International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

An Intelligent 64-bit parallel CRC for high-speed communication system applications

Ajay Sudhir Bale[1], Karmanyaraj Singh Yadav[2], Mahboob Alam[3], Abhinav Shrivastava[4], Raj A. Varma[5], Rajdeep Singh Solanki[6] and Mamta B. Savadatti*[7]

Submitted: 27/05/2023 **Revised**: 07/07/2023 **Accepted**: 24/07/2023

Abstract*:* In the networking context, CRC plays a critical function in detecting mistakes. It is essential to improve the pace of CRC creation in order to keep up with the speed of data transmission. The cyclic redundancy check is well-known among engineers (CRC). Many people are becoming aware that it is used to identify bit mistakes in communication channels and that it is fundamentally a residual of the modulo-2-long division operation. These linear feedback shift registers (LFSRs), which process data serially, often are used in the hardware implementation of CRC calculations as a crucial technique for dealing with data mistakes. This CRC code's serial computation cannot reach a high throughput. The throughput of CRC computations may be substantially increased by using constant concurrent CRC calculations. The CRC-16BISYNC protocol, CRC32 for error detection in Ethernet; CRC8 for ATM; CRC-CCITT for the X-25 set of rules, disc storage, XMODEM, and SDLC are all examples of applications that utilize CRCs of various types. We have focused on the main features and the trends of 64bit parallel CRC architecture and applications.

Keywords: *Redundancy check, Error control coding, Shift register, Parallel CRC calculation.*

1. Introduction

Cyclic redundancy check (CRC) is an error-detecting algorithm [11] that is often used in virtual networks and unintentional changes. A fast test cost is applied to each block of information that enters those structures, and this cost is dependent entirely on the remainder of a polynomial department in their contents. CRC technology, also known as Cyclic (CRC), may be used to verify the integrity of a process in the context of facts or facts compression. It has utilised within the communication community as well as the information garage period, among other places.

The Cyclic Redundancy Check (CRC) is a strong coping with data mistakes that is extensively used in data transmission and storage systems. A wide range of other areas, including the testing of integrated circuits and defect detection, have also benefited from this technology.

[1] *Dept. of ECE, New Horizon College of Engineering, Bengaluru, India ajaysudhirbale@gmail.com; ORCID ID : 0000-0002-5715-9739*
[2] *Viterbi School of Engineering , (Electrical and Computer Engineering), University of Southern California (USC) , California; ksyadav@usc.edu*
[3] *Manav Rachna International Institute of Research and Studies Faridabad; mahboobalam.fet@mriu.edu.in*
[4] *Assistant Professor, Symbiosis Law School (SLS), Symbiosis International (Deemed University) (SIU), Vimannagar, Pune, Maharashtra, India; abhinav.shrivastava@symlaw.ac.in*
[5] *Assistant Professor, Symbiosis Law School (SLS), Symbiosis International (Deemed University) (SIU), Vimannagar, Pune, Maharashtra, India; raj.varma@symlaw.ac.in*
[6] *Assistant professor, Medi-caps university indore, Computer science department, rajdeep.solanki@medicaps.ac.in;*
[7]*Dept. of ECE, New Horizon College of Engineering, Bengaluru, India mamta.savadatti@gmail.com;; ORCID ID : 0000-0003-0679-7084*
** Corresponding Author Email: mamta.savadatti@gmail.com*

The cyclic redundancy test (CRC) is widely used in networking settings to determine whether or not errors were introduced at some point across physical connections. In this article, we demonstrate that the CRC computation is performed at some point during the routing packets by means of, which include a body header and a body trailer, respectively. CRC code is located on the inside of the body trailer, next to the bumper. As a result of our research, we discovered that the finest unique fields located at the commencement of a body are changed as the body travels via interconnecting devices. The process of error correction returns to computers. When the 7-bit ASCII character set was used on certain computers, 8-bit check bits were employed to ensure that the characters were delivered properly. A character's bottom 7 bits are set to 1 if there are an odd number of 1s in the lower 7 bits; otherwise, the character's 8th bit is set to 0. Despite the fact that this is a simplified method, some key principles shared with CRC are as follows:

Check data has been added to the data to be sent (duplicate). This is not a comprehensive list. Detects particular faults that are intended to be checked. It is specifically designed to properly identify every one-bit mistake in each seven-bit block of data, but it has the potential to fail if more severe data corruption occurs. The receiver calculates the data it gets from the sender using the same CRC method that was used by the sender. A successful transmission or compression operation is indicated by a computed value that is equal to or greater than the code check of the frame. The linear feedback register structure, which is used to implement the traditional CRC method, is used to implement the algorithm. Because it is a serial coding technique, it is simple to understand and put into practise. The calculation time, on the other hand, is very sluggish when using the traditional CRC method. High-speed data transmission is not possible with this device. The parallel CRC [3-5] method is capable of meeting the demands of high-speed data transmission. Parallel CRC is a popular technique that may be implemented in either software or hardware. A consequence of the sluggish processing speed of software, algorithms have been implemented in many communication devices by hardware in order to achieve high throughput. According to the concept of conventional serial CRC algorithms, the MATRIX-based parallel CRC PIPELINE Method is a high-speed coding algorithm that is based on a matrix.

