FPGA Implementation of Rate Matching in 5G NR PDSCH

Rochak Jain‡, Naveed Anjum‡, Samudrala Soujanya†, Harsha Rudramuniyappa†,
Aviral Jain ‡, Ashutosh Bisht‡, Ekant Sharma‡ and Prem Singh†
† Department of ECE, IIIT Bangalore, India
‡ Department of ECE, IIT Roorkee, India

***Abstract*— 5G new radio (NR) distinguishes itself by offering remarkable features such as significantly higher throughput and lower latency compared to 4G. In 5G, low density parity check (LDPC) encoding is employed as the channel coding scheme for the transmission of data bits. Subsequent to the LDPC encoding stage, bit selection and bit interleaving operations are performed. Rate matching plays an important role in selecting specific encoded bits for transmission, utilizing techniques like shortening and repetition to support hybrid automatic repeat request functionality. To address the issue of latency encountered in processing large transport blocks, we propose parallel algorithms for the rate matching. These algorithms are implemented on field programmable gate arrays, and then the performance of the optimized algorithms is compared with the existing algorithms specified in the 3rd generation partnership project standards.**

I. Introduction

The fifth generation cellular communication, also known as 5G new radio (NR), has brought significant advancements in wireless communication. One crucial aspect of 5G NR is the efficient transmission of data, which is achieved through various functionalities such as low density parity check (LDPC) encoding and rate matching, in the physical data channel. LDPC encoding is an error control coding technique used in 5G NR to provide reliable communication. Rate matching, on the other hand, is a key algorithm in 5G NR that ensures the compatibility of data rates between a source and the channel. It adjusts the size of the transmitted data to match the channel capacity and then rearranges the order of data bits before transmission to improve the error resilience of the transmitted data by spreading out consecutive bits, mitigating the impact of burst errors. Field programmable gate arrays (FPGA) implementation plays a vital role in optimizing the performance of these techniques and allows for the efficient execution of the aforementioned algorithms.

A previous study [1] has extensively analyzed rate matching algorithm based on circular buffers for long term evolution (LTE) systems. In addition, prior research [2] has investigated circular buffer rate matching and bit selection algorithms specifically tailored for 5G LDPC codes. The work [3] proposes an efficient rate matching implementation specifically designed for LTE systems. Furthermore, a separate study [4] analyzes the decoding latency of LDPC codes for 5G NR. In this paper, we propose a low latency algorithm for bit selection and bit interleaving in rate matching along with its FPGA implementation, and its performance is compared with the rate matching algorithm given in the 3rd generation partnership project (3GPP) standards.

In section II, we present an overview of the bit selection and bit interleaving procedures. In Section III, we present a detailed description of the proposed algorithm for rate matching, highlighting the key aspects of the algorithm and comparing it with the rate matching algorithm outlined in 3GPP standards. In section IV, we present the implementation of the aforementioned algorithm on an FPGA, evaluating its performance in terms of resource utilization and latency.

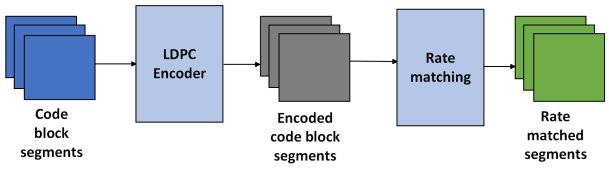

Fig. 1: Downlink data processing chain.

II. Rate Matching in 5G NR

Rate matching is a technique used to align the bit count in each encoded segment with the available transmission resources. 5G NR follows limited buffer rate matching (LBRM), where the input bits are written into a circular buffer, and subsequently selected based on the redundancy version (RV) index [5] set by the scheduler. The rationale behind rate matching is explained as follows: the decoding latency of LDPC codes [6] at the receiver is influenced by the code rate employed during transmission, whereby a higher code rate leads to reduced decoding latency, and conversely, a lower code rate increases decoding latency. LBRM helps achieve a flexible code rate by limiting the minimum code rate, thereby reducing decoding latency. The input, comprising systematic bits, filler bits, and parity bits, is fed into rate matching from the preceding channel encoder, which utilizes LDPC encoding for data transmission in 5G as shown in Fig. 1. In order to understand rate matching, it is essential to familiarize with the basic principles of LDPC encoder.

