PIPELINED BLOCK-SERIAL DECODER ARCHITECTURE FOR STRUCTURED LDPC CODES

Tejas Bhatt, Vishwas Sundaramurthy, Victor Stolpman and Dennis McCain
Nokia Research Center, Irving, TX, USA
{tejas.bhatt, vishwas.sundaramurthy, victor.stolpman, dennis.mccain}@nokia.com

ABSTRACT

We present a pipelined block-serial decoder architecture for structured LDPC Codes, implementing the layered-mode belief-propagation. We introduce the concept of LLR-update and mirror memory to enforce a pipelined decoding schedule. The pipelined architecture improves the latency of the LDPC decoder by about 2x-3x and has negligible performance loss when implemented with clever layer scheduling. We also present a low-complexity check-node architecture suitable for block-serial processing and utilize the properties of the *min* approximation to significantly reduce the memory requirement. The proposed architecture is suitable for mobile devices with data-rates of tens of mbps.

1. INTRODUCTION

Low-Density Parity-Check (LDPC) codes have gained a lot of attention because of their capability to achieve near Shannon limit performance [1]. Although, the LDPC decoding algorithm offers a lot of parallelism, the randomness and lack of structure in LDPC codes lead to complex interconnect and demand massive hardware resources and memory to fully exploit parallelism [2]. This has lead to design of structured LDPC codes that are suitable for high throughput hardware implementation [3]-[6]. In this paper, we present decoder architecture for irregular, partitioned codes based on permutation matrix(IPP), where the parity-check matrix is partitioned into non-overlapping block-columns and block-rows(also referred to as *layers* or *supercodes*) [3], [4]. Structured codes significantly reduce area, power and memory of the decoder and perform close to randomly designed LDPC codes [3]. Partitioned LDPC codes have been proposed as a forward error correction scheme in standards such as IEEE 802.11n and 802.16e, with data-rates ranging from tens to hundreds of Mbps [5], [6].

In [3], Mansour and Shanbhag have proposed a layered-mode belief-propagation (L-BPA [4]) schedule for partitioned LDPC codes, where extrinsic messages are passed between the layers in each iteration, leading to faster convergence. One bottleneck in L-BPA is the dependency between the layers, which increases the decoding latency, especially for serial implementation. In this paper, we present simple modifications to L-BPA schedule that allows pipelining between different layers and reduces the latency by 2x-3x. By cleverly scheduling the order of *layers* in iteration, we obtain the performance similar to that of layered-decoding. We then present a block-serial architecture for pipelined L-BPA that is scalable for throughput and hardware area. The proposed architecture is best suited for structured LDPC codes that have large variable-node (bit-node) degree and many *layers*. A reduced complexity check-node processor architecture is developed that is suitable for serial implementation.

2. LAYERED BELIEF PROPAGATION

Consider a (*N, K*) LDPC code with parity-check matrix partitioned in to *L* layers and *C* block-columns and without loss of generality, assume that each sub-matrix is a cyclically shifted identity matrix of dimension *S*. Let, $\mathbf{R}[i]$ ($\mathbf{C}[j]$) be the set of the positions of columns(rows) of parity-check matrix $\mathbf{H}$ such that, $\mathbf{R}[i]=\{j|\mathbf{H}_{i,j}=1\}$ ($\mathbf{C}[j]=\{i|\mathbf{H}_{i,j}=1\}$). ρ_l denotes the degree of the check-nodes in layer l and v_c denotes the degree for the variable-nodes in block-column c. Further, $c_i v_j^{[q]}$ denotes the extrinsic message from check-node i to variable-node j and $v_j c_i^{[q]}$ denotes the extrinsic message from variable-node j to check-node i at q^{th} iteration. λ_j and $L(x_j)^{[q]}$ are soft-input and updated log-likelihood-ratio (LLR), $L(x_j)^{[q]}=\log\{\Pr(x_j=1)/\Pr(x_j=0)\}$ for bit-node j at q^{th} iteration. Also, define $\psi(x)=-\log(\tanh(|x/2|))=\psi^{-1}(x)$.

In L-BPA, one iteration of conventional belief-propagation is broken into *L* sub-iterations and in each sub-iteration new check-node messages are computed for one layer. An improved message passing schedule then utilizes the updated check-node messages in the next sub-iteration to compute check-node messages for the next layer. The algorithm is now summarized [3], [4].

