ELKOMIKA: Jurnal Teknik Energi Elektrik, Teknik Telekomunikasi, & Teknik Elektronika

ISSN(p): 2338-8323 | ISSN(e): 2459-9638 | Vol. 8 | No. 1 | Halaman 58 - 68
DOI : http://dx.doi.org/10.26760/elkomika.v8i1.58 Januari 2020

CRC 8-bit Encoder-Decoder Component in FPGA using VHDL

ANDHI RACHMAN SALEH[1], SUNNY ARIEF SUDIRO[2]

[1]Department of Electrical Engineering, Faculty of Industrial Technology
Universitas Gunadarma, Indonesia
[2]STMIK Jakarta STI&K, Jakarta, Indonesia
Email: andhis@hotmail.co.id

Received 4 April 2019 | *Revised* 8 Juli 2019 | *Accepted* 24 September 2019

ABSTRAK

Cyclic Redundancy Check (CRC) adalah salah satu jenis dari deteksi kesalahan yang digunakan pada pengiriman data. CRC umumnya digunakan di jaringan digital dan perangkat penyimpanan untuk mendeteksi perubahan tidak disengaja pada data asli. CRC memiliki keandalan yang tinggi dalam pengiriman data karena CRC menggunakan algoritma berdasarkan cyclic code. Pada artikel ini generator polinomial yang digunakan dalam encoder dan decoder adalah CCITT $X^8 + X^2 + X + 1$ *dan dengan lebar bit data 8 bit. CRC-8-CCITT biasanya digunakan pada Asynchronous Transfer Mode (ATM) headers, Integrated Services Digital Network (ISDN) HEC, dll. Pada penelitian ini dilakukan perancangan dan diterapkan dengan menggunakan VHDL. Software pendukung yang digunakan untuk mengimplementasikan VHDL adalah Xilinx ISE 8.1i. Rancangan encoder dan decoder CRC ini memiliki komponen yang lebih effisien.*

Kata kunci*: Cyclic Redundancy Check (CRC), VHDL Language, Xilinx ISE 8.1i*

ABSTRACT

Cyclic Redundancy Check (CRC) is one type of error detection used in data transmission. CRC commonly used in digital networks and storage devices to detect accidental changes to raw data. CRC has high reliability in data transmission because uses algorithms based on cyclic codes. In this article the polynomial generator used in the encoder and decoder is the CCITT $X^8 + X^2 + X + 1$ *and with a width of 8 bits data bits.CRC-8-CCITT usually used at Asynchronous Transfer Mode (ATM) headers, Integrated Services Digital Network (ISDN) HEC, etc. This article presents design and implementation of a component using VHDL. The supporting software used to implement VHDL is Xilinx ISE 8.1i. This CRC encoder and decoder design have more efficient components.*

Keywords*: Cyclic Redundancy Check (CRC), VHDL Language, Xilinx ISE 8.1i*

1. INTRODUCTION

Cyclic Redundancy Check (CRC) is one type of error detection used in data transmission. CRC commonly used in digital networks and storage devices to detect accidental changes to raw data. For example, in everyday life sending data from one device to another using USB, sending data using Bluetooth, sending data using Ethernet, etc. CRC has high reliability in data transmission because CRC has an algorithm based on cyclic codes. In the CRC algorithm there is a main key in detecting errors in a file, namely a polynomial generator. In its design sometimes requires a complex logic because to demand conformity of the results of the mathematics resulting in requiring a lot of resources. In this journal we will design the CRC using the VHDL language and trimming the logic so that the components used are fewer but the accuracy is still appropriate.

In this journal, the polynomial generator used in the encoder and decoder is the CCITT X^8+X^2+ X+1 and with a width of 8 bits data bits.CRC-8-CCITT usually used at Asynchronous Transfer Mode (ATM) headers, Integrated Services Digital Network (ISDN) HEC, etc.

Cyclic Redundancy Check (CRC) is one type of error detection with the use of redundancy in this method. Redundancy is adding additional bits to the data to be sent. This technique of using redundancy is quite popular in its use. Besides CRC there are also types that use redundancy such as, Simple Parity Check, Two-dimensional Parity check and Checksum **(Forouzan & Fegan, 2007)**.

