Power Characterization of a Gbit/s FPGA Convolutional LDPC Decoder

by

Si-Yun Li

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, 2012

© Si-Yun Li 2012

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

In this thesis, we present an FPGA implementation of parallel-node low-density-parity-check convolutional-code (PN-LDPC-CC) encoder and decoder. A 2.4 Gbit/s rate-1/2 (3, 6) PN-LDPC-CC encoder and decoder were implemented on an Altera development and education board (DE4). Detailed power measurements of the FPGA board for various configurations of the design have been conducted to characterize the power consumption of the decoder module. For a E_b/N_0 of 5 dB, the decoder with 9 processor cores (pipelined decoder iteration stages) has a bit-error-rate performance of 10^{-10} and achieves an energy-per-coded-bit of 1.683 nJ based on raw power measurement results. The increase in E_b/N_0 can effectively reduce the decoder power and energy-per-coded-bit for configurations with 5 or more processor cores for $E_b/N_0 < 5$ dB. The incremental decoder power cost and incremental energy-per-coded-bit also hold a linearly decreasing trend for each additional processor core. Additional experiments are performed to account for the effect of the efficiency of the DC/DC converter circuitry on the raw power measurement data. Further experiments have also been conducted to quantify the effect of clipping thresholds, bit width for each processor core on bit-error-rate (BER) performance, power consumption, logic utilization of the decoder. A "6Core" decoder with growing bit-width log-likelihood ratios (LLRs) has been found to have a BER performance near that of a "6Core" 6-bit decoder while consuming similar power, and logic utilization to that of a 5-bit "6Core" decoder.

Acknowledgements

I would like to thank my supervisors Dr. Vincent Gaudet and Dr. Duncan Elliott for their guidance, support, funding, and their confidence in me and my work.

Many thanks to Dr. Manoj Sachdev and Dr. Siddharth Garg for taking the time to read my thesis as well as allowing me to borrow the DE4 board from Dr. Sachdev's group.

Thanks to Dr. Chris Backhouse for lending me the extra equipments for power measurements.

I would also like to thank Dr. Zhengang Chen for developing the PN-LDPC-CC codes and Dr. Tyler Brandon for providing HDL code for the encoder and decoder modules, and for making my project possible.

A big thank you to Philip Marshall, Russell Dodd, Brendan Crowley and Stephen Holmes for their support and guidance for LDPC-related problems and discussions.

Special thanks to Logan Gunthrope and Gary Block for their extensive support in HDL, LDPC-CC, Tcl and FPGA-related matters.

To Danny Tsuei, thank you for showing me how to make use of the JTAG-MM interface with the DE4 board, which greatly sped up the data-gathering portion of my work.

Thank you to Karl Jensen for his extensive support with LaTeX-related problems.

Special thanks to Altera for the donation of the DE4 board.

Lastly, I would like to thank University of Waterloo for the QEIIGSST scholarship, and NSERC and PetroCanada for funding.

Dedication

This is dedicated to my parents and friends.

Table of Contents

List of Tables

List of Figures

Nomenclature

ASIC	Application-Specific Integrated Circuit
AWGN	Additive White Gaussian Noise
BER	Bit-Error-Rate
BPSK	Binary Phase-Shift Keying
FIFO	First-In First-Out
FPGA	Field-Programmable Gate Array
LDPC	Low-Density Parity-Check
LDPC-BC	LDPC Block Codes
LDPC-CC	LDPC Convolutional Codes
LFSR	Linear Feedback Shift Register
LLR	Log-Likelihood Ratio

PN-LDPC-CC	Paralllel-Node Low-Density Parity-Check Convolutional-Code
Tcl	Tool Command Language

H	Parity-check matrix (PCM)
E_b/N_0	Energy-per-bit divided by spectral noise density
ρ	node-parallelization factor
T_s	code period
T_s'	group period, T_s/ρ

Chapter 1

Introduction

1.1 Motivation

The goal of forward error control is to minimize the net error rate of a communications channel and to improve the overall efficiency of data transmission over noisy channels by adding redundancy to data to avoid complete data retransmission. Low-density parity-check (LDPC) codes, in particular, are a class of linear error correcting codes which were first proposed by Gallager in 1962 [3]. LDPC block codes (LDPC-BC) are characterized by a seemingly random parity-check matrix, known as the H matrix, with mostly 0's and a small fixed number of 1's. However, despite its high performance, the decoding of LDPC codes was impractical to implement due to limitations imposed by the processing capabilities available at the time.

After being forgotten, LDPC codes were rediscovered by MacKay and Neal [4]. Through the use of a probabilistic decoding algorithm, LDPC codes were proven to perform substantially better than standard convolutional codes could, and their performance approaches

Shannon's theoretical channel capacity limit [5]. Since then, LDPC codes have become one of the most intensive research areas in coding theory. There has been a lot of research done to analyze the behaviour and performance of LDPC codes over different channels (e.g. binary erasure channel, binary symmetric channel, additive white Gaussian noise channel) and under various constraints. Various approaches have been taken to design and construct good LDPC codes to achieve channel capacity-approaching performance [6]. LDPC codes have since been widely adopted in numerous digital communications standards, such as the 10GBase-T Ethernet (IEEE 802.3an), wireless-N network standard (IEEE 802.11n), IEEE 802.16e, DVB-S2, an enhanced specification for digital video broadcasting, and wireless personal area network (802.15.3c). Numerous designs of LDPC block code decoders have been shown in recent literature to achieve a decoding throughput in the range of multi-Gb/s through optimizations in decoding algorithms, architecture, and code construction [7, 8, 9, 10, 11, 12, 13, 14].

Turbo codes are another family of popular error-correction codes with capacity-approaching performance, which were first introduced in 1993 [15]. In parallel with the development of LDPC codes, there has also been a lot of active research done in the area of turbo codes. They have been incorporated in 3G and 4G mobile telephony standards such as HSPA, EV-DO and LTE, as well as in other standards, such as WIMAX standard (IEEE 802.16), and DVB-RCS, a specification for interactive satellite communication system. Numerous turbo decoders with Gbit/s performance have been reported in literature [16, 17, 18, 19].

LDPC convolutional codes (LDPC-CC), on the other hand, were first proposed in [20] as a convolutional variant of the original LDPC codes, and past studies have shown that well-designed LDPC-CCs have the same capacity-approaching BER performance as LDPC-BCs [20, 21]. The flexibility to handle frames of arbitrary lengths, a characteristic shared with convolutional codes, makes LDPC-CC a suitable candidate for practical implementations

of communication scenarios, such as video streaming and packet-switching networking. LDPC-CCs have been incorporated in the next-generation High Definition Power Line Communication (HD-PLC) specification [22], and it has also been proposed for use in Global Navigation Satellite System (GNSS) along with its block-code variant [23]. The implementation of LDPC-CC encoding has also been shown to be simpler than its block-oriented counterpart [24, 25]. Recent literature on LDCC-CC-based decoders has also demonstrated the ability to achieve Gbit/s decoding throughput [26, 27, 28].

Past implementations of LDPC decoders have mostly used Field Programmable Gate Arrays (FPGA) as an intermediate tool to prototype their designs instead of targeting it as the final platform [29, 30, 31], because of its relatively slower clock speed, larger area and higher power consumption compared to full-custom application-specific integrated-circuit (ASIC) implementation. However, FPGA does provide much faster turnaround time than ASIC. Therefore, it would be interesting to consider the feasibility of using state-of-the-art FPGAs as final design platform. Recent literature on LDPC decoders implemented on FPGAs has shown improvements in decoding throughput and some of them have even been shown to achieve throughput in the range of several Gigabits per second (Gbit/s) [32, 33, 31, 34]. However, the lack of reported power measurement figures from existing FPGA implementations of LDPC decoder designs has made it difficult to analyze the gap in power consumption between FPGA implementations and ASIC implementations of LDPC decoders, and to explore FPGA-specific architectural tradeoffs.

This thesis presents an FPGA implementation of a parallel-node low-density-parity-check-convolutional-code (PN-LDPC-CC) decoder and a set of detailed power measurement data conducted using a simple power measurement set-up. This work verifies the ability of an FPGA-based PN-LDPC-CC decoder to achieve decoding throughput in the Gbit/s range while consuming a reasonable amount of power, which demonstrates the feasibility

of using FPGA as a final design platform. Architectural tradeoffs of the PN-LDPC-CCs are further investigated, and further experiments have also been conducted to quantify the effect of clipping thresholds, bit width for each processor core on BER performance, power consumption, logic utilization of the decoder. A "6Core" decoder with growing bit-width LLRs has been found to have a BER performance near that of a "6Core" 6-bit decoder while consuming similar power, and logic utilization to that of a 5-bit "6Core" decoder.

1.2 Thesis Organization

This thesis is organized as follows. Chapter 2 includes the background on LDPC-BC as well as LDPC-CC, and the gap between FPGA and ASIC implementation, and a summary of performance results from the existing ASIC and FPGA LDPC decoders. Chapter 3 presents a background on the PN-LDPC-CC encoder and decoder architectures as well as the other system modules implemented in our experiment. Chapter 4 provides a description of the power measurement method employed for further power characterization of our system. Chapter 5 presents the measurement results of our FPGA implementation and the resulting discussion. Chapter 6 presents the results from a performance optimization of the decoder by varying the clipping threshold, precision of each decoder processor core. Conclusions, contributions, and further research directions are presented in Chapter 7.

Chapter 2

Background

This chapter first presents a description on LDPC, decoding of LDPC and LDPC-CC codes respectively in Section 2.1 to 2.3. Then a general background on the gap between FPGA and ASIC implementation is outlined in Section 2.4 along with a summary of the performance results from existing ASIC and FPGA implementations of LDPC decoders.

2.1 LDPC Definitions

Low-density parity-check (LDPC) codes are a class of linear error-control codes, which were first introduced in 1962 [3]. An LDPC code is a linear block code of rate $R = k \,/\, n$, characterized by a sparse *(n - k)* by n parity-check matrix, H. The length of the code is denoted by n, the number of message bits is denoted by k, and $(n - k)$ is the number of the parity (redundant) bits in the codeword. An example for a rate-1/2 LDPC code with $n = 16$ and $k = 8$ is shown in Figure 2.1. All the valid codewords x of an LDPC code must satisfy the constraint $Hx^T = 0$. An LDPC code can also be represented by

a bipartite graph, known as the Tanner Graph [35]. The corresponding Tanner Graph for the H matrix in Figure 2.1 is depicted in Figure 2.2. Each codeword bit corresponds to a variable node, labeled v_i, and each parity bit corresponds to a check node, labeled c_i. Each entry h_{ij} of 1 in the H matrix corresponds directly to an edge (or connection) between variable node i and check node j in its graphical representation.

The H matrix of a *regular* LDPC code contains a fixed number of of 1's in each column, denoted by d_v, and a fixed number of 1's in each row, denoted by d_c. In other words, for a regular LDPC code, the number of edges connected to a check node and the number of edges connected to a variable node are constant. An *irregular* LDPC, on the other hand, has nodes of varying degrees defined by a degree distribution. Although LDPC codes are block codes, there also exist LDPC convolutional codes (LDPC-CC), which are further described in Section 2.3.

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

Figure 2.1: *Example of H matrix for a rate-1/2 LDPC code with $n = 16$, $k = 8$*

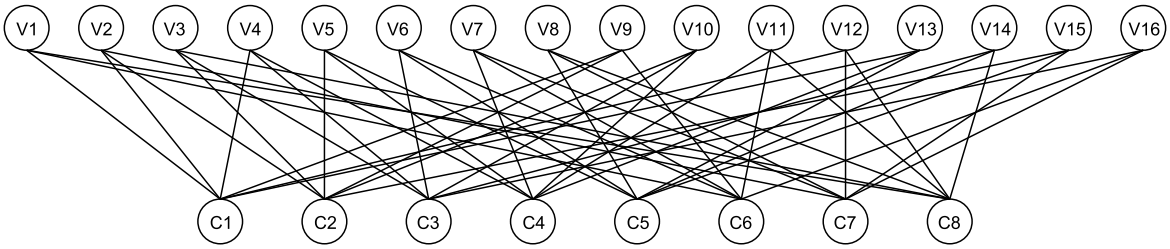


Figure 2.2: *Tanner graph representation of the* H *matrix in Figure 2.1. Variable nodes are shown on top, and check nodes at the bottom*

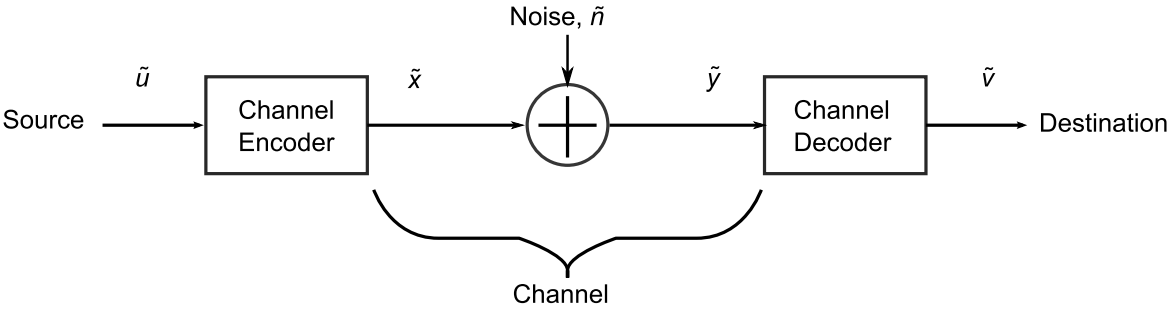


Figure 2.3: *Basic Communication System. Channel modulator and demodulator are not shown in the picture.*

2.2 LDPC Decoding

In a basic communication system, as depicted in Figure 2.3, an information message $\tilde{u}$ is encoded by the channel encoder to a coded sequence $\tilde{x}$ with added redundancy to provide error protection. The coded sequence $\tilde{x}$ is then modulated by the modulator to allow transmission through the channel, where it gets distorted by noise. At the receiving side, the received signal is demodulated by the demodulator. The demodulated sequence $\tilde{y}$ then goes through the channel decoder, where it gets decoded into received encoded message, $\tilde{v}$, and sent to destination.

For an LDPC decoder, the received channel messages are expressed in the form of

log-likelihood-ratios (LLRs), which represent the scaled and quantized channel samples of original encoded data with added noise scaled according to a E_b/N_0, the ratio of the energy-per-transmitted bit versus the spectral noise density. An LLR is defined by Equation 2.1, where $P(x_i = 0/1|y_i)$ is the probability that x_i is zero or one respectively knowing the value of y_i, for an additive white Gaussian noise (AWGN) channel based on binary phase-shift keying (BPSK) modulation scheme. Equation (2.4) describes the operation at the variable node that computes the final decoder outputs.

$$LLR_i(y) = \ln \frac{P(x_i = 0|y_i)}{P(x_i = 1|y_i)} \tag{2.1}$$

LDPC codes can be iteratively decoded using the Sum-Product Algorithm (SPA), which is a general-purpose algorithm for probabilistic inference based on factor graphs [36]. The variable-node and check-node operations of SPA can be generalized in Equation (2.2) and (2.3), where L_i^0 represents the initial received LLR at variable node i, $\lambda_{i\to j}$ represents an LLR from variable node i to check node j, $L_{j\to i}$ represents an LLR from check node j to variable node i, d_c denotes degree of the current check node, and d_v denotes the degree of the current variable node.

$$\lambda_{i\to j} = L_i^0 + \sum_{\substack{k=1 \\ k\neq j}}^{d_v} L_{k\to i} \tag{2.2}$$

$$L_{j\to i} = 2\tanh^{-1}\left(\prod_{\substack{k=1 \\ k\neq i}}^{d_c} \tanh\left(\frac{\lambda_{k\to j}}{2}\right)\right) \tag{2.3}$$

$$\lambda_{i\to j}^n = L_i^0 + \sum_{k=1}^{d_v} L_{k\to i} \tag{2.4}$$

Let y_i denote the received message at variable node i, $i \in \{1, d_v\}$, from an AWGN channel. The SPA decoding procedure can be described as the following steps, where the extrinsic principle of excluding self-message is applied at both the check-node and variable-node operations:

1. Set $n = 0$. Initialize LLR L_i^n as $\frac{2}{\sigma_N^2} y_i$ for each variable node i, where σ_N is the noise variance and $\sigma_N^2 = \frac{N_0}{2}$, where N_0 is the noise power spectral density.

2. Variable node LLRs are passed along all edges to the connected check nodes as inputs:$\lambda_{i \to j} = L_i^n$, where $j \in \{1, d_c\}$.

3. At the check node, the outgoing LLR sent from the current check node j along the edge to each connected variable node i is computed using Equation (2.3).

4. At the variable node, the outgoing LLR sent from the current variable node i to each connected check node j is computed using Equation (2.2).

5. The result for the current decoding iteration n is $\lambda_{i \to j}^n$, the sum of the result of the variable node computations, computed using the Equation (2.4).

6. If $\lambda_{i \to j}^n$ satisfies the constraint $\lambda_{i \to j}^n \cdot H^T = 0$ or a fixed number of iterations are reached, terminate decoding and output the current result. If neither condition is met, set $n = n + 1$, return to Step 2 to start the next iteration.

However, while being capable of achieving excellent bit-error-rate (BER) performance, the implementation of the SPA has been shown to be rather demanding of hardware resources, because it is an iterative numerically intensive algorithm that performs arithmetic operations on real-valued information of large blocks of data. However, the SPA is an inherently parallel algorithm. A good alternative to SPA is the Min-Sum algorithm [37, 38, 39],

which is an approximation of the SPA that offers reduced complexity. The simplification of the SPA is achieved by approximating the check-node operation depicted in Equation 2.3 with the operation described in Equation 2.5 instead, where the magnitude of the outgoing message on edge i is the minimum of the incoming message magnitudes excluding the one from edge i, and its sign is the XOR of the signs of the incoming messages except the one from edge i.

its inherent parallelism,

$$L_{j\to i} = \left(\prod_{\substack{k=1 \\ k\neq i}}^{d_c} sign\left(\lambda_{k\to j}\right) \right) \times \min_{\substack{k=1,\ldots,d_c \\ k\neq i}} |\lambda_{k\to j}| \tag{2.5}$$

This approximation eliminates the use of LUTs to implementation *tanh* calculations, which in turn reduces the complexity in hardware implementation. However, a loss of up to 1.03 dB is observed in decoding performance after the application of the min-sum approximation [40]. There also exists a number of modifications proposed for the original min-sum approximation, such as the offset min-sum algorithm and the normalized min-sum algorithm, which both perform very close to the ideal sum-product algorithm or even outperform that in some cases, but have hardware complexity closer to that of min-sum algorithm [39].

2.3 LDPC-CC Definitions

LDPC convolutional codes (LDPC-CC) were first proposed in [20]. An LDPC-CC bears the same characteristics as conventional convolutional codes, where a current code bit depends on present and previous information bits only. An LDPC-CC can be characterized by

$$H^T = \begin{bmatrix} \ddots & & \ddots & & & & 0 \\ & H_{m_s}^T(t) & \cdots & H_0^T(t) & & & \\ & & \ddots & \vdots & \ddots & & \\ & & & H_{m_s}^T(t+m_s) & \cdots & H_0^T(t+m_s) & \\ 0 & & & & \ddots & \vdots & \ddots \end{bmatrix}$$

Figure 2.4: *H matrix for LDPC-CC. Note the diagonal band structure of non-zero elements.*

a parity-check matrix, H. However, unlike LDPC block codes (LDPC-BC), which have finite-length parity-check matrices, the parity-check matrix for LDPC-CC in infinite in length. Like LDPC-BC, all of the valid codewords w of an LDPC-CC must satisfy the condition, where $wH^T = 0$. The transposed form of the H matrix for an LDPC-CC with the rate $R = b \ / \ c$ $(b < c)$ is shown in Figure 2.4, where the $H_i^T(t)$ $(i = 0, 1, ..., m_s)$ represents the periodic time-varying sub-matrices of size $c \times (c - b)$ and the parameter m_s is called the code memory. For a finite code period T_s, $H_i^T = H_i^T$ $(t + T_s)$ for all i and t. If $T_s = 1$, the LDPC-CC is considered time-invariant, else time-varying. The overall constraint length of the rate -b / c LDPC-CC is defined as $\nu = c \times (m_s + 1)$. For *regular* LDPC-CCs, both the column weight (K) and the row weight (J) of H^T are constant, and such a code can be referred to as an (m_s, J, K) LDPC- CC code. For a (m_s, J, K) code, the column weight (K) defines the number of LLRs entering check-node from the variable-node, while the row weight (J) defines the number of LLRs entering each variable-node from the check-nodes. While the LDPC decoding algorithms described in Section 2.2 still apply for LDPC-CC, the iterations are realized as pipelined "processor cores". Additional parallelism can be achieved through design of the LDPC-CC code to allow the encoding and decoding of ρ information bits per cycle, as demonstrated in [26], where ρ is defined as the parallelization ratio.

2.4 State-of-the-art LDPC Decoders

An LDPC decoder can be implemented in a digital Application-Specific Integrated Circuit (ASIC) or Field-Programmable Gate Array (FPGA). There have also been analog implementations of LDPC decoders [41, 42]. Digital ASIC implementation is usually preferred as it provides higher clock frequency, smaller area and lower power consumption than FPGA implementation. In a recent paper, Kuon and Rose presented experimental measurements of area, speed, and power consumption to analyze the gap between ASICs and FPGAs [43]. They suggest that by making use of the available hard heterogeneous blocks (such as memory, DSP blocks) on FPGAs, the resulting gap in area can be narrowed, as can the gap in power consumption. However, they also suggest that the possibility of narrowing the gap in clock speed performance would depend largely on how well the designs are tailored to the functionality of the DSP block.

