Hardware Implementation of Single-carrier Time-domain Turbo Equalization for Severe Multipath Acoustic Communication Channels

Jinfeng Li and Y. Rosa Zheng
Dept. of ECE, Lehigh University, Bethlehem, PA 18015-3084, USA.

Global Oceans 2020: Singapore - U.S. Gulf Coast | 978-1-7281-5446-6/20/$31.00 ©2020 IEEE | DOI: 10.1109/IEEECONF38699.2020.9389170

***Abstract*—This paper proposes a low-latency FPGA implementation for Turbo equalization to combat very long multipath fading channels where the Intersymbol-interference (ISI) channel length is on the order of 100 taps. Turbo equalization is essential for such severe multipath channels, but exhibits very large latency and high computational complexity due to its sequential and iterative data processing on large-scale matrix arithmetic. This paper proposes an FPGA acceleration architecture to exploit the Hermitian symmetric property of the channel Gram matrix and convolutional nature of Sequential Interference Cancellation (SIC), and successfully implements a linear Turbo equalizer of 100 taps on a Xilinx Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit. The architecture is able to support two turbo iterations for a 1024-symbol block size and achieve 200 kilo-symbols-per-second (ksps) transmission rate.**

I. Introduction

Turbo equalization is essential for single-carrier modulation (SCM) receivers in severe multipath wireless channels where channel length L is often on the order of 100 taps or strong intersymbol interference is spread over more than 50 symbols. Such severe multipath channels are often encountered, for example, in underwater acoustic (UWA) wireless communications [1], multi-drone base-station wireless networks [2] and multiple base-station digital broadcast systems [3], etc. The common approach to combat the severe ISI channels is to avoid large-size equalizers in the receiver by either shortening the effective channel with time reversal or utilising transform-domain transmission schemes, such as the Orthogonal Frequency-Division Multiplexing (OFDM) [4], Filter Bank Multi-Carrier (FBMC), or Non-Orthogonal Multiple Access (NOMA) [5] schemes. These approaches significantly reduce the computational complexity, but suffer from performance bottleneck in terms of high bit error floor, high peak-to-average ratio, slow to track Doppler spread, etc.

Single-carrier modulation with Turbo equalization improves robustness and reliability of UWA communications, especially in severe Doppler and multipath channels. Many advanced Turbo equalization algorithms are well proved by field experiments with post-experiment data processing [1]. However, real-time hardware implementation is rather difficult due to the constraints in hardware resources and latency requirements.

Most of the turbo-equalization works have focused on the MIMO-OFDM system, such as [6]–[10], where the related matrix size is typically 4×4 only. The existing works in SCM turbo equalization include [11]–[13] where the maximum channel length is less than 20 taps or frequency-domain equalization is used to reduce the equalizer length. For example, [11] implemented a single-carrier turbo equalization on the TMS320C5509, but the achieved maximum data rate is 207 Kbit/s per iteration for QPSK modulation with only 16-taps channel. The work in [13] designed a single carrier-Frequency Domain Equalization (FDE) system on FPGA, which converted the equalization in FD with only one channel tap. The work in [12] also implemented a single carrier frequency domain turbo equalization for underwater acoustic communication but on DSP, the effect data rate in the experiments is only 770 bit/s.

The computational complexity mainly comes from matrix arithmetic such as matrix inversion and multiplication in calculating equalizer coefficients and sequential interference cancellation (SIC). When the channel length is on the order of a hundred symbols, the sizes of the equalizer matrices would be as large as 200×200. However, most of the existing implementations of matrix arithmetic only go to half of the required sizes. Recently, high-level synthesis (HLS) has been offered by Xilinx [14] or Intel [15] for large-scale matrix inversion. Although the solutions based on entire column vector access can reduce an $\mathcal{O}(N^3)$ process to $\mathcal{O}(N^2)$ steps with N being the size of the matrix, the latency is too high to meet the requirement of our application.

In this paper, we propose an FPGA acceleration architecture to implement turbo equalization in severe multipath wireless channels with low latency as well as relatively small area. The main acceleration mechanism to achieve low latency is to keep all the matrix arithmetic on the row-wise computation in both SIC and adaptive equalizer design. In the SIC part, we propose a low-latency computational algorithm by exploiting the convolution of shifted data blocks. For a block length of N_{blk} symbols, SIC is completed within around $2N_{blk} + L$ clock cycles, which is much lower than the $N_{blk} \times L$ by direct matrix multiplication. For the equalizer design, we leverage the Hermitian symmetric property of the matrix to be inverted, utilize the LDL^H-decomposition similar to [16] but with iterative manner [14] for matrix inversion. We propose a block RAMs storage strategy for intermediate matrices to eliminate the switching time between the column-wise access and row-wise access needed in different inversion steps. Benefiting from the proposed storage strategy, we also propose a row-wise computation architecture based on Hadamard product for the multiplication of an upper triangular matrix and a diagonal matrix, which reduces the latency from $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$.

The architecture is successfully implemented for a 151-tap linear equalizer on a Xilinx Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit, and is able to support two Turbo iterations for a 1024-symbol block size and achieve 200 kilo-symbols-per-second (ksps) transmission rate.

II. Preliminary

Consider a single carrier modulation (SCM) communication system as shown in Fig. 1. At the transmitter side, the information bits b_i are first encoded and interleaved. The interleaved coded bits $c_{k,q}$ are grouped into Q bits per set and are mapped into *M-ary* symbols x_k, where $Q = \log_2 M$. The complex symbols are then up-sampled and passed through the pulsing shaping filter. The signal is directly modulated to the single carrier frequency and then transmitted to the multipath channel.

