A High-Speed Fully-Programmable VLSI Decoder for Regular LDPC Codes

Euncheol Kim, Nikhil Jayakumar, Pankaj Bhagwat, Anand Selvarathinam, Gwan Choi, Sunil P Khatri
Department of Electrical Engineering, Texas A&M University, College Station, TX 77843

ABSTRACT

This paper presents a VLSI implementation of a Low-Density Parity Check (LDPC) decoder that achieves 2.4 Gbps throughput yet permits real-time configuration of (1) rate, (2) code length, and (3) the parity equations. This decoder can be programmed in the field, much like an FPGA. We describe the architectural, circuit-level and layout-level details of our implementation. Our design can handle variable rate codes of length up to 1024, and is implemented in a 0.1μm VLSI fabrication process. Our design has a die size of 12mm by 8mm and a power consumption of 7W. This implementation can extended to handle longer codes in a partially parallel manner, and allow for on-the-fly modification of the code.

1. INTRODUCTION AND PREVIOUS WORK

LDPC codes are known to achieve information rates very close to the Shannon limit when iteratively decoded [1]. Also, decoders for these codes have arithmetic computation requirements that are an order of magnitude less than Turbo decoders [2] for similar bit-error rate (BER) performance. Algorithms for decoding LDPC codes also have the advantage of being inherently parallel. In principle, this permits the use of multiple parallel processing elements to increase the throughput of the decoder. However, in practice, due to the high complexity of interconnects between processing units, exploiting this parallelism is a considerable challenge. One way to make the decoder more amenable to hardware implementation is by imposing a structure on the parity check matrix of the code. Examples of this approach are [3,7,12,13,18]. In [3] the authors present a decoder for a class of highly structured LDPC codes (known as ALDPC). In [8], the authors reported a standard-cell based ASIC implementation of an LDPC decoder, with a throughput of 1 Gbps. Similarly, in [7], a structured fully parallel custom VLSI implementation of an LDPC decoder was reported, with very high throughputs. The designs presented in [7] and [8] are of fixed code length and rate architectures. Our implementation, in contrast, allows completely general field configuration while simultaneously achieving throughputs that are higher than those of custom implementations such as [8]. In [19], a semi-parallel reconfigurable decoding approach is described, using FPGAs. The design implements a family of (3,6) codes using a special parity check matrix structure. The throughputs achieved are up to 127Mbps.

Our implementation achieves approximately 2.4Gbps with code length of 1024; it can be scaled to handle *longer codes*. Further, the design allows switching between different codes *on the fly*, making the implementation useful in highly secure application scenarios. The impact of the presented chip is two-fold. Two obvious application areas that can immediately benefit from the presented decoder design are: (1) communication and storage devices where coding requirements may vary unpredictably, and upgradeability or maintainability may be costly or impossible, such as in deep-space communication or in soft radio applications; (2) a code-development/analysis environment where an extensive and swift analysis of a variety of code configurations is indispensable. This device permits analysis of code performance in a real-time setting wherein a BER analysis at a high SNR is tractable. This device permits "configure and instrument" during run-time thereby eliminating the slow and expensive process of setting up the experiments for each configuration of decoder design and code data.

This paper is organized as follows: Section 2 overviews LDPC code, Section 3 describes our approach, with implementation details, Section 4 provides simulation results, and Section 5 draws several concluding remarks and future work.

2. BRIEF REVIEW OF LDPC CODES

An LDPC code is a class of parity check codes which can be fully defined by parity–check matrix **H**. To be specific, it is defined as the null space of a very sparse *M*x*N* parity check matrix **H**. An LDPC code is represented by a bipartite graph, called a Tanner graph, in which one set of *N bit* or *variable* nodes corresponds to the set of codewords, while another set of *M check* nodes corresponds to the set of parity check constraints. Each edge corresponds to a non-zero entry in the parity check matrix **H.**

LDPC decoding is performed using an iterative algorithm called belief-propagation (BP). The structure of BP decoding algorithm directly matches the Tanner graph, and in principle, enables us to make use of the parallelism that is native to LDPC codes. The following steps summarize the BP algorithm [5]:

1. Initialize each variable node with the intrinsic (channel) information, and compute the variable-to-check messages.
2. Pass the variable-to-check messages from variable nodes to check nodes along the edges of the Tanner graph.
3. At each check node, generate the check-to-variable message using all incoming messages from the incident variable nodes.
4. Pass the check-to-variable messages from check nodes to variable nodes along the edges of Tanner graph.
5. At each variable node, update the estimate of the corresponding bit (using the incoming message and intrinsic information) and compute the outgoing variable to check message.
6. Repeat steps 2-5 until either a valid codeword is obtained

In this paper we restrict the discussion to *regular* LDPC codes. In regular LDPC codes the bit node and check node degrees are constants (denoted by j and k respectively). The *rate* of the code is an indicator of amount of redundancy that is added to the data. For a regular (j, k) code it is expressed as R = (k-j)/k, or alternatively in terms

of dimensions of the **H** matrix as R = (c-r)/c, where, c is the number of columns and r is the number of rows in the **H** matrix.

