VLSI Implementation of a Non-Binary Decoder
Based on the Analog Digital Belief Propagation

VLSI Implementation of a Non-Binary Decoder Based on the Analog Digital Belief Propagation

Muhammad Awais, Guido Masera, Guido Montorsi and Maurizio Martina
Politecnico di Torino, Department of Electronics and Telecommunication, Italy
{muhammad.awais,guido.masera,guido.montorsi,maurizio.martina}@polito.it

***Abstract*—This work presents the VLSI hardware implementation of a novel Belief Propagation (BP) algorithm introduced in [1] and named as Analog Digital Belief Propagation (ADBP). The ADBP algorithm works on factor graphs over linear models and uses messages in the form of Gaussian like probability distributions by tracking their parameters. In particular, ADBP can deal with system variables that are discrete and/or wrapped. A variant of ADBP can then be applied for the iterative decoding of a particular class of non binary codes and yields decoders with complexity independent of alphabet size M, thus allowing to construct efficient decoders for digital transmission systems with unbounded spectral efficiency. In this work, we propose some simplifications to the updating rules for ADBP algorithm that are suitable for hardware implementation. In addition, we analyze the effect of finite precision on the decoding performance of the algorithm. A careful selection of quantization scheme for input, output and intermediate variables allows us to construct a complete ADBP decoding architecture that performs close to the double precision implementation and shows a promising complexity for large values of M. Finally, synthesis results of the main processing elements of ADBP are reported for 45 nm standard cell ASIC technology.**

***Index Terms*—Belief propagation, APP estimation, iterative decoding, non binary LDPC, Analog Digital Belief Propagation,VLSI decoder, decoder architecture.**

I. Introduction and motivation.

Since their rediscovery by MacKay, LDPC codes have been extensively adopted in both next-generation wired and wireless standards due to their near-Shannon limit performance. Moreover, LDPC codes over non binary alphabets of size M can show better performance over their binary counterparts with proper encoding design and code length [2]. However, the significant improvement comes along with the penalty of high decoding complexity. The locally optimal, yet the most complex, iterative decoding algorithm of non binary LDPC codes is the belief propagation (BP) algorithm. Since the size of messages varies with the size of the alphabet M, a straightforward implementation of BP results in memory and complexity requirements of the order of $O(M)$ and $O(M^2)$ respectively. In order to reduce the complexity of non binary decoding, several suboptimal decoding schemes have been proposed in recent years.

The first straightforward simplification is obtained at check nodes by replacing the discrete convolution of messages, with complexity $O(M^2)$, with the product of the message Fourier transforms. The use of FFT brings down the complexity to $O(M \log M)$. In [3], the authors introduce a log-domain version of this approach that has advantages in terms of numerical stability.

Some further simplifications have been proposed in [4] with the Extended Min Sum (EMS) algorithm, where message vectors are reduced in size by keeping only those elements in the alphabet with higher reliability. In [5] the same authors propose a hardware implementation of the EMS decoding algorithm for non-binary LDPC codes.

In [6] the Min-Max algorithm is introduced with a reduced complexity implementation called selective implementation, which can reduce by a factor 4 the operations required at the check nodes; however, complexity is still in the order of $O(M^2)$.

Several studies on VLSI implementation of non binary decoders based on the previous algorithms have been presented in literature [7]–[13]. The results of such studies confirm that all non binary decoders require complexity growing with the size of the alphabet M.

The ADBP proposed in [1] represents a breakthrough in the reduction of the complexity and memory requirements with respect to previous proposed algorithms, as for ADBP both complexity and memory requirements are *independent* of the size M of the alphabet. The main simplification of ADBP is due to the fact that messages are not stored as vector of size M containing the likelihood of the discrete variables (or equivalently their log-likelihood ratios-LLR) but rather as the two moments, or related quantities, of some suitable predefined class of Gaussian-like distributions. ADBP can be casted into the general class of expectation-propagation algorithms described by Minka [14]. The main contribution in [1] is the definition of a suitable class of distributions for the messages and the derivation of the updating equations for the message parameters at the sum and repetition operations of the graph.

It should be noticed that ADBP cannot be applied to all types of linear codes over $GF(M)$ as multiplication by field elements different from ± 1 is not allowed in the graph. This ensemble of codes has been analyzed in [15] and [16], where it is shown that it is capacity achieving as its distance spectrum approaches that of random codes as the underlying graph connectivity grows.

The exact ADBP updating equations however are not suitable for a straightforward implementation due to the

presence of complex non linear operations. Some simplifications to the updating equations have been presented in [17]. In this paper we tackle the problem of VLSI implementation of ADBP with a systematic approach. In section II we start by reporting the exact updating equations of ADBP and consider its special application to the decoding of non binary codes. In section III we introduce some simplifications to the updating equations and evaluate their impact on the performance of the decoder. In section IV we present the results of the fixed point implementation of ADBP obtained by optimizing the bit width of input, output and intermediate quantities of the decoder processing elements. In section V we report the architecture of the designed core processors and the synthesis results for $M = 10$ up to $M = 64$. The provided results confirm that implementation of ADBP is feasible with small complexity and more importantly that complexity is independent of M.

II. The ADBP algorithm.

ADBP is a particular version of the BP algorithm that allows to perform in a very efficient way the BP for *linear* systems where variables can be either discrete or wrapped or both.

From the complexity point of view ADBP is equivalent to Gaussian BP over linear system as described in [18].

Let us define the class of **gaussian messages** as

$$\mathcal{G} \triangleq \left\{ G(\mu, K, x) \propto e^{-\frac{K}{2}|x-\mu|^2}, K \in \mathbb{R}^+, \mu \in \mathbb{R} \right\}$$

where μ and K denote respectively the mean and concentration of a Gaussian message. With continuous variables and messages belonging to $\mathcal{G}$, the following simple updating rules for linear real systems can be derived corresponding to the sum, repetition and axis scaling operations.

$$\text{Sum} \rightarrow G_1 * G_2 = G(\mu_1 + \mu_2, (K_1^{-1} + K_2^{-1})^{-1}) \in \mathcal{G} \quad (1)$$

$$\text{Repetition} \rightarrow G_1 \cdot G_2 = G\left(\frac{\mu_1 K_1 + \mu_2 K_2}{K_1 + K_2}, K_1 + K_2\right) \in \mathcal{G} \quad (2)$$

$$\text{Scaling} \rightarrow \alpha G = G(\alpha\mu, K/\alpha^2) \in \mathcal{G}. \quad (3)$$

In [18], it is shown that several powerful estimation techniques can be derived as particular instances of this algorithm.

ADBP introduced in [1] adds the possibility of wrapping and/or discretizing the random variables involved in the system. Wrapping of variables induces a wrapping of the corresponding messages and requires to use the class of **wrapped gaussian messages**:

$$\mathcal{W} \triangleq W[\mathcal{G}] = \{W[G(\mu, K, x)], K \in \mathbb{R}^+, \mu \in [0, M[\};$$

where M is the wrapping period and following wrapping operator is introduced

$$W[L(x)] \triangleq \sum_i L(x - iM) = s_M(x) * L(x), \quad (4)$$

In (4), the symbol $*$ indicates the convolution operation and $s_\Delta(x) \triangleq \sum_i \delta(x - i\Delta)$ is the train of pulses. The discretization of system variables on the other side, induces a *sampling* of their corresponding messages, leading to the introduction of the class of **sampled gaussian messages** : $\mathcal{S} \triangleq S[\mathcal{G}]$ where we have defined the sampling operator

$$S[L(x)] \triangleq s_M(x) \cdot L(x),$$

Notice that Fourier transforms of sampled gaussian messages are wrapped gaussian messages with imaginary mean.

