Power Characterization of a Digit-Online FPGA Implementation of a Low-Density Parity-Check Decoder for WiMAX Applications

by

Manpreet Singh

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2014

AUTHOR'S DECLARATION

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Low-Density Parity-Check (LDPC) codes are a class of easily decodable error-correcting codes. Published parallel LDPC decoders demonstrate high throughput and low energy-per-bit but require a lot of silicon area. Decoders based on digit-online arithmetic (processing several bits per fundamental operation) process messages in a digit-serial fashion, reducing the area requirements, and can process multiple frames in frame-interlaced fashion. Implementations on Field-Programmable Gate Array (FPGA) are usually power- and area-hungry, but provide flexibility compared with application-specific integrated circuit implementations. With the penetration of mobile devices in the electronics industry the power considerations have become increasingly important. The power consumption of a digit-online decoder depends on various factors, like input log-likelihood ratio (LLR) bit precision, signal-to-noise ratio (SNR) and maximum number of iterations.

The design is implemented on an Altera Stratix IV GX EP4SGX230 FPGA, which comes on an Altera DE4 Development and Education Board. In this work, both parallel and digit-online block LDPC decoder implementations on FPGAs for WiMAX 576-bit, rate-3/4 codes are studied, and power measurements from the DE4 board are reported. Various components of the system include a random-data generator, WiMAX Encoder, shift-out register, additive white Gaussian noise (AWGN) generator, channel LLR buffer, WiMAX Decoder and bit-error rate (BER) Calculator. The random-data generator outputs pseudo-random bit patterns through an implemented linear-feedback shift register (LFSR).

Digit-online decoders with input LLR precisions ranging from 6 to 13 bits and parallel decoders with input LLR precisions ranging from 3 to 6 bits are synthesized in a Stratix IV FPGA. The digit-online decoders can be clocked at higher frequency for higher LLR precisions. A digit-online decoder can be used to decode two frames simultaneously in frame-interlaced mode. For the 6-bit implementation of digit-online decoder in single-frame mode, the minimum throughput achieved is 740 Mb/s at low SNRs. For the case of 11-bit LLR digit-online decoder in frame-interlaced mode, the minimum throughput achieved is 1363 Mb/s. Detailed analysis such as effect of SNR and LLR precision on decoder power is presented. Also, the effect of

changing LLR precision on max clock frequency and logic utilization on the parallel and the digit-online decoders is studied. Alongside, power per iteration for a 6-bit LLR input digit-online decoder is also reported.

Acknowledgements

First of all, I would like to thank my supervisor Dr. Vincent Gaudet for his guidance, support, funding, and confidence in me and my work.

I am very thankful to Dr. Philip P. Marshall and Si-Yun Li for providing me with all the necessary help for making this work possible.

Many thanks to all the research group members including Brendan Crowley, Si-Yun Li, Bahareh Ebrahimi, Navid Bahrani, Rachna Srivastava, Chris Ceroici, Pierce Chuang for their valuable advice and feedback from time to time.

A bunch of thanks to Dr. Hiren Patel and Dr. Liang Liang Xie for accepting to act as readers for my thesis.

I also would like to mention how grateful I am to have worked alongside with some amazing people like Dr. Edgar Mateos Santillan, Dr. Bill Bishop, Dr. Wayne Loucks, Dr. Rodolfo Pellizzoni, Dr. Marcio Juliato, Dr. Carlos Moreno, Najma Jose, Bahaedinne Jlassi, Mortaja AlQassab, Peter Dawoud, Farhad Haghighizadeh, Alborz Rezazadeh Sereshkeh, Roger Sanderson, during my assignments as a Teaching Assistant at University of Waterloo. Many of these people I run into during my casual walks through University corridors and we have some good chats about future, past and sometimes present.

And, I am thankful to University of Waterloo for providing me with the coolest office ever; I really enjoyed partially jealous looks from anyone who visited my office for the first time. I am so glad to have had some great office mates at University of Waterloo, mostly they are great because they were away most (all) of the time, leaving me as the sole owner of the office. Saud Wasly, thank you very much for the support and guidance in times of need. You are among the most astute people I know.

I had the good fortune of meeting many people at University of Waterloo; sadly, I won’t be able put down names of everyone. I would really like to thank you for being a part of my life, however small it was, for it contributed towards the kind of person I am today. Thanks.

Dedication

Dedicated to all my dear ones whose love and support helped me make it through

Table of Contents

List of Figures

List of Tables

.

Chapter 1
Introduction

Forward error correction is a common technique used to control errors when transmitting data over noisy and unreliable channels. By adding controlled redundancy to a stream of data, enough information can be provided to a receiver so that it can detect or correct the occurrence of most errors. Low-Density Parity-Check Block Codes (LDPC-BC) are a class of forward error-correcting codes proposed by R.G. Gallager in 1962 [1]. These codes can communicate data at very low bit error rates (BER), asymptotically reaching a capacity limit given by Shannon's capacity Theorem [2]:

$$C = B log_2 \left(1 + \frac{S}{N}\right) \tag{1.1}$$

In the case of an Additive White Gaussian Noise (AWGN) channel, for a specific bandwidth B and signal-to-noise ratio (S/N), the maximum possible information capacity (C) is given by Equation (1.1). In practical applications, effective utilization of available capacity is required since signal bandwidth and transmission power are usually limited. LDPC decoding typically relies on Iterative Decoding Algorithms implemented on Very-Large-Scale integrated (VLSI) chips. Parallel implementations of such decoders can run at very high throughputs. The area of parallel LDPC decoders increases rapidly with an increase in message precision, with interconnect wiring possibly occupying more area than actual logic units [3], [4]. In a successful attempt to reduce wiring congestion with increased message precision, digit-online decoders based on digit-serial arithmetic were proposed in [5]. Previously proposed bit-serial decoders [4], [6], which although proving effective at reducing wiring congestion, suffered from a disadvantage of not being able to perform consistent and continuous serial processing (some nodes work on a most-significant-bit-first basis, and others least-significant-bit-first). In contrast, digit-online decoders employ techniques of digit-online arithmetic [7], [8] and are capable of processing all the data in a consistent most-significant-digit (MSD)-first order at all

computational nodes. This allows for increased pipelining and reductions in clock cycle requirements, and therefore, higher throughputs can possibly be achieved.

In this thesis the work in [5] is extended, by studying and comparing bit-parallel and digit-online decoder implementations of WiMAX 576-bit, rate-3/4 codes that were able to fit on an Altera Stratix IV field-programmable gate array (FPGA). Using a power measurement setup from [9], [18], various aspects of both decoders are characterized and power numbers are reported. In Chapter 2 discusses LDPC Decoding and various popular LDPC decoder implementation techniques, and provides a thorough literature review of FPGA-based LDPC decoders. Digit-online arithmetic and digit-online decoding algorithms are discussed in Chapter 3. Chapter 4 explains the FPGA implementations of the decoder systems and discusses DE4 power measurement setup along with a discussion of power dissipation factors in FPGA. In Chapter 5 reports and discusses the simulation and synthesis results for bit-parallel and digit-online decoders for a WiMAX 576-bit, rate-3/4 LDPC code. BER curves and power results for the FPGA implementation are also reported. Chapter 6 concludes the thesis along with a discussion of future work.

Chapter 2
Background

2.1 LDPC Codes

Low-density parity-check (LDPC) codes are a class of linear error correcting codes defined by a sparse parity-check matrix **H** of dimensions M x N, such that all valid code words of size N (represented by vector **x**) satisfy the relation $\mathbf{Hx}^{\mathbf{T}} = \mathbf{0}$ [1]. For binary regular LDPC codes the number of 1's in **H** is fixed for each row and column, while for binary irregular LDPC codes the number of 1's is variable. The code rate of a block code is given by 1 – M/N. Lower code rates provide better error performance; however, with lower code rates the information content in a code word decreases and thus information throughput is reduced. There exists a trade-off where the designer chooses between the desired information throughput and code-rate. An example of a regular (3, 6) LDPC code H-matrix is shown in Figure 2.1. For a (3, 6) regular LDPC code, the number of 1's in a row of H-matrix is equal to 6 and number of 1's in each column is equal to 3. The distribution of 1's in H-matrix is random, and is required to keep the matrix sparse to guarantee a good-performance LDPC code. Tanner graphs [10] are bi-partite graphs that are popularly used to represent LDPC codes. Figure 2.2 shows the Tanner graph

$$H = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Figure 2.1: An H-matrix example of a (3, 6) regular LDPC code with N = 16 and M = 8, the code.

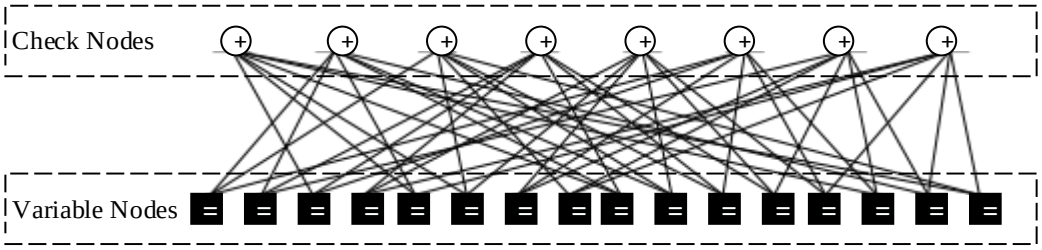

Figure 2.2: An example of a Tanner graph. Message passing occurs between the connected Variable and Check Nodes until all constraints are met or an early termination occurs after maximum allowed number of iterations

for the **H**-matrix represented in Figure 2.1. A Tanner graph consists of two sets of nodes: Variable nodes and Check nodes. The number of rows in the **H**-matrix corresponds to the number of check nodes in the Tanner graph while the number of **H**-matrix columns is equal to the number of variable nodes. Every element $\mathbf{h}_{ij} = 1$ in the **H** matrix corresponds to an edge between a variable node *i* and check node *j*. Since this is an example of a regular code, all variable nodes have equal number of edges and thus a degree $d_v = 3$ and all check nodes have a degree $d_c = 6$. To satisfy the constraint for a variable node, as represented by an equal sign, all the incoming symbols over the edges should carry equal values. For check nodes the constraints are satisfied if the XOR function calculated from all the input symbols results in a zero. The distribution of edges in the Tanner graph represents the LDPC decoder structure. Actual LDPC decoder designs are based on the Tanner graph design, with messages moving across the graph edges and nodes evaluating and iteratively trying to satisfy the constraints in the code.

2.2 LDPC Decoding

Given the size of the received block of data, if maximum-likelihood decoding is used to simultaneously satisfy all constraints, the algorithm required for finding the optimal solution will be NP-hard [11]. Therefore, for the purpose of decoding LDPC codes, iterative decoding

algorithms are suitable. Variable nodes are initialized with probabilistic information representing channel outputs, and messages are exchanged between the variable and check nodes until all the constraints are met. For the purpose of achieving higher throughput, the number of iterations is usually limited. The channel information entering the decoder is represented as log-likelihood ratios (LLRs), which is a relative probability specified on a logarithmic scale, and quantized to a desired bit precision. Mathematically, LLR can be expressed as:

$$\lambda(p_0,p_1) = \ln(\frac{p_0}{p_1}) \tag{2.1}$$

where p_0 is the probability that the transmitted bit could be a 0 and likewise p_1 is the probability that the transmitted bit can be a 1. As a general rule, higher bit precision LLR inputs are better for decoding since they provide more accurate channel information and help the decoder to converge faster. Increasing the bit precisions above 6 bits offers marginal improvement in the error performance [12]; however, increasing LLR precision increases the decoder input buffer and decoder size requirements significantly. For the case of serial decoders, increasing LLR bit precision is not much of an issue for decoder size; however, the throughput is negatively affected, as higher number of clock cycles are required to decode a code word. In highly pipelined architectures like the digit-online decoder, increasing LLR precision allows for deeper pipelining, which allows it to be clocked at higher frequencies. Thus, the throughput of digit-online decoders remains relatively unaffected despite increase in LLR precisions [15]. The type of quantization scheme also affects the error performance of the decoder; in [12] it is shown that using a non-uniform quantization scheme results in better performance than a uniform quantization scheme. In [13], [14], it is shown that using an optimized quantization scheme for 4 bits of precision achieves BER performance similar to the design using uniform quantization using 6 bits per message.

2.2.1 Sum-Product Decoding

As discussed previously, the data flow structure of LDPC decoders is based on bipartite graphs known as Tanner graphs; decoding uses iterative message-passing algorithms. One such algorithm is the sum-product (SP) algorithm, also referred to as belief-propagation (BP). The SP algorithm factors the complicated global function into a number of simpler local functions depending on a subset of the variables. The messages in each iteration of the SP algorithm are updated according to the SP update rule given in [16]. Variable node *v* outputs a message sent on edge *e* to check node *c*, which is the sum of the local function with all the messages received at *v* on all the edges other than *e*; this constitutes a half iteration. For the next half of the iteration, check node *c* outputs the product of the local function with all the messages received at *c* other than the ones received at *e*. The variable and the check node operations can be represented in form of following Equations (2.2), (2.3) and (2.4). The channel LLRs that are provided to the decoder variable nodes in the beginning of decoding are denoted as L_c.

$$\text{Variable Node}: \quad \lambda_i = L_c + \sum_{\substack{j=1 \\ j \neq i}}^{d_v} L_j \tag{2.2}$$

$$\text{Varible Node, final iteration}: \quad \lambda_{Final} = L_c + \sum_{j=1}^{d_v} L_j \tag{2.3}$$

$$\text{Check Node}: \quad L_p = 2\tanh^{-1}\left(\prod_{\substack{p=1 \\ p \neq q}}^{d_c} \tanh\left(\frac{\lambda_q}{2}\right)\right) \tag{2.4}$$

d_v and d_c denote the variable node and check node degrees respectively. Check node also monitors if the constraints for the code are satisfied, and can indicate the variable nodes to terminate the decoding early and perform a hard-decision of the results.

2.2.2 Min-Sum Decoding

SP algorithm provides near-optimal decoding performance for LDPC codes [17], [19], but is inefficient to implement in silicon. The SP variable node implementation largely requires only summation operations, which is not very costly to implement; however, SP check node operations are relatively expensive to implement in hardware. For long low-rate codes, SP check nodes consume lot of silicon area and hence not considered a very area efficient choice. The min-sum (MS) algorithm provides a simpler implementation of check nodes. The check node update equation is modified to Equation (2.5) in MS algorithm:

$$L_p = |\lambda_{min,p}| \left(\prod_{\substack{p=1 \\ p \neq q}}^{d_c} \text{sign}(\lambda_q) \right) \tag{2.5}$$

where

$$\text{sign}(x) = \begin{cases} +1 & \text{for all } x \geq 0 \\ -1 & \text{for all } x < 0 \end{cases} \tag{2.6}$$

The variable node update equation remains the same as Equation (2.2) and (2.3). Although MS provides easier implementation than SP, it comes at a slight performance loss. The MS decoder implementation typically requires a few tenths of a dB more transmitted power at same BER performance than the SP decoder implementation. It was found, as explained in [20], [21] using the density-evolution of the MS algorithm that the output LLRs from the check nodes tend to have a higher magnitude than LLRs output by SP check nodes. It was suggested that it is possible to regain most of the coding loss by introducing minor modifications in the algorithm to somehow reduce the magnitude of MS check node output LLRs to match those of SP check nodes. Most popular modifications in use include the use of correction factors, and are referred to as: 1) Offset correction, and 2) Normalization techniques. The offset correction

$$L_p = \max(|\lambda_{min,p}| - \beta) \left(\prod_{\substack{p=1 \\ p \neq q}}^{d_c} \text{sign}(\lambda_q) \right) \tag{2.7}$$

$$L_p = \frac{|\lambda_{min,p}|}{\alpha} \left(\prod_{\substack{p=1 \\ p \neq q}}^{d_c} \text{sign}(\lambda_q) \right) \tag{2.8}$$

involves subtraction of an offset value (β) from all the incoming LLRs in the check nodes. The normalization technique involves dividing the LLRs in check node by a factor α (α > 1), such that the resulting LLRs are smaller. The check node update equations for the offset correction (offset-MS) and the normalization (normalized-MS) are shown in Equation (2.7) and (2.8), respectively. Both techniques provide very similar performance benefits [21]. However, [20] notes that the results of the offset-MS can be readily extended for the discretized density-evolution cases, and suggests that the offset-MS is a good choice for practical implementation since it can be extended easily to quantized values.

2.3 LDPC Decoder Implementations and Comparisons

2.3.1 Parallel LDPC Decoding

Most straightforward implementations of parallel LDPC decoders instantiate a Tanner graph directly in hardware. The internal structure of the variable nodes and the check nodes depends on the decoding algorithm. Parallel LDPC decoder nodes process multi-bit LLR messages in parallel, so each edge in the Tanner graph corresponds to multiple interconnect wires.

One of the early parallel LDPC decoders was implemented in 160 nm technology [21], [22], decoding at a throughput of 1 Gb/s with 64 iterations per frame and clocked at a frequency of

64 MHz. Using parallel LDPC decoder implementations, high throughput and low power consumption per decoded frame can be achieved.

However, a parallel LDPC decoder architecture results in low logic-densities and high routing congestion. Most of the chip area is utilized for routing the connections between the variable and check nodes. While designing parallel LDPC decoders for long code sizes and high-precision input LLR messages, it becomes really hard to connect the nodes in the decoder. The number of parallel wires for each Tanner graph edge increases with increasing LLR precision. In the case of implementation of a parallel LDPC decoder on an FPGA where the routing resources are already limited, providing connections in the decoder places a lot of stress on the routing resources. A shared extrinsic memory is used between the variable node update units (VNUs) and the check node update units (VNUs) to achieve low routing congestion but highly parallel computations in [23]. Onizawa et al. report a variant of the parallel LDPC decoder in a new decoding algorithm referred to as a flooding-type update algorithm [24]. The longer wires between the nodes are divided appropriately in several sub-wires by insertion of flip flops; thus reducing the length of longer wires and enabling the decoder to be clocked at a higher frequency.

Apart from parallel decoder implementations, many "partially" parallel LDPC decoder implementations are presented in the literature. Zhang *et al.* implemented a partially parallel structured LDPC code decoder employing time-multiplexed routers to reduce parallelism and groups the highly connected check nodes and variable nodes in the decoder as local units [25], [26]. Zhong *et al.* present a partially parallel structured LDPC code decoder architecture with shared memory blocks for storing the iterative decoding messages and the channel messages [27], [28]. A clustering algorithm for partitioning a Tanner graph in clusters, such that the inter-cluster communications are minimized was proposed by Al-Rawi *et al.* [29]. To reduce the wiring congestion in the LDPC code decoders a message broadcasting technique is demonstrated in [30] which reduces the amount of information needed to be conveyed from the check nodes to the variable nodes. A half-broadcasting technique reduces average node-to-node wirelength by about 26%. Other techniques based on bit-flipping (BF), referred to as weighted-bit-flipping (WBF), modified-WBF (MWBF) and improved-MWBF have been

presented in [31], [32], [33] and [34] respectively. BF algorithm works in an iterative fashion and flips one or more bits per iteration until all the parity checksum are satisfied [34]. BF algorithm works well with LDPC codes, with performance losses of 1 dB or less in some cases as compared with sum-product algorithm [30].

2.3.2 Serial LDPC Decoder Implementation

A bit-serial implementation of an LDPC decoder is able to reduce interconnect and routing complexity to a large extent and allows for longer code lengths. The multi-wire connections between nodes are replaced by single wires transmitting bits serially, resulting in significant reduction in the interleaver wiring. This implementation however, comes at a price of reduced throughput and higher switching activity in the decoder interconnects. With increase in LLR message precision, more number of clock cycles will be required for decoding each code word, as the number of bits to be transmitted serially across the decoder nodes increases.

Implementing satisfactory throughput in serial LDPC decoders is challenging. In [23], [35] and [36] a bit-serial message passing scheme is presented. The check node update units (CNUs) process the data bit-serially in MSB-first sign-magnitude format, while the variable node update units (VNUs) process the data serially in LSB-first 2's complement format. Therefore, the messages need to be stored and format conversions are required at the output of VNUs and CNUs. To maximize the utilization of decoder hardware two successive frames are simultaneously decoded in the block-interlaced mode; thus, doubling the throughput. The bit-serial LDPC decoder implementation presented by Darabiha *et al.* [4] employs variable nodes with parallel adders. The messages arriving from the check nodes are buffered and converted to parallel inputs and addition operations are performed in a single clock cycle. Processing two frames at once allows for double the throughput; however, the design of parallel adders is much complex than serial adders.

Digit-online decoding of LDPC block codes was proposed by Marshall *et al.*[5] ,[15], which involves deeply pipelined digit-serial processing in the decoder. Instead of using conventional sign-magnitude format, messages in digit-online decoder are expressed in the form of signed-

binary digit format. The variable nodes and the check nodes process signed-binary format LLR messages in a most-significant-digit (MSD)-first fashion without requiring any changes in message format or order of digits at the node inputs.

