International Journal of Modern Engineering Research (IJMER) Vol.2, Issue.4, July-Aug. 2012

Cyclic Redundancy Check Generation Using Multiple Lookup Table Algorithms

ABSTRACT: The primary goal of this paper is to generate cyclic redundancy check (CRC) using multiple lookup table algorithms. A compact architecture of CRC algorithm (Slicing-by-N algorithm) based on multiple lookup tables (LUT) approach is proposed. This algorithm can ideally read large amounts of data at a time, while optimizing their memory requirement to meet the constraints of specific computer architectures. The focus of this paper is the comparison of two algorithms. These two algorithms are Slicing by-N-algorithm and Sarwate algorithm, in which slicing by-N-algorithm can read arbitrarily 512 bits at a time, but Sarwate algorithm, which can read only 8 bits at a time. This paper proposes the generation of CRC using slicing by 8 algorithm. In this, message bits are chunked to 8 blocks. All are processed at a time. Proposed Slicing-by-8 algorithm can read 64 bits of input data at a time and it doubles the performance of existing implementations of Sarwate algorithm.

Keywords: CRC, LUT, Slicing-by-N.

1. Introduction

Cyclic Redundancy Check (CRC) is one of the methods of detecting the errors in the information during transmission. CRC is an error-checking code that is widely used in data communication systems and other serial data transmission systems [10]. CRC are used for detecting the corruption of digital content during its production, transmission, processing or storage. CRC algorithms treat each bit stream as a binary polynomial and calculate the remainder from the division of the stream with corresponding to the remainder is transmitted together with the bit stream. At the receiver side, CRC algorithms verify that the correct remainder is has been received. Long division is performed using modulo-2 arithmetic [3]. CRC is a polynomial-based block coding method for detecting errors in blocks or frames of data. A set of check digits is computed for each frame scheduled for transmission over a medium that may introduce error and is appended to its end. The computed check digits are known as the frame check sequence (FCS). A CRC value is calculated as a remainder of the modulo-2 division of the original transmitted data with a specific CRC generator polynomial. For example, Ethernet uses the 32-bit polynomial value,

$G(x) = 1 + x + x2 + x4 + x5 + x7 + x8 + x10 + x11 + x12 + x16 + x22 + x23 + x26 + x32$ (1) To find the FCS, first a number of zeroes equal to the number of FCS digits to be generated are appended to the message M(x). This is equivalent to multiplying M(x) by 2n, where "n" is the number of FCS digits. This value is then divided by the generator polynomial G(x) (1), which contains one more digit than the FCS. The division uses modulo-2 arithmetic, where each digit is independent of its neighbour and numbers are not carried or borrowed, thus additions and subtractions are performed via an exclusive-OR (XOR) function. The remainder R(x) is appended to the end of the message before transmission. At the receiver, the message plus the FCS is divided by the same polynomial. If the remainder is zero then it can be assumed that no error has occurred [2]. To accelerate the CRC generation process, a number of algorithms have been proposed. Among these algorithms the most commonly used today is the algorithm proposed by Sarwate. The Sarwate algorithm reads 8 bits at a time from stream and calculates the CRC value by performing lookups on a table of 256 32-bit entries [4]. Looking to overcome limitations of processing 8 bits of data at a time, new algorithm have been proposed [4] and they can read arbitrarily large amount of data at a time. Recently time is the major concern. So in order to process large amount of data at a time, Multiple Lookup based approach is more efficient. Multiple Lookup based approach contains five CRC algorithms, called Slicing by-N algorithm ($N \epsilon 4, 8, 16, 32, 64$), which is used to read up to 512 bits at a time. So performance of the system should be increased. Here proposing Slicing by-8 algorithm to read 64 bits at a time. Here proposed an efficient design of CRC generator using Slicing by-N algorithm (N=8). In this algorithm, input message stream is sliced into N slices and each slice has 8 bits. So using this Slicing by-8 algorithm, it can read 64 bits at a time and it triples the performance of existing implementation of Sarwate algorithm. In this algorithm, input data stream is sliced into 8 slices. Each slice has 8 bits and these total 64 bits are processed at a time. In this design 8 Look Up tables (LUT) are used, which contain the pre-computed CRC values. These CRC values are generated by using LFSR method and 256 combinations of CRC values corresponding to 8 bit input stream are generate. In this work CRC32 standard is used. Each slice returned

CRC values from each LUT and all are XORed to get the final CRC value. This algorithm can reduce the memory usage and also number of operations significantly reduces compared to Sarwate algorithm.

