Low Power IEEE 802.11n LDPC Decoder Hardware

Merve Peyic, Hakan A. Baba, Ilker Hamzaoglu, Mehmet Keskinoz

Electronics Engineering, Sabanci University, Tuzla 34956 Istanbul, Turkey
mervep@su.sabanciuniv.edu, altugb@su.sabanciuniv.edu, hamzaoglu@sabanciuniv.edu, keskinoz@sabanciuniv.edu

Abstract—In this paper, we present a low power hybrid low-density-parity-check (LDPC) decoder hardware implementing layered min-sum decoding algorithm for IEEE 802.11n Wireless LAN Standard. The LDPC decoder hardware, which has 27 check node datapaths and 24x162 variable node memory, is implemented in Verilog HDL and verified to work correctly in a Xilinx Virtex II FPGA. For 648 block length and 1/2 code rate, on a Xilinx Virtex II FPGA, the LDPC decoder hardware implementation works at 83.5 MHz and it can process 60.68 Mbps. For 648 block length and 5/6 code rate, on a Xilinx Virtex II FPGA, the LDPC decoder hardware implementation works at 71.5 MHz and it can process 113.78 Mbps. The power consumption of the implementation on a Xilinx Virtex II FPGA is estimated as 2052 mW for 648 block length and 1/2 code rate and 1989 mW for 648 block length and 5/6 code rate using Xilinx XPower tool. In this paper, we propose two novel techniques, sub-matrix reordering and differential shifting, for reducing the power consumption of a LDPC decoder hardware. We applied glitch reduction, sub-matrix reordering and differential shifting techniques to our LDPC decoder hardware. These techniques do not affect the bit error rate (BER) of a LDPC decoder. For block length 648 and code rate 1/2, these three techniques together reduced the power consumption of the LDPC decoder hardware in total by 23.7% to 1,565.84 mW. For block length 648 and code rate 5/6, they together reduced the power consumption of the LDPC decoder hardware in total by 38.98% to 1,214.22 mW.

I. INTRODUCTION

In communication systems, Forward Error Correction (FEC) techniques are used to detect and/or correct the errors on the received bit streams. Low-density-parity-check (LDPC) codes are an example of ECCs which were first proposed by Robert Gallager in 1960 [1] and rediscovered by MacKay after 30 years in mid 1990s [2]. They are now used as error correction code in many communication standards such as IEEE 802.11n, the recently developed wireless LAN standard.

The parity check matrix of an LDPC code determines the BER, the throughput and the complexity of the LDPC decoder. The parity check matrixes used in IEEE 802.11n standard have layered structures and they consist of shifted versions of identity matrixes concatenated to form 12 different matrixes for 648, 1296 and 1944 block lengths and 1/2, 2/3, 3/4 and 5/6 code rates [3]. The 324x648 parity check matrix used in IEEE 802.11n standard for 648 block length and 1/2 code rate is shown in Figure 1. A layer consists of multiple rows (parity check equations) and concatenation of these layers forms the whole parity check matrix. For example, the parity check matrix for 1/2 code rate consists of 12 layers and each layer is composed of 24 sub-matrixes of size 27x27 which are either null matrixes or shifted versions of identity matrixes.

Several decoding algorithms for LDPC codes have been proposed in the literature [4]. In this paper, we used the min-sum decoding algorithm with layered belief propagation in log-likelihood ratio (LLR) domain, because it satisfies the throughput and BER requirements of IEEE 802.11n standard and it has low computational complexity and fast convergence.

Since a parallel LDPC decoder hardware is not scalable for large parity check matrixes [5], in this paper, we present a low power hybrid LDPC decoder hardware for IEEE 802.11n wireless LAN standard. The LDPC decoder hardware has 27 check node datapaths and 24x162 variable node memory. The hardware is implemented in Verilog HDL and verified to work correctly in a Xilinx Virtex II FPGA. For 648 block length and 1/2 code rate, on a Xilinx Virtex II FPGA, the LDPC decoder hardware implementation works at 83.5 MHz and it can process 60.68 Mbps if it does 3 iterations (36 sub-iterations) for each codeword. For 648 block length and 5/6 code rate, on a Xilinx Virtex II FPGA, the LDPC decoder hardware implementation works at 71.5 MHz and it can process 113.78 Mbps if it does 3 iterations (12 sub-iterations) for each codeword.

