单位代码 10006
学　　号 13021264
分 类 号 TN914.3

北京航空航天大學
BEIHANG UNIVERSITY

毕业设计(论文)

LDPC 译码器架构调研及
低复杂度可重配置移位网络设计优化

学 院 名 称　电子信息工程学院
专 业 名 称　通信工程
学 生 姓 名　何沃洲
指 导 教 师　Jinhong Yuan、刁为民

2017 年 6 月

本人声明

我声明，本论文及其研究工作是由本人在导师指导下独立完成的，在完成论文时所利用的一切资料均已在参考文献中列出。

作者：何 沃 洲

签字：

时间：2017 年 6 月

北京航空航天大学

本科生毕业设计（论文）任务书

Ⅰ、毕业设计（论文）题目：

LDPC 译码器架构调研及低复杂度可重配置移位网络设计优化

Ⅱ、毕业设计（论文）使用的原始资料（数据）及设计技术要求：

（1）原始资料主要来源于 IEEE Xplore 数据库收录的关于 LDPC 译码器架构的若干篇最新论文；

（2）设计技术要求是实现支持 100MHz 以上时钟频率、吞吐量可达数百兆 Hz 的 LDPC 译码器架构。

Ⅲ、毕业设计（论文）工作内容：

（1）对 LDPC 基础的译码算法和译码器结构进行了理论整理；

（2）对文献已提出的 LDPC 译码器架构优化方法进行了调研和综述，从提取出最具有参考意义的若干点共性及特点；

（3）研究了 LDPC 译码器中低复杂度可重配置移位网络的实现。在文献现有的若干种实现上进行了资源优化，得出了适用于 WiMAX 和 WLAN 中 LDPC 码标准的多模式可重配置移位网络的最优设计。

Ⅳ、主要参考资料：

[1] R. Gallager, “Low-density parity-check codes,” IRE Trans. Inf. Theory[J], vol. 8, p. 21–28, Jan. 1962.

[2] Hailes, P., et al., A Survey of FPGA-Based LDPC Decoders[J]. IEEE Communications Surveys and Tutorials, 2016. 18(2): p. 1098-1122.

[3] Guilloud, F., et al., Generic Description and Synthesis of LDPC Decoders[J]. IEEE Transactions on Communications, 2007. 55(11): p. 2084-2091.

[4] Orlitsky, A., K. Viswanathan, and Z. Junan, Stopping set distribution of LDPC code ensembles[J]. IEEE Transactions on Information Theory, 2005. 51(3): p. 929-953.

电子信息工程学院 学院 通信工程 专业类 130231 班

学生 何 沃 洲

毕业设计（论文）时间： 2017 年 02 月 15 日至 2017 年 05 月 30 日

答辩时间： 2017 年 06 月 05 日

成　　绩：

指导教师： Jinhong Yuan、刁为民

兼职教师或答疑教师（并指出所负责部分）：

系（教研室） 主任（签字）：

注：任务书应该附在已完成的毕业设计（论文）的首页。

LDPC 译码器架构研究及低复杂度可重配置移位网络设计优化

学生：何沃洲

指导老师：Prof. Jinhong Yuan

摘 要

低密度奇偶校验（LDPC）码是数字通信系统里应用广泛的一种前向纠错编码。对于长码，LDPC 码可以通过迭代译码无限地逼近香农限。置信度传播（BP）算法为 LDPC 译码器的设计提供了一个框架，但在硬件的具体实现上，从并行度到节点调度都有很多可能的选择。

评价一个译码器性能的好坏离不开 7 个指标：吞吐量、处理延迟、硬件资源、纠错能力、能耗效率、带宽利用率以及灵活性。本文从这些最重要的参数出发，对 LDPC 译码器的优化设计进行了综述和总结，得出了最有利于设计者参考的若干共性。

此外，本文将通过 Benes 网络设计实现 LDPC 译码器中进行信息传递的一个重要模块——可重配置移位网络，并基于硬件资源消耗和关键路径长短两个评价指标对网络进行优化，得到适用于 WiMAX 和 WLAN 的 LDPC 码标准的最佳设计。实验结果表明，该网络能够以较低的资源消耗支持 140MHz 以上的时钟频率，在两个指标上达到了较好的折衷。

关键词：LDPC 译码器，译码器架构，置信度传播，可重配置移位网络

Research on LDPC Decoder Architecture and Design of Optimized Low-complexity Permutation Network

Author: Wozhou He

Supervisor: Prof. Jinhong Yuan

Abstract

The low-density parity-check (LDPC) code is a kind of forward error correction codes, which is widely applied to modern communication systems. LDPC codes can perform very close to Shannon linit by iterative decoding when the codeword is long. Based on the belief propagation (BP) algorithm, the design of a practical LDPC decoder has many options on its architecture, like parallelism and secheduling, when it comes to the hardware implementation.

There are seven key characteristics to evaluate the performance of a practical LDPC decoder, that is, the processing throughput, processing latency, hardware resource requirement, error correction capability, processing energy efficiency, bandwidth efficiency and flexibility. The paper makes a literature review on the optimized LDPC deocder architecture designs published based on the key characterisitics above and also summarizes the common practices they have made for further reference.

In addition, the paper designs and implements an important module of the LDPC decoder to perform the message-passing, the reconfigurable permutation network, by Benes Network. Also, the permutation network is optimized based on two key evaliation factors, the hardware resources and the critical path, according to the LDPC code standard of WiMAX and WLAN proposal. The results show that the permatation network can work with a clock frequency of over 140MHz and a relatively lower complexity, which achieves a good tradeoff between the two factors.

Key words: LDPC decoder, decoder architecture, belief propagation, reconfigurable permutation network

目 录

1 绪论 1
1.1 课题背景及目的 1
1.2 国内外研究状况 1
1.3 课题的研究方法 2
1.4 论文构成和内容安排 2
2 LDPC 译码器 4
2.1 数字通信系统概述 4
2.2 LDPC 码概述 5
2.3 LDPC 译码器结构 9
2.3.1 节点调度 10
2.3.2 检验节点和变量节点的更新 10
2.3.3 校验节点更新算法的简化 12
2.3.4 LDPC 译码器设计的其它要素 15
2.4 LDPC 译码器架构优化调研 16
2.4.1 LDPC 译码器的算法实现优化 18
2.4.2 LDPC 译码器的调度优化 21
2.4.3 LDPC 译码器的存储管理优化 24
3 可重配置移位网络 26
3.1 网络结构 27
3.2 网络控制 27
3.3 网络实现 28
3.3.1 控制信号产生算法 29
3.3.2 改进的控制信号产生算法 33
3.4 网络性能分析 36
结论 40
致谢 41
参考文献 42