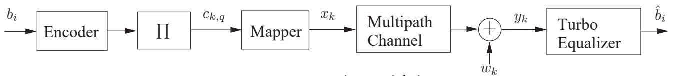

Fig. 1. A Communication System Employing Turbo Equalization.

Assuming that the channel impulse response (CIR) coefficients h_l and the channel length L are known by channel estimation, the received symbol at time instant k after carrier demodulation and symbol synchronization is written as

$$y_k = \sum_{l=0}^{L-1} h_l x_{k-l} + w_k \tag{1}$$

where x_{k-l} is the transmitted symbol at time instant $k-l$, and w_k is the sampled noise at time instant k, which is modeled as adaptive white Gaussian noise (AWGN) with zero mean and variance σ_w^2.

The structure of the linear minimum mean square error (LMMSE) turbo equalizer is depicted in Figure 2. It consists of a soft-input soft-output (SISO) LMMSE equalizer, a SIC, soft symbol demapper and mapper, de-interleaver and iterleaver, and a maximum *a posteriori* probability (MAP) decoder. The covariance estimator is used to design the adaptive equalizer coefficients $\mathbf{g}$. For low-complexity implementation, we assume the covariance keeps the same for every symbol in a block, $\mathbf{g}$ is updated once at the start of each iteration of a block.

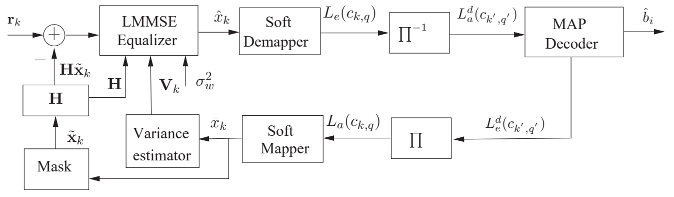

Fig. 2. Block diagram of the LMMSE turbo receiver.

The equalizer output $\hat{x}_k$ is mapped to the bit extrinsic LLRs $L_e(c_{k_q})$ which is considered as the *a priori* LLR for the MAP decoder after deinterleaver, denoted as $L_a^d(c_{k',q'})$. The MAP decoder computes the *a posteriori* LLR and outputs the extrinsic LLR $L_e^d(c_{k',q'})$. Its interleaved version becomes the *a priori* LLR of the equalizer for the next iteration. To close the loop, the SIC unit consists of a mask function that zeroes out the kth symbol from the soft symbol vector $\bar{x}_k$ and the resulting symbol vector $\tilde{x}_k$ passes through the channel to reconstruct the Inter-Symbol Interference (ISI). The received signal removes the reconstructed ISIs before being fed to the adaptive equalizer.

Let the linear equalizer have K_1 post-cursor taps and precursor K_2 taps. Stacking $N = K_1 + K_2 + 1$ received symbols into vectors, equation (1) can be rewritten in the matrix form

$$\mathbf{r}_k = \mathbf{H}\mathbf{x}_k + \mathbf{w}_k \tag{2}$$

where

$$\begin{aligned}
\mathbf{r}_k &= [y_{k-K_2}, y_{k-K_2+1}, \cdots, y_{k+K_1}]^T \\
\mathbf{x}_k &= [x_{k-K_2-L+1}, x_{k-K_2-L+2}, \cdots, x_{k+K_1}]^T \\
\mathbf{w}_k &= [w_{k-K_2}, w_{k-K_2+1}, \cdots, w_{k+K_1}]^T \\
\mathbf{H} &= \begin{bmatrix} h_{L-1} & \cdots & h_0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & h_{L-1} & \cdots & h_0 \end{bmatrix}
\end{aligned} \tag{3}$$

and $\mathbf{H} \in \mathcal{C}^{(K_1+K_2+1)\times(K_1+K_2+L)}$.

The LMMSE equalizer estimates the transmitted symbols one by one via

$$\hat{x}_k = \mathbf{g}^H(\mathbf{r}_k - \mathbf{H}\tilde{\mathbf{x}}_k) \tag{4}$$

with

$$\begin{aligned}
\tilde{\mathbf{x}}_k &= [\bar{x}_{k-K_2-L+1}, \ldots, \bar{x}_{k-1}, 0, \bar{x}_{k+1}, \ldots, \bar{x}_{k+K_1}] \\
\mathbf{g} &= (\sigma_w^2 \mathbf{I}_N + \mathbf{H}\mathbf{V}\mathbf{H}^H)^{-1}\mathbf{s} \\
\mathbf{s} &\triangleq \mathbf{H}[\mathbf{0}_{1\times(K_2+L-1)}, 1, \mathbf{0}_{1\times K_1}]
\end{aligned} \tag{5}$$

where $\bar{x}_k$ is the soft mapper output, $\mathbf{V} \in \mathcal{C}^{(K_1+K_2+L)\times(K_1+K_2+L)}$ is a diagonal matrix of the mean of the variance of soft mapper $\bar{x}_k$ except the (K_2+L)th diagonal element is 1 [17].

Based on (5), the computation of $\mathbf{g}$ for a block of symbols in each iteration involves large-scale matrix multiplication and matrix inversion, as $K_1 + K_2 + L$ is often on the order of 200. The equalization (4) for each symbol in the block needs the calculation of a matrix and a vector. These causes high computational complexity and latency in the real-time implementation of the turbo equalization. Moreover, the latency will increase linearly with the times of turbo iteration, as in one turbo iteration, one part can only start when the previous part is complete.

