A Novel Design Methodology for High-Performance Programmable Decoder Cores for AA-LDPC Codes *

Mohammad M. Mansour and Naresh R.Shanbhag
iCIMS Research Center, ECE Dept.
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
1308 W. Main Street, Urbana, IL 61801
(mmansour,shanbhag)@uiuc.edu

ABSTRACT

A new parameterized-core-based design methodology targeted for programmable decoders for low-density parity-check (LDPC) codes is proposed. The methodology solves the two major drawbacks of excessive memory overhead and complex on-chip interconnect typical of existing decoder implementations which limit the scalability, degrade the error-correction capability, and restrict the domain of application of LDPC codes. Diverse memory and interconnect optimizations are performed at the code-design, decoding algorithm, decoder architecture, and physical layout levels, with the following features: 1) Architecture-aware (AA)-LDPC code design with embedded structural features that significantly reduce interconnect complexity, 2) faster and memory-efficient turbo-decoding algorithm for LDPC codes, 3) programmable architecture having distributed memory, parallel message processing units, and dynamic/scalable transport networks for routing messages, and 4) a parameterized macro-cell layout library implementing the main components of the architecture with scaling parameters that enable low-level transistor sizing and power-rail scaling for power-delay-area optimization. A 14 mm^2 programmable decoder core for a rate-$\frac{1}{2}$, length 2048 AA-LDPC code generated using the proposed methodology is presented, which delivers a throughput of 1.6 Gbps at 125 MHz and consumes 760 mW of power.

1. INTRODUCTION

With the renewed interest in iterative decoding via message-passing in coding theory and the introduction of the concept of codes defined on graphs [1], low-density parity-check codes [2] have emerged as serious competitors to the well-known turbo codes [3] in terms of error-correction capability. However, efficient hardware implementation techniques of LDPC decoders remain largely immature compared to their turbo decoder counterparts, clearing the way for turbo codes to occupy main stream applications ranging from wireless applications to fiber-optics communications. Hence, the quest for new hardware design methodologies for LDPC decoders has become a topic of increasing interest, gradually promoting LDPC codes as the coding technique of choice for next generation applications.

The design of LDPC decoder architectures departs from the traditional decoder design in that it is intimately related to the structure of the parity-check matrix defining the code [4]. An LDPC code of length n is defined by a random sparse parity-check matrix $\mathbf{H}_{m\times n}$, where m is the number of parity-check equations, and is typically described by a bipartite graph whose reduced adjacency matrix is $\mathbf{H}$ as shown in Fig. 1. To the m rows of $\mathbf{H}$ there corresponds a set of m *check* nodes on one side of the partition, and to the n columns a set of n *bit*-nodes on other side of the partition in the graph. Two nodes in the graph are connected if their corresponding entry in $\mathbf{H}$ is non-zero. In a *regular* (c,r)-LDPC code, bit-nodes have degree c and check-nodes have degree r. The number and location of the non-zeros in $\mathbf{H}$ determine the computational load performed by the decoder as well as the memory size and the interconnect complexity needed to route and store the computation results.

LDPC codes are decoded iteratively using Gallager's two-phase message-passing (TPMP) algorithm [2] which computes iteratively extrinsic probability values associated with each bit-node using disjoint parity-check equations that the bit participates in [2]. Each iteration consists of two phases of computations in which updates of all bit-nodes are done in **phase 1** by sending messages to neighboring check-nodes, and then updates of all check-nodes are done in **phase 2** by sending messages to neighboring bit-nodes. Updates in each phase are independent and can be parallelized.

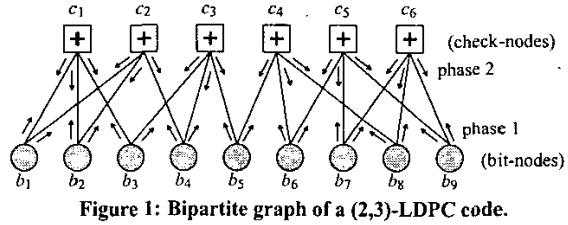


Figure 1: Bipartite graph of a (2,3)-LDPC code.

LDPC codes are not standardized in the sense that a system designer has the flexibility of formatting the data according to a desired size and code-rate depending on the channel conditions, the required level of coding gain, or other system considerations, which make a code-programmable decoding platform extremely desirable. Moreover, with the emerging process technologies and the constantly evolving communications standards and applications, a parameterizable decoder core that is portable across technology generations with predicable design quality is even more attractive. However, state-of-the-art significantly falls short of these objectives, and only custom, FPGA-based, or synthesis-based hard decoders have been attempted. In parallel decoder implementations [5] that mimic the topology of the bipartite graph, the randomness in communicating messages on the graph edges results in complex interconnect and poses serious implementation challenges for large n in terms of placement of $m+n$ function units and routing more than $2ncb$ wires, where b is the message bit-width. Custom algorithms are needed for placement and buffer insertion to reduce route lengths and routing congestion, achieve timing closure, and increase hardware utilization. For large n, parallel decoder implementations quickly become intractable. Consider the ring placement strategy shown in Fig. 2 in which $m = N_1M_1$ check function units are surrounded by $n = 4N_2M_2$ bit function units having respective aspect ratios of w_1/h_1 and w_2/h_2. It can be shown [6] that in a 0.18μm CMOS technology, the average interconnect length $\bar{l}(n)$ is