2.3.3 Stochastic LDPC Decoding

Stochastic decoding is another method of reducing the interconnect complexity involved in LDPC decoders. Stochastic techniques were introduced in 1960s [37], and have been successfully used in implementation of neural networks [38] and are increasingly becoming popular in LDPC decoder implementations. Stochastic LDPC decoders are similar to bit-serial LDPC decoders in sense that extrinsic messages are communicated over single wires, thus interleaver wiring requirements are lower. The variable node and the check nodes operation can be implemented with simple architectures. For example, multiplication operation can be done as easy as ANDing two inputs together and a J-K flip flop can implement a division operation.

Therefore, stochastic LDPC decoders allow for low-complexity computational nodes and reduced routing congestion. Stochastic LDPC decoding uses a unique way of representing probabilistic messages as Bernoulli sequences indicating values in between 0.0 and 1.0 [39], [40], [41]. At the transmission side LLRs are turned into stochastic streams based on their probabilities, therefore, the encoding scheme is not unique and different sequences are possible for same probability values. For a message size of S bits, if r bits are 1 then the probability P represented by this message is given by:

$$P = (r/S) \tag{1.9}$$

The frequency of 1s in the probabilistic message is equal to the probability P. Stochastic decoders, however do not get as large of a coding gain as sum-product decoders. Also, they are sensitive to the level of switching activity in the decoder and result in reduced decoding performance with reduction in switching activity. The lack of switching activity (latching)

worsens at high SNRs in which received LLR probabilities reach either 0 or 1. There are different techniques proposed in the literature like NDS (noise-dependent scaling), regenerative EMs (edge memories) and IMs (internal memories), which help to provide high switching activity in stochastic nodes [40], [41]. A parallel stochastic decoder capable of a maximum throughput of 61.3 Gb/s and very low error floor is presented in [42].

2.3.4 Analog LDPC Decoders

Apart from digital LDPC decoders, an analog circuit for decoding LDPC codes was reported by Hemati *et al.* in 2006 [43]. The analog LDPC decoder employs iterative analog min-sum (MS) algorithm to decode a (32, 8) LDPC code. The node circuits in the analog decoder are based on current-mirrors and fabricated in 180-nm CMOS technology. The data throughput for the analog LDPC decoder is 6 Mb/s and power consumption is about 5 mW. However, designing analog LDPC decoders requires high process control to design accurate current mirror circuits. Analog LDPC decoders suffer from physical non-idealities such as component mismatch, thermal noise effects and short-channel effects; therefore, they have been only considered for small LDPC codes [44].

2.4 LDPC Decoder Implementation on FPGA

FPGAs provide reduced opportunities for parallelism but increased flexibility when compared with ASIC implementations of LDPC decoders. The routing requirements of the Tanner graph edges pose a lot of strain on the programmable interconnect fabric of an FPGA, especially for significantly longer routings. Since FPGAs are more suitable for datapath-intensive designs and have interconnects optimized for local routing [45], the implementation of less parallel and constrained LDPC decoders is more suitable for FPGAs. A parallel LDPC decoder synthesized on FPGA is reported by Zarubica *et al.* in [46] which is able to reach 12 Gb/s throughput. The biggest decoder in [46] to fit on the FPGA has a frame length of 1200 bits with 3-bit input channel LLR precision.

FPGAs also provide the benefit of already available extra hardware like block memories and DSP blocks. In [47] Zarubica *et al.* make use of the distributed memories available on the FPGA for storing the messages needed in the decoding iterations and suggests that use of memories can enable implementation of decoders for large code sizes. An implementation of 9984-bit block LDPC code decoder is reported in [47]. A partially parallel LDPC decoder architecture is synthesized on a Xilinx Virtex-5 FPGA for a rate-1/2 code of size 648 bits by Kim *et al.* [48]. A summary of the decoder along with reported power data for the decoder is shown in Table 2.1.

A bit-serial LDPC decoder presented in [4] is implemented on an Altera Stratix EP1S80 FPGA and occupies about 84% of logic resources. The maximum clock frequency of the decoder is 61 MHz and it can reach a throughput of 650 Mb/s with early termination. The use of FPGAs in implementation of stochastic decoders has been demonstrated by Tehrani *et al.* in [41] and [49], achieving throughputs upto 706 Mb/s and 1.66 Gb/s respectively.

Gunnam et al. present a multi-rate decoding architecture for irregular LDPC codes for WiMAX (IEEE 802.16) standard, resulting in significant savings in memory and routing requirements of the decoder [50]. Table 2.1 shows a summary of the decoders synthesized on the FPGA in [50]. A modified 2-bit min-sum LDPC decoding algorithm is proposed by Chandrasetty *et al.* resulting in a reduced implementation complexity of the decoder [51]. With a slight drop in the BER performance, the decoder can be implemented in about 18% less FPGA resources. An improvement on this implementation is presented in [52] by keeping different length intrinsic (LLR messages entering the decoder from channel) and extrinsic (LLR messages exchanged between the variable and the check nodes) messages. Reducing extrinsic message length reduces the interconnect complexity and also simplifies the check node operation. The LDPC decoder in [52] decodes rate-1/2 code words of size 1152 bits.

Two optimization techniques, vectorization and folding are presented by Chen *et al.* in [53] for effective utilization of block RAM resources available on the FPGA. Both these techniques build on the fact that the block RAMs available in FPGAs are of configurable aspect ratios and are dual-ported with very fast access times. Vectorization attempts to pack multiple messages

References	Darabiha et al., [4]	Zarubica et al., [46]	Gunnam et al. [50]	Tehrani et al. [36]	Kim et al. [48]	Sulek et al. [54]
Year	2006	2007	2007	2008	2010	2013
FPGA	Altera Stratix EP1S80	Xilinx V4 XC4VLX200	Xilinx V2 XC2V8000	Xilinx V4 XC4VLX200	Xilinx V5 XC5VLX155T	Xilinx V6 XC6VLX240T
LDPC Code Type	RS –based LDPC	Parallel PEG based (6, 3)	OMS Algorithm	Parallel Stochastic	Partially parallel	GF(q) LDPC Decoding
Code Size	(480, 355)	(1200, 600)	576 b (multi rates)	(1056, 528)	(648, 324)	(480, 240)
Throughput	650 Mb/s	12 Gb/s	41-70 Mb/s	1.66 Gb/s	110 Mb/s	6 Mb/s
LLR Message	3 bit	3 bit	5 bit	6 bit (input)	-	8 bit
Max. Iterations	15	10	-	700 (DC_{max}) (1DC/clk)	8	10
Reported BER	10^{-5} at 5dB*	-	-	10^{-8} at 4.25 dB	-	7×10^{-6} at 2 dB*
f_{Max}	61 MHz	100 MHz	110 MHz	222 MHz	100 MHz	180 MHz
Logic Utilization	84 % (66,588 LEs)	45%	-		-	7 %
Slices (Xilinx)	-	40,613	1,640	46,097	7,081	
LUTs	-	69,038	2,982	68,112	19,761	10,916
Block RAM	-	-	38	-	24	26
Decoder Power	-	-	-	-		
E/b	-	-	-	-		-

Table 2.1: Summary of some of the discussed FPGA based decoders

into the same word by utilizing configurable width of block RAM, while folding attempts to take advantage of configurable depth of block RAMs by allowing messages from different submatrices of the code to share the same physical block RAM. Li *et al.* present an FPGA implementation of 2.4 Gb/s, rate-1/2, (3, 6) convolutional encoder and decoder and discuss a detailed decoder power analysis for various aspects of the FPGA implementation [9]. In recent years, non-binary or higher-order Galois-field (GF) LDPC decoders implemented on FPGAs have been reported in literature. In case of Galois-field GF($q = 2^p$) codes the decoding complexity grows exponentially [54] with 2^{pd_c} (d_c is the maximal nonzero entities in parity check matrix row). Non-binary decoders benefit from the extra available FPGA resources like configuration logic blocks, block RAMs and multipliers. [55] also presents non-binary decoders for different order implementations. Table 2.2 shows the important aspects of the works mentioned in this section which report the power data for the FPGA implementations.

References	Chen'11 [53]	Chandrasetty'11 [52]	Li'13 [9]	This work	
Year	2011	2012	2013	2014	
FPGA	Xilinx V4 XC4VLX200	Xilinx V5 XC5VLX110T	Altera StratixIV EP4SGX230	Altera StratixIV EP4SGX230	
LDPC Code Type	Quasi-Cyclic Irregular	(3, 6) Regular MMS	PN- LDPC-CC (3, 6)	Digit-online LDPC decoding	
Code Size	(3969, 3213)	(1152, 576)	T_s = 192	(576, 432)	
Max.Throughput	1.474 Gb/s	11.7 Gb/s	2.4 Gb/s	740 Mb/s	1363 Mb/s
LLR Message	6 –bit	4 – bit (input)	4 –bit	6-bit	11-bit(interlaced)
Max. Iterations	15	10	9	10	10
Reported BER	-	$3x10^{-6}$ at 3.9 dB	10^{-8} at 4.25 dB	$1.2x\ 10^{-10}$ at 4 dB	-
f_{Max}	195.7 MHz	138 MHz	75 MHz	90 MHz	140 MHz
Logic Utilization	-	-	83 %	91%	90%
Slices (Xilinx)	62,362	10,823	-	-	
LUTs	98,003	39,024	-	-	
Block RAM	330	-	-	-	
Decoder Power	7632 mW	1130 mW	4105 mW	1248 mW (at 450 Mb/s)	1029 mW (at 450 Mb/s)
E/b	5.18 nJ	-	1.71 nJ	2.77 nJ(avg.)	2.29 nJ (avg.)

Table 2.2: Summary of FPGA based decoders with reported power data

2.5 Altera Stratix IV FPGAs

An FPGA is a semiconductor device consisting of programmable logic components, programmable interconnects and I/Os. Present-generation FPGAs also contain additional features such as configurable embedded SRAMs, DSP blocks and high-speed transceivers. FPGAs can be programmed to replicate the functionality of basic gates as well as complex logic functions.

An Altera Stratix IV FPGA core is fabricated in 40nm technology and is made up of multiple LABs (Logic Array Blocks). Each LAB consists of 10 ALMs (Adaptive Logic Modules) along with interconnect and control circuitry. An ALM is the basic building block of Stratix IV FPGA and provides features for efficient logic usage. An ALM consists of a variety of resources consisting of one 8-input ALUT (Adaptive Look Up Table), two dedicated full

adders and two dedicated registers as shown in Figure 2.3. One ALM can implement variety of functions including all 2,3,4,5,6-input functions and certain 7-input functions [56].

Figure 2.4 shows two examples from normal-mode of the ALMs, implementing 3-input, 5-input and 6-input functions. This mode is suited for implementing general logic applications or combinatorial functions.

For implementation of addition circuits, comparators and counters, ALMs can be used in arithmetic-mode; shown in Figure 2.5. The adders can form large carry chains, the 4-input LUTs provide for pre-adder logic. ALM in arithmetic mode supports use of adder's carry output along with combinational logic outputs simultaneously. The carry chains in adders can run to very long lengths, from one ALM to next ALM and in between LABs.

Quartus® II can allocate carry chains up to the length of a vertical column in FPGA fabric. Similarly, registers in ALMs can be cascaded together to form long register chains to implement large shift registers. Figure 2.6 shows example of data chains in a Stratix IV LAB.

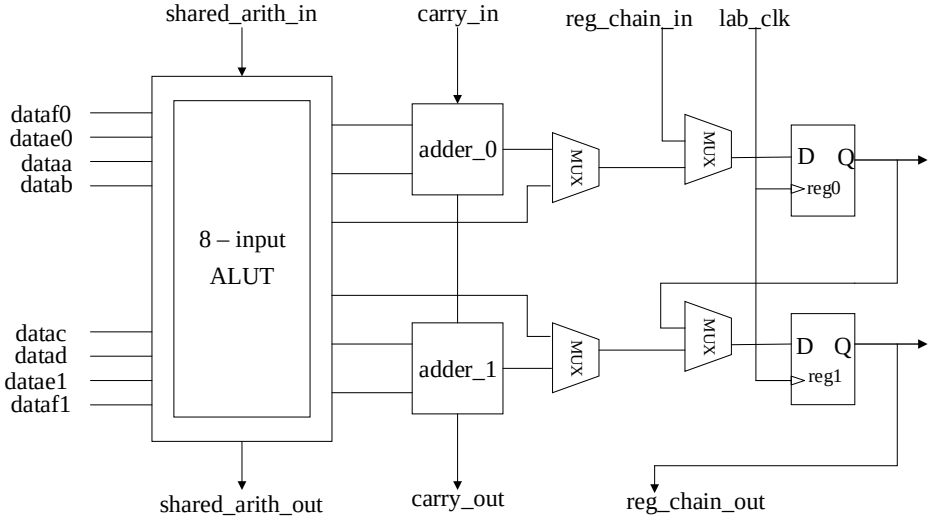

Figure 2.3 : High level diagram of a Stratix IV ALM

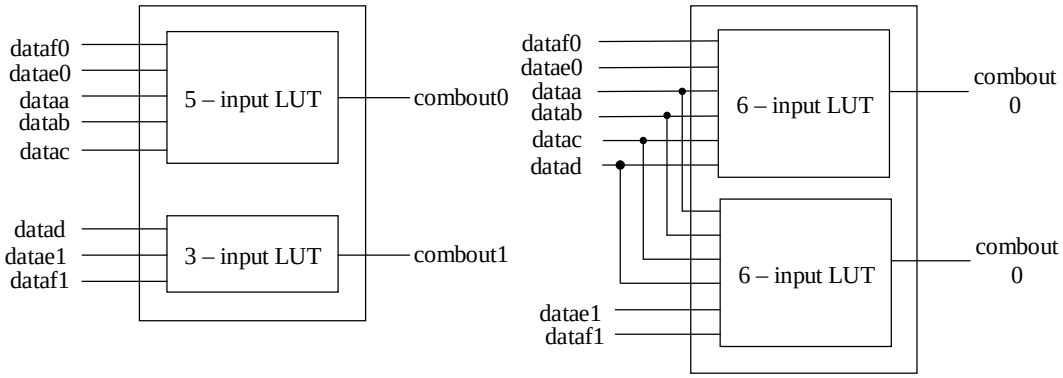

Figure 2.4: Normal mode allows (a) two distinct functions a 5-input and a 3-input; (b) Two 6-input functions with shared inputs to be implemented in a single Stratix IV ALM

.

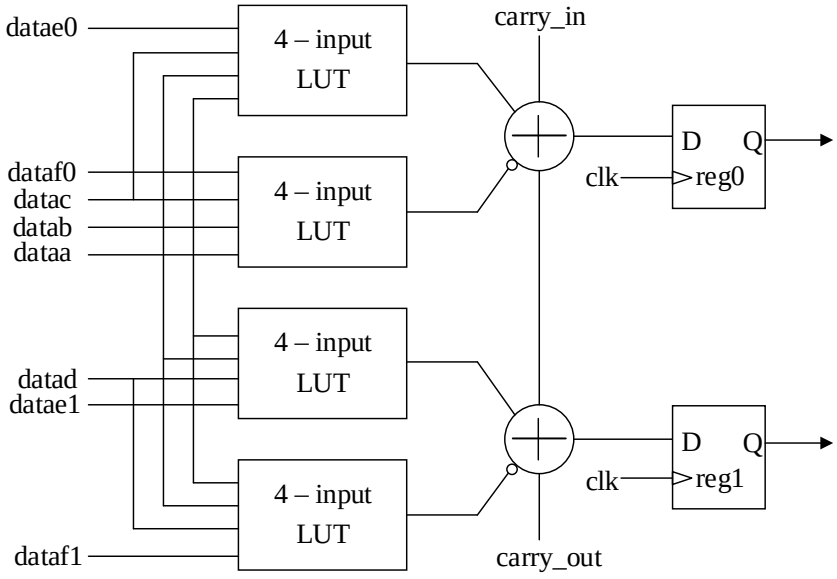

Figure 2.5 : Stratix IV ALM in Arithmetic mode

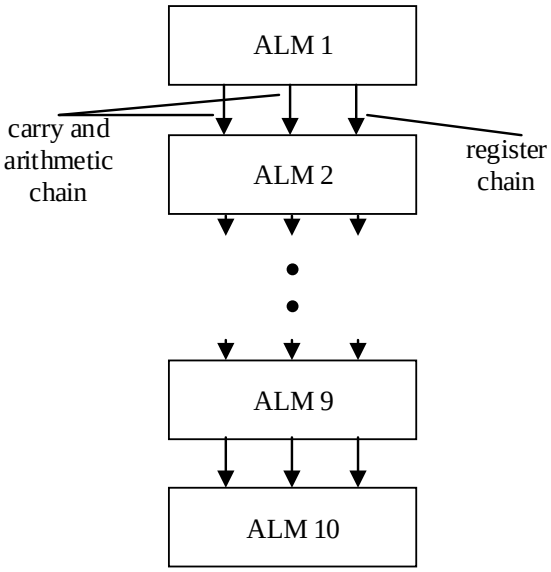

Figure 2.6: Carry, Arithmetic and Register chains in Stratix IV

\

Chapter 3
Digit-Online Decoder

In this Chapter, a brief overview of the architecture of a digit-online LDPC decoder and its operation is discussed. This Chapter mainly builds on the works from [5], [15], [57].

3.1 Digit-Online Arithmetic

An online algorithm processes inputs piece-by-piece without needing a complete set of inputs to begin the calculations. For the processing to be digit-online [58], [59], [60] the operation should be able to produce $i - \Delta$ most-significant digits of a result knowing only i digits out of the s digits of precision of input ($s>i$), with Δ being the initial delay to produce first output. Thus, such a processing can achieve a very high level of pipelining, with a new set of outputs being produced every clock cycle.

The digit-online decoder in [57] employs the MS algorithm with offset correction. The variable nodes in a digit-online decoder perform an addition operation and the check nodes implement selection networks to select the minimum of the inputs. An offset correction operation is performed in the check nodes. The operations in a digit-online decoder are performed in a serial manner, with the most-significant digits (MSDs) processed first. It is not possible to perform MSD-first addition using the conventional notations without receiving all of the bits of the inputs being added, because carry values need to be propagated from the LSB. Thus a processing algorithm using conventional notations is not very advantageous from the point-of-view of serial processing. A pipelined digit-online process requires taking in new set of inputs and generating a new set of outputs every clock cycle.

To achieve highly pipelined continuous processing at the decoding nodes, the work in [5], [15], [57] uses redundant notation to represent numbers and represents them in form of generalized digits rather than bits. A class of number representations known as signed-digit

representation scheme is shown in [57], which allows for most-significant digit (MSD) to least-significant digit (LSD) addition of numbers.

Unlike conventional number representations where an integer radix r (r >1) each digit can only assume out of 0, 1,…., r-1 values, signed-digit representation allows for representations using negative integers as well. For a conventional number representation digits take values from set {0, 1} while for signed-binary the values from set $\{\bar{1}, 0, 1\}$ are allowed. $\bar{1}$ represents -1, and borrows 1 from the digit before it. Thus, the representations of numbers are no longer unique. For example, $11\bar{1} = 4 + 2 + (-1)$ and $101 = 4 + 0 + 1$ represent same number in decimal format. Since the channel LLRs are provided to the decoder in conventional binary notation, they need to be converted in signed-binary digit format in the decoder.

Conversion of conventional sign-magnitude binary representation into signed-binary representation is shown in [57]. For conversion of positive numbers, the MSD of the resultant signed-binary number is set to zero and the remaining digits are left as is; however, now this number is stored with a greater number of bits. For the negative number conversion, the MSD is set to zero, and the remaining digits are stored as negative copies of the corresponding binary bits. The digit-online decoder in [5], [15], [57] converts channel LLRs into signed-binary representation from sign-magnitude representation in the first decoding iteration. The conversion from signed-binary representation to conventional binary representation becomes increasingly expensive to implement with increasing LLR precisions. After the decoding iterations are complete, a thresholding operation is performed to produce hard outputs. Performing this thresholding operation only requires the knowledge about the sign of final LLRs output from the variable nodes, which is relatively inexpensive and easy to implement.

When decoding LDPC codes with the MS algorithm using offset correction, the operations that need to be performed in the decoder are: addition, subtraction, compare-select and sign detection. All of these operations are explained in detail in [5], [15], [57] to work with redundant (signed-binary) notation with most-significant-digit (MSD)-first digit-online processing.

3.2 LDPC Encoder

The VHDL files for the block LDPC encoder are generated using a C++ routine. The WiMAX standard supports frame sizes ranging from 576 bits to 2304 bits and supported code rates are 1/2, 2/3, 3/4, 5/6 with a minimum throughput requirement of 100 Mb/s. For this work, a 576-bit, rate-3/4 WiMAX Encoder is generated. The base matrix (.hbm) and half a- list (.alist) files provide information about the encoder structure to the C++ routine, while the size of the code word is decided through the expansion factor (Z) passed on to the routine. The encoder takes in a 432-bit input, generates and appends 144 check bits to generate a 576-bit frame as shown in Figure 3.1. The check bits of the frame are calculated as per the code constraint equations.

3.3 Digit-Online Decoder

In an iterative LDPC decoder, each decoding iteration consists of messages flowing from the variable nodes to the check nodes and back. Thus, the total pipeline length (p) for one decoding iteration is equal to the sum of the variable node pipeline stages (Δ_v) and the check node pipeline stages (Δ_c) (Figure 3.2). Each pipeline stage stores and processes one digit, thus length of the input message should be less than $\Delta_v + \Delta_c$ digits (total pipeline stages in one iteration), since guard digits need to be added into the message to prevent overflow in the variable nodes. The number of guard digits depends on the maximum variable node degree ($d_{v,max}$) and the way values are being added in variable node [57]; however, an optimum number is given by $\lceil \log 2(dv, \max) \rceil$.