The performance of several state-of-the-art ASIC-based LDPC decoders that is reviewed in Section 2.4.1. Preference is given to designs with reported power measurements to demonstrate the trend of power consumptions of ASIC-based LDPC decoders. A summary of several Gbit/s FPGA decoders is also provided in Section 2.4.2.

2.4.1 ASIC-based LDPC Decoders

An LDPC decoder that supports four code rates of IEEE 802.15.3c applications is presented in [13]. The proposed decoder utilizes row-based layered scheduling, and achieves a maximum throughput of 5.79 Gbit/s at a clock frequency of 197 MHz. Darabiha *et al.* introduced a highly-parallel decoder architecture with low routing overhead and proposed a method to detect early convergence of iterative decoder to allow reduction in dynamic power [9]. A bit-serial fully-parallel LDPC decoder employing the proposed architecture

and optimization is reported to achieve a total throughput of 3.3 Gbit/s. A Reed-Solomon-based LDPC (RS-LDPC) decoder suitable for 10GBASE-T Ethernet is presented in [14]. The resulting decoder delivers a throughput of 47.7 Gbit/s at a clock frequency of 700 MHz, and is capable of achieving a BER of 10^{-14} at SNR of 5.5 dB with the use of a post-processing algorithm. to Lastly, an LDPC-CC decoder fabricated in a 90-nm process is reported to achieve 2.37 Gbit/s at a clock frequency of 198 MHz [28]. The proposed decoder employs the on-demand variable node activation scheduling to improve convergence speed, and a node-level optimization to increase decoding throughput. A summary of the results from the aforementioned ASIC-based LDPC decoders is included in Table 2.1.

2.4.2 FPGA-based LDPC Decoders

Darabiha *et al.* introduced a bit-serial decoding scheme that addresses the issue in interconnect complexity in fully parallel implementations of LDPC decoders, along with a simplified check node architecture [29]. The reduction in interconnect complexity is achieved by allowing the multi-bit messages to be communicated between variable node and check node over single wires. The viability of both ideas is demonstrated on a fully parallel FPGA implementation of a (480, 355) Reed-Solomon-based LDPC decoder, which has been shown to achieve a decoding throughput of 650 Mbps at a clock frequency of 61 MHz. A fully parallel LDPC decoder based on revised stochastic decoding was presented by Tehrani *et al.* [30]. The method of noise-depending-scaling (NDS) and the use of edge memories (EM) have been applied to the original stochastic decoding scheme [44]. The resulting decoder has been shown to achieve a decoding throughput of 706 Mbps at a signal-to-noise ratio of 3 dB. In addition to the methods of NDS and of EM, internal memories (IMs) are used for each subnode in the high-degree variable nodes to reduce the

Table 2.1: *Summary of ASIC-based Implementation Results*

	[13]	[9]	[14]	[28]
CMOS Technology	65-nm	130-nm	65-cm	90-cm
Type of LDPC Code	LDPC-BC	LDPC-BC	LDPC-BC	LDPC-CC
LDPC Code	(672, k) k=336, 420,504,588	(660, 480)	(2048, 1723)	(491, 3, 6)
Code Rate	1/2, 5/8, 3/4, 7/8	0.74	0.84	1/2, 2/3, 3/4,4/5,5/6
Core Area [mm^2]	1.56	7.3	5.35	2.24
Iteration or Processor Count	5	15	8	5
Input Quantization	4-bit	4-bit	4-bit	6-bit
BER	10^{-6} at 9 dB for k = 588, 16-QAM modulation	$< 10^{-8}$ at 5.5 dB	$< 10^{-12}$ at 5.5 dB	10^{-5} at 2.5 dB
Supply Voltage [V]	1	1.2	1.2	1 V
Clock Fequency [MHz]	197	300	700	198
Throughput [Gbit/s]	39.9	3.30 (coded)	47.7	2.37
Power [mW]	450	398	2800	284
Energy Efficiency [pJ/bit]	11.3	120 (coded)	58.7	0.12

occurrence of the hold state in a high-degree variable node [32]. A 1.66 Gbit/s fully parallel stochastic decoder based on the irregular WiMAX (1056, 528) LDPC code using the aforementioned techniques has been shown to achieve a BER of 10^{-8} at an E_b/N_o of 4.25 dB. In [33], an FPGA-based decoder based on the 1200-bit rate-1/2 LDPC code using a modified min-sum algorithm has been shown to achieve a decoding throughput of 6 Gbit/s. Chandrasetty *et al.* proposed a modified 2-bit Min-Sum Algorithm (MMS2), which is implemented on an FPGA-based LDPC decoder that achieves an average throughput of 10.2 Gbit/s at an E_b/N_o of 4dB [31]. The difference between the proposed MMS2 algorithm and the traditional Min-Sum algorithm lies in the variable node operation. In MMS2, the variable node operation involves an additional mapping step that maps the computed sum in higher precision to a 2-bit message to be passed to the check node. The mapping is based on a threshold value obtained from Monte-Carlo simulations.

FPGA-specific architectural techniques named *vectorization* and *folding* are proposed by Chen *et al.* in [34] for quasi-cyclic LDPC decoders. *Vectorization* takes advantage of the configurable data width of embedded memory on FPGAs by packing multiple messages into the same physical word, which is loaded and stored simultaneously. However, to concurrently process the messages delivered in each memory access and to take care of the data alignment and addressing, additional logic resources are needed. The extra required hardware resources may result in degraded performance, or it may not even fit on the FPGA due to the alignment logic and interconnect complexity. An additional tool, named QCSyn, which synthesizes a vector architecture for a given quasi-cyclic code, is also developed to ensure high resource utilization. *Folding*, on the other hand, is a memory virtualization technique that allows large LDPC codes to be implemented on commercially-available FPGAs with a small number of available block RAMs by mapping messages corresponding to multiple sub-matrices in the same physical block RAMs.

Results of all the aforementioned implementations based on FPGAs are presented in Table 2.2. None of the included implementations have reported any power values with the exception of the *vectorization*-based implementation [34], which has not identified how power values were obtained. Therefore, this thesis focuses on the power characterization of an FPGA-based LDPC-CC decoder. The next chapter describes the architecture of our FPGA implementation of the PN-LDPC-CC, and detailed power measurement set-up is provided in Chapter 4.

Table 2.2: *Summary of FPGA Implementation Results*

Ref.	[29]	[30]	[32]	[34]	[34]	[31]	[33]	This work
FPGA	Stratix EP1S80	Virtex4 [2]	Virtex4 [2]	Virtex4 [2]	Virtex4 XC4VLX160	Virtex4 [2]	Virtex4 [2]	Stratix IV EP4SGX230
LDPC Code	RS-based (480, 355)	(1024, 512)	Irregular (1056, 528)	Irregular (3969, 3213) QC	(8176, 7156) QC	Regular (3, 6)	PEG-based (6,3)	PN-LDPC-CC (3,6) T_s = 192
f_{CLK}	61 MHz	212 MHz	222 MHz	195.7 MHz	212.2 MHz	123 MHz	100 MHz	75 MHz
Throughput	650 Mbit/s	706 Mbit/s	1.66 Gbit/s	1.474 Gbit/s	713.8 Mbit/s	10.2 Gbit/s	12 Gbit/s	2.4 Gbit/s
BER	-	10^{-6} at 3 dB	10^{-8} at 4.25 dB	-	10^{-12} at 4.25 dB	10^{-5} at 3.9 dB	-	10^{-8} at 4.25 dB
# of Iter.	15	15	-	15	15	-	10	9 (proc.)
Logic Utilization	66588 (84%)	-	-	-	-	-	-	83%
Slices	-	32875	46097	62362 (70%)	17856 (26%)	33345	40613 (45%)	-
Slice FF	-	-	-	98003 (55%)	27210 (20%)	15691	18945 (10%)	-
4-I/P LUTs	-	-	-	111035 (62%)	27046 (20%)	58,053	69038 (38%)	-
Block RAM	-	-	-	330 (98%)	80 (28%)	-	-	-
Power	-	-	-	7632 mW	3033 mW	-	-	4105 mW [1]
Energy/bit	-	-	-	5.18 nJ	4.25 nJ	-	-	1.71 nJ [1]

[1] Measured, *raw data*
[2] XC4VLX200

Chapter 3

FPGA Implementation

This chapter first provides a brief summary on Parallel-Node Low-Density Parity-Check Convolutional Codes (PN-LDPC-CCs) in Section 3.1. Then a detailed description on the architecture of the PN-LDPC-CC-based encoder and decoder implemented is provided in Section 3.2 and 3.3. Lastly, an overview of the remaining modules of the system implemented on FPGA is presented in Section 3.4.

3.1 PN-LPDC-CC

Architecture-aware PN-LDPC-CCs were initially developed by Chen *et al.* [26]. Both implementation-oriented constraints and performance-oriented constraints are applied in the construction of PN-LDPC-CCs to allow parallelism in the encoder and decoder architecture and to also ensure BER performance of the code. For PN-LDPC-CC, the parity-check matrix H^T is organized into groups of sub-matrices $H_{i'}^T(t')$ of size $\rho \times \rho$, where ρ is defined as the node-parallelization factor, for parallel processing. The code period T_s and code

$$H_{i'}^{T}(t') = \begin{bmatrix} H_{i\cdot\rho+\rho-1}^{T}(\rho\cdot t) & H_{i\cdot\rho+\rho-1}^{T}(\rho\cdot t+1) & \cdots & H_{i\cdot\rho+\rho-1}^{T}(\rho\cdot t+\rho-1) \\ H_{i\cdot\rho+\rho-2}^{T}(\rho\cdot t) & H_{i\cdot\rho+\rho-2}^{T}(\rho\cdot t+1) & \cdots & H_{i\cdot\rho+\rho-2}^{T}(\rho\cdot t+\rho-1) \\ \vdots & \vdots & \ddots & \vdots \\ H_{i\cdot\rho}^{T}(\rho\cdot t) & H_{i\cdot\rho}^{T}(\rho\cdot t+1) & \cdots & H_{i\cdot\rho}^{T}(\rho\cdot t+\rho-1) \end{bmatrix}$$

Figure 3.1: *Sub-matrice $H_{i'}^{T}(t')$ for PN-LDPC-CC*

memory m_s are also measured in terms of such groups, where the group period, T_s', is defined as the ratio of T_s/ρ, and group memory, m_s', is defined as m_s/ρ, hence, T_s and m_s must be multiples of ρ by design. The phase parameter, ϕ, $\phi \in [0, T_s)$, refers to the specific row in the parity-check matrix, is also used to specify check constraints at each instant in time. Additionally, for PN-LDPC-CC, a new parameter group phase, ϕ', $\phi \in [0, T_s')$ is introduced. The mapping from $H_i^T(t)$ to $H_{i'}^T(t')$ is shown in Figure 3.1.

In [1], it is shown that ρ can be increased significantly with little impact on the BER performance, and the main factors affecting BER performance are code period (T_s) of the code and the decoding algorithm.

A series of circuit optimizations to the PN-LDPC-CC encoder/decoder architecture presented in [26] and the corresponding synthesis results in terms of energy-per-encoded-bit versus throughput and area versus throughput for each improvement are reported by Brandon in [1]. The throughput for both the encoder and the decoder is be defined as the product of clock frequency and node parallelization factor, ρ, divided by the rate of LDPC-CC code, R, as shown in Equation (3.1).

$$N_{enc/decThroughput} = \frac{f_{clk} \times \rho}{R} \tag{3.1}$$

Our implementation of the PN-LDPC-CC encoder and decoder is based on a pre-existing PN-LDPC-CC architectures by Brandon [1], which are described in details in

Section 3.2 and 3.3.

3.2 PN-LDPC-CC Encoder

A high-level architecture of the PN-LDPC-CC-based encoder implemented in our studies is depicted in Figure 3.2. The input data[ρ-1:0] is connected to each register bank regX[ρ-1:0], $X \in \{0, T'_s\}$ of the T'_s encoder nodes, where T'_s is the group period of the PN-LDPC-CC, and ρ is the parallelization ratio of the code. An example of a single encoder node based on a $T_s = 192$, $\rho = 16$, rate-1/2 (3, 6) PN-LDPC-CC is shown in Figure 3.5. The data in the encoder node is updated according to the regPhase[T'_s-1:0] signals, but remains in the flip-flop until it gets replaced in the next period. The parity output, $v(t)$, is then actively chosen from one set of the T'_s encoder node registers based on the regPhase signals, as shown in the parity output circuitry in Figure 3.4. For a given rate-1/2 PN-LDPC-CC code with $\rho = 16$, the encoder outputs 16 information bits, $u(t)$, and 16 parity bits, $v(t)$, per clock cycle.

The group of phase signals, regPhase, is shown in Figure 3.2 and 3.5, which control the "info" and "parity" updates, are created with one-hot encoding logic, where only one register is being set upon the reset of the encoder, as depicted in Figure 3.3.

As shown in Figure 3.5, the register bank inside each encoder node is implemented as a circular buffer, where it is constrained so that it becomes the next register bank in the next phase, and the input data accepted by the constrained register bank is unique to each phase.

The output parity vector, $v(t)$, is multiplexed out from one of the T'_s register banks with width of ρ based based on the phase specified by the regPhase signals, as shown in

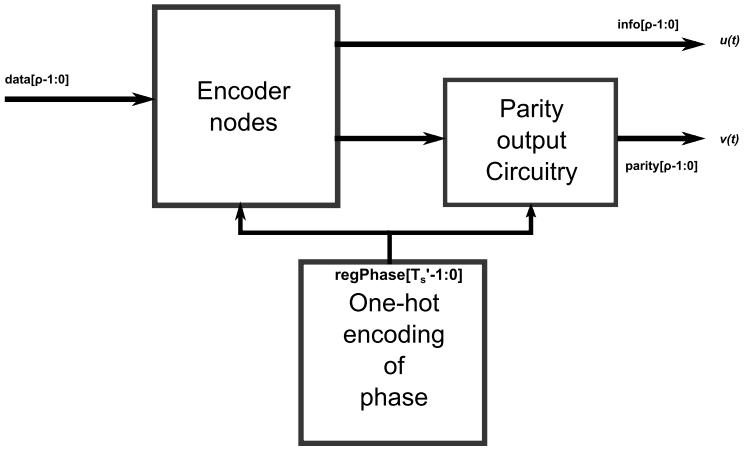


Figure 3.2: *A high-level block diagram of the implemented PN-LDPC-CC encoder. An example of a single encoder node is shown in Figure 3.5. The one-hot encoding of the phase signal is shown in Figure 3.3. Figure 3.4 depicts the parity output circuitry.*

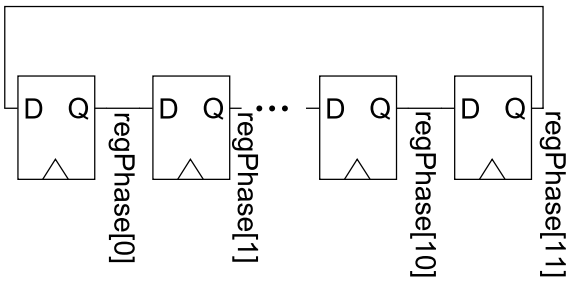


Figure 3.3: *One-hot encoding of the phase signal for the $T_s = 192$, $\rho = 16$, rate-1/2 (3,6) PN-LDPC-CC, group period $T_s' = 12$. [1]*

Figure 3.4.

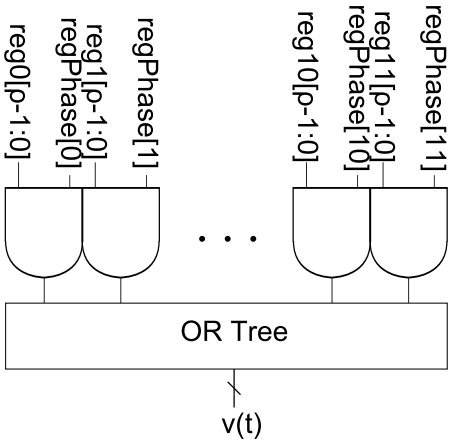


Figure 3.4: *Parity output circuitry for the $T_s = 192$, $\rho = 16$, rate-1/2 (3,6) PN-LDPC-CC, group period $T'_s = 12$. [1]*

The presented encoder node shown in Figure 3.5 includes two optimizations by Brandon [1], which are outlined in Section 3.2.1 and 3.2.2.

3.2.1 Gate-Swapping

The technique named "gate-swapping" is applied to the design of the encoder node. It conditionally swaps the 3-input XOR with a simpler 2-input XOR and a 2-input OR gate when the phase-gated data ("info") and register ("parity") updates inside an encoder node do not occur in the same phase [1]. In Figure 3.5, an encoder node that meets the requirement for "gate-swapping" is depicted. The "gate-swapped" area shown in Figure 3.5 is where the swapping takes place. It can be seen that the regPhase signals, which control the "info" updates in this depicted encoder node are regPhase[5], regPhase[8], and regPhase[10]. The signals that control the "parity" update are regPhase[1] and regPhase[9].

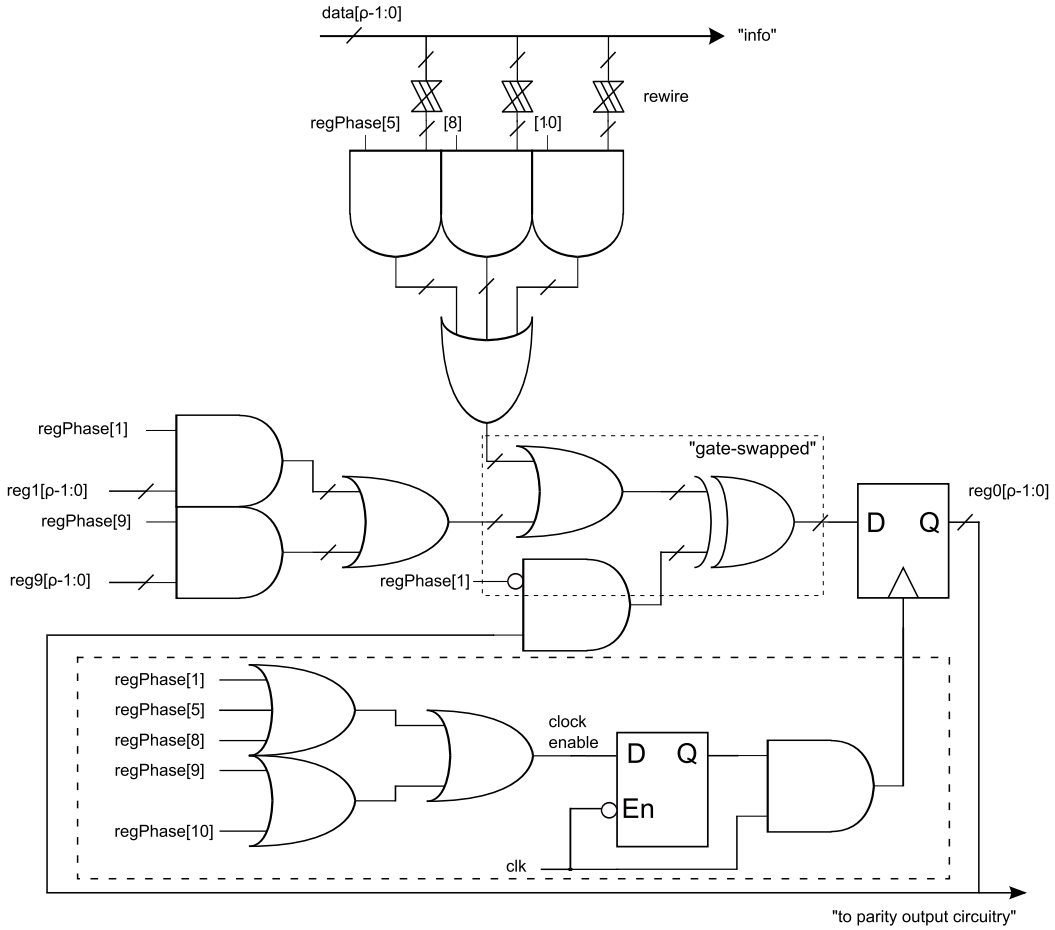


Figure 3.5: *A single encoder node from the implemented architecture in this thesis for the* $T_s = 192$, $\rho = 16$, *rate-1/2 (3,6) PN-LDPC-CC. [1]*

In this case, the phase signals controlling the "info" and the "parity" updates in this encoder node are different; so this allows these phase-gated data and register updates to be combined with a 2-input OR gate, and the result of that is then fed into a 2-input XOR gate. However, in the cases, where the "info" and "parity" updates are controlled by the same phase signals, the 3-input XOR cannot be replaced by a 2-input OR gate and a 2-input XOR. The number of occurrence of simultaneous "info" and "parity" updates varies among all PN-LDPC-CCs. A consistent small reduction in power and area is observed by Brandon among the implemented encoders based on various PN-LDPC-CCs with the application of this technique in [1].

3.2.2 Clock-gating

Latch-based clock-gating [45] is applied to the encoder node design. As depicted at the bottom portion of Figure 3.5, the regPhase phase signals associated with for that particular encoder node are ORed together and the result is latched. The clock is ANDed with the output of the latch to form a new clock signal that controls the encoder-node registers. Brandon noted in [1] that the phase signals have switching rates of $1/2 \cdot T_s'$ per clock cycle and the clock switches once per cycle, so gating the clock with the phase signals can lead to a reduction in dynamic power. A significant reduction in power consumption for all PN-LDPC-CCs using the resulting architecture is observed [1].

