Semi-Parallel Reconfigurable Architectures for Real-Time LDPC Decoding

Marjan Karkooti and Joseph R. Cavallaro

Center for Multimedia Communication
Department of Electrical and Computer Engineering
Rice University, 6100 Main St., Houston, TX 77005-1892.
{marjan, cavallar}@rice.edu

Abstract

This paper presents a semi-parallel architecture for decoding Low Density Parity Check (LDPC) codes. A modified version of Min-Sum algorithm has been used which has the advantage of simpler computations compared to Sum-Product algorithm without any loss in performance. Special structure of the parity check matrix of the proposed code leads to an efficient semi-parallel implementation of the decoder for a family of $(3,6)$ *LDPC codes. A prototype architecture has been implemented in VHDL on programmable hardware. The design is easily scalable and reconfigurable for larger block sizes. Simulation results show that our proposed decoder for a block length of* 1536 *bits can achieve data rates up to* 127 *Mbps.*

Keywords: Reconfigurable architecture, FPGA implementation, channel coding, parallel architecture, area-time tradeoffs.

1. Introduction

Future generations of wireless devices will need to transmit and receive high data rate information in real-time. This poses a challenge to find an optimal coding scheme that has good performance and can be efficiently implemented in hardware. Error correcting codes insert redundancy into the transmitted data stream so that the receiver can detect and possibly correct errors that occur during transmission.

Low Density Parity Check(LDPC) codes are a special case of error correcting codes that have recently been receiving a lot of attention because of their very high throughput and very good decoding performance. Inherent parallelism of the decoding algorithm for LDPC codes, makes it very suitable for hardware implementation.

Gallager [4] proposed LDPC codes in the early $1960's$, but his work received no attention until after the invention of turbo codes, which used the same concept of iterative decoding. In 1996, MacKay and Neal [7] re-discovered LDPC codes. While standards for Viterbi and turbo codes have emerged for communication applications, the flexibility of designing LDPC codes allows for a larger family of codes and encoder/decoder structures. Some initial proposals for LDPC codes for DVB-S2 are emerging [6].

In the last few years some work has been done on designing architectures for LDPC coding. This area is still very hot and researchers are looking for the best design in the trade-offs between area, time, power consumption and performance. Here we mention some of the most related work in this area. Blanksby and Howland [1] directly mapped the Sum-Product decoding algorithm to hardware. They used the fully parallel approach and connected all the functional units with wires regarding the Tanner graph connections. Although this decoder has very good performance, the routing complexity and overhead makes this approach infeasible for larger block lengths (e.g. more than 1000 bits). Also, implementation of all the processing units enlarges the area of the chip.

Another approach is to have a semi-parallel decoder, in which the functional units are reused in order to decrease the chip-area. Semi-parallel architecture takes more time to decode the codeword and the throughput is lower than a fully parallel architecture. Zhang [11] offered an FPGA implementation of a $(3,6)$ regular LDPC semi-parallel decoder which achieves up to 54 Mbps symbol decoding throughput. He used a multi-layered interconnection network to access messages from memory. Mansour [8] proposed a 1055 bit, rate 0.4, $(3,5)$ regular semi-parallel decoder architecture which is low power. He used a fully-structured parity check matrix which led to a simpler memory addressing scheme than [11]. Chen [2] implemented a semi-parallel architecture for a rate $1/2$, 8088 bit irregular LDPC code both on FPGA and ASIC. They used a multiplexer network to select the special inputs for the processing units. Their architecture can achieve up to 40Mbps for

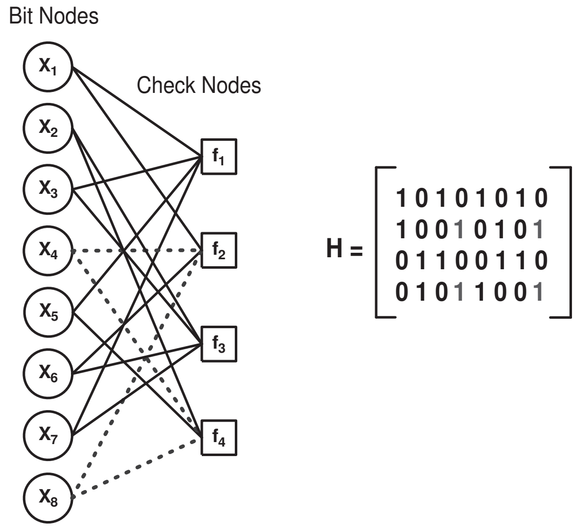

Figure 1. Tanner graph of a parity check matrix.