Step-1: Initialization- $c_i v_j^{[0]} = 0$, $\forall j \in \mathbf{R}[i]$

Step-2: Iterative decoding, For $q = 1,2,3,...,Q$

Part-A-B (Horizontal/Vertical Step):

@ Each Layer ($l = 0,1,2,...,L-1$)

$\forall j \in \mathbf{R}[i]$, $i = l \times S + [0,1,2,...,S-1]$,

$$M\left(c_i v_j^{[q]}\right) = \psi^{-1}\left[\sum_{j' \in \mathbf{R}[i]\setminus j} \psi\left(\left|\lambda_{j'} + \sum_{i \in \mathbf{C}[j']\setminus i} c_{i'} v_{j'}^{[q-1]}\right|\right)\right] \quad (1)$$

$$S\left(c_i v_j^{[q]}\right) = (-1)^{\rho_i - 1} \prod_{j' \in \mathbf{R}[i]\setminus j} sign\left(\left|\lambda_{j'} + \sum_{i \in \mathbf{C}[j']\setminus i} c_{i'} v_{j'}^{[q-1]}\right|\right) \quad (2)$$

$$c_i v_j^{[q]} = -S\left(c_i v_j^{[q]}\right) \times M\left(c_i v_j^{[q]}\right) \quad (3)$$

Part-C (LLR Update and Hard Decision):*

$$L\left(x_j\right)^{[q]} = \lambda_j + \sum_{i \in \mathbf{C}[j]} c_i v_j^{[q]} \quad (4)$$

IF $L\left(x_j\right)^{[q]} > 0$, $\hat{x}_j = 1$, ELSE $\hat{x}_j = 0$.

**For fixed iterations, can be deferred until the last iteration.*

Syndrome computation ($\mathbf{s} = \hat{\mathbf{x}}\mathbf{H}^{\mathrm{T}}$) may be used after each iteration to implement early stopping criteria. Layered-decoding improves convergence and cuts the required decoder iterations by half. Accumulated variable-to-check (V2C) messages are not stored but computed at every layer leading to significant memory reduction in two different ways [3]; (1) avoids storage of V2C messages, saving $\sum_n \upsilon_n$ memory locations; and (2) saves check-to-variable (C2V) message memory by overwriting C2V message of the next layer(s) to be processed by newly computed C2V messages. Savings in C2V memory is achieved at the expense of extra computations in each layer. This approach is attractive for the LDPC codes with small maximum variable-node degree, but leads to higher latency or higher hardware resources for the LDPC codes with large degrees.

The decoder architecture proposed in [3] accesses and processes all the sub-matrices in a layer in parallel and requires substantial hardware resources. It is suitable only for very high throughput applications. For mobile devices, with strict area and power constraints, a block-serial implementation that processes each sub-matrix serially with lesser hardware requirements is more attractive. However, the bock-serial implementation typically increases the decoder latency; thereby limiting the throughput.

3. PIPELINED SCHEDULE FOR L-BPA

Layered-mode belief-propagation improves performance by passing the updated extrinsic messages between the layers within a conventional decoding iteration. As shown in Figure 1, each layer is processed serially and extrinsic information is updated at the end of each layer, causing dependency between the layers. *Read* and *Write* procedures represent memory read/write for C2V messages and LLRs, and *Process* represents actual computation of extrinsic messages. In this schedule, sub-iteration for the next layer does not start until the completion of the *Write* for the current layer due to possible overlap between the layers. Moreover, in a block-serial implementation, the *Write* procedure operates on each sub-matrix in a serial fashion, resulting in a longer latency for each sub-iteration. This limits the throughput of L-BPA, especially for a structured code with many layers (e.g. lower code-rate) or higher check-node degree (e.g. higher code-rate).