In 5G NR, LDPC is used as a channel encoding scheme [7] where two base graphs, namely BG_1 with dimensions 46×68 and a mother code rate of $\frac{1}{3}$, and BG_2 with dimensions 42×52 and a mother code rate of $\frac{1}{5}$, have been defined. These base graphs are designed to accommodate varying code block sizes, with the maximum code block size being 8448 bits for BG_1 and 3840 bits for BG_2. The general structure of a base graph is illustrated in Fig. 2, where the submatrix $\mathbf{A}$ represents the systematic bits, and the submatrices $\mathbf{B}$ and $\mathbf{D}$ correspond to the parity bits. To accommodate a broad spectrum of payload sizes, a set of 51 lifting sizes (Z_c) has been defined in Table 5.3.2-1 of TS 38.212 [5], and every element with value 1 in the base graph is substituted with a $Z_c \times Z_c$ identity matrix, circularly shifted based on the corresponding shift coefficient. Similarly, each element with

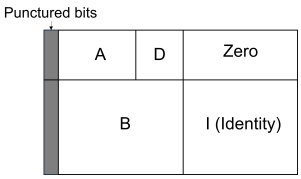

Fig. 2: LDPC base graph structure.

value 0 in the base graph is replaced with a $Z_c \times Z_c$ all-zero matrix. The number of bits per code block at the input of the LDPC encoder, denoted as K, is determined by the choice of the base graph and the lifting size Z_c. If base graph 1 is opted for, then K is calculated as $22 * Z_c$, and if base graph 2 is chosen, then K is determined as $10 * Z_c$. During LDPC encoding, the information bits corresponding to the first two columns of the base graph are disregarded. Therefore, the number of bits per code block at the output of the LDPC encoder is given by $N = 66 * Z_c$ for base graph 1, and $N = 50 * Z_c$ for base graph 2. If the size of the segmented code block after CRC addition is less than K, filler bits are added to make the size to the code block equal to K, prior to the encoding process.

Bit selection encompasses the elimination of filler bits introduced prior to LDPC encoding, followed by the selection of bits, wherein a specific segment of bits is chosen based on the RV index. Subsequently, the selected bits are rearranged using a bit interleaver.

A. Bit selection in rate matching

In the bit selection process, the number of bits in the code blocks is aligned with the number of available resources. Each encoded block is written into a circular buffer, and subsequently, an appropriate number of bits are read, starting from a specific bit location indicated by the RV index, disregarding any filler bits. In cases where the number of available resources exceeds the length of the encoded block, the encoded block is repeated to fill the available resources. Conversely, if the number of available resources is insufficient, the coded block is truncated to match the available resources. Bit selection in 5G NR involves the

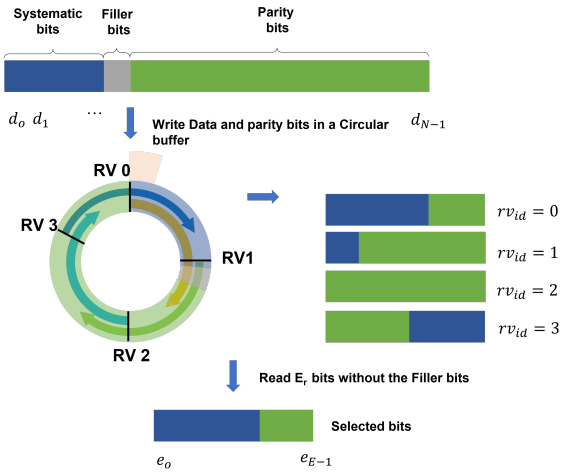

Fig. 3: Bit selection for redundancy version 0.

following steps as depicted in Fig. 3.