3.3 PN-LDPC-CC Decoder

The decoder implemented in our design is formed by cascading identical decoder processors in series. A single decoder processor consists of 2ρ variable-node (vnode) units, ρ check-

node (cnode) units and multiple memories for storage purpose. Figure 3.6 depicts a high-level architecture of a single decoder processor for a rate-1/2 (3, 6) PN-LDPC-CC.

The operation of a decoder processor can be described as follows: as shown in Figure 3.6, the processor takes in channel LLRs or the information and parity LLRs from a previous decoder processor and stores them in the respective info and parity registers. The information/parity input databus has a initial bit-width of $4\rho q$, where 4 represents the degree of the variable node, ρ is the node parallelization factor and q is the bit-width of the LLRs. With the application of TMS, the LLR bit-width of storage elements is reduced by 1 bit by dropping the LSB, with the exception of the channel samples, as denoted in Figure 3.6. LLRs from the info and parity registers are read and routed to the appropriate check-node units in each phase. The processed LLRs from the check-node operations are then written back to the same registers that they were read from, with the exception of one of the parity LLRs, which is routed directly to a variable-node unit. Simultaneously, the variable-node units obtain the processed LLRs from the info and parity registers. The output LLRs from the variable-node units become the outputs of the decoder processor, which are fed into the subsequent decoder processor for another decoding iteration, or they become the input of the hard-decision slicer to form a stream of decoded info bits. The info and parity registers used for storage all have a common clock-gated structure, which is shown in Figure 3.7.

The depicted architecture includes a series of optimizations suggested in [1], which are summarized in the following subsections with their effects on the performance of the PN-LDPC0CC design.

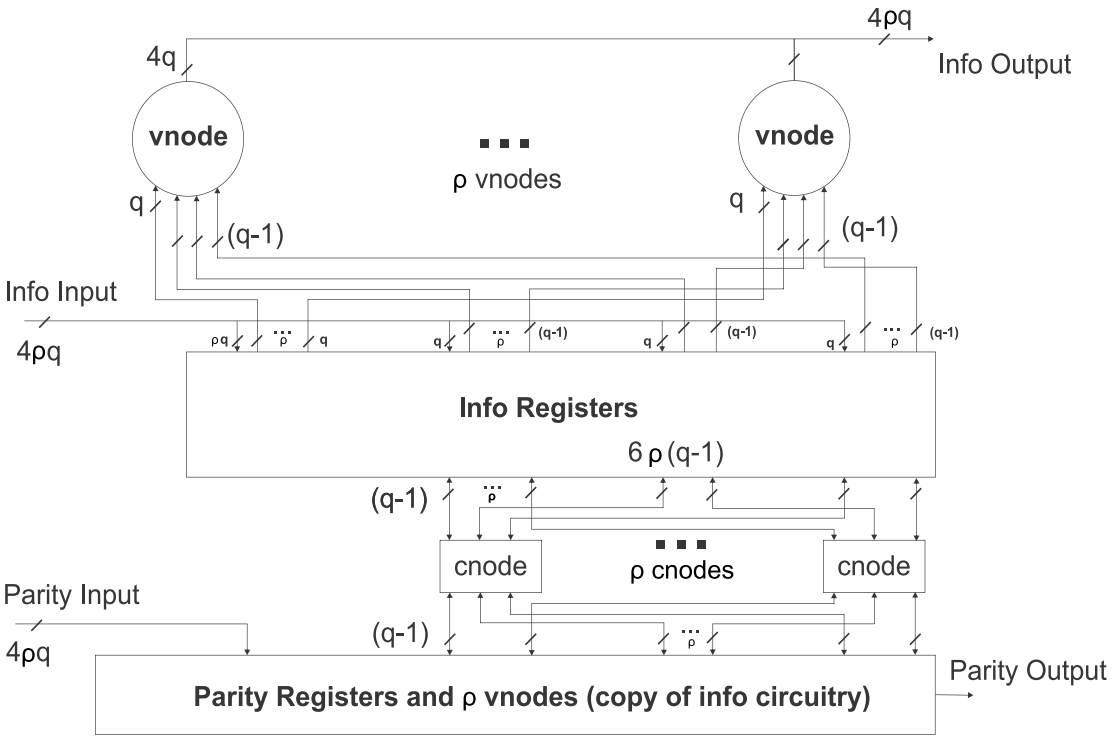


Figure 3.6: *A single decoder processor. [1]*

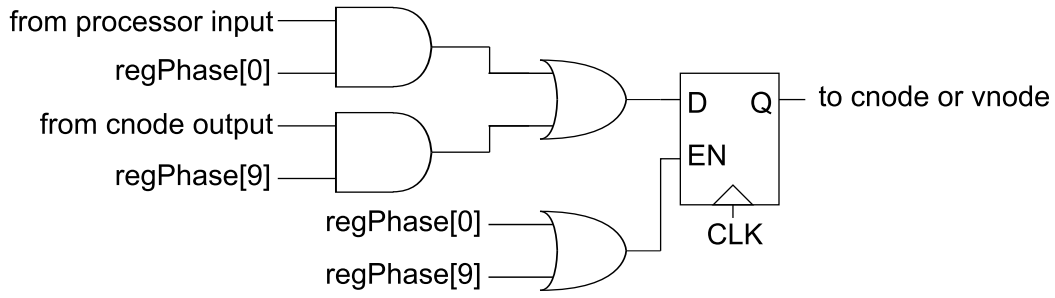


Figure 3.7: *An example of a clock-gated register [1]*

3.3.1 Truncated Min-Sum Check Sum Operation

Brandon [1] has employed a technique named truncated min-sum check sum operation (TMS) for the design of the check node unit. The TMS technique conditionally subtracts a constant value from the check-node LLR magnitudes when the least-significant-bit (LSB) of the LLR is set. This subtracted constant is equal to the maximum value that the LSB can represent. The application of TMS has demonstrated an decrease in energy-per-decoded-bit and area [1].

3.3.2 Removal of Reset Circuitry in Check-Node

Due to the nature of LDPC-CC decoder being deterministic in that the order of operations only depends on the code and not the data, the values in uninitialized registers are unimportant. With this knowledge, the maximum LLR magnitude can be multiplexed into the check-node instead of allowing the LLR values from uninitialized memory locations into the check-node. This is achieved by keeping each input into the check-node at the maximum magnitude until a predetermined phase is reached. The LLR values from the memory are used as inputs to the check-nodes once this predetermined phase is reached. Using this method allows the reset circuitry/memory initialization to be eliminated from the design. Although the effect of the removal of the reset circuitry is minimal on the energy-per-decoded-bit based on simulation results, it does further reduce area [1].

3.3.3 Removal of Saturation Bit

The removal of the saturation bit in the sign-magnitude representation of LLRs is achieved by approximating the function of the saturation bit using the maximum LLR magnitude.

This modification has been shown to have an advantage of reducing power consumption and area [1]. However, the removal of saturation bits also results in a small loss in BER performance [1].

3.3.4 Clock-gated Registers

The technique of clock-gating is applied on the info and parity registers. An example of how a register is clock-gated with the phase signal is depicted in Figure 3.7. This technique does not only further reduce area, but it also reduces the energy-per-decoded-bit [1].

3.4 System Design

The design, LP4, is written in Verilog HDL, compiled using Altera Quartus II Version 11.0 for the FPGA implementation on DE4, which is an Altera development and education board that features a Stratix IV 4SGX230 FPGA. An encoder and decoder based on a rate-1/2 (3,6) PN-LDPC-CC with $T_s = 192$, $\rho = 16$ [26] are implemented using a verified version of the sample HDL code provided by Brandon [1] for the resulting encoder and decoder architectures described in Section 3.2 and 3.3 along with an additive white Gaussian noise (AWGN) channel model based on binary phase-shift keying (BSPK) modulation, a random pattern generator, a first-in first-out (FIFO) buffer, and a simple error counter on the DE4. The overall operation is controlled by a counter-based unit that enables/starts the operation of different modules based on their known latency.

In this design, ρ test bits are generated by the random number generator, which is implemented as a conventional linear feedback shift register (LFSR) with configurable seed value. A copy of the ρ test bits is fed into the encoder module and the FIFO buffer.

For every ρ info bits entering the encoder, it generates ρ code bits. The noise output from the AWGN generator of the channel module is scaled according to the desired E_b/N_0 value and added to the info and code bits generated by the encoder. The sum of the signal and noise is then scaled by a linear function and quantized to 4-bit LLRs, where each LLR consists of one sign bit and three fractional magnitude bits. The quantized LLRs are then fed into the decoder module. The decoded data is then compared with the original data from the FIFO that was previously fed to the encoder. The error counter keeps track of the number of information bits and detected errors until a pre-defined target number of errors is reached. The overall signal flow is shown in Figure 3.8.

A total of 11 configurations of the design containing different numbers of decoder processors have been implemented at a fixed clock frequency of 75 MHz using the default compilation options on Quartus II to maintain consistency of the experiment. To avoid the need to re-compile each configuration for every combination of the parameters such as the predefined E_b/N_0 and target number of detected errors, a JTAG interface [46] system allows for the flexibility of modifying registers after a design is compiled and downloaded onto the FPGA. The BER data gathering process is simplified with this JTAG interface since the values in the registers that store the total error counts and the elapsed clock cycle counts can be read for every combinations of E_b/N_0 and target number of detected errors.

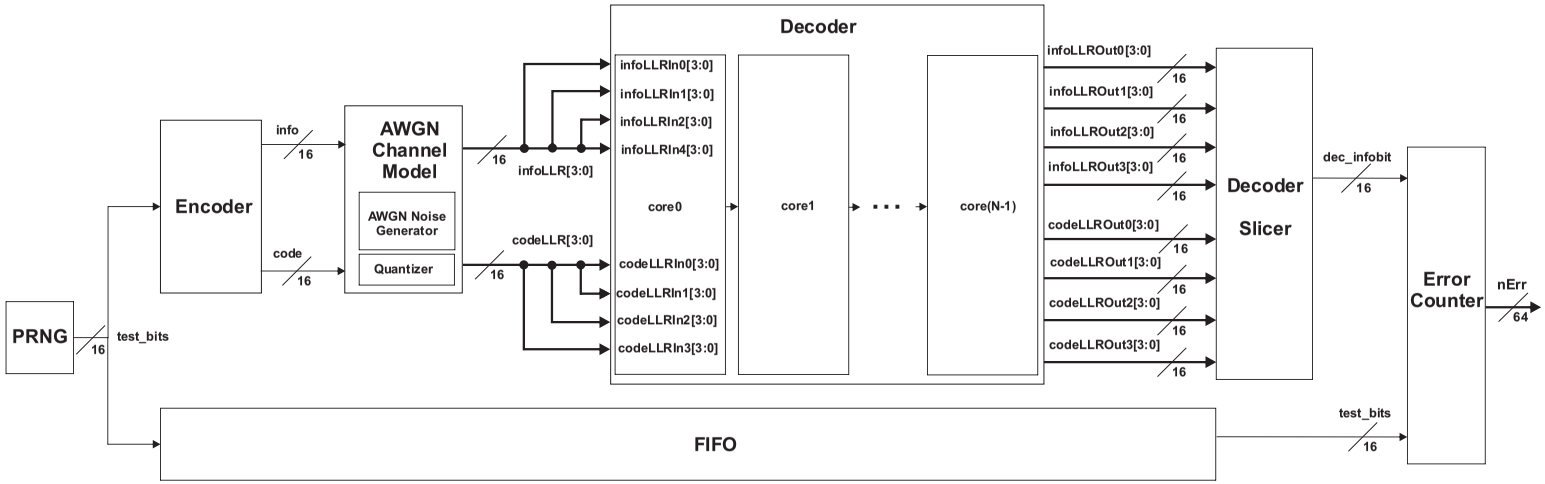


Figure 3.8: *LP4 block diagram. Counter-based control logics are not shown.*

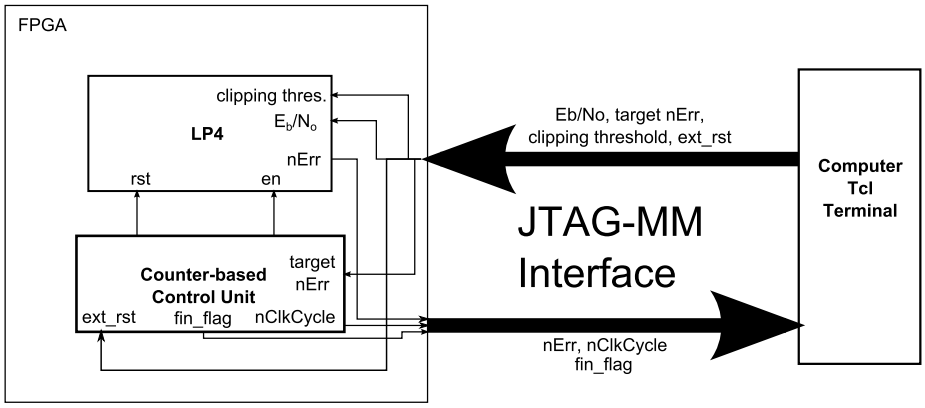


Figure 3.9: *Interaction between LP4 core on the FPGA and computer Tcl terminal*

A simplified version of the interaction between the FPGA and the computer Tool Command Language (Tcl) terminal is shown in Figure 3.9. Programmable registers storing parameters such as E_b/N_0 ratio, target number of errors (*target nErr*), clipping threshold, and a soft reset (*ext_rst*) are mapped to pre-defined memory locations to allow read/write access from the JTAG memory-mapped (JTAG-MM) interface. On the other hand, registers that get updated by the LP4 core and its counter-based control unit such as actual number of errors (*nErr*), number of clock cycles required to detect the target number of errors (*nClkCycle*), and a flag that indicates the completion of data gathering process (*fin_flag*), are mapped to neighbouring memory locations to allow access in a single memory-read operation by the JTAG-MM interface. Doing so ensures that the *nClkCycle* value corresponds exactly to the *nErr* value, which are then used to calculate BER off-chip using Equation 3.2.

$$BER = \frac{nErr}{nClkCycle \times \rho} \tag{3.2}$$

The *fin_flag* register is constantly being monitored by the Tcl terminal, where the values

in*nErr* and *nClkCycle* are read and recorded to a text file when actual *nErr* is equal to the predefined *target nErr*. Details and example code for configuration of this JTAG interface are included in Appendix A.

Detailed power measurements have been conducted using the FPGA implementation presented above with the measurement steps described in Chapter 4. The BER performance and power measurement results obtained from the implementation are then presented and further discussed in Chapter 5.

Chapter 4

Power Measurement

This chapter, first discusses the available methods for determining the power consumption of the LDPC-CC decoder in Section 4.1. Section 4.2 describes the chosen power measurement method. The limitations of the chosen method are then presented in Section 4.3 along with the estimated lower and upper bounds of the DC/DC efficiency presented in Section 4.3.1. Furthermore, Section 4.3.2 described an additional experiment to analyze the efficiency of the DC/DC converter circuitry and presented the relevant results.

4.1 Available Power Measurement Methods

In this section, we describe several power measurement methods available for the characterization of the decoder power on an FPGA board. In order to characterize the power consumption of the LDPC-CC decoder, it is necessary to capture the total power consumption of the Stratix IV FPGA on Altera DE4.

The first power measurement method makes use of PowerPlay, which is an power ana-

lyzer tool provided in The Altera Quartus II Development Software [47]. PowerPlay gives a detailed breakdown of the total power consumed by all the modules in the design using captured signal activities from full post-fit netlist timing simulation. The estimation from PowerPlay power analyzer has an uncertainty of ±15% from actual power consumption when used with accurate design information [48]. From the PowerPlay report, the estimated power consumption of the LDPC-CC decoder can be extracted. Although the signal activities from the full post-fit netlist timing simulation (gate-level simulation) on ModelSim can be used by PowerPlay to give the most accurate power estimation, its long simulation time [47], and the difficulty of incorporating SNR info in the simulation, are two of the main disadvantages of this method [49]. As the size of the design grows, its gate-level simulation time also increases; the requirement for computer resources such as processing power and memory also increases. For the LDPC-CC decoder design with 10 processor cores, signal activity data cannot be generated because ModelSim is unable to start the gate-level simulation due to a memory error. Therefore, PowerPlay Analyzer can only be used to provide an accurate power estimation for our design of LDPC-CC decoder with less than 10 cores.

The manufacturer of the Altera DE4 FPGA board, Terasic, also provides a power measurement tool that makes use of a Nios II processor to facilitate the use of the onboard power measurement circuitry, which consists of two multi-channel differential 24-bit Linear Technology LT2418 delta-sigma analog-to-digital converters (ADC) with sense resistors to measure the small voltage drop across the resistors. This method is capable of reporting the total power consumed by the FPGA chip in real-time. However, in our experiments, a consistent discrepancy of over 90% is observed between the average decoder power based on PowerPlay results from configurations without the Nios II measurement unit and that from the Nios II method, as shown in Table 4.1. One of the possible causes for the observed

Table 4.1: *Comparison of average decoder power results from various methods for various numbers of decoder cores for the $T_s = 192$, $\rho = 16$ rate-1/2 (3, 6) PN-LDPC-CC code with 4-bit LLRs at E_b/N_0 = 2 dB running at a clock frequency of 75 MHz on Altera DE4. Values from columns labeled "PowerPlay", "$P_{Board,raw}$", "$P_{Board,mapped}$" are based on configurations without the Nios II power measurement unit. Three sets of measurements are taken at different time instants for each calculation of P_{Board} value.*

nCore	PowerPlay Estimation [W]	$P_{Board,raw}$[1][W]	$P_{Board,mapped}$[2][W]	Nios II Results[3][W]
1	0.174	0.668	0.393	0.000755
2	0.509	1.09	0.618	0.001264
3	0.956	1.666	0.896	0.001097
4	1.419	2.187	1.118	0.001788
5	1.849	2.693	1.307	0.002074

[1] $P_{Board,raw}$: $P_{Dec} = P_{Total\,Board,raw} - P_{Total\,Board\,w/o\,Dec,raw}$

[2] $P_{Board,mapped}$: $P_{Dec} = P_{Total\,Board,mapped} - P_{Total\,Board\,w/o\,Dec,mapped}$

[3] Nios II Results: $P_{Dec} = P_{Total\,Chip} - P_{Total\,Chip\,w/o\,Dec}$

discrepancy is that PowerPlay estimations are only performed on configurations without the Nios II unit due to the inability to perform gate-level simulation with the inclusion of the Nios II unit. Therefore, an alternate method is employed to provide more reasonable power measurements in order to determine the power consumed by the decoder core in our design, as well as to provide an alternate method for comparison with the Nios II results.

The next power-estimation method involves taking the board power measurements for the design with various numbers of decoder cores, and the board power measurement for the configuration with no decoder core is also recorded as the base case. By subtracting the base case, the decoder power is obtained. Power measurements using this method are also

obtained for our comparison, as shown in Table 4.1. It can be seen that the average decoder power values from this method agree with the PowerPlay estimations with a difference of around 50% for configurations with 4 or 5 cores, which is slightly closer to the ±15% difference reported in [48]. The difference is likely caused by the fact the decoder power values from the column "$P_{Board,raw}$ are based on raw power measurements, which do not take into account the effect of DC/DC converter efficiency. However, the differences for the configurations with 1 to 3 cores are much greater, which are likely to be a result of measurement errors in our method, in addition to the influence from the efficiency of the DC/DC converter circuitry. An additional experiment has been done to model the DC/DC efficiency, which is described in details later in Section 4.3.2. By mapping the board measurement data to the results obtained from the method in Section 4.3.2, the influence from the DC/DC efficiency is reduced, the difference between the mapped data and the PowerPlay estimation results is further reduced. The results reported in Table 4.1 focus on using the PowerPlay estimations as a reference for comparison, which may not be a fair comparison for results obtained from the Nios II method. Therefore, the board measurement method along with the mapping step described in Section 4.3.2 are also used to obtain power measurements from the configurations containing the Nios II measurement unit, which are then summarized in Table 4.2. The "raw" and Nios II data are also plotted in Figure 4.1, where they are then fitted linearly. The "raw" and "mapped" values reported in Table 4.2 are largely different from the ones reported in Table Table 4.2, which were measured from the configurations without Nios II unit. This suggests that the Nios II unit may not have been correctly incorporated into our design. Additionally, while it is suggested that our design may not have been configured properly to work with along with Nios II unit, it is interesting to note that the Nios II reported values are roughly an order of magnitude smaller than the "mapped" values in a consistent manner. This also suggests

Table 4.2: *Comparison of decoder power results from various methods for various numbers of decoder cores for the $T_s = 192$, $\rho = 16$ rate-1/2 (3, 6) PN-LDPC-CC code with 4-bit LLRs at E_b/N_0 = 2 dB running at a clock frequency of 75 MHz on Altera DE4. All measurements are performed on the configurations with the Nios II unit.*

Number of Cores	$P_{Board,raw}$[1][W]	$P_{Board,mapped}$[2][W]	Nios II Results[3][W]
1	0.0106	0.00713	0.000755
2	0.0154	0.0103	0.00126
3	0.0201	0.0135	0.00110
4	0.0236	0.0158	0.00179
5	0.0378	0.0253	0.00207

[1] $P_{Board,raw}$: $P_{Dec} = P_{Total\,Board,raw} - P_{Total\,Board\,w/o\,Dec,raw}$

[2] $P_{Board,mapped}$: $P_{Dec} = P_{Total\,Board,mapped} - P_{Total\,Board\,w/o\,Dec,mapped}$

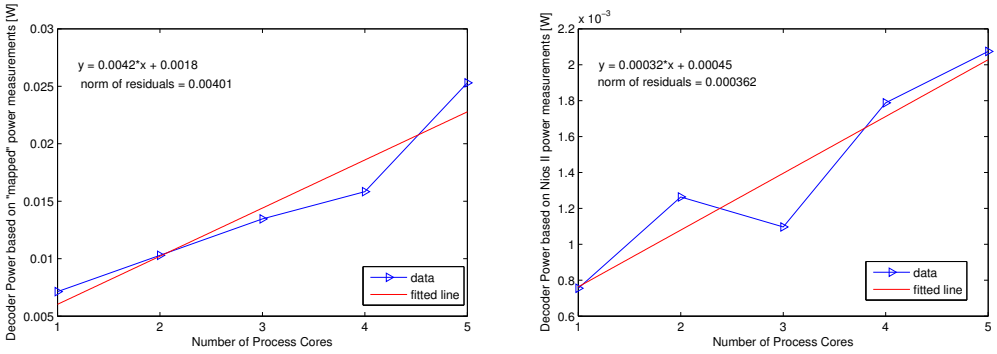
[3] Nios II Results: $P_{Dec} = P_{Total\,Chip} - P_{Total\,Chip\,w/o\,Dec}$

that there may be one or more factors incorrectly specified in the calculation steps used by the Nios II unit.

