electronics

MDPI

Article

An Efficient Parallel CRC Computing Method for High Bandwidth Networks and FPGA Implementation

Ling Zhang, Shanwei Ye, Zhuo Gou, Xuefei Yang, Qilin Dai, Fuqiang Wang and Yingcheng Lin *

School of Microelectronics and Communication Engineering, Chongqing University, Chongqing 401331, China
* Correspondence: linyc@cqu.edu.cn

Abstract: A cyclic redundancy check (CRC) is a widely used technique in data communication for detecting data transmission errors. However, existing FPGA-based parallel CRC hardware implementation schemes often face problems of excessive resource utilization and timing convergence difficulties in high-bandwidth networks. In addition, these problems are further exacerbated by the variable length of the end of the checksum data frame during data transmission. To address these challenges, this paper proposes a parallel CRC computation method based on precomputed seed values for bit-width normalization (named PSV-WN-CRC). The algorithm selects the corresponding primitive seed value according to the length of the data frame tail and converts the CRC computation with arbitrary bit-width to the CRC computation with fixed bit-width, thus adapting to the case of the indefinite length of the data frame tail. Based on this algorithm, this paper designs an efficient parallel CRC circuit on FPGA to reduce the consumption of resources. The experimental results show that the CRC algorithm and circuit proposed in this paper implemented on Virtex UltraScale+ FPGAs with 1024-bit wide CRC consumes only 5981 LUTs and achieves a maximum throughput of 392.2 Gbps. The method effectively reduces the resource consumption and improves the maximum throughput as compared to three advanced works.

Keywords: cyclic redundancy check; FPGA; parallel CRC; high-bandwidth network

Citation: Zhang, L.; Ye, S.; Gou, Z.; Yang, X.; Dai, Q.; Wang, F.; Lin, Y. An Efficient Parallel CRC Computing Method for High Bandwidth Networks and FPGA Implementation. *Electronics* **2024**, *13*, 4399. https://doi.org/10.3390/electronics13224399

Academic Editor: Alexander Barkalov

Received: 12 October 2024
Revised: 6 November 2024
Accepted: 8 November 2024
Published: 10 November 2024

1. Introduction

A cyclic redundancy check (CRC) [1–3] is a checksum technique for detecting errors in data transmission and is widely used in the field of data communication [4]. A checksum value is generated by performing a series of mathematical operations on a data packet and then that checksum value is appended to the data packet. The receiver performs the same calculations after receiving the data and compares the result with the received checksum value to detect whether an error has occurred in the transmission. In conventional serial CRC, the data enters the CRC computation unit for processing on a bit-by-bit basis, whereas parallel CRC increases the speed of computation by processing multiple bits of data at the same time, which improves the efficiency of the processing [2,3,5].

In 1961, Peterson et al. [6] introduced the concept of CRC, which is based on the principle that the data to be transmitted are considered as a polynomial, dividing the polynomial by a fixed generating polynomial, and then appending the remainder as a checksum value to the data. After receiving these data, the receiver performs the same division operation and compares the resulting remainder with the checksum value appended by the sender to detect the presence of errors. The implementation of two types of bit-by-bit serial CRC computation circuits is also given. In 1971, Patel [7] proposed a method of constructing a multi-channel CRC register capable of processing binary data in parallel, allowing parallel processing of binary data to generate CRC checksum codes. This multi-channel circuit offers greater advantages over conventional serial CRC circuits in terms of eliminating data buffering and increasing the speed of checksums. Sarwate [8], in 1988, proposed a lookup table-based method for CRC computation by processing 8 bits, i.e., one byte of

data at a time, which is more efficient than conventional parallel computation methods. In 1990, Albertengo et al. [9] proposed a hardware parallel CRC circuit design method based on digital systems theory and z-transforms, which allows the designer to derive logical equations for parallel CRC circuits for any generating polynomials and is a fundamental model for parallel CRC computation.

In recent years, as the demand for data communications and high-performance computing grows, programmable logic devices such as the Field Programmable Gate Array (FPGA) have been increasingly used in data transmission and network communications. FPGAs play a key role in a variety of application scenarios, including data centers [10–15], communication networks [16–20], video processing [21–25], and automated driving [26–30], due to their flexibility and parallel computing capabilities. Especially in high-bandwidth and low-latency applications, FPGAs provide a parallel computing platform capable of meeting real-time processing requirements, making them ideal for realizing efficient CRC checksum circuits.

However, as the future network bandwidth demand continues to increase to 100 G and above, in high-performance application scenarios [31–35] requiring low latency and high bandwidth, traditional FPGA-based parallel CRC hardware implementation schemes often face the challenges of excessive hardware resource utilization and difficult timing convergence. For example, in the pursuit of low-latency, high-bandwidth RDMA NICs (RNIC), the resource overhead and timing convergence faced by the RNIC hardware design have become important issues, and the parallel CRC module is the key module with large logic resource consumption and difficult timing convergence. Especially in data transmission, the problem of mismatch between the width of the transmitted data and the bit width of the data bus often occurs, and this problem usually appears at the end of the data packet, so it is also called the indefinite length of the end of the check data frame. At present, in order to solve this problem, a multi-path selection method is usually used. This method selects different computation paths based on the number of data bytes and requires different CRC computation logic for each case smaller than the parallel width. As the parallel width increases, the resources required for CRC computation increase significantly.

To address these challenges, this paper proposes an efficient parallel CRC computation method for high-bandwidth networks, which is named PSV-WN-CRC. In this paper, we design an efficient parallel CRC circuit based on this method and synthesize and implement it in Virtex UltraScale+ FPGAs, and we also experimentally test the implementation results for performance parameters such as resource consumption, maximum throughput, and maximum running clock frequency. The experimental results show that the proposed in this paper meets the requirements of stability and high performance while effectively reducing the consumption of logic resources by CRC circuits.

In the light of the above considerations, the main contributions of the work in this paper can be summarized as follows:

- Aiming at the problems of hardware CRC circuits, which are caused by the variable length of the checksum data frame tail and the difficulty of convergence in timing, this paper proposes a bit-width normalized CRC parallel computation method based on the precomputation of the original seed value, called PSV-WN-CRC. The computation method selects the corresponding primitive seed value according to the data frame tail length, and converts the arbitrary bit-width CRC computation into the fixed bit-width CRC computation, so as to effectively adapt to the case of variable data frame tail length.
- Based on the PSV-WN-CRC computation method, this paper designs an efficient parallel CRC circuit for FPGAs. The implementation and experiments are performed in Virtex UltraScale+ FPGAs. The experimental results show that the efficient parallel CRC circuit designed in this paper consumes only 5981 LUTs for realizing 1024-bit wide CRC computation and achieves a maximum throughput of 392.2 Gbps. Compared with three advanced works, this method effectively reduces resource consumption and improves the maximum throughput.

The paper is organized as follows: In Section 2, this paper presents some important pieces of literature in related research areas. In Section 3, the derivation process of the PSV-WN-CRC algorithm is described in detail, an example computation of a primitive seed and the circuit design of the PSV-WN-CRC algorithm are given. In Section 4, this paper performs performance tests in Virtex UltraScale+ FPGAs with different bus widths, compares the results with existing schemes, and tests the stability of the CRC module in a 100 G RNIC system. Section 5 is the conclusion section.

2. Related Work

With the emergence of various high-bandwidth networks and high-performance data transmission application areas, the data communication rate of each channel has increased significantly, and parallel CRC computation methods need to be further optimized. The main goal is to increase the maximum clock frequency while ensuring limited resource consumption, thus increasing the throughput of the circuit.

A software-based design framework for CRC generation algorithms is proposed in [36], where the "slicing-by-4" and "slicing-by-8" algorithms are designed to effectively improve CRC performance, but this method is not applicable to FPGAs.

A fully adaptive CRC accelerator based on a lookup table algorithm was designed on an FPGA in [37] to generate CRCs for any known CRC polynomials However, the CRC accelerator faces challenges in meeting the demands of high-performance applications with respect to maximum frequency and throughput.

A pipelined computation method for CRC checksums was proposed in [38] to address the issue of indefinite length at the end of the checksum data frame, allowing computation for data that do not span the full width of the input. However, this method still underperforms in terms of metrics like maximum operating frequency and throughput.

A parallel CRC circuit design proposed in [39] divides the data into multiple blocks, utilizing a lookup table to support CRC computation for each block. This method allows for CRC circuits with various input data lengths, parallel bit-widths, and generating polynomials; however, its performance is optimal only at 32-bit and 64-bit data widths, making it unsuitable for high-performance, high-bit-width applications.

A 64-bit parallel CRC architecture for high-speed applications is proposed in [40], based on F-matrices that generate 32nd-order polynomials, effectively reducing the number of cycles required to compute CRCs by 50%.

A CRC computing architecture for high-speed data transmission is proposed in [41], designed to achieve high throughput and low resource consumption by processing multiple independent packets simultaneously in each clock cycle.

The stride-by-5 algorithm is proposed in [42] to reduce FPGA resource consumption, along with a backtracking pipeline algorithm to address the zero-filling problem, enabling a low-cost, programmable FPGA-based parallel CRC scheme. However, this approach requires a large pipeline order.

The PQC-CRC method is proposed in [43] to address the issue of complementary zeros in the final loop of parallel CRC computation by employing the partial quotient compensation technique.

Kishore et al. proposed a lookup table-based segmented CRC-32 parallel architecture in [44], utilizing approximately 16 lookup tables, each 256 entries of 32 bits, to implement a 128-bit wide CRC computation. This design achieved a 10 Gbps throughput at a clock frequency of 83.33 MHz.

Das [45] proposed and implemented a CRC algorithm called Stride-x in 2023, aiming to improve computational speed and resource utilization efficiency through segmented computation. Cai et al. [46] investigated and implemented a typical lookup-table-based CRC algorithm in 2024 and designed a zero-padding strategy to support byte-multiple input data lengths. The above two approaches were tested and analyzed only below 256 bus bit widths and the performance of the circuit was not verified at high bus bit widths.

