We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

6,300

Open access books available

172,000

International authors and editors

190M

Downloads

Our authors are among the

154

Countries delivered to

TOP 1%

most cited scientists

12.2%

Contributors from top 500 universities

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index
in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

Ultra-High Speed LDPC Code Design and Implementation

Jin Sha¹, Zhongfeng Wang² and Minglun Gao¹

¹Nanjing University, ²Broadcom Corporation

¹China, ²U.S.A.

1. Introduction

The digital communications are ubiquitous and have provided tremendous benefits to everyday life. Error Correction Codes (ECC) are widely applied in modern digital communication systems. Low-Density Parity-Check (LDPC) code, invented by Gallager (1962) and rediscovered by MacKay (1996), is one of the two most promising near-optimal error correction codes in practice. Since its rediscovery, significant improvements have been achieved on the design and analysis of LDPC codes to further enhance the communication system performance. Due to its outstanding error-correcting performance (Chung 2001), LDPC code has been widely considered in next generation communication standards such as IEEE 802.16e, IEEE 802.3an, IEEE 802.11n, and DVB-S2. LDPC code is characterized by sparse parity check matrix. One key feature associated with LDPC code is the iterative decoding process, which enables LDPC decoder to achieve outstanding performance with moderate complexity. However, the iterative process directly leads to large hardware consumption and low throughput. Thus efficient Very Large Scale Integration (VLSI) implementation of high data rate LDPC decoder is very challenging and critical in practical applications.

A satisfying LDPC decoder usually means: good error correction performance, low hardware complexity and high throughput. There are various existing design methods which usually encounter the limitations of high routing overhead, large message memory requirement or long decoding latency. To implement the decoder directly in its inherent parallel manner may get the highest decoding throughput. But for large codeword lengths (e.g., larger than 1000 bits), to avoid routing conflict, the complex interconnection may take up more than half of the chip area. Both serial and partly parallel VLSI architectures are well studied nowadays. However, none of these approaches are good for very high throughput (i.e., multi-Gb/s) applications.

In this chapter, we present the construction of a new class of implementation-oriented LDPC codes, namely shift-LDPC codes to solve these issues integrally. Shift-LDPC codes have been shown to perform as well as computer generated random codes following some optimization rules. A specific high-speed decoder architecture targeting for multi-Gb/s applications namely shift decoder architecture, is developed for this class of codes. In

contrast with conventional decoder architectures, the shift decoder architecture has three major merits:

- 1) Memory efficient. By exploring the special features of the min-sum decoding algorithm, the proposed architecture stores the message in a compact way which normally leads to approximately 50% savings on message memory over the conventional design for high rate codes.
- 2) Low routing complexity. Through introducing some novel check node information transfer mechanisms, the complex message passing between variable nodes and check nodes can be alleviated by the regular communication between check nodes and thus the complex global interconnection networks can be replaced by local wires. One important fact is that in the new architecture, check node processing units have few and fixed connections with variable node processing units.
- 3) High parallel level of decoding. The architecture can normally exploit more levels of parallelism in the decoding algorithm than conventional partially parallel decoder architectures. In addition, the decoding parallel level can be further linearly increased and the critical path can be significantly reduced by proper pipelining.

The chapter is organized as follows. Section 2 gives an overview of LDPC codes, with the main attention being paid to the decoding algorithm and decoder architecture design. Section 3 introduces the code construction of shift-LDPC codes and Section 4 presents the decoder design with shift decoder architecture and demonstrates the benefits of proposed techniques. In Section 5, we consider the application of shift architecture to some well known LDPC codes such as RS-based LDPC codes and QC-LDPC codes. Section 6 concludes the chapter.

2. Review of LDPC codes

2.1 LDPC decoding algorithms

LDPC codes are a set of linear block codes corresponding to the $M \times N$ parity check matrix H , which has very low density of 1's. M is the parity check number and N is the codeword length. With the aid of the bipartite graph called Tanner graph (Tanner 1981), LDPC codes can be effectively represented. There are two kinds of nodes in Tanner graph, variable nodes and check nodes, corresponding to the columns and rows of parity check matrix respectively. Check node f_i is connected to variable node c_j only when the element h_{ij} of H is a 1. Fig.1 shows the parity check matrix H of an $(8, 4)$ regular LDPC code and its Tanner graph. A LDPC code is called (c, t) regular if the number of 1's in every column is a constant c and the number of 1's in every row is also a constant t .

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

(b)

Fig. 1. An LDPC code example. (a) Tanner graph. (b) Parity check matrix.

The typical LDPC decoding algorithm is the Sum-Product (or belief propagation) algorithm. After variable nodes are initialized with the channel information, the decoding messages are iteratively computed by all the variable nodes and check nodes and exchanged through the edges between the neighbouring nodes (Kschischang 2004).

The modified min-sum decoding algorithm studied in Guilloud (2003), Chen (2005) and Zhao (2005) is similar to the Sum-Product algorithm, with a simplification of check node process. It has some advantages in implementation over the Sum-Product algorithm, such as less computation complexity and no requirement of knowledge of SNR (signal-to-noise ratio) for AWGN channels. In the modified min-sum decoding algorithm, the check node computes the check-to-variable messages R_{cv} as follows:

$$R_{cv} = \alpha \times \prod_{n \in N(c) \setminus v} \text{sign}(L_{nc}) \times \min_{n \in N(c) \setminus v} |L_{nc}| \quad (1)$$

where α is a scaling factor around 0.75, L_{vc} is the variable-to-check messages, $N(c)$ denotes the set of variable nodes that participate in c -th check node. The variable node computes the variable-to-check messages L_{vc} as the following:

$$L_{vc} = \sum_{m \in M(v) \setminus c} R_{mv} + I_v \quad (2)$$

where $M(v) \setminus c$ denotes the set of check nodes connected to the variable node v excluding the variable node c , I_v denotes the intrinsic message of variable node v .

2.2 Implementation options

From the decoding algorithm described above, it can be seen that the calculations between different variable nodes or check nodes are independent at each iteration. Therefore, LDPC codes are especially suitable for parallel implementation. In addition, the computation in the decoder is fairly simple, which means low logic consumption. The main obstacle in the implementation of fully parallel LDPC decoders is the complicated interconnection between variable node processing units (VNU) and check node processing units (CNU). This interconnection complexity is due to the random-like locations of ones in the code's parity-check matrix and it is a crucial problem for practical fully parallel decoders when the code block length is large. The routing congestion or interconnection problem will arise and thus causes high area consumption and low logic utilization in the decoder. For instance, with 4-bit precision of probability messages, the 52.5 mm² die size of the (1024, 512) decoder in Blanksby (2002) has a logic utilization of 50% in its core and the rest of the core area is occupied by wires. There are some approaches developed to alleviate this routing

congestion problem. Darabiha (2008) uses bit-serial architectures and message broadcasting technique to reduce the number of wires in fully parallel decoders. Sharifi Tehrani (2008) presents a stochastic decoder design where the messages or probabilities are converted to streams of stochastic bits and complex probability operations can be performed on stochastic bits using simple bit-serial structures.