Since we are unable to configure the Nios II unit to give values that are closer to that of the PowerPlay results at this point, we end up using the board measurement method together with the mapping step described in Section 4.3.2 for the rest of our experiments. A detailed description of this method is provided in the following sections.

4.2 Board Power Measurement Method

In order to determine the power consumption of the decoder module, we have chosen to measure the total power consumed by the entire FPGA board. The power measurement of



(a) *"mapped"* **(b)** *"Nios II*

Figure 4.1: *Decoder power obtained on configurations of 1 to 5 cores with Nios II unit incorporated using board power measurement method with mapping step and Nios II measurement unit*

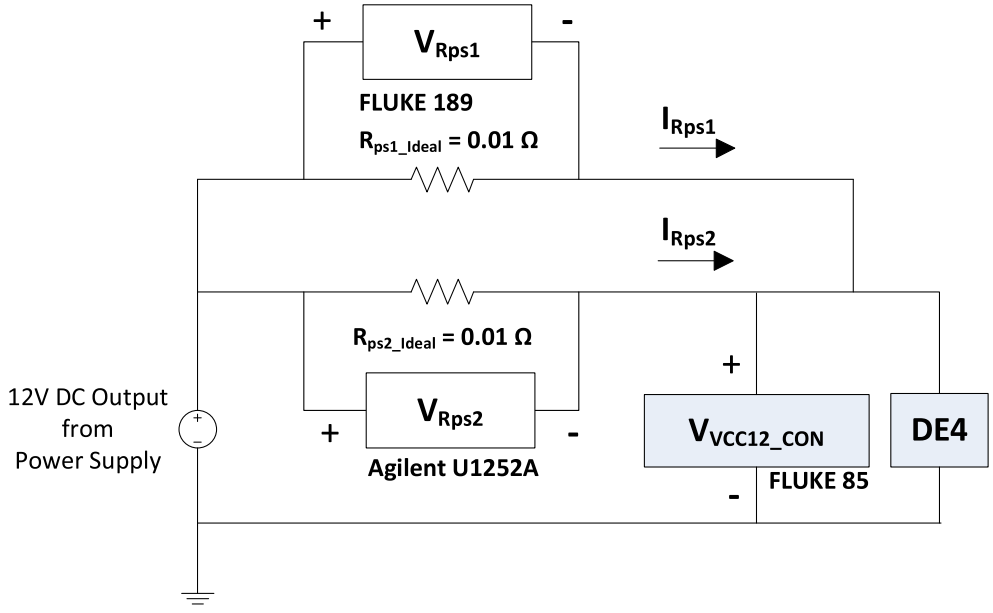


Figure 4.2: *Power Measurement Set-up*

DE4 is made possible by inserting a custom unit that consists of a pair of 8-pin Molex power connectors and necessary wire connections between the Molex receptacle of the AC/DC power supply and the matching Molex header on the FPGA board. The custom unit allows two 0.01-Ω (1% tolerance) current sense resistors to be put in series with the 12-V power rail, where the voltage drops measured across these resistors can then be used to calculate the total current drawn by Altera DE4 board, and an additional digital multimeter is used to measure the input voltage, V_{VCC12_CON} for the FPGA board. Detailed circuit connections are shown in Figure 4.2. The total power consumed by the FPGA board is calculated using Equation (4.1). Three sets of current and voltage readings are captured at different time instants to calculate the total power, and the average of that is considered as the board power consumption for that case.

$$P_{FPGA\,Board} = (I_{Rps1} + I_{Rps2}) \times V_{VCC12\_CON} \tag{4.1}$$

The 12-V supply powers several components and DC-DC converters, which in turn power several more components. All power consumption on the FPGA board is kept constant, except for varying the number of decoder cores. Therefore, the measured power of the "0Core" configuration that contains all the modules except the decoder, $P_{Board,E_b/N_0,0Core}$, for every different E_b/N_0 value can be used as the base case for that particular E_b/N_0, which is then subtracted from the corresponding total measured board power, $P_{Board,E_b/N_0,nCore}$, to calculate the decoder power, $P_{Dec,E_b/N_0,nCore}$, as shown in Equation (4.2). Since the clock frequency for all our experiment is fixed at 75 MHz, the resulting coded throughput of all configurations is 2.4 Gbit/s, using Equation (3.1). The energy-per-coded-bit is calculated using Equation (4.3).

$$P_{Dec,E_b/N_0,nCore} = P_{Board,E_b/N_0,nCore} - P_{Board,E_b/N_0,0Core} \tag{4.2}$$

$$Energy - per - coded - bit_{E_b/N_0,nCore} = \frac{P_{Dec,E_b/N_0,nCore}}{CodedThroughput} \tag{4.3}$$

4.3 DC/DC Converter Efficiency

4.3.1 Estimation Based on Data Sheet

The power measurement setup described in Section 4.2 has assumed an efficiency of 100% for the DC/DC converter on Altera DE4. Based on the detailed schematic of DE4 [50] given by their manufacturer, Terasic, a 3-phase buck converter with three LTM4601 regulators is used to convert the ideal 12-V supply voltage down to a 0.9-V supply voltage for the FPGA core. This 3-phase buck converter topology is capable of providing a maximum load current of 36 A at an output voltage of 0.9 V. Under ideal condition, the output current for each phase, LTM4601, is the total output current divided by 3, the number of phases in this particular setup.

To obtain a rough estimate for upper and lower bounds of the efficiency of the LTM4601, the currents drawn by the FPGA core in both cases need to be calculated. Among all the board power measurements data obtained for the LP4 designs at 110 configurations with 0 to 10 cores at SNR of 1 to 10 dB, the maximum measured board power is 14.97 W, and the minimum measured power is 9.69 W. Both measurements contain the total power consumed by all the onboard components powered by the 12-V supply. In order to isolate the power consumed by the FPGA core, the total power for the base case, where the FPGA core

only contains the programming for the control logic of the onboard PLL, is also measured. This is to ensure that all the other components on the DE4 board are operating under the same conditions as the cases where the other other power measurements of the design are taken. The FPGA core power can then be obtained by subtracting the power measured in the base case from the maximum and minimum measured power. The calculated values for the maximum and minimum FPGA core power here are considered to be the inputs of the DC/DC converter circuitry. To calculate the maximum and minimum currents drawn by the FPGA core, which are on the output side of the DC/DC converter, the efficiency of converter must be again assumed to be 100%, in which case the output power of the DC/DC converter is equal to the calculated input power described above.

Using the assumption above, the maximum and minimum currents drawn by the FPGA core can then be obtained by dividing the respective calculated power by the ideal FPGA core supply voltage, 0.9 V. Using the calculated current values for both cases, an estimate for their respective efficiency can be obtained from Figure 4.3, which shows the efficiency curve from the LTM4601 data sheet [2]. However, since the efficiency curve for 0.9-V output voltage is not provided in Figure 4.3, the efficiency for the 0.9-V output is estimated by subtracting 40% of the difference between the efficiencies for that of 0.6-V and 1.2-V from the efficiency value for 1.2-V output voltage. The 40% difference is chosen based on the overall decreasing trend in Figure 4.3. The measured power consumptions and respective estimated efficiency are summarized in Table 4.3. Since the load currents are calculated by assuming the efficiency is 100% and the output voltage is fixed at 0.9 V, the actual load currents in the two cases are expected to be lower than the calculated values, which means the reported efficiencies in the Table 4.3 are also over-eastimated.

Using the estimation method above, it can be concluded that the efficiency of the DC/DC converter circuitry is below 50% for the case with the minimum measured board

power, and is upper-bounded by 73% for the case with maximum measured board power. However, this only gives us a range for the DC/DC efficiency from data sheet, which does not provide the DC/DC efficiency for each different value of the output current. In order to provide a more accurate way to account for the effect of the DC/DC efficiency on our power measurement data, we decided to perform an additional experiment, which is described in Section 4.3.2.

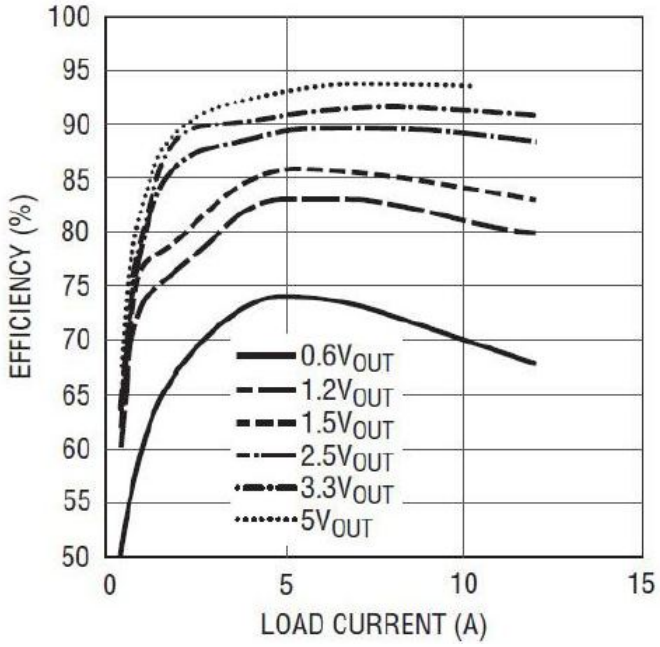


Figure 4.3: *Efficiency versus Load Current with 12-V Input for LTM4601 Regulator. (Taken from the LTM4601 data sheet [2])*

4.3.2 Estimation Based on Experimental Results

From the maximum and minimum calculated FPGA core power values shown in Table 4.3, the equivalent resistance values can be calculated by assuming the DC/DC efficiency to be

Table 4.3: *Summary of measured board power result, deduced output current and efficiency for each LTM4601 under various conditions. The FPGA core power is calculated by subtracting the measured power for the base case with having only PLL control logic on the FPGA core from the measured board power for the cases with the full design on the FPGA core.*

	Base Case (with only PLL control logic)	P_{Max} Measured	P_{Min} Measured
Total Board Power	9.142 W	14.971 W	9.667 W
FPGA Core Power	-	5.829 W	0.525 W
Estimated Current Drawn by FPGA Core	-	6.477 A	0.583 A
Output Current of Each LTM4601	-	2.159 A	0.194 A
Estimated Efficiency	-	73%	<50%

100% and the output voltage to be constant at 0.9 V. The calculated values are shown in Table 4.4. Using the calculated resistance values, a range of resistence values are selected between 0.1 Ω and 1.6 Ω for our DC/DC efficiency experiment.

Table 4.4: *Calculated equivalent resistance for the FPGA core power listed in Table 4.3 with the assumption that V_{VCC0P9} remains constant at 0.9 V*

	Maximum	Minimum
Calculated FPGA Core Power, P [W]	5.829	0.525
Calculated Resistance [Ω]	0.139	1.543

In our experiment, the FPGA chip is programmed with a configuration containing the minimum amount of logic so the total input power to the DC/DC converter circuitry is kept at a minimum to allow wider range of dissipated power of the varying resistance to

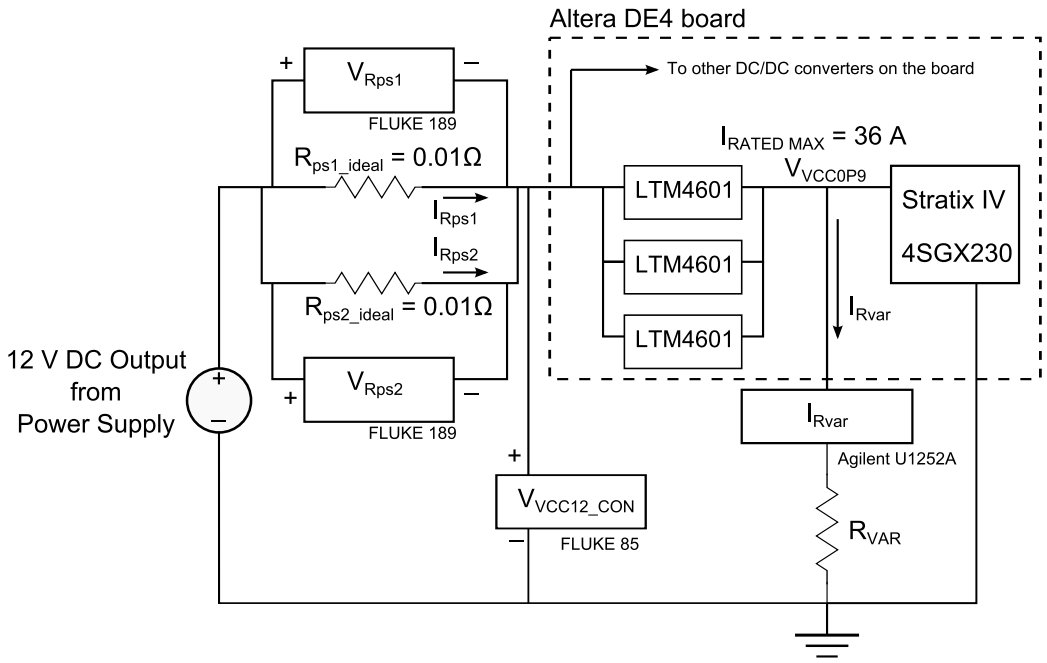


Figure 4.4: *Schematic for DC/DC efficiency experiment*

be observed. A wire is soldered to the V_{VCC0P9} node to allow a varying resistance to be put in parallel with the FPGA chip, and the current going through the varying resistance is measured with an ammeter put in series with the varying resistance. The experiment set-up is depicted in Figure 4.4. In our experiments, while we are aware that V_{VCC12_CON} is connected to other DC/DC converters that power other components on the board, no variation in these other loads was detected. It is also assumed that the dissipated power on the varying resistance is equivalent to the difference between the power consumed by the logic of the design programmed on the FPGA and the power consumed by the FPGA with minimum configuration, in which case, the latter is assumed to remain constant. Based on this consumption, the power consumed by the varying resistance is plotted in Figure 4.5 against the input power, which is simultaneously measured on the input end of the DC/DC converter circuitry as the resistance is varied from 0.1 Ω to 1.6 Ω. The data plotted in

Figure 4.5 is then curve-fitted to Equation (4.4), with a maximum percent error of 3.14%. While a quadratic fitting is not physically meaningful here, it does provide an analytic approximation so long as it is not used for extrapolation.

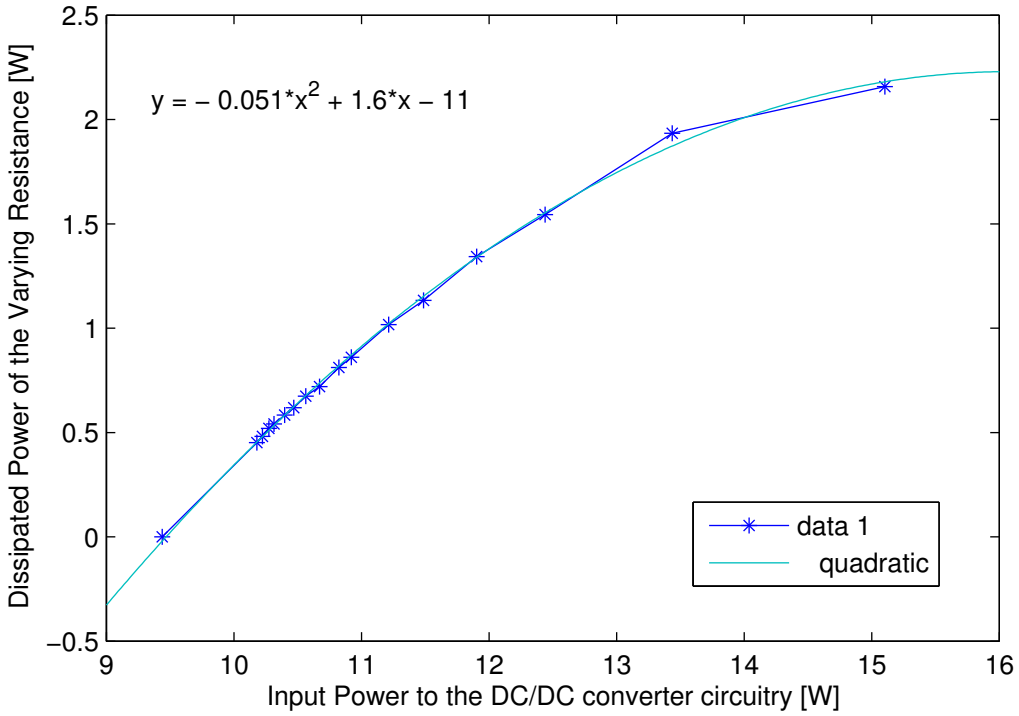


Figure 4.5: *Input and output power measured for the DC/DC converter circuitry on the Altera DE4 using the setup denoted in Figure 4.4*

$$y = -0.051 \times x^2 + 1.6 \times x - 11 \tag{4.4}$$

The measurements from this experiment has indirectly depicted the efficiency of the DC/DC converter circuitry. By mapping the existing set of 110 power measurement values described in Section 3.4 using Equation (4.4), the resulting values have then taken into account the efficiency of the DC/DC converter circuitry. An example of the mapped values

of decoder power with different number of cores are shown in Table 4.5 to be compared with the PowerPlay estimated results shown in Table 4.1. By taking the efficiency of the DC/DC converter circuitry into account, the percentage difference calculated with respect to the PowerPlay estimation for the "4Core" and "5Core"' configurations are reduced from around 40% as mentioned in Section 4.1 to 22%.

Table 4.5: *Comparison of mapped power measurement values with estimated values from PowerPlay of decoder power of different number of processor cores for the $T_s = 192$, $\rho = 16$ rate-1/2 (3, 6) PN-LDPC-CC code with 4-bit LLRs at $E_b/N_0 = 2$ dB running at a clock frequency of 75 MHz on Altera DE4*

nCore	Mapped Power Measurements [W]	PowerPlay Estimation [W]	% Difference (w.r.t. PowerPlay Values)
3	0.965	0.956	0.85%
4	1.212	1.420	-14.61%
5	1.449	1.849	-21.66%

The input voltage V_{VCC0P9} is also measured again at the the minimum and maximum parallel resistances used in the experiment above, and it turns out that the voltage does not remain constant when the resistance is varied from 0.1 Ω to 1.6 Ω, as shown in Table 4.6. This is most likely due to the finite output resistance of the AC power adapter, which is the external power supply and the DC/DC converter circuitry in this case. While this observation is indeed different from our earlier assumption made on the value of V_{VCC0P9} for the calculation of the equivalent resistance, it does not seem to have affected our results plotted in Figure 4.5.

Table 4.6: *Measured input voltage for FPGA chip at various parallel resistances*

Resistance [Ω]	Start-up (Non-configured)	Configured
0.1	0.6386 V	0.641 V
1.6	0.879 V	0.8786 V

4.4 Chapter Summary

In this chapter, we have discussed the available power measurement methods with the chosen FPGA board and explained why the chosen method has been employed. A further experiment has been developed and performed to model the efficiency of the DC/DC converter circuitry, and its results are applied on the existing set of power measurement values. Further discussion on the resulting values are presented in Chapter 5.

Chapter 5

Measurement Results and Discussion

The performance of our LDPC-CC decoder is evaluated based on its power consumption, energy-per-coded-bit, BER performance, area, throughput, and clipping threshold, c_{th}, which is a threshold value chosen to clip the received channel values during quantization so that the received values are uniformly quantized into sign-magnitude representation in the range [-c_{th}, c_{th}]. BER performance depends on several factors: the LDPC code that the decoder is based on, the number of iterations (each implemented as a physical decoder processor core in the case of LDPC-CC), the bit-precision of LLRs, as well as E_b/N_0, which is the normalized signal-to-noise ratio (SNR) per bit applied in the channel.