III. FPGA Implementation

In contrast to software-programmable solutions by CPUs and GPUs where the latency is limited by the data transfer paths of a load-store architecture, FPGA is a hardware-programmable solution with high degree of parallelism as well as flexible and configurable data movements, thus is promising to support high-performance matrix arithmetic. In this paper, we employ the row-wise parallelism, which means that all the elements in a row of the matrix are involved in the computation at the same time.

The proposed accelerated turbo equalization architecture is depicted in Figure 3. Received vector $\mathbf{r}$ is stored in a RAM block. CIR $\mathbf{h}$ is got from channel estimation. Vector Processing (VP) unit consists of a group of $\mathbf{N}$ complex multiplier units followed by a $\log_2(\mathbf{N})$-stage adder tree. Row-wise operation (vector path, bold lines in the figure) is in the matrix inversion, SIC, and the computing to generate matrix $\mathbf{A} = \sigma_w^2 \mathbf{I}_K + \mathbf{HVH}^H$. $\mathbf{A}$ is stored in a group of RAM blocks. Each pair of RAM block stores a column of $\mathbf{A}$ including the real part and the imaginary part. The estimated symbols are de-interleaved through a RAM block whose *write* and *read* address are pre-stored in a ROM. The same case is valid for interleaving the output of the MAP decoder. After *bits2symbol* block which is implemented by storing the *tanh* function in a ROM, the extrinsic information of received symbols got in one turbo iteration are fed back into SIC part for the next iteration.

A. SIC

SIC is used to avoid the instability caused by positive feedback during the iterative operation, which means, the *a priori* information of symbol x_k should not be used during the iterative detection of x_k itself.

Since $\mathbf{H}$ is a matrix, the calculation of $\mathbf{H}\tilde{\mathbf{x}}_k$ in (4) for each symbol needs $K_1 + K_2 + 1$ times even if we parallel the computation in row-wise. For a block of N_{blk} symbols, the needed clock cycles would be $N_{blk} \times (K_1 + K_2 + 1 + N_{Astage})$, where $N_{Astage} = \lceil \log_2(L) \rceil$ as only L non-zero elements are in a row.

However, as $\tilde{\mathbf{x}}_{k+1}$ is left shifted from $\tilde{\mathbf{x}}_k$ like in (6), they only have four different elements and the resulted interference are like in (7). Based on this, we make two simplifications and propose a low-latency computing algorithm.

$$\begin{aligned} \tilde{\mathbf{x}}_k &= [\bar{x}_{k-K_2-L+1}, \ldots, \bar{x}_{k-1}, 0, \bar{x}_{k+1}, \ldots, \bar{x}_{k+K_1}] \\ \tilde{\mathbf{x}}_{k+1} &= [\bar{x}_{k-K_2-L+2}, \ldots, \bar{x}_{k}, 0, \bar{x}_{k+2}, \ldots, \bar{x}_{k+K_1+1}] \end{aligned} \tag{6}$$

$$\begin{aligned} \mathbf{H}\tilde{\mathbf{x}}_k &= [a_{0,k}, \ldots, a_{K_2-1,k}, a_{K_2,k}, a_{K_2+1,k}, \ldots, a_{K_2+K_1,k}] \\ \mathbf{H}\tilde{\mathbf{x}}_{k+1} &= [a_{1,k}, \ldots, a_{K_2,k}, a_{K_2,k+1}, a_{K_2+1,k+1}, \ldots, a_{K_2+K_1,k+1}] \end{aligned} \tag{7}$$

1) Simplification 1: The first K_2 elements of $\mathbf{H}\tilde{\mathbf{x}}_{k+1}$ have been already calculated in $\mathbf{H}\tilde{\mathbf{x}}_k$. Therefore, only $K_1 + 1$ terms need to be recalculated for the $(k+1)_{th}$ symbol. One term is because the new element shifted in $\tilde{\mathbf{x}}_{k+1}$. The other K_1 terms are due to the different term that set to be zero in $\tilde{\mathbf{x}}_{k+1}$. The needed clock cycles are reduced to $N_{blk} \times (K_1 + 1 + N_{Astage})$.

2) Simplification 2: Modify (5) as

$$\tilde{\mathbf{x}_k}' = [\bar{x}_{k-K_2-L+1} \ldots \bar{x}_{k-1} \quad \bar{x}_k \quad \bar{x}_{k+1} \ldots \bar{x}_{k+K_1}] \tag{8}$$

so that $\mathbf{H}\tilde{\mathbf{x}}_k$ is obtained by

$$\mathbf{H}\tilde{\mathbf{x}}_k = \mathbf{H}\tilde{\mathbf{x}_k}' - \mathbf{h}\bar{x}_k. \tag{9}$$

The resulted vectors $\mathbf{H}\tilde{\mathbf{x}_k}'$ for all the symbols in the block will be first calculated at the start of SIC by the convolution of $\mathbf{h} \circledast \bar{\mathbf{x}}$, which needs about $N_{blk} + L + N_{Astage}$ clock cycles. The multiplication of $\mathbf{h}\bar{x}_k$ only needs one clock cycle in row-wise parallel. By fully taking the advantage of parallel and pipelining, the clock cycles needed for the SIC of N_{blk}-symbol block are reduced to $N_{blk} + L + N_{Astage} + N_{blk}$.