CRCs are based on the theory of cyclic error-correcting codes. The use of systematic cyclic codes, which encode messages by adding a fixed-length check value, for the purpose of error detection in communication networks, was first proposed by W. Wesley Peterson in 1961 **(Peterson & Brown, 1961)**. In journals W. Wesley Peterson and D. T. Brown provide a new perspective on cyclic codes, which is enough to use basic mathematics and understand the nature of hamming and fire codes.

Within the network, Cyclic Redundancy Check (CRC) is one type of method used to detect errors in data transmission. In sending data packets CRC has high credibility to maintain data packets during shipping caused by noise. Each crc data transmission generates a unique code on the CRC generator then sent to the CRC checker **(Peterson & Brown, 1961)**. Although CRC is widely applied in networks and data storage, CRC can also be applied to other uses such as applications start-up vertification, load-time vertification and program and data correctness validation **(Ritter, 1986)**.

In The OSI Layer, CRC works on layer 2, namely Data Link. Data Link has two main function namely, Data Link Control and Media Access Control. Data Link Control is responsible for the design and procedures for communication between two adjacent nodes: node-to-node communication and media access control, or how to share the link. The Data Link Control has functions that include Framing, Flow and Error Control, and software implemented protocols that provide smooth and reliable transmission of frames between nodes **(Forouzan & Fegan, 2007)**.

CRC has one main part, the generator polynomial. This generator polynomial functions as a divider in the CRC algorithm. The use of CRC must use the same generator polynomial on the encoder or at the decoder. The basic form of a polynomial divisor is like that of ordinary polynomials except that the generator polynomial presents a binary code into a polynomial.

Just as if there is a G(x) generator polynomial with code $X^6 + X + 1$ it will represent the code 1000011. As in Figure 1 **(Ghosh, Mitra, Mukhopadhyay, Dawn, & Ghosh, 2013)**.

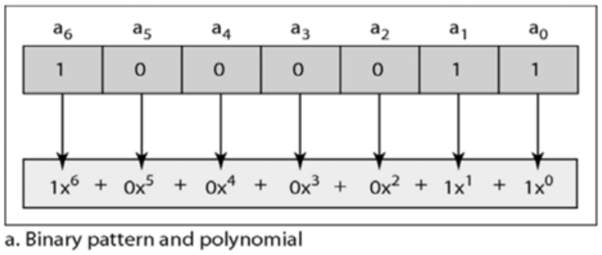

a_6 a_5 a_4 a_3 a_2 a_1 a_0
1 0 0 0 0 1 1
$1x^6 + 0x^5 + 0x^4 + 0x^3 + 0x^2 + 1x^1 + 1x^0$
a. Binary pattern and polynomial

Figure 1. Represents a Polynomial Generator (Ghosh, Mitra, Mukhopadhyay, Dawn, & Ghosh, 2013)

In the calculation there is almost no difference made on the encoder or the decoder. The difference that occurs is where the encoder has an augmented dataword and the decoder has remainder. The initial encoder process generates a unique CRC code, the data is entered first, then the data will be copied into the encoder processing, then the data will receive a number of bits according to the CRC used, with binary 0. Then divide the data bits against the polynomial generator. In this division, one bit is shifted in each division and to determine whether the remainder of the division is then divided by the polynomial divisor or not, it is necessary to pay attention to the MSB. Division is complete until the last data bit in augmented dataword has been used. After completing the distribution, the remaining results will be used as a unique CRC code. As shown in Figure 2.

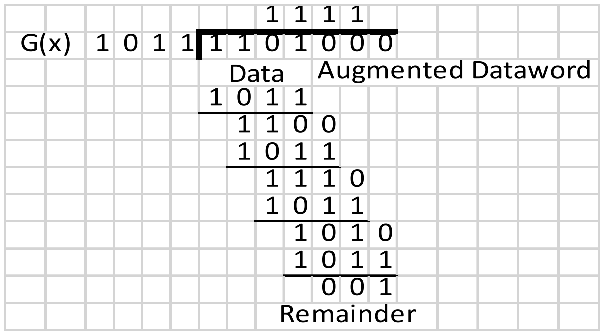


Figure 2. Encoder

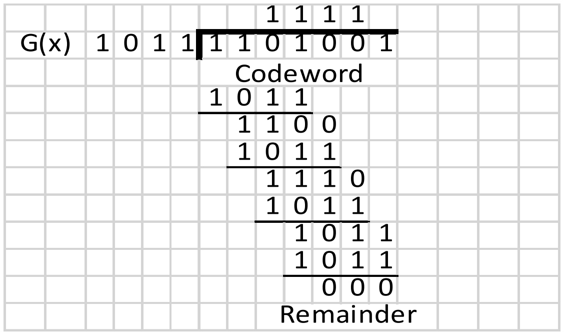


Figure 3. Decoder

In Figure 3. shows the calculations that occur in the decoder. The incoming data decoder is called a codeword. The Codeword will be divided by the same polynomial generator used by the encoder. The division process is the same as the encoder and from the calculation results taken is Remainder. Figure 3. shows that when there are no errors, the remainder values are 0.

