2019 International Conference on Advanced Science and Engineering (ICOASE),
University of Zakho, Duhok Polytechnic University, Kurdistan Region, Iraq

Optimized FPGA Implementation of the CRC Using Parallel Pipelining Architecture

Noor Najeeb Qaqos
Information Technology Department
Duhok Polytechnic University
Duhok, Iraq
noor.qaqos@dpu.edu.krd

***Abstract*—Cyclic Redundancy Check (CRC) plays a major role in data storage, communication systems and networking environment fields to detect errors. The speed of transmitting data with optimized hardware utilization are the main challenges nowadays. Thus, CRC calculation becomes a bottleneck for the system implementations. The aim of this paper is to design and implement the CRC5 and CRC8 systems that are used for USB token packet and ATM protocols, respectively. A parallel pipelining method is used to implement the proposed CRC architecture for both CRC encoder and decoder systems to achieve high throughput data with optimized hardware resources. A proposed architecture doesn't base on Look-Up Table (LUT) to store pre-calculated CRC values or F-matrix in its implementation as in the previous works. The system designed and implemented based on Spartan-3E FPGA chip, Very High-Speed Integrated Circuit Description Language (VHDL) used to write the related code. The whole proposed architecture is functionally simulated and verified using the Xilinx ISE 9.2i simulator.**

Keywords—CRC, Optimized Implementation, Parallel-Pipelining, FPGA, Spartan 3E-kit, VHDL, Xilinx

I. INTRODUCTION

The integrity of data that is transmitted over a noisy channel can be protected by using codes for error detection and correction of disrupted data. There are many techniques to detect and correct errors, but a CRC is the widely adopted code. It represents the most common, efficient and reliable data communication method, which is first described by Peterson and Brown in 1961 [1]. The CRC requires less hardware resources and more easily implementation as compared with other techniques, so this is the reason that why CRC is being used for more than three decades, despite of developing many error detection and correction advance techniques.

CRC has a wide range of applications in data storage devices and communication systems. CRC uses binary division technique. The most common hardware implementation of CRC computation is the Linear Feedback Shift Register (LFSR) [2], which handle the data in serial way and this calculation can not a achieve a high throughput. Therefore, to solve this problem researches are moving to parallel CRC generator architecture, but parallel processing at the same time leads to long Critical Path which is the bottleneck of the system that leads to decreasing its speed. So to reduce the critical path, a pipelined technique must be used and added to the parallel architecture to obtain a parallel pipelining method. Hence the proposed design will get a high throughput by using a parallel technique and a high speed from the pipeline method, taking into consideration the optimal use of hardware resources. A proposed parallel pipelining architecture is designed and implemented in this work, by dividing the CRC calculation architecture into many stages to increase processing speed and increase throughput at the same time. Therefore, in the proposed architecture, the time of clock is equal to the time requires to implement the most delayed stage. In addition to that, the proposed architecture is not based on LUTs (SRAM in FPGA) in its implementation to achieve high throughput data with optimized hardware resources. The proposed architecture implemented on FPGAs, which are programmable logic devices that represent the most common and frequently devices used to implement a wide range of digital systems [3].

This paper is organized as the following: Besides this introductory section. Section II introduces to the brief description of CRC encoder and decoder system. CRC related works are discussed in section III, Section IV describes the proposed CRC architecture, while the hardware implementation of the architecture represents in section V. Implementation summary, results and their discussion are presented in section VI. Finally, the conclusions are drawn based on obtained results.

II. CRC GENERATION

A. Encoder system

The basic calculation of CRC is as the following: if we assume the length of a transmitted data is equal to (a) bits. Firstly, k 0s are appended to (a). Where k is the polynomial degree of CRC. This operation can be done as a multiplication with 2^k resulting in $M = a * 2^k$. Secondly, the result M is divided by generator polynomial P using module 2 arithmetic. The k bits remainder (checksum) of the mentioned operation represent by CRC. Finally, the CRC is appended and replaced with the k 0s resulting in (a+CRC) bits to create the codeword that is transmitted to the receiver side. Figs. 1a and 2a show the encoder side and its mathematical computation of CRC process [2].

