VLSI IMPLEMENTATION FOR LOW DENSITY PARITY CHECK DECODER*

W. L. Lee and Angus Wu

Electronic Design Automation Center
Department of Electronic Engineering
City University of Hong Kong
Tat Chee Avenue, Kowloon Tong, Kowloon, Hong Kong.
Email: angus.wu@cityu.edu.hk, Tel: +852-27889391, Fax: +852-27887791

ABSTRACT: In this paper, a low complexity digital Low Density Parity Check (LDPC) turbo code decoder architecture for real-time cellular personal communication application is presented. The proposed VLSI decoder architecture alleviates the use of complex operations such as combinational arithmetic, exponent computations and reduce intermediate storage as well as interleaving latency by incorporates in-place algorithm, index look-up table and address counter. Besides, output section and termination of iteration are implemented by simple decision logic. The entire decoder is designed and synthesized using Synopsys VHDL computer aided design tool.

1. INTRODUCTION

Turbo codes are a new class of error correction codes that were introduced along with a practical decoding algorithm in [1]. The importance of turbo codes is that they enable reliable communications with channel capacity efficiencies close to the theoretical limit predicted by Claude Shannon in [2]. This constituted a significant gain in efficiency over other coding techniques known at the time [3]. Low Density Parity Check (LDPC) turbo code was discovered to represent a significant breakthrough in coding over the past several years because of its amazing error correcting capability. Therefore, this type of code has been very attractive for applications in real-time digital communication. However, there are relatively lack of basic papers discussing the architecture implementation in this topic. This paper proposes a VLSI implementation on the digital decoding part. To verify its functionality, simulations and synthesis were performed using Synopsys VHDL.

After a brief review on LDPC code decoding algorithm in Section 2, the proposed VLSI implementation of the decoder is described in Section 3. The simulation results are discussed in Section 4 followed by conclusion in Section 5.

2. LDPC DECODING ALGORITHM

Error correction coding scheme is a necessary part of most real-time cable and wireless digital communication systems nowadays. Figure 1 shows the overall functional block diagram of the communication system. In the transmitter side, turbo encoder generates a parity matrix $P_1 = (p_1^{(1)}, \ldots, p_m^{(1)})$ with m rows associated with an information matrix (or known as dimension) $D_1 = (d_{1,1}^{(1)}, \ldots, d_{m,n}^{(1)})$ with m rows and n columns where $p_m = d_{m,1} \oplus \ldots \oplus d_{m,n}$. Similarly, another parity matrix $P_2 = (p_1^{(2)}, \ldots, p_m^{(2)})$ associated with the interleaved dimension $D_2 = (d_{1,1}^{(2)}, \ldots, d_{m,n}^{(2)})$ is generated where information bits in D_2 are the same in D_1 with bit position shuffled. Until parity matrices of all j dimensions are generated, matrices D_1 and P_1, ..., P_j are assumed to be transmitted over an additive white Gaussian noise (AWGN) channel.

In the receiver side, both coded information and parity signal are inputted to the Analog-to-Digital-Converter (ADC) at the very beginning. ADC provides necessary process for the digitization before decoding process. The corresponding received matrices after digitized and quantized are $Q = (q_{1,1}, \ldots, q_{m,n})$, $V_1 = (v_1^{(1)}, \ldots, v_m^{(1)})$, ..., $V_j = (v_1^{(j)}, v_2^{(j)}, \ldots, v_m^{(j)})$.

Parity Likelihood Ratio (PLR) algorithm [4] is a fast decoding method for LDPC codes. It is obtained by changing the subtraction operation into division on the probabilities in the original Sum Product Algorithm [5]. The equations for generating extrinsic information $u_{m,n}$ in PLR algorithm could be summerized as the following steps:

- **Initialization:** Initially, the decoder assigns the quantized information Q to dimension Q_1, ..., Q_j with the same predefined shuffle rule as encoder. The quantized parities will be V_1, ..., V_j accordingly.

$$u_{m,n}^{(j)} = 1 \quad \forall\, m,n,j$$
$$i = 1$$
$$j = 1$$

where i is the iteration number and j is the dimension number.

- **Updating:** Calculate $q_{m,n}$ as

$$q_{m,n} = q_{m,n} / u_{m,n}$$

- **Horizontal Forward Step:** Compute the intermediate variables $\tilde{q}_{m,n}$ and $\hat{q}_m$ as

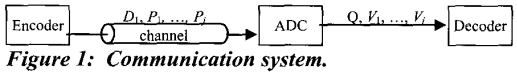

Figure 1: Communication system.

* This work is supported by CityU PAG grant 7100049.