2. System Architecture

Cyclic Redundancy Check is an error detecting codes that are widely used due to their capability to detect the alteration of data. Different algorithms are used for generating CRC. Existing algorithms are LFSR method and Sarwate algorithm. Here proposing an algorithm, this has five algorithms, called Slicing by-N algorithm which is used to generate CRC fastly. Every CRC algorithm treat input bit stream as a binary polynomial and calculate the remainder from the division of the stream with standard generator polynomial. In this work CRC32 standard is used as a generator polynomial. The binary words corresponding to the remainder are transmitted together with input stream. In this work slicing by-8 algorithm is used to generate CRC and it can read 64 bits at a time. The main disadvantage of existing table-driven CRC generation algorithms is their memory space requirement when reading a large number of bits at a time. To solve this problem, a new algorithm that slices the CRC value produced in every iteration as well as the data bits read into small terms. These terms are used as indexes for performing lookups on different tables in parallel. For example, Slicing-by-4 algorithm can read 32 bits of input data at a time and it doubles the performance of existing implementations of Sarwate algorithm, while Slicing-by-8 triples the performance and reads 64 bits of input data at a time. In this way, here proposed algorithm is capable of reading 64 bits at a time, as opposed to 8, while keeping its memory space requirement to 8KB.

2.1 General Block Diagram of CRC Generation

The basic block diagram of CRC generator is shown in fig.1.

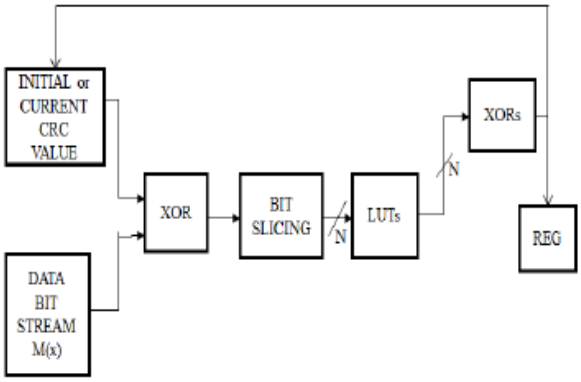

Fig.1 Block diagram of basic CRC Generator.

In this CRC generator, the data stream is firstly XORed with initial CRC value. In this work CRC32 standard is used as a generator polynomial. In the case of CRC32 standard, the initial CRC value is 0xFFFFFFFF [1] and other cases, that is, in other CRC standard initial CRC value is cleared. The output of the XOR block is then sliced into N slices and each slice has 8 bits. Each is given to each LUT for calculating the CRC value. Numbers of LUTs are same to N. Using this Slicing by-N algorithm up to 512 bits are processed at a time, that is, this Slicing by-N algorithm composed of five algorithms (N ϵ 4, 8, 16, 32, 64) in which fifth algorithm is used to read 512 bits at a time.

2.2 Block Diagram of CRC Generator Using Slicing By8

Here proposing CRC generation using slicing by-8 algorithm which is used to read 64 bits at a time. The block diagram of Slicing by-8 algorithm is shown in fig.2 is

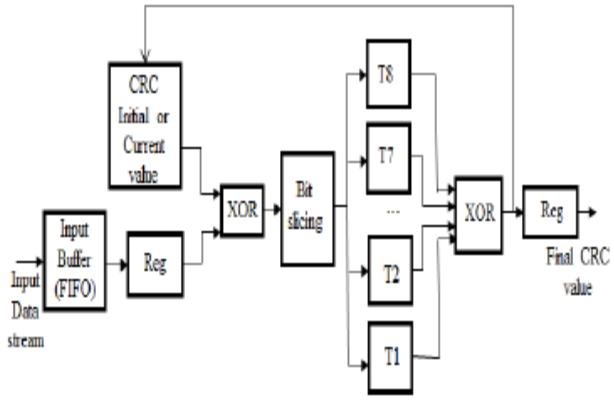

Fig.2 Block diagram of CRC Generator using Slicing by-8 Algorithm

In this, primarily the input data stream is stored in Buffer and buffer has size five and each location is 64 bit wide. Initially first 64 bit data is taken and it is XORed with initial CRC value. Here using CRC32 standard and its initial value is 0xFFFFFFFF. So the modified bit stream is got. This bit stream is sliced into 8 slices and each has 8 bit long. Each 8 bit data is given to each LUT. LUTS have the pre-computed CRC values. These LUTs have 256 entries with 32 bit wide, that is, 8 bit data has 28 (= 256) combinations of values, each has its own CRC value. The CRC value corresponding to each bit stream is getting from each LUT. Finally all CRC values are XORed to get the final CRC value.

2.2.1 Slicing-by-8 Algorithm Descriptions

In this CRC generation process the pre computed value of CRC first stored in LUTs. The long division process is to pre-compute the current remainder that results from a group of bits and place the result in a table. Before the beginning of the long division process all possible remainders which result from groups of bits are pre-computed and placed into a Look up Table. In this way, several long division steps can be replaced by table lookup step. The benefit from slicing comes from the fact that modern processor architectures comprise large cache units. These cache units are capable of storing moderate size tables. If tables are stored in an external memory unit, the latency associated with accessing these tables may be significantly higher than when tables are stored in a cache unit. Slicing is also important because it reduces the

number of operations performed for each byte of an input stream when compared to Sarwate. For each byte of an input stream the Sarwate algorithm performs the following: (i) an XOR operation between a byte read and the most significant byte of the current CRC value; (ii) a table lookup; (iii) a shift operation on the current CRC value; and (iv) an XOR operation between the shifted CRC value and the word read from the table. In contrast, for every byte of an input stream the Slicing by- 8-algorithm performs only a table lookup and an XOR operation. This is the reason why the Sicing-by-8 algorithm is faster than the Sarwate algorithm.