The method has the potential to decrease circuit latency while simultaneously increasing data processing speed. CRCs may be produced in two ways: by hand or by computer. A. The creation of serial CRCs. B. Generation of CRCs in parallel. LFSRs (linear feedback shift registers) are often used in hardware implementations of CRC computations because they allow data to be processed in a sequential manner. Serial calculation of CRC codes, on the other hand, is not capable of providing high throughput. This article discusses the creation of parallel CRCs in the 64-bit domain. Although a parallel CRC with 32 bits generates gigabits/sec, it is not appropriate for high-speed applications such as Ethernet because of its limited size.

1.1. Serial CRC generation

In this instance, the CRC verification is done in a sequential manner. Only one bit and each clock pulse are used as data input. When there is a wait between subsequent data inputs and the information is encoded with the identical CRC value, the outcome will be 0; otherwise, the outcome will show a non-zero number. We will then be able to tell whether the information is valid or distorted by using this form.[1]

.

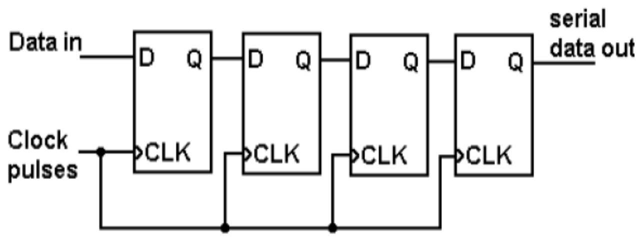

Fig. 1. Flip-Flop connected to form IO Shift register

A linear feedback shift register (LFSR) circuit is usually used to do CRC computations, and it is often constructed in VLSI (Very-Large-Scale Integration) technology, which can only process one bit per cycle. computations have lately gained popularity, and one byte or several bytes may often be handled in parallel, depending on the situation. When attempting to achieve parallelism, it is common practice to. Unfortunately, the parallelism techniques increase the length of the worst-case temporal route, resulting in speedups that are less than optimal in. Furthermore, as the degree of parallelism grows, so does the amount of space and power that must be allocated to the system. A new approach to integrating CRC hardware is required as a consequence, and we are seeking for a solution that would while still using a reasonable amount of space and power. CRC may be generated in dualistic ways:

1. Serial CRC generation

2. Parallel CRC generation

1.2. Hardware Implementation of CRC

When it comes to the hardware implementation of CRC calculations, linear feedback shift registers (LFSRs) come in handy since they handle data serially. In contrast, the serial computation of CRC codes is incapable of achieving high throughput. Therefore, in order to resolve this issue, we are moving to Parallel CRC generation (PCR). A 32-bit parallel CRC may produce gigabits per second, but it is not as appropriate as Ethernet due to its limited memory capacity. In conventional is used to do the CRC computation, which is accomplished by treating each bit of the message one by one. Figure 1 illustrates a common serial CRC that makes use of LFSRs. There are a total of 32 registers; the middle ones have been removed sake of brevity. It is the XOR operation that is shown in the picture, which is a combinational logic operation. One bit is transferred into the memory with each clock pulse. In the same manner that manual long division works, this circuit operates as well. The XOR gates in Fig. 1 are responsible for holding the that correspond to the stated. However, while method to calculating CRCs is usually done in hardware, it is possible to implement this algorithm in software provided the amount of bit-by-bit processing required is not excessive.