The more general methods to alleviate the routing problem is through reducing the parallelism (by using partially parallel or serial processing) and using storage elements to store the intermediate messages passed along the edges of the graph (e.g., see Mansour 2002, Yeo 2003, Cocco 2004). Various approaches are investigated in the literature at both code design and hardware implementation levels. One approach is to design "implementation-aware" codes (e.g., see Boutillon 2000, Zhang 2002, Mansour 2003a, Liao 2004, Zhang 2004, Sha 2009). In this approach, instead of randomly choosing the locations of ones in the parity-check matrix at the code design stage, the parity-check matrix of an LDPC code is decided with constraints allowing a suitable structure for decoder implementation and providing acceptable decoding performance. In these cases, the problem becomes how to reduce the complexity incurred by the message storage elements and how to speedup the decoding process. For example, one important subclass of LDPC codes with regular structure is quasi-cyclic (QC) LDPC codes which has received the most attentions and has been selected by many industry standards. Due to their regular structure, QC-LDPC codes lend themselves conveniently to efficient hardware implementations. Not only the partly parallel decoder architectures for QC-LDPC codes require simpler logic control (e.g., see Chen 2004b, Wang 2007) but also the encoders can be efficiently built with shift registers (e.g., see Li 2006). Construction of QC-LDPC codes with good error performances is therefore of both theoretical and practical interest. Various construction methods of QC-LDPC codes have been proposed and the satisfying error correcting performances are reported (e.g., see Fossorier 2004, Chen 2004a).

On the other hand, LDPC decoders can be implemented with a programmable architecture or processor, which lend themselves to Software Defined Radio (SDR). SDR offers flexibility to support codes with different block lengths and rates, however, Seo (2007) shows that the throughput of SDR-based LDPC decoders is usually low. In addition to digital decoders, continuous time analog implementations have also been considered for LDPC codes. Compared to their digital counterparts, analog decoders offer improvements in speed or power. However, because of the complex and technology-dependent design process, the analog approach has been only considered for very short error-correcting codes, for example, the (32, 8) LDPC code in Hemati (2006).

In the following, a new ensemble of LDPC codes, called shift-LDPC codes, will be introduced to mitigate the above decoder implementation problems.

3. Shift-LDPC Code Construction

3.1 Construction of Shift-LDPC Code

Shift-LDPC code is constructed in a well-regulated way. A regular (N, M) (c, t) shift LDPC code example is shown in Fig. 1, with $N=t \times q$ and $M=c \times q$, where each submatrix has a dimension of $q \times q$. As displayed, the parity check matrix consists of $c \times t$ submatrices. The structures of the leftmost c submatrices H_{i1} ($1 \leq i \leq c$) are random column permutations of the identity matrix. The other submatrices are decided by these H_{i1} submatrices.

Fig. 2. The construction of shift-LDPC parity check matrix and an example of the (2,3) shift-LDPC code

The 1's in the shift LDPC code parity check matrix are arranged as the example in Fig. 1. At first, the 1's in the leftmost submatrices are arranged randomly under the constraint of keeping one "1" in each row and one "1" in each column. Thus the submatrix H_{i1} is a permutation matrix of identity matrix. Next, for each submatrix on the right hand side, the 1's are cyclic-shifted up by 1 space. The "1" at the top is moved down to the bottom. Finally we can get a (c, t) regular shift-LDPC code.

It should be noted that shift-LDPC code is a kind of code specially designed for hardware implementation. In fact, the shift decoder architecture is designed first and then the code construction is found. This follows the "decoder-first code design" methodology proposed in Boutillon (2000).

3.2 Code examples and performance

Given a fixed block length, column weight, and row weight, the ensemble of shift-LDPC codes can have considerable variations in performance. An effective method to find good LDPC codes is to find the codes with large girth. Girth is referred to the length of the shortest cycle in a Tanner graph. A large girth generally improves the bit error performance of the codes. Hence, LDPC codes with large girth are particularly desired. By exploiting the structured property of shift-LDPC codes, an efficient girth optimization algorithm is developed. Normally, by using this girth optimization soft program, the cycle-4 can be eliminated, and the cycle-6 can be avoided for moderate code rate shift-LDPC codes.

Through extensive simulation, we found that the performance of the optimized shift LDPC codes can be comparable to the computer generated random codes. Fig. 2 shows the performance comparison between a girth optimized (1008, 504) (3, 6) shift-LDPC code, the same size rate-1/2 code downloaded from Mackey's website and a (960,480) irregular WiMAX code as well as comparison between a (8192, 7168) (4, 32)-regular shift-LDPC code and a girth optimized QC-LDPC code. It can be seen that the WiMAX code performs better due to its irregular property, and in other cases the shift-LDPC codes have comparable performance. The performance of shift-LDPC codes could be further improved by introducing limited irregularity, e.g., to zero out some of the submatrices, which will cause tiny hardware overhead.

Shift-LDPC codes are a kind of implementation oriented codes and are constructed after the decoder design. Thus in this chapter we will focus on the efficient decoder design instead of the code performance. On the other hand, the shift decoder architecture can be applied to a large number of known structured LDPC codes such as quasi cyclic LDPC codes and RS

based LDPC codes. For example, QC-LDPC codes can be converted to shift architecture compliant code with proper matrix permutation. It will be explained later in the chapter.

Fig. 3. Performance of (1008, 504) (3, 6) shift-LDPC code and (8192, 7168) (4, 32) shift-LDPC code compared with the same length WiMAX standard code and computer generated random codes

4. Shift LPDC Decoder Architecture

In this section, we present the shift decoder architecture specifically designed for shift-LDPC codes.

4.1 Decoding schedule

Firstly, let us discuss about the decoding schedule. For a regular (N, M) (c, t) shift-LDPC code, we assume q variable node processing units and M check node processing units are instantiated in the decoder where q is the size of submatrix (q=N/t=M/c). Fig. 3 illustrates the decoding flow for a simple shift-LDPC code example with q=4. The schedule can be applied to other cases with different q, c, and t parameters.

