Implementation of Low Density Parity Check Decoders using a New High Level Design Methodology

Syed Mahfuzul Aziz and Minh Duc Pham
School of Electrical & Information Engineering, University of South Australia, Mawson Lakes, Australia
Email: {Mahfuz.Aziz, Minh.Pham}@unisa.edu.au

***Abstract*—Low density parity check (LDPC) codes are error-correcting codes that offer huge advantages in terms of coding gain, throughput and power dissipation. Error correction algorithms are often implemented in hardware for fast processing to meet the real-time needs of communication systems. However hardware implementation of LDPC decoders using traditional hardware description language (HDL) based approach is a complex and time consuming task. This paper presents an efficient high level approach to designing LDPC decoders using a collection of high level modelling tools. The proposed new methodology supports programmable logic design starting from high level modelling all the way up to FPGA implementation. The methodology has been used to design and implement representative LDPC decoders. A comprehensive testing strategy has been developed to test the designed decoders at various levels. The simulation and implementation results presented in this paper prove the validity and productivity of the new high level design approach.**

***Index Terms*—Error correction coding, digital systems, digital communication, logic design, FPGA.**

I. Introduction

Information passing through a practical communication channel may be corrupted in transit by noise present in the channel [1]. Therefore it is of paramount importance for communication systems to have adequate means for the detection and correction of errors in the information received over communication channels. Turbo codes and LDPC (low density parity check) codes are most commonly used for error detection and correction nowadays [2]. Both of these codes provide coding gains [3] close to Shannon's limit [4]. LDPC codes however outperform turbo codes in terms of coding gain for large SNR [5, 6]. LDPC code of length 1 million with a coding rate (ratio of information bits to the sum of information and parity bits) of 0.5 and BER of 10-6 provides a capacity which is only 0.13db from Shannon's limit [5]. Further advantages of using LDPC codes are given in [7]. These include less computational complexity as compared to turbo codes, ability to pipeline the decoder to increase throughput at the cost of registers and some latency, and less number of iterations than turbo codes. Increased number of iterations reduces the throughput and increases power dissipation [5, 8]. The studies conducted in [7] and [8] indicate that lesser number of iterations in LDPC codes helps achieving higher throughput and decreases power dissipation. Another factor which contributes to increasing the throughput of LDPC decoder is the degree of parallelism which is adjustable [9, 10]. Another reason of using LDPC codes is that it is relatively easier to implement than turbo codes [11].

Despite all these advantages of LDPC codes the random parity check matrix makes the wiring between variable and check nodes complex, especially for large matrices. This leads to increased routing congestion in the decoder and eventually the size of the LDPC decoder increases and speed decreases [12, 13]. Complexity of practical implementation makes high throughput very difficult to achieve [14, 15]. Moreover, designing LDPC decoders in VHDL (Very high speed integrated circuit Hardware Description Language) becomes a cumbersome task when the size of the design increases. Huge amount of time and effort are required to model such large designs in VHDL [16]. It results in decrease in productivity. It becomes a nightmare for the designer to write VHDL code for thousands of connections and to make the changes required. Therefore, hardware implementation of LDPC decoder remains a challenge.

This paper examines high level modelling and synthesis techniques of LDPC decoders using emerging industry tools. It compares the high level approaches with traditional hardware description language based approaches in terms of modelling complexity, efforts and time. It also compares the results obtained for a representative LDPC decoder design using the high level approaches and using a traditional HDL based approach.

II. Design Approach

This paper investigates a new high-level modelling and synthesis methodology for LDPC decoder using state of

Project number: ITEE-09/CGD-12, University of South Australia.
Corresponding author: Dr Syed Mahfuzul Aziz, Email: mahfuz.aziz@unisa.edu.au

doi:10.4304/jcp.5.1.81-90

the art tools. As opposed to using only hand written VHDL codes for the entire design, this research examines high level design methods using Simulink, involving a combination of blocks designed with predefined library components and with embedded Matlab codes. A VHDL (VHSIC Hardware Description Language) code is generated automatically from the high level Simulink model using Mathworks's Simulink HDL coder (version 1.2). The entire design process is captured in Fig. 1.

Simulink HDL coder gives the flexibility of integrating different design approaches, namely Matlab programs, Simulink models and VHDL codes. Simulink and Matlab programs provide a much higher level of abstraction than VHDL. Therefore our design approach utilizes as many Simulink library blocks and Matlab blocks as possible to design various modules of the decoder provided these modules can be successfully processed by the HDL coder for automatic generation of VHDL codes. All the modules are then integrated in a top level Simulink model to produce the overall decoder model. VHDL code is generated automatically using Simulink HDL coder along with an optional test bench. This test bench can be used in ModelSim (a HDL simulator) to check correctness of the design. The auto-generated VHDL code can also be used in Altera's Quartus II or Xilinx ISE (Integrated Software Environment) for synthesis and implementation on desired FPGA (Field Programmable Gate Array). Automatic HDL code generation will lead to reduction in design efforts thereby increasing design productivity.

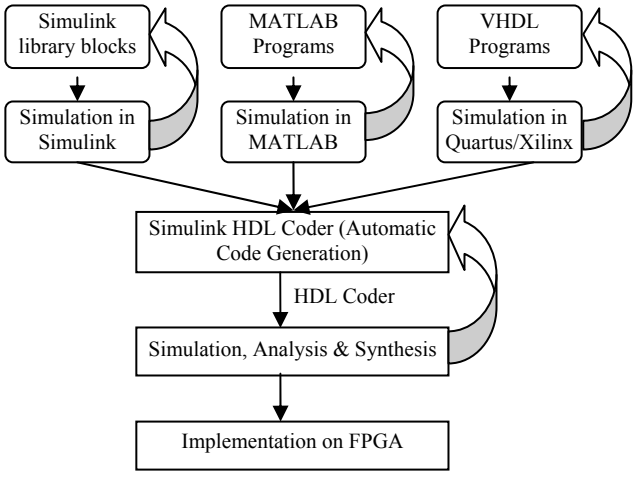

Figure 1. Proposed design flow.