It is worth noting that many of the above hardware implementation methods are implemented using hardware description languages (HDLs), such as Verilog [47]. In FPGA programming, Verilog is a widely used hardware description language [48], especially suitable for designing high-performance parallel computing circuits such as CRC check circuits. Compared to higher-level programming methods such as HLS, Verilog allows designers to precisely control hardware resources at a lower level of abstraction, optimizing the utilization of logic resources and timing performance [49]. As a result, many high-performance applications choose to use Verilog for hardware development to take full advantage of the parallel processing capabilities of FPGAs [50,51].

Many of the above studies have effectively reduced the hardware resource consumption of CRC circuits and improved the data throughput, but the main contribution of most of the studies focuses on more efficient data processing through, e.g., pipelining, lookup tables, and other architectures. In contrast, this paper is dedicated to presenting an efficient parallel CRC computation method to improve resource utilization and clock frequency by simplifying the computational logic level. A detailed description of the PSV-WN-CRC algorithm can be found in Section 3 of this paper.

3. Proposed Work

Section 3 describes the efficient parallel CRC computation method and its hardware implementation architecture proposed in this paper. First, Section 3.1 discusses in detail the PSV-WN-CRC, which achieves the unified processing of different bit-width data by selecting appropriate raw seed values. Then, Section 3.2 describes the specific implementation scheme of the PSV-WN-CRC algorithm in hardware, including the circuit structure design and the functional decomposition of each module. This organizational structure aims to gradually demonstrate the complete process from algorithm design to hardware implementation, laying the foundation for subsequent experimental verification.

3.1. PSV-WN-CRC

The elaboration of PSV-WN-CRC consists of two parts. First, the basic calculation principle of PSV-WN-CRC is elaborated through mathematical derivation. Then, the calculation method of primitive seeds is provided, and the practical application of the method is illustrated with cases.

3.1.1. Basic Theory of the PSV-WN-CRC

CRC check is a redundancy code for detecting data errors based on cyclic codes, which belong to linear packet codes in channel coding. Let the input data be a binary sequence of length k, which is mapped into a binary sequence of length $n(n > k)$ after linear transformation, we call such a grouping code an (n, k) grouping code. According to the definition of generating polynomials for a cyclic code, if $\mathbf{g}(X)$ is an $(n-k)$-th degree polynomial and is a factorization of $(x^n + 1)$, then an (n, k)-cyclic code can be generated from $\mathbf{g}(X)$, and $\mathbf{g}(X)$ is said to be the generating polynomial for this cyclic code. After derivation, the iterative operation process of serial CRC can be represented by the Linear Feedback Shift Register (LFSR), whose circuit structure is shown in Figure 1. In the figure, m_i is the input data stream. g_i denotes the generating polynomial coefficients of CRC. r_i denotes the CRC checksum value. $\oplus$ is the "bitwise XOR" operation. The feedback mechanism within the LFSR determines which bits will be fed back to the inputs of the register based on the selected CRC polynomial setting. During each clock cycle, the state of the LFSR is updated based on the CRC polynomial and the input data bits. When all data bits have been processed, the final value in the LFSR becomes the generated CRC.

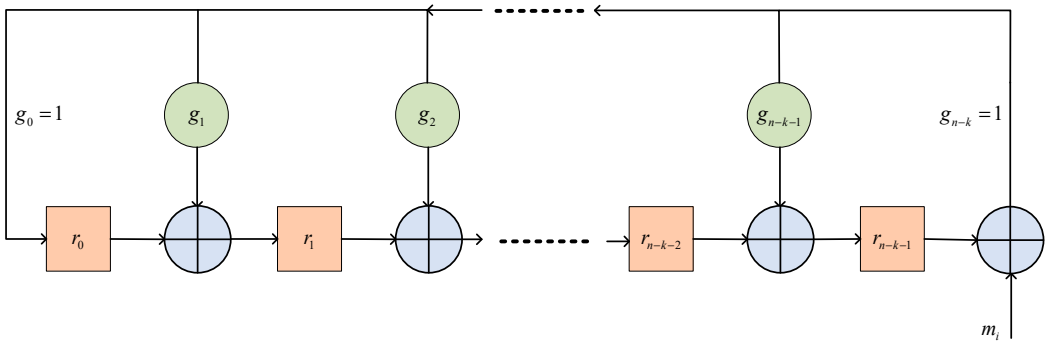


Figure 1. Serial CRC circuit structure based on linear feedback shift register circuit.

The parallel CRC algorithm is derived from the serial CRC. For generating polynomials up to $n-k$ times, the CRC checksum value of $\mathbf{u}_m = (m_{n-k}, m_{n-k+1}, \ldots, m_{n-2}, m_{n-1})^\top$ for input data of length k. We first define a column vector $\mathbf{r}_i = (r_{n-k-1}, r_{n-k-2}, \ldots, r_1, r_0)^\top$ of length $n-k$ with which to represent the CRC checksum value, where r_0 is the low bit of the CRC checksum value and r_{n-k-1} is the high bit of the CRC checksum value. Then, from Figure 1, the matrix expression of the CRC checksum value $\mathbf{r}_{n-k}$ when the last bit of data is entered is given in the form:

$$\mathbf{r}_{n-k} = \begin{bmatrix} r_{n-k-1}^{(n-k)} \\ r_{n-k-2}^{(n-k)} \\ \vdots \\ r_1^{(n-k)} \\ r_0^{(n-k)} \end{bmatrix} = \begin{bmatrix} g_{n-k-1} & 1 & 0 & \cdots & 0 \\ g_{n-k-2} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ g_1 & 0 & 0 & \cdots & 1 \\ 1 & 0 & 0 & \cdots & 0 \end{bmatrix} \otimes \left(\begin{bmatrix} r_{n-k-1}^{(n-k+1)} \\ r_{n-k-2}^{(n-k+1)} \\ \vdots \\ r_1^{(n-k+1)} \\ r_0^{(n-k+1)} \end{bmatrix} \oplus \begin{bmatrix} m_{n-k} \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} \right), \tag{1}$$

where we let the **F** matrix be:

$$\mathbf{F} = \begin{bmatrix} g_{n-k-1} & 1 & 0 & \cdots & 0 \\ g_{n-k-2} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ g_1 & 0 & 0 & \cdots & 1 \\ 1 & 0 & 0 & \cdots & 0 \end{bmatrix} = [\mathbf{G} | \frac{\mathbf{I}}{\mathbf{0}}] \tag{2}$$

Expressing Equation (1) in terms of the **F**-matrix and expressing $\mathbf{r}_{n-k-1}$ in the right equation in terms of $\mathbf{r}_{n-k-2}$ yields the following equation:

$$\mathbf{r}_{n-k} = \mathbf{F} \otimes [\mathbf{r}_{n-k+1} \oplus \begin{bmatrix} m_{n-k} \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}] = \mathbf{F} \otimes [\mathbf{F} \otimes \mathbf{r}_{n-k+2} \oplus \mathbf{F} \otimes \begin{bmatrix} m_{n-k+1} \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} \oplus \begin{bmatrix} m_{n-k} \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}] \tag{3}$$

Since the **F** matrix consists of polynomial coefficients and a unit matrix, it has the following properties:

$$\begin{bmatrix} m_{n-k} \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} = \mathbf{F} \otimes \begin{bmatrix} 0 \\ m_{n-k} \\ \vdots \\ 0 \\ 0 \end{bmatrix} \tag{4}$$

According to the properties of the F-matrix in Equation (4), Equation (3) can be expressed as:

$$\begin{aligned}\mathbf{r}_{n-k}(X) &= \mathbf{F} \otimes [\mathbf{F} \otimes \mathbf{r}_{n-k+2}(X) \oplus (\mathbf{F} \otimes \begin{bmatrix} m_{n-k+1} \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}) \oplus (\mathbf{F} \otimes \begin{bmatrix} 0 \\ m_{n-k} \\ \vdots \\ 0 \\ 0 \end{bmatrix})] \\ &= \mathbf{F} \otimes [\mathbf{F} \otimes \mathbf{r}_{n-k+2}(X) \oplus \mathbf{F} \otimes (\begin{bmatrix} m_{n-k+1} \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ m_{n-k} \\ \vdots \\ 0 \\ 0 \end{bmatrix})] \\ &= \mathbf{F}^2 \otimes [\mathbf{r}_{n-k+2}(X) \oplus \begin{bmatrix} m_{n-k+1} \\ m_{n-k} \\ \vdots \\ 0 \\ 0 \end{bmatrix}] \\ &= \mathbf{F}^{k-1} \otimes [\mathbf{r}_{n-1}(X) \oplus \begin{bmatrix} m_{n-2} \\ m_{n-3} \\ \vdots \\ m_{n-k} \\ 0 \end{bmatrix}]\end{aligned} \tag{5}$$

Similarly, Equation (5) can be continued by iterating $\mathbf{r}_{n-k}$ to an expression about $\mathbf{r}_{n-1}$. Since the CRC checksum adds the checksum's initial value $\mathbf{r}^{init}$ to the polynomial remainder operation as a way to improve the error detection ability of the CRC algorithm, the expression for $\mathbf{r}_{n-1}$ is:

$$\mathbf{r}_{n-1} = \mathbf{F} \otimes [\mathbf{r}^{init} \oplus \begin{bmatrix} m_{n-1} \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}] \tag{6}$$

Bringing Equation (6) into the iterative equation for $\mathbf{r}_{n-k}$, the expression for $\mathbf{r}_{n-k}$ with respect to the input data $\mathbf{u}_m$, the $\mathbf{F}$ matrix, and the initial value of the checksum $\mathbf{r}^{init}$ can finally be obtained as shown in the following equation:

$$\mathbf{r}_{n-k}(X) = \mathbf{F}^{(k)} \otimes [\mathbf{r}^{init} \oplus \begin{bmatrix} m_{n-1} \\ m_{n-2} \\ \vdots \\ m_{n-k+1} \\ m_{n-k} \end{bmatrix}] = \mathbf{F}^{(k)} \otimes [\mathbf{r}^{init} \oplus \mathbf{u}_m] \tag{7}$$