- The filler bits are removed, and then the appropriate bits are selected based on RV index. Each encoded block is denoted as $d_0, d_1, \ldots, d_{N_r-1}$, where N_r represents the length of the r-th encoded block which contains the systematic bits, the filler bits, and the parity bits. These input bits are written into a circular buffer of length N_{cb} where $N_{cb} = min(N_r, N_{ref})$, $N_{ref} = \lfloor \frac{TBS}{C \times R_{LBRM}} \rfloor$. Here TBS denotes the transport block size which is determined using clause 6.1.4.2 of TS 38.214 [8] and represents the length of the original data block before cyclic redundancy check (CRC) addition, segmentation and encoding. $R_{LBRM} = \frac{2}{3}$ as defined in [8], and C is the number of code blocks or segments. If $N_r > N_{cb}$, the trailing bits $d_{N_{cb}}, d_{N_{cb}+1}, \ldots, d_{N_r}$ are discarded, and the remaining bits are written into the circular buffer.
- The bits $e_0, e_1, \ldots, e_{E_r-1}$ are read from the circular buffer, disregarding the filler bits. The starting bit location k_0 depends on the base graph selected for LDPC and the RV index as per Table 5.4.2.1-2 in TS 38.212 [5]. The number of bits read for the rth encoded block is calculated using $E_r = N_L Q_m \lfloor \frac{G}{N_L Q_m C} \rfloor$ if $r \leq C - mod(\frac{G}{N_L Q_m}, C) - 1$, otherwise $E_r = N_L Q_m \lceil \frac{G}{N_L Q_m C} \rceil$, where N_L is the scheduled number of transmission layers, Q_m is the modulation order, G is the total number of coded bits available for transmission, C is the number of scheduled code block segments. The selected bits are input to the bit interleaver for further processing.

B. Bit interleaving in rate matching

In scenarios where a set of adjacent bits are subject to deep fade conditions, the receiver cannot effectively recover these symbols. Therefore, it is preferable to have errors dispersed randomly rather than concentrated in neighboring bits. To address this, the bits are interleaved before allocating them to REs. In the bit interleaver, the previously selected input bits are permuted by writing them into a table row-wise, and then reading them column-wise, effectively altering the order of the bits.

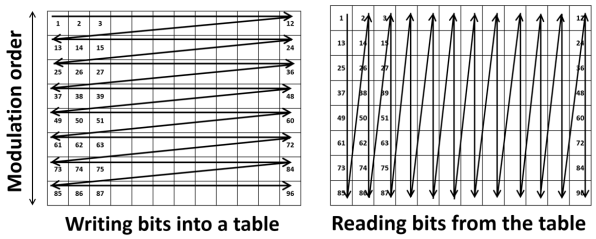

Fig. 4: Bit interleaving procedure.

Each encoded block after bit selection denoted as $e_0, e_1, \ldots, e_{E-1}$, serves as the input to the bit interleaver. These bits are sequentially written into a table row-wise, and subsequently read column-wise, as shown in Fig. 4. The number of rows in the table is equal to the modulation order Q_m, and the number of columns is $\frac{E}{Q_m}$, where E is the number of bits at the input of the bit interleaver.

III. Proposed Algorithm for Rate Matching

The proposed implementation of bit selection and bit interleaving in rate matching in high-level synthesis, vivado HLS, offers the advantages of low latency and low resource

utilization. Vivado HLS allows designers to describe IP functionality using high-level languages like C, C++, or SystemC. This higher level of abstraction simplifies verification and accelerates development compared to traditional RTL design. The HLS tool generates synthesizable RTL code based on the high-level language description, making it compatible with different FPGA devices. Vivado HLS also provides various optimization methods to enhance the functionality and space efficiency of the synthesized IP cores. These optimizations improve speed and reduce resource usage without explicit RTL coding.

A. *Efficient Low-Latency Algorithm for Rate Matching*

The rate matching algorithm for LDPC code, as outlined in the 3GPP standards, is illustrated in Fig. 5. After receiving the output from the LDPC encoder, the data contains extraneous filler bits that must be eliminated before applying the bit selection and bit interleaving operations. Therefore, our proposed approach divides the rate matching algorithm into two components: filler bit removal and interleaver. These algorithms are implemented as intellectual property (IP) modules in Vivado HLS, which is part of the centric design flow. The rationale behind generating two distinct IP blocks on FPGA from the rate matching algorithm specified in the 3GPP standards is to minimize latency during the rate matching process and enable parallel data processing to expedite the output generation. The filler bit removal IP plays a vital role in the LDPC block's data processing pipeline by identifying and removing the filler bits in the output, ensuring data integrity. The modified data, along with control signals t_{keep} and t_{last}, is then transmitted to the interleaver IP for further processing. The interleaver block then receives the refined bit stream, to perform further bit selection and bit interleaving operations on the data.

Before delving into the optimized algorithm employed for rate matching, it is crucial to understand the rate matching algorithm specified in the 3GPP standards and how it is optimized, modified, and implemented to construct an IP on an FPGA. To elucidate the functionality of the filler bit removal IP on the FPGA, we refer to Algorithm 1. The process starts with receiving encoded data from the LDPC encoder. The essential configuration parameters for the algorithm are retrieved from the configuration data bus to ensure smooth execution. These parameters aid in calculating burst locations and essential parameters, as specified in Table I, facilitating accurate identification of filler bit positions in the encoded data. The encoded output consists of systematic bits, filler bits, and parity bits arranged sequentially as shown in Fig. 5.