1 绪论

1.1 课题背景及目的

低密度奇偶校验（Low-Density Parity Check, or LDPC）码是数字通信系统里广泛应用的一种差错控制码，代表着现代通信领域研究最为热门的一类前向纠错编码（Forward Error Correction, or FEC），尤其是在 2016 年 10 月 3GPP 组织选择了 LDPC 码作为 5G 的中长码编码方案之后。LDPC 码最早由麻省理工学院（MIT）的 Gallager 在其 1962 年的博士论文中提出。虽然有着优越的性能，但限于当时的计算机技术，LDPC 码在它问世的那个年代被认为复杂度过高而不适用于实际通信系统的应用，因此在长达数十年的时间里被忽略。到了 90 年代的时候，由于当时流行的 Turbo 码的专利保护，Mackay 和 Neal 重新研究了 LDPC 码而使它迎来了生涯里的高潮。由于当今机器日益增长的计算能力，LDPC 码已经成为了很多商业化通信系统的关键组成部分，例如 WLAN，WiMAX，DVB-S2，CCSDS 以及 ITU G.hn 等等。

LDPC 码有着一系列在实际通信应用中受到青睐的性质，特别是它可以由低复杂度的硬件处理器来译码。跟 Turbo 码一样，LDPC 码可以通过迭代运算进行译码，实现非常接近于香农限的纠错性能，特别是当码长很长的时候。但相比起 Turbo 码，LDPC 码的译码有着一系列的架构可供选择，包括采用的算法和并行度等等，这使得 LDPC 译码器有了更多可能的设计方案，也成为了现在的研究热点。因此，充分调研各种架构之间的区别和联系，通过比较得出它们的共通和差异之处，对于设计特定条件下的最优译码器架构，具有很重要的参考意义。

1.2 国内外研究状况

LDPC 码的研究自从 90 年代被重新发现时便开始兴起，其中，译码作为通信终端去除噪声干扰、恢复信源信息的最重要的步骤，更成为了 LDPC 码研究的重中之重。文献[1, 2]等对 LDPC 译码器的架构和算法做出了综述，其中文献[1]指出了 LDPC 译码器的最重要的几个性能指标之间的折衷平衡，可以作为分析一个译码器架构的出发点和设计一个译码器架构的依据。

一个 LDPC 译码器在置信度传播算法这个通用架构下的优化，在过去的 10 年里一直是研究的热点。从具体的硬件优化，到算法上的改进，再到结合码的构造进一步地降

低误码平台，研究者们一直努力用尽可能少的资源消耗，以尽可能好的性能向香农限一点点的逼近。由于 2.4 节进行了专门的综述，这里的绪论就先不展开讨论了。

1.3 课题的研究方法

LDPC 译码器的设计实际上是一系列性能指标之间的折衷，包括吞吐量（Processing Throughput），处理延迟（Processing Latency），硬件资源（Hardware Resource Requirement）、纠错能力（Error Correction Capability）、能耗效率（Processing Energy Efficiency）、带宽利用率（Bandwidth Efficiency）、灵活性（Flexibility）[1]。这些性能指标取决于译码器的系统设计，包括译码器架构（Architecture）、LDPC 码本身（LDPC Code employed）、所用的算法（Algorithm）以及迭代次数（Number of Iteration）等等。有两点是值得注意的：一是这些性能指标的好坏并不仅仅取决于译码器架构本身，例如带宽利用率跟调制方式有关，采用的码的结构直接影响纠错性能等等；另一点是各个性能指标之间都存在着折衷平衡，这个在后边会集中讨论。本文对各种架构的综述和分析，也将围绕这 7 个性能指标展开。

实际中衡量一个 LDPC 译码器的性能的时候，往往需要把它实现出来进行测试。其中最简单的方式就是利用 FPGA，通过硬件描述语言编程，然后综合和下载快速地进行模型的逻辑验证，尤其适用于比较误比特率（Bit Error Rate, or BER），因为一个跑若干天的计算机仿真在 FPGA 上可能只需要几个小时就完成了。经过 FPGA 验证之后，译码器的版图设计可能还会进一步制成专用集成电路（Application-Specific Integrated Circuits, or ASIC）。ASIC 相对 FPGA 开发成本更高但可以达到更好的性能。本文对架构设计进行验证时，主要依据 Vivado 2017.1 的布局布线后仿真结果，通过资源报告和关键路径报告等进行再优化。

1.4 论文构成和内容安排

本文首先对 LDPC 译码器的算法和结构进行了总结和整理，然后对文献中 LDPC 译码器的架构进行综述，最后针对译码器内移位网络的实现做了优化和验证。

本文的章节安排如下：

本章为绪论，简要介绍了 LDPC 码及其译码器的发展和研究背景，罗列了本文的研究方法和主要内容。

第 2 章将对数字通信系统中的 LDPC 码及其基础的译码算法、通用的硬件架构进行

概述，并综述了最近文献中关于译码器架构的优化设计。

第 3 章将基于 Benes 网络设计实现 LDPC 译码器中的一个重要模块——可重配置移位网络，通过资源消耗和关键路径的性能分析，得到针对 WiMAX 和 WLAN 的 LDPC 码标准的最优化设计。

最后对本文工作进行总结，得出结论，并提出未来的研究方向。

2　LDPC 译码器

本章从数字通信系统中的差错控制编译码出发，对 LDPC 码的构造和主要译码算法进行了概述，然后结合 LDPC 译码器的若干个最重要的性能指标，对关于译码架构的文献所提出的优化方案进行了综述，总结出其中最具参考意义的共性和特点。

2.1 数字通信系统概述

图 2.1 是一个简化的数字通信系统模型。信息 $\mathbf{m}=\{m_j\}_{j=1}^{K}$是一个 K bits 的向量，经过 FEC 编码之后成为 N bits 的码字 $\mathbf{c}=\{c_j\}_{j=1}^{N}$，其中$N>K$，这过程中增加了 $M=N-K$ 个冗余比特。码率 R 定义为信息比特数 K 和码字比特数 N 的比值：

$$R=\frac{K}{N}=1-\frac{M}{N} \tag{2.1}$$

M 个冗余比特由 K 个信息比特计算得到，并不携带额外的信息。然而，在 FEC 译码过程中，它们将参与接收码字中差错比特的检测及纠正。

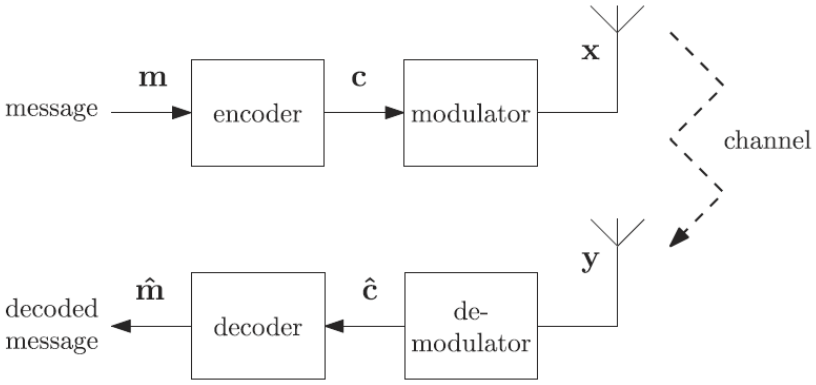


图 2.1　数字通信系统简单示意图

经过编码后的码字 **c** 经过调制后进入信道。在众多文献中，二进制相移键控（Binary Phase-Shift Key, or BPSK）是研究基于 FPGA 的 LDPC 译码器最为常用的调制方式，而实际的通信系统往往会使用更高阶的调制方式如 QPSK 或者 16QAM 等等。BPSK 将会产生经过调制的符号向量 $\mathbf{x}=\{x_j\}_{j=1}^{N}$，其中

$$x_j=\begin{cases}+\sqrt{E_S}, & c_j=0\\ -\sqrt{E_S}, & c_j=1\end{cases} \tag{2.2}$$

