VLSI Architectures for Turbo Decoding Message Passing Using Min-Sum for Rate-Compatible Array LDPC Codes

Kiran Gunnam, Weihuang Wang, Gwan Choi, Mark Yeary[*]
Dept. of Electrical Engineering, Texas A&M University, College Station, TX-77840
[*] Dept. of Electrical and Computer Engineering, University of Oklahoma, Norman, OK-73109

Abstract

Turbo decoding message passing (TDMP) or layered decoding has been proposed for the decoding of Low-Density Parity-Check (LDPC) codes using a trellis based BCJR algorithm in check node units (CNU). We present a new architecture for supporting rate compatible array LDPC codes that uses an offset min-sum decoding algorithm instead of the BCJR. The proposed architecture utilizes the value–reuse properties of min-sum and block-serial scheduling of computations, along with TDMP. This results in reduction of logic, interconnect requirements, and memory requirements of the decoder. The synergetic combination of TDMP, min-sum and array codes result in the following features: removal of memory needed to store the sum of channel values and variable node messages, 40%-72% savings in storage of extrinsic messages depending on rate of the codes, reduction of routers by 50% and increase of throughput by 2x. Variation of the proposed architecture is also presented for hardware resource-throughput trade-off. Implementation on the FPGA achieves throughput of 1.36 Gbps for each iteration.

1. Introduction

Low-Density Parity-Check (LDPC) codes and Turbo codes are among the known near Shannon limit codes that can achieve very low bit error rates for low signal-to-noise ratio (SNR) applications [1]. When compared to the decoding algorithm of Turbo codes, LDPC decoding algorithm has more parallelization, low implementation complexity, low decoding latency, as well as no error-floors at high SNRs. LDPC codes are considered for virtually all the next generation communication standards.

A binary (N, K) LDPC code is a linear block code of codeword length N and information block length K that can be described by a sparse $M \times N$ parity check matrix where M denotes the number of parity check equations. LDPC codes can be decoded by the Gallager's iterative two phase message passing algorithm (TPMP) which involves check node update and variable node update as two phase schedule. Various algorithms are available for check node updates and widely used algorithms are sum of products (SP), min-sum (MS) and Jacobian based BCJR (named after its discoverers Bahl, Cocke, Jelinik and Raviv). Mansour and Shanbhag [2] introduced the concept of turbo decoding message passing (TDMP), which is sometimes also called as layered decoding, using BCJR for their architecture-aware LDPC (AA-LDPC) codes. TDMP offers 2x throughput and significant memory advantages when compared to TPMP. This is later studied and applied for different LDPC codes using sum of products algorithm and its variations in [3-4. TDMP is able to reduce the number of iterations required by up to 50% without performance degradation when compared to the standard message passing algorithm. A quantitative performance comparison for different check updates was given by Chen and Fossorier et al [5]. Their research showed that offset based min-sum with 5-bit quantization could achieve same bit-error rate (BER) performance as that of floating point SP and BCJR with less than 0.1 dB penalty in SNR. While fully-parallel LDPC decoder designs [6] suffered from complex interconnect issues, various semi-parallel implementations based on structured LDPC codes [2, 7-11] alleviate the interconnect complexity. Among several structured codes proposed, array LDPC codes, AA-LDPC, block-LDPC and cyclotomic based LDPC are the popular ones and they all share the property that the H matrix is constructed out of cyclic shifted version of identity matrix and null matrices. This paper focuses on the implementation for rate compatible array codes.

In this work, we propose to apply TDMP for the offset MS for rate-compatible array LDPC codes. The main contribution is an efficient architecture which utilizes the value–reuse property of offset MS, cyclic shift property of structured LDPC codes and block serial scheduling. The resulting decoder architecture has, to our best knowledge, the lowest requirements of logic, interconnection and memory The rest of the paper is organized as follows. Section 2 introduces the background of array LDPC cods and min-sum the decoding algorithm. Section 3 presents the equations which facilitate the decoding process. Section 4 presents the value-reuse property and new micro-architecture structure for CNU. The data flow graph and architecture for TDMP using offset MS is included in Section 5. Section 6 shows the FPGA implementation results and performance comparison with related work and Section 7 concludes the paper.

2. Background

2.1 Array LDPC Codes

A brief review of the array codes is provided to facilitate the TDMP and the decoder architecture. The reader is referred to [11] for a comprehensive treatment on Array LDPC codes. The array LDPC parity-check matrix is specified by three parameters: a prime number p and two integers k and j such that $j,k < p$. It is given by

$$H_A = \begin{bmatrix} I & I & I & \cdots & I \\ I & \alpha & \alpha^2 & ... & \alpha^{k-1} \\ I & \alpha^2 & \alpha^4 & ... & \alpha^{2(k-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ I & \alpha^{j-1} & \alpha^{(j-1)2} & \cdots & \alpha^{(j-1)(k-1)} \end{bmatrix} \quad (1a)$$

where I is the $p \times p$ identity matrix , and α is a $p \times p$ permutation matrix representing a single left or right cyclic shift of I. Power of α in H denote multiple cyclic shifts, with the number of shifts given by the value of the exponent. In the following discussion, we use the α as a $p \times p$ permutation matrix representing a single left cyclic shift of I .Rate-compatible array LDPC codes are modified version of above for efficient encoding and multi-rate compatibility in [11] and their H matrix has the following structure