Parallelizing CRC computation has been a significant emphasis in recent years due to the increasing need for faster processing rates in today's applications. To improve the system's overall performance, Cheng and Partii investigated and combined it using the pipelining and retiming approaches. Due to their use of operations such as multiplication and division on the generating polynomial, the parallel long [12]. BCH encoders are very efficient in speeding up the parallel linear feedback shift register (LFSR) computation. Unfortunately, the expense of putting this plan into action is very substantial. Campobello and colleagues devised a linear systems theory-based method to unrolling the serial implementation, which they published in 2011. In this method, the underlying premise is that the packet size is a multiple of the CRC input size. When message fragments are out-of-order, an incremental method for computing the CRC on the fragments is possible. Each incoming its part independent of the arrival of any other segments, to the message's CRC at the time of arrival, enabling the message to go straight to the top-layer protocol without any further processing. Additionally, a method to compute the CRC in parallel has been presented by a number of software-based algorithms, and the concept is still widely employed today. Walma developed a hardware-based approach that concentrates on computation in order to increase throughput. For CRC computation, the Linear Feedback Shift Register (LFSR) is one of the most well-known hardware alternatives available (LFSR). LFSRs are made up of a small number of flip-flops (FFs) and logic gates. Because of fundamental design, the bits may be processed in a serial fashion. Some applications, such as high-speed data transfer, are totally insufficiently served by this serial implementation due to its insufficient speed. Consequently, the CRC is calculated in parallel, with consecutive units of w bits being processed at the same time as one another. It is both necessary and beneficial to do so. Only two layers of gates are required to construct parallel CRC hardware, which is the same as for any other combinatorial circuit. Protocols are employed in communication networks to meet the ever-increasing demands for speed that are placed on them. As a consequence, ASIC-based circuits can keep up with the increasing demands for performance. This is most likely going to continue to be the case in the foreseeable future. It is essential to achieve the required processing speed since If analysis remains incomplete at wire velocity, packets will be refused. As novel protocols, including IEEE 802.11n WLAN and UWB (Ultra-Wide Band), offer significantly greater throughput requirements, have emerged in recent years, new protocols [21] with even higher throughput requirements are expected to emerge in the near future as well. For example, IEEE 802.3ak (10 Gbps) was standardized in 2003, and currently work has started on standardizing IEEE 802.3 (100 Gbps) in the same year. As a result, fulfilling these requirements is becoming an increasingly significant concern. Most communication protocols use CRC as an effective method of detecting transmission problems, and as a result, high-speed CRC computation is required. In to handle these high throughput CRCs an acceptable frequency, it is necessary to process multiple bits in parallel and pipeline route via which the data is sent.

2. Related Work

[1] The present state of the network environment is characterized by high-speed data transmission. Cyclic redundancy check (CRC) is a technique identifying mistakes in data transfer that is absolutely necessary. Data transmission rates are difficult to achieve, and speed and synchronization are required to accelerate CRC creation. Following the serial structure, we provided a recursive phrase, which was then used to generate the parallel structure. We propose a [17] 64-bit parallel CRC that is based on a F matrix of order 32 of the generating polynomial, as described in this work. The suggested architecture is hardware efficient, and the number of cycles needed to create a generative polynomial CRC of the same order is decreased by 50% compared with the previous design. The Xilinx ISE simulator is used to functionally verify the whole design from start to finish. Circular redundancy checking is widely utilized as an essential technique for addressing data problems in a variety of disciplines, including data transmission areas such as and data reduction. LFSRs (linear feedback shift registers) are often hardware implementations of CRC computations because they allow data to be processed in a sequential manner. Serial calculation of CRC codes, on the other hand, is not capable of providing high throughput. Parallel CRC calculation, on the other hand, has the potential to substantially increase the throughput of CRC computation. CRC32, for example, has a 32-bit parallel computational performance that may reach several gigabits per second when used in parallel. High-speed applications like as Ethernet, on the other hand, are now insufficient.

[2] This paper offers a Fast Cyclic Redundancy Check (CRC) method that executes CRC computations in all message lengths, regardless of the message length. No matter how long the message is, the chosen message algorithm chops it into blocks first, regardless of its length. Each block has a fixed size equal to the degree of the generating polynomial, which is the same for all of them. Then it conducts a CRC computation using just the parallel chunked block lookup table and combines the result using the XOR operation, which completes the process. That which caused the pipeline to become an issue was the input from previous implementations. With SMIC0.13mCMOS technology, the suggested method overcomes this issue and performs a 32-bit CRC pipeline computation with a high level of accuracy. This method can calculate data rather than the whole width of the input rather than the entire width of the input. In addition, our algorithm has extremely low latency in the pipeline, and by utilizing this approach, we can quickly increase the only parallelism while only having a small effect on the overall time. Using simulations, it has been shown pipeline CRC is more efficient than the existing pipeline CRC implementation.

By comparing the to the received CRC, an error was discovered. The CRC method adds just the bit with the smallest number of messages (32 bits for CRC32), making it particularly effective at identifying single mistakes and bursts of errors. CRC32 is a 32-bit CRC algorithm. Demonstrate your ability to execute. I'll demonstrate. Given that CRC algorithms are good at error detection and are straightforward to implement in hardware, CRC algorithms are extensively employed today to identify digital data corruption that may occur during the creation, transmission, and storage of digital data. Furthermore, CRC just found a new use of a standard for general purpose mobile communication systems, f1which is used to identify the length of variable-length message communications messages, which was previously unknown.

