Design and implementation of LT codec architecture with optimized degree distribution

S. M. Shamsul Alam^{a)} and GoangSeog Choi^{b)}

*Department of Information and Communication Engineering, Chosun University,
309, Pilmun-daero, Dong, GwangJu, 501–759, Korea*

a) alam_ece@yahoo.com

b) gschoigs@chosun.ac.kr

Abstract: In this paper, we present an architecture for ASIC realizations of the Luby Transform (LT) encoder and decoder. To determine the efficiency of the LT Codec architecture, the encoder and decoder are implemented with a core area of 9 mm^2 in TSMC 180-nm 1-poly 6-metal and Samsung 130-nm complementary metal–oxide–semiconductor (CMOS) technology. An empirically modified Robust Soliton degree distribution technique is applied for LT Codec implementation and its performance is analyzed in terms of chip area and cycle count. Instead of including a random generator in the register transfer level (RTL) design, we use different look-up tables (LUTs) for degree distribution, edge routing, addressing and inverse edge routing. Therefore, this architecture is efficient for hardware implementation and occupies less area inside the chip. The result shows that an area of 2.3 mm^2 is required for whole encoder and decoder implementation using TSMC library, of which 0.08 mm^2 is used for encoder implementation. Finally, a modified Robust Soliton degree distribution technique is presented and evaluated in terms of cycle count for different number of iterations using Tensilica tool. Result shows that it takes very less iterations which is more beneficial for hardware implementation of LT Codec.

Keywords: application specific integrated circuit (ASIC), Luby Transform (LT) encoder and decoder, degree distribution, check node unit (CNU), variable node unit (VNU), coder and decoder (Codec)

Classification: Electron devices, circuits, and systems

References

- [1] H. Jenkač and T. Mayer: Proc. IST Mobile Summit (2005).
- [2] J. Byers, M. Luby and M. Mitzenmacher: IEEE J. Sel. Areas Commun. **20** (2002) 1528.
- [3] C. Howland and A. Blanksby: IEEE Conf. Custom Integrated Circuits (2001) 293.

- [4] T. Zhang and K. K. Parhi: IEEE Workshop on Signal Processing Systems (2001) 25.
- [5] H. Wang: Master's thesis, Technical University of Delft, The Netherlands (2006).
- [6] K. Zhang, X. Huang and C. Shen: IEEE Workshop on Signal Processing Systems (2008) 210.
- [7] C. M. Chen, Y. Chen, T. C. Shen and J. K. Zao: IEEE Conf. Evolutionary Computation (CEC), (2010) 1.
- [8] T. Brandon, R. Hang, G. Block, V. C. Gaudet, B. Cockburn, S. Howard, C. Giasson, K. Boyle, P. Goud, S. S. Zeinoddin, A. Rapley, S. Bates, D. Elliot and C. Schlegel: Integration, the VLSI J. **41** [3] (2008) 385.
- [9] S. M. Shamsul Alam and GoangSeog Choi: ACM J. (Submitted paper).
- [10] Tensilica Product "ConnX D2 DSP Engine," (2012), http://www.tensilica.com/uploads/pdf/connx_d2.pdf.

1 Introduction

The binary erasure channel (BEC) is a real world channel environment which is a common communication channel model used in coding theory and information theory. Since the absence of feedback concept in forward error correction channel, advanced adaptation schemes or reliable transmission modes are infeasible in the BEC environment [1]. Therefore, research has been done to fulfill the BEC requirements. Luby *et al.* explained a channel code with potentially limitless redundancy (rateless) and used it to solve the reliable broadcast problem in BEC [2]. This coding scheme is known as the fountain code. Luby Transform (LT) code and Raptor code are two such fountain codes based on its degree distribution function. These codes have been extensively proposed to solve the transmission problem through wired internet and the resulting behaviors are investigated on erasure channels. Like the low density parity check (LDPC), the decoding part of the LT code includes an iterative belief propagation algorithm or Log-BP algorithm. So, the decoder architecture of the LT code has followed a similar architecture to that of the LDPC decoder. In [3], the LDPC decoder was implemented by using parity check matrix directly mapped into the hardware. In [4], the VLSI architecture of LDPC was studied and authors tried to reduce the gap between decoding throughput and hardware complexity.