When the wrapping period M is a multiple of the sampling interval, wrapping and sampling operators commute ($S[W[L]] = W[S[L]]$), so that it is possible to introduce the class of Digital messages or **D-Messages**, which consists of messages that are both wrapped and sampled:

$$\mathcal{D} \triangleq W[S[\mathcal{G}]] = S[W[\mathcal{G}]].$$

Examples of linear systems that use discrete and wrapped variables i.e. integers in the range $[0, M-1]$ are linear non-binary encoders, whose non binary codewords $\mathbf{c}$ satisfy

$$\mathbf{Hc} = \mathbf{0}$$

where the mod M sum is assumed in the equation and the coefficients of the parity check matrix $\mathbf{H}$ are bounded to be in the set $\{1, 0, -1\}$.

ADBP using D-Message can then be applied for the iterative decoding of members of this code ensemble, yielding decoders with complexity independent from the cardinality of the alphabet M.

In [1] it is shown that, in contrast to what happens for gaussian messages (equations (1-3)), wrapped messages are *not* closed under multiplication and sampled gaussian messages are not closed under convolution. As a consequence D-messages are not closed w.r.t repetition and sum operations.

In particular, while the updating equation (2) is reliable also for wrapped messages for large concentration of the messages, it fails to provide accurate results when the concentration is small. This is due to the non negligible effect of the aliasing on the replicas introduced by wrapping. For the same reason, equation (1) fails to provide accurate results for sampled gaussian messages with *high* concentration.

However, accurate approximations of the output messages belonging to the same class of the inputs can be found in both cases. This is obtained by exploiting the correspondence between members of the class of wrapped gaussian messages of period M with those of the class of Von Mises or Tychonov messages with the same period:

$$\mathcal{V} \triangleq \left\{ V(\mu, K_v, x) \propto e^{K_v \cos\left(A(x-\mu)\right)}, K_v \in \mathbb{R}^+, \mu \in [0, M[\right\},$$

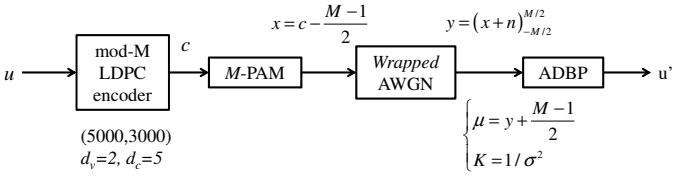


Figure 1: Block diagram of high spectral efficiency transmission system employing ADBP

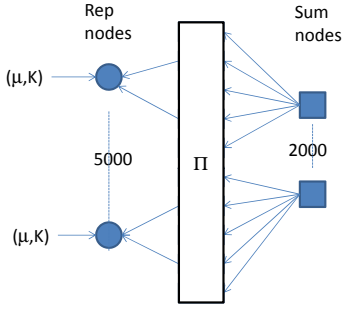


Figure 2: Tanner graph of LDPC code adopted in proposed ADBP algorithm

Where $A \triangleq \frac{2\pi}{M}$. The mapping between the two distributions preserves the mean and transforms the concentration according to

$$K_w \quad \Leftrightarrow \quad A^2 f(K_v) \tag{5}$$

$$f(K_v) \quad \triangleq \quad \left(2\log\left(\frac{I_0(K_v)}{I_1(K_v)}\right)\right)^{-1} \tag{6}$$

Where $I_n(x)$ denotes the modified Bessel functions of order n.

In summary, by skipping the details (mentioned in [1]), in the repetition operation, the output distribution $W[G_1] \cdot W[G_2]$ can be approximated for *low* message concentrations as

$$\begin{aligned} K_{vi} &\triangleq f^{-1}\left(\frac{K_{w,i}}{A^2}\right) \qquad i = 1,2 \\ K_{w,3} &= A^2 f\left(\sqrt{K_{v,1}^2 + K_{v,2}^2 + 2K_{v,1}K_{v,2}\cos(A(\mu_1 - \mu_2))}\right) \\ \mu_3 &= \frac{1}{A}\arctan\left(\frac{K_{v,1}\sin(A\mu_1) + K_{v,2}\sin(A\mu_2)}{K_{v,1}\cos(A\mu_1) + K_{v,2}\cos(A\mu_2)}\right) \end{aligned} \tag{7}$$

Similarly, a good approximation of the message $S[G_1] * S[G_2]$ for the sum operation, which is valid for *large* message concentrations can be obtained by exploiting the same correspondence, but in the transform domain, yielding:

$$\begin{aligned} K_{vi} &\triangleq f^{-1}\left(\frac{1}{K_{s,i}}\right), \quad \mu = (l_i + \alpha_i), \quad \gamma_i = \alpha_i . K_{vi}, \quad i = 1,2 \\ K_{s,3} &= 1/f\left(\sqrt{K_{v,1}^2 + K_{v,2}^2 + 2K_{v,1}K_{v,2}\cosh(\gamma_1 - \gamma_2)}\right) \\ 2\gamma_3 &= \log\left(\frac{K_{v,1}e^{\gamma_1} + K_{v,2}e^{\gamma_2}}{K_{v,1}e^{-\gamma_1} + K_{v,2}e^{-\gamma_2}}\right), \quad l_3 = l_1 + l_2 \end{aligned} \tag{8}$$

where the real quantity μ is expressed by separating its integer (l) and fractional part (α) $\in [-0.5, 0.5]$. The ADBP algorithm which uses exactly the updating rules (7)-(8) is named as *exact*-ADBP algorithm and denoted in the following with the acronym *eADBP*.

III. Simplifications of the updating equations.

Although ADBP algorithm introduces the fundamental complexity breakthrough of making the iterative decoding of non binary codes independent from the alphabet size M, equations (7) and (8) for updating the D-message parameters are still too complex for a hardware implementation. In this section we introduce some simplifications of the ADBP updating equations for the specific purpose of using it for decoding of non binary codes in a digital transmission system.

The considered full transmission system for high spectral efficiencies, shown in Fig.1, consists of an mod-M LDPC encoder, an M-PAM modulator, a *wrapped* AWGN channel and the *ADBP decoder*.

The encoder is a regular LDPC encoder with $K = 3000$ input M-ary symbols, and $N = 5000$ output symbols. The constant variable degree is $d_v = 2$ and the check node degree is $d_c = 5$.

The M-ary output symbols c of the encoder are transmitted using a M-PAM constellation, with a natural order mapping $x = c - (M-1)/2$. The outputs of the wrapped gaussian channel are obtained wrapping the output of a regular AWGN channel in the interval $[-M/2, M/2]$. Wrapping reduces the channel capacity but at the same time make it input symmetric, so that transmission of the all zero sequence can be assumed with the employed linear encoder[1].

Furthermore, the likelihoods at the output of the wrapped gaussian channel with M-PAM inputs take naturally the form of digital messages with parameters

$$\begin{aligned} \mu &= y + (M-1)/2 \bmod M \\ K &= \frac{1}{\sigma^2} = \frac{E_s}{N_0}\frac{12}{M^2-1}, \end{aligned}$$

so that it can be easily interfaced with ADBP.

The ADBP decoder takes as input messages the pairs (μ, K) and performs 10 iterations using the flooding schedule. The tanner graph of the code is reported in Fig.2.