When the length of the data is less than or equal to the highest power of the polynomial, i.e., $k \leq r$, the expression for $\mathbf{F}^{(k)}$ is:

$$\mathbf{F}^{(k)} = \mathbf{F}^k = [\mathbf{F}^{k-1}\mathbf{G}\ \mathbf{F}^{k-2}\mathbf{G}\ \cdots\ \mathbf{F}\mathbf{G}\ \mathbf{G} \Big| \frac{\mathbf{I}}{\mathbf{0}}], \tag{8}$$

the expression for $\mathbf{r}_{n-k}$ at $k \leq r$ can be expressed as:

$$\mathbf{r}_{n-k} = \mathbf{F}^k \otimes [\mathbf{r}^{init} \oplus \mathbf{u}_m] \tag{9}$$

When the length of the data is greater than the highest power of the polynomial, i.e., $k > r$, the $\mathbf{F}^{(k)}$ is an $r \times k$ matrix and can be expressed as follows:

$$\mathbf{F}^{(k)} = [\mathbf{F}^{k-1}\mathbf{G}\ \mathbf{F}^{k-2}\mathbf{G}\ \cdots\ \mathbf{F}\mathbf{G}\ \mathbf{G}], \tag{10}$$

the expression for $\mathbf{r}_{n-k}$ at $k > r$ can be expressed as:

$$\begin{aligned} \mathbf{r}_{n-k} =& [\mathbf{F}^{k-1}\mathbf{G}\ \mathbf{F}^{k-2}\mathbf{G}\ \cdots\ \mathbf{F}\mathbf{G}\ \mathbf{G}] \otimes [\begin{bmatrix} \mathbf{r}^{init} \\ \mathbf{0}^{(k-r)\times 1} \end{bmatrix} \oplus \mathbf{u}_m] \\ =& \mathbf{F}^{(k)} \otimes [\begin{bmatrix} \mathbf{r}^{init} \\ \mathbf{0}^{(k-r)\times 1} \end{bmatrix} \oplus \mathbf{u}_m] \end{aligned}, \tag{11}$$

based on the input CRC checksum polynomial $\mathbf{G}$ and the data length k, $\mathbf{F}^{(k)}$ can be calculated, which can be brought into Equation (11) to obtain an expression for the XOR of the CRC checksum value $\mathbf{r}_{n-k}$ with respect to the initial value $\mathbf{r}^{init}$ and the input data $\mathbf{u}_m$. A circuit to perform this XOR operation can be designed in the hardware. In this way, each bit of the CRC check digit can be calculated in parallel, realizing the process of parallel CRC calculation.

A further derivation of the parallel CRC computation is carried out next. The derivation process reveals that the data can be shifted by utilizing the property of the unit matrix in **F** matrix. When the length of the data is greater than the highest power of the polynomial, i.e., $k > r$, the data can be shifted by complementary zeros by means of a matrix consisting of a unit matrix and polynomial coefficients of a similar form to the **F**-matrix, as shown in the following equation:

$$\begin{bmatrix} m_{k-1} \\ m_{k-2} \\ \vdots \\ m_1 \\ m_0 \end{bmatrix} = \begin{bmatrix} g_{r-1} & 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & & \vdots \\ g_0 & 0 & \cdots & 1 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} 0 \\ m_{k-1} \\ m_{k-2} \\ \vdots \\ m_1 \\ m_0 \end{bmatrix} = \begin{bmatrix} \mathbf{G} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} 0 \\ m_{k-1} \\ m_{k-2} \\ \vdots \\ m_1 \\ m_0 \end{bmatrix}, \tag{12}$$

continuing with the shift leads to equation:

$$\begin{bmatrix} m_{k-1} \\ m_{k-2} \\ \vdots \\ m_1 \\ m_0 \end{bmatrix} = \begin{bmatrix} \mathbf{FG} & \mathbf{G} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ m_{k-1} \\ m_{k-2} \\ \vdots \\ m_1 \\ m_0 \end{bmatrix} = \begin{bmatrix} \mathbf{F}^{a-1}\mathbf{G} & \mathbf{F}^{a-2}\mathbf{G} & \cdots & \mathbf{G} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} 0^{(a\times 1)} \\ m_{k-1} \\ m_{k-2} \\ \vdots \\ m_1 \\ m_0 \end{bmatrix} \tag{13}$$

With the above data shifting method, it is possible to convert the formula for the data length of k parallel CRC. Since the data length is generally much larger than the check digit length in the actual CRC checking process, it is not discussed in detail in the case of $k \leq r$. When $k > r$, Equation (11) can be expressed as:

$$\begin{aligned} \mathbf{r}_{n-k} =& \mathbf{F}^{(k)} \otimes [\begin{bmatrix} \mathbf{r}^{init} \\ \mathbf{0}^{(k-r)\times 1} \end{bmatrix} \oplus \mathbf{u}_m] \\ =& \mathbf{F}^{(k)} \otimes [\begin{bmatrix} \mathbf{F}^{a-1}\mathbf{G} & \mathbf{F}^{a-2}\mathbf{G} & \cdots & \mathbf{G} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix} \otimes (\begin{bmatrix} \mathbf{0}^{(a\times 1)} \\ \mathbf{r}^{init} \\ \mathbf{0}^{(k-r)\times 1} \end{bmatrix} \oplus \begin{bmatrix} \mathbf{0}^{(a\times 1)} \\ \mathbf{u}_m \end{bmatrix})] \end{aligned}, \tag{14}$$

where $\mathbf{F}^{(k)}$ is an $r \times k$ matrix and the shift matrix is a $k \times (k+a)$ matrix, multiplying these two matrices gives the following equation:

$$\begin{aligned}\mathbf{r}_{n-k} =& [\mathbf{F}^{k-1}\mathbf{G} \ \cdots \ \mathbf{FG} \ \mathbf{G}] \otimes \begin{bmatrix} \mathbf{F}^{a-1}\mathbf{G} & \cdots & \mathbf{G} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix} \otimes \left(\begin{bmatrix} \mathbf{0}^{(a\times 1)} \\ \mathbf{r}^{init} \\ \mathbf{0}^{(k-r)\times 1} \end{bmatrix} \oplus \begin{bmatrix} \mathbf{0}^{(a\times 1)} \\ \mathbf{u}_m \end{bmatrix} \right) \\ =& [\mathbf{F}^{k+a-1}\mathbf{G} \ \cdots \ \mathbf{FG} \ \mathbf{G}] \otimes \left(\begin{bmatrix} \mathbf{0}^{(a\times 1)} \\ \mathbf{r}^{init} \\ \mathbf{0}^{(k-r)\times 1} \end{bmatrix} \oplus \begin{bmatrix} \mathbf{0}^{(a\times 1)} \\ \mathbf{u}_m \end{bmatrix} \right) \\ =& \mathbf{F}^{(k+a)} \otimes \left(\begin{bmatrix} \mathbf{0}^{(a\times 1)} \\ \mathbf{r}^{init} \\ \mathbf{0}^{(k-r)\times 1} \end{bmatrix} \oplus \begin{bmatrix} \mathbf{0}^{(a\times 1)} \\ \mathbf{u}_m \end{bmatrix} \right)\end{aligned} \tag{15}$$

As can be seen from Equation (15), the matrix $\mathbf{F}^{(k)}$ of the CRC checksum value operation expression is converted to matrix $\mathbf{F}^{(k+a)}$ by the shift operation of the $\mathbf{F}$ matrix, which enables the conversion of the parallel CRC circuit with an input data bit width of k to a parallel CRC circuit with an input data bit width of $k+a$. This conversion method provides an effective solution to the problem of variable length at the end of the data frame encountered in CRC checksums. Observing Equation (15), it is possible to XOR two sets of 0 matrices for $k+a$ in the last equation, and then using the distributive law, two expressions representing the CRC operation can be obtained as follows:

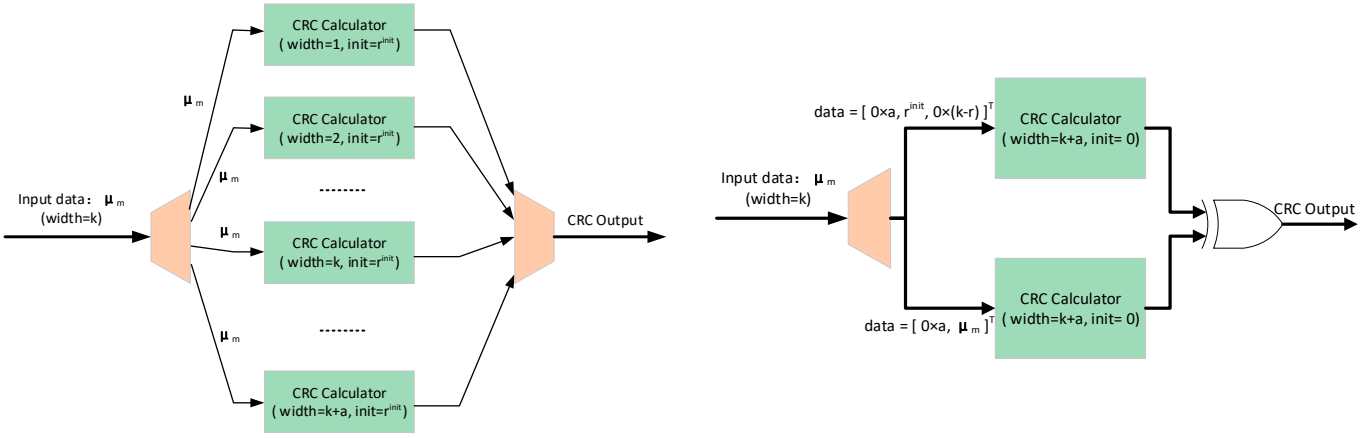
$$\begin{aligned}\mathbf{r}_{n-k} =& \mathbf{F}^{(k+a)} \otimes \left(\begin{bmatrix} \mathbf{0}^{(a\times 1)} \\ \mathbf{r}^{init} \\ \mathbf{0}^{(k-r)\times 1} \end{bmatrix} \oplus \mathbf{0}^{(k+a)\times 1} \oplus \mathbf{0}^{(k+a)\times 1} \oplus \begin{bmatrix} \mathbf{0}^{(a\times 1)} \\ \mathbf{u}_m \end{bmatrix} \right) \\ =& \mathbf{F}^{(k+a)} \otimes \left(\mathbf{0}^{(k+a)\times 1} \oplus \begin{bmatrix} \mathbf{0}^{(a\times 1)} \\ \mathbf{r}^{init} \\ \mathbf{0}^{(k-r)\times 1} \end{bmatrix} \right) \oplus \mathbf{F}^{(k+a)} \otimes \left(\mathbf{0}^{(k+a)\times 1} \oplus \begin{bmatrix} \mathbf{0}^{(a\times 1)} \\ \mathbf{u}_m \end{bmatrix} \right)\end{aligned}, \tag{16}$$