In this paper, a novel hardware architecture of LT coder and decoder (Codec) is proposed and the application specific integrated circuit (ASIC) implementation of LT Codec is discussed. Moreover, the result of this newly modified chip architecture fabricated by using Taiwan Semiconductor Manufacturing Corporation (TSMC) 180 nm technology is presented. An efficient architecture is offered for LT Codec which includes implementation of degree distribution in encoder and LLR operation, check node, and variable node operation in decoder. Its decoder architecture is based on the principle of the sum product algorithm.

2 Hardware architecture of the LT coder

To implement the LT coder in the hardware environment, it is difficult to optimize a tradeoff between design complexity and coding performance. The problem of Ideal Soliton distribution is solved by the Robust Soliton distribution by increasing the number of ripple size similar to $\ln(k/\delta)\sqrt{k}$ [5]. Chin *et al.* [7] discussed how to handle the degree distribution by using the Pareto front formed by partial optimal solutions and provided the degree distribution to different LT code applications. In this paper, a custom LUT is used to design the degree distribution function of the LT code. Basically the degree distribution part of this proposed architecture is formed on the modification of Robust Soliton Distribution explained in equation (1). The Robust Soliton Distribution is given below:

$$\tau(d) = \begin{cases} \frac{s}{K} \frac{1}{d} & \text{for } d = 1, 2, \dots, (K/S) - 1 \\ \frac{s}{K} \log(S/\delta) & \text{for } d = K/S \\ f(a, b, c) = a(b + c) \end{cases} \quad (1)$$

where, $\tau(d)$ is the degree distribution, $S \equiv c \log_e(K/\delta)\sqrt{K}$ is the expected number of degree-one. This modified degree distribution takes less iteration for successful decoding.

In implementation result section, we mentioned the iteration result using this empirically modified degree distribution. Therefore, by using this distribution, the hardware architecture of the LT code is shown and the result of this architecture implemented by TSMC and Samsung library is discussed in this paper.

2.1 HW architecture of encoder

In an encoder, a long output encoded sequence can be produced from k input symbols $\{S_1, S_2, S_3, S_4, \dots, S_k\}$ as $c_i = S_{i,1} \oplus S_{i,2} \oplus S_{i,3} \oplus \dots \oplus S_{i,d}$ [6].

The value of degree d is taken randomly from a degree distribution function explained in equation (1). In order to take the random degree d , a random number generator is required but in this architecture we have used one table generated from equation (1) that removes the requirements of random number generator.

Figure 1 shows the hardware architecture of the LT encoder for 128 input bits and 256 output bits. Here two look-up tables (LUTs) are used to satisfy this distribution. If the degree distribution is 4, then pick 4 consecutive rows of address message column (4, 6, 3, 2) that point out the message value of the corresponding address of the message signal. Here, the address of the message signal is randomly distributed and the combined operation of the column for degree distribution and the address of the message satisfies the distribution mentioned in equation (1). These same LUTs are also used for decoding of the encoded signal. The message signals identified by one row of the degree distribution column are added and the result is stored in a temporary register. For example, in the degree distribution column, the

Fig. 1. Architecture of LT Encoder

degree value is 4 then the message signals of address (4, 6, 3, 2) positions are identified as (1, 0, 1, 0) respectively, and the result of this addition is stored in a temporary register as 2 and after applying the modulo 2 operation, the encoded signal for degree distribution 4 is 0. The 256 bit encoded signal is generated according to the same procedure as used for the 128 rows of degree distribution column and 128 bit message signal.