Slicing by-N algorithm says that the current input stream is XORed with the current CRC and then the modified stream is produced. This modified stream is then sliced into N slices and each has one byte wide. Each slice is given to the LUTs, which have the pre-computed CRC values. Output of these LUTs is then XORed to get the current CRC. Initially CRC value is 0xFFFFFFFF. Here slicing by-8 algorithm is explained. The steps of this algorithm as follows:

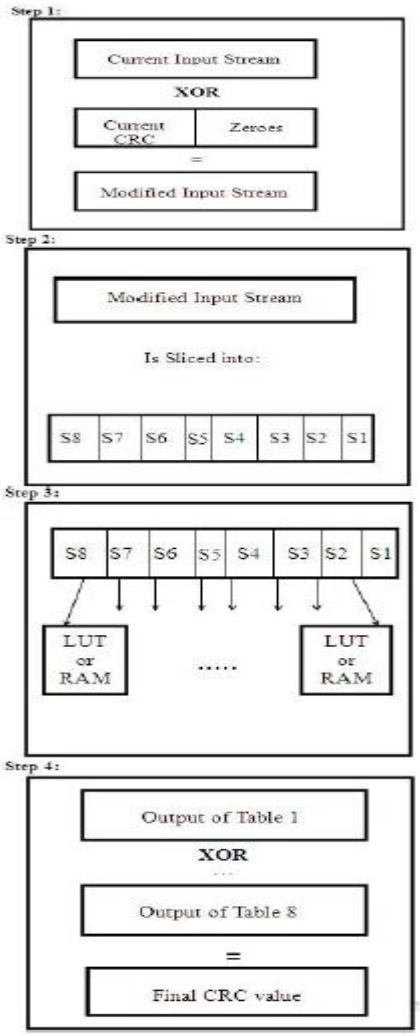

Fig.3 The Slicing by-8 Algorithm

These steps explain how this algorithm works in CRC generators. In the first step, the current input stream is XORed with the current CRC value. The input stream is 64 bit wide and CRC value is 32 bit wide. In order to perform XOR operation, 32 bit wide zeroes is appended to the current CRC value and is XORed with input stream. The modified bit is produced as output. In the second step this modified bit stream is sliced into eight slices. In the third step, each slice is given to LUTs as input and CRC value corresponding to input is produced from LUTs. In the fourth step, these outputs are XORed to get final CRC value. is given to the LUTs, which have the pre-computed CRC values. Output of these LUTs is then XORed to get the current CRC. Initially CRC value is 0xFFFFFFFF. Here slicing by-8 algorithm is explained. The steps of this algorithm as follows: Fig.3 The Slicing by-8 Algorithm These steps explain how this algorithm works in CRC generators. In the first step, the current input stream is XORed with the current CRC value. The input stream is 64 bit wide and CRC value is 32 bit wide. In order to perform XOR operation, 32 bit wide zeroes is appended to the current CRC value and is XORed with input stream. The modified bit is produced as output. In the second step this modified bit stream is sliced into eight slices. In the third step, each slice is given to LUTs as input and CRC value corresponding to input is produced from LUTs. In the fourth step, these outputs are XORed to get final CRC value.

2.3 Block Diagram of CRC Generator Using Sarwate Algorithm

Here explains the CRC generation using Sarwate algorithm which is used to process only 8 bit at a time. A more efficient approach to CRC computation in software was described by Sarwate [2]. This technique uses a table of pre computed effects on the shift register of 8-bit bytes, which allows the computation to run at one cycle per byte (instead of one cycle per bit) [4]. The long division process is a compute-intensive operation because it requires in the worst case one shift operation and one XOR logical operation for every bit of a bit stream. Most software-based CRC generation algorithms, however, perform the long division process quicker than the bit-by-bit marking technique. One commonly used technique for accelerating the long division process is to pre-compute the current remainder that results from a group of bits and place the result in a table. In this way, several long division steps can be replaced by a single table lookup step [3] [6].