*This work was supported with funds from NSF under grants CCR 99-79381 and CCR 00-85929.

lower-bounded by

$$\bar{l}(n) \geq 0.15\sqrt{n} \text{ mm}$$

for $w_1/h_1 = 5$ and $w_2/h_2 = 2$, assuming uniform distribution of wires across function units. In addition, for a rate-$\frac{1}{2}$ regular $(3,6)$-LDPC code, the area of the decoder $A(n)$ is lower-bounded by

$$A(n) \geq 0.045\sqrt{n} \text{ mm}^2,$$

and the total number of wires $N(n)$ for $b = 4$ is given by

$$N(n) = 24n.$$

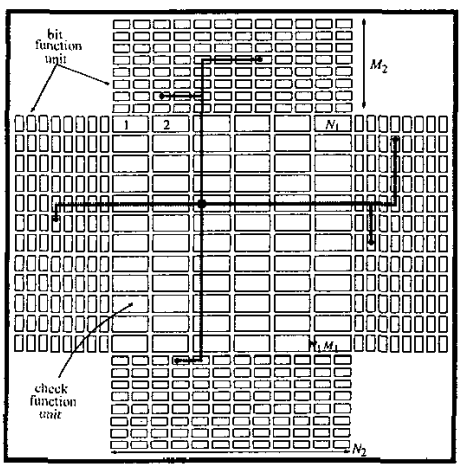


Figure 2: Ring placement strategy of function units.

Figure 3 plots the functions $\bar{l}(n), A(n)$, and $N(n)$ for typical values of n. The figure clearly demonstrates that parallel architectures scale unfavorably with the code length and become impractical to implement. On the other hand, serial decoder architectures [7], in which computations are folded onto a subset of function units and message communication takes place through centralized memory, require significant memory overhead that amounts to four times the number of edges in the graph and suffer a throughput penalty due to the serial processing bottleneck.

This paper attempts to break the architectural dependence on the code properties by proposing a new design methodology that jointly performs diverse optimizations at various system abstraction levels, namely, at the code-design, decoding algorithm, decoder architecture, and physical layout levels. The outcome is: 1) a class of structured or architecture-aware LDPC (AA-LDPC) codes having regularity features that significantly reduce interconnect complexity, 2) a faster and memory-efficient turbo-decoding algorithm for LDPC codes together with a new message update mechanism immune to quantization effects, 3) a code-programmable architecture having distributed memory, parallel message processing units, and dynamic and scalable transport networks for storing, processing, and routing messages, respectively, capable of decoding ensembles of AA-LDPC codes, and 4) a parameterized macro-cell layout library containing layout implementations of the main components of the architecture which are characterized by a set of scaling and other feature parameters used by a core-optimizer that enable low-level transistor sizing and power-line scaling for power-delay-area optimizations. Section gives an overview of the methodology and its subsections highlight the details. Section presents a programmable AA-LDPC decoder core generated using the proposed methodology, and Section concludes the paper.

2. DESIGN METHODOLOGY

In this section, a parameterized-core-based design methodology for high-throughput and memory-efficient programmable decoder cores

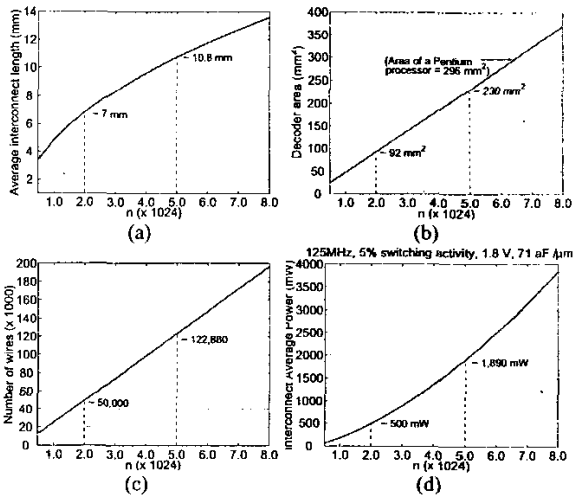


Figure 3: Parallel decoder architectures: (a) Average interconnect length, (b) decoder area, (c) number of wires, and (d) average interconnect power as a function of code length.

for LDPC codes is proposed. Figure 4 shows a flow graph of the methodology. The following subsections highlight the main aspects of the methodology and their impact on reducing memory overhead and interconnect complexity.