3. OUR APPROACH

Our parallel implementation of the regular-code decoder accommodates selectable rates from the set {7/10, 6/9, 5/8, 4/7, 3/6, 2/5, 1/4}. Our design is fully on-line programmable. We serialize the routing of messages between processing elements. For example, in a rate ½ (3, 6) code a bit node sends out three messages. Instead of sending three messages on three different buses we send them serially on a single bus. This reduces the size of the routing network and also the overall die size, allowing the design to be clocked at a higher speed. The hardware we have developed is similar in this sense to a FPGA implementation of an LDPC decoder, except that it is developed specifically for the task of LDPC decoding. This is unlike general FPGAs designed to implement arbitrary functionality. This is the reason for the higher throughputs achieved by our decoder.

We have carried out the architectural, circuit-level and layout level implementation of our design, to obtain accurate area estimates of the final IC. Further, we have performed accurate 3-D parasitic capacitance (using SPACE-3D [15]) and resistance extraction. Then we used these in a SPICE [9] simulation model, using a 0.1µm fabrication process, to obtain accurate delay and power estimates of the circuit.

The key objectives of our design were to develop:

1. A message passing structure that is completely programmable within a useful range of rates.
2. A reconfigurable message delivery path that permits a run-time rearrangement of the H-matrix.
3. A placement of VN, CN and message-passing structures that minimizes the critical lengths and achieves high throughput.
4. A floorplan that is scaleable to accommodate longer code lengths.
5. A multiple-frame pipelining approach, where every stage including the message-passing cycle is balanced. The design is pipelined so that two frames are processed at the same time. When one frame transfers data from VNs to CNs, the other transfers date from CNs to VNs. The transfer delays are balanced.

Our programmable LDPC decoder is organized with Check-node computation Units (CUs) arranged in a horizontal row, in the center of the IC die, as shown in Figure 1. Variable-node computation Units (VUs) are arranged in the center of the die as well, in a vertical column. This positioning of the CUs and VUs reduces the wire-lengths in the design by a factor of 2. Wire delays in a VLSI IC increase quadratically with wire length, hence this placement of CUs and VUs results in a delay-efficient design. As shown in Figure 1, the majority of the IC consists of Switch-box Units (SUs), which allow for the programmable connection of any VU to any CU (and vice versa). The transfer of messages from VUs to CUs occurs simultaneously with the transfer of messages from CUs to VUs, in our approach. In the rest of this paper, we assume that VUs drive their data horizontally, while CUs drive their data in a vertical direction. Assume that SU_{ij} is at the intersection of the horizontal wire from VU_i and the vertical wire from CU_j.

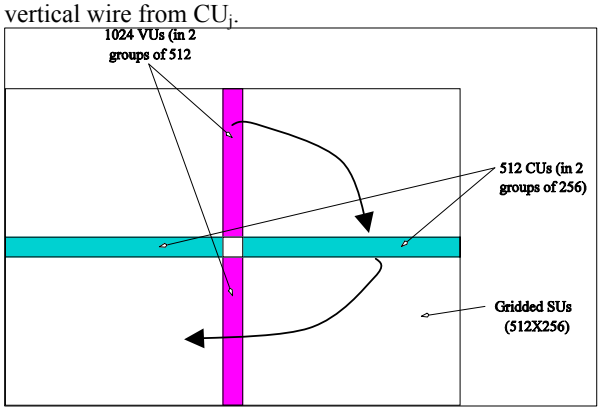

Figure 1: Architecture of LDPC decoder IC

Consider the simple Tanner graph shown in Figure 2. In this figure, the Variable Nodes (VNs) are shown to the left, while the Check Nodes (CNs) are on the right. In any iteration, VN1 needs to send data values a and d to the CNs it is connected to. Similarly, VN2 needs to send b and e, while VN3 needs to send c and f to the CNs. This transfer takes 6 cycles in total. The 6-bit string in each rectangle of Figure 3 describes the cycle in which these transfer operations are conducted. It indicates the cycle in which an SU (which would reside in the rectangle) connects its corresponding VU to its CU. For example, VN1 transfers data value a to CN1 in the first cycle, while d is transferred to CN2 in cycle 4. In the first 3 cycles, data values a, b and c are transferred to CN1, while in the next 3 cycles, values d, e and f are transferred to CN2.