as can be seen from Equation (16), the CRC checksum value can be expressed as the XOR result of two CRC checksum values with a checksum initial value of zero. The above equation can be expressed in the CRC checksum formula as:

$$\begin{aligned}&\mathbf{r}_{n-k} = CRC^{(k)}(init = \mathbf{r}^{init}, data = \mathbf{u}_m) \\ &= CRC^{(k+a)}(init = \mathbf{0}, data = \begin{bmatrix} \mathbf{0}^{(a\times 1)} \\ \mathbf{r}^{init} \\ \mathbf{0}^{(k-r)\times 1} \end{bmatrix}) \oplus CRC^{(k+a)}(init = \mathbf{0}, data = \begin{bmatrix} \mathbf{0}^{(a\times 1)} \\ \mathbf{u}_m \end{bmatrix})\end{aligned} \tag{17}$$

As can be seen, Equation (17) successfully converts the CRC checksum value of the input data bit width of k to the XOR operation of the CRC checksum value of two input data bit widths of $k+a$. The before and after conversion structure of the CRC calculation is shown in Figure 2. The traditional multi-path selection method requires the design of several CRC calculators with different widths according to different data lengths, as shown in Figure 2a. After the conversion, only two CRC calculators with the same width are needed to complete the CRC calculation, as shown in Figure 2b. Moreover, since $\mathbf{r}^{init}$ in Equation (17) is known, the result of the first term of the XOR operation can be calculated in advance of the CRC checksum value of this term according to the different cases of the length of the end of the frame when the circuit is designed, and the result of the first term of the XOR operation thereof is referred to as the primitive seed in this paper. The CRC calculator at the top of the picture in Figure 2b can be pre-calculated in advance. Therefore, the CRC calculator above in the actual circuit would be designed as a data selector that selects the appropriate primitive seed based on the width of the input data. In summary, this method can calculate the CRC checksum value of a packet of arbitrary bit width with

only one CRC calculation circuit with a fixed input data bit width, which effectively reduces hardware resource consumption.



(**a**) (**b**)

Figure 2. Structure before and after conversion of CRC calculations: (**a**) traditional multi-path selection method; (**b**) proposed PSV-WN-CRC method.

3.1.2. Calculation of Primitive Seeds

Since the data of modern data communication systems and protocols are usually aligned in 4 bytes, an example of raw seed calculation for a CRC circuit with a bus transmission bit width of 64 bytes and a minimum data unit of 4 bytes is given here. According to Section 3.1.1, the CRC32 expression for an arbitrary data length of n is shown in Equation (18). This equation represents the CRC32 checksum value for an input data bit width of n and a checksum initial value of 32′hFFFFFFFF. Equation (18) can be converted to the XOR operation of Equations (19) and (20) according to Equation (17).

$$CRC32^{(n)}(init = 32\prime hFFFFFFFF, m = \mathbf{u}_m) \tag{18}$$

$$CRC32^{(512)}(init = 0, m = \{32\prime hFFFFFFFF, (n-32)\prime b0\}) \tag{19}$$

$$CRC32^{(512)}(init = 0, m = \{(512-n)\prime b0, \mathbf{u}_m\}) \tag{20}$$

where Equation (19) is the CRC checksum value for a bus width of 512, a checksum initial value of 0, and input data of 32′hFFFFFFFF low complementary zeros. Equation (20) is the CRC checksum value for a bus width of 512, a checksum initial value of 0, and input data as the original data. Since the initial value of the CRC calculation is 0, the zero in the high bit of the parity data will not affect the calculation result. Therefore, after passing the conversion formula, the data of any length can be complemented with high bits of zero to 512 bits. One reason for this is that CRC algorithms are designed to be transparent to high consecutive zeros at an initial value of 0, meaning that they do not change the final result of the CRC calculation. This property comes from the mathematical definition of CRC, which treats the input data as a polynomial and performs a modulo-division operation on that polynomial. In this process, high zeros are simply the higher terms of the polynomial that are not set, and do not change the coefficients of the polynomial, nor do they affect the final remainder. The number of complementary zeros will determine how many zeros need to be complemented in the lower bits of 32′hFFFFFFFF in Equation (19), so the result of Equation (19), the original seed, can be calculated ahead of time according to the different end-frame data lengths.

Since the smallest data unit is 4 bytes, there are 16 cases of tail data length in a CRC circuit with a bus width of 64 bytes, i.e., 0, 4, 8, 12, . . ., 60 bytes. We need to calculate the primitive seed corresponding to these 16 cases in advance. The high complementary zero

length y of the first data is determined from the tail frame data length x, which results in the length of the initial value 32′hFFFFFFFF low complementary zero z. This is then substituted into Equation (19) to compute the result of the primitive seed. The primitive seed for different tail frame data lengths is shown in Table 1.

Table 1. Primitive seed for different data frame tail lengths.

Data Frame Tail Lengths [1]	Data High Complementary Zero Length [2]	Initial Value Low Complementary Zero Length [3]	Primitive Seed (Hex)
0	0	No zero-completion operation required [4]	FFFFFFFF
4	60	64 − 60 − 4 = 0	C704DD7B
8	56	64 − 56 − 4 = 4	6904BB59
12	52	64 − 52 − 4 = 8	099C5421
16	48	64 − 48 − 4 = 12	552D22C8
20	44	64 − 44 − 4 = 16	4E26540F
24	40	64 − 40 − 4 = 20	FBAC7C3A
28	36	64 − 36 − 4 = 24	6811F1FE
32	32	64 − 32 − 4 = 28	4A55AF67
36	28	64 − 28 − 4 = 32	54B292A9
40	24	64 − 24 − 4 = 36	7243C868
44	20	64 − 20 − 4 = 40	C799DB3E
48	16	64 − 16 − 4 = 44	5632EEB0
52	12	64 − 12 − 4 = 48	F20F2BCC
56	8	64 − 8 − 4 = 52	6D5AEC34
60	4	64 − 4 − 4 = 56	EF6EB7DF

[1–3] The terms indicated in footer 1, 2, and 3 are described in the text as x, y, and z, respectively. Their units are bytes. [4] This case means that the input data width is an integer multiple of the bus transmission width, and no zero-complement operation is required.

The calculation of the high complementary zero length of the data header in Table 1 is obtained by the remaining length of the tail data length with respect to the data bus bit width of 64 bytes, i.e., $y = 64 - x$. The low complementary zero length of the initial value is calculated by subtracting the high complementary zero length of the data header from the data bus bit width of 64 bytes and then subtracting the bit width of the initial value itself of 4 bytes, i.e., $z = 64 - y - 4$. The low complementary zero length of the initial value is then calculated by substituting it into Equation (20) to arrive at the primitive seed. One of the more special cases is when the length of the trailing data is equal to 0 bytes. This case represents an input data width that is an integer multiple of the bus transmission width, without the need for a complementary zero operation, whose original seed is 32′hFFFFFFFF.

3.2. PSV-WN-CRC Hardware Implementation

The hardware implementation part consists of two parts: data preprocessing and CRC checksum circuit structure. Based on the previously proposed computational method, the preprocessing operation of the input data is first introduced to prepare for the subsequent checksum. Then the structure of the CRC hardware checksum circuit is described, showing the key steps of hardware implementation of PSV-WN-CRC.

3.2.1. Data Preprocessing

After the calculation method of PSV-WN-CRC is clarified, the input data stream needs to be processed with high complementary zeros and shifts according to the bus width, and the processing of the data stream by the CRC circuit module is shown in Figure 3. Firstly, the data packet is zero-complemented, shifted and spliced according to the data length at the end of the frame, and the effective data length of the spliced *CRC_data* is the bus transmission bit width for the rest of the cycles, except for the first cycle which requires high zero-complementing.

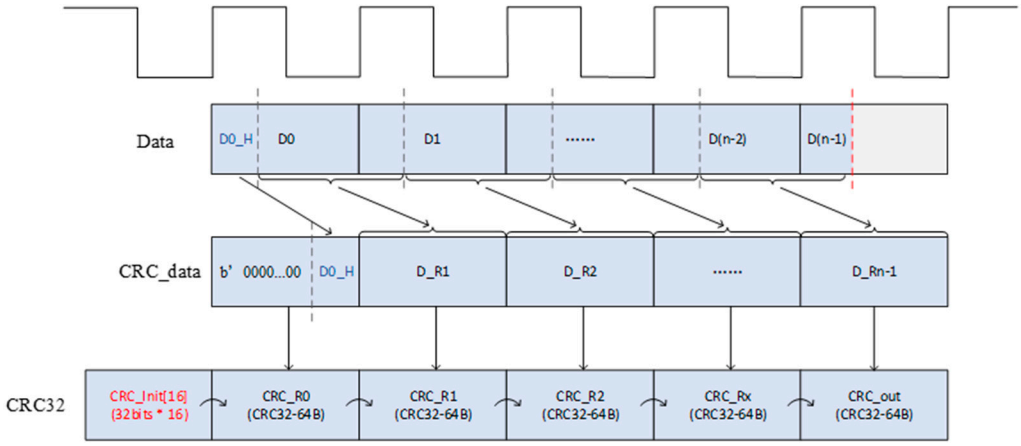


Figure 3. CRC processing timing for variable data lengths.