FPGA and 188Mbps for ASIC. All these architectures have used either Sum-Product or BCJR algorithms.

Contributions of this paper are as follows: First, we designed a structured parity check matrix which is suitable for semi-parallel hardware design and is very efficient in terms of the memory usage. Instead of storing the locations for all the $ones$ in the matrix, we can store certain "block shift values" and then restore the addresses using counters. Second, we introduce a semi-parallel architecture for decoding LDPC codes that is scalable to be used for a variety of block lengths. The decoder is the first implementation of Modified Min-Sum algorithm and achieves very good performance with low complexity.

The paper is organized as follows: Sections 2 and 3 will give an overview of LDPC codes and their encoding/decoding algorithms. Section 4 proposes the architecture for LDPC decoder. Implementation issues and results will be discussed in this part. We will show that by using a structured parity check matrix, a scalable hardware architecture has been designed. Concluding remarks will follow in section 5.

2. Low Density Parity Check Codes

Low Density Parity Check codes are a class of linear block codes corresponding to the parity check matrix H. The parity check matrix H of size$(N-K)\times N$ consists of only $zeros$ and $ones$ and is very sparse which means that the density of $ones$ in this matrix is very low. Given K information bits, the set of LDPC codewords C in the code space of length N, spans the null space of the parity check matrix H in which: $CH^T = 0$.

For a (W_c, W_r) regular LDPC code each column of the parity check matrix H has W_c $ones$ and each row has W_r $ones$. If degrees per row or column are not constant, then the code is *irregular*. Some of the irregular codes have shown better performance than regular ones [3], but irregularity results in more complex hardware and inefficiency in terms of re-usability of functional units. In this work we have considered regular codes to achieve full utilization of processing units. Code rate R is equal to K/N which means that $(N-K)$ redundant bits have been added to the message so as to correct the errors.

LDPC codes can be represented effectively by a bipartite graph called a "Tanner" graph. There are two classes of nodes in a Tanner graph, "Bit Nodes" and "Check Nodes". The Tanner graph of a code is drawn according to the following rule: "Check node $f_j, j = 1, ..., N-K$ is connected to Bit node $x_i, i = 1, ..., N$ whenever element h_{ji} in H (parity check matrix) is a one". Figure 1 shows a Tanner graph made for a small parity check matrix H. In this graph each Bit node is connected to two check nodes (Bit degree=2) and each Check node has a degree of $four$.

3. Encoding and decoding

In order to encode a message M of K bits with LDPC codes, one might compute $C = MG$ in which C is the N-bit codeword and $G_{K\times N}$ is the generator matrix of the code. At first glance, encoding may seem to be a computationally extensive task, but there exist some reduced complexity algorithms for encoding of the LDPC codes [10]. In this paper, our focus is on the decoder. We will discuss the issues in decoder design in more detail.

Min-Sum algorithm is an approximation of the sum-product algorithm in which a set of calculations on a nonlinear function $\phi(x) = -log(tanh(x/2))$ is approximated by a minimum function. In the literature, it has been shown that scaling the soft information during the decoding using Min-Sum algorithm results in better performance. By using density evaluations, Heo [5] showed that scaling factor of 0.8 is optimal for $(3, 6)$ LDPC code. We call this version of the algorithm "Modified Min-Sum" algorithm.

Figure 2 shows a comparison between the performance of Sum-Product, Min-Sum and Modified Min-Sum algorithms. It can be seen that scaling the soft information not only compensates for the loss of performance because of approximation, but also results in superior performance compared to the Sum-Product algorithm, because of the reduction in overestimation error. Modified Min-Sum is used as the decoding algorithm in our architecture.

