Journal of Network Communications and Emerging Technologies (JNCET)
Volume 11, Issue 8, August (2021)

Zynq FPGA Implementation of LDPC Based on Bit Flipping Algorithm

Arwa H. Ashou
Computer Engineering Department, Collage of Engineering, University of Mosul- Iraq
arwahafidh@gmail.com

Dhafir A. Alneema
Computer Engineering Department, Collage of Engineering, University of Mosul- Iraq
dhafir.abdulfattah@uomosul.edu.iq

Published online: 21 August 2021

ABSTRACT

A channel or medium, such as wired or wireless, is used to transmit data from a source to a receiver. The channel medium, as well as external noise, affect the reliability of received data, causing signal interference and introducing errors in transmitted data. LDPC codes are one of the error correction codes with the quickest and most efficient level of information security and correction during contact. As a result of its robust success in correcting errors and ability to satisfy all of the specifications of the 5G communications, this code has been almost universally adopted in networking standards such as satellites and communications networks for error correction. The hardware implementation, however, is a challenge for researchers due to its high complexity and long run time. LDPC codes are one of the most codes for FPGA implementation. This work introduces a hardware LDPC of Bit Flipping algorithm decoder using HLS "High-Level Synthesis" techniques. Given that flexibility is one of the key advantages of FPGAs, HLS is commonly used as a good tool to synthesize hardware. In this work, an optimization techniques including array partitioning and loop unrolling to speed up the execution time and to minimize latency are utilized. The implementation of the BF architecture on Zynq-7000, ZC702 Evaluation Board Part xc7z020clg484-1 is presented in this work and for simulation results, Xilinx Vivado HLS 18.3 is used.

Index Terms – LDPC, Bit Flipping, FPGA, HLS, ZYNQ.

1. INTRODUCTION

In communication systems, the mechanism that sends information from Tx_transmitter to Rx_receiver through an intermediate communication_channel must be robust and reliable. This is a difficult task in the real world because there are many mistakes. As a result, detecting and correcting errors when transmitting information has become a very important task [1]. LDPC codes are Forward Error Correction (FEC) codes which automatically detect and correct errors. Robert Gallager [2] invented them in 1962. However, the LDPC codes were overlooked for a long time before Mackay and Neal re_discovered them [3], because the transistor era had just begun, and hardware technology did not yet cover the difficulty of LDPC encoding and decoding [4]. For FEC, due to their high computational complexity and dominance of highly organized algebraic block and convolutional codes [5]. The LDPC codes have been shown to work in cellular systems [4], Furthermore, In DVB-S2, the LDPC code was introduced as the ECC. LDPC codes are also used in wireless network protocols, such as IEEE 802.11n WLAN and IEEE 802.16e Wi-MAX [6]. And In wired networks, such as 10GBase-T Ethernet [7].

The key characteristics of LDPC Codes are that They are capable of providing efficient encoding and decoding with reduced decoding time, latency, and error-floors at high SNR and low BER For low SNR [8], [9]. LDPC codes have better error detection and correction capabilities. LDPC codes can be used to add parity bits to a message before sending it to the receiver, allowing the receiver to determine what message the sender want to send [10].

LDPC codes have been widely applied in recent research, due to its excellent performance, and ability to work very near the capabilities of Shannon. However, some challenges in implementing the program include building a finite block length LDPC code, reducing the encoding and decoding complexity, storing the parity check matrix, and minimizing the long latency and number of iterations [7]. These codes have a high level of parallelism in implementation. Many parameters influence the complexity and throughput of an LDPC decoder, including block length, code rate, processing node complexity, interconnection complexity, number of iterations, and parallelism level. parallelism in decoding can provide high throughput. FPGA's flexibility makes it better for designing LDPC decoders than general-purpose processors [9]. Since HLS can synthesize a high-level hardware application, it's very useful to reduce design time and explore a range of design options [11] [12]. This work, designs Bit Flipping algorithm with using HLS techniques. The BF is the efficient LDPC algorithm, with low hardware complexity and good error performance

[8]. The Bit Flipping algorithm is referred to as a "hard decision" algorithm. The BF decoders exchange only one bit of information iteratively between their processing units known as Bit Nodes and Check Nodes. As a result, these short messages necessitate very simple computation blocks, which accounts for the simplicity of such decoders. When a bit of the received sequence is toggled, the Bit Flipping algorithm currently in use exchanges the number of parity failures. The iterative decoding algorithm is a control operation performed by the parity nodes on the received bits of the bit nodes [13], [14].

A ZC702 board was used as a hardware platform for implementing the design proposed for this work and is filled with a Zynq-7000 XC7Z020 AP SoC. On a given chip, it contains a PS and a programmable (PL) that are integrated in the SoC style [15].

2. RELATED WORK

The main goal of this section is to focus on the studies that are more related to the topic of this paper. Hence, this section is divided into three main subsections, each of which includes a promising area of improvement.