上式中E_S表示每符号的发送能量。对于信道，大部分的研究都会考虑加性高斯白噪声（Additive White Gaussian Noise, or AWGN）。跟一个随机的噪声向量叠加以后，接收

到的符号向量 **y** 可表示为

$$y_j = x_j + \mathcal{CG}(0, N_0) \tag{2.3}$$

上式中$\mathcal{CG}(0, N_0)$表示一个复高斯分布，其中N_0表示噪声的功率谱密度。

信噪比 SNR 表示为：

$$\text{SNR} = \frac{E_S}{N_0} = R \times \frac{E_b}{N_0} \tag{2.4}$$

y 经过解调得到接收码字$\hat{\mathbf{c}}$，$\hat{\mathbf{c}}$ 经过译码得到接收信息$\hat{\mathbf{m}}$，如图 2.1 所示。以下章节集中研究的内容就是对于 LDPC 码的译码过程设计。

FEC 译码器的纠错能力受到来自解调器（demodulator）的信息形式的影响。硬判决可以直接由符号向量得到码字向量，但实际应用中为了充分发挥 FEC 的纠错能力并不会这么做，而是采用软判决。软判决的依据是对数似然比（Logarithmic-Likelihood Ratio, or LLR）。LLR 可表示为后验概率（A Posteriori Probability, or APP）比值的对数：

$$\text{LLR}_i = \log \frac{P(c_i=0|y_i)}{P(c_i=1|y_i)} \tag{2.5}$$

上式中c_i表示发送的比特，y_i表示接收到的符号。LLR 的符号位决定对应的码字比特最有可能是 0 还是 1，而幅度大小则衡量这个判决正确的可能性有多大。之所以用对数形式来表示 LLR 是因为它有利于减小似然比的变化范围，使结果一般都会落在-10 到+10 之间，而且使概率之间的运算变成硬件更容易操作的加法运算。因此 LLR 被广泛地应用在 LDPC 译码中。

当使用 BPSK 调制加上 AWGN 信道时，解调器可以通过下面的关系式由接收符号 **y** 计算得到来自信道的信息 LLR：

$$\text{LLR}_i = 4 \times R \times \frac{E_b}{N_0} \times \text{Re}(y_i) \tag{2.6}$$

2.2 LDPC 码概述

这一节将会简要地概述 LDPC 码的结构以及编码。虽然这篇文章主要讨论 LDPC 译码器，但为了完整性在此也一并进行介绍。

上面已经提到，编码器在 *K* bits 的信息中增加 *M* bits 的冗余得到 *N* bits 的码字。如果编码后的码字仍然含有原来的 *K* 个信息比特便是系统码，如果已经没有了则是非系统码。*K* bits 的信息编码后将产生多达2^K种可能的排列。LDPC 码的纠错能力实际上就取决于这2^K种可能排列中任意两种之间的差异最小值，也就是最小汉明距（Minimum Hamming Distance）。具有较大的最小汉明距的码的两个码字之间便不容易混淆。

例如，一种系统码的信息比特数 K = 6、码字长度 N = 10，则冗余或者校验比特数 $R = N - K = 4$，码率 $R = \frac{K}{N} = 0.6$。这个码可以写成

$$\mathbf{c} = \{c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}\} \tag{2.7}$$

其中$c_1, c_2, c_3, c_4, c_5, c_6$是 6 个信息比特，而$c_7, c_8, c_9, c_{10}$是 4 个校验比特。校验比特可以由信息比特的模 2 和得到：

$$c_7 = c_4 \oplus c_6 \tag{2.8a}$$

$$c_8 = c_1 \oplus c_3 \oplus c_5 \oplus c_6 \tag{2.8b}$$

$$c_9 = c_2 \oplus c_5 \tag{2.8c}$$

$$c_{10} = c_1 \oplus c_2 \oplus c_6 \tag{2.8d}$$

以上方程就引入了生成矩阵 **G**（Generator Matrix）的概念，从而方便地表示编码的过程。对于系统码，**G** 可以表示为

$$\mathbf{G} = [\boldsymbol{I_K}\ \mathbf{A}] \tag{2.9}$$

上式中，$\boldsymbol{I_K}$是$K \times K$的单位矩阵，而**A**代表每一个方程的系数，这样 **G** 就可以写成：

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix} \tag{2.10}$$

这样系统码字 **c** 就可以通过 $\mathbf{c} = \mathbf{m} \times \mathbf{G}$ 得到。例如当发送信息是$\mathbf{m} = [0\ 1\ 1\ 1\ 0\ 1]$时，就得到码字$\mathbf{c} = [0\ 1\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 0]$。

把式 (2.8) 移项可以得到校验方程：

$$c_4 \oplus c_6 \oplus c_7 = 0 \tag{2.11a}$$

$$c_3 \oplus c_5 \oplus c_6 \oplus c_8 = 0 \tag{2.11b}$$

$$c_2 \oplus c_5 \oplus c_9 = 0 \tag{2.11c}$$

$$c_1 \oplus c_2 \oplus c_6 \oplus c_{10} = 0 \tag{2.11d}$$

同样地，校验方程也可以写成校验矩阵 **H**（Parity Check Matrix）的形式。系统码 **H** 矩阵的形式是：

$$\mathbf{H} = [\boldsymbol{A^T}\ I_M] \tag{2.12}$$

分别有对应每个校验方程的 M 行以及对应每个码字比特的 N 列。于是上面提到的这个码的校验矩阵 **H** 便是：

$$\mathbf{H}=\begin{bmatrix}0&0&0&1&0&1&1&0&0&0\\1&0&1&0&1&1&0&1&0&0\\0&1&0&0&1&0&0&0&1&0\\1&1&0&0&0&1&0&0&0&1\end{bmatrix} \tag{2.13}$$

LDPC 就是一种线性分组码。然而对于 LDPC 码，校验矩阵 **H** 一般并不写成这种形式。LDPC 码的纠错能力取决于 **H** 中每一行和每一列中非零元素的个数，也就是码重（weight）。更具体地来说，码重是 1 的列是无法进行纠错的，但这可以通过行变换来避免。(2.13)的 **H** 矩阵经过变换可以得到：

$$\mathbf{H}=\begin{bmatrix}1&0&0&1&1&0&1&0&1&1\\0&1&1&0&1&0&0&1&0&1\\1&1&1&0&0&1&0&1&1&0\\0&1&0&1&1&1&1&0&1&0\end{bmatrix} \tag{2.14}$$