III. Overview of LDPC Codes and Decoding Algorithm

Low density parity check (LDPC) codes are a class of linear block codes which are used for error detection and correction [17]. LDPC codes were first discovered by Gallager in 1962. In 1981, Tanner worked on LDPC codes and came up with important tanner graphs or bipartite graphs [18]. LDPC codes could not be implemented at the time of invention because the technology was not advanced, though they were studied again after almost three decades because they are more advantageous than any other error-correcting codes. LDPC codes are extensively used in standards such as 10Gigabit Ethernet (10GBaseT) & digital video broadcasting (DVB-S2) [19].

There are different algorithms which could be used for decoding purposes. We used the min-sum decoding algorithm [14, 20], which is a special case of sum-product algorithm [18, 21, 22]. Sum-product algorithm reduces the computational complexity and makes the decoder numerically stable [18]. Assume that the messages from the host communication system are represented by I and are passed on to the decoder for error correction. The LDPC decoder consists of a number of variable nodes (v) and check nodes (c). The operation of the min-sum algorithm can be summarised as follows [14]:

A. Variable Node Operation

A variable node performs the operation given in (1) and passes the outputs to check nodes.

$$L_{cv} = \sum_{m \in M(v) \backslash c} R_{mv} + I_v \qquad (1)$$

where, I_v is the input to variable node v, also known as Log Likelihood ratio (LLR), L_{cv} is the output of variable node v going to check node c, $M(v)\backslash c$ denotes the set of check nodes connected to variable node v excluding the check node c, R_{mv} is the output of check nodes going to variable node v.

B. Check Node Operation

A check node receives messages from variable nodes and performs the operation given by (2):

$$R_{cv} = \prod_{n \in N(c) \backslash v} sign(L_{cv}) \times \min_{n \in N(c) \backslash v} | L_{cv} | \qquad (2)$$

where, R_{cv} is the output of check node c going to variable node v.

C. Parity Check

Every check node also checks whether the parity condition is satisfied by looking at the sign of the messages coming from the variable nodes. Until all the parity checks from all the check nodes are satisfied the messages are sent back to variable nodes and the variable nodes do the operation specified in part (A), otherwise the decoder stops the process.

Min-sum decoding algorithm uses soft decision. However, hard decision is taken on the new LLR (I_v). If the new LLR is negative then the output bit would be a 1 otherwise a 0. Interested readers can find the details of the sum-product and min-sum decoder algorithms in [18, 20-22].

IV. Representative Decoder Designs

LDPC codes can be represented by an M x N sparse matrix, usually called H matrix. The H matrix contains mostly zeros and a small number of 1s [9, 10]. It can also

be represented by a graph called bipartite or tanner graph. Tanner graphs contain variable and check nodes. The decoder prototype designed in this research is based on the matrix shown in Fig. 2. It is a very small matrix compared to the real matrices. This 10x5 matrix can be transformed into a tanner graph having 10 variable nodes and 5 check nodes as is shown in Fig. 3. The signals go from the variable nodes to the check nodes and from the check nodes back to variable nodes. Typically different sets of wires are used for the signals going from variable nodes to check nodes and vice versa. The process is iterative and goes until all the parity checks are satisfied. The messages in our prototype designs are 4-bit long.

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

Figure 2. A 10x5 sparse matrix.

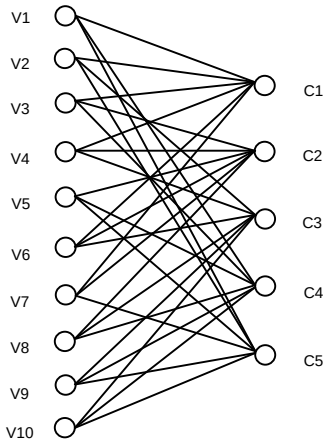

Figure 3. Tanner graph.

A. Design 1

The approach taken in the first decoder design was to use a combination of embedded Matlab blocks and Simulink library blocks. No handwritten VHDL blocks were used. A generic Simulink model was first developed for each of a variable node and a check node.

Check node design in Simulink: The check node has been designed in the same way as the variable node. The Simulink model of the check node is shown in Fig. 4. It uses blocks from Simulink library (absolute and bitwise XOR). The check node finds the minimum of all the inputs and performs the parity checks. It contains a control block, which controls its operation.

Variable node design in Simulink: The variable node, shown in Fig. 5, has been designed in Simulink using the basic *add* block from the Simulink library. The control block controls the operation of the variable node. The variable node has four inputs, the first three come from various check nodes and the fourth one is the raw LLR, which is an external input supplied by the host communication system. It performs the operation given in (1) and passes the outputs to the check nodes to which it is connected.

Once the Simulink models are developed for the check node and the variable node, the VHDL codes for each of these can be generated automatically using the *HDL Coder* tool according to the design flow presented in Fig. 1. The functionality of each of these models can be verified at both Simulink and VHDL levels. Each VHDL model can also be synthesized for a target FPGA.

10x5 LDPC decoder: A LDPC decoder with 10 variable and 5 check nodes has been designed using instances of the variable and check node components described above. The various nodes are interconnected in a way that corresponds to the Tanner graph of Fig. 3.