To distinguish the MSD of an LLR from other digits, each processing node in the decoder receives a globally synchronized control input (Figure 3.3). The vector version of this control

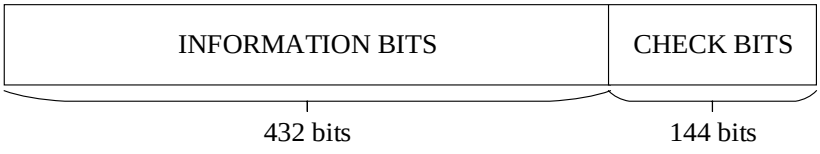

Figure 3.1 : A 576-b encoder frame output from the LDPC Encoder

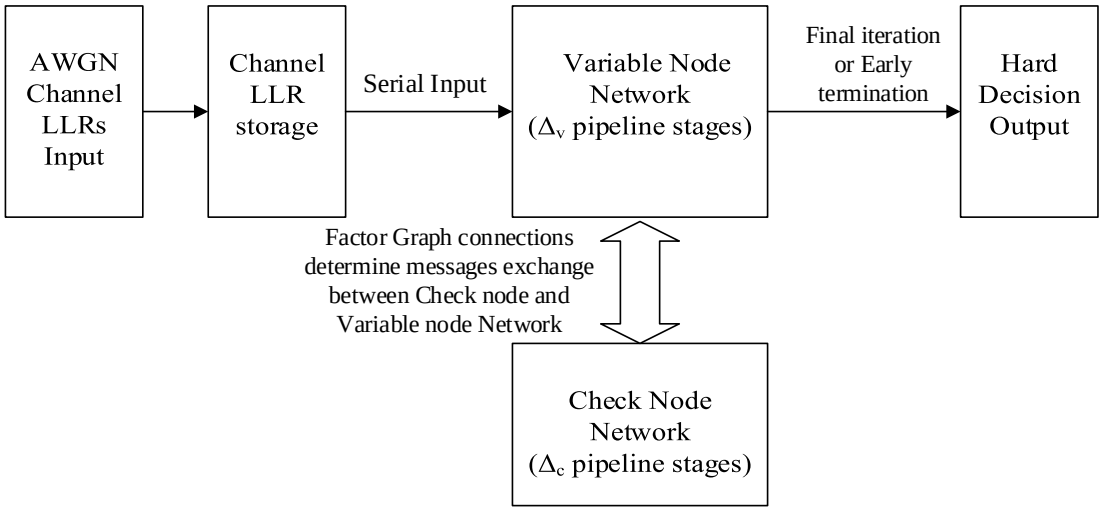

Figure 3.2 : Digit-online decoder system diagram. Channel LLRs are sent serially to the variable nodes. The messages are passed between variable node array and check node array. Each iteration involves message going to check nodes and back to variable nodes for a total pipeline length of $\Delta_v + \Delta_c$.

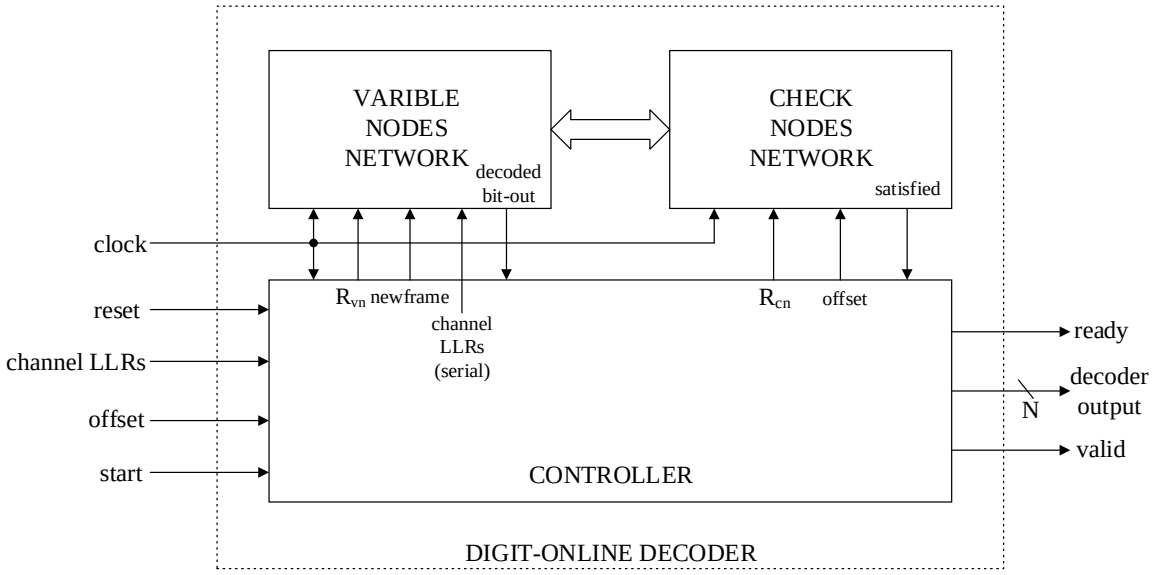

Figure 3.3 : Various components of a digit-online decoder. Main signals are shown as well.

signal is called **R**, with $\mathbf{r}_i$ being equal to 1 if the current i^{th} input digit to a node is an MSD. This signal is required to reset the state machines in check nodes and break carry chains in adders in variable nodes for the new LLR messages [57].

R is expanded to the length of the pipeline, with multiple 1s in the R vector for decoding more than one frames simultaneously. The distance between 1s depends on the precision of LLRs. The processing nodes work on a continuous stream of data and need to know only where the LLR message starts and ends; thus, variable length LLR messages can be processed easily.

3.3.1 Variable Node Structure

Variable nodes perform the addition operation on LLR messages as shown in equation below:

$$\lambda_i = L_{channel} + \sum_{\substack{j=1 \\ i \neq j}}^{\Delta_v} Lj \tag{3.1}$$

The addition trees in the variable nodes have been optimized for minimum Δ_v and minimum required guard digits [57]. Optimization for small number of required guard digits also reduces overhead associated with their generation. An example structure of a degree-4 variable node is represented in Figure 3.4.

Figure 3.5 shows the variable node structure indicating the variable node outputs for various stages of decoding. In the first decoding iteration of a new frame, since the channel values are provided to the variable nodes in conventional sign-magnitude format, they are converted to signed-binary format and are output to the check nodes. A separate circuit is used for the conversion of sign-magnitude to signed-binary values. New channel values are loaded into the local channel memory in the variable nodes for use in future iterations. The local channel memory in the variable nodes is implemented as a circular shift register with output of the shift register connecting to the input when the new_frame signal is 'low'. The local channel

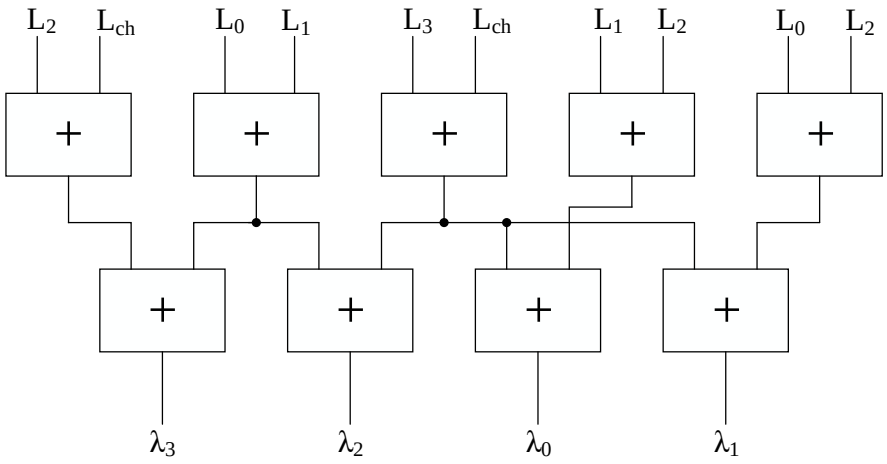

Figure 3.4 : An example degree-4 variable node. The LLRs, L_{ch} (channel LLRs) and L_0, L_1, L_2, L_3 are added digit-serially in the adders represented by boxes with '+' sign

memory has a length equal to the pipeline length of $(\Delta_c + \Delta_v)$ stages. Since new values in the local memory are only loaded in the first decoding iteration of a new frame, the local channel memory provides channel values in a digit-serial manner to the addition circuitry for the remaining decoding iterations of the frame. In the final iteration, new old channel values are shifted out to perform the final addition operation and the new frame channel values are loaded in the local channel memory. The final addition operation consists of adding $\Delta_v + 1$ values (all incoming edge messages and local channel values). After final addition since the next step is hard-decision output, only the sign of the final output LLR (λ_{final}) is of significance. So even if an overflow occurs in final addition, the sign of the result needs to be preserved while the magnitude can be incorrect [57].

3.3.2 Check Node Structure

The check nodes perform the minimum input selection operation through the use of selection networks. To compensate for the performance loss as compared with SP algorithm, an offset correction is performed in check nodes. If decoder constraints are met before the final iteration, check nodes output a 'satisfied' signal to indicate early termination of decoding to the controller. The architecture of a check node is shown in Figure 3.6. The sign of the incoming

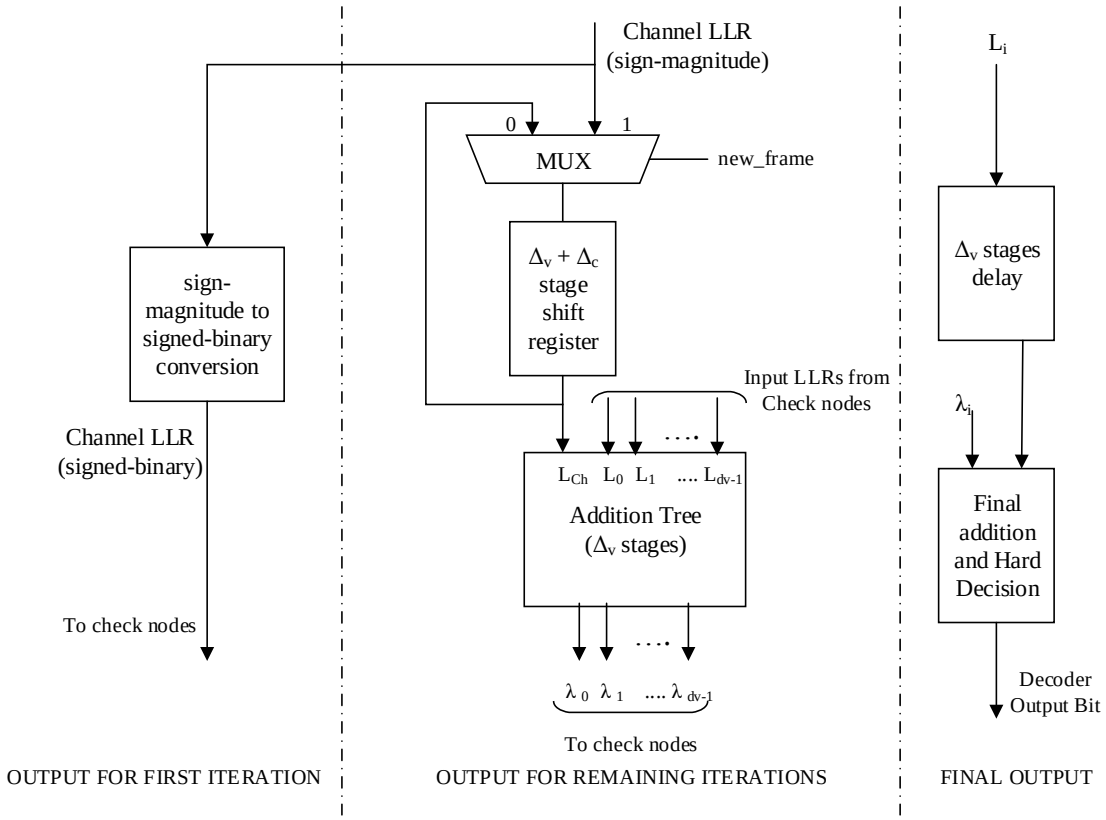

Figure 3.5: Variable node structure. In the first iteration new channel LLRs are received serially and converted to signed-binary format and sent to check nodes. New channel values are also loaded in the shift register for use in later iterations. The process for final output is illustrated on the right hand side.

LLRs in the check nodes is determined and sent to the XOR network tree to check if the code constraints are meeting already and early termination can be performed.

The magnitude of the incoming LLRs undergoes offset correction. The offset value is stored in conventional binary format and is subtracted from the LLR magnitudes using digit-online subtraction circuits [57]. The resultant magnitudes if negative, are replaced with all zeroes otherwise kept as is. Offset correction applied on the inputs of the selection networks requires more subtraction circuits than offset correction applied at selection network outputs; however,

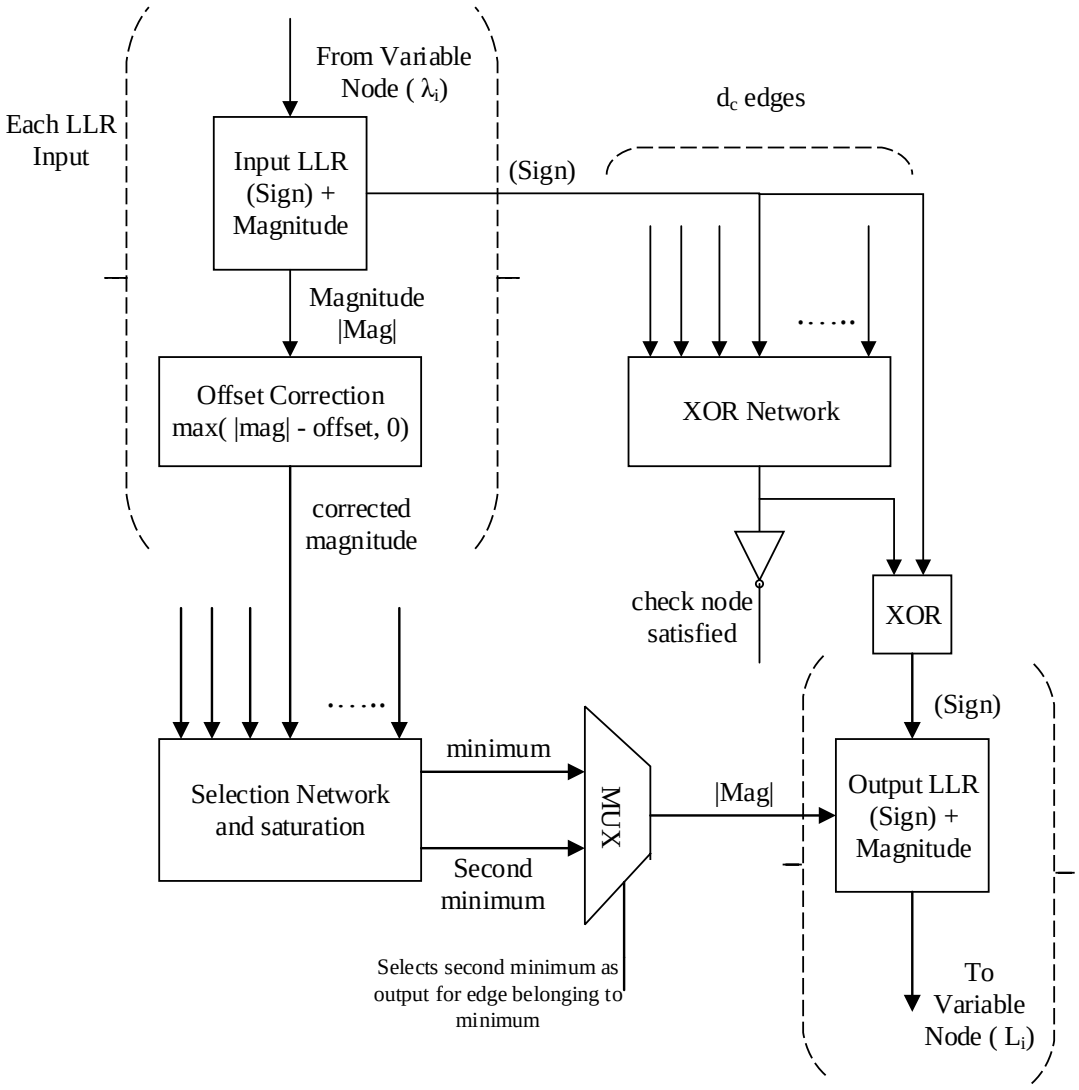

Figure 3.6: Digit-online check node structure. The sign of the incoming serial LLR is sent to an XOR network to identify if check node constraints are satisfied. Magnitude of LLR is sent for offset correction, preceding a selection network to calculate the minimum values. Appropriate magnitude is selected and then combined with sign to be sent to variable node edge.

[57] notes that performing offset correction at selection network outputs lead to incorrect results for digit-online processing. The selection network in the check nodes selects a minimum and a second-minimum magnitude. [61] provides an excellent online tool with various implementation options for generating selection networks up to 32 inputs. The WiMAX 576-bit, rate-3/4 LDPC decoder discussed in this thesis requires selection networks of 14 and 15 inputs; these selection network have been generated using "best" algorithm option in the online tool, giving the minimum number of comparisons required for implementation [57].

When processing MSD-first serially in redundant notation, the selection of minimum values is complicated. The numbers that appear smaller in the beginning may not be so when received completely. While comparing numbers the difference between the numbers is tracked as long as it is big enough and it is safe to make a decision on which number is smaller [57]. For the case of compared numbers being equal, a number is selected as the smaller one arbitrarily. Each comparator comparing two numbers A and B, issues a signal AIsSmaller, which is '1' if the number A is smaller of the two. This signal may change its value as the comparison progresses digit-by-digit, however this should occur only certain times to ensure correct results. The signal AIsSmaller from the individual comparators in the selection tree is used to track the minimum input magnitudes.

The minimum values at the outputs of the check nodes are checked for overflow, and if required they are saturated before sending to the variable nodes. The minimum number is output at all the check node edges except for the edge where this number came from, on which the second-minimum value is sent instead. The sign of the outputs is decided according to what other edges suggest the output sign should be. Thus, the sign of each output LLR is found by XORing the input sign for that edge to the output of the XOR network.

3.3.3 Controller

The controller is responsible for synchronizing the data flow between the variable and the check nodes. For irregular codes, the nodes are of varying degrees and there is a need to ensure that the pipeline path-length for all the node types is same and equal to $\Delta_v + \Delta_c$. If required,

extra registers can be added in the nodes during synthesis to gain desired loop latency. The control signal R, as discussed before is generated by controller to keep track of the MSDs of the messages during serial processing. Controller issues R_{vn} signal for the variable nodes and R_{cn} signal for the check nodes. When the variable nodes are receiving the MSD of a message, R_{vn} signal is asserted '1'; while R_{cn} is '1' when the check nodes are receiving the MSD. R_{vn} and R_{cn} signals can be seen in the timing diagrams of the digit-online decoder as shown in Figure 3.7 and Figure 3.8.

Channel LLRs and offset values are loaded from the channel LLR buffer into the decoder in bit-parallel manner. Controller sends channel LLR values to the variable nodes and the offset value to the check nodes in bit-serial fashion in the beginning of decoding. new_frame signal acts as a selection input for the MUX in the variable node structure shown in Figure 3.5, in the first decoding iteration, new_frame signal is kept '1'. While new_frame is 1, the MUX in the variable node loads the new channel values to the local channel memory. After first iteration, new_frame signal is pulled to 0. This converts the local channel memory in a circular shift register, providing channel LLR values for future iterations.

The controller also monitors the number of decoding iterations performed. The maximum number of iterations allowed (i_{max}) per code word is fixed at the time of design. After performing i_{max} iterations, the decoder signals the variable nodes to finish decoding current code word and perform a hard-decision (thresholding operation). Hard-decision output bits from individual variable nodes are appended together and output as a decoded code word. It is possible that the check node constraints are satisfied before the i_{max} iterations are reached (early termination criterion). Controller detects early termination from the satisfied signal generated at the check nodes. Each check node 'n' generates its satisfied signal CNsatisfied(n). The controller performs an AND operation on CNsatisfied from all the check nodes and if result is a '1' then it signals the next iteration to be final iteration.