The power consumption of the implementation on a Xilinx Virtex II FPGA is estimated as 2052 mW for 648 block length and 1/2 code rate and 1989 mW for 648 block length and 5/6 code rate using Xilinx XPower tool. In this paper, we propose two novel techniques, sub-matrix reordering and differential shifting, for reducing the power consumption of an LDPC decoder hardware. We applied glitch reduction, sub-matrix reordering and differential shifting techniques to our LDPC decoder hardware. These techniques do not affect the BER of an LDPC decoder. For block length 648 and code rate 1/2, these three techniques together reduced the power consumption of the LDPC decoder hardware in total by 23.7% to 1,565.84 mW. For block length 648 and code rate 5/6, they together reduced the power consumption of the LDPC decoder hardware in total by 38.98% to 1,214.22 mW.

Several hybrid LDPC decoder hardware architectures are proposed in the literature [6, 7, 8, 9, 10, 11, 12]. Some of these LDPC decoders are proposed for IEEE 802.11n standard. Our LDPC decoder hardware is similar to the LDPC decoder hardware proposed in [8] for DVB-S2 standard. The power consumption is only reported in [11] for an ASIC implementation. We, therefore, could not compare the power consumption of our LDPC decoder hardware with the other LDPC decoders.

The rest of the paper is organized as follows. Section II describes LDPC codes and layered min-sum LDPC decoding algorithm. The LDPC decoder hardware architecture is presented in Section III. The power consumption reduction for the LDPC decoder hardware is explained in Section IV. The implementation results are given in Section V. Section VI concludes the paper.

0	–	–	–	0	0	–	–	0	–	–	0	1	0	–	–	–	–	–	–	–	–	–	–
22	0	–	–	17	–	0	0	12	–	–	–	–	0	0	–	–	–	–	–	–	–	–	–
6	–	0	–	10	–	–	–	24	–	0	–	–	–	0	0	–	–	–	–	–	–	–	–
2	–	–	0	20	–	–	–	25	0	–	–	–	–	–	0	0	–	–	–	–	–	–	–
23	–	–	–	3	–	–	–	0	–	9	11	–	–	–	–	0	0	–	–	–	–	–	–
24	–	23	1	17	–	3	–	10	–	–	–	–	–	–	–	–	0	0	–	–	–	–	–
25	–	–	–	8	–	–	–	7	18	–	–	0	–	–	–	–	–	0	0	–	–	–	–
13	24	–	–	0	–	8	–	6	–	–	–	–	–	–	–	–	–	–	0	0	–	–	–
7	20	–	16	22	10	–	–	23	–	–	–	–	–	–	–	–	–	–	–	0	0	–	–
11	–	–	–	19	–	–	–	13	–	3	17	–	–	–	–	–	–	–	–	–	0	0	–
25	–	8	–	23	18	–	14	9	–	–	–	–	–	–	–	–	–	–	–	–	–	0	0
3	–	–	–	16	–	–	2	25	5	–	–	1	–	–	–	–	–	–	–	–	–	–	0

Figure 1. Parity Check Matrix for 648 block length and 1/2 code rate

II. LDPC Codes

LDPC decoding is done based on a parity check matrix which consists of "0"s and "1"s defining the parity check equations. An example 4x8 parity check matrix is shown in Figure 2. An MxN parity check matrix has M parity check equations and N variables. For an MxN parity check matrix, M check nodes and N variable nodes exchange information between themselves iteratively according to the LDPC decoding algorithm. "1"s in the parity check matrix determine the connections between the variable nodes and the check nodes. The information exchange is done only between the nodes connected to each other. LDPC decoding process for the 4x8 parity check matrix is shown in Figure 3.