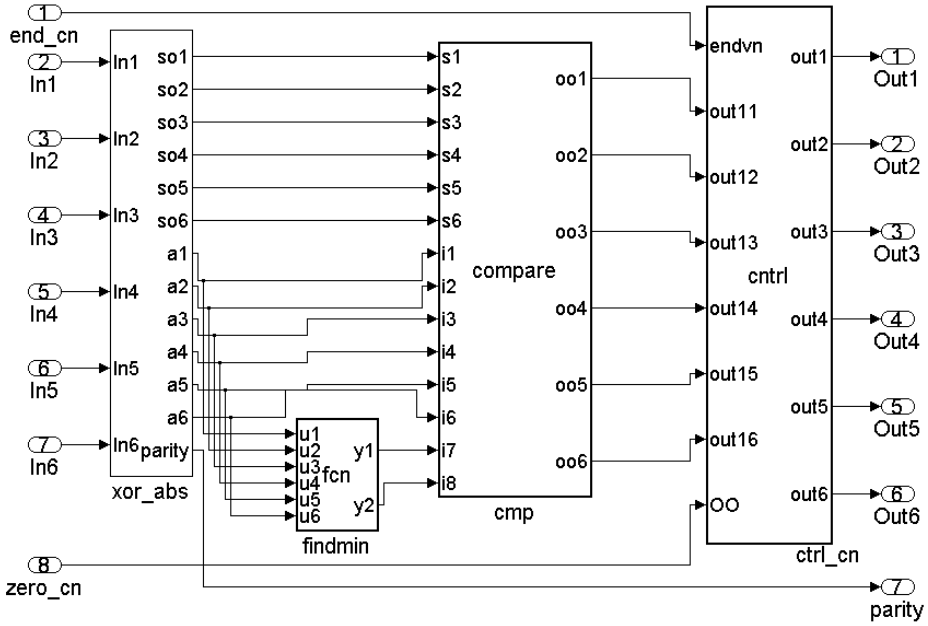

Figure 4. Check node design in Simulink.

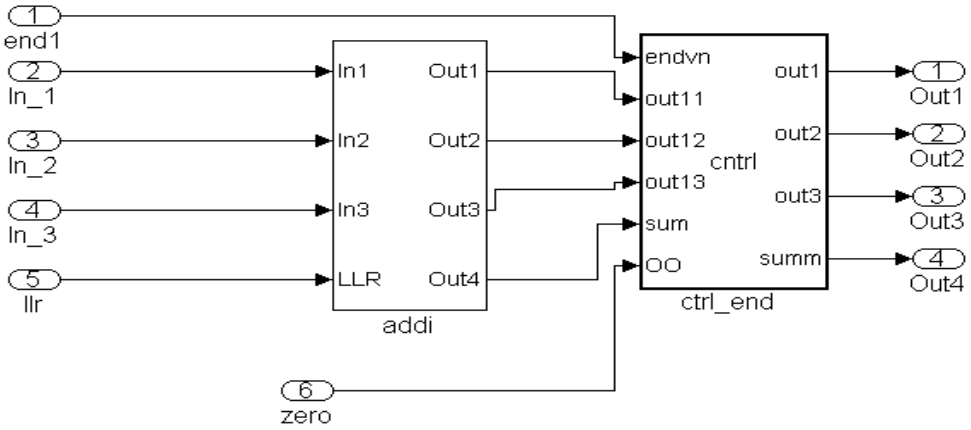

Figure 5. Variable node design in Simulink.

B. Design 2

Design 2 uses a combination of embedded Matlab blocks, Simulink library blocks and VHDL/Link for ModelSim blocks. 'Link for ModelSim' is a utility that enables modules coded in VHDL to be embedded in Simulink models. Design 2 is different from the first design in that it uses serial communication of messages between variable and check nodes. This is achieved using SIPO (serial-in-parallel-out) and PISO (parallel-in-serial-out) registers at each input and output port in all the nodes. This greatly simplifies the interconnections by reducing the number of wires four times at the cost of some extra registers. The variable and check node components are same and are connected in the same way as in Design 1. The PISO and SIPO components are coded in VHDL and are used in the Simulink model with the help of the 'Link for ModelSim' utility (to link the VHDL blocks with ModelSim for simulation and code generation purposes). The way PISO and SIPO are used in a variable node is shown in Fig. 6. Check nodes have been modified in exactly the same manner. The inputs and outputs of all the variable and check nodes are now 1-bit long. The effects of using a VHDL block in a Simulink design are discussed in the next section. The HDL coder does not generate the VHDL code of the Link for ModelSim blocks. The VHDL code for these blocks needs to be added before synthesizing the code.

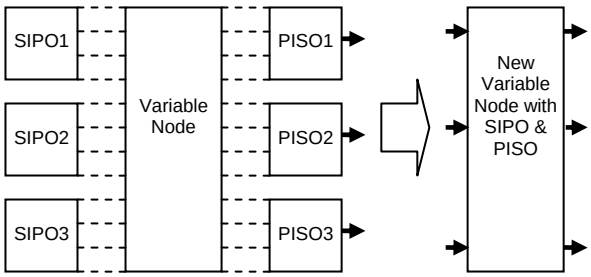

Figure 6. A variable node with PISO and SIPO.

V. Results And Analysis

A. Convergence Test

The convergence characteristics of the LDPC decoders presented in this paper have been tested using VHDL testbench and compared with that of a Matlab code for a functionally equivalent decoder. For this purpose the VHDL code automatically generated from the Simulink models was used. Fig. 7 shows the spread of the number of iterations for Design 2 by using plots of (mean number of iterations + standard deviation) and (mean number of iterations − standard deviation) versus SNR (E_b/No). It is clear that convergence is achieved in 6 iterations. This is consistent with the convergence result of a functionally equivalent decoder shown in Fig. 8, designed using Matlab code [23].

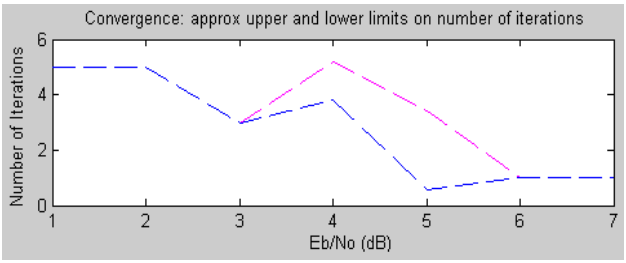

Figure 7. Spread of the number of iterations for Design 2 obtained from VHDL testbench.