Algorithm 1 Proposed SIC algorithm

```
1:  Function SIC ($\mathbf{r}, \mathbf{h}, \bar{\mathbf{x}}, \mathbf{g}$);
    Input: Received symbols vector $\mathbf{r}$, CIR $\mathbf{h}$, soft mapper
    output vector $\bar{\mathbf{x}}$, equalizer coefficients $\mathbf{g}$
    Output: Estimated soft symbols $\hat{\mathbf{x}}$
2:  Initialize vectors $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}$
3:  for $i = 0 : N_{blk} + L - 1$ do /*Stage 0*/
4:      $a_i = \mathbf{h} \cdot \bar{\mathbf{x}}_i$ /*via the vector processing unit*/
5:      $\bar{\mathbf{x}}_{i+1} \longleftarrow \bar{\mathbf{x}}_i$
6:  end for
7:  Initialize vectors $\mathbf{a}_j, \mathbf{z}_j$
8:  for $j = 0 : N_{blk} - 1$ do /*Stage 1*/
9:      $\mathbf{b}_j = \bar{x}_j \times \mathbf{h}$
10:     $\mathbf{c}_j = \mathbf{a}_j - \mathbf{b}_j$ /*via the adders*/
11:     $\mathbf{d}_j = \mathbf{r}_j - \mathbf{c}_j$
12:     $\hat{\mathbf{x}}_j = \mathbf{g} \cdot \mathbf{d}_j$ /*Equalization*/
13:     $\mathbf{a}_{j+1} \longleftarrow \mathbf{a}_j$;
14:     $\mathbf{r}_{j+1} \longleftarrow \mathbf{r}_j$;
15:     $\bar{x}_{j+1} \longleftarrow \bar{x}_j$
16: end for
```

3) Proposed SIC Algorithm: The proposed algorithm has two stages. The first stage (lines 2-6) calculates the convolutional products vector $\mathbf{a}$ of CIR $\mathbf{h}$ and the soft mapper output vector $\bar{\mathbf{x}}$. In each loop iteration, $\bar{\mathbf{x}}_i$ shifts in a new $\bar{x}$, and shifts out the oldest one. The second stage (lines 7-15) calculates SIC results for all the symbols in a block. The product vector $\mathbf{b}_j$ of soft mapper output $\bar{x}_j$ and CIR $\mathbf{h}$ is first calculated. It is removed from the convolutional products vector $\mathbf{a}_j$ corresponding to j_{th} symbol that was calculated in the first stage. Subsequently, the reconstructed interference is removed from the received symbols vector $\mathbf{z}_j$ (line 11). At last, the $\mathbf{a}_{j+1}, \mathbf{r}_{j+1}, \bar{x}_{j+1}$ for the next loop iteration are updated by shifting one new element in and shifting out the oldest element, respectively.

4) Architecture: Figure 4 shows the architecture of the proposed SIC algorithm. Vectors $\bar{\mathbf{x}}$ and $\mathbf{r}$ are stored in two RAM blocks separately. CIR $\mathbf{h}$ is always latched as one input of the complex multipliers. In Stage 0, the other inputs of the complex multipliers are the vector $\tilde{\mathbf{x}}'$ shifted by one $\bar{x}$ every clock cycle. The products feed into the multi-stages adder tree

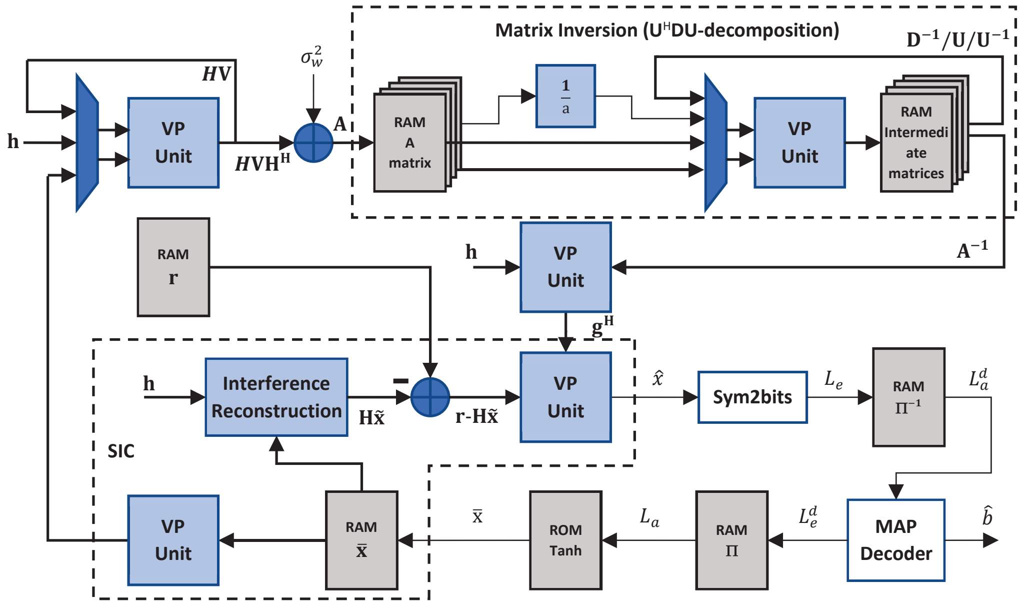

Fig. 3. Proposed accelerated turbo equalization architecture.

to calculate the convolutional results a, which are then written into a RAM block. This process is pipelined until the last soft mapper output goes through all the CIR taps.