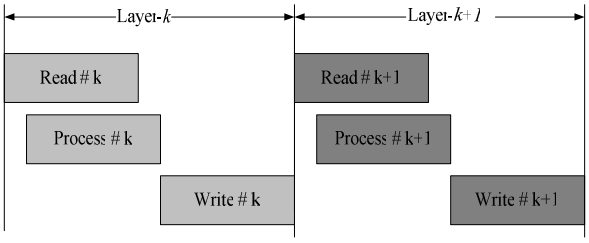

Figure 1 – Schedule of Layered Belief-Propagation

Figure 2 depicts the proposed pipelined schedule, where *Read* and *Process* part of the next layer are overlapped with the *Write* of the current layer. Hence, the advantage of the layer update may not be reflected to the adjacent layer/s. The L-BPA schedule described in section-2 is not suitable to implement a pipeline between interconnected layers. To enable pipelining, we propose simple modifications to L-BPA schedule that updates LLRs after each sub-iteration by adding the difference between new and old C2V messages to the old LLRs. The V2C messages are computed by subtracting old C2V messages from previously updated LLRs, which reduces the computations in V2C messages at the expense of a small increase in C2V memory.

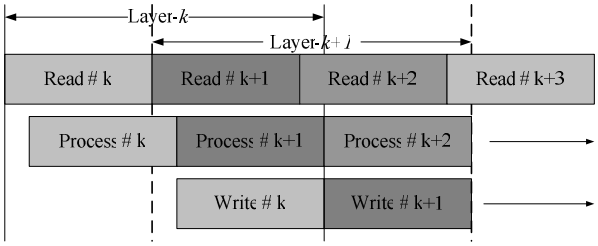

Figure 2 – Schedule of Pipelined Layered-Belief Propagation

The modifications to L-BPA schedule are now summarized. The remaining part is similar to section-2.

Step-1: $L\left(x_j\right)^{[0]} = \lambda_j \;\&\; c_i v_j^{[0]} = 0$

Step-2: Iterative decoding, For $q = 1,2,3,...,Q$

Part-A-B (Horizontal/Vertical Step):

……

$$M\left(c_i v_j^{[q]}\right) = \psi^{-1}\left[\sum_{j' \in \mathbf{R}[i] \setminus j} \psi\left(\left|L\left(x_{j'}\right)^{[q-1,l-1]} - c_i v_{j'}^{[q-1]}\right|\right)\right] \quad (5)$$

$$S\left(c_i v_j^{[q]}\right) = (-1)^{\rho_i - 1} \prod_{j' \in \mathbf{R}[i] \setminus j} sign\left(L\left(x_{j'}\right)^{[q-1,l-1]} - c_i v_{j'}^{[q-1]}\right) \quad (6)$$

$$c_i v_j^{[q]} = -S\left(c_i v_j^{[q]}\right) \times M\left(c_i v_j^{[q]}\right) \quad (7)$$

$$\Delta L\left(x_j\right)^{[q,l]} = c_i v_j^{[q]} - c_i v_j^{[q-1]} \quad (8)$$

Part-C (LLR Update Step):

$$L\left(x_j\right)^{[q,l]} = L\left(x_j\right)^{[q-1,l-1]} + \Delta L\left(x_j\right)^{[q,l]} \quad (9)$$

For clarity, bit-node LLRs are additionally indexed with the layer (l), although it is transparent in actual implementation. An extra subtract operation is required at the check-node-processor (CNP) to compute adjustment to LLRs ($\Delta L(x_j)$). The schedule allows us to overlap the processing of different layers; thereby reducing the overall decoding latency.

4. PIPELINED BLOCK-SERIAL ARCHITECTURE

In this section we develop a pipelined block-serial architecture based on the modified decoder schedule. Each sub-matrix in a parity-check matrix is treated as a block within which all the involved check-nodes are processed in parallel using S functional units. Alternatively, the decoder may support only F ($\leq S$) parallel functional units. In order to reduce the latency of layered decoding, a pipeline is enforced between the layers and over full parity-check matrix. The *Read* for the next layer begins as soon as the *Read* for current layer is completed; thereby avoiding the latency of the memory *Write* of each layer. Different computations of decoding (*Process*) are partitioned in one or more stages of the pipeline. The *Read*, *Process* and *Write* procedures are carried out simultaneously and span over one or more layers. Memory contention between *Read* and *Write* procedures can be avoided by changing the order of sub-matrix *Read*/*Write* within a layer. Since *Read* and *Write* may happen simultaneously, dual-port memories are required.