Variable nodes receive soft information, the likelihood ratio of probabilities of that bit being 1 or 0, from the channel and this information is iteratively passed between check nodes and variable nodes to satisfy the parity check equations specified by the parity check matrix [1, 2]. This operation can be done in logarithmic domain to simplify multiplication operations to addition operations in which case the decoder gets log-likelihood ratios (llr) from the channel [4]. This algorithm can be further simplified to min-sum decoding algorithm with a small degradation in BER. The steps of the min-sum decoding algorithm are shown below:

i. Take the llr values from the channel for each variable node as the initial variable node messages.

$$Q_n = LLR(n) \tag{1}$$

ii. Update each check node with the variable node messages they are connected to, according to the min-sum algorithm.

$$R_{mn} = \prod_{n' \in C \backslash n} sign(Q_{n'm}) \times \min_{n' \in C \backslash n}(Q_{n'm}) \tag{2}$$

where C is the set of variable nodes connected to a check node.

iii. Update each variable node with the check node messages they are connected to.

$$Q_{nm} = LLR(n) + \sum_{m' \in V \backslash m} R_{m'n} \tag{3}$$

where V is the set of check nodes connected to a variable node.

iv. After each layer, calculate the decoder output by summing up all check node messages for each variable node.

$$Q_n = LLR(n) + \sum_{m' \in V} R_{m'n} \tag{4}$$

v. Finally the hard decision is made according to the soft decoder outputs.

When the min-sum decoding algorithm is implemented using a hybrid LDPC decoder hardware, its BER performance can be improved by using layered decoding technique in which message updates are not done only after finishing the whole parity check matrix but also after finishing each layer of the parity check matrix [6, 7]. The layered decoding can be used for the parity check matrixes with layered structure such as the parity check matrixes used in IEEE 802.11n standard. For example, for the parity check matrix used for 648 block length and 1/2 code rate in IEEE 802.11n standard, after the 27 check nodes finishes the min-sum algorithm for the variable nodes they are connected to in one layer, these variable nodes are updated and the 27 check nodes uses these updated messages for the next layer. Since message updating is also done after finishing each layer in an iteration, the time spent for processing a layer is called a sub-iteration. Therefore, for the parity check matrix used for 1/2 code rate, 12 sub-iterations are done in one iteration.

III. LDPC Decoder Hardware

In this paper, we present a hybrid LDPC decoder hardware implementation of the parity check matrixes specified in the IEEE 802.11n standard for 648 block size [3]. As shown in Figure 4, our hardware architecture is similar to the LDPC decoder hardware proposed in [8]. Since sub-matrix size of the parity check matrixes is 27x27, we used 27 check node datapaths for implementing the min-sum decoding algorithm for one layer in parallel. After variable-node updates are finished for one layer, the next layer of the parity check matrix is processed resulting in a hybrid LDPC decoder implementation.