Presented in this paper are theoretical findings that may be used to the implementation of high-speed hardware for parallel CRC checksums. The serial implementation [3] that has been extensively published in the literature and then defined a recursive expression that derives from a parallel version that was also well documented. Contrary to earlier efforts, this latest draught is more efficient and compact while remaining independent of the technology utilized in its execution. The result we propose is that the number of bits parallelized may polynomial in our example. Finally, we have created a high-level parametric code that can build circuits on its own when just given the input of Polynomial and no other parameters. Cycle redundancy checks are extensively utilized as a strong method of communicating data and dealing with data mistakes from storage devices because of their simplicity. Furthermore, it is applicable to a wide range of other fields, such as integrated circuit testing and logic fault detection. Another well-established hardware method [16] for CRC computations is the Linear Feedback Shift Register (LFSR), which comprises of a logic gate with multiple flip-flops and is one of the most often used hardware alternatives (FF). The bits are handled in sequence by this straightforward design. The speed of this is completely inappropriate in certain circumstances, such as high-speed data transmission, when it is absolutely necessary. The parallel computation of CRC, in which units of w bits are processed at the same time, is desirable or necessary in such a situation.

This method is based when an Ethernet frame is sent from a router or interconnect device, only the tiny portions at the beginning of the frame are changed, rather than the whole frame. Our fast CRC update technique has been enhanced to include parallel CRC calculation, and it can be scaled to accommodate the number of bits that are processed in parallel as needed. This technique may also help to decrease the amount of data flow and the amount of power used by the CRC compute unit. In addition, this white paper offers proof of the effectiveness of a fast CRC update method for accuracy.

The review [4] offers a Fast Cyclic Redundancy Check (CRC) method that executes CRC computations in parallel for all lengths, regardless of the message length. No matter how long the message is, the chosen message algorithm chops it into blocks first, regardless of its length. Each block has a of the generating polynomial, which is the same for all of them. Then it conducts a CRC computation using just the parallel chunked block lookup table and combines the result using the XOR operation, which completes the process. That which caused the pipeline to become an issue was the input from previous implementations. With SMIC0.13mCMOS technology, the suggested method overcomes this issue and performs a 32-bit CRC pipeline computation with a high level of accuracy.

[6] High-speed IP lookup engines are required because of the quick growth in network link speeds. Trie-based designs are ideal options for high-throughput pipeline solutions. The tri-levels to the pipeline stages will, however, unequally share the memory among them. To address this issue, several fresh pipeline architectures have been put up. However, that non-linear pipeline topology brings with it a host of brand-new performance problems, such as decreased throughput and variable latency. For trie-based IP lookups, this article offers a linear pipeline architecture. Our architecture achieves high throughput of one query every clock cycle while maintaining evenly distributed memory. It allows additional flexibility in how tree nodes can be mapped to pipeline phases and supports NOPs. On commercialized FPGAs, we tested our designs and cutting-edge solutions to gauge their performance. 80 Gbps post-location and path throughput, up to two times that of the benchmark solution. Without interfering with active IP lookup processes, progressive route updates are supported and can be scheduled

RCAM based and SRAM based solutions. The IP lookup results of the TCAM based engine clock, but the to the low speed of TCAM1. SRAM is superior to TCAM. However, conventional SRAM-based engines need cycles to do a lookup. The usage of pipelines can considerably increase processing speed, as numerous academics have noted. It is straightforward to map each tri-level to a separate pipeline stage having separate memory, processing, and logic for tri-based IP lookups. One IP packet only passes through one of the pipeline's many phases. can be found for one clock period [18]. However, this method distributes the challenge nodes disproportionately across the various pipeline stages. This has been identified as a major problem in pipeline architecture. An imbalanced pipeline requires more time to access larger memory stages that contain more tri-nodes. It is also updated more frequently in proportion to the number of trinodes stored in local memory. Intensive root inserts, a large stage can lead to a memory overflow. Therefore, it becomes a frequently used bottleneck like this and can affect the [13] overall performance of pipeline

3. Methods

3.1. Existing method

Serial CRCs are traditionally generated using linear feedback shift registers, which are utilized in the conventional method of generating them (LFSR). The binary divisions are the most important operation performed by the LFSR while performing CRC calculations. When doing binary divisions, a sequence of shifts and subtractions may be utilized to get the desired result. In modulo 2 arithmetic, multiplication is equivalent to AND (indicated by the letter "&" in this article), while binary-addition and binary-subtraction are equal to bit-wise XOR (&).

CRCs may be produced in one of two ways: by hand or by computer.

A. Serially generated CRC generation.

B. Parallelly generated CRC generation.

For most of the part, CRC calculations are performed in hardware using LFSRs (linear feedback shift registers), which have been designed to process data in a serially-based method. In contrast, the serial computation of CRC's code is incapable of achieving high throughput. Consequently, we have been moving towards parallel CRC generation in order to resolve such issues.