这样形式的 **H** 矩阵消除了码重为 1 的列，但和实际使用的 LDPC 码依然有差距。LDPC 码的校验矩阵 **H** 是非常稀疏的，里面含 0 的个数远远多于 1 的个数。显然(2.14)的 **H** 矩阵并不符合这个要求，因为它的码长 N 只有 10，远远短于实际使用的往往长达数百甚至上千比特的 LDPC 码。

由于 **H** 矩阵对于 LDPC 译码的重要性，所以实际使用的 LDPC 码往往由 H 矩阵给出。设计一个具有强大纠错能力的码的 **H** 矩阵正是现在 LDPC 编码研究的重点。

校验矩阵 **H** 可以通过 Tanner 图的形式表示。作为描述 LDPC 码的一个数学模型，Tanner 图由两种节点及节点间的连线组成。M 个校验节点（Check Node）分别对应 **H** 矩阵每一行的校验方程，而 N 个变量节点（Variable Node）分别对应 **H** 矩阵每一列的码字比特。如果某个校验节点和某个变量节之间通过一条边（edge）连接表示 **H** 矩阵该行该列的元素是 1。因此 Tanner 图与 **H** 矩阵之间是一一对应的关系。通常用度数（degree）来表示与某个节点相连的其它节点数，所以校验节点和变量节点的度数实际上就对应于刚刚提到的 LDPC 码的行重和列重，是设计 LDPC 码时的重要参数。如果所有的校验节点都有一样的度数D_c，所有的变量节点都有一样的度数D_v，那么就把这样的 LDPC 码称为正则的（regular），否则就是非正则的（irregular）。对于正则 LDPC 码可以方便地得到：

$$D_c \times M = D_v \times N，即\ D_v = D_c \times (1-R) \tag{2.15}$$

下面通过图 2.2 给出的 LDPC 码就是非正则的。非正则 LDPC 码的度数用平均值来表示，例如对于图 2.2 的 LDPC 码 $D_c = 5.75, D_v = 2.3$。

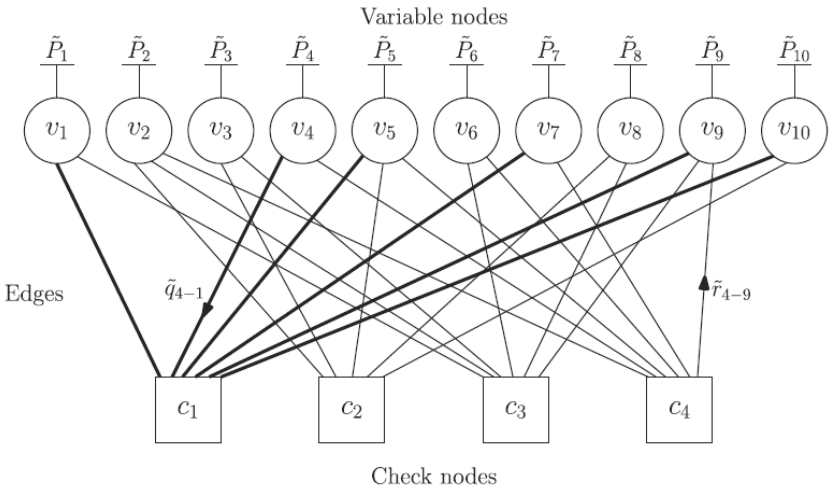


图 2.2　一种非正则 LDPC 码的 Tanner 图

当 Tanner 图不存在短环的时候，节点之间传递的外部信息相关性小，容易收敛到最优的译码结果。当 Tanner 图存在短的循环时，节点间外部信息的传递互相干扰而影响纠错性能。Gallager 在 1962 年提出的 LDPC 码的 **H** 矩阵就是随机构造的。当码长较长时，随机构造码可以获得很好的性能；但对于中短码长，Tanner 图中会引入较多短的闭环而影响性能。

码构造的核心问题是寻找不存在停止集（Stopping Set）[3]和 Tanner 图短环的 **H** 矩阵。目前主要的构造方法有：一定约束前提的随机搜索法（渐进边增长 Progressive Edge-Growth）、几何代数构造法（有限几何 LDPC 码 Finite Geometry LDPC Codes）等等。

早期的 LDPC 码构造大多由计算机按照一定的规则搜索得到，然后生成随机的排列，虽然性能很好，但是编译码结构复杂而不利于硬件实现，所以没有得到广泛应用。后来林舒等学者利用代数方法构造了有限几何空间的 LDPC 码得到的 **H** 矩阵具有循环或准循环结构，性能接近于随机构造的 LDPC 码但更利于硬件实现，因此非常合适高效快速的编译码，具有较高的应用价值。现实中通信系统使用的 LDPC 码标准基本都具有准循环结构（Quasi-Cyclic LDPC Codes, or QC-LDPC）。QC-LDPC 码的 **H** 矩阵可以划分为若干个大小相等的子矩阵块（Submatrix），每个子矩阵块要么是空矩阵（Null），要么由单位矩阵 **I** 经过循环位移得到。因此 **H** 矩阵可以利用基矩阵（Base Matrix）的形式来表示，如下面图 2.3 中 IEEE 802.11n (WLAN) 标准中某一种码的基矩阵表示，这样的结构非常有利于在硬件上实现低复杂度的存储器寻址或者网络连接，而又不至于在纠错性能上损失太多，所以在 DVB-S2、IEEE 802.11n、IEEE 802.16e 等标准上得到广泛的应用。