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

Figure 2. A 4x8 Parity Check Matrix

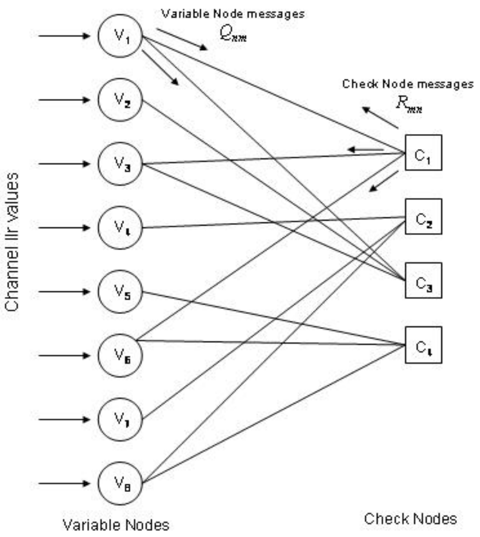

Figure 3. LDPC Decoding for the 4x8 Parity Check Matrix

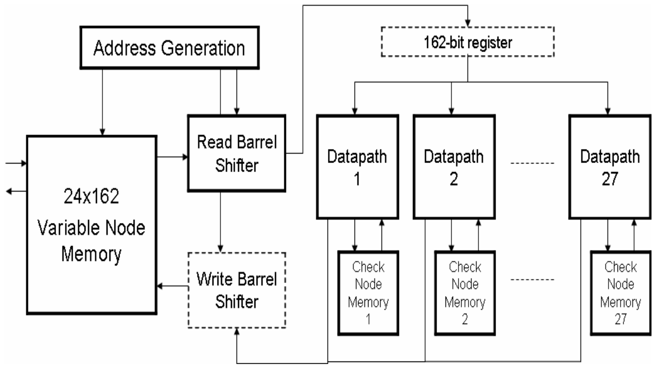

Figure 4. LDPC Decoder Hardware Architecture

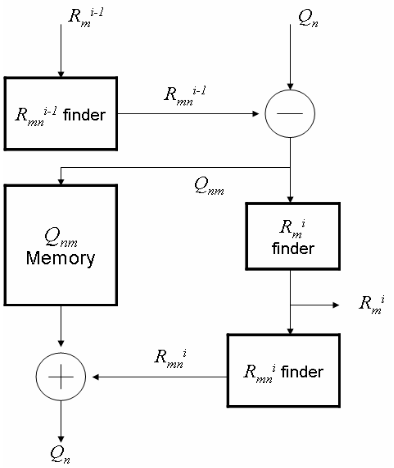

Figure 5. Check Node Datapath

The hardware architecture consists of a 24x162 variable node memory, 2 barrel shifters, 27 check node datapaths and 27 12x38 check node memories. 24x162 bit memory is used to store the 648 variable node messages each one being 6-bit including 1 sign bit. The variable node memory is organized such that in each word 27*6 = 162 bit messages are stored to send 27 variable node messages to 27 check node datapaths in parallel.

Since the sub-matrixes of the IEEE 802.11n standard are shifted versions of 27x27 identity matrixes, before sending variable node messages, the word has to be shifted by the read barrel shifter to send the correct variable node messages to each check node datapath. Then the updated variable node messages are written back to memory after they are shifted back to their original position by write barrel shifter.

In the 648 block-length and 5/6 code rate parity check matrix, each check node is connected to 22 variable nodes. Therefore to compute the check node message as in equation 2, each check node datapath is sent the variable node messages in 22 cycles. In our decoder hardware, instead of storing all variable node messages for every check node, we only store their sum for every variable node, calculated as in equation 4.

Then, as shown in Figure 5, in the check node datapath, the check node message, sent in the previous iteration, is subtracted from the total variable node message to extract the individual variable node message for that check node, as in equation 5.

$$Q_{nm}{}^{i} = Q_n - R_{mn}{}^{i-1} \tag{5}$$

After calculating Q_{nm} for all 22 variable node messages, the block "*$R_m{}^i$ finder*" finds the minimum and one-but-minimum magnitudes among the 22 Q_{nm} messages and sends a 38-bit length message containing 4-bit min and one-but-min magnitudes, 5-bit index of the minimum, 24-bit signs of 24 variable nodes, of which only 22 are used for each layer, and 1-bit for xor of the signs of 22 variable nodes. This 38-bit compressed message is stored in 4x38 check node memories. The "*$R_{mn}{}^{i-1}$ finder*" and "*$R_{mn}{}^i$ finder*" in a check node datapath are used to decompress the 38 bit R_m messages and find the individual check-to-variable node messages. The 24x5 Q_{nm} memories keep the Q_{nm} values which will later be added with R_{mn} to finally update the variable node sending the Q_n to the variable node memory.

IV. Power Consumption Reduction

The LDPC decoder hardware is implemented in Verilog HDL. The Verilog RTL design is synthesized to a 2V8000ff1157 Xilinx Virtex II FPGA with speed grade 5 using Mentor Graphics Precision RTL 2005b. The resulting netlist is placed and routed to the same FPGA using Xilinx ISE 8.2i.