2.1 Architecture-Aware LDPC Codes

The interconnect problem stemming from the inherent randomness of LDPC codes is addressed by designing structured or **architecture-aware** LDPC (AA-LDPC) codes having regularity features favorable for an efficient and scalable decoder implementation. The ensemble of AA-LDPC codes of length n and rate R is defined by a block parity-check matrix $\mathbf{H}$ having B block rows and D block columns such as the one shown in Fig. 5, where each block is an $S \times S$ sub-matrix and S is a code-independent parameter. These submatrices are required to be either all-zeros $S \times S$ matrices or *permutation* matrices. An $S \times S$ binary permutation matrix is simply the identity matrix $\mathbf{I}_{S\times S}$ whose rows (or equivalently columns) are randomly permuted. Hence, a regular (c,r)-LDPC code would have r permutation matrices per block row and c permutation matrices per block column, and is denoted as a $[D,B,S,c,r]$-AA-LDPC code. A particular choice of permutation matrices and their positions in $\mathbf{H}$ define an instance of the ensemble of $[D,B,S,r]$-AA-LDPC codes. The code length is given by $n = BS$, and the code rate is $R \geq 1 - D/B$. Figure 5 shows a parity-check matrix of a regular $[6,12,8,6]$-AA-LDPC code.

The main advantages of AA-LDPC codes over other classes of LDPC codes are twofold. First, they transform the LDPC decoding problem employing the TPMP algorithm into a turbo-decoding problem [4] in which only one type of messages is processed, thus eliminating the storage required to save multiple check-to-bit messages (a savings of 75%). This follows from the fact that the ones in the rows in each block row of $\mathbf{H}$ do not overlap, and consequently, the block rows can be processed independently by passing messages only between adjacent block rows as opposed to potentially all rows as is the case with the TPMP algorithm. Second, taking the next step further towards an efficient and scalable decoder implementation (as compared to [5]), the structure of AA-LDPC codes reduces the complexity of the interconnection network when it comes to forwarding and retrieving messages between the non-zero entries as defined in $\mathbf{H}$. Observe that the ones in the rows of $\mathbf{H}$, absent any structure, would generally have random column indices requiring r $(n{:}1)$-(de)multiplexers to access r messages

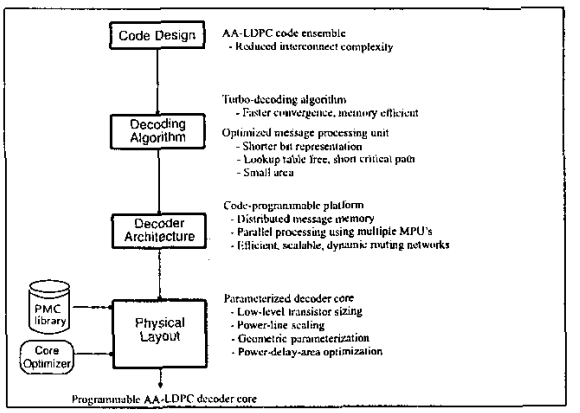


Figure 4: Proposed design methodology.

$$\mathbf{H} = \begin{array}{|cccccccccccc|} \hline \mathbf{P}_{11} & & \mathbf{P}_{12} & \mathbf{P}_{13} & & & \mathbf{P}_{14} & & \mathbf{P}_{15} & & & \mathbf{P}_{16} \\ & \mathbf{P}_{21} & \mathbf{P}_{22} & & & \mathbf{P}_{23} & \mathbf{P}_{24} & \mathbf{P}_{25} & & & & \mathbf{P}_{26} \\ & \mathbf{P}_{31} & & \mathbf{P}_{32} & \mathbf{P}_{33} & \mathbf{P}_{34} & & & \mathbf{P}_{35} & \mathbf{P}_{36} & & \\ \mathbf{P}_{41} & \mathbf{P}_{42} & \mathbf{P}_{43} & & \mathbf{P}_{44} & & & & \mathbf{P}_{45} & & \mathbf{P}_{46} & \\ & & & & & \mathbf{P}_{51} & \mathbf{P}_{52} & \mathbf{P}_{53} & & \mathbf{P}_{54} & \mathbf{P}_{55} & \mathbf{P}_{56} \\ \mathbf{P}_{61} & & & \mathbf{P}_{62} & \mathbf{P}_{63} & & & \mathbf{P}_{64} & & \mathbf{P}_{65} & \mathbf{P}_{66} & \\ \hline \end{array}$$

$B = 12$, $D = 6$

Figure 5: Parity-check matrix of a regular $[6,12,8,6]$-AA-LDPC code ($\mathbf{P}_{ij}$ is 8×8).

corresponding to the row. This solution quickly becomes impractical for large n, or when multiple rows are accessed in parallel to increase decoding throughput. Moreover, the overhead of the control mechanism of the (de-)multiplexers which keeps track of the column positions of all the ones in the parity-check matrix becomes too complex to implement. On the other hand, with the prescribed structure of AA-LDPC codes, S rows can be accessed in parallel with a complexity of $rS\log(S)$ in terms of (2:1)-multiplexers, a reduction of order

$$\frac{rS(BS-1)}{rS\log(S)} = \frac{n-1}{\log(S)} \sim O(n).$$

Further, the reduction in control overhead is proportional to

$$\frac{rS\log(n)}{rS/2\log(S)} = \frac{2\log(n)}{\log S} \sim O(\log(n)).$$

While the parity-check matrix shown in Fig. 5 has desirable architectural properties, it is not a priori clear whether LDPC codes having such structure would achieve comparable bit-error rate (BER) performance to randomly constructed codes of similar complexity. In [8–10], it was shown that indeed AA-LDPC codes based on cyclotomic cosets [8] and Ramanujan graphs [9, 11] have BER performance that compares favorably with randomly generated codes.