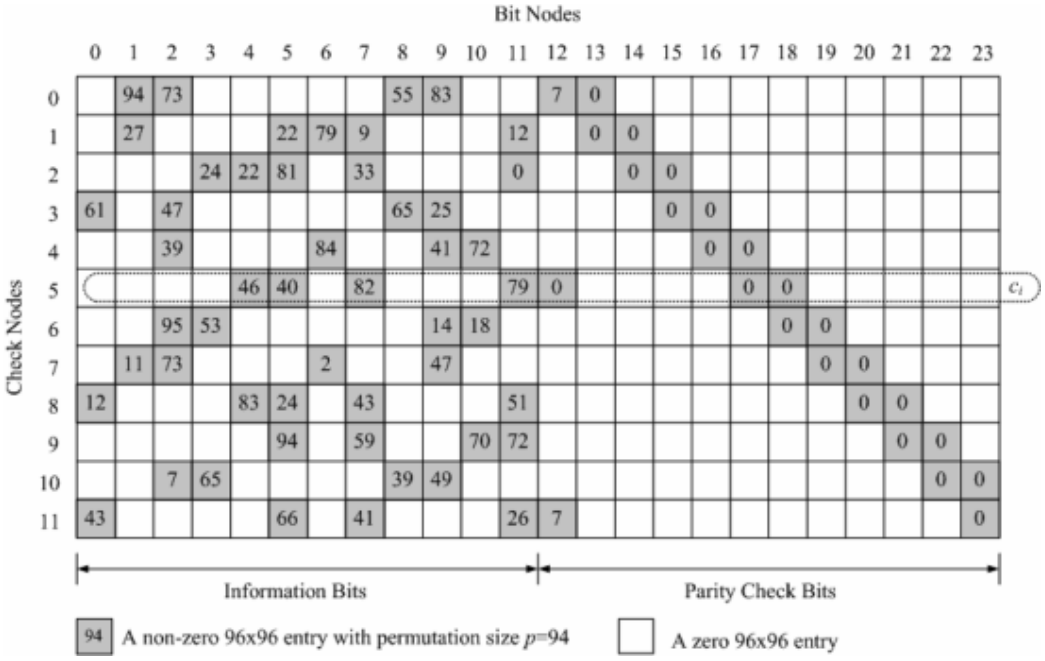


图 2.3 IEEE 802.11n 标准中其中一种码率 1/2 的 LDPC 码基矩阵[4]

2.3 LDPC 译码器结构

LDPC 译码的算法主要分为硬判决和软判决两种。硬判决算法包括比特翻转算法等等，虽然实现简单，但性能损失较大，在这不做讨论。软判决算法是指置信度传播（Belief Propagation, or BP）算法。BP 算法通过迭代达到较好的性能，而且有利于并行处理，是 LDPC 译码器研究的主流算法。

利用 BP 算法迭代译码的过程中，信息以 LLR 的形式通过 Tanner 图中节点间的连线在校验节点和变量节点之间传递。在上面图 2.2 的 Tanner 图中，校验节点c_1与变量节点v_1，v_4，v_7，v_{10}相连，接受来自这 4 个变量节点的信息并进行更新。值得注意的是，在 Tanner 图中传递的信息是外部信息（Extrinsic Information），也就是来自不包括目标节点的其它连接节点的信息的处理结果。例如，上图中c_1更新完之后传到v_1的外部信息由v_4，v_7和v_{10}传来的信息计算得到，传到v_4的外部信息则由v_1，v_7和v_{10}传来的信息得到。

信息传到某个节点后将首先把该节点“激活”，然后由节点更新得到送到与它相连的其它节点的外部信息。节点被“激活”的顺序称为调度（Scheduling），调度对 LDPC 译码器的纠错能力以及其它性能指标都有着非常重要的影响。下面篇幅将依次对 LDPC 译码器的调度以及检验节点和变量节点的更新方法进行概述。

2.3.1 节点调度

调度决定了 LDPC 译码过程中校验节点和变量节点更新的顺序。相关文献提出了 LDPC 译码器很多的调度算法，下面将介绍其中最为重要的两种：洪水调度（Flooding）和分层调度（Layered）。

洪水调度是最简单直接的调度算法，又称为双相信息传递调度（Two-Phase Message-Passing, or TPMP）。在每次迭代中，所有的校验节点会被同时“激活”，更新完之后将外部信息同时送到与其相连的变量节点，所有的变量节点也会被同时“激活”，更新得到外部信息并反馈到对应的检验节点，由此完成一次迭代。也就是说，检验节点和变量节点并没有同时处于更新状态，每一组节点都会等待另一组节点把信息更新完并传递过来之后再进行更新，这就约束了硬件利用效率（Hardware Utilization Efficiency, or HUE）的提高。改进方案将会在 2.4 节中进行讨论。

分层调度又称为 Turbo 译码信息传递（Turbo Decoding Message-Passing, or TDMP），可以分为基于行的分层（Row-Layered）以及基于列的分层（Column-Layered）两种，它们之间比较相似，下面以基于行的分层调度为例。基于行的分层调度每次更新 **H** 矩阵的一行（即对应某一个校验节点），这一行“1”位置对应的变量节点都会被更新，完成一次子迭代。当一个校验节点更新完毕以后就会转到下一个校验节点，直到所有的校验节点都依次被更新完成，则当前迭代结束。相比于洪水调度，分层调度的优点是每次子迭代结束后变量节点马上就得到更新，从而被后面的子迭代所利用，所以得以收敛得更快，相同条件下往往只需要洪水调度所需迭代次数的一半便可以收敛到同样的误码率，吞吐量得以提高。

2.3.2 检验节点和变量节点的更新

LDPC 译码过程中，变量节点和校验节点交替地进行更新。变量节点的更新相对比较简单，且不随算法变化而变化，只需要进行 LLR 的加和（包括来自解调器的内部信息 LLR）。而校验节点更新信息的算法有很多种，其中被研究得最多的就是和积算法（Sum-Product Algorithm, or SPA）以及最小和算法（Min-Sum Algorithm, or MSA），其中 MSA 实际上就是 SPA 的近似化简。

结合上一条所述的调度算法，以下将分别基于洪水调度和分层调度给出信息的更新和传递过程。实际上它们的基础是统一的，只不过结合不同的调度所表现出来的形式不

一样而已。

用λ_n表示第 n 个变量节点VN_n的内部消息（对于 BPSK 可参考 2.6 式），$R_{m,n}$表示第 m 个校验节点CN_m传递到第 n 个变量节点VN_n的 CTV 信息，$Q_{n,m}$表示第 n 个变量节点 VN_n传递到第 m 个校验节点CN_m的 VTC 信息，$\mathcal{M}(m)$表示与校验节点CN_m连接的变量节点的集合，$\mathcal{N}(n)$表示与变量节点VN_n连接的变量节点的集合，$\mathcal{M}(m)/n$表示与校验节点CN_m连接且排除了VN_n的变量节点的集合，$\mathcal{N}(n)/m$表示与变量节点VN_n连接且排除了CN_m的校验节点的集合。

对于洪水调度，信息传递和更新的过程可以表示如下[5]：

1、初始化：开始迭代时，对于所有的 M 个校验节点，把所有的 CTV 信息$R_{m,n}$都置为零，以后的每次迭代开始时从存储器中读取；

2、变量节点更新：在第 k 次迭代中，对于每个变量节点，由所有与其相连接的校验节点传来的 CTV 信息计算送到每个校验节点对应的 VTC 信息：

$$Q_{n,m}^{(k)} = \lambda_n + \sum_{m\prime\in\mathcal{N}(n)/m} R_{m',n}^{(k-1)} \tag{2.16}$$

3、校验节点更新：在第 k 次迭代中，对于每个变量节点，由所有与其相连接的校验节点传来的 CTV 信息计算送到每个对应校验节点的 VTC 信息：

$$R_{m,n}^{(k)} = \Big(\prod_{n'\in\mathcal{M}(m)/n} \mathrm{sign}\Big(Q_{n\prime,m}^{(k)}\Big)\Big) \cdot \psi^{-1}\Big[\sum_{n'\in\mathcal{M}(m)/n} \psi(|Q_{n\prime,m}^{(k)}|)\Big]$$

$$= \Big(\prod_{n'\in\frac{\mathcal{M}(m)}{n}} \mathrm{sign}\left(Q_{n',m}^{(k)}\right)\Big) \cdot \psi^{-1}\Big[\sum_{n'\in\mathcal{M}(m)} \psi\left(\left|Q_{n\prime,m}^{(k)}\right|\right) - \psi\left(\left|Q_{n,m}^{(k)}\right|\right)\Big] \tag{2.17}$$

