August 2014

Master's Degree Thesis

Novel Hardware Architecture of the LT CODEC with Non-Belief Propagation Based Decoding Algorithm

Graduate School of Chosun University

Department of Information and Communication Engineering

Md. Tariq Hasan

Novel Hardware Architecture of the LT CODEC with Non-Belief Propagation Based Decoding Algorithm

비 BP 복호 알고리즘을 적용한 새로운 LT 코덱 하드웨어 구조

August 25, 2014

Graduate School of Chosun University

Department of Information and Communication Engineering

Md. Tariq Hasan

Novel Hardware Architecture of the LT CODEC with Non-Belief Propagation Based Decoding Algorithm

Advisor: Prof. GoangSeog Choi, PhD

A thesis submitted in partial fulfillment of the requirements for a Master's degree

April, 2014

Graduate School of Chosun University

Department of Information and Communication Engineering

Md. Tariq Hasan

Graduate School of Chosun University
GwangJu, Republic of Korea (South)

CERTIFICATE OF APPROVAL

MASTER'S THESIS

May, 2014

This is to certify that the master's thesis of

Md. Tariq Hasan

has been approved by the examining committee for the thesis requirement for the Master's degree in Engineering

Committee Chairperson ..... 변재영
Prof. Jae-Young Pyun

Committee Member ..... 김영식
Prof. Young-Sik Kim

Committee Member ..... 최광서
Prof. GoangSeog Choi

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND SYMBOLS

ARQ	Automatic Repeat request
ASIC	Application Specific Integrated Circuit
BEC	Binary Erasure Channel
BP	Belief Propagation
BP	Belief Propagation
BSC	Binary Symmetric Channel
C2S	Check-node to *s*
CAD	Computer Aided Design
CN	Check Node- LT encoded output
CNG	Check Node Generator
CNPU	Check Node Processing Unit
CODEC	COder-DECoder
CPU	Central Processing Unit
CUU	Check- Node Update
DDU	Degree Distribution Unit
DGU	Degree Generation Unit
DSP	Digital Signal Processing
EDA	Electronic Design Automation
FEC	Forward Error Correction
FPGA	Field-Programmable Gate Array
G2G	*G*2 Generator
GMU	Generator Matrix-generation Unit
GSI	Giant (or Giga) Scale Integration
GUU	Generator-matrix Update Unit

HDL	Hardware Description Language
IC	Integrated Circuit
IID	Independent and Identically Distributed
IP	Intellectual Properties
ISD	Ideal Soliton Distribution
LCG	Linear Congruential Generator
LFSR	Linear Feedback Shift Register
LT	Luby Transform
LUT	Lookup Table
MG	Matrix Generator
MUX	Multiplexer
NAK	negative acknowledgement
NoC	Network-on-Chip
PAK	Positive Acknowledgement
POS	Product of Sum
PU	Permutation Unit
PU	Permutation Unit
RNG	Random Number Generator
RRVN	Recently Recovered Variable Node
RSD	Robust Soliton Distribution
RTL	Register Transfer Logic
SEIU	Single Edge Identification Unit
SoC	System-on-Chip
SOP	Sum of Product
SPA	Sum Product Algorithm
STA	Static Timing Analysis
ULSI	Ultra Large Scale Integration

VHDL	Very-High-Speed-Integrated-Circuit Hardware Description Language
VLSI	Very Large Scale Integration
VN	Variable Node- Input message of an LT encoder

?	Unknown symbol for lost or erased one
μ	RSD
a	Multiplier
b	Increment
c	Check node or a constant
col_pos	Column position
csum	Cumulative sum
d	Degree
deg	Memory to store degrees according to probabilities
e	Erasure probability
G	Generator Matrix
g_in	Input generator matrix
K	Number of input message bits
m	modulus
N	Total number of check nodes generated
prob	Memory to store probabilities of the respective degrees
ran	Random number
rand	Random number produced by the LFSR
ro_pos	Row position
S	Number of degree-one check node
s	Variable Nodes
se_flag	Register to store single edge status

tp	Temporary register to hold probability
tsum	Register to hold temporary value of a column sum
X_0	*Seed*, initial value of the random number generator
X_t	Channel input, at time *t*
Y_t	Channel output, at time *t*
Z	Sum of ρ (d) and $\tau(d)$
δ	Probability of decoding failure
ρ	Degree distribution usually ISD
τ	Degree distribution usually RSD

ABSTRACT

Novel Hardware Architecture of the LT CODEC with Non-Belief Propagation Based Decoding Algorithm

Md. Tariq Hasan
Advisor: Prof. GoangSeog Choi, Ph.D.
Department of Information and
Communication Engineering
Graduate School of Chosun University

Present technology- EDA tools, technology independent hardware description languages, have made possible to design almost a complete system on a chip, and design time is reduced to engineer-month. As a result, the computer, processor, internet, wireless communication, etc. are getting new shape and dimension swiftly. In a typical automatic repeat request based communication, some transmitted symbols may be lost or erased due to the adverse effects- excessive delay or buffer over flow, of the binary erasure channel, where data is either received or lost, even though higher order of error correcting code is applied. There are couples of solutions of the problem, such as RS code and Tornado code. However, these codes are not rateless and true fountain codes to combat against the adverse effects of the binary erasure channel. In 1998 M. Luby proposed the first rateless fountain code known as LT codes. In this thesis a novel hardware architecture of the LT codec is presented where non- BP based decoding algorithm is applied. A new and fully functional LT codec architecture is designed with an

efficient degree distribution unit using Verilog HDL in ModelSim. To perform permutation operation, different initial valued or time shifted counters have been used to get pretty well permutations and an effect of randomness. Any kind of degree distribution can be implemented with the degree generator architecture. Usually, LT codes performs well as the number of input bits goes higher, and the encoder will generate ideally endless streams of encoded bits like fountain. However, for hardware implementation there should be a limitation. Here the codec will take 128 bits as input and produce 256 encoded output bits. The encoder, along with the random number generator unit, distribution generator unit and generator matrix unit performed quite satisfactorily. The decoder with its internal functional units – check node and generator matrix processing unit, single degree check nodes identification unit, value assignment unit, check nodes update and generator matrix update units performed quite satisfactorily as it successfully decoded the encoder generated check nodes to original message bits. The result-waveforms and distribution figures shows expected performances as the implemented distribution and the original distribution are pretty same. Matrices are stored in memories and address indicates the respective column. Encoding and decoding processes do not have complex mathematical functions such as tanh(), ln(), and inverse matrix, etc of soft BP decoding and inverse matrix based hard decoding processes respectively. Instead of using LUTs and predefined arrays, here a simple random number generator and a degree generator have been used to get a distribution. From the simulation it is found that the proposed LT codec takes 257.5 cycle counts and 2.575 μs for encoding and decoding instead of 5,204,861 minimum cycle counts and 4.43s of the design mentioned in the previous works where iterative soft BP decoding was used in ASIC and ASIP implementation of the LT codec.

한 글 요 약

비 BP 복호 알고리즘을 적용한 새로운 LT 코덱 하드웨어 구조

하산 타릭
지도 교수:최광석
정보 통신공학과
대학원, 조선대학교

현재 독립적인 하드웨어 기술 언어인 EDA 툴 기술은 칩 에 거의 완전한 시스템 을 설계하는 것이 가능 했고, 디자인 설계 시간을 줄였다. 이 결과 , 기타 컴퓨터 프로세서 , 인터넷, 무선 통신 은 신속히 새로운 형상 및 치수 를 받고 있다. 일반적인 자동 반복 요청 기반의 통신 에서 전송 된 일부 기호는 , 데이터를 수신 하거나 분실 하거나 이진 소거 채널 의 손실 또는 삭제 로 인해 부작용 - 과도한 지연 이나 흐름을 버퍼링 할 수있다 그럼에도 불구하고 오류의 고차 정정 코드 가 적용된다 . 이러한 RS 코드와 토네이도 코드는 문제의 솔루션 커플 이다. 그러나 이러한 코드는 이진 소거 채널의 역 효과에 대해 대처하기 위해 레이트리스 진정한 분수 코드이지 않다. 1998 년 M. 지도 Luby 는 LT 코드로 알려진 첫 번째 레이트리스 분수 코드를 제안했다. 이 논문 에서 LT 코덱 의 새로운 하드웨어 아키텍처가 비 BP 기반 디코딩 알고리즘 이 적용되는 곳에 제공된다. Verilog HDL과 ModelSim을 사용하고 특별히 설계된 정도(차수) 생성기를 이용하여 새롭고 완전하게 동작하는 LT 코덱 구조를 설계했다. 높은 순열과 무작위 효과를 얻기 위하여 다른 초기 값들 혹은 시-변위 카운터들이 사용되었다. 어떠한 정도(차수) 분포도 이

정도(차수) 생성기 구조로 구현되어 질 수 있다. 일반적으로 입력 비트들의 수가 많을수록 LT 부호들은 잘 동작하고 부호기는 분수와 같이 끝없는 부호화된 비트 스트림을 생성할 것이다. 그러나 하드웨어 구현에서는 한계가 있다. 여기서는 코덱은 입력으로 128비트를 가지고 256비트의 출력 비트들을 생성할 것이다. 랜덤 수 생성기, 분포 생성기 및 생성 매트릭스기와 함께 부호기는 아주 만족스럽게 동작했다. 내부기능 유닛들 - 일차 체크 노드 인식 유닛, 값 할당 유닛, 체크 노드 개선 및 생성 매트릭스 개선 유닛들을 가진 복호기가 아주 만족스럽게 동작했다. 파형들과 분포 그림들은 구현된 분포와 원 분포들이 매우 같다는 예상 성능을 보여준다.

매트릭스들은 메모리에 저장되고 어드레스는 해당 열을 나타낸다. 부호 및 복호 과정들은 BP 복호 과정에서의 tanh(), ln() 등과 같은 복잡한 수학적인 항들을 가지고 있지 않다. 분포들을 얻기 위하여 LUT들과 미리 정의된 배열들을 이용하는 대신에 간단한 난수 발생기와 정도(차수) 생성기들을 사용하였다. 아주 좋은 무작위 효과들을 얻기 위한 매트릭스 생성하기 위하여 다른 초기 값들의 클록들이 사용된다. 부호와 복호를 위해 517 사이클 수와 2.585μS가 걸린다. 반면에 LT 코덱의 ASIC과 ASIP 구현에서 사용된 반복적인 연 BP 복호에서 부호와 복호에서 5,204,861 최소 사이클 수와 4.43s가 걸렸다.

I. INTRODUCTION

In a traditional Automatic Repeat Request (ARQ) based data communication, extra feedback path is required and yet sometimes receiver fails to receive the transmitted packets and send a message to the upper layer that the packet is lost. Few solutions have been proposed to solve the problem, such as Tornado codes. However, those are not true rateless fountain codes. Then in the year 1998, Michael Luby came out to solve the problem. The codes that he proposed named after him as Luby Transform codes and abbreviated as LT codes. These codes have some fascinating characteristics. Because of its importance, it is being adopted in many standards.

On the other hand, invention of transistor, microelectronics, then integrated circuits, Electronic Design Automation (EDA) tools, multi-core chip, etc.- have changed the concept of VLSI design and implementation, and hence the world and the way people used to live. Whole system is being integrated on a chip as System-on-Chip (SoC); people are using embedded system in their daily life. Here a novel architecture of the LT codec (COder-DECder) has been proposed with an excellent degree distribution unit and a random permutation unit.

In this chapter, background problem, motivation, research goal and organization of the thesis are presented.

A. Background Problem

In the traditional ARQ based data communication, feedback path is required to exchange positive acknowledgement (PAK) and negative acknowledgement (NAK). However, according Shannon, this extra feedback path is wasteful [1].

Because, if a number of receiver request to resend of the packets, sender may get busy and this may increase its work load. Besides, these packets might be lost or erased due to the adverse effect of the binary erasure channel (BEC). To solve this problem, M. Luby proposed first true rateless fountain codes, called LT code. The efficiency of the code increases as the code size increases [2]. Now a hardware architecture of the LT codec is required for practical implementation of the codes.

B. Motivation

The binary erasure channel, introduced by Elias in 1954 [3], is the real world and simplest channel model where data arrive correctly or lost due to buffer overflow or excessive delay experienced in the transmission [4]. LT codes are the first true rateless codes introduced by Michael Luby in 1998 to solve the erasure problem [2] [5]. Instead of sending message bits along with the redundant bits like other block codes, the information of the transmitting message are distributed among a number of encoded bits in LT codes. Therefore, if some of the transmitted bits are lost or erased in BEC, source information can be recovered from the remaining intact bits. Because of its efficiency and universal characteristics, it might be used in many applications [6], [7], [8]. It will have a good impact in the field of digital system design and communication if an efficient hardware architecture of the LT codec is found.

C. Research Goal

The main goal of this research is twofold. First, we would like to study the basics of LT codes as Luby has proposed in his paper and hence find the design parameters to design a hardware for efficient implementation of a degree-distribution and a generator matrix. These are important parts of the performance of the LT codec. Second, we would like to design a hardware architecture of LT

codec as Luby has presented the codes using non-Belief Propagation (BP) based hard decision decoding algorithm. It is quite challenging to have an efficient random number generator and degree generator of a specific degree distribution. With all these modules an LT codec will be designed which will perform as LT encoder and decoder as Luby has described the fountain code.

D. Organization of the Thesis

In this thesis, background study of the LT encoding and decoding, latest research on it, hardware design methodology, proposed architecture, and simulation outcomes are organized in the following way.

Chapter I: Introduction – it includes background problem, motivation, research goal and organization of the thesis.

Chapter II: Digital design methodologies – it deals with the choices of implementation, electronic design automation, important features of HDLs, design flows, design methodologies, and HDL design levels.

Chapter III: Luby's proposed LT Codes – it describes the background theories, degree distributions, encoding process, decoding process, and applications of the LT codes.

Chapter IV: Related works on hardware architecture of the LT codec – it covers literature review or previous works done on the hardware architecture of the LT codec.

Chapter V: Proposed hardware architecture of the LT encoder – it discusses proposed overall architecture of the LT codec, design hierarchy, random number generator unit, degree generator unit, proposed hardware architecture of the generator matrix, and hardware architecture of the LT encoder.

Chapter VI: Proposed hardware architecture of the LT decoder – it focuses on the LT decoder, check node processing unit, generator matrix construction unit, identification of single edge check nodes and C2S units, check nodes update unit, generator matrix update unit, and input-output waveforms of the codec.

