ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering
Vol. 2, Issue 6, June 2013

DESIGN AND VERIFICATION OF AN AUTOMATED CRC ENGINE USING VERILOG HDL

Prof. Dhiraj Jain [1], Mr. Hiren G. Patel [2]

Professor, Dept. of ECE, I.T.M. Engineering College, Bhilwara, Rajasthan, India [1]

PG Student [VLSI], Dept. of ECE, I.T.M. Engineering College, Bhilwara, Rajasthan, India [2]

ABSTRACT: The CRC or cyclic redundancy check is a widely used technique for error checking in many protocols used in data transmission. The aim of this project is to design the CRC RTL generator or a tool that calculates the CRC equations for the given CRC polynomials and generates the Verilog RTL code .This block deals with the calculation of equations for standard polynomials like CRC-4, CRC-8, CRC-16, CRC-32 and CRC-48, CRC-64 and also user defined proprietary polynomial. To use PERL as the platform it also aims at having a simpler user interface. To generate the RTLs for any data width and for any standard polynomial or user defined polynomial, this design aims to be complete generic. The RTLs generated by this tool are verified by System Verilog constrained random testing to make it more robust and reliable.

Keywords: HDL, CRC-tools, PERL, RTL.

I.INTRODUCTION

A CRC (Cyclic Redundancy Check) [1] is a popular error detecting code computed through binary polynomial division. To generate a CRC, the sender treats binary data as a binary polynomial and performs the modulo-2 division of the polynomial by a standard generator (e.g., CRC-32[2]). The remainder of this division becomes the CRC of the data, and it is attached to the original data and transmitted to the receiver. Receiving the data and CRC, the receiver also performs the modulo- 2 division with the received data and the same generator polynomial. Errors are detected by comparing the computed CRC with the received one. The CRC algorithm only adds a small number of bits (32 bits in the case of CRC-32) to the message regardless of the length of the original data, and shows good performance in detecting a single error as well as an error burst. Because the CRC algorithm is good at detecting errors and is simple to implement in hardware, CRCs are widely used today for detecting corruption in digital data which may have occurred during production, transmission, or storage. And CRCs have recently found a new application in universal mobile telecommunications system standard for message length detection of variable-length message communications [3].

Traditionally, the LFSR (Linear Feedback Shift Register) circuit is implemented in VLSI (Very-Large-Scale Integration) to perform CRC calculation which can only process one bit per cycle [4]. In this project the method used for the generation of CRC polynomials is based on the LFSR CRC implementation, where the CRC is calculated by passing each data bit every cycle, feeding the most significant bit first. Depending upon the data of the MSB register in the LFSR, shifting and XOR operations occur. This serial LFSR implementation is converted into a one shot or single cycle operation that is realized into a combinational circuit. Based on this method the CRC polynomials are generated.

II.CYCLIC REDUDANCY CHECK

A CRC is an error-detecting code. Its computation resembles a polynomial long division operation in which the quotient is discarded and the remainder becomes the result, with the important distinction that the polynomial coefficients are calculated according to the carry-less arithmetic of a finite field. The length of the remainder is always less than the length of the divisor (called the generator polynomial), which therefore determines how long the result can be. The definition of a particular CRC specifies the divisor to be used, among other things.

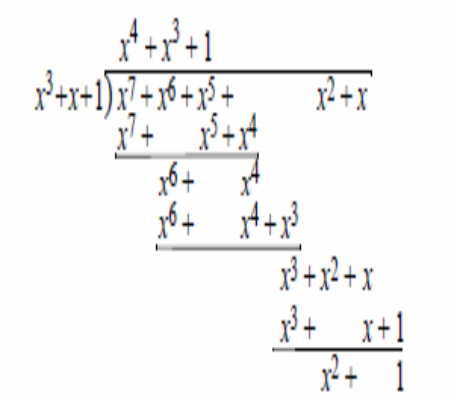
The CRC is based on polynomial arithmetic, in particular, on computing the remainder of dividing one polynomial in GF (2) (Galois field with two elements) by another. It is a little like treating the message as a very large binary number, and computing the remainder on dividing it by a fairly large prime such as 2^32-5.Intuitively, one would expect this to give a reliable checksum. A polynomial in GF (2) is a polynomial in a single variable x whose coefficients are 0 or 1.