978-1-5386-9343-8/19/$31.00 ©2019 IEEE

B. Decoder system

The decoder receives the codeword that is possibly corrupted through transmission. The receiver divides the received codeword by checker (which is a replica of generator) and calculate the syndrome (remainder), if the syndrome is equal to zero this means that there are no errors in the receiving message and the message is exactly the same as it is transmitted by the sender, otherwise the message is corrupted through the transmission channel and then discarded. Figs. 1b and 2b show the decoder side and its mathematical computation of CRC process [2].

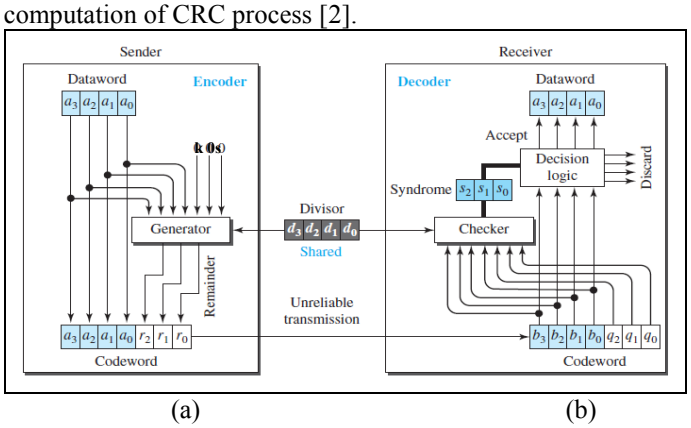

Fig. 1. Encoder and decoder process.

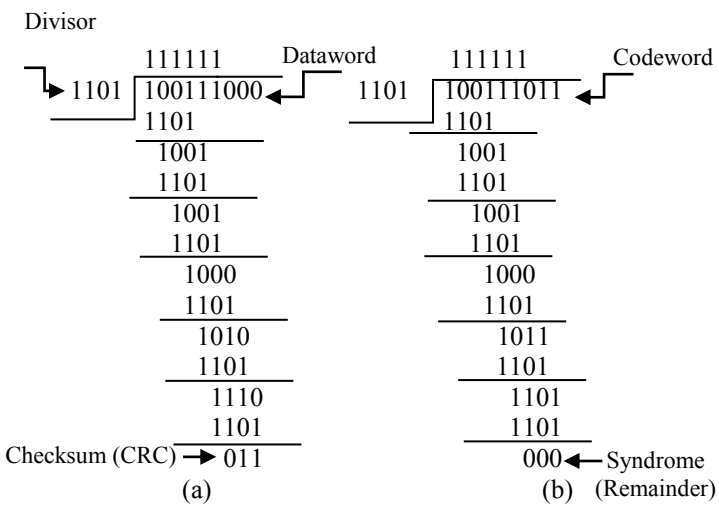

Fig. 2. CRC mathematical calculation.

III. Related Works

In traditional hardware implementations, a Linear Feedback Shift Register is a simple circuit based on shift registers and XOR gates perform the CRC calculation by handling the message one bit at a time. Fig. 3 shows the architecture of CRC5 as an example. It is accomplished on hardware by handling one bit of input data at a time [4]. Binary division is the main operation of LFSR for CRC generation which can be performed by using some of shift and subtract operations. Firstly, all the remainder registers rest to zero, MSB of the message is entered first to the circuit and then the data is shifted to the right or added (XORED) with the feedback values. Once the whole data bits are shifted into the registers the remainder registers represent the checksum (CRC) bits that are concatenated with the data bits before transmitting to the receiver side[5].

CRC-5 Generator Polynomial

$$P(x)=x^5 + x^2 + 1 \quad (1)$$

The coefficients of $P(x)$ can be constructed and represented in binary form as:

$P_k= P_5 , P_4 , P_3 , P_2 , P_1 , P_0$

P_k={100101}.

The checksum will be generated after n+k cycle, where n is the number of input bits and k is the degree of generator polynomial. In the case of CRC5 for n=11bits as input data and polynomial degree k=5. The serial CRC generation will produce the result after 16 clock cycle. A low throughput and high latency of data are big problems in this implementation because the process of input data will be one bit in every clock pulse.