3.2.2. CRC Circuit Structure

Figure 4 illustrates the structure of the PSV-WN-CRC calculation circuit for a bus transmission bit width of 64 bytes. The specific hardware implementation of this method is written in Verilog language. The hardware implementation of this method is written in the Verilog language, and the input and output interfaces of the circuit follow the AXI-Stream protocol, including the *axis_tvalid*, *axis_tready*, *axis_tdata*, *axis_tlast*, and *axis_tuser* signal interfaces. The *axis_tvalid* signal is used to indicate whether the current data is valid or not, while the *axis_tready* signal indicates whether the receiver is ready to receive the data, and the data will only be transmitted when both *axis_tvalid* and *axis_tready* are high. The *axis_tdata* is responsible for transmitting the data body, i.e., the data that participates in the CRC checksum. And the *axis_tlast* is used to identify the end of the data stream, helping the circuit to recognize the end position of the packet. The *axis_tuser* signal carries additional information about the packet to facilitate data parsing. The entire CRC circuit consists of several functional modules, including a data parsing circuit module, a left and right shift circuit module, a data splicing circuit module, a CRC calculation circuit module, a raw seed selection circuit module, and a CRC data insertion circuit module. This design, based on the AXI-Stream interface, enables the circuit to efficiently process high-speed data streams and realize real-time CRC checksum calculations.

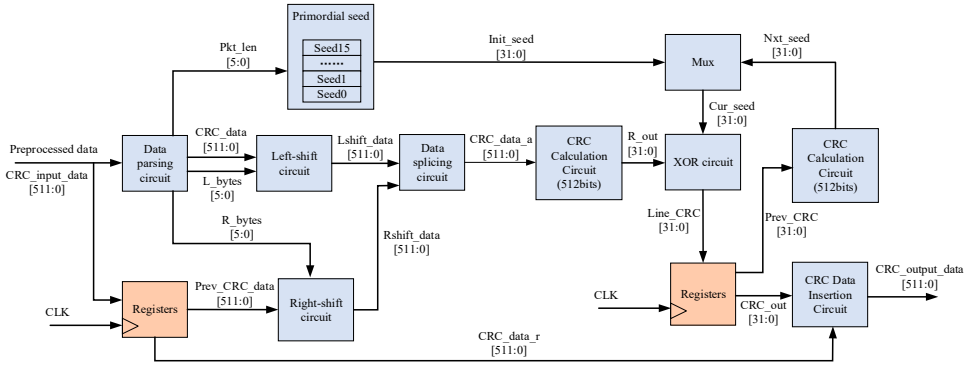


Figure 4. CRC circuit structure block diagram.

The data parsing circuit module first extracts the effective packet length, *Pkt_len*, from the packet, which is one of the key parameters for the CRC calculation. Based on *Pkt_len*, the module calculates the number of bytes that need to be shifted left bytes *(L _bytes)* and the number of bytes that need to be shifted right bytes *(R_bytes)* for each cycle of data. These values are used to adjust the splice position of the data in each clock cycle to ensure that the data can be aligned correctly for CRC calculation.

For the left-shift and right-shift circuit modules, because the network data handled by the hardware is the high data stored in the low address and the low data stored in the high address, which belongs to the big-end byte order, and is different from the small-end byte order of the human habitual way of reading and writing numerical values, the data that needs to be shifted right in the schematic diagram actually needs to be shifted left in the hardware. Therefore, the left-shift and right-shift operations are crucial to the adjustment of the data position. For the input data, on the one hand, it is necessary to go through the left shift circuit to obtain the left side data calculated by CRC in this cycle, and on the other hand, it is necessary to register a cycle through the register, and in the next cycle, it is necessary to go through the right shift circuit to obtain the right side data calculated by CRC in the next cycle.

The data splicing circuit module splices the left side data of this cycle with the right side data of the previous cycle to obtain the 512-bit CRC calculation data, in which the right side data is all zeros when it is in the first cycle of the packet transmission, i.e., the high bits of the data are complemented with zeros. The data to be computed is passed through a matrix-transformed CRC calculation circuit module to obtain the initial parity value *R_out* of the input data. The CRC calculation circuit module, on the other hand, is implemented in parallel through a LFSR in order to process a 512-bit data stream and generate a 32-bit checksum code.

The primitive seed selection circuit module is designed to act on the first cycle of data transmission to select the appropriate primitive seed value based on *Pkt_len*. Sixteen different raw seed values are pre-stored in the circuit to accommodate different packet lengths. The selected seed value is then differentiated from the initial checksum value of the current cycle, *R_out*, to obtain the initial CRC calculation result, *Line_CRC*, which is stored in a register to provide the basis for calculations in subsequent cycles.

In the intermediate cycle of data transmission, the CRC calculation value *Prev_CRC* of the previous cycle is used as the initial value for the current cycle. This value is then calculated by the CRC calculation circuit module to obtain a new seed *Nxt_seed*. this seed is selected as the current cycle's arithmetic seed *Cur_seed* by means of a selector, which is iso-orthogonal to the current cycle's preliminary checksum value, *R_out*, to obtain the updated CRC value *Line_CRC*. this method allows the circuit to continuously accumulate and update the CRC value during each clock cycle. This method allows the circuit to accumulate and update the CRC value in each clock cycle.

The CRC data insertion circuit module acts on the last cycle of data transmission, and the module passes the final CRC calculation, *CRC_out*, through the CRC Data Insertion Module to perform a bit-flip operation (i.e., XOR the checksum value with 32'hFFFFFFFF) and byte ordering to conform to the big-end format. The bit-flip operation is designed to prevent the receiver from adding or subtracting zeros to or from the end with the checksum value remaining unchanged, making it impossible to detect transmission errors. After bit flipping and byte order adjustment, the final CRC value is written into the ICRC field of the packet.

The CRC circuit module on the receiving side follows the same calculation process as the circuit on the sending side. Because the CRC field undergoes a bit-flip operation at the sender side, the final calculated residual will be a known constant value called residual or residue value if the received data is error-free. The process of calculating the residual value at the receiver side can be illustrated by Equation (21), where X is the bit position operator that serves to represent the position of its corresponding bit in the data:

$$\begin{aligned}(\mathbf{u}_m(X)X^r + \overline{\mathbf{R}})\mathrm{mod}\mathbf{G} &= (\mathbf{u}_m(X)X^r + X^{r-1} + \cdots + X + 1 - \mathbf{R})\mathrm{mod}\mathbf{G} \\ &= (\mathbf{u}_m(X)X^r + \mathbf{R} + (X^{r-1} + \cdots + X + 1))\mathrm{mod}\mathbf{G}'\end{aligned} \tag{21}$$

The inverse code is expressed in Equation (21). Also, because the checksum data has the following properties:

$$(\mathbf{u}_m(X)X^r + \mathbf{R})\mathrm{mod}\mathbf{G} = 0, \tag{22}$$

the expression for the residual value at the receiver side is:

$$(\mathbf{u}_m(X)X^r + \overline{\mathbf{R}})\mathrm{mod}\mathbf{G} = (X^{r-1} + \cdots + X + 1)\mathrm{mod}\mathbf{G} \tag{23}$$

For the polynomial 32′h04C11DB7 of CRC32, the residual value can be calculated from Equation (23) as 32′hC704DD7B. Since the receiver side can be exemplified with the same CRC circuit module as the sender side, the residual value is adjusted by the bit-flip and byte-order adjustment to obtain the final checksum value of 32′h2144DF1C. i.e., when the final checksum value calculated at the receiving end is 32′h2144DF1C, it means that the received data have no errors.

4. Experimental Section

In order to verify the effectiveness and efficiency of the proposed PSV-WN-CRC method, functional and performance tests of the computational method and hardware circuit are conducted in this paper. The functional tests mainly examine the effectiveness of the algorithm and circuit, while the performance tests focus on comparing PSV-WN-CRC with existing state-of-the-art algorithms to highlight its performance advantages.

4.1. Functional Test

Figure 5 shows the simulated waveform of the parallel CRC module on the transmitter side. By inputting the test data into the CRC calculation tool we were able to determine that the checksum value of the test data after preprocessing and output processing is 32′hCF5B5075. This checksum value not only matches the CRC checksum value in the simulation of Figure 5 but also matches the CRC field added to the output data, which verifies the correct functioning of the transmitter logic of the parallel CRC circuit module. Therefore, the logic function of the parallel CRC circuit module is correct.

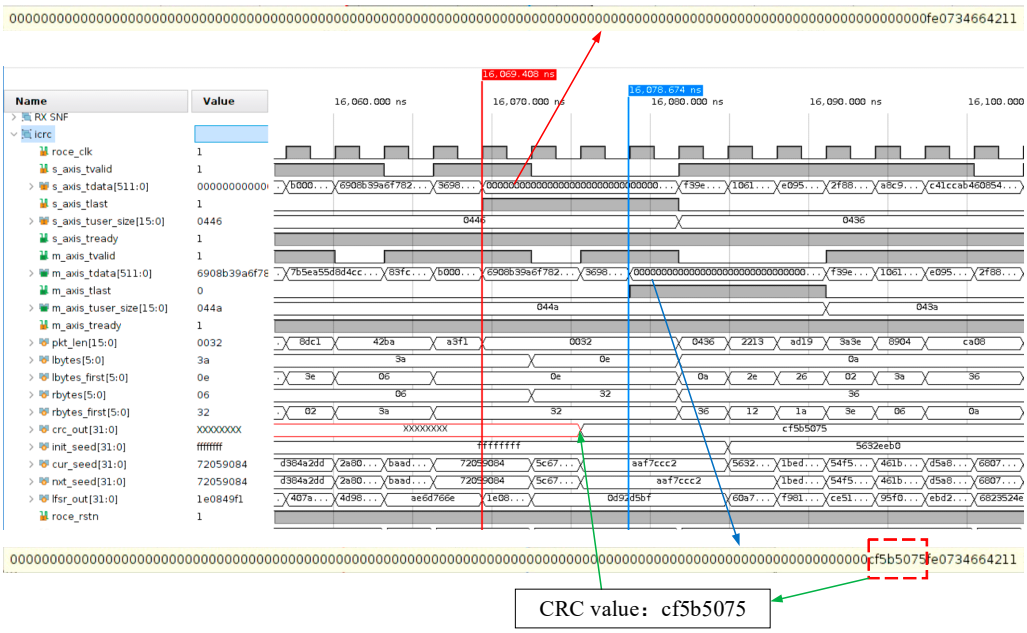


Figure 5. Simulation diagram of the transmitter CRC circuit module.