2.2 Turbo-Decoding of AA-LDPC Codes

Since the rows in each block row in an AA-LDPC code do not overlap, a block row in $\mathbf{H}$ can be viewed by itself as a parity-check matrix of an even parity-check code having support only on rS bit positions. Consequently, $\mathbf{H}$ is considered as the concatenation of D constituent codes [4, 9] which can be decoded in tandem in a "turbo-decoding" fashion by S message processing units (MPU's) that operate in parallel [8, 10]. The MPU's together form a soft-input soft-output (SISO) decoder which is designated by $\mathbf{D}$. The algorithm is called the turbo-decoding message-passing (TDMP) algorithm for LDPC codes. Let δ denote the received channel information about the bits, and $\lambda_i, i = 1, \cdots, D$, denote the extrinsic reliability information obtained by decoding the bits assuming they belong only to the ith constituent code. Let γ denote the total or *posterior* reliability information known about the bits, or $\gamma = \delta + \sum_{i=1}^{D} \lambda_i$. Decoding proceeds according to the **extrinsic principle** which asserts that decoder $\mathbf{D}_i$ takes as input all information known so far about the bits that was previously generated *not* using $\mathbf{D}_i$, and generates as output updated extrinsic information λ_i using the constraints of the ith constituent code.

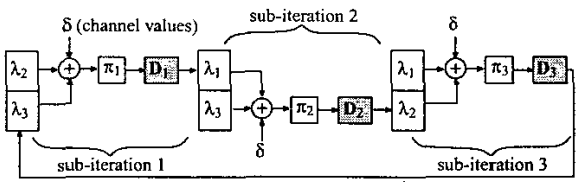


Figure 6: Block diagram of the TDMP algorithm showing the message exchange between the decoders $\mathbf{D}_i$, interleavers π_i, and λ-memory for $D = 3$.

The pseudo-code of the TDMP algorithm is listed below, and is described in Fig. 6 for $D = 3$. The algorithm performs $D = 3$ decoding sub-iterations on the block rows of $\mathbf{H}$. Starting from block row 1, extrinsic reliability values λ_1 are computed for each bit using SISO decoder $\mathbf{D}_1$ and the input channel reliability values δ, assuming that the bit belongs to the code defined by block row 1 (i.e., using λ_2 and λ_3, not λ_1). This extrinsic information is fed as *a priori* information through an interleaver (π_1) to SISO decoder $\mathbf{D}_2$ operating on the second block row. The interleaver can be factored into at most B S-to-S independent permuters following the structure of the AA-LDPC code. $\mathbf{D}_2$ in turn updates the extrinsic reliability values assuming that the bits belong to the code defined by block row 2 and generates updated values for λ_2. The process is repeated for the third block row. A single update of messages based on one block row is referred to as a *sub-iteration*, and a round of updates across all the block rows constitutes a single decoding iteration. In the final iteration, hard decisions are made based on the *posterior* reliability values γ read from the third SISO decoder.

Algorithm 1 $\gamma = \text{TDMP}(\delta)$

$\gamma \leftarrow \delta$
$\lambda_i \leftarrow 0,\ i = 1, \cdots, D$
for $k = 1$ to MAXITER **do**
  for $i = 1$ to D **do**
    $[\Lambda, \Gamma] \leftarrow \mathbf{D}_i(\lambda_i, \pi_i(\gamma[I_i]))$ $\quad \{I_i = \text{index set}\}$
    $\lambda_i \leftarrow \Lambda$
    $\gamma[I_i] \leftarrow \pi_i^{-1}(\Gamma)$.
  end for
end for

The TDMP algorithm has two main advantages over the TPMP algorithm [4, 9]: (1) It eliminates the storage required to save *multiple* check-to-bit messages and replaces them with a *single* message corresponding to the most recent check-message update, a savings of

$$\frac{4nc - nc}{4nc} \times 100\% = 75\%,$$

and (2) it exhibits a faster convergence behavior requiring between 20%-50% fewer decoding iterations to converge for a given signal-to-noise ratio (and hence higher decoding throughput) compared to the TPMP algorithm.