In this instance, the CRC check is performed in a serial manner. It is anticipated that the data input will be single (binary), and that there will be one data input for each clock pulse. It is necessary to wait between successive data inputs. The outputs will become zero (0) if the data is encoded with the same CRC's value, or it will become a non-zero value if the data is not.

The CRC circuit can be implemented using shift registers and XOR gates because CRC computations are performed in GF (2). The linear feedback shift register (LFSR) based CRC generator is a popular name for this technique. [19] The degree of the generating polynomial determines the number of flip-flops required to construct the shift register.

The same architecture can be utilized by both the transmitter and the receiver to generate and verify CRC codes. With a generating polynomial of degree k, k flip-flops are required to create an LFSR-based CRC generator. All of the flip-flops in an LFSR-based CRC circuit use the same clock and clear signals. The coefficients in the generator polynomial are denoted by pi's in the illustration. The XOR of the output of the (i - 1) th flip-flop and the output of an AND gate with D and xk−1 as input is used to calculate the i 'th flip-input. Flip flop's It is equivalent to a shift operation if either of the coefficient's pi is zero

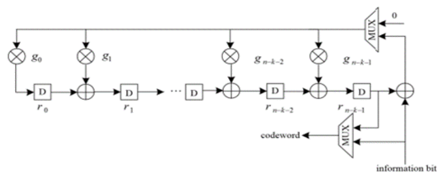

Fig. 2. Basic LFSR Architecture with polynomial degree k [9]

Only a single bit of the input stream is moved in at a time in the LFSR-based circuit. As a result, (m + k) number of clock cycles will be required to calculate the CRC-k function of an m-bit long message.

There are a variety of approaches to generating CRCs at the same time.

a. Table based procedure for Calculating the CRC of Pipelined Data.

b. Update the CRC in a short period of time

c. F matrix centered Parallel CRC creation.[14]

d. Algorithm for unfolding, retiming, and pipelining.

Because of the heavy pipelining, the LUT base design delivers a smaller memory LUT. Input, LUT3, LUT2, and LUT1 are all part of the table foundation architecture. LUT3 is made up of input CRC values followed by 12-bytes of zeros, an LUT2 made up of 8 bytes, and an LUT4 is made up of 4 bytes. Essentially, this approach can increase throughput. There is no requirement for a lookup table in the rightmost block. Because CRC-32and 4-byte blocks are assumed in this architecture. The CRC value of a binary string is the string itself if its length is less than the degree of the CRC generator. Because the rightmost block corresponds to A4, it has no following zero and hence has the same CRC as the block.

Combining the results of XOR and LUT4 together, as well as the outputs from LUT4, i.e., the CRC of the value from the iteration with the previous 16 bytes of zeros concatenated. Results from the output of LUT4 being used to check for a repeatability of the value from the prior iteration with 16-bytes of zeros concatenated. We include a pre-XOR stage immediately before the four-input XOR - gate in order to reduce the length of the critical path and improve performance [20]. Additional blocks can be added without cumulating the critical path of the pipeline, allowing the algorithm to become more scalable. The pre-XOR stage is comprised of a XOR-gate (2 i/p) and a delay of LUT4, with a throughput of 16 bytes per cycle. The pre-XOR stage is comprised of the following components.

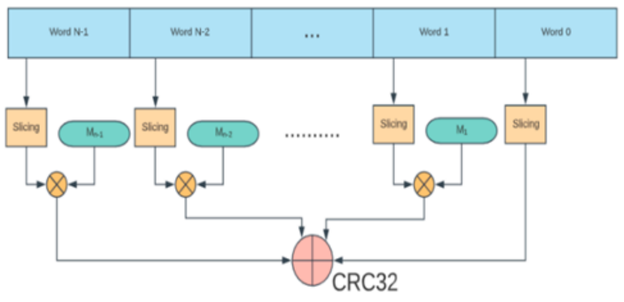

Fig. 3. LUT based CRC [10]

Parallel processing is a technique for boosting throughput by producing many outputs at the same time. Retiming is a technique for boosting the clock-rate of a circuit by minimising the critical route computation time as in figure 3

Instead of computing CRC for all of the data bits each time, the quick CRC update algorithm calculates CRC for only the bits that have changed. There are several methods for generating parallel CRCs, each with its own set of benefits and drawbacks. Table-based architecture necessitates the use of a pre-calculated LUT; hence it will not be utilized for generalized CRC generations. Fast-CRC update techniques necessitate the use of a safeguard to retain the old CRC as well as the data. The number of iterations bound rises as the architecture unfolds.

In pipelined systems, as in figure 4, the most effective critical-path is minimized by adding pipeline latches along the perilous data channel. This may be used to simultaneously increases the clock-frequency and sample the speed, and to simultaneously decrease the current power consumption, depending on the application. The iteration bound is defined as the sum of all loop boundaries multiplied by a certain number of iterations. The loop-bound efficiency is defined as ‘t/w’, where t is the loop's computation time and w denote the number of delayed elements inside the loop.