$$H = \begin{bmatrix} I & I & I & \cdots & I & I & \cdots & I \\ O & I & \alpha & \cdots & \alpha^{j-2} & \alpha^{j-1} & & \alpha^{k-2} \\ O & O & I & \cdots & \alpha^{2(j-3)} & \alpha^{2(j-2)} & & \alpha^{2(k-3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & & \vdots \\ O & O & \cdots & \cdots & I & \alpha^{(j-1)} & \cdots & \alpha^{(j-1)(k-j)} \end{bmatrix} \quad (1b)$$

were O is the $p \times p$ null matrix. The LDPC codes defined by H in (1b) have a codeword length $M = jp$, number of parity checks $M = kp$, and an information block length $K = (k - j)p$. The family of rate-compatible codes are obtained by successively puncturing the leftmost p columns and the topmost p rows. According to this construction, a rate-compatible code within a family can be uniquely specified by a single parameter, say, q , with $0 < q \le j - 2$. To have a wide range of rate-compatible codes, we can also fix j , p and select different values for the parameter k . Since all the codes share the same base matrix size p , the same hardware implementation can be used. It is worth mentioning that this specific form is suitable for efficient linear-time LDPC encoding [11]. The systematic encoding procedure is carried out by associating the first $N - K$ columns of H with parity bits and the remaining K columns with information bits.

2.2 Min-sum decoding of LDPC

In this paper, we follow the notation as in [5] and expand equations in alternate forms as necessary for illustrating various properties of the min-sum decoding scheme. Assume binary phase shift keying (BPSK) modulation (a 1 is mapped to -1 and a 0 is mapped to -1) over an additive white Gaussian noise (AWGN) channel. The received values y_n are Gaussian with mean $x_n = \pm 1$ and variance σ^2 .

The reliability messages used in belief propagation based min-sum algorithm can be computed in two phases, viz. (1) check node processing and (2) variable node processing. The two operations are repeated iteratively untill the decoding criterion is satisfied. This is also referred to as standard message passing or two phase message passing (TPMP). For the i^{th} iteration, $Q_{nm}^{(i)}$ is the message from variable node n to check node m , $R_{mn}^{(i)}$ is the message from check node m to bit node n , $\mathrm{M}(n)$ is the set of the neighboring check nodes for variable node n , $\mathrm{N}(m)$ is the set of the neighboring variable nodes for check node m . The message passing for TPMP are calculated in the following steps:

(1) check node processing: for each m and $n \in \mathrm{N}(m)$,

$$R_{mn}^{(i)} = \delta_{mn}^{(i)} \kappa_{mn}^{(i)} \quad (2)$$

$$\kappa_{mn}^{(i)} = \left|R_{mn}^{(i)}\right| = \min_{n' \in \mathrm{N}(m) \setminus n} \left|Q_{n'm}^{(i-1)}\right|. \tag{3}$$

The sign of check node message $R_{mn}^{(i)}$ is defined as

$$\delta_{mn}^{(i)} = \left(\prod_{n' \in \mathrm{N}(m) \setminus n} \mathrm{sgn}\left(Q_{n'm}^{(i-1)}\right) \right) \tag{4}$$

where $\delta_{mn}^{(i)}$ takes value of $+1$ or -1

(2) Variable node processing: for each n and $m \in \mathrm{N}(n)$,

$$Q_{nm}^{(i)} = L_n^{(0)} + \sum_{m' \in \mathrm{M}(m) \setminus m} R_{m'n}^{(i)} \tag{5}$$

where the log-likelihood ratio of bit n is $L_n^{(0)} = y_n$.

(3) Decision

$$P_n = L_n^{(i)} + \sum_{m \in M(n)} R_{mn}^{(i)} \tag{6}$$

A hard decision is taken where $\hat{x}_n = 0$ if $P_n(x_n) \geq 0$, and $\hat{x}_n = 1$ if $P_n(x_n) < 0$. If $\hat{x}_n H^T = 0$, the decoding process is finished with $\hat{x}_n$ as the decoder output; otherwise, go to step (1). If the decoding process doesn't end within some maximum iteration, stop and output an error messenger.

The reliability messages $R_{mn}^{(i)}$ computed in min-sum algorithm are overestimated. The overestimation is corrected in offset min-sum algorithm by subtracting a positive constant β from the magnitude of the $R_{mn}^{(i)}$ in the following way:

$$R_{mn}^{(i)} = \delta_{mn}^{(i)} \max\left(\kappa_{mn}^{(i)} - \beta, 0\right) \tag{7a}$$

Overestimation in $R_{mn}^{(i)}$ can also be corrected by a scaling factor λ

$$R_{mn}^{(i)} = \delta_{mn}^{(i)} \lambda \kappa_{mn}^{(i)} \tag{7b}$$

Both β and λ depends on the code parameters and the iteration number [5]. For (3, 6) rate 0.5 code, β is computed as 0.15 using the density evolution technique and can be used as constant for all iterations[2]. We used the procedure in [2] to compute β as 0.52 for (3, 32) rate 0.9 code and 0.75 for (5, 25) rate 0.8 array code. Offset MS is computationally less complex to implement as adder instead of multiplier is needed. Though a multiplier can be realized with constant wire shifts as done in [12], this does not permit the possibility of changing the correction factor in run time to support a code with different parameters based on channel conditions.