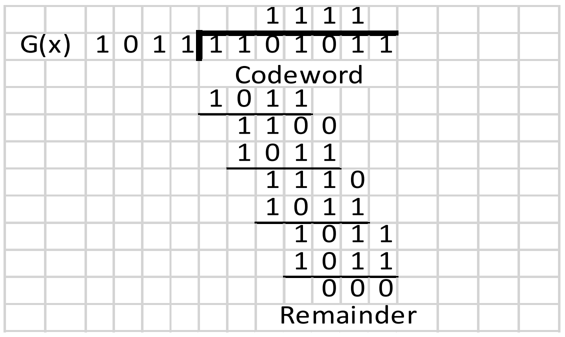


Figure 4. Decoder when Remainder Error

In Figure 4. shows that when the decoder receives a codeword there has been a change when sending data.

2. CRC SCHEME

In this journal the polynomial generator used in the encoder and decoder is the CCITT $X^8 + X^2 + X + 1$. CRC-8-CCITT usually used at Asynchronous Transfer Mode (ATM) headers, Integrated Services Digital Network(ISDN)HEC,etc.

2.1 Encoder

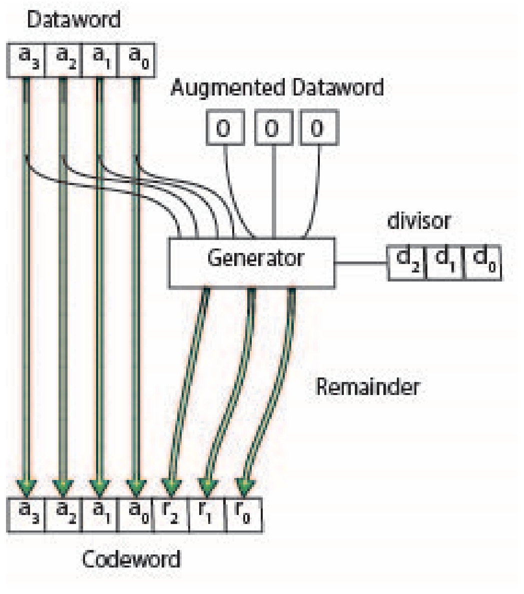


Figure 5. Encoder Diagram

The CRC encoder is the part that generates a unique CRC code that will be carried along with the data to be sent. CRC component is designed in accordance with the algorithm in Figure 2 **(Satran, Sheinwald, & Shimony, 2005)**. In Figure 5, it is explained that the copying of the dataword which as the original data will be sent and then entered the generator block is then combined with augmented dataword and then the division process is done as in Figure 2. The output is the remainder of the process. Then do the merger between remainder data with the original data dataword then it is called codeword.

2.2 Decoder

The CRC decoder is a component part of the receiver. Where the decoder will check the received codeword if there is an error during the sending process or not. If an error occurs in the codeword, the decision logic block will discard the received codeword while if there is no error, the dataword will be taken from the received codeword and the dataword will be forwarded to the next process **(Satran, Sheinwald, & Shimony, 2005)**

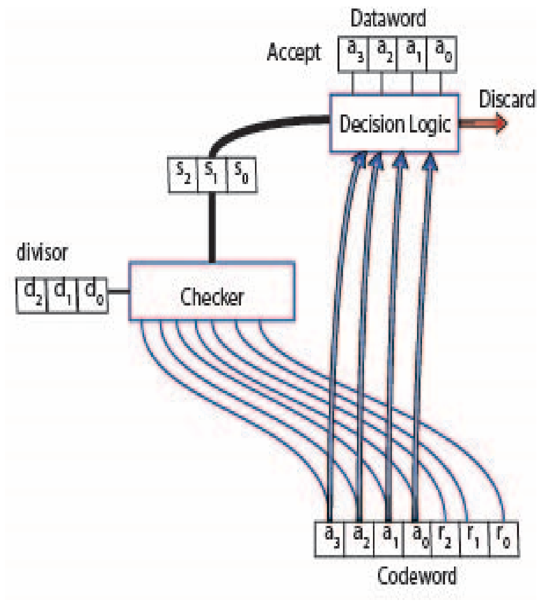


Figure 6. Decoder Diagram

In Figure 6, the decoder process scheme in which the received codeword is checked on the checker component is then calculated as shown in Figure 3. The results taken from the calculation are remainder. then remainder is sent to the decision logic component to decide whether the codeword received has an error or not.