5.1 BER Performance

As previously mentioned in Section 3.4, our design is based on the $T_s = 192$, $\rho = 16$ rate-1/2 (3, 6) PN-LDPC-CC code, where each decoder core has a constant bit-width of 4 for the LLRs. A constant clipping threshold, c_{th}, of 1.33 is used in the quantization operation

for all configurations. The resulting FPGA implementations with 1 to 10 decoder cores all run at a constant clock frequency of 75 MHz and a constant coded throughput of 2.4 GHz, where 75MHz is a common maximum frequency achieved by different configurations of the implementation. The BER performance of the FPGA implementation is plotted in Figure 5.1, based on the measurement results from the DE4 board. It can be seen that the improvement in BER performance gained from additional cores gradually decreases. At an E_b/N_0 of 6 dB, the "6Core" configuration is able to achieve an BER of 10^{-10}. If the number of cores is increased to 9 or 10, the same BER performance can be achieved at a lower E_b/N_0 of 5 dB. Table 5.1 summarizes the maximum reported clock frequencies on Quartus II and the corresponding calculated coded throughputs for configurations with 1 to 10 decoder cores. Configurations with 1 to 3 cores have slightly higher maximum clock frequency, while configurations with 4 or more cores all have maximum clock frequency around 80 MHz.

5.2 Power Measurement Results

The decoder power consumption and energy-per-coded-bit for each configuration is plotted against E_b/N_0 in Figure 5.2 and against the number of decoder cores in Figure 5.3. For both figures, the top plot, labeled *"Raw"*, is based on the raw measurements obtained using the method described in Section 4.2, while the bottom plot, labeled *"Mapped"*, is based on the mapped data, which has taken into account the effect of the efficiency of the DC/DC converter circuitry using the method stated in Section 4.3.2. The difference between the two plots in Figure 5.2 and 5.3 is caused by the efficiency of the DC/DC converter circuitry, where the raw data has assumed an efficiency of 100%, while the processed values in the bottom plot are mapped to the experimental values determined in Section 4.3.2, which

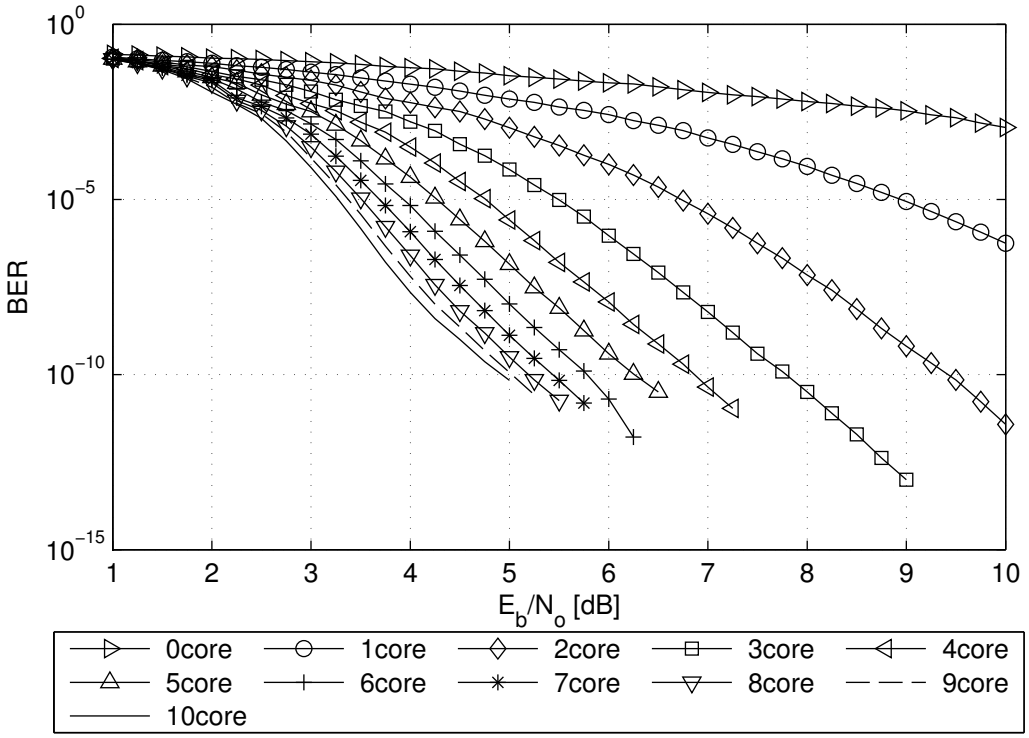


Figure 5.1: *BER for various numbers of decoder cores for the $T_s = 192$, $\rho = 16$ rate-1/2 (3, 6) PN-LDPC-CC code with 4-bit LLRs at various E_b/N_0 obtained from measurements on DE4 based a target number of errors of 1000 based on a BPSK-based AWGN channel.*

Table 5.1: *Maximum achievable clock frequency and maximum coded throughput for configurations with 1 to 10 decoder processor cores based on reported values for "slow 900mV 85C model" on Quartus II using default compilation options for an assigned clock constraint of 75 MHz*

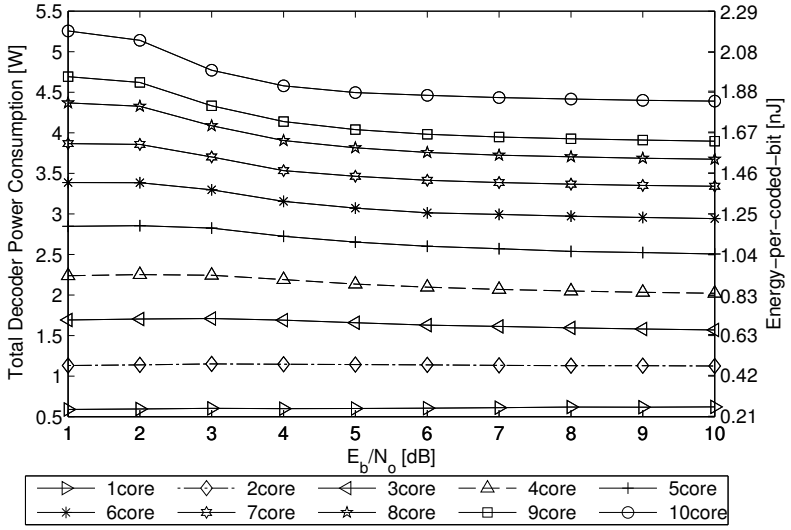
nCore	f_{max} [MHz]	Max. Coded Throughput [Gbit/s]
1	85.5	2.74
2	83.1	2.66
3	83.5	2.67
4	79.6	2.55
5	79.3	2.54
6	80.2	2.57
7	79.3	2.54
8	80.3	2.57
9	79.6	2.55
10	79.6	2.55

model efficiency.

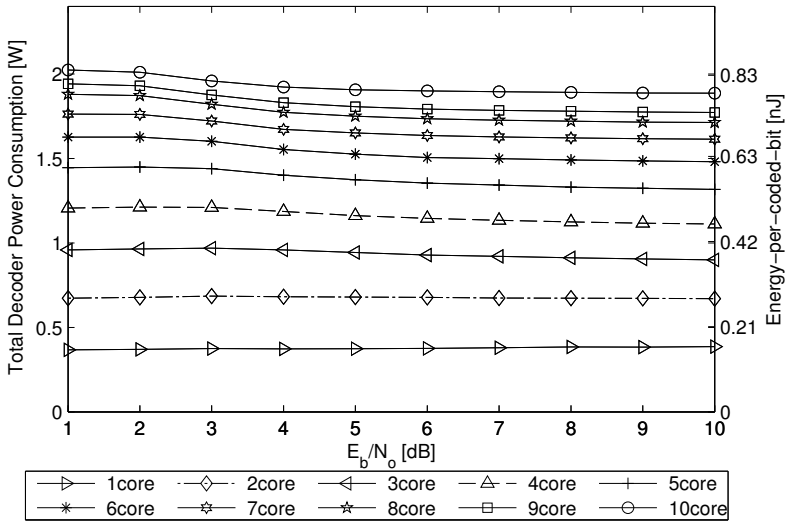
On both plots in Figure 5.2, the power consumed by the decoder and the energy-per-coded-bit gradually decrease as E_b/N_0 increases, as has been observed for other LDPC decoders in the literature [49, 51]. In particular, in Figure 5.2a, the decrease in decoder power and energy-per-coded-bit caused by the increase in E_b/N_0 are more obvious for configuration with 4 or more processor cores until E_b/N_0 reaches 6 dB. While the same trend can be observed in Figure 5.2b, the effect of E_b/N_0 is less significant.

In Figure 5.3, the decoder power and energy-per-coded-bit show a close-to-linear relationship with the number of processor cores in raw data, while a gradually increasing trend can be observed in the plot with mapped values. The later decoding stages, implemented as physical copies of decoder cores, in the decoding chain, can be said to consume less power than the earlier decoding stages. This implies that less work is being done in the later stages of the decoding process, which also agrees with the trend of the BER performance observed in Figure 5.1, where improvement in BER performance is less significant for configurations with 6 or more cores.

To further analyze the relationship between decoder power consumption, energy-per-coded-bit, E_b/N_0, number of processor cores, the incremental decoder power and energy-per-coded-bit values are obtained by subtracting the decoder power and energy-per-coded-bit values for the $(N-1)$-core configuration from the decoder power and energy-per-coded-bit at N-core configuration, and are then plotted against E_b/N_0 in Figure 5.4 and against number of cores in Figure 5.5. In Figure 5.4a, the incremental decoder power and energy-per-coded-bit gradually decrease for every dB of increase in E_b/N_0 with the exception of the values for the "1Core" and "10Core" configurations. The same trend appears in Figure 5.4b. While the magnitude of decrease due to the increase in E_b/N_0 is less obvious after considering the effect of DC/DC efficiency, the difference between the behaviour from

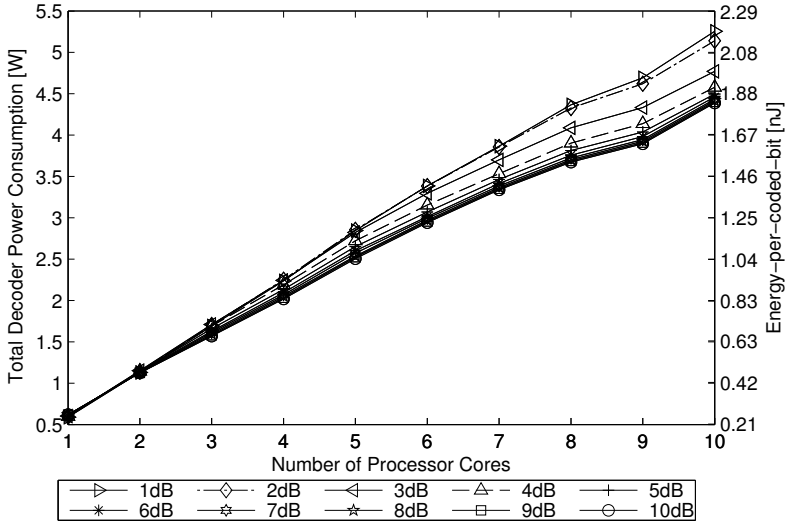


(a) *Raw*

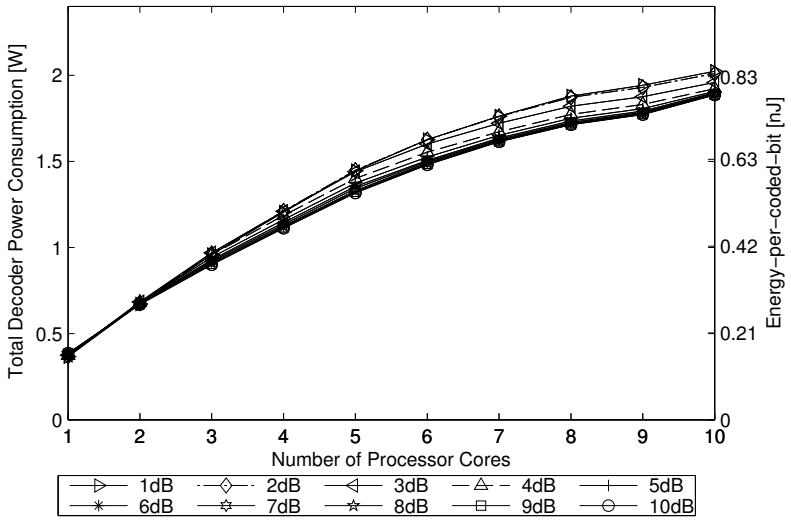


(b) *Mapped*

Figure 5.2: *Average measured decoder power and energy-per-coded-bit at various E_b/N_0 for various numbers of decoder cores on DE4 with 4-bit LLRs at a 75MHz clock with coded throughput of 2.4 Gbit/s for the $T_s = 192$, $\rho = 16$, rate-1/2 (3,6) PN-LDPC-CC code based on three sets of measurements for each data point.*

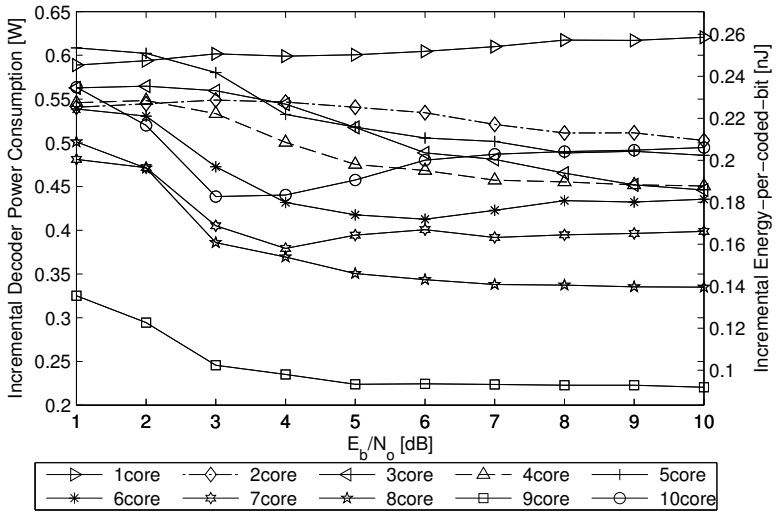


(a) *Raw*

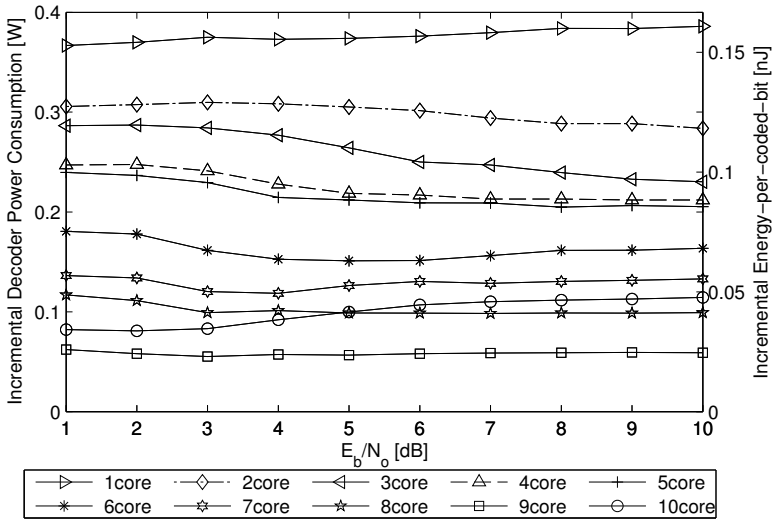


(b) *Mapped*

Figure 5.3: *Average measured decoder power and energy-per-coded-bit at various E_b/N_0 for various numbers of decoder cores on DE4 with 4-bit LLRs at a 75MHz clock with coded throughput of 2.4 Gbit/s for the $T_s = 192$, $\rho = 16$, rate-1/2 (3,6) PN-LDPC-CC code based on three sets of measurements for each data point.*



(a) *Raw*



(b) *Mapped*

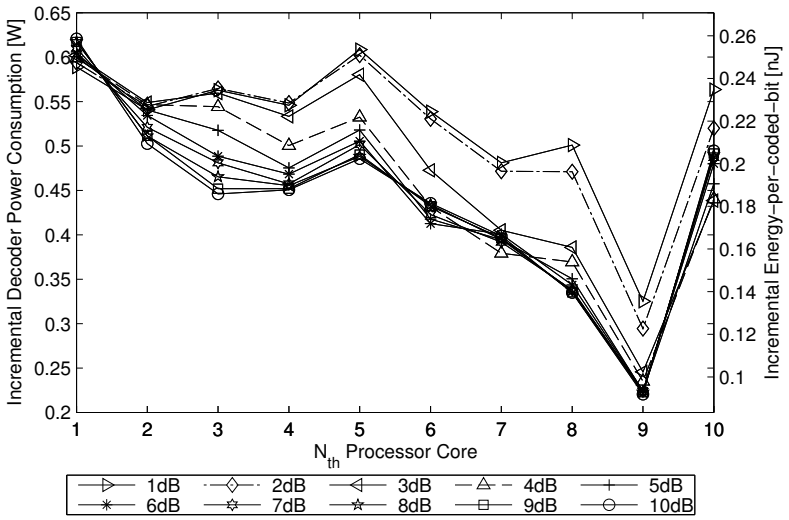
Figure 5.4: *Incremental measured decoder power and energy-per-coded-bit for every dB of increase in E_b/N_0 for various numbers of decoder cores on DE4 with 4-bit LLRs at a 75MHz clock with coded throughput of 2.4 Gbit/s for the $T_s = 192$, $\rho = 16$, rate-1/2 (3,6) PN-LDPC-CC code.*

the "1Core" and "10Core" configurations and the rest are much easier to see in Figure 5.4b. In both of the plots in Figure 5.5, we observe a generally decreasing trend in the incremental decoder power and energy-per-coded-bit for each additional processor core, which again agrees with our conclusion drawn from Figure 5.3 above.

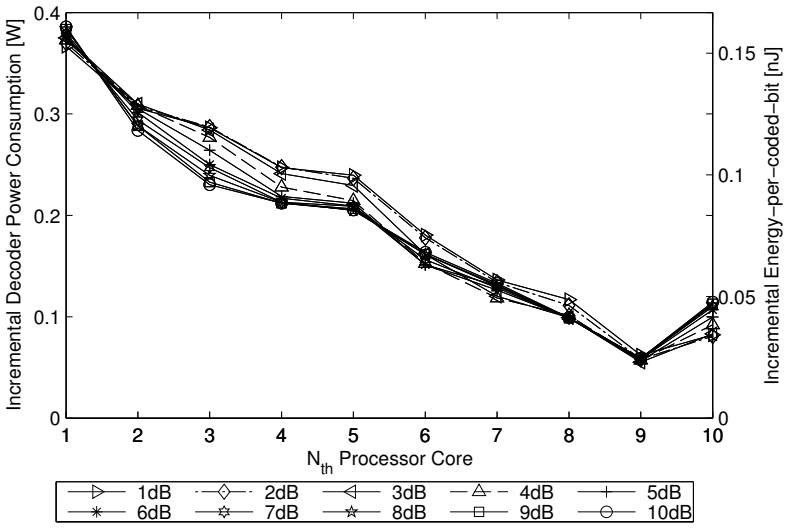
In Figure 5.4, the inconsistent behaviour of "1Core" is a result of the measurement errors for the "1Core" configuration as well as the "0Core" base case, which is also used in the calculation of the incremental power and energy-per-coded-bit for the "1Core" case. In Figure 5.4, the behaviour of the "10Core" configuration is mainly caused by the measurement data for the "9Core" configuration, the inconsistent behaviour of the "9Core"' case can be better-observed in Figure 5.5. The cause of the "9Core" behaviour can be mainly explained by the inherent randomness within the CAD methodologies used in Quartus II for placement and routing. In Figure 5.4, although the overall incremental values for the "9Core" are lower than that of the other configurations, the curve itself still maintains the same decreasing trend as the other curves when E_b/N_0 is increased.

5.3 Logic Utilization

In Figure 5.5, the full and incremental logic utilization for all configurations on the Altera DE4 are presented. For the 9^{th} core, the incremental power from the 8^{th} core is significantly lower than that of the other cores. Furthermore, it can be seen from Figure 5.6 that the increase in incremental logic utilization at 9^{th} core from the 8^{th} core is significantly greater than that of the others. The aforementioned behaviours of the "9Core" configuration seem to suggest that a higher logic utilization can result in lower decoder power and energy-per-coded-bit, but they could also be just the result of the inherent randomness in the CAD algorithms used in Quartus II. We verified this by performing additional compilations of



(a) *Raw*



(b) *Mapped*

Figure 5.5: *Incremental measured decoder power and energy-per-coded-bit for each additional core at various E_b/N_0 on DE4 with 4-bit LLRs at a 75MHz clock with coded throughput of 2.4 Gbit/s for the $T_s = 192$, $\rho = 16$, rate-1/2 (3,6) PN-LDPC-CC code.*

the “8core”, “9core”, and “10core” configurations with different random number seeds specified at the Fitter stage on Quartus II. The resulting incremental logic utilization and the incremental decoder power values varied by 11% as shown in Figure 5.7, and 0.362 W respectively with different seeds based on the raw data, and they have not shown any consistent trend in their relationship, which confirms that the earlier observed relationship is most likely caused by the randomness in the synthesis tools. This experiment shows that results (area in the form of logic utilization for the case for FPGA) are dependent on the random seed generators used during the Fitter stage, but the variation in this experiment does not fully explain the variation around the results from different runs of the “9Core” design in Figure 5.7.