Table 1 shows a comparison between the number of calculations needed for each of the decoding algorithms for a $(3, 6)$ LDPC code in each iteration of decoding. From the table it is clear that Modified Min-Sum algorithm substi-

Table 1. Complexity comparison between algorithms per iteration.

Algorithm	Addition	Func.	Shift
Log-Sum-Prod.	$24(N-K)+7N$	$12(N-K)$	-
Min-Sum	$24(N-K)+7N$	-	-
Mod.Min-Sum	$24(N-K)+10N$	-	$6N$

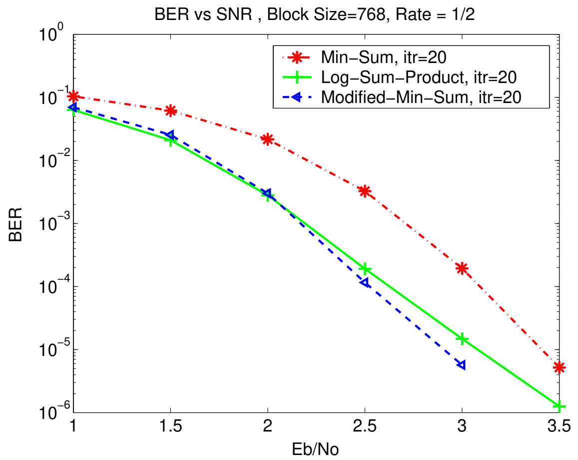

Figure 2. Comparison of different decoding algorithms.

tutes the costly function evaluations with addition and shift. Although Modified Min-Sum has a few more additions than other algorithms, it is still preferred since nonlinear function evaluations are omitted.

The function $\phi(x) = -\log(\tanh(x/2))$ is sensitive to quantization error which results in loss of the decoder performance. Either direct implementation or look up tables can be used to implement this function. Direct implementation is costly for hardware [1]. Look-up tables (LUT) are very sensitive to the number of quantization bits and number of LUT values [11]. Since in each functional unit several LUTs should be used in parallel, they can take a large area of the chip. Omitting the need for this function in the decoding, saves us some area and complexity.

All of the above iterative decoding algorithms have the following steps; they only differ in the messages that they pass among nodes.

- Initialization: Read the values from channel in each Bit node x_i and send the messages to corresponding Check nodes f_j.
- Iteration : Compute the messages at Check nodes and pass a unique message to each Bit node.
- Compute messages at Bit nodes and pass to Check nodes.
- Threshold the values calculated in each Bit node to find a codeword.
- If the codeword satisfies all the parity check equations or if maximum number of iteration is reached then stop, otherwise continue iterations.

We consider an AWGN (Additive White Gaussian Noise) channel and BPSK (Binary Phase Shift Keying) modulation of the signals.

4. Architecture design

The structure of the parity check matrix has a major role in the performance of the decoder. Finding a good matrix is an essential part of the decoder design. As mentioned earlier, parity check matrix determines the connections between different processing nodes in the decoder according to the Tanner graph. Also, degree of each node is proportional to the amount of computations that should be done in that node. For example a $(3, 12)$ LDPC has twice as many connections as a $(3, 6)$ code, which results in twice as many messages to be passed across the nodes and the memory needed to store those messages is twice the memory required for a $(3, 6)$ code. Chung et.al.[3] showed that $(3, 6)$ is the best choice for rate $1/2$ LDPC code. We have used a $(3, 6)$ code in our design.

In each iteration of the decoding, first all the Check nodes receive and update their messages and then, in the next half-iteration all the Bit nodes update their messages. If we choose to have a one-to-one relation between processing units in the hardware and Bit and Check nodes in the Tanner graph, then the design will be fully parallel. Obviously, a fully parallel approach takes a large area; but is very fast. There is also no need for central memory blocks to store the messages. They can be latched close to the processing units [1]. With this approach, the hardware design can be fixed to relate to a special case of the parity check matrix.