3. CRC SIMULATION AND TEST

3.1 Encoder

Encoder has a role to produce Codewords from every incoming data. Codeword itself has a unique CRC code and data information that will be sent. Based on the design in Figure 5. Then get the design results in Figure 7 and Figure 8.

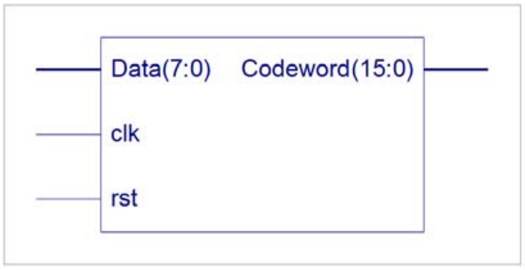
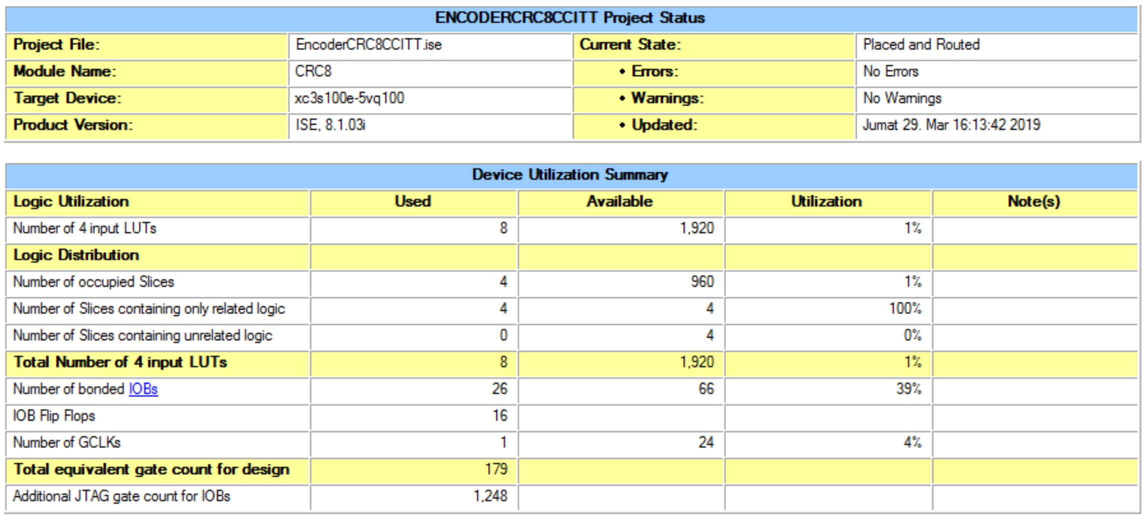


Figure 7. Encoder CRC8

In Figure 7. There are clk, first, Data and Codeword ports. Each port has its own function, such as a clk port to receive the clock signal input produced by a generator. Rst port is used for the reset / intrusion process if an unwanted error occurs. So when Rst port receives a high signal it will reset temporarily if it receives a low signal there is no reset process. The Data Port has an 8-bit data width and is used to enter every data that will be processed. The codeword port has a 16 bit data width and is used to display the results of the process combining the remaining values of the modulo process and original data.



ENCODERCRC8CCITT Project Status			
Project File:	EncoderCRC8CCITT.ise	**Current State:**	Placed and Routed
Module Name:	CRC8	**• Errors:**	No Errors
Target Device:	xc3s100e-5vq100	**• Warnings:**	No Warnings
Product Version:	ISE, 8.1.03i	**• Updated:**	Jumat 29. Mar 16:13:42 2019

Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of 4 input LUTs	8	1,920	1%	
Logic Distribution				
Number of occupied Slices	4	960	1%	
Number of Slices containing only related logic	4	4	100%	
Number of Slices containing unrelated logic	0	4	0%	
Total Number of 4 input LUTs	8	1,920	1%	
Number of bonded IOBs	26	66	39%	
IOB Flip Flops	16			
Number of GCLKs	1	24	4%	
Total equivalent gate count for design	179			
Additional JTAG gate count for IOBs	1,248			

Figure 8. Inside of Encoder CRC8

In Figure 8 displays the design results and components used in the CRC Encoder design using the CRC8-CCITT standard. In Figure 8 it can be seen that the most influential design component is Number of 4 Input LUTs, Number of occupied Slices, Number of Slices containing only related logic and Number of bonded IOBs.