3.3.4 Single Frame Decoding

Figure 3.7 shows timing diagrams of the various signals in the decoder when beginning decoding and loading a new frame. Figure 3.8 shows the timing diagram when finishing the decoding of current frame and starting with a new frame respectively in single-frame mode of decoding. The decoder is initialized by pulling reset 'low'. An offset value must be applied before initializing the decoder, since it is latched in with reset going 'low'. In an actual design, the offset is supplied in the HDL code, and its value is available to the decoder pre-initialization. After being initialized, the decoder asserts a ready = '1' signal indicating it is ready to take in channel LLR values. The channel LLRs are made available to the decoder and the decoder is signaled to start decoding by asserting start signal 'high'. The Start signal is

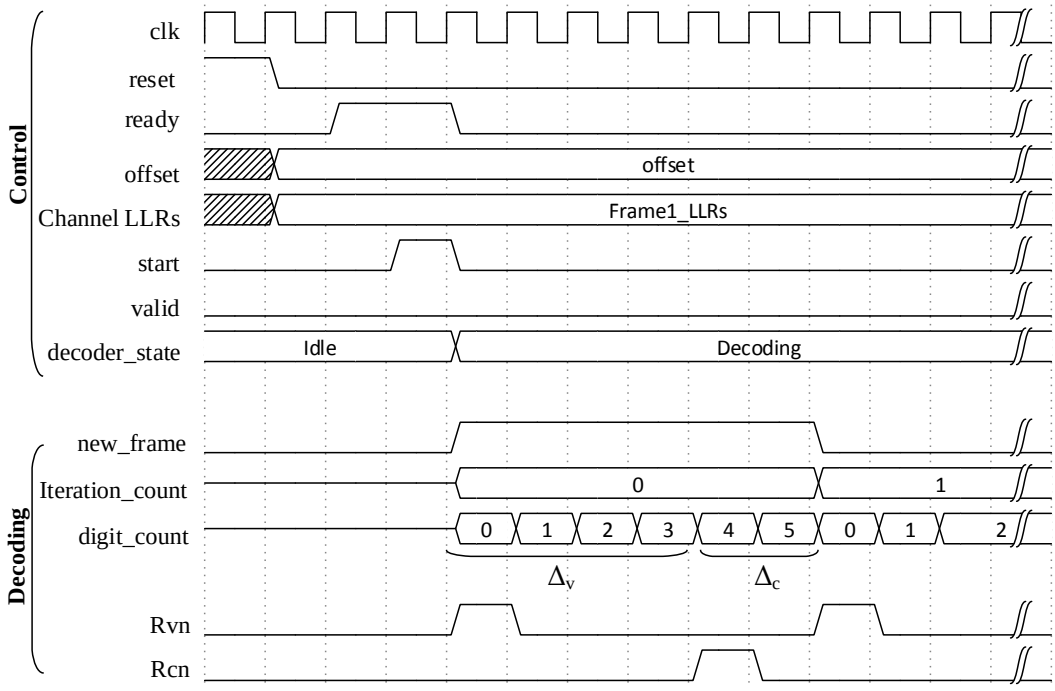

Figure 3.7 : Beginning of decoding a frame in single-frame mode. The decoder goes into Idle state when reset signal goes 0, and asserts 'ready' signal as 1. On receiving a 1 on 'start' pin, the decoder loads in the channel LLRs, pulls in 'ready' signal low and goes into DECODING state beginning with Iteration 0 for decoding the frame. In this example $\Delta_v = 4$ and $\Delta_c = 2$, therefore the length of the pipeline is equal to $\Delta_v + \Delta_c = 6$.

pulled ‘low’ after one clock cycle; the decoder pulls ready signal ‘low’ as well. To keep the iterations going on continuously, the decoder asserts ready signal little before the last iteration finishes as shown in Figure 3.8. Thus, before the final iteration is over, the channel LLR buffer is able to respond with new frame LLRs. Next iteration is first iteration of the new frame, the old frame values are sent into the variable node pipeline for hard-decision decoding. The delay for the final decoded code word is represented in Figure 3.8. This delay is dependent on the pipeline depth in the decoder and is given by $t_{delay,bitout}$ ($=\Delta_{v,sat} + \Delta_v + \Delta_c$) clock cycles. Thus after $t_{delay,bitout}$ clock cycles, the resulting hard-decoded frame is output and the decoder is asserts valid signal ‘high’ for one clock cycle. The iteration count keeps incrementing from 0 to n-1 in a continuous circular fashion and decoder outputs a decoded frame every n iterations.

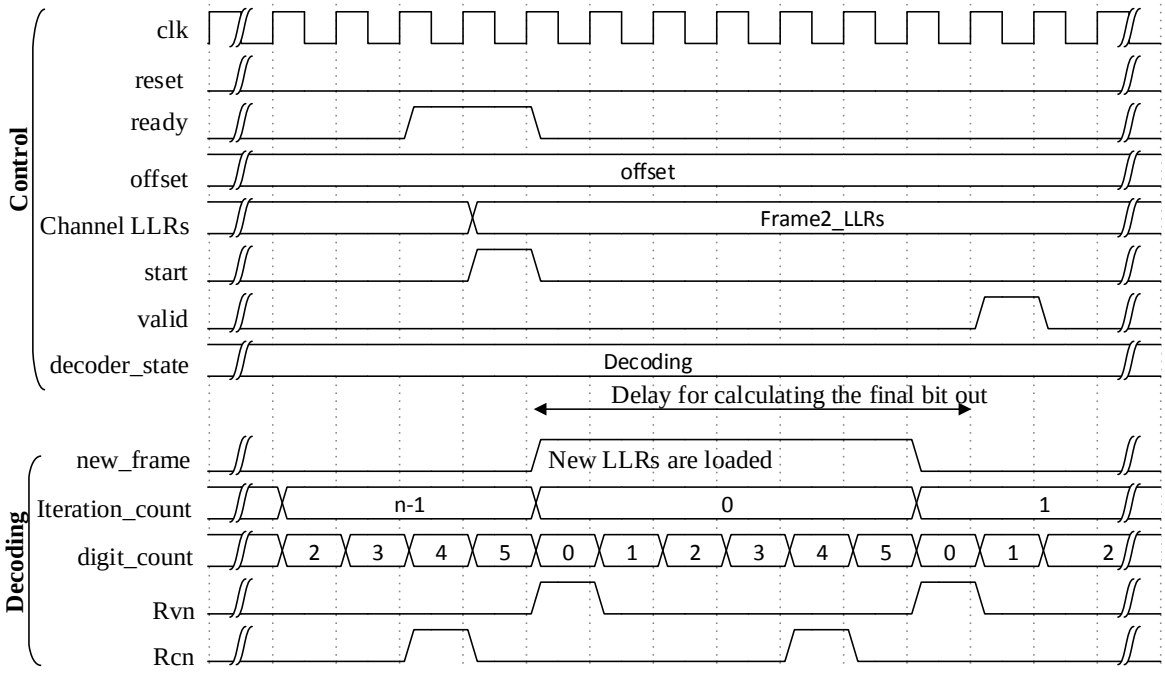

Figure 3.8: Finishing decoding previous frame, hard-decision output and loading next frame for single-frame decoding. The decoder hard-decision output of previous frame at beginning of iteration 1 of new frame. Iteration 0 is used up in hard-decision calculation of previous frame.

3.3.5 Frame-Interlaced Decoding

Frame-interlaced decoding enables the digit-online decoder to decode 2 frames simultaneously. However, the LLR precision of the frames being decoded is halved. The frame-interlaced mode does not require major changes in the decoder structure. Thus, for a slight change in implementation area, it is possible to achieve nearly twice the throughput [5], [15], [57]. Figure 3.9 illustrates the timing diagrams for various signals when beginning decoding and loading new frames. Figure 3.10 shows timing diagram when finishing decoding of previous frames and loading in new frames in frame-interlaced mode.

Pulling reset signal ‘low’ initializes the decoder. Upon initialization, the decoder asserts a ready signal ‘high’ when it is ready to start decoding. The channel LLR buffer responds by providing channel LLRs and asserting start bit ‘high’. The input channel LLRs consist of LLRs

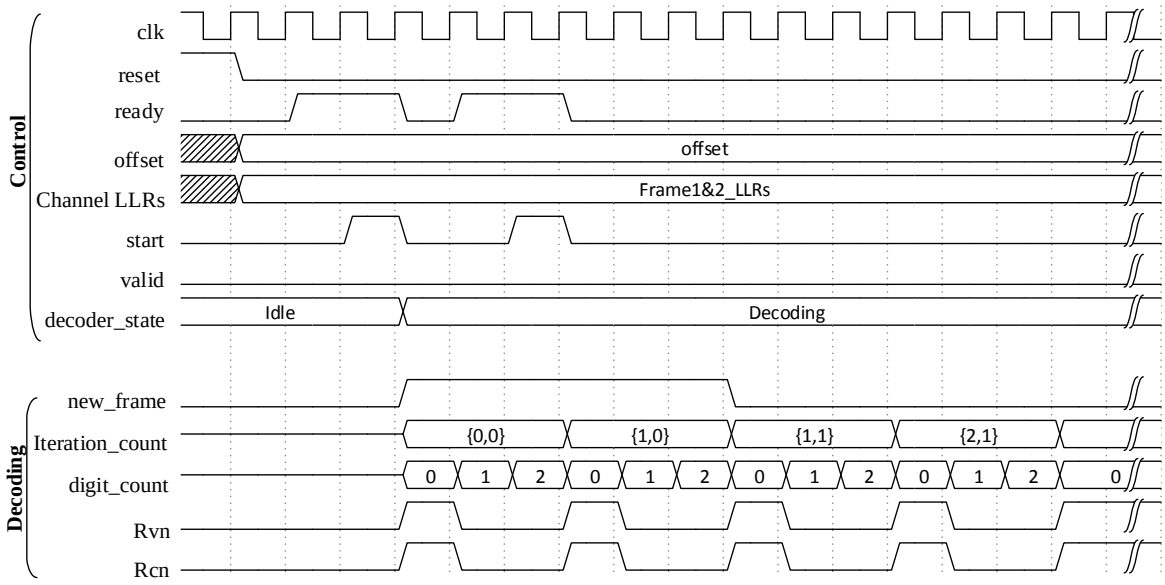

Figure 3.9 : The beginning of frame-interlaced decoding. Each iteration from single frame decoding is split in two iterations. Iteration number {1,0}, represents iteration 1 for frame 1 and iteration 0 for frame 2. There is a frame counter implemented as well to keep track of the frame number. The pipeline length for each frame is halved.

for both frame 1 and frame 2 LLRs in an interlaced manner. The total pipeline length of the decoder is distributed between the two frames. The new_frame signal stays '1' for the duration of iteration {0,0} and {1,0}. The channel LLRs are loaded in variable nodes in this duration. The processing nodes in the digit-online decoders do not require information about which frame they are processing; they only need to know when MSDs for a frame appear. Thus, structurally frame-interlaced decoding is very similar to single-frame decoding. However the control signal differ, as shown in Figure 3.9 and Figure 3.10, R_{vn} and R_{cn} signals are asserted twice in each frame iteration.

When finishing decoding, a ready signal is asserted 'high' before the iteration counter resets to {0,0} to signal channel LLR buffer to send in new channel LLRs. The decoded frames are output after a delay similar to as explained in Single-frame decoding, the decoder output changes at each valid signal as shown in Figure 3.10.

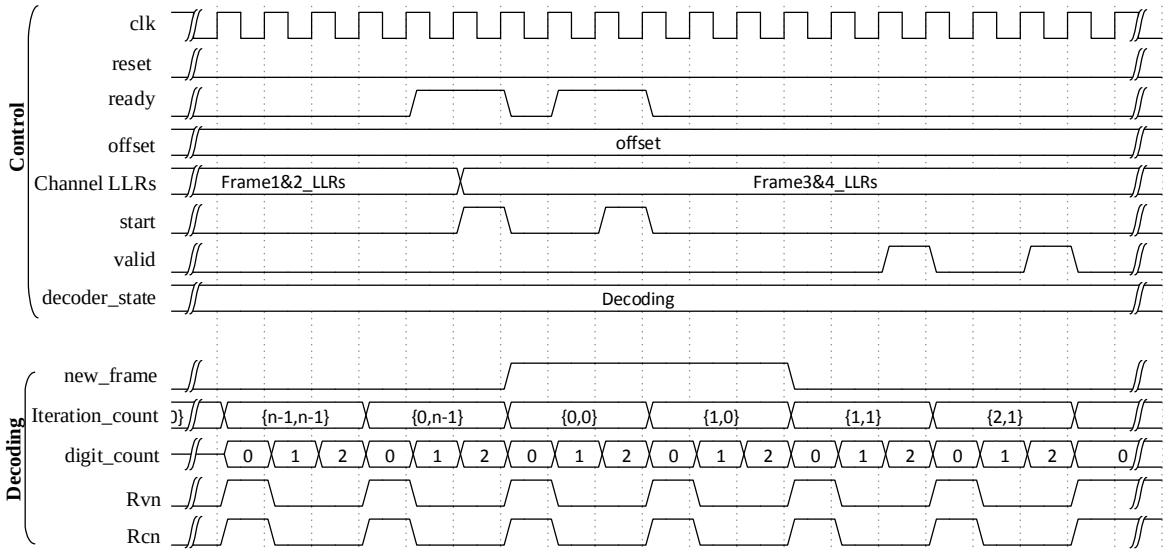

Figure 3.10: Finishing decoding current frames and loading new frames in frame-interlaced decoding. The frames are output after a delay required for hard-decision decoding. Δ_v and Δ_c may not be equal and frames may not be in same decoding iteration.

The channel LLR values that are introduced to the decoder in conventional sign-magnitude format. The AWGN channel module generates channel LLRs and stores them in the channel LLR buffer module, until the decoder is ready to receive new LLRs. LLRs are converted in signed-binary format in the variable nodes only when the decoding begins, doing so reduces memory and fan-in requirements, since signed-binary format requires more number of bits to represent a number.

The hard-decision output of the frame being decoded is performed if either of the two events occur: 1) maximum number of iterations have taken place, or 2) all the check node equations are satisfied (early termination). The maximum number of iterations (i_{max}) is estimated according to the throughput requirements and rate of code convergence. Minimum throughput of an LDPC decoder is inversely proportional to i_{max}.

Chapter 4
Test Unit and Power Measurement Setup

4.1 Introduction

FPGA designs are considered more power and area hungry than their ASIC counterparts. However, FPGAs offer many advantages over ASICs such as fast prototyping abilities, allowing users to change and quickly verify idea in hardware rather than waiting for fabrication of a custom ASIC chip. This saves time and cost per FPGA chip if the quantity of product to be deployed is small. The reconfigurability associated with FPGAs increases product lifetime, and any incremental modifications occurring in future designs can be programmed into the FPGAs, as long as they fit in current FPGA, without the need to redesign the whole board.

4.1.1 FPGA Implementation

The design is implemented on an Altera Stratix IV GX EP4SGX230 FPGA, which comes on Altera DE4 development and education board. Most of the design is written in VHSIC (Very High Speed Integrated-Circuit) Hardware Description Language (VHDL), with the exception of AWGN Channel Module which is written in Verilog HDL. Figure 4.1 shows various functional blocks in the system.

A random data generator outputs pseudo-random bit patterns through a linear-feedback shift register (LFSR). The bit patterns from the random data generator are fed to WiMAX encoder as information bits (432 bits). Encoder generates check bits according to the code constraints, appends them to the information bits and outputs the resulting bit pattern as an encoded frame (576 bits). A shift-out register works in a parallel-in serial-out shift register manner. The shift-out register buffers the encoded frame of 576 bits and shifts out x bits per clock cycle. For x =32 bits means that 576 bits of the frame are shifted out completely in 576/32 = 18 clock

cycles. The number of shift-out register output bits control the flow of channel LLRs to the decoder. Thus, for a 50 MHz clock source, shift-out register with x= 32 bits allows for 2.77 million frames sent to the decoder in one second.

The shift-out register output bits are added with pseudo-random noise samples and scaled as per the assigned signal-to-noise ratio (SNR) in AWGN channel module and quantized to desired precision LLRs. The AWGN generator works on principal of Box-Muller transform, and generates LLRs for each input bit. Channel LLR buffer, stores the output LLRs from the AWGN module for each encoded bit, until all the LLRs for encoded frame are received. When all LLRs for a frame are received in channel LLR buffer, it waits for ready signal from the WiMAX decoder. On receiving '1' on ready signal, the channel LLR buffer provides the decoder with channel LLR values and signals the decoder to start decoding by asserting '1' on start signal. The start signal stays '1' for one clock cycle and goes back to '0'.

The final part of the system is a bit-error-rate (BER) calculation module. The BER module buffers the encoded and the decoded frames when the valid signal from appears from both the modules. Keeping track of the frame numbers is tricky; thus, to ensure proper operation three most recent frames from the encoder are buffered and compared with decoded frame. An XOR operation between the buffered encoder and decoder frames is performed, any bit errors

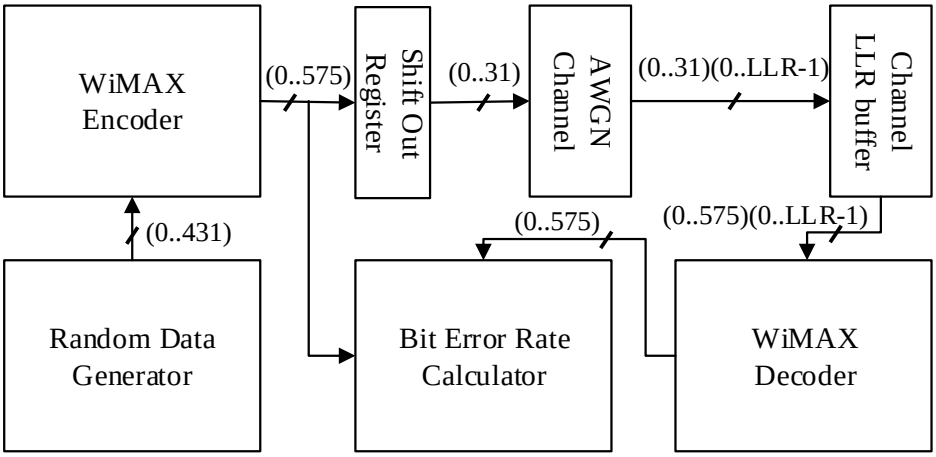

Figure 4.1: A block diagram of the FPGA implementation, showing all the component modules. The arrows indicated the flow of data, numbers on the arrows mention the bus-width

between the two frames results in 1s at that specific bit positions. Counting the number of 1s occurring in the output of XOR operation gives us the bit errors per frame. To calculate the BER, a variable keeps track of the number of frames processed until a set number of errors is reached. The decoder area forms the majority of the design's area on FPGA; an average decoder takes about 85-90% of the design area.

The initial approach in this work was to study WiMAX 1056-bit, rate-1/2 codes on FPGA, since a larger code provides better coding gain. Therefore, a bit-parallel LDPC decoder for code word size 1056 bits and rate-1/2 was synthesized on the FPGA. The size of a bit-parallel LDPC decoder increases with input LLR message precision. The biggest 1056-bit, rate-1/2 bit-parallel LDPC decoder that could fit in the FPGA had LLR message precision of 4 bits. The digit-online LDPC decoder for code word size 1056 bits, rate-1/2 was unable to fit on the FPGA for any LLR precision. A digit-online check node tends to occupy a lot more logic resources as compared with a digit-online variable node (see Appendix A). Going for a higher rate code like rate-3/4 reduces the check nodes in decoder by half in number, also a smaller code with size 576 bits was chosen. The biggest 576-bit, rate-3/4 bit-parallel LDPC decoder that can fit on the FPGA has LLR precision of 6 bits. For the 576-bit, rate-3/4 Digit-Online LDPC Decoder, up to an input LLR precision of 13 bits was successfully synthesized on the FPGA. Higher input precision cases might be able to fit as well, but they were not tested. Thus choosing a smaller size code with size 576 bits provides more test cases over a larger range of LLR precisions.

4.2 FPGA Core Power Measurement

For the measurement of FPGA core power, the setup from [18] is used. A diagram showing the FPGA board power measurement setup is shown in Figure 4.2. The power characterization technique reported in [18] relates the power dissipated in the FPGA core to the actual measured power consumed by the Altera DE4 board. The power measurement setup consists of 0.01-Ω resistors connected to the power supply lines to DE4 board. There are two power supply lines onto the DE4 power supply: one being the 12V and, the other 3.3 V supply. The

power to the FPGA core is supplied through 12V supply, which is converted to 0.9V for FPGA core using three LTM4601 regulators on the Altera DE4 board [62]. Therefore, the power supplied to the board is calculated by measuring the current through the 12V supply lines.

The voltage drop across the 0.01-Ω resister connected in series with the supply lines is measured and current through the resistors is calculated by dividing the measured voltage with 0.01-Ω.

The current across the two 12 V supply lines are added to find out the total input current to the FPGA board.

The input power to the board is found by using Equation (4.1), reproduced from [18] :

$$P_{FPGA\_Board} = (I_{R1} + I_{R2}) * V_{vcc12} \quad (4.1)$$

The FPGA board power when minimum logic is programmed on the FPGA core can be calculated as $P_{BoardMin}$ from (4.1). The design is synthesized on the FPGA chip and the input power to the FPGA board is measured as $P_{BoardwDesign}$.