2.3 Low-Complexity Message Computation

The commonly employed message-update mechanism based on Gallager's equations [2] is prone to quantization noise which results in increased decoding latency and switching activity (and hence power

consumption) in the decoder. An alternative approach for computing messages is to use a simplified form of the BCJR algorithm [12] tailored to the syndrome trellis of an even parity-check code. A reduced-complexity message update unit is presented that eliminates the need for lookup-tables, hence simplifying implementation cost considerably especially in parallel MPU implementations. In the proposed method, messages are updated using a simple "max-quartet" bivariate function $Q(x,y)$ defined as

$$Q(x,y) = \max(x,y) - \max(x+y,0) + \max\left(\frac{5}{8} - \frac{|x-y|}{4},0\right) - \max\left(\frac{5}{8} - \frac{|x+y|}{4},0\right), \quad (1)$$

which approximates the key equations of the BCJR algorithm in differential form. It can be shown that $Q(x,y)$ is a simpler (implementation-wise) and more accurate approximation of the ideal key equations than other approximations available in the literature (e.g., [13]). In terms of $Q(x,y)$, the key equations of the BCJR algorithm simplify to

$$\Delta\alpha' = Q(\Delta\alpha, \gamma - \lambda), \qquad \Delta\beta' = Q(\Delta\beta, \gamma - \lambda),$$
$$\Lambda = Q(\Delta\alpha, \Delta\beta), \qquad \Gamma = \Lambda + (\gamma - \lambda). \quad (2)$$

Figure 7 shows a logic circuit implementing (1), and Fig.'s 8-9 show a parallel and a serial MPU implementation of (2).

2.4 Programmable Decoder Architecture

A programmable decoder architecture implementing the TDMP algorithm for regular $[D,B,S,r]$-AA-LDPC code ensembles is shown in Fig. 10. The architecture is composed of six main components: 1) A set of S MPU's for message computation, 2) S memory modules (λ-Memory) each of size $D \times r$ that store the extrinsic messages pertaining to all constituent codes, 3) a dual-port memory module (γ-Memory) of size $B \times S$ that stores the posterior reliabilities of all bits, 4) a memory module (**H**-Memory) of size $D \times r$ that stores the column positions of the permutation matrices in **H** used to index γ-Memory, 5) two networks Ω and Ω^{-1} of size $S/2\log(S)$ for routing messages between γ-Memory and the MPU's, and 6) a dual-port memory module (π-Memory) of size $Dr \times S/2\log(S)$ that stores the switch control for the networks.

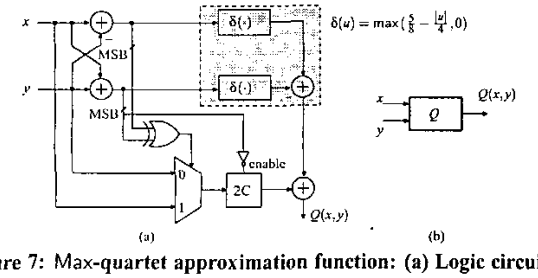


Figure 7: Max-quartet approximation function: (a) Logic circuit, and (b) block symbol.

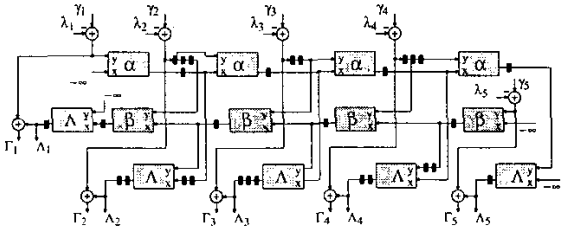


Figure 8: Parallel MPU implementation.

For simplicity, serial MPU's are assumed. The **H**- and π-Memory modules are programmed according to the desired instance of the ensemble of $[D,B,S,r]$-AA-LDPC codes. The λ-Memory is organized

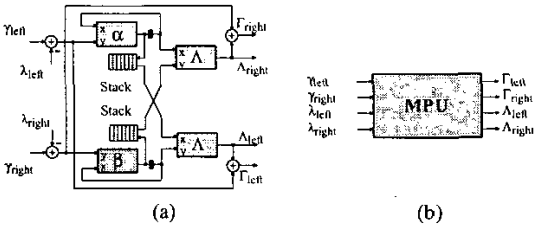


Figure 9: (a) Serial MPU, and (b) block symbol.

to deliver maximum bandwidth to the MPU's using single read/write ports without the need for switching networks. The jth λ-Memory stores the jth row of each block row in **H**, and the MPU's consume S rows from λ-Memory in parallel. The Ω and Ω^{-1} networks are shuffle-exchange networks consisting of an array of $S/2 \times \log(S)$ 2-by-2 switches and controlled by the π-Memory as shown in Fig. 11, capable of routing $2^{S/2\times\log(S)}$ out of $S!$ permutations without conflicts. Other *rearrangeable* multistage interconnection networks such as the Clos and Beneš networks widely used in multiprocessor, fiber optics, and photonics applications can be employed if arbitrary permutation routing capability is needed.