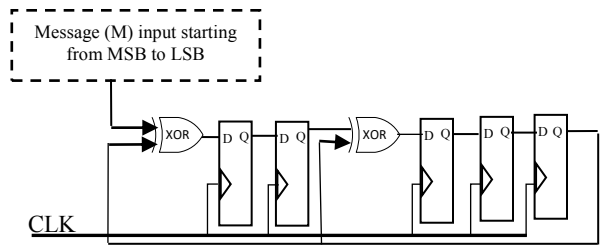

Fig. 3. Serial LFSR architecture.

The development that has been happened in data storage, networks, communication systems and their applications have made us in need of faster processing speed with optimized hardware resources. Therefore, the CRC implementation has been changed from serial to parallel processing to increase the throughput of the processing. The primary work on parallel CRC calculation was presented by Pie in 1992 [6], succeeding that many various techniques generated for CRC parallel implementation. These techniques include: Table-based algorithm [7][8][9][10], Fast CRC updating [11], F- matrix based on parallel CRC[12] and retiming, unfolding and pipelining method[13][14]. Despite of parallel processing is an effective method to increase the throughput of the systems due to increasing the number of input bits that can be processed by one clock cycle. But at the same time, the big problem of parallel processing is that it leads to a long critical path. The long critical path means that the time period of clock cycle to execute the parallel operation increased which leads to decrease of circuit speed. The increase of throughput that accomplished by parallel processing will be decreased by the reducing of circuit speed. In addition to that, an increasing in the hardware resources when a system is implemented using a parallel processing technique.

In this work, a parallel pipeline technique without using look-up table method is used to speed up the CRC calculation by splitting the CRC parallel architecture into many sub-stages to reduce the critical path to obtain a fast hardware implementation. Therefore, the proposed architecture will do an optimization between the speed of the implementation and the hardware resource utilization (without using BRAMs in FPGA).

IV. Proposed Architecture

A novel proposed CRC architecture and its implementation are presented in this paper. The architecture based on parallel pipeline technique to increase the throughput of data by taking in the consideration optimized implementation of hardware resources. Fig. 4 shows the CRC core architecture that designed using parallel pipelining technique, which is based on a simple conventional mathematical calculation that is shown in Fig. 2. The proposed design can be used to implement the different degree of CRC polynomial but it is more efficient for a low polynomial degrees than higher ones. The input data is denoted by (D) and the Polynomial degree by (P). Whereas (n) and (k) represent the number of bits of data (D) and polynomial degree (P), respectively. The message (M) represents the data bits (D) with k 0s appended bits. Figs. 5 and 6 show the flowchart for the encoder and decoder circuit of the proposed CRC core, respectively. Where (&) is the concatenation operator in VHDL language.

From Fig. 4, It can be realized that:

1- The number of input message bits (M) in each pipeline stage is equal to (n+k) for both encoding and decoding processes. where in the encoding side, k 0s are appended with the data, while the CRC bits are replaced with k 0s that send to the receiver side.

2- The number of bits in each stage that are **XORED** with polynomial or zeros is equal to k, the leftmost bit of the polynomial is not needed in the calculation because the result of operation is always 0, regardless what the value of this bit. The reason is that the XOR inputs of leftmost bits are either both 0s or 1s.

3- The number of stages in the proposed pipeline will be equal to the number of input parallel bits to the design (n).

4- The multiplexer is used to check the leftmost bit in each stage and decide if the data will be XORED with the polynomial bits or with p 0s, if the leftmost bit ='1' then XOR with the polynomial or else will XORED with p 0s

5- The CLK and CLR signals are used to synchronize the core and reset the resisters (Flip-Flops), respectively.

6- The long critical path in the proposed design is an XOR gate only, so the slowest path is equal to the time to execute one XOR gate.

Thus, the proposed architecture increases the throughput of CRC by increasing the frequency of the system.

V.CRC Hardware Implementation

There are many techniques which are used to implement the CRC system on hardware, but FPGA represents the best one of them. In spite of that there are many drawbacks of FPGAs like the restriction of resources available on the FPGA that will limit the design size and features, but at the same time it can perform a humongous amount of parallel data processing very quickly and efficiently. So, in this work the FPGA is used to implement the CRC system which is presented in Fig.7. The system includes two CRC core circuits (CRC5 and CRC8) that proposed in Fig. 4 which is used for different applications.