Table 2 shows a comparison between the resources for a parallel, semi-parallel or serial implementation of the decoder. In this table , W_c is the degree of Bit nodes, W_r is the degree of the Check nodes, b is the number of the bits per message and S is the folding factor for the semi-parallel design.

Implementing LDPC decoding algorithm in fully-serial architecture has the smallest area since it is sufficient to have just one Bit Functional Unit (BFU) and one Check Functional Unit (CFU). The fully-serial approach is suitable for Digital Signal Processors (DSPs) in which there are only a few functional units available to use. However, speed of the decoding is very low in a serial decoder.

To balance the trade-off between area and time, the best strategy is to have a semi-parallel design. This involves the creation of "l_c" CFUs and "l_b" BFUs, in which

Table 2. LDPC decoder hardware resource comparison.

Design Parameters	Fully Parallel	Semi Parallel	Fully Serial
Code Length	N	N	N
Information Length	K	K	K
Code Rate	K/N	K/N	K/N
BFU	N	N/S	1
CFU	$N-K$	$(N-K)/S$	1
Memory Bit	$(W_c+1)Nb$	$(W_c+1)Nb$	$(W_c+1)Nb$
Wire	$2(W_c+1)Nb$	$(W_c+1)Nb/S$	$2(W_c+W_r)b$
Time Per Iteration	T	ST	$T/2(2N-K)$
Counter (Address Generator)	0	$W_r(W_c+1)$	1
Address Decoder (for Memories)	0	$W_r(W_c+1)$	1
Memory Type	Scattered Latches	Several Memory Blocks	One Memory Block

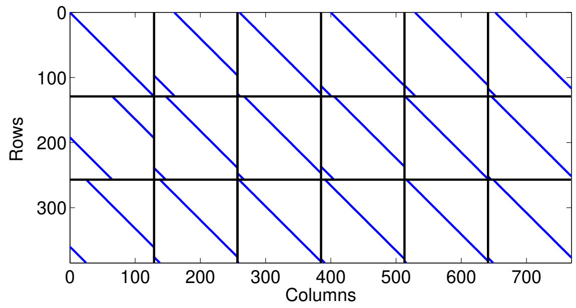

Figure 3. Parity Check Matrix of a (3,6) LDPC code.

$l_c << N-K$ and $l_b << N$ and then the reuse of these units throughout decoding time. For semi-parallel design, the parity check matrix should be structured in order to enable re-usability of units. Also, in order to design a fast architecture for LDPC decoding, we should first design a good H matrix which results in good performance. Following the block-structured design similar to [8], we have designed H matrices for $(3,6)$ LDPC codes.

Figure 3 shows the structured parity check matrix that has been used in this paper. The matrix consists of $(3\times 6 = 18)$ blocks of size s in which s is a power of two. Each $s\times s$ block is an identity matrix that has been shifted to the right a_{mn} times, $m = 1,...,3, n = 1,...,6$. The shift values can be any value between 0 and $s-1$, and have been determined with a heuristic search for the best performance in the codes of the same structure. Our approach is different from [8] since the sub-block length is not a prime number. Also, shifts are determined by simulations and searching for the best matrix that satisfies our constraints (with the highest girth [9]). Figure 4 shows a comparison between the performance of two sets of $(3,6)$ LDPC codes of rate $1/2$ and block lengths of 768 and 1536 designed with above structure. To give some comparison points [11] uses a LDPC code of length 1020 which achieves BER of 10^{-5} and 0.3×10^{-5} for SNR of 2.7 and 3.1dB respectively.