Chapter VII: Simulation results – it explains the performance of the proposed degree distribution unit, and the architecture of the LT codec.

Chapter VIII: Conclusion – this chapter summarizes the research works and recommends the possible future works.

II. DIGITAL DESIGN METHODOLOGIES

In the previous chapter, thesis objectives, motivation, background problem, thesis organization are presented, and here, digital design methodologies are explored as the beginning of the main subject. In the past 40 years, electronics as well as digital design have grown very sharply from SSI to VLSI, ULSI and finally GSI. In the earlier stage, there were very small numbers of transistors or logic gates inside an IC (Integrated Circuit). However, Intel is thinking about 10nm transistor technology by 2015 [9]. This means the number of transistors is increasing very rapidly in the chip and the number would be more than $2{\cdot}6\times10^{09}$ in near future to satisfy the Moore's law. To handle this huge number of transistor, electronic design automation (EDA) tools have been used since 1983 [10]. Hardware description languages (HDLs) have become popular to design a digital system; however, the use was very limited – only for logic verification, and design engineers had to translate the HDL-based design into a schematic circuit with interconnections between gates manually [10], [11]. Then the birth of synthesis changed this concept and design engineer could design a system at a register transfer level (RTL) by the use of an HDL. Here they need to specify how the data flows between registers and how the design processes the data. The details of gates and their interconnections to implement the system were automatically extracted by logic synthesis tools from the RTL description. The design engineers no longer had to manually place gates to build digital circuits. They could describe complex circuits at an abstract level in terms of functionality and data flow by designing those circuits in HDLs. Logic synthesis tools implements the specified functionality in terms of gates and gate interconnections.

A. Choices of Implementation

A circuit or a system can be implemented in different ways. Figure 1 shows the typical ways of design, and a brief description of these are mentioned below.

In the *full custom* or simply *custom* design, each transistor and interconnection are designed and laid out manually. This is also applicable for the standard cell design initially to prepare the standard cell library. This approach maximizes the performances- minimum chip area for high density of transistor, very high clock speed, etc. Usually this is followed when the volume of production will be very high, however, it is very time consuming and labor-intensive job [12].

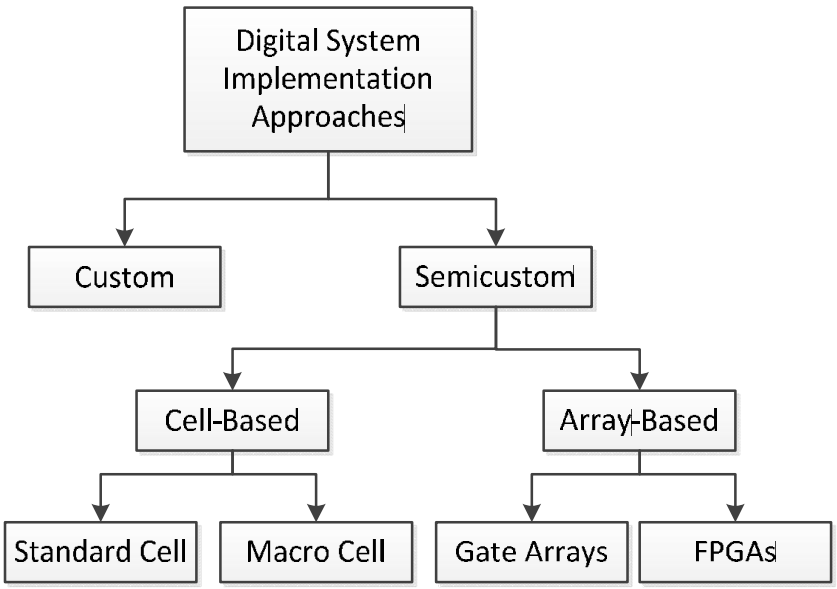

Figure 1. Digital system implementation approaches.

In *semicustom* approach, pre-designed standard cell libraries or available off the shelf hardware and HDL codes are used. It can be cell based or array based.

In the *standard cell based* design, pre-designed layouts from a vendor are used. Gate level simulation is performed of an RTL and usually all layers are customized for medium-high density and performance, and it takes reasonable time for design [13].

In the *Macrocell-based* design, predefined macro blocks (IPs–intellectual properties) of analog and digital units, such as microprocessor, RAM, etc., are collected from a vendor. Gate level or behavioral simulation is performed to produce high density and high performance chip in a short time.

Gate array (SOP or POS) uses fixed sizes of array for the predefined transistors and these are connected via metal. Gate level simulation is performed for medium density, and medium performance chip and it takes short time for design [13].

Field-Programmable Gate Array (FPGA) uses programmable logic blocks and through program connections can be made between blocks and units; therefore, no layers are customized. Devices are expensive but the design tools are cheap. It takes short time for the design and implementation; however, it is not a real ASIC (Application Specific Integrated Circuit).

B. Electronic Design Automation

Electronic Design Automation (EDA) is the collection of methodologies, algorithms and tools, which assist and automate the design, synthesis, floor planning, place and route, verification, design rule check, mapping, and testing of electronic systems [14]. EDA has completely changed the way that electronic engineers design and manufacture integrated circuits. Before the invention and practice of EDA tools, usually, chips were designed and laid out manually. After the system design according to the system specifications, mask information, translation or mapping from transistors and interconnections to graphics was

managed by hand or manually. Developers started to automate the design with the drafting by the mid 1970s, for automatic placement and routing.

Afterwards in 1980s, the text book 'Introduction to VLSI Systems' by Craver Mead and Lynn Conway, made a dramatic change in VLSI design and showed a path which led to design a chip with the help of programming languages. This made design, simulation and verification possible before the actual fabrication of the chip. Initially EDA tools were produced academically. However, in 1981 EDA tools started its journey as commercial products. At the very beginning, Daisy Systems, Mentor Graphics, and Valid Logic Systems were founded around this time, and collectively known as DMV. In 1981, the U.S. Department of Defense began financial supporting of VHDL (Very-High-Speed-Integrated-Circuit Hardware Description Language) as a hardware description language and in 1986, Verilog was first introduced as a hardware description language by Gateway Design Automation [15]. In the 1990s, large second-order partial differential equation of heat was solved for energy optimization, Fast Fourier and discrete cosine transform were applied to solve the multi-layer Green's function for full-chip thermal analysis; for higher reliability, quality assurance, and removing lithographic limitations, and anomalous operation conditions, technology-centric EDA tools were evolved; for accurate system design, model checking, SAT solver, static analysis etc. were applied. All these evolutions led to the rise of System-on-Chip (SoC), where many components such as microprocessors, custom IPs, and even analog circuits are integrated in a single die [16]. The driving force of Moore's law causes the growth of transistors in a single chip and the operating frequency indirectly. However, due to power consumption and thermal density constraints, operating frequency cannot be increased indefinitely according to the prediction of Moore's law. Then the design engineers introduced multi-core architecture in a chip to accommodate CPUs, DSPs, IPs, memories, analog circuits,

etc. to obtain higher performance without a proportional increase in operating frequency. Connectivity and data transfer between the cores can be done in a similar way as it is followed in the computer networks, and this lead to a new era of Network-on-Chip (NoC) in the early 2000 [16]. By the new millennium, the VLSI world made a quantum leap by using the device size in the order of nano-meter, which led to a new-generation EDA design tools to manage nano-scale issues such as leakage currents, to develop multi-scale modeling for the beyond-Moore's-law technologies such as carbon nano-tubes, tunneling FETs, quantum dots, single electron transistors, and molecular devices. Design abstraction has transformed from polygon drawing, to schematic entry, then to RTL specification using hardware description languages (HDLs), and most recently to behavior specifications—for example, in C, C++, SystemC, and Matlab code [14].

C. Important Features of HDLs

Now design engineers can design complex digital system with hardware description language. It has following features to design, compile and simulate any digital system [10] [17].

- Design engineers do not need to think about the specific fabrication technology. It is logic synthesis tools' job to automatically convert the design to any fabrication technology.

- For new technology, design engineers do not need to redesign the digital system. The synthesis tools will create a new gate-level *netlist* for the new technology by optimizing the circuit in area and timing for the new technology.

- Designer can optimize and modify the RTL description until it fulfills the expected functionality, and system specifications. In this way bugs can be eliminated early.

- HDL is text based and portable. It is more portable than a schematic design and does not need any drawing tools.

- It is more like computer programming language.

- A system can be designed in a behavioral or structural manner.

- HDL permits to synthesize the intended digital system very fast as compared to the schematic design.

- Easy to test the design with test bench as the test bench is also text based, portable and repeatable. This functional verification can be done before the fabrication in the design cycle.

The sophisticated EDA tools- HDL tools, synthesis tools, and others, have made this field easily manageable, and now-a-days system designers cannot but think of it.

D. Design Flow

A general design flow of a digital system design of any reasonable complexity is shown in Figure 2 [10], [11] [17]. In any design, specifications are defined first. These specifications describe abstractly the functionality, interface, and overall architecture of the digital circuit to be designed. However, at this stage, the design engineers do not need to think about how they will implement this circuit. A behavioral description is then created to analyze the design in terms of functionality, performance, compliance to standards, and other high-level issues.

Behavioral descriptions are written with HDL [10]. The behavioral description is converted to an RTL description in an HDL manually.

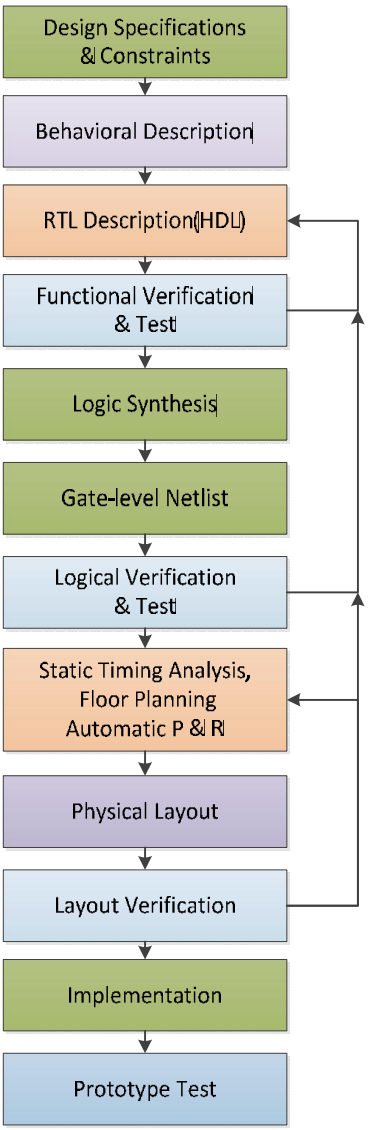

Figure 2. Design flow of a typical digital system design.

Here the design engineer has to describe the data flow that will implement the desired digital circuit. In the design cycle, the designers can test the functionality of

the design using test bench and remove design flaws, bugs, modify the description according to the specifications and functionality. Then logic synthesis tools automatically convert the high-level description, like RTL description, to a gate-level *netlist*, which is a description of the system in terms of gates and connections between them. The RTL can be modified according to requirement if there is any error or to check the functionality of the system. The stages from the design specification to the synthesis are known as *front-end.* On the other hand, the stages from *place and route* to GDSII (Graphic Database System)[1] [18] file generation are known as *back-end.* After static timing analysis (STA), the gate-level *netlist* can be input to an automatic *place and route* tool, which creates a layout [10]. The layout is verified and then fabricated on chip. Thus, most digital design activity is concentrated on manually optimizing the RTL description of the circuit. After the completion of RTL description, EDA tools are ready to assist the design engineers for the next processes. Designing at RTL level has reduced design cycle times from engineer-years to a few engineer-months. Behavioral synthesis tools have begun to emerge recently. Here these tools can create RTL descriptions from a behavioral or algorithmic description of the circuit directly. As these tools get more developed, digital system design will become similar to high-level computer programming in near future. Design engineers will simply implement the algorithm in an HDL at a very abstract level. CAD or EDA tools will help the design engineers to convert the behavioral description to a final chip. Usually a large number of skilled people are involved in the whole process of chip design.

[1] GDSII, is a database file format introduced by Calma. Now it is owned by Cadence Design Systems [18].

E. Design Methodologies

It is good to practice hierarchical design methodologies for efficient digital system design and organization. Usually there two types of digital design methodologies. These are

- Top-down hierarchy and
- Bottom-up hierarchy.

1. Top-Down Hierarchy

In a top-down hierarchy, the top-level block is defined and the sub-blocks necessary to build the top-level block are identified. The sub-blocks are further divided until easy and manageable leaf cells are found, which are usually the cells that cannot further be divided. Figure 3 shows the top-down hierarchy [19].

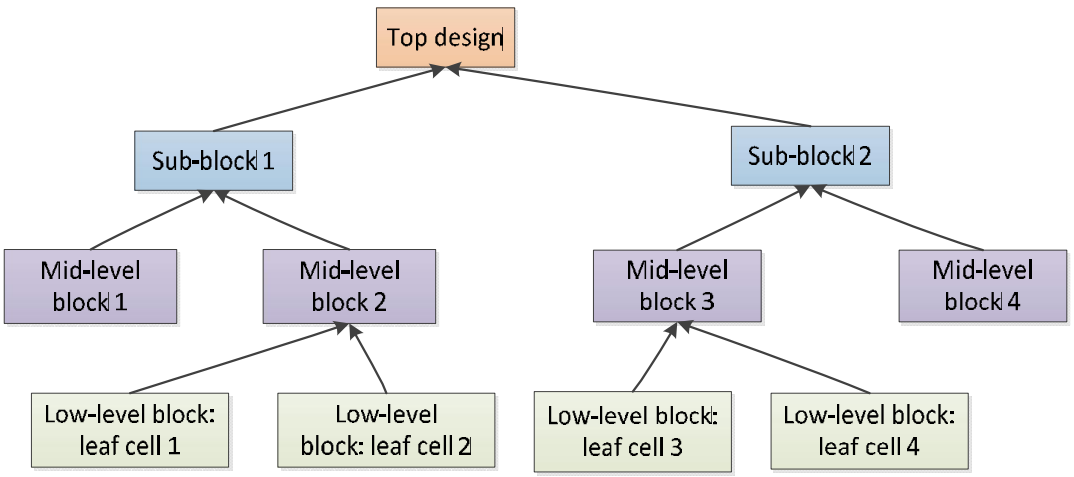

Figure 3. Top-down hierarchy of a significant complexity digital system.