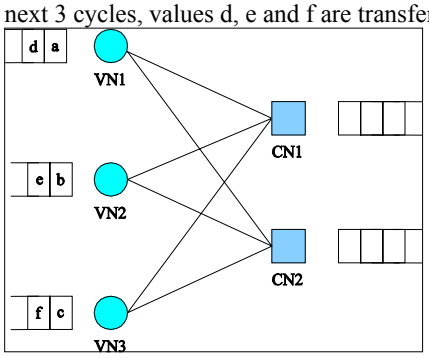

Figure 2: Sample Tanner Graph *G*

Note that any 6-bit string has exactly one '1' value. Note also that any message from the VNs is static for 3 clock cycles while they are consumed by the CNs.

For our implementation (in which the out-degree of check-nodes is at most 10, and the out-degree of variable-nodes is 3), we need 30 such cycles for a transfer. We implement a 5-bit counter in each SU, which is pre-loaded with the value of the cycle in which the SU is active (i.e. in which it connects its VU to its CU). When this down-counter reaches a 0 count, the required connection is made. Note that the registers comprising the counter are all scan [20] enabled, allowing us to upload them with

arbitrary values in the field. This feature enables the field programmability of our design.

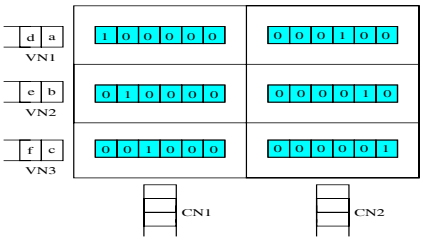

Figure 3: Message Transfer Timing for Graph *G*

The data transfer from CUs to VUs occurs in parallel with the transfer of data from VUs to CUs. The circuit level diagram that describes the SU to CU transfer is shown in Figure 4. The data from VUs is first driven horizontally. Suppose the data from VU_i needs to be selectively driven to CU_j. If the output of SU_{ij} is '1' (SU_{ij} has down-counted to 0), then the data from VU_i is driven to CU_j. The circuit for this transfer is shown in Figure 4. In this figure, the SELECT signals represent the outputs of SUs. In any transfer, exactly one SELECT signal in any column j can be '1', ensuring that the data transferred to CU_j corresponds to that driven by VU_i, if the SELECT value of SU_{ij} is '1'. Breaking the vertical wire into 5 segments (using the 5 OR gates shown in Figure 4) reduces the maximum length of any vertical segment, significantly reducing data propagation delay. A similar circuit is used for transferring data from CUs to VUs (which occurs in parallel with data transfer from VUs to CUs).

The layout level diagram of the SU is also shown in Figure 4. It consists of a 5-bit counter, and occupies an area of 10µm by 12.6µm. Using these dimensions, we were able to determine the length of the longest wires in the vertical and horizontal directions, allowing us to model the delays in the design precisely. By modifying the SU such that the SELECT signal fires when the count reaches two or more distinct values, we can handle longer codes (and/or allow the ability to switch between different codes on-the-fly.

The implementation of the CU and VU operations is performed in a pipelined fashion. VU operations are illustrated in Figure 5 (the CU operations are similar, with the exception that no channel information is used, and there is a stage of logtanh (logarctanh) computations before (after) the logic core, which are performed using Programmable Logic Arrays (PLAs) [17]. The speed requirements for these operations are minimal, since the critical delay path for our design is the transfer of data from CUs to VUs (and vice versa). We employ a simple pipelined add-accumulate operation in the VUs. For the VUs, in the first cycle, we utilize channel information (signal C is a 1 in the first cycle). Finally, the self-information is subtracted before date is stored in registers for subsequent transfer to CUs.

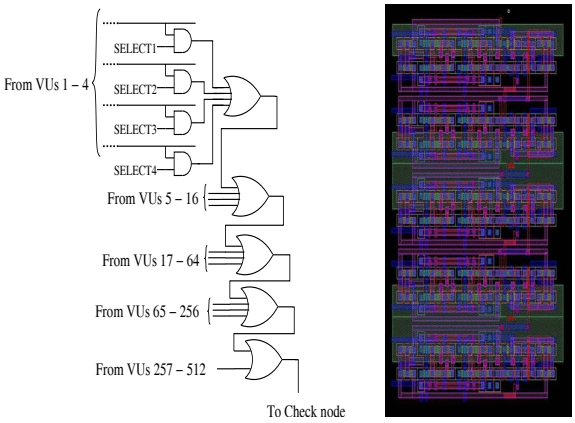

Figure 4: Circuit Diagram and Layout for Switch Unit

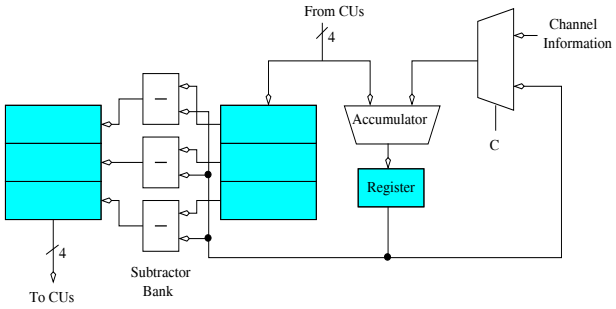

Figure 5: Circuit Diagram of Variable-node Unit (VU)

Using the dimensions of the switch core, we set out to estimate the worst case delay of the switching core. This was done by modeling the circuit from Figures 4 and 5, and augmenting this information with parasitic resistance and capacitance information obtained from SPACE-3D [15]. The devices in the circuit were assumed to be implemented in a 0.1µm fabrication process, and wire dimensions (thickness, ILD) for this process were used from [16]. We assigned metal layers 1 and 2 for local layout of the SU circuitry. To minimize the delay variation of the design due to cross-talk, we used metal layers 3 (horizontal) and 6 (vertical) for messages from VUs to CUs, and metal layers 5 (horizontal) and 4 (vertical) for messages from CUs to VUs. This arrangement ensures that layers which are switching are sufficiently far apart, minimizing their inter-layer cross-coupling capacitance. The 3-D extracted capacitance values for our 0.1µm process are indicated in Table 1, for the wiring dimensions we utilized in our simulations.

Layer i	$C_{i,0}$	$C_{i,i}$	$C_{i,i+1}$
2	-	-	32.80
3	1.48	72.81	26.87
4	1.39	72.43	30.13
5	1.53	39.75	35.77
6	7.09	48.67	-

Table 1: 3-D Parasitic Capacitance Values (aF/µm).

4. EXPERIMENTAL RESULTS

The clock period of the decoder is 60ps (worst case delay from VU to a SU) + 600ps (worst case delay from a SU to a CU). This includes the delay degradation due to worst-case cross-talk induced by neighbors of the switching wire. The propagation delay for signals driven from CUs to VUs is smaller, and therefore not critical. For a rate 1/2 code of length 1024, the resulting decoding throughput is computed as 1024/T, where T is the decoding delay for a code word. In our implementation, T = 30 (cycles) x 700ps (propagation delay) x 20 (iterations) x 2 x 0.5 (since VU-CU messages are transmitted simultaneously as CU-VU messages) = about 420ns, which results in a decoding throughput of 2.44Gbps.

The power consumption of our design consists of clock power (3W) and switching power (4W assuming simultaneous VU-CU and CU-VU transfers). Static power was found to be negligible. In comparison, the Flarion FPGA based decoder (which is also fully programmable) has a throughput of 100Mbps , computed for a parallelism factor of 128, rate of 0.5, code length of 1024, using 20 decoding iterations, with the FPGA operating at 100MHz. These values were obtained from [14]. In [8], the ASIC solution yields a non-programmable code with a throughput of 1Gbps and power consumption of 0.7W. Our throughput is considerably higher, though our power consumption is also higher on account of the need to drive signals halfway across the die.

The simulation in Figure 6 shows the longest path signal propagation delay for the OR tree logic of Figure 4. In this figure, the outputs of 2nd, 3rd and 4th level of the hierarchical OR tree are shown together with clock signal. Propagation delay of logic data '1' and '0' takes less than 0.6ns, as indicated in the figure.

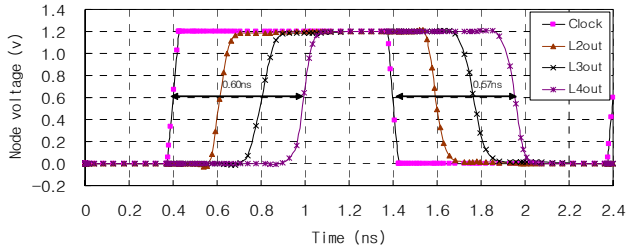

Figure 6: SPICE Plot Showing Propagation Delay

5. CONCLUSIONS

We have presented a fully programmable VLSI implementation of an LDPC decoder, using a 0.1µm fabrication process (12mm x 8mm IC). Circuit simulations indicate that our implementation achieves a decoding throughput of 2.4Gpbs with a power consumption of about 7W, while allowing full programmability of the LDPC code. This is an order of magnitude higher than existing fully programmable FPGA implementations [14], and is about twice the throughput of a non-programmable ASIC solution [8]. In the future, we plan to use our approach to implement longer codes by modifying the switch units (SU). Additionally, we plan to utilize this approach to implement codes which can change on the fly, allowing for an added degree of security in highly sensitive data transfer applications.

6. REFERENCES

[1] D. J. C. Mackay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *IEE Electronics Letters*, vol.33, no.6, pp.457-458, Mar. 1997.

[2] E. Yeo, P. Pakzad, B. Nikolic, and V. Anantharam, "VLSI architectures for iterative decoders in magnetic recording channels," *IEEE Trans. Magnetics*, vol.37, no.2, pp. 748-755, Mar. 2001.

[3] Pankaj Bhagawat, Momin Uppal, and Gwan Choi, "FPGA Based Implementation of Decoder for Array Low-Density Parity-Check Codes", IEEE J. Solid-State Circuits, Vol.37, 2002, pp. 404-412.

[4] T. Bhatt, K. Narayanan, and N. Kehtarnavaz, "Fixed point DSP implementation of low-density parity check codes," in *Proc. Ninth DSP Workshop, Hunt, Texas*, October 2000.

[5] T. Zhang and K. K. Parhi, "An FPGA Implementation of (*3,6*)-Regular Low-Density Parity-Check Code Decoder", *EURASIP Journal on Applied Signal Processing, special issue on Rapid Prototyping of DSP Systems,* May 2003 vol. 2003, no. 6, pp. 530-542.

[5] E. Eleftheriou and S. Olcer, "Low density parity-check codes for digital subscriber lines", Proc. ICC'2002, New York, pp.1752-1757(2002).

[6] A. Selvarathinam, G. Choi, A. Prabhakar, K. Narayanan, and E. Kim, "A massively scaleable decoder architecture for low-density parity-check codes", *Proceedings of International Symposium on Circuits and Systems*, May 2003, vol.2, pp.61-64

[7] V. Nagarajan, N. Jayakumar, S. Khatri and O. Milenkovic, "High-Throughput VLSI Implementations of Iterative Decoders and Related Code Construction Problems", *Proceeding, IEEE Global Telecommunications Conference (GLOBECOM)*, October 2004, vol. 1, pp 361-365.

[8] A.J.Blanksby and C.J.Howland, "A 690-mW 1024-b, Rate ½ Low-Density Parity-Check Code Decoder," *IEEE Journal of solid-state circuits*, vol.37, No.3, Mar 2002

[9] L. Nagel, "Spice: A computer program to simulate computer circuits," in *University of California, Berkeley*

UCB/ERL Memo M520, May 1995.

[10] J.M.Rabaey, "Digital Integrated Circuits: A design Perspective," *Prentice Hall Electronics and VLSI series*

[11] "The Berkeley Predictive Technology Model." http://www-device.eecs.berkeley.edu/_ptm/.

[12] T.Zhang and K.Parhi, "Join-(3,k)-Regular LDPC Code and Decoder/Encoder Design," *submitted to IEEE Trans. On Signal Processing*

[13] H. Zhong, and T. Zhang, "Design of VLSI Implementation-Oriented LDPC Codes," *preprint*.

[14] Flarion Technology, Vector-LDPC Coding Solution Data Sheet. http://www.flarion.com/products/vector.asp

[15] SPACE Layout to Circuit Extractor, Delft University of Technology, http://www.space.tudelft.nl

[16] S. Khatri, A. Mehrotra, R. Brayton, A, Sangiovanni-Vincentelli, R. Otten, "A Novel VLSI Layout Fabric for Deep Sub-micron Applications", *Proceedings, Design Automation Conference,* June 1999, pp. 491-496.

[17] S. Khatri, R. Brayton, and A. Sangiovanni-Vincentelli, "Cross-talk immune VLSI design using a network of PLAs embedded in a regular layout fabric," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, (Santa Clara, CA), pp. 412–418, Nov 2000.

[18] M. Mohiyuddin, A. Prakash, A. Aziz, W. Wolf, "Synthesizing Interconnect Efficient Low Density Parity Check Codes, *DAC 2004,* June 7–11, 2004, San Diego, California, USA.

[19] M. Karkooti and J. Cavallaro, "Semi-parallel Reconfigurable Architectures for Real-time LDPC Decoding," *Proc. IEEE International Conference on Information Technology (ITCC),* pp. 579-585, Volume 1, Las Vegas, NV April 2004