0-7803-7057-0/01/$10.00 ©2001 IEEE.

$$\tilde{q}_{m,n} = f(q_{m,1},\ldots,q_{m,k},\ldots,q_{m,n}) \quad for\ k \neq n$$
$$\hat{q}_m = f(q_{m,1},\ldots,q_{m,n})$$

where k is the current column number.

- **Vertical Backward Step:** Update a_m as

$$a_m = \begin{cases} v_m & if\ i=1, m=M \\ v_m & if\ i \neq 1, j=J, m=M \\ v_m \cdot f(\hat{q}_m, a_{m+1}^{(i-1,j+1)}) & if\ i \neq 1, j \neq J, m=M \\ v_m \cdot f(\hat{q}_m, a_{m+1}^{(i-1,j)}) & otherwise \end{cases}$$

where M and J is the last row in each dimension and last dimension in each iteration respectively.

- **Vertical Forward Step:** Update b_m as

$$b_m = \begin{cases} v_m \cdot \hat{q}_m & if\ j=1, m=1 \\ v_m \cdot f(\hat{q}_m, b_{m-1}^{(i,j-1)}) & if\ j \neq 1, m=1 \\ v_m \cdot f(\hat{q}_m, b_{m-1}^{(i,j)}) & otherwise \end{cases}$$

- **Horizontal Backward Step:** Compute the intermediate variables $\hat{v}_m$ as

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

- **Extrinsic Information:** Calculate $u_{m,n}$ as

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

- **Output:** At completion,

$$u_{m,n} = \begin{cases} 0 & if\ u_{m,n} > 1 \\ 1 & otherwise \end{cases}$$

In the above equations, the Parity Likelihood Ratio function f is defined by $f(x, y) = (1 + x\ y) / (x + y)$. Among these steps, except initialization and output step, are carried out recursively in dimension and then iteratively. That is the first dimension attempts to calculate the extrinsic information which would be interleaved and be applied to the second dimension. Then, second dimension will compute another extrinsic information matrix and so on, until the j[th] extrinsic information matrix is obtained. For the next iteration, the j[th] extrinsic information block is deinterleaved and passed to the first dimension for calculation.

3. LDPC DECODER IMPLEMENTATION

Figure 2 shows architecture of the proposed decoder. After the ADC process, the information and parities are quantized into 64 levels with base 2. To avoid complex floating point calculation, the decoder is operated in logarithmic domain using 6-bit sign-magnitude representing level index ±0, ±1, …, ±31. During calculation of PLR, a 64 × 64 look-up table on level indices is used. This method is very fast since it saves a lot of calculation step and simplifies horizontal steps to only involve table searching.

This decoder has 4 dimensions (j = 4) of each 256 rows (m = 256) and 4 columns (n = 4) with 16 iterations (i = 16). The reason for choosing this combination of parameters is that it requires less resources and can achieve bit error rate (BER) the closest to unquantized decoding scheme [4].

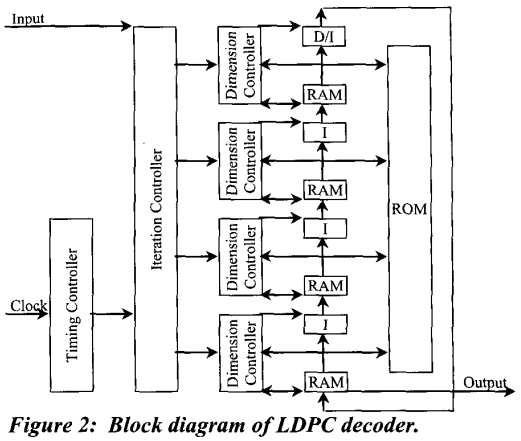

Figure 2: Block diagram of LDPC decoder.

3.1 Input section

Level index of information and parity matrices are assumed to be inputted in the first iteration at a period of 64 clock cycles. This is because once data are inputted from ADC to decoder, the updating step and the horizontal forward step can be carried out simultaneously.

3.2 Memory modules

The proposed memory design are tailored for efficient handling of input and calculation concurrently. The algorithm can be categorized into three similar operations:

- Division on input data $q_{m,k}$ and extrinsic information $u_{m,k}$, the result is put back to $q_{m,k}$.
- Apply f-function on $q_{m,k}$ and $\tilde{q}_{m,k}$, the result is put back to $\tilde{q}_{m,k}$.
- Apply f-function on $q_{m,k}$ and $\hat{q}_m$, the result is put back to $\hat{q}_m$.

These three operations can be performed in parallel due to their similar nature with same input data $q_{m,k}$. Moreover, an in-place algorithm is applied. In-place is an algorithm that destination of result occupied the same storage locations as source data. Therefore, no additional memory is required for temporarily storing the intermediate value of $q_{m,n}$, $\tilde{q}_{m,n}$ and $\hat{q}_m$.