The decoder completes a decoding sub-iteration pertaining to the ith code in $(r+4)$ cycles as follows:

1- The MPU's read all the extrinsic messages λ_i for code i from λ-Memory.

2- The corresponding posterior messages are read from γ-Memory and permuted using the Ω network according to code i.

3- Updated λ messages are written back to λ-Memory.

4- Updated γ messages are written back to γ-Memory after inverse permuting using the Ω^{-1} network.

A single decoding iteration is completed in $(r+4)D$ cycles.

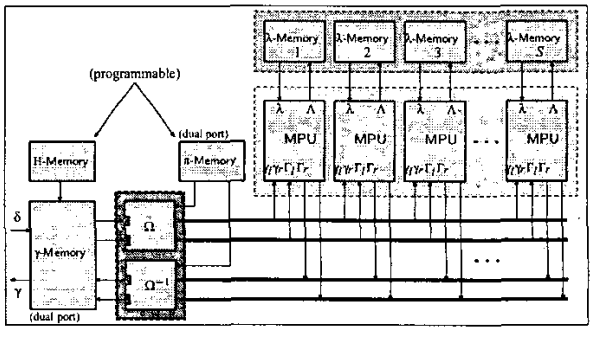


Figure 10: TDMP decoder architecture.

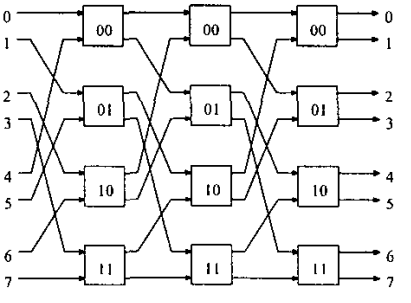


Figure 11: Omega network.

2.5 Parameterized Macro- Cell Library

Core-based IC design methodologies offer valuable tradeoffs between the high quality of full-custom designs and the short design cycle time of synthesis-based design methodologies. Particularly effective to communication systems, in which applications, standards, and process technologies constantly change and evolve towards optimum system energy and throughput efficiencies, are those cores that offer in addition flexibility between portability across technology generations and predictability of design quality. This section presents an effective approach towards achieving the above objectives using a parameterized-core-based IC design methodology targeted for AA-LDPC decoders. Using this methodology, a custom-quality layout of an AA-LDPC decoder is synthesized using high-level algorithmic and architectural specifications without passing through the automated synthesis, place, and route step. The novelty in this approach is the ability to perform low-level transistor sizing, power-rail scaling, and other geometric modifications through a small set of parameters that characterize the main building blocks of the core.

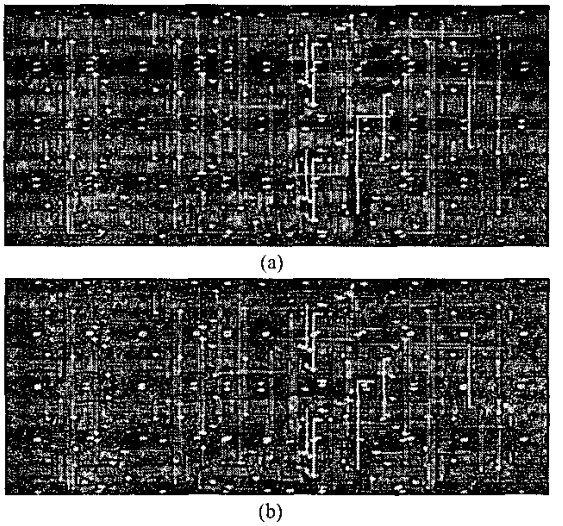
The methodology is based on a hierarchical parameterized cell layout library. The first level is a parameterized leaf-cell (PLC) library that accommodates all basic cells available in a standard cell library. In addition, each PLC in the library is characterized by a set of scaling parameters ($\underline{\sigma}$) assigned to individual or groups of transistors within the cell depending on its functionality, fan-in, and fan-out. All PLC's are designed using the 1-D layout strategy in which the P- and N-type transistors are placed across a horizontal centerline running parallel to the power rails. Scaling up to five times the minimum size is done vertically within the cell, while the transistor folding technique is employed for larger scaling factors. Moreover, the power rails are assigned a width parameter (ω) that is determined based on the maximum current drawn by the cell as a function of $\underline{\sigma}$. A PLC can be instantiated in virtually any size to within the technology's minimum feature size by setting its $\underline{\sigma}$ and ω parameters. A current macro-model based on [14] is used to determine the $\underline{\sigma}$ parameters from the delay specs, the ω parameter, as well the power consumption of the PLC.

