High Performance Table-Based Architecture for Parallel CRC Calculation

Yuanhong Huo, Xiaoyang Li, Wei Wang, Dake Liu, *Senior Member, IEEE*

***Abstract*—A high performance table-based architecture implementation for CRC (cyclic redundancy check) algorithms is proposed. The architecture is designed based on a highly parallel CRC algorithm. The algorithm first divides a given message with any length into bytes. Then it performs CRC computation using lookup tables among the divided bytes in parallel. At last, the results are XORed to obtain the CRC value of the given message. The algorithm is table-based and can accelerate different CRC algorithms. Based on the algorithm, the architecture is designed to accelerate CRC algorithms with high parallelism and flexibility. The architecture is configurable and can support CRC algorithms such as CRC32, CRC24, CRC-CCITT, CRC16, CRC8. CRC value of 128-bit input data can be generated in one cycle. Our method also allows calculation over data that is less than 128-bit wide without increasing hardware cost. With 128-bit input each clock cycle, the throughput of the proposed architecture reaches up to 100 Gbps by utilizing 16 KB SRAM (Static Random Access Memory) with about 12% area reduction compared with previous work.**

***Index Terms*—CRC generation, parallel algorithm, parallel architecture.**

I. Introduction

CRCs are used to detect the corruption of digital content during production, transmission, processing, or storage. CRC algorithms treat each bit stream as a binary polynomial. They calculate the remainder from the division of the stream with a standard generator polynomial. The remainder of this division becomes the CRC value of the data. Then, the remainder is attached to the original data and transmitted to the receiver. At the receiver side, CRC algorithms verify if the correct data packet has been received by checking its remainder.

Broadband communications, such as 100 Gbps Ethernet, need high speed real time CRC computing. It is significant to meet the speed requirement since packets will be dropped if processing is not completed within proper time. Broadband wireless communications need also high speed real time CRC computing to minimize the time for acknowledgement. In order to generate CRC with high throughput at a reasonable frequency and silicon cost, processing multiple bits in parallel is desirable. Parallel CRC is essential for low computing latency required by system specifications of wireless and broadband communications.

Traditionally, CRC calculation is performed utilizing the LFSR (Linear Feedback Shift Register) circuit. LFSR circuit can be implemented in VLSI (Very Large Scale Integration) or be implemented as software running in a processor. Original LFSR can only process one bit each cycle. Recently, to calculate CRC in parallel has been much investigated. Common method such as Table-based algorithm [3][10], Fast CRC update [8], F-matrix based parallel CRC generation [9] and Unfolding, retiming and pipelining algorithm [11] are used to achieve parallelism. However, when the bit-width of input data becomes too large, for example, 128 bit, the circuit critical path delay of the worst case keeps increasing in F-matrix based parallel CRC generation algorithm. The benefits of the parallelization become less significant. The throughput of Unfolding, retiming and pipelining algorithm is far less than 100 Gbps. Fast CRC update algorithms only calculate CRC for the intermediate results of the changed fields. When utilizing traditional Table-based algorithm without any optimization, the required area and power consumption will increase explosively as the degree of parallelism increases.

In this work, we optimize the traditional Table-based CRC algorithm to speed up the CRC calculation while maintaining reasonable area and power consumption. First, a long input bit stream is divided into a set of short bit segments. Then, a set of small tables are designed to calculate the CRC of these short segments. At last, the result of these short segments are XORed to get the CRC of the long input bit stream. Based on the optimized algorithm, a table-based hardware architecture for calculating CRC is proposed. Since the characteristics of different CRC algorithms are similar, they can all be accelerated by the proposed algorithm and architecture with different configuration of the lookup tables. The architecture can support different CRC algorithms by reusing the hardware. By taking advantage of CRC's properties, it is able to calculate the CRC value of 128-bit data in parallel. Experimental results show that the throughput of CRC algorithms such as CRC32, CRC24, CRC-CCITT, CRC16 etc can be up to 100 Gbps utilizing the proposed architecture.