	RAM 1	RAM 2			RAM 3
		column 1	column 2	column 3	column 4	
phase 1	A=A/$u_{m,1}$ (column 1 stored)		A	A	A	A
phase 2	B=B/$u_{m,2}$ (column 2 stored)	B		B=f(B,A)	B=f(B,A)	B=f(B,A)
phase 3	C=C/$u_{m,3}$ (column 3 stored)	C=f(C,B)	C=f(C,A)		C=f(C,B) (data 4 stored)	C=f(C,B)
phase 4	D=D/$u_{m,4}$ (column 4 stored)	D=f(D,C) (data 1 stored)	D=f(D,C) (data 2 stored)	D=f(D,B) (data 3 stored)		D=f(D,C) (data stored)

Figure 3: Operation timing of the memory module.

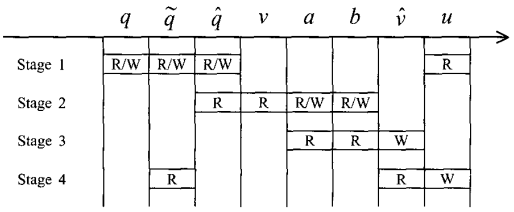

Figure 4: Timing of dimension updating.

In Figure 3, the operations performed on the three memories are indicated for a sequence of four phases, where four phasès represent the processing of four incoming index A, B, C, D inputted to one row of the three RAMs. They are RAM 1 for storing $q_{m,n}$, RAM 2 for storing $\tilde{q}_{m,n}$ and RAM 3 for storing $\hat{q}_m$.

From updating step to extrinsic information step, the whole operation applied to a dimension can be divided into four stages as shown in Figure 4. The R/W section in the figure are implemented as in-place algorithm. After the four phases finished in a dimension, stage 1 completed. Next, other 3 intermediate values a_m, b_m and $\hat{p}_m$ will be calculated simultaneously. The extrinsic information can start calculation in stage 4 once intermediate values are completely calculated in stage 2 and 3. Thus, only 16 stages are required for updating all four dimensions in one iteration.

The key to the sequential nature of memory location access is the address generator in dimension controller. The generator is designed as a 14-bit binary counter. An address generated can be seen as concatenation of four segments, dimension number (2 bits), stage number (2 bits), row number (8 bits) and column number (2 bits). It should be noted that in stage 3, a_m values are processed in reverse order. The inverted row number segment of the address eliminates the additional down counting logic in reverse address order generation.

3.3 Interleavers and deinterleaver

It has been proven that, at least for high SNR, the best performances are obtained with randomly generated interleaving patterns [6]. This means the required sequence of addresses cannot be obtained through simple computations, but quite large read-only memories (ROMs) must be allocated [6].

Therefore, the ROM size can be expressed in terms of the interleaving word length. In Figure 5, ROM address represents destination address for writing data in present dimension while ROM data stores shuffled address for reading data from 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 address generator. This decreases the latency problem due to data interleaving since a long duration stage for data shuffle process is eliminated. This operations are then repeated alternating between ROM and RAMs.

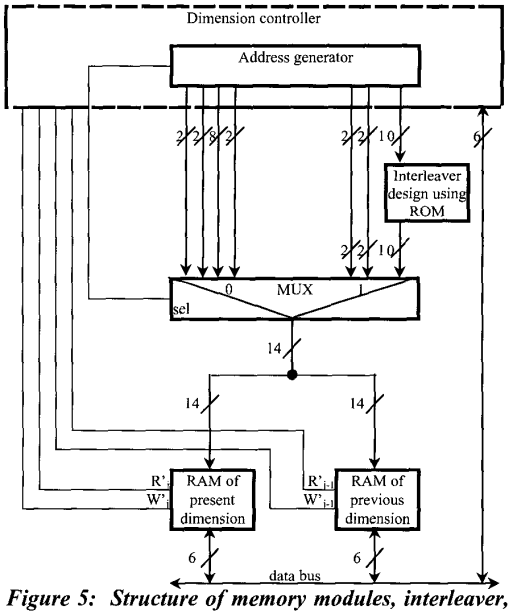

Figure 5: Structure of memory modules, interleaver, dimension controller and address generator unit.

3.4 Control unit

Besides address generation, dimension controller also plays an important role in memory read-write controlling. Because of the need to continuously applying f-function to the received data following the order of phase and stage within a dimension, memories and look-up tables require a lot of read-write and chip-select signals. A dimension controller incorporating a big finite state machine is introduced so that all control signals employed are being well matched and synchronized. A small portion of its state table is shown in Table 1.

The function of iteration controller is to activate one necessary dimension controller at a time and pass iteration number for dimension controller to use. The architecture 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.

Table 1: Sequence of control signal generated by dimension controller in Figure 5.

