Enhanced CRC Computation Through Dynamic Pipelined Architecture

Mohammad Saifali Shaikh*, Sathish V*, Sandeep Rangappa Chudappagol*, Abhishek Sangamesh Sanakal*, Dr. Jayanthi P.N*
*Electronics and Communication Department, R. V. College of Engineering, Bengaluru, India
Email: {msaifalishaikh.ec22, sathishv.ec22, sandeeprc.ec22, abhishekss.ec22, jayanthipn}@rvce.edu.in
R. V. College of engineering, Bengaluru, India

***Abstract*—In digital communication and storage systems, the Cyclic Redundancy Check (CRC) is a crucial error-detection mechanism that is frequently used to detect problems during data transmission or storage and guarantee data integrity. Due to sequential data processing and reliance on single-edge clock triggering, traditional CRC implementations frequently suffer from issues like high latency and limited throughput, which can impair performance in high-speed systems. This project offers a Verilog-implemented pipelined CRC computing architecture that uses alternating clock edges and a x+1 (x refers to number of bits) stage XOR division pipeline to get around these restrictions. The design efficiently doubles data throughput and lowers latency by running computations on both rising and falling clock edges. This enables parallel processing, which improves the efficiency of CRC calculations. First, last, and valid control signals are included in the architecture to regulate data flow and guarantee synchronization throughout pipeline stages. Incoming data is processed in real-time by this system, which also computes the CRC checksum and compares it to an expected checksum to confirm data integrity. Because the pipelined design maximizes hardware utilization, it is perfect for high-speed communication protocols and FPGA-based implementations. Performance research shows that the suggested architecture achieves better real-time error detection, reduced delay, and higher throughput than conventional single-edge-triggered CRC designs. This method provides a low-latency scalable solution for next-generation error detection methods, which is especially advantageous for applications in digital signal processing, embedded security systems, high-performance computing, and fault-tolerant data transmission networks.**

***Index Terms*—Cyclic Redundancy Check (CRC), Error detection, Pipelined architecture, FPGA (Field-Programmable Gate Array), Sequential processing, Latency, Parallel processing, clock synchronization, data integrity**

I. INTRODUCTION

In digital communication systems, the Cyclic Redundancy Check (CRC) algorithm is a vital tool for identifying potential transmission mistakes. Because of its ease of use and effectiveness, the CRC algorithm is the most widely used of the different CRC schemes. It may be applied to a variety of communication protocols and low-resource contexts. Based on the input data, it creates a fixed-size checksum that is subsequently used to confirm the accuracy of the data that was received. Error management procedures are activated when differences between the transmitted and received checksums are found. There have been errors. As data volumes and communication speeds continue to rise, the conventional CRC algorithm, despite its efficacy, confronts difficulties.

Conventional CRC calculation techniques usually use sequential processing, which processes each piece of data individually. This causes a lot of latency and performance snags, especially in systems with high speeds. The real-time demands of contemporary communication protocols are difficult for these traditional approaches to match as data transmission rates increase and systems become more sophisticated. This issue is most noticeable in systems where fast data integrity checks are essential, such wireless communication, embedded devices, and Internet of Things applications. Researchers and engineers are now investigating cutting-edge ways to get around these restrictions as a result of the increased demand for quicker and more effective CRC computation methods.

A method called pipelining, which was taken from digital circuit design, divides the CRC computation process into multiple steps so that each step can analyze distinct data segments concurrently. This technique allows for the simultaneous execution of several CRC calculations, which effectively lowers the overall latency. This method significantly improves the CRC calculation's performance because each data bit is handled without waiting for the preceding one to finish. This parallelism is perfect for high-throughput applications since it enables a notable improvement in processing speed.

An FPGA (Field-Programmable Gate Array) platform is used in the project to develop the pipelined CRC calculator. This platform is ideal for such jobs because of its reconfigurability and capacity to manage concurrent processes well. The FPGA-based solution guarantees excellent resource utilization and low latency when executing the pipelined CRC algorithm. Compared to conventional processor-based systems, FPGAs provide notable speed and flexibility advantages since they can manage several tasks at once, which lowers the total amount of time needed to calculate the CRC checksum for every data packet. Scalability is another benefit of FPGA implementation, which enables the system to adjust to different data volumes and transmission rates.