The generator polynomials of the implemented CRC system include:

- CRC5 Polynomial $P(x)=x^5 + x^2 + 1$ (1)
- CRC8 Polynomial $P(x)=x^8 + x^2 + x + 1$ (2)

Table I shows the CRC circuits with their details. Spartan-3E 500k FPGA kit is used as the hardware implementation for the proposed architecture. The number of data input bits that used for each polynomial is 11 bits for CRC5 and 64 bits for CRC8 achieving a hamming distance (HD) equal to 4 [15]. The proposed CRC core which is clarified in Fig. 4 is used to design and implement the complete CRC system shown in Fig. 7. Control signal (CS) is used to select one of the two CRC cores, whereas ED signal is used to select the encoder or decoder operation on the input data.

TABLE I. CRC Circuits with their Details

CS	CRC Selection	Polynomial bits	Applications
0	CRC5	0X05	USB token packet
1	CRC8	0X07	ATM Protocols

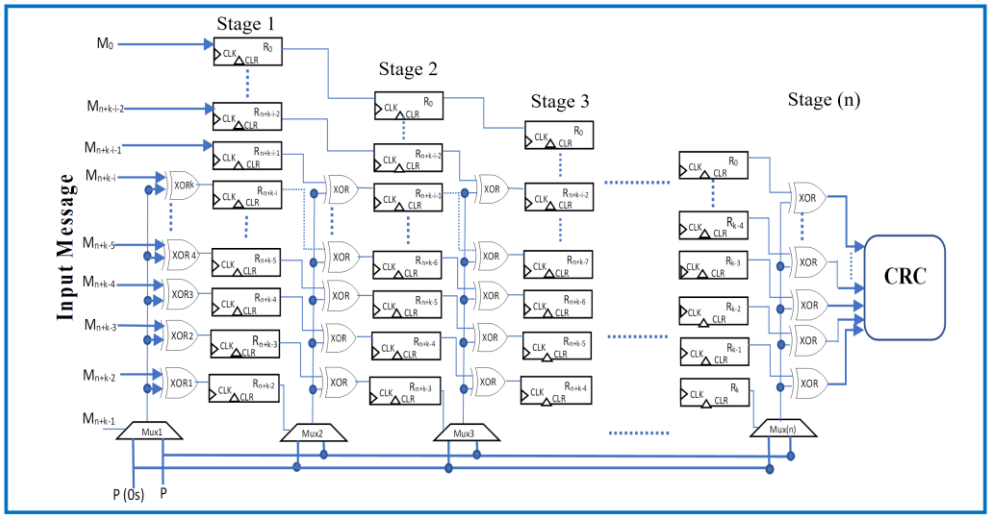

Fig. 4. Proposed CRC core architecture (2≤ i≤n+k).