Each 128-bit burst is processed individually once the start and end positions of the filler bits are detected. Unwanted filler bits are removed from the intermediate bursts while preserving the paramount data present in the first and last filler bit bursts (since the whole burst that contains the first and last filler bit location's information can't be removed directly). The systematic bits are then concatenated with the parity bits to create the final output, for the interleaver IP, to process. The output of the filler bit removal IP comprises the data for bit selection and bit interleaving, along with the t_{keep} and t_{last} signals. These signals provide essential information

Algorithm 1: Algorithm: filler bit removal

Input : $cnData[128]$, $inData[128]$
Output: $outData[128]$, $t_{keep}[7]$, $t_{last}[1]$

Extract parameters TBS, C, $K\_$, Bg_no, Z_c from $cnData$;
Calculate N, K, Ncb, $Nref$;
Calculate a_1, a_2, t_1, t_2, a, b, idx (refer Table I);
for $k \leftarrow 1$ **to** C **do**
  for $i \leftarrow 1$ **to** a **do**
    `// outData stream writes: out,`
    `// inData stream reads: in`
    `// reading before filler bits`
    if $i < a_1$ **then**
      out: in(127, 0) $t_{keep} = 127$;
    else if $i < a_1$ *and* $i = a$ **then**
      out: in(127, idx), $t_{keep} = 127 - idx$;
    else if $i = a_1 = a_2$ *and* $t_2 > 0$ **then**
      out: in((127, t_1), ($t_2 - 1$, 0)), $t_{keep} = 127 - t_1 + t_2$;
    else if $i = a_1$ **then**
      out: in(127, idx), $t_{keep} = 127 - idx$;
    `// flush the bursts between` a_1 `and` a_2
    `// reading after filler bits`
    else if $i = a_2$ **then**
      if $a = a_2$ **then**
        out: in($t_2 - 1$, idx), $t_{keep} = t_2 - idx - 1$;
      else
        out: in($t_2 - 1$, 0), $t_{keep} = t_2 - 1$;
    else
      if $i = a$ **then**
        out: in(127, idx), $t_{keep} = 127 - idx$;
      else
        out: in(127, 0) $t_{keep} = 127$;
  end
  Flush out the remaining bursts from $a + 1$ to b.
end

to the interleaver IP. Specifically, t_{keep} indicates which bits within a given burst should be included for processing, while t_{last} determines the last accepted input burst. After the output is received from the filler bit removal IP, it is then sent to the interleaver IP, for further processing. The subsequent steps of the rate matching process can now be subdivided into distinct processes. Initially, interleaver IP performs bit selection. This step involves the selection of specific bits from the remaining data. Then, depending upon the length of the rate matched block (E), and the LDPC encoder output (N) for each segmented code block, we perform a shortening or repetition process over data received in the form of a circular buffer (N_{cb}) for the encoded output. After which, depending on the redundancy version for the circular buffer being retrieved from the configuration, we perform the left circular shift on the entire transport block. Finally, the bit interleaving is performed based on the modulation index, ensuring an optimized arrangement of the bits tailored to the specific modulation scheme. Collectively, these subprocesses contribute to the effective rate matching of the

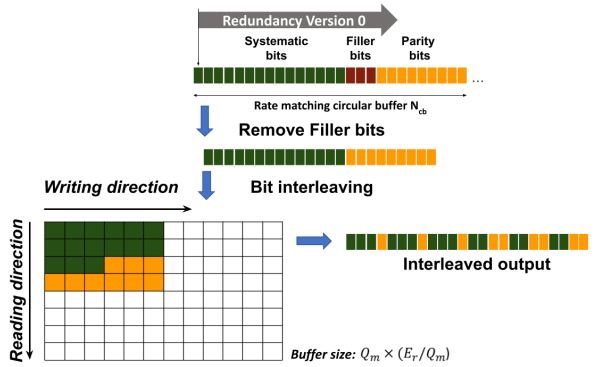

Fig. 5: Rate matching algorithm as per 3GPP standards.