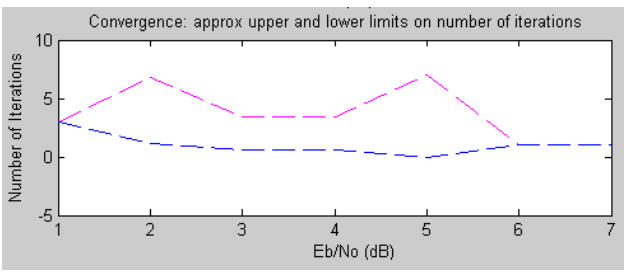

Figure 8. Spread of the number of iterations for a functionally equivalent Matlab code.

B. Algorithm Performance

The performance of our LDPC algorithm has been evaluated by simulating the decoder over AWGN channel and plotting the Bit-Error-Rate (BER) against Signal-to-Noise-Ratio (SNR) [24, 25]. Fig. 9 shows the BER performance of Design 2 obtained from simulation of the high-level Simulink model. Fig. 9 also shows the BER plot of the unencoded BPSK channel. Clearly our decoder demonstrates increasing gain in BER with increasing SNR compared to the unencoded BER. This proves that the proposed high-level design method can deliver competitive designs with the desired gain in BER performance [24, 25].

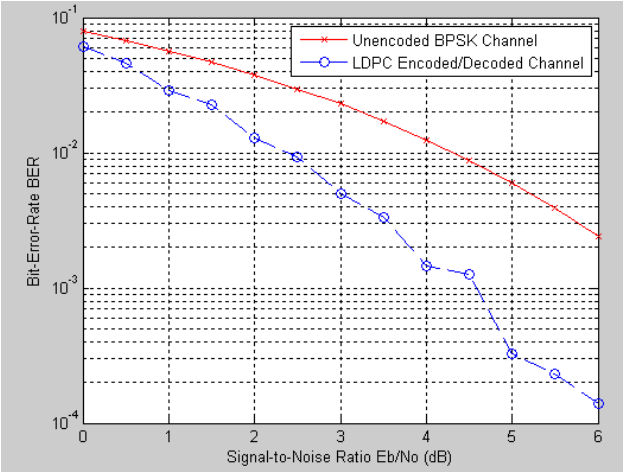

Figure 9. Performance simulation of the LDPC decoder.

C. Synthesis

Synthesis results for the high-level designs are given below and are compared with their 'VHDL only' counterparts. The results were obtained using Altera's Quartus II software with Cyclone II EP2C70F672C6 as the target FPGA device. Table 1 compares the resources used by our Design 1 and its maximum operating frequency (Fmax) with the same decoder designed solely using hand coded VHDL. Our design uses 2.1% of the FPGA's logical elements, only 120 registers and has a maximum frequency of 42.96 MHz. Clearly our Simulink design uses much less resources and is faster than the *VHDL-only* design. The higher register count in the VHDL design is due to the presence of dedicated registers for latching variable and check node outputs, which were removed from the Simulink design. Table 2 compares our Design 2 with its *VHDL-only* counterpart. It uses 3.5% of the logic elements, which is higher than the amount used by the *VHDL-only* design (2.8%).

TABLE I.
DESIGN 1 COMPARED WITH VHDL-ONLY COUNTERPART

	Design 1	VHDL only design
Logical elements	1432 (2.1%)	1492 (2.2%)
Combinational functions	1432 (2.1%)	1492 (2.2%)
Total registers	120 (0.2%)	686 (1%)
Maximum frequency (Fmax)	42.96MHz	37.3 MHz

TABLE II.
DESIGN 2 COMPARED WITH VHDL-ONLY COUNTERPART

	Design 2	VHDL only design
Logical elements	2416 (3.5%)	1916 (2.8%)
Combinational functions	2416 (3.5%)	1916 (2.8%)
Total registers	480 (0.7%)	881 (1.3%)
Maximum frequency (F_{max})	67.20MHz	67.72MHz

The reason for the higher number of logic elements in our design is that the control unit contains handshaking circuitry to facilitate communication of large amount of data between the PC and the FPGA board for testing. However, the *VHDL-only* design has a simple control unit and does not include any such handshaking circuitry. Our Design 2 uses nearly half the number of registers and achieves nearly the same maximum frequency (Fmax). The time required by our high-level approach for successful modelling, simulation and synthesis of the LDPC designs was almost a quarter of that required by the hand coding method.

D. Behavioral Simulation of VHDL Model

Fig. 10 shows the functional simulation result for the VHDL model of Design 2 generated from its top level Simulink model. A set of raw LLRs (6, 6, 2, 4, 7, 4, -2, 6, 4, 7) are applied to the variable nodes. The signals *end_o_vn* & *end_o_cn* are the control signals. The parity becomes 1 in the third iteration when all the parity checks are satisfied and the decoder stops the iterations. The corrected LLRs output by the variable nodes are 7, 7, 7, 7, 7, 7, -8, 4, 5, 7.

E. Hardware Implementation and Testing

After fully simulating the Simulink as well as the VHDL models of Design 1 and Design 2, both designs were implemented on a Xilinx Spartan 2E FPGA. The FPGA platform we used is shown in Fig. 11. It contains three separate modules: the USB communication module for communicating with the PC, the main FPGA module housing the Xilinx Spartan IIE, and the I/O module. The LLRs are generated by a MATLAB program and are stored in a text file on the PC. A LabVIEW program running on the PC sends these LLRs to the decoder via the USB module and receives the decoded LLRs back along with the parity information. The decoded LLRs are written into a separate file by the LabVIEW program and analysed for correctness by a MATLAB program by comparing with the LLRs generated by simulation of the Simulink and/or VHDL models. The PC controls the operation of the decoder on the FPGA through a number of handshaking signals as is shown in Fig. 12, for example start/stop, max_iteration etc. The I/O module has been used to display useful runtime information, for example the number of iterations completed by the decoder for each LLR. This enables us to obtain a visual indication that the LDPC decoder is operating. The performance results obtained from the implemented decoders are presented in the next sub-section along with the performance of Simulink and VHDL models.