As mentioned before, the LTM4601 regulators down-convert the input voltage to the board to 0.9V for FPGA chip. Assuming that 100% of input power is converted to the FPGA core

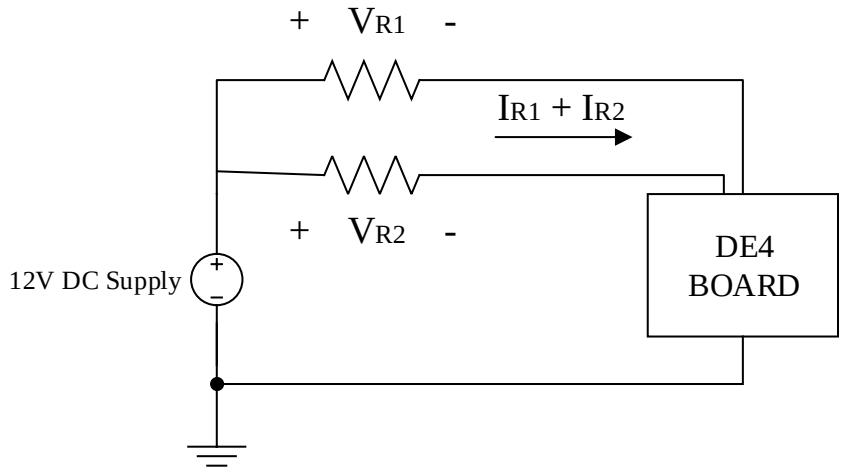

Figure 4.2 : Power measurement setup for measuring power inDE4 board as demonstrated in [18]

power through voltage converters, the incremental FPGA core power for design can be calculated by subtracting $P_{BoardMin}$ from $P_{BoardwDesign.}$ However, in [18] it was noted that LTM4601 does not work at 100% efficiency; rather the conversion efficiency follows a curve, modeled by Equation (4.2), reproduced from [18]:

$$P_{FPGAcoreAct} = -2.039*(P_{FPGAwDesign}) + 16.93*(P_{FPGAwDesign})^{0.5} - 32.8 \quad (4.2)$$

where $P_{FPGAcoreAct}$ is the actual power consumed by the FPGA core. Therefore, using Equation (4.2) the measured board power can be mapped to actual FPGA core power.

4.3 FPGA Power Dissipation Discussion

The ease of programmability and flexibility offered by FPGAs make them less power-efficient than custom ASICs for implementing a given logic circuit. The FPGA configuration circuitry and configuration memory consume more silicon area, which results in longer wire lengths and higher interconnect capacitances. Further additions to capacitive load are caused by programmable routing switches in the FPGA interconnect structures. The power consumed in FPGA core consists of dynamic and static power.

In CMOS circuits the dynamic power is proportional to average capacitance, switching activity and voltage supply according to the relation:

$$P_{avg} = \frac{1}{2} \sum C_i f_i V^2 \quad (4.3)$$

Thus dynamic power is due to the logic transitions in the signals in the capacitive circuit. Interconnect in an FPGA's programmable routing fabric contributes a considerable portion of

the dynamic power. Dynamic power can also be reduced by reducing the switching activity in the non-critical areas of the circuit.

The static power in logic circuits is the due to the leakage caused in the powered-on circuit even when no switching activity occurs. Leakage power is proportional to transistor count and is dependent on transistor width. A reduction in transistor width causes an almost linear reduction in leakage and dynamic power [63]. FPGA leakage power is dissipated in both the used (active mode leakage) and unused parts (sleep mode leakage) of the chip.

In the older process technologies, dynamic power used to be major source of power dissipation in a digital circuit. However, going to sub-50-nm ranges of fabrication, the static or leakage power proves to be an important portion of the power consumption of a chip.

Chapter 5
Results and Discussion

Bit-parallel and digit-online decoders have been synthesized in the FPGA for decoding LDPC code words of size 576 bits and rate-3/4. The maximum iterations have been kept at 10 for both the decoder types. In this chapter, the bit-parallel LDPC decoder implementation is discussed first and later digit-online LDPC decoder implementation is discussed.

5.1 WiMAX 576-bit, Rate-3/4 Bit-Parallel LDPC Decoder

The bit-parallel decoder is a straightforward implementation of a Tanner graph in hardware. The decoding algorithm used is the min-sum algorithm with offset correction. The variable nodes perform the summation operation on the incoming messages; the check nodes after offset correction, select two minimum values out of the corrected LLR messages. The LLR messages are transmitted in parallel over multiple wires across the decoder nodes. The bit-parallel decoder is written in VHDL and its operation is briefly described below:

In the beginning of a new frame, channel values are loaded in all the variable nodes. In the first iteration, the variable nodes simply output the new channel values to the check nodes. The check nodes perform offset correction, select the minimum values and send back to variable nodes where the incoming LLR messages are added with channel LLR values. It is made sure that the variable (LLRsum) used to save the result of the addition are big enough, even if all the LLRs being added have magnitudes equal to maximum magnitude. The variable node output on each edge is calculated by subtracting the incoming LLR on that edge from LLRsum. If the magnitude of resulting LLR is beyond the allowed range, a positive maximum LLR or a negative maximum LLR is output depending on the sign of resulting LLR. The check nodes perform the operation of selecting the minimum of all incoming LLRs on the edges. Signs of all the incoming LLRs are checked to see if the check node constraints are satisfied and early termination criterion applies. If the check node constraints are satisfied, the variable

nodes are signaled to finish decoding and output the decoded code word after a thresholding operation. Offset correction is performed in the check node by subtracting the offset values from the minimum LLR magnitudes selected by the selection network. If the offset value is greater than the magnitude of minimum LLRs then all 0s are output, otherwise the message resulting after offset subtraction is output. The sign of the output LLR is decided based on the sign of the other LLRs. The interleaver specifies the connection between different nodes in the decoder. The interleaver files are generated from C++ routines by parsing the Base matrix (.hbm) and A-list (.alist) files for 576 bit, rate-3/4 WiMAX code.

The controller for the bit-parallel decoder is very similar to the controller described for digit-online decoder but much simpler. When beginning a new frame, the controller loads new channel LLRs into the variable nodes and supplies the offset values to all check nodes. The controller monitors number of iterations and finishes the decoding if the maximum number of iterations is reached or an early termination occurs. When finishing decoding, the decoder appends the hard decoded bits from all the variable nodes and outputs the decoded frame.

5.1.1 ModelSim® Simulation

Before implementation of the decoder on the FPGA, a test bench was written to ensure correct operation in VHDL for use in the ModelSim® software. Few of the issues faced while trying to simulate bit-parallel decoder in ModelSim® were:

5.1.1.1 Simulation of Altera Megafunctions in ModelSim®:

The AWGN module uses Altera megafunction LPM_MULT, which cannot be directly simulated in ModelSim®, since ModelSim® does not have definitions for it. There is a requirement for the inclusion of 220model and altera_mf library files from Altera Quartus® II in the ModelSim® project to fix this.

5.1.1.2 Mixed-language compilation:

The project uses both VHDL and Verilog HDL. Compiling mixed languages in Quartus® II is not an issue. However, the version of ModelSim® in windows did not seem to support it. ModelSim® SE version in Linux supports fast mixed language compile and simulation. A screenshot from the ModelSim® simulation of the bit-parallel decoder for an input LLR precision of 5 bits and 10 iterations is shown in Figure 5.2.

5.1.2 FPGA Synthesis

After ensuring that the control signals are working fine and decoder decodes code words correctly, the design was moved to Altera Quartus® II software. Parallel LDPC decoders have long interconnection wires and since every edge in Tanner graph is actually a multi-bit parallel connection in hardware, this implementation is not very FPGA friendly.

Area requirements in implemented bit-parallel LDPC code decoders increase very rapidly with increase in input LLR message precision as shown in Figure 5.1(a). The FPGA logic utilization

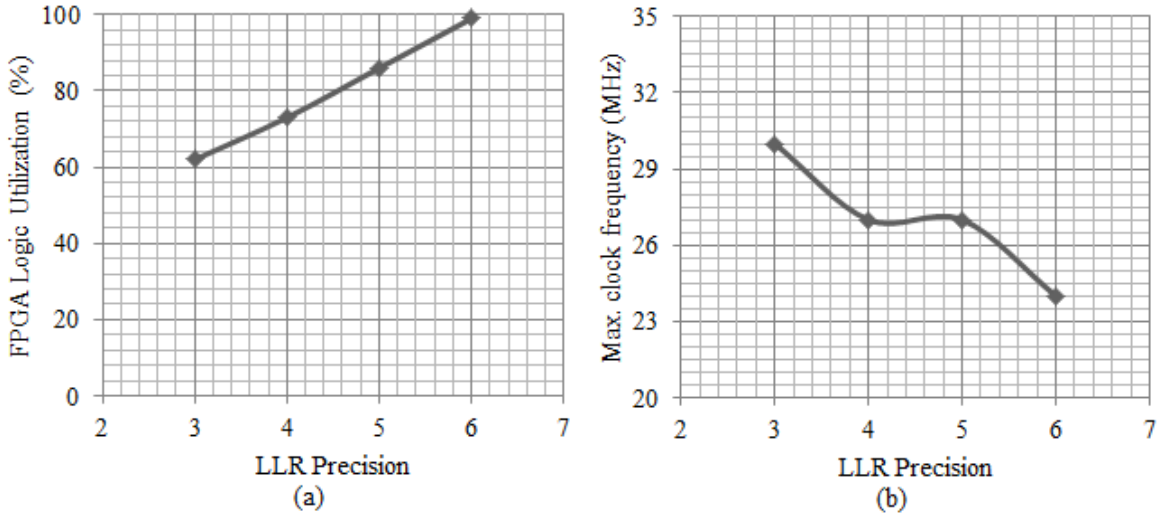

Figure 5.1: For a bit-parallel offset-MS LDPC decoder implementation for a code size of 576 bits, rate-3/4; (a) represents change in FPGA logic utilization versus LLR precision, and (b) represents f_{max} values for changing LLR precision

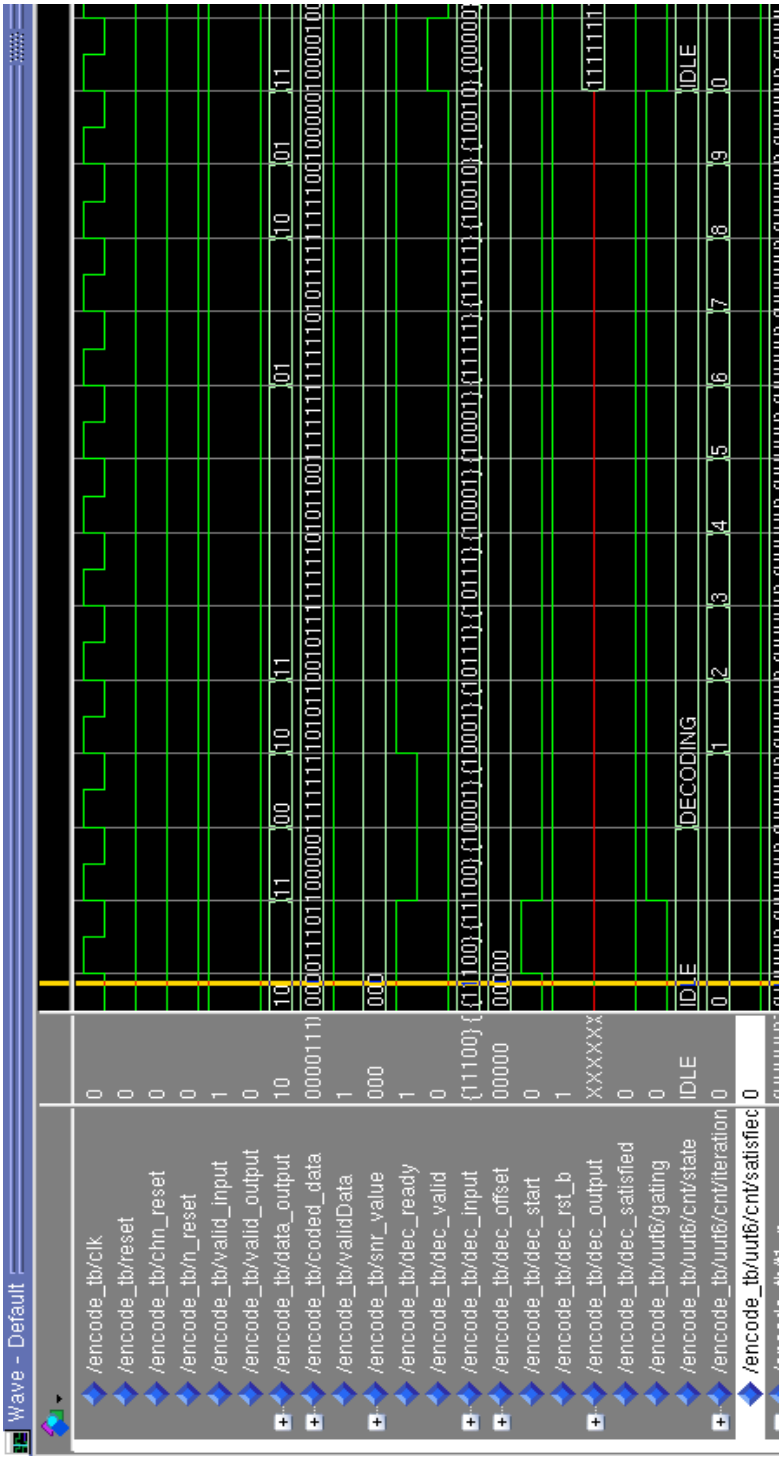

Figure 5.2: ModelSim® simulation for 576 bit, rate-3/4 WiMAX bit-parallel decoder for input LLR precision of 5 bits and maximum iteration number = 10

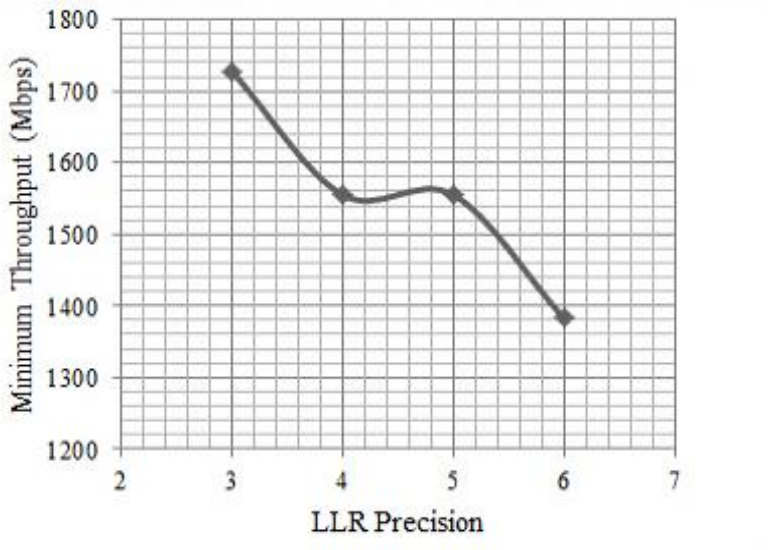

Figure 5.3: Variation in minimum throughput for variations in input LLR decoder implementations

is about 62% for an input LLR precision of 3 bits and increases almost linearly with increasing LLR precision up to 99% logic utilization at 6-bit input LLR precision. Bit-parallel decoders with LLR message precision more than 6 bits were unable to fit in the FPGA.

TimeQuest timing analyzer tool in Quartus® II was used to perform timing analysis as well as to ensure the setup and hold requirements in the circuit were met. The maximum clock frequency (f_{max}) for the design is determined in both the fast- and the slow-corners using TimeQuest timing analyzer. Figure 5.1(b) shows the variation of maximum clock frequency (f_{max}) for the bit-parallel decoder with different LLR input precisions. The reported f_{max} for the designs is the lowest value of f_{max} from all the analyzed timing corners. The f_{max} for the bit-parallel decoder demonstrates a slightly decreasing pattern with increasing input LLR precision. For the smallest decoder with 3-bit input LLR messages f_{max} is about 30 MHz, while for the bit-parallel decoder with input LLR precision 6 bits wide it is about 24 MHz.

Minimum coded throughput versus input LLR precision for a bit-parallel decoder is shown in Figure 5.3. Maximum iterations allowed for decoding a code word are 10; thus, the bit-parallel decoder takes 11 clock cycles for decoding a code word. Therefore, throughput of the

bit-parallel decoder is only affected by f_{max}, therefore, coded throughput versus LLR precision for a bit-parallel decoder forms a curve similar to that of f_{max} versus LLR precision.

The SignalTap II logic analyzer was used to verify signals and debug FPGA hardware. Once the design was successfully synthesized on FPGA, the power analysis using the power measurement setup discussed previously was performed.

5.1.2.1 Use of Gating Signal

In the design implemented on FPGA since the decoder is provided with frames at a controlled rate. It was found on synthesis that the decoder kept consuming power at low SNRs even after performing a hard decision output. The reason for this was that until a new frame was loaded in the decoder, the nodes were trying converge the messages to satisfy the constraints, so other words, the decoding was still going on in the background. A quick fix solution for this was use of a latch-based clock-gating signal. The clock input to the check nodes is ANDed with the output of the latch to form a new clock signal. The clock-gating latch is transparent while the decoder performs decoding iterations before hard decision output; after the hard decision, the controller signals the latch to block the clock signal until a new frame is loaded. This signal can be seen in the ModelSim® screenshot in Figure 5.5. The gating signal turns ‘on’ before the decoder goes into the ‘decoding’ state and stays on until the decoding is done for 10 iterations. As Gating signal goes low, the nodes do not consume any dynamic power.

5.1.2.2 BER Performance

The bit error rate versus SNR for a bit-parallel LDPC decoder with input LLR precision of 3 bits is shown in Figure 5.4. The value of offset used is 001 and maximum of 10 iterations per frame are performed.

5.2 WiMAX 576-bit, Rate-3/4 Digit-Online LDPC Decoder

Like the bit-parallel decoder, the digit-online decoder was first tested in ModelSim® for correct functioning. Afterwards, the decoder was implemented in FPGA and power stats were studied.

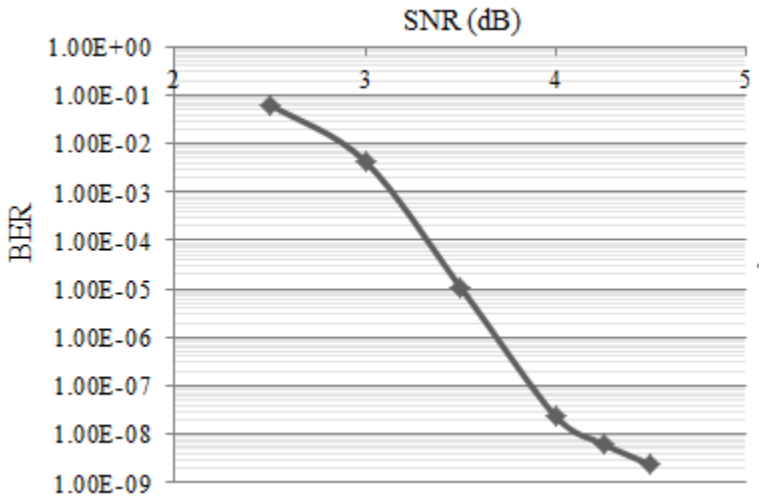

Figure 5.4: BER curve for bit-parallel decoder. The input LLR precision is 3 bits. Offset value is 001 and maximum 10 decoding iterations per frame

5.2.1 ModelSim® Simulation

The ModelSim® simulation screenshot for a 6-bit LLR precision digit-online decoder is shown in Figure 5.5. The control signals in the digit-online decoder are very similar to those of the bit-parallel decoder with a few new signals. The length of an iteration depends on the pipeline length in the decoder and increases with increase in LLR precision. It can be seen in Table 5.1 that each iteration in a 6-bit LLR digit-online decoder is 7 clock cycles long. 6-bit conventional sign-magnitude LLRs are converted into 5 digits in signed-binary format. Each digit occupies one pipeline stage in the digit-online decoder, the pipeline length as shown in Table 5.1 is more than LLR length in digits, the difference is filled with guard digits to prevent overflow in the decoder nodes.

5.2.2 FPGA synthesis

The area requirements in a digit-online decoder change very slowly with changes in LLR precision in contrast with the bit-parallel decoder. A digit-online decoder with 6 bit input LLR

Figure 5.5 : ModelSim® simulation showing main signals in digit-online decoder. Input LLR precision is 6 bits and maximum iterations = 4. The decoder simulation was slightly modified for better illustration

precision uses about 91% of FPGA's logic resources. Increasing the input LLR precision of the digit-online decoder, the decoder size remains fairly constant as shown in the Figure 5.6. However, it was found that the lower LLR precision decoder implementations tend to use more ALUTs (Adaptive Lookup Tables) and fewer DLRs (Dedicated Logic Registers) as compared to high LLR precision decoders implementations (see Appendix A). As input LLR precision increases in digit-online decoder the number of pipeline stages increase as well. Higher pipelining also allows for higher clock frequencies, thus compensates for the more number of clock cycles required for decoding a frame with increasing LLR precisions.

The FPGA logic utilization in case of the digit-online decoders for different LLR message sizes is shown in Figure 5.6. As can be seen, the area occupied by the digit-online decoder does not vary much unlike bit-parallel decoder. The logic utilization on average changes by 5-6% when the synthesis optimization settings are changed from 'Balanced' to 'Area' or 'Speed'. Selecting the fitter effort out of 'Standard fit', 'Fast fit' or 'Auto Fit' has impact not only on the logic utilization, but also on the design compilation time.

Performing a 'Standard Fit' is most time consuming, with average compilation time of about 4-5 hours. A 'Fast Fit' or 'Auto Fit' reduces the average compilation time to about 60-90 minutes. The difference in area utilization between the 'Standard fit' and 'Fast Fit' on average is about 6-7%. The variations in the logic utilizations for various decoder configurations can be attributed to inherent randomness in the CAD algorithms of Quartus® II.