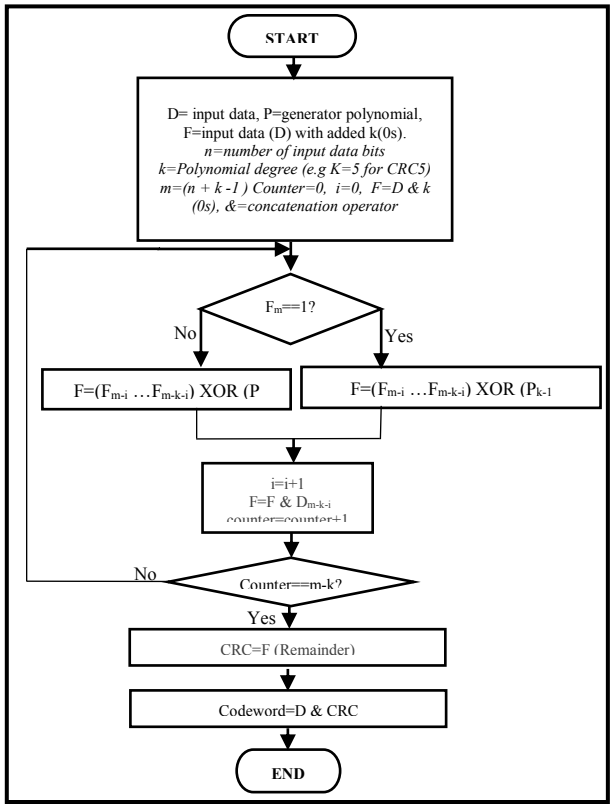

Fig. 5. Flowchart of the proposed encoder CRC process

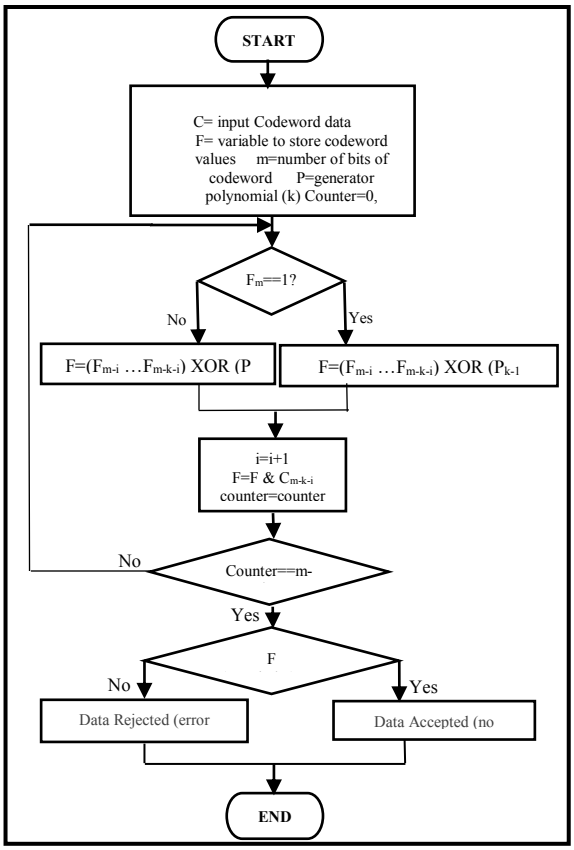

Fig. 6. Flowchart of the proposed decoder CRC process.

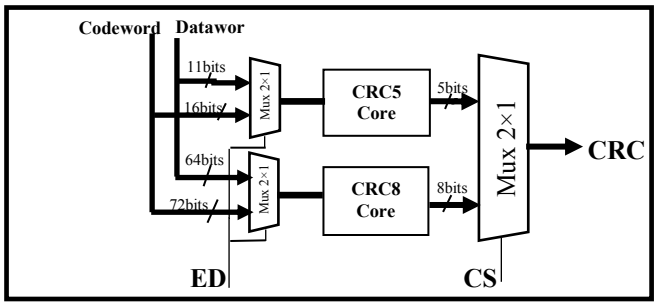

Fig. 7. Proposed architecture of CRC system.

The optimal number of logic components which is used in the proposed CRC cores can be calculated as in table II. Where k and n represent the number of polynomial degree and input data bits, respectively.

TABLE II. OPTIMAL NUMBERS OF LOGIC COMPONENTS IN EACH CRC CORE

Cores	Number of Stages (S)	Number of XORs = number of XORs in each stage × S	Number of Muxs.= number of Mux in each stage ×S	Number of Flip Flops= $\sum_{F=k+1}^{n+k-1} F$
CRC5	11	5×11=55	5×11=55	105
CRC8	64	8×64=512	8×64=512	2520

VI. IMPLEMENTATION SUMMARY AND RESULTS

The proposed architecture is implemented on FPGA Spartan-3E kit. The system synthesized, and the result verified using Xilinx ISE 9.2i simulator. VHDL used as a description hardware language of the CRC design. The top level of the complete CRC system and Register Transfer Level of the proposed CRC core (e.g. CRC5) are presented in Figs. 8 and 9, respectively. The utilization resources that used for implementing each of the CRC cores are shown in table III. Table IV shows the maximum frequency, throughput and the latencies for each core and the complete CRC system.