The results of its development, use, and testing demonstrate that the Dynamic Pipelined CRC computations are a high-speed error detection solution that functions effectively for modern digital communication networks. In order to overcome the performance bottlenecks present in conventional CRC

calculation techniques, the project integrates pipelining into the process, resulting in significant gains in processing efficiency, speed, and scalability. The architecture may be readily modified for real-time applications in embedded systems, Internet of Things devices, and wireless communication thanks to the FPGA implementation, which also reduces latency. With this initiative, the bar for error detection methods is raised, opening the door for further developments in data integrity solutions that can satisfy the ever-increasing requirements of contemporary electronic systems.

II. Related Works

The development of Cyclic Redundancy Check (CRC) algorithms has been a critical area of research due to their essential role in ensuring error detection and data integrity in digital communication systems. Traditional CRC algorithms, such as CRC-8, CRC-16, and CRC-32, have been widely adopted because of their simple but effective error detection capabilities. These algorithms generate a checksum based on the input data, which is then used to verify the integrity of the received data. If discrepancies are found between the transmitted and received checksums, it indicates potential data corruption. However, despite their simplicity and reliability, these traditional CRC implementations face inherent limitations in high-speed systems, primarily due to their sequential processing method. In these conventional designs, the data is processed bit by bit, leading to increased latency, reduced throughput, and an inability to keep pace with modern, high-speed communication systems.

As communication systems continue to evolve and data transmission speeds increase, a concerted effort has been made in recent years to overcome the limitations of traditional CRC algorithms. One of the most promising advancements has been the development of parallel CRC algorithms. These algorithms take advantage of parallel processing techniques to compute multiple bits of the CRC simultaneously, significantly enhancing throughput and reducing processing time. By leveraging lookup tables, matrix-based methods, or even hardware accelerators, parallel CRC algorithms can process large data blocks in a fraction of the time required by traditional sequential methods. However, while parallel methods offer increased throughput, they often come with increased resource demands. These resource-intensive solutions can place a heavy burden on hardware platforms, particularly on Field-Programmable Gate Arrays (FPGAs), where resource usage, such as memory and logic elements, is a crucial consideration.

The Pipelined CRC Calculations take a unique approach by combining pipelining with efficient resource utilization, setting them apart from traditional parallel CRC algorithms. In this approach, the CRC calculation process is divided into multiple stages, each handling a specific part of the calculation. This architecture enables the parallel processing of data bits, allowing multiple bits to be processed simultaneously across the pipeline stages. By breaking the calculation down into smaller steps, this design achieves higher throughput while minimizing resource consumption, making it particularly well-suited for FPGA implementations. Unlike conventional parallel methods, which often require extensive resources to store lookup tables or perform matrix-based computations, the pipelined approach distributes the workload more evenly, ensuring that the system runs efficiently without excessive resource usage. This results in not only a faster CRC calculation but also a more scalable design that can be adapted to different system sizes and performance requirements.

FPGA-based implementations have become increasingly important in the field of CRC calculation due to their flexibility, high performance, and ability to handle parallel operations efficiently. FPGAs offer a customizable environment where algorithms can be optimized for specific applications, making them an ideal platform for implementing custom CRC designs. The Pipelined CRC Calculations take full advantage of these FPGA capabilities by using the inherent parallelism of the FPGA to accelerate CRC computation. This design ensures that CRC computation is carried out with minimal latency, meeting the real-time requirements of high-speed communication systems. Additionally, the FPGA's reconfigurability allows for rapid prototyping and optimization, which is particularly useful for evolving communication protocols or custom error detection schemes.

