Hardware Architecture for Modified Sequential LDPC Decoder

ANGUS WU and W. L. LEE
Department of Electronic Engineering
City University of Hong Kong
Tat Chee Avenue
HONG KONG

Abstract: - Iterative decoding of Low Density Parity Check (LDPC) codes using the Parity Likelihood Ratio (PLR) algorithm have been proved to be more efficient compared to conventional Sum Product Algorithm (SPA). However, the nature of PLR algorithm tends to put numerious pieces of data to this decoder and perform computation intensive operations, which is a major challenge for building a practical real-time LDPC decoder. In this paper, it makes use of extrinsic information clipping and calculation step merging techniques to simplify the decoding algorithm. These approaches reduce the number of quantization levels while still remaining the algorithmic performance promised by random codes. Moreover, a modified sequential architecture is proposed for LDPC decoding that decreases the decoding latency and reduces the memory storage compared to existing direct sequential design. Simulation results show that the proposed architecture results in time and memory savings of up to 92.26% and 59.56% respectively over conventional direct sequential implementation.

Key-Words: - LDPC codes, sequential architecture, VLSI implementation, PLR algorithm

1 Introduction

Turbo codes [1] and LDPC codes [2] are the two best known codes that are capable of achieving low bit error rates (BERs) at low signal to noise ratios (SNRs) [3][4]. They are recent breakthroughs in coding theory that promise to push the areal density of the magnetic recording channel to its limits [5]. LDPC codes were proposed by Gallager in 1962 [2], and the performance is very closed to the Shannon limit [6]. However, LDPC codes were not pursued for due to implementation complexity [7]. Nevertheless, the interest in iterative decoding algorithms has led to rediscover of LDPC codes [7] by MacKay and Neal [8], [9]. Like turbo codes, LDPC codes also belong to the general class of powerful concatenated codes that employing pseudo-random encoders and iterative decoders [10].

LDPC codes are similar to turbo codes in many aspects but are widely considered as serious competitors to turbo codes [11] in terms of performance and complexity as well as their similar philosophy bases: constrained random code ensembles and iterative decoding algorithm [12]. Recent advances in error correcting codes (ECCs) have shown that irregular LDPC codes can achieve reliable transmission at SNRs extremely close to the Shannon limit on the additive white Gaussian noise (AWGN) channel, outperforming turbo codes of the same block size and code rate [13].

There are some variation of LDPC decoding algorithm, Sum Product Algorithm (SPA) and Parity Likelihood Ratio (PLR) algorithm. Although all current implementations of the decoder employ SPA for decoding, direct implementation of SPA can be very sensitive to the quantization effect. It has been proven that PLR technique can greatly reduce the quantization level requirement [14], which leads to significant reduction in decoding costs. As the horizontal step of PLR algorithm only involve look up table for the PLR function, their respective realization are straightforward and involve only table searching.

LDPC code applications become more on handling parity check in communication systems such as cellular mobile phone and video conferencing which requires real-time encoding or decoding of transmitted data. However, the randomness of LDPC code and large amount of intermediate processing data results in stringent memory requirements that amount to an order of magnitude increase in complexity [4]. Therefore, reducing the size of decoder becomes an increasingly concern for feasible VLSI implementation.

A direct approach for implementing a sequential decoder architecture [16] would be to alleviates the use of complex operations, apply in-place algorithm, synchronise timing signal, incorporate address counter and look-up tables for simplifying combinational arithmetic, solving latency problem due to interleaving process, utilising memory

modules and further optimizing chip area. Besides, it takes the most significant bit in output section to eliminate exponent calculation and employs simple decision logic at output section for termination of iteration. These simplified the architecture complexity and results in a sub-optimal decoder design. However, a major problem with this approach is that even though many simplifications have been made, the size of chip is still significantly large due to the memory requirement and the big finite state machine. In addition, the 6-bit architecture has been shown to be only marginally satisfied with the specification of real-time transaction. Therefore, both decoding time and chip size is the main drawback of the direct sequential architecture for practical VLSI implementation.

In this paper, we have taken into account the clipping of extrinsic information [14] and combining horizontal backward step with extrinsic information calculation to simplify decoding process. Here, in our proposed architecture, we modify the direct sequential decoder with the clipping and merging technique to boost decoding performance. Simulation results show that the proposed architecture is more efficient and less storage requirement than the direct sequential one.