In Figure 8 the number of components 4 LUT Input uses 8 components from FPGA availability for 1920 components. The slice component describes the total components used based on the logic used in the design and from these results we can find out whether there are components that are not too used or not. The bound IOB component explains how many input and output port pins are used in the design

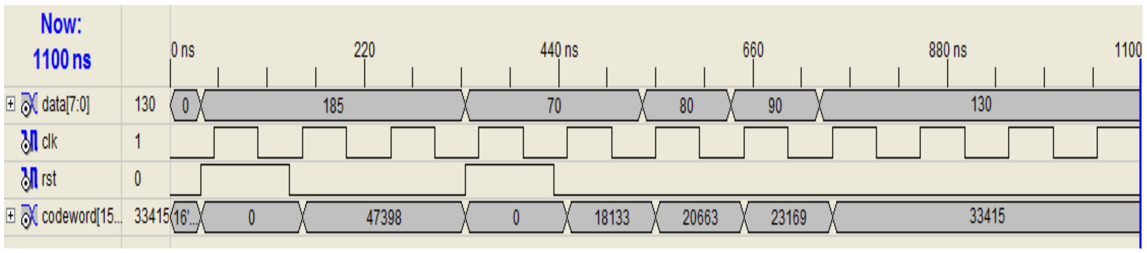


Figure 9. CRC8 Simulation Encoder

In Figure 9. Shows the simulation results of incoming data until the condition occurs if the Rst port receives a high signal it will be reset to the CRC8 encoder. The data tested in Figure 9 is in Table 1.

Tabel 1 displaying the data tested then made into two formats, Decimal(10) and Biner(2). After the data in Table 1 is entered, the Encoder process is carried out. the results of the Encoder process are found on the Codeword port. For results on the Codeword port can be seen in Table 2.

Table 1. Data Entered

Data$_{(10)}$	Data$_{(2)}$
185	10111001
70	01000110
80	01010000
90	01011010
130	10000010

In Table 2 displays the value of data in the Codeword port in two formats, Decimal(10) and Biner(2). The 1st and 3rd data shows when port Rst gets a high signal.

Tabel 2. Codeword

Codeword$_{(10)}$	Codeword$_{(2)}$
0	0000000000000000
47398	1011100100100110
0	0000000000000000
18133	0100011011010101
20663	0101000010110111
23169	0101101010000001
33415	1000001010000111

3.2 Decoder

The decoder has a role to check Codeword that has been sent by the encoder and then processes the data into original data and determines whether the data has errors or not. The decoder design is built based on Figure 6 so that it is obtained as shown in Figure 10 and 11.