Fig. 4. Decoding schedule for a sample shift-LDPC code

The leftmost q columns are processed first, then the second left-most q columns, and so on. So q columns are processed concurrently in one clock cycle. The column process and the row process are interleaved. The whole check node process is divided into t steps. In each clock cycle, q VNUs get M check-to-variable messages and compute the M variable-to-check messages, so that M CNUs get one message each, so each CNU can deal with one step of the check node process. With this decoding schedule, we can finish one iteration in t clock cycles. It is normally much faster than the traditional partly parallel decoder architectures. By combining the decoding schedule and the min-sum algorithm, the decoding process can be expressed as below:

Min-Sum Algorithm and Decoding Schedule

- 1: **Initialization:** $L_{vc} = I_v$, $i = 0, 1, \dots, N-1$;
 - 2: **repeat**
 - 3: **for** $k = 0$ **to** $k = d_c - 1$ **do**
 - 4: { process of kq th $\sim (k+1)q$ th columns }
 - 4: compute R_{cv} from row process result of last iteration
 - 5: magnitude of $R_{cv} = \text{minimum or 2nd-minimum}$
 - 6: sign of $R_{cv} = \prod_{n \in N(c)} \text{sign}(L_{nc}) \times \text{sign}(L_{vc})$
 - 7: for q columns process :
 - 8:
$$L_{vc} = \sum_{m \in M(v)} R_{mv} + I_v - R_{cv}$$
 - 9: for all check nodes (receive one L_{vc} per row):
 - 10: update minimum; 2nd-minimum; location of minimum;
 - 11: record signs
 - 12: **endfor**
 - 13: **until** max iteration times reached or convergence to a codeword
 - 14: **Output:** decoded bit
-

Table 1. Decoding schedule designed for shift decoder architecture

4.2 Overall decoder architecture

The overall decoder block diagram is shown in Fig. 4. q variable node processing units and M check node processing units are instantiated in the decoder. The critical part of this implementation is the network connecting VNUs and CNUs. The shuffle network A transmits variable-to-check messages; the shuffle network B transmits check-to-variable messages. Shuffle network B is the same as shuffle network A except that the data flow is in the reverse direction.

In LDPC decoding algorithms, CNUs only communicate with VNUs. The row processes of different check nodes are independent of each other and so do the variable nodes processes. In shift-LDPC decoder architecture, a novel block called *CNU communication network* is added into the decoder. This new block brings some communications between CNUs. By introducing these originally unwanted communications, the complexity of connections

between CNUs and VNUs and thus the complexity of shuffle networks can be significantly reduced. This will be shown next.

Fig. 5. Overall shift decoder block diagram. Messages are iteratively exchanged between VNUs and CNUs through the shuffle network

For random codes, the shuffle network routing complexity is normally intolerable, while for shift-LDPC codes we introduced above, the shuffle networks become really simple. Through the introduction of the *CNU communication network*, we can ensure that each CNU processes the variable-to-check messages from and transmit the check-to-variable messages to a fixed VNU during the entire decoding process. Therefore, the shuffle network connecting CNUs and VNUs only consists of $M \cdot b$ wires, where we assume each message is quantized as b bits. (Normally the quantization bits b is chosen as 6.) In contrast, the number of wires required by original LDPC decoding algorithm is $M \cdot t \cdot b$. For high rate codes, for example, t can be as large as 32.

Fig. 5 explains why each CNU always computes with the messages from a fixed VNU during the entire decoding process. With the simple shuffle network, CNU i is connected with VNU x . The row process of each check node is separated into t steps with one variable-to-check message being processed in each step. At the first clock cycle of an iteration, having received the message from VNU x , CNU i performs the first step of i -th row process. After that, CNU i passes the i -th row process intermediate result to CNU $i+1$ through the *CNU communication network*. Meanwhile, CNU i receives the $(i-1)$ -th row process intermediate result from CNU $i-1$. In the second clock cycle of the iteration, CNU i still receives the variable-to-check message from VNU x . This message and the row process intermediate result in CNU i are arranged both corresponding to row $i-1$. Therefore, CNU i can perform the second step of $(i-1)$ -th row process. At the same time, CNU $i+1$ is performing the second step of i -th row process. For variable node processing, VNU x receives the check-to-variable

message in sequential from row i to $i-1, i-2, \dots$, etc. In the same way, the *CNU communication network* also can ensure that VNU x only need to receive its message from CNU i .

Fig. 6. Data flow of row process intermediate result

To show the decoding procedure more clearly, we illustrate the whole process with the simple (12, 4) (2, 3) shift LDPC code presented in Fig. 1 as an example. The procedure for one entire iteration is shown in Fig. 6. It can be seen that the *CNU communication network* is separated into two independent networks: *CNU communication network (intra iteration)* and *CNU communication network (inter iterations)* which transfer the row processing results during iteration and between iteration respectively.

The connections between CNUs and VNUs are fixed and simplified. CNU 1 connects with VNU 4; CNU 2 connects with VNU 2...These connections are based on the 1's positions in the leftmost submatrices.

One time iteration consists of three steps or three cycles (the row weight t equals 3). In each step/clock cycle, each CNU generates a check-to-variable message, passes the message to its connected VNU and processes a variable-to-check message which is received from its connected VNU. At the starting of iteration, in the first clock cycle, CNU 1 process the data of row 1; CNU 2 process the data of row 2. After this one step row process, the row process intermediate results are shifted between CNUs through the *CNU communication network (intra iteration)*. The result according to row 1 is passed to CNU 2; the result according to row 2 is passed to CNU 3, etc.

In the second clock cycle, CNU 2 still transmits to and gets message from VNU 2. This time the variable-to-check message it gets is corresponding to row 1, thus it can deal with one step of row 1 process. Likely CNU 3 performs one step of row 2 process; CNU 4 performs one step of row 3 process; etc. The third step is performed in the same way.

Once the iteration is finished, the row process results will be transferred back to appropriate CNUs for the check-to-variable messages generation in next iteration. The results are transferred through the *CNU communication network (inter iterations)*. This network is a one-to-one communication network as shown in Fig. 6.

As a result, the communication between CNUs and VNUs required by the original LDPC decoding algorithm is decomposed into three kinds of connections: the simplified connection between CNUs and VNUs; the *CNU communication network (intra iteration)* and the *CNU communication network (inter iterations)*.