The rest of paper is organized as follows. Section 2 presents the two algorithm simplification approaches of PLR algorithm for decoding LDPC codes. Then, Section 3 proposes a new optimized version of modified sequential architecture for reducing decoding time and memory requirement. Next, Section 4 presents some simulations results followed by some concluding remarks provides in Section 5.

2 Algorithm Simplifications

PLR algorithm [14] is an efficient decoding algorithm for LDPC code. It is obtained by changing the subtraction operation into division on the probabilities in the original SPA. In PLR algorithm, there are several decoding steps including initialization, updating, horizontal forward step, vertical backward step, vertical forward step, horizontal backward step, extrinsic information calculation and output step. Among these steps, except initialization and output step, are carried out iteratively.

2.1 Extrinsic Information Clipping

By incorporating the clipping methodology into the sequential architecture, both low complexity and low decoding latency can be achieved. Clipping is an algorithm that the index of extrinsic information is clipped between +7 and –7 inclusive before updating the whole dimension. During the whole iterative calculation process, if clipping algorithm is applied, then only 15 iterations are enough to decode a block of data. More important, the decoder can operated in logarithmic domain using 5-bit sign-magnitude representing quantization level index ±0, ±1, ±2, …, ±15. This 1-bit reduction in architecture not only decrease the size of finite state machine inside the control unit but also reduce the size of look-up table by four times. Although it seems that extra clock cycle are needed for handling clipping process, there exist some wait states in the original sequential design. Those extra clock cycle for clipping can be fitted into those wait states so as to maintain the decoding time. Therefore, the decrease in quantization levels and number of iterations becomes more significant in reducing hardware requirements.

2.2 Decoding Steps Merging

The operations of horizontal backward step and extrinsic information calculation on the decoding process are two of the computation steps inside iterative cycle. Therefore, combining of horizontal backward step

$$\hat{v}_m = \begin{cases} b_m & if \;\; m = 1 \\ f(a_{m-1}, b_m) & otherwise \end{cases} \quad (1)$$

and extrinsic information calculation

$$u_{m,n} = d_{m,n} \cdot f(\hat{q}_{m,n}, \hat{v}_m) \quad (2)$$

can be rewritten in the following form.

$$u_{m,n} = \begin{cases} q_{m,n} \cdot f(\tilde{q}_{m,n}, b_m) & if \;\; m = 1 \\ q_{m,n} \cdot f(\tilde{q}_{m,n}, a_{m-1}, b_m) & otherwise \end{cases} \quad (3)$$

Then, there is no need to store the intermediate variable $\hat{v}_m$ of horizontal backward step. The PLR result *f(.)* of horizontal backward step can be temporary stored in the registers of control unit during calculation of extrinsic information $u_{m,n}$ as in other steps. As a result, merging equation not only reduce memory storage for intermediate variables but also further minimize the size of finite state machine due to the decrease in number of read-write cycles.

3 Implementation of Modified Sequential Architecture

Due to the inherent sequential nature of the decoding algorithm, sequential architecture only consists of a small number of units. The architecture has a control unit for controlling enable and read-write signal of memory modules. RAMs are used for storing data

and intermediate variables while ROMs are design for holding look-up tables and shuffle rule. The design of the interconnection of these blocks is simplified to take advantage of information exchange so as to increase the decoding efficiently.

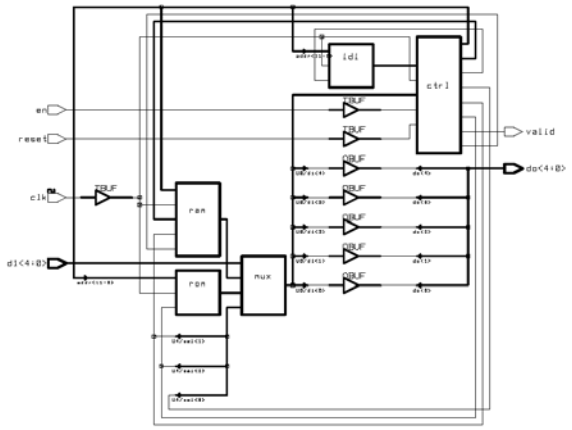

Fig. 1 Block diagram of sequential decoder architecture for LDPC code.

The block diagrams are illustrated in Fig. 1. It implements LDPC decoder by employing PLR algorithm and realizing 4 dimensions. The synthsized LDPC code block has a total of 1024 data and 256 parity checks, where each parity check computed using entries from 4 information.