The rest of this paper is structured as follows: Section II

Y. Huo, is with School of Computer Science and Technology, Beijing Institute of Technology, Beijing, China. (e-mail: hyh@bit.edu.cn).

X. Li, W. Wang and D. Liu are with School of Information and Electronics, Beijing Institute of Technology, Beijing, China. (e-mail: hsiaoyonglee@gmail.com; 3120130295@bit.edu.cn; dake@bit.edu.cn).

978-1-4673-6762-2/15/$31.00 ©2015 IEEE

provides a survey of related work on CRC algorithms. Section III presents the proposed parallel CRC algorithm. The parallel architecture is introduced in Section IV. Section V evaluates the proposed algorithm and architecture. In Section VI some concluding remarks are provided.

II. Related Work

There have been successful architectures to accelerate CRC algorithms. The researches can be divided into two categories: serial CRC and parallel CRC. Paper [1] is one of the most popular implementation of serial CRC. In its implementation, a simple circuit based on shift registers performs the CRC calculation by handling the message one bit at a time. Processing one bit per cycle greatly limits the throughput of CRC circuits. So there are many researches generate CRC in parallel. These techniques include:

1) Table-Based algorithm
2) Fast CRC update
3) F-matrix based parallel CRC generation
4) Unfolding, retiming and pipelining algorithm

Table-Based algorithm provides lookup table (LUT) for various input bit stream and can obtain high throughput. It should pre-calculate CRC value and store it in LUT. It's necessary to change LUT when the polynomial is changed. In paper [10], a framework for designing a family of fast cyclic redundancy code generation algorithms is presented. Their algorithms can read 32 and 64 bits of data each time, while optimizing their memory requirement to meet the constraints of specific computer architectures. Their algorithms can also be implemented in software using commodity processors instead of specialized parallel circuits. When implemented on Intel IA32 processor, the throughput can be 2.19 cycles per byte. In paper [3], the authors first chunk the message into blocks, each of which has a fixed size equal to the degree of the generator polynomial. Then they perform CRC computation using lookup tables instead of LFSRs. The results are combined together by XOR operations. Utilizing their method, the throughput can reach 56.21 Gbps and the silicon area is 14587 μm^2 at 130 nm CMOS technology.

In fast CRC update technique, it is not necessary to calculate CRC each time for all the data bits, instead, one only needs to calculate CRC for those bits that are changed. Paper [8] presents this method. They calculate the intermediate results of the changed fields using the parallel CRC calculation and performs a single step update afterwards. Consequently, the number of cycles needed to recalculate the CRC codes is greatly reduced. They estimate that the theoretical throughput can reach about 56 Gbps assuming the frame size distributions are realistic.

In F-matrix based parallel CRC generation, data input and each element of F-matrix generated from given generator polynomial are ANDed, then the result will be XORed with present state of CRC checksum. Paper [9] is one of the best in this field. It presents 64 bit parallel CRC architecture based on F-matrix with the order of generator polynomial is 32. When processing 64 bytes data, their method costs 9 clock cycles.

Retiming is used to increase clock rate of circuit by reducing the computation time of critical path. For example, paper [9] applies unfolding technique to pipelined architecture to increase the throughput of the circuit. Then it applies retiming to the architecture to reduce the critical path delay. Paper [9] also points out that the design is not efficiently applicable for the LFSR architecture of any generator polynomial. It is efficient for the generator polynomials with many zero coefficients between the second and third highest order nonzero coefficients. When processing the same input data, the serial implementation of CRC-9 costs 9 clock cycles while their method costs 5 clock cycles without increasing the hardware cost.