Fig. 7. Decoder scheduling of the simple shift-LDPC code in Fig. 1

In the following, we will introduce the architecture of CNU applying the min-sum decoding algorithm. The (8192, 7168) (4, 32) shift LDPC code is chosen as the design example.

4.3 Architecture of check node processing unit

The check node processing unit executes the check-to-variable message computation and the magnitude comparison between the variable-to-check message and the intermediate result of row process. Fig. 7 shows the architecture of CNU. The old register, new register, and sign register store the row process results of last iteration, the intermediate results of row process, and the sign bits of the check-to-variable messages respectively. To manage the message storage more efficiently, only row process results are stored in a compressed way that only the minimum magnitude, 2nd-minimum magnitude, the location of the minimum and the sign bits are saved.

The data in new registers and old registers are always being passed between CNUs through the *CNU communication network (intra iteration and inter iterations)*. The signs in the sign register are from the same VNU, so they do not need to be passed. In each clock cycle, sign register shifts in a sign bit of the variable-to-check message and shifts out a sign bit of a check-to-variable message.

Fig. 8. Architecture of check node processing unit

Because only one variable-to-check message is processed in one time, the computation in CNU has very low complexity. The magnitude comparison part compares the magnitude of input message with the current row process intermediate result to update the magnitudes, sign and index. Then the updated row process intermediate result is registered and passed to its next CNU neighbour through the *CNU communication network (intra iteration)*. The message computation part selects the proper message magnitude according to the index value, and computes the sign of the message. At the beginning of each iteration, it performs the message scale calculation ($a = 0.75$).

4.4 Architecture of variable node processing unit

The architecture of variable node processing unit is the same with other LDPC decoder designs. It can be designed with only combinational logic. Fig. 8 shows the VNU architecture. The inputted check-to-variable messages are firstly converted from sign-magnitude format to two's complement format. Then the adder tree computes the variable-to-check messages according to equation (2). Finally the messages are converted back to sign-magnitude format and transmitted to CNUs.

Fig. 9. Architecture of the variable node processing unit. StoT module converts from the sign-magnitude format to two's complement format. TtoS is for reverse conversion

4.5 Implementation results

To further increase the decoding speed, more parallel levels can be added to the decoder architecture. For example, to double the parallel level, $2 \times q$ VNUs will be instantiated. The architecture of CNU should be changed to process two messages per clock cycle. Then each iteration can be finished in $t/2$ clock cycles. A double parallel level decoder example is designed for the (8192, 7168) (4, 32) shift LDPC code to investigate the tradeoff between hardware cost and decoding speed.

In addition, to increase the clock frequency, some pipeline levels can be added to the data path. The critical path is from old register, through CNU message computation part, to VNU computation, to CNU row processing part, and finally ends at the new register. Pipeline will increase the number of clock cycles needed for each iteration. For example, a three stage pipeline will add two clock cycles to each iteration.

Gate count	CNU combination logic	VNU combination logic	total combination logic	frequency (MHz)	Throughput per iteration (Gbps)	Message area (mm ²)	Total area (mm ²)
One parallel level design	148	953	395K	160	40.96	6.1	10
One parallel + pipelining	148	1410	512K	317	76.4 ★	6.1	11.3
Double parallel level	342	953	838K	151	77.3	6.1	14
Double parallel + pipelining	342	1410	1072K	317	153	6.1	16.7

Table 2. Synthesis result for (8192, 7168) (4, 32) shift LDPC decoder under 0.18μm technology

Table 2 shows the synthesis result of four design examples: one-level parallel design, one-level parallel with pipelining, two-level parallel design, and two-level parallel with

pipelining. Due to the large number of CNUs and VNUs (submatrix size $q = 256$), the bottom-up synthesis strategy is applied. These results are achieved with $0.18\mu\text{m}$ technology. It can be seen that, by applying pipelining, the clock frequency can be almost doubled, and the overhead is 81 registers per CNU. The one-level parallel design with pipelining can achieve the best trade-off between speed and hardware complexity.

For the message storage, it is implemented with registers. The transferring messages take $(16 \times 2 + 32) \times 1024 = 65536$ bits. In comparison, the traditional partially parallel architecture (e.g., Wang (2007)) needs to store $8192 \times 4 \times 6 = 196224$ bits in total. It saves 67% of the message memory needed. The initial intrinsic information takes $8192 \times 6 = 49152$ bits. It is the same for both cases.

To further examine the efficiency of the shift decoder architecture, backend design of the sample $(8192, 7168) (4, 32)$ shift LDPC decoder is completed. The technology used for implementation is a $0.18\mu\text{m}$ CMOS process with 6 metal layers. Fig. 9 shows the floor-plan and layout of the decoder chip with a die size of $4.1\text{ mm} \times 4.1\text{ mm}$. The logic density is 70%. The placement of CNUs is critical to reducing the routing congestion of CNU communication networks. The CNU array located in the centre of the chip is specially arranged: the 1024 CNUs are aligned with 31 CNUs per row, so that the communications between CNUs can be locally routed.

Fig. 10. The floor-plan and layout of the $(8192, 7168) (4, 32)$ shift LDPC decoder

	Partly parallel Mansor (2006)	Ultra sparse Brack (2007)	Proposed
Code length	2048	9600	8192
Code rate	1/2	3/4	7/8
Edges	6144	26400	32768
Quantization	4b	6b	6b
Algorithm	Min-Sum	Min-Sum	Min-Sum
technology	180nm	65nm	180nm
frequency	125 MHz	500MHz	317 MHz
Iterations	10	10	15
Throughput	640Mb/s	1.45Gb/s	5.1 Gb/s
Area (mm2)	14.3	0.504	11.3/16.8
Area [scaled to 90nm]	3.57	1.008	2.82 /4.2
Hardware Efficiency	1778	14384	25500

Table 3. Comparisons with other decoder architectures

Table 3 shows the decoder implementation results compared with some other LDPC decoder architectures. The throughput achieved here is 5.1 Giga bits per second at a maximum iteration times of 15. One parameter “Hardware Efficiency” as (Throughput * Iterations / Area) is defined to evaluate the efficiency of each architecture. The area metrics are scaled to resemble 90nm CMOS results (65nm by a factor of 2 and 180nm by a factor of 1/4). From this table, we can see that the proposed design can achieve more than 70% improvement in hardware efficiency compared with an advanced existing design while the higher clock speed benefited from much more advanced CMOS technology is not considered for this comparison. Otherwise the improvement would be even more significant. So shift decoder architecture is very efficient for high-speed LDPC decoder implementation.