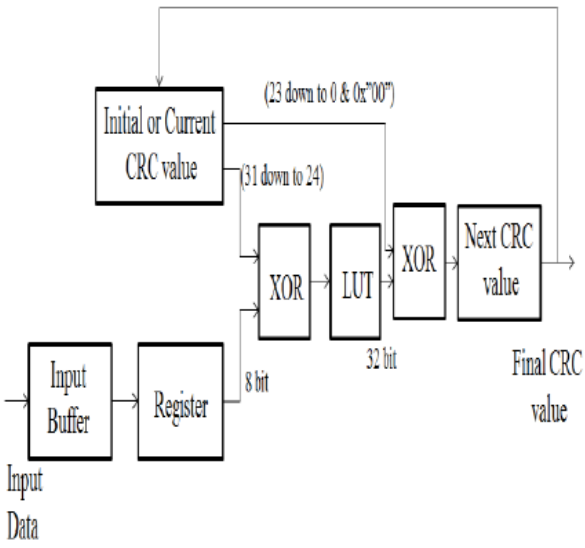

Fig.4 Block diagram of CRC Generator using Sarwate Algorithm

In both Sarwate algorithm and Slicing by 8 algorithms, Look Up Table is used for storing the 2_8 combinations of CRC values. These CRC values are computed using long division process. Long division steps are explained above. Initially the CRC value is set to 0XFFFFFFFF. The first 8 bit from the input stream is taken and is XORed with most significant 8 bits of the initial CRC value. This modified byte is given to the LUT, it produce the corresponding CRC value. It is finally XORed with the 24 least significant bits of the current CRC value, shifted by 8 bit positions to the left, which produce the next CRC value and this CRC value is used for next iteration.

2.3.1 Sarwate Algorithm Description

The most representative table-driven CRC generation algorithm used today is the algorithm proposed by Dilip.V.Sarwate. The length of the CRC value generated by the Sarwate algorithm is 32 bits. The Sarwate algorithm is more complicated than the straightforward lookup process because the amount of bits read at a time (8 bits) is smaller than the degree of the generator polynomial. Initially, the CRC value is set to a given number which depends on the standard implemented (e.g., this number is 0xFFFFFFFF for CRC32) [1]. For every byte of an input tream the algorithm performs the following steps:

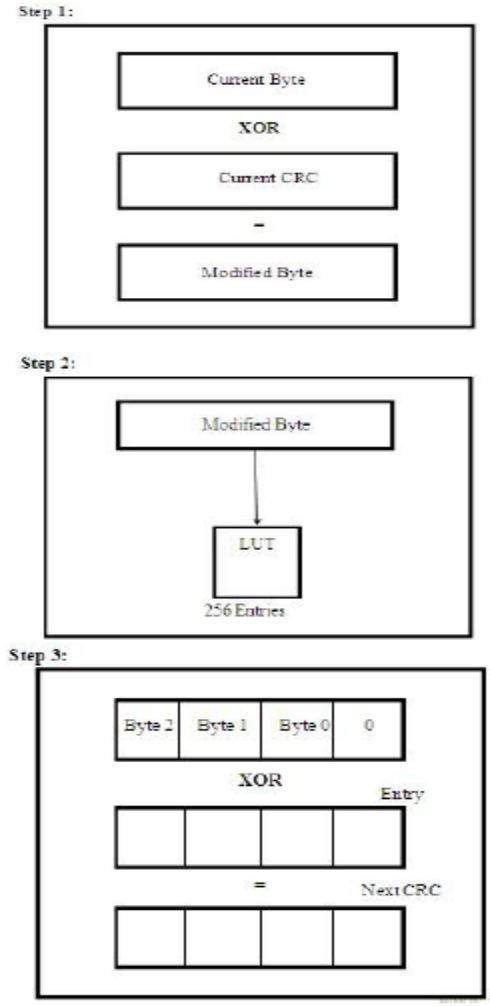

Fig.5 Sarwate Algorithm

In first step, the algorithm performs an XOR operation between the most significant byte of the current CRC value and the byte from the stream which is read. The 8-bit number which is produced by this XOR operation is used as an index for accessing a 256 entry table (Step 2). The lookup table used by the Sarwate algorithm stores the remainders from the division of all possible 8-bit numbers shifted by 32 bits to the left with the generator polynomial. The value returned from the table lookup is then XORed with the 24 least significant bits of the current CRC value, shifted by 8 bit positions to the left (Step 3). The result from this last XOR operation is the CRC value used in the next iteration of the algorithm"s main loop. The iteration stops when all bits of the input stream have been taken into account. Sarwate algorithm has been designed when computer architectures supported XOR operation with only eight bits. Today"s processors support operations with 32 and 64 bits values, and if this algorithm is extended it would require lookup tables of 2_{32} = 4G entries for processing 32 bits of data at a time, and 2_{64} = 16G for 64 input data. These tables

can‟t fit into a cache and would cause significant latency problem if in RAM.

3. Simulation Results

The simulation was done using Modelsim PE 10.0c Simulator and the output waveforms is obtained as shown in fig. 6, 7,8,9,10. Slicing by-N algorithm is one of the CRC algorithms, which is used to read up to 512 bits at a time. It is Multiple Look Up based approach that is, in this N LUTs are used. In this paper, Slicing by-8 algorithm is used which it is used to read 64 at a time. Each 64 bits are sliced into 8 slices and each slice has 8 bits. In this algorithm 8 LUTs are used. Each slice is given to each LUT and these LUTs contain all 28 combinations of CRC values. Outputs of LUTs are CRC values corresponding to each slice and all outputs from LUTs are XORed to get final CRC value. Similarly Sarwate algorithm is used to read 8 bits at a tme and in this, only one LUT is used for iteration.