3.1 Input and Output Section

In each iteration, there are four equal time slots for performing large amount of calculations in the four dimensions. However, the first iteration is not for calculation but for input and output data. The output step will process when the signal of first iteration is asserted. Resulting value will be outputted after the decoding of the last dimension in the last iteration before next decoder input. The resulting values can be reduced by only taking the most significant bit. This design reduces the exponent computation to convert the indices back from logarithmic domain. After that, information and parity will be inputted to the decoder respectively. Once data are inputted from ADC to decoder, the updating step and the horizontal forward step can be carried out simultaneously.

3.2 Interleavers and Deinterleaver

The randomness of the interleaver output sequence makes it difficult to realize in low complexity combinational circuit. A direct interleaver implementation uses two banks of buffers alternating between read and write for consecutive sectors of data. The latency through an interleaver is therefore equal to the block size [15].

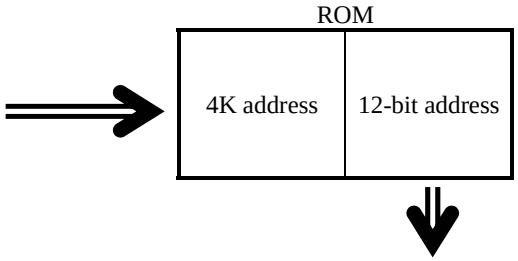

Fig. 2 Interleavers and deinterleaver implemented using ROM.

The basic block interleaver design uses a minimal amount of control logic. Using ROM for high-speed implementation, the interleaver inputs are data position in current dimension, while outputs are data position in previous dimension. In updating step, when reading in shuffled extrinsic information from last dimension, the RAM address is read from ROM data and the ROM address is read from control unit. This decreases the latency problem due to data interleaving since a long duration stage for data shuffle process is eliminated. The read-read operations are then repeated alternating between ROM and RAM as in sequential design. More sophisticated interleaver designs yield improved error rate performance, but result in increased implementation complexity [15]. Therefore, the implementation of the described basic interleaver provides a lower limit on complexity [15].

3.3 Control Unit

In the control unit, timing controller, iteration controller and dimension controllers are responsible for implementing decoding steps recursively. For that reason, iteration controller is to activate one necessary dimension controller at a time and pass iteration number for dimension controller to use. The whole control unit incorporates simple decision logic that uses a sign-controlled signal from the timing controller to indicate the first and final iteration of a data block. A simple state machine like the one shown in Fig. 3 is used to maintain the state of each iteration in the decoder. A cycle can be in one of four states: output state will output results of the last dimension which are stored in memory module. Upon that, received data moves into the memory to overwrite previous data block in the input state and all extrinsic information variables will be

reinitialized in the initialization state. This ends the first iteration.

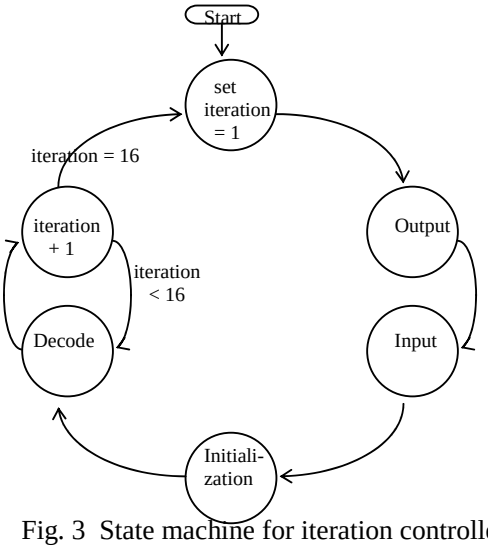

Fig. 3 State machine for iteration controller of the control unit.

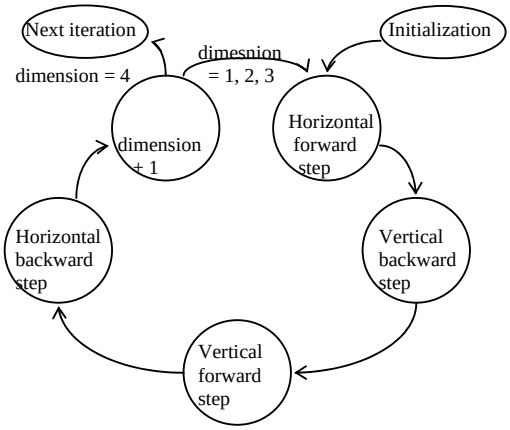

Fig. 4 State machine for dimension controller of the control unit.