3. Vector Equations for TDMP for Array Codes

In TDMP, the array LDPC can be viewed as concatenation of j layers or block rows similar to observations made for AA-LDPC codes in [2]. In TDMP, after the check node processing is finished for one block row, the messages are immediately used to update the variable nodes, whose results are then provided for processing the next block row of check nodes. This differs from TPMP where, all check nodes are processed first and then the variable node messages will be computed. Each decoding iteration in the TDMP is composed of j number of sub-iterations. In the beginning of the decoding process, variable messages are initialized as channel values and they are used to process the check nodes of the first block row. After completion of that block row, variable messages are updated with the new check node messages. This concludes the first sub-iteration. In similar fashion, result of check node processing of the second block row is immediately used in the same iteration to update the variable node messages for third block row. The completion of check-node processing and associated variable-node processing of all block rows constitutes one iteration.

To develop the vector equations for TDMP for array LDPC codes, we first assume that the H matrix has the structure in (1a). We later discuss how the same equations are applicable to the H matrix of interest, rate compatible array codes defined by (1b), with minor initialization adjustments.

We can represent R and Q messages by the following matrices

$$Rm=\begin{bmatrix} R_{1,Row[1][1]} & R_{1,Row[1][2]} & \cdots & R_{1,Row[1][k]} \\ R_{2,Row[2][1]} & R_{2,Row[2][2]} & \cdots & R_{2,Row[2][k]} \\ \vdots & \vdots & \vdots & \vdots \\ R_{jp,Row[jp][1]} & R_{jp,Row[jp][1]} & \cdots & R_{jp,Row[jp][k]} \end{bmatrix} \quad Qm=\begin{bmatrix} Q_{1,Col[1][1]} & Q_{2,Col[2][1]} & \cdots & Q_{kp,Col[kp][1]} \\ Q_{1,Col[1][2]} & Q_{2,Col[2][2]} & \cdots & Q_{kp,Col[kp][2]} \\ \vdots & \vdots & \vdots & \vdots \\ Q_{1,Col[1][j]} & Q_{2,Col[2][j]} & \cdots & Q_{kp,Col[kp][j]} \end{bmatrix} \tag{8}$$

where $Row[m][1,2\cdots,k]=N[m]$, $Col[1,2,\cdots,j][n]=M[n]$

We now represent the R and Q messages in a p x p block as p x 1 vectors

$$\vec{R}_{m,n}=\left[Rm_{1+(m-1)p,n},...,Rm_{v+(m-1)p,n},...,Rm_{p+(m-1)p,n}\right]^T \tag{9}$$

$$\vec{Q}_{m,n}=\left[Qm_{m,1+(n-1)},...,Qm_{m,v+(n-1)p},...,Qm_{m,p+(n-1)p}\right]^T \tag{10}$$

$\forall v=1,2,....,p$, $\forall m=1,2,....,j$, and $n=1,2,....,k$,

$$\vec{R}=\begin{bmatrix} \vec{R}_{1,1} & \vec{R}_{1,2} & \cdots & \vec{R}_{1,k} \\ \vec{R}_{2,1} & \vec{R}_{2,2} & \cdots & \vec{R}_{2,k} \\ \vdots & \vdots & \vdots & \vdots \\ \vec{R}_{j,1} & \vec{R}_{j,1} & \cdots & \vec{R}_{j,k} \end{bmatrix}, \vec{Q}=\begin{bmatrix} \vec{Q}_{1,1} & \vec{Q}_{1,2} & \cdots & \vec{Q}_{1,k} \\ \vec{Q}_{2,1} & \vec{Q}_{2,2} & \cdots & \vec{Q}_{2,k} \\ \vdots & \vdots & \vdots & \vdots \\ \vec{Q}_{j,1} & \vec{Q}_{j,2} & \cdots & \vec{Q}_{j,k} \end{bmatrix} \tag{11}$$

Do the TDMP by doing the message processing in each block row in serial fashion till the decoding criterion is satisfied. The TDMP can be described with the following steps:

$$\vec{R}_{l,n}^{(0)}=0, l=1,2,\cdots,j \tag{12}$$

$$\vec{P}_n=\vec{\Lambda}_n \tag{13}$$

For $\forall i=1,2,\cdots$, $\forall l=1,2,\cdots,j$, $\forall n=1,2,\cdots,k$

$$\vec{Q}_{l,n}^{(i)}=\vec{P}_n-\left[\vec{R}_{l,n}^{(i-1)}\right]^{S(l,n)} \tag{14}$$

$$\vec{R}_{l,n}^{(i)}=f\left(\left[\vec{Q}_{l,n'}^{(i)}\right]^{-S(l,n)}\right), \forall n'=1,2,\cdots k \tag{15}$$

$$\vec{P}_n=\vec{P}_n-\left[\vec{R}_{l,n}^{(i-1)}\right]^{S(l,n)}+\left[\vec{R}_{l,n}^{(i)}\right]^{S(l,n)}=\vec{Q}_{l,n}^{(i)}+\left[\vec{R}_{l,n}^{(i)}\right]^{S(l,n)} \tag{16}$$

where $s(l,n)$ denotes the shift coefficient for the l^{th} block row and n^{th} block column of the H matrix; $\left[\vec{R}_{l,n}^{i-1}\right]^{S(l,n)}$ denotes that the vector $\vec{R}_{l,n}^{i-1}$ is cyclically shifted up by the amount $s(l,n)$. A negative sign on the $s(l,n)$ indicates it is cyclic down shift (equivalent cyclic left shift). $f(\cdot)$ denotes the check node processing. This check node processing can be done using BCJR, SP or MS. For the proposed work we will be using MS as defined in equation 2 and 3. The above equations can be transformed into the following set of equations:

$\forall l=1,2,\cdots,j$

$$\left[\vec{Q}_{l,n'}^{(i)}\right]^{-S(l,n)}=\left[\vec{P}_n\right]^{-S(l,n)}-\vec{R}_{l,n}^{(i-1)} \tag{17}$$

$$\vec{R}_{l,n}^{(i)}=f\left(\left[\vec{Q}_{l,n'}^{(i)}\right]^{-S(l,n)}\right), \forall n'=1,2,\cdots k \tag{18}$$

$$\left[\vec{P}_n\right]^{-S(l,n)}=\left[\vec{Q}_{l,n'}^{(i)}\right]^{-S(l,n)}+\vec{R}_{l,n}^{(i)} \tag{19}$$

If we are processing a block row in serial fashion using p check node units (17), then the output of the CNU will also be in serial form. As soon as the output vector $\vec{R}_{l,n}^{(i)}$ corresponding to each block column n in H matrix for a block row l is available, this could be used to produce updated sum $\left[\vec{P}_n\right]^{-S(l,n)}$ (18). This could be immediately used in (16) to process

block row $l+1$ except that the shift $s(l,n)$ imposed on $\vec{P}_n$ has to be undone and a new shift $s(l+1,n)$ has to be imposed. This could be simply done by imposing a shift corresponding to the difference of $s(l+1,n)$ and $s(l,n)$. Note that due to the slight irregularity in Array LDPC matrix defined in (1b), each block row l has a node degree $j-l+1$. The variable nodes in each block column k has a node degree (k, j). We have to devise a simple control mechanism to address this. One possible way to deal with this check node irregularity is setting the check node degrees in CNU processor unit based on the block row that is being processed. Another simpler way to facilitate implementation is to assume that all the block rows have equal check node degree and set the check node messages corresponding to null blocks in H matrix to zero in order not to affect the variable node processing. $\vec{R}_{l,n}^{(i)}=0$ if $n<l$ in each iteration *i*. Similarly the variable node messages belonging to the null blocks are always set to positive infinity in order not to affect the check node processing. $\vec{Q}_{l,n}^{(i)}=\infty$ if $n<l$. For check node update based on SP or MS, the message with maximum reliability won't affect the CNU output. In the specific case of MS, it is easy to see this as the CNU magnitude is dependent on the two least minimum and the sign bit processing is based on XOR operation.

4. Value-reuse Properties of Check Node Processing of Min-Sum Decoding

Equation (3) is determined by identifying the two minimum values corresponding to this check sum. Check node processing can be achieved with at most $2d_c$ comparisons in serial search, or with at most $d_c+\log 2d_c-2$ comparisons with the help of a binary tree [5]. One of the key contributions is the observation that min-sum, normalized min-sum and offset min-sum decoding algorithm share the value-reuse property, as explained below, hence reduce the message passing requirement of the decoder. For each check node m, $\left|L_{mn}^{(i)}\right|$ $\forall n\in \mathrm{N}(m)$ takes only 2 values. Define the least minimum and the second least minimum of the entire set of the messages from various variable nodes to the check node m as

$$M1_m^{(i)}=\min_{n'\in \mathrm{N}(m)}\left|Q_{mn'}^{(i-1)}\right|,\ M2_m^{(i)}=\min_{n'\in \mathrm{N}(m)\,2nd}\left|Q_{mn'}^{(i-1)}\right|. \tag{20}$$

Now equation (2) becomes

$$\kappa_{mn}^{(i)}=M1_m^{(i)},\ \forall n\in \mathrm{N}(m)\setminus k$$
$$=M2_m^{(i)},n=k \tag{21}$$

The check node message $R_{mn}^{(i)}$ $\forall n\in \mathrm{N}(m)$ then takes only 3 values. Since $\forall n\in \mathrm{N}(m)$, $\delta_{mn}^{(i)}$ takes value of either $+1$ or -1 and $\left|R_{mn}^{(i)}\right|$ takes only 2 values. Equation (2) gives rise to only 3 possible values, $M1_m^{(i)}$, $-M1_m^{(i)}$, and either $M2_m^{(i)}$ or $-M2_m^{(i)}$ for the whole set $R_{mn}^{(i)}$ $\forall n\in \mathrm{N}(m)$. In a VLSI implementation, this property greatly simplifies the logic and reduces the memory needed to store CNU outputs. Since only the two least minimum numbers need to be identified, the number of comparison can be reduced. In the CNU proposed in [12], 2* d_c comparators are used to compute Equation (3) associated with that check node.

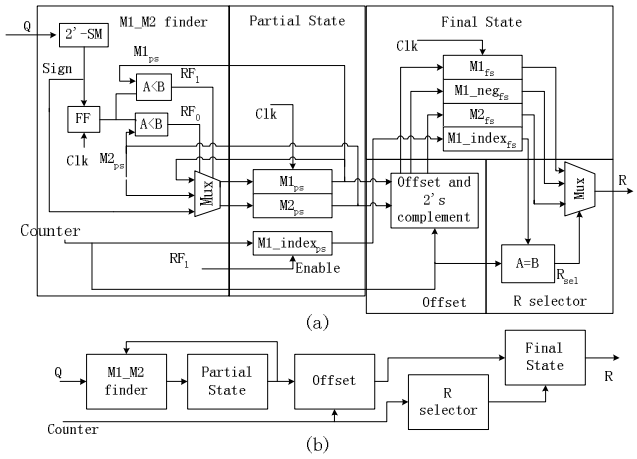

Figure1: Serial CNU for offset min-sum using value reuse property

We present efficient serial implementations for CNU. Figure 1(a) shows the CNU micro-architecture for (5, 25) code. In the first 25 clock cycles of the check node processing, incoming variable messages are compared with the two up-to-date least minimum numbers (partial state, PS) to generate the new partial state, M1 which is the least minimum value, M2 which is the second minimum value and index of M1. The final state (FS) is then computed by offsetting the partial