Retiming is a method for relocating delay components in a circuit without it affecting the I/O characteristics of the circuit. It helps to reduce the critical path of the system by keeping the system's latency constant. Retiming is a technique used in synchronous circuit design that has a broad variety of applications.

These submissions include decreasing the clock period of the circuit, decreasing the number of registers in the circuit, lowering the power consumption of the circuit, and logic synthesis. In order to increase the clock rate of a circuit, it may be utilized to reduce the calculation time of the critical route [15]. This route is the one with the longest calculation time among all zero-delay pathways, and the time it takes to calculate reflects the circuit's lower limit on the total clock period. The critical route and the iteration limit are the two factors that have an impact on the frequency with which operations are performed. The Floyd Warshall method, as well as the algorithm given in, are used to determine when to retire.

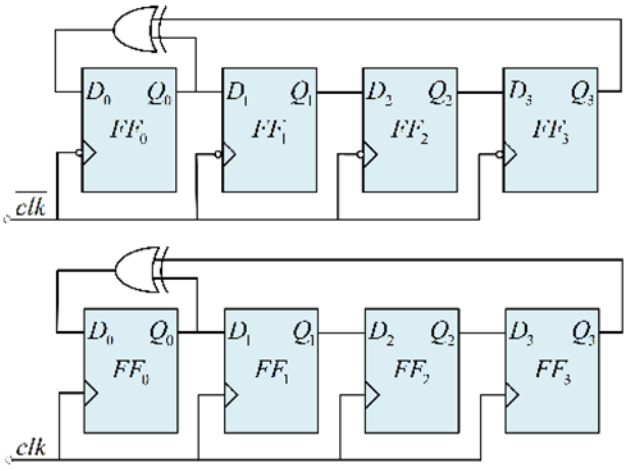

Fig. 4. Basic LFSR Architecture [11]

As depicted in figure 5, d represents serial data input, Q represents the current state (produced CRC), Q' represents the future state, and p represents the generator polynomial. The following equations describe how the basic LFSR design works.

After a (k+ms) cycle, a Frame Check Sequence (FCS) will be generated, (k denotes the number of data-bits while m denotes the ranking of the generating polynomial). If the order of the generator polynomial is 32 for a 32-bit serial CRC, the serial CRC will be created after 64 cycles.

The highest order of G, corresponds to the most important bit of P, p32(x)i.e., coefficient of x32. Similarly, order p31 is the figure of x31, which in this case is a zero. The next bits are positioned in the

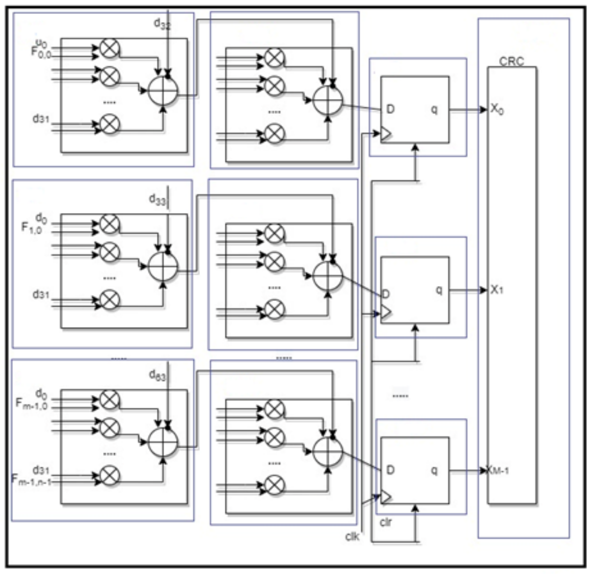

Fig. 5. Stages in a Pipeline CRC

same order as the coefficients in G(x). P is referred to as the generator, as it is the only polynomial that corresponds with the generating polynomial G(x).

The following are the basic steps in calculating CRwC:

- An unsigned-binary integer D, which could be a few hundreds of bits long, is seen as the message to be safeguarded. In this work, the length of D is indicated by k.
- D is appended with m-0s. This task can be thought of as a 2m multiply function that yields D * 2*m.
- Using modulo 2 arithmetic, generator P divides the binary number D *2m. The remainder of this procedure is represented by C, which is an m-bit CRC code.
- The CRC-code called as C is appended to the extended message, bring about in result of D * 2m + C, which will be sent. In fact, addition of C is the same as trading the appended m-0s with C, thus no real addition is required.
- Finally, if no transmission mistakes occur, the residual should be 0 at the Receiving end while completing the CRC on D*2m+C.
- Based on the preceding discussion, the main operation of CRC calculations is binary divisions. A series of shifts and subtractions can be used to perform binary divisions. Furthermore, totaling and deduction are similar to bitwise XOR in modulo-2-arithmetic, while multiplication is equivalent to AND. As a result, shifts and bitwise XOR can be used to do binary divisions in modulo 2 arithmetic. Shifts and bitwise XOR, or binary divisions, are performed via linear feedback shift registers (LFSRs). The serialized and analogous CRC computation implementations grounded on LFSRs are detailed in the following sections.