The maximum clock frequency (f_{max}) for the design is determined in both the fast- and the slow-corners using TimeQuest timing analyzer just like the bit-parallel decoder implementation. Figure 5.7 (a) shows the variation of maximum clock frequency (f_{max}) for decoder implementation with different LLR input precisions. The reported f_{max} for the designs is the lowest value of f_{max} in all the analyzed corners. The f_{max} for the digit-online decoder increases with increasing input LLR precision implementations, going from 90 MHz for 6-bit input LLR decoder indicating that f_{max} for FPGA based digit-online decoders starts saturating near 145 MHz. It might be possible to clock the design at higher clock rates by use of appropriate synthesis settings.

Input LLR Precision (bits)	Input LLR Precision (digits)	Δ_v	Δ_c	Pipeline Length
6	4	4	3	7
8	7	5	4	9
9	8	6	4	10
11	10	7	5	12
13	12	7	7	14
11 (frame interlaced)	4 (each frame)	7	5	12
13 (frame interlaced)	5 (each frame)	7	7	14

Table 5.1: Table shows the digit-online decoder pipeline length for a different input LLR precisions in bits. The LLR values are converted from sign-magnitude format to signed-binary format in the decoder and stored in form of digits. Δv and Δc is the length of the variable node pipeline and check node pipeline repectively.

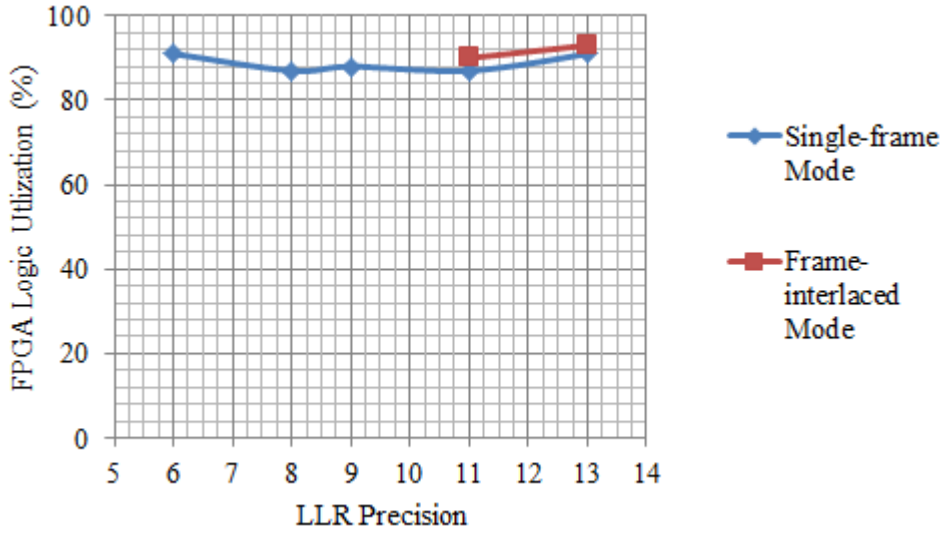

Figure 5.6 : Graph showing variation in FPGA logic utilization for variation input LLR precision configurations of the digit-online decoder.

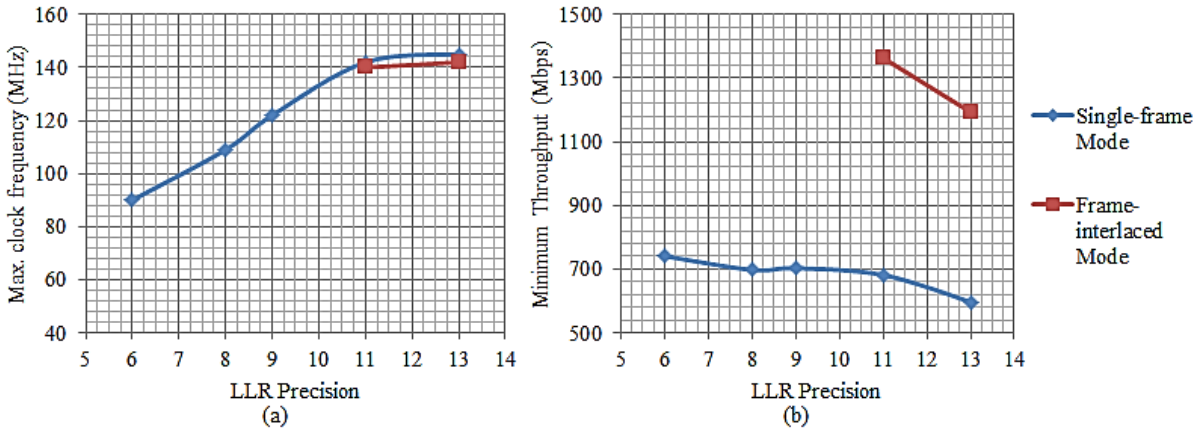

Figure 5.7: Plot showing variation of (a) f_{max} and, (b) minimum throughput vs LLR Precision for digit-online decoder.

Minimum coded throughput versus input LLR precision for the digit-online decoder is shown in Figure 5.7(b). A code word is decoded in a maximum of 10 iterations. The number of clock cycles per iteration depends on the LLR precision as discussed before. Therefore, minimum coded throughput can be calculated from f_{max} and number of clock cycles required for decoding a code word for a digit-online decoder implementation and multiplying by code word size (576 bits).

The SignalTap II logic analyzer was used to verify signals and debug FPGA hardware. Once the design was successfully synthesized on FPGA, the power analysis using the power measurement setup discussed previously was performed.

5.2.3 BER Performance

A BER curve for the digit-online decoder was calculated using the BER calculation module synthesized in the FPGA. The BER calculation module counts a minimum of 250 errors at each SNR value. The input code word to the digit-online decoder is of size 576 bits and rate-3/4, input LLR precision is 6 bits and the decoder performs a maximum of 10 decoding iterations.

5.2.3.1 BER curve

An offset value of 000011 was found to provide the best BER performance. The observed bit error rate (BER) performance versus SNR for the 6-bit digit-online decoder synthesized on the FPGA is shown in Figure 5.8.

5.2.3.2 Effect of Offset

The BER of the digit-online decoder for different offset values was calculated as shown in Figure 5.9. As can be seen, the choice of the offset affects the BER performance a lot, especially if a large value of the offset is chosen. It was found that an offset value of 000011 gives the best error performance for a 6-bit LLR digit-online decoder configuration. BER performance for an offset value of 000000 basically corresponds to MS decoding without offset correction. It can be seen from Figure 5.9, for an offset value of 000000 the BER is 5.6×10^{-9} while with an appropriate choice of offset value the BER can go as low as 8.77×10^{-10} for same SNR value of 3.75 dB.

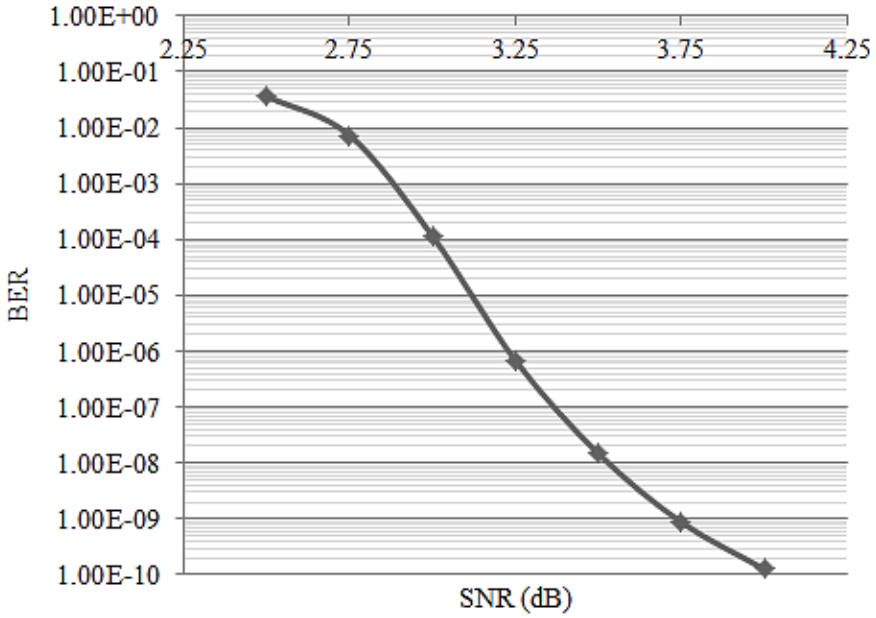

Figure 5.8: BER vs SNR curve for a digit-online decoder. The input LLR precision is 6 bits. Offset value is 000011.

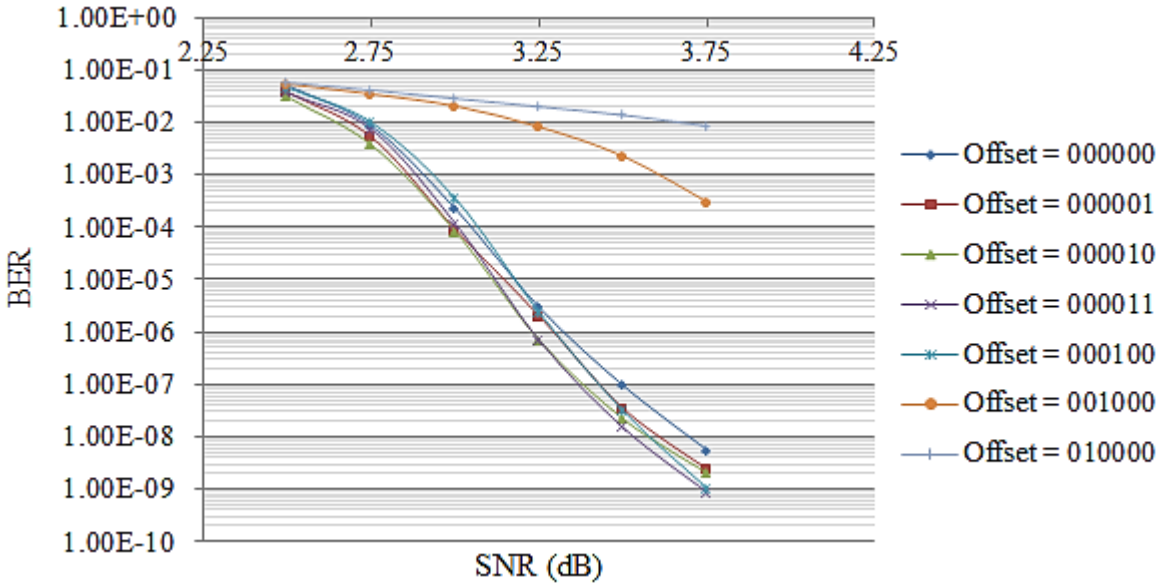

Figure 5.9: Effect of offset on BER vs SNR curve for a synthesized 6-bit input LLR digit-online decoder for a code word size of 576bits, rate -3/4. The decoder performs maximum of 10 iterations.

5.3 Power Characterization Experiment

For the power characterization experiment, a constant throughput (coded) value of 450 Mb/s is chosen for all the LDPC decoder implementations to get an idea of how expensive a particular implementation choice is in terms of power consumption to achieve a specific throughput. For the bit-parallel decoder, the code words are introduced at a controlled rate from the shift-out register in Figure 4.1. The shift-out register shifts-out 18 of encoded frame bits per clock cycle; thus, an encoded frame is shifted out completely every 32 clock cycles, and a new encoded frame is loaded in the shift-out register. For an input clock frequency of 25 MHz this corresponds to 781, 250 frames/second, for a frame size of 576 bits corresponds to 450 Mb/s coded throughput. The bit-parallel decoder decodes a code word in 11 clock cycles and asserts a ready signal (Chapter 4); however, a new code word is not introduced until the next 21 clock cycles and for that duration the decoder nodes are clock-gated to halt any unnecessary switching activity in decoder.

The implementation of the 450 Mb/s throughput is slightly different in case of digit-online decoder. Since, the digit-online decoder requires more clock cycles for decoding a code word in comparison with the bit-parallel decoder; it is fed with new code word LLRs as soon as it is ready to accept new code words. Since the number of clock cycles required for decoding a code word in a digit-online decoder increases with increase in LLR message precision because of increased pipeline stages; higher LLR precision digit-online decoder configurations are clocked at increasingly higher frequencies as shown in Table 5.2 to keep a throughput of 450 Mb/s. One of the reasons for choosing a coded throughput value of 450 Mb/s for power characterization is because it is easily achievable by all the decoder configurations. The bit-parallel decoder implementation requires the number of bits out from the shift-out register to be a divisor of 576 bits for a simple implementation. The chosen number of bits out for 450 Mb/s is 18; the nearest choices are 16 or 24, which correspond to a throughput of 400 Mb/s or 675 Mb/s respectively. The latter throughput cannot be achieved by 13-bit LLR configuration in digit-online decoder, while the former is smaller than the current selection; it is better to go with as high as throughput possible for better power characterization.

Digit-online decoder LLR precision	Clock frequency (MHz)
6-bit	55
8-bit	70
9-bit	78 (*80)
11-bit	94 (*95)
13-bit	110
11-bit (frame-interlaced mode)	47 (*45)
13-bit (frame-interlaced mode)	55

Table 5.2: Table showing operating clock frequencies for a throughput of 450 Mb/s for the power characterization experiment. The mode of operation is single-frame mode, unless mentioned. All decoder configurations perform 10 decoding iterations.

5.3.1 FPGA Core Power Consumption

The power input to the DE4 board is measured and mapped to the FPGA core power using Equation (4.2). Figure 5.10 shows a plot of the FPGA core power plot versus change in SNR values of the input LLRs for the bit-parallel and digit-online LDPC decoder implemented on FPGA. The LDPC decoder power tends to decrease with increase in SNR of input LLR messages because of reduced switching activity at the decoder input and in the decoding iterations.

From Figure 5.10, it can be observed that the FPGA core power increases with increase in the LLR message precision. For the bit-parallel LDPC decoder implementation, this is because the decoder size increases with increase in input message precision; thus the dynamic as well as static power increases. For a digit-online decoder, area of decoder implementation does not vary much with different input message precisions, therefore, the static power remains relatively constant. But for this experiment, higher precision digit-online decoder configurations are clocked at higher frequencies to make up for longer iterations, thus resulting in increasing FPGA core power due to increased switching activity with increasing input LLR precision. A few of the cases from Figure 5.10 stand out and require further explanation.

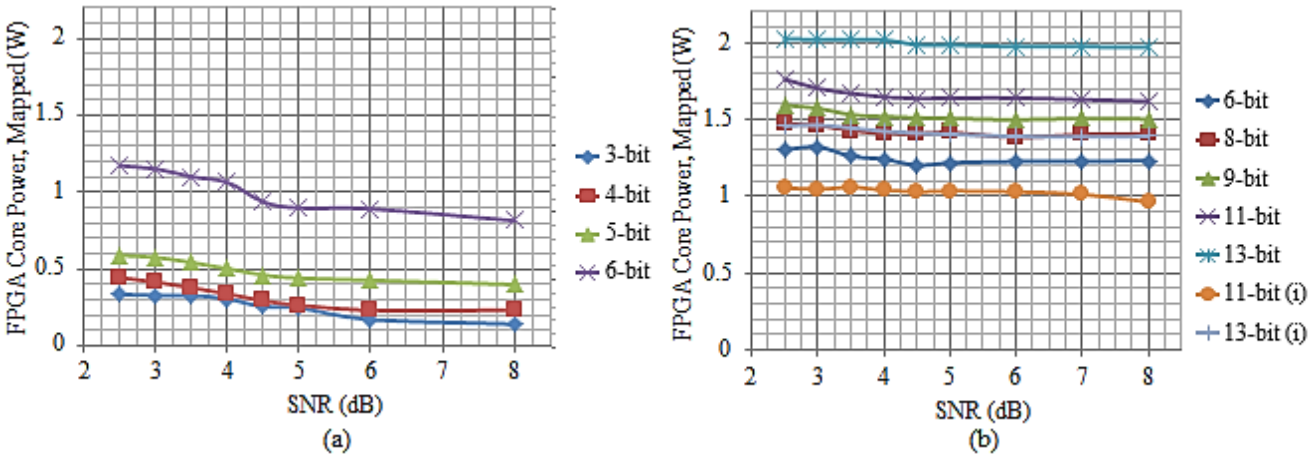

Figure 5.10: FPGA core power (mapped)(W) vs SNR plot of different LLR precisions for (a) bit-parallel decoder and, (b) digit-online decoder.

5.3.1.1 5-bit and 6-bit LLR Input Bit-Parallel Decoder

The 6-bit LLR decoder configuration in Figure 5.10(a) consumes about twice as much power as compared with 5-bit LLR decoder configuration. While the logic utilization between these two configurations does not change that much at about 86% for 5-bit configuration and 99% for 6-bit configuration, the 6-bit LLR decoder configuration has very high interconnect usage (61% average) as compared with 38% average for 5-bit configuration. High logic utilization and interconnect usage not only adds additional dynamic power but also adds considerable static power dissipation to the FPGA core power. A plot showing the average interconnect usage in the decoder configurations for different LLR message precisions is shown in Figure 5.11. Two cases in digit-online decoders require further discussion.

5.3.1.2 Single Frame Mode 11-bit and 13-bit LLR Digit-Online Decoder Configuration

It can be seen in Figure 5.10(b), there is a larger gap in core power dissipations in between the 11-bit and 13-bit configurations of the digit-online decoder. The logic utilization in the case of 13-bit LLR digit-online decoder is increased by 4%. Also from Figure 5.11, it can be seen that

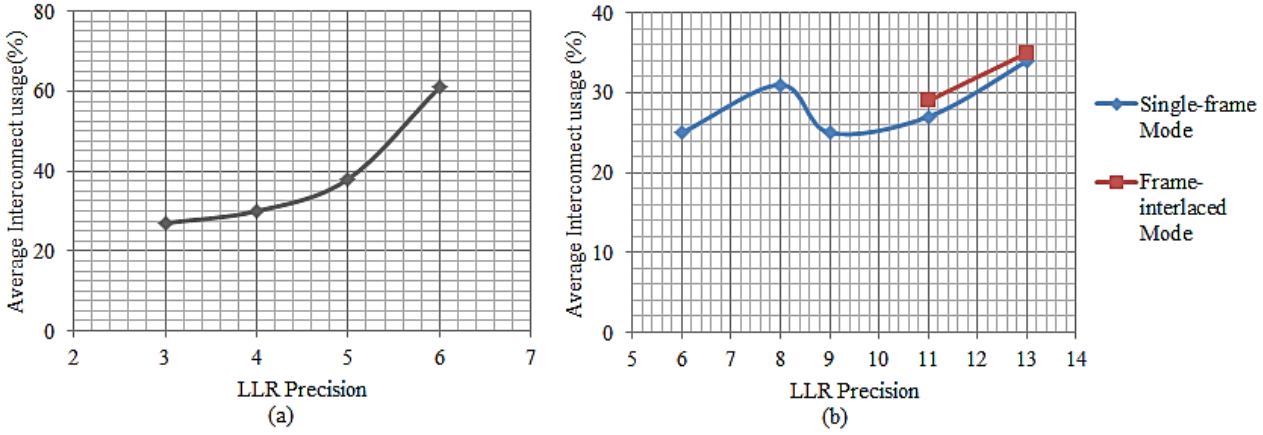

Figure 5.11: Average interconnect usage variation with changing input LLR precision for (a) bit-parallel decoders and, (b) digit-online decoders synthesized on the FPGA.

the interconnect usage in case of a 13-bit LLR digit-online decoder configuration the average interconnect usage is about 34% as compared with 27% in the case of 11-bit LLR digit-online decoder. In the case of a serial message passing decoder like digit-online decoder, the power dissipated in the interconnect forms an important part of the decoder power consumption. Thus, increase in logic utilization and interconnect usage results in increased power consumption in case of 13-bit LLR input digit-online decoder.

5.3.1.3 Single-Frame Mode 6-bit, 8-bit LLR and Frame-Interlaced Mode 13-bit LLR Digit-Online Decoder Configuration

The input clock frequency for 6-bit LLR and 13-bit (interlaced) LLR digit-online decoder configurations is 55 MHz. Thus, it was expected for these two configurations to have nearly equal power consumptions. Instead, the power consumption of 13-bit (interlaced) LLR digit-online decoder configuration is nearly equal to 8-bit LLR digit-online decoder.

The 13-bit (interlaced mode) LLR digit-online decoder configuration utilizes only about 2% more FPGA logic than 6-bit decoder configuration but ends up consuming about 10% more routing resources and results in higher power consumption. The 8-bit LLR digit-online decoder configuration takes up 6% less logic and 4% less routing resources on FPGA; thus, even though it is clocked at higher frequency, it results in nearly equal power consumption with 13-bit (interlaced) LLR digit-online decoder configuration. Another factor for increased core power in the case of 13-bit (interlaced) LLR configuration compared with 6-bit configuration is increased static power in the synthesized LBAs in Stratic IV FPGA. Quartus® II software evaluates slacks in the different parts of the circuit and assigns individual LBAs in either high-performance or low-power modes [64]. In the chip planner tool in Quartus® II software the high-performance logic blocks are shown as yellow and low-power blocks are shown in blue color. Figure 5.12 shows a screenshot from the Chip Planner tool in Quartus® II for both configurations. The transistors in the blue tiles have reduced back-bias voltage and hence are difficult to turn-on, the yellow tiles on the other hand are easier to turn-on. Thus, yellow tiles in the FPGA core have higher leakage currents and more static power dissipation.