2. Bottom-Up Hierarchy

In a bottom-up design methodology, first the building blocks that are available to design engineer are identified. The bigger cells are designed using these bottom

level building blocks. These cells are then used for higher-level blocks until the top-level block of the system is designed. Figure 4 shows the bottom-up hierarchy [19].

Usually a hybrid of top-down and bottom-up hierarchies are followed in practice and it depends totally on front-end design engineers.

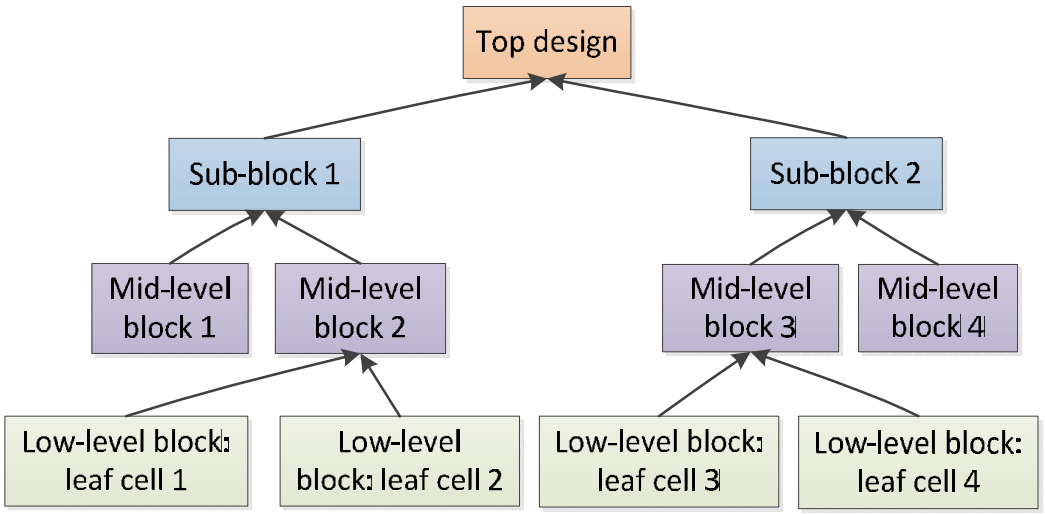

Figure 4. Bottom-up hierarchy.

F. HDL Design Level

HDL language can be both behavioral and structural language. For example: Verilog HDL. Here module is the basic building block. Internals of each module can be defined at four levels of abstraction, depending on the requirements of the design. The module behaves identically with the external environment irrespective of the level of abstraction at which the module is described. The internals of the module are hidden from the environment [10]. Thus, the level of abstraction to describe a module can be changed without any change in the environment. The levels are defined following.

Behavioral or Algorithmic Level

This is the highest level of abstraction provided by Verilog HDL. A module can be implemented in terms of the desired design algorithm without the concern of the hardware implementation details. Designing at this level is very similar to C programming.

Dataflow Level

At this level, the module is designed by specifying the data flow. The design engineer is aware of how data flows between hardware registers and how the data is processed in the design.

Gate Level

The module is implemented in terms of logic gates and interconnections between these gates. Design at this level is similar to describing a design in terms of a gate-level logic diagram.

Switch Level

This is the lowest level of abstraction provided by Verilog. A module can be implemented in terms of switches, storage nodes, and the interconnections between them. Design at this level requires knowledge of switch-level implementation details [10].

Verilog allows the designer to mix and match all four levels of abstractions in a design. The term register transfer level (RTL) is frequently used for a Verilog description that uses a combination of behavioral and dataflow constructs and is acceptable to logic synthesis tools. Usually, the higher the level of abstraction, the more flexible and technology independent the design is. As one goes lower toward switch-level design, the design becomes technology dependent and inflexible, and a small change in codes can cause a significant change in the design.

In this chapter, choices of implementation, design flow, digital design hierarchy, HDL design levels are discussed with necessary figures. One of the important characteristics of HDL is that it is independent of fabrication technology. Design engineers can concentrate only on design or functionality. All EDA tools, design flexibility, level of abstraction, and eventually the automation have decreased the design and fabrication time from engineer-years to engineer-months. This is really important from the commercial point of view. On the other hand, design engineers are adopting multi-core in a single chip to increase the performance of the chip instead of increasing the proportional operating frequency, and this has opened a new field of study, which is known as NoC and routing algorithms.

III. LUBY'S PROPOSED LT CODES

In the previous chapters, thesis objectives, motivation, background problem, VLSI implementation choices, electronic design automation, important features of HDLs, system design flow, design methodologies, and design levels are presented. Here binary erasure channel (BEC), fountain codes and Luby's proposed LT codes are discussed. Shannon showed that the feedback path for traditional ARQ-based packet data communication is wasteful. Besides, a receiver sometimes sends the upper layer that the packet is not received or lost although high level error correcting and detecting code is applied. Michael Luby came out with a solution that can solve this type of problem and remove the feedback channel to send PAK or NAK.

A. Erasure Channel

A channel can be modeled to find the adverse effects of it and the solution of the effects. The binary erasure channel, introduced by Elias in 1954, is the real world

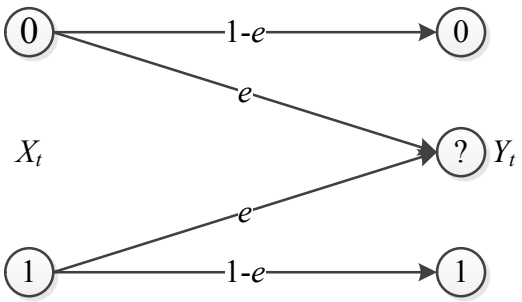

Figure 5. Binary erasure channel.

and simplest channel model where data arrive correctly or lost due to buffer overflows or excessive delays experienced in transmission [6]. In this channel, data

is lost or erased but is never corrupted as shown the model in Figure 5 for the 1-bit BEC (e). Here time is discrete and indexed by t. The transmitter and receiver are synchronized, and present state is t. Channel input, at time t is denoted by X_t and for binary channel $X_t \in \{0,1\}$ and corresponding output can be found as $Y_t \in \{0, 1, ?\}$, where '?' indicates an erasure. Each transmitted bits is either erased with probability e, or received correctly: $Y_t \in \{ X_t, ?\}$ and $p\{Y_t = ?\} = e$ [6].

In a traditional data communication, correct and faithful communication is achieved by exchanging control signals between a sender and a receiver. In brief it follows following scenario [1].

- Messages are converted into small manageable packets.
- Error detecting (and correcting) codes are applied to encode each packet.
- Encoded packets are transmitted through a binary symmetric channel (BSC).
- If the decoded message has any error or encoded bits are lost, receiver sends negative acknowledgement and tell the sender resend the packet.

This protocol is advantageous that it will work regardless of the erasure probability. Faithful and correct message reception may fail even though higher level of error-correcting codes is applied. However, according to Shannon, there is no need for the feedback channel and actually it is wasteful. The capacity of the forward channel is $(1-e)\cdot L$ bits, whether there is any feedback or not for L number of bits per packet. Reliable communication should be possible at this rate, with the help of an appropriate forward error correcting code [1]. The wastefulness of the retransmission protocols can be understood by the following example and Figure 6. Consider a broadcast scenario where one broadcast unit sends to large number receivers. Each receiver receives a random fraction $(1-e)$ of the packets and rest of the fraction, e is erased or lost. The sender has to retransmit every packet as every

receiver might have lost a fraction of a packet. Those retransmissions will be redundant as every receiver will have already received most of the retransmitted packets.

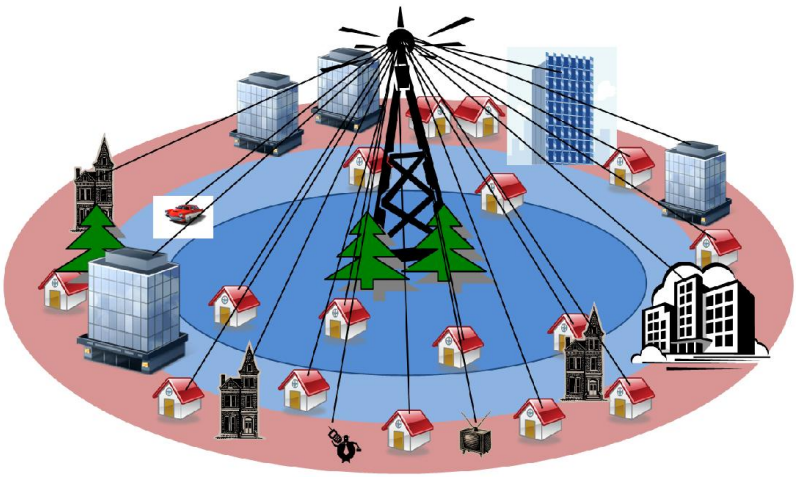

Figure 6. A broadcast unit is sending packets to a large number of receivers.

The scenario will be the worst if the number of receiver becomes more. Thus, it is required to make erasure-correcting codes that require no feedback or almost no feedback path.

B. Fountain Codes

In the fountain codes, the encoder is like a fountain, as shown in Figure 7, and generates randomly an arbitrary number of encoded symbols or droplets from a given set of message symbols and sends to the receivers. The receivers have to collect these symbolic droplets until its cup fills up and recover the whole message. For example, the encoder generates K number of droplets or symbols and each drop contains n number of bits. That is total number of bit is $K \cdot n$. Now anyone who wishes to receive the encoded file holds a cup under the metaphorical fountain and

collects drops until the number of drops in the cup is little larger than K. Then they can recover the original whole file or message.

Figure 7. Fountain where sufficient amount of droplets are collected according to one's requirements.

1. Luby Transform Codes

Michael Luby, in 1998, invented the codes known as LT (Luby Transform) codes as the solution for erasure problem of the BEC. In his proposal, the information of the transmitting message is usually distributed among a number of encoded output bits. If some of the bits are erased or lost with erasure probability e due to the negative impact of the BEC, still message can be recovered from the $(1-e)\cdot M$, rest of the intact received encoded bits, where M is the total number of encoded bits. On the other hand, receiver does not need any feedback path to send acknowledgement [1].

2. Characteristics of the LT Codes

LT codes have some important and fascinating characteristics. Some of the important characteristics and attributes of the LT codes are mentioned following [2], [4], [5].

- It is the first rateless erasure or fountain codes. It is rateless in a sense that encoder generates endless stream of encoded outputs like natural fountain and the receiver has to collect as many as it needs to decode the whole message. Practically a transmitter will transmit the encoded bits until each receiver has enough unerased encoded bits to recover the associated message bits—i.e., until each receiver has filled its cup from the digital fountain. Thus, fountain codes provide reliable dissemination of information without the complexity and congestion incurred by the conventional forward error correction (FEC) codes and ARQ techniques based communication [20].

- The encoded bits can be generated on the fly, as many or as few as required by the receivers. Again, the decoder can recover the exact replica of the data from any set of the encoder generated bits [2].

- No matter what the probability of erasure or loss model is of the BEC, encoded output bits can be generated as many as required and transmitted over the channel until a receiver gets sufficient number of encoded bits to recover the whole message.

- Furthermore, encoded bits can be received in any order and it is not required to follow the order that it was sent.

- Message can be recovered from the minimum number of encoded bits. This means that LT codes are optimal with respect to any erasure channel.

- LT codes are universal. It means that it is very efficient as the data length grows.

3. LT Encoding

An LT encoder will receive K number of message bits ($s_0, s_1, s_2, s_3, \ldots, s_{k-1}$), which are also known as variable nodes u, and d_n degree from $\rho(d)$ degree distribution.

$$c_n = \sum_{k=1}^{K} s_k G_{kn} \tag{1}$$

d_n message bits are chosen uniformly at random from u to get output encoded bits, known as check nodes, c by modulo-2 addition and the corresponding column of the generator matrix is generated. Equation (1) can be used to describe the encoding process mathematically where n is the position of the check node and corresponds the column number of the generator matrix G [1]. Table 1, explains the LT encoding process.

Table 1: Encoding Algorithm of the LT codes

Algorithm 1: LT Encoding
1. ***Repeat***
2. *Select a degree d_n from the degree distribution $\rho(d)$*
3. *Select uniformly at random d_n distinct input message bits*
4. *Perform modulo-2 addition among the selected d_n input message bits*
5. ***Until*** *maximum number of the encoded bits are generated*

4. An Example of Encoding

Figure 8 shows a simple example of the encoding process. Here number of symbols, $K = 3$; number of bits per symbols, $n = 1$. In an encoding session, say for example, four degrees $\{d_0, d_1, d_2, d_3\} = \{1, 3, 2, 3\}$ generated from to a degree distribution, and by selection d variable nodes at random uniformly, the generator matrix G is formed as shown in Equation (2).

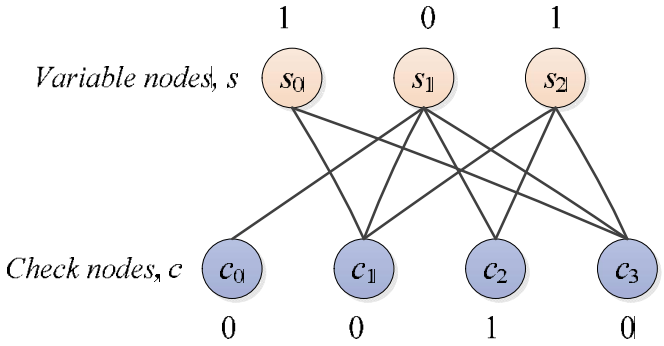

Figure 8. The LT encoding process with the generator matrix G.

For a given message {1, 0, 1} the encoded output will be {0, 0, 1, 0} according to the Equation 1, and encoding algorithm. From the graph of Figure 8, c_1 will be generated as $s_0 \oplus s_1 \oplus s_2 = 1 \oplus 0 \oplus 1 = 0$. Rest of the check nodes will be generated similarly.

$$G = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \tag{2}$$

5. Degree Distributions

Degree distribution is the important part of the design, and the efficiency and performance of the codes depend on it. Luby proposed two distributions- Ideal Soliton Distribution (ISD), and Robust Soliton Distribution (RSD). Besides these distributions, there are other distributions, such as Pareto degree distribution, power degree distribution, Shokrallahi proposed distributions for Raptor code in [4].