Variable	Definition
a_1, a_2	Burst location having first and last filler bit.
t_1	Bit location before first filler bit in a_1
t_2	Bit location of the last filler bit in a_2
a	Number of Bursts in circular buffer ($N_{cb}/128$)
b	Total number of bursts ($N/128$)
idx	Bit location of N_{cb} in b

TABLE I: Parameters for filler bit removal algorithm

encoded bits within the system.

To improve its performance and reduce latency in FPGA implementation, significant enhancements have been made to the interleaver IP. These enhancements encompass modifications to the algorithm itself and the utilization of available features in vivado HLS. After processing and storing the data, a comprehensive modification was made to the algorithm by introducing a reverse-mapping formula. This formula enabled the reversed execution of the steps of shortening/repetition, left circular shift, and bit interleaving. By utilizing copied arrays, the operation facilitated parallel access in the pipelined architecture, reducing latency. The application of parallelism pragmas in vivado HLS played a crucial role in improving performance. These pragmas allowed for the synthesis of RTL code with a pipelined architecture and overlapping, further reducing latency over iteration intervals. By making multiple independent accesses to different arrays with copied contents within a single iteration, latency in the loop was minimized. The pipelined architecture enabled the overlapping of the four distinct stages of the loop, leading to a reduction in overall latency.

In our FPGA implementation, we conducted processing on up to 152 code blocks for maximum transport block size and 4 transmission layers. To illustrate, we consider an example where the aggregate number of bits slated for transmission (G) equals 45360 bits, and the total count of segmented code blocks (C) is 4. Consequently, the length of each rate matched code block (E) = 11340.

To achieve 4-fold iteration parallelism in the $interleave\_loop$ for interleaving after the filler bit removal, the trip count of the loop was set to $\frac{11340}{4}$, in order to perform 2835 iterations on all the bits to be transmitted, taking 4 at a time. The $interleave\_loop$ had an iteration latency of 72 clock cycles, while the initiation latency was only 1 clock cycle due to optimizations that eliminated sequential bottlenecks. Therefore, the total latency to process

Algorithm 2: Optimized algorithm: rate matching of LDPC output

Input : $inData[128]$, $cnData[128]$, $t_{keep}[7]$, $t_{last}[1]$
Output: $outData[96]$

Extract parameters Rv, TBS, C', $K\_$, Bg_no, Z_c, Q, V, G, Cr from $cnData$;
Calculate N, K, Ncb, $Nref$;
Set $Qm = 2 \times (Q+1)$, $F = K - K\_$;
Set $E_r = (V+1)Q_m \lfloor \frac{G}{(V+1)Q_m C'} \rfloor$;
Calculate index for left circular shift ($k0$) from RV, Ncb, N, Z_c, Bg_No, then adjust it;
Initialize $arr1[200]$, $arr2[200]$, $arr3[200]$, $arr4[200]$;
codeblock_loop: for $k \leftarrow 0$ **to** C **do**
  read_loop: repeat
    Extract data, t_{keep}, and t_{last} from $inData$;
    Extract the req. bits from $inData$ using t_{keep};
    Concatenate those bits to the array elements;
    `// For Instance: Burst b1, b2:` t_{keep}`: 110, 128, is stored like` $arr[0] : b1(0, 109),\ b2(0, 17)$, `128 bits` $arr[1] : b2(18, 127)$ `110 bits`
    Make copies of the original array;
    Add t_{keep} to the keep total kp_{tl} variable;
  until t_{last} *is not triggered*;
  define function $rev\_map(idx, arr)$;
  Return $arr[\lfloor \frac{(\lfloor idx/Qm \rfloor + (idx\%Qm) \times \lfloor E/Qm \rfloor + k0)\%kp\_tl}{128} \rfloor][128 - ((\lfloor idx/Qm \rfloor + (idx\%Qm) \times \lfloor E/Qm \rfloor + k0)\%kp\_tl)\%128]$;
  interleave_loop: for $idx \leftarrow 0$ **to** $E - 1$ **by** 4 **do**
    if $k = 96$ **then**
      Write $temp$ on $outData$, set $k = 0$;
    $temp[95 - k] = rev\_map(idx, arr1)$;
    $temp[95 - k - 1] = rev\_map(idx + 1, arr2)$;
    $temp[95 - k - 2] = rev\_map(idx + 2, arr3)$;
    $temp[95 - k - 3] = rev\_map(idx + 3, arr4)$;
    $k+ = 4$;
  end
  Handle the last code block if necessary;
  Store the final output data and write it to $outData$;
end

11,340 bits was reduced to 2835 (overlapped iterations) $+72$ (initial iteration) -1 (initiation latency), resulting in 2906 clock cycles, which originally was supposed to be $2835 \times 72 = 204120$ clock cycles. This improvement signifies a performance boost of 70 times. This holistic approach significantly improved the efficiency of the algorithm and reduced latency. Independent pipelining was subsequently applied to the remaining steps of the algorithm to optimize performance further.