Figure 6 shows the simulated waveforms of the parallel CRC module at the receiver side. The test data are the data outputs from the CRC module at the transmitter side, which were verified. The simulation shows that the final CRC check value is 32′h2144DF1C, which corresponds to the residual value of CRC32 before output processing, 32′hC704DD7B, thus verifying the correct function of the parallel CRC circuit module's received logic.

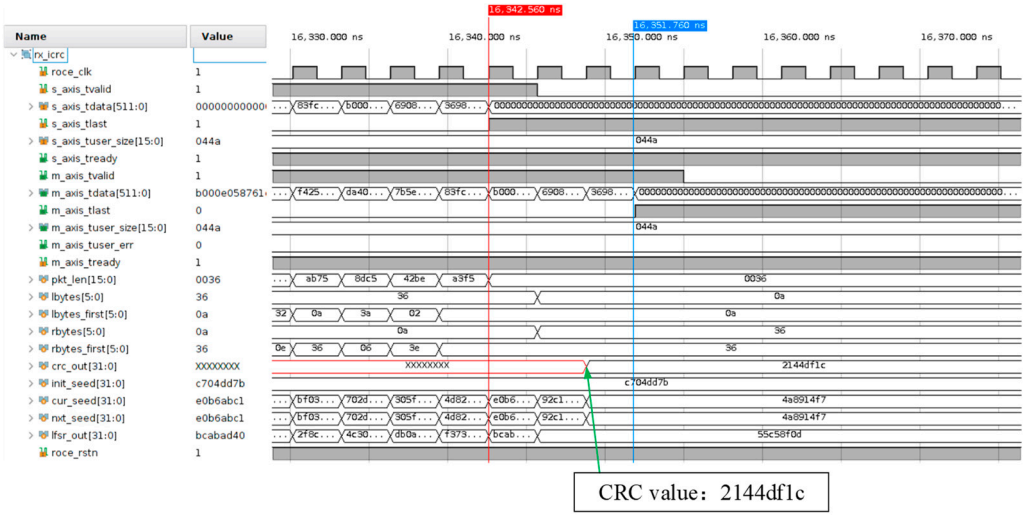


Figure 6. Simulation diagram of the receiver end CRC circuit module.

4.2. Performance Test

In this section, we synthesize and implement the CRC-32 circuit based on the proposed PSV-WN-CRC and measure the resource consumption, maximum operating frequency, and maximum throughput of the circuit at different data bus widths (w = 64, 128, 256, 512, 1024). The experimental results are also compared with papers [41,43,46] and traditional multi-path selection methods. The method proposed in this paper, along with the above three methods, uses Virtex UltraScale+ XCVU7P FPGAs to synthesize and implement, and the resource consumption and maximum operating frequency of the circuit is checked through the resource consumption report and timing analysis report generated by the EDA tool Vivado, which results in the maximum throughput. The version of Vivado is 2020.2. Finally, we apply the algorithm to a 100 G-class RNIC system built on a Xilinx U280 board for stability testing.

Measurements of the resource consumption, maximum operating frequency, and maximum throughput of the circuit for various CRC circuit implementation methods are shown in Table 2. In terms of resource consumption, corresponding to different data bus widths, the resource consumption of the PSV-WN-CRC circuits proposed in this paper is 510, 818, 1550, 3519, and 5981 LUTs, respectively. For the other four architectures, only the architectures in [41] are supported for implementation in large bus widths (w = 512, 1024). The resource consumption of PSV-WN-CRC proposed in this paper is 26.8–39.7% lower than that of the architectures in [41]; while the architectures in [43,46] as well as the traditional multi-path selection implementations perform better only when the bus width is small and are not suitable for applications with large bus widths. At small bus widths (w = 64, 128, 256), the resource consumption of PSV-WN-CRC is 39.7–71.1% lower than that of the architectures in [43], and especially at a bus width of 256, the use of LUTs is reduced by 71.1% compared to the paper [43]. At small bus widths (w = 64, 128, 256), PSV-WN-CRC consumes on average 86.7% less resources than traditional multi-path selection methods. Also, the graph shows that the resource consumption of architectures in [46] is slightly lower than that of PSV-WN-CRC proposed in this paper, but the maximum throughput of architectures in [46] is much lower than that of PSV-WN-CRC. The comparison of maximum throughput and indexes (Max Throughput/Resource Consumption) is given below.

In terms of maximum operating frequency and maximum throughput, the experiment analyzes the maximum operating frequency of the circuit based on the timing report. Then, the maximum throughput is obtained by converting according to the bus width. Corresponding to different data bus widths, the maximum operating frequencies achievable by the PSV-WN-CRC circuit proposed in this paper are 806.4, 684.9, 588.2, 459.8, and 392.2 Mhz. The maximum throughput is 50.4, 85.6, 147.1, 229.9, 392.2 Gbps. The maximum throughput of PSV-WN-CRC is higher than that of the architectures in [41] by an average

of 7.7%, 6.9–24.0% higher than the architectures in [43], 112.4–144.1% higher than the architectures in [46], and 23.1% higher than the average of traditional multi-path selection methods.

Table 2. Performance comparison of parallel CRC methods.

	Bus Width (w)	LUTs	Fmax (Mhz)	Throughput (Gbps)	Throughput/LUTs
PSV-WN-CRC	64	510	806.4	50.4	0.099
	128	818	684.9	85.6	0.105
	256	1550	588.2	147.1	0.095
	512	3519	459.8	229.9	0.065
	1024	5981	392.2	392.2	0.066
PQC-CRC in [43]	64	846	650.2	40.6	0.048
	128	1663	577.6	72.2	0.043
	256	5368	550.2	137.6	0.026
	512	- *	-	-	-
	1024	-	-	-	-
Architecture in [41]	64	697	698.7	43.7	0.063
	128	1358	649.6	81.2	0.060
	256	2218	564.8	141.2	0.064
	512	4983	450.8	225.4	0.045
	1024	8935	352.3	352.3	0.039
Stride-1 in [46]	64	454	319.7	19.9	0.044
	128	673	288.5	36.1	0.054
	256	1345	273.2	68.3	0.051
	512	-	-	-	-
	1024	-	-	-	-
Multi-path Selection	64	864	613.9	38.4	0.043
	128	4276	552.8	69.1	0.017
	256	11683	486.3	121.6	0.011
	512	-	-	-	-
	1024	-	-	-	-

* Represented here, this method is not applicable to applications with large bus widths.

In order to compare the performance of several methods more intuitively, an index (Max Throughput/Resource Consumption) is introduced here to measure the maximum throughput that can be achieved per unit of resource consumption. It indicates which of the different methods can achieve higher throughput with less resource consumption. As shown in the last column of Table 2, the PSV-WN-CRC proposed in this paper is able to achieve higher throughput with less resource consumption compared to several other implementations. The throughput can reach 229.9 Gbps and 392.2 Gbps when the input bit width is 512 and 1024, which can satisfy the high-performance applications with high throughput requirements.

Finally, this CRC circuit is applied to the 100 G RNIC system built on the Xilinx U280 board to realize the RDMA communication between RNICs, and the test environment is shown in Figure 7. The data transmission clock frequency of the RNIC is 322 MHz, and the data transmission bit width is 512 bits. Among them, to ensure data correctness in high-speed transmission, the CRC circuit designed in this paper is embedded in the link layer of RNIC to perform real-time checksums on each RDMA packet. In the functional test, the input, output signals and *crc_out* signals of the CRC modules are captured by the Integrated Logic Analyzer (ILA) tool, including the parallel CRC modules on the respective transmit and receive paths at the request and response ends. Figure 8 shows the data encoding process of the parallel CRC module on the transmit path at the request side, and the *crc_out* signal is the CRC checksum value of each packet. Figure 9 shows the data-checking process of the parallel CRC module of the receiving path at the response end, where the input data is the request packet received at the response end, the output

is the packet with successful checking, and the *crc_out* signal is the CRC checking value, and the value of 32′h2144DF1C indicates that the CRC checking is correct. After testing, the CRC module runs stably in the system and meets the requirements of low resource consumption, high bandwidth and low latency.

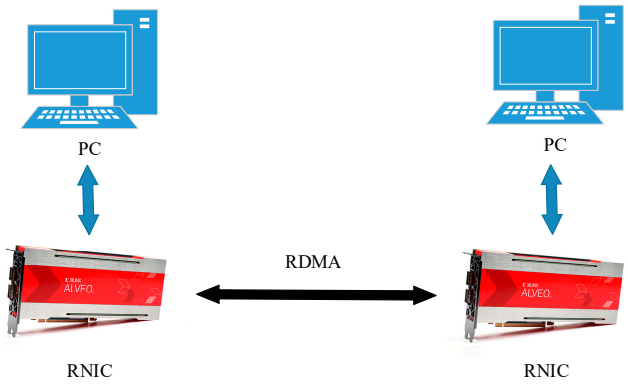


Figure 7. RNIC system test environment.