Control signal generated					Meaning of state in dimension controller	
sel	R'_{j-1}	W'_{j-1}	R'_j	W'_j	Operation	Memary accessed
1	0	1	1	1	Read from	Prior dimension with predefined shuffle address
0	1	1	1	0	Write to	Present dimension with normal counting address
0	1	1	0	1	Read from	Present dimension with normal counting address
0	1	1	1	0	Write to	Present dimension with normal counting address

According to the clock signal inputted as shown in Figure 2, timing controller will generate clock signals with clock period is the multiple of power-of-2. Among these signals, there are three main synchronize clock signals in the architecture. One is equivalent to the input clock signal. It has the highest clock rate and is used to control the state sequence time. Another clock signal with period 64 times input clock is the same as input data frequency. It is for address generation in each dimension. The other clock signals are for iteration number and dimension number generation.

3.5 Output Section

The output step will process when the signal of final iteration is asserted. Resulting value will be outputted at stage 4 of the last dimension in the last iteration with data rate the same as decoder input. The resulting values can be reduced by only taking the most significant bit. This design reduces the exponent computation to convert the index back from logarithmic domain.

4. SIMULATION RESULTS

The proposed design are simulated with Synopsys CAD tools using 0.35μm CMOS standard cell library at the supply voltage of 3.0V. From the synthesis, the worst case operating frequency is 85MHz. Since time for processing the whole data block is 16777216 clock cycles, the data rate inputted to the decoder should be 1.328125 MHz. Then, the time for decoding a data block becomes 0.197 seconds which is less than 0.2 seconds and is feasible in real-time application.

5. CONCLUSION

Turbo codes represent an important advancement in the area of real time communications. The extraordinary performance of Low Density Parity Check (LDPC) turbo code is due to the combination of parallel concatenated coding, recursive encoders, pseudo-random interleaving, and iterative decoder structure. However, there are some drawbacks associated with such decoders for practical VLSI implementation. The main drawback is that the decoding algorithms are very complicated because they typically require many complex operations such as combinational arithmetic, exponent and logarithmic calculus as well as an extensive iterative process [8]. Furthermore, a very long duration of data interleaving processes is needed for achieving the performance as advertised [7]. In addition, the decoders typically require a large memory for intermediate storage which implies requiring a large silicon chip area [8].

To achieve real time application in a feasible implementation, the complexity of architecture should be reduced. This paper proposed a low-complexity decoder architecture which incorporates address counter and index table look-up to simplify combinational arithmetic. Besides, it takes the most significant bit in output section to eliminate exponent calculation. In addition, synchronous power-of-2 timing signal reduces iteration decision logic. Nevertheless, in-place algorithm utilise memory modules. Also, the use of ROM as an interleaver can solve latency problem due to interleaving process. 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. The size is the main drawback of the turbo code decoder for practical VLSI chip implementation.

Future work on this subject includes the design of a parallel architecture decoder with small chip area constraint. In this case, the architecture will be performed using reconfigurable FPGAs. The technology will progress from dynamic to the more demanding on-the-fly reconfiguration necessary for the next generation of intelligent and adaptive hardware.

6. REFERENCES

[1] C. Berrou, A. Glavieux and P. Thitimasjshima, "Near Shannon Limit Error-correcting Coding and Decoding: Turbo-codes (1)," *Proc. of the IEEE Int. Conf. on Communications*, Geneva, Switzerland, vol. 2, May 1993, pp. 1064-1070.

[2] Claude. E. Shannon, "A Mathematical Theory of Communication," *Bell Sys. Tech. J.*, vol. 27, 1948, pp. 379-423 and 623-656.

[3] Matthew C. Valenti, "Turbo Codes and Iterative Processing," *Proc. of the IEEE Int. Conf. on Communications*, Geneva, Switzerland, vol. 2, May 1993, pp. 944-968.

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

[5] R. G. Gallager, "Low Density Parity Check Codes," *IRE Tran. on Information Theory*, IT-8, 1962, pp.21-28.

[6] Guido Masera, Gianluca Piccinini, Massimo Ruo Roch and Maurizio Zamboni, "VLSI Architectures for Turbo codes," *IEEE Tran. on Very Large Scale Integration (VLSI) Systems*, vol. 7, no. 3, September 1999, pp. 369-379.

[7] Sangjin Hong, Joonhwan Yi and Wayne E. Stark, "VLSI Design and Implementation of Low-complexity Adaptive Turbo-code Encoder and Decoder for Wireless Mobile Communication Applications," *IEEE Workshop on Signal Processing Systems*, 1998, pp. 233-242.

[8] Sangjin Hong and Wayne E. Stark, "VLSI Circuit Complexity and Decoding Performance Analysis for Low-power RSC Turbo-code and Iterative Block Decoders Design," *Proc. of the IEEE Military Communications Conf.*, vol. 3, 1998, pp. 708-712.