2.1. Utilization-Related Approaches

The implementation of the LDPC hardware has attracted many researchers around the world. One of these studies is Yasoubi [16] that focused on the what is called DVB-S2. In the decoder design, the researcher found that Lookup Table (LUT) approximation was the fastest one compared to other approaches (e.g., Piece-Wise Linear PWL). However, LUT requires more resources during the implementation phase, which is one of the LUT limitations. To overcome this issue, the researcher proposed to use the Range Addressable Lookup Table (RALUT) due to the structural nature of DVB-S2. Yasoubi also proposed to use a memory mapping that allowed to implementation of 360 functional units. In this context, each unit was optimized aiming at reducing the hardware resources on the FPGA. The authors also suggested a design for the RALUT for the "Hyperbolic Tangent Function". Usually, the RALUT is distributed on input uniformly, instead, the author used LUT that is in a scale that is "Non-Uniform", which increased the "Near-Zero" values. Moreover, the approach used Zynq XC7Z030 in assessing the complexity of the proposed approach. The experimental results showed that the speed is increased as a result of using the LUT. However, the LUT requires more memory space, therefore, the RALUT can be utilized in decreasing resource usage. Furthermore, the results when using Xilinx and XC7Z030 reflected a better performance in terms of speed.

Another study that was performed by Unal [17] proposed a method that used the FPGA framework in accelerating the LDPC codes simulations. The authors performed several experiments that were based on PGaB and GaB algorithms. The results depicted that these algorithms significantly decreased the time consumption to an hours-scale from a years-scale. In fact, this allowed performing the analysis of Error Correction at unattainable resolution with CPU-based simulations. The author also performed an analysis on Error Pattern aiming at distinguishing all the possible 4-bit patterns of error in a code word. Moreover, identified all the possible patterns of error that could not be corrected by GaB in less than 5 hours, which was estimated in a CPU-based to be achieved in approximately "7800" days. The author presented a comparison of resources throughput, utilization, and clock rate when the algorithm tested in Tanner Code using "Virtex6 FPGA". The design and performance of the LDPC decoder was studied by Raju and Prasad [9]. The used Bit Flipping algorithm in the decoding of the regular LDPC with an 8x16 matrix, 4 for row weight, 2 for column weight, Code word Length of 16 and code rate of ½. The decoder and encoder were designed using Verilog and HDL. The simulations were performed using "XC-Vivado14.2" and "QuestaSim10.4c". The design was synthesized by "LeonardoSpectrum 2014b.4" and "XC-Vivado14.2". Also, the implementation was performed using Nexys 4-DDRXC7A100TCSG324-2L FPGA. The use of the Bit Flipping algorithm had also attracted Reddy and Rao [18]. They used Verilog HDL and Xilinx ISE Design Suite 12.4 to design an LDPC decoder and encoder block. The Xilinx SPARTAN 3E XC3S500EFG320 FPGA was used to carry out the implementations. The simulation results for encoder, decoder, AWGN channel, and detector were obtained using an LDPC architecture for 8-bit message blocks using a standard parity check matrix with 8 rows and 16 columns. Orabi et al. [19] performed a comparison among different decoding algorithms such as SPA, BF, and Two-Stage Hybrid decoding. The comparison was based on the "Delay Time" and "Memory Usage".

2.2. Complexity-Related Approaches

The issues of complexity and throughput of the LDPC decoders have been extensively studied in the literature since there is a trade-off between them. Singh and Sharma [20] suggested an approach for decreasing the complexity level and increase the throughput of the LDPC decoder. They used a "partially parallel" architecture that led to reducing the level of complexity when it comes to check node complexity and routing. Their design was highly concurrent, which enabled to obtain a "Maximum Symbol Throughput" of 92.95 Mbps with decoding iterations of 18 at maximum. Moreover, the work implemented a 9216 bit and "Rate of ½", (3,6) LDPC decoder on "Xilinx XC3D3400A" from "Spartan-3A DSP". Their proposed design was simulated using the "ISim" simulator and was synthesized using "Xilinx ISE design suite 13.1". One of the factors that reduced resource utilization was the BP algorithm. The performance was close to the performance of soft decoding algorithm SPA and Hard

Bit-Flipping decoding algorithms BFA. In the latter, the bit node operation is considered a soft operation that is able to enhance performance. While the check node operation is considered a hard operation that decreases the complexity. Here, the quantization check-to-bit and bit-to-check messages can be decreased to only one bit, which in turn, decreases the complexity. The pipelining involved in the (3,6) LDPC decoder results in gaining more throughput. Finally, the work of Singh and Sharma used only 36-bit node columns in the LDPC matrix and 18 rows. In the same the above context, the authors [21] Chandrasetty and Aziz proposed an algorithm that aimed to reduce the implementation complexity of the LDPC decoding. The proposed approach depended on the "simple hard-decision" technique and used concepts inspired by "Soft Channel" information aiming at enhancing the performance of the decoder. The approach was validated with the "WLAN -IEEE 802.11n" standard and examined on Xilinx Virtex 5 FPGA. The experimental results showed that the average throughput was approximately 16.2 Gbps and the performance of BER was 10-5 at Eb/No of 6.25 dB. Compared to the BEF algorithm, the authors' algorithm outperformed the BEF in terms of the average iterations. Moreover, the algorithm outperformed the SPA algorithm in terms of the reduced resources. This work provided a comparison of the performance among SPA, SMPA, BFA.