The power consumption of the LDPC decoder hardware implementation on a Xilinx Virtex II FPGA is estimated using Xilinx XPower tool. In order to estimate the dynamic power consumption, timing simulation of the placed and routed netlist of the LPDC decoder hardware implementation is done using Mentor Graphics ModelSim SE for 10 codewords and 10 iterations and the signal activities are stored in a VCD file. This VCD file is used for estimating the power consumption of the LDPC decoder hardware using Xilinx XPower tool.

The dynamic power consumption of the LDPC decoder hardware implementation for 648 block length, and 1/2 and 5/6 code rates on a Xilinx Virtex II FPGA at 33 MHz are shown in Table I and Table II. The dynamic power consumption of the LDPC decoder hardware is divided into three categories; signal power, logic power and clock power. Signal power is the power dissipated in routing tracks between logic blocks. Logic power is the amount of power dissipated in the parts where computations take place. Clock power is due to clock tree used in the FPGA. Since the LDPC decoder hardware is interconnection dominant, a significant amount of power, 58.37% of total power consumption of 1/2 code rate and 60.88% of total power consumption of 5/6 code rate, is dissipated in routing tracks.

In this paper, we propose two novel techniques, sub-matrix reordering and differential shifting, for reducing the power consumption of the LDPC decoder hardware.

In the hybrid LDPC decoder hardware designs, a read barrel shifter is used for shifting the current variable node values after reading them from the variable node memory and a write barrel shifter is used for shifting the new variable node values produced by the check node datapaths before writing them to the variable node memory. In differential shifting technique, new variable node values produced by check node datapaths are written to variable node memory without being shifted. Therefore, in the next iteration, the current variable node values are shifted by the difference between the previous write shift amount and the current read shift amount, i.e. the previous write shift and the current read shift are done together by the read barrel shifter.

Therefore, implementing the differential shifting technique in the LDPC decoder hardware is done by removing the write barrel shifter, by properly updating the shift amounts for the read barrel shifter and by changing the initial variable node memory organization to make it suitable for the differential shift amounts.

24	–	–	–	11	9	–	–	2	–	–	10	0	0	–	–	–	–	–	–	–	–	–	–
22	7	–	–	17	–	19	25	12	–	–	–	–	0	0	–	–	–	–	–	–	–	–	–
11	–	19	–	20	–	–	–	12	–	24	–	–	–	0	0	–	–	–	–	–	–	–	–
23	–	–	11	10	–	–	–	1	22	–	–	–	–	–	0	0	–	–	–	–	–	–	–
21	–	–	–	10	–	–	–	2	–	9	11	–	–	–	–	0	0	–	–	–	–	–	–
1	–	23	1	14	–	3	–	10	–	–	–	–	–	–	–	–	0	0	–	–	–	–	–
1	–	–	–	18	–	–	–	24	18	–	–	26	–	–	–	–	–	0	0	–	–	–	–
15	24	–	–	19	–	5	–	26	–	–	–	–	–	–	–	–	–	–	0	0	–	–	–
21	23	–	15	22	10	–	–	17	–	–	–	–	–	–	–	–	–	–	–	0	0	–	–
4	–	–	–	24	–	–	–	17	–	21	6	–	–	–	–	–	–	–	–	–	0	0	–
14	–	12	–	4	8	–	14	23	–	–	–	–	–	–	–	–	–	–	–	–	–	0	0
5	–	–	–	20	–	–	15	16	14	–	–	1	–	–	–	–	–	–	–	–	–	–	0

Figure 6. Differential Shift Amounts for the Parity Check Matrix for 648 Block Length and 1/2 Code Rate