There are advantages and disadvantages for each technique to generate CRC value. Fast CRC update technique requires extra memory to store the old CRC value and data. Unfolding architecture increases the number of iteration bound and the parallel degree is limited. The F-matrix based architecture can improve the parallel degree greatly. But when the input data is more than 64 bit long, the delay of the XOR tree can be the critical path of the circuit. To accelerate CRC generation at the parallel degree of 128 bit per cycle, a new table-based architecture is recommended. By optimizing the CRC algorithms, the proposed architecture can achieve the throughput of 100 Gbps. It can also accelerate common CRC algorithms such as CRC8, CRC16, CRC-CCITT, CRC24, CRC32 by configuring the architecture.

III. Proposed CRC Algorithm

CRC is widely used due to its capability to detect the accidental error of data. When data is stored on or transmitted through media, it can be modified by many reasons, including Gaussian noise, hard drive malfunctions, and faulty physical connections. CRC typically demonstrates a good Hamming distance [11]. They are suitable for such error detection.

Mathematically, CRC value is computed for a fixed-length bit stream. CRC algorithms treat each bit stream as a binary polynomial $A(x)$ and calculate the remainder $R(x)$ from the division of $A(x)$ with a standard "generator" polynomial $G(x)$. For a n bit message $a_{n-1}a_{n-2}...a_0$, it can be treated as a polynomial as follows:

$$A(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + ... + a_1x + a_0 \tag{1}$$

where a_{n-1} is the Most Significant Bit (MSB) and a_0 is the Least Significant Bit (LSB) of the message. Given the degree $m-1$ generator polynomial,

$$G(x) = g_{m-1}x^{m-1} + g_{m-2}x^{m-2} + ... + g_0 \tag{2}$$

where $g_{m-1} = 1$ and $g_i \in \{0,1\}$ for all $0 \le i \le m-2$, $A(x)$ is multiplied by x^{m-1} and divided by $G(x)$ to find the remainder.

$$CRC(A(x)) = R(x) = A(x)x^{m-1} \bmod G(x) \tag{3}$$

```
crc = INIT_VALUE;
while(p_buf < p_end) {
  crc  = (crc >> 8) ^ table[(crc ^ * p_buf++) & 0x000000FF];
}
return crc ^ FINAL_VALUE;
```

Fig. 1. The Sarwate algorithm

CRC value of the message is defined as the coefficients of the remainder polynomial. After CRC processing is completed, the binary words corresponding to $R(x)$ are transmitted together with the bit stream associated with $A(x)$. At the receiver side, CRC algorithms check whether $R(x)$ is the correct remainder. The division is performed using modulo-2 arithmetic. Additions and subtractions are "carry-less" in modulo-2 arithmetic. In this case, additions and subtractions are equal to the XOR logical operation.

Traditional table-based CRC calculation always take 4-bit or 8-bit data as input. The Sarwate algorithm [4] is one of the most popular ones. By performing an XOR operation between the least significant byte of the current CRC value and a new byte from the input data and by performing a table lookup, the Sarwate algorithm determines how the current CRC value is modified when a new byte is taken into consideration. The lookup table used by the Sarwate algorithm stores the remainders from the division of all possible 8-bit numbers shifted to the left by 32 bits with the generator polynomial. Paper [4] and [5] present the detailed justification and proof of correctness of the Sarwate algorithm. The Sarwate algorithm can be described as Fig. 1.

The serial computation described above can be rearranged into a parallel configuration. Two theorems to achieve parallelism in CRC computation are used.

Theorem 1: Let $A(x) = A_1(x) + A_2(x) + ... + A_n(x)$ over $GF(2)$. Given a generator polynomial, $CRC(A(x)) = \sum_{i=1}^{n} CRC(A_i(x))$

Theorem 2: Given $B(x)$, a polynomial over $GF(2)$, $CRC(x^k B(x)) = CRC(x^k CRC(B(x)))$ for any k.

Both theorems can be easily proved using the properties of $GF(2)$. The proof of these theorems can be seen in paper [3] and [10]. Fig. 2 shows Theorem 1 more intuitively.

Based on the two theorems discussed above, an algorithm that can process N bytes data in parallel is proposed. The algorithm is designed as follows.