5. Apply the shift architecture to known LDPC codes

5.1 RS-based LDPC code

In Djurdjevic (2004), the authors constructed a kind of LDPC codes based on Reed-Solomon codes and called them RS-based LDPC codes. These codes have good minimum distances, girth properties, and perform very well with iterative decoding. One code constructed in this way was chosen as the forward error correction coding scheme in IEEE 802.3an 10GBase-T Ethernet standard (LAN/MAN CSMA/CD Access Method). RS-based LDPC codes are constructed in an algebraic way and by exploiting the shift-structured properties hidden in the parity check matrices, the shift decoder architectures presented above can be applied to this kind of code. In the following, we will explain the shift properties in these codes by going through the code generation procedure.

We start from the extended $(q, 2, q - 1)$ RS code C_b over $GF(q)$ which has a length of q and two information symbols. By using the generation matrix

$$G = \begin{bmatrix} 1 & 1 & \cdots & 1 & 1 & 0 \\ 1 & \alpha^{q-2} & \cdots & \alpha^2 & \alpha & 1 \end{bmatrix}, \quad (3)$$

we can get the extended RS code with q^2 codewords in total. Let $\mathbf{v}$ be a nonzero codeword in C_b with weight q , for example $\mathbf{v} = (1, \alpha^{q-2}, \cdots, \alpha^2, \alpha, 1)$. Then, the set $C_b^{(0)} = \{c\mathbf{v} : c \in GF(q)\}$ of q codewords forms the subcode of C_b . Set $C_b^{(0)}$ always contains the all zero codeword and $q-1$ weight q codewords, no matter which weight q codeword $\mathbf{v}$ is chosen.

Partition C_b into q additive cosets, $C_b^{(0)}, C_b^{(1)}, \dots, C_b^{(q-1)}$ based on the subcode $C_b^{(0)}$. Each coset $C_b^{(i)}$ is composed of q codewords, $W_0^{(i)}, W_1^{(i)}, \dots, W_{q-1}^{(i)}$ and each codeword has a length of q , as below:

$$C_b^{(i)} = \begin{bmatrix} W_0^{(i)} \\ W_1^{(i)} \\ \vdots \\ W_{q-1}^{(i)} \end{bmatrix} = \begin{bmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,q-1} \\ w_{1,0} & w_{1,1} & \cdots & w_{1,q-1} \\ \vdots & \vdots & \ddots & \vdots \\ w_{q-1,0} & w_{q-1,1} & \cdots & w_{q-1,q-1} \end{bmatrix} \\ = \begin{bmatrix} 1+\alpha^i & \alpha^{q-2}+\alpha^i & \alpha^{q-3}+\alpha^i & \cdots & \alpha+\alpha^i & 1 \\ \alpha+\alpha^i & 1+\alpha^i & \alpha^{q-2}+\alpha^i & \cdots & \alpha^2+\alpha^i & \alpha \\ \alpha^2+\alpha^i & \alpha+\alpha^i & 1+\alpha^i & \cdots & \alpha^3+\alpha^i & \alpha^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \alpha^{q-2}+\alpha^i & \alpha^{q-3}+\alpha^i & \alpha^{q-4}+\alpha^i & \cdots & 1+\alpha^i & \alpha^{q-2} \\ 0+\alpha^i & 0+\alpha^i & 0+\alpha^i & \cdots & 0+\alpha^i & 0 \end{bmatrix}. \quad (4)$$

Then the cosets are arranged together to get the $q^2 \times q$ matrix H_{rs}

$$H_{rs} = \begin{bmatrix} C_b^{(0)} \\ C_b^{(1)} \\ \vdots \\ C_b^{(q-1)} \end{bmatrix}. \quad (5)$$

Replace each symbol in H_{rs} with a location vector:

$$z(\alpha^i) = (z_{-\infty}, z_0, z_1, z_2, \cdots, z_{q-2}), \quad (6)$$

where the i -th component $z_i = 1$ and all the other components equal zero. The exponential value can be denoted as the position of "1" in the location vector replacement. Finally, choose a $d_v \times d_c$ subarray from H_{rs} to get a (d_v, d_c) regular LDPC code. This can also be stated as that select d_v cosets randomly and then select d_c columns in them, finally replace each symbol with a location vector to get a sparse parity check matrix H .

A coset example is given here to show the matrix properties clearly. It is generated in GF(8):

$$\begin{bmatrix} 4 & 6 & 7 & 3 & 1 & 5 & 0 & 1 \\ 0 & 4 & 6 & 7 & 3 & 1 & 5 & 2 \\ 5 & 0 & 4 & 6 & 7 & 3 & 1 & 3 \\ 1 & 5 & 0 & 4 & 6 & 7 & 3 & 4 \\ 3 & 1 & 5 & 0 & 4 & 6 & 7 & 5 \\ 7 & 3 & 1 & 5 & 0 & 4 & 6 & 6 \\ 6 & 7 & 3 & 1 & 5 & 0 & 4 & 7 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 0 \end{bmatrix}.$$

(7)

The examination of the matrix and equation (4) reveals the four following properties:

- 1) Each column is composed of the q elements of Galois Field $GF(q)$.
- 2) q th row is a special row. It is composed with only one value a^i except the last one “0”.
- 3) q th column is a special column. It is always “1, a , a^2 , ... a^{q-2} , 0”.
- 4) Except the special row and special column, the remaining $q-1 \times q-1$ matrix is a special type of circulant in which each row is the right cyclic-shift of the row above it while the first row is the right cyclic-shift of the last row.

Careful readers will find that property 4) is exactly the shift property which is introduced earlier in the chapter. Thus the shift decoder architecture can be applied to this kind of RS-based LDPC codes. Fig. 10 shows the location vector replacement in RS-based LDPC code construction and the mapping of the parity check matrix to decoder hardware. The chosen code sample is a (32, 24) (1, 4) RS-based LDPC code generated in $GF(8)$.

Fig. 11. The location vector replacement in RS-based LDPC code construction and the mapping of the parity check matrix to decoder hardware

Fig. 12. Block diagram of the shift decoder design for the matrix defined in Fig.10

Fig. 11 shows the block diagram of decoder designed for matrix defined in Fig. 10. It can be decomposed into two parts: the processing units and the message storage registers. There is one row result registers (RR) corresponding to each CNU. One RR is composed of current iteration row result registers (RRC) and last iteration row result registers (RRL). Both contain minimum value, 2nd-minimum value, minimum value index and sign bit.

Since the offset value in q th row is fixed at "2", q th row's message always comes from VNU 2. CNU 7 is only in charge of the q th row's processing and RR 7 stores processing result of the q th row, which corresponds to the special row property 2).