The second level is a parameterized macro-cell (PMC) library specific to AA-LDPC decoder cores which implements the MPU's, Ω-networks, and memory modules of a programmable AA-LDPC decoder using the PLC library. The PLC's in a PMC inherit their $\underline{\sigma}$ and ω parameters from the global parameters that characterize the PMC. A core-optimizer is used to determine the optimal parameter settings for a PMC. For example, Fig. 12(a) shows an un-optimized PMC implementing the "max-quartet" function of Fig. 7, and Fig. 12(b) shows an optimized version for a delay of 10 ns. Figure 13 shows pipelined implementations of the parallel and serial MPU's shown in Fig.'s 8 and 9. The PMC of Figure 13(a) corresponds to a parallel MPU with $r = 5$ containing 13 Q blocks, 29 flip-flops, and 10 adders while the PMC of Fig. 13(b) corresponds to a serial MPU with $r = 6$ and 6 pipeline stages containing four Q blocks, 2 stacks, and 4 adders. Both PMC's are optimized for a stage delay of 10 ns. Figure 14 shows an optimized PMC of the Ω-network of Fig. 11 with $S = 64$ and delay of 10 ns. Similar PMC's for the memory modules were generated (see Fig. 15).

3. IMPLEMENTATION EXAMPLE

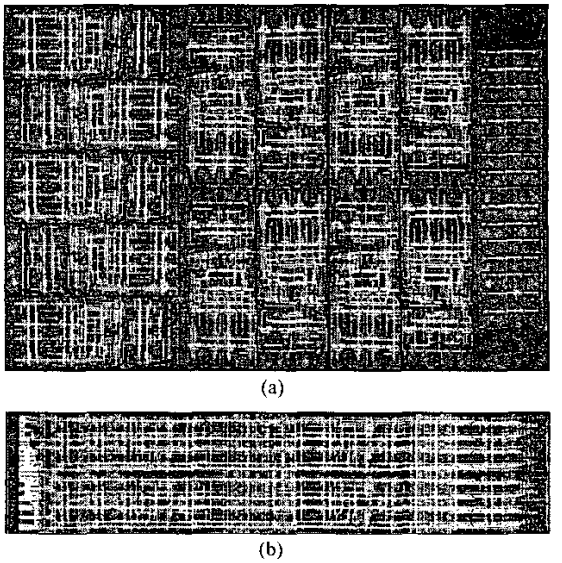
To demonstrate the effectiveness of the proposed core-based design methodology, a TDMP decoder core targeted for rate-$\frac{1}{2}$, length 2048, $[16, 32, 64, 6]$-AA-LDPC code ensembles was generated using the PMC library. The core has a four bit datapath, and is implemented in a 0.18μm, 1.8 V CMOS technology. Figure 17 shows the layout of the core. The decoder achieves a throughput of 1.6 Gbits/s at a clock frequency of 125 MHz and consumes 760 mW of power. Figure 18(a) plots the power distribution of the core amongst the memory (53.9%), MPU (37.7%), and Ω-network (8.4%) blocks. Figure 18(b) shows a similar plot for area distribution: memory (36.2%), MPU (55.3%), and Ω-network (8.5%).

The core has an area of 14 mm^2 (3.75 times smaller than the decoder of [5]), attains a higher throughput by 60%, is capable of running at a higher clock frequency due to the absence of routing congestion and relatively short route lengths, and more importantly, decodes an LDPC code of twice the length. In the light of Fig. 3 and the problems existing LDPC decoder implementations suffer from, up to the authors' knowledge the proposed methodology constitutes a superior approach in almost all respects to all known implementations in the literature. Figure 16 illustrates how the proposed decoder area scales with code length versus existing techniques.



(a) (b)

Figure 12: PMC for the "max-quartet" function: (a) Un-optimized, and (b) delay-optimized for 10 ns.



(a) (b)

Figure 13: PMC for the (a) parallel MPU, and (b) serial MPU optimized for a stage delay of 10 ns.

4. CONCLUSION

A new design methodology that solves the problems of memory overhead and interconnect complexity of current-day LDPC decoder

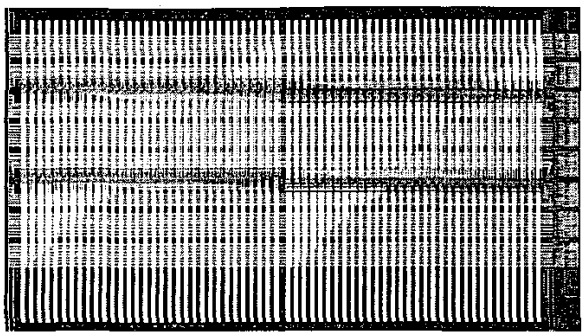


Figure 14: PMC for the Ω-network of Fig. 11.

implementations has been proposed. The methodology performs memory and interconnect optimizations at various system abstraction levels and generates LDPC decoder cores with architectural programmability and layout parametrization capabilities, which bode well with the demands of next generation communications applications. A decoder core demonstrating the effectiveness of the proposed methodology has been presented.