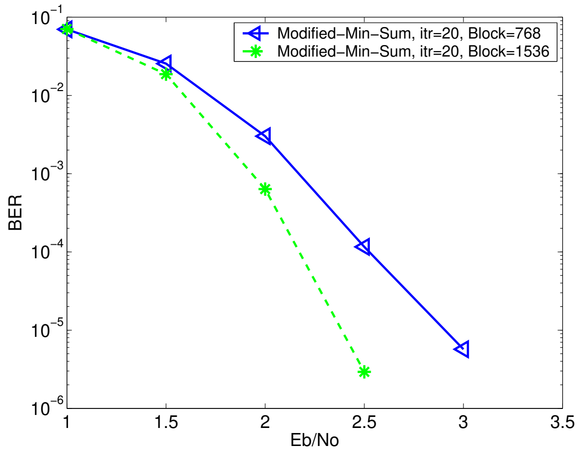

Figure 4. Simulation results for the decoding performance of different block lengths.

4.1. Reconfigurable architecture

For LDPC codes, increasing the block length results in a performance increase. That is because the Bit and Check nodes receive some extrinsic information from the nodes that are very far from them in the block. This increases the error correction ability of the code. Having a scalable architecture which can be scaled for different block lengths enables us to choose a suitable block length N for different applications. Usually N is in the order of $500\sim 10000$ for practical uses. Our design is flexible for block lengths of $N = 6\times 2^{\theta}$ for a (3,6) LDPC code. As an example for $\theta = 8$, N is equal to 1536. By choosing different values for θ we can get different values for the block length. We will discuss the statistics and design of the architecture for block length 1536 bits. The proposed LDPC decoder can be scaled for any block length $N = 6\times 2^{\theta}$. The largest block length is determined with the physical limitations of the platform such as FPGA or ASIC. It should be noted that changing the block length is an off-line process, since a new bitstream file should be compiled to download to the FPGA.

The overall architecture for a $(3,6)$ LDPC decoder is shown in figure 5. This semi-parallel architecture consists

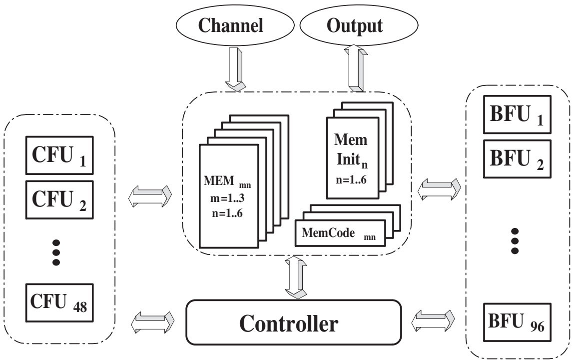

Figure 5. Overall architecture of a semi-parallel LDPC decoder.

of $W_c \times W_r = 3 \times 6 = 18$ memory units ($MEM_{mn}, m = 1, ..., W_c, n = 1, ..., W_r$) to store the values passed between Bit nodes and Check nodes and Wr memories ($MemInit_n$) to store the initial values read from the channel. $MemCode_{mn}$ stores the code bits resulted from each iteration of the decoding. This architecture has several Bit Functional Units and Check Functional Units that can be reused in each iteration. Since the code rate is $1/2$, there are twice as many columns in the parity check matrix as rows, which means that the number of BFUs should be two times the number of CFUs to balance the time spent on each half-iteration. For the block length of 1536, we have chosen the parallelism factor of $S = 16$, which means that we have $(1536 - 768)/16 = 48$ CFUs and 96 BFUs. Each of these units is used 16 times in each iteration. These units perform computations on different input sets that are synchronized by the controller unit.

Figure 6 shows the interconnection between memories, address generators and CFUs that are used in the first half of iterations. In each cycle $ADGC_{mn}$ generate addresses of the messages for the CFUs. Split/Merge (S/M) units pack/unpack messages to be stored/read to/from memories. To increase the parallelism factor, it is possible to pack more messages (i.e. δ) to put to a single memory location. This poses a constraint on the design of H matrix, since the shift values should all be multiples of δ. The finite state machine "control unit" supervises the flow of messages in/out of memories and functional units.

Figure 7 shows the Architecture for Check Functional Units (CFUs). Each CFU has $Wr = 6$ inputs and 6 outputs. This unit computes the minimum among different choices of five out of six inputs. CFU outputs the result to output ports corresponding to each input which is not included in the set. For example $out1$ is the result of:

$$out1 = \min(abs(in2), abs(in3), ..., abs(in6)) \quad (1)$$

in which $abs(.)$ is the absolute value function.