5.3.2 Energy-Per-Coded-Bit (E/b)

Energy per coded bit (E/b) provides an important metric to compare energy expense requirements for an LDPC decoder. E/b can be calculated as:

$$\text{Energy per coded bit (E/b)} = \frac{P_{FPGACore}}{Coded\ throughput} \tag{5.1}$$

Energy per bit values are calculated from the FPGA core power values reported in Figure 5.10, for a coded throughput value of 450 Mb/s and are shown in Figure 5.13. Average energy-per-coded-bit values for the bit-parallel and digit-online decoder are discussed in chapter 6.

5.3.3 Energy Per Iteration

For the purpose of finding out the power expenses per iteration in the digit-online decoder, two 6-bit LLR decoder configurations with maximum 4 and 6 iterations were synthesized in the FPGA.

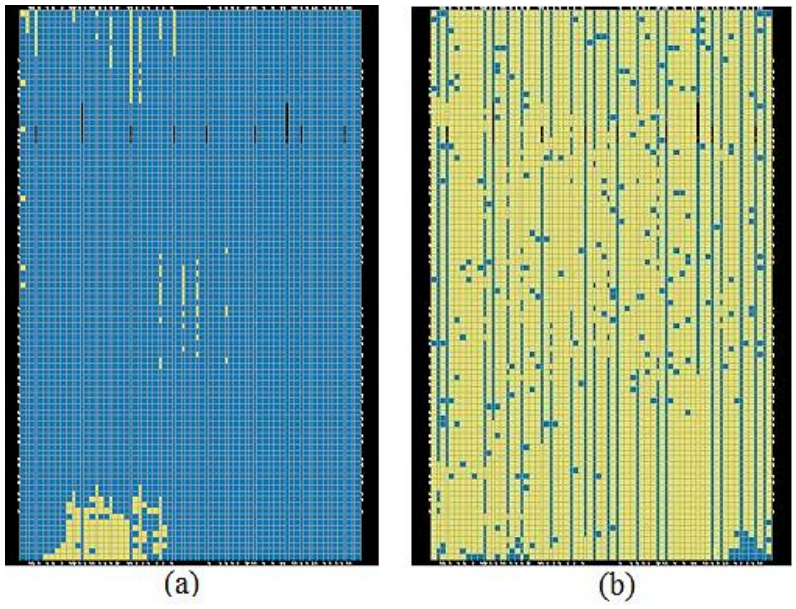

Figure 5.12: FPGA synthesis of Digit-Online decoder design showing low-power (blue) and high-performance (yellow) tiles for (a) 6-bit LLR configuration and, (b) 13-bit LLR configuration. Low-power tiles in Stratix IV FPGA have reduced back-bias transistor voltage, which reduces sub-threshold leakage power.

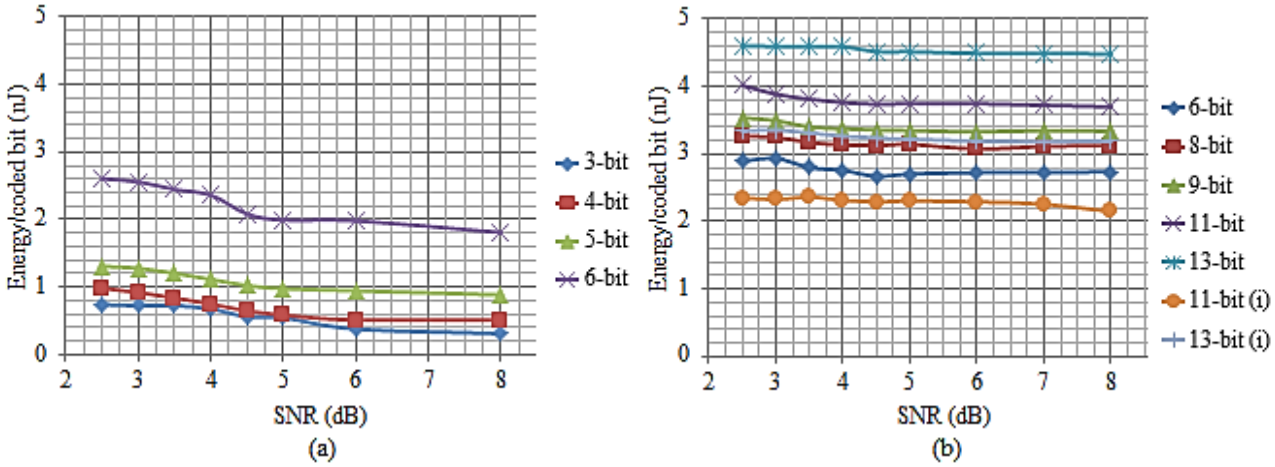

Figure 5.13: Energy-per-coded-bit values for different (a) bit-parallel and, (b) digit-online decoder configurations plotted vs SNR(dB).

The power dissipated in the FPGA core consists of both dynamic and static components. The FPGA resource utilizations in 4 and 6 maximum iteration cases are almost equal; thus, the static power dissipation in both decoders can be considered to be almost equal as well. The dynamic power consumed in the digit-online decoder can be broken down into two main components: 1) power consumed in loading new frame LLRs, 2) power consumed in performing decoding iterations. The power consumed in each decoding iteration will likely be different, since LLRs tend to converge as decoding progresses and switching activity reduces. However, since the LLRs are processed in redundant representation, for calculations here, the power consumption in iterations is assumed to be equally distributed.

For the digit-online decoder with maximum 4 iterations, let's assume 'x' is the power consumed due to loading of new frames and 'y' is the power consumed in the decoding iterations performed in 1 second. Therefore, the contribution of the x and y to the power, $P_{core,\ 4\ iterations}$ (C1) consumed in the digit-online decoder can be written in form of Equation (5.2) as:

$$1*x + y = C1 \tag{5.2}$$

where C1 is the mapped FPGA core power for 6-bit LLR digit-online decoder with maximum 4 iterations.

The digit-online decoder with maximum 6 iterations will process less number of frames in same time as compared with the digit-online decoder with maximum 4 iterations. Thus, the power consumed in loading a new frame x, is multiplied by a factor of 0.67. Since the iterations in the digit-online decoder simply keep incrementing in circular fashion, the number of iterations taking place in both the decoders is still the same. Thus, power component due to y remains unchanged. Therefore, the consumed FPGA core power, $P_{core,\ 6\ iterations}$ (C2), can be given by the expression:

$$0.67*x + y = C2 \tag{5.3}$$

Equations (5.2) and (5.3) can be solved for x and y to find out the power consumed in loading a new frame in the decoder and power consumed in performing decoding iterations in one second.

To verify the results, the values of x and y are used to estimate power consumption for 6-bit digit-online decoder with a maximum 10 iterations using the expression:

$$\text{Estimated FPGA core power, } P_{core,\ 10\ iterations} = 0.4*x + y \tag{5.4}$$

Calculations are performed at power data from different SNRs and shown in

Table 5.3. For a digit-online decoder with input LLR precision 6 bits, each decoding iteration is 7 clock cycles long. Therefore, the number of iterations per second for an input clock frequency of 55 MHz can be calculated as 7.86×10^6. The average value of y can be used to estimate the average energy used per iteration, which comes out to be 0.17 μJ.

SNR	C1-C2	x = Power (frame-loading)(W) = (C1-C2) *3	y = Power (iterations)(W)	Estimated power for 10 iterations (W)	Measured power for 10 iterations (W)	Difference in Est. and Meas. Pow.
2.5	0.042737	0.128211074	1.253455	1.30474	1.395665	0.090925
3	0.065233	0.195699101	1.227595	1.305875	1.414139	0.108264
3.5	0.046186	0.138556773	1.289295	1.344718	1.353299	0.008581
4.0	0.040775	0.122326417	1.332588	1.381518	1.334112	-0.04741
4.5	0.031903	0.095708588	1.345735	1.384018	1.295069	-0.08895
5	0.040413	0.121238052	1.347027	1.395522	1.309815	-0.08571
6	0.022518	0.067555062	1.387359	1.414381	1.319575	-0.09481
7	0.040051	0.120154344	1.361341	1.409402	1.319575	-0.08983
8	0.00445	0.013350292	1.450478	1.455818	1.324435	-0.13138
Average :		**0.111422189**	**1.332764**	1.377332	1.340631	-0.0367

Table 5.3 : Table shows calculation steps for estimating power consumption for a 6-bit LLR digit-online decoder with 10 decoding iterations and the difference between the actual measured and estimated values. The average power consumed in loading a frame and performing decoding iterations is shown in bold.

Chapter 6
Summary and Future Work

6.1 Thesis Summary

In this thesis, FPGA implementations of bit-parallel and digit-online LDPC decoders have been presented. The chosen code size and code rate is among the ones supported by WiMAX (IEEE 802.16) standard. The reported results include FPGA synthesis specific results like the effect of changing input log likelihood ratio (LLR) precision on the decoder size, the maximum clock frequency and decoder power consumption. The effects on decoder power consumption with changing signal-to-noise ratio of the input LLRs for different configurations of both the decoder types have been studied and reported.

Bit-error-rate (BER) for a 6-bit precision LLR input digit-online decoder synthesized on the FPGA is reported as 1.2×10^{-10} at 4.0 dB. The effect of offset value on the BER performance of the 6-bit LLR input digit-online decoder is analyzed and BER for different offset values is reported. For the digit-online decoder configurations synthesized on the FPGA, 11-bit LLR precision decoder configuration gives the highest throughput of 1363 Mb/s in frame-interlaced mode.

A power characterization experiment to compare the power consumed in the bit-parallel and digit-online decoders decoding at a throughput of 450 Mb/s is reported. The energy-per-coded-bit values for all the decoder configurations are presented. Table 6.1 reports the average values of the energy-per-coded-bit for different decoder configurations. The lowest E/b values for the digit-online decoder are shown. Energy-per-iteration is reported for a 6-bit LLR digit-online decoder. The power consumed in the FPGA core is due to loading of new frames and the decoder performing the decoding iterations, each of which are reported separately.

A breakdown of the entity resource utilization for the bit-parallel decoder and the digit-online decoder is presented in appendix A. The entities in the bit-parallel decoder tend to

Decoder type	Bit-parallel				Digit-online (single frame)					Digit-online (frame-interlaced)	
LLR Precision(bits)	3	4	5	6	6	8	9	11	13	11	13
Average E/b (nJ)	0.58	0.72	1.09	2.23	**2.77**	3.15	3.39	3.79	4.54	**2.29**	3.25

Table 6.1 : A table showing average E/b values for different decoder implementations, the observed minimum E/b values for digit-online decoder are shown in bold.

increase almost linearly in logic resource utilization with increase in message precisions. This is to be expected, since with increase in the message precision the size of the adders and comparators in the decoder nodes also increases. The entities in the digit-online decoder do not show a specific pattern to the increase in message precision, with the exception of controller. The size of controller increases linearly with increase in the LLR message precision, because of the size of the new frame LLRs being stored in the controller.

The size of the individual nodes in the digit-online decoder fluctuates around the average values listed in Table 6.2. Since the digit-online decoder processes messages in a digit-serial format, increase of message precision in the decoder causes slight increase in registers used each node because of increased pipelining. The size of the adders and the selection networks in the decoder nodes, however, mainly depends on the degree of the node. It might be informative to look at the size of the individual check nodes and variable nodes and predict the size of the decoder for a specific code size.

Decoder Entity	Avg. ALUTs	Avg. DLRs	Avg. ALMs
Check Node (degree 14)	730	293.6	470.1
Check Node (degree 15)	783.4	312.8	501.2
Variable Node (degree 2)	13.2	16.8	15.2
Variable Node (degree 3)	25.6	21.2	19.8
Variable Node (degree 4)	39.8	38.8	39.6

Table 6.2: Table showing average values of various entities in the 576 bit, rate-3/4 digit-online decoder

6.2 Future Work

The 8-bit LLR and 13-bit LLR configurations of digit-online decoder show higher interconnect usage as compared to their neighbors and thus have higher power consumption. No special attempts have been made to reduce logic utilization and interconnect routing resources in case of digit-online decoders. Using appropriate constraints in the Quartus® II, it might be possible to further optimize logic utilization and interconnect usage in these decoder configurations.

The LLRs in the digit-online decoder are loaded in parallel and are stored in ALM resources, this not only causes high usage of routing resources at the input of decoder but also cause a huge switching activity when a new frame is loaded. Instead, it might be possible to distribute the loading of LLRs over a few clock cycles and employ block memory available on the FPGA to store the LLR messages. However, as seen in Section 5.3.3, the power consumed in the loading of a new frame is very less as compared with power consumed in performing decoding iterations. It might be interesting to implement digit-online decoder configurations for higher code sizes and code rates on a bigger FPGA.

Bibliography

[1] R. Gallager, “Low-density parity-check codes,” in *IEEE Trans. Inf. Theory*, vol. 8, no. 1, pp. 21-28, Jan. 1962

[2] C. E. Shannon, “A mathematical theory of communication,” in *ACM SIGMOBILE Mobile Comp. and Commun, Review*, vol. 5, no. 1, pp. 3–55, Jan. 2001.

[3] A. J. Blanksby and C. J. Howland, “A 690-mW 1 Gb/s 1024-b, rate-1/2 low-density parity-check code decoder,” in *IEEE J. Solid-State Circuits*, vol. 37, no. 3, pp. 404-412, Mar. 2002.

[4] A. Daraibha, A. C. Carusone, and F. R. Kschischang, “A bit serial approximate min-sum LDPC decoder and FPGA implementation,” in *Proc.2006 IEEE Int. Symp. Circuits and Systems*, pp.149-152, May 2006.

[5] P. A. Marshall, V. C. Gaudet, and D. G. Elliot, “Deeply pipelined digit-serial LDPC decoding,” in *IEEE Trans. Circuits and Sysems I, Reg. Papers*, vol. 59, no. 12, pp. 2934-2944, Dec. 2012.

[6] 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. Elliott, and C. Schlegel, “A scalable LDPC decoder ASIC architecture with bit-serial message exchange,” in *Integration VLSI J.*, vol. 41, no. 3, pp. 385–39, May 2008.

[7] M. Ercegovac, “Online arithmetic: An overview,” in *Proc. SPIE V.495: Real Time Signal Processing VII*, pp. 86–93, Aug 1984.

[8] M. J. Irwin and R. M. Owens, “Digit-pipelined arithmetic as illustrated by the paste-up system: A tutorial,” in *Computer*, vol. 20, no. 4, pp. 61–73, Apr 1987.

[9] S.J. Li, T. L. Brandon, D. G. Elliott, and V. C. Gaudet, "Power Characterization of a Gbit/s FPGA Convolutional LDPC Decoder," in *IEEE Workshop on Sig. Proc. Sys. (SiPS)*, pp. 294-299, Oct. 2012

[10] R. M. Tanner, “A recursive approach to low complexity codes,” in *IEEE Trans. on Inf. Theory*, vol. 27, pp. 533-598, Feb, 2001.

[11] X. Huang, “Near perfect decoding of LDPC codes,” in *Proc. 2005 Int. Symp. Information Theory (ISIT 2005)*, pp. 302–306, Sept 2005.

[12] T. Zhang, Z. Wang, and K. K. Parhi, "On finite-precision implementation of low density parity check codes decoder," in *2001 Int. Symp. Circuits and Systems(ICAS 2001),* vol. 4, pp. 202–205, May 2001.

[13] J. Sha, Z. Wang, M. Gao, and L. Li, "Multi-Gb/s LDPC code design and implementation," in *IEEE Trans. Very Large Scale Integr. (VLSI) Syst,* vol. 17, no. 2, pp. 262-268, Feb. 2009.

[14] C. Zhang, Z. Wang, J. Sha, L. Li, and J. Li, "Flexible LDPC decoder design for multi gigabit-per-second applications," in *IEEE Trans. Circuits and Systems I, Reg. papers,* vol. 57, pp. 116-124, Jan. 2010.

[15] P. A. Marshall, V. C. Gaudet, and D. G. Elliot, "Effects of Varying Message Precision in Digit-Online LDPC Decoders," in *IEEE Workshop on Sig. Proc. Sys. (SiPS),* pp. 7-12, Oct. 2012.

[16] F. R. Kschischang, B. J. Frey, and H. A. Loeliger, "Factor graphs and the sum-product algorithm," in *IEEE Trans. Inf. Theory,* vol. 47, no. 2, pp. 498-519, Feb. 2001.

[17] S. Y. Chung, G. D. Forney, Jr., T. J. Richardson, and R. Urbanke, "On the design of low-density parity-check codes within 0.0045dB of the shannon limit," in *IEEE Commun. Lett.,* vol. 5, pp. 58-60, Feb. 2001.

[18] Si-Yun Li, "Power Characterisation of a Gbit/s FPGA Convolutional LDPC Decoder," *M.A.Sc Thesis,* University of Waterloo, 2012.

[19] J. Zhao, F. Zarkeshvari and A. H. Banihashemi, "On implementation of min –sum algorithm and its modifications for decoding LDPC codes" in *IEEE Trans. Commun.* vol.53, no. 4, pp. 549-554, Apr. 2005.

[20] J. Chen and M. P. C. Fossorier, "Density evolution for BP-based decoding algorithms of LDPC codes and their quantized versions," in *Proc. IEEE GLOBECOM, Taipei,* vol.2, pp. 1378-1382, Nov. 2002.

[21] A. J. Blanksby and C. J. Howland, "A 690-mW 1 Gb/s 1024-b, rate-1/2 low-density parity-check code decoder," in *IEEE J. Solid-State Circuits,* vol. 37, no. 3, pp. 404-412, Mar 2002.

[22] C. J. Howland and A. J. Blanksby, "Parallel decoding architectures for low-density parity-check codes," in *IEEE Proc. ICAS,* vol. 4, pp. 742-745, May 2001.

[23] A. Darabiha, A. C. Carusone, and F. R. Kschischang, "Power reduction techniques for LDPC decoders," in *IEEE J. Solid-State Circuits,* vol. 43, pp. 1835-1845, Aug, 2008.

[24] N. Onizawa, T. Ikeda, T. Hanyu, V. Gaudet, "3.2 Gb/s 1024-b rate-1/2 LDPC decoder chip using a flooding-type update-schedule algorithm," in *Proc. 50th IEEE Midwest Symp. Circuits Systems, Brest, France,* pp. 217-220, Aug. 2007.

[25] Z. Zhang, V. Anantharam, M. J. Wainwright, and B. Nikolic, "An efficient 10GBASE-T ethernet LDPC decoder design with low error floors," in *IEEE J. Solid-State Circuits,* vol. 45, no. 4, pp. 843-855, Apr. 2010.

[26] Z. Zhang, V. Anantharam, M. J. Wainwright, and B. Nikolic, "A 47Gb/s ldpc decoder with improved low error performance," in *Symp. VLSI Circuits Dig. Kyoto,* pp. 286-287, Jun. 2009.

[27] H. Zhong and T. Zhang, "Block-LDPC: A practical LDPC coding system design approach," in *IEEE Trans. on Circuits and Systems I,* vol. 52, no. 4, pp. 766-775, Apr. 2005.

[28] H. Zhong and T. Zhang, "Design of VLSI implementation-oriented LDPC codes," in *Proc. IEEE Semiann. Vehicular Technology Conf.,* vol.1, pp. 670-673, Oct. 2003.

[29] G. Al-Rawi, J. Cioffi, and M. Horowitz, "Optimizing the mapping of low-density parity-check codes on parallel decoding architectures," in *Proc. IEEE ITCC, Las Vegas,* pp. 578-586, Apr. 2001.

[30] A. Darabiha, A. C. Carusone, and F. R. Kschischang, "Block-interlaced LDPC decoders with reduced interconnect complexity," in *IEEE Trans. Circuits Syst. II,* vol. 55, no. 1, pp. 74-78, Jan, 2008.

[31] Y. Kou, S. Lin, and M. P. C. Fossorier, "Low-density parity-check codes based on finite geometrics: A rediscovery and new results," in *IEEE Trans. Inf. Theory,* vol. 47, no. 7, pp. 2711-2736, Nov. 2001.

[32] A. Nouh and A. Banihashemi, "Bootstrap decoding of low-density parity-check codes," in *IEEE Commun. Lett.,* vol. 6, pp. 391-393, Sept. 2002.

[33] J. Zhang and M. Fossorier, "A modified weighted bit-flipping decoding of low-density parity-check codes," in *IEEE Commun. Lett.,* vol. 8, pp. 165-167, Mar. 2004.

[34] M. Jiang, C. Zhao, Z. Shi, and Y. Chen, "An improvement on the modified weighted bit flipping decoding algorithm for LDPC codes," in *IEEE Commun. Lett.*, vol. 9, pp. 814-816, Sep. 2005.

[35] T. L. Brandon et. al., "A scalable LDPC decoder ASIC architecture with bit-serial message exchange," in *Integration VLSI J.*, vol. 41, no. 3, pp. 385-398, May, 2008.