In summary, while traditional CRC algorithms have been instrumental in providing reliable error detection for digital communication systems, their sequential processing nature presents limitations in terms of latency and throughput in high-speed applications. To address these shortcomings, the Pipelined CRC Calculations leverage the principles of pipelining and parallel processing to offer a more efficient and scalable solution. By breaking down the CRC calculation process into multiple stages and utilizing FPGA hardware to enable parallel processing, this approach increases throughput, reduces latency, and optimizes resource utilization. The Pipelined CRC Calculations provide a robust solution for high-speed communication systems, ensuring faster and more reliable error detection while making efficient use of available hardware resources. This advancement in CRC calculation represents a significant step forward, offering a scalable and low-latency solution to meet the ever-growing demands of modern communication systems.

III. Methodology

A. System Design and CRC Calculation Scheme

The Dynamic Pipelined CRC Calculation utilizes a pipelined architecture as showed in Fig. 1, which divides the CRC calculation process into multiple stages, each responsible for processing a portion of the input data. This approach enables parallel processing, allowing multiple parts of the data to be handled simultaneously, which significantly reduces the overall latency of the system. The input data is treated as a continuous stream of bits, with each stage of the pipeline contributing to the final CRC value by performing specific operations like shifting and XORing. The output from one stage is passed to the next, ensuring that the pipeline

continuously processes data without idle time, improving throughput and making the design highly efficient for real-time applications.

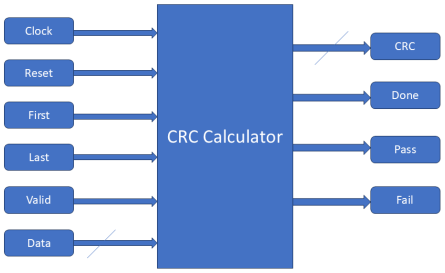

Fig. 1 System design

The CRC algorithm relies on polynomial division, where the input data is divided by a predefined generator polynomial. This division is performed bit by bit, and the remainder of the division becomes the CRC value, which is appended to the original data for error detection. In the dynamic pipelined design, each stage of the pipeline processes a portion of the polynomial division, with intermediate results passed from one stage to the next. This method ensures that the division is car- ried out incrementally across stages, allowing for parallelism and reducing the time required to compute the final CRC value.

By using this pipelined approach, the Dynamic Pipelined CRC Calculation can efficiently handle high-speed data streams while maintaining low latency. The intermediate re- sults that are passed between stages ensure that the CRC calculation progresses continuously, with each stage contribut- ing to the final checksum without waiting for the previous stage to complete its entire operation. The final CRC value, once computed, is appended to the data and serves as a checksum for error detection. This design not only accelerates the computation process but also makes the system suitable for applications where real-time performance and reliability are essential.

B. Data Frame Structure

The input data for the Dynamic Pipelined CRC Calculation is organized into discrete frames, each consisting of a fixed number of bits. These frames act as containers for the data being processed, and their fixed size ensures a predictable structure for the CRC calculation. The number of bits in each frame is typically determined based on the application and communication protocol requirements, ensuring that the data is handled efficiently by the system. These frames are processed one by one, but the pipelined architecture allows multiple frames to be processed in parallel, significantly improving the overall throughput of the system.

The pipelined architecture of the Dynamic Pipelined CRC Calculation ensures that while one frame is being processed by a given stage, other frames are simultaneously processed in different stages of the pipeline. This parallelism allows the system to maintain high throughput, as data is continuously flowing through the pipeline, with each stage handling different bits or portions of the data. This approach reduces the overall processing time, enabling faster CRC calculation and allowing the system to handle higher data rates.

Each stage of the pipeline performs a specific part of the CRC calculation process, such as shifting, XOR operations, and polynomial division. As data passes through these stages, intermediate results are generated and passed to the next stage. By the time the data reaches the final stage of the pipeline, all the necessary operations have been completed, and the final CRC value can be determined. The result of the CRC calculation is a single byte, which consists of 8 bits. This byte represents the CRC checksum for the frame, and it is appended to the original data frame.