2.3. Other-Recent-Aspects Approaches

In [22], multiple PGDBF decoder implementations for LDPC codes have been proposed. The first design is a conventional LFSR implementation of the random generator, and the second design is IVRG, which is a new method that uses binary sequences generated by the LDPC decoder. When compared to the non-probabilistic version, both PGDBF implementations greatly increase error correction efficiency while retaining the same high throughput. However, in the case of the LFSR-PGDBF, the performance benefit requires a large hardware overhead, while the IVRGPGDBF has a 10% overhead and thus appears to be a viable solution for very high throughput applications. The research [23] was presented by F. S. H. Mahmood and et al. A DCSK communication system has been improved using LDPC codes with an FPGA design using the BF decoding method and its implementation has been carried out on the Xilinx Spartan-6 kit with the assistance of the Xilinx SG design tools.

3. IMPLEMENTATION OF LDPC ENCODING AND DECODING BASED ON BIT FLIPPING ALGORITHM

In the past years a variety of designs have employed the HLS_technique. Its popularity is growing as complex algorithms can be converted into efficient hardware implementation more quickly. Therefore, we try to use HLS_technique in this work, the LDPC BF algorithm decoder. Some optimization directives, like the Loop Unrolling and Array Partitioning directives, are also applied to enhancing the performances of our designed algorithm that helps the hardware within the HLS environment to speedup execution_time and reduce latency.

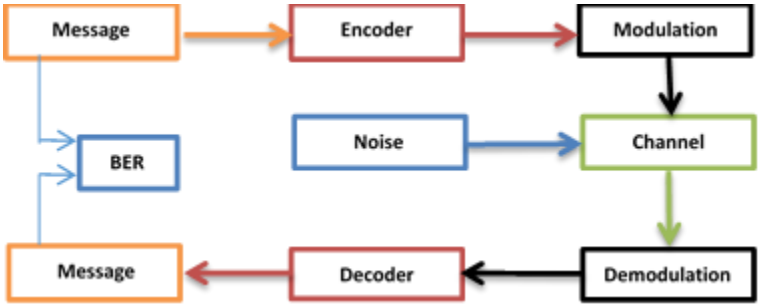


Figure 1 The General Diagram of Digital Communication System

There is increasing use of the LDPC in digital communication. Generally, a digital communication system consists of the encoder, modulator, communication channel, demodulator, and decoder as shown in figure 1.

The implementation of each block is briefly described as follow:

Step 1: The message is generated randomly by using the Matlab, then reading the message from the text file and storing it into character array to encode all message characters.

Step 2: Encoder

At this step, the code word is created by the original message and the generator matrix (G), however, this work uses a Binary regular LDPC code, the size of generator matrix is 8*16 regular and coding rate is 0.5. The equation (1) of the matrix G as follow:

$$G = [\, I_{n-k} : P_{n-k} \,]_{k-n} \qquad (1)$$

Whereas I: identity matrix and P: parity matrix.

The code word calculated by equation (2):

$$\text{Code word} = \text{Message} * \text{G\_Matrix} \quad (2)$$

When the code rate = 0.5, The parts of code word are, the first is the original message and the second is the parity and this is called systematic code as the equation (3):

$$\text{Code word} = [\ \text{Message} : \text{Parity}\ ]\ 1*n \quad (3)$$

Figure (2) shows a flowchart of the Encoding process when the generation matrix size 8 * 16 and the message length is 8 bits.

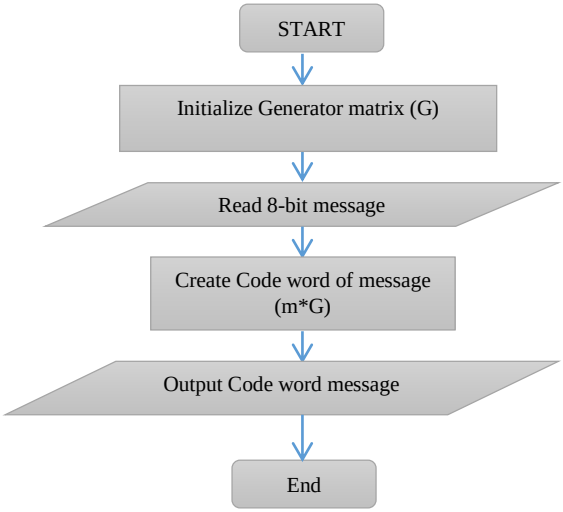


Figure 2 LDPC Encoder Flowchart

The program of encoder was written in C++ language using Vivado HLS. The directive uses unrolling for parallelism [24]. The device where used to implement the design was Zynq-7000 ZC702 Evaluation Board Part xc7z020clg484-1, where it has 53200 LUT, 106400 FF, 220 DSP48E and 280 BRAM-18K. The targeted clock set to 10 ns.