state. It should be noted that the final state include only $M1_m^{(i)}$, $-M1_m^{(i)}$, $\pm M2_m^{(i)}$ with offset correction. Figure 1(b) is the block diagram of the same architecture. M1_M2 finder computes the two least numbers, according to the incoming data and the current minimum numbers stored in Partial State. The Offset module applies the offset correction, and stores the results in the Final State module. R Selector then select the output R messages. While the final state has dependency on offset correction, the offset is dependent on the completion of partial state. In operation, the Final State and Partial State will operate on different check nodes. The proposed serial CNU finds the least two minimum numbers with 2 comparators in a serial fashion and reduces the number of offset-correction computation from k to 2. Normally CNU (check node unit) processing is done using the signed magnitude arithmetic for equations (2-4) and VNU (variable node unit processing) equation (5) is done in 2's complement arithmetic. This requires 2's complement to signed conversion at the inputs of CNU and signed to 2's complement at the output of CNU. In the proposed scheme, 2's complement is applied to only 2 values instead of k values at the output of CNU. The value re-use property also reduces the memory requirement significantly. Conventionally, number of messages each CNU store is equal to the number of edges it has, that is k. Now only four unit of information is needed: the three values that $R_{mn}^{(i)}$ may take and the location of $M1_m^{(i)}$, then check node message to the VNU is readily chosen by multiplexing. Up to 90% saving in addition operation and memory in the CNU could be achieved by utilizing this value re-use property, depending on different coding rates.

5. Architecture Using TDMP and Min-Sum

5.1 Optimally scaled architecture

A new data flow graph is designed based on the TDMP and on the value reuse property of min-sum algorithm described above. For ease of discussion, we will illustrate the architecture for a specific structured code: Array code of length 1525 described in section 2, $j=5$, $k=25$ and $p=61$, the discussion can be easily generalized to any other structured codes. First, functionality of each block in the architecture is explained. A check node process unit (CNU) is the serial CNU based on offset min-sum described in previous section. The CNU array is composed of *p* computation units that compute the partial state for each block row to produce the R messages in block serial fashion. Since final state of previous block rows, in which the compact information for CNU messages is stored, is needed for TDMP, it is stored in register banks. There is one register bank of depth $j-1$, which is 4 in this case, connected with each CNU. Each final state is the same as the final state register bank in the CNU. Besides the shifted Q messages, the CNU array also take input of the sign information for previous computed Q messages in order to perform R selection operation. The sign bits are stored in sign FIFO. The total length of sign FIFO is k and each block row has p one bit sign FIFOs. We need $j-1$ of such FIFO banks in total. *p* number of R select units is used for R_{old}. An R select unit generates the R messages for $25(=k)$ edges of a check node from 3 possible values stored in final state register associated with that particular check node in a serial fashion. Its functionality and structure is the same as the block denoted as R select in CNU. This unit can be treated as de-compressor of the check node edge information which is stored in compact form in FS registers. The generation of R messages for all the layers in this way amounts to significant memory savings- which would be quantified in later section. The shifter is constructed as cyclic down logarithmic shifter to achieve the cyclic shifts specified by the binary encoded value of the shift. The logarithmic shifter is composed of $\log 2(p)$ stages of p switches. A logarithmic shifter works by successively shifting (or, not, depending on a bit of the shift) the input by powers of two, such that the result, the product of each of these successive shifts, is shifted overall by the value of the shift index: the first stage shifts by 1 place if bit 0 of the index is set; the next by 2 places if bit 1 is set; the next by 4 places if bit 2 is set; and so on until the last stage, which shifts by *n/2* places if bit *(log2n)-1* of the index is set. Since cyclic up shift is also needed in the operation of the decoder, cyclic up shift by u can be simply achieved by doing cyclic down shift with $p-u$ on the vector of size p .This is a slight improvement to the use of Omega networks [2] which need the storage of switching sequences of the order of $jk(p/2)\log 2(p)$ bits for every shift that is needed in the decoding process.

The decoding operation proceeds as per the vector equations described in section 3. In the beginning of the decoding process, P vector is set to received channel values in the first k clock cycles(i.e. the first sub-iteration) as the channel values arrive in chunks of p while the output vector of R select unit is set to zero vector. The multiplexer array at the input of cyclic shifter is used for this initialization. The CNU array takes the output of the cyclic shifter serially and the partial state stage will be operating on these values. After k clock cycles, partial state processing will be complete and the final state stage in CNU array will produce the final state for each check node in 2 clock cycles. Then R select unit within the each CNU unit starts generating k values of check node messages in serial fashion. The CNU array thus produces the

check node messages in a block serial fashion as there are p CNUs are operating in parallel. The P vector is computed by adding the delayed version of the Q vector (which is stored into a FIFO SRAM to till the serial CNU produces the output) to the output vector R of the CNU. Note that the P vector that is generated can be used immediately to generate the Q vector as the input to the CNU array as CNU array is ready to process the next block row. This is possible because CNU processing is split into three stages as shown in the pipeline diagram and partial state stage and final state stage can operate simultaneously on two different block rows. Now the P message vector will undergo a cyclic shift by the amount of difference of the shifts of the block row that is processed and the previous block row that was just processed. Note that this shift value can be either positive or negative indicating that an up shift or down shift need to be performed by the cyclic shifter. The shifted P sum messages are subtracted by R message to get the shifted version of Q messages.