The CRC checksum appended to the data frame serves as an error detection mechanism. When the data frame is received by the destination system, the CRC value can be recalculated using the same method. If the calculated CRC matches the appended value, the data is considered valid; otherwise, it indi- cates that an error has occurred during transmission, allowing the receiver to detect and handle the error appropriately.

By structuring the input data into frames and leveraging the dynamic pipelined architecture, the system can process large amounts of data efficiently and reliably, making it ideal for real-time applications in high-speed communication systems, embedded systems, and IoT devices.

C. Pipelined Architecture

The pipelined architecture of the Dynamic Pipelined CRC Calculation is a crucial element of its design, structured into multiple stages that each handle a specific part of the CRC calculation process as show cased in Fig. 2. This segmented approach breaks down the task into smaller, more manageable operations, enabling the system to handle the calculation efficiently. Each stage in the pipeline is dedicated to performing a particular operation, such as shifting, XORing, or applying other logic operations on the input data. This specialization ensures that the process is both modular and efficient, allowing each stage to focus on a specific aspect of the overall CRC calculation.

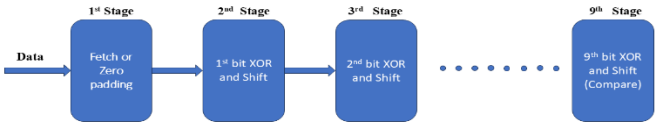

Fig.2 Pipelined Architecture

The stages are connected in a linear fashion, meaning the output from one stage becomes the input for the next. This arrangement allows for a continuous flow of data through the pipeline, where each subsequent stage builds upon the work of the previous one. As the data moves through each stage, the

operations become progressively more refined, ultimately leading to the final CRC value. This linear connection ensures that each stage operates in synchronization with the others, and the design maintains a streamlined, predictable processing flow, making it easier to manage and optimize the system's performance.

One of the key advantages of this pipelined architecture is the ability to process multiple data frames in parallel. While one frame is being processed in one stage, other frames can be processed simultaneously in other stages of the pipeline. This parallelism is a direct result of the segmented, linear architecture, where each stage works independently but in coordination with the others. This setup significantly increases the throughput of the system, as multiple frames are processed concurrently rather than sequentially. In addition, by dividing the process into smaller stages, the system experiences reduced latency, meaning that the time between input and the final CRC output is minimized. This makes the dynamic pipelined design ideal for high-speed applications, where real-time processing is crucial and data throughput is a top priority.

FPGA Implementation

The Dynamic Pipelined CRC Calculation is implemented on an FPGA (Field-Programmable Gate Array) platform using a hardware description language (HDL) such as Verilog or VHDL. FPGAs are chosen for their flexibility, allowing for the design and reconfiguration of hardware circuits to meet specific application requirements. The use of an HDL enables the description of hardware behavior at a high level of abstraction, allowing for easier and more efficient design, simulation, and testing of the pipelined CRC calculation system. Verilog or VHDL provides the necessary constructs to define the sequential and parallel logic operations within the pipeline, which are essential for efficiently computing the CRC in multiple stages.

FPGAs offer significant advantages in terms of perfor- mance and customization. They are capable of executing tasks in parallel, which aligns perfectly with the pipelined nature of the CRC calculation. Unlike traditional proces- sors, which execute instructions sequentially, FPGAs can simultaneously process multiple bits or stages of data, significantly accelerating the CRC calculation process. The flexibility of FPGAs also allows for optimization in terms of resource usage and performance. Designers can customize the architecture to balance the trade-off between throughput and resource consumption, ensuring that the system meets both performance and area constraints.

The implementation of the Dynamic Pipelined CRC Cal- culation on the FPGA platform is optimized for low re- source utilization, making it highly efficient for resource- constrained environments. In such environments, there may be strict limits on available memory, processing power, or physical area, and the FPGA allows for fine- tuned resource management. The pipelined design minimizes the need for extensive memory usage by using sequential, distributed computation across the stages, which reduces the overall footprint of the system. Additionally, by leveraging the parallelism inherit in the FPGA architecture, the system achieves high throughput without requiring a large number of hardware resources. This makes the implementation well-suited for embedded systems, IoT devices, or other applications where space and power are limited but performance is still critical.