Ideal Soliton Distribution

The distribution is defined in Equation (3). Here, $\rho(d)$ represents the probability of degree d. The expected degree of this distribution can be found rough by $\log_e K$ [1], [2].

$$\rho(d) = \begin{cases} \frac{1}{K} & \\ \frac{1}{d(d-1)} & \text{for } d = 2, 3, ..., K \end{cases} \tag{3}$$

This degree distribution performs poorly, because of poor distribution and generation of degree one. It means, in the decoding process there will be no degree one check nodes and, sometimes a few variable nodes will have no connections at all with any of the check nodes [1].

Robust Soliton Distribution

As the performance of ISD was not as good as expected, RSD was formulated as following.

$$S = c \cdot \sqrt{K} \log_e (K/\delta) \tag{4}$$

$$\tau(d) = \begin{cases} \frac{S}{Kd} & \text{for } d = 1, 2, ...(K/S)\text{-}1 \\ \frac{S}{K}\log_e(S/\delta) & \text{for } d = K/S \\ 0 & \text{for } d > K/S \end{cases} \tag{5}$$

$$\mu(d) = \frac{\rho(d) + \tau(d)}{Z} \tag{6}$$

where, $Z = \sum_d [\rho(d) + \tau(d)]$.

S is the expected number of degree-one check nodes and is defined in Equation (4) with a constant c of order 1 and decoding failure probability δ [1], [2]. It will ensure sufficient number of single-degree check nodes required for decoding.

Finally RSD, $\mu(d)$ can be found from the Equation (6) after normalizing the summation of $\rho(d)$ and $\tau(d)$ by Z.

6. LT Decoding

The only purpose of degree-one check nodes is to initiate decoding process, and if there is no single degree check node due to erasure or poor degree distribution, the decoding may halt or fail. The decoding algorithm is mentioned in Table 2.

Table 2: Decoding Algorithm of the LT codes

Algorithm 2:LT Decoding
1. ***Repeat***
2. ***if*** $d = 1$
3. $s_k \leftarrow c_n$
4. $c'_n \leftarrow (c_n \oplus s_k)$ (*update the check nodes*)
5. $d \leftarrow d$-1 (*release edge or reduce degree*)
6. ***Until*** *all input bits recovered*

It is required to reconstruct the generator matrix G before the decoding process. The decoder will receive the transmitted bits and decode it. If some of the bits are erased or lost due to the adverse effect of the BEC, it will discard those bits and corresponding column from the matrix G which is shown in Figure 9 by *ash* color and letter 'X'. After matrix construction and getting intact check nodes, the decoder will start decoding and in short, it will perform following steps one by one.

- Identify the check nodes having degree one. Here, column index and row index of the degree-one check nodes are determined.
- Value assignment, *i.e.* $s \leftarrow c$.

- Check nodes updates – the check nodes which have edge connection with the recently recovered variable node (RRVN) *s*, are updated to make those check nodes independent of the *s*.

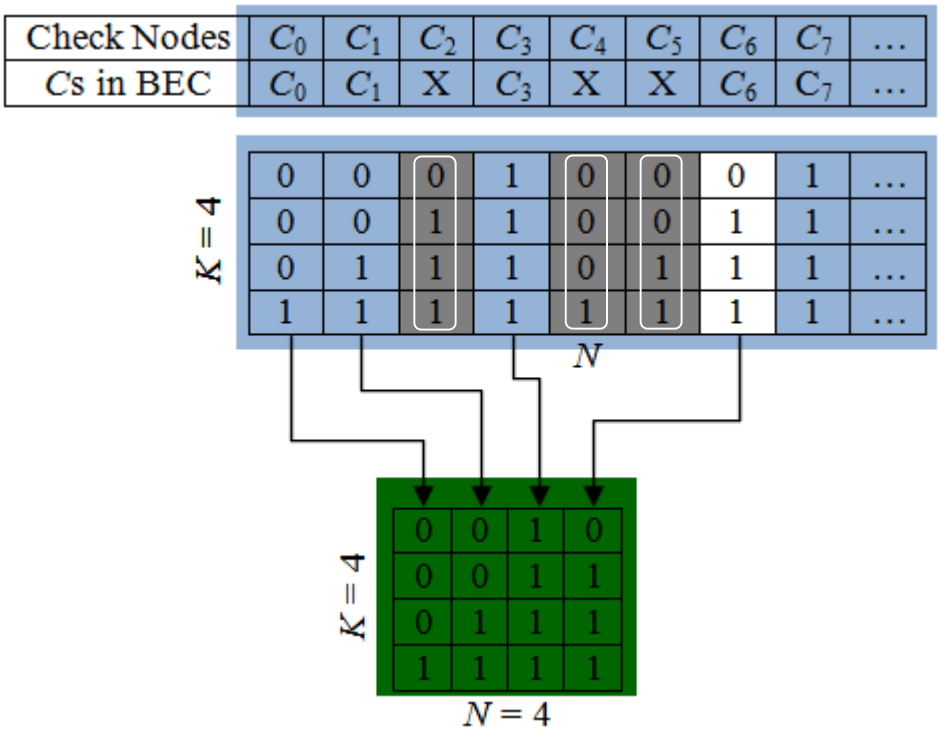

Figure 9. Check node selection and corresponding generator matrix formation for 4 bit message and encoded output bits.

- Edge updates or matrix updates. The edge connections among the RRVN *s* and updated *c*s are no longer required. Therefore, those edges are replaced with 0s, which means there are no edge connections.

The above process will continue until all the variable nodes are recovered.

There are other decoding processes except the above mentioned one. These are

- Belief propagation (BP) based soft decoding

- Gaussian elimination based decoding etc.

In the BP based decoding, the algorithm is realized by sum product algorithm (SPA).

C. An Example of the LT Decoding

In a decoding session, for a given G and encoded output bits {0, 0, 1, 1}, it is required to find out the transmitted message.

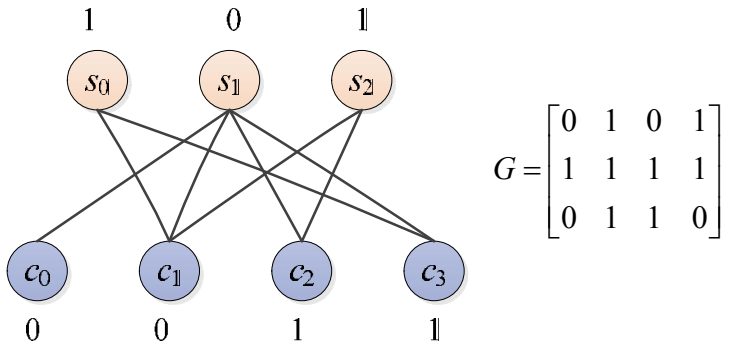

Figure 10. Decoding process for a given matrix G and check nodes.

Step 1: Find the check nodes having degree 1. Here c_0 has the degree one. Its column position, $col\_pos = 0$ and row position, $ro\_pos = 1$.

Step 2: Assign the value of degree-one check node to the variable node. Here the value of c_0 is assigned to s_1 as

$$s[ro\_pos] = c[col\_pos]$$

$$\text{or, } s[1] = c[0].$$

And ultimately s_1 will have 0.

Step 3: Update the check nodes. s_1 is connected to all four nodes. All nodes are needed to be updated except c_0. This is done by the XOR operation with c_0 and all the check nodes which are connected to the RRVN, s_1. Therefore, after update the check nodes will have {0, 0, 1, 1}.

Step 4: Update the generator matrix, *G*. This operation is performed as

$$G[ro\_pos] = 0$$

and after update, the matrix will have the following form.

$$G = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

Above steps will be repeated until all the message signals are recovered. At the end recovered message will be {1, 0, 1}.

D. Applications

Fountain codes like LT code has a number of applications. It can be applied where data are lost due to the erasure nature of the medium; or in another word, the LT codes can be applied in the system, which can be modeled as the BEC.

This is age of information and usually magnetic hard drives are widely used to store information on it. However, this device is quite unreliable and stored data are lost forever at a rate of 0.001/day [1]. LT codes can be used to store data on it. All the data will be LT encoded, and can be stored on the drive here and there like spray. If some of the encoded symbols are lost, all the data can be recovered from the intact rest of the encoded symbols.

LT codes can be used to efficiently broadcast data without interruption. Consider that ten thousand subscribers are watching world cup football from a satellite where if a frame is lost at the receivers, these will send ARQ to the sender. If the erasure probability, $e = 10^{-3}$, on an average $e \cdot K$ packets will lost in every house for K number of transmitted packets. In the traditional network, each house will request for retransmission of the lost packets. Total retransmission request will be $10{,}000eK$. It means the broadcast unit may need to retransmit the all packets again. Therefore, it will transmit the entire broadcast more than one to ensure that all the subscribers can watch the match perfectly.

However, if the packets are encoded with LT codes, each subscriber needs to receive only slightly more than K number of packets to recover the whole message which will ensure congestion free transmission and low work load of the broadcast unit.

Random code construction is the important characteristic of the LT codes. The variable nodes which will be used for encoding are selected uniformly at random and the number depends on the degree, d. This random selection is also challenging part for the design. Degree-one check nodes are merely used to initiate the decoding process. Usually without degree-one check node decoding is not possible. Sufficient number of degree-one check nodes is ensured by the well defined degree distribution.

IV. RELATED WORKS ON HARDWARE ARCHITECTURE OF THE LT CODEC

In the previous chapter, basics of the LT codes, encoding, decoding, and applications of the codes are discussed with figures and examples where required. A generator matrix can be constructed from the degrees and this matrix can be applied for encoding and decoding process of the LT codes.

Now in this chapter, related works on hardware design and architecture of the LT codec are discussed. In the last couple of decades, digital system design and hence electronics has made a rapid change in design process and automation, and opened an unlimited possibilities. On the other hand, due to some important characteristics, wide spread applications and very high impact on future communication, the LT codes have drawn much attention of the digital system design engineers. They are taking challenges to design the universal LT codec with fascinating specifications.

A. Already Proposed Architectures

As mentioned earlier that the LT codes have some very important characteristics to solve some of the problems of present technology. Therefore, this has drawn much attention of the students from different universities, academicians and researchers. Some are working on the LT codes and others are working on the architecture of the codec.

Han in his master's thesis [21] presented two BEC channel models for the performance analysis of the LT code. He has also presented hardware architecture of the LT encoder and decoder. Here, variable nodes are chosen or the generator matrix is produced by taking the *degree* and *index* from *degree generator* and

index register's output respectively. In another register called *neighbor*, variable nodes' positions are stored according to the *degree* and *index*. However, address or the *index* stored randomly. Then according to the *index* and *degree* values *neighbors* are selected. *Index* will indicate the initial address of the *neighbor*, and according to the *degree*, subsequent positions will be selected as shown in Figure 11. It also includes three-stage architecture of the decoder.

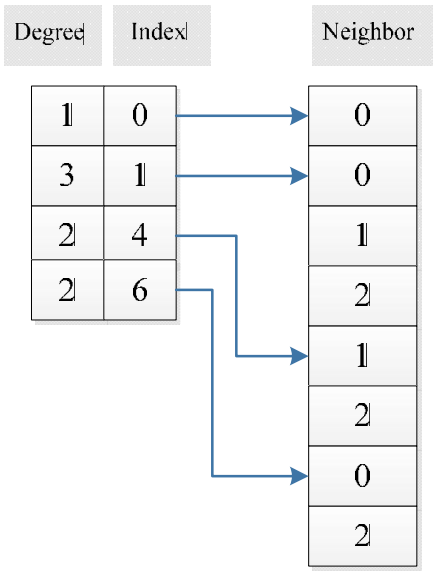

Figure 11. Method of generator matrix generation proposed in [21].

Kai *et. al.* [22] have also followed almost the same architecture of the generator matrix, and the codec along with router and reverse router. The architecture was used for storing non-zero elements in the generator matrix for Belief Propagation (BP) based soft decoding.

Alam *et. al.* in [23] have implemented the LT codec in ASIC using TSMC and Samsung libraries, and showed the performance in terms of area and cycle count. Here they used empirically modified RSD and almost same idea and architecture as

shown in Figure 11 for matrix construction. They also used the BP based soft decoding of the LT codes.

For the matrix construction, almost same idea and architecture was used but Han used hard decoding while rest of the previous works used BP based soft decoding and the corresponding algorithm and architecture.

V. PROPOSED HARDWARE ARCHITECTURE OF THE LT ENCODER

In the previous chapters, LT codes, encoding-decoding algorithms and different hardware implementations of the LT codec are discussed. Encoding-decoding algorithms are formulated, which can be used to find the hardware architecture of the codec. In this chapter, a new hardware architecture of the LT encoder is presented as Luby proposed the codes, with necessary figures and explanation.

A. Proposed Overall Architecture of the LT CODEC

Hardware design for the LT codec is quite challenging due to its random code construction characteristic and variable data length. However, Figure 12 shows the overall architecture of the LT codec. Here an encoder may contain

- A random number generator (RNG),
- A degree distribution unit (DDU) or degree generation unit (DGU),
- A generator matrix generation (MG) unit,
- A permutation unit (PU) and finally
- A check nodes generator (CNG).

In short, a RNG will generate a random number; the DDU will receive the random number as input and generate a degree according to RSD or any other predefined definition of the distribution. The PU will generate $K!$ permutations ideally and choose one, and send permutation information to matrix generator, MG. It will make edge connections by assigning 1s (or 0s for no edge connection) between

check and variable nodes according to the magnitude of degree generated by the DGU.