The process used to generate the encoded signal is given below:

1. Create the two lists D and A that represent the degree and address of the message table, respectively.
2. Take a variable x that indicates the first element of table D .
3. Find the value of degree number d and let $D(x) = d$ and let $A(x) = A(x - 1) + d$.
4. Take d numbers from 0 to $k - 1$ message column, where k is the length of the original message by using the address $A(x)$ and save the result of their addition into the temporary register.
5. Apply modulo 2 operation on the temporary register column.
6. Repeat steps 1 to 5 until the codeword is formed.

Here the LUTs D and A are generated from equation (1) using a computer program.

2.2 LT decoding architectures

An (n, k) LT code is typically decoded according to a bipartite graph known as a factor graph which corresponds to the parity check matrix H of the code [8]. This kind of factor graph contains two nodes: variable nodes and parity check nodes. During this decoding process, the messages are exchanged back and forth in a number of decoding iterations between the variable nodes and check nodes. Channel decoding in an LT decoder is based on the log

likelihood ratio (LLR) of a binary random variable $X \in \{\pm 1\}$ defined by the following equation:

$$LLR(X) \triangleq \log \left(\frac{\Pr\{X = +1\}}{\Pr\{X = -1\}} \right) \quad (2)$$

where $LLR(X)$ represents the LLR corresponding to bit X , and $P(X = 0, 1)$ represents the probability that bit X is equal to 0 or 1. The LT decoder operates based on the sum product algorithm by passing the message (LLR values) on tanner graph. Let $L(t_{i,j})$ denote an L value message passed from check node i to variable node j and $L(h_{i,j})$ denote an L value message passed from variable node i to check node j .

Then from [1], $L(t_{i,n})$ can be written as:

$$L(t_{i,j}) = 2 \tanh^{-1} \left(\tanh \frac{L(\hat{c}_i)}{2} \cdot \prod_{n \in N_i, n \neq j} \tanh \frac{L(h_{n,i})}{2} \right) \quad (3)$$

where $L(\hat{c}_i)$ denotes the received L value of the codeword from the channel. Similarly, the L value $L(h_{i,j})$ depends on the messages passed to variable node i . So $L(h_{i,j})$ can be obtained by [1]

$$L(h_{i,j}) = \sum_{e \in \varepsilon_i, e \neq j} L(t_{e,i}) \quad (4)$$

Similarly the L value about the decoding decision [1]

$$L(\hat{u}_i) = \sum_{e \in \varepsilon_i} L(t_{e,i}) \quad (5)$$

In order to execute these equations, the decoding is performed by transferring messages from the check node to the variable node and the variable to the check node units. For example, equations (3) and (4) are responsible for implementing the check node unit (CNU) and the variable node unit (VNU) and equation (5) is used as the decoding final stage. In decoding architecture, these equations are implemented in different stages and the working principle of this architecture is discussed in the next section.

I) CNU operation:

In the CNU module, LLR memory is used for check node operation while the message is passing through the check node. Like encoder, the same degree distribution table is used so that when the degree is one, the counter counts the position of unity degree and CNU memory stores the message of the count address value from LLR memory. Then, the counter counts further and when the degree is not equal to one, the message from LLR of that count address is multiplied with the message from VNU memory through the operations presented in figure 2 (a). The CNU memory therefore has messages for degree one and updated messages for a degree greater than one. Messages pass through these CNU nodes and updated messages are stored in the CNU memory. The operations of CNU are executed as below:

1. Search for a row in degree table where $d(i) = 1$.

Fig. 2. Hardware architecture of the LT Decoder: (a) CNU architecture, (b) VNU architecture, and (c) Final decoding stage

2. Take the message from LLR memory and store it in CNU memory, $L(i) = C(i)$.
3. Search for a row in degree table where $d(i) = x$, $x \neq 1$.
4. For each x , temporary register $T(j) = T(j) * V(j)$ and $C(j) = T(j) * L(j)$ where $j = 0, 1, \dots, x - 1$.