At the start of Stage 1, RAM block storing $\bar{\mathbf{x}}$ is reread from the beginning address and one element is latched at the other sides of the complex multipliers. The products are directly sent to a group of adders whose one side is a vector shifted by one element read from RAM block storing $\mathbf{a}$ every clock cycle. The difference vector is latched as the inputs of another group of adders whose one side is a vector shifted by one element read from RAM block storing $\mathbf{z}$ every clock cycle. The final results are latched as the input of complex multipliers for the following equalization to calculate $\hat{x}$.

5) Performance Analysis: By exploiting the convolution of shifted data blocks in the SIC operation, the clock cycles needed for a N_{blk}-symbol block are reduced to the order of $2N_{blk}+L+N_{Astage}$ compared with directly matrix arithmetic. For a typical case in underwater acoustic communication where $L = K_1 = 100, K_2 = 50, N_{blk} = 1024, N_{Astate} = 8$, the clock cycles are reduced from more than 100000 to around 2100 without considering additional clock cycles needed in the implementation.

B. Matrix Inversion

From equation (3), we found that matrix $\mathbf{A}$ is an $N \times N$ Hermitian symmetric matrix, and the diagonal elements would be never zero. Therefore, there is possible a group of a lower-triangle matrix $\mathbf{L}$ and a diagonal matrix $\mathbf{D}$ such that $\mathbf{A} = \mathbf{LDL}^H$. The inversion of $\mathbf{A}$ is obtained by $\mathbf{A}^{-1} = (\mathbf{L}^{-1})^{\mathbf{H}}\mathbf{D}^{-1}\mathbf{L}^{-1}$ or $\mathbf{U}^{-1}\mathbf{D}^{-1}(\mathbf{U}^{-1})^{\mathbf{H}}$ when we use an upper-triangle matrix $\mathbf{U}$.

Normally, to guarantee numerical stability, a row permutation matrix $\mathbf{P}$ is introduced (pivoting) to avoid overflow in

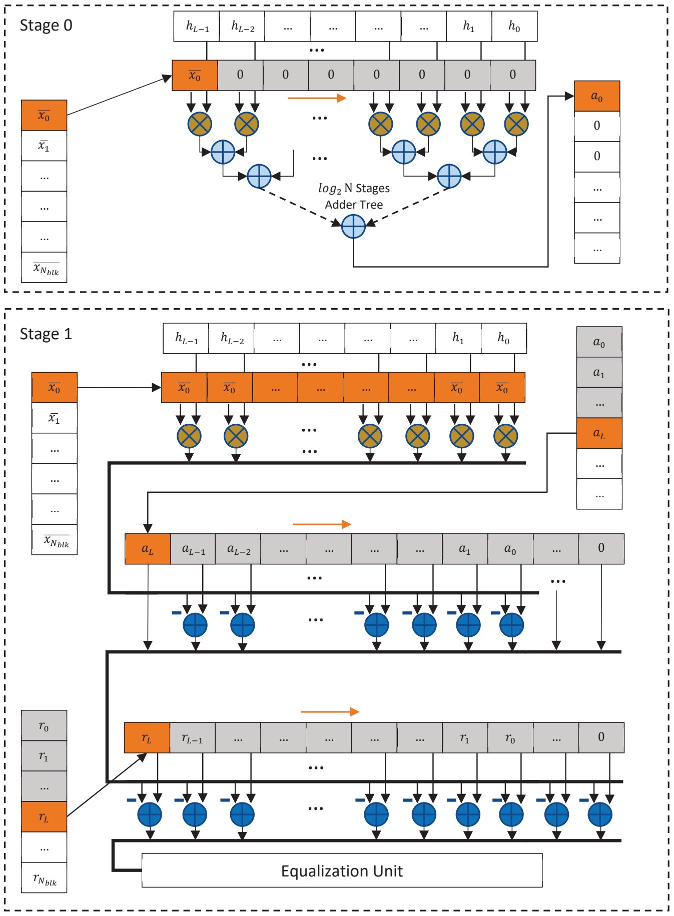

Fig. 4. Proposed SIC Algorithm Architecture

the reciprocal operation. However, as the diagonal elements in $\mathbf{A}$ are real numbers and much larger than other elements in the same column, displacement matrix $\mathbf{P}$ is not needed in this implementation.

For the conjugate multiplication, we construct a complex multiplier unit similar as [18], that consists of four multipliers and two adders. The add/sub function of both adders is enabled. Figure 5 shows how this unit calculates the multiplication with a complex number or its conjugate.

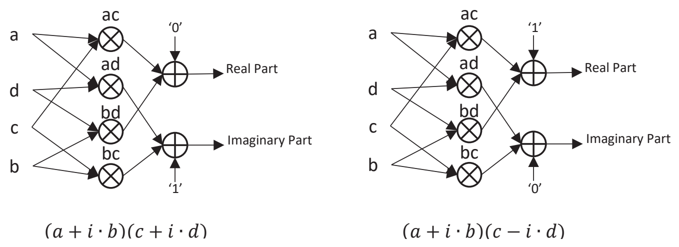

Fig. 5. Complex multiplier unit.

LDL^H-based inversion algorithm has three steps: decomposition, inversion, and multiplication. This paper adopts the iterative computation manner and pipeline architecture [14] in all these three steps.

Since in the original inversion process, data access needs to be switched between row-wise and column-wire in different inversion steps, we propose a block RAMs storage strategy to solve this problem so that all data accesses are row-wise, by which the data copy time is eliminated. Besides, we propose a row-wise computation architecture based on Hadamard product for the multiplication of an upper triangular matrix and a diagonal matrix, which reduces the latency from $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$.