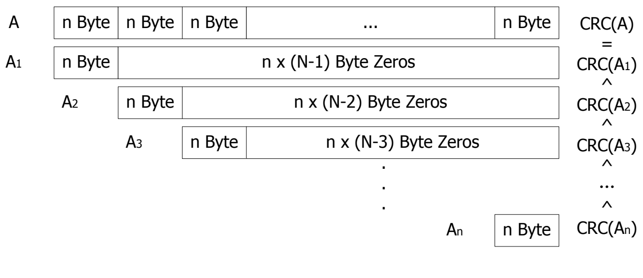

Fig. 2. Optimized CRC calculation

```
crc = INIT_VALUE;
while(p_buf < p_end) {
  crc  ^= *(uint32_t *)p_buf;
  p_buf += 4;
  tmp[0] = table[15][crc & 0x000000FF];
  tmp[1] = table[14][(crc >> 8) & 0x000000FF];
  tmp[2] = table[13][(crc >> 16) & 0x000000FF];
  tmp[3] = table[12][(crc >> 24) & 0x000000FF];
  for(i=4; i<16; i++) {
     tmp[i] = table[15-i][(*p_buf) & 0x000000FF];
     p_buf++;
   }
   for(i=0; i<16; i++) {
     crc ^= tmp[i];
   }
}
return crc ^ FINAL_VALUE;
```

Fig. 3. The proposed parallel CRC algorithm

1) divide the input bit stream into a series of n byte blocks, considering the size of lookup table increases greatly as the number of bit of each block increases, each block is designed with 8-bit data. And each lookup table contains 2^8=256 items.
2) XOR CRC value with the first 4 bytes of input data;
3) utilize the 4-byte result of step 2) and the remained $N-4$ bytes as the addresses of lookup tables and N CRC values are obtained;
4) XOR these N CRC values and the CRC value of the N bytes input data can be figured out.

The C language description of the proposed algorithm when N is equal to 16 is listed as Fig. 3. The value in "table[n]" is the CRC value of 8-bit data appended with $n \times 8$ bit zeros. The tables can be calculated beforehand and used for parallel CRC generation when the "generator" polynomial $G(x)$ is determined. The tables can be figured out using traditional algorithm according to paper [1].

In this way, M bit input bit stream can be calculated within $\lceil M/(N \times 8) \rceil$ rounds. Each round $N \times 8$ bit data are processed, when the n bit input bit stream is less than $N \times 8$ bit, it can be solved by prepending $N \times 8 - n$ bit zeros to the input data and the result can be figured out correctly. When N is equal to 16, the algorithm divides 128-bit input data into 16 bytes and utilizes 16 tables to process these 16 bytes in one iteration. These 16 tables cost $16 \times 256 \times 32$ bit memory, which is much less than $2^{128} \times 32$ when utilizing traditional table-based algorithm. So the proposed algorithm can achieve high parallelism with acceptable hardware cost.

IV. Proposed CRC Architecture

The algorithms introduced in section III can be implemented using general purpose processor (GPP). When using GPP, paper [3] shows processing each byte costs at least 2.19 cycles. In fact, if 16 bytes are input, the lookup tables can be accessed