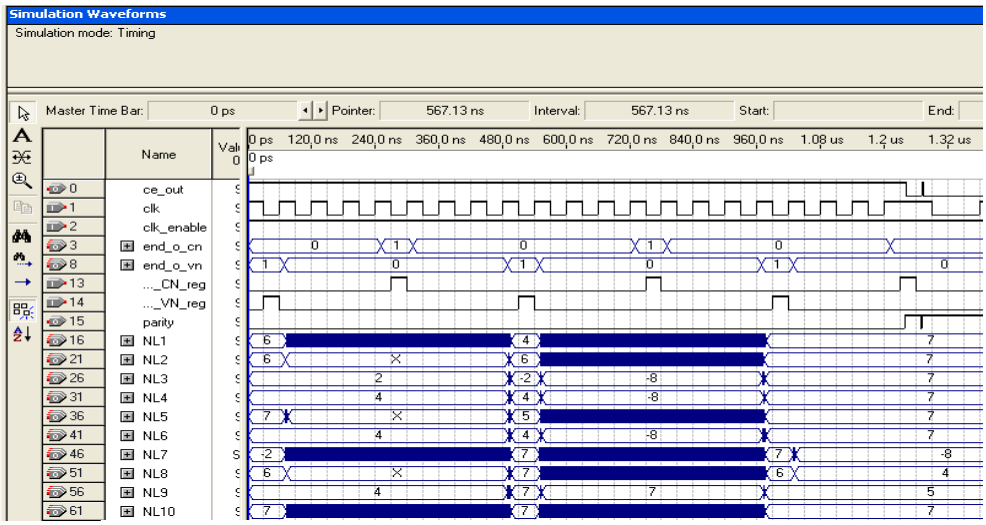

Figure 10. Behavioral simulation results of decoder Design 2.

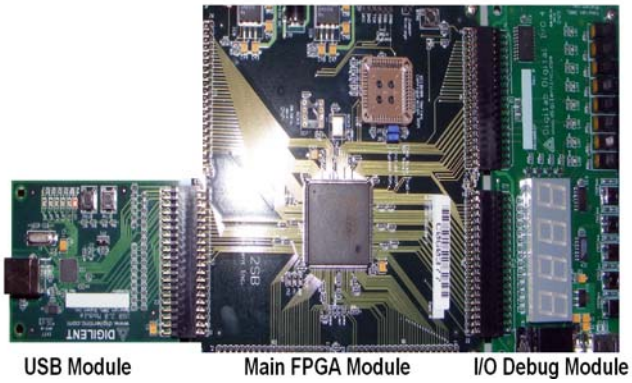

Figure 11. The FPGA platform used to implement the decoders.

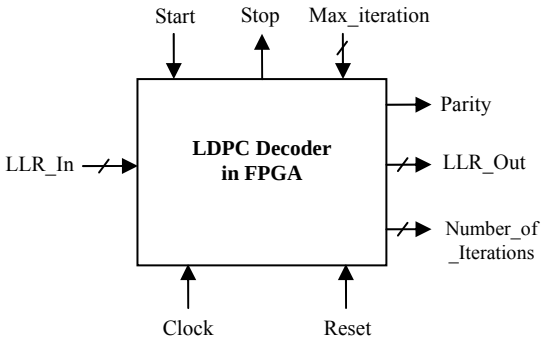

Figure 12. Block diagram of the LDPC decoder implemented on the Xilinx Spartan 2E FPGA device.

F. Comprehensive Testing Strategy

The Matlab and Simulink environment used in the high level design methodology make it very easy and fast to build a full test system for testing the whole design. In this section we present a comprehensive testing strategy for LDPC decoders whereby we can test the decoder outputs from three different environments simultaneously. Fig. 13 shows the proposed testing strategy. The test system allows the design to be simultaneously tested in Simulink, VHDL (ModelSim) and on the FPGA.

The *LDPC Encode Test Data* module generates a sequence of LDPC encoded test data and sends these to the Simulink simulation model, VHDL testbench and FPGA at the same time. After decoding is done in the three environments, the data are sent back to the *Test Data Analysis* module. This module analyses the decoded data from the three environments and compares the parity information for correctness. We have used this scheme to validate the decoder designs presented in this paper.

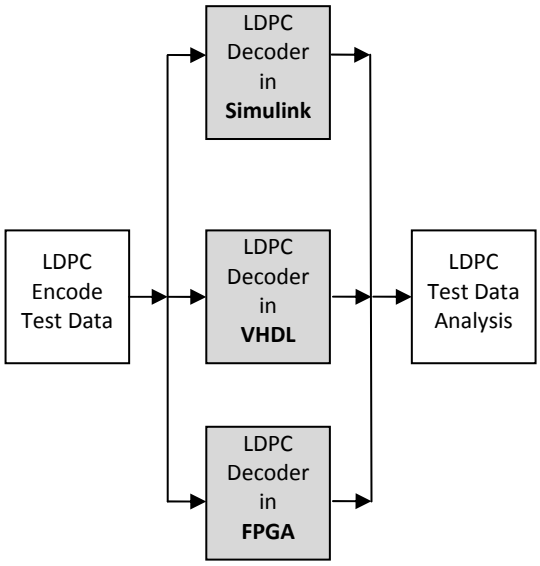

Figure 13. Structure of the testing system.