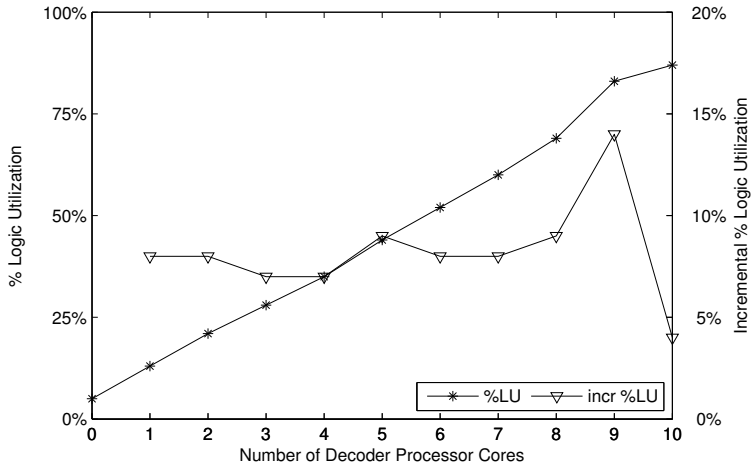


Figure 5.6: *Logic utilization and incremental logic utilization for various configurations with different numbers of decoder processor cores*

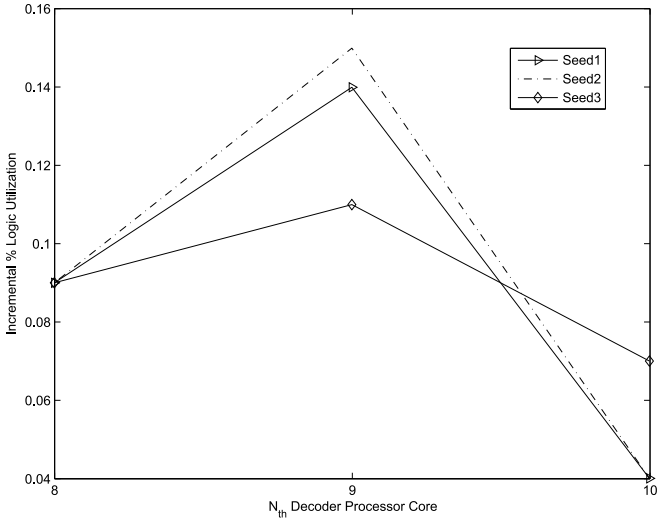


Figure 5.7: *Incremental logic utilization for various configurations with 8, 9, 10 decoder processor cores*

5.4 Chapter Summary

We also attempted to measure the encoder power consumption. However, using our power measurement method, the configuration without an encoder turns out to consume slightly more power than the configuration with an encoder. We conclude that the power consumption of the encoder module is minimal in comparison to the FPGA power fluctuations caused by the randomness in the synthesis tools. We will further quantify and explore this behaviour in future work.

To summarize, we have demonstrated the chosen LDPC-CC code is capable of achieving a BER of 10^{-10} with 6 or more decoder cores at E_b/N_0 of 6 dB. The trend of increase in decoder power and energy-per-coded-bit from the increase of process cores slows down for later cores in the decoding chain, which identifies less decoding work is being performed in those cores. For 10^{-10} BER performance, having an additional core on the“ 9Core” config-

uration would cost an additional of 0.457 W based on the raw data and the improvement in BER performance is less than 5×10^{-11}. An increase of 0.25 dB or 0.50 dB in E_b/N_0 would be required for the "8Core", "7Core" configurations to achieve the same BER performance as well as saving 0.224 W, and 0.350 W respectively, compared to the "9Core" configuration.

In Table 2.2, the result from this work with 9 decoder cores at a E_b/N_0 of 4.25 dB is summarized along with the results of other existing LDPC-BC decoders described in Section 2.4.2. Our work achieves a higher throughput than both implementations from [34] on a lower clock frequency. The average incremental energy-per-coded-bit for our design for 7 to 10 decoder cores at E_b/N_0 dB is approximately 0.15 nJ per additional core. Using this approximated value, our design with 15 cores at an E_b/N_0 between 4 and 5 dB consumes around 2.61 nJ per-coded-bit, which is around half of what is consumed by the first implementation from [34].

A further exploration on the relationship between bit-precision for each core, clipping threshold and resulting BER performance, decoder power, energy-per-coded-bit using the 6-core configuration is included in Chapter 6.

Chapter 6

Power-Driven Architectural Exploration

There are many factors that influence an LDPC decoder's performance, which can be evaluated by its area (in the form of logic utilization for FPGA), BER performance, and power consumption as well as energy-per-bit. Clipping thresholds and bit width of LLRs are two of the factors that have been left as constant in the reference design described in Section 3.4. Higher bit-precision in LLR is expected to produce better BER performance, but the improvement comes at the cost of extra hardware in the form of higher logic utilization for FPGA, and higher power consumption. It has been also suggested that optimizing clipping threshold, c_{th}, can lead to improvement in BER performance [52]. Our goal is to examine the trade-offs between power consumption, logic utilization and bit-precision, clipping threshold. Instead of performing an exhaustive search on all possible combinations of clipping threshold and bit resolution, the effect of each parameter is examined separately.

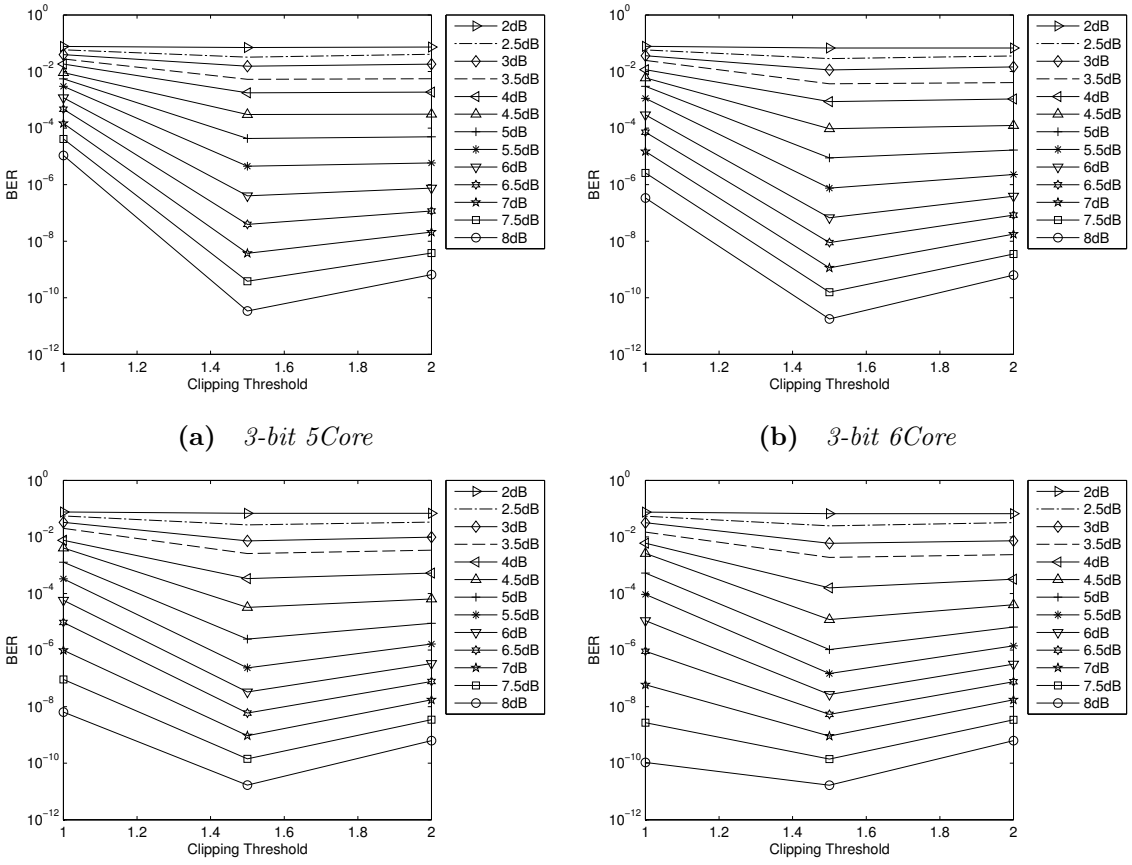
The effect of clipping thresholds on BER performance is first examined in Section 6.1.

Then the rate of saturation events for each core in the 6-core pipelined decoding chain is then analyzed in Section 6.2. Based on the results from the rate of saturation events, a test case with varying LLR width for each core in a "6Core" configuration is compared with the constant 4-bit, 5-bit, and 6-bit implementations with same number of cores. Then the results of the aforementioned experiment are used to analyze the effects of clipping thresholds, bit-resolution for each decoder core on the power consumption, energy-per-coded-bit, and BER performance of the decoder in Section 6.3.

6.1 Clipping Threshold versus BER Performance

A slightly modified version of the reference design, which is called LP4a, is developed to further simplify the data gathering process. The channel output LLRs, and all the intermediate output LLRs from each processor core are fed in their respective hard-decision decoders (decoder slicer), and the outputs of those decoder slicers are compared with the original copy of their corresponding test data bits. The numbers of detected errors for the output from different processor cores are monitored by different counters, and stored in different programmable registers to allow read access from the JTAG-MM interface. Using this modified model reduces the amount of required individual compilations, and the time required for data collection for determining BER performance. However, this version is not suitable for power measurement application since the compiled design contains all the additional logic to achieve separate monitoring of the numbers of errors as well as all the decoder cores.

The architecture of LP4a is employed for a design with constant bit-width of 3, and a design with constant bit-width of 4. The 3-bit version and the 4-bit version of the designs are swept with a range of clipping thresholds, and the BER performances with 5 to 8 cores



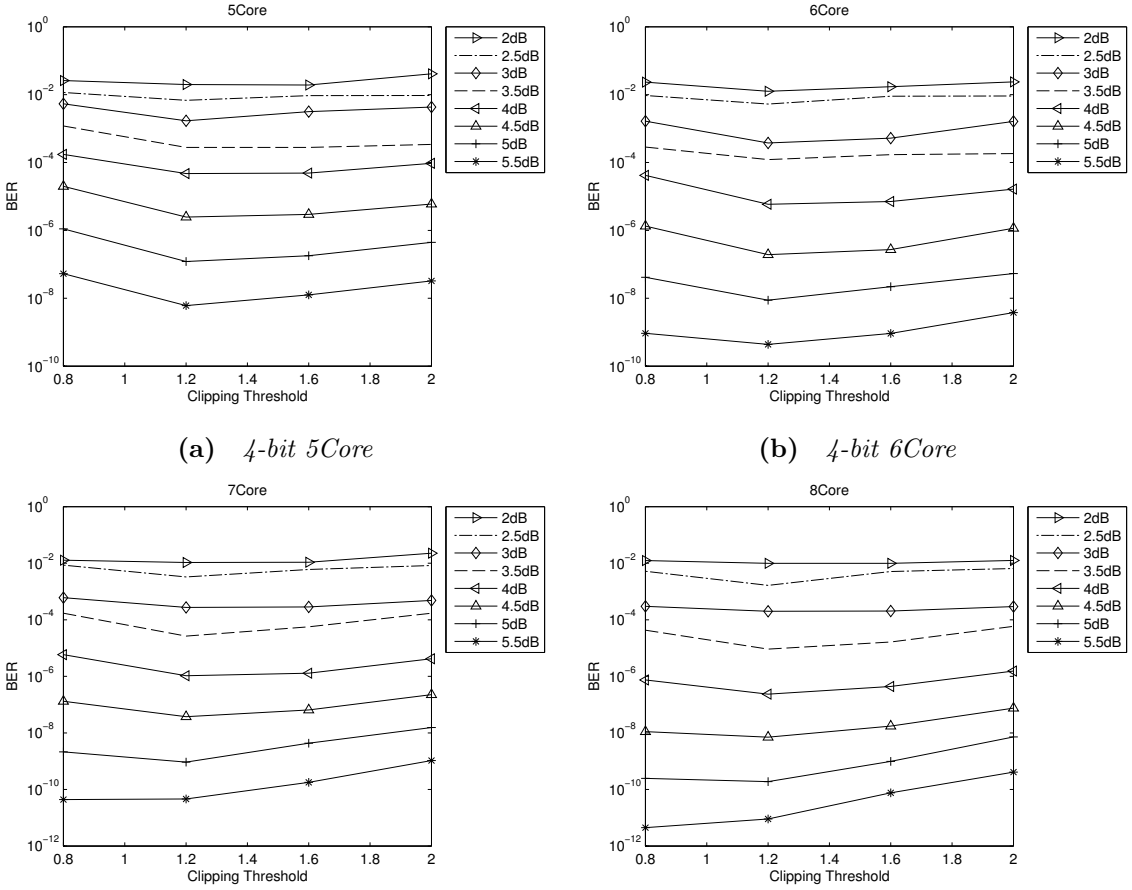
(a) *3-bit 5Core* **(b)** *3-bit 6Core*

(c) *3-bit 7Core* **(d)** *3-bit 8Core*

Figure 6.1: *Information BER at E_b/N_0 of 2 dB to 8 dB for configurations with 5 to 8 decoder cores on DE4 with 3-bit LLRs at a 75MHz clock with coded throughput of 2.4 Gbit/s for the $T_s = 192$, $\rho = 16$, rate-1/2 (3,6) PN-LDPC-CC code using clipping thresholds from 1 to 2 during quantization.*

are captured and plotted in Figure 6.1 and 6.2. The purpose of this is to obtain a general idea on the relationship between clipping threshold and BER performances. The case with constant bit width of 3 bits is also considered here for reference purpose.

As shown in Figure 6.1, as the number of cores increases, the BER performance of



(a) *4-bit 5Core* **(b)** *4-bit 6Core*

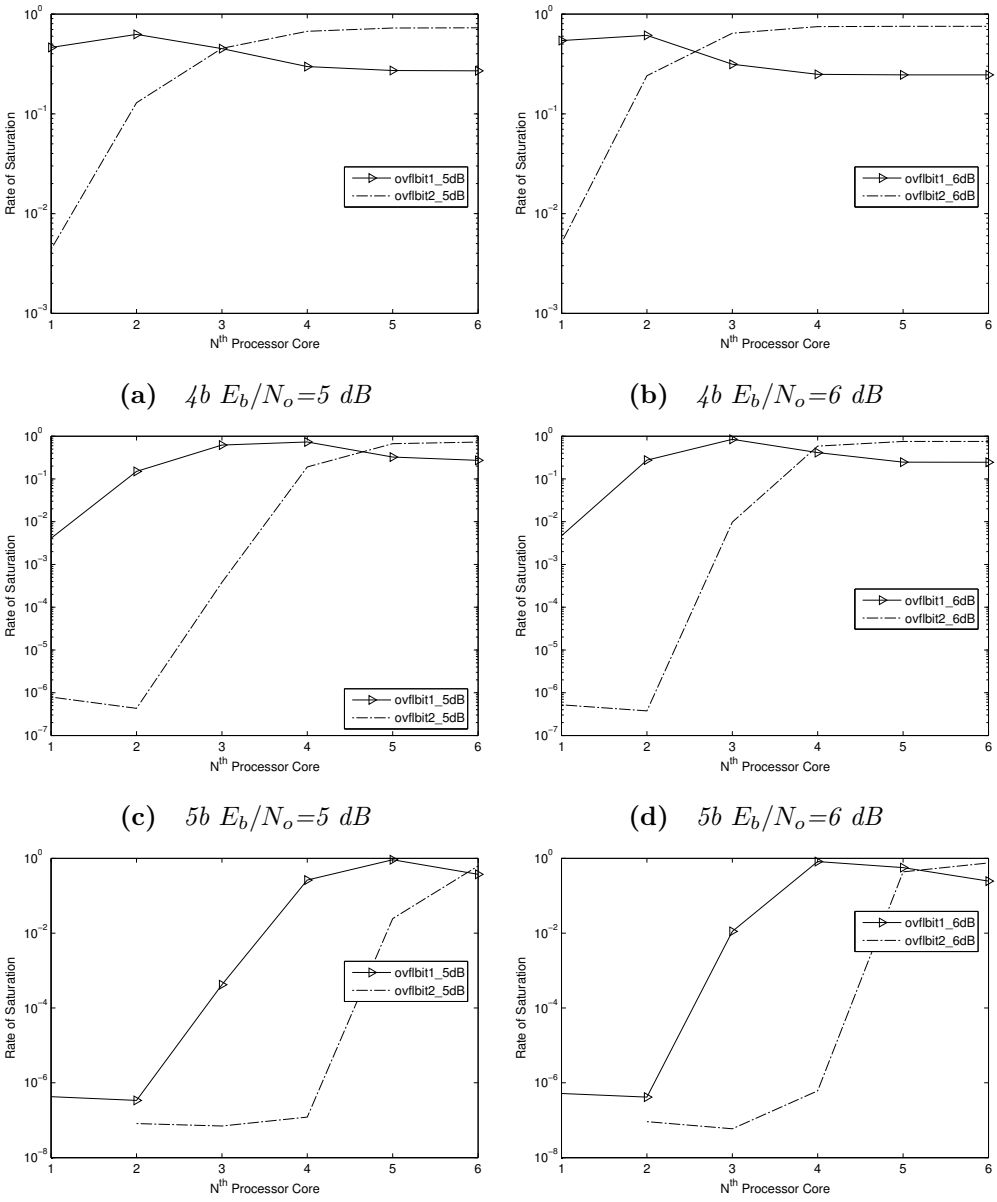
(c) *4-bit 7Core* **(d)** *4-bit 8Core*

Figure 6.2: *Information BER at E_b/N_0 of 2 dB to 5.5 dB for configurations with 5 to 8 decoder cores on DE4 with 4-bit LLRs at a 75MHz clock with coded throughput of 2.4 Gbit/s for the $T_s = 192$, $\rho = 16$, rate-1/2 (3,6) PN-LDPC-CC code using clipping thresholds from 0.8 to 2 during quantization.*

designs with increasing number of cores at the same E_b/N_0 gradually becomes less sensitive to the change in clipping threshold. It can be also seen that the optimal value of clipping threshold in each sub-figures in Figure 6.1 decreases at the same E_b/N_0 as the number of cores increases. The increase in E_b/N_0 provides a consistent improvement in BER performance as expected throughout the change in clipping thresholds. The effect of clipping thresholds on BER performance at higher E_b/N_0 at lower number of cores is more significant as their curves are deeper. In Figure 6.2, the same aforementioned characteristics for Figure 6.2 remain applicable. The shape of BER performance curves for the 4-bit implementation is much less sensitive to the change in clipping thresholds as the curves are shallower than that of their 3-bit versions.

6.2 Rate of LLR Saturation

Before moving to the evaluation of the effect of bit-width on BER performance, we have chosen to first evaluate the rate of LLR saturation in each decoder cores. Rate of saturation is defined as the fraction of calculations that lead to a saturation event in variable nodes in each decoder core. A saturation event is defined as the act of saturating the magnitude of the LLRs if overflow is detected in the next two higher bit-positions during the summation step in variable node. For example, in our 4-bit implementation where bit3 of the LLR represents the sign, and bit2 to bit0 are used to represent the magnitude bits, two extra bits, bit3 and bit4 are inserted between the sign bit and the magnitude bits for the summation step inside the variable node, and the sign bit of the sum is temporarily stored in bit5. If bit3 and bit4 become 1's during the summation stage, then the output LLR from the variable node becomes a concatenation of the sign bit of the sum and three magnitude bits of 1's to represent the maximum magnitude.