Started from second iteration is the data processing state, it follows the state machine as shown in Fig. 4. For sequential implementation, four states are carried out consecutively. If it reaches maximum number of dimensions, it will carry on to next iteration and start horizontal forward state without re-initialize any variables. Until it reaches maximum number of iterations, it will back to output state and repeats the same cycle. Moreover, dimension controller, which incorporating a big finite state machine, for memory read-write controlling is employed so that all control signals employed are being well matched and synchronized.

With the proposed methodology, the timing controller, iteration controller and four dimension controllers can be combined into one single control unit. It consists only one finite state machine but perform the same operation and have the same decoding effect as the direct sequential architecture.

```
for all iterations {
  if (first iteration) {
    output decoded data block;
    input received data block;
    initialize extrinsic information;
  }
  else {
    for all dimensions {
      for all rows {
        registers ← shuffled information from last dimension –
                    extrinsic information from current
                    dimension;
        for all columns {
          temporary RAM block ← f(all data in the row except
                                  current column);
        }
        d̂ ← f(all data in the row);
      }
      a255 = d̂255
      for (row = 254 downto 0) {
        arow ← prow+1 + f( d̂row , arow+1);
      }
      b0 = p0 + d̂0
      for (row = 1 to 255) {
        bi ← pi + f( d̂row , brow-1);
      }
      for all rows {
        register ← q add with clipped f( q̂ , a, b);
        extrinsic information ← register – q;
        q ← register;
      }
    } // end for
  } // end if
} // end for
```

Fig. 5 Implementation of PLR algorithm in the control unit with processing details abstracted.

In iterative programs, like LDPC code decoding, execution proceeds as a sequence of sequential iterations, where at each iteration all parallel processes corresponding to logical function and variables can execute independently, but each logical function then needs to communicate values computed during that iteration with other variables it is connected, before it can commence its next iteration. As shown in Fig. 5, the control flow is done in such a way that every iteration, a logical function sends data to its logical neighbours and then waits until it receives messages back from all of next iteration to any of these neighbours.

3.4 Memory

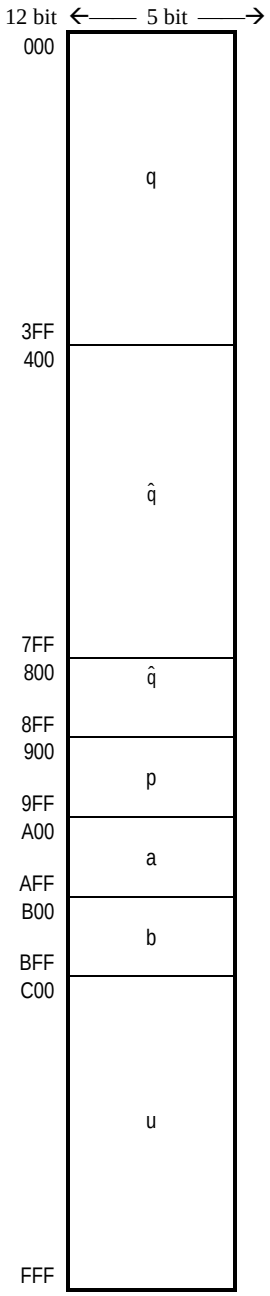

Fig. 6 RAM configuration for one dimension.

Although there are a large number of intermediate variables, some values that serve as local variables, which will not be referenced again in the next dimension or iteration, can share a temporary register inside the control unit. According to this motivation, the size of memory can be reduced so that the overall average cost of implementation can be minimized.

Both data and intermediate results will be stored into the RAM. The allocation of variables referred to as a memory map for a RAM is shown in Fig. 6. The memory map allows a RAM that performs decoding to become switching received data, intermediate variables and extrinsic information. The switch happens simply by telling the control unit to execute at a given location in the RAM. Treating intermediate variables in the same way as decoding data greatly simplifies the RAM address calculation in the control unit. Fig. 7 shows the formats of RAM address to connect the fields of decoding step to the algorithm.

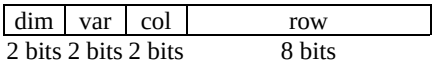

Fig. 7 RAM addrress format.