TABLE III. AREA UTILIZATION AND MAXIMUM FREQUENCY

Core (GCLK=1)	Utilization Area (%)			
	Number of slices (out of 4656)	Number of F.Fs. (out of 9312)	Number of LUTs (out of 9312)	Number of bounded I/O (out of 190)
CRC5	51 (1%)	107 (1%)	96 (1%)	55 (28%)
CRC8	422 (9%)	2531 (27%)	801 (8%)	68 (35%)
Complete CRC system	445 (9%)	2564 (27%)	870 (9%)	79 (41%)

TABLE IV. CRC CORES MAXIMUM FREQUENCIES, THROUGHPUTS AND LATENCIES

Core (GCLK=1)	Maximum Frequency (MHZ)	Throughput (Gbps)	Latency =Number of clock cycle / maximum frequency
CRC5	278.5	3.069	11 clock cycle (39 nsec.)
CRC8	278.5	17.824	64 clock cycle (230nsec.)
Complete CRC system	278.5	3.069 **or** 17.824	Based on the selected core

Figs. 10, 11, 12 and 13 explain the simulation results of the CRC two cores. The input and output data of the simulation are illustrated in table V. It shows some samples of the input data that are used in the simulation and the generated CRC output of the two CRC cores for both encoder and decoder processes. After inputting the data from high to low bits parallelly to core system, selecting the desired core using CS signal and determine the encoding/decoding operation of the system using ED signal. Checksum (CRC) will be the output of the cores. Error signal is used to inform us that an error occurs to the transmitted data. In the sender side, the CRC is appended with the transmitted data and the message sends to the receiver side. For the checking process, the message is checked with the same polynomial and then the CRC output is generated, if the output is zero means it is error free (Error signal will be zero), but if the output is not zero, an error has taken place to the data through the transmission operation (Error signal will be one). The errors that occur in data are bold labeled and underlined as described in table V. The data are tested using online CRC calculation to verify the correct work of the system [16].

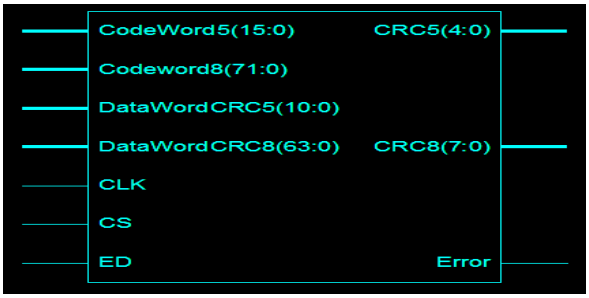

Fig. 8. Top level for the complete CRC system.

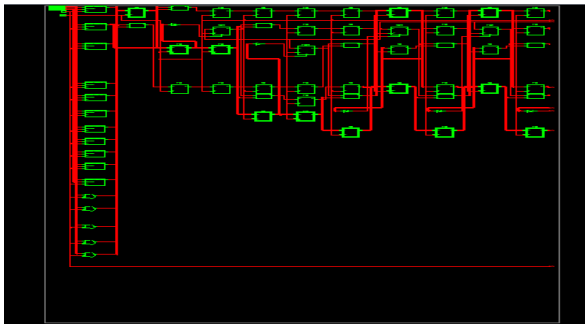

Fig. 9. RTL Schematic for the proposed CRC core (e.g. CRC5).

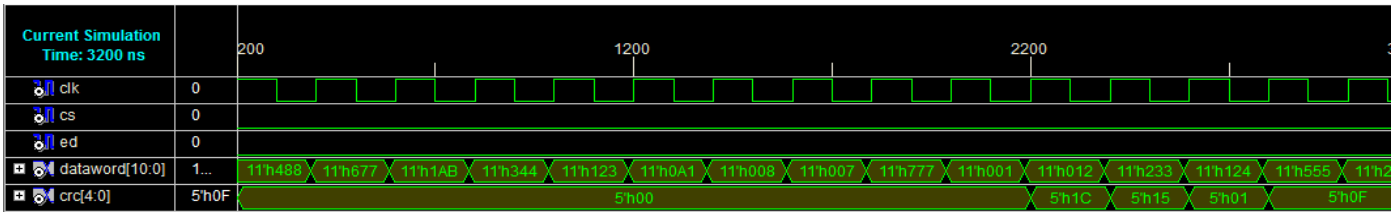

Fig. 10. CRC5 encoder simulation results.