IV. Implementation on FPGA

A. FPGA Architecture

The overall architecture of the proposed algorithm is shown in Fig. 6. Within the filler bit removal IP, data from the

LDPC encoder is read at a rate of 128 bits per burst. To facilitate the processing of each burst during filler bit removal, a retentive/processing memory with a word width of 128 bits is designed. The major RTL component utilized in the IP core's design is the lookup tables (LUTs) for expression logic within the algorithm. The usage of LUTs can be observed in the expression: $output.data = temp.range(t_2 - 1, index)$; which requires logical right shift operators for synthesis in the RTL. Furthermore, LUTs are utilized in the range read operations within the if-else ladders. Another significant

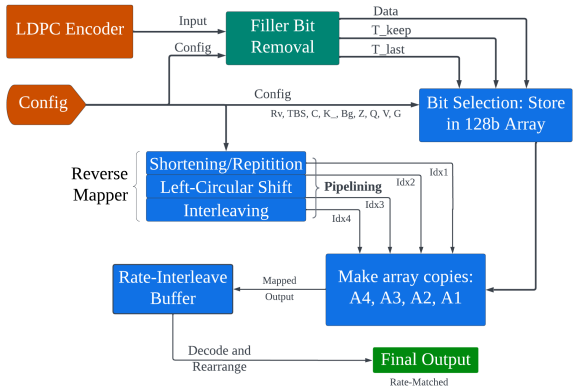

Fig. 6: Overall architecture for the proposed parallel algorithm.

LUT expense is incurred by the multiplexer and the reading of buffer memory through the read_buffer function instance in the original algorithm. Additionally, flip-flops (FFs) are employed to store the parameters retrieved from the config data and utilized in the algorithm. Pipelining is employed in the FPGA implementation within the for-loops.

When applying pipelining to loop iterations using vivado HLS, the loop is divided into multiple stages, allowing concurrent processing of iterations. Vivado HLS identifies these stages during synthesis, resulting in a hardware implementation with multiple pipeline stages. Similarly, in the implementation of the interleaver IP on the FPGA, the $interleave\_loop$ is divided into different pipeline stages, enabling parallel and overlapping execution of iterations.

Resource	***Utilization***	***Available***	***Utilization %***
LUT	30921	425280	7.12
FF	32085	850560	3.77
BRAM_18K	16	2160	0.74
DSP48E	18	4272	0.42
URAM	0	80	0

TABLE II: Resource utilization of interleaver IP.

The interleaver IP heavily relies on LUTs and FFs as the major RTL components. These components are essential due to the intricate calculations performed during the instantiation of the reverse mapper function, as mentioned in Algorithm 2, which is a critical aspect of the rate matching process. Our optimized approach implements 4-way parallelism, significantly reducing latency while increasing resource utilization. Experimenting with 8-way parallel optimization further reduced latency but led to a substantial increase in resource utilization, with FFs reaching 7% and LUTs reaching 14%.

Resource	***Utilization***	***Available***	***Utilization %***
LUT	7945	425280	1.86
FF	1092	850560	0.13
BRAM_18K	0	2160	0
DSP48E	0	4272	0
URAM	0	80	0

TABLE III: Resource utilization for filler bit removal IP

IP (with specifications)	**Latency** (min - max)
Filler bit removal	0.66us - 0.37ms
Interleaver (original)	1.91us - 30.0ms
Interleaver (optimized)	2.58us - 0.97ms

TABLE IV: Latency report of filler bit removal IP and interleaver IP operating at a clock frequency of 100 MHz