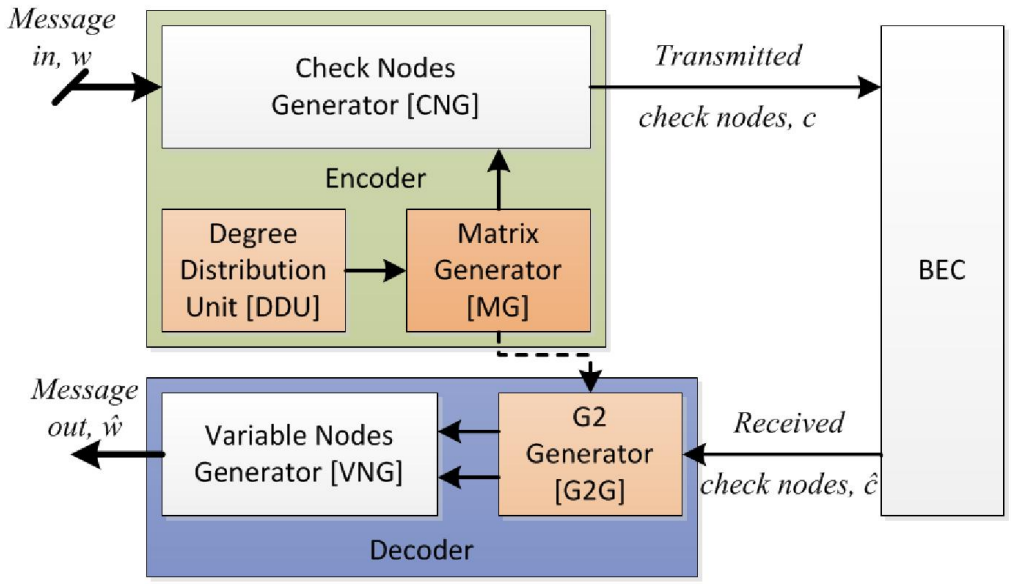

Figure 12. Overall hardware architecture of the LT codec as Luby has proposed the codes.

If the value of a degree is 2, there will be two 1s in the column of the generator matrix, *G* as shown in the previous chapters. Finally the check nodes generator, CNG, will generate *N* number of encoded output bits where each of the check nodes will be generated by *d* variable nodes. Usually, these encoded output bits will be transmitted through a BEC.

The decoder will receive the transmitted bits and decode it. If some of the bits are erased or lost due to the negative impact of the BEC, it will discard those bits and corresponding columns from the matrix *G*. Practically, a decoder will also have all the units like encoder except the CNG to decode the message. At the receiving end, it can generate *G* with the help of same *seed* and degree distribution.

B. Design Hierarchy

Usually there are two different types of design hierarchies as mentioned earlier. These are

1. Top-down and
2. Bottom-up and

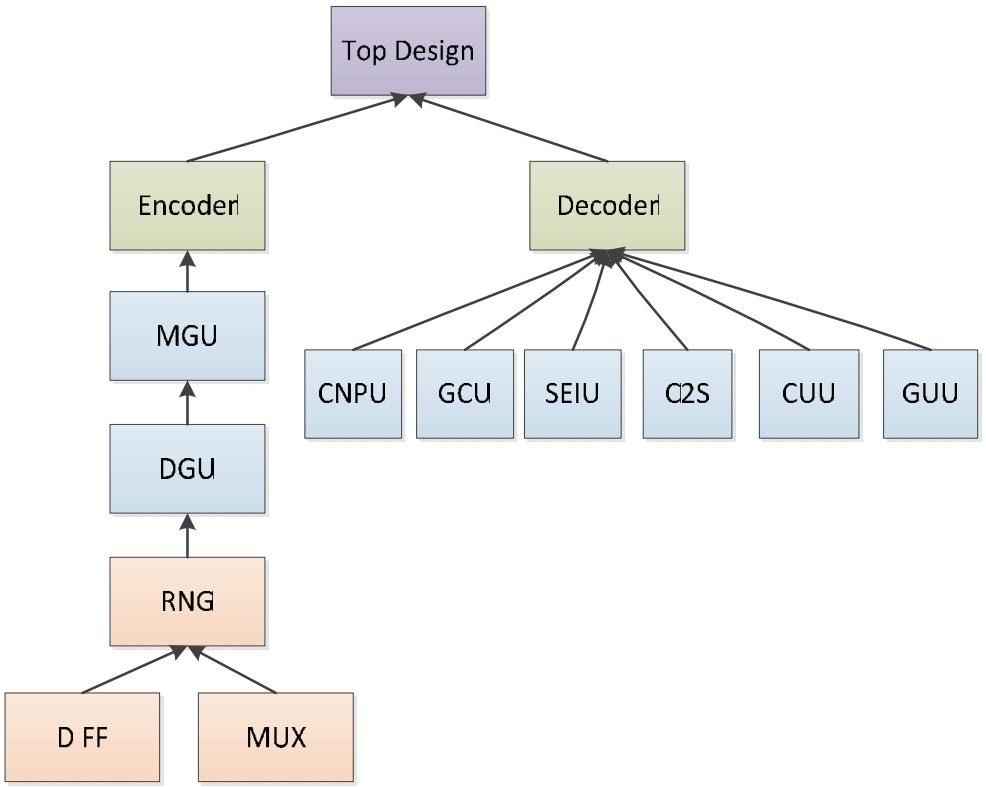

Figure 13. Bottom-up design hierarchy is followed for the LT codec hardware design.

In this hardware design, bottom-up type design hierarchy is followed as shown in Figure 13. To generate random number, Linear Feedback Shift Register (LFSR) is used, which consists of the leaf cell D flip-flop and 2-to-1 MUX.

Similarly upper-level units are designed. Some functional units are designed and integrated with another unit to make it easy and manageable. Figure 14 shows the hardware architecture of the codec where the encoder part is emphasized.

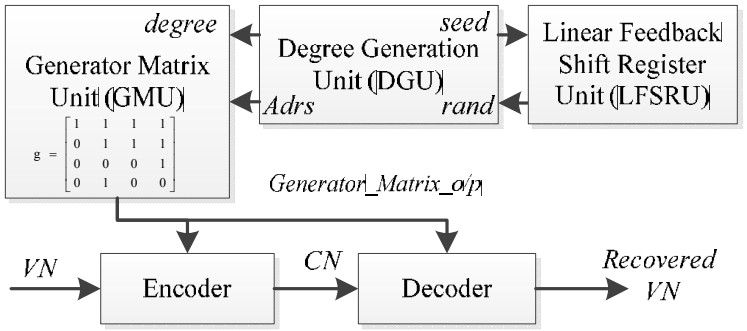

Figure 14. Hardware architecture of the codec where the encoder part is shown in detail.

Each unit is described following in brief.

C. Random Number Generator Unit

The code construction nature of the LT codes is random as mentioned earlier. Thus its performance mostly depends on the randomness of the degree and random construction of the generator matrix, *G*. Random numbers are independent and identically distributed (IID). Natural phenomenon like cosmic background radiation, atmospheric noise, etc. measured over a very short time, show true randomness [24]. Different devices, like gamma ray counters, noise diodes [25], two resistors with different temperature [26] can be used to generate true random number. However, these devices make a system complex and may not provide desired outputs [25]. On the other hand, if an algorithm is used to generate random numbers, usually it will provide an impression of randomness, however, the output

will not be true random because of its repeatable patterns, since this is computed in deterministic way. This random number is known as pseudorandom number.

Definition: *A pseudorandom number generator is a structure* $\mathcal{R} = (S, s_0, T, U, G)$, *where* S *is a finite set of states*, $s_0 \in S$ *is the initial state* (*or seed*), *the mapping* T: $S \rightarrow S$ *is the transition function*, U *is a finite set of output symbols*, *and* G: $S \rightarrow U$ *is the output function*.

The states are finite, therefore, the output of the generator will revisit its previous outcomes [25]. Hardware implementable several pseudorandom number generators are described in the following sections.

1. Linear Feedback Shift Register

The shift register with feedback, whose output bit is a linear function of its previous sate, is known as Linear Feedback Shift Register (LFSR).

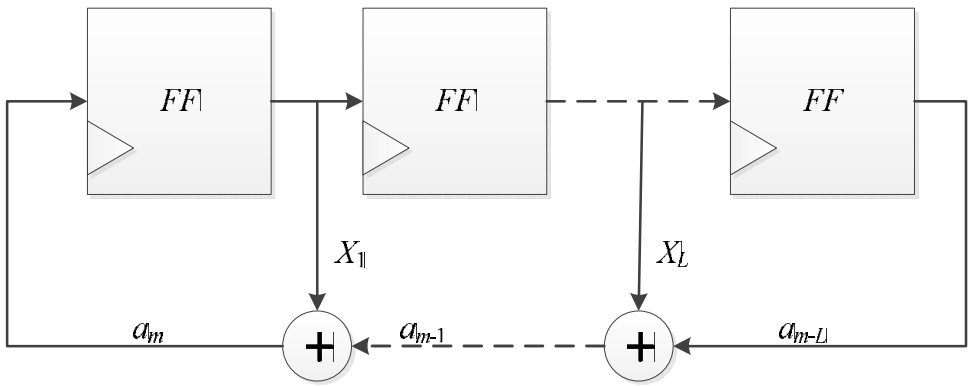

Figure 15. An L-stage typical linear feedback shift register.

A typical linear feedback shift register of length L is shown in Figure 15. The output of the mth iteration can be found as

$$a_m = \sum_{i=1}^{L} a_{m-i} X_i \tag{7}$$

where X = 0 or 1, known as taps and depends on feedback connection [27]. An 8-bit LFSR is designed with the feedback or characteristic polynomial $P(X)$ is shown in Figure 16 [28].

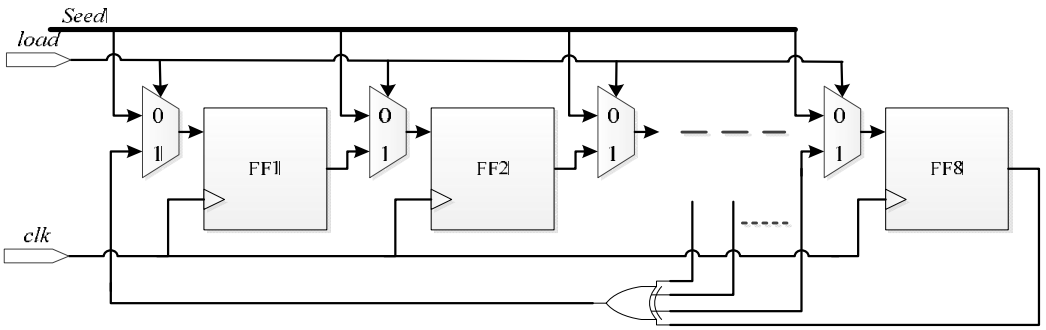

Figure 16. An LFSR random number generation unit designed using the polynomial P(X).

$$P(X) = X^8 + X^7 + X^6 + X^5 + X^4 + 1 \tag{8}$$

Figure 17. Output waveforms of the LFSR RNG.

Here, using leaf cell D flip-flop and 2-to-1 MUX, upper level unit of the LFSR is designed. Different *Seed* can be loaded to obtain different pattern of the random

number. Figure 17 shows the output waveforms, where *out* is the 8-bit output and *ran* is the randomly selected 4-bit output of the RNG.

2. Linear Congruential Generator

The mathematical relation that is used for the linear congruential generator (LCG) can be found by the Equation (9).

$$X_n = (a \cdot X_{n-1} + b) \bmod m \tag{9}$$

$$X_n = c \bmod m \tag{10}$$

where X_{n-1} is previous number and X_n is the next random number, a is the multiplier, b is the increment, and m is the modulus.

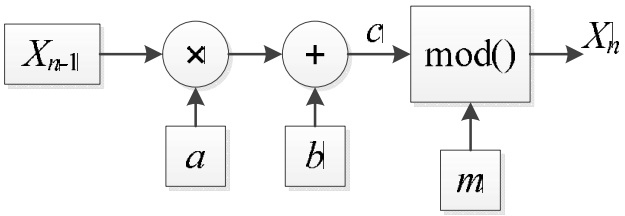

Figure 18. Architecture of the LCG.

And above parameters must satisfy the following conditions.

$$0 < m,$$
$$0 < a < m,$$
$$0 \le b < m,$$
$$0 \le X_0 < m$$

Here, X_0 is the initial value or *seed* and this can be changed to get different pattern of random numbers [25]. It is easy to implement the Equation 9 and Figure 18 shows hardware architecture of the LCG.

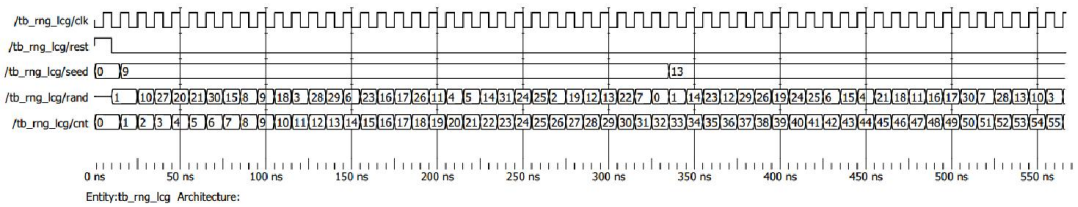

Figure 19. Output waveforms of LCG RNG.

In the *mod* unit, *m* will be subtracted from *c* until the subtracted result is less than *m* or zero. Figure 19 shows the output waveforms of the random number generated by the LCG and *rand* is the output random number of the unit.

D. Degree Generation Unit

Figure 20 pictorially describes the proposed architecture of the degree generation unit (or degree distribution unit). Whatever the name is, it will take the random from the RNG unit and generate random numbers according to the distribution defined in it. The mathematics behind the generation of the random number according to a distribution is the comparison of the random number with cumulative sum of the given probabilities as mentioned in Equation 11. It will receive a random number from the RNG unit which is using a LCG or linear feedback shift register and the number is generated according to a *seed*.

$$rand \leq csum[k] \tag{11}$$

The DGU generated random numbers will be used as degree for the subsequent units. Any degree distribution can be loaded any time if superior degree

distribution is found instead of RSD. From the stored probabilities, cumulative probabilities are calculated according to the formula

$$csum[j] = csum[j\text{-}1] + prob[j]|_{j\neq 0} \tag{12}$$

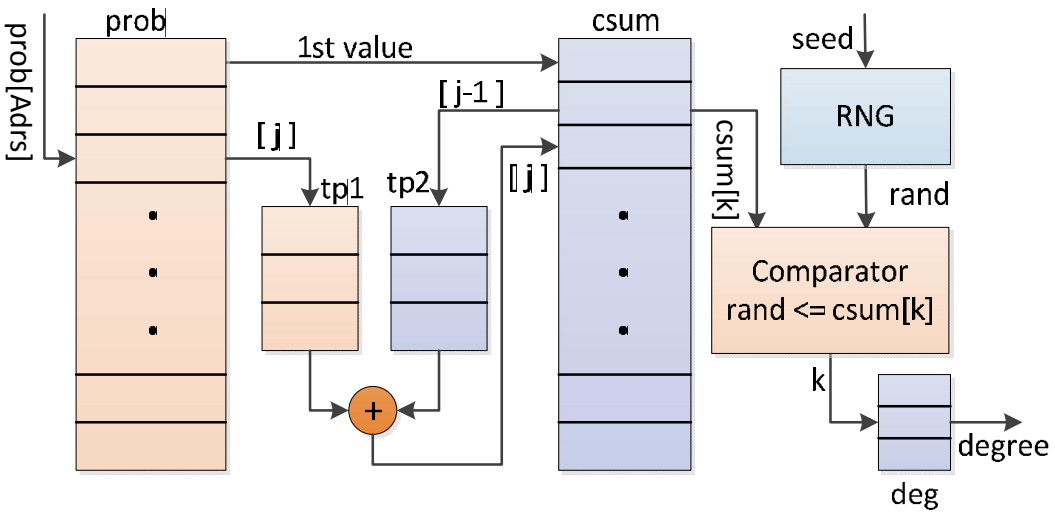

Figure 20. Hardware architecture of the degree generation unit. tp1 and tp2 are the temporary registers for holding probability (prob) and cumulative sum of the probabilities (csum) respectively. prob and csum are memories for storing probabilities and cumulative sum respectively.

with the help of two temporary registers tp_1 and tp_2, and stored in memory *csum*. On the other hand, a random number, *rand* is received from RNG unit. This random number is compared with every cumulative sum of probabilities. There will be at least one value of *csum* for which *rand* will be less than or equal to *csum*. First value will be considered, and degree can be found as *deg*[*k*] where the memory *deg* stores the degrees according to the probability. Here, *seed* can be changed to generate different sequence of the RNG.