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering
Vol. 2, Issue 6, June 2013

Addition and subtraction are done modulo 2 – that is, they are same as the exclusive or operator. For example, the sum of the polynomials:
$x^3 + x + 1$ and
$x^4 + x^3 + x^2 + x$
is $x^4 + x^2 + 1$, as is their difference. These polynomials are not usually written with minus signs, but they could be, because a coefficient of –1 is equivalent to a coefficient of 1.Multiplication of such polynomials is straightforward. The product of one coefficient by another is the same as their combination by the logical and operator, and the partial products are summed using exclusive or. Multiplication is not needed to compute the CRC checksum. Division of polynomials over GF (2) can be done in much the same way as long division of polynomials over the integers. Below is an example [5].

The reader might like to verify that the quotient of $x^4 + x^3 + 1$ multiplied by the divisor of $x^3 + x + 1$, plus the remainder of $x^2 + 1$, equals the dividend.



The CRC method treats the message as a polynomial in GF(2). For example, the message 11001001, where the order of transmission is from left to right (110…) is treated as a representation of the polynomial $x^7 + x^6 + x^3 + 1$.

Common Name	r	Generator	
		Polynomial	Hex
CRC-12	12	$x^{12}+x^{11}+x^3+x^2+x+1$	80F
CRC-16	16	$x^{16}+x^{15}+x^2+1$	8005
CRC-CCITT	16	$x^{16}+x^{12}+x^5+1$	1021
CRC-32	32	$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+$ $x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$	04C11DB7

Table 1: Generator polynomial of some CRC codes [5]

To develop a hardware circuit for computing the CRC checksum, we reduce the polynomial division process to its essentials.

The process employs a shift register, which we denote by CRC. This is of length r (the degree of G) bits, not as you might expect. When the subtractions (exclusive or's) are done, it is not necessary to represent the high-order bit, because the high-order bits of G and the quantity it is being subtracted from are both 1. The division process might be described informally as follows[5]:

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering
Vol. 2, Issue 6, June 2013

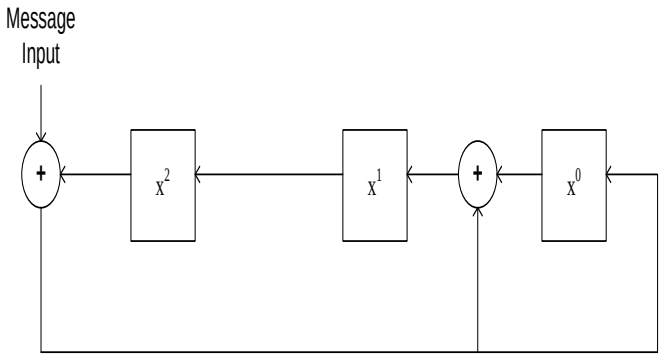


Figure 1: CRC circuit for $G=x^3 + x + 1$

III.IMPLEMENTATION OF CRC RTL GENERATOR

CRC Generator is a command line application that generates verilog code for CRC of any data width starts from 1 bit and no inherent upper limit and any standard polynomial or user defined polynomial. The code is written in Perl and is cross platform compatible. This tool provides CRC RTL generator which can be used at transmitter for CRC checksum generation and the receiver end for verification.

The generated CRC module is synthesizable verilog RTL. The method used for the generation of CRC polynomials is based on the LFSR CRC implementation, where the CRC is calculated by passing each data bit every cycle, feeding the most significant bit first.

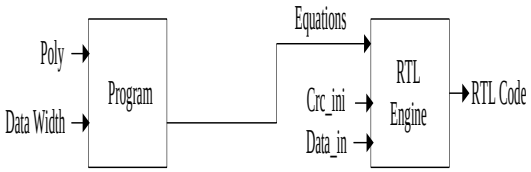


Figure 1: Block Diagram of CRC RTL Generator

Depending upon the data of the MSB register in the LFSR, shifting and XOR operations occur. This serial LFSR implementation is converted into a one shot or single cycle operation that is realized into a combinational circuit. Based on this method the CRC polynomials are generated. Once all the RTL's of different polynomials are generated then the user can use these RTL's to calculate the CRC of entire packet.