Fig. 14 shows the performance plot (BER) obtained from the FPGA and compares it with the BER obtained from the Simulink model. The performance plots from the VHDL testbench and FPGA are also compared in Fig. 15. These figures show a close match among the performance plots obtained from three different environments and therefore prove that the design has been correctly implemented and run on the FPGA device.

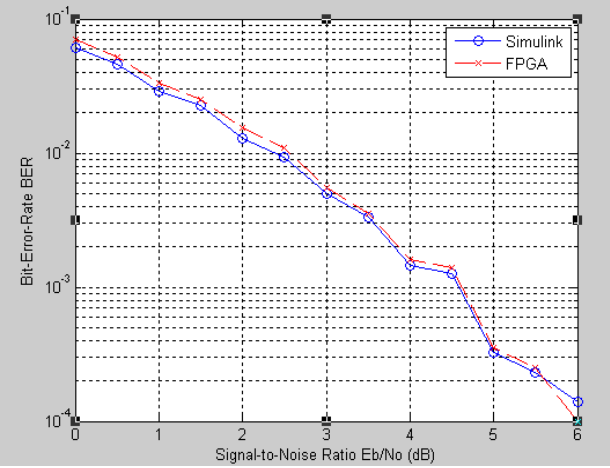

Figure 14. Bit-Error-Rates obtained from FPGA implementation compared with Simulink simulation.

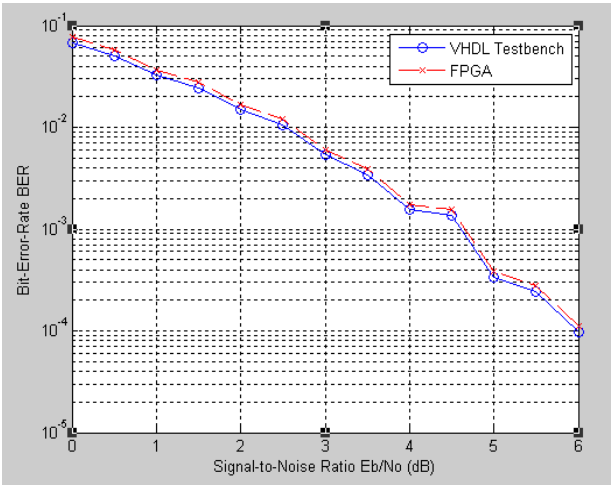

Figure 15. Bit-Error-Rates obtained from FPGA implementation compared with VHDL testbench.

VI. Analysis of the High Level Design Method

The high-level design methodology presented in this paper reduces design complexity, effort and time. For example, to design the variable and check nodes of Fig. 4 and Fig. 5 completely in VHDL a designer has to write behaviours of quite a few modules in VHDL and manually do the port mapping for the top level design. As the design gets larger and larger it becomes very difficult to code everything in VHDL. It not only takes huge effort and time, but also managing complex designs and reusing the designs are often quite difficult. A great deal of expertise and experience in VHDL is also required. For a large LDPC decoder with hundreds of nodes, the interconnections among the variable and check nodes could easily become a designer's nightmare if the entire design has to be coded in VHDL. An alternative, intuitive and highly efficient design method has been a long standing desire of the engineers engaged in the design of complex digital systems such as LDPC decoders.

The high level methodology we have presented in this paper offers a very attractive alternative. The designs can be done intuitively in Simulink using predefined Simulink library blocks and blocks made from high-level Matlab code. The design complexity, effort and time are reduced drastically because the need for manually writing complex VHDL code is almost eliminated. Our estimates have shown that the time required to successfully design, simulate and synthesise a LDPC decoder using hand coded VHDL is almost four times that required by the proposed high-level methodology. Our decoder Design 2 had 1069 lines of VHDL code generated automatically by HDL Coder from the top level Simulink model as opposed to only 225 lines of hand written VHDL code. The main reason for the large code produced by HDL Coder is that it used a very large number of internal signals to generate the VHDL description of the decoder. This is something we did not have any control over. Of course our decider Design 2 had additional handshaking circuitry to facilitate communication between the PC and the FPGA board for testing purpose. This contributed to the larger code to some extent. However, we did not optimise the auto-generated VHDL code. Yet our decoder designs compare favourably with the hand coded designs (see Tables 1 and 2).

Another important aspect of the proposed high-level design methodology is that design reuse requires much less effort because changes are made either at block level in Simulink or in high-level Matlab code. In addition Matlab is a software programming language that is used much more widely than hardware description languages like VHDL and Verilog. Therefore designers without specific skills in hardware languages are able to design complex digital systems without much problem. Even software engineers and algorithm developers are able to quickly implement and test their high-level designs due to the ability to automatically generate HDL descriptions from the Simulink models. This will surely offer great flexibility and efficiency in the design and reuse of complex LPDC decoders. There are some other benefits of the high level design methodology:

- Different parts of the model can be enveloped using the 'create subsystem' property of Simulink. It reduces the design complexity for large designs.
- User created Simulink library blocks provide great flexibility because the revisions made to the user defined library seamlessly propagate through the entire model.
- HDL Coder can generate either VHDL or Verilog descriptions from the high-level Simulink models, allowing greater flexibility in the choice of the hardware description language.

In the high-level design methodology, it is also easy and fast to build a complete test system. The Matlab and Simulink library provides a lot of powerful tools for generating and analysing test data such as graphical plot, scripts, etc.

The design methodology we have presented requires the use of some emerging design tools and library functions, such as Simulink hardware library and

embedded Matlab blocks, Link for ModelSim and HDL Coder tools. Because these are very recent developments the library of Simulink blocks to support the functions a designer needs is rather limited at the present time. Similarly the capability of HDL Coder is limited to conversion of a few commonly used Simulink blocks and a few embedded Matlab blocks. Some of the specific limitations are discussed below.