E. Proposed Hardware Architecture of the Generator Matrix

The generator matrix-generation unit (GMU) will receive degree from DGU. This degree means how many 1s there will be in a column of the generator matrix *G*. If degree is 3, there will be three 1s in a column. This is one of the critical parts of the LT encoder. In the LT encoding process, *d* variable nodes will be selected uniformly at random.

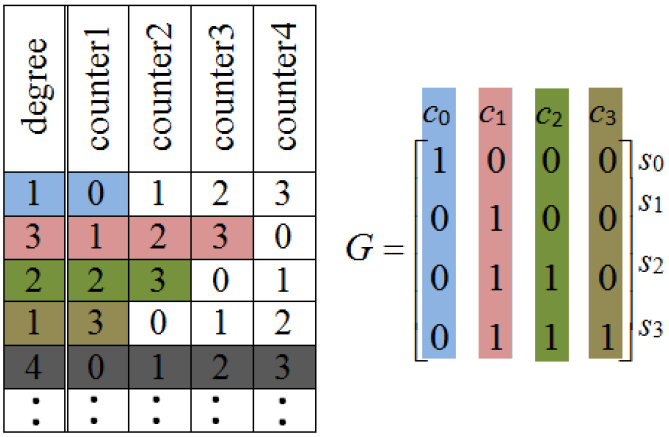

Figure 21. Generator matrix, G generation using degrees and different counters of unlike initial values.

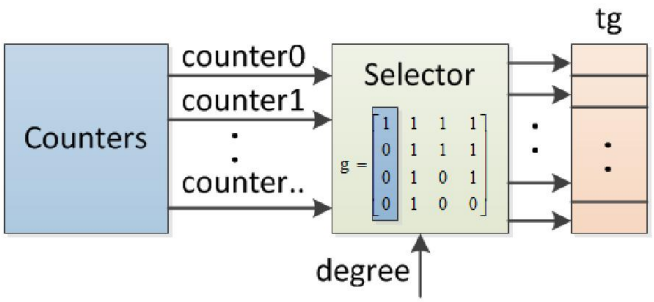

Figure 22. GMU architecture with different counters with unlike initial values.

To do that it is better to have the permutations of *K* message bits. Design becomes

critical as K grows. In this architecture, ten counters with different initial values are proposed to get the effect of permutations and randomness. The idea is shown in Figure 21. These different initial values of the counters will make sure that the counts are non-overlapping. For example, there are four counters with different initial conditions. For the column or address 01, the received degree is 3. Then the GMU will select first three counters and a 1 will be stored in every row position indicated by the corresponding counter's count. That is, here $counter_1 = 1$, $counter_2 = 2$, and $counter_3 = 3$. In column c_1, it will have 1 in row position 1, 2 and 3; rest of the position will have 0s. In this way, a single column of the generator matrix, *scog*, is generated and stored in the temporary register *tg*, as shown in Figure 22, to send it to the encoder and decoder units. Counters can be customized with different initial values and increment with some conditions to have better sparse matrix.

F. Proposed Hardware Architecture of the LT Encoder

In this section, check nodes or encoded output bits are generated from a column of

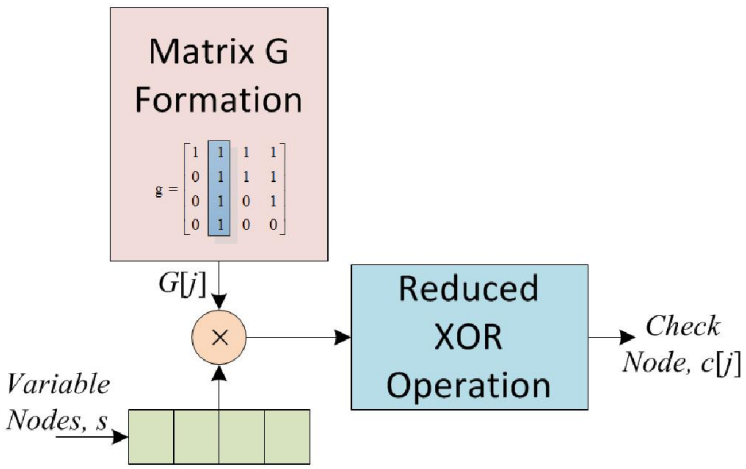

Figure 23. Here a column is multiplied with the message bits s to get d message bits and then these d message bits are added together by modulo-2 addition which produces a single bit of the respective check node.

the generator matrix, G and input variable nodes, s. Figure 23 shows the encoder module and Figure 24 shows the logical operation in detail. A column number of the matrix is used as the address. According to the address, the respective column of the G will be chosen and these values will be sent for the logical multiplication or AND operation with the variable nodes, s.

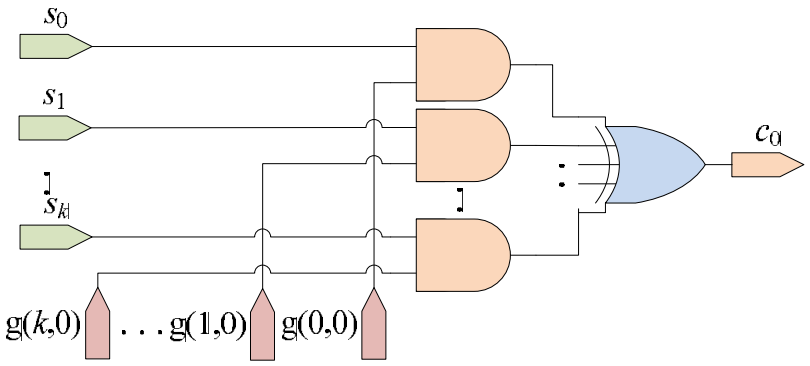

Figure 24. Multiplication and modulo-2 addition can be implemented by this simple logical operation.

Figure 25. Output waveforms generated by the proposed LT encoder.

From this multiplication, corresponding d variable nodes will be selected as the column contains d 1s and (K-d) 0s. These selected variable nodes will be added together by modulo-2 addition or reduced XOR operation which will provide the encoded output bit of a single check node.

There will be *N* such units, as shown in Figure 24 for *N* number of check nodes. Figure 25 shows the output waveforms generated by the proposed encoder.

In this chapter, the design hierarchy that has been followed is presented. Different units of the LT encoder are discussed. In the check nodes generation, first generator matrix is constructed, then encoding operation is done with the help of the generator matrix, and the output waveforms show that the encoder is encoding well along with its all units and subsequent levels. Proposed decoder architecture will be discussed in the next chapter.

VI. PROPOSED HARDWARE ARCHITECTURE OF THE LT DECODER

Random number generation, degree distribution, performing permutation and selecting any one of the permutation, are the pretty critical parts of the encoder section. However, the proposed encoder is working perfectly. In this chapter, the proposed hardware architecture of the LT decoder is presented.

A. LT Decoder

This is one of the challenging sections of the LT codec. Major functional unit are mentioned below and overall architecture is shown in Figure 26.

- Reception of the intact (existent or not erased) check nodes by the CNPU (check node processing unit) of a receiver.
- Generator matrix, *G* construction for the corresponding received check nodes by the GCU (generator matrix construction unit).
- Identification of the degree-one check node with the SEIU (single edge identification unit) - that determines row index and column index of a single edge in a column of the matrix, *G*.
- The value of *c* is assigned to *s* by the C2S assignment unit.
- Check node register updating by the CUU (check nodes update unit) and
- Matrix updating by assigning zeros to every column in a particular row index by the GUU (generator matrix update unit).

These functional units are discussed next.

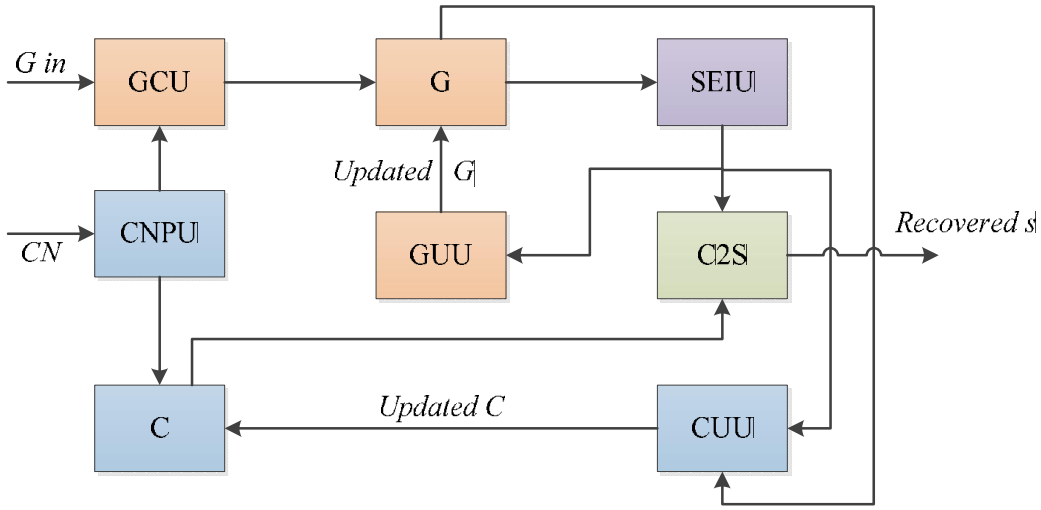

Figure 26. The detail architecture of the LT decoder.

B. Check Node Processing Unit

Figure 27 shows the architecture of the check node processing unit which actually will receive the intact check nodes generated by the encoder unit and will discard the erased check nodes. Here, input check nodes will be stored in *cin* register and first $K'(K' \geq K)$ intact check nodes will be received according to the following equations.

$$c[A] = cin[i] \tag{13}$$

$$A = A + 1 \tag{14}$$

If a single check node $cin[i]$ is not erased and if it is within first K' bits, it will be received in $c[A]$ according to the Equation 13; and the address A will be updated according to the Equation 14.

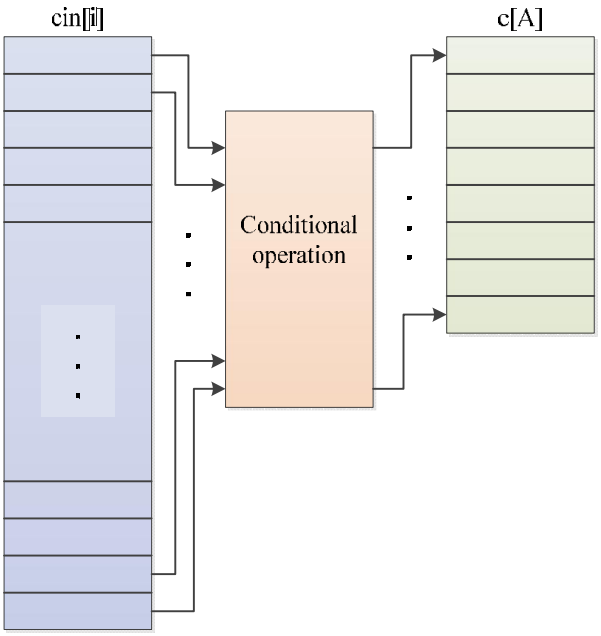

Figure 27. The architecture of the CNPU.

Figure 28. Output waveforms of the check node processing unit.

Figure 28 shows that first 128 bits of *cin* are lost, and in the last 128 bits it contains two 1s- one in the 128th and second one in the 255th bit positions while rest 126 bits have zeros. That is, last 128 bits which contain 10000…..0001 are accepted as intact bits for further processing.

C. Generator Matrix Construction Unit

Figure 29 shows the architecture of the generator matrix, *G* construction unit. It will use the same conditional operation as the CNPU and will perform the operation as Equation 15. The address will be updated as previous one according to the Equation 14.

$$g[A] = g\_in[i] \tag{15}$$

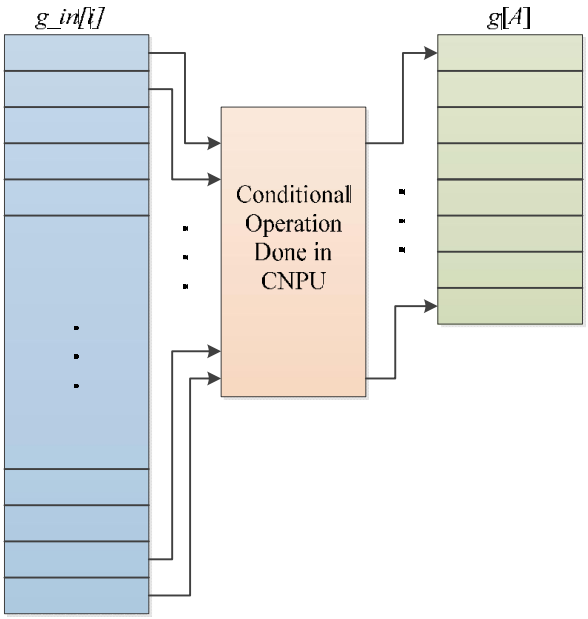

Figure 29. Generator matrix G construction unit.