A. Simplifications of the repetition update equations

As pointed out in the Fig. 2, at repetition nodes one of the involved messages is always the channel message. The standard deviation of this message, related to E_s/N_0

[1]The use of ADBP in conjunction with the regular AWGN channel, as well as with other types of modulation set requires some modifications of the computation of the input messages parameters. Since in this paper we focus primarily in the implementation issues, we decide to use the wrapped AWGN to simplify the analysis. Notice that the loss of the capacity induced by wrapping becomes negligible for large value of E_s/N_0.

over the channel, is typically much smaller than the wrapping period M. In this case the exact expression of ADBP (7) can be replaced by the simpler expression (2) that neglects the aliasing effect of replicas of the wrapped gaussian. Notice however that for the proper computation of the output mean in (2) one should consider, among all possible replicas associated to the wrapped gaussian distribution, those that have closest μ_i mod$M$. As a consequence the following approximation can be used to obtain the mean of $W[G1] \cdot W[G2]$.

$$\mu_3 \approx \frac{\bar{\mu}_1 K_{w,1} + \bar{\mu}_2 K_{w,2}}{K_{w,1} + K_{w,2}} \bmod M \tag{9}$$

Where $\overline{\mu}_1 = (\mu_1 + k_1 M)$ and $\overline{\mu}_2 = (\mu_2 + k_2 M)$, and the integers $k_1$and k_2 should be chosen so as to minimize $|\mu_1 - \mu_2|$.

Eq. (9) requires 2 multiplications, one sum and one division. In order to simplify it, we first derive the indexes associated to the maximum and minimum values of the concentrations.

$$L \triangleq \arg\max_i(K_{w,i}), \quad l \triangleq \arg\min_i(K_{w,i})$$

we then write (9) as

$$\mu_3 = \mu_L + A(\mu_l - \mu_L)_{-M/2}^{M/2} \bmod M \tag{10}$$

where

$$A \triangleq \frac{K_{w,l}}{K_{w,1} + K_{w,2}},$$

and the notation $(\mu_l - \mu_L)_{-M/2}^{M/2}$ means that the difference $\mu_l - \mu_L$ should be taken mod M in the range $[-M/2, M/2]$.

Expression (10) only requires two sums and the multiplication by the number A, which is always bounded in $[0, 1]$.

B. Simplifications of sum update equations

In sum update the use of the correct expression (8) instead of (1) is required as the concentration of messages actually increases during iterations. In this case the simplification of (8) is obtained by considering the following approximation for the function $f(x)$

$$f(x) \approx -(2\log(x/2))^{-1}$$

which is valid for $x < 1$.

Using this approximation we can write the concentration of the characteristic functions as

$$K_{v,i} = 2\exp(-K_{s,i}/2)$$

and approximate the output message concentration as follows:

$$\begin{aligned} K_{s,3} &= -2\log\left(\frac{1}{2}\sqrt{K_{v,1}^2 + K_{v,2}^2 + 2K_{v,1}K_{v,2}\cosh(\gamma_1 - \gamma_2)}\right) \\ &= -\log\left(\frac{1}{4}\left[K_{v,1}^2 + K_{v,2}^2 + 2K_{v,1}K_{v,2}\cosh(\gamma_1 - \gamma_2)\right]\right) \\ &= -\log\left(e^{-K_{s,1}} + e^{-K_{s,2}} + 2e^{-\frac{K_{s,1}+K_{s,2}}{2}}\cosh(\gamma_1 - \gamma_2)\right) \\ &= \min^*\left(K_{s,1}, K_{s,2}, \frac{K_{s,1} + K_{s,2}}{2} - |\gamma_1 - \gamma_2|^*\right) \end{aligned} \tag{11}$$

where we introduced the familiar operator $\max^*(a,b) = \log(e^a + e^b) = \max(a,b) + \log(1 + e^{-|a-b|})$ and the derived operators $\min^*(a,b) = -\max^*(-a,-b)$ and $|x|^* = \max^*(x,-x)$.

Similarly we can derive the following expression for the output mean (3rd equation in (8))

$$\begin{aligned} 2\gamma_3 &= \log\left(\frac{K_{v,1}e^{\gamma_1} + K_{v,2}e^{\gamma_2}}{K_{v,1}e^{-\gamma_1} + K_{v,2}e^{-\gamma_2}}\right) \\ &= \max^*\left(-\frac{K_{s,1}}{2} + \gamma_1, -\frac{K_{s,2}}{2} + \gamma_2\right) \\ &\quad -\max^*\left(-\frac{K_{s,1}}{2} - \gamma_1, -\frac{K_{s,2}}{2} - \gamma_2\right) \end{aligned} \tag{12}$$

A further simplification to the $\min^*$ that neglects the correction term $\log(1 + e^{-|a-b|})$ and obtains the *true min* is considered. The ADBP algorithm that uses the updating rules of (10), (11) and (12) with *true min* approximation is named as the *simplified-ADBP* and denoted by the acronym *sADBP*.

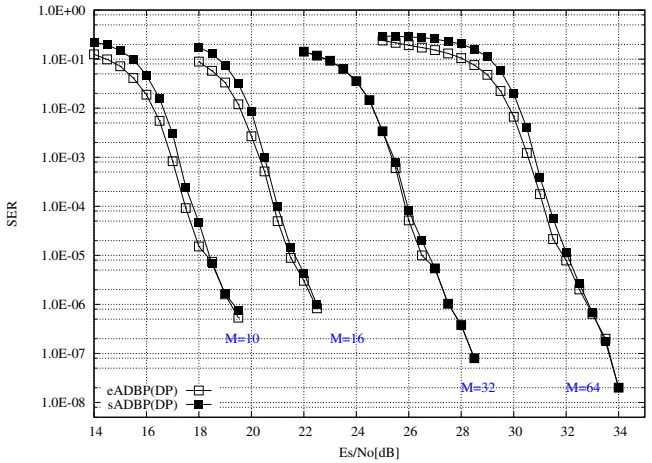
IV. Fixed Point Model

The fixed point (FP) model of proposed algorithm is implemented in C language. All variables in the decoding algorithm are represented in 2'*s* compliment notation as $[I, F]$ with I bits for integer part (which includes the sign bit unless stated otherwise) and F bits for fractional part. The performance of the decoder is expressed as symbol error rate (SER) as a function of signal to noise ratio (E_s/N_o).

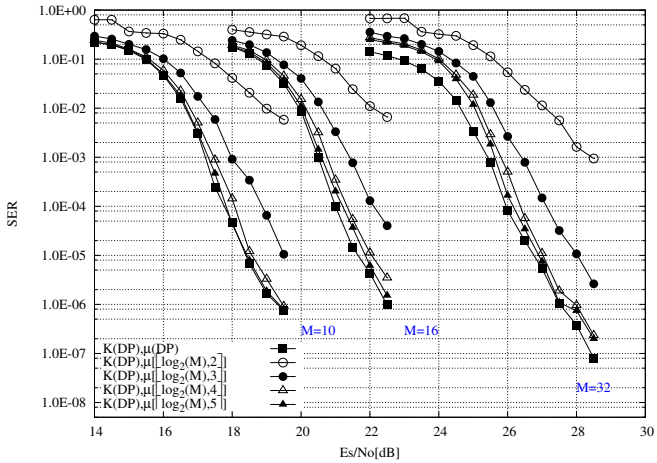
At first, the approximations used in Eq.(10)-(12) are validated by simulating in double precision (DP), both *eADBP* and *sADBP* algorithms. Fig. 3.a shows the results obtained from this step for $M = 10, 16, 32$ and 64. For all the four cases of M reported in the figure, the simulation results show that the *sADBP* algorithm performs close to the *eADBP* algorithm and therefore, can be safely investigated for FP implementation.