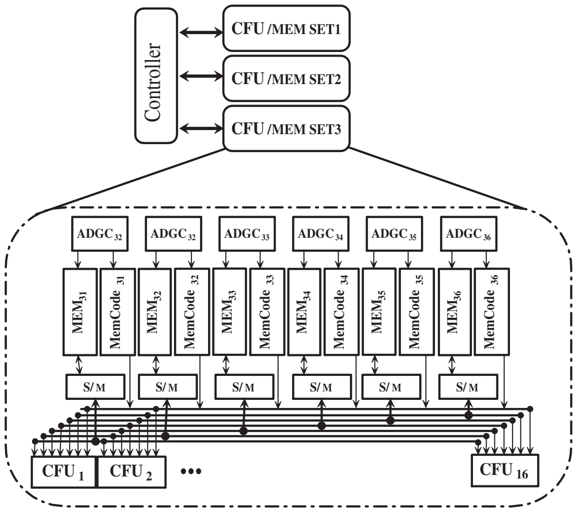

Figure 6. Connections between memories, CFUs and address generators.

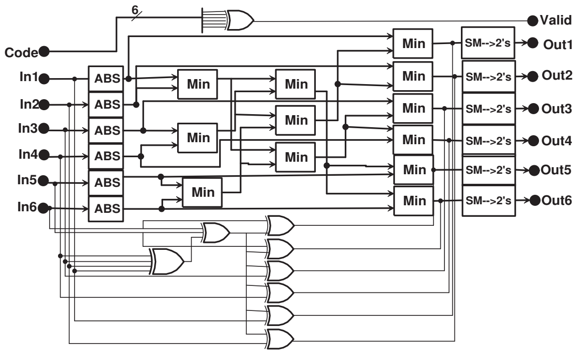

Figure 7. Check Functional Unit (CFU) architecture

Also, during the computations of the current iteration, CFU checks the code bits resulting from the previous iteration to check if the code bits satisfy the corresponding parity check equation (step 5 of the decoding algorithm). After the first half of the iteration is complete, the result of all parity checks on the codeword will be ready too. With this strategy, computations in Check nodes and Bit nodes can be done continuously without the need to wait for checking the codeword resulting from the previous iteration. This increases the speed of the decoding.

The interconnection between BFUs and memory units and address generators $ADGB$ is shown in figure 8. Locations of the messages in the memories are such that a single address generator can service all the BFUs. Controller makes sure that all the units are synchronized.

The architecture of a Bit Functional Unit is shown in the figure 9. This unit adds different combinations of its inputs

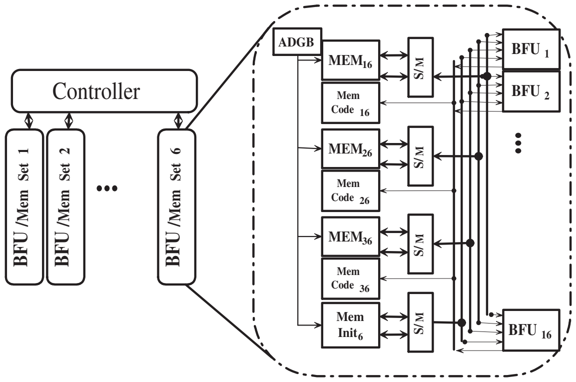

Figure 8. Connections between memories, BFUs and address generators.

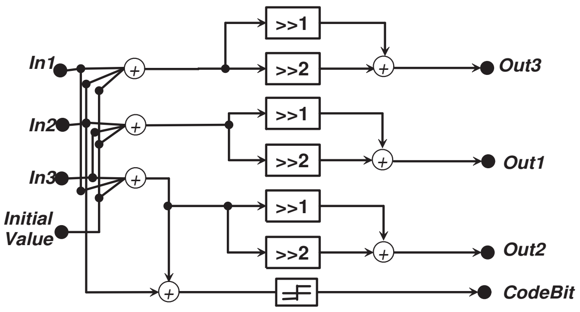

Figure 9. Bit Functional Unit (BFU) architecture

and scales them with a scaling factor of 0.75 which is done with shift and addition. Also, it thresholds the summation of its inputs to find the code-bit corresponding to that Bit node.