IV.CRC PARAMETERS

Data width: Width of the data ranges from 1 bit and no inherent upper limit.

Poly: Standard or any user defined Polynomial.

Equations: The remainder equations, these are the functions of data input and initial state remainders.

Data in: Input data to the verilog code.

Crc_ini: Input initial remainder to the verilog code.

RTL Engine: RTL Engine that generates the CRC bits.

Inputs-Polynomial: Which is one among the above mentioned standard polynomials or a user defined proprietary polynomial.

Data Width:-Starting from 1 bit and no inherent upper limit.

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering
Vol. 2, Issue 6, June 2013

Outputs -Verilog RTL code which in turn has its inputs as partial remainder, data (with the same width mentioned in the computational block) and output as final remainder.

RTL Engine -The RTL Engine takes Data stream, Initial Remainder as input and generates the RTL code as output using the equations from the Program block directly.

Program Block -The program block is the main block of the design. It calculates the polynomial equations that help in building the XOR tree.

V.PLATFORM USED

PERL (Practical Extraction and Report Language): The major internal data structures in the Perl interpreter that represent Perl language elements. Our extractor interrogates the Perl internals just before the execution phase of the Perl script. At that moment the internal data structures are ready to be used for fact extraction. Perl is a compiling interpreter. Instead of interpreting line-by-line the script file, it reads the entire script file, converts it to an internal representation, and then executes the instructions [18].

PERL (Practical Extraction And Report Language) is used as platform for generating the RTL codes because of the constructs available .Perl had useful data structures like Hashes, Arrays, Array of Hashes, Hash of Arrays , which are very much useful in generating the polynomial equations.

VI.SIMULATION AND RESULTS

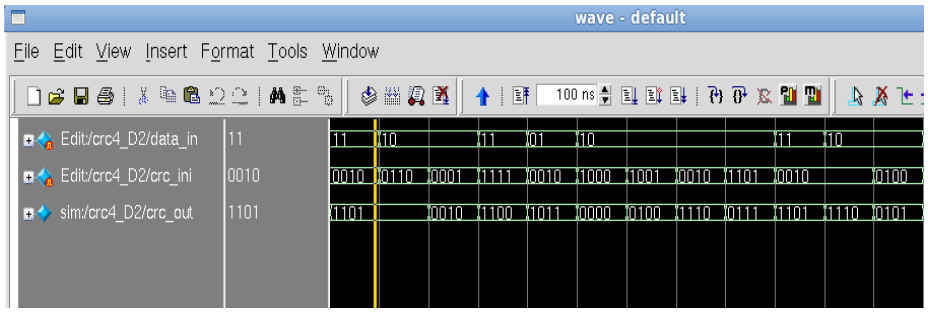


Figure 3: Simulation results for CRC4 of data width is 2

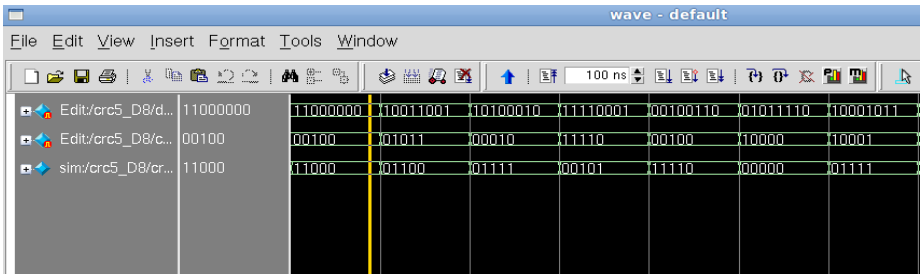


Figure **Ошибка! Текст указанного стиля в документе отсутствует.**: Simulation results for CRC5 of data width is 8