The initial step of FP implementation of *sADBP* decoder is the determination of appropriate number of I and F bits for input quantities (i.e. μ and K). This acts as a starting point to implement the internal datapath of decoder keeping in view the performance and complexity constraints. To achieve this, we analyze the DP performance offered by the $sADBP$ algorithm when quantized inputs are applied. This analysis is carried out in two steps.

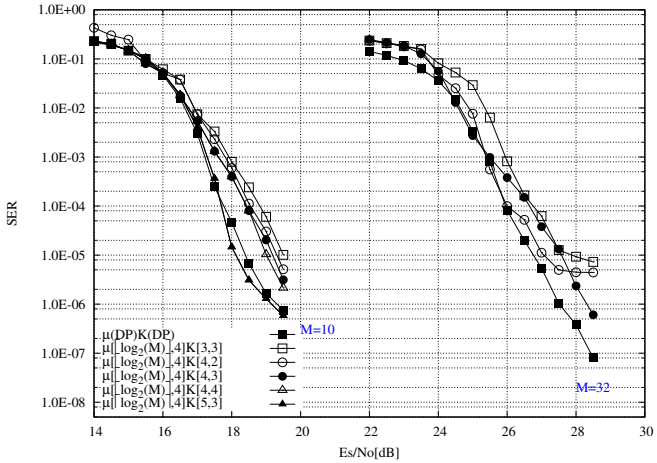
Quantization of μ only: In this step, only the μ is quantized as $[I, F]$ and applied to the input of sum and repetition functional units whereas, the K and internal data path is in DP. As discussed in section II, the μ of D-messages with wrapping period M is a real number that lies in the range $[0, M[$. Therefore, the 2'*s* complement representation of integer part of μ requires $\lceil \log_2 M \rceil$ bits for the magnitude and 1 bit for the sign. Since the input μ is always positive, most significant sign bit is always '0' and neglected. Whereas, the number of fractional bits of μ is a design parameter that could be changed to



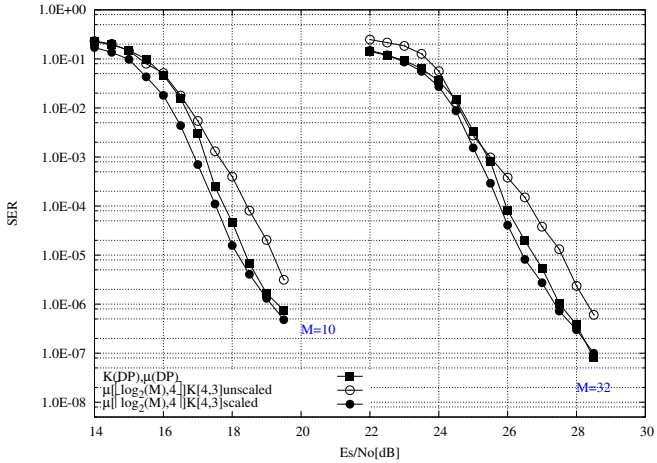
(a) DP performance of *eADBP* and *sADBP* algorithms



(b) DP performance of *sADBP* algorithm with quantized μ only



(c) DP performance of *sADBP* with quantized μ and K (unscaled)



(d) Effect of K scaling on DP performance of $sADBP$ algorithm

Figure 3: ADBP Simulation Results (a) DP performance of *eADBP* and *sADBP* algorithms (b) (c) (d) Effects of input quantizations on the DP implementation of *sADBP* algorithm.

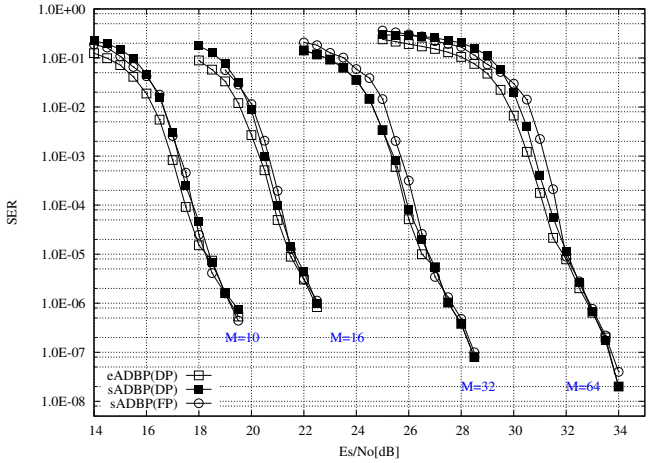


Figure 4: Complete FP simulation results for sADPB.

trade off performance with hardware complexity. The simulation results of this step are plotted in Fig. 3.b, which shows the *sADBP* performance with quantized μ with fixed ($\lceil \log_2 M \rceil$) integer and variable (2 to 5) fractional bits. For the three cases of M shown in Fig. 3.b, the simulation results demonstrate that at least 4 fractional bits are required for μ in order to obtain performance within 0.3 dB to the DP implementation.

Quantization of μ and K: In this step, both inputs i.e. μ and K are quantized as $[I, F]$ and applied to functional units. μ is quantized as obtained from the above step whereas, for K we simulate various combinations of integer and fractional bits. The simulation results of this step are shown in Fig. 3.c. Since the magnitude of K increases with E_s/N_o, the schemes with large I and $F \geq 3$ are required to achieve low *SER* at high E_s/N_o. For example, $K[5,3]$ shows superior results for both $M = 10$ and 32 while $K[4,3]$ also shows comparable results with fewer number of bits. From the simulation results, we conceived that considerable improvement in *SER* performance could be achieved at high E_s/N_o by first scaling down the magnitude of K before applying

to the input of functional units. Whereas, the updated K at the output of the units is scaled up by the same factor. This scaling helps to represent a large value of K with lower number of integer bits and avoids overflow during intermediate calculations thus achieving better numerical stability. In addition, it is also conceived that scaling factor of 16 when used with [4, 3] quantization scheme provides a good performance-complexity trade off. The results of this step are shown in Fig. 3.d for $M = 10$ and 32 which shows significant improvement in *SER* performance of *sADBP* algorithm achieved with the help of K scaling. It may be noted that the same [4, 3] quantization for K and scaling factor of 16 is also applicable for $M = 16$ and 64.

In the next step, the internal datapath of decoder is optimized for FP implementation using the quantized inputs (μ and K). The signals involved at each intermediate step of computation are optimized by simulating various combinations of integer and fractional bits. Figure 4 shows the complete FP simulation results of the proposed decoder. The curves in the figure show that for each case of M, the FP implementation of *sADBP* performs very close to the DP implementations of *sADBP* and *eADBP* algorithm. The hardware architecture and datapath complexity details specific to the FP simulation curves of Fig. 4 are discussed in the next section.

V. Hardware architecture and Synthesis Results.

The ADBP decoding architecture is similar to standard non binary (NB) LDPC decoders. As discussed in Sec. III and shown in Fig. 2, the Tanner graph for ADBP consists of two kinds of computation nodes i.e. sum nodes and repetition nodes that operate on d_c and d_v messages respectively. The decoding process involves a flooding schedule in which a single decoding iteration consists of two steps: a) Horizontal scan: in which all sum nodes update their output messages and b) Vertical scan: in which all repetition nodes update their output messages. As discussed before, a single message in ADBP decoding is a vector containing two values i.e. μ and K of the corresponding message class.

In this section, we will first discuss the digital architecture for binary (i.e. two input) sum and repetition nodes and then we will propose a generalized processing element (PE) that implements the extension of these operations over $d_v(d_c)$ messages.

Binary Repetition Node

The repetition node implements the function $W[G1].W[G2]$ using simplified version (10). This equation is implemented as shown by the steps in Algorithm 1.