Step 3: Modulation

This process is done to represent the bits using BPSK after the code word has been created from the previous step, by applying the following equation (4), the BPSK representation of bits is obtained:

$$\text{Representation of (BPSK)} = 1 - (2*\text{bit\_value}) \quad (4)$$

Hence, the bit with the value 1 is represented by the value -1. Whereas the representation value is +1 when the bit value is equal to zero. At this step the bits have been represented in BPSK.

Step 4: AWGN generation

The number of bits for the noise must be equal to the number of bits for the code word, Noise was configured with (Eb/N0) values for seven different values, which are (0 to 7 db) in each program. Additive White Gaussian noise is generated (AWGN) by using a number of ready-made functions and some mathematical equations to obtain the final value of the noise . At this step, every noise bit is defined. It was also considered that the channel is not diminished. After the noise was generated it was combined with the BPSK signal to represent the signal with the addition of noise, so that the transmission of the signal through the transmission channel is simulated and then transferred to the next stage, which is the decoding stage .

Step 5: Demodulation

This operation was done to convert the bits to 0 and 1 by comparing the value of the bits produced after the decomposition process, if the value was negative This means that the bit value is 1, and if the value is positive, then the bit is equal to zero. At this step, each bit was defined as integer.

Step 6: Decoder

At this step the noise and parity bits will be removed from the received code word in order to obtain the original message, Here the bit flipping algorithm will be used, we have the parity check matrix H which consists of two parts: The first part is the parity matrix transpose, and the second part is the identity matrix as follows:

$$H = [\ P^{T}{}_{n-k} :\ I_{n-k}\ ]_{K*N} \quad (5)$$

It is necessary to find the HT matrix to compute the syndrome, Where the equation for syndrome is

$$\text{Syndrome} = \text{Received code word} * H^{T} \quad (6)$$

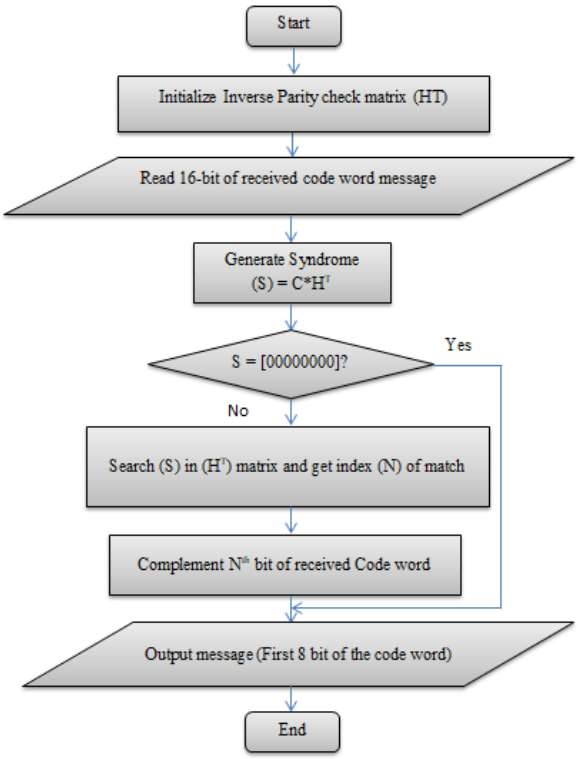


Figure 3 Bit Flipping Decoder Flowchart

If syndrome is zero, it means there is no error in the received code word, But If syndrome is not equal to zero, there is an error, and its location is determined and corrected through the following, the located of the wrong bit can be found by searching for the syndrome value in the H^{T} matrix, the two vectors will match at one of the rows of the HT matrix. The row number in which the 2 vectors match is the location of the wrong bit in the received code word. And it is corrected at the final of decoder process, where it is flipped from zero to one or vice versa, Then the parity is separated from the coded word to get the original message after correcting the received code word. Figure (3) shows a flowchart of the Bit Flipping LDPC Decoding process if the parity check matrix size of 8 * 16 and the code word length is 16 bits.

In our design, we used the Zynq hardware platform which consists of processing_system(PS) and programmable_logic(PL), which is equivalent to a conventional integrated_memory of FPGA, several peripherals, and high_speed communication interfaces.

Step 7: Calculate the BER value

At this step, the sum of the number of different bits between the bits extracted from the previous process with the bits formed in the first operation was calculated by dividing the sum of the errors in the decoded bits by the total number of bits.