A. Current Limitations

Some common blocks in the Simulink library are not currently supported by the HDL Coder for automatic generation of VHDL code, e.g. flip-flops. Although we could build the PISO and SIPO registers easily in Simulink using the flip flops from its library the registers were not converted to VHDL by the HDL Coder. A careful selection of supported Matlab functions and Simulink library blocks may help addressing this type of problems, but not necessarily always. Some other limitations we experienced are listed below:

- Multiple instances of some modules (components) are used in the LDPC decoder, and in fact in most modular designs utilising a hierarchical design methodology. For example, in our LDPC decoder multiple instances of the check and variable nodes, and PISO and SIPO registers are used. The HDL Coder dumps the full behaviour of each component as many times as it is instantiated in the design. This produces redundant instances of component behaviour in the generated VHDL code. It is necessary for the designer to edit the auto-generated code.
- There are some functions which are supported by the HDL Coder, but it produces the output in a particular data type. For example, the sign function of Matlab gives output only in 8-bit integer (int8) format. It is not possible to change these default data types in the current version. The only option is to manually edit the auto-generated HDL code.
- Link for ModelSim: This utility enables blocks designed in VHDL to be included in Simulink models. However the ports of the blocks that use Link for ModelSim get interchanged while simulating in Simulink. HDL code cannot be generated for the Simulink model in this situation. The code can be generated only after correcting the design. This makes the design process difficult and time consuming. The other drawback of blocks utilising Link for ModelSim is that these blocks may make changes in the auto-generated code. In case of the LDPC decoder these blocks changed the signals of type signed to unsigned. Once again this requires manually editing the auto-generated VHDL code.

VII. CONCLUSIONS

In this paper a new high-level intuitive design methodology based on Simulink for modelling, synthesis and implementation of LDPC decoders has been presented. It utilises the higher level of abstraction offered by the Simulink modelling environment. The modelling, simulation and synthesis process utilises a combination of emerging design tools and associated library functions. These include the Simulink HDL Coder and 'Link for ModelSim' tools, and embedded Matlab and Simulink hardware library blocks. Two versions of 10x5 LDPC decoders have been designed, simulated, synthesised and successfully implemented on a Xilinx Spartan 2E FPGA device. A comprehensive testing strategy has been adopted to test the decoders at all levels, from the high level Simulink model through VHDL all the way up to hardware implementation. Our testing strategy supports simultaneous testing of the decoders at the three levels which is useful for real-time debugging. The proposed high level design methodology facilitates the creation of such testing strategy very quickly because the test data generated at the high level is used for testing at all three levels. This helps to reduce the time of evaluating and testing the design as well as ensures that the final design is efficient and error-free. The performance figures on Bit-Error-Rates obtained at the three levels compare favourably with those of the LDPC decoders reported in the literature.

The proposed high level design methodology offers great advantages in terms of design complexity, effort and time compared to a HDL-only design method. Our Design 1, completed using the new methodology, uses less FPGA resources and achieves higher F_{max} compared to its HDL-only designs. Our Design 2 achieves nearly the same F_{max} as its VHDL-only counterpart, but uses slightly higher number of logic elements due to the inclusion of additional handshaking circuitry required for testing the design on FPGA. Given the significant reduction in design effort and time, the above results make the proposed design methodology a very attractive one. We envisage that with further enrichment of the Simulink Library blocks, and enhancements of HDL Coder and Link for ModelSim tools, design methodologies similar to the one presented in this paper will eventually replace tedious HDL-based design approach. This will pave the way for cost effective design and reuse of a new generation of complex high performance LDPC decoders.

ACKNOWLEDGEMENT

This work has been supported in part by a research grant from the Division of IT, Engineering and the Environment of the University of South Australia (UniSA). The authors wish to thank Mr Sunil Sharma for his initial investigations into developing variable and check node models using Simulink and related tools. The authors also acknowledge Prof Bill Cowley of UniSA's Institute of Telecommunications Research for his useful suggestions and critical feedback. The authors understand that Prof Cowley's time has been partially supported through a research grant from Sir Ross and Sir Keith Smith fund. Finally the authors would like to thank Dr Mark Ho of the School of Electrical and Information Engineering of UniSA for his suggestions on the performance simulations of the decoder.

REFERENCES

[1] R. G. Gallager, *Low-Density Parity-Check Codes*. Cambridge, Mass: Monogram, 1963.

[2] S. Johnson, *Introducing Low-Density Parity-Check Codes*. Australia: University of Newcastle, 2006.

[3] S. Lin and D. J. Costello, *Error Control Coding: Fundamentals and Applications*. New Jersey: Prentice Hall, 2004.

[4] B. Reiffen, "Sequential Decoding for Discrete Input Memoryless Channels," *IRE Trans. Inf. Theory*, vol. 8, no. 3, pp. 208-220, April 1962.

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

[6] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 47, pp. 619-637, February 2001.

[7] J. Nguyen, B. Nikolic, and E. Yeo, *Design of a low density parity check iterative decoder*. University of California, Berkley: EECS, College of Engineering, 2002.

[8] S. Hong and W. Stark, "Design and implementation of a low complexity VLSI turbo-code decoder architecture for low energy mobile wireless communications," *J. VLSI Signal Processing*, vol. 24, pp. 43-57, 2000.

[9] M. Karkooti, P. Radosavljevic, and J. R. Cavallaro, "Configurable, High Throughput, Irregular LDPC Decoder Architecture: Tradeoff Analysis and Implementation," *Proc. Int. Conf. Application Specific Systems, Architectures and Processors*, pp. 360-367, September 2006.

[10] M. Karkooti, P. Radosavljevic, and J. R. Cavallaro, "Configurable LDPC Decoder Architectures for Regular and Irregular Codes", *J. Signal Processing Systems,* vol. 53, pp. 73-88, October 2008.