3.1 Simulation waveforms

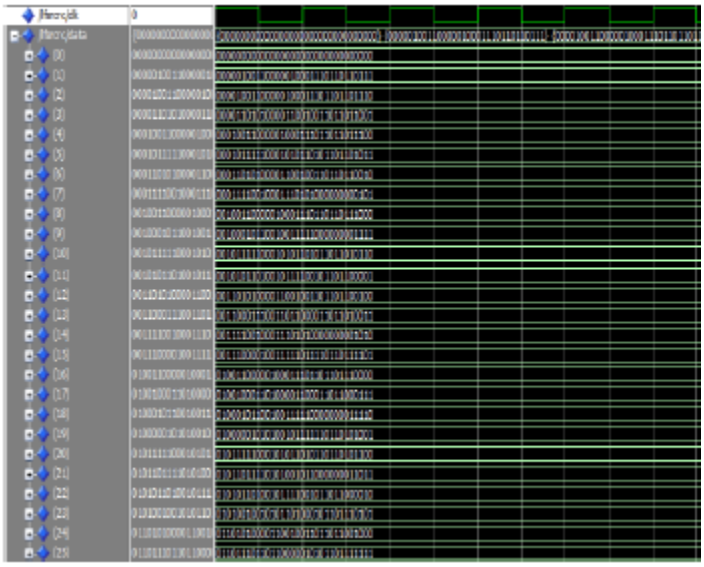

Fig.6 256 Combinations of CRC Using LFSR Method

Fig.6 shows the 256 combinations of CRC is generated using LFSR method in VHDL. In this, initially reset („rst‟) is „1‟, all are cleared. After that, reset is „0‟, and calculating each CRC corresponding bits from “00000000” to “11111111”. These are getting from the signal „data‟

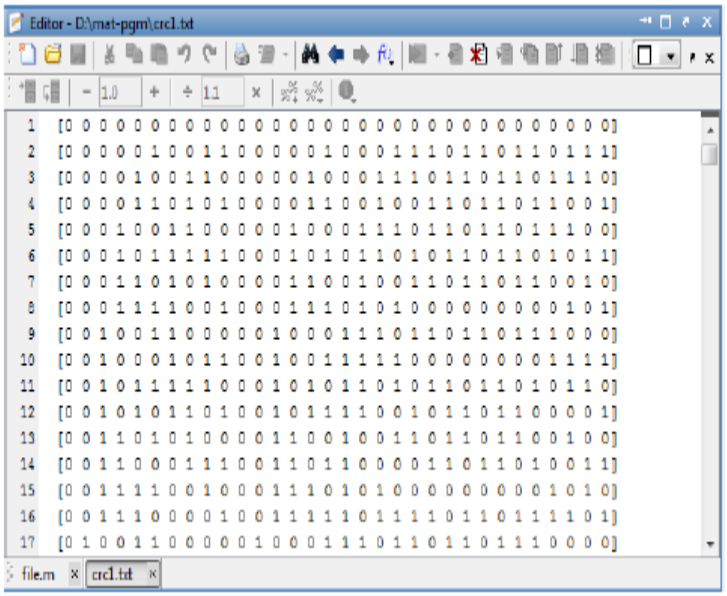

Fig.7 256 Combinations of CRC values using MATLAB

Fig.7 shows 256 combinations of 32-bit CRC values can be generated by MATLAB. Using MATLAB, generated these CRCs by using the „Deconv‟ function. This function is used for polynomial division. By dividing the input stream with the standard generator polynomial, which is 33-bits, getting the remainder which is 32 bit long. These generated CRC values are stored in a text file by generating a text file using „file generate‟ and „file write‟ functions.

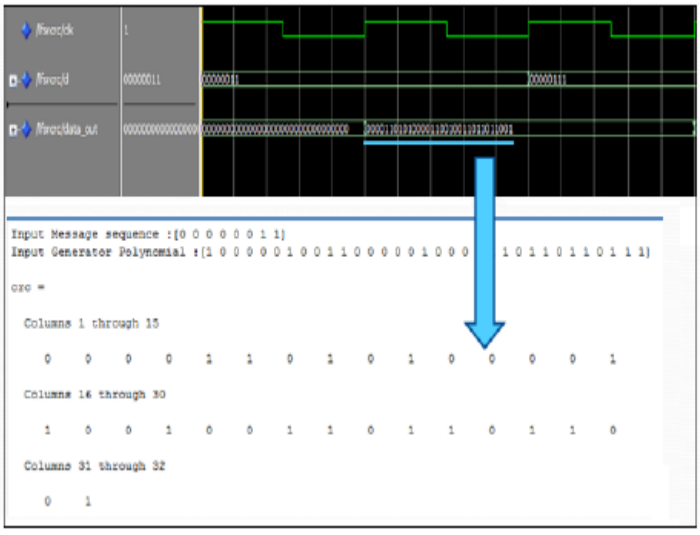

Fig.8 Comparison output for CRC values

The comparison result of CRC generation using MATLAB and LFSR method got the same. Ie, generated CRC values using LFSR method is cross verified by MATLAB. This comparison gives the result that the CRC values produced by LFSR method are correct.