I. Results and Discussion

The designed CRC-8 computation module was simulated to validate its functionality and correctness. The module was tested with an 8-bit input data stream using a 100 MHz clock and the CRC-8 polynomial ($x^8 + x^2 + x + 1$, represented as 0x07). The expected CRC value for the given test dataset was set to 0x02, which was used to verify the accuracy of the computed CRC output. The implementation followed a 9-stage pipeline approach, where each stage performed a shift operation and applied the polynomial division as required. The data was processed sequentially, ensuring efficient and structured execution of the CRC algorithm.

To control the data flow, the module utilized valid, first, and last signals. The valid signal ensured that only meaningful data was processed, while the first and last signals indicated the beginning and end of the data stream, respectively. A counter was also used to track the processing stages, ensuring correct alignment of the input data. The final CRC value was extracted from the ninth pipeline stage (reg_stage9[15:8]) after the required iterations. Once the entire data sequence was processed, the module asserted the done signal to indicate completion. The correctness of the computed CRC was verified by comparing it with the expected value, where the module generated a pass signal if the values matched and a failure signal otherwise.

The simulation was performed using a testbench that applied a 32-byte input dataset to the CRC module. Each byte of data was processed in sequential clock cycles, and the final CRC output was checked against the expected value. The results confirmed that the computed CRC value matched the expected result, thereby validating the correctness of the design. The module effectively identified CRC matches and mismatches, ensuring robust error detection. The pipeline-based implementation optimized the processing speed by distributing the computation across multiple clock cycles. The overall results demonstrate that the designed CRC-8 module functions correctly and can be effectively used for data integrity verification in practical applications.

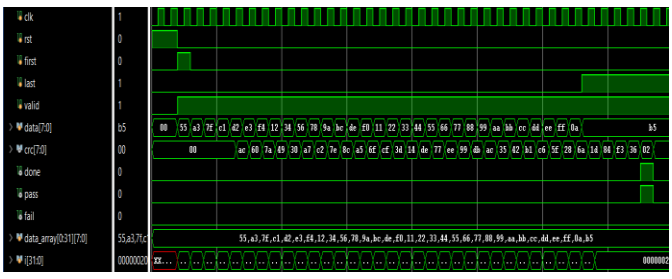

Fig. 3 Output waveform of CRC calculation

The given waveform represents the simulation output of the CRC-8 computation module implemented in Verilog. The CRC module processes an 8-bit input data stream, computes the cyclic redundancy check (CRC) using a polynomial division method, and verifies the computed CRC against an expected value (0x02). The testbench drives various control signals, including clock (clk), reset (rst), valid (valid), first (first), and last (last), which dictate the operation of the CRC pipeline.

At the start of the simulation, the reset (rst) signal is asserted high, initializing all internal registers to zero. Once reset is de-asserted, the first signal is briefly set, indicating the start of data processing. The valid signal remains high, ensuring that the module continuously processes incoming data bytes. The testbench provides a predefined sequence of 32 data values, as seen in the data [7:0] signal, including values like 0x55, 0xA3, 0x7F, and so on. These values are fed into the CRC computation pipeline, where each stage performs polynomial XOR division to generate the final CRC value.

The CRC output (crc[7:0]) evolves as data is processed through the pipeline. The CRC calculation is performed in multiple pipeline stages, where each stage shifts the data left and conditionally XORs it with the polynomial (0x07). The final computed CRC value is 0x02, which matches the expected result defined in the module. This correctness is confirmed by the pass signal, which is asserted at the end of the computation, indicating that the computed CRC is correct. Additionally, the done signal is asserted, marking the completion of the CRC check, while the fail signal remains low, meaning no errors were detected.