D. Identification of Single Edge Check Nodes and C2S Units

Here, the column sum of the matrix *G* is determined to find the single edge check nodes, and hence its row and column positions for the use of next sections. A complete column is copied to temporary register *tg* and column sum is calculated and stored in *tsum*. If the value of *tsum* is 1, a 1 is stored in *se_flag* register at the address of column index.

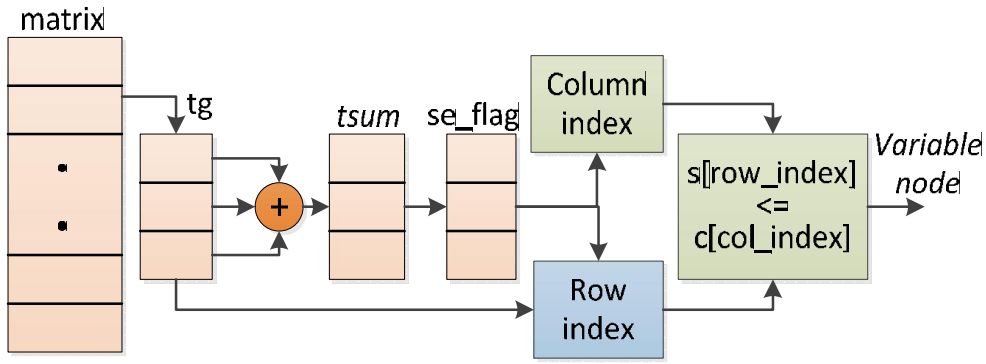

Figure 30. SEIU and C2S units – column sum calculator, finding the row and column indexes of a single edge check node and the value of the check node is assigned to the variable node.

The same *se_flag* register and *tg* are used to determine the row position or index of the single edge in a column by searching the 1 in *tg*. The proposed architecture is shown in Figure 30. Then the value of *c*, is assigned to the *s* register according to the column and row indexes, as

$$s[row\_index] = c[col\_index] \tag{16}$$

This variable node, *s* is known as recently recovered variable node (RRVN).

E. Check Nodes Update Unit

Now the check nodes in *c* must be updated to make these independent of the newly recovered variable node or RRVN. This architecture is depicted in Figure 31. Here, the value of *c* which is very recently assigned to *s* is stored in temporary register *tc*. Now, every *c* which is connected to the RRVN, *s* is updated by doing XOR operation with *tc*, and the updated *c* values are stored in the respective indexes. The Equations (17) and (18) explains the above operations clearly.

$$tc = c[col\_index] \tag{17}$$

$$c[j] = c[j] \oplus tc\big|_{j \neq col\_index} \tag{18}$$

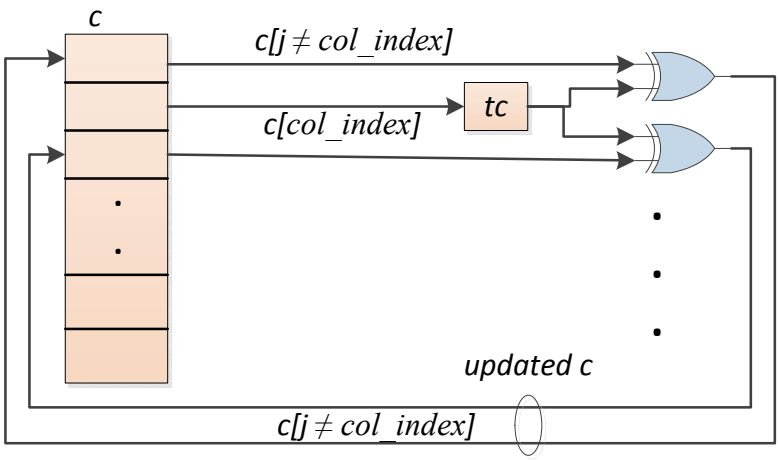

Figure 31. The register where c is stored is updated by performing XOR operation with every check node or c which is connected with the newly recovered s and c value which is recently assigned to the s.

F. Generator Matrix Update Unit

Next step is to update the matrix, G to make the check nodes free or disconnected from the RRVN, s. This is shown in the Figure 32. Here, each column of the generator matrix is copied to the temporary register tg_2 and this tg_2 is updated as

$$tg_2[row\_index] = 0 \tag{19}$$

and all the contents of tg_2 is stored in matrix G according to the column indexes. After this operation, entire row will have 0s in every column. In this way the edge update or G update operation is completed by the GUU using the indexes information of SEIU.

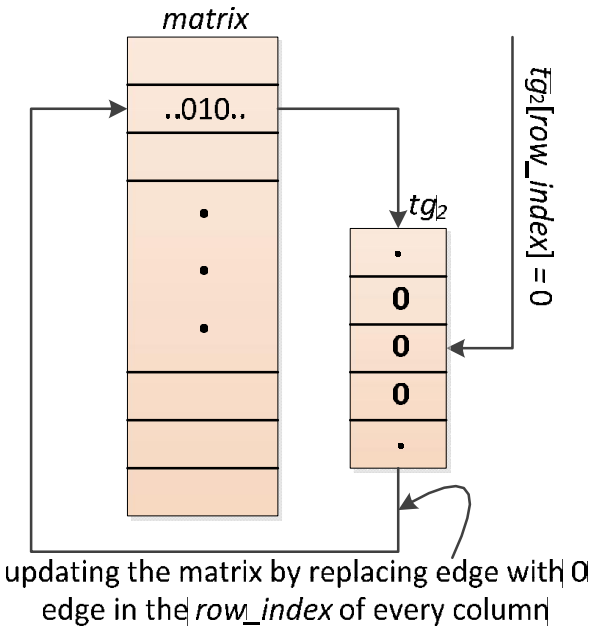

Figure 32. Generator matrix, G update unit. Each column of the generator matrix is updated in the respective row position by replacing the edge value 1 with 0.

G. Input-output Waveforms of the CODEC

Using above architecture, a complete codec is designed, compiled and simulated using Verilog® HDL in ModelSim. As mentioned earlier that some of the units are integrated with other units to make it easy and operationally manageable. Figure 33 shows the input-output waveforms of the codec. It takes arbitrary message 128'hefac….43ff as the input or variable nodes (*sin*) and generates 256'hfffff….dedc155 as the encoded output (*cout*) of the encoder. These are used as the input of the decoder and generate 128'hefac….43ff at the output (*sout*) of the decoder which are actually the recovered variable nodes.

Actually, a decoder will also have all units for the construction of the generator matrix for the given distribution and *seed*. However, here in the proposed

architecture, encoder generated matrix is also used in the decoder unit for simplicity.

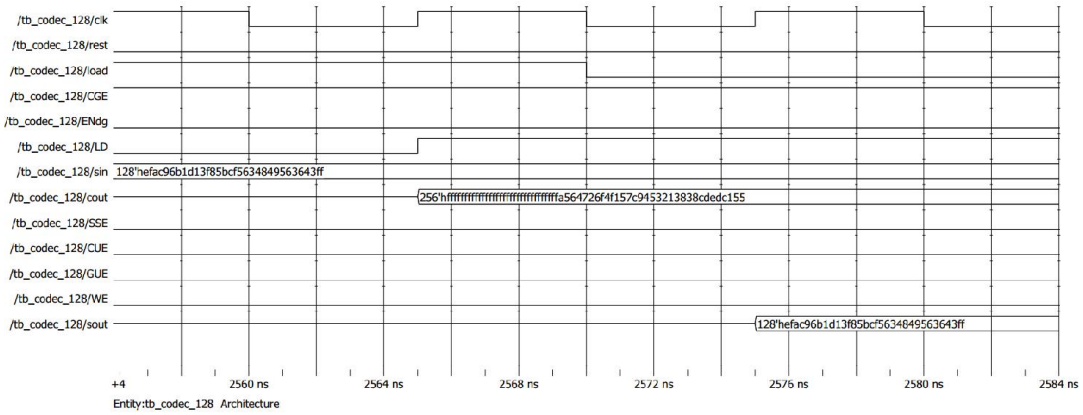

Figure 33. Input-output waveforms of the LT codec.

After getting the intact check nodes and the corresponding matrix, it will start to calculate the sum of a column and find out the single edges of the matrix, and hence at the end it will get the column and row indexes of a single degree check node. With these addresses, the decoder will assign the check node value to a variable node which eventually is the recovered variable node. Then it will update the check nodes and the matrix sequentially. These steps will continue until all the variable nodes are recovered.

VII. SIMULATION RESULTS

In the previous chapters, method of digital system design, LT codes, encoding-decoding, hardware architecture of encoder and decoder are discussed. Here design, simulation and results are the main points of discussion.

A. Performance of the Proposed Degree Distribution Unit

A hardware architecture is proposed which can generate random numbers of any predefined distribution. The performance of the degree distribution unit depends on the performance of the generation of the input random number. Here, a linear feedback shift register is used with different *seed*s for the generation of the random numbers. Figure 34 shows the output waveform of the degree distribution unit. It generates output random numbers according to the predefined degree distribution almost exactly. *ran* is the output random number or the degree.

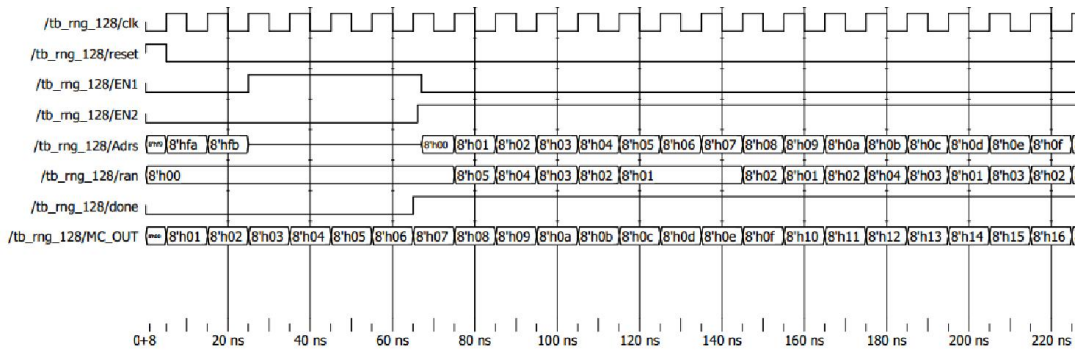

Figure 34. ran (degree) is the output random number generated by the DGU (or DDU) according a distribution and it follows the distribution almost exactly.

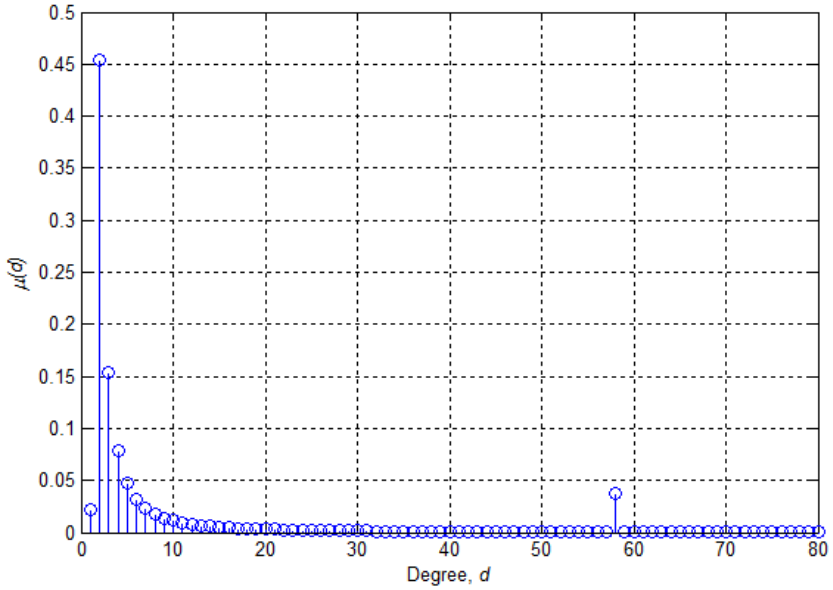

Figure 35. RSD degree distribution for K = 128, δ = 0.2, and c = 0.03.

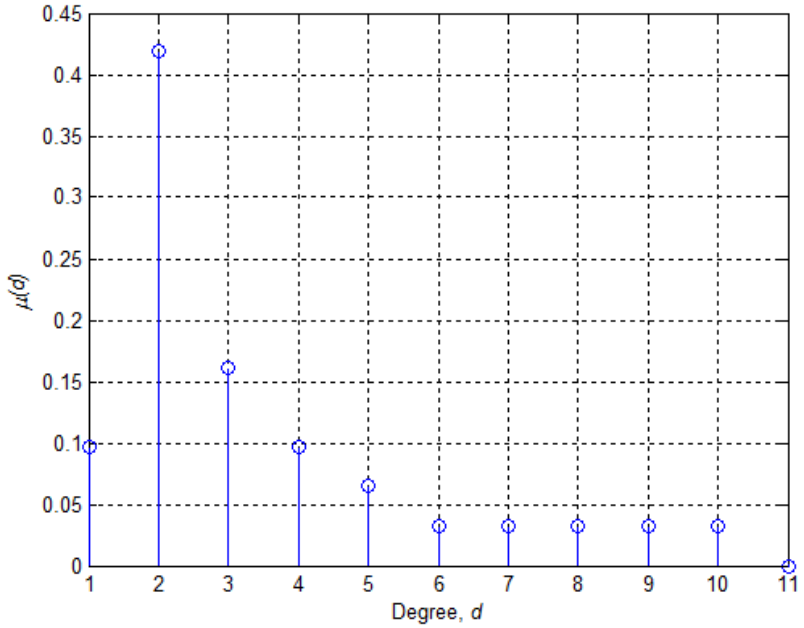

Figure 36. RSD degree distribution implemented by the proposed hardware.

Figure 35 shows the RSD degree distribution and Figure 36 shows the hardware implemented degree distribution. An important feature of the architecture is that any distribution can be implemented in it as shown in Figure 37. The performance of the DGU is shown by the hardware implemented or approximated distribution, which is pretty well following the original distribution.