Fig. 5.A shows the datapath architecture of binary repetition node functional unit (RN-FU). CMP1-CMP3 are binary comparators, A1-A4 are 2's complement binary adders, M1-M2 are binary multipliers and Mux1-4 are the 2 input multiplexers. The figure also shows

Algorithm 1 Repetition Node Update for *sADBP* Algorithm

Inputs: $K_{w,i}$, $\mu_i = (l_i + \alpha_i)$, $i = 1, 2$
Outputs: $K_{w,3}$, μ_3
Scaling: $K_{x,i} = \frac{K_{w,i}}{16}$, $i = 1, 2$
$K_{w,3}$**-Computation:**
$K_{x3} = K_{x1} + K_{x2}$, $K_{w,3} = K_{x3} \times 16$
μ_3**-Computation:**
if $(K_{x1} > K_{x2})$ **then**
  $\mu_{t1} = \mu_2 - \mu_1$, $K_t = K_{x2}$, $\mu_{t2} = \mu_1$
else
  $\mu_{t1} = \mu_1 - \mu_2$, $K_t = K_{x1}$, $\mu_{t2} = \mu_2$
end if
if $(\mu_{t1} > \frac{M}{2})$ **then**
  $\mu_{t3} = \mu_{t1} - M$, $\mu_{t4} = \mu_{t3}$
else if $(\mu_{t1} < -\frac{M}{2})$ **then**
  $\mu_{t3} = \mu_{t1} + M$, $\mu_{t4} = \mu_{t3}$
else
  $\mu_{t4} = \mu_{t1}$
end if
$v_3 = \frac{1}{K_{x3}}$, $a1 = K_t \times v_3$
$a2 = a1 \times \mu_{t4}$
$\mu_3 = \mu_t 2 + a2$

the input, output and intermediate variables of Listing 1. μ_i and K_{xi}, $i = 1, 2$ are the input mean and scaled concentrations respectively. In FP implementation, the *wrapped message* concentration $K_{w,i}$ at the input of the node is represented in $[I, F]$ form as $[8, 0]$ which is scaled down by 16 by simply moving the decimal point to left by 4 places i.e. $K_{xi} = [4, 4]$. Reverse is true for scaling up by 16 at the output of the node. In order to decrease the hardware complexity and propagation delay, we adopted *division via multiplication by reciprocal* technique where the quantity $v_3 = \frac{1}{K_{x3}}$ is implemented using a look up table (LUT). The repetition node is implemented as a fully pipelined structure with pipelining depth $L = 3$ (shown in dotted lines in Fig.5.A).

Binary Sum Node

The binary sum node implements the function $S[G1] * S[G2]$ using (11) and (12) and the main computational steps are shown in Algorithm 2.

The hardware architecture reported in Fig.5.b consists of binary comparators Cmp1-Cmp5 that compute the *true min* of two inputs, 2's complement adders A1-A16, multipliers M1-M3 and a LUT to implement $v_3 = \frac{1}{K_{x2}}$. The sum node is also a fully pipelined structure with $L = 3$ pipeline stages.

It may be noted that although the generalized architecture for both sum and repetition node remains the same for all values of M, the complexity as well as decoding performance is dependent upon the $[I, F]$ representation of all variables. For the FP simulation results shown in Fig. 4, the binary sum and repetition nodes datapath

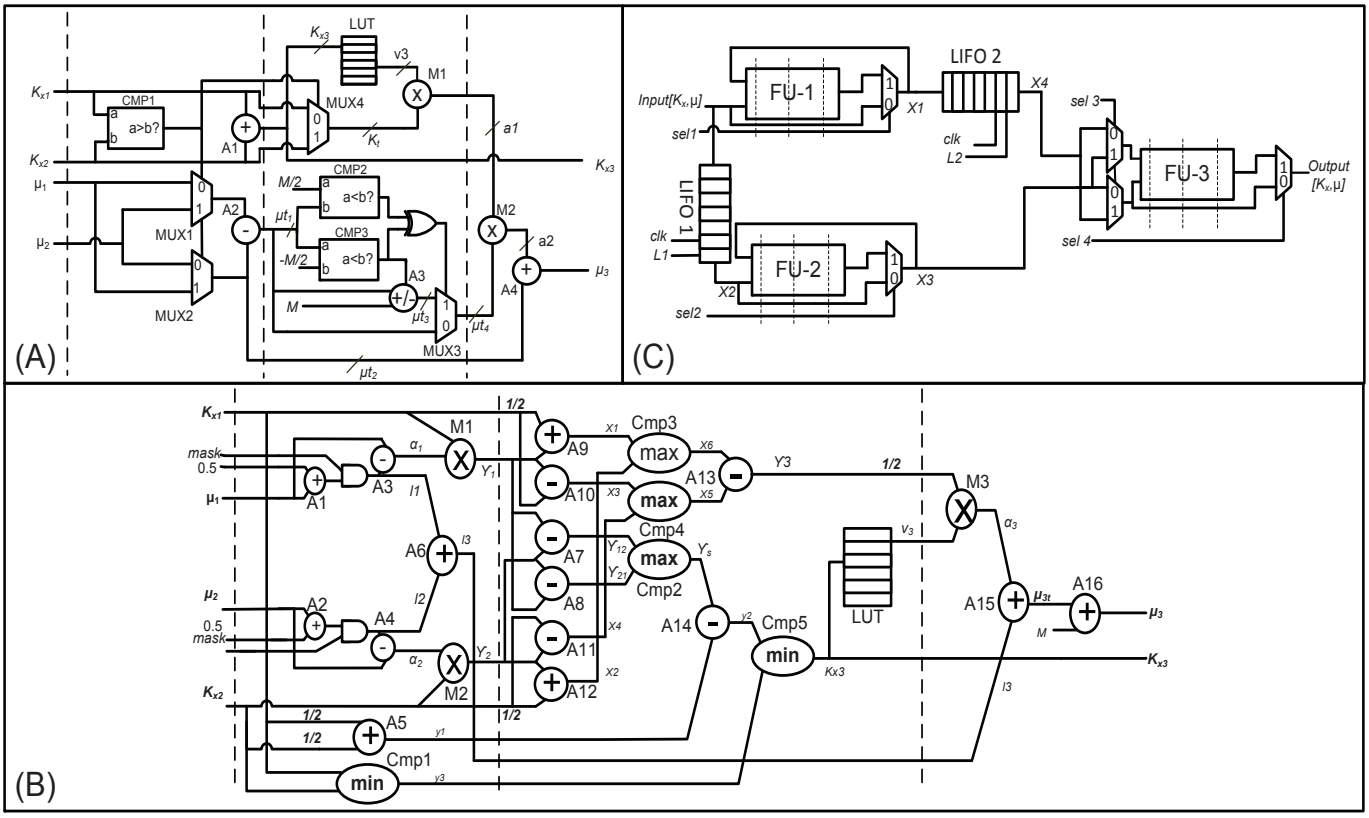


Figure 5: ADBP Hardware Architecture (A) Binary Repetition Node (B) Binary Sum Node (C) Forward Back Processing Element with serial read/write.