Each iteration share same memory space in the RAM. This sharing is made possible by not assigning iteration field in the range of RAM address. The "dim" field is contained in bits 13~12. The 8-bit row number is in positions 7~0. The data and intermediate variables to be read or write are specified by "var" fields at position 11~10. The column number is in bit position 9~8. However, this is not true for field var=10_2 while the "col" field will be used to indicate intermediate variables to be access as well. The two kinds of datapath can then use one address format.

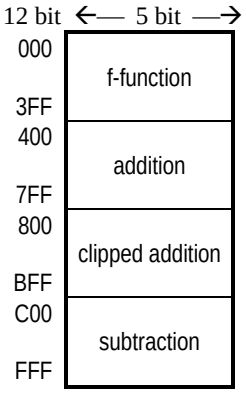

Fig. 8 ROM configuration.

Look-up tables are resided in ROM as shown in Fig. 8. The four kinds of operation in logarithm domain that it implements are f-function, addition, clipped addition and subtraction. Starting top down, the f-fucntion starts at 000_{16}. At the other end, the subtraction starts at $C00_{16}$. The addition starts at 400_{16}. Clipped addition is next and it can look up from 800_{16} to BFF_{16}.

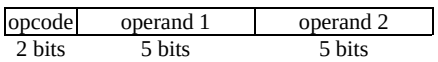

Fig. 9 ROM address format.

The ROM address format is set to make it easy to perform table look-up. It is concatenated by a 2-bit opcode and two 5-bit operands as shown in Fig. 9. The 5-bit operand fields are sign-and-magnitude notation and the look-up result is also 5-bit sign-and-magnitude index in logarithm domain. Address format for f-fucntion operation, which has an opcode of 00_2, can be implemented by a combinational circuit. Then, the opcode becomes a select signal input to the multiplexer for choosing between combinational result and ROM output.

4 Simulation results

The direct sequential and the proposed modified sequential decoder were synthesized with Synopsys computer aided design tool using 0.38 microns technology under 3V supply based on a 1024 bit, rate-1/2 LDPC code. This corresponds to one of the block sizes and code rates proposed for 3G wireless turbo codes. In our simulation, we adopt 4 dimensions of each 256 rows and 4 columns with 16 iterations including one iteration for input-output section. The effect of using clipping and merging under Virtex implementation is shown in Fig. 10 and Fig. 11.

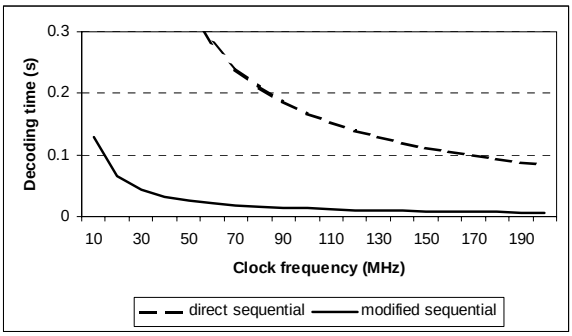

Fig. 10 Performance of direct and modified sequential LDPC decoder.

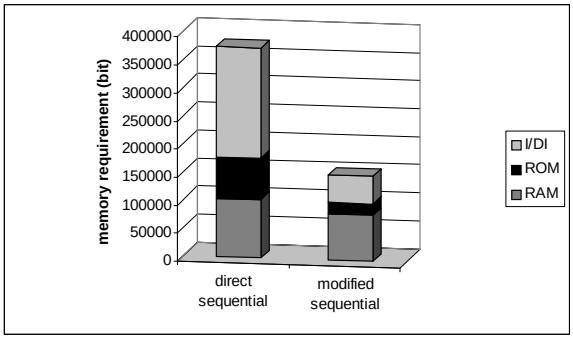

Fig. 11 Memory requirement comparison between direct and modified sequential architecture.

It can be observed that the performance of sequential decoder with the proposed modified scheme is very superior to that of the original direct case. The total decoding time of the modified decoder is 1298432 clock cycles which was 92.26% less than the direct sequential decoder. In addition, the memory requirement of the modified one was 151552 bits which was 59.56% less than the direct sequential one. The reduction in control logic implies that less silicon is required for implementing such decoder. Simulation results suggested that the proposed decoder is more efficient in terms of decoding speed and chip area. As far as both speed and area is concerned, the proposed architecture shortens the decoding latency and requires less memory storage. Hence, it is more feasible in real-time mobile communication applications.