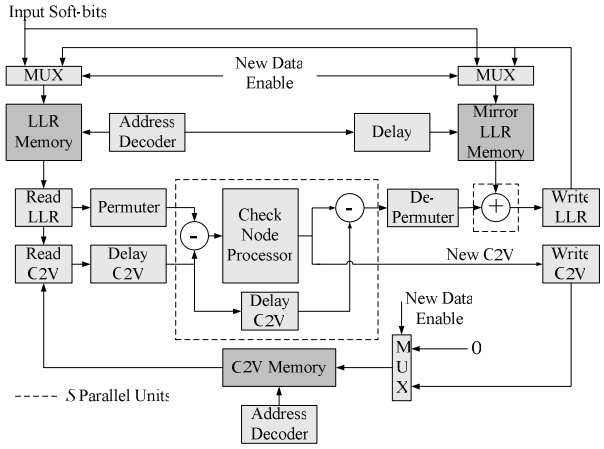

Figure 3 – Pipelined Block-Serial Architecture

Figure 3 shows the data flow for improved pipelined architecture. CNP computes adjustment on bit-node LLRs and at the end of each sub-iteration, the LLRs are updated using the available LLR-updates. A second *Read* is required to fetch the previously updated LLR value. To facilitate this, a mirror LLR memory is implemented that keeps the copy of LLRs, increasing the overall decoder memory requirement by N soft-words. However, the new architecture reduces the decoding latency by more than half. Both original and mirror LLR memories are initialized with soft inputs and C2V memory is initialized with zeros. The permuter implements message routing network and routes the LLRs to the correct CNP. For the LDPC codes designed with cyclically shifted identity matrix, the permuter can be implemented efficiently with staged multiplexers (MUX). The stages of the permuter can be pipelined, marginally increasing the depth of the pipeline. De-permuter implements inverse permutation on the LLR adjustments ($\Delta L(x_j)$).

4.1. Serial Check Node Processor Architecture

CNP involves logarithm and tan-h operations and is computationally the most complex part of the decoder. Operation $\psi(x)$ is commonly implemented using a look-up table (LUT) or well-known low-complexity *min* approximation. The performance of *min* algorithm can be improved by scaling the V2C messages [7]. The *min* approximation yields,

$$\psi^{-1}\left(\psi(x)+\psi(y)\right) \approx sign(x) * sign(y) * \min(x, y) \quad (10)$$

In block-serial processing, one input enters one of the S parallel CNP and one output per CNP is written back in a single clock. This knowledge is utilized to employ *accumulate and subtract* strategy to reduce the total number of computations from $\rho_l \times (\rho_l - 1)$ to $2\rho_l$ and obtain simplified CNP implementation [8]. Ignoring sign manipulation, the CNP update can be re-written as,

$$acc = \sum_{j \in \mathbf{R}[i]} \psi\left(\left|L\left(x_j\right)^{[q-1]} - c_i v_j^{[q-1]}\right|\right) \quad (11)$$

$$M\left(c_i v_j^{[q]}\right) = \psi^{-1}\left[acc - \psi\left(\left|L\left(x_j\right)^{[q-1]} - c_i v_j^{[q-1]}\right|\right)\right] \quad (12)$$

Figure 4 illustrates serial CNP architecture. Sign computation can be efficiently implemented using X-ORs. A FIFO of length $\rho_{\max} + 2$ is required to retain and delay $\psi(x)$ values. One advantage of the serial CNP architecture is that the number of processing units ($\psi(x)$, adder etc.) remains constant for any check-node degree, allowing efficient hardware layout and usage.

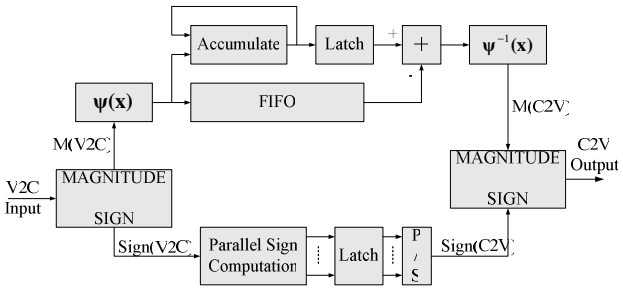

Figure 4 – CNP Architecture for Block-Serial Implementation

Simplification in *min* approximation can be obtained by observing that the magnitude of a C2V message is either minimum (MIN) or 2nd minimum (MIN2) value of the inputs. Hence,

$$k_1 = \underset{j' \in R[i]}{\arg\min} \left(\left|L\left(x_{j'}\right)^{[q-1]} - c_i v_{j'}^{[q-1]}\right|\right) \quad (13)$$

$$MIN = \left|L\left(x_{k_1}\right)^{[q-1]} - c_i v_{k_1}^{[q-1]}\right| \quad (14)$$

$$MIN2 = \min\left(\left|L\left(x_{j'}\right)^{[q-1]} - c_i v_{j'}^{[q-1]}\right|\right),\ j' \in R[i] \setminus k_1 \quad (15)$$

$$M\left(c_i v_{k_1}^{[q]}\right) = MIN2,\ M\left(c_i v_j^{[q]}\right) = MIN, \forall j \neq k_1 \quad (16)$$