1) Proposed Matrices storage strategy: Figure 6 shows the proposed storage strategy for $\mathbf{D}^{-1}$, upper triangular matrix $\mathbf{U}$ and its inversion matrix $\mathbf{U}^{-1}$. All the diagonal elements of $\mathbf{D}^{-1}$ are stored at the address zero of all the RAM blocks. $\mathbf{U}$ is stored as its transpose $\mathbf{U}^T$. Therefore, in the decomposition step, the elements are stored from RAM block 0 to block $N-1$, in each block the address is started from $n+1$ (n is the row number) like what the black arrow is shown in the Figure 6(a).

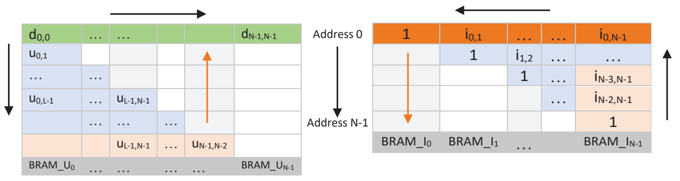

Fig. 6. Proposed matrix storage strategy for (a) $\mathbf{D}^{-1}$ and $\mathbf{U}$. (b) Inversion matrix $\mathbf{U}^{-1}$.

In this way, we keep the row-wise operation in the following inversion step where the matrix needs to be read in column-wise. As the orange arrow is shown in Figure 6(a), the same address of all RAM blocks is read simultaneously to fetch one column of $\mathbf{U}$, from bottom to top. The resulted inversion elements are stored in the reverse way, which is from RAM block $N-1$ to block 0, in each block, the address is decreased from $N-1$ to 0, as what the black arrow is shown in the Figure 6(b).

Algorithm 2 Proposed algorithm for $\mathbf{U}^{-1}\mathbf{D}^{-1}$

```
1:  Function Matrix Inversion ($\mathbf{A}$);
    Input: Matrix $\mathbf{A}$ (conjugate symmetric)
    Output: Inversion Matrix $\mathbf{W}$
2:  LU decomposition to get matrix $\mathbf{U}$ and vector $\mathbf{dinv}$
3:  Inverse $\mathbf{U}$ based on proposed storage strategy to get matrix $\mathbf{U}^{-1}$
4:  Initialize matrix $\mathbf{M}$
5:  Initialize row vector $\mathbf{i} \in \mathbf{U}^{-1}, \mathbf{m} \in \mathbf{M}$
6:  for j=0:N-1 do
7:      $\mathbf{m}_j = \mathbf{dinv} \circ \mathbf{i}_j$ /*Hadamard product*/
8:      $\mathbf{i}_{j+1} \longleftarrow \mathbf{i}_j$;
9:  end for
10: $\mathbf{W} = \mathbf{M} \times \mathbf{I}^H$ /*Iterative manner*/
```

2) Proposed multiplication algorithm for upper-triangle matrix and diagonal matrix: The proposed algorithm for the multiplication of $\mathbf{U}^{-1}\mathbf{D}^{-1}$ is shown in Algorithm 2. As we store all the diagonal elements of $\mathbf{D}^{-1}$ at the same address 0 of all the RAM blocks, we fetch them as a vector $\mathbf{dinv}$ at the same clock and feed them into the complex multipliers. The loop (line 6-9) calculates the Hadamard product of each row of $\mathbf{U}^{-1}$ and $\mathbf{dinv}$ until the last row as what the orange arrow is shown in Figure 6(b). The architecture of the proposed row-wise computation of $\mathbf{U}^{-1}\mathbf{D}^{-1}$ is depicted in Figure 7.

The last matrix multiplication is based on iterative update approach, and $\mathbf{L}^{-1}$ needs to be read in column-wise. As we discussed, $\mathbf{L}^{-1} = (\mathbf{U}^{-1})^{\mathbf{H}}$. We read the same address of all the RAM blocks to fetch a row of $\mathbf{U}^{-1}$ to multiply with rows of $\mathbf{U}^{-1}\mathbf{D}^{-1}$ as the column of $\mathbf{L}^{-1}$. Conjugate operation is completed by controlling the addition or subtraction function of the complex multipliers in Figure 5.

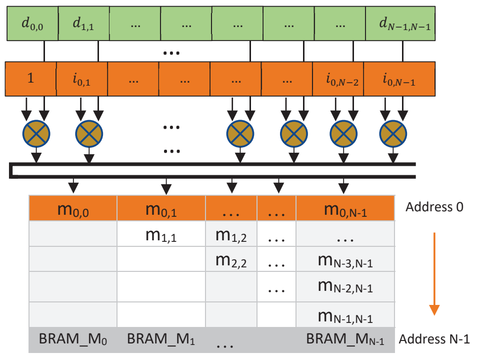

Fig. 7. Proposed row-wise computation of $\mathbf{U}^{-1}\mathbf{D}^{-1}$.

3) Performance Analysis: Beyond the reduction from $\mathcal{O}(N^3)$ process to $\mathcal{O}(N^2)$ steps, we eliminate the row and column alternating accesses time and reduce $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$

steps in the multiplication of the upper triangular matrix and a diagonal matrix. Though we benefit from the Hermitian symmetric characteristic in this implementation, the proposed matrices storage strategy and row-wise computation architecture for $\mathbf{U}^{-1}\mathbf{D}^{-1}$ can be used in the general matrix inversion based on LU decomposition.