Algorithm 2 Sum Node Update for *sADBP* Algorithm

Inputs: $K_{s,i},\ \mu_i = (l_i + \alpha_i),\ i = 1, 2$
Outputs: $K_{s,3},\ \mu_3$
Scaling: $K_{xi} = \frac{K_{s,i}}{16},\ i = 1, 2$
$K_{s,3}$**-Computation:**
for $i = 1$ to 2 **do**
  $l_i = \lfloor \mu_i + 0.5 \rfloor,\ \alpha_i = \mu_i - l_i,\ \gamma_i = \alpha_i \times K_{xi}$
end for
$\gamma_{12} = \gamma_1 - \gamma_2,\ \gamma_{21} = \gamma_2 - \gamma_1$
$\gamma_s = \max(\gamma_{12}, \gamma_{21})$
$y1 = \frac{K_{x1}}{2} + \frac{K_{x2}}{2}$
$y3 = \min(K_{x1}, K_{x2})$
$y2 = y1 - \gamma_s$
$K_{x3} = \min(y3, y2),\ K_{s,3} = 16 \times K_{x3}$
μ_3**-Computation:**
$x1 = \gamma_1 + \frac{K_{x1}}{2},\ x2 = \gamma_2 + \frac{K_{x2}}{2}$
$x3 = \gamma_1 - \frac{K_{x1}}{2},\ x4 = \gamma_2 - \frac{K_{x2}}{2}$
$x5 = \max(x3, x4)\ ,\ x6 = \min(x1, x2)$
$\gamma_3 = \frac{1}{2}\{x5 - x6\},\ v_3 = \frac{1}{K_{x3}}$
$\alpha_3 = v_3 \times \gamma_3,\ l_3 = l_1 + l_2$
$\mu_{3t} = l_3 + \alpha_3$
$\mu_3 = (\mu_{3t} + M) \mod M$

detail is given in Tab. I.a and I.b. The left most column of both tables shows all computation steps involved in the updates of Algorithm 1 and 2 whereas, the next columns show for $M = 10$ to 64, the quantization detail $[I, F]$ of operands as well as of the result of each step. The quantities $\mu_i, K_{w,i}, K_{s,i}, K_{xi}, l_i : i = 1, 2, 3$ are always positive therefore, MSB of their magnitude part is always '0' and hence excluded from $[I, F]$ representation. For all other quantities the $[I, F]$ includes the sign bit. In some cases where the operands have unequal number of I and F bits; arithmatic shift left(right) operation is performed on one or both of them in order to allign their decimal points. In addition, the tables also show the hardware modules (of Fig. 5.A and 5.B) involved in each step along with their complexity in terms of bitwidth.

The details of Tab. I.a and I.b reveal that moving from $M = 10$ to 64 requires a 1 bit increase in the bitwidth of those hardware modules which involve μ_i in computation. This results in a slight but affordable increase in overall complexity of sum and repetition functional nodes.

Processing Element

The sum and repetition functions discussed above are associative i.e. for three inputs I_1, I_2, I_3

$$\boxplus(I_1, I_2, I_3) = \boxplus(\boxplus(I_1, I_2), I_3)$$

Operation / Signals	Quantization Details $[I,F]$									Hardware Modules	Bitwidth		
	M=64			M=16			M=10,16				M=64	M=32	M=10,16
	Op.1	Op.2	Result	Op.1	Op.2	Result	Op.1	Op.2	Result				
μ_1, μ_2	[6.4]			[5.4]			[4.4]						
$K_{s,1}, K_{s,2}, K_{s,3}$	[8,0]			[8,0]			[8,0]						
K_{x1}, K_{x2}	[4,3]			[4,3]			[4,3]						
$l_i = int(\mu_i + 0.5), i = 1,2$	[6,4]	[6,4]	[7,0]	[5,4]	[5,4]	[6,0]	[4,4]	[4,4]	[5,0]	A1,A2	10	9	8
$\alpha_i = \mu_i - l_i,\ i = 1,2$	[6,4]	[7,0]	[1,4]	[5,4]	[6,0]	[1,4]	[4,4]	[5,0]	[1,4]	A3,A4	10	9	8
$\gamma_i = \alpha_i \times K_{xi}, i = 1,2$	[1,4]	[4,3]	[2,7]	[1,4]	[4,3]	[2,7]	[1,4]	[4,3]	[2,5]	M1,M2	7	7	7
$\gamma_{12} = \gamma_1 - \gamma_2$	[2,7]	[2,7]	[3,6]	[2,7]	[2,7]	[3,6]	[2,5]	[2,5]	[3,5]	A7	9	9	7
$\gamma_{21} = \gamma_2 - \gamma_1$	[2,7]	[2,7]	[3,6]	[2,7]	[2,7]	[3,6]	[2,5]	[2,5]	[3,5]	A8	9	9	7
$\gamma_s = max(\gamma_{12}, \gamma_{21})$	[3,6]	[3,6]	[3,6]	[3,6]	[3,6]	[3,6]	[3,5]	[3,5]	[3,5]	Cmp2	9	9	8
$\frac{K_{xi}}{2},\ i = 1,2$	[4,3]		[4,3]	[4,3]		[4,3]	[4,3]		[4,3]				
$y1 = \frac{K_{x1}}{2} + \frac{K_{x2}}{2}$	[4,3]	[4,3]	[4,3]	[4,3]	[4,3]	[4,3]	[4,3]	[4,3]	[4,3]	A5	7	7	7
$y3 = min(K_{x1}, K_{x2})$	[4,3]	[4,3]	[4,3]	[4,3]	[4,3]	[4,3]	[4,3]	[4,3]	[4,3]	Cmp1	7	7	7
$y2 = y1 - \gamma_s$	[4,3]	[3,6]	[4,3]	[4,3]	[3,6]	[4,3]	[3,5]	[3,5]	[4,3]	A14	9	9	8
$K_{x3} = min(y1, y2)$	[4,3]	[4,3]	[4,3]	[4,3]	[4,3]	[4,3]	[4,3]	[4,3]	[4,3]	Cmp5	7	7	7
$xi = \gamma_i + \frac{K_{xi}}{2}, i = 1,2$	[2,7]	[4,3]	[5,3]	[2,7]	[4,3]	[5,3]	[2,5]	[4,3]	[4,3]	A9,A12	7	7	7
$x3 = \gamma_1 - \frac{K_{x1}}{2}$	[2,7]	[4,3]	[5,3]	[2,7]	[4,3]	[5,3]	[2,5]	[4,3]	[4,3]	A10	7	7	7
$x4 = \gamma_2 - \frac{K_{x2}}{2}$	[2,7]	[4,3]	[5,3]	[2,7]	[4,3]	[5,3]	[2,7]	[4,3]	[4,3]	A11	7	7	7
$x5 = max(x3, x4)$	[5,3]	[5,3]	[5,3]	[5,3]	[5,3]	[5,3]	[4,3]	[4,3]	[4,3]	Cmp4	8	8	7
$x6 = min(x1, x2)$	[5,3]	[5,3]	[5,3]	[5,3]	[5,3]	[5,3]	[4,3]	[4,3]	[4,3]	Cmp3	8	8	7
$\gamma_3 = \frac{(x5 - x6)}{2}$	[5,3]	[5,3]	[5,3]	[5,3]	[5,3]	[5,3]	[4,3]	[4,3]	[4,3]	A13	8	8	7
$v_3 = \frac{1}{K_{x3}}$	[2,4]		[2,4]	[2,4]		[2,4]	[2,4]		[2,4]	LUT	128x6 bit		
$a3 = v3 \times \gamma_3$	[2,4]	[5,3]	[7,4]	[2,4]	[5,3]	[7,4]	[2,4]	[4,3]	[3,4]	M3	8	8	7
$l3 = l1 + l2$	[7,0]	[7,0]	[8,0]	[6,0]	[6,0]	[7,0]	[5,0]	[5,0]	[6,0]	A6	7	6	5
$\mu_{3t} = l3 + a3$	[8, 0]	[1,4]	[8,4]	[7,0]	[1,4]	[7,4]	[6,0]	[3,4]	[6,4]	A15	8	7	6
$\mu_3 = (\mu_{3t} + M) \mod M$	[8, 4]	[6,4]	[6,4]	[7,4]		[5,4]	[6,4]	[4,4]	[4,4]	A16	8	7	6