(a) *4b E_b/N_o=5 dB* **(b)** *4b E_b/N_o=6 dB*

(c) *5b E_b/N_o=5 dB* **(d)** *5b E_b/N_o=6 dB*

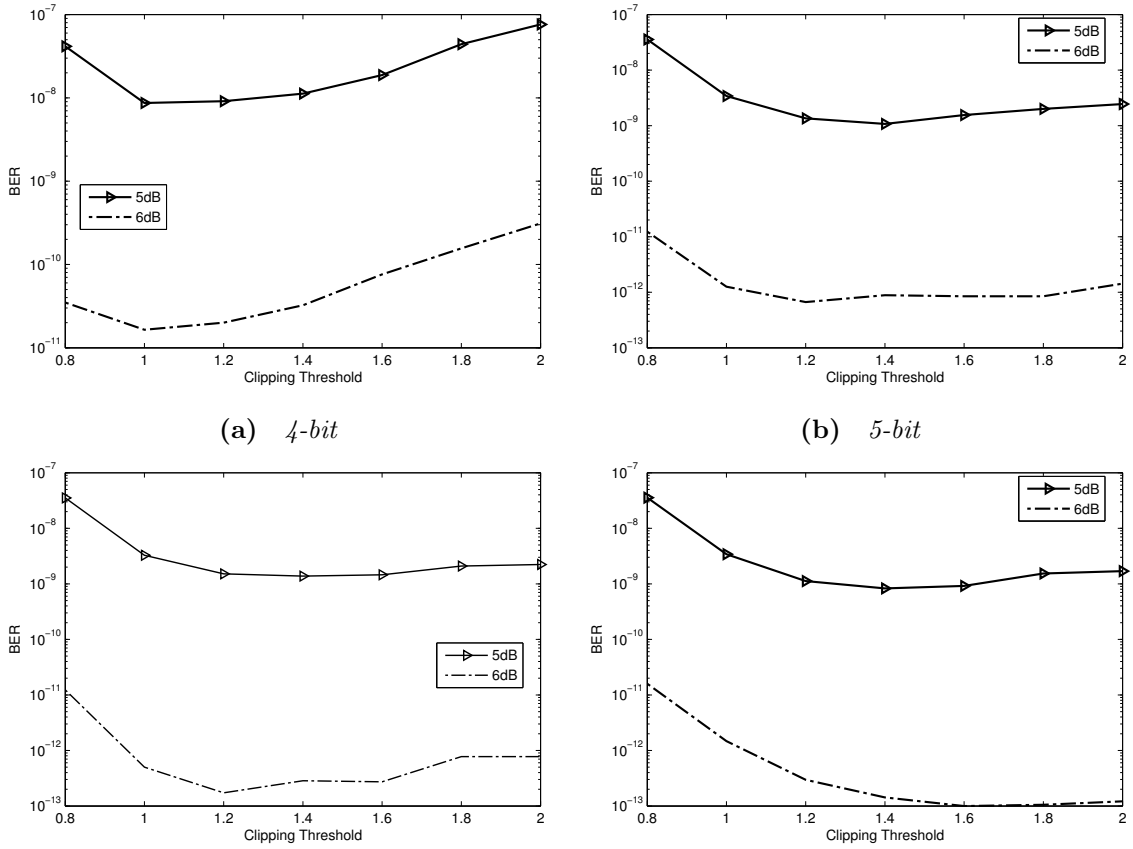
(e) *6b E_b/N_o=5 dB* **(f)** *6b E_b/N_o=6 dB*

Figure 6.3: *Rate of Saturation Events for "6Core" decoder with constant 4-bit, 5-bit, 6-bit LLRs at E_b/N_0 of 5, 6 dB*

The LP4a version described in Section 6.1 is slightly modified to include the functionality to keep track of the rate of saturation events in each processor core. The rate of saturation events captured for a "6Core" implementation at E_b/N_0 of 5 dB and 6 dB are included in Figure 6.3. The *ovflbit1* labeled in the plots corresponds to the aforementioned *bit3*, while the *ovflbit2* is *bit4*. The intersection point of the *ovflbit1* curve and the *ovflbit2* curve occurs at the later core in the pipelined chain of decoder processor cores as bit width is increased, while the increase of E_b/N_0 has an effect of pushing the intersection point back to an earlier core in the chain. Beyond the intersection point, the gap between the rate of saturation events for two of the overflow bits becomes narrower. The intersection point occurs around 3^{rd} core for the 4-bit implementation, between 4^{th} and 5^{th} core for 5-bit implementation, and between 5^{th} and 6^{th} core for the 6-bit implementation. This behaviour suggests a possibility that the BER performance may benefit from an additional bit in the magnitude bits in the core that the intersection point occurs, and is to be verified by the experiment described in Section 6.3.

6.3 Results and Analysis

Based on the results from the rate of saturation events presented in the previous section, a combination of "445566" is chosen as the bit width of the output LLRs of each decoder core in the 6-core pipelined chain based on the same order specified in the sequence, where the first "4" is the bit width of the first decoder core in the chain, and the last "6" is that of the last core. The extra bit in the magnitude bits is generated by saving the first neighbouring overflow bit of the most significant bit of the magnitude bits during the summation step in the variable node. The BER performance is obtained for the cases of constant 4-bit, 5-bit, 6-bit and "445566", as shown in Figure 6.4, over a range of clipping thresholds based on



(a) *4-bit* **(b)** *5-bit*

(c) *445566* **(d)** *6-bit*

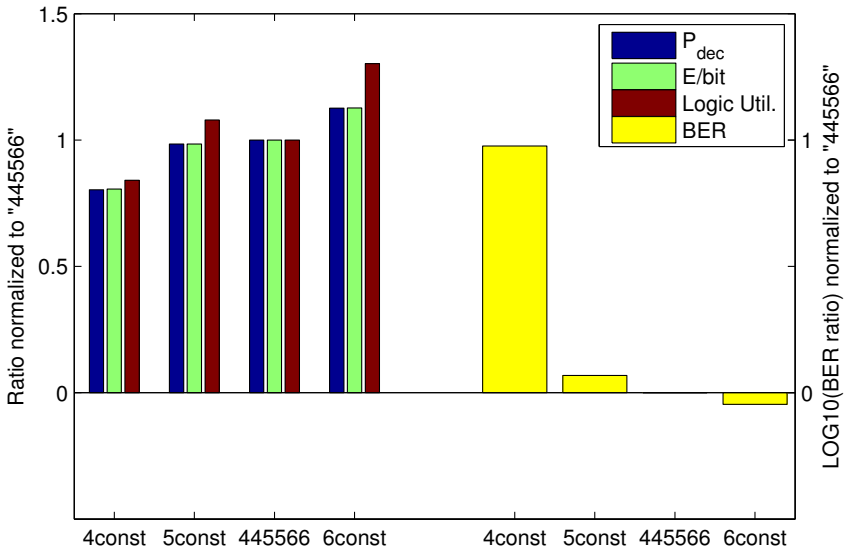
Figure 6.4: *Information BER at E_b/N_0 of 2 dB to 8 dB for configurations with 6 decoder cores on DE4 with various bit-widths for LLRs at a 75MHz clock with coded throughput of 2.4 Gbit/s for the $T_s = 192$, $\rho = 16$, rate-1/2 (3,6) PN-LDPC-CC code using clipping thresholds from 0.8 to 2 during quantization. The 5-dB curves are based on a target number of errors of 100, while the 6-dB curves are based on a target number of errors of 5.*

the same 4-bit quantized output LLRs from a BPSK-based AWGN channel. The 4-bit, 5-bit, 6-bit cases are implemented to provide references for the chosen "445566" case. The BERs plotted for an E_b/N_0 of 5 dB are based a target number of errors of 100, while the BERs for 6 dB are based 5 errors to reduce emulation time. The low target number of errors, 5, resulted in measurement error at the bottom of the 6-dB curves. However, while this has affected the locating of the optimal clipping threshold, it has not affected the main purpose of this experiment, which is to observe the effect on BER from growing the bit width of the LLRs of consecutive cores. The clipping threshold for lowest BER at the same E_b/N_0 increases as the LLR width increases.

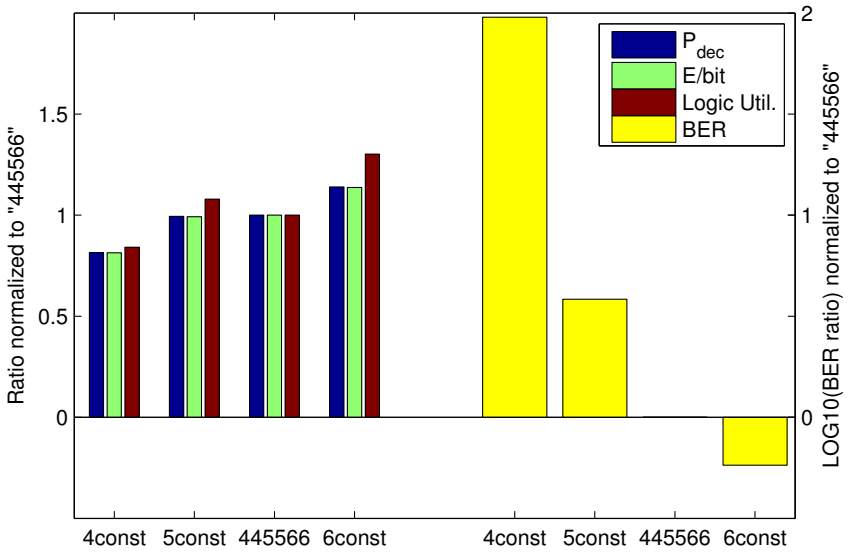
The decoder power measurements and energy-per-coded-bit for the configuration with the best BER performance in each "6Core" implementation for E_b/N_0 of 5 dB and 6 dB shown in Figure 6.4 are summarized in Table 6.1 and 6.2 along with the respective clipping threshold, c_{th}, BER, and logic utilization.

Table 6.1: *Decoder power measurements and energy-per-coded-bit, for PN-LDPC-CC decoder with 6 cores with various bit widths taken at E_b/N_0 of 5 dB at various clipping thresholds. The target number of error of 100 is used for BER data gathering for all cases. Energy-per-coded-bit, E/bit, is calculated using Equation (3.1). Results with subscript "raw" are based on raw power measurement. Results with subscript "mapped" are based on the mapped power numbers using the results in Section 4.3.2.*

LLR Width	Clipping Threshold	P_{raw} [W]	P_{mapped} [W]	E/bit_{raw} [nJ]	E/bit_{mapped} [nJ]	BER	Logic Util.
4 const.	1	2.92	1.47	1.22	0.613	8.67×10^{-9}	53%
5 const.	1.4	4.00	1.80	1.67	0.749	1.07×10^{-9}	68%
445566	1.4	4.12	1.83	1.72	0.761	9.16×10^{-10}	63%
6 const.	1.4	5.60	2.06	2.33	0.858	8.28×10^{-10}	82%



(a) $E_b/N_0 = 5dB$



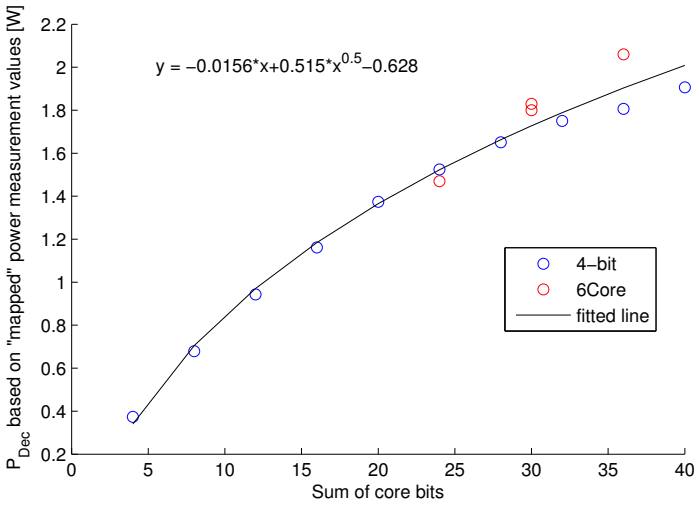
(b) $E_b/N_0 = 6dB$

Figure 6.5: *Decoder power, energy-per-coded-bit (E/bit), Logic Utilization, BER normalized to that of the "445566" case. The decoder power and energy-per-coded-bit values are based on the "mapped" values from the DC/DC converter efficiency experiment.*

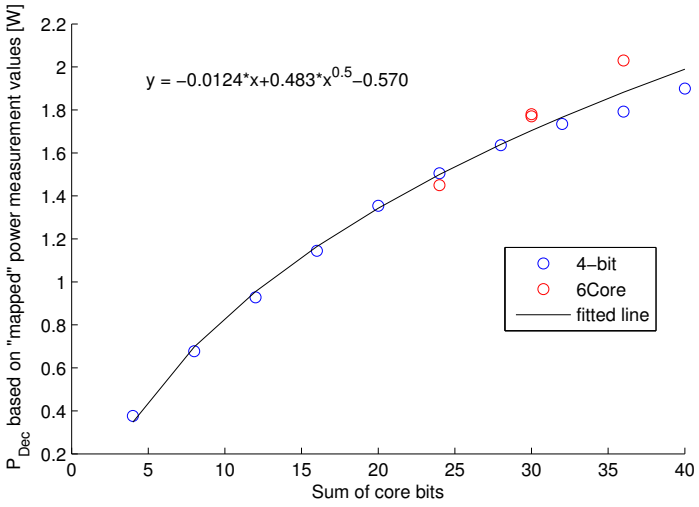
Table 6.2: *Decoder power measurements and energy-per-coded-bit, for PN-LDPC-CC decoder with 6 cores with various bit widths taken at E_b/N_0 of 6 dB at various clipping thresholds. The target number of error of 5 is used for BER data gathering for all cases. Energy-per-coded-bit, E/bit, is calculated using Equation (3.1). Results with subscript "raw" are based on raw power measurement. Results with subscript "mapped" are based on the mapped power numbers using the results in Section 4.3.2.*

LLR Width	Clipping Threshold	P_{raw} [W]	P_{mapped} [W]	E/bit_{raw} [nJ]	E/bit_{mapped} [nJ]	BER	Logic Util.
4 const.	1	2.86	1.45	1.19	0.605	1.65×10^{-11}	53%
5 const.	1.2	3.89	1.77	1.62	0.737	6.65×10^{-13}	68%
445566	1.2	3.94	1.78	1.64	0.743	1.73×10^{-13}	63%
6 const.	1.6	5.29	2.03	2.20	0.845	9.96×10^{-14}	82%

The "445566" case turns out to consume similar power as the constant 5-bit implementation, but has a BER performance that is closer to that of the constant 6-bit implementation. The "445566" case also happens to have a lower logic utilization than the 5-bit implementation. These trends are consistent from the tested E_b/N_0 from 5.0 dB to 6.0 dB, but the effect of varying LLR width is more significant for 6 dB, which is the higher E_b/N_0 tested, as depicted in Figure 6.5. The results listed in Figure 6.1 and 6.2 have demonstrated the "445566" case is more power-efficient and area-efficient than the 6-bit implementation while having only a slightly worse BER performance. At similar costs in logic utilization and decoder power as well as energy-per-coded bit, the "445566" case is a better option than the constant 5-bit implementation, as it provides a better BER performance. The lower logic utilization consumed by the "445566" implementation than the 5-bit implementation is most likely to be a result of the inherent randomness in the CAD algorithms used on Quartus II.



(a) $E_b/N_0 = 5dB$, *R-square* $= 0.9789$



(b) $E_b/N_0 = 6dB$, *R-square* $= 0.9826$

Figure 6.6: *Decoder power for 4-bit configurations and "6Core" configurations with 4-bit, 5-bit, 6-bit, and "445566" at* E_b/N_0 *of 5 and 6 dB versus sum of core bits, where sum of core bits are defined as the sum of LLR bit width used in each processor core for each configuration.*

Furthermore, the decoder power values obtained from the four aforementioned cases of "6Core" at two E_b/N_o and the decoder power values for various configurations with constant 4-bit LLR width from Chapter 5 are plotted against the sum of core bits in Figure 6.6. The fitted lines in both sub-figures in Figure 6.6 show a gradual increase in decoder power as the sum of core bits increases. Since only the result from one case of variable LLR widths are included, the measure, "sum of core bits", conveys the same relationship as the measure, number of cores. However, if data for more cases of variable LLR widths are available, "sum of core bits" is a better measure than the number of cores for analyzing the behaviour of decoder power.

The results from the "445566" experiment agrees with the hypothesis made at the end of Section 6.2, which suggested that the increase of bit-width at the core, where we observe intersection points of curves of overflow bits in Figure 6.3, can lead to a possible improvement in BER performance.

Chapter 7

Conclusions

7.1 Summary and Contribution

In this thesis, we presented an FPGA implementation of a rate-1/2 (3,6) PN-LDPC-CC encoder and decoder with $T_s = 192$ and $\rho = 16$ with a coded throughput of 2.4 Gbit/s on an Altera DE4 board. The decoder achieves a BER of approximately 10^{-10} with 9 processor cores, and an energy-per-coded bit of 1.683 nJ based on raw power measurements at an E_b/N_0 of 5 dB.

The set-up and model used to measure the power consumption of the decoder, and the efficiency of DC/DC converter circuitry on the DE4 board are described in Chapter 4. The measured power consumed by the decoder with various cores at various E_b/N_0 using a constant bit-width of 4 and clipping threshold of 1.33 is presented in Chapter 5. This power measurement method can be used for any design implemented on the Altera DE4.

We observe that, as expected the increase of E_b/N_0 is more effective at reducing the decoder power and energy-per-coded bit for configurations with 5 or more cores before

E_b/N_0 exceeds 5 dB. A decreasing incremental decoder power cost has also been observed for each additional processor core. The logic utilization also holds a positive increasing trend with the increase of the number of processor cores.

The effect of clipping threshold on the BER is analyzed in Chapter 6. The improvement in BER performance of decoder with increasing number of cores received from the change of clipping threshold gradually decreases at the same E_b/N_0 with constant LLR bit-width of 3 or 4. The increase of LLR bit-width from 3 to 4 also makes the decoder less sensitive to the change in clipping threshold. Rate of saturation events of two overflow bits in each decoder core are monitored for decoders with 4 to 8 cores at bit with LLR bit-width of 4 to 6. The results are used to select a test case with growing LLR width in the order "445566" for a "6Core" configuration. The results from the chosen test case are compared with constant LLR bit-width implementations with 4-bit, 5-bit, 6-bit processing quantized 4-bit AWGN channel LLRs.

While the test case is not necessarily using all the optimal parameters, it achieves a BER performance that is closer to that of the 6-bit configuration, while consuming similar logic utilization, power, and energy-per-coded-bit to that of the 5-bit implementation at E_b/N_0 of 5, 6 dB. This observation has suggested the feasibility of using increasing LLR bit-width of decoder cores to improve BER performance without increasing hardware and power costs. This finding suggests an alternate way to achieve BER improvement without making further optimizations in decoding algorithms, variable node and check node architecture designs.

7.2 Future Research

The presented FPGA decoder is based on a shorter code from the 26 rate-1/2 codes proposed in [26]. The test pattern generator, AWGN channel model, BER error counter and control logic as well as the JTAG-MM interface can be adapted to examine the performance of the other proposed PN-LDPC-CC with minor modifications. With the use of JTAG-MM interface from Altera, the change in BER performance, power consumption resulted from clipping threshold, E_b/N_0, bit-width, target number of errors, and number of cores can be obtained with reduced amount of required re-compilations. Additionally, the presented design can also be adapted to an ASIC implementation with minor changes to allow evaluation of the potential of the PN-LDPC-CC-based decoder.

The effect in BER performance of the presented "445566" case employing the idea of growing LLR width can be further verified by setting a higher target of number of errors to minimize measurement errors, provided more time is given. The idea of growing LLR bit-width can also be further verified using other proposed PN-LDPC-CC in [26] to confirm the consistency of its effect on BER performance and power consumption as well as energy-per-coded-bit.

In our experiments, the coding throughput is kept constant since no efforts have been made to increase the maximum achievable clock frequency of the design on the Altera DE4. Tools such as Design Space Explorer on Quartus II [47] can also be employed to further optimize the maximum clock frequency and logic utilization of the design.

Architectural changes other than the change of LLR width of each core in the pipelined decoder chain can be further explored to optimize the performance of the PN-LDPC-CC decoder on FPGAs.

Finally, the proposed power measurement method can be also used to characterize the

power consumption of other FPGA-based designs, including LDPC-CC and LDPC-BC decoders.

7.3 Publication Arising out of Thesis

Si-Yun J. Li, Tyler L. Brandon, Duncan G. Elliott, and Vincent C. Gaudet, "Power Characterization of a Gbit/s FPGA Convolutional LDPC Decoder," accepted for lecture presentation of the *IEEE Workshop on Signal Processing Systems (SiPS)*, Québec City, QC, 6 Pages, Oct. 2012

References

[1] T. Brandon, "Parallel-node low-density parity-check convolutional code encoder and decoder architectures," Ph.D. dissertation, University of Alberta, 2010.

[2] *LTM4601 Datasheet*, Linear Technology, Version C. Accessed Aug. 2, 2012. [Online]. Available: ⟨http://cds.linear.com/docs/Datasheet/4601fc.pdf⟩

[3] R. Gallager, "Low-density parity-check codes," *IRE Trans. on Information Theory*, vol. 8, no. 1, pp. 21–28, Jan. 1962.

[4] D. MacKay and R. Neal, "Near Shannon limit performance of low density parity check codes," *Electronics Letters*, vol. 32, no. 18, p. 1645, aug 1996.

[5] D. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. on Information Theory*, vol. 45, no. 2, pp. 399–431, Mar. 1999.

[6] T. Richardson, A. Shokrollahi, and R. Urbanke, "Design of provably good low-density parity check codes," in *Proc. of IEEE Int. Symp. on Information Theory*, 2000, p. 199.

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

[8] A. Darabiha, A. Chan Carusone, and F. Kschischang, "Multi-Gbit/sec low density parity check decoders with reduced interconnect complexity," in *Proc. of IEEE Int. Symp. on Circuits and Systems*, vol. 5, May 2005, pp. 5194–5197.

[9] A. Darabiha, A. Chan Carusone, and F. Kschischang, "Power reduction techniques for LDPC decoders," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 8, pp. 1835–1845, Aug. 2008.

[10] L. Liu and C. Shi, "Sliced message passing: High throughput overlapped decoding of high-rate low-density parity-check codes," *IEEE Trans. on Circuits and Systems I: Regular Papers*, vol. 55, no. 11, pp. 3697–3710, Dec. 2008.

[11] Z. Cui, Z. Wang, and Y. Liu, "High-throughput layered LDPC decoding architecture," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 4, pp. 582–587, Apr. 2009.

[12] K. Zhang, X. Huang, and Z. Wang, "A high-throughput LDPC decoder architecture with rate compatibility," *IEEE Trans. on Circuits and Systems I: Regular Papers,*, vol. 58, no. 4, pp. 839–847, Apr. 2011.

[13] S. Yen, S. Hung, C. Chen, H. Chang, S. Jou, and C. Lee, "A 5.79-Gb/s energy-efficient multirate LDPC codec chip for IEEE 802.15.3c applications," *IEEE Journal of Solid-State Circuits*, vol. 47, no. 9, pp. 2246–2257, Sep. 2012.

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

[15] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes. 1," in *Proc. of IEEE Int. Conf. on Communications*, vol. 2, may 1993, pp. 1064–1070.

[16] C. Wong and H. Chang, "High-efficiency processing schedule for parallel turbo decoders using QPP interleaver," *IEEE Trans. on Circuits and Systems I: Regular Papers*, vol. 58, no. 6, pp. 1412–1420, June 2011.