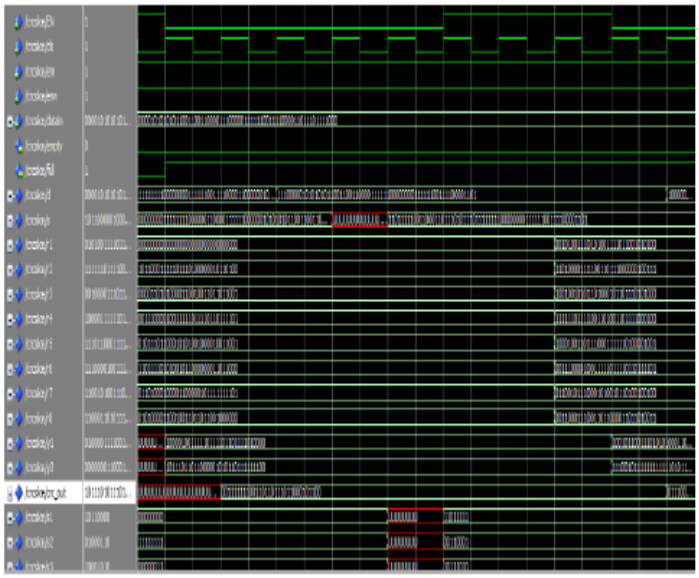

Fig.9 Output of CRC generation using Slicing by 8 Algorithm

Fig.9 shows the simulation result of CRC generation using Slicing by 8 algorithm. This result shows that the initial 64 bit from the input stream is taken and is modified with current CRC value by XORing. This output is given at the signal „s". this output is slices into 8 slices, which is given at signal s1-s8. These output is given to each LUTs. This LUT produced corresponding CRC values of each byte(8 bytes), which is given at the signal „r1-r8" and all are XORed to get next CRC value which is used for next iteration. This final CRC value is indicated by the signal „crc_out".

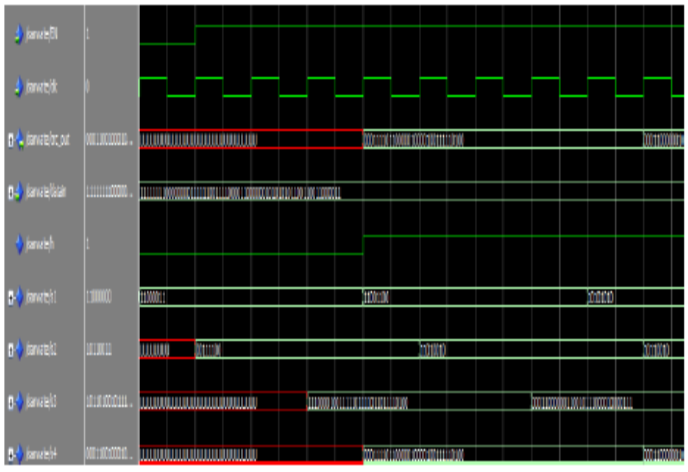

Fig.10 Output of CRC Generation Using

Sarwate Algorithm Fig.10 shows the simulation result of CRC generation using Sarwate algorithm. In this, the initial 8 bit data from the input message stream is taken („s1") which is XORed with initial CRC value (0XFFFFFFFF), ie, in this result if h is „low" then the message bit is XORed with initial value and if h is „high", then the input is XORed with MSB bit of current CRC value. This modified bit is indicated by the signal „s2". This modified bit is given to LUT. In this, only one LUT is used for storing the pre computed CRC values. This LUT produce corresponding CRC value. This output is indicated by the signal „s3". This output is XORed with 24 least significant bits of the current CRC value which is shifted by 8 bit positions to the left. This is the next CRC value. It is indicated by the signal „s4". This CRC output is used for next iteration to get the modified bit and finally get the correct CRC value. This final CRC output is indicated by the signal „crc_out".

4. Synthesis Results

4.1 Comparison of Device Utilization

This comparison results explain how much amount of area are required for different algorithms.

4.1.1 Device Utilization Summary of Slicing by 8 and Sarwate Algorithm with 64 bit Input Stream

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slices	183	4656	3%
Number of Slice Flip Flops	202	9312	2%
Number of 4 input LUTs	246	9312	2%
Number of bonded IOBs	98	232	42%
Number of BRAMs	1	20	5%
Number of GCLKs	1	24	4%

Fig.11 Device Utilization Summary of Sarwate Algorithm with 64 bit data

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slices	238	4656	5%
Number of Slice Flip Flops	416	9312	4%
Number of 4 input LUTs	96	9312	1%
Number of bonded IOBs	98	232	42%
Number of BRAMs	4	20	20%
Number of GCLKs	1	24	4%

Fig.12 Device Utilization Summary of Slicing by 8 Algorithm with 64 bit data

These two summaries shows that number of slices used by Slicing by 8 algorithm is lightly greater compared to Sarwate algorithm, ie, 3% of slices are used in Sarwate algorithm and 5% of slices are used in Slicing by 8 algorithm. But the usage of 4-input LUTs are less in Slicing by 8 algorithm compared to Sarwate algorithm, ie, 2% of 4-input LUTs are used in Sarwate algorithm and 1% in Slicing by 8 algorithm. And IOB

usage is same in Slicing by 8 algorithm and Sarwate algorithm, ie, 42% of IOBs are used.

