

# Design and Implementation of a Structured LDPC Decoder on FPGA\*

Jason Kwok-San Lee, Benjamin Lee, Jeremy Thorpe,

Kenneth Andrews, Sam Dolinar, Jon Hamkins<sup>†</sup>

{kwoklee, leeb, jeremy}@caltech.edu {andrews, sam, hamkins}@shannon.jpl.nasa.gov

Jet Propulsion Laboratory, California Institute of Technology

## Abstract

We present a design framework for the implementation for a Low-Density Parity-Check (LDPC) decoders. We employed a scalable decoding architecture for a certain class of structured LDPC codes. The codes are designed using a small  $(n, r)$  protograph that is replicated  $Z$  times to produce a decoding graph for a  $(Z \times n, Z \times r)$  code. Using this architecture, we have implemented a decoder for a (4096, 2048) LDPC code on a Xilinx Virtex-II 2000 FPGA, and achieved decoding speeds of 31 Mbps with 10 fixed iterations. The implemented message-passing algorithm uses an optimized 3-bit non-uniform quantizer that operates with 0.2dB implementation loss relative to a floating point decoder.

## 1 Introduction

Error correcting codes are widely used in digital communications to allow effectively error-free communications to occur over noisy channels. Low-Density-Parity-Check (LDPC) codes[1] have recently received a lot of attention because of their excellent error-correcting capability and highly parallelizable decoding algorithm. LDPC codes have been shown to be able to perform close to the Shannon limit[2]. They also can achieve very high throughput because of the parallel nature of their decoding algorithms. In the past decade or so, much of the research on LDPC codes has focused on the analysis and improvement of codes under decoding algorithms with floating point precision. However, to make LDPC codes practical in the real world, the design of an efficient hardware architecture is crucial.

There are several reasons to explore the implementation of LDPC code using reconfigurable hardware. First, research on constructing a good LDPC code involves empirical testing of various algorithm parameters and must be verified by simulations. Running simulations in software is a slow process. Reconfigurable hardware would provide a good way to speed up these simulations. Second, once verified, the same algorithms could be employed using the same reconfigurable hardware for proto-

types. In the future, error correcting system can be easily upgraded to the same design with a larger block length thus better performance, once a newer and larger FPGA is on the market. Third, communication standards involving the use of capacity approaching codes are still evolving. By implementation in reconfigurable hardware, communication systems would have the flexibility to adopt the new standards when they are developed and comply with standards in use in different geographic locations. Fourth, reconfigurable hardware allows communication systems to adaptively switch between different codes adjusting to the noise environments, power requirements, data block lengths, and other variable parameters.

LDPC codes are often specified by bipartite graphs consisting of two kinds of processing nodes, called variable nodes and check nodes. Each variable node is connected to channels to receive transmitted bits; each check node represents a parity check constraint. Each variable nodes is connected to a few check nodes through a set of edges.

In general, the construction of a good LDPC code requires a large number of nodes with disorganized interconnections, which complicates the implementation of due to long global interconnects. In this work, we propose a scalable architecture of a structured LDPC

\*The work described was funded by the IND Technology Program and performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

<sup>†</sup>Communications Systems & Research Section, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA.

$j^{\text{th}}$  check node (see figure 2).

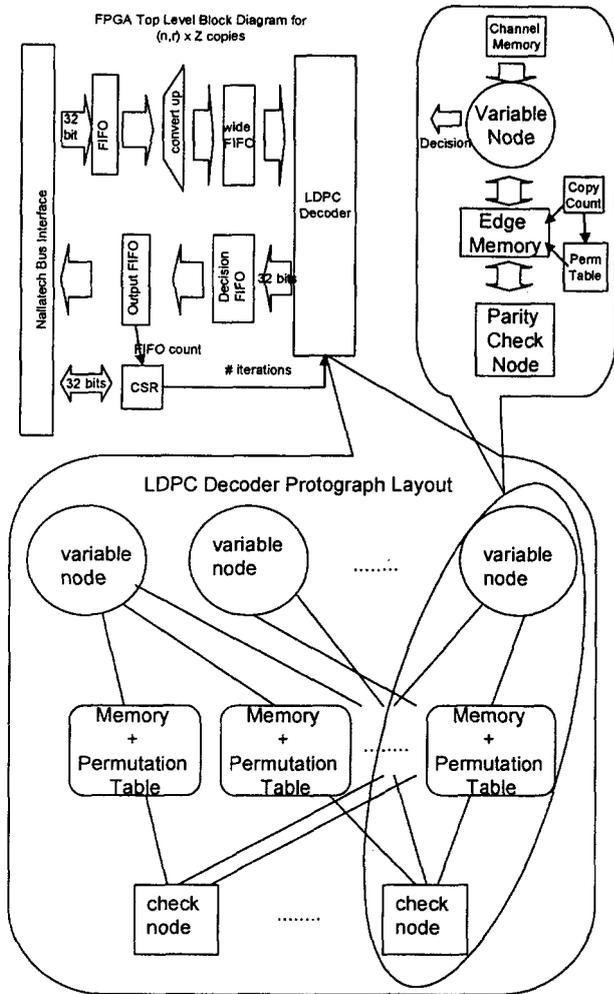


Figure 2. Our FPGA decoder architecture

### 2.3 Computation Scheduling

The cornerstone of our hardware architecture is the scheduling of message-updates in space and time. One iteration consists of a check node phase, followed by a variable node phase. In each phase, there are  $Z$  computation cycles (See figure 3).

In the check node phase, all check node modules read messages from the edge memory in ascending order, update the messages, and write their results back to the edge memory in ascending order. This computation across all  $r$  check node units occurs in parallel.

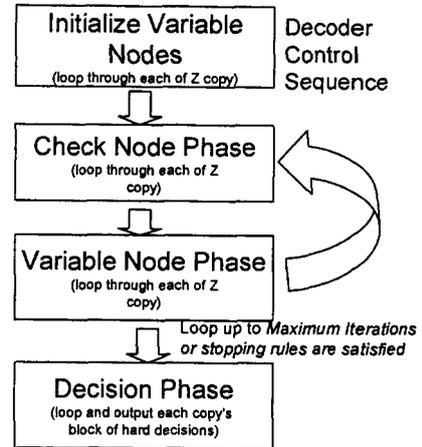


Figure 3. Decoder control sequence

In the variable node phase, all variable node modules read messages from the edge memory in permuted order, update the messages, and write back the edge memory in permuted order. The computation across all  $n$  variable node units also occurs in parallel. The decoding stops at the maximum iteration number, or when a stopping rule is satisfied.

Although this work was underway before the Flarion decoder patent was published, we can now make a useful comparison to that architecture. Flarion's design operates on all  $Z$  copies of the template LDPC graph in parallel and processes the individual nodes serially. In this manner, memory and processing can be centralized and a Single-Instruction-stream-Multiple-Data-stream (SIMD) instruction is used to access all  $Z$  messages[5].

In contrast, our system has multiple decentralized processing elements with multiple separate memories. All nodes in the template LDPC graph are operated on simultaneously in parallel and each of the  $Z$  copies are processed serially (see figure 4).

$Q_{ch}(ch) / Q_v(v) / Q_c(c)$	$ch / v / c$
-4	$ch < -3.3$ $v < -18$ $-5 \leq c < 0$
-3	$-3.3 \leq ch < -2.2$ $-18 \leq v < -12$ $-9 \leq c < -5$
-2	$-2.2 \leq ch < -1.1$ $-12 \leq v < -6$ $-26 \leq c < -9$
-1	$-1.1 \leq ch < 0$ $-6 \leq v < 0$ $c < -26$
0	$0 \leq ch \leq 1.1$ $0 \leq v \leq 6$ $c > 26$
1	$1.1 < ch \leq 2.2$ $6 < v \leq 12$ $9 < c \leq 26$
2	$2.2 < ch \leq 3.3$ $12 < v \leq 18$ $5 < c \leq 9$
3	$ch > 3.3$ $v > 18$ $0 \leq c \leq 5$

## 4 Structured LDPC Implementation Methodology

1. Choose a small  $(n, r)$  protograph by some methods (e.g. [7]).
2. Replicate the protograph  $Z$  times and apply a "girth conditioning" algorithm such as Progressive-Edge-Growth (PEG)[8] to permute the end points of each set of edges to obtain a large  $(Z \times n, Z \times r)$  code graph that does not contain short cycles.
3. Generate a decoder design by applying the protograph and the chosen permutations to parameterized Verilog HDL
4. Automatically synthesize, place and route design using Xilinx XST

Particular attention is given to the degree distribution of the small protograph chosen, as the larger code graph will have the same degree distribution.

## 5 Implementation

In order to avoid detrimental arithmetic precision effects and the complexity of collating a large number of inputs

and output at each processing unit, implementations of LDPC decoders benefit from regular code with a small maximum edge degree.[10] Therefore, for all our protographs implemented, we use a regular  $(3, 6)$  code in our LDPC decoder, in which each variable node connects to three check nodes and each check node connects to six variable nodes.

Our LDPC decoder runs an iterative quantized belief propagation algorithm. One iteration consists of a check node phase, followed by a variable node phase.

### 5.1 Check Node Processing Unit

In the check node phase, all  $r$  check node modules read messages from the edge memory in ascending order, update the messages, and write their results back to the edge memory in ascending order. This computation occurs in parallel across all  $r$  check node units.

The input messages for all  $r$  check node modules are 3-bit. A reconstruction function  $\phi_c$  maps the 3-bit input message into a 8-bit unreliability magnitude. 8 bits are used to avoid overflowing the sum of the six-input adder. After summing up the 6 unreliability magnitudes, the sum is then subtracted from each input unreliability magnitude to give out their respective updated 8-bit unreliability magnitudes. The updated 8-bit unreliability values, along with the updated sign bits, are then quantized into 3-bit by  $Q_c$ . The results are written back to the edge memory.

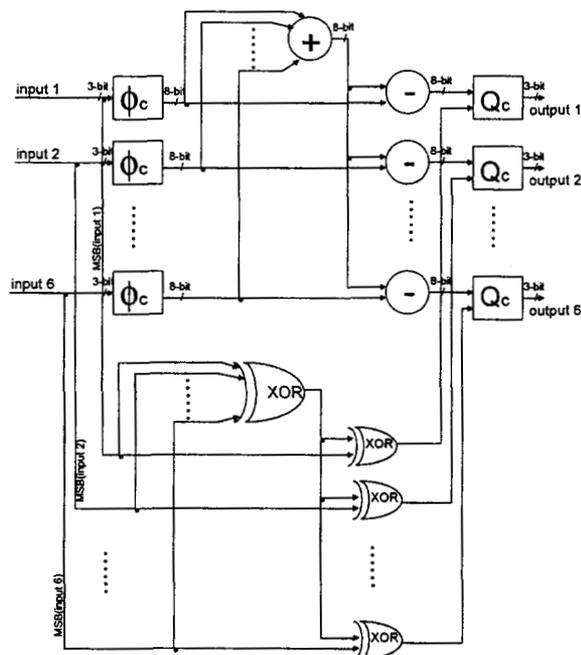


Figure 5. Check Node Processing Unit Circuit

Xilinx Virtex-II 2000 FPGAs provide fully synchronous dual-port Block RAM for memory use. However, using Block RAM in our design led to long routes from the Block RAMs to the processing logic, which increased the maximum delay unaccepted. Instead, we opted to use distributed RAM in our LDPC decoder design. All  $n$  variable node units,  $r$  check node units, and edge memory units could therefore be placed tightly over the FPGA area (See figure 8, 9, 10). As a result, routes between the adjacent logic and memory are minimized, making it easier for the placement and routing tool to meet timing requirements.

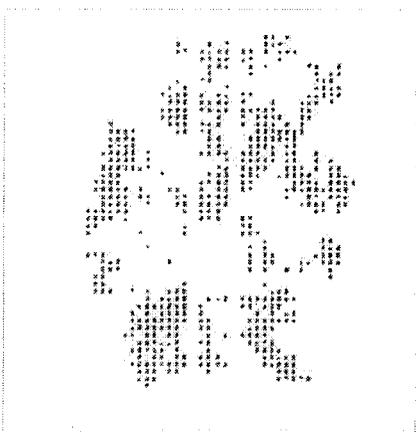


Figure 8. Distributed location of  $n$  variable nodes' logic over the FPGA area

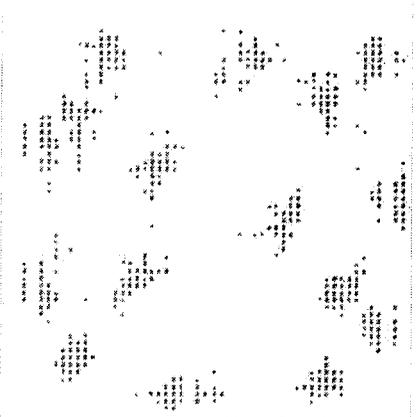


Figure 9. Distributed location of  $r$  parity check nodes' logic over the FPGA area

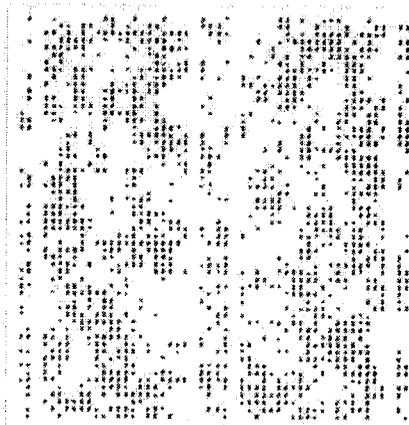


Figure 10. Locations of all distributed memory modules over the FPGA area

## 6 Performance

### 6.1 Speed/Throughput

We measured the real decoding throughput by the FPGA decoder of a  $(128 \times 32, 128 \times 16)$  LDPC code at fixed iteration numbers without stopping rules.

# iteration	Throughput without communication overhead (Mbps)	Throughput with communication overhead (Mbps)
1	314.47	10.01
10	31.45	7.74
20	15.72	6.20
50	6.29	3.91
100	3.14	2.41
150	2.10	1.74
200	1.57	1.37
250	1.26	1.12

The measured delay consists of communication overhead and decoder latency, in which decoder latency is proportional to the number of iterations. The decoder latency is 3.18 ns/bit/iteration. The communication overhead is 97.1 ns/bit in our tests. Communication overhead includes the buffer delay outside decoder module, and the time delay writing to and reading from the FPGA board.

- [4] J. Thorpe "Low-Density Parity-Check (LDPC) Codes Constructed from Protographs", IPN Progress Reports 42-154, April-June 2003
- [5] T. Richardson, "Methods and Apparatus for Decoding LDPC Codes," United States Patent No.: US 6,633,856 B2, Oct. 14, 2003.
- [6] H. Zhong and T. Zhong, "Design of VLSI Implementation-Oriented LDPC Codes," IEEE Semiannual Vehicular Technology Conference (VTC), Oct. 2003
- [7] J. Thorpe, K. Andrews, S. Dolinar, "Methodologies for Designing LDPC Codes Using Protographs," submitted to 2004 IEEE International Symposium on Information Theory.
- [8] X. Hu, E. Eleftheriou, and D. Arnold, "Progressive edge-growth Tanner graphs," Global Telecommunications Conference, 2001. GLOBECOM '01. IEEE , Volume: 2 , 25-29 Nov. 2001
- [9] J. Thorpe "Low-Complexity Approximations to Belief Propagation for LDPC Codes," Unpublished.
- [10] E. Yeo, B. Nikolic, and V. Anantharam, "Iterative Decoder Architectures," Communications Magazine, IEEE , Volume: 41 , Issue: 8 , Aug. 2003.