II) VNU operation:

As shown in figure 2 (b), each variable node contains 4 LUTs. Two new LUTs termed as edge information and index tables are included in VNU operation. These additional tables consist of nodes and edge information provided by the degree distribution function. The VNU function unit takes data from CNU memory and stores it in VNU memory after following the operation of node routing and inverse node routing explained in figure 2 (b). In VNU, the processing unit accumulates messages serially from the check node and stores them in the variable node memory. The operation of VNU can be written as below:

Search for a row in address table $A(i)$ such that $A(i) = K$ and the increment of K depends on index table $I(j)$. Here, j is the variable of the Index table and its increment depends on Degree table $D(l)$. Edge information table controls the value of l in a prescribed manner. So, for $A(i) = K$, VNU processing unit accumulates LLR message format from CNU memory and stores it in the VNU memory unit.

III) Final decoding stage:

After finishing the CNU and VNU operations, CNU memory contains all the nodes and edges of the processing information. Degree and address LUTs are used for generating addresses for the decoder memory. Then, data read from the decoder memory are taken as the decoded output. Figure 2 (c) shows the final decoding stage architecture. Its algorithm is given below:

1. For each element of degree $D(i)$, increase the index variable k until $k = D(i)$.
2. When $k > D(i)$, then $k = 0$.
3. For every value of i and k , take the value from the address table A .
4. Using this address value, store the information from CNU memory in the decoder memory.
5. Finally, the decoded output is generated from the decoder memory.

From the above discussion, the whole decoding process is explained through the LLR operation, CNU and VNU processing unit and final decoding stage. It is performed by passing messages from check nodes to variable nodes and vice versa. Therefore, this decoding is an iterative process and messages are decoded from the code value after certain iterations. This 144 quad flat package pin LT Codec chip is fabricated by applying TSMC 180 nm technology and its performance is detailed in the next section.

3 Implementation result

Based on the architecture explained above, this LT Codec architecture is implemented by using TSMC 180-nm and Samsung 130-nm standard cell library, and 1-poly and 6-metal complementary metal-oxide-semiconductor (CMOS) technology. Here, a single clock is used for synchronizing all operations of LT Codec. Instead of using a random number generator, LUTs are used to reduce the area inside the core and decrease the complexity in the

Table I. Implementation Result

Parameters		TSMC	Samsung
LT Codec (n,k)		(256,128)	(256,128)
Clock (MHz)		50	50
Technology		TSMC 180 nm	Samsng 130 nm
Metal layers		6	6
Combinational area (μm ²)	Area	779012.949320	111726.946341
No combinational area (μm ²)		1582591.378503	158435.787793
Total cell area (μm ²)		2361604.250000	270162.718750
Gate count		236K	162K
Cell count		1491	1449

encoding and decoding processes. Table I shows the resource usages of the blocks in LT Codec.

This LT Codec design has been synthesized by Synopsys design compiler version z-2007.03-SP1 and the generated area report is presented in table I. Three memories are used in the LT decoder (LLR, CNU and VNU) and these register memories occupy 1,854,898 (μm^2) and 214,309 (μm^2) of the total cell area for TSMC and Samsung library respectively. Therefore, the cell area is further reduced by applying a compile memory instead of a register memory.

In LT Codec, the encoder occupies an area of only 89253 (μm^2) for TSMC and 9794 (μm^2) for Samsung library, which is less than that of the decoding part of this chip. Therefore, Samsung library takes very less area compared to the TSMC library. Analysis of the VNU architecture reveals that it requires four LUTs and uses more register memory compared to CNU. Hence, one VNU occupies an area of 14123 (μm^2), which is almost double the area of CNU.