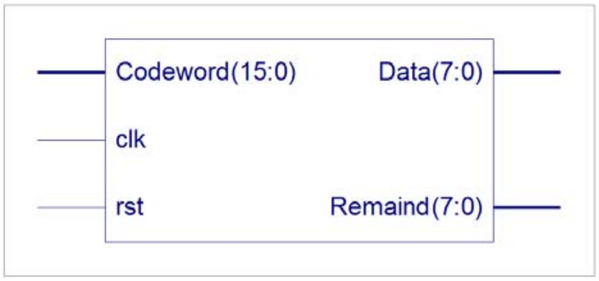
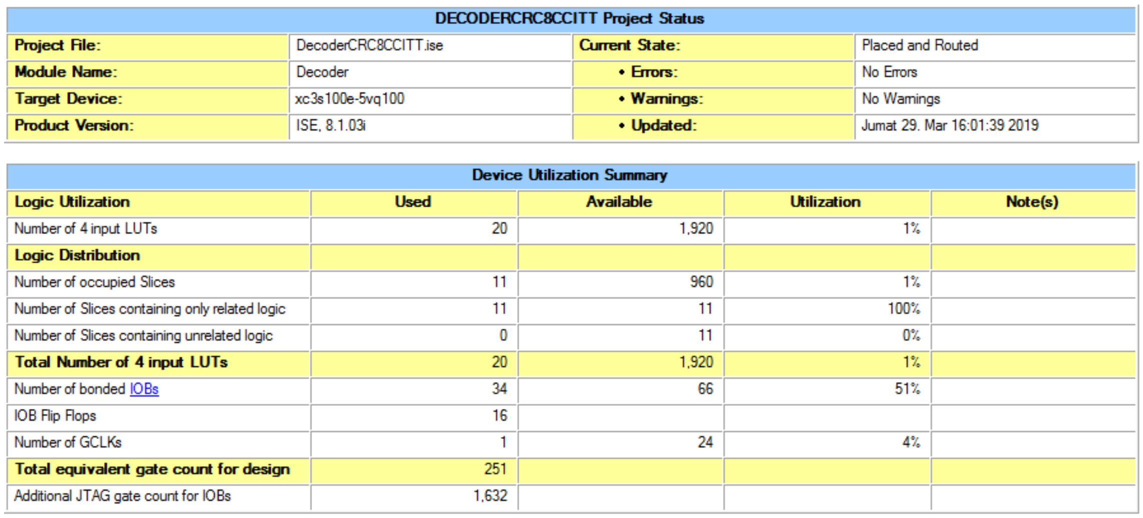


Figure 10. Decoder CRC8

In Figure 10 Displays the results of the decoder design by having 5 ports which each has its own role. The clk port functions to accept enter the clock signal generated by the Clock generator. Rst port functions for the reset / intrusion process if an unwanted error occurs. So when Rst port receives high signal, it will reset while if it receives low signal there is no reset process. The codeword port has a 16 bit data width and is used to enter codeword data. The data port has a data width of 8 bits and this port is the output of the checker results. On the remaind port it has an 8 bit data width, this port will show the checker results whether there is an error on Codeword during transmission or not.



DECODERCRC8CCITT Project Status			
Project File:	DecoderCRC8CCITT.ise	**Current State:**	Placed and Routed
Module Name:	Decoder	**• Errors:**	No Errors
Target Device:	xc3s100e-5vq100	**• Warnings:**	No Warnings
Product Version:	ISE, 8.1.03i	**• Updated:**	Jumat 29. Mar 16:01:39 2019

Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of 4 input LUTs	20	1,920	1%	
Logic Distribution				
Number of occupied Slices	11	960	1%	
Number of Slices containing only related logic	11	11	100%	
Number of Slices containing unrelated logic	0	11	0%	
Total Number of 4 input LUTs	20	1,920	1%	
Number of bonded IOBs	34	66	51%	
IOB Flip Flops	16			
Number of GCLKs	1	24	4%	
Total equivalent gate count for design	251			
Additional JTAG gate count for IOBs	1,632			

Figure 11. Inside of Decoder CRC8

In Figure 11. Displays the device utility used from the utility available in the FPGA.

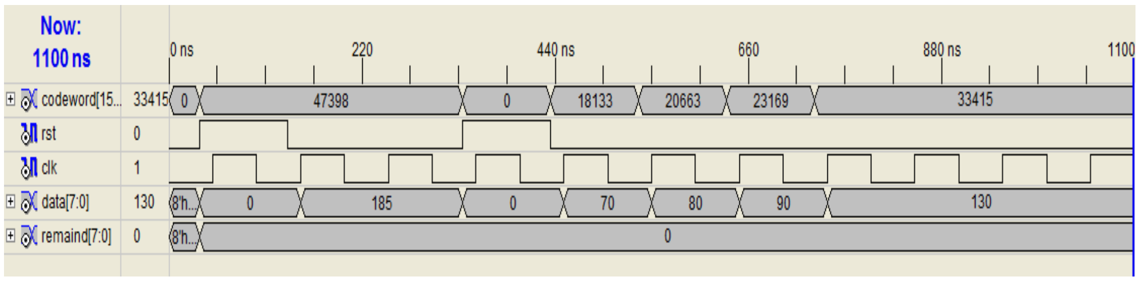


Figure 12. Decoder CRC8 Simulation

In Figure 12 shows the simulation results from the incoming data to make the condition that if the Rst port receives a high signal it will be reset to the CRC8 decoder. The data tested in Figure 12 are in Table 3.