4.1.2 Device Utilization Summary of Slicing by 8 and Sarwate Algorithm with 512 bit Input Stream

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slices	253	4656	5%
Number of Slice Flip Flops	275	9312	2%
Number of 4 input LUTs	384	9312	4%
Number of bonded IOBs	102	232	43%
Number of BRAMs	1	20	5%
Number of GCLKs	1	24	4%

Fig.13 Device Utilization Summary of Sarwate Algorithm with 512 bit data

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slices	310	4656	6%
Number of Slice Flip Flops	492	9312	5%
Number of 4 input LUTs	234	9312	2%
Number of bonded IOBs	102	232	43%
Number of BRAMs	4	20	20%
Number of GCLKs	1	24	4%

Fig.14 Device Utilization Summary of Slicing by 8 Algorithm with 512 bit data

These two summaries also gave the same comparison result with the input is 512 bits. Usage of 4-input LUTs are less and usage of slices are more in Slicing by 8 algorithm compared to Sarwate algorithm. 5% of slices are used in Sarwate algorithm and 6% in Slicing by 8 algorithm. 4% of 4-input LUTs are used in Sarwate algorithm and 2% in Slicing by 8 algorithm. The above two comparisons, ie, for 64 bits and for 512 bits, the result shows that for 64 bits of data, number of slices used by Slicing by 8 is 2% greater compared to Sarwate algorithm and for 512 bits of data, number of slices used by slicing by 8 is only 1% greater compared to Sarwate algorithm. This result explains that by increasing the number of bits processed, number of slices used in Slicing by 8 algorithm is reduced compared to Sarwate algorithm. Also these two summaries explains that number of 4-input LUTs used by Sarwate algorithm is twice greater than Slicing by 8 algorithm.

4.2 comparison of time consumption

Fig.15 shows the comparison graph of time consumption. This graph clearly says that CRC generator with Slicing by 8 algorithm consume less time compared to Sarwate Algorithm and LFSR method. Slicing by 8 algorithm have minimum delay compared to other algorithms.

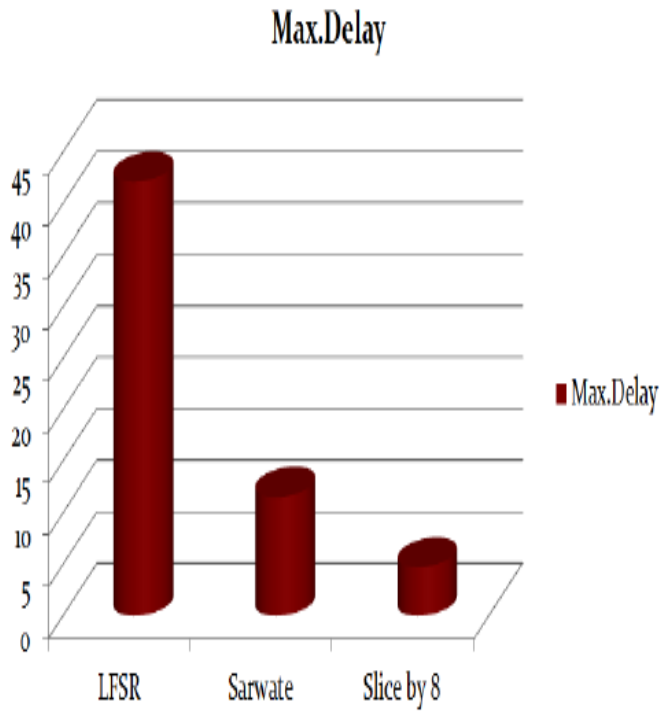

Fig.15 comparison of time delay

The below table clearly explains the comparison of different Algorithms for CRC generation purposes.

Algorithm\Data	Number of Slices	Number of 4-Inout LUTs	Number of IOBs	Maximum delay
LFSR Method	1258	2301	43	42.142
Sarwate	253	384	102	11.56
Slice by 8	310	234	102	3.455