[17] C. Leroux, G. Le Mestre, C. Jego, P. Adde, and M. Jezequel, "A 5-Gbps FPGA prototype of a (31,29)2 Reed-Solomon turbo decoder," in *Proc. of Int. Symp. onTurbo Codes and Related Topics*, Sep. 2008, pp. 67–72.

[18] S. Karim and I. Chakrabarti, "Design of efficient high throughput pipelined parallel turbo decoder using QPP interleaver," in *Proc. of Int. Conf. on Multimedia, Signal Processing and Communication Technologies (IMPACT)*, Dec. 2011, pp. 248–251.

[19] A. Tu, "A gigabit/second turbo decoder on field programmable gate array: Supporting TSAT channel coding requirement," in *Proc. of IEEE Military Communications Conference*, Oct. 2006, pp. 1–5.

[20] A. Jiménez-Feltström and K. Zigangirov, "Time-varying periodic convolutional codes with low-density parity-check matrix," *IEEE Trans. on Information Theory*, vol. 45, no. 6, pp. 2181–2191, Sep. 1999.

[21] A. Pusane, K. Zigangirov, and D. Costello, "Construction of irregular LDPC convolutional codes with fast encoding," in *Proc. of IEEE Int. Conf. on Communications*, vol. 3, June 2006, pp. 1160–1165.

[22] H. Koga, "Next-generation power line communications and standardization," *Panasonic Technical Journal*, vol. 56, no. 1, pp. 16–21, Apr. 2010. [Online]. Available: ⟨http://panasonic.co.jp/ptj/v5601/pdf/p0104.pdf⟩

[23] "Advanced technologies from ISRO," Indian Space Research Organization (ISRO), 2010. [Online]. Available: ⟨http://www.isro.gov.in/ttg/pdfuploads/TTICLDPCC.pdf⟩

[24] R. Swamy, S. Bates, and T. Brandon, "Architectures for ASIC implementations of low-density parity-check convolutional encoders and decoders," in *Proc. of IEEE Int. Symp. on Circuits and Systems*, vol. 5, May 2005, pp. 4513–4516.

[25] T. Brandon, J. Koob, L. van den Berg, Z. Chen, A. Alimohammad, R. Swamy, J. Klaus, S. Bates, V. Gaudet, B. Cockburn, and D. Elliott, "A compact 1.1-Gb/s encoder and a memory-based 600-Mb/s decoder for LDPC convolutional codes," *IEEE Trans. on Circuits and Systems I: Regular Papers*, vol. 56, no. 5, pp. 1017–1029, May 2009.

[26] Z. Chen, T. Brandon, D. Elliott, S. Bates, W. Krzymien, and B. Cockburn, "Jointly designed architecture-aware LDPC convolutional codes and high-throughput parallel encoders/decoders," *IEEE Trans. on Circuits and Systems I: Regular Papers*, vol. 57, no. 4, pp. 836–849, Apr. 2010.

[27] Y. Ueng, Y. Wang, L. Kan, C. Yang, and Y. Su, "Jointly designed architecture-aware LDPC convolutional codes and memory-based shuffled decoder architecture," *IEEE Trans. on Signal Processing*, vol. 60, no. 8, pp. 4387–4402, Aug. 2012.

[28] C. Chen, Y. Lin, H. Chang, and C. Lee, "A 2.37-Gb/s 284.8 mW rate-compatible (491,3,6) LDPC-CC decoder," *IEEE Journal of Solid-State Circuits*, vol. 47, no. 4, pp. 817–831, Apr. 2012.

[29] A. Darabiha, A. Chan Carusone, and F. Kschischang, "A bit-serial approximate min-sum LDPC decoder and FPGA implementation," in *Proc. of IEEE Int. Symp. on Circuits and Systems*, May 2006.

[30] S. Sharifi Tehrani, S. Mannor, and W. Gross, "An area-efficient FPGA-based architecture for fully-parallel stochastic LDPC decoding," in *Proc. of IEEE Workshop on Signal Processing Systems*, Oct. 2007, pp. 255–260.

[31] V. Chandrasetty and S. Aziz, "FPGA implementation of high performance LDPC decoder using modified 2-bit min-sum algorithm," in *Proc. of Second Int. Conf. on Computer Research and Development*, May 2010, pp. 881–885.

[32] S. Sharifi Tehrani, S. Mannor, and W. Gross, "Fully parallel stochastic LDPC decoders," *IEEE Trans. on Signal Processing*, vol. 56, no. 11, pp. 5692–5703, Nov. 2008.

[33] R. Zarubica, S. Wilson, and E. Hall, "Multi-Gbps FPGA-based low density parity check (LDPC) decoder design," in *Proc. of IEEE Global Telecommunications Conference (GLOBECOM)*, Nov. 2007, pp. 548–552.

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

[35] R. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. on Information Theory*, vol. 27, no. 5, pp. 533–547, Sep. 1981.

[36] F. Kschischang, B. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Trans. on Information Theory*, vol. 47, no. 2, pp. 498–519, Feb. 2001.

[37] N. Wiberg, H. Loeliger, and R. Kotter, "Codes and iterative decoding on general graphs," in *Proc. of IEEE Int. Symp. on Information Theory*, Sep. 1995, p. 468.

[38] M. Fossorier, M. Mihaljevic, and H. Imai, "Reduced complexity iterative decoding of low-density parity check codes based on belief propagation," *IEEE Trans. on Communications*, vol. 47, no. 5, pp. 673–680, May 1999.

[39] J. Chen, A. Dholakia, E. Eleftheriou, M. Fossorier, and X.-Y. Hu, "Reduced-complexity decoding of LDPC codes," *IEEE Trans. on Communications*, vol. 53, no. 8, pp. 1288–1299, Aug. 2005.

[40] A. Anastasopoulos, "A comparison between the sum-product and the min-sum iterative detection algorithms based on density evolution," in *Proc. of IEEE Global Telecommunications Conference (GLOBECOM)*, vol. 2, 2001, pp. 1021–1025.

[41] S. Hemati, A. Banihashemi, and C. Plett, "A 0.18-um CMOS analog min-sum iterative decoder for a (32,8) Low-Density Parity-Check (LDPC) code," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 11, pp. 2531–2540, Nov. 2006.

[42] C. Kong and S. Chakrabartty, "Analog iterative LDPC decoder based on margin propagation," *IEEE Trans. on Circuits and Systems II: Express Briefs*, vol. 54, no. 12, pp. 1140–1144, Dec. 2007.

[43] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 203–215, Feb. 2007.

[44] S. Sharifi Tehrani, W. Gross, and S. Mannor, "Stochastic decoding of LDPC codes," *IEEE Communications Letters*, vol. 10, no. 10, pp. 716–718, Oct. 2006.

[45] T. Lang, E. Musoll, and J. Cortadella, "Individual flip-flops with gated clocks for low power datapaths," *IEEE Trans. on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 44, no. 6, pp. 507–516, June 1997.

[46] *Altera Avalon Interface Specifications*, Altera, Version 1.0. [Online]. Available: ⟨http://www.altera.com/literature/manual/mnl_avalon_spec.pdf⟩

[47] *Altera Quartus II Handbook*, Altera, Version 12.0.0. [Online]. Available: ⟨http://www.altera.com/literature/hb/qts/quartusii_handbook.pdf⟩

[48] Altera, "40-nm FPGA power management and advantages," White Paper, Altera, Dec. 2008, version 1.2. [Online]. Available: ⟨http://www.altera.com/literature/wp/wp-01059-stratix-iv-40nm-power-management.pdf⟩

[49] A. Blanksby and C. Howland, "A 690-mw 1-gb/s 1024-b, rate-1/2 low-density parity-check code decoder," *IEEE Journal of Solid-State Circuits*, vol. 37, no. 3, pp. 404–412, Mar. 2002.

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

[51] B. Crowley, "Predicting the switching activity of low-density parity-check decoders through density evolution," Master's thesis, University of Alberta, 2009.

[52] J. Zhao, F. Zarkeshvari, and A. Banihashemi, "On implementation of min-sum algorithm and its modifications for decoding low-density parity-check (LDPC) codes," *IEEE Trans. on Communications*, vol. 53, no. 4, pp. 549–554, Apr. 2005.

[53] A. Pusane, A. Jiménez-Feltström, A. Sridharan, M. Lentmaier, K. Zigangirov, and D. Costello, "Implementation aspects of LDPC convolutional codes," *IEEE Trans. on Communications*, vol. 56, no. 7, pp. 1060–1069, July 2008.

[54] E. Guizzo, "Closing in on the perfect code [turbo codes]," *IEEE Spectrum*, vol. 41, no. 3, pp. 36–42, Mar. 2004.

APPENDICES

Appendix A

A Lean Host-FPGA Interface Using JTAG-MM

This application note explains how to implement low-hardware-cost communication between an Altera FPGA and a host computer using the JTAG-MM interface, giving the host read-write accesses to registers created in the FPGA design. A soft-core microcontroller is not required on the FPGA, avoiding the associated hardware and power cost.

The main motivation for employing this interface is to allow parameterized experiments and interrogate internal state, such as in the BER experiments, where parameters like E_b/N_0 and clipping threshold can be modified without re-compilations.

We make the following assumptions:

- This tutorial assumes on slight experience on using Quartus II and proper configured installation of Quartus II and ModelSim Altera Starter Edition.
- The overall flow should apply to newer versions of **SOPC Builder** and **System**

Console with minor modifications, but please note that this guide is specifically written for Quartus II 11.0 **Build 157**.

- Altera is in the process of replacing **SOPC Builder** with **Qsys**, while the main idea of the flow should still apply, there is no guarantee that the steps for **SOPC Builder** will work with **Qsys**.

A.1 Checklist

- Quartus II Version 11.0 **Build 157** (I know it would work for Version 10.x as well)
- An Altera FPGA board (I used DE4)
- An existing Quartus II project with a top-level HDL file that contains all the compatible I/O signal descriptions for the target FPGA
- **my_dut.v** included in Appendix B
- **sw.tcl** and **hw.tcl** included in Appendix C

A.2 Detailed steps for JTAG-MM

A.2.1 Define an SOPC system

1. Open the existing Quartus II project and from the main Quartus II window, go to **Tools > SOPC Builder**

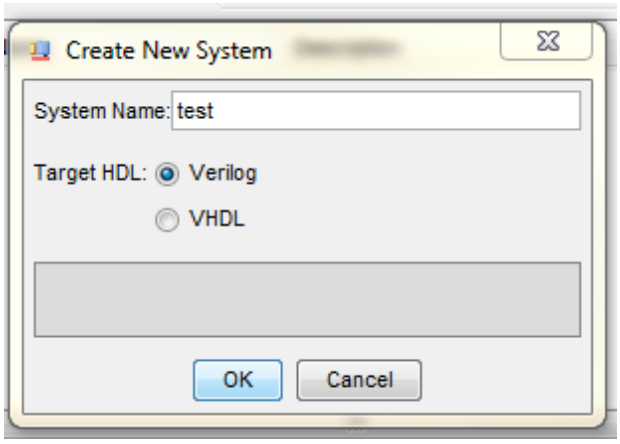


Figure A.1: *Create a new SOPC system*

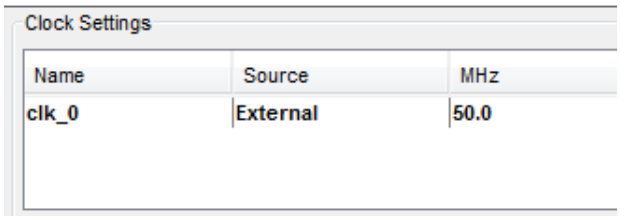


Figure A.2: *Clock settings for SOPC*

2. SOPC Builder will ask you to **Create New System**, as shown in Figure A.1, give it a name and pick the Target HDL, and click OK. For our simple test case, I'll just call it **"test"**, and I'll pick **Verilog** as the Target HDL.

3. Since I will be connecting the SOPC system to a 50MHz clock, I'll just leave the **Clock Settings** at default, as shown Figure A.2. I believe this would need to be modified if you plan to connect your SOPC module to a different clock.

4. Under the tab **System Contents** on the left of your **Altera SOPC Builder** window, double-click **New component..** under **Project** to import your pre-written **my_dut.v**.

5. Under the tab **HDL Files** on the **Component Editor**, specify the **location** of your my_dut.v file and **Top Level Module** (if you have included more than one HDL module). Go through all the other tabs on the **Component Editor** window to make sure everything is imported properly. Click **Finish** when you are done and Click **Yes, Save** to save **my_dut 1.0** and **my_dut_hw.tcl** file.

6. Under **Library**, expand **Bridges and Adapters** and then expand **Memory Mapped**, double-click **JTAG to Avalon Master Bridge** to add that to your SOPC system. If you plan to test your setup in *simulation mode* first, please *check the box* before **Use PLI Simulation Mode**, as shown in Figure A.3. (Leave it *unchecked* if you plan to test this on *hardware*) Click **Finish** to finish adding the module.

7. Under **Library**, double-click **my_dut** for that to be added to your SOPC system, and click **Finish** to finish adding.

8. Connect **my_dut_0** to the **master_0** as indicated in Figure A.4.

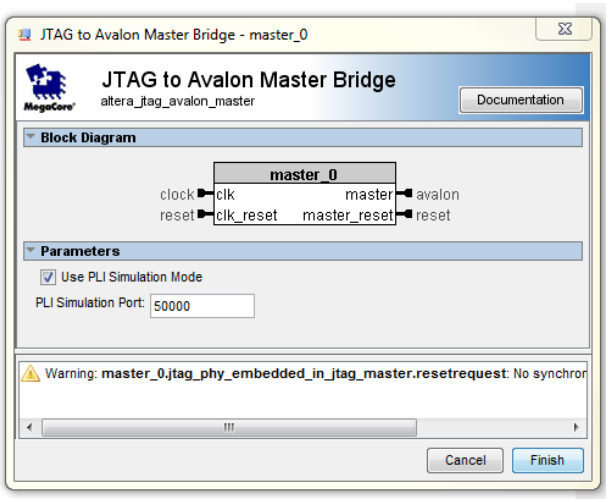


Figure A.3: *Configure* ***JTAG to Avalon Master Bridge***

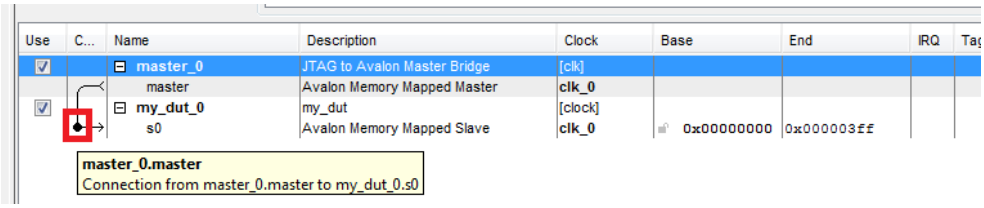


Figure A.4: *Connect components for SOPC system.*

9. **Save** the system.

10. Click on the tab **System Generation**, check the box before **Simulation. Create project simulator files**. Then click **Generate** at the bottom of your Altera SOPC Builder window to generate your SOPC system.

11. Under the project folder, open a file named "test_inst.v" (this should have been generated in the previous step) and copy the content (a sample instantiation of the SOPC module) to your top level HDL file. ("test" corresponds to the name I have assigned to this SOPC system in Step 2.) *I find that* ***master_read*** *and* ***master_write*** *commands would time out on System Console if the "rst" port of the SOPC instance is left unconnected. I am not sure if that's because I've missed any other intermediate steps.*

12. Update the clock and reset signal connections based on your target FPGA. In the case of DE4, I am connecting **clk_0** to **OSC_50_BANK2**, **reset_n** to **CPU_RESET_n**, as shown in Figure A.5.

```
test test_inst
{
.clk_0(OSC_50_BANK2),
.reset_n(CPU_RESET_n)
}
```

Figure A.5: ***HDL code*** *for connecting SOPC system*

13. **Save** all the changes and click **Compile Design** on the **Tasks** sub-window.

A.2.2 Simulation Mode

1. Go back to the **Altera SOPC Builder** window, go to the tab **System Generation**, click on **Run Simulator** to launch **Modelsim (Altera Starter Edition)**.

2. Enter **s** on the Command window of Modelsim to load the design. If you've done everything right so far, you should see the message shown in Figure A.6 on Modelsim.

```
# Server: creating socket at address 127.0.0.1, port 50000. Waiting for client...
```

Figure A.6: *Modelsim message after loading the design*

3. Now minimize the Modelsim window, go back to the **Altera SOPC Builder** window, go under **Tools** and select **System Console**.

4. Under Tcl Console of **System Console**, load **sw.tcl** by entering the following command (you may need to **cd** into the appropriate directory that contains the sw.tcl script though):

```
source sw.tcl
```

5. Go to **ModelSim** window, add all signals you would like to verify to the wave window. Type "**run 100 ms**" in the command window but *do not* press enter yet. *(I chose a rather long simulation time here on purpose; feel free to adjust this as needed.)*

6. Return to **System Console**, enter the following line in Tcl Console:

```
master_read_32 $cm 0x0 1
```

(The tcl command above reads one 32-bit number from the memory address 0x0, $cm is defined in sw.tcl to point to the path of the master bridge). Go to **ModelSim**

window, now press Enter for the "**run 100 ms**" command. You should now see what is shown in Figure A.7 on your **System Console**.

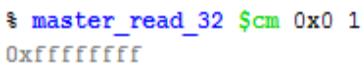


Figure A.7: *Returned value on* ***System Console*** *after a read command*

7. Then we can write to the "**test**" register defined in **my_dut.v** using the following command:

   ```
   master_write_32 $cm 0x0 0xFFFFFFFF
   ```

   Assuming you have added the "**test**" register to the ModelSim wave window, you should now see the value in "test" being *changed* from 0x0 to 0xFFFFFFFF. Or you can simply double-check using the same *read* command in Step 6 , and you should see the message shown in Figure A.8:

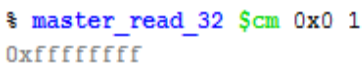
   

   Figure A.8: *Returned value on* ***System Console*** *after a write command*

8. Exit **System Console** and **ModelSim** after you are done. *(Note: You would find it's impossible to exit from ModelSim without killing the related processes under Windows Task Manager. I do not have a solution for this.)*

A.2.3 Hardware Mode

1. In Altera SOPC Builder window, right-click the **JTAG to Avalon Master Bridge** module in your SOPC system and choose **Edit** from the menu, **uncheck** the box for **Use PLI Simulation Mode**.

2. **Save** the change for the SOPC system and click **Generate** the bottom of the window to update the script files.

3. Go to the main **Quartus II** window, and **Compile Design** again.

4. **Download** the updated programming file to your target FPGA.

5. Launch **System Console** from **Altera SOPC Builder** window.

6. For **Quartus II v11.0 Build 157**, under Tcl Console of **System Console**, load hw.tcl by entering the following command (you may need to **cd** into the appropriate directory that contains the hw.tcl script though):

   ```
   source hw.tcl
   ```

   For **Quartus II v11.1 Build 173**, you need to use the following commands instead of what is included in **hw.tcl** to connect to the FPGA:

   ```
   set masters [ get_service_paths master ]
   set cm [lindex $masters 0]
   open_service master $cm
   ```

7. The **master_read_32** and **master_write_32 Tcl** commands used in the Simulation mode can be used here the same way to *read from* and *write to* the target register. Additional available Tcl commands can be found in **Analyzing and Debugging Designs with the System Console** document on Altera's website.

Appendix B

Verilog Code for device-under-test

```
module my_dut
(
        /* the I/O's of this module needs to have the following format:
        1. prefix: avs_
        2. s0_ : this needs to match with the number of the dut slave
        assigned on SOPC, use s0_ if you only have one MM slave
        3. write, read, address, writedata, readdata: for a full list
        of available Avalon-MM signals, please refer to Table: 3-1
        on Avalon Interface Specifications document
        */

        // clock & reset interface
        input csi_clock_clk,
```

```
        input csi_clock_reset,

        // MM slave
        input                avs_s0_write,
        input                avs_s0_read,

        // Width of address can be resized as needed
        input                [7:0]                avs_s0_address,
        input                [31:0]        avs_s0_writedata,
        output        reg        [31:0]        avs_s0_readdata
);

reg [31:0]        test;

always @ (posedge csi_clock_clk or posedge csi_clock_reset) begin

   if ( csi_clock_reset) begin
                test <= 0;
        end else if (avs_s0_write) begin // if write signal is high
                if (avs_s0_address == 0) begin
                        test <= avs_s0_writedata;
                end
        end else if (avs_s0_read) begin // if read signal is high
                if (avs_s0_address == 0) begin
                        avs_s0_readdata <= test;
```

```
                end
        end

end

endmodule
```

Appendix C

Tcl Scripts

C.1 Tcl Script for Simulation Mode

```
#
# The following needs to match with the name of the master bridge
# defined on SOPC
#
set console_master_name "master_0"



#
# The following needs to match with the PLI Simulation Port
# defined on SOPC
#
```

```
set pli_master_port_number "50000"

#
# Obtain service paths from Simulation mode
#
set master_service_paths [add_service pli_master $console_master_name \
$pli_master_port_number]

#
# Open Service
#
open_service master $master_service_paths

#
# Create a shorter variable for master path
#
set cm $master_service_paths
```

C.2 Tcl Script for Hardware Mode

```
#
# Obtain service path for master
#
set cm [ get_service_paths master ]


#
# Open service
#
open_service master $cm
```