With the shift decoder architecture, two decoder examples are designed. The target code is the (2048, 1723) (6, 32) LDPC code generated from (64, 32, 2) RS code. A 6×32 sub-array at the upper left corner of H_{rs} is selected.

Table 4 shows the synthesis results under 90nm CMOS technology of two design examples: one basic design and a four times parallel level design. Compared to the basic design, the four times parallel level design can achieve an approximately 3x decoding speed with 2x hardware consumption. Table 3 also compares the shift architecture decoders with state-of-the-art bit serial fully parallel LDPC decoder (Darabiha (2008)) and partially parallel LDPC decoder (Chen (2003)). Comparing the throughput to area ratio metric, it can be clearly seen that the shift architecture-based designs are very much hardware efficient.

Design	Module	Gate count × number	Total gate	parameters	Clock speed	throughput
basic 34 cycles	CNU	148 × 384	420K	6 bit quan 8 iterations	500MHz	3.8 Gbps
	VNU	1450 × 64				
	storage	2048 × 6 bits initial 64 × 384 bits message				
4parallel 10 cycles	CNU	515 × 384	820K	6 bit quan 8 iterations	400MHz	10 Gbps
	VNU	1450 × 256				
	storage	2048 × 6 bits initial 64 × 384 bits message				
Darabiha (2008)	(2048, 1723) (6,32) RS-based LDPC code fully parallel architecture		2230K	4 bit quan 8 iterations	250MHz	16 Gbps
Chen (2003)	8088 codeword length, 1/2 rate partially parallel architecture		742K	6 bit quan 25 iterations	212MHz	188 Mbps

Table 4. Synthesis results of (2048, 1723) RS-LDPC code decoder under shift architecture and comparisons with other designs

5.2 Quasi-Cyclic LDPC codes

Quasi-cyclic LDPC (QC-LDPC) codes are well known that they can achieve comparable performance with equivalent random-like LDPC codes. In addition, more importantly, they are well suited for hardware implementation because of the regularity in their parity check matrices. In particular the encoder of a QC-LDPC code can be easily built with shift registers while usually it is hardware complex to encode an LDPC code.

In general, a regular (c, t) quasi-cyclic LDPC code is defined as its parity check matrix H ($M \times N$) consisting of a $c \times t$ array of submatrices. The dimension of each submatrix is $q \times q$ ($q = M/c = N/t$). Every submatrix H_{ij} is a circulant matrix of identity matrix I . Fig.12 shows an example of $(24, 8)$ $(2, 3)$ regular QC-LDPC code.

Fig. 13. The construction of QC-LDPC parity check matrix H_{qc} : an array of circulant submatrices

Fig. 14. The transformation from the sample QC-LDPC parity check matrix into a shift like LDPC code through column permutation

Fig. 13 illustrates the way to transform the QC-LDPC code parity check matrix into a shift kind matrix whose decoder can be implemented with the shift decoder architecture. Firstly the q columns are distributed to t block columns of H_{qcs1} in a round-robin fashion. Then the second q columns are permuted in the same way and so on until all columns are distributed into new matrix H_{qcs1} . Careful readers will find that a new property is induced in the new matrix that in some submatrices there are multiple “1”s in one row. This newly induced property requires a special CNU design which processes multiple messages instead of one at each step. It will cause some additional logic delay in CNU. In Cui (2008a), the authors proposed an efficient matrix permutation optimization method to minimize the maximum row weight. Normally, it can be no bigger than 2. In addition, in Cui (2008a) the authors optimized the column layered decoding algorithm and applied it to shift decoder architecture. By this way, the iteration number to converge can be reduced and the decoding throughput can be further increased.

There are other matrix transformation methods to apply the extended shift decoder architecture on QC-LDPC codes. For example, the quasi cyclic matrix can be transformed into a row shift construction by row permutations as shown in Fig. 14. Firstly the q rows are distributed to 4 block rows of H_{qcs2} in a round-robin fashion (i.e., rows A-H of H_{qc} are distributed to row 1, 5, 9, 13, 2, 6, 10 and 14 of H_{qcs2}). Then the second q rows are permuted in the same way and so on until all rows are distributed into new matrix H_{qcs2} . The quasi cyclic matrix is converted to a form as:

$$H_{qcrs} = \begin{bmatrix} A_1 & A_2 & A_3 & \cdots & A_t \\ A_1\alpha & A_2\alpha & A_3\alpha & \cdots & A_t\alpha \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_1\alpha^{q/m} & A_2\alpha^{q/m} & A_3\alpha^{q/m} & \cdots & A_t\alpha^{q/m} \end{bmatrix}$$

(8)

where a is a $q \times q$ permutation matrix representing a single right cyclic shift. m is an integer such that q can be divided by m .

Fig. 15. The transformation from the sample QC-LDPC parity check matrix into a shift like LDPC code through row permutation

After the matrix conversion, the row shift property can be exploited to apply a row shift decoder architecture just like the presented column shift architecture. By this means, the row layered decoding algorithm should be applied. Fortunately the row layer algorithm can increase the convergence speed significantly (Mansour (2003b)). Interested readers can be referred to Cui (2008b) for more information.

5.3 Array LDPC code

Array LDPC code is a more structured QC-LDPC code which can achieve comparable performance to random codes. Efficient encoding, minimum distance properties, and performance of array-code based LDPC codes for binary as well as multilevel modulation are addressed in Fan (2000). The parity check matrix based on array codes can be represented as follows:

$$H_{array} = \begin{bmatrix} I & I & I & \cdots & I \\ I & \alpha & \alpha^2 & \cdots & \alpha^{t-1} \\ I & \alpha^2 & \alpha^4 & \cdots & \alpha^{2(t-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ I & \alpha^{c-1} & \alpha^{2(c-1)} & \cdots & \alpha^{(c-1)(t-1)} \end{bmatrix} \tag{9}$$

Where I is a $q \times q$ identity matrix with a prime number q . α is a permutation matrix with a single cyclic right shift or left shift. For example, a (2209, 2024) (4, 47) array LDPC code can be constructed with $q = 47$. We use this code in the decoder explanation. For array LDPC codes, they have the similar shift property as shift-LDPC codes. Some modifications in the CNU communication network can be enough for the decoder implementation. Fig. 15 shows the modified CNU connections. They are grouped into four

CNU groups. Each group has 47 members. Group 1 contains from CNU-1 to CNU-47, in charge of the row operation of row 1~47. Group 2 contains from CNU-48 to CNU-94, in charge of the row operation of row 48~94... There is no communication between groups. The difference between array LDPC codes and shift LDPC codes is that the shift values are changing from 0 to 3 instead of keeping at 1. In addition, for this special kind of LDPC codes, the *CNU communication network (intra iteration)* is exactly the same with *CNU communication network (inter iteration)*, thus these two networks can be incorporated into one simple network and the routing complexity can be further simplified. As a result, except the CNUs in Group 1, every 47 CNUs in a group form a chain and each CNU only communicates with its two neighbors. For example, in CNU group 3, CNU- i only gets data from CNU- $(i-2)$ and delivers data to CNU- $(i+2)$.