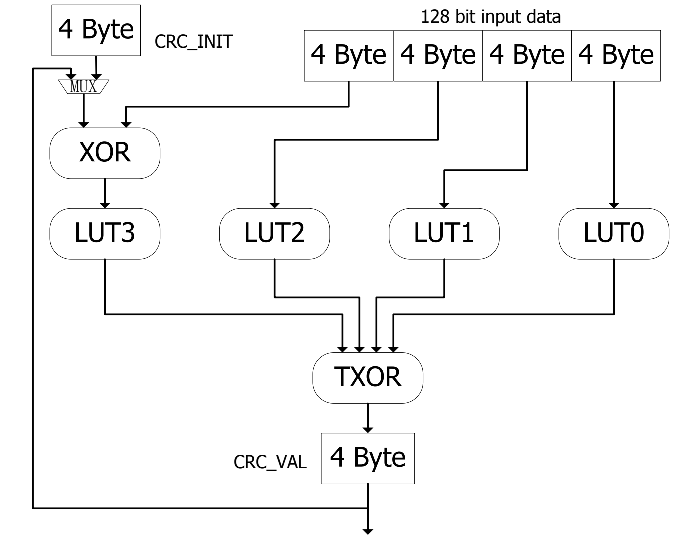

Fig. 4. parallel CRC architecture

by these 16 bytes in parallel. Then, the 16 outputs of lookup tables can be XORed to generate one CRC value. If these steps can be finished in one cycle, 16 bytes can be processed within one cycle.

To calculate the CRC value of 128-bit data in one cycle, a highly parallel architecture is proposed. The architecture costs 16 KB SRAM to store 16×256×32 bit keywords and several XOR gates to update CRC value. The architecture can accelerate CRC32 as well as other CRC algorithms such as CRC8, CRC16, CRC-CCITT, CRC24, etc. In the rest of this section, we will first discuss how CRC32 is accelerated. Then, we will explain how other CRC algorithms can be accelerated by reusing the architecture.

A. Parallel Architecture

Fig. 4 shows the proposed architecture. The inputs include 128-bit data to be calculated and 32-bit CRC value. The output is 32-bit new CRC value. The data is arranged according to little endian pattern. In the first place, the first four bytes of the input data are XORed with CRC value and four new bytes are obtained. Then, the four bytes and the remained 12 bytes of input data are acted as addresses for the 16 SRAMs, as Fig. 4 shows, LUT0-3 contain 4 SRAMs each and every SRAM contains 256 32-bit keywords. At last, the outputs of these 16 SRAMs are passed to TXOR module. The datapath of TXOR is shown in Fig. 5. In the TXOR module, these 16 32-bit words are XORed and the new CRC value can be figured out. All these works can be completed within one cycle. The "MUX" is used to choose CRC initial value in the beginning. Otherwise, it will choose the output of the TXOR module.

B. Last Iteration

Since the proposed architecture processes 128-bit data every cycle, it is necessary to consider how to accelerate the last input block as the last block may contain less than 128-bit data in some case. As the CRC value of zero is always zero, the problem can be solved by prepending zeros to the last block if it contains less than 128-bit data. As any data XORed with zero remains unchanged, the result is correct on the proposed architecture. In this way, we don't need extra hardware or clock cycles. In this paper, we assume the number of bit of the last block is no less than the number of bit of the generator polynomial. For example, when processing CRC32, the last block should contain no less than 32-bit data.

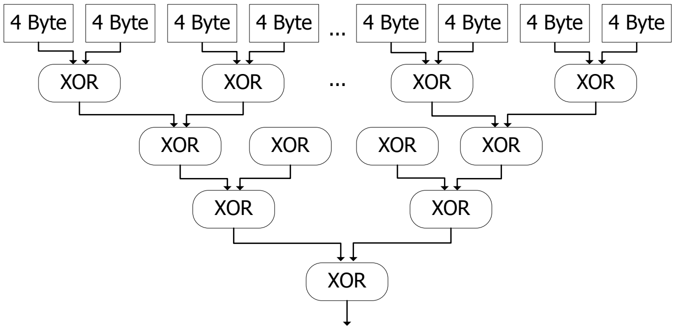

Fig. 5. The datapath of TXOR

C. Compatibility for Other CRC Algorithms

The architecture proposed can not only accelerate CRC32 but also accelerate other CRC algorithms. The calculation of different CRC algorithms are similar except the content of lookup tables is different and the bit width of CRC value is different. By using our method, these problems can be solved by configuring the SRAMs with different CRC keyword tables and prepending zeros to the CRC value that is less than 32-bit wide. Take CRC24 as an example, 24-bit CRC value takes up the lower 24 bits of 32-bit "CRC_VAL" shown in Fig. 4 and the higher 8 bits store zeros. Firstly, the first 4 bytes of the input data is XORed with "CRC_VAL" and three new bytes are obtained while the fourth byte remains. Secondly, the 4 bytes and the remained 12 bytes of input data are used as addresses for the 16 SRAMs. The SRAMs store 24-bit keywords of CRC24 and output corresponding CRC keywords of the 16 input bytes. Lastly, the outputs of these 16 SRAMs are passed to the TXOR module. The output of TXOR is 32-bit CRC whose higher 8 bits are ignored. Other CRC algorithms with less number of bit can be processed in a similar way.