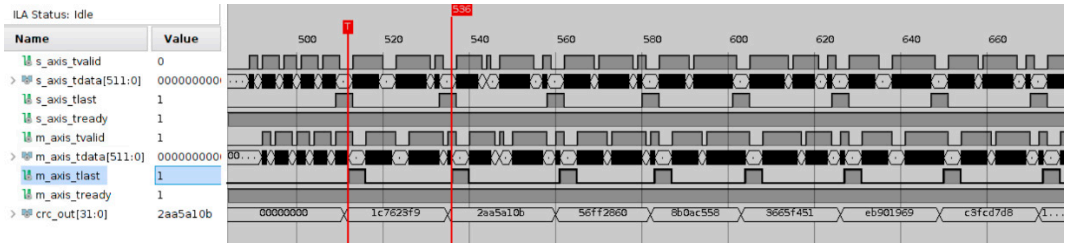


Figure 8. Waveform diagram of the parallel CRC module on the request side sending path.

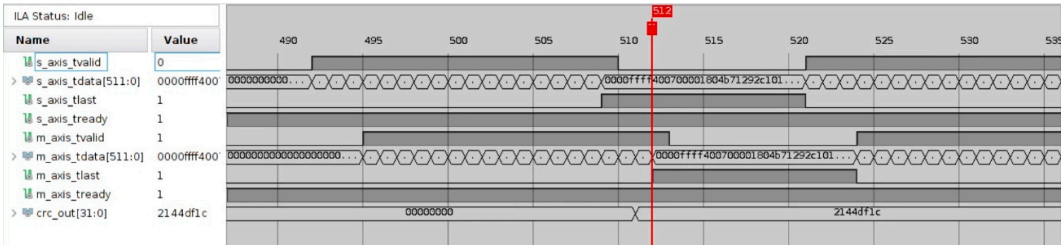


Figure 9. Waveform diagram of the parallel CRC module on the response side receiving path.

The experimental results show that the PSV-WN-CRC proposed in this paper outperforms the three advanced methods in terms of resource consumption and maximum throughput, and is able to provide efficient CRC computation for high-bandwidth networks such as RNICs, which further verifies the efficiency and practicality of the method.

5. Discussions

The CRC method proposed in this paper has the potential for future applications in high-bandwidth networks. With the increasing data transmission rate, data-checking techniques need to reduce resource consumption and delay while ensuring integrity. The PSV-WN-CRC method in this paper is suitable for high-throughput scenarios and can operate efficiently in high-bandwidth networks such as RNICs, which may provide a low resource-consuming verification scheme for the construction of data centers and next-generation communication networks.

In the data center space, interconnect technology is progressively relying on RDMA to enable large-scale data transfers between servers. RDMA technology allows network

devices to directly access host memory, which significantly reduces CPU loads and network latency, and dramatically improves the efficiency of data transfers. However, with the exponential growth of network bandwidth demand in data centers, 400 G and 800 G networks are expected to be widely used within data centers by 2026. The PSV-WN-CRC method proposed in this paper was applied to RNIC systems supporting RDMA communication in data centers, which can achieve efficient error checking in RDMA environments, and is particularly suitable for data-intensive applications requiring low latency and high throughput. In next-generation 5 G/6 G communication networks, the method can be used for high-speed data transmission from the base station to the core network to meet the high demand for real-time and accuracy. For example, during the transmission of a large number of data streams between the user equipment and the base station, the data checking speed and stability can be significantly improved by introducing the PSV-WN-CRC method, which provides a guarantee for the smooth operation of the network.

Although the method in this paper shows certain advantages, it may still face the challenges of adaptability and performance optimization in practical applications. For example, in response to the demand for diverse CRC polynomials in different network protocols, the current proposed circuit structure needs to modify the circuit and regenerate the bitstream if it needs to adapt to different CRC polynomials, which is a relatively time-consuming process. In order to improve the adaptability, the method can be extended in the future to refer to the dynamic reprogramming using HWICAP, so as to enhance the adaptability of the circuit in multiple scenarios. In terms of performance optimization, the circuit design proposed in this paper involves left and right shift circuits as well as CRC calculation circuits. However, since shift and splice operations increase the critical path delay, they may have an impact on the timing of the circuit, especially in high-frequency application scenarios with large bus widths, where longer critical path delays can limit the clock frequency of the circuit. Future improvement directions can consider techniques such as pipelining or parallel shifting to improve the timing stability of the left-shifted and right-shifted circuits, thereby increasing the maximum operating frequency of the circuit and hence the transmission bandwidth. In addition, the CRC circuit in this paper uses a LFSR to handle the CRC calculation task. Since the computation of the LFSR relies on the feedback chain, it may lead to longer critical paths, thus affecting the timing convergence. Improvements can also be made by considering the use of a pipelined design to enhance the timing stability of the CRC computation circuit, thereby increasing the overall circuit's operating frequency and data throughput capability.

In summary, the PSV-WN-CRC method proposed in this paper has a wide range of applications in high-bandwidth networks and provides an efficient means of checking with low resource consumption for future 5G/6G communication and data center interconnection technologies. Future research can be carried out to further optimize the applicability and performance of the method to meet more diversified application requirements and continue to improve the system performance in combination with other hardware acceleration techniques.

6. Conclusions

Aiming at the problems of over-utilization of CRC circuit resources and difficult timing convergence as well as variable length of data frame tails in high-bandwidth networks, this paper proposes a new parallel CRC method, PSV-WN-CRC, from the perspective of simplifying the calculation logic, and details the derivation process and mathematical proof of the algorithm. This paper also proposes a corresponding CRC circuit structure and describes the hardware implementation in detail by module. The method only needs to use a fixed-width CRC calculation circuit to realize the CRC calculation of different bit-width data, and the calculation cycle is only one clock cycle. The experimental results show that compared with three advanced works, the method and circuit proposed in this paper can achieve the goals of lower resource utilization and higher maximum throughput and can

operate stably in a high-bandwidth network such as RNIC to meet the requirements of low resource utilization, high-speed processing and low latency. In addition, there is still room for further optimization of the left-shift and right-shift circuits and the CRC calculation unit, which can further improve circuit performance. This method provides an efficient verification scheme with low resource consumption for future next-generation communication technologies and high-bandwidth application scenarios such as data centers.

Author Contributions: Conceptualization, L.Z.; methodology, Y.L.; software, Z.G.; validation, Z.G., S.Y. and X.Y.; formal analysis, Q.D.; data curation, F.W.; writing—review and editing, S.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This work was funded in part by the Key Project of the National Natural Science Foundation of China under Grant No. 62334008.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Pei, T.-B.; Zukowski, C. High-speed parallel CRC circuits in VLSI. *IEEE Trans. Commun.* **1992**, *40*, 653–657. [CrossRef]
2. Wang, X.; Zhuang, L.; Lu, Q.; Wang, S. The Research of Parallel CRC Pipeline Algorithm Based on Matrix Transformation. In Proceedings of the 2012 International Conference on Computer Science and Service System, Nanjing, China, 11–13 August 2012; pp. 162–165.
3. Akagic, A.; Amano, H. Performance analysis of fully-adaptable CRC accelerators on an FPGA. In Proceedings of the 22nd International Conference on Field Programmable Logic and Applications (FPL), Oslo, Norway, 29–31 August 2012; pp. 575–578.
4. *802.3bs-2017*; Amendment 10: Media Access Control Parameters, Physical Layers, and Management Parameters for 200 Gbps and 400 Gbps Operation. IEEE: Piscataway, NJ, USA, 2017.
5. Qaqos, N.N. Optimized FPGA implementation of the CRC using parallel pipelining architecture. In Proceedings of the 2019 International Conference on Advanced Science and Engineering (ICOASE), Zakho, Iraq, 2–4 April 2019; pp. 46–51.
6. Peterson, W.W.; Brown, D.T. Cyclic codes for error detection. *Proc. IRE* **1961**, *49*, 228–235. [CrossRef]
7. Patel, A.M. A multi-channel CRC register. In Proceedings of the May 18–20, 1971, Spring Joint Computer Conference, Atlantic City, NJ, USA, 18–20 May 1971; pp. 11–14.
8. Sarwate, D.V. Computation of cyclic redundancy checks via table look-up. *Commun. ACM* **1988**, *31*, 1008–1013. [CrossRef]
9. Albertengo, G.; Sisto, R. Parallel CRC generation. *IEEE Micro* **1990**, *10*, 63–71. [CrossRef]
10. Weerasinghe, J.; Abel, F.; Hagleitner, C.; Herkersdorf, A. Enabling FPGAs in hyperscale data centers. In Proceedings of the 2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom), Beijing, China, 10–14 August 2015; pp. 1078–1086.
11. Weerasinghe, J.; Polig, R.; Abel, F.; Hagleitner, C. Network-attached FPGAs for data center applications. In Proceedings of the 2016 International Conference on Field-Programmable Technology (FPT), Xi'an, China, 7–9 December 2016; pp. 36–43.
12. Tarafdar, N.; Lin, T.; Fukuda, E.; Bannazadeh, H.; Leon-Garcia, A.; Chow, P. Enabling Flexible Network FPGA Clusters in a Heterogeneous Cloud Data Center. In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 22–24 February 2017; pp. 237–246.
13. Bertolino, M.; Pacalet, R.; Apvrille, L.; Enrici, A. Efficient scheduling of FPGAs for cloud data center infrastructures. In Proceedings of the 2020 23rd Euromicro Conference on Digital System Design (DSD), Kranj, Slovenia, 26–28 August 2020; pp. 57–64.
14. Hoozemans, J.; Peltenburg, J.; Nonnemacher, F.; Hadnagy, A.; Al-Ars, Z.; Hofstee, H.P. Fpga acceleration for big data analytics: Challenges and opportunities. *IEEE Circuits Syst. Mag.* **2021**, *21*, 30–47. [CrossRef]
15. Mbongue, J.M.; Saha, S.K.; Bobda, C. Domain Isolation in FPGA-Accelerated Cloud and Data Center Applications. In Proceedings of the 2021 Great Lakes Symposium on VLSI, Virtual Event, USA, 22–25 June 2021; pp. 283–288.
16. Chamola, V.; Patra, S.; Kumar, N.; Guizani, M. FPGA for 5G: Re-configurable hardware for next generation communication. *IEEE Wirel. Commun.* **2020**, *27*, 140–147. [CrossRef]
17. Visconti, P.; Velázquez, R.; Soto, C.D.-V.; De Fazio, R. FPGA based technical solutions for high throughput data processing and encryption for 5G communication: A review. *TELKOMNIKA (Telecommun. Comput. Electron. Control)* **2021**, *19*, 1291–1306. [CrossRef]
18. Bartzoudis, N.; Rubio Fernández, J.; López-Bueno, D.; Román Villarroel, A.; Antonopoulos, A. Agile FPGA Computing at the 5G Edge: Joint Management of Accelerated and Software Functions for Open Radio Access Technologies. *Electronics* **2024**, *13*, 701. [CrossRef]
19. Gou, M.; Wang, B.; Zhang, X. Development of Multi-Motor Servo Control System Based on Heterogeneous Embedded Platforms. *Electronics* **2024**, *13*, 2957. [CrossRef]