This architecture can also be used for the structured irregular codes with some minor modifications. For example, assume that the parity check matrix of the irregular code is similar to figure 3, but it has 4 block rows and 7 block columns in which some of the blocks are full of zeros, then we can have an irregular code with row degrees of $6, 7$ and column degrees of $3, 4$. We should add some circuitry so that for the blocks full of zero in the parity check matrix, it sends a zero message to the corresponding inputs of the BFU/CFUs. In this case the BFUs will have 5 input/outputs and CFUs will have 8 input/outputs.

4.2. FPGA architecture

For real-time hardware, fixed-point computations are less costly than floating point. A fixed-point decoder uses quantized values of the soft information. There is a trade-off between the number of quantization bits, area of the design, power consumption and performance. Using more bits de-

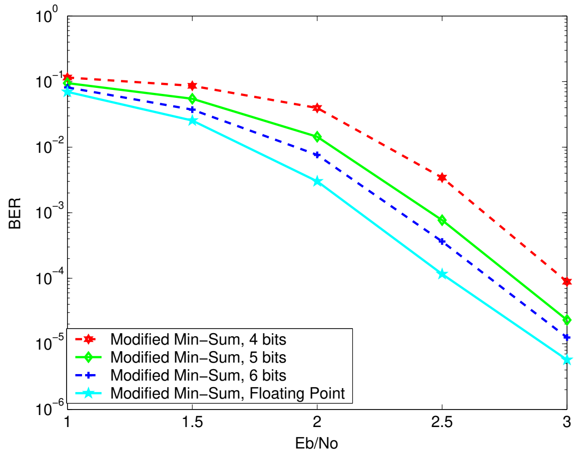

Figure 10. Comparison between different quantization levels.

creases the bit error rate, but increases the area and power consumption of the chip. Also, depending on the nature of the messages, the number of bits used for integer or fractional part of the representation is important. Our simulations show that using 5 bits for the messages is enough for good performance. These messages will be divided into one sign bit, two integer bits and two fractional bits. Figure 10 shows the performance of the decoder using $4, 5, 6$ bits and the floating point version.

Since the memory blocks in the FPGA have no more than two ports, we need to increase the number of the message read/writes in each clock cycle in the dual-port memories. We pack eight message values and store them in a single memory address. This enable us to read $2 \times 8 = 16$ messages per memory per cycle.

A prototype architecture has been implemented by writing VHDL (Hardware Description Language) code and targeted to a Xilinx VirtexII-3000 FPGA. Table 3 shows the utilization statistics of the FPGA. Based on the Leonardo Spectrum synthesis tool report, the maximum Clock frequency of this decoder is 121 MHz. Considering the parameters of our design, it takes 96 cycles to initialize the memories with the values read from the channel, 32 cycles for each CFU and BFU half-iterations, and 48 cycles to send out the resulting codeword. Assuming that the decoder does μ iterations to finish the decoding, the data rate can be calculated with the following equation:

$$Datarate = \frac{(blocklength \times decoder frequency)}{cycles} \quad (2)$$

and,

$$\begin{aligned} cycles &= \frac{N}{2\lambda} + \mu\left(\frac{2(N-K)}{l_c} + \frac{2N}{l_b}\right) + \frac{N-K}{l_c} \\ &+ \frac{(N-K)}{2\lambda} = (96 + \mu \times (32+32) + 32 + 48) \end{aligned}$$

Table 3. Xilinx VirtexII-3000 FPGA utilization statistics.

Resource	Used	Utilization rate
Slices	11,352	79%
4 input LUTs	20,374	71%
Bonded IOBs	100	14 %
Block RAMs	66	68 %

In which N is the block length, K is number of the information bits, λ is the packing ratio for the messages in the memories, l_b is number of BFUs, and l_c is the number of CFUs. With maximum number of iterations, $\mu = 20$ (worst case), the data rate can be 127 Mbps. This architecture is suitable for a family of codes with similar structure as described earlier and different block lengths, parallelism ratios and message lengths.