Table II shows the result for implementing LT Codec using Tensilica tool. This performance is evaluated in terms of cycle count, time count and number of iterations. Here, Tensilica tool had been used to measure the complexity of LT Codec implementation in iterative manner. It is mentioned earlier that the decoding complexities depend on the number of iterations required for recovering message from encoded signal. On the other hand the minimum number of iteration, which is required to get error free decoded signal, depends on the maximum value of degree in check node operation. Table II shows this iteration result of LT decoding process. This iterative result has been done by using Tensilica tool. Before discussing the different parameters of table II, we will discuss an ASIP oriented design process using Xtensa Xplorer (XX) integrated development environment (IDE) as the

Table II. Simulation for different number of iteration using Tensilica tool

# of Iterations	XRC_D2MR_MAC		DC_C_106 micro	
	Cycle Count	Time Count (s)	Cycle Count	Time Count (s)
1	5,204,861	4.43	6,983,593	4.97
2	19,182,518	15.60	26,034,519	18.31
3	35,128,884	27.71	48,036,337	33.79
4	52,840,982	42.21	72,696,444	50.98
5	71,951,305	57.18	99,135,487	69.74
6	92,160,200	74.41	127,678,043	90.72
7	115,082,566	92.73	159,731,114	113.43
8	164,837,807	128.70	194,314,345	137.03
10	189,915,708	151.39	264,271,320	185.73

design framework under Tensilica tool. Using the XX, it is possible to integrate software development, processor optimization and multiple-processor system-on chip (SoC) architecture into one common platform. From it, we can profile our input application code to identify the cycle consumed by the function used in input design. Then we can make necessary change to speed up that code. There are various configuration blocks in the Xtensa architecture.

These configurations are selected upon the nature of input application. Based on these properties of this architecture, we have taken two different configurations of architecture to simulate our input application. Then we apply some custom logic levels to processor for accelerating the processor performance. In this paper, two configurations known as XRC_D2MR_MAC and DC_C106 micro are used for simulation and show the performance in terms of cycle count and time. The architecture XRC_D2MR_MAC is a member of the family of communication configuration known as ConnX D2 DSP engine [10]. In addition LT Codec architecture and its hardware operations are matched with the resources of communication configurations like XRC_D2MR_MAC. For example the XRC_D2xx configuration includes dual 16-bit multiply-accumulate (MAC) units and 40-bit register file to the base RISC architecture of the Xtensa LX processor. This engine uses two-way SIMD (single instruction, multiple data) instructions to provide high performance on vectorizable C code. It implements an improved form of VLIW instructions and five-stage pipeline. For this reason this architecture is suitable for implementing LT Codec application. Similarly DC_C106 micro is a part of Diamond or General Purpose Controller. So this architecture can commonly be used for every input application. As it is mentioned earlier that the configuration is chosen on the basis of input design, so XRC_D2MR_MAC shows very good result compared to diamond controller processor.

For Xtensa Xplorer it takes only 9 iterations for successfully decoding the encoded signal [9]. However, it is possible to reduce the number of required iterations by modifying the degree distribution in the encoder. If we analysis the decoding part of input design, the whole complexity of decoding algorithm drops to the number of iterations of the message passing algorithm. Therefore, number of cycles are increasing exponentially with respect to the number of iterations. For example for single iteration it will take 5,204,861 cycles for implementing the LT Codec design. However for 10 iterations it takes more than 189 M cycles which is very high compared to the single iteration. So it is very important to trade off between several issues: degree distribution, architecture structure of processor configuration, architecture of LT encoder and decoder, and finally the status of BEC. This paper shows the response of LT Codec architecture considering those tradeoffs.

4 Conclusions

This article has described the implementation of architecture for an LT encoder and decoder. This LT Codec implementation was transformed from

register transfer level (RTL) code specification into the final chip fabrication that was synthesized using ASIC design flow (Synopsys frontend and backend tools) by applying Samsung and TSMC 180-nm 6 metal CMOS process. A new degree distribution function was applied in the encoding and decoding techniques of LT Codec and the degree distribution was realized by applying LUTs. Moreover, routing and reverse routing algorithms of CNU and VNU operations were realized using memories that stored the non-zero elements in check node and variable node memories. The superior performance of this LT Codec architecture in terms of chip area and number of iterations will help to design an efficient channel coding scheme in reliable wireless communication.

Acknowledgments

This research was supported by the Basic Science Research Program through the National Research Foundation of Korea funded by the Ministry of Education, Science and Technology (2010-0022661). GoangSeog Choi is a corresponding author.