C. MAP decoder and other components

1) Interleaver and De-interleaver: In this implementation, we use MATLAB code to generate the random interleave and de-interleave sequences and store them in the ROM. One entry of the ROM consists of two parts. The high part is the address for interleaving, the low part is the address for de-interleaving. Normally, the interleaver size in UAC is like one thousand, the ROM resource needed is limited.

2) MAP decoder: The architecture for MAP decoder can refer to [19]. However, in this paper, besides output the LLR of original information bit, we modified the MAX log-MAP decoder to generate the extrinsic information of each encoded bit as the *a priori* information feedback to the equalizer after bit-to-symbol LLR mapping.

3) Sym2bits blcok: Since the input of the MAP decoder is the bit LLR value, Sym2bits block maps the soft symbols output from equalizer to bit LLR values. The implementation can refer to [20].

4) Hyperbolic Tangent Sigmoid Function: Bit-to-symbol soft extrinsic information approximated mapping can refer to [21].

The main challenge here is the implementation of the hyperbolic tangent function. Piece-wise linear (PWL) approximation and lookup tables are two popular adopted solutions. Due to the usage of multipliers in its design, PWL approximation usually requires several clock cycles and larger areas. However, the lookup table solution offers high operation speed at a little cost of ROM resources. In this implementation, we approximate the hyperbolic tangent function with R points that are uniformly distributed across the entire input range -4 to 4 where $R = 2^p$, p is the number of bits to represent the bit LLR value L_a. The read address t of the ROM is given by

$$t = \begin{cases} T_{pos}, & \text{if } L_a \geq 4 \\ L_a + 2^{p-1}, & \text{if } -4 < L_a < 4 \\ T_{neg}, & \text{if } L_a \leq -4 \end{cases} \tag{10}$$

where T_{pos} is the address storing 1 and T_{neg} is the address storing -1.

IV. Synthesis and Performance Results

We implemented the proposed accelerator on Xilinx Zynq UltraScale+ MPSoC ZCU102 Evaluation board which has an Ultrascale+ FPGA (XCZU9EG-2FFVB1156, 274,080 LUTs, 548,160 FFs, 912 36Kb BRAMs, 2,520 DSP48Es). Table I lists a detail of the resource utilization of each module in turbo equalization. Different kinds of resource utilization of this design reach a good balance. The maximum frequency can be acheived is around 253 MHz.

TABLE I
Resource utilization summary when $N_{blk} = 1024, N = 151$

Size	Clocks	BRAM	DSP48E	LUT	FF
Gram matrix	500	906	1216	142084	212336
Matrix Inversion	184914				
SIC	10360	8			
MAP decoder	3084	18	0	4163	3077
others	1090	4	48	$< 1\%$	$< 1\%$
Total	199948	52%	50%	54%	41%

This resource utilization is under the 32-bit data width complex matrix arithmetic. Figure 8 plots the average error [14] as a function of different matrix size.

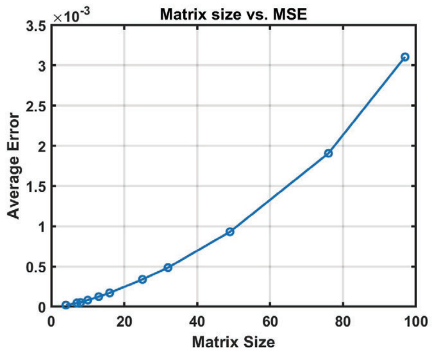

Fig. 8. Error as function of matrix size for a word length of 32 bits.

From Table I, we can find that the latency of matrix inversion dominates the whole process time. The latency of each step in the matrix inversion is shown in Table II. The reciprocal operation in the decomposition step has a latency of about 41. If the number of rows in the iteration is greater than 37 considering the additional pipeline depth, this latency would be hidden in the processing time of the decomposition iteration.

TABLE II
Steps Latency of Matrix Inversion

Steps	Latency (clock cycles)	N=151
U	$19L^2/2 - 18L + 54 (N > 37)$	93536
	$109L/2 + 41 (N \leq 37)$	NA
$\mathbf{U}^{-1}$	$63L^2/8 + 15/4L + 6$	79131
$\mathbf{U}^{-1}\mathbf{D}^{-1}$	$3L/2 + 7$	157
Multiplication	$9L^2/8 + 39/4L + 8 + \log_2(3L/2)$	12090

A latency comparison of this work with HLS results [14] is depicted in 9.

The total processing time for a 1024-symbol block when $L = 100, N = 151$ becomes 1.6 ms with the operational frequency of 125 MHz. Therefore, this architecture is able to support two turbo iterations and achieve 200 kilo-symbols-per-second (ksps) transmission rate.

V. Conclusion

In this paper, we propose an FPGA accelerator for turbo equalization in severe multipath wireless channels. By exploiting the Hermitian symmetric property of the channel

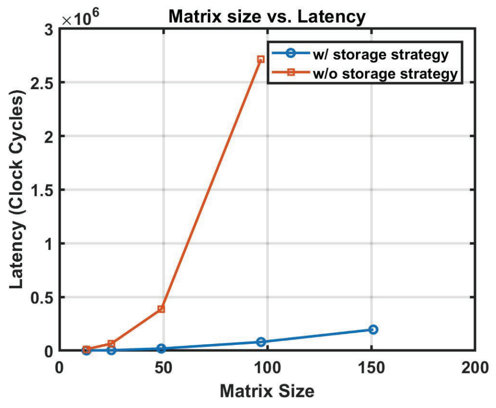

Fig. 9. Latency as function of matrix size.