The snapshot of the pipeline of the decoder is shown in Figure 3. Here the partial state stage in CNU (CNU PS) is operating on the 2nd block row from clock cycles labeled as 0 to 24 (note that these numbers will not denote the actual clock numbers as the snapshot is shown in the middle of the processing). Final state stage in CNU (CNU FS) can not start until the end of PS processing, that is clock cycle 25. As soon as the FS is done in clock cycle 26, R select is able to select the output R messages, and P and Q messages processing starts. With the first block of Q message ready, PS for next block row can be started immediately..

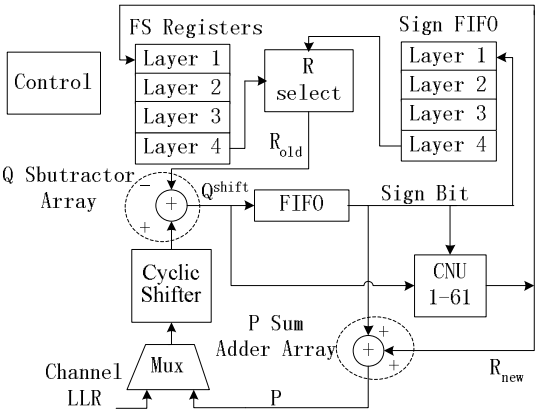

Figure 2: LDPC Decoder using Turbo Decoding for Message Passing and min-sum as SISO decoder

The control unit also contains the information of Array code parameters such as j, k, q – these could be changed to support multi-rate decoding. The family of rate-compatible codes obtained by successively puncturing the leftmost p columns and the topmost p rows in the H matrix defined in (1b) q times. Changing q from 0 to $3(= j-2)$ gives the code rates of 0.8 to 0.909. Changing k values from 15 to 61 while fixing $j = 5$ results in code rates from 0.666 to 0.91. The Q FIFO need to be of maximum depth p as the k can take a maximum value equal to p .

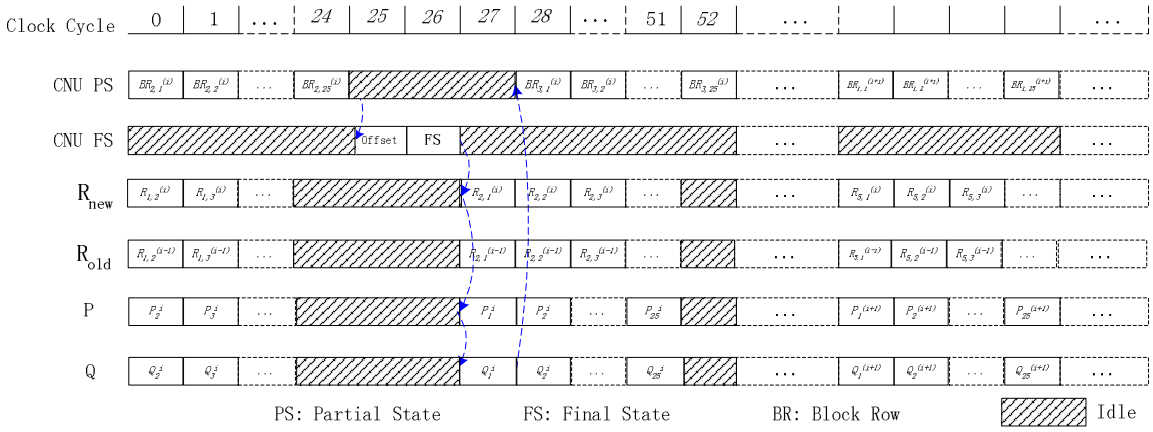

Figure 3: Block serial processing and 3-stage pipelining for TDMP using min-sum

The BER performance of Min Sum algorithm for TDMP and TPMP for (5,25) code of length 1525 with 5-bit quantization for R and Q messages, 6-bit quantization for P messages has a penalty of 0.1 dB for maximum number of 100 TDMP iterations. When the maximum number of TDMP iterations is set to 10, the total performance degradation is about 0.2 dB. The final BER performance the proposed decoder is 10^{-5} at 3.7dB and 10^{-6} at 4dB. The reference for the above comparisons is the SNR required for floating point SP in TPMP schedule with maximum number of 200 iterations to achieve the same BER. The 5-bit quantized SP in TPMP schedule needs around 20 iterations to achieve within 0.2 dB of its floating point performance. So the use of TPMP for min-sum decoding scheme for array LDPC codes results in a 2x

throughput improvement similar to what is being reported in for BCJR and SP [2-4]. For comprehensive treatment of rate compatible array codes and their BER performance, the reader is referred to [11].