其中，$\psi(x)$是 Gallager 函数，可表达为$\psi(x) = \psi^{-1}(x) = \ln\frac{e^x+1}{e^x-1}$

4、奇偶校验：对于 N 个变量节点，分别计算对应后验概率的 LLR

$$Q_n = \lambda_n + \sum_{m\in\mathcal{N}(n)} R_{m,n}^{(k)} \tag{2.18}$$

根据 LLR 进行判决$\widetilde{c_n} = \begin{cases} 1, \text{if } Q_n \le 0 \\ 0, \text{其它} \end{cases}$

当$\widetilde{c_n} \cdot \mathbf{H}^{\mathbf{T}} = 0$，或者迭代次数达到预先设定的最大值$I_{ite}$时，便停止译码。

以上的迭代过程实际上对应于全并行的 LDPC 译码器架构。在这个架构中，每个变量节点分别对应一个变量节点处理单元（Variable-Node Processing Unit, or VNPU），每个校验节点也分别对应一个校验节点处理单元（Check-Node Processing Unit, or CNPU）。两种节点处理单元之间在硬件中由一个非阻塞网络进行连接。假设时钟频率用f_{clk}来表示，那个这个全并行 LDPC 译码器的吞吐量就可以表示为：

$$T_f = \frac{N \cdot f_{clk}}{2I_{ite}} \tag{2.19}$$

对于一个 N 比特长有 M 比特校验位的(D_v, D_c)正则 LDPC 码，设计一个全并行 LDPC 译码器需要 N 个D_v输入的 VNPU 以及 M 个D_c输入的 CNPU 以及一个复杂的连接网络，尽管吞吐量可以做到很高，但硬件复杂度同样很大，这样全并行的设计并不是最优的。关于译码器并行度的讨论将在下一节中提到。

对于分层调度，由于每次子迭代只在某一层（layer）进行而不是整个 **H** 矩阵，所以下面使用不同的符号以示区别。第 j 层的信息传递和更新的过程可以表示如下[6]：

1、初始化：从存储器中读取对应后验概率的 LLR Λ_n和对应先验概率的$r_{j,n}$（也就是 CTV 信息）；

2、计算 VTC 信息：

$$q_{j,n_j} = \Lambda_{n_j} - r_{j,n_j} \tag{2.20}$$

3、更新 CTV 信息：

$$\overline{r_{j,n_j}} = \left(\prod_{n_{j'} \in \mathcal{M}(j)/n_j} \operatorname{sign}\left(q_{j,n_{j'}}\right)\right) \cdot \psi^{-1}\left[\sum_{n_{j'} \in \mathcal{M}(j)/n_j} \psi\left(\left|q_{j,n_{j'}}\right|\right)\right] \tag{2.21}$$

4、回写对应后验概率的 LLR：

$$\overline{\Lambda_{n_j}} = q_{j,n_j} + \overline{r_{j,n_j}} \tag{2.22}$$

逐层完成更新之后便完成了一次迭代。从上面更新和传递的过程中发现，在分层调度中，CNPU 和 VNPU 之间功能的界限已经变得模糊了。也就是说，VNPU 实际上是可以省略的，因为简单的加和在 CNPU 也可以很方便地进行。所以分层更新的过程中实际上变成了 CNPU 进行更新并和 LLR 存储器及 CTV 信息存储器交换信息交替进行的过程。这样，对于每个子矩阵块行并行处理的 LDPC 译码器的吞吐量就可以表示为

$$T_f = \frac{N \cdot f_{clk}}{m_b \cdot I_{ite}} \tag{2.23}$$

分层调度有一个重要的前提，就是同时处理的层中“1”的位置不能重叠，否则在更新对应后验概率的 LLR 时就会发生冲突。也就是说，只要不违背这个前提，译码的时候就可以把不含有相同“1”位置的层任意地简并在一起更新。对于“1”位置有重叠的层的简并，在下一节 2.4.1 会有讨论。

2.3.3 校验节点更新算法的简化

上一条中校验节点更新用的算法是 SPA，下面对其进行简化得到 MSA。

上面(2.17)和(2.21)式使用的 Gallager 函数是一个递减的非线性函数，在硬件上只能

通过查找表(LUT)和局部线性拟合的方法来实现，会消耗很多的硬件资源。利用 Gallager 函数单调递减以及反函数是它自身的这两个性质，可以对(2.17)式做以下的近似简化：

$$R_{m,n}^{(k)} = \left(\prod_{n'\in\mathcal{M}(m)/n} \operatorname{sign}\left(Q_{n',m}^{(k)}\right)\right)\cdot\psi^{-1}\left[\sum_{n'\in\mathcal{M}(m)/n}\psi\left(\left|Q_{n',m}^{(k)}\right|\right)\right]$$

$$\approx \left(\prod_{n'\in\mathcal{M}(m)/n} \operatorname{sign}\left(Q_{n',m}^{(k)}\right)\right)\cdot\psi^{-1}\left[\psi\left(\min_{n'\in\mathcal{M}(m)/n}\left|Q_{n',m}^{(k)}\right|\right)\right]$$

$$= \left(\prod_{n'\in\mathcal{M}(m)/n} \operatorname{sign}\left(Q_{n',m}^{(k)}\right)\right)\cdot \min_{n'\in\mathcal{M}(m)/n}\left|Q_{n',m}^{(k)}\right| \tag{2.24}$$

(2.21)式的化简同理。以上的近似可以在纠错性能损失极小（通常在 0.5dB 以内[4]）的同时大大地降低 CNPU 的复杂度。由于 LDPC 译码器计算的复杂度很大程度上取决于 CNPU，所以 MSA 现在已经越来越多地取代 SPA 成为译码器节点更新算法的主流。此外，为了补偿用最小项近似两次 Gallager 函数的性能损失，MSA 往往会在形式上做一些修正，例如加上偏置β（Offset-MSA）或乘以修正因子α（Scaled-MSA），其中α的理论最佳值是 0.8，但为了硬件实现方便往往取为 0.75[7]。