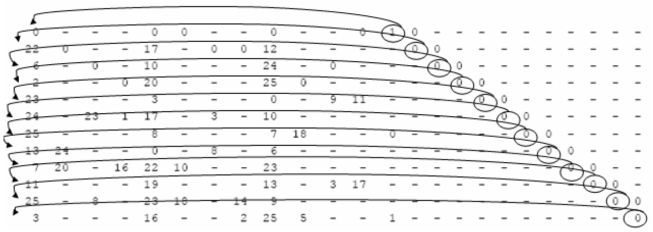

Figure 7. Sub-matrix Reordering for the Parity Check Matrix for 648 Block Length and 1/2 Code Rate

TABLE I. POWER CONSUMPTION OF LDPC DECODER HARDWARE FOR 1/2 CODE RATE

Power (mW)	Initial Hardware	Glitch Reduction	Sub-Matrix Reordering	Differential Shifting
Clock	550.49	566.23	570.64	547.47
Logic	305.11	264.83	262.62	218.83
Signal	1,198.02	1,031.13	996.70	798.80
Total	2,052.55	1,863.10	1,830.69	1,565.84

TABLE II. POWER CONSUMPTION OF LDPC DECODER HARDWARE FOR 5/6 CODE RATE

Power (mW)	Initial Hardware	Glitch Reduction	Sub-Matrix Reordering	Differential Shifting
Clock	488.43	496.11	486.34	495.23
Logic	288.62	238.73	227.00	173.64
Signal	1,211.37	794.60	768.50	544.45
Total	1,989.85	1,530.30	1,482.75	1,214.22

Since there is no write shifter, after the last layer the updated variable node messages will be written to variable node memory in the read shifted order and in the next iteration, in the first layer the variable node messages has to be read shifted by taking into account the read shift amounts of the last layer. Therefore, each variable node message received from the channel is written to a variable node memory word after shifted by the read shift amount of the last layer to make the shift amounts of the first layer consistent for all iterations. The differential shift amounts for the parity check matrix of 1/2 code rate is shown in Figure 6.

In the hybrid LDPC decoder hardware design, the sub-matrixes in one layer of a parity check matrix are processed by the check node datapaths sequentially starting from the first sub-matrix until the last sub-matrix in the parity check matrix. Processing the sub-matrixes in one layer of a parity check matrix by the check node datapaths in a different order does not affect the BER of an LDPC decoder. Therefore, in sub-matrix reordering technique, the sub-matrixes in one layer of a parity check matrix are processed by the check node datapaths in the order that results in a smaller amount of switching activity by both reading the same 162-bit variable node memory word and shifting it with the same shift amount in the consecutive clock cycles as much as possible.

As shown in Figure 1 for rate 1/2, in the parity check matrixes used in IEEE 802.11n, some sub-matrixes in consecutive layers are shifted with the same shift amount. For example, as shown in Figure 7, in the parity check matrix for 648 block length and 1/2 code rate, in both the first and the second layers the 13th sub-matrix is shifted by 0, therefore while processing the first layer we read the 13th sub-matrix the last and while processing the second layer we read the 13th sub-matrix the first in order to avoid reading a different variable node memory word which will result in unnecessary switching activity. Therefore, the sub-matrixes in this parity check matrix are processed by the check node datapaths in the below order.

Layer 1: 12 – 0 – 4 – 5 – 8 – 11 – 13
Layer 2: 13 – 0 – 1 – 4 – 6 – 7 – 8 – 14
Layer 3: 14 – 0 – 2 – 4 – 8 – 10 – 15
....

Layer 11: 22 – 0 – 2 – 4 – 5 – 7 – 8 – 23
Layer 12: 23 – 0 – 4 – 7 – 8 – 9 – 12

Glitch is a spurious transition at a node within a single cycle before the node settles to the correct logic value [13]. Unlike ASICs, in which signals can be routed using any available silicon, FPGAs implement interconnects using fixed metal tracks and programmable switches. The relative scarcity of programmable switches often forces signals to take longer routes than would be seen in an ASIC. As a result, the potential for unequal delays among signals, and hence the creation of glitches, is more likely than that in an ASIC. Thus, reducing glitches by pipelining is an effective power reduction technique for FPGAs. Pipeline registers can be inserted after the read barrel shifter, shown as dashed rectangle in Figure 4, for reducing the glitches in the LDPC decoder hardware.