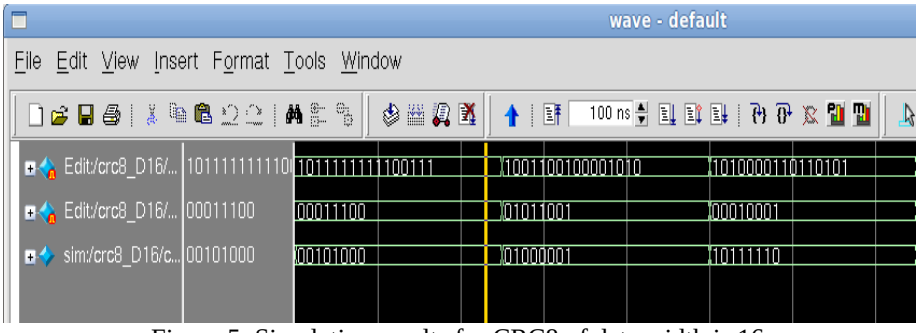


Figure 5: Simulation results for CRC8 of data width is 16

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering
Vol. 2, Issue 6, June 2013

VI.CONCLUSION

In this report the objectives that is to design a tool that generates a verilog RTL, that calculated the checksum for the given data polynomial and CRC polynomial on Perl and generating RTL for any data width and any polynomial.

To calculate the CRC equations for the given CRC polynomials designed a tool that generates the verilog code for any standard polynomials like CRC8, CRC16, CRC24, CRC32 and also any user defined polynomial and data width. The RTLs generated by this tool are verified by system verilog constrained random verification to make it more robust and reliable. Hence the CRC applications are successfully Designed and verified.

REFERENCES

[1] Yan Sun and Min Sik Kim," A Table-Based Algorithm for Pipelined CRC Calculation," IEEE international conference on communications (icc).PP 1-5, publication year 2010.
[2] K. Brayer and J. J. L. Hammond, "Evaluation of error detection polynomial performance on the AUTOVON channel," in Conference Record of National Telecommunications Conference, vol. 1, pp. 8–21 to 8–25. 1975.
[3] 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, publication year 2007.
[4] G. Campobello, M. Russo, and G. Patanè, "Parallel CRC realization", IEEE Trans. Comput., vol. 52, no.10, pp. 1312–1319, Oct. 2003.
[5] http://www.hackersdelight.org/crc.pdf - 2009-07-28 accessed on August/2012.
[6] Qiaoyan Yu and Paul Ampadu," Adaptive Error Control for NoC Switch-to-Switch Links in a Variable Noise Environment," sIEEE international symposium on defects and fault tolerenceof vlsi systems, PP. 352 – 360, . publication year 2008.
[7] Shu Lin, Daniel J. Costello, Jr. Error Control Coding: Fundamentals and Applications. Prentice Hall. ISBN 0-13-283796-X.1983.
[8] http://www.interlakenalliance.com/Interlaken_Protocol_Definition_v1.2.pdf accessed on September/2012.
[9] http://en.wikipedia.org/wiki/Error_detection_and_correctin.
[10] Heidi Joki, Jarkko Paavola and Valery Ipatov "Analysis of Reed-Solomon Coding Combined with Cyclic Redundancy Check in DVB-H link layer,"2nd international symopsium on wireless communication systems , page(s) 313 - 317 , publication year: 2005.
[11] T.V.; Gaitonde, S.S.; Micro ―"A tutorial on CRC computation by Ramabadran," micro IEEE ,Vol.: 8, Issue: 4: Page(s): 62 - 75. 1988.
[12] http://en.wikipedia.org/wiki/Cyclic_redundancy_check accessed on July/2012.
[13] Ross N. Williams, ―A Painless guide to CRC error detection algorithms‖ Version: 3, Date: 19 August 1993.
[14] /LINK/F_crc_ http://www.repairfaq.org/filipg v32.html accessed on August/2012.
[15] http://en.wikipedia.org/wiki/Mathematics_of_CRC#Bitfilters. Accessed on August2012.
[16] Daniel L.Moise KennyWong, "Extracting Facts from Perl Code",reverse engineering WCRE'06, 13th IEEE conference, pages 1-10, publication year 2006.
[17] www.testbench.in/CR_01_constrained_random_verification.html. Accessed on April 2011.
[18] http://en.wikipedia.org/wiki/Error_detection_and_correction#Cryptographic_hash_functions. Accessed on august 2012.
[19] Tsonka S. Baicheva ―Determination of the best CRC codes with up to 10-bit redundancy.