根据 MSA 算法，检验节点更新时要完成的工作就是从D_c个 VTC 信息中找到绝对值最小的两个（分别记为min_1和min_2），以及min_1的序号，以及对 VTC 信息的符号位进行异或（XOR）计算。这里基于树状结构实现了适用于 IEEE 802.11n（WLAN）及 IEEE 802.16e（WiMAX）LDPC 码标准的 MSA 算法模块 MinSum_Processor。这两种码标准中行度数最高的是 22，所以 MinSum_Processor 把D_c设为 22，按照可能的最多输入进行设计。下面图 2.4 表示了分层调度中 CNPU 的结构[8]，可以看见 MinSum_Processor 的顶层通过数据选择器得到送往各个变量节点的 CTV 信息。CTV 信息的计算包括符号位的异或以及寻找最小值和次小值，其中 CNPU 的复杂度主要取决于后者。树状结构对输入的D_c个信息两两进行比较找到大值和小值，然后通过连接模块再两两比较找到每 4 个输入中的最小值和次小值，成为树状结构中的第二阶，后面的阶再两两比较如此类推，并传递最小值在输入中的序号，因此共需要($[\log_2 D_c]$+1)即 5 阶便可以得到D_c个信息中的最小值、次小值以及最小值的序号。

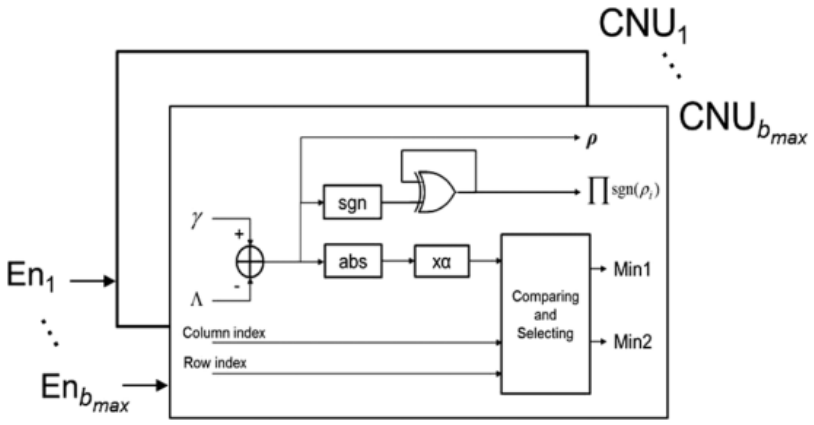


图 2.4　分层调度中 CNPU 的结构示意图

把 CTV 信息的位数设为 6（包括 1 位符号位），设定输入的最小值是 00001，来自高位的第 2 个信息，次小值是 00010，从下面图 2.5 中可以看到 MinSum_Processor 计算送往各变量节点的 CTV 信息，符号位等于除这一位以外其它变量节点的 VTC 信息符号位的异或，幅度等于除这一位以外其它变量节点的 VTC 信息幅度的最小值，满足 MSA 的要求。

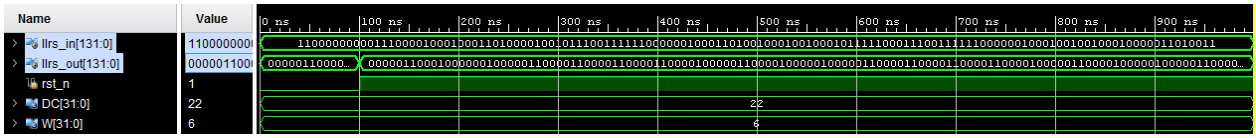


图 2.5　$D_c = 22$的 MinSum_Processor 输入输出关系

对校验节点利用利用一个时钟周期完成 MSA，在 Vivado 2017.1 平台上进行综合，综合和布局布线策略选择“Default”，器件选择 Xilinx Virtex-7，型号 x7vx485tffg1157-1。可以得到 MinSum_Processor 的主要性能，包括资源消耗（通过 LUT 和 FF 使用数来衡量）和最高时钟频率f_{max}，列举在下面的表 2.1 中。

表 2.1　MinSum_Processor 的资源消耗和最高时钟频率

LUT	FF	f_{max}
500	264	117M Hz

针对 MinSum_Processor 中寻找最小值、次小值及最小值序号的实现，文献[9]中有专门的讨论，从基于超前进位加法的两数比较出发，比较了分别基于树状（TS）、排序（XS）以及筛选（RS）的三种结构，发现 RS 结构能达到延时和资源的最佳折衷，也就是，平均分组并找到每组最小的两个数（暂不区分最小值和次小值），再两两比较层层选出最小的两个数，最后一次比较区分出最小值和次小值及最小值的序号，如下面图 2.6 所示。

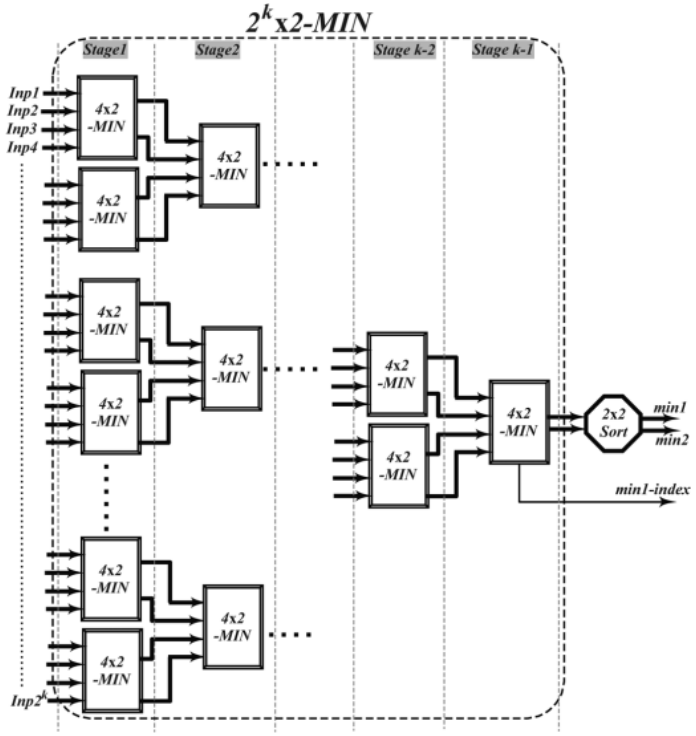


图 2.6 基于筛选（RS）的找最小值、次小值及最小值序号的结构示意图[9]

文献[10]还了进一步改进了 MSA，只找 VTC 信息的最小值而不找次小值（Single-MSA），当需要计算送往最小值信息来自的变量节点的 CTV 信息时，使用最小值加偏置 $\tilde{\beta}$ 修正来代替次小值从而补偿性能损失，其中 $\tilde{\beta}$ 取为 0.125。

2.3.4 LDPC 译码器设计的其它要素

除了以上提到的 LDPC 译码过程，一个译码器的具体实现还涉及一系列的设计要素，包括并行度（Parallelism）、LLR 的形式（Representation）以及停止迭代的标志（Stopping Criteria）等等。下面将逐条进行讨论。