B. Hardware Implementation and Simulation

Fig. 7 shows the hardware implementation of the rate matching. The proposed algorithm is implemented in RTL using verilog on the xilinx zynq ultraScale ZCU111 (xczu28dr-ffvg1517-2-e) evaluation board. To evaluate the algorithm's performance, an end-to-end simulation is conducted for all possible configurations. Data generators are used to provide input and configuration data directly to the filler bit removal IP, which is connected to the interleaver IP. The generated output is compared with the output generated by the 5G NR toolbox in MATLAB for correctness. One specific configuration is considered for evaluation, with a transport block size of 848 bits and a target rate matched output length of **7168** bits. The redundancy version is set to 0, the number of code blocks is 1, the modulation index is 2, lifting size Z_c is 88, and the base graph 2 is used. The clock frequency is set to 100MHz with an uncertainty of 12.5%. Using this configuration, the output is generated on the evaluation board and verified. The total latency for this configuration is measured to be 23.0421 microseconds. The simulated results can be observed in Fig. 8.

C. Resource Utilization and Latency

Table II presents the resource utilization report for the interleaver IP. The moderate utilization of LUTs and FFs can be observed, although there is room for improvement. The relatively high utilization is primarily attributed to the parallel computing approach employed in the reverse mapper loop within the IP. Table III presents the resource utilization of the filler bit removal IP, which is comparatively low since the primary objective of this IP is to remove filler bits. Both IPs were evaluated on the xilinx zynq ultraScale ZCU111 evaluation board operating at a clock frequency of 100MHz.

Table IV presents the latency report for the filler bit removal IP and provides a comparison between the original and optimized versions of the interleaver IP. The latency has been significantly improved, reducing from 30 milliseconds to 900 microseconds for processing the maximum transport block size which is 12,77,992 bits, achieving a latency optimization of approximately $\frac{1}{33}$. However, this optimization comes at the cost of increased FPGA resource utilization. Therefore, it is crucial to find an optimal trade-off for the selection of M in M-paralleled pipelining to achieve the desired optimization