(a) Sum Node Functional Unit

Operation / Signals	Quantization Details $[I,F]$									Hardware Modules	Bitwidth		
	M=64			M=32			M=10,16				M=64	M=32	M=10,16
	Op.1	Op.2	Result	Op.1	Op.2	Result	Op.1	Op.2	Result				
μ_1, μ_2	[6.4]			[5.4]			[4.4]						
$K_{w,1}, K_{w,2}, K_{w,3}$	[8,0]			[8,0]			[8,0]						
K_{x1}, K_{x2}	[4,3]			[4,3]			[4,3]						
$K_{x3} = K_{x1} + K_{x2}$	[4,3]	[4,3]	[4,3]	[4,3]	[4,3]	[4,3]	[4,3]	[4,3]	[4,3]	A1	7	7	7
$if(K_{x1} > K_{x2})$	[4,3]	[4,3]	1	[4,3]	[4,3]	1	[4,3]	[4,3]	1	CMP1	7	7	7
$\mu_{t1} = \mu_2 - \mu_1$ or $\mu_{t1} = \mu_1 - \mu_2$	[6,4]	[6,4]	[7,3]	[5,4]	[5,4]	[6,3]	[4,4]	[4,4]	[5,3]	A2	10	9	8
										MUX1	10	9	8
$K_t = K_{x1}$or K_{x2}	[4,3]		[4,3]	[4,3]		[4,3]	[4,3]		[4,3]	MUX4	7	7	7
$\mu_{t2} = \mu_1$or μ_2	[6,4]		[6,4]	[5,4]		[5,4]	[4,4]	[4,4]	[4,4]	MUX2	10	9	8
$if(\mu_{t1} > \frac{M}{2})$	[7,3]	[7,3]	1	[6,3]	[6,3]	1	[5,3]	[5,3]	1	CMP2	10	9	8
$elseif(\mu_{t1} < -\frac{M}{2})$	[7,3]	[7,3]	1	[6,3]	[6,3]	1	[5,3]	[5,3]	1	CMP3	10	9	8
$\mu_{t3} = \mu_{t1} \pm M$	[7,3]	[7,3]	[7,3]	[6,3]	[6,3]	[6,3]	[5,3]	[5,3]	[5,3]	A3	10	9	8
$\mu_{t4} = \mu_{t3}$or μ_{t1}	[7,3]		[7,3]	[6,3]		[6,3]	[5,3]		[5,3]	MUX3	10	9	8
$v_3 = \frac{1}{K_{x3}}$	[0,6]		[0,6]	[0,6]		[0,6]	[0,6]		[0,6]	LUT	128x6 bits		
$a1 = K_t \times v_3$	[4,3]	[0,6]	[0,7]	[4,3]	[0,6]	[0,7]	[4,3]	[0,6]	[0,7]	M1	7	7	7
$a2 = a1 \times \mu_{t4}$	[0,7]	[5,3]	[5,3]	[0,7]	[6,3]	[4,3]	[0,7]	[5,3]	[4,3]	M2	10	9	8
$\mu_3 = \mu_{t2} + a2$	[6,4]	[5,3]	[6,4]	[5,4]	[4,3]	[5,4]	[4,4]	[4,3]	[4,4]	A4	10	9	8

(b) Repetition Node Functional Unit

Table I: Sum and repetition functional nodes complexity detail specific to FP performance curves of Fig. 4

Where $\boxplus$ operator denotes the binary sum or repetition function. In order to extend these functions to process more than two input messages, the following update rule must be satisfied

$$I_{y,i} = \boxplus_{j \neq i}^{n}(I_{x,j}) \tag{13}$$

$I_{x,j}(I_{y,i})$ denote the input(output) $[\mu, K]$ message vectors and n denotes the node degree (d_v or d_c). The above equation states that the output message corresponding to edge i of a node (sum or repetition) is the result of $\boxplus$ operator over all input messages except the input message i. The update rule of (13) is implemented in this work by adopting a forward-backward (FB) strategy with serial read and write. Fig.5.c. shows the generalized architecture of proposed FB processing element (PE) which consists of three functional units (FUs) that implement the $\boxplus$ operator and two last in first out (LIFO) memory units which hold the intermediate results. The functionality of PE explained here in context of CN processing holds equally applicable for VN processing.

Thanks to the pipelined implementation of FUs, the PE is able to process one edge of Tanner graph per clock cycle. The number of parity check equations in pipeline is $L = 3$. In the first L clock cycles PE receives the first message of L parity equations, in the next L cycles it receives the second message of L equations and so on. This process continues until all d_c messages of L parity check equations have been received. At the output, the messages are produced in reverse order i.e. starting from message number d_c of equation number L up to message number 1 of equation number 1.

The width (i.e. number of bits per row) of LIFO1 and LIFO2 is equal to sum of total number of bits for FP representation of μ and K , whereas the depth (i.e. number of rows) is given as

$$d_{Lifo} = (d_c - 1)L \tag{14}$$

Both sum and repetition PE's have a throughput of one edge per clock cycle. Therefore, if the clock frequency is C, the PE's for both sum and repetition operator will provide the full set of updated edges $E = N\eta$ in $2N_{it}(Nd_v + L^{'})/C$ seconds, where $L^{'}$ is an additional constant that takes into account the latency of the two processors.

More generally, with P parallel processors the throughput is given as

$$T = C\frac{PR_c \log M}{2N_{it}(d_v + P\frac{L^{'}}{N})} \backsimeq C\frac{PR_c \log_2 M}{2N_{it}d_vP} \tag{15}$$

Where R_c denotes the code rate and d_v is the average variable degree of LDPC code, N_{it} is the number of decoding iterations.

The synthesizable IP core of the ADBP decoder is written in VHDL and synthesis is performed on 45nm standard cell ASIC technology using Synopsys Design Vision tool at a target clock rate $C = 300$ MHz.

M	n	A_{Sum} [μm^2]	A_{Rep} [μm^2]	d_{Lifo}	A_{Lifo} [μm^2]	A_{CN} [mm^2]	A_{VN} [mm^2]	T.P Mbps	G.C CN	G.C VN
10,16	4	2340	1710	9	1600	0.0102	0.0083	18	9.6K	7.8K
	5	2340	1710	12	1900	0.0108	0.0090		10.2K	8.5K
	6	2340	1710	15	2200	0.0113	0.0096		10.6K	9.1K
	8	2340	1710	21	2800	0.0126	0.0099		11.9K	9.3K
32	4	2530	2060	9	1700	0.0110	0.0093	22.5	10.3K	8.8K
	5	2530	2060	12	2000	0.0115	0.0100		10.8K	9.4K
	6	2530	2060	15	2300	0.0121	0.0107		11.4K	10.1K
	8	2530	2060	21	2900	0.0134	0.0109		12.6K	10.3K
64	4	3010	2530	9	1900	0.0128	0.0114	27	12.1K	10.7K
	5	3010	2530	12	2100	0.0132	0.0118		12.5K	11.1K
	6	3010	2530	15	2500	0.0140	0.0126		13.2K	11.9K
	8	3010	2530	21	3200	0.0154	0.0139		14.5K	13.1K

Table II: Synthesis Results of ADBP processing nodes.

Table II shows the synthesis results of the main processing elements of ADBP algorithm for various values of M. Following notations are adopted in table to represent the area figures.