Changing the block-size of the codeword changes the sizes of the memory blocks. If we assume that the codes are still $(3, 6)$ and have a parity check matrix similar to figure 3, then all the CFUs, BFUs and address generators can be used for the new architecture. The size of the memories changes and there will be a slight modification in the address generator units because they should address a different number of memory words. This can be done by changing the size of the counters used in the address generators. Since the counters are parametric in the VHDL code, this can be done with a new compilation of the code using these new values.

4.3. LabVIEW implementation

An alternative design has been implemented using LabVIEW FPGA from National Instruments. This architecture has the same characteristics as the VHDL version. The only difference is that it is implemented using the graphical GUI of LabVIEW and runs in the co-simulation mode. In this model, data input-output is done in the host PC and decoding in the FPGA. This enables us to use the LDPC decoder in our end-to-end communication testbed at the Center for Multimedia Communication (CMC) at Rice University and connect it directly to National Instruments radios and other hardware.

5. Conclusion

A semi-parallel architecture for decoding LDPC codes has been designed and implemented on Xilinx VirtexII FPGAs. The special structure of the parity check matrix simplifies the memory addressing and results in the efficient storage of the matrix. Modified-Min-Sum algorithm has the advantage of good decoding performance with simple computations in the functional units. The semi-parallel architecture is easily scalable for different block sizes, message lengths and parallelism factors. For a $(3, 6)$ LDPC code with the block length of 1536 bits, the decoder achieves a data rate of up to 127 Mbps.

6. Acknowledgements

This work was supported in part by a National Instruments Fellowship, and by NSF under grants ANI-9979465, EIA-0224458, and EIA-0321266.

References

[1] A. Blanksby and C. Howland. A 690-mW 1-Gbps 1024-b, Rate-1/2 Low-Density Parity-Check Code Decoder . *Journal of Solid State Circuits*, 37(3):404–412, Mar 2002.

[2] Y. Chen and D. Hocevar. A FPGA and ASIC Implementation of Rate 1/2 8088-b Irregular Low Density Parity Check Decoder. *IEEE Global Telecommunications Conference, GLOBECOM*, 2003.

[3] S. Chung, T. Richardson, and R. Urbanke. Analysis of Sum-Product Decoding of Low-Density Parity-Check Codes Using a Gaussian Approximation. *IEEE Trans. on Inform. Theory*, 47(2):657–670, Feb 2001.

[4] R. Gallager. Low-Density Parity-Check Codes. *IRE Trans. on Inform. Theory*, 8:21–28, Jan 1962.

[5] J. Heo. Analysis of Scaling Soft Information on Low Density Parity Check Codes. *Elect. Letters*, 39(2):219–221, Jan 2003.

[6] L. Lee. LDPC Code, Application to the Next Generation Wireless Communication Systems, 2003. Fall VTC, Panel Pres. by Hughes Network.

[7] D. MacKay and R. Neal. Near Shannon Limit Performace of Low Density Parity Check codes. In *Elec. Letters*, volume 32, pages 1645–6, Aug 1996.

[8] M. Mansour and N.Shanbhag. Low Power VLSI Decoder Architectures for LDPC Codes. *Proc. of the Int. Symp. on Low Power Electronics and Design.*, pages 284–289, 2002.

[9] Y. Mao and A. Banihashemi. A Heuristic Search for Good Low-Density Parity-Check Codes at Short Block Lengths. *IEEE Int. Conf. on Comm.*, pages 41–44, Jun 2001.

[10] T. R. R. Urbanke. Efficient Encoding of Low-Density Parity Check Codes. *IEEE Trans. on Inform. Theory*, 47(2):638–656, Feb 2001.

[11] T. Zhang. *Efficient VLSI Architectures for Error-Correcting Coding*. PhD thesis, University of Minnesota, Jul 2002.