The waveform validates the efficient functioning of the CRC-8 pipeline. Each byte of data undergoes nine processing stages, ensuring an accurate CRC computation. The correct output confirms that the designed module successfully implements error detection and is reliable for data integrity verification applications. The structured pipeline approach ensures that the CRC computation is performed efficiently while maintaining accuracy.

CONCLUSION

The Dynamic Pipelined CRC Calculations represent a significant advancement in CRC calculation methods, providing a high-speed, efficient, and scalable solution for error detection in digital communication systems. By utilizing the principles of pipelining and parallel processing, the project effectively reduces latency and resource utilization, making it highly suitable for real-time applications in high-speed communication systems, embedded systems, and IoT devices. The FPGA implementation of the Dynamic Pipelined CRC Calculations highlights the potential of pipelined architectures to enhance the performance of error detection algorithms. The results indicate that the pipelined design achieves high throughput, low latency, and efficient resource utilization, positioning it as a robust solution for modern electronic systems. Future work on the Dynamic Pipelined CRC Calculations could involve integrating additional error detection and correction techniques, as well as expanding the pipelined architecture to accommodate other CRC algorithms. By continuing to build on the principles of pipelining and parallel processing, this project lays the foundation for future advancements in error detection methods, effectively addressing the challenges posed by increasingly complex and high-speed communication systems.

REFERENCES

[1] A. Sabharwal, P. Schniter, D. Guo, D. W. Bliss, S. Rangarajan, and R. Wichman, “Full-duplex wireless communication: Challenges, solutions, and future research directions,” *Proceedings of the IEEE*, vol. 102, no. 3, pp. 131–152, Mar. 2014.

[2] R. Young and A. Sharma, “I3C: Next Generation Interface for IoT and Automotive Applications,” in *IEEE Conference on Emerging Technologies*, 2019, pp. 223-228, doi: 10.1109/IEEECONF.2019.8392243.

[3] A. Adnan, M. Waheed, and M. Hamid, “Design and Implementation of an I2C Communication Protocol on FPGA,” in *International Conference on Communication Systems*, 2015, pp. 645-650, doi: 10.1109/IEEECONF.

2015.7428645.

[4] F. Bader, M. M. A. Zain, and J. Habibi, “Adaptive Full-Duplex Communication in Dense IoT Environments,” *IEEE Internet of Things Journal*, vol. 8, no. 3, pp. 1561-1575, Mar. 2021, doi: 10.1109/JIOT.2021.3069992.

[5] H. Wang, Y. Chen, and S. Xu, “Collision-Free Full-Duplex Communication Protocols for IoT Networks,” *IEEE Transactions on Vehicular Technology*, vol. 68, no. 2, pp. 1205-1218, Feb. 2019, doi: 10.1109/TVT.2019.2956789.

[6] A. Sahai, G. Patel, and A. Sabharwal, “Pushing the limits of full- duplex: Design and real-time implementation,” *IEEE Communications Magazine*, vol. 52, no. 6, pp. 112-119, Jun. 2014, doi: 10.1109/MCOM.2014.6829967.

[7] S. Dhawan and K. Gupta, “A Study on the Performance of Full-Duplex Communications in Wired Networks,” in *IEEE International Conference on Consumer Electronics (ICCE)*, 2020, pp. 543-548, doi: 10.1109/ICCE.2020.02436.

[8] A. Elsayed and M. Erol-Kantarci, “Survey on Full-Duplex Wireless MIMO and Network Coding: Brief History, Recent Advances, and Future Challenges,” *IEEE Access*, vol. 9, pp. 101782–101803, 2021.

[9] Xilinx, Inc. *Designing with FPGAs*. [Online]. Available: https://www.xilinx.com

[10] J. E. Smith and B. M. Foss, “A high-speed bidirectional communication protocol for embedded systems,” *IEEE Transactions on Industrial Electronics*, vol. 57, no. 3, pp. 895–905, 2010.

[11] Z. Tan and Y. Wen, “Efficient group communication in full-duplex wireless systems,” *IEEE Communications Letters*, vol. 19, no. 12, pp. 2206–2209, 2015.