- A_{Sum} and A_{rep} denote respectively the area in μm^2 of a two input sum and repetition node FU.
- A_{Lifo} denote the area in μm^2 of a single LIFO for a given node degree n.
- A_{CN} and A_{VN} denote respectively the area in $mm^2$of a single sum and repetition PE of degree n.

The table also reports the gate count ($G.C$) i.e. total number of 2-input Nand gates for both CN and VN PEs for various values of M and node degrees n. The throughput for the whole decoder is also reported in table which is calculated using (15) for full serial architecture i.e. $P = 1$ with $R_c = \frac{3}{5}$ and $d_v = 2$.

The exact comparison in terms of performance and hardware complexity of complete ADBP decoder with the state of the art NB LDPC decoders is difficult due to different design choices e.g. cardinality of Galois fields, CMOS technology process, operating frequency, parallelism, different block lengths, code rates and variable (check) node degree distributions. However, the complexity comparison at the PE level is possible and presented here. One recent work in the domain of non binary LDPC decoders is [13] in which the authors propose an M=32 decoder with Trellis based implementation of forward backward check node. The main processing core of [13] consists of an iterative decoder processor (IDP) which implements the combined functionality of a single CN with $d_c = 27$ and VN with $d_v = 4$. The IDP is synthesized on 90 nm technology and has a gate count $G.C$ of 5 Million gates. In case of ADBP, the combined area of sum and repetition PE's of degrees $d_c = 27$ and $d_v = 4$ respectively is 0.038 mm^2 at 45nm. This area is multiplied by 4 to obtain equivalent area at 90nm technology. Finaly, the result is divided by the area of a single 2-input nand gate to obtain the $G.C$ at 90nm which is 0.06 Million gates. This clearly demonstrates the logic area advantage of ADBP over the exisiting NB decoders. In addition, the non binary LDPC decoder of [13] achieves a throughput of 234 Mbit/s with parameters N_{it}, C, K and R_c equal to 5, 250MHz, 726 and 0.86 respectively. For the same parameters, the proposed ADBP decoder is able to achieve a throughput of 833Mbits/sec

which is almost 3.5 times higher than the decoder in [13]. However, the simple regular mod-M encoder used in this paper actually does not provides performances competitive with those of LDPC encoders constructed on fields. The exceptional complexity reduction achieved from using the ADBP however motivates for further research effort in the design of good LDPC encoders within the class.

The synthesis results of Tab. II clearly demonstrate that the ADPB algorithm achieves a comparable throughput with affordable complexity. In addition the complexity increases very slightly moving to higher cardinalities which shows feasibility of this algorithm for high values of M.

VI. Conclusions.

After almost two decades of research, binary LDPC codes have gained a wide diffusion in several fields and excellent implementations exist for their decoding. However, the problem of developing efficient decoders for non binary LDPC codes is far from being solved. In particular, the implementation complexity of non binary LDPC decoders tends to grow rapidly with the cardinality of the symbol alphabet, which severely limits the achievable spectral efficiency. In the first part of this paper, we introduce several simplifications in the previously proposed Analog Digital Belief Propagation algorithm. Provided simulation results show that these approximations do not affect significantly decoding performance, but enable the practical implementation of ADBP. In the second part of the paper, the fixed point model of ADBP is developed, showing that between 5 and 10 bits must be allocated to represent external and internal quantities with limited or null effect on performance. Finally, in the last part of the work, we propose and detail the implementation architecture of key processing nodes. Synthesis results obtained for multiple sizes of the symbol alphabet prove that: (i) the required area to implement processing nodes is affordable, and (ii) the complexity grows very weakly with the size of the alphabet.

References

[1] G. Montorsi, "Analog digital belief propagation," *Communications Letters, IEEE*, vol. 16, no. 7, pp. 1106–1109, 2012.

[2] M. C. Davey and D. J. MacKay, "Low density parity check codes over GF (q)," in *Information Theory Workshop, 1998*. IEEE, 1998, pp. 70–71.

[3] H. Wymeersch, H. Steendam, and M. Moeneclaey, "Log-domain decoding of LDPC codes over GF(q)," in *Communications, 2004 IEEE International Conference on*, vol. 2, 2004, pp. 772–776 Vol.2.

[4] D. Declercq and M. Fossorier, "Decoding Algorithms for Nonbinary LDPC Codes Over GF(q)," *Communications, IEEE Transactions on*, vol. 55, no. 4, pp. 633–643, 2007.

[5] A. Voicila, F. Verdier, D. Declercq, M. Fossorier, and P. Urard, "Architecture of a low-complexity non-binary LDPC decoder for high order fields," in *Communications and Information Technologies, 2007. ISCIT '07. International Symposium on*, 2007, pp. 1201–1206.

[6] V. Savin, "Min-Max decoding for non binary LDPC codes," in *Information Theory, 2008. ISIT 2008. IEEE International Symposium on*, 2008, pp. 960–964.

[7] C. Spagnol, E. Popovici, and W. Marnane, "Hardware Implementation of GF(q) LDPC Decoders," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 56, no. 12, pp. 2609–2620, 2009.

[8] J. Lin, J. Sha, Z. Wang, and L. Li, "An Efficient VLSI Architecture for Nonbinary LDPC Decoders," *Circuits and Systems II: Express Briefs, IEEE Transactions on*, vol. 57, no. 1, pp. 51–55, 2010.

[9] C. Zhang and K. Parhi, "A Network-Efficient Nonbinary QC-LDPC Decoder Architecture," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 59, no. 6, pp. 1359–1371, 2012.

[10] X. Chen, S. Lin, and V. Akella, "Efficient configurable decoder architecture for nonbinary Quasi-Cyclic LDPC codes," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 59, no. 1, pp. 188–197, 2012.

[11] X. Chen and C.-L. Wang, "High-Throughput Efficient Non-Binary LDPC Decoder Based on the Simplified Min-Sum Algorithm," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 59, no. 11, pp. 2784–2794, 2012.

[12] Y.-L. Ueng, C.-Y. Leong, C.-J. Yang, C.-C. Cheng, K.-H. Liao, and S.-W. Chen, "An Efficient Layered Decoding Architecture for Nonbinary QC-LDPC Codes," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 59, no. 2, pp. 385–398, 2012.

[13] Y.-L. Ueng, K.-H. Liao, H.-C. Chou, and C.-J. Yang, "A High-Throughput Trellis-Based Layered Decoding Architecture for Non-Binary LDPC Codes Using Max-Log-QSPA," *Signal Processing, IEEE Transactions on*, vol. 61, no. 11, pp. 2940–2951, 2013.

[14] T. P. Minka, "Expectation propagation for approximate bayesian inference," in *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 2001, pp. 362–369.

[15] U. Erez and G. Miller, "The ML decoding performance of LDPC ensembles over Z_q," *Information Theory, IEEE Transactions on*, vol. 51, no. 5, pp. 1871–1879, 2005.

[16] A. Bennatan and D. Burshtein, "On the application of LDPC codes to arbitrary discrete-memoryless channels," *Information Theory, IEEE Transactions on*, vol. 50, no. 3, pp. 417–438, 2004.

[17] G. Montorsi, "Analog digital belief propagation: from theory to practice," in *Communications (ICC), 2012 IEEE International Conference on*. IEEE, 2012, pp. 2591–2595.

[18] H. Loeliger, J. Dauwels, J. Hu, S. Korl, L. Ping, and F. Kschischang, "The factor graph approach to model-based signal processing," *Proc. IEEE*, vol. 95, no. 6, pp. 1295–1322, 2007.