V. EVALUATION

The proposed architecture is synthesized in verilog implementation and three previous algorithms are selected for comparison: a typical parallel CRC algorithm [6], a pipelined CRC algorithm [7] and a table-based CRC algorithm [10]. These three hardware based approaches are chosen because they are more comparable: [6] is a commonly used parallel CRC calculation algorithm, [7] is one of the most efficient CRC calculation algorithms proposed recently and [10] is the most similar method to ours.

TABLE I. COMPARISON BETWEEN PROPOSED ALGORITHM AND EXISTING ALGORITHMS

Algorithms	Maximum Frequency (MHz)	Power (mW)	Area (Scaled to 65 nm) (μm^2)	Throughput (Gbps)	Area Efficiency (Gbps/ $k\mu m^2$)	Normalized Throughput (bit/cycle)	Comments	
							Input Bit Width	Technology
Ordinary CRC algorithm in [6]	347.8	NA	3576 (894)	20.68	23.13	59.46	64	130 nm CMOS
Pipelined CRC algorithm in [7]	561.1	NA	16494 (4124)	35.28	8.55	62.88	64	130 nm CMOS
Table based CRC algorithm in [10]	878.3	NA	14587 (3647)	56.21	15.41	64	64	130 nm CMOS
Proposed algorithm	833.3	3.79	3261 (3261)	106.66	32.71	128	128	65 nm CMOS

TABLE II. CRITICAL PATH AND LOOKUP TABLE SIZE AS THE THROUGHPUT INCREASES

Lookup table size (KB)	Critical path	Throughput (bytes/cycle)
8	$D_{LUT}+4D_{XOR}$	*8*
16	$D_{LUT}+5D_{XOR}$	*16*
32	$D_{LUT}+6D_{XOR}$	*32*
64	$D_{LUT}+7D_{XOR}$	*64*

The comparison result of our method and three previous algorithms is shown in TABLE I. The result of three previous algorithms are got at the condition of 1.2V, 130 nm CMOS technology and the bit width of input data is 64. The proposed method is experimented using 1.2V, 65 nm CMOS technology and the bit width of input data is 128. The second column in the left is the maximum frequency of different method. Column three shows the power of the proposed architecture. Since the power of SRAM depends greatly on the manufacturer, the result doesn't include the power of SRAM. The fourth column shows the silicon area used. The areas in parentheses are the areas scaled to 65 nm CMOS technology. They are figured out by dividing the area obtained at 130 nm CMOS technology by 4. The fifth column shows the throughput of these methods. Column six shows the area efficiency of different methods, it is obtained by dividing the throughput by the scaled area. The seventh column shows the normalized throughput, they are calculated by dividing the throughput by the maximum frequency. The last column shows the experimental condition of these algorithms. The architecture introduced by [6] utilizes less silicon area than ours but the proposed method achieves the best area efficiency among all these methods.

Among the previous works, the result of paper [10] has the best throughput. The proposed architecture achieves better throughput since it can process 128-bit data each cycle. The SRAM used in [10] is 8 KB while in this work it is 16 KB. When the cost of SRAMs is the same as [10], the silicon area can be decreased 12% using the proposed method since no specialized hardware is utilized for computing the CRC value of input data when it is less than 128 bit. Besides, the proposed architecture can support different CRC algorithms while [10] only supports CRC32.