5.2 Scalable architecture

Note that the throughput of the architecture is increased by increasing p of the code and scaling the hardware accordingly. While the complexity of computational units scale linearly with p, the complexity of cyclic shifter increases with the factor $(p/2)\log_2 p$. So it is necessary to change the architecture for large values of p. Alternatively it may be needed in low throughput applications to have low parallelization. To suit this requirement, minor changes in the proposed architecture are necessary. Let us assume the desired parallelization is $M < p$. For the ease of implementation choose M close to the powers of 2. The cyclic shifter needed is $M \times M$. Since It is needed to achieve $p \times p$ cyclic shift with consecutive shifts of $M \times M$,it is necessary that the complete vector of size p is available in M banks with the depth $s = (ceil(p/M))$ and shifting is achieved in part by the cyclic shifter and in part by the address generation. Now all the CNU and variable node processing is done in a time division multiplexed fashion for each sub-vector of length M so as to process the vector of size p to mimic the pipeline in Figure 3. Now instead of taking one clock cycle to process a block column in a block row (layer), it takes s clock cycles. The FS register bank external to the CNU and FS registers in CNU are now implemented as M banks of memory with depth equal to js and word length is 22 bits. In addition we need another memory with M banks with depth equal to s to store the partial state with the word length 17 bits as we need to store and retrieve (M1, M2, M1 Index and cumulative sign). Channel values need to be stored in a buffer of size p as the decoder needs any M values out of this buffer at each clock cycle. Note that this architecture consuming less number of logic when compared to fully scaled architecture, its memory requirements increased slightly. One way to look at the memory requirements of the scalable architecture($M < p$) is: All the logic resources, FS registers and PS registers are scaled down from p to M while having an external SRAM of size equal to the number of FS and PS registers used in the case of fully scaled architecture(i.e. $M = p$). Note that the exact memory bank organization can be changed by grouping different messages together, so less number of memory banks is possible.

6. Discussion and FPGA implementation results

The proposed TDMP architecture features large memory savings, up to 2x throughput advantage as well as 50% less interconnection complexity. Before quantifying the savings, a brief description of TPMP architectures optimized for array LDPC codes is necessary. Due to the nature of scheduling in the architecture possible due to structured properties of cyclotomic LDPC and array codes, it was shown in [9, 10] that there is need to store only one class of messages even while doing the simultaneous processing of R and Q messages belonging to different iterations. In [10] the requirement to store one class of messages is fulfilled through storing the intermediate messages in an SRAM FIFO whose total size is $\sim jkp$ messages. While p CNUs are associated with these FIFOs, p VNUs are associated with FIFOs whose total size is equal to jp messages to do the decoding in block serial fashion such that all the intermediate and final computations are done such that processing for one block column at a time is handled. The channel intrinsic values are stored in a memory whose size is equivalent to $5kp$, assuming 5-bit quantization. Now the TDMP permits us to use a running sum which is initialized to channel LLR values in the first iteration. So there is no memory needed to store the channel LLR values as these values are implicitly stored in the p messages. Also note that due to the nature of scheduling in the architecture possible due to structured properties of array codes, there is no need to store the P messages at all as P-message vector is consumed immediately whenever it is assigned new values. In addition, TDMP will make the p VNUs need not have FIFO with the depth $j = 5$. This is savings of $5jp$ memory bits. The total savings of memory bits as a direct consequence of employing TDMP for array LDPC codes when compared to TPMP architecture [9, 10] is $5p(k + j)$ memory bits.

In terms of throughput and interconnect advantage, to achieve the same BER as that of TPMP schedule on MS, TDMP schedule on MS need half the number of iterations. This essentially doubles the throughput. If we use 5-bit precision in R messages to achieve almost the same performance as that of the floating point SP. Moreover, this architecture requires only one cyclic shifter instead of two cyclic shifters as in case of TPMP architecture for array LDPC codes [9, 10].

It is also observed that instead of storing all the R messages, the compressed information cumulative sign, $M1_m^{(i)}$, $-M1_m^{(i)}$, either $M2_m^{(i)}$ or $-M2_m^{(i)}$ and index of $M1_m^{(i)}$ is stored. R select unit can generate the R message by the use of a index comparator and the XOR of the cumulative sign and the sign bit of the corresponding Q message which comes from the Sign FIFO. This results in a reduction of around 50%-90% of R memory based on the rate of the code when compared to BCJR algorithm or Sum of Products algorithm. Since the sign information of R memory is still stored implicitly in the form of sign information of Q and this can not be reduced in further compact form, the total savings in R memory is around of 40%-72% for 5 bit messages based on the rate of the code. So, in an architecture designed to support the different rates, the savings of R memory is 72%. In addition, to compute updated P, the input Q is buffered and available to the modules until CNU finishes the serial min-sum processing. The same principle is used by buffering the input until both forward-backward recursions are complete using a buffer at the outputs of lambda-memory modules [2]. This means that R has to be stored in a FIFO whose depth is equal to the latency of the CNU which is around k. Since there are p units, we need to have $5pk$ memory bits where the factor 5 comes due to the number of bits used to represent the soft messages. Similarly, due to the nature of the min-sum algorithm, there is no need of internal FIFO to carry the magnitude information. Only sign FIFO is needed and this sign information can be used from the external FIFO to the CNU.

Table 1: FPGA implementations and performance comparison

	No. Slice	No. LUT	Slice FF	BRAM	Frequency (MHz)	Throughput	code rate	code length	Code Type /Decoding
M. Karkooti et al [12]	11352	20374	NA	66	121	127 Mbps	0.5	1536	Structured (similar to array LDPC) MS, TPMP, 20 iterations
T. Brack et al [13]	14475	NA	NA	165	100	180 Mbps	0.8	1000-3000	irregular, MS, TPMP, 10 iterations
Y. Chen et al [14]	NA	72621	6779	32	44	40 Mbps	0.5	8888	irregular , SP, TPMP 24 iterations
Proposed (Optimally Scaled) M=61 P=61	6002	7713	9981	12 (33489 bits)	112	(1366*code rate/number of iterations) Mbps 125.4 Mbps for rate 0.918 code of length 3721. 68.3 Mbps for rate 0.5 code of length 610	0.5-0.9	610-3721	Rate compatible Array LDPC P=61, j=5, k=10 to 61, q=0 to 3 MS, TDMP Max. 10 iterations
Proposed (Scalable, M<P) M=61 P=347	6182	8022	10330	102 (55173 bits)	112	(2277*code rate/number of iterations) Mbps 113 Mbps for rate 0.5 code of length 2082. 159 Mbps for rate 0.7 code of length 3470.	0.5-0.7	2082-3470	Rate compatible Array LDPC P=347, j=3, k=6 to 10, q=0 to 1 MS, TDMP Max. 10 iterations

All implementations are on Virtex 2 family FPGA , and the specific device for the proposed implementation is 2V8000-5.