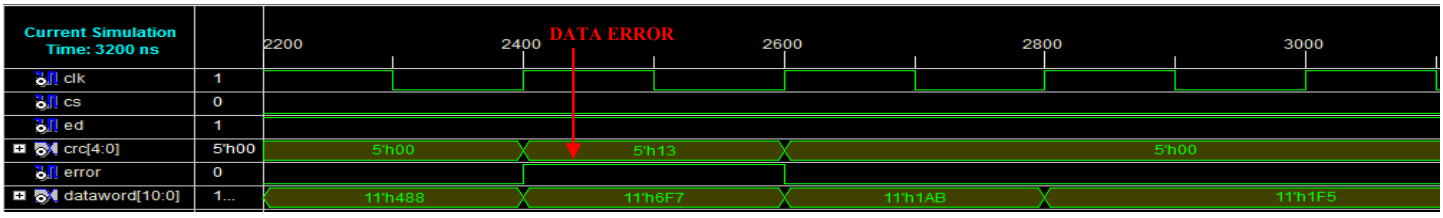

Fig. 11. CRC5 decoder simulation results.

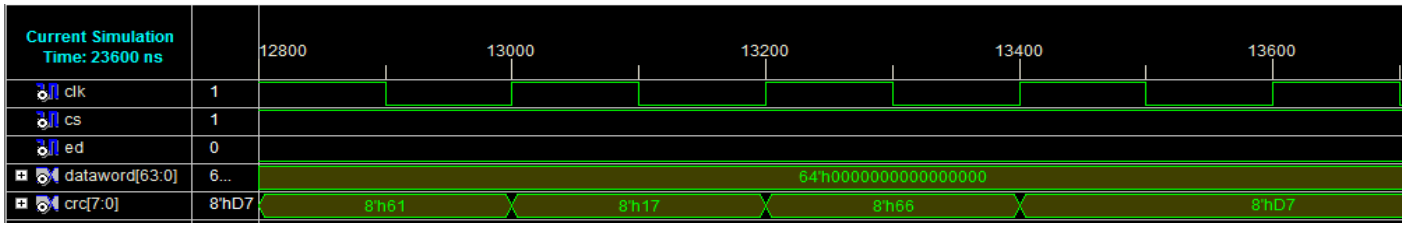

Fig. 12. CRC8 encoder simulation results.

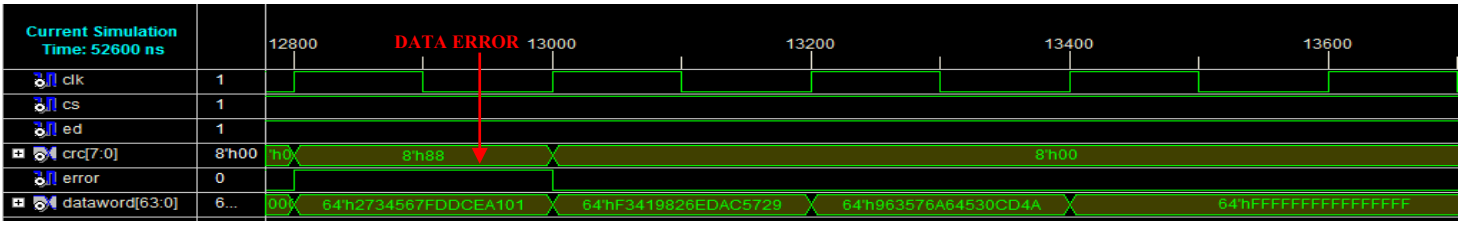

Fig. 13. CRC8 decoder simulation results.

TABLE V. SIMULATION INPUTS AND OUTPUTS OF DATA TO CRC SYSTEM

Cores	CS	ED	Input data (Hex.)	CRC (Hex.)
CRC5	0	0	488	1C
			677	15
			1AB	01
			1F5	0F
		1	911C	00 (no error)
			CEF5	13 (error)
			3561	00 (no error)
			3EAF	00 (no error)
CRC8	1	0	1734567FDDCEA101	61
			F3419826EDAC5729	17
			963576A64530CD4A	66
			FFFFFFFFFFFFFFFF	D7
		1	**2**734567FDDCEA10161	88 (error)
			F3419826EDAC572917	00 (no error)
			963576A64530CD4A66	00 (no error)
			FFFFFFFFFFFFFFFFD7	00 (no error)

From Figs. 10, 11, 12 and 13, it can be seen that the first output will be generated after (n) clock cycles, where n is the number of input data bits (D) and other CRC outputs will be generated in succession each clock cycle.