4. Simulation

Beginning with the most significant bit (MSB) and concluding with the m_0 appended to the data message, these message data are moved from the leftward. When all of the messages have been moved in, the balance of the binary-division, i.e., the CRC encryption, is stored in the shift registers. The LFSRs are thought to be a time-invariant linear arrangement that operates in discrete time.

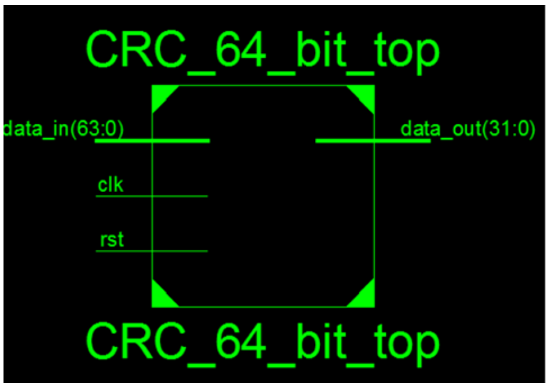

Fig. 7. RTL Schematics for CRC_64_bit_top

Evaluation for Area & Delay report is shown in Table 1 and simulation review as in Figure 8-10, These implementations are done using Xilinx by considering the inputs and data after referring to [22].

Table. 1. Area & Delay

	Area	Delay(ns)	Delay(ns)
Parallel CRC	32	32	32

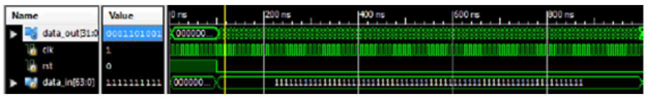

Fig. 8. RTL Schematics- CRC Test

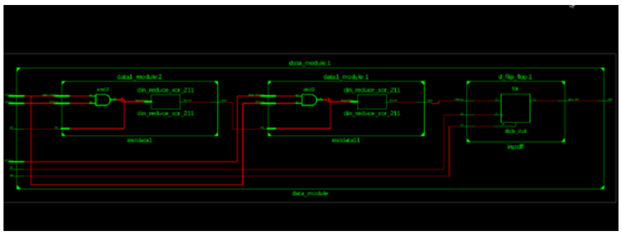

Fig. 9. RTL Schematics- CRC Test

Fig. 10. Simulation Output for CRC

5. Conclusion

In this work we are generating CRC code by employing a F matrix parallel CRC generation method. By comparing with the traditional method of serial CRC generation method we are going to get the high throughput and fewer delay. In proposed architecture 64 bits are parallel processed and order of generator polynomial is 32-bits. This generated CRC codes are employed in the Ethernet protocols and where the high-speed devices are utilized in the communication systems. Based on the results of the relevant survey, it is clear that serial execution is not favored if high-speed data transfer is essential. As a result, we've moved to a less-time-consuming parallel method of execution. The trade-offs among adaptability, efficiency, and cost that are allowed by conventional diverse designs based on microprocessors and digital signal processors (DSPs) have been expanded upon by CRC.

References

[1] Mathukiya, H. H., & Patel, N. M. (2012). A Novel Approach for Parallel CRC Generation for High Speed Application. 2012 International Conference on

Communication Systems and Network Technologies. doi:10.1109/csnt.2012.131

[2] Y. Sun and M. S. Kim, “High performance table-based algorithm for pipelined CRC calculation,” J. Commun., vol. 8, no. 2, pp. 128–135, 2013, doi: 10.12720/jcm.8.2.128-135.

[3] Campobello, Giuseppe et al. “Parallel CRC Realization.” IEEE Trans. Computers 52 (2003): 1312-1319.

[4] C. Cheng and K.K. Parhi, "High-Speed Parallel CRC Implementation Based on Unfolding, Pipelining and Retiming", IEEE Trans. Circuits and Systems-II: Express Briefs, 53(10), pp. 1017-1021, Oct. 2006

[5] P. S. Hajare and K. Mankar, “Design and Implementation of
Parallel CRC Generation for High Speed Application,” IOSR J.
VLSI Signal Process. (IOSR-JVSP, vol. 5, no. 3, p. 1, 2015, doi:
10.9790/4200-05320105.

[6] P. Sushma, “64-bit parallel CRC Generation for High Speed Applications,” Int. J. Comput. Appl. Inf. Technol., vol. 5, no. I, pp. 29–34, 2014.