TABLE.1 Comparison of Device Utilization

4.3 Graphical Representation of Comparison Results

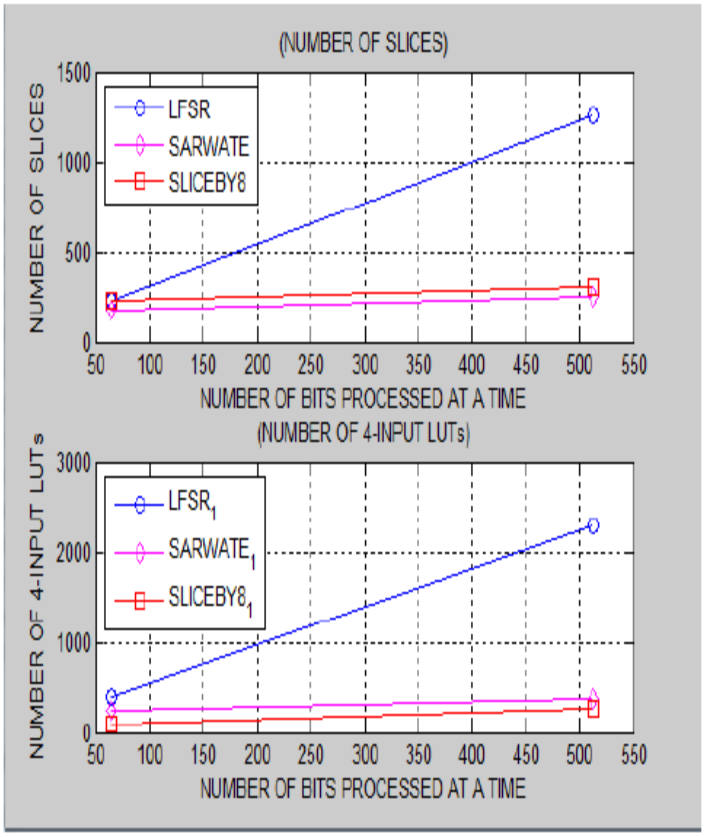

Fig.16 Graphical Representation of Comparison of Device Utilization using MATLAB

Fig.16 shows the graphical representation of comparison results of device utilization in different CRC generation methods. The first graph represents the number of slices used by LFSR, Sarwate, Slicing by 8 methods. This graph shows that slices used by LFSR are very much higher than Sarwate and Slicing by 8. The first graph represents the number of slices utilized by LFSR, Sarwate and Slicing by 8 for, 64 bits processed and also 512 bits processed. The second graph represent the number of 4-input LUTs. This graph also shows that LFSR method is high area consuming method. The second graph represents the number of 4-input LUTs are utilized by LFSR, Sarwate, Slicing by 8 in the case of both 64 bits processing and 512 bits processing.

5. Conclusion

The design of CRC generator using Multiple Look Up based approach is proposed. In this paper, slicing by-8 algorithm is designed, and compares this algorithm with the existing algorithms, that is, with Sarwate algorithm and LFSR method. In this work, first generated the CRC values using LFSR method and generated all 256 combinations of CRC using LFSR method. In this work, also generated these 256 combinations of CRC values using MATLAB.Outputs produced from MATLAB and LFSR method is cross verified. This clearly explains that these two CRC values are same. Designed Look Up Table (LUT) and stored all CRC values in the LUT. LUT contains 256 entries with 32 bit. The input stream is firstly stored in a buffer. In a buffer, five locations with each location has 64 bit wide. These 64 bit data is sliced into eight slices and each is given to each LUT. CRC values are generated from each LUTs corresponding to input stream and all are XORed and get the final CRC value. In this work CRC-32 is used for generating CRC values. So Slicing by-N algorithm can read arbitrarily large amount of data at a time that is used to reduce the time requirement. So this method is applied to CRC generator with Slicing by-N algorithm will be proposed in iSCSI (internet Small computer system interface).

Acknowledgement

We would like to thank the Principal, HOD and all the teaching and non-teaching staffs of TKM Institute of Technology for helping to complete the work as mentioned in the paper.

References

[1] Amila Akagic, Hideharu Amano. “Performance Evaluation of Multiple Lookup Tables Algorithms for generating CRC on an FPGA”, 1st International Symposium on Access Spaces (ISAS), IEEE-ISAS 2011.

[2] C. Toal, K. McLaughlin, S. Sezer, and Xin Yang. “Design and Implementation of a Field Programmable CRC Circuit Architecture”. *Very Large Integration (VLSI) Systems, IEEE Transactions on*, aug. 2009.

[3] F. L. Berry M. E. Kounavis., “A Systematic Approach to Building High Performance Software-Based CRC Generators”. In *ISCC ’05: Proceedings of the 10th IEEE Symposium on Computers and Communications*, pages2, 2005.

[4] Michael E. Kounavis, Frank L. Berry, “Novel Table Lookup-Based Algorithms for High-Performance CRC Generation”, IEEE transactions on Computers, vol. 57, november 2008

[5] S.M. Joshi, P.K. Dubey, and M.A. Kaplan. “A new parallel algorithm for CRC generation”. In Communications, IEEE International Conference, pages 1764 –1768 vol.3, 2000.

[6] Abhijeet Joglekar, Michael E. Kounavis, and Frank L. Berry. “A scalable and high performance software iSCSI implementation”. Proceedings of the 4th conference on USENIX Conference, page 20, 2005.

[7] Yan Sun, Min Sik Kim. “A Pipelined CRC Calculation Using Lookup Tables”, IEEE Communications Society subject matter experts for publication in the IEEE CCNC 2010 proceedings.

[8] J. Bhasker. “AVHDL Primer”, Prentice Hall, 2007.

[9] Kennedy, Davis. “Electronic Communication Systems”, Tata McGraw Hill, 1999.

[10] C. Borrelli. IEEE 802.3 Cyclic Redundancy Check. http://www.xilinx.com/support/documentation/application notes/xapp209.pdf.