Gram matrix and convolutional nature of the SIC, we reduce the processing time needed for large-scale matrix arithmetic, where the main latency of the turbo equalization comes from. We implement the proposed accelerator architecture on a Xilinx Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit. It is able to support Two turbo iterations for 1024-symbol block size and achieve 200 ksps transmission rate.

References

[1] Y. R. Zheng, J. Wu, and C. Xiao, “Turbo equalization for single-carrier underwater acoustic communications,” *IEEE Communications Magazine*, vol. 53, no. 11, pp. 79–87, November 2015.

[2] M. Mozaffari, W. Saad, M. Bennis, Y. Nam, and M. Debbah, “A tutorial on uavs for wireless networks: Applications, challenges, and open problems,” *IEEE Communications Surveys Tutorials*, vol. 21, no. 3, pp. 2334–2360, thirdquarter 2019.

[3] L. Dai, Z. Wang, and Z. Yang, “Next-generation digital television terrestrial broadcasting systems: Key technologies and research trends,” *IEEE Communications Magazine*, vol. 50, no. 6, pp. 150–158, June 2012.

[4] M. Rizk, A. Baghdadi, M. Jézéquel, Y. Mohanna, and Y. Atat, “Design and prototyping flow of NISC-based flexible MIMO turbo-equalizer,” in *2014 25th IEEE International Symposium on Rapid System Prototyping*, Oct 2014, pp. 16–21.

[5] G. Liu, Z. Wang, J. Hu, Z. Ding, and P. Fan, “Cooperative NOMA Broadcasting/Multicasting for Low-Latency and High-Reliability 5G Cellular V2X Communications,” *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 7828–7838, Oct 2019.

[6] L. Boher, R. Rabineau, and M. Helard, “Fpga implementation of an iterative receiver for mimo-ofdm systems,” *IEEE Journal on Selected Areas in Communications*, vol. 26, no. 6, pp. 857–866, 2008.

[7] A. R. Jafri, D. Karakolah, A. Baghdadi, and M. Jézéquel, “Asip-based flexible mmse-ic linear equalizer for mimo turbo-equalization applications,” in *2009 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2009, pp. 1620–1625.

[8] B. Han, Z. Yang, and Y. R. Zheng, “Fpga implementation of qr decomposition for mimo-ofdm using four cordic cores,” in *2013 IEEE International Conference on Communications (ICC)*. IEEE, 2013, pp. 4556–4560.

[9] M. Schwall, T. Bose, and F. K. Jondral, “On the performance of sc-mmse-fd equalization for fixed-point implementations,” in *2014 8th International Symposium on Turbo Codes and Iterative Information Processing (ISTC)*. IEEE, 2014, pp. 97–101.

[10] M. Rizk, A. Baghdadi, M. Jézéquel, Y. Mohanna, and Y. Atat, “Design and prototyping flow of flexible and efficient nisc-based architectures for mimo turbo equalization and demapping,” *Electronics*, vol. 5, no. 3, p. 50, 2016.

[11] R. Le Bidan, C. Laot, D. Leroux, and A. Godet, “Real-time turbo-equalization on the tms320c5509-improving the reliability of broadband wireless links,” 2004.

[12] B. Peng and H. Dong, “Dsp based real-time single carrier underwater acoustic communications using frequency domain turbo equalization,” *Physical Communication*, vol. 18, pp. 40–48, 2016.

[13] J. Xu, J. Sun, and Q. Kang, “Design of single carrie-frequency domain equalization system based on fpga,” 2018.

[14] M. Ruan, “Scalable Floating-point Matrix Inversion Design Using Vivado High-Level Synthesis,” in *Application Notes*. Xilinx, 2017, pp. 1–20.

[15] P. White, “Advanced QRD Optimization with Intel HLS Compiler,” in *White Paper*. Intel, 2018, pp. 1–18.

[16] C. Ingemarsson and O. Gustafsson, “On fixed-point implementation of symmetric matrix inversion,” in *2015 European Conference on Circuit Theory and Design (ECCTD)*. IEEE, 2015, pp. 1–4.

[17] M. Tuchler, R. Koetter, and A. C. Singer, “Turbo equalization: principles and new results,” *IEEE transactions on communications*, vol. 50, no. 5, pp. 754–767, 2002.

[18] M. Rizk, A. Baghdadi, M. Jézéquel, Y. Mohanna, and Y. Atat, “Quantization and fixed-point arithmetic for mimo mmse-ic linear turbo-equalization,” in *2013 25th International Conference on Microelectronics (ICM)*. IEEE, 2013, pp. 1–4.

[19] V. Belov and S. Mosin, “Fpga implementation of lte turbo decoder using max-log map algorithm,” in *2017 6th Mediterranean Conference on Embedded Computing (MECO)*. IEEE, 2017, pp. 1–4.

[20] B. Han, Z. Yang, and Y. R. Zheng, “Efficient implementation of iterative multi-input–multi-output orthogonal frequency-division multiplexing receiver using minimum-mean-square error interference cancellation,” *IET Communications*, vol. 8, no. 7, pp. 990–999, 2014.

[21] H. Lou and C. Xiao, “Soft-decision feedback turbo equalization for multilevel modulations,” *IEEE Transactions on Signal Processing*, vol. 59, no. 1, pp. 186–195, 2010.