A sample decoder is designed based on an ALTERA FPGA EP2C35. 50MHz clock speed and 120Mbps throughput (20 iterations) are achieved with 23k logic elements and 26k memory bits. Interested readers are referred to Sha (2006) for more information.

Fig. 16. The CNU communication network designed for (2209, 2024) (4, 47) array LDPC code

6. Concluding Remarks

Since MacKay's (1997) rediscovery of LDPC codes, the LDPC decoder design has experienced considerable development and enhancement over the last ten years. Various optimization techniques on both decoding algorithms and decoder designs have been developed. This chapter provided a brief overview of the problems in existing LDPC decoder designs and presented a novel shift-structured decoder design approach to tackle these problems. The codes suited for this kind of decoder architecture are called shift-LDPC codes. Several decoder samples are discussed to illustrate the effectiveness of the proposed architecture. In addition, it was shown in the chapter that some popular classes of LDPC codes such as RS-based LDPC codes and QC-LDPC codes can be implemented with the shift decoder architecture through simple matrix permutations or architecture extensions. As a conclusion, the presented shift decoder architecture will be a competent candidate in future high-speed communication system designs.

7. References

- A. Darabiha, A. C. Carusone, & F. R. Kschischang. (2008). Power Reduction Techniques for LDPC Decoders, *IEEE Journal of Solid-State Circuits*, vol. 43, no. 8, pp. 1835-1845
- A. J. Blanksby & C. J. Howland. (2002). A 690-mW 1-Gbps 1024-b, rate-1/2 Low-Density Parity-Check code decoder, *IEEE J. Solid-State Circuits*, vol. 37, no. 3, pp. 404-412
- D. J. C. MacKay & R. M. Neal. (1996). Near Shannon limit performance of low density parity check codes, *Electron. Lett.*, vol. 32, pp. 1645-1646
- E. Boutillon, J. Castura, & F. R. Kschischang, (2000). Decoder-First Code Design, *Proceedings of the 2nd International Symposium on Turbo Codes and Related Topics*, Brest, France, pp. 459-462
- F. Guilloud, E. Boutillon & J.L. Danger. (2003). λ -Min Decoding Algorithm of Regular and Irregular LDPC Codes, *Proc. 3rd International Symposium on Turbo Codes and Related Topics*, pp. 451-454
- E. Liao, E. Yeo & B. Nikolic. (2004). Low-density parity-check code constructions for hardware implementation, *Proc. IEEE Int. Conf. on Commun.* vol. 5, pp. 2573-2577
- E. Yeo, B. Nikolic & V. Anantharam. (2003) Iterative decoder architectures, *IEEE Commun. Mag.*, vol. 41, pp. 132-140
- F. R. Kschischang, B. J. Frey & H. A. Loeliger. (2001). Factor graphs and the sum-product algorithm, *IEEE Trans. Inf. Theory*, vol. 47, pp. 498-519
- G. Liva, S. Song, L. Lan, Y. Zhang, S. Lin, & W. E. Ryan. (2006). Design of LDPC Codes: A Survey and New Results. *J. Comm. Software and Systems*, vol. 2, pp. 191
- I. Djurdjevic, J. Xu, K. Abdel-Ghaffar, & S. Lin. (2004). Construction of low-density parity-check codes based on Reed-Solomon codes with two information symbols, *IEEE Commu. Lett.*, vol. 8, no. 7, pp. 317-319
- J. Chen, A. Dholakia, E. Eleftheriou, M. P. C. Fossorier, & X. Hu. (2005). Reduced-complexity decoding of LDPC codes, *IEEE Trans. Commun.*, vol. 53, pp. 1288-1299
- J. L. Fan. (2000). Array codes as low-density parity-check codes, *Proc. 2nd Int. Symp. Turbo Codes and Related Topics Brest*, France, pp. 543
- J. Sha, M. Gao, Z. Zhang, L. Li, Z. Wang. (2006). An FPGA Implementation of array LDPC decoder, *IEEE Asia Pacific Conference on Circuits and Systems*, pp. 1675-1678
- J. Sha, Z. Wang, M. Gao, & Li. (2009). Multi-Gb/s LDPC Code Design and Implementation, *IEEE Trans. on VLSI Systems*, vol. 17, no. 2, pp. 262-268
- LAN/MAN CSMA/CD Access Method, *IEEE 802.3 Standard* Online available: <http://standards.ieee.org/getieee802/802.3.html>
- J. Zhao, F. Zarkeshvari, & A. H. Banihashemi. (2005). On implementation of min-sum algorithm and its modifications for decoding Low-Density Parity-Check (LDPC) codes, *IEEE Trans. Commun.*, vol. 53, no. 4, pp. 549-554
- L. Chen, J. Xu, I. Djurdjevic & S. Lin. (2004a) Near-Shannon limit quasi-cyclic low-density parity-check codes, *IEEE Trans. Commun*, vol. 52, pp. 1038
- M. Cocco, J. Dielissen, M. Heijligers, A. Hekstra, & J. Huisken. (2004). A scalable architecture for LDPC decoding, *Proc. Design, Automation and Test in Europe*, vol. 3, pp. 88-93
- M. M. Mansour & N. R. Shanbhag. (2002). Low power VLSI decoder architecture for LDPC codes, *Proc. IEEE Int. Symp. on Low Power Electron. Design*, pp. 284-289
- M. M. Mansour & N. R. Shanbhag, (2003a). Architecture-Aware Low-Density Parity-Check Codes, *Proc. IEEE ISCAS*, pp. 57-60