To enforce and maintain consistency in the pipeline for *min* approximation, MIN and MIN2 must be computed in a single pass, since new value enters the CNP at every clock. Figure 5 shows the proposed implementation strategy for the CNP employing *min* approximation. MIN and MIN2 values of the inputs are computed in parallel using 2 comparators, a 2x1 MUX and a 4x1 MUX. New C2V messages are computed using the index of the MIN, sign vector and a compare-select unit, eliminating the need for FIFO. Size of the extra comparator is *ceil*($\log2(\rho_{\max})$) bits.

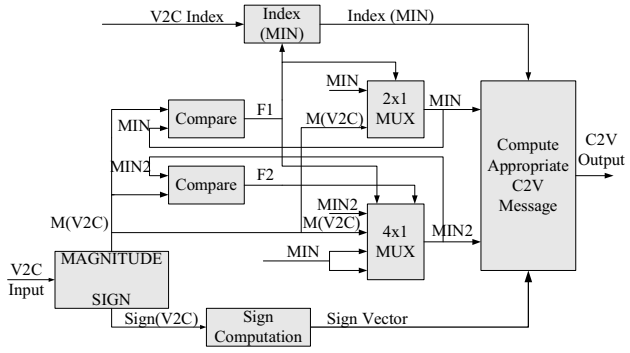

Figure 5 – Low Complexity CNP (Min-Sum Core)

4.2. Memory Savings for Min-Sum Implementation

We use dual-port RAMs for C2V and LLR memories. With well-balanced pipeline stages and clever partitioning, single-port RAMs can be used for C2V memory. The total memory requirements of

the decoder, including the mirror memory, is $b(2N + LS\ \rho_{\max})$ bits, where b is word-length of C2V messages. For *min* approximation C2V memory can further be conserved by storing sign and magnitude separately. Since the magnitude of C2V message is either MIN or MIN2 of the input V2C messages, overall memory requirement can be reduced significantly by storing MIN, MIN2, sign of different C2V messages and the index of MIN value. For each check-node the decoder requires; 2(b-1) bits to store magnitudes, $\rho_{\max}$ bits to store sign and *ceil*(log2($\rho_{\max}$)) bits to store index of MIN value. The overall savings in C2V memory is,

$$S_{C2V} = \left(1 - \frac{(2(b-1) + \rho_{\max} + \lceil \log 2(\rho_{\max}) \rceil)}{b \times \rho_{\max}}\right) \times 100\% \quad (17)$$

Assuming 8-bit precision, about 60% savings in C2V memory can be achieved for rate ½ code with maximum check-node degree of 8 and about 75% savings can be achieved for rate 5/6 code with maximum check-node degree of 20. Extra logic may be needed to convert C2V messages from sign-magnitude to 2's complement format.

4.3. Latency Improvement for Pipelined Architecture

Let us assume that permuter, V2C computation, CNP, de-permuter and LLR-update operations shown in Figure 3 has P pipeline stages, each consuming 1-clock cycle. Then, overall latency (clocks/iteration) of the block-serial layered LDPC decoder without any pipeline between the layers is,

$$\Gamma_{LBPA} = \sum_{l=1}^{L} (2\rho_l + P - 1) \quad (18)$$

In order to enforce pipeline, the processing time for each layer must be equal. For irregular LDPC codes with different check-node degrees for different layers, pseudo computation cycles are inserted to balance the pipeline. They do not alter the end result, but increase the decoding latency. The LDPC codes in [3]-[6] allow efficient pipelining as the check-node degrees of various layers are almost same. The latency of the pipelined decoder for the LDPC code with maximum check-node degree $\rho_{\max}$ is,

$$\Gamma_{pipeline} = L \times \rho_{\max} + P - 1 \quad (19)$$

It should be noted that in the original L-BPA schedule, the CNP directly computes LLR instead of LLR adjustment and needs 1 less decoder stage than the pipelined schedule. Neglecting the differences in the decoder stages with and without pipeline mode, the improvement in latency is,

$$\Gamma_{improved} = L \times (\rho_{\max} - 1) + P \times (L - 1) \quad (20)$$

For example, for a rate ½ LDPC code with base-matrix dimension (12, 24) and maximum check-node degree of 7, the improvement in latency with 14 decoder stages is roughly 3x. Considering small pipeline inefficiencies, the proposed pipelined architecture offers 2x-3x improvement in decoder throughput.