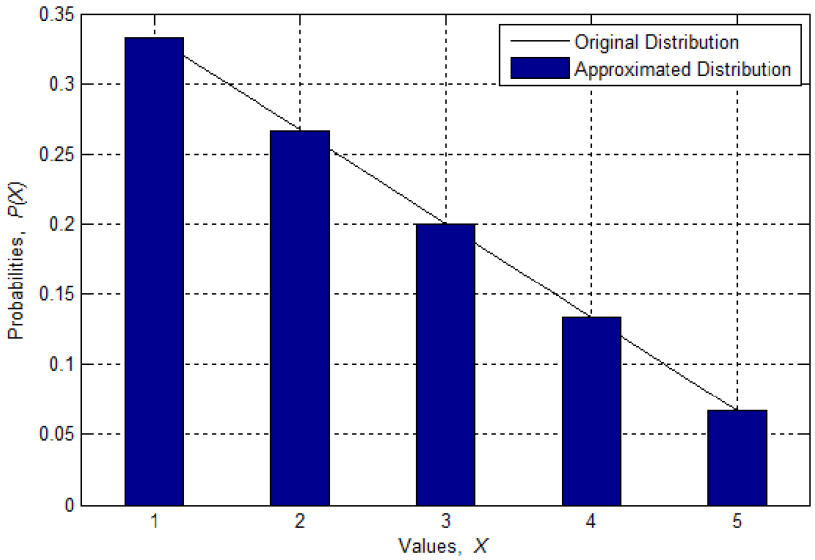

Figure 37. Performance of the degree distribution unit, DDU.

B. Performance of the Proposed Architecture of the LT CODEC

The LT codec architecture was designed as Luby proposed the LT codes in his paper. Since the LT code is a fountain code and it will ideally generate limitless bits at the output. However, in practice, this is not feasible. Therefore, this architecture is designed for 128 message bits and 256 encoded output bits. Figure 38 shows the input-output waveforms of the codec. Here, *sin* represents the input variable nodes or *s*; *cout* represents the encoded output or check nodes generated by the encoder; and *sout* indicates the decoded or recovered variable nodes or $\hat{s}$. 128'hefac96……..3643ff are used as message bits and encoded output bits are

256’hfff…… dc155. These 256’hfff…… dc155 are used by the decoder to recover the original message bits 128’hefac96……..3643ff.

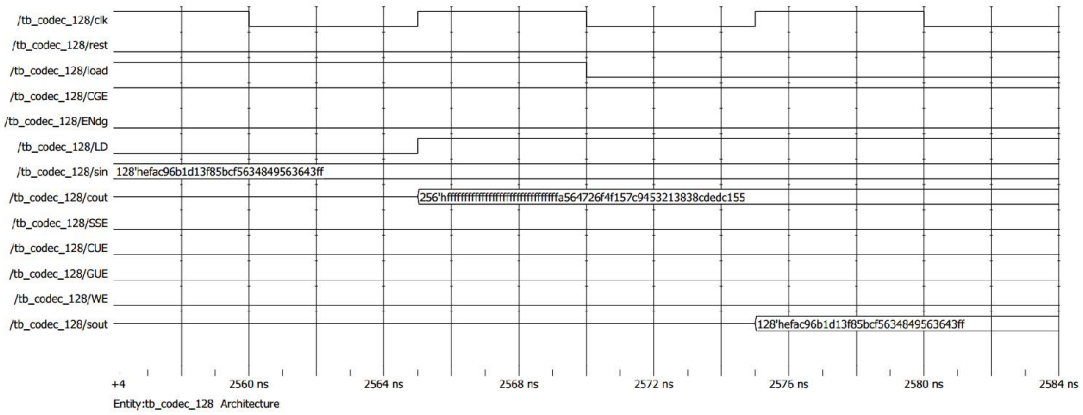

Figure 38. Input-output waveforms of the LT codec.

Using Verilog® HDL and ModelSim tool, a new and fully functional LT codec architecture is proposed with a specially designed degree generator; permutation was performed by different initial values or time shifted counters, the encoder and decoder units were designed as the Luby proposed the codes in his paper. Any kind of degree distribution can be implemented with this architecture and the result-waveforms and figures shows expected performance. Here, multiple counters were used to get the permutation result for encoding to have an effect of pretty randomness. The encoder and decoder performed quite satisfactorily. The Figure 38 shows that decoding operation is completed in 2575 ns for 100MHz clock. This means the encoding and decoding operation takes 257.5 cycles to complete the operation.

VIII. CONCLUSIONS

In the previous chapter, the results are discussed. In this chapter, summary of the whole works are presented and the recommendation of the future work will be used to conclude the chapter.

A. Summary

The whole works can be divided into three main categories.

1. Hardware architecture of the proposed degree distribution unit.
2. Hardware architecture of the proposed LT encoder and
3. Hardware architecture of the proposed LT decoder.

The code construction nature of the LT codes is random. To design an efficient LT codec, a good random number generator, degree distribution unit, permutation unit and a random selector are required. Here, an efficient a hardware architecture is presented which can generate random number according to the distribution. It will take a random number and compare it with the cumulative sum of probabilities; based on the comparison result a random number will be generated which will follow the defined distribution.

In the proposed LT codec architecture, first generator matrix is constructed with the help of degree, *d* (random number according to the distribution) generated by the DGU. Each column of the generator matrix is generated from the degree, and using different initial valued counters to provide the pretty well random permutation effect. This degree will select *d* counters having different initial values to avoid collision in the matrix generation. A single 1 is placed in the column

where the value of a counter indicates the row position and the degree determines the number of 1s and hence the number of counters. The matrix will be stored in a memory for encoding and decoding. Each column is used to perform AND operation with the variable nodes to obtain the *d* variable nodes. These *d* variable nodes are used for reduced XOR operation to obtain a single check node. This process is repeated until sufficient numbers of check nodes are generated.

In the decoding, matrix based hard decision decoding was used, which did not include any inverse matrix operations, and the process is named as non-Belief Propagation based decoding algorithm.

In the decoder, after getting intact check nodes and the corresponding generator matrix, sum of each column is calculated and hence the position of the degree-one check node is identified. This position is used to recover a check node, update the check nodes, and afterwards to update the generator matrix. This process will continue until all the variable nodes are recovered.

In short, here an architecture of the LT codec is proposed as Luby has described the code, and simple matrix-based encoding and decoding process have been adopted. Matrices are stored in memories, and an address indicates the respective column. Encoding and decoding processes do not have complex mathematical operations such as tanh(), ln(), etc. as these are found in soft BP (Belief Propagation) decoding process.

Instead of using LUTs (Lookup Tables) and predefined arrays, here simple random number generator and degree generator are used to get a distribution. Different initial valued counters are used to generate the generator matrix to get pretty well random effects.

It takes 257.5 *cycle counts* and 2.575 μs for encoding and decoding instead of 5,204,861 minimum *cycle counts* and 4.43s of the design mentioned in [23] where iterative soft BP decoding was used in ASIC and ASIP implementation of the LT codec.

B. Future Work

In this, proposed LT codec architecture, matrix based encoding and decoding methods are used as Luby described this code in his paper. However, Hamming distance based decoding has some fascinating features. For example, it can decode even if there is no degree-one check node. In future, this algorithm can be studied, formulated and based on the algorithm an architecture can be designed to obtain a complete or universal LT codec which will not be prone decoding failure problem due to the lack of degree-one check node.

On the other hand, the performance of the LT codes improves as the number of input variable nodes increases. Thus, 2048-input LT code can be designed with parallel processing.

REFERENCES

[1] D. MacKay, "Fountain Codes," in *IEE Proceedings- Communications*, 2005.

[2] M. Luby, "LT Codes," in *43 rd Annual IEEE Symposium on Foundations of Computer Science*, 16–19 November 2002.

[3] P. Elias, "Coding for Two Noisy Channels," in *3rd London Symp. Information Theory*, London, U.K, 1955.

[4] A. Shokrollahi, "Raptor Codes," *IEEE Transactions on Information Theory,* vol. 52, no. 6, pp. 2551-2567, 2006.

[5] M. Asim and G. Choi, "An Optimized Framework for Degree Distribution in LT Codes Based on Power Law," *J. Cent. South Univ.,* pp. 2693-2699, 2013.

[6] T. Richardson and R. Urbanke, Modern Coding Theory, NY: Cambridge University Press, 2008.

[7] "3GPP TS 26.346, Techn. Spec. Group Serv. and Sys. Aspects; Multimedia Broadcast/Multicast Service (MBMS); Protocols and codecs, V7.4.1," June 2007.

[8] T. Mladenov, S. Nooshabadi and K. Kim, "Efficient Incremental Raptor Decoding Over BEC for 3GPP MBMS and DVB IP-Datacast Services," *IEEE Transactions on Broadcasting,* vol. 57, no. 2, p. 313, June 2011.

[9] M. Bohr and K. Mistry, "Intel's Revolutionary 22 nm Transistor Technology," Intel, May 2011.

[10] S. Palnitkar, Verilog HDL A guide to Digital Design and Synthesis, SunSoft Press , 1996.

[11] P. J. Ashenden, Digital Design: An Embeded Systems Approach Using Verilog, MA: Morgan Kaufmann Publishers, 2008.

[12] Wikimedia, "Full Custom," Wikimedia Foundation, Inc, 15 March 2013. [Online]. Available: http://en.wikipedia.org/wiki/Full_custom. [Accessed 2 April 2014].

[13] U. V. C. Laboratory, "Design Styles," 2005. [Online]. Available: http://vlsicad.ucsd.edu/courses/ece260b-w05/pdf/ece260b-w05-design-style.pdf. [Accessed 02 April 2014].

[14] R. Brayton and J. Cong, "NSF Workshop Report: Electronic Design Automation Past, Present, and Future," 08 July 2009. [Online]. Available: http://cadlab.cs.ucla.edu/nsf09/. [Accessed 02 April 2014].

[15] Wikipedia, "Electronic Design Automation," Wikimedia Foundation, Inc, 05 April 2014. [Online]. Available: http://en.wikipedia.org/wiki/Electronic_design_automation. [Accessed 02 May 2014].

[16] K. Tatas, K. Siozios, D. Soudris and A. Jantsch, Designing 2D and 3D Network-on-Chip Architectures, New York: Springer, 2014.

[17] S. Lee, Advance Digital Logic Design Using Verilog, State Machines, and Synthesis for FPGAs, Canada: Thomson, 2006.

[18] Wikimedia, "GDSII," Wikimedia Foundation, Inc, 15 May 2014. [Online]. Available: http://en.wikipedia.org/wiki/GDSII. [Accessed 18 May 2014].

[19] J. Cavanagh, Verilog HDL Digital Design and Modeling, FL: CRC Press, 2007.

[20] S. Puducheri, J. Kliewer and T. E. Fuja, "The Design and Performance of Distributed LT Codes," *IEEE Transactions on Information Theory,* pp. 3740-3754, 2007.

[21] H. Wang, "Hardware Designs for LT Coding," Delft University of Technology, Netherlands, 2006.

[22] K. Zhang, X. Huang and C. Shen, "Soft Decoder Architecture of LT Codes," in *IEEE Workshop on Signal Processing Systems*, Washington, 2008.

[23] S. M. S. Alam and G. Choi, "Design and Implementation of LT CODEC Architecture with Optimized Degree Distribution," *IEICE Electronics Express,* pp. 1-10, May 29, 2013.

[24] Wikimedia, "Random Number Generation," Wikimedia Foundation, Inc, 25 April 2014. [Online]. Available: http://en.wikipedia.org/wiki/Random_number_generation. [Accessed 01 May 2014].

[25] J. Banks, Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice, USA: John Wiley & Sons, 1998.

[26] R. e. d. R. Franco, "R.F. elettronica di Rota Franco," 2014. [Online]. Available: http://www.rf-microwave.com/datasheets/2582_generic_NSG-10A_01.pdf. [Accessed 01 May 2014].

[27] J. L. Massey, "Shift Register Synthesis and BCH Decoding," *IEEE Transaction on Information Theory,* Vols. IT-15, no. 1, pp. 122-127, 1969.

[28] P. Schaumont, "Virginia Tech Department of Electrical and Computer Engineering," 2012. [Online]. Available: http://rijndael.ece.vt.edu/schaum/slides/ddii/lecture6.pdf. [Accessed 21 April 2014].

ACKNOWLEDGEMENTS

First and foremost, I offer my sincerest gratitude to my supervisor Professor GoangSeog Choi for his excellent guidance, caring, patience, and providing me with comfortable research environment. His encouragements to pursue my master's degree at SoC Design Lab and continuous support have been a great assistance to achieve a significant milestone in my career. Without his help, this thesis would not have been completed. He is the friendliest, caring, easy going and encouraging advisor that anyone could wish for.

I would also like to show appreciation to all faculty members of the Department of Information and Communication Engineering, Chosun University, specially the members of the thesis examining committee Professor Jae-Young Pyun, and Professor Young-Sik Kim for their valuable advices, comments, and insight throughout my research. In addition, I would like to thank my department to provide me the atmosphere to cultivate and augment my knowledge.

My family members in Bangladesh have always supported me with their distant love and sympathy. The painful and cruel experience and fact of being away from home, friends and family helped me to realize the value and importance of friends and family in every instance. I would like to thank my respectable mother, father and my dear brother, sisters for their constant prayers during my study, support and condolence.

I would also like to show my gratitude to all the members of Chosun University, specially to the members of International Office and Graduate Schools.

During my daily life in Korea, I have been supported by lab-mates, classmates, dorm-mates, cheerful friends and always smiling local people. Specially S. M.

Shamsul Alam, Muthukumar, Shajeel Iqbal, 구정주, 김철홍, 전준철, 안성민, 김범석, 선장균, 박한빛, 정대성, and 장현정 will remain in my mind forever for their philanthropist nature and generous assistances. Whenever I got stuck, they helped whole heartedly.

This research was supported by the NIPA under the Global IT Scholarship program and the NIIED (National Institute for International Education). I am really grateful to the Korean Government and the people of Korea. The honour is mine to be with them.

Graduate School of Chosun University
GwangJu, Republic of Korea (South)

CERTIFICATE OF APPROVAL

MASTER'S THESIS

May, 2014

This is to certify that the master's thesis of

Md. Tariq Hasan

has been approved by the examining committee for the thesis requirement for the Master's degree in Engineering

Committee Chairperson ..... 변재영
Prof. Jae-Young Pyun

Committee Member ..... 김영식
Prof. Young-Sik Kim

Committee Member ..... 최광서
Prof. GoangSeog Choi