20. Ni, X.; Cen, Y.; Tyagi, T.; Enemali, G.; Arslan, T. 5G Enabled Dual Vision and Speech Enhancement Architecture for Multimodal Hearing-Aids. *Electronics* **2024**, *13*, 2588. [CrossRef]
21. de Sousa, M.A.d.A.; Pires, R.; Del-Moral-Hernandez, E. SOMprocessor: A high throughput FPGA-based architecture for implementing Self-Organizing Maps and its application to video processing. *Neural Netw.* **2020**, *125*, 349–362. [CrossRef] [PubMed]
22. Sarkar, S.; Bhairannawar, S.S. Efficient FPGA architecture of optimized Haar wavelet transform for image and video processing applications. *Multidimens. Syst. Signal Process.* **2021**, *32*, 821–844. [CrossRef]
23. Sarkar, S.; Bhairannawar, S.S.; KB, R. FPGACam: A FPGA based efficient camera interfacing architecture for real time video processing. *IET Circuits Devices Syst.* **2021**, *15*, 814–829. [CrossRef]
24. Jia, C.; Hang, X.; Wang, S.; Wu, Y.; Ma, S.; Gao, W. Fpx-nic: An fpga-accelerated 4k ultra-high-definition neural video coding system. *IEEE Trans. Circuits Syst. Video Technol.* **2022**, *32*, 6385–6399. [CrossRef]
25. Jeong, D.; Lee, M.; Lee, W.; Jung, Y. FPGA-Based Acceleration of Polar-Format Algorithm for Video Synthetic-Aperture Radar Imaging. *Electronics* **2024**, *13*, 2401. [CrossRef]
26. Leal, D.P.; Sugaya, M.; Amano, H.; Ohkawa, T. Automated integration of high-level synthesis fpga modules with ros2 systems. In Proceedings of the 2020 International Conference on Field-Programmable Technology (ICFPT), Maui, HI, USA, 9–11 December 2020; pp. 292–293.
27. Kojima, A. Autonomous driving system implemented on robot car using soc fpga. In Proceedings of the 2021 International Conference on Field-Programmable Technology (ICFPT), Auckland, New Zealand, 6–10 December 2021; pp. 1–4.
28. Uetsuki, T.; Okuyama, Y.; Shin, J. CNN-based End-to-end Autonomous Driving on FPGA Using TVM and VTA. In Proceedings of the 2021 IEEE 14th International Symposium on Embedded Multicore/Many-Core Systems-on-Chip (MCSoC), Singapore, 20–23 December 2021; pp. 140–144.
29. Li, Y.; Li, S.E.; Jia, X.; Zeng, S.; Wang, Y. FPGA accelerated model predictive control for autonomous driving. *J. Intell. Connect. Veh.* **2022**, *5*, 63–71. [CrossRef]
30. Sciangula, G.; Restuccia, F.; Biondi, A.; Buttazzo, G. Hardware acceleration of deep neural networks for autonomous driving on FPGA-based SOC. In Proceedings of the 2022 25th Euromicro Conference on Digital System Design (DSD), Maspalomas, Spain, 31 August–2 September 2022; pp. 406–414.
31. Dimitrova, R.S.; Gehrig, M.; Brescianini, D.; Scaramuzza, D. Towards low-latency high-bandwidth control of quadrotors using event cameras. In Proceedings of the 2020 IEEE International Conference on Robotics and Automation (ICRA), Paris, France, 31 May–31 August 2020; pp. 4294–4300.
32. Du, J.; Jiang, C.; Wang, J.; Ren, Y.; Debbah, M. Machine learning for 6G wireless networks: Carrying forward enhanced bandwidth, massive access, and ultrareliable/low-latency service. *IEEE Veh. Technol. Mag.* **2020**, *15*, 122–134. [CrossRef]
33. Niwa, N.; Amano, H.; Koibuchi, M. Low-latency high-bandwidth interconnection networks by selective packet compression. In Proceedings of the 2021 Ninth International Symposium on Computing and Networking (CANDAR), Matsue, Japan, 23–26 November 2021; pp. 56–64.
34. Velayutham, A. Optimizing sase for low latency and high bandwidth applications: Techniques for enhancing latency-sensitive systems. *Int. J. Intell. Autom. Comput.* **2023**, *6*, 63–83.
35. Kok, C.L.; Siek, L. Designing a Twin Frequency Control DC-DC Buck Converter Using Accurate Load Current Sensing Technique. *Electronics* **2024**, *13*, 45. [CrossRef]
36. Kounavis, M.E.; Berry, F.L. Novel table lookup-based algorithms for high-performance CRC generation. *IEEE Trans. Comput.* **2008**, *57*, 1550–1560. [CrossRef]
37. Akagic, A.; Amano, H. High-speed fully-adaptable CRC accelerators. *IEICE Trans. Inf. Syst.* **2013**, *96*, 1299–1308. [CrossRef]
38. Walma, M. Pipelined cyclic redundancy check (CRC) calculation. In Proceedings of the 2007 16th International Conference on Computer Communications and Networks, Honolulu, HI, USA, 13–16 August 2007; pp. 365–370.
39. Nie, Y.; Cai, F.; Zhang, K.; Zhong, S.; Luo, H. A Formal Design of Parallel CRC Circuit. In Proceedings of the 2023 5th International Conference on Electronic Engineering and Informatics (EEI), Wuhan, China, 30 June–2 July 2023; pp. 449–452.
40. Hajare, P.S.; Mankar, K. Design and Implementation of Parallel CRC Generation for High Speed Application. *IOSR J. VLSI Signal Process. (IOSR-JVSP)* **2015**, *5*, 1.
41. Kekely, L.; Cabal, J.; Kořenek, J. Effective fpga architecture for general crc. In Proceedings of the Architecture of Computing Systems–ARCS 2019: 32nd International Conference, Copenhagen, Denmark, 20–23 May 2019; Proceedings 32, 2019. pp. 211–223.
42. Liu, H.; Qiu, Z.; Pan, W.; Li, J.; Zheng, L.; Gao, Y. Low-cost and programmable CRC implementation based on FPGA. *IEEE Trans. Circuits Syst. II Express Briefs* **2020**, *68*, 211–215. [CrossRef]
43. Zhou, Z.; Wang, R. A Method of High-Speed Parallel CRC Computation. In Proceedings of the 2023 5th International Conference on Electronic Engineering and Informatics (EEI), Wuhan, China, 30 June–2 July 2023; pp. 298–303.
44. Kishore, P.; Pal, B.A.; Kishore, L.N.; Revathi, C.V. Implementation of Table-Based Cyclic Redundancy Check (CRC-32) for Gigabit Ethernet Applications. In Proceedings of the 2023 4th International Conference for Emerging Technology (INCET), Belgaum, India, 26–28 May 2023; pp. 1–4.
45. Das, A. Block-Wise Computation of Cyclic Redundancy Code Using Factored Toeplitz Matricesin Lieu of Look-Up Table. *IEEE Trans. Comput.* **2022**, *72*, 1110–1121. [CrossRef]

46. Cai, F.; Nie, Y.; Zhang, K.; Luo, H.; Li, Y. A high-speed CRC-32 Implementation on FPGA. In Proceedings of the 2024 4th International Conference on Neural Networks, Information and Communication (NNICE), Guangzhou, China, 19–21 January 2024; pp. 1665–1668.
47. Thomas, D.; Moorby, P. *The Verilog® Hardware Description Language*; Springer Science & Business Media: Berlin, Germany, 2008.
48. Salauyou, V.; Zabrocki, Ł. Coding techniques in Verilog for finite state machine designs in FPGA. In Proceedings of the Computer Information Systems and Industrial Management: 18th International Conference, CISIM 2019, Belgrade, Serbia, 19–21 September 2019; Proceedings 18, 2019. pp. 493–505.
49. Stock, F.; Koch, A.; Hildenbrand, D. FPGA-accelerated color edge detection using a Geometric-Algebra-to-Verilog compiler. In Proceedings of the 2013 International Symposium on System on Chip (SoC), Tampere, Finland, 23–24 October 2013; pp. 1–6.
50. Rose, J.; Luu, J.; Yu, C.W.; Densmore, O.; Goeders, J.; Somerville, A.; Kent, K.B.; Jamieson, P.; Anderson, J. The VTR project: Architecture and CAD for FPGAs from verilog to routing. In Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays, Monterey, CA, USA, 22–24 February 2012; pp. 77–86.
51. Qiu, M.; Yu, S.; Wen, Y.; Lü, J.; He, J.; Lin, Z. Design and FPGA implementation of a universal chaotic signal generator based on the Verilog HDL fixed-point algorithm and state machine control. *Int. J. Bifurc. Chaos* **2017**, *27*, 1750040. [CrossRef]