4.4. Performance of Pipeline Layered Decoding

Figure 6 compare the block error rate of pipelined and original L-BPA for R-1/2 (N=1536) and R-5/6 (N=2400) LDPC codes specified in [5]. The decoder implements layered-min-sum (LMS) algorithm with 10 decoding iterations and has 14-pipeline stages. Pipelined decoding with an alternate schedule is also shown in the same figure, labeled Pipeline-2, where the layers are processed in other than linear order. The pipelined schedule alters the message passing mechanism of the L-BPA as the adjacent layer(s) may not benefit from the advantage of the layered update. However, the performance loss incurred by the pipelined decoder is negligible and as shown in the figure a pipelined decoder with effective scheduling performs as good as the L-BPA.

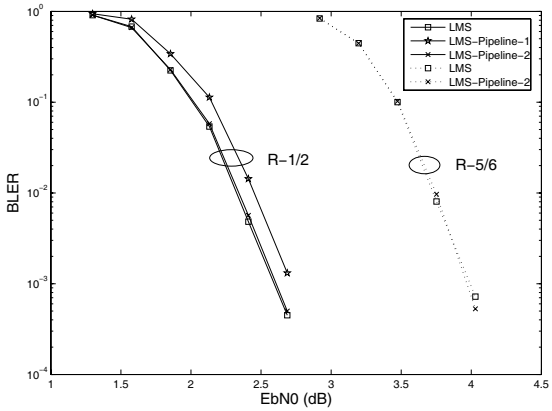

Figure 6 – Performance of Pipelined Layered-Min-Sum

5. CONCLUSION

Pipelined block-serial decoder architecture is presented for structured LDPC codes. The pipelined architecture improves the latency of the layered decoder by 2x-3x, with a small increase in memory and hardware requirements. Simplified CNP architecture is developed that is suitable for block-serial implementation and the overall decoder implementation requires significantly less hardware resources than parallel layer implementation. We also show that the overall check-node memory requirement of the decoder can be reduced by about 50%-75% for *min-sum* algorithm.

6. REFERENCES

[1] D.J.C. Mackay and R.M. Neal, "Near Shannon Limit Performance of Low Density Parity Check Codes", *IEEE Electron. Lett*., vol. 32, no. 18, pp. 1645-1646, Aug 1996.

[2] C. Howland and A. Blanksby, "Parallel decoding architectures for low density parity check codes", *IEEE Int. Symp. on Circuits and Systems*, vol. 5, no. 2, pp. 58-60, Feb 2001.

[3] M.M. Mansour and N.R. Shanbhag, "High-Throughput LDPC Decoders", *IEEE Trans. on VLSI Systems*, vol. 11, pp. 976-996, Dec 2003.

[4] D.E. Hocevar, "A Reduced Complexity Decoder Architecture via Layered Decoding of LDPC Codes", *in Proc., IEEE Workshop on Signal Processing Systems(SiPS)*, pp. 107-112, Oct 2004.

[5] V. Stolpman et. al., "Irregular Structured LDPC Codes", IEEE 802.16 Broadband Wireless Working Group, contribution IEEE C802.16e-04/264, Aug 2004.

[6] IEEE WirelessMAN 802.16, "Air Interface for Fixed and Mobile Broadband Wireless Access Systems", P802.16e/D11, Sept 2005.

[7] J. Heo, "Analysis of Scaling Soft Information On Low-Density Parity-Check Code", *IEEE Electron. Lett.*, vol. 39, no. 2, pp. 219-221, Jan 2003.

[8] T. Bhatt et. al., "Fixed point DSP Implementation of Low-Density Parity-Check Codes", in *Proc. of the 9th IEEE DSP Workshop*, Hunt, TX, Oct 2000.