4. RESULTS AND DISCUSSIONS

This paper illustrates the processes of encoding and decoding were carried out using Vivado Design Suite and optimizing the design needed to fulfill timing request. The Vivado Design Suite 2018.3 release from Xilinx supports high-level synthesizing Zynq

ZC702 and SoC. When the Generator matrix of 8*16 with row weight is 7 and column weight is 6. If the message length is 8 bits and the code word length is 16 bits. The synthesis results of encoder when using array partitioning and loop unrolling directives are shown in figure (4).

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	4.890	1.25

Latency (clock cycles)

Summary

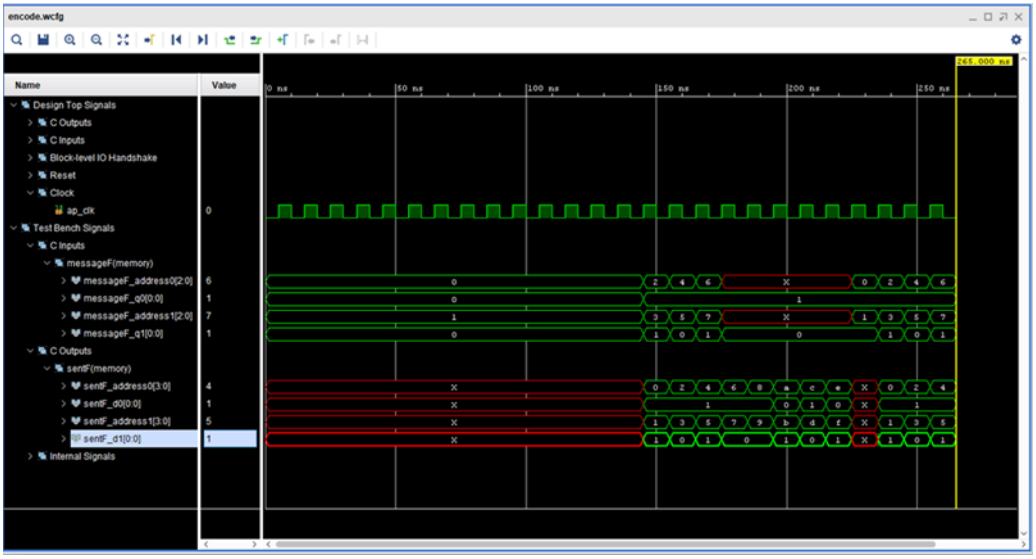
Latency		Interval		
min	max	min	max	Type
8	8	8	8	none

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	52
FIFO	-	-	-	-
Instance	-	-	-	-
Memory	-	-	-	-
Multiplexer	-	-	-	255
Register	-	-	23	-
Total	0	0	23	307
Available	280	220	106400	53200
Utilization (%)	0	0	~0	~0

Figure 4 LDPC Encoder Synthesis Results



Figures 5 Input and Output Data to the Encoder with Using Loop Unrolling Directive

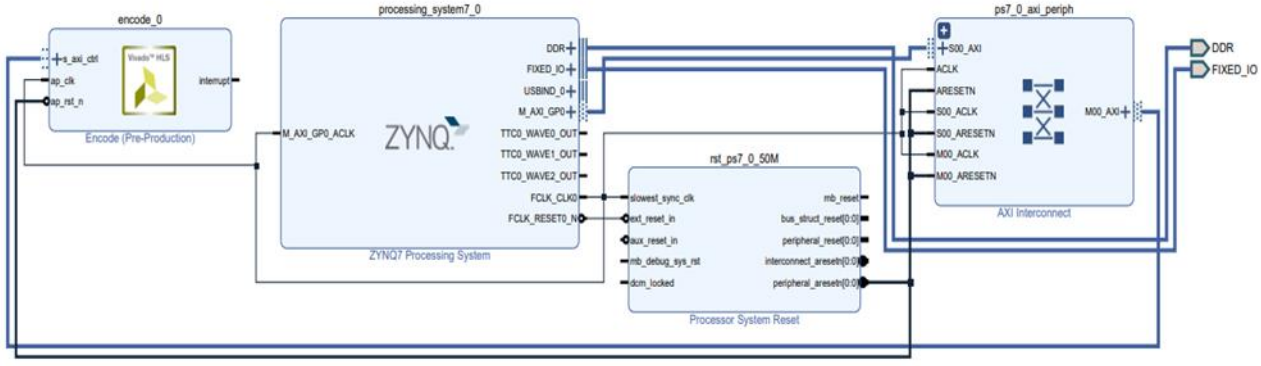


Figure 6 Implementation of Encoder

If the message is (11101110), The code word will be (1110111010011001) with using array partitioning and loop unrolling directives, the waveform of encoder shown in figure (5). The input data to the encoder is (11101110) from address $(0)_{hex}$ to address

$(7)_{hex}$ and the output data from the encoder is (1110111010011001) from address $(0)_{hex}$ to address $(F)_{hex}$ as shown in this figure. Figure (6) illustrates the proposed design for the encoder using Zynq_7000 platform. The Zynq_AXI_lite interface_connection is used to connect the algorithm_core with other FPGA hardware as the PS.

Figure (7) represents the results of executing the LDPC encoder after projecting the design onto the Zynq board, which was done using the SDK. The figure shows the settings for the Serial Port (COM3, Baud rate and Number of bits) and shows the success of the process of reading and writing the data used in this encoder and stored in the SD Card, as well as the process of encoding the message (11101110) sent and the corresponding Code Word (1110101011011001).

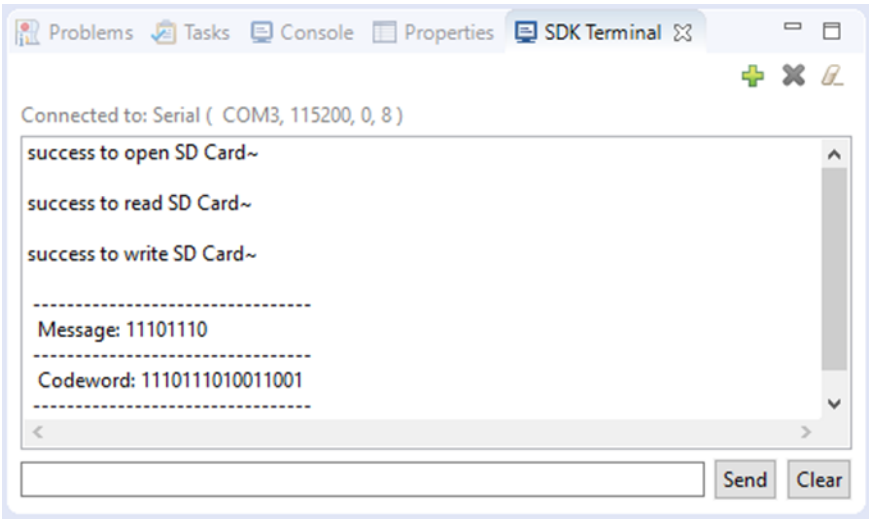


Figure 7 Implementation of Encoder on Zynq

Summary

Resource	Utilization	Available	Utilization %
LUT	515	53200	0.97
LUTRAM	60	17400	0.34
FF	622	106400	0.58
BRAM	2	140	1.43

Figure 8 Summary of the Resources Needed for the Encoder Design Process

The utilization report of the encoder design when it is implemented on Zynq-7000 ZC702 Evaluation Board Part xc7z020clg484-1 given below in figure (8). At the Receiver, the synthesis results of Bit Flipping LDPC Decoder when using array partitioning and loop unrolling directives are shown in figure (9).

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.726	1.25

Latency (clock cycles)

Summary

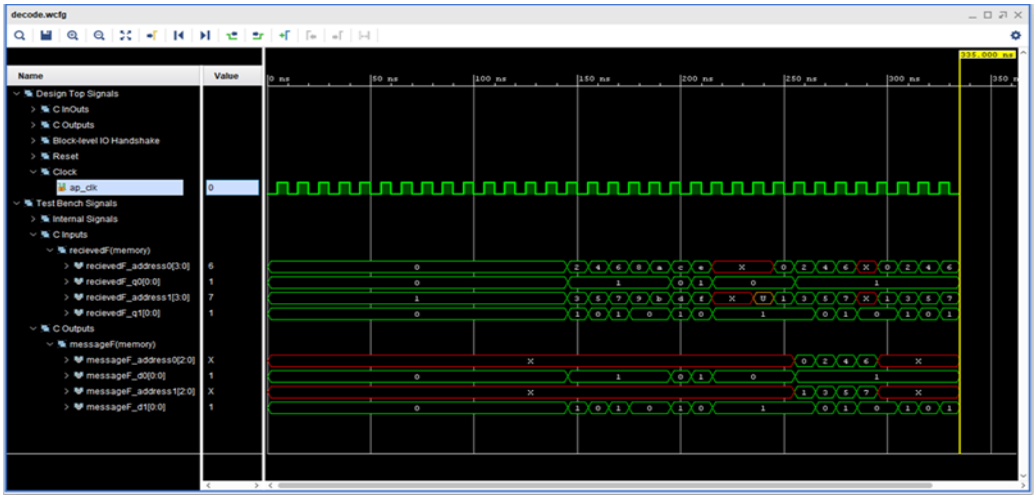
Latency		Interval		
min	max	min	max	Type
15	39	15	39	none

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	145
FIFO	-	-	-	-
Instance	-	-	153	1430
Memory	-	-	-	-
Multiplexer	-	-	-	261
Register	-	-	44	-
Total	0	0	197	1836
Available	280	220	106400	53200
Utilization (%)	0	0	~0	3

Figure 9 Bit Flipping LDPC Decoder Synthesis Results



Figures 10 Input and Output Signals to the Decoder with Using Loop Unrolling Directive

The input data to the decoder, if there is no noise, will be (1110111010011001) from address $(0)_{hex}$ to address in $(F)_{hex}$ and the output data from the decoder is (11101110) from address $(0)_{hex}$ to address $(7)_{hex}$. when using array partitioning and loop unrolling directives, the waveform of the Bit Flipping LDPC Decoder as shown in figure (10). Figure 11 illustrates the proposed design for the Decoder using Zynq_7000 platform. The Zynq_7000 platform. The Zynq_AXI_lite interface_connection is used to connect the algorithm_core with other FPGA hardware as the PS.

As for Figure (12) it represents the results of implementing LDPC Decoder after projecting the design onto the Zynq board, which was done through the use of the SDK program. Code Word (1110101011011001) and retrieve the original message (11101110). The utilization report of the decoder design when it is implemented on Zynq-7000 ZC702 Evaluation Board Part xc7z020clg484-1 given below in figure (13).

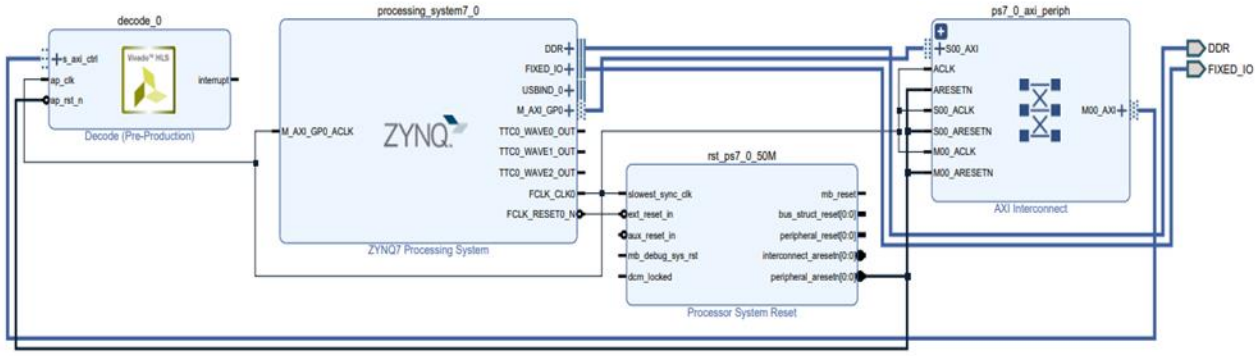


Figure 11 Implementation of Decoder

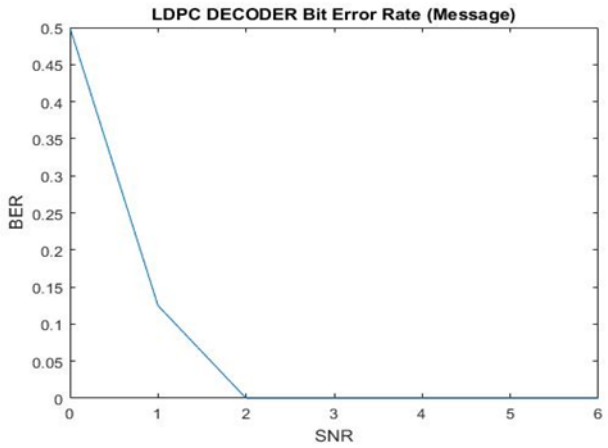
Figure 12 Implementation of Decoder on Zynq

Summary

Resource	Utilization	Available	Utilization %
LUT	670	53200	1.26
LUTRAM	60	17400	0.34
FF	800	106400	0.75
BRAM	2	140	1.43

Figure 13 Summary of the Resources Needed for the Encoder Design Process

The performance of the Bit Flipping LDPC Decoder was analyzed and the software simulation results of the BER of the hardware decoder is shown in figure (14). It can be seen, the reverse relation between BER and SNR.



Figures 14 BER Performance of the Bf Decoder

5. CONCLUSION

The LDPC Bit Flipping Algorithm decoder is used in this work. LDPC "Encoder and Decoder" was developed utilizing HLS technique in "Xilinx Vivado18.3" and for a "regular-LDPC" matrix of "8x16" with 7 rows and 6 columns. "Xilinx Vivado18.3" on Zynq-7000 Evaluation Board, Part xc7z020clg484-1 was used to simulate and synthesize these designs. The system's results showed that it works properly, and that such a decoder can be used with a new communication system because of its high throughput, low complexity, and low BER.

REFERENCES

[1] S. Pawankar and N. Mohota, "High Performance LDPC Decoder design using FPGA," in *2019 9th International Conference on Emerging Trends in Engineering and Technology-Signal and Information Processing (ICETET-SIP-19)*, 2019, pp. 1-4.
[2] R. Gallager, "Low-density parity-check codes," *IRE Transactions on information theory*, vol. 8, pp. 21-28, 1962.
[3] D. J. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electronics letters*, vol. 32, pp. 1645-1646, 1996.
[4] H. Kim, *Design and Optimization for 5G Wireless Communications*: John Wiley & Sons, 2020.
[5] K. D. Rao, *Channel coding techniques for wireless communications*: Springer, 2015.
[6] T.-D. Chiueh, P.-Y. Tsai, L. I-Wei, and T.-D. Chiueh, *Baseband receiver design for wireless MIMO-OFDM communications*: Wiley Online Library, 2012.
[7] S. A. Alabady, "Binary and non- binary low density parity check codes: a survey," *Int J Inf Eng Appl*, vol. 1, pp. 104-117, 2018.
[8] M. G. Prasad, C. C. Reddy, and J. C. Babu, "VLSI Implementation of decoding algorithms using EG-LDPC Codes," *Procedia computer science*, vol. 115, pp. 143-150, 2017.
[9] P. Raju and P. S. Prasad, "Design of an LDPC Decoder and Its Performance," *International Journal of Electrical and Electronics communication*, vol. 1, 2017.
[10] N. J. Gaurihar, I. R. Khadse, T. S. Ghonade, A. Borkar, A. Singh, and M. Patil, "Design and implementation of LDPC codes and turbo codes using FPGA," *Int. Res. J. Eng. Technol.*, vol. 3, pp. 1683-1687, 2016.
[11] G. Choi, K.-B. Park, and K.-S. Chung, "Optimization of FPGA-based LDPC decoder using high-level synthesis," in *Proceedings of the 4th International Conference on Communication and Information Processing*, 2018, pp. 256-259.
[12] A. T. Ali and D. A. F. Alneema, "Design Analysis of Turbo Decoder Based on One MAP Decoder Using High Level Synthesis Tool," *Al-Rafidain Engineering Journal (AREJ)*, vol. 25, pp. 70-77, 2020.

[13] T. T. Nguyen-Ly, V. Savin, K. Le, D. Declercq, F. Ghaffari, and O. Boncalo, "Analysis and design of cost-effective, high-throughput LDPC decoders," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, pp. 508-521, 2017.
[14] L. S. N. Rao, K. V. Sathyajith, and Y. N. Yadav, "Encoding and Decoding of LDPC Codes using Bit Flipping Algorithm in FPGA," *International Journal of Innovative Research in Computer and Communication Engineering*, vol. 5, 2017.
[15] B. M. Younis and A. K. Younis, "Hardware accelerator for anti-aliasing Wu's line algorithm using FPGA," *Telkomnika*, vol. 19, pp. 672-682, 2021.
[16] M. Yasoubi, "An Efficient Hardware Implementation of LDPC Decoder," Concordia University, 2020.
[17] B. Unal, "Low-Density Parity-Check Code Decoder Design and Error Characterization on an FPGA Based Framework," 2019.
[18] B. S. Reddy and V. S. R. Rao, "FPGA Implementation of LDPC Encoder and Decoder using Bit Flipping Algorithm," *International Journal of Science and Research*, vol. 6, pp. 1683-1690, 2017.
[19] H. A. Orabi, A. Zekry, and G. Gomah, "Implementation for two-stage hybrid decoding for low density parity check (LDPC) codes," *International Journal of Computer Applications*, vol. 80, 2013.
[20] A. Singh and S. G. Sharma, "Implementation of the Low complexity and High throughput LDPC Decoder," 2013.
[21] V. A. Chandrasetty and S. M. Aziz, "FPGA implementation of a LDPC decoder using a reduced complexity message passing algorithm," *Journal of Networks*, vol. 6, p. 36, 2011.
[22] K. Le, D. Declercq, F. Ghaffari, C. Spagnol, E. Popovici, P. Ivanis, *et al.*, "Efficient realization of probabilistic gradient descent bit flipping decoders," in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2015, pp. 1494-1497.
[23] F. S. H. Mahmood F. Mosleh1, Aya H. Abdulhameed3, "FPGA Hardware Co-Simulation of DCSK Based on Bit-Flipping LDPC Decoding Using Xilinx System Generator," *International Journal of Advanced Science and Technology*, vol. 29,, p. 18, 2020.
[24] Amer T. Ali and Dhafir A. Alneema, " Design and Analysis Combining Two Algorithms in One Turbo Decoder ", *IJIRCCE*, Vol. 8, Issue 8, August 2020.

Authors

Arwa H. Ashou received the B.Sc. degree in Computer Engineering from Mosul University, Mosul, Iraq, in 2011. She is currently working toward the M.Sc. degree in Computer Engineering at the College of Engineering in Mosul University. She is interested in doing research in Computer Architecture, FPGA technology, and Digital Communications and Signal Processing Circuits Design.

Dhafir A. Alneema is a Lecturer at the Computer Engineering Department, University of Mosul, Mosul, Iraq. He received his PhD in Computer Engineering from University of Mosul in 2008. His current research focuses on designing and implementing the different architectures for communication and digital signal processing systems using FPGA.