We prototyped the proposed multi-rate decoder architecture with a parallelization factor equal to p on Xilinx Virtex 2V8000-5 device. The synthesis results are given in Table 1. It is difficult to compare directly with other FPGA implementations due to the use of different structured codes and different code parameters. The TPMP architecture based on SP [10] is compared against the TDMP architecture based on MS as both are optimized for array LDPC codes. The results are shown in Figure 4.

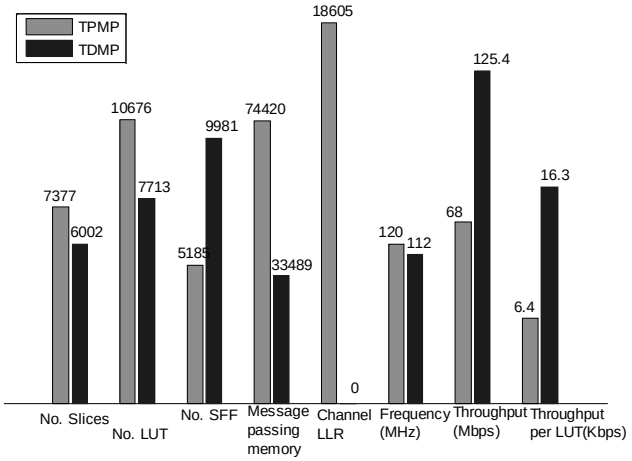

Figure 4: Comparison of TPMP-SP and TDMP-MS architectures optimized for array LDPC codes

7. Conclusion

We present memory efficient multi-rate decoder architecture for turbo decoding message passing of structured LDPC codes using the min-sum algorithm for check node update. Our work offers several advantages when compared to the other state of the art LDPC decoders in terms of significant reduction in logic, memory and interconnect. This work retains the key advantages offered by the original TDMP work – however our contribution is in using the value-reuse properties of offset MS algorithm and devising a new TDMP decoder architecture to offer significant additional benefits. We also presented the variations of architecture that offer a throughput-hardware resource trade-off.

References

[1] D.J.C. MacKay and R.M. Neal. "Near Shannon Limit Performance of Low Density Parity Check codes" *Electronics Letters*, volume 32, pages 1645-1646, Aug 1996.

[2] M. Mansour and N. Shanbhag, "A 640-Mb/s 2048-bit programmable LDPC decoder chip," *IEEE Journal of Solid-State Circuits*, vol. 41, no.3, pp. 684- 698, March 2006.

[3] H. Sankar and K.R. Narayanan, "Memory-efficient sum-product decoding of LDPC codes," *Communications, IEEE Transactions on*, vol.52, no.8pp. 1225- 1230, Aug. 2004

[4] Hocevar, D.E., "A reduced complexity decoder architecture via layered decoding of LDPC codes," *Signal Processing Systems*, 2004. SIPS 2004. IEEE Workshop on , .pp. 107- 112, 13-15 Oct. 2004

[5] J. Chen and M. Fossorier, ``Near Optimum Universal Belief Propagation Based Decoding of Low-Density Parity Check Codes", *IEEE Transactions on Communications*, vol. COM-50, pp. 406-414, March 2002.

[6] Blanksby, A.J.; Howland ,C.J, A 690-mW 1-Gb/s 1024-b, rate-1/2 low-density parity-check code decoder", *Solid-State Circuits, IEEE Journal of*, Vol.37, Iss.3, Mar 2002 Pages:404-412

[7] T. Zhang and Parhi, "A 54 Mbps (3, 6)-regular FPGA LDPC decoder," *IEEE Workshop on Signal Proc. Systems*, 2002. (SIPS '02), pp. 127-132, Oct. 2002.

[8] A. Selvarathinam, G.Choi, K. Narayanan, A. Prabhakar, E. Kim, "A massively scalable decoder architecture for low-density darity-check codes", in *Proceedings of ISCAS*, Bangkok, Thailand, 2003.

[9] Olcer, S., "Decoder architecture for array-code-based LDPC codes," *Global Telecommunications Conference*, 2003. GLOBECOM '03. IEEE , vol.4, no.pp. 2046- 2050 vol.4, 1-5 Dec. 2003

[10] K. Gunnam, G. Choi and M. B. Yeary, "An LDPC decoding schedule for memory access reduction", *IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 5, May 2004.

[11] Dholakia, A.; Olcer, S., "Rate-compatible low-density parity-check codes for digital subscriber lines," *Communications, 2004 IEEE International Conference on*, vol.1, no.pp. 415- 419, 20-24 June 2004

[12] M. Karkooti and J. Cavallaro, "Semi-parallel reconfigurable architectures for real-time LDPC decoding," *Proceedings of International Conference on Information Technology, Coding and Computing*, vol. 1, pp. 579-585, 2004.

[13] T. Brack, F. Kienle and N. Wehn. "Disclosing the LDPC Code Decoder Design Space", *Design Automation and Test in Europe (DATE) Conference*, pp. 200-205, March 2006.