We first applied glitch reduction, then applied sub-matrix reordering and finally applied differential shifting techniques to our LDPC decoder hardware implementation. These techniques do not affect the BER of an LDPC decoder. The impact of these techniques on the power consumption of LDPC decoder hardware for block length 648 and code rate 1/2 is shown in Table I and for block length 648 and code rate 5/6 is shown in Table II.

For block length 648 and code rate 1/2, glitch reduction technique reduced the power consumption of the LDPC decoder hardware by 189.45 mW, sub-matrix reordering technique further reduced the power consumption of the LDPC decoder hardware by 32.41 mW and differential shifting technique further reduced the power consumption of the LDPC decoder hardware by 264.85 mW. Therefore, these three techniques together reduced the power consumption of the LDPC decoder hardware in total by 23.7% to 1,565.84 mW.

For block length 648 and code rate 5/6, glitch reduction technique reduced the power consumption of the LDPC decoder hardware by 459.55 mW, sub-matrix reordering technique further reduced the power consumption of the LDPC decoder hardware by 47.55 mW and differential shifting technique further reduced the power consumption of the LDPC decoder hardware by 268.53 mW. Therefore, these three techniques together reduced the power consumption of the LDPC decoder hardware in total by 38.98% to 1,214.22 mW.

TABLE III. AREA OF LDPC DECODER HARDWARE FOR 1/2 CODE RATE

Category	Initial Hardware	Glitch Reduction	Differential Shifting
Function Generators	16,136	15,731	13,850
CLB Slices	11,303	11,038	10,153
DFFs	4,401	4,759	4,734
Block RAMs	116	118	118

TABLE IV. AREA OF LDPC DECODER HARDWARE FOR 5/6 CODE RATE

Category	Initial Hardware	Glitch Reduction	Differential Shifting
Function Generators	14,048	13,143	11,260
CLB Slices	10,154	9,404	8,588
DFFs	4,777	4,812	4,812
Block RAMs	89	91	90

V. IMPLEMENTATION RESULTS

The LDPC decoder hardware is implemented in Verilog HDL. The implementation is verified with RTL simulations using Mentor Graphics ModelSim SE. RTL simulation results for both 1/2 and 5/6 code rates matched the results of MATLAB models of the LDPC decoding algorithm for 1/2 and 5/6 code rates.

The Verilog RTL design is synthesized to a 2V8000ff1157 Xilinx Virtex II FPGA with speed grade 5 using Mentor Graphics Precision RTL 2005b. The resulting netlist is placed and routed to the same FPGA using Xilinx ISE 8.2i. The LDPC decoder hardware implementation works at 45.5 MHz for 648 block length and 1/2 code rate and it works at 45.5 MHz for 648 block length and 5/6 code rate. The FPGA resource usages of the LDPC decoder implementations for 648 block length and 1/2 and 5/6 code rates are shown in Table III and IV respectively.

After applying glitch reduction technique, the LDPC decoder hardware implementation works at 55.5 MHz for 648 block length and 1/2 code rate and it works at 55.5 MHz for 648 block length and 5/6 code rate. After applying glitch reduction technique, the FPGA resource usages of the LDPC decoder implementations for 648 block length and 1/2 and 5/6 code rates are shown in Table III and IV respectively.

Applying sub-matrix reordering technique did not affect the frequency and area of the LDPC decoder implementations.

After further applying differential shifting technique, for 648 block length and 1/2 code rate, the LDPC decoder hardware implementation works at 83.5 MHz and it can process 60.68 Mbps if it does 3 iterations (36 sub-iterations) for each codeword, and for 648 block length and 5/6 code rate, it works at 71.5 MHz and it can process 113.78 Mbps if it does 3 iterations (12 sub-iterations) for each codeword. After applying differential shifting technique, the FPGA resource usages of the LDPC decoder implementations for 648 block length and 1/2 and 5/6 code rates are shown in Table III and IV respectively.