VII. CONCLUSIONS

In this paper, a novel method for CRC implementation using parallel pipelining technique is proposed. The proposed architecture is based on a simple mathematical calculation of CRC. It can calculate the checksum for CRC5 and CRC8 for input data width equal to 11 and 64bits respectively. Look-up tables (e.g. SRAMs in FPGAs) to store pre-computed CRC or F-matrix values are not required in the proposed architecture as in the previous works. The simulation results show that the proposed design can reduce the critical path, decrease the chip area and at the same time increasing the throughput rate by increasing the clock speed of the circuit. The complete proposed CRC architecture systems can be executed under 278.5 MHz, so it can be achieved the throughput of 3.069 and 17.824Gbps for CRC5 and CRC8, respectively.

REFERENCES

[1] W. Peterson and D. Brown, "Cyclic Codes for error detection," Proceedings of the IRE, vol. 49, no. 1, pp. 228-235, 1961.

[2] B. Forouzan," Data communication and networking," 4th edition, 2007.

[3] H. Sadeeq and A. Abdulazeez," Hardware implementation of firefly optimization algorithm using FPGAs," International Conference on Advanced Science and Engineering (ICOASE), pp.30-35, 2018.

[4] T. Ramabadran and S. Gaitonde," A tutorial on CRC computations," IEEE micro, vol. 8, no. 4, pp. 62-75, Aug. 1988.

[5] V. Mukati," High speed parallel architecture and pipelining for LFSR," International Journal of Scientific Research Engineering and Technology (IJSRET), IEERET Conference Proceeding, 3-4 Nov. 2014.

[6] T. Pie and C. Zukowski," High- speed parallel CRC circuit in VLSI," IEEE Transactions on Communication, vol. 40, no. 4, pp. 653-657, Apr. 1992.

[7] Y. Huo, X. Li, W. Wang and D. Liu," High performance table-based architecture for parallel CRC calculation," In: Local and Metropolitan Area Networks (LANMAN), IEEE International Workshop on, pp. 1-6, Apr. 2015.

[8] P. Anand and Bajarangbali," Design of high-speed CRC algorithm for ethernet on FPGA using reduced Lookup table algorithm," In India Conference (INDICON), IEEE Annual, pp. 1-6, Dec. 2016.

[9] D. Sarwate," Computation of cyclic redundancy check via table lookup," Communication of the ACM, vol. 31, no. 8, pp.1008-1013, Aug. 1988.

[10] Y. Sun and M. Kim," A pipelined CRC calculation using lookup tables," 7th IEEE Consumer Communications and Networking Conference (CCNC), pp. 1-2, 9-12 Jan. 2010.

[11] W. Lu and S. Wong," A fast CRC update implementation," IEEE workshop on high Performance Switching and Routing, pp. 113-120, Oct. 2003.

[12] H. Mathukiya and N. Patel," A novel approach for parallel CRC generation for high speed application," In: Communication Systems and Network Technologies (CSNT), International Conference on IEEE, pp. 581-585, May 2012.

[13] A. Chowdary and K. Swamy," Implementation of 'n' bit parallel CRC using unfolding, retiming and pipelining for high speed applications," Indian Journal of Research, vol. 3, pp. 65-67, Nov. 2014.

[14] S. Singh, S. Sujana, I. Babu and K. Latha," VLSI implementation of parallel CRC using pipelining, unfolding and retiming," IOSR Journal of VLSI and Signal Processing (IOSR-JVSP), vol. 2, no. 5, pp.66-72, May- Jun. 2013.

[15] P. Koopman and T. Chakravarty, "Cyclic Redundancy Code (CRC) polynomial selection for embedded networks," IEEE International conference on dependable systems and networks (ICDSN), pp. 145-154, June 2004.

[16] http://www.ghsi.de/pages/subpages/Online%20CRC%20Calculation/indexDetails.php?