- M. M. Mansour & N. R. Shanbhag. (2003b). High throughput LDPC decoders, *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 11, pp. 976-996
- M.M. Mansour & N. R. Shanbhag. (2006). A 640-Mb/s 2048-bit programmable LDPC decoder chip, *IEEE J. Solid-State Circuits*, vol. 41, no. 3, pp. 684- 698
- M. P. C. Fossorier. (2004). Quasi-cyclic low-density parity-check codes from circulant permutation matrices, *IEEE Trans. Inf. Theory*, vol. 50, no. 8, pp. 1788-1793
- R. G. Gallager. (1962). Low-density parity-check codes, *IRE Transactions on Information Theory*, vol. IT-8, pp. 21-28
- R. M. Tanner. (1981). A recursive approach to low complexity codes, *IEEE Trans. Inf. Theory*, vol. IT-27, pp. 533-547
- S. Hemati, A. Banihashemi, & C. Plett. (2006). A 0.18 μm analog min-sum iterative decoder for a (32,8) low-density parity-check (LDPC) code, *IEEE J. Solid-State Circuits*, vol. 41, pp. 2531-2540
- S. Seo, T. Mudge, Y. Zhu & C. Chakrabarti. (2007). Design and analysis of LDPC decoders for software defined radio, *Proc. IEEE Workshop on Signal Processing Systems*, Shanghai, China, pp.210-215
- S. Sharifi Tehrani, S. Mannor & W. J. Gross. (2008). Fully Parallel Stochastic LDPC Decoders *IEEE Trans. Signal Processing*, Vol. 56, no. 11, pp. 5692 - 5703
- S. Y. Chung, G. D. Forney, T. J. Richardson & R. Urbanke. (2001). On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit, *IEEE Commun. Lett.*, vol. 5, pp. 58-60
- T. Brack, M. Alles, T. Lehnigk-Emden, F. Kienle, N. Wehn, & L. Fanucci. (2007). Low Complexity LDPC Code Decoders for Next Generation Standards, *Proc. Design, Automation and Test in Europe*, pp. 1-6
- T. Zhang & K. Parhi. (2002). A 54Mbps (3,6)-regular FPGA LDPC decoder, *Proc. IEEE Sips'*, pp. 127-132
- T. Zhang & K. K. Parhi. (2004). Joint (3,k)-regular LDPC code and decoder/encoder design, *IEEE Trans. Signal Process.*, vol. 52, no. 4, pp.1065-1079
- Y. Chen & D. Hocevar. (2003). A FPGA and ASIC implementation of rate 1/2, 8088-b irregular low density parity check decoder, *Proc. IEEE GLOBECOM*, San Francisco, CA, pp. 113-117
- Y. Chen & K. K. Parhi. (2004b). Overlapped message passing for quasi-cyclic low density parity check codes, *IEEE Trans. Circuits Syst. I*, vol. 51, pp. 1106-1113
- Z.-W. Li, L. Chen, L.-Q. Zeng, S. Lin & W.H. Fong. (2006). Efficient encoding of quasi-cyclic low-density parity-check codes, *IEEE Transactions on Communications*, Vol. 54 , no. 1, pp. 71 - 81
- Z. Cui, Z. Wang, X. Zhang & Q. Jia. (2008a). Efficient decoder design for high-throughput LDPC decoding, *APCCAS*, pp. 1640-1643
- Z. Cui, Z. Wang & Y. Liu. (2008b). High-throughput layered LDPC decoding architecture, *IEEE Trans. VLSI Systems*, vol. 17, no. 4, pp. 582-587
- Z. Wang & Z. Cui. (2007). Low-complexity high-speed decoder design for quasi-cyclic LDPC codes, *IEEE Trans. on VLSI Systems*, vol. 15, no. 1, pp. 104-114

IntechOpen

IntechOpen

VLSI

Edited by Zhongfeng Wang

ISBN 978-953-307-049-0

Hard cover, 456 pages

Publisher InTech

Published online 01, February, 2010

Published in print edition February, 2010

The process of Integrated Circuits (IC) started its era of VLSI (Very Large Scale Integration) in 1970's when thousands of transistors were integrated into one single chip. Nowadays we are able to integrate more than a billion transistors on a single chip. However, the term "VLSI" is still being used, though there was some effort to coin a new term ULSI (Ultra-Large Scale Integration) for fine distinctions many years ago. VLSI technology has brought tremendous benefits to our everyday life since its occurrence. VLSI circuits are used everywhere, real applications include microprocessors in a personal computer or workstation, chips in a graphic card, digital camera or camcorder, chips in a cell phone or a portable computing device, and embedded processors in an automobile, et al. VLSI covers many phases of design and fabrication of integrated circuits. For a commercial chip design, it involves system definition, VLSI architecture design and optimization, RTL (register transfer language) coding, (pre- and post-synthesis) simulation and verification, synthesis, place and route, timing analyses and timing closure, and multi-step semiconductor device fabrication including wafer processing, die preparation, IC packaging and testing, et al. As the process technology scales down, hundreds or even thousands of millions of transistors are integrated into one single chip. Hence, more and more complicated systems can be integrated into a single chip, the so-called System-on-chip (SoC), which brings to VLSI engineers ever increasingly challenges to master techniques in various phases of VLSI design. For modern SoC design, practical applications are usually speed hungry. For instance, Ethernet standard has evolved from 10Mbps to 10Gbps. Now the specification for 100Mbps Ethernet is on the way. On the other hand, with the popularity of wireless and portable computing devices, low power consumption has become extremely critical. To meet these contradicting requirements, VLSI designers have to perform optimizations at all levels of design. This book is intended to cover a wide range of VLSI design topics. The book can be roughly partitioned into four parts. Part I is mainly focused on algorithmic level and architectural level VLSI design and optimization for image and video signal processing systems. Part II addresses VLSI design optimizations for cryptography and error correction coding. Part III discusses general SoC design techniques as well as other application-specific VLSI design optimizations. The last part will cover generic nano-scale circuit-level design techniques.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Jin Sha, Zhongfeng Wang and Minglun Gao (2010). Ultra-High Speed LDPC Code Design and Implementation, VLSI, Zhongfeng Wang (Ed.), ISBN: 978-953-307-049-0, InTech, Available from: <http://www.intechopen.com/books/vlsi/ultra-high-speed-lidpc-code-design-and-implementation>

INTECH
open science | open minds

www.intechopen.com

InTech Europe

University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China

Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen

IntechOpen

© 2010 The Author(s). Licensee IntechOpen. This chapter is distributed under the terms of the [Creative Commons Attribution-NonCommercial-ShareAlike-3.0 License](https://creativecommons.org/licenses/by-nc-sa/3.0/), which permits use, distribution and reproduction for non-commercial purposes, provided the original is properly cited and derivative works building on this content are distributed under the same license.

IntechOpen

IntechOpen