以上已经提到，BP 算法实际上直接对应着全并行的 LDPC 译码器架构，在硬件上把每一个校验节点和变量节点对应的处理单元都分别实现，然后通过复杂的网络相连接。全并行的架构可以达到很高的吞吐量，但同时也要付出消耗大量硬件资源的代价。特别是对于码长达到上千的 LDPC 长码来说，节点间的连接网络消耗的硬件资源甚至可能比所有节点处理单元消耗的还要多。而且，全并行架构只能支持某种特定的 **H** 矩阵结构因此不具有多模式的灵活性。这使得全并行的译码器架构很不适合实际应用。与之相对的，全串行的译码器架构通过把资源时分复用，分别只利用一个处理单元来更新校验节点或变量节点，完成一个迭代可能需要上千个时钟周期。虽然全串行只消耗很少的硬件资源且灵活性好，但它的吞吐量往往没法达到实际要求。为了达到全并行高吞吐量和全串行

低复杂度的折衷，现实的 LDPC 译码器往往采用半并行（Partially-Parallel）的架构。这样架构的译码器有很灵活的并行度。而 QC-LDPC 码恰好非常适合半并行架构的实现。

LLR 在节点间传递时以若干位来表示。通过增加 LLR 的位宽来提高其分辨率及动态范围可以得到更好的纠错性能，但同样地必然带来硬件资源消耗的增加。对于 LLR 形式固定（包括小数点位置及位宽都保持不变）的设计，为了方便地选择最适合当前设计的 LLR 形式，可以参考 B.ten Brink 给出的外部信息转移图（Extrinsic Information Transfer Chart）[11]，从中可以直观地读出 LLR 的各种形式对译码器纠错能力的定量的影响。此外，研究指出 LLR 的非均匀量化可以一定程度上减少硬件资源消耗[12]，但同时也将因为引入更复杂的加法器使得硬件复杂度增加[13]。

LDPC 译码器设计时还需考虑满足什么样的条件时停止迭代译码。通常情况下，每次迭代结束后都会对当前的判决结果$\tilde{\mathbf{c}}$进行奇偶校验，如果发现是许用码字则停止译码。但在某些信道较不理想的情况下，接收码字$\hat{\mathbf{c}}$的错误比特太多而超出了纠错能力范围，如果不设定其它的停止标志的话可能将无限地迭代下去造成硬件和功耗的浪费。所以，一般还会设定当迭代次数已经达到了某个阈值I_{ite}的时候便让迭代停止。此外，某些设计也提到了另外的停止标志[6]，当连续两次或者几次的判决结果$\tilde{\mathbf{c}}$都没有发生变化，那么就停止译码（Early Stopping），这样的停止标志更多地应用在分层调度中，这样每一层更新结束之后都有可能停止译码，而不用等待整个迭代结束了再进行判断[14] 。

2.4 LDPC 译码器架构优化调研

当一个 LDPC 译码器在硬件上具体实现的时候，绪论中提到的 7 个性能指标就成为了评价一个译码器的依据，也就是，吞吐量、处理延迟、硬件资源、纠错能力、能耗效率、带宽利用率和灵活性。有三点值得说明的是：第一，这 7 个性能指标之间存在一定的依赖性，例如硬件资源消耗大的设计往往能耗也会相应地增加；第二，一个方面的改变往往会同时影响多个参数，例如并行度的增加可能同时带来吞吐量和硬件资源消耗的增加；另外，7 个性能指标也不仅仅取决于译码器本身，例如纠错能力受到码结构本身的影响，带宽利用率受到调制方式的影响。这就使 LDPC 译码器的设计引入了 7 个性能指标之间的折衷平衡。实际情况中往往无法面面俱到地到对所有的 7 个性能指标进行比较和讨论。以往的研究往往比较关注其中的 4 个，吞吐量、硬件资源、灵活性和纠错能力，它们可以比较方便地比较不同设计之间的差异，而其它的 3 个则关注得相对较少，

尤其是处理延迟，和吞吐量和码长有关，但很少被量化地讨论。但是，当研究 LDPC 译码器的硬件架构时，不能只关注某一两个性能指标，或者单纯地牺牲某一个性能指标换取另一个的改善，而是应该思考，在给定的条件下，怎么折衷地设计译码器的架构才能最大限度地满足目的，以及，基于所使用的 LDPC 码结构，如何相应地优化硬件做到 7 个性能指标的平衡点整体地改善。这一节将会从这一点出发，对近年关于 LDPC 译码器设计的主要文献进行调研，提取出最有利于参考的共性和特点。

LDPC 译码器的硬件实现主要有基于 FPGA 和基于 ASIC（VLSI）两种。他们的灵活性都可以用各自所支持的码标准种类来表示。对于 FPGA 实现来说，硬件资源可以通过 LUT 及 FF 的使用量来衡量；对于 ASIC 来说，则往往通过芯片面积来衡量。此外某些基于 ASIC 的文献会提到 pJ/bit/iter 为单位的能耗效率。还要注意的是，在具体比较 FPGA 实现和 ASIC 实现时要注意区分，因为 ASIC 作为成型的芯片往往具有比 FPGA 更高的集成度和处理速度，以及更好的可靠性。

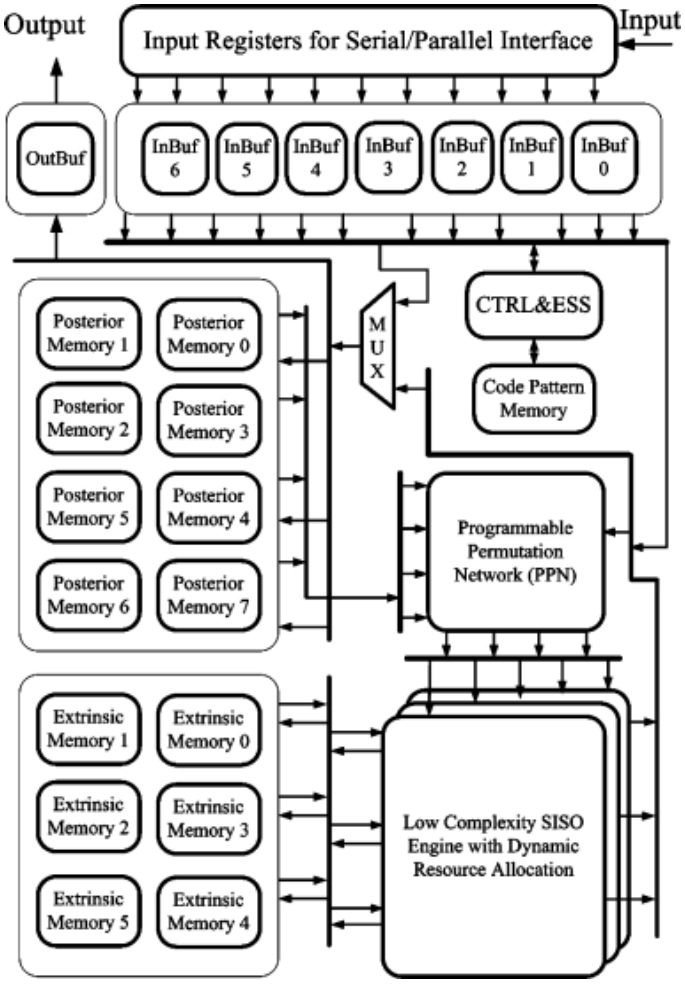