VI. CONCLUSIONS

In this paper, we presented a low power hybrid LDPC decoder hardware implementing layered min-sum decoding algorithm for IEEE 802.11n Wireless LAN Standard. The hardware is implemented in Verilog HDL and verified to work correctly in a Xilinx Virtex II FPGA. For 648 block length and 1/2 code rate, on a Xilinx Virtex II FPGA, the LDPC decoder hardware implementation works at 83.5 MHz and it can process 60.68 Mbps. For 648 block length and 5/6 code rate, on a Xilinx Virtex II FPGA, the LDPC decoder hardware implementation works at 71.5 MHz and it can process 113.78 Mbps.

The power consumption of the implementation on a Xilinx Virtex II FPGA is estimated as 2052 mW for 648 block length and 1/2 code rate and 1989 mW for 648 block length and 5/6 code rate using Xilinx XPower tool. In this paper, we also proposed two novel techniques, sub-matrix reordering and differential shifting, for reducing the power consumption of a LDPC decoder hardware. We applied glitch reduction, sub-matrix reordering and differential shifting techniques to our LDPC decoder hardware. These techniques do not affect the BER of a LDPC decoder. For block length 648 and code rate 1/2, these three techniques together reduced the power consumption of the LDPC decoder hardware in total by 23.7% to 1,565.84 mW. For block length 648 and code rate 5/6, they together reduced the power consumption of the LDPC decoder hardware in total by 38.98% to 1,214.22 mW.

REFERENCES

[1] R. G. Gallager, “Low density parity check codes”, *IRE Transations on Information Theory*, vol. 8, pages 21-28, 1962.

[2] D. MacKay and R. Neal, “Near shannon limit performance of low density parity check codes”, *Electronics Letters*, volume 32, pages 1645-1646, August 1996.

[3] “IEEE 802.11n Wireless LAN Medium Access Control MAC and Physical Layer PHY specifications”, IEEE 802.11n-D2.0, 2007.

[4] Lingyan Sun, *“Implementation and Evaluation of Iterative Soft Detection / Decoding Using Field Programmable Gate Array”,* PhD Thesis, Carnegie Mellon University, Aug. 2005.

[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 Journal of Solid-State Circuits*, vol 37, pages 404–412, 2002.

[6] M. Mansour and N. Shanbhag, “Architecture-aware low-density parity-check codes”, *IEEE Int. Symp. on Circuits and Systems,* May 2003.

[7] M. Mansour and N. Shanbhag, “A 640-Mb/s 2048-bit programmable LDPC decoder chip,” *IEEE J. of Solid-State Circuits*, vol. 41, no.3, pp. 684- 698, March 2006.

[8] J. Dielissen, A. Hekstra, V. Berg, “Low cost LDPC decoder for DVB-S2**”,** *Design, Automation & Test in Europe Conference,* March 2006.

[9] T. Brack, M. Alles, T. Lehnigk-Emden, F. Kienle, N. Wehn, Lapos, N.E. Insalata, F. Rossi, M. Rovini, L. Fanucci, “Low Complexity LDPC Code Decoders for Next Generation Standards”, *Design, Automation & Test in Europe Conference,* April 2007.

[10] J. Dielissen, A. Hekstra, “Non-fractional parallelism in LDPC Decoder implementations”, *Design, Automation & Test in Europe Conference,* April 2007.

[11] Weihuang Wang and Gwan Choi, “Minimum-Energy LDPC Decoder for Real-Time Mobile Application”, *Design, Automation & Test in Europe Conference,* April 2007.

[12] K. Gunnam, G. Choi, W. Wang, M. Yeary, “Multi-Rate Layered Decoder Architecture for Block LDPC Codes of the IEEE 802.11n Wireless Standard”, *IEEE Int. Symp. on Circuits and Systems,* May 2007.

[13] S. J. E. Wilton, S-S. Ang and W. Luk, "The Impact of Pipelining on Energy per Operation in Field-Programmable Gate Arrays", *International Conference on Field-Programmable Logic and its Applications*, August 2004.