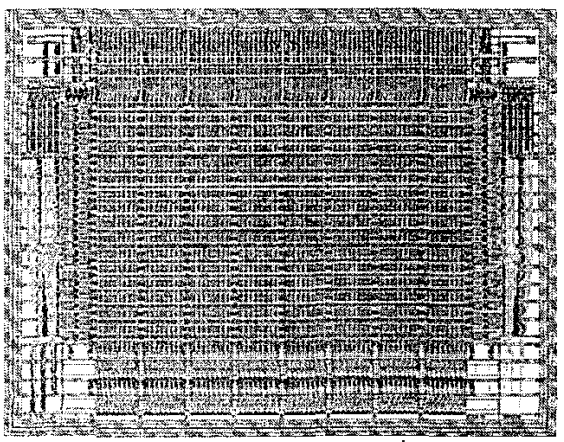


Figure 15: PMC for a 32×64 dual-port SRAM module.

5. REFERENCES

[1] N. Wiberg, *Codes and decoding on general graphs*, Ph.D. thesis, Linköping University, Sweden, 1996.

[2] R. G. Gallager, *Low-Density Parity-Check Codes*, MIT Press, Cambridge, MA, 1963.

[3] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon Limit Error-Correcting Coding and Decoding: Turbo Codes," in *IEEE Int. Conf. on Communications*, 1993, pp. 1064–1070.

[4] M. M. Mansour and N. R. Shanbhag, "Turbo decoder architectures for low-density parity-check codes," in *GlobeCOM 2002, Taiwan*, Nov. 2002.

[5] C. Howland and A. Blanksby, "Parallel decoding architectures for low density parity check codes," in *Proc. of 2001 IEEE Int. Symp. on Circuits and Systems, Sydney*, May 2001, pp. 742–745.

[6] M. M. Mansour, *VLSI Architectires for Iterative Channel Decoders*, Ph.D. thesis, University of Illinois at Urbana-Champaign, USA, 2003.

[7] E. Yeo et al., "VLSI architectures for iterative decoders in magnetic recording channels," *IEEE Trans. on Magnetics*, vol. 37, no. 2, pp. 748–755, Mar. 2001.

[8] M. M. Mansour and N. R. Shanbhag, "Low-power VLSI decoder architectures for LDPC codes," in *ISLPED 2002, Monterey, CA*, Aug. 2002, pp. 284–289.

[9] M. M. Mansour and N. R. Shanbhag, "Memory-efficient turbo decoder architectures for LDPC codes," in *SiPS 2002, San Diego, CA*, Oct. 2002.

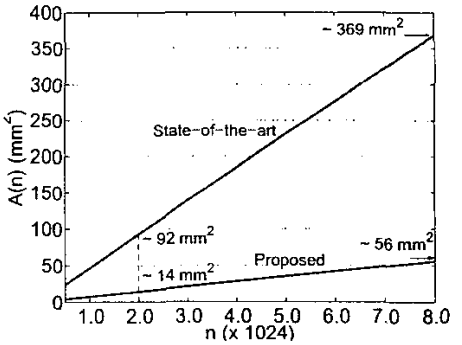


Figure 16: Scaling of decoder area with code length.

[10] M. M. Mansour and N. R. Shanbhag, "High-throughput memory efficient decoder architectures for LDPC codes," *submitted to IEEE Transactions on VLSI Systems*, 2002.

[11] M. M. Mansour and N. R. Shanbhag, "Construction of LDPC codes from Ramanujan graphs," in *Conf. on Info. Sciences and Systems, Princeton University*, Mar. 2002.

[12] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. on I.T.*, pp. 284–287, Mar. 1974.

[13] X. Y. Hu et al., "Efficient implementations of the sum-product algorithm for decoding LDPC codes," in *GLOBECOM 2001*, 2001, vol. 2, pp. 1036–1036E.

[14] Makram M. Mansour, Mohammad M. Mansour, and A. Mehrotra, "Modified Sakurai-Newton current model and its applications to CMOS digital circuit design," in *IEEE Computer Society Annual Syposium on VLSI*, Feb. 2003, pp. 62–29.

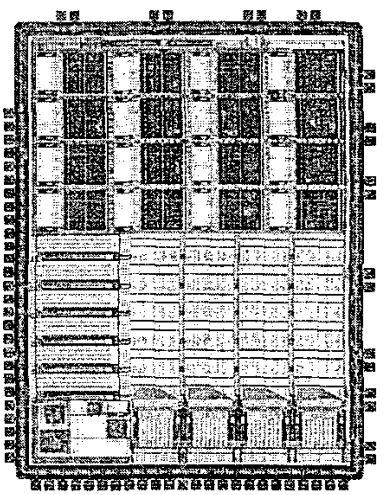
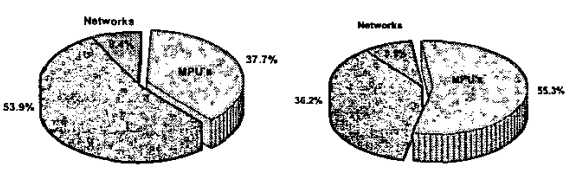


Figure 17: Layout of the AA-LDPC decoder core.



(a) (b)

Figure 18: Decoder core: (a) Power, and (b) area distribution.