Tabel 3. Codeword decoder

Codeword$_{(10)}$	**Codeword$_{(2)}$**
0	0000000000000000
47398	1011100100100110
0	0000000000000000
18133	0100011011010101
20663	0101000010110111
23169	0101101010000001
33415	1000001010000111

In Table 3 displays the value of data in the Codeword port in two formats, Decimal(10) and Biner(2). The 1st and 3rd data shows when rst port gets a high signal.

Table 4. Data decoder

Data$_{(10)}$	**Data$_{(2)}$**
185	10111001
70	01000110
80	01010000
90	01011010
130	10000010

In Table 4 displays the results of data that Checker has done to proceed to the next process. on the Remainder port in Figure 10 shows no Codeword errors received then the data will continue. if the Checker detects a data error, the Remainder port will display the data value.

In the design of this journal, there are several components used in making CRC Encoder and Decoder. In the Encoder there are a number of components 8 numbers of 4 input LUTs, 4 number of occupied slices and 26 number of bonded IOBs. While in the decoder there are a number of components 10 number of 4 input LUTs, 5 number of occupied slice and 34 number of bonded IOBs. The results of the components obtained can be compared with previous studies listed in the Table.

Table 5. Comparison of Encoder Components

Encoder	**Number of slices**	**Number of LUTs**	**Number of bonded IOBs**
CRC8 **(Saleh, Saleh, & Saad, 2018)**	30	54	40
CRC8 **(P, A, & Kotain, 2012)**	22	8	N/A
CRC8(proposed)	4	8	26

In Table 5 shows a comparison of the number of components used in the CRC8 encoder. when compared with the second and first studies, the second study uses fewer components. When viewed from the component the number of slices has a difference of 8 components, while the number of Lut has a difference of 46 components. When compared with the second study with this study, the difference in the number of slice components was 18 components lower than the second study.

Table 6. Comparison of Decoder Components

Decoder	Number of slices	Number of LUTs	Number of bonded IOBs
CRC8 **(Saleh, Saleh, & Saad, 2018)**	36	94	35
CRC8 **(P, A, & Kotain, 2012)**	28	111	N/A
CRC8(proposed)	11	20	34

In Table 6 shows a comparison of the number of components used in the CRC8 decoder. when compared with the first and second studies, the second study uses fewer components in the Number of slice components while the first study is in the Number of LUTs component. When viewed from the component the number of slices has a difference of 8 components, while the number of Lut has a difference of 17 components. When compared with the second research with this research, the difference in the number of slice components was 17 components lower than the second research and the difference in the Number of LUTs components was 74 components lower than the first research.

4. CONCLUSION

This research designed and implemented using VHDL. The supporting software used to implement VHDL is Xilinx ISE 8.1 i. The design in this journal has been successful because it is in accordance with what is desired and has been tested and the results are appropriate. Like in an encoder where it is assigned to produce a unique code which is then combined with the original data into a codeword. In the decoder where it is assigned to parse the received codeword and confirm whether the codeword received has an error or not. If an error occurs, the data will be discarded while if no error occurs then the data will continue. The design results is more efficient because when compared between each study, the components used are fewer and the CRC design is as expected. From the use of hardware in this series, it is expected that in the future it can be integrated with other systems.

REFERENCE

Forouzan, B. A., & Fegan, S. C. (2007). *Data Communications and Networking.* New York: McGraw-Hill Higher Education.

Ghosh, D., Mitra, A., Mukhopadhyay, A., Dawn, A., & Ghosh, D. (2013). A Generalized Code For Computing Cyclic Redundancy. *International Journal of Students Research in Technology & Management, 1*(2), 192 - 202.

P, P. S., A, R., & Kotain, A. S. (2012). FPGA Implementation of Single Bit Error Correction using CRC. *International Journal of Computer Applications, 52*(10), 2 - 6.

Peterson, W. W., & Brown, D. T. (1961). Cyclic Codes for Error Detection. *Proceedings of the IRE, 49*(1), 228-235.

Ritter, Terry. (1986, February 11). The Great CRC Mystery. *Dr. Dobb's Journal of Software Tools*, hal. 26-34.

Saleh, A. A., Saleh, K. M., & S. A.-A. (2018). Design and Simulation of CRC Encoder and Decoder Using VHDL. *International Scientific Conference of Engineering Sciences, 1*(3), 221-225.

Satran, J., Sheinwald, D., & Shimony, I. (2005). Out of Order Incremental CRC Computation. *IEEE Transactions On Computers, 54*, 1178-1181.