[7] W. Jiang and V. K. Prasanna, “A memory-balanced linear pipeline architecture for trie-based IP lookup,” Proc. - 15th Annu. IEEE Symp. High-Performance Interconnects, HOT Interconnects, pp. 83–90, 2007, doi: 10.1109/HOTI.2007.10.

[8] M. Dasari, “Design and Implementation of Parallel CRC Generator for 64-Data Bit,” vol. 1, no. December, pp. 151–156, 2014.

[9] J. Kang, J. S. An and B. Wang, "An Efficient FEC Encoder Core for VCM LEO Satellite-Ground Communications," in IEEE Access, vol. 8, pp. 125692-125701, 2020, doi: 10.1109/ACCESS.2020.3007923.

[10] Tran, D., Aslam, S., Gorius, N., & Nehmetallah, G. (2021). Parallel Computation of CRC-Code on an FPGA Platform for High Data Throughput. Electronics, 10(7), 866. doi:10.3390/electronics10070866

[11] G. Dai, W. Xie, X. Du, M. Han, T. Ni, and D. Wu, “Memristor-Based D-Flip-Flop Design and Application in Built-In Self-Test,” Electronics, vol. 12, no. 14, p. 3019, Jul. 2023, doi: 10.3390/electronics12143019.

[12] Sun, Y., & Kim, M. S. (2010). A Table-Based Algorithm for Pipelined CRC Calculation. 2010 IEEE International Conference on Communications. doi:10.1109/icc.2010.5501903

[13] W. Jiang and V. K. Prasanna, "A Memory-Balanced Linear Pipeline Architecture for Trie-based IP Lookup," 15th Annual IEEE Symposium on High-Performance Interconnects (HOTI 2007), Stanford, CA, USA, 2007, pp. 83-90, doi: 10.1109/HOTI.2007.10.

[14] R. O. S. Juan and H. S. Kim, "Utilization of DSP algorithms for Cyclic Redundancy Checking (CRC) in Controller Area Network (CAN) controller," 2016 International Conference on Electronics, Information, and Communications (ICEIC), Danang, Vietnam, 2016, pp. 1-4, doi: 10.1109/ELINFOCOM.2016.7562991.

[15] Panda, G., Satapathy, S. C., Biswal, B., & Bansal, R. (2018). Microelectronics, Electromagnetics and Telecommunications. Fourth ICMEET, 802.

[16] N. N. Qaqos, "Optimized FPGA Implementation of the CRC Using Parallel Pipelining Architecture," 2019 International Conference on Advanced Science and Engineering (ICOASE), Zakho - Duhok, Iraq, 2019, pp. 46-51, doi: 10.1109/ICOASE.2019.8723800.

[17] Y. Huo, X. Li, W. Wang and D. Liu, "High performance table-based architecture for parallel CRC calculation," The 21st IEEE International Workshop on Local and Metropolitan Area Networks, Beijing, China, 2015, pp. 1-6, doi: 10.1109/LANMAN.2015.7114717.

[18] Das, "Block-Wise Computation of Cyclic Redundancy Code Using Factored Toeplitz Matricesin Lieu of Look-Up Table," in IEEE Transactions on Computers, vol. 72, no. 4, pp. 1110-1121, 1 April 2023, doi: 10.1109/TC.2022.3189574.

[19] J. Cho and W. Sung, "Efficient Software-Based Encoding and Decoding of BCH Codes," in IEEE Transactions on Computers, vol. 58, no. 7, pp. 878-889, July 2009, doi: 10.1109/TC.2009.27.

[20] Y. -K. Chang, F. -C. Kuo, H. -J. Kuo and C. -C. Su, "LayeredTrees: Most Specific Prefix-Based Pipelined Design for On-Chip IP Address Lookups," in IEEE Transactions on Computers, vol. 63, no. 12, pp. 3039-3052, Dec. 2014, doi: 10.1109/TC.2013.109.

[21] Sun, Yan & Kim, Min. (2010). A Pipelined CRC Calculation Using Lookup Tables. 1 - 2. 10.1109/CCNC.2010.5421679.

[22] Hajare, P. S., & Mankar, K. (n.d.). Design and implementation of parallel CRC generation for high-speed application. https://doi.org/10.9790/4200-05320105

[23] Harsh, S. ., Singh , D., & Pathak , S. (2021). Efficient and Cost-effective Drone – NDVI system for

Precision Farming. International Journal of New Practices in Management and Engineering, 10(04), 14–19. https://doi.org/10.17762/ijnpme.v10i04.126

[24] Vadivu, N. S., Gupta, G., Naveed, Q. N., Rasheed, T., Singh, S. K., & Dhabliya, D. (2022). Correlation-based mutual information model for analysis of lung cancer CT image. BioMed Research International, 2022, 6451770. doi:10.1155/2022/6451770