5 Conclusions

This paper presented a modified architecture for sequential LDPC decoder. The decoder is simulated and synthesized with Synopsys computer aided design tool for Virtex implementation. We proposed incorporate extrinsic information clipping and calculation step merging into a sequential architecture to achieve high performance low requirement decoder. By comparing the simulation results, it can be observed that with the proposed scheme, the decoding speed is nearly doubled and the required memory storage is nearly halved under 3V supply.

Acknowledgement:
This work is supported by CityU PAG grant 7100049.

References:

[1] C. Berrou, A. Glavieux and P. Thitimajshima, "Near Shannon Limit Error-correcting Coding and Decoding: Turbo Codes," *IEEE ICC*, 1993, pp. 1064-1070.

[2] R. G. Gallager, "Low Density Parity Check Codes," *IRE Transactions on Information Theory*, vol. 8, January 1962, pp.21-28.

[3] Igal Sason, Shlomo Shamai, "Improved Upper Bounds on the Ensemble Performance of ML Decoded Low Density Parity Check Codes,"

IEEE Communications Letters, vol. 4, no. 3, March 2000, pp. 89-91.

[4] Mohammad M. Mansour and Naresh R. Shanbhag, "Low-power VLSI Decoder Architectures for LDPC Codes," *Proceedings of the International Symposium on Low Power Electronics and Design*, 2002, pp. 284-289.

[5] Thomas Mittelholzer, Ajay Dholakia and Evangelos Eleftheriou, "Reduced-complixity Decoding of Low Density Parity Check Codes for Generalized Partial Response Channels," *IEEE Transactions on Magnetics*, vol. 37, no. 2, March 2001, pp. 721-728.

[6] Hisashi Futaki and Tomoaki Ohtsuki, "Low-density Parity-check (LDPC) Coded OFDM Systems," *IEEE Vehicular Technology Conference*, vol. 1, 2001, pp. 82-86.

[7] Chris Howland and Andrew Blanksby, "A 200mW 1Gb/s 1024-bit Rate-1/2 Low Density Parity Check Code Decoder," *IEEE Conference Custom Integrated Circuits*, 2001, pp. 293-296.

[8] D. J. C. Mackay and R. M. Neal, "Near Shannon Limit Performance of Low Density Parity Check Codes," *Electronic Letters*, vol. 32, August 1996, pp. 16454-1646.

[9] D. J. C. Mackay and R. M. Neal, "Near Shannon Limit Performance of Low Density Parity Check Codes," *Electronic Letters*, vol. 33, March 1997, pp. 457-458.

[10] Ping Li and Keying Y. Wu, "Concatenated Tree Codes: A Low-complexity, High-performance Approach," *IEEE Transactions on Information Theory*, vol. 47, no. 2, February 2001, pp. 791-799.

[11] Tong Zhang, Zhongfeng Wang and Keshab K. Parhi, "On Finite Precision Implementation of Low Density Parity Check Codes Decoder," *IEEE International Symposium on Circuits and Systems*, vol. 4, 2001, pp. 202-205.

[12] Thomas J. Richardson and Rudiger L. Urbanke, "Efficient Encoding of Low-density Parity-check Codes," *IEEE Transactions on Information Theory*, vol. 47, no. 2, February 2001, pp. 638-656.

[13] Jilei Hou, Paul H. Siegel, Laurence B. Milstein, "Performance Analysis and Code Optimization of Low Density Parity-check Codes on Rayleigh Fading Channels," *IEEE Journal on Selected Areas in Communications*, vol. 19, no. 5, May 2001, pp. 924-934.

[14] Ping Li and W. K. Leung, "Decoding Low Density Parity Check Codes With Finite Quantization Bits," *IEEE Communications Letters*, vol. 4, no. 2, February 2000, pp. 62-64.

[15] Engling Yeo, Payam Pakzad, Borivoje Nikolic and Venkat Anantharam, "VLSI Architectures for Iterative Decoders in Magnetic Recording Channels," *IEEE Transactions on Magnetics*, vol. 37, no. 2, March 2001, pp. 748-755.

[16] W. L. Lee and Angus Wu, "VLSI Implementation for Low Density Parity Check Decoder," *Proceedings of IEEE International Conference on Electronics, Circuits and Systems*, September 2001, pp. 1223-1226.