[11] Y. Lei, L. Hui, and R. C. J. Shi, "Code Construction and FPGA Implementation of a low-error-floor multi-rate low-density Parity-check code decoder," *IEEE Trans. Circuits & Systems I*, vol. 53, pp. 892-904, April 2006.

[12] A. Darabiha, C. A. Carusone, R. F. Kschischang, and E. S. Rogers, "Multi-Gbit/sec Low Density Parity Check Decoders with Reduced Interconnect Complexity," *Proc. IEEE Int. Symp. Circuits & Systems*, vol. 5, pp. 5194-5197, 2005.

[13] A. Darabiha, C. A. Carusone, and R. F. Kschischang, "Block-Interlaced LDPC Decoders With Reduced Interconnect Complexity," *IEEE Trans. Circuits & Systems I I*, vol. 55, pp. 74-78, January 2008.

[14] J. Sha, M. Gao, Z. Zhang, Li Li, and Z. Wang, "An FPGA implementation of array LDPC decoder," *Proc. IEEE Asia Pacific Conf. Circ. & Systems*, pp. 1675-1678, December 2006.

[15] Mauro Cocco, "A scalable architecture of LDPC Decoding," *Proc. Design, Automation & Test in Europe Conf.*, vol. 3, pp. 88-93, February 2004.

[16] J. A. Wicks and J. R. Armstrong, "Efficiency ratings for VHDL behavioral models," *IEEE Proc. Southeasrcon'98*, pp. 401-404, April 1998.

[17] M. Eroz, F.W. Sun, and L.N. Lee, "DVB-S2 low density parity check codes with near Shannon limit performance," *Int. J. Satellite Communications and Networking*, pp. 269-279, 2004.

[18] W. E. Ryan, *An Introduction to LDPC Codes*. USA: The University of Arizona, 2003.

[19] T. Mohsenin and B. M. Baas, "Split-Row: A reduced complexity, high throughput LDPC decoder architecture," *Proc. Int. Conf. on Computer Design (ICCD 2006)*, San Jose, CA, pp. 220-225, 1-4 Oct 2007.

[20] Z. Jianguang, F. Zarkeshvari, and A. H. 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, April 2005.

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

[22] Sae-Young Chung, T. J. Richardson, and R. L. Urbanke, "Analysis of Sum-Product Decoding of Low-Density Parity-Check Codes Using a Gaussian Approximation," *IEEE Trans. Info. Theory*, vol. 47, no. 2, pp 657-670, 2001.

[23] S. M. Aziz and S. Sharma, "New Methodologies for High Level Modeling and Synthesis of Low Density Parity Check Decoders", *Proc. 11th Int. Conf. on Computers and IT (ICCIT 2008)*, Khulna, pp. 276-281, 24-27 December 2008.

[24] D. Sridhara and T. E. Fuja, "LDPC Codes Over Rings for PSK Modulation," *IEEE Trans. Inf. Theory*, vol. 51, no. 9, pp. 3209-3220, September 2005.

[25] J. K. S. Lee and J. Thorpe, "Memory-Efficient Decoding of LDPC Codes," *Proc. IEEE Int. Symp. on Information Theory (ISIT 2005)*, Adelaide, Australia, pp. 459-463, 4-9 November 2005.

Syed Mahfuzul Aziz received Bachelor and Masters Degrees, both in electrical & electronic engineering, from Bangladesh University of Engineering & Technology (BUET) in 1984 and 1986 respectively. He received a Ph.D. degree in electronic engineering from the University of Kent (UK) in 1993 and a Graduate Certificate in higher education from Queensland University of Technology, Australia in 2002.

He was a Professor in BUET until 1999, and led the development of the teaching and research programs in integrated circuit (IC) design in Bangladesh. He joined the University of South Australia in 1999, where he is currently an associate professor and the inaugural academic director of first year engineering program. In 1996, he was a visiting scholar at the University of Texas at Austin when he spent time at Crystal Semiconductor Corporation designing advanced CMOS integrated circuits. He was a visiting professor at the National Institute of Applied Science Toulouse, France in 2006, where he has collaborations in the area of nanoscale CMOS technology modelling and integration with educational IC design tools. He has been involved in numerous industry projects in Australia and overseas, and has attracted funding from reputed research organisations such as the Australian Defence Science and Technology Organisation (DSTO), and the Pork CRC (Cooperative Research Centre), Australia. He has authored eighty five refereed research papers. His research interests include digital CMOS IC design and testability, modelling and FPGA implementation of high performance processing systems, biomedical engineering and engineering education.

Dr Aziz is a senior member of IEEE and a member of Engineers Australia. He has received numerous professional awards. These include: an *Excellent Achievement Award* in Networking and Internet System Development (1998) from the Centre of the International Co-operation for Computerisation, Japan; the International Network for Engineering Education and Research *Achievement Award* (2007); a *Citation* for outstanding contributions to student learning from both the Australian Learning & Teaching Council (2007) and the Australasian Association for Engineering Education (AaeE–2007); an AaeE *Award for Teaching Excellence - Highly Commended* (2008). He has served as member of the program committees of many international conferences. He reviews papers for the IEEE Transactions on Computer and Electronics Letters, UK. Recently he has been appointed a reviewer of the National Priorities Research Program, a flagship funding scheme of the Qatar National Research Fund.

Minh Duc Pham received B.S. degree in electronic engineering from HCM National University of Technology, Vietnam in 2003 and M.S. degree in microsystems technology from the University of South Australia in 2008. He has been working as an ASIC/FPGA engineer since 2003 for Arrive Technologies Inc, a fab-less silicon supplier of Disruptive Next Generation Solutions for PDH, SONET, SDH and Ethernet Internetworking. Mr Pham is currently working as a Research Assistant in the School of Electrical and Information Engineering of the University of South Australia. His research interests are in the fields of VLSI implementation of communication systems such as SoC for next generation networking, automation in VLSI design, forward error correction and coding theory.