The algorithm proposed can be configured to process different number of bytes each cycle. TABLE II. shows the hardware cost increase linearly as the throughput increases. The second column in TABLE II. is the critical path delay of different situation. D_{LUT} is the delay of lookup table and D_{XOR} is the delay of XOR operation. Using the proposed algorithm, the depth of XOR tree increases as the number of input data increases. Let N be the number of input data, the delay of XOR tree D_{XOR_tree} can be described as:

$$D_{XOR\_tree} = D_{XOR} \times \log_2(N) \qquad (4)$$

Users can choose different configuration according to the requirements of throughput and the limit of lookup table size.

VI. CONCLUSION

In this paper, a new fast cyclic redundancy check algorithm using lookup tables instead of linear feedback shift registers is proposed. The algorithm can calculate different CRC algorithms of any length of message in parallel. Based on the algorithm, we further proposed a parallel hardware architecture. The gate count is less than 1 K and the total memory cost is 16 KB. Introducing SRAMs to fulfill LUT function in parallel, the architecture achieves considerably higher throughput than existing serial or byte-wise lookup table CRC algorithms. The architecture can be executed under 833.33 MHz and we thus achieved the performance of 128 bit× 833.33 MHz = 106.66 Gbps. Compared with other table-based methods, the proposed method consumes less silicon area by reusing hardware. It can also support different CRC algorithms by configuring the SRAMs with different lookup tables. The throughput can be improved further by increasing parallelism utilizing more SRAMs for LUT with small delay increase in the critical path.

ACKNOWLEDGMENT

The finance supporting from National High Technical Research and Development Program of China (863 program) 2014AA01A705 is sincerely acknowledged by authors.

REFERENCES

[1] S. L. Shieh, P. N. Chen, and Y. S. Han, “Flip CRC modification for message length detection,” IEEE Transactions on Communications, vol. 55, no. 9, pp. 1747–1756, 2007.

[2] T. V. Ramabadran and S. S. Gaitonde, “A tutorial on CRC computations,” IEEE Micro, vol. 8, no. 4, pp. 62–75, Aug. 1988.

[3] M. E. Kounavis, and L. B. Frank. “Novel table lookup-based algorithms for high-performance CRC generation.” Computers, IEEE Transactions on 57.11 (2008): 1550-1560, 2008.

[4] D. V. Sarwate, “Computation of Cyclic Redundancy Checks via Table Lookup,” Comm. ACM, vol. 31, no. 8, pp. 1008-1013, Aug. 1988.

[5] R. N. Williams, “A Painless Guide to CRC Error Detection Algorithms,” technical report, http://www.ross.net/crc, Aug. 1993.

[6] A. Simionescu, “CRC tool computing CRC in parallel for ethernet.” (2001): 1-5, 2001.

[7] Y. Sun and M. S. Kim, “A pipelined crc calculation using lookup tables,” in Proceedings of IEEE Consumer Communications and Networking Conference (CCNC), Jan. 2010.

[8] W. Lu and S. Wong, “A Fast CRC Update Implementation”, IEEE Workshop on High Performance Switching and Routing ,pp. 113-120, Oct. 2003.

[9] H. H. Mathukiya , and M. P. Naresh. “A Novel Approach for Parallel CRC generation for high speed application.” Communication Systems and Network Technologies (CSNT), 2012 International Conference on. IEEE, 2012.

[10] Y. Sun, and S. K. Min. “A table-based algorithm for pipelined CRC calculation.” Communications (ICC), 2010 IEEE International Conference on. IEEE, 2010.

[11] S. Sangeeta, et al. “VLSI Implementation of Parallel CRC Using Pipelining, Unfolding and Retiming.” IOSR Journal of VLSI and Signal Processing 2.5 (2013): 66-72. 2013

[12] G. Castagnoli, S. Brauer, and M. Herrmann, “Optimization of Cyclic Redundancy Check Codes with 24 and 32 Parity Bits”, IEEE Trans. Comm., vol. 41, no. 6, pp. 883-892, 1993.