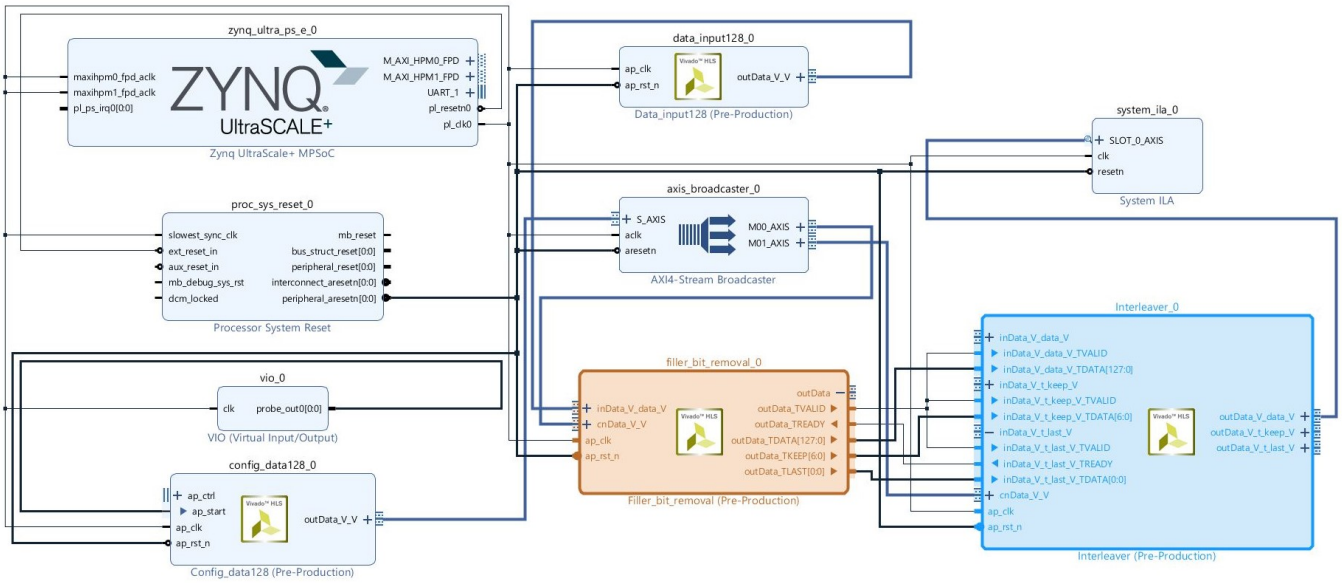

Fig. 7: Hardware implementation of the rate matching

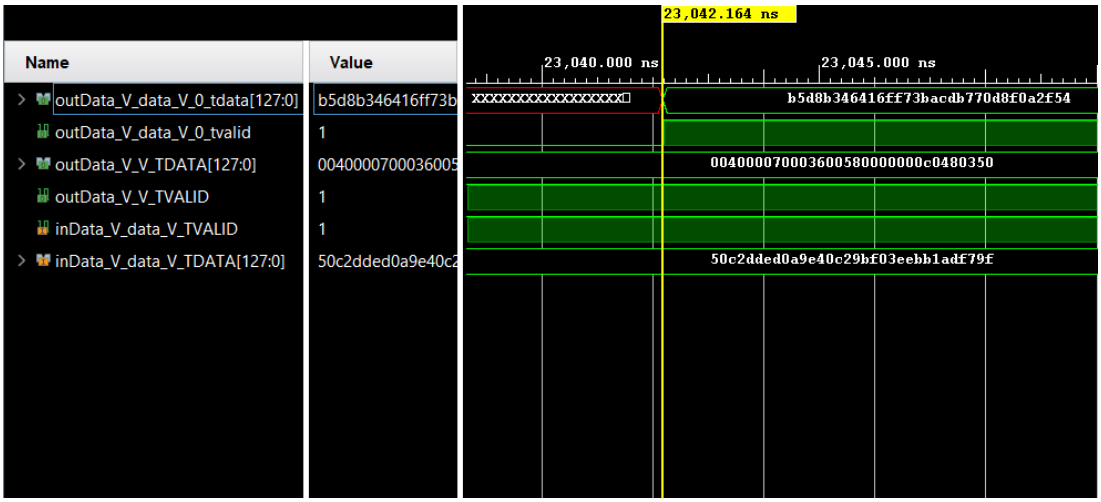

Fig. 8: Waveform of the simulated result.

(in our the optimal latency and resource utilization were found for M = 4).

V. Conclusion

In this paper, we presented FPGA implementations of bit selection and bit interleaving in rate matching for 5G NR, aiming to optimize latency and resource utilization. Our optimizations resulted in significant latency improvements, with the interleaver IP achieving a $\frac{1}{33}$ times latency reduction. We explored 4-way and 8-way parallelism to balance resource usage and latency, with the former offering a four-fold increase in resource utilization. The resource utilization of the filler bit removal IP remained low due to its specific function. Our findings contribute to efficient 5G NR implementations, enhancing data processing and transmission. Future work should focus on further resource optimization and exploring additional parallelism techniques to meet evolving communication standards.

References

[1] J. T. Cheng, A. Nimbalker, Y. W. Blankenship, B. K. Classon, and K. T. Blankenship, "Analysis of circular buffer rate matching for LTE turbo code," in *Proceedings of the 68th IEEE Vehicular Technology Conference, VTC Fall 2008, 21-24 September 2008, Calgary, Alberta, Canada*. IEEE, 2008, pp. 1–5.

[2] F. Hamidi-Sepehr, A. Nimbalker, and G. Ermolaev, "Analysis of 5g LDPC codes rate-matching design," in *87th IEEE Vehicular Technology Conference, VTC Spring 2018, Porto, Portugal, June 3-6, 2018*. IEEE, 2018, pp. 1–5.

[3] C. Ma and P. Lin, "Efficient implementation of rate matching for lte turbo codes," in *2010 2nd International Conference on Future Computer and Communication*, vol. 1, 2010, pp. V1–704–V1–708.

[4] H. Wu and H. Wang, "Decoding latency of LDPC codes in 5g NR," in *29th International Telecommunication Networks and Applications Conference, ITNAC 2019, Auckland, New Zealand, November 27-29, 2019*. IEEE, 2019, pp. 1–5.

[5] 3GPP, "5G;NR;Multiplexing and channel coding," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 38.212, 7 2018, version 15.2.0.

[6] H. Wu and H. Wang, "Decoding latency of ldpc codes in 5g nr," in *2019 29th International Telecommunication Networks and Applications Conference (ITNAC)*, 2019, pp. 1–5.

[7] T. Richardson and S. Kudekar, "Design of low-density parity check codes for 5g new radio," *IEEE Communications Magazine*, vol. 56, no. 3, pp. 28–34, 2018.

[8] 3GPP, "5G;NR;Physical layer procedures for data," 3rd Generation Partnership Project (3GPP), Technical Specification (TS) 38.214, 10 2018, version 15.3.0.