[36] A. Darabiha, A.C. Carusone, and F. R. Kschischang, "A 3.3-Gbps bit-serial block-interlaced min-sum LDPC decoder in 0.13-μm CMOS," in *Proc. Custom Integrated Circ. Conf.*, pp. 459-462, Sep. 2007.

[37] J. T. Tou, "Advances in information systems science," *Springer*, 1969

[38] B. D. Brown and H. C. Card, "Stochastic neural computation. I. Computational elements," in *IEEE Trans. Comput.*, vol. 50, pp. 891-905, Sep, 2001.

[39] W. J. Gross, V. Gaudet, and A. Milner, "Stochastic implementation of LDPC decoders," in *Proc. 39th Asilomar. Conf. Signals, Systems, Computers, Pacific Grove, CA*, pp. 713-717, Nov. 2005.

[40] S. S. Tehrani, W.J. Gross, and S. Mannor, "Stochastic decoding of LDPC codes," in *IEEE Commun. Lett.*, vol. 10, pp. 716-718, Oct. 2006.

[41] S. S. Tehrani, S. Mannor, and W.J. Gross, "Fully parallel stochastic LDPC decoders," in *IEEE Trans. Signal Process.*, vol. 56, no. 11, pp. 5692-5703, Nov. 2008.

[42] S. S. Tehrani, A. Naderi, G.-A. Kamendje, S. Hemati, S. Mannor, and W. J. Gross, "Majority based Tracking Forecast Memories for Stochastic LDPC decoding," in *IEEE Trans. Signal Process.*, vol. 58, no. 9, pp. 4883–4896, Sept. 2010.

[43] S. Hemati, A. H. Banihashemi, and C. Plett, "A 0.18-μm CMOS Analog min-sum iterative decoder for a (32, 8) low-density parity-check (LDPC) code," in *IEEE J. Solid-State Cir.*, vol. 41, Nov. 2006.

[44] M. Zargham, C. Schlegel, J. P. Chamorro, C. Lahuec, F. Seguin, M. Jezequel and V. Gaudet, "Scaling of analog LDPC decoders in sub-100 nm CMOS processes," in *Integration VLSI J.*, vol. 43, no. 4, pp. 365-377, Sept. 2010.

[45] E. Yeo, B. Nikolic, and V. Anantharam, "Iterative decoding architectures," in *IEEE Commun. Mag.*, vol. 41, pp. 132-140, Aug. 2003.

[46] R. Zarubica, S. G. Wilson, and E. Hall, "Multi-Gbps FPGA-based Low-Density Parity- Check (LDPC) Decoder Design," in *IEEE Global Telecommun. Conf., Washington,* pp. 548-552, Nov. 2007.

[47] R. Zarubica and S. G. Wilson, "A solution for memory collision in semi-parallel FPGA-based LDPC decoder design," in *IEEE ACSSC'07, Pacific Grove,* pp. 982-986, Nov. 2007.

[48] M. H. Kim, T. D. Park, C. S. Kim, and J. W. Jung, "An FPGA design of low power LDPC decoder for high-speed wireless LAN," in *IEEE Intl. Conf. on Commun. Tech.,* pp. 1460-1463, Nov. 2010.

[49] S. S. Tehrani, S. Mannor, and W. J. Gross, "An area-efficient FPGA-based architecture for fully-parallel stochastic LDPC decoding," in *Proc. IEEE Workshop on Sig. Process. Systems (SiPS), Shanghai*, pp. 255-260, Oct. 2007.

[50] K. K. Gunnam, G. S. Choi, M. B. Yeary, and M. Atiquzzaman, "VLSI architectures for layered decoding for irregular LDPC codes of WiMAX," in *Proc. IEEE Int. Conf. on Commun.*, pp. 4542-4547, Jun 2007.

[51] V. A. Chandrasetty and S. M. Aziz, "FPGA Implementation of High Performance LDPC Decoder using Modified 2-bit Min-Sum Algorithm," in *Proc. of 2nd Intl. Conf. on Comp. R&D, Kuala Lumpur*, pp. 881-885, May. 2010.

[52] V. A. Chandrasetty and S. M. Aziz, "An area efficient LDPC decoder using a reduced complexity min-sum algorithm," in *Integration VLSI J.*, vol. 45, no. 2, pp. 141-148, Mar. 2012.

[53] X. Chen, J. Kang, S. Lin, and V. Akella, "Memory system optimization for FPGA-based implementation of quasi-cyclic LDPC codes decoders," in *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 58, no. 1, pp. 98-111, Jan. 2011.

[54] W. Sulek, M. Kucharczyk, and G. Dziwoki, "GF(q) LDPC decoder design for FPGA implementation," in *IEEE Proc. 10th Annual Consumer Commun. and Networking Conf., Las Vegas*, pp. 445-450, Jan 2013.

[55] C. Spagnol, E. M. Popovici, and W. P. Marnane, "Hardware implementation of GF(2^m) LDPC Decoders," in *IEEE Trans. Circuits Syst. I, Reg. papers*, pp. 2609-2620, Dec. 2009.

[56] Altera Stratix IV documententation Chapter 2: Logic Array Blocks and Adaptive Logic Modules

[57] P. A. Marshall, "Digit-Online LDPC Decoding," *Ph.D. dissertation*, University of Alberta, 2013.

[58] M. D. Ercegovac, "On-line arithmetic: An overview," in *Real Time Signal Processing VIII, Proc. SPIE*, vol. 495, pp. 86-93, Aug. 1984.

[59] R. M. Owens, "Techniques to reduce the inherent limitations of fully digit on-line arithmetic," in *IEEE Trans. Comput.*, vol. C-32, no. 4, Apr. 1983.

[60] A. Avizienis, "Signed-digit number representations for fast parallel arithmetic," in *IRE Trans. Electron. Comput.*, vol. EC-10, pp. 389-400, Sept. 1961.

[61] www.pages.ripco.net/~jgamble/nw.html

[62] Altera Stratix IV Development and Education Board (DE4) Schematic, *Terasic*, Version 1.0.

[63] D. Nyugen, A. Davare, M. Orshansky, D. Chinnery, B. Thompson, and K. Keutzer, "Minimization of dynamic and static power through joint assignment of threshold voltages and sizing optimization," in *Proc. Int. Symp. Low-Power Electronics Design (ISLPED '03)*, pp. 158-163, Aug. 2003.

[64] www.altera.com/literature/wp/wp-01059-stratix-iv-40nm-power-management.pdf

Appendix A
FPGA Resource Utilization Analysis

Bit-parallel and digit-online LDPC decoders discussed in this thesis decode LDPC code words of size 576 bit and rate-3/4. Both the implemented decoders consists of controller, check nodes (degrees 14 and 15) and variable nodes (degrees 2, 3 and 4). It is informative to look how FPGA resource utilization in each entity varies with increase in LLR message precisions input to the decoders. Resource utilization plots showing ALUTs (Adaptive Look Up Tables), DLRs (Dedicated Logic Registers) and ALMs (Arithmetic Logic Modules) usage are shown.

A.1 Bit-Parallel Decoder

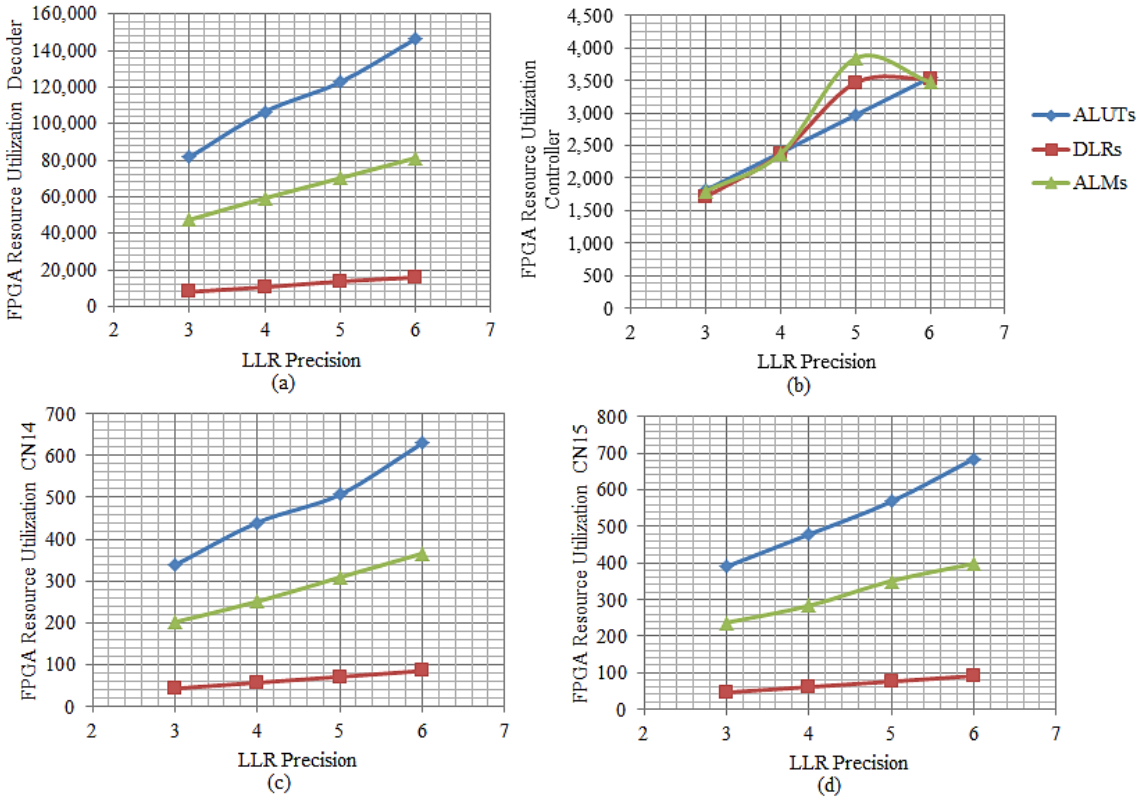

Figure A.1: Resource utilization plots for (a) bit-parallel decoder, (b) controller, (c) degree-14 check node (avg.), (d) degree-15 check node (avg.) are shown vs LLR precision.

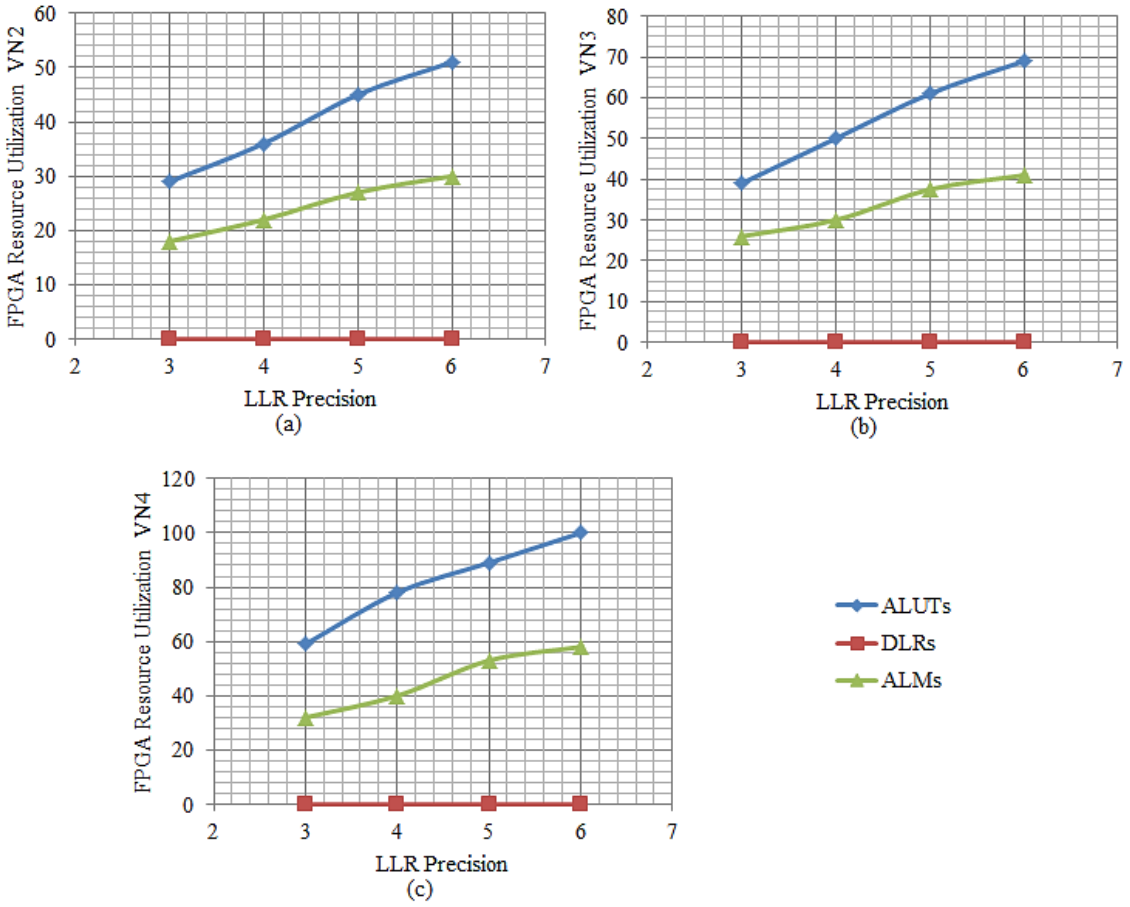

Figure A.2: Average resource utilization plots for bit-parallel decoder (a) degree-2 variable node, (b) degree-3 variable nodes and (c) degree-4 variable node are shown vs LLR precision.

A.2 Digit-Online Decoder

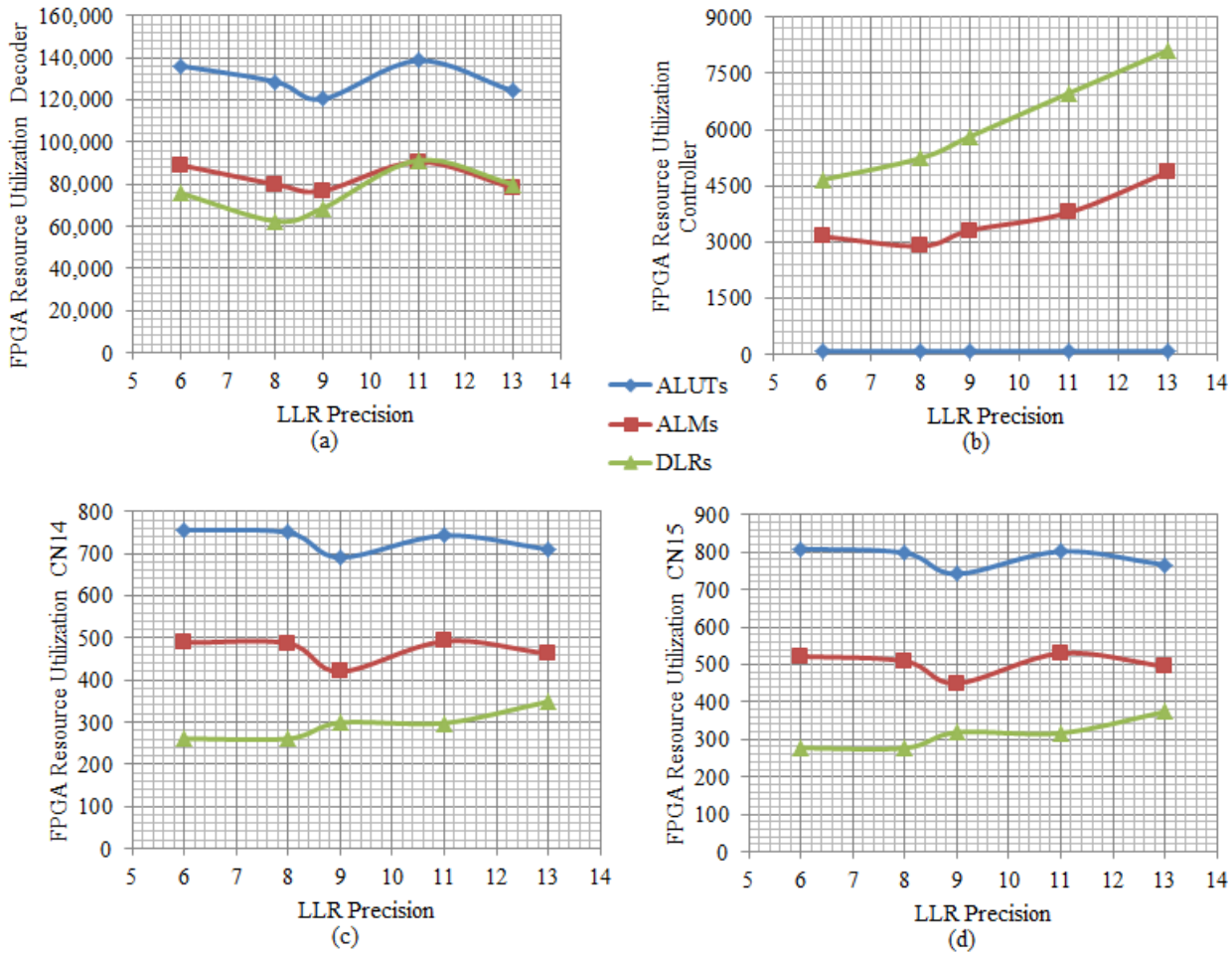

Figure A.3: Resource utilization plots for (a) digit-online decoder, (b) controller, (c) degree-14 check node (avg.), (d) degree-15 check node (avg.) are shown vs LLR precision.

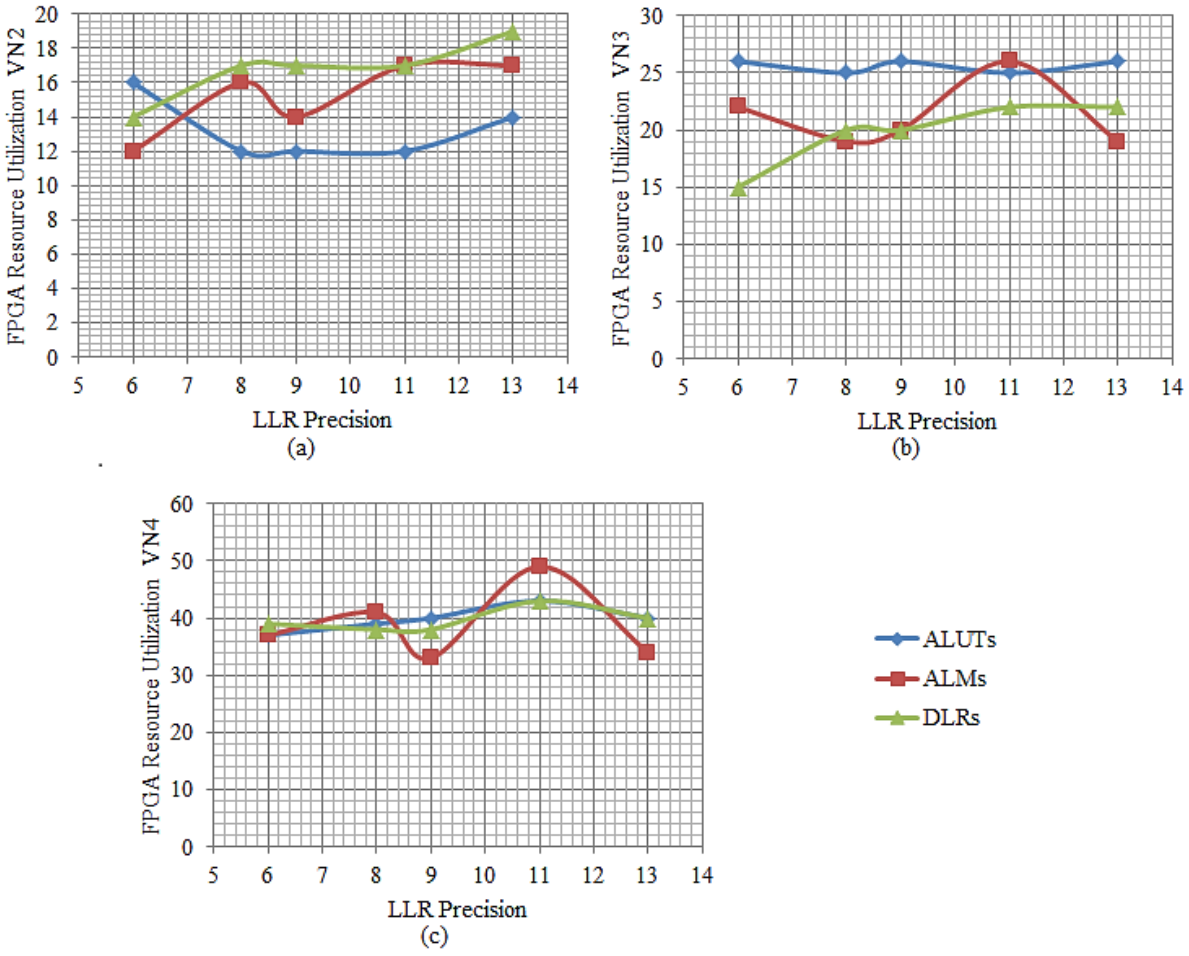

Figure A.4 : Average resource utilization plots for digit-online decoder (a) degree-2 variable node, (b) degree-3 variable nodes and (c) degree-4 variable node are shown vs LLR precision.