Design and Hardware Implementation of a Message-Passing Decoder for LDPC Codes.

Jeremy Thorpe
Technical Advisor: Ilya Dumer

June 14, 2000

Contents

1 Executive Summary

The practice of error-control coding will become increasingly important with the coming of wireless communications technologies. However, it is only in the last 5 years that huge theoretical advances have been made, particularly in the area of iterative decoding methods, which promise to change forever its practice.

One class of error-control codes which is able to be attacked using iterative decoding is Low-Density Parity Check (LDPC) codes. In this project, I propose a very efficient hardware solution which approximates the well known belief propagation decoder (also called the optimal or ideal message passing decoder). This solution is unique, though it is based on a solution proposed in [2].

I also implement this solution in hardware on an FPGA chip, the Xilinx Virtex-300. The source code was written in VHDL and compiled by Xilinx Foundation Series software version 2.1i. However, since the implementation was done in a high level language, it should be easily portable to other architectures, including ASIC design.

1.1 Keywords:

Error-correcting codes, Low-Density Parity Check Codes, Message-passing decoder

2 Introduction

2.1 Importance of ECC's

Today, wireless technologies are poised to become as ubiquitous as the internet is now. However, one of the technical obstacles that must be overcome for this to happen is the comparitively high power required to transmit over any large distance or in electromagnetically noisy environments. One part of the solution to this problem is to develop powerful error-correcting codes which can achieve very low error-probabilities at close to the channel capacity. To be practical, the encoding and decoding procedures should have low complexity so that they do not use too much power themselves. Low-complexity solutions also contribute to fast execution and smaller, cheaper microchips.

2.2 LDPC codes

A well-known kind of error-correcting codes is Low-Density-Parity-Check codes, first discovered by Robert Gallager in the early 1960's. These codes have what's technically known as "good" performance under message-passing decoding because for any fixed signal to noise ratio (SNR) above some threshhold, they have arbitrarily low output error probability.

3 Problem Statement

Study the structure, encoding and decoding algorithms for Low-Density Parity Check (LDPC) codes. Based on the ideal message-passing decoder described in [2], or another reasonable decoder, design a realizable decoding procedure that uses small (a few bits) messages.

Implement this design in hardware, and verify that it works by whatever means are appropriate. Create a working demonstration of the decoder which uses the physical hardware. All other components of the communication system (figure 1) shoulc be created or simulated as necessary, in order to create a working whole.

3.1 Technical Specifications

- Operating range = 4-6 db.
- Output bit-error probability $< 10^{-4}$.
- Achieve coding gain of 3 db.
- Accept soft-decision input.
- Decoding speed $>$ 1 kbps.
- Information Block length 50-200.

4 Solution

In the research phase of this project it was determined that the ideal message-passing decoder has performance far superior to several other kinds of message-passing decoders. The design is therefore built as an approximation to this algorithm. Also during this phase, a general matlab simulator was created with which different designs can easily be simulated and their performance compared.

4.1 Technical Overview

The message-passing decoder is implemented as a collection of communicating "miniprocessors." Each miniprocessor is designed to calculate soft-decision estimates of a single bit of data given a very few estimates. Such miniprocessors are connected in the form of a Tanner graph, figure 2.

There are two kinds of miniprocessors. The ones on the left of the tanner graph represent channel symbols. They recieve an estimate directly from the channel and three estimates from the parity-check nodes (right miniprocessors). The right miniprocessors recieve estimates of 6 bits of information from the channel symbol nodes (left miniprocessors) and generate 6 estimates of the same 6 information bits, each based on the other 5 incoming estimates. The precise design rule is discussed in the following section.

4.2 Hardware Design

Since each miniprocessor recieves and generates only a very few bits of information per iteration, it is possible to search the entire design space for the optimal decoder. Based on such a search, the following design rules were obtained. However, the design rules can be replaced in case better rules are later found.

4.2.1 Left miniprocessors

Left miniprocessors (figure 3) recieve a total of 4 (multi-bit) signals and generate 4 signals. The 3-bit message coming from the channel is thought to have the message alphabet $[0, 1, 2, 3, 4, 5, 6, 7]$. 0 indicates the strongest probability of that bit being 0, while 7 indicates the strongest probability of being 1. The 2-bit messages incoming from the parity check are thought of as being from the alphabet $[0, 1, 2, 3]$, and

4.2.2 Design rule of left miniprocessor

On the first iteration, only the channel code is considered. Each outgoing estimate is given as follows:

Input	Output
$0-2$	0
3	1
4	2
$5-7$	3

For the rest of the iterations, each outgoing message is calculated based on the sum of the other two incoming messages plus the channel message. The total range is $[0-7]+[0-3]+[0-3]=[0-13]$ (read: between 0 and 13). Estimates are given as follows:

Input	Output
$0-5$	0
6	1
7	2
$8-13$	3

On the final iteration, the single-bit outgoing estimate of the channel is taken as a function of the sum of all incoming estimates, which take the range $[0-16]$. The design rule is to take the estimate to be 1 if the sum is at least 8.

4.2.3 Right miniprocessors

Right miniprocessors (figure 4) recieve a total of 6 2-bit signals and generate 6 signals. Their design rule is as follows:

4.2.4 Design rule of right miniprocessor

The signals are taken to be $'1'$ or $'0'$ and 'strong' or 'weak' depending on the input value in the obvious way. The outgoing signal is $'1'$ if the sum mod 2 of incoming messages is $'1'$. Also, the outgoing signal is 'strong' if all inputs are 'strong' and 'weak' otherwise.

4.3 Hardware Verification

Although the graphs in this report are generated through matlab simulation, the equivalence of the hardware decoder with software decoder was rigorously tested. The primary means of testing was to take random signals and decode first using Matlab and then using the decoder. In these tests, the output signals matched, indicating that the two were indeed equivalent.

4.4 Alternate Solution Analysis

The obvious alternate solution is to use some kind of all software device running on a traditional microprocessor. The real disadvantage with this solution would be that it would be incomparably slower than the hardware solution which has been implemented because operations would certainly have to be serialized, leading to a slowdown from constant to linear (in n) time for decoding per iteration.

Another strong possiblity for an alternate solution would be to realize some hybrid hardware-software approach. Such an approach would utilize the design for the miniprocessors, but would use some sort of software instructions in order to permute the data. This approach still requires some thought, since the details have not been worked out yet.

5 Discussion

5.1 Overview

While several of the design criteria were not quite within the specification, I consider the project to be quite successful overall. As demonstrated in the presentation, the decoder achieves significant coding gain (4.9 db) relative to the channel error probability.

5.2 Technical Results

A summary of the technical results are shown in figure 5. Note that 3 of the specifications were not met. However, each of these 3 specifications would have been met by using a slightly longer code. Unfortunately, the decoder for a 64-bit code was the largest one that would fit onto the FPGA that was used (300,000 gates).

Looking at the right of the graph in figure 6, the top line represents the results of uncoded modulation, with which the performance of error-correcting codes is usually compared. The next line down is the implemented design. The performance begins to outperform uncoded modulation at a SNR of 2.6 db. at 6 db, the bit error rate drops to about $2.12 \cdot 10^{-4}$. The next line down is the performance of the optimal message-passing decoder. The performance of the decoder I designed is within about 0.3 db of this curve. The lowest line represents the performance of my design extended to a $n = 256$-bit code.

The most significant shortcoming of this project was that only 64-bit codes were able to be implemented. This corresponds to only 64 left-miniprocessors and 32 right-miniprocessors being synthesized onto the FPGA board. The goal was to achieve at least a 100 bit code. However, the results could not be improved without either buying a larger and more expensive FPGA or using more powerful tools to re-design the miniprocessors at a much lower level.

If the design were to have accomodated a 256-bit codeword, it would have handily achieved all of the specifications at a SNR of 5 db.

One last relevant design criteria is the comparison of the discrete design to the ideal message passing decoder. In fact, in all the cases that were tested, the design performs within 0.5 db of the ideal message-passing decoder, which corresponds to only about 12 percent more power transmitted.

5.3 Remarks

I sincerely believe that this decoder design could not only be practical, but commercially viable. While there are certainly architectural enhancements to be made, such as extending it to a hybrid hardware/software solution, this project has demonstrated the solution's feasibility.

6 Cost Analysis

Item	Cost
Xess XSV prototyping board	$899
XSV-300 chip	$300
Development software (Xilinx Foundation v2.1)	$95

Further development would likely require tens of thousands of dollars for development and millions of dollars (at least) for full-time production of the ASIC design.

7 Patents

There do not appear to be any patents on this material either granted or pending. As far as the future of this work, I plan to use this work as constituting prior art if I may possibly do so, in order to prevent patents from being granted in this area. I do not (currently) intend to apply for a patent myself.

7.1 Open source

I plan to make this project open-source, and will licence all of the source code under the BSD licence.

8 Conclusions

8.1 Impact

I do not know what impact this will have, but the potential is certainly quite high. In terms of speed of decoding, this simple FPGA implementation already rivals the fastest decoders that I have heard of. Practically, the design will have great value only when more nodes, corresponding to longer codes, can be implemented, since short codes still perform far away from the theoretical bounds for message-passing decoding.

8.2 Demonstration

The demonstration (figure 7), to be performed on June 14 will use a 24 bit windows bitmap (.bmp) file. A program has been written which reads one such file and writes two such files. The input file is the uncorrupted image, which is broken into 32-bit segments and then encoded into 64-bit codewords. Then, the image is corrupted and sent to the XSV board for decoding.

9 References

References

[1] T. Richardson et al., "Design of Provably Good Low-Density Parity Check Codes" submitted IEEE IT.

[2] T. Richardson, Rudiger Urbanke, "The Capacity of Low-Density Parity Check Codes under Message-Passing Decoding"

[3] M. Luby, M. Mitzenmacher et al., "Analysis of Low Density Codes and Improved Designes Using Irregular Graphs," in *Proceedings of the 30th annual ACM Symposium of Theory of Computing,* pp. 249–258, 1998.

[4] David J. C. MacKay, "Good Error-Correcting Codes Based on Very Sparse Matrices" in *IEEE Transactions on Information Theory, Vol. 45 No. 2 March 1999.*

10 Appendix

10.1 Appendix 1–list of figures

10.2 Appendix 2–contents of project disk

- readme.txt –file explaining how to compile for, load, and communicate with the XSV board.
- fpga/ –vhdl source code for the FPGA (Virtex PQ300) chip on the XSV board, with other required files.
- cpld/ –vhdl source code for the CPLD chip on the XSV board, with other required files.
- cpp/ –c++ source code for the program that communicates with the board.
- matlab/ –matlab code for generating test vectors and simulating the quantized and ideal message passing decoding algorithms.
- tex/ –this report, in LaTeX format.
- presentation/ –powerpoint presenatation.