SSRG International Journal of Computer Science and Engineering
ISSN: 2348 – 8387 /doi:10.14445/23488387/IJCSE-V8I7P102

Volume 8 Issue 7, 10-15, July 2021
©2021 Seventh Sense Research Group®

# HLS Design of Min Sum Decoding Algorithm on Zynq

Arwa H. Ashou[#1], Dhafir A. Alneema[*2]

[#]*Computer Engineering Department, College of Engineering, University of Mosul, Mosul, Iraq*

Received Date: 19 June 2021
Revised Date: 23 July 2021
Accepted Date: 04 August 2021

***Abstract -*** *The Low-Density Parity Check (LDPC) codes are an important aspect of 5 G communication systems. This code is a forward error correction block code that corrects errors by iteratively performing decoding operations. Using High-Level Synthesis (HLS) techniques, however, this paper presents a high-performance Min Sum LDPC decoder. HLS for FPGAs is widely used as a good hardware synthesis tool due to one of the key advantages of FPGAs is flexibility. This paper uses an optimization technique including array partitioning and loop unrolling to minimize latency and increase throughput. The results showed that the implementation speed was increased. For simulation results, Xilinx Vivado HLS 18.3 is used on Zynq-7000 Evaluation Board Part xc7z020clg484-1.*

**Keywords:** *LDPC, Min Sum, FPGA, HLS, ZYNQ.*

## I. INTRODUCTION

Gallagher discovered the "Low_Density Parity_Check" (LDPC) code in 1962[1], which is a type of "linear block code". This code has a high error-correction performance that is close to the Shannon limit [2], as well as a sparse parity check matrix [3]. Furthermore, because of the intermediate decoding complexity combined with the high level of parallelism in hardware implementation, these codes were well suited for new wireless communication systems. LDPC codes are widely used in Wireless LAN (IEEE 802.11n), Mobile WiMax (IEEE 802.16e) [6], and 10 Gb/s Ethernet (10GBASE-T). In addition to satellite television, space communications, and other industries [7]. The Density Parity Check (LDPC) codes are used extensively for various communication and storage systems due to their excellent ability to correct errors [8].

The LDPC codes are two kinds, binary and non-binary[9] [10]. For large block lengths, binary LDPC codes demonstrate good error correction and use of channel capacity. Because of the short cycles in the parity matrix, short word lengths show poorer performance. Galois field GF(q) [11] has defined non-binary LDPC codes [12]. Binary LDPC codes can be classified into regular and irregular LDPC codes [13] [14]. If column and row weights are constant, the LDPC code will be regular. Otherwise, The LDPC code is irregular [11].

In this paper, we propose a binary regular LDPC Min-Sum decoding algorithm using the optimization techniques of HLS tools (array partitioning and loop unrolling) to minimize latency [15] [16]. The Min-Sum algorithm is the most efficient LDPC decoding algorithm, which is referred to as a soft decision algorithm. The minimum algorithm is widely used to implement LDPC decoders in hardware. Since no estimates of the Signal-Noise (SNR) channel over the AWGN channel are required, it also is low in complexity and robust to quantize [17].

## II. RELATED WORK

The implementation of the LDPC hardware has attracted many researchers around the world. One of these studies is Yi, and Xiaorong [18] proposed a partial-parallel approach for the LDPC decoder. The general architecture of the design is MIMD-based, while the internal calculation unit is SIMD-based. A programmable method was used with the processor, and Normalized Minimum Sum (NMS) was involved. The results reflected higher fast calculation speed and a simple chip layout. The authors used "Xilinx Kintex-7" and "Verilog" for writing code of the decoder ("QC-LDPC") with a ¾ rate and 2304 bits of code length. The authors also used software called VC6.0 to randomly generate the binary sequence (source sequence). The source sequence was coded using MATLAB. The modulation was performed using BPSK on the AWGN channel. The signal-to-noise ratio was 2 dB, with the proposed structure fits the case when having a multi-bit rate and multi-code long decoders[19].

The Min-Sum algorithm has been widely involved in the design of the LDPC decoders. This is due to the efficiency that can be obtained when using it. Many developers around the world use the original version of this algorithm. However, many researchers adjust the algorithm by performing modifications aiming at fitting the needs. Liu et al [20] presented an approach for modifying the Min-Sum Algorithm (MSA). The modification included a threshold called "Threshold Attenuated", and here the algorithm was called TAMSA. Its goal was to improve the Bit-Error Rate (BER) with no additional cost required compared to the conventional AMSA. The TAMSA showed that the modified version of MSA did not require extra area or power for circuits. Besides, the authors in [20] investigated the layered features of the aforementioned approaches (MSA, AMSA, and TAMSA). Romani [21] an approach that aimed to accelerate the LDPC decoder in 5G networks. The author used software developed by OpenairInterface Software Alliance consortium (OAI). The code was explored using OAI and then optimized using the SDAccel environment under Xilinx, which included the corresponding bitstream. In implementing the decoder, HLS was used and the synthesizing was performed using Vivado HLS.

Another study that was performed by Anantharaman et al. [22] aimed to modify the MSA algorithm and then optimize it for the LDPC decoders. The implementation was performed for (8 x 12) parity-check matrix, using Zynq-702 board, and the results were obtained using Vivado. The optimized MSA used a factor that was introduced in the horizontal step aiming at decreasing the iterations of the decoder as well as reducing the complexity of the FPGA used. Furthermore, Pawankar and Mohota [23] proposed a low complexity approach of the LDPC decoders. Three processes were involved, the encoding included 324 message bit that created 648 bit encoded message.

The second process included the use of the AWGN channel for passing the message, then the third process used MSA to produce the exact message. Variable Node Unit (VNU) and Check Node Unit (CNU) were the main components of the LDPC decoder structure, which decreased the complexity of the proposed structure. Fewer slice resources were obtained using the MSA algorithm. Also, minimum resources were consumed since the design included multiplexed structure. The work of Zhou and Zhang [24] suggested two obfuscation designs for the LDPC. The results showed that both of the proposed approaches were resistant to the removal attacks, AppSAT, and Sat. Also, the throughput was reduced to 1/3 and the error rate was increased with an area overhead of approximately 0.33%. Alvarez et al [25] proposed a design for the Non-Binary LDPC (NB-LDPC). The design was implemented for the “NB-LDPC” code (128,64) under the “Virtex-5QV FPGA”. The results showed that the proposed design outperformed the Standard-Binary Design of the LDPC decoder with 0.7dB of coding gain.

Li et al. [25] [26]. proposed a graph-based iterative detection approach for the LDPC decoder. The designed 2D detector included downtrack that was based-on Sum-Product algorithm (SPA). The LDPC-decoder was based-on Simplified-Check-Node (SCN) aiming at providing soft information for the channel of the 2D detector. The experimental results showed that the proposed approach outperformed the trellis-based BCJR over 2x2 2D channels.

Kuc [27] proposed an approach that proved the decoder operations correctness and was implemented using Intel Cyclone. The authors also estimated the power consumption, which is useful when it is needed to know the future consumption of power. In this approach, the authors implemented a matrix of 512x1024 for parity check with (3,6) regular code. The MSA algorithm was also involved aiming at reducing the complexity of the approach.

The study of Boudaoud and Abdelmounim [28] also presented a parallel design that aimed to produce a low complexity LDPC in applications that need a high rate of data transmission. For this purpose, they use FPGA Altera EPC4CE115F29C7 and the MSA algorithm. The simulations validated the Bit Error Rate (BER) of the proposed design. The experiments were performed based on three measurements; latency, data rate, complexity, and SNR Vs BER when varying the quantifications and the number of iterations.Another study [29] tried to optimize the MSA algorithm performed by Zhang and Qi. The proposed approach gained low complexity with a performance that was approximately close to the one gained by Sum-Product decoding. The proposed approach was well-fitted to the VLSI implemented on the Xilinx FPGA family.Heidari, T., & Jannesari [30] used a QC-LDPC decoder that was based on the MSA algorithm. The proposed approach aimed to increase the throughput of the decoder codes. They performed processing on the rows and columns to reduce the clock cycle for each iteration. The design was developed for the 802.16e standard and a ½ rate with 2304 code length.

## III. Implementation of Min Sum Decoding Algorithm

In this paper, the ZC702, composed with the Zynq-7000, has been used as a hardware platform to carry out the design proposed. Combined “processing system” (PS) and “programmable logic” (PL) are integrated into one single chip [31]. Fig. 1 illustrate the proposed design for Min Sum algorithm only using this platform.

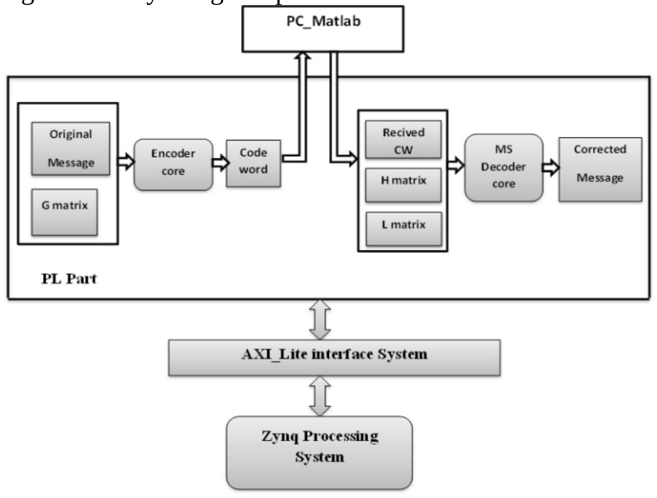


**Fig. 1: Design based on Min Sum Algorithm on Zynq Board**

This interface connection is made with the Zynq AXI lite interface connection with a hardware core of the min Sum algorithm with other hardware elements (e.g., processing system) in the FPFA environment. Then, the corrected message will be sent to the PC through a serial port for further processing by the hardware core of the designed algorithm. The algorithm Min Sum has been proposed to allow high-speed, efficient LDPC decoding. The best performing decoding method is the Min Sum algorithm [32] [33]. It is a creed propagation iterative decoding algorithm that is better for decoding codes (LDPC). This algorithm consists of 3 steps to complete the decoding algorithm as shown in the flowchart fig. 2. These steps are the Initialization step, the Horizontal step, and the Vertical step. [32].

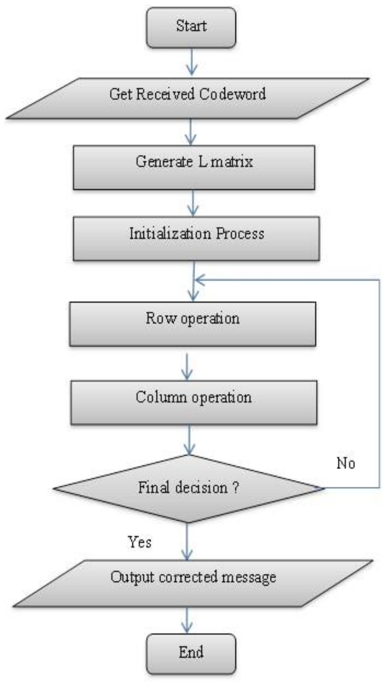


**Fig. 2: Min Sum Decoder Flowchart**

**Step1:** Generating H matrix from G matrix

$$G = [I\ P] \tag{1}$$

G matrix :

$$\begin{pmatrix}
1&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1\\
0&1&0&0&0&0&0&0&1&0&0&1&1&1&1&1\\
0&0&1&0&0&0&0&0&1&1&0&0&1&1&1&1\\
0&0&0&1&0&0&0&0&1&1&1&0&0&1&1&1\\
0&0&0&0&1&0&0&0&1&1&1&1&0&0&1&1\\
0&0&0&0&0&1&0&0&1&1&1&1&1&0&0&1\\
0&0&0&0&0&0&1&0&1&1&1&1&1&1&0&0\\
0&0&0&0&0&0&0&1&0&1&1&1&1&1&1&0
\end{pmatrix}$$

$$H = [P^T\ I] \tag{2}$$

H matrix :

$$\begin{pmatrix}
0&1&1&1&1&1&1&0&1&0&0&0&0&0&0&0\\
0&0&1&1&1&1&1&1&0&1&0&0&0&0&0&0\\
1&0&0&1&1&1&1&1&0&0&1&0&0&0&0&0\\
1&1&0&0&1&1&1&1&0&0&0&1&0&0&0&0\\
1&1&1&0&0&1&1&1&0&0&0&0&1&0&0&0\\
1&1&1&1&0&0&1&1&0&0&0&0&0&1&0&0\\
1&1&1&1&1&0&0&1&0&0&0&0&0&0&1&0\\
1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&1
\end{pmatrix}$$

To Generate L matrix

Assuming CW generated by LDPC encoder after applying Additive White Gaussian noise with SNR=2 dB.

The Codeword will be :

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| -2.0365 | -3.2868 | -0.9743 | 1.3409 | -0.7558 | 0.2736 | -1.4532 | 0.1983 | -1.2979 | -0.9599 | 1.0210 | -0.6603 | -1.6402 | 2.0212 | 1.8491 | -2.1641 |

Then L matrix :

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0.000000 | -3.286800 | -0.974300 | 1.340900 | -0.755800 | 0.273600 | -1.453200 | 0.000000 | -1.297900 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 0.000000 | 0.000000 | -0.974300 | 1.340900 | -0.755800 | 0.273600 | -1.453200 | 0.198300 | 0.000000 | -0.959900 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| -2.036500 | 0.000000 | 0.000000 | 1.340900 | -0.755800 | 0.273600 | -1.453200 | 0.198300 | 0.000000 | 0.000000 | 1.021000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| -2.036500 | -3.286800 | 0.000000 | 0.000000 | -0.755800 | 0.273600 | -1.453200 | 0.198300 | 0.000000 | 0.000000 | 0.000000 | -0.660300 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| -2.036500 | -3.286800 | -0.974300 | 0.000000 | 0.000000 | 0.273600 | -1.453200 | 0.198300 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | -1.640200 | 0.000000 | 0.000000 | 0.000000 |
| -2.036500 | -3.286800 | -0.974300 | 1.340900 | 0.000000 | 0.000000 | -1.453200 | 0.198300 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 2.021200 | 0.000000 | 0.000000 |
| -2.036500 | -3.286800 | -0.974300 | 1.340900 | -0.755800 | 0.000000 | 0.000000 | 0.198300 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.849100 | 0.000000 |
| -2.036500 | -3.286800 | -0.974300 | 1.340900 | -0.755800 | 0.273600 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | -2.164100 |

**Step 2**: Apply row operation

In place computation using L matrix

To calculate the value of the elements, the two smallest values are taken without a sign. The smallest value for which the absolute value is taken and given to all non-zero elements. As for the next least value, it is given to replace the minimum value after the absolute value is taken for it as well. While calculating the signs of the elements, the signs of the elements of the entire class are multiplied. This value is called a party. Then the sign of each element is calculated by multiplying the party with the previous sign.

Row 1 in L matrix

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | MIN2 | MIN1 | | | | | | | | | | |
| 0.000000 | -3.286800 | -0.974300 | 1.340900 | -0.755800 | 0.273600 | -1.453200 | 0.000000 | -1.297900 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |

After Row operation in row 1:

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0.000000 | 0.273600 | 0.273600 | -0.273600 | 0.273600 | -0.755800 | 0.273600 | 0.000000 | 0.273600 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |

Row 2 in L matrix

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | MIN1 | | | | | MIN2 | | | | | | | | |
| 0.000000 | 0.000000 | -0.198300 | 0.198300 | -0.198300 | 0.198300 | -0.198300 | 0.273600 | 0.000000 | -0.198300 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |

After Row operation in row 2:

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0.000000 | 0.000000 | -0.198300 | 0.198300 | -0.198300 | 0.198300 | -0.198300 | 0.273600 | 0.000000 | -0.198300 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |

Row operation after 8 rows :

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0.000000 | 0.273600 | 0.273600 | -0.273600 | 0.273600 | -0.755800 | 0.273600 | 0.000000 | 0.273600 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 0.000000 | 0.000000 | -0.198300 | 0.198300 | -0.198300 | 0.198300 | -0.198300 | 0.273600 | 0.000000 | -0.198300 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 0.198300 | 0.000000 | 0.000000 | -0.198300 | 0.198300 | -0.198300 | 0.198300 | -0.273600 | 0.000000 | 0.000000 | -0.198300 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 0.198300 | 0.198300 | 0.000000 | 0.000000 | 0.198300 | -0.198300 | 0.198300 | -0.273600 | 0.000000 | 0.000000 | 0.000000 | 0.198300 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 0.198300 | 0.198300 | 0.198300 | 0.000000 | 0.000000 | -0.198300 | 0.198300 | -0.273600 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.198300 | 0.000000 | 0.000000 | 0.000000 |
| -0.198300 | -0.198300 | -0.198300 | 0.198300 | 0.000000 | 0.000000 | -0.198300 | 0.974300 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.198300 | 0.000000 | 0.000000 |
| -0.198300 | -0.198300 | -0.198300 | 0.198300 | -0.198300 | 0.000000 | 0.000000 | 0.755800 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.198300 | 0.000000 |
| 0.273600 | 0.273600 | 0.273600 | -0.273600 | 0.273600 | -0.755800 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.273600 |

**Step 3:** Apply column operation

In place computation using L matrix

- New values

Sum[j] = r[j] + sum of all entries in column j

New entry = Sum – (Old entry)

Codeword will be:

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| -2.0365 | -3.2868 | -0.9743 | 1.3409 | -0.7558 | 0.2736 | -1.4532 | 0.1983 | -1.2979 | -0.9599 | 1.0210 | -0.6603 | -1.6402 | 2.0212 | 1.8491 | -2.1641 |

The L matrix will be

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0.000000 | -3.286800 | -0.974300 | 1.340900 | -0.755800 | 0.273600 | -1.453200 | 0.000000 | -1.297900 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 0.000000 | 0.000000 | -0.974300 | 1.340900 | -0.755800 | 0.273600 | -1.453200 | 0.198300 | 0.000000 | -0.959900 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| -2.0365 | 0.000000 | 0.000000 | 1.340900 | -0.755800 | 0.273600 | -1.453200 | 0.198300 | 0.000000 | 0.000000 | 1.021000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| -2.0365 | -3.286800 | 0.000000 | 0.000000 | -0.755800 | 0.273600 | -1.453200 | 0.198300 | 0.000000 | 0.000000 | 0.000000 | -0.660300 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| -2.0365 | -3.286800 | -0.974300 | 0.000000 | 0.000000 | 0.273600 | -1.453200 | 0.198300 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | -1.640200 | 0.000000 | 0.000000 | 0.000000 |
| -2.0365 | -3.286800 | -0.974300 | 1.340900 | 0.000000 | 0.000000 | -1.453200 | 0.198300 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 2.021200 | 0.000000 | 0.000000 |
| -2.0365 | -3.286800 | -0.974300 | 1.340900 | -0.755800 | 0.000000 | 0.000000 | 0.198300 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.849100 | 0.000000 |
| -2.0365 | -3.286800 | -0.974300 | 1.340900 | -0.755800 | 0.273600 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | -2.164100 |

Column operation in column 1:

Sum=  -1.564600

Column operation in column 2:

Sum= -2.739600

After Column operation in column 8:

Sum=

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| -1.564600 | -2.739600 | -0.823700 | 1.190300 | -0.208600 | -1.634600 | -0.981300 | 1.381200 | -1.024300 | -1.158200 | 0.822700 | -0.462000 | -1.564600 | -2.739600 | -0.823700 | 1.190300 |

Decision:

If sum[j]>=0, Decision on bit j=0

If sum[j]<0, Decision on bit j=1

Assuming BPSK 0→1, 1→-1

After the first iteration:

Sum= 1 1 1 0 1 1 1 0 1 1 0 1 1 1 1 0, message and parity

Then message = 1 1 1 0 1 1 1 0

The message was corrected for more iteration; continue with the new L matrix.

The new L matrix will be:

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0.000000 | -3.013200 | -1.097300 | 1.463900 | -0.482200 | -0.878800 | -1.254900 | 0.000000 | -1.297900 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 0.000000 | 0.000000 | -0.625400 | 0.992000 | -0.010300 | -1.832900 | -0.783000 | 1.107600 | 0.000000 | -0.959900 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| -1.762900 | 0.000000 | 0.000000 | 1.388600 | -0.406900 | -1.436300 | -1.179600 | 1.654800 | 0.000000 | 0.000000 | 1.021000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| -1.762900 | -2.937900 | 0.000000 | 0.000000 | -0.406900 | -1.436300 | -1.179600 | 1.654800 | 0.000000 | 0.000000 | 0.000000 | -0.660300 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| -1.762900 | -2.937900 | -1.022000 | 0.000000 | 0.000000 | -1.436300 | -1.179600 | 1.654800 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | -1.640200 | 0.000000 | 0.000000 | 0.000000 |
| -1.366300 | -2.541300 | -0.625400 | 0.992000 | 0.000000 | 0.000000 | -0.783000 | 0.406900 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 2.021200 | 0.000000 | 0.000000 |
| -1.366300 | -2.541300 | -0.625400 | 0.992000 | -0.010300 | 0.000000 | 0.000000 | 0.625400 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.849100 | 0.000000 |
| -1.838200 | -3.013200 | -1.097300 | 1.463900 | -0.482200 | -0.878800 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | -2.164100 |

The core design of the Min Sum Decoder using the zynq platform is presented as shown in the flow chart in Fig 3.

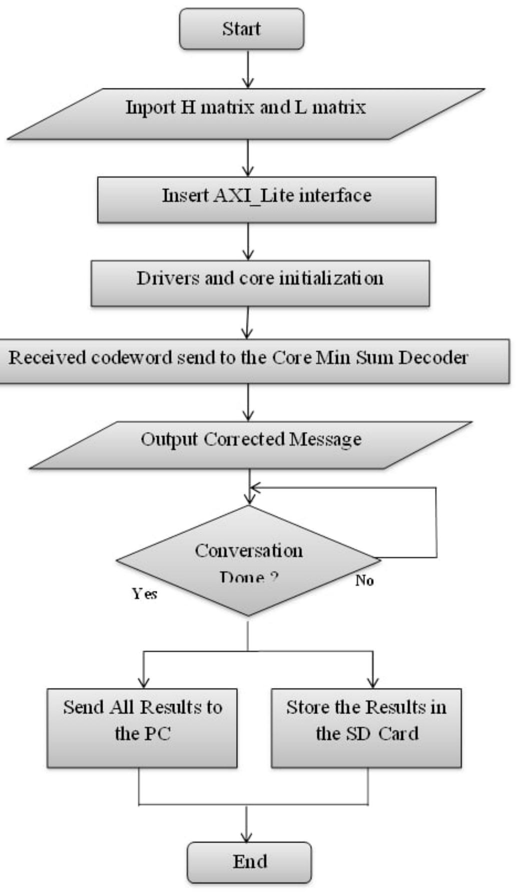
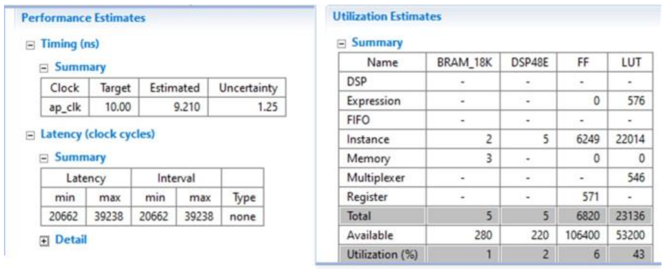

Start
Inport H matrix and L matrix
Insert AXI_Lite interface
Drivers and core initialization
Received codeword send to the Core Min Sum Decoder
Output Corrected Message
Conversation Done ?
Yes
No
Send All Results to the PC
Store the Results in the SD Card
End

**Fig. 3: Min Sum Decoder Core Flowchart**

## IV. RESULTS

The Vivado Design Suite package is used in this project to implement all design processes and optimize the design to meet the time limits required. Xilinx's 2018.3 Vivado Design Suit now supports Zynq ZC702 with various FPGA devices. The synthesis results of the Min Sum Decoder when using array partitioning and loop unrolling directives are shown in fig. 4.



Performance Estimates

Timing (ns)

Summary

| Clock | Target | Estimated | Uncertainty |
|---|---|---|---|
| ap_clk | 10.00 | 9.210 | 1.25 |

Latency (clock cycles)

Summary

| Latency | | Interval | | |
|---|---|---|---|---|
| min | max | min | max | Type |
| 20662 | 39238 | 20662 | 39238 | none |

Detail

Utilization Estimates

Summary

| Name | BRAM_18K | DSP48E | FF | LUT |
|---|---|---|---|---|
| DSP | - | - | - | - |
| Expression | - | - | 0 | 576 |
| FIFO | - | - | - | - |
| Instance | 2 | 5 | 6249 | 22014 |
| Memory | 3 | - | 0 | 0 |
| Multiplexer | - | - | - | 546 |
| Register | - | - | 571 | - |
| Total | 5 | 5 | 6820 | 23136 |
| Available | 280 | 220 | 106400 | 53200 |
| Utilization (%) | 1 | 2 | 6 | 43 |

**Fig. 4: Min Sum Decoder synthesis results**

The input data to the MS decoder with 9 iterations , if SNR=1 dB, will be (-2.0365 -3.2868 -0.9743 1.3409 -0.7558 0.2736 -1.4532 0.1983 -1.2979 -0.9599 1.0210 -0.6603 -1.6402 2.0212 1.8491 -2.1641 ) and the output data from the

MS decoder is (11101110). when using array partitioning and loop unrolling directives, the waveform of the Min Sum Decoder, as shown in fig. 5.

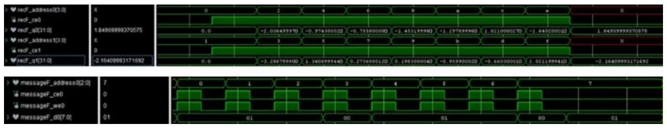


**Fig. 5: Input and output signals to the MS Decoder using loop unrolling directive.**

Fig. 6 shows the proposed Min Sum Decoder design using the platform Zynq 7000. The zynq "AXI lite interface" is used to connect the algorithm's hardware core to other FPGA hardware components, such as the zynq processing system, where the algorithm is executed.

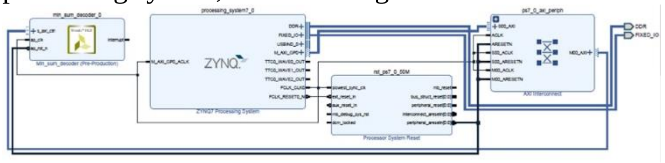
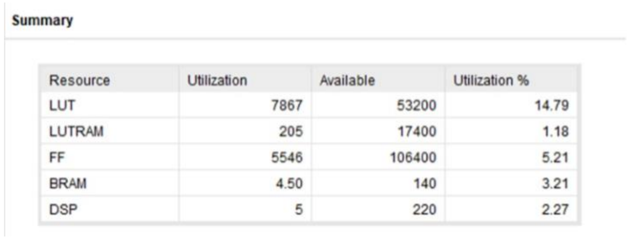


**Fig. 6: Implementation of Min Sum Decoder.**

The utilization report of the Min Sum decoder design when it is implemented on Zynq-7000 ZC702 Evaluation Board Part xc7z020clg484-1 is given below in fig. 7.

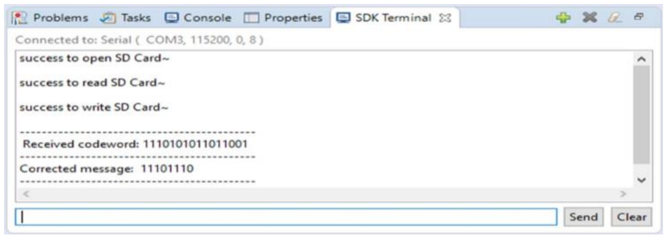


Summary

| Resource | Utilization | Available | Utilization % |
|---|---|---|---|
| LUT | 7867 | 53200 | 14.79 |
| LUTRAM | 205 | 17400 | 1.18 |
| FF | 5546 | 106400 | 5.21 |
| BRAM | 4.50 | 140 | 3.21 |
| DSP | 5 | 220 | 2.27 |

**Fig. 7: Summary of the resources needed for the Min Sum Decoder design process.**

Fig. 8 represents the results of implementing Min Sum Decoder after projecting the design onto the Zynq board, which was done through the use of the SDK program. The received codeword is (1110101011011001), and retrieve the original message is (11101110).



```
Connected to: Serial ( COM3, 115200, 0, 8 )
success to open SD Card~
success to read SD Card~
success to write SD Card~
--------------------------------
Received codeword: 1110101011011001
--------------------------------
Corrected message: 11101110
--------------------------------
```

**Fig. 8: Implementation of Min Sum Decoder on Zynq.**

The Min Sum LDPC decoder performance has been analyzed, and BER hardware decoder software simulation results can be shown in fig. 9. It can be seen the reverse relation between BER and SNR.

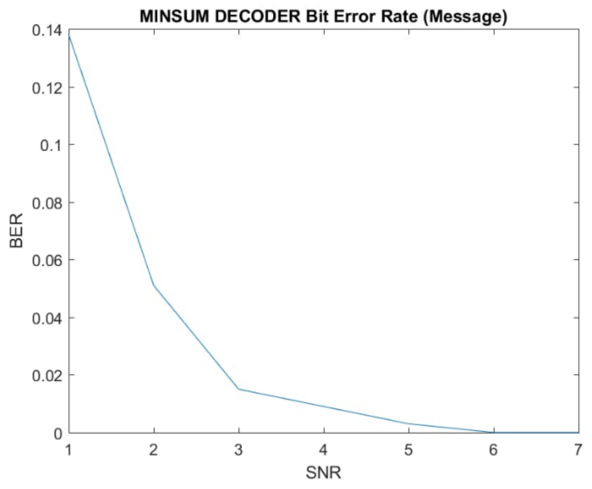


**Fig. 9: BER performance of the MS Decoder.**

## V. CONCLUSIONS

The Min Sum Algorithm decoder is used in this work. LDPC Encoder and Min Sum Decoder were developed using the HLS technique in Xilinx Vivado 18.3 and for a regular LDPC matrix of 8x16 with row weight of 7 and column weight of 6. Xilinx Vivado 18.3 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] R. Gallager., Low-density parity-check codes., IRE Transactions on information theory, 8 (1962) 21-28.

[2] D. J. MacKay and R. M. Neal., Near Shannon limit performance of low-density parity-check codes, Electronics Letters, 32(1996) 1645-1646.

[3] J. Campello, D. S. Modha, and S. Rajagopalan., Designing LDPC codes using bit-filling., in ICC 2001. IEEE International Conference on Communications. Conference Record (Cat. No. 01CH37240) (2001) 55-59.

[4] P.Vemaiah and S.Kannappan., Efficient Performance Analysis of IEEE802.11a Standard in Mobile Environment. SSRG International Journal of Electronics and Communication Engineering., 2 (6) (2015) 1-4.

[5] Shahzeb Hussain, Namrata Bhadri, and Md. Shaayan Hussain., Advancements in Wireless Communication., SSRG International Journal of Electronics and Communication Engineering., 7 (9) (2020) 1-4.

[6] G. Falcao, V. Silva, J. Marinho, and L. Sousa., LDPC decoders for the WiMAX (IEEE 802.16 e) based on multicore architectures., in WIMAX New Developments, ed: IntechOpen, (2009).

[7] H. Gharaee, M. Kiaee, and N. Mohammadzadeh., A high-throughput FPGA implementation of quasi-cyclic LDPC decoder., IJCSNS, 17(2017) 140.

[8] 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) 256-259.

[9] X. Zhang, VLSI architectures for modern error-correcting codes: Crc Press, (2017).

[10] M. C. Davey and D. J. MacKay., Low-density parity-check codes over GF (q), in 1998 Information Theory Workshop (Cat. No. 98EX131), (1998) 70-71.

[11] S. A. Alabady., Binary and non-binary low-density parity-check codes: a survey., Int J Inf Eng Appl, 1(2018) 104-117.

[12] N. Noorshams and M. J. Wainwright., Stochastic belief propagation: A low-complexity alternative to the sum-product algorithm, IEEE Transactions on Information Theory, 59(2012) 1981-2000.

[13] S. Lin and D. J. Costello, Error control coding., Prentice hall, 2(2001).

[14] K. D. Rao, Channel coding techniques for wireless communications: Springer, (2015).

[15] A. T. Ali and D. A. Alneema., Design Analysis of Turbo Decoder Based on One MAP Decoder Using High-Level Synthesis Tool, Al-Rafidain Engineering Journal (AREJ), 25 (2020) 70-77.

[16] Amer T. Ali and Dhafir A. Alneema., Design and Analysis Combining Two Algorithms in One Turbo Decoder, IJIRCCE, 8(8) (2020).

[17] Y. Liu, W. Tang, and D. G. Mitchell., Efficient Implementation of a Threshold Modified Min-Sum Algorithm for LDPC Decoders, IEEE Transactions on Circuits and Systems II: Express Briefs, 67(2020) 1599-1603.

[18] Q. Yi and Z. Xiaorong., Design of programmable parallel LDPC decoder, in 2017 IEEE 9th International Conference on Communication Software and Networks (ICCSN), (2017) 66-70.

[19] S.-W. Choi, G.-P. Kim, and J.-K. Kim., An LDPC decoder architecture for multi-rate QC-LDPC codes, in 2011 IEEE 54th International Midwest Symposium on Circuits and Systems (MWSCAS), (2011) 1-4.

[20] L. Romani., Hardware Acceleration of 5G LDPC using datacenter-class FPGAs, Politecnico di Torino, (2020).

[21] R. Anantharaman, K. Kwadiki, and V. P. Kerehalli Shankar Rao., Hardware Implementation Analysis of Min-Sum Decoders, Advances in Electrical and Electronic Engineering, 17(2019) 179-186.

[22] 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) 1-4.

[23] J. Zhou and X. Zhang., Algorithmic Obfuscation for LDPC Decoders, arXiv preprint arXiv:2104.03814, (2021).

[24] Y. Li, Z. Qin, L. Zou, Y. Qin, and Q. Lu., Graph-Based Detection and LDPC Decoding over 2D Intersymbol Interference Channels, J. Commun., 16(2021) 91-98.

[25] Z. Qin, Y. Li, Y. Qin, Q. Lu, and X. Liu., Graph-based detection and reduced-complexity LDPC decoding over 2D intersymbol interference channels, in Twelfth International Conference on Signal Processing Systems, (2021) 117190X.

[26] M. Kuc, W. Sułek, and D. Kania., Hardware implementation of the LDPC decoder in the FPGA structure, in AIP Conference Proceedings, (2021) 050002.

[27] B. M. Younis and A. K. Younis., Hardware accelerator for anti-aliasing Wu's line algorithm using FPGA, Telkomnika, 19 (2021) 672-682.

[28] A. Boudaoud, M. El Haroussi, and E. Abdelmounim., VHDL Design and FPGA Implementation of LDPC Decoder for High Data Rate., International Journal of Advanced Computer Science and Applications, 8(2017).

[29] S. ZHANG and W. Qi., Joint Design of Quasi-cyclic Low-Density Parity-Check Codes and Performance Analysis of Multi-source Multi-relay Coded Cooperative System, 41(2019) 2325-2333.

[30] T. Heidari and A. Jannesari., Design of high-throughput QC-LDPC decoder for WiMAX standard, in 2013 21st Iranian Conference on Electrical Engineering (ICEE), (2013) 1-4.

[31] M. R. Islam, D. S. Shafiullah, M. M. A. Faisal, and I. Rahman., Optimized min-sum decoding algorithm for low-density parity-check codes., International Journal of Advanced Computer Science and Applications, 2(2011) 168-174.

[32] V. A. Chandrasetty and S. M. Aziz., An area-efficient LDPC decoder using a reduced complexity min-sum algorithm, Integration, 45(2012) 141-148.

[33] X. Chen and C.-L. Wang., High-throughput efficient non-binary LDPC decoder based on the simplified min-sum algorithm, IEEE Transactions on Circuits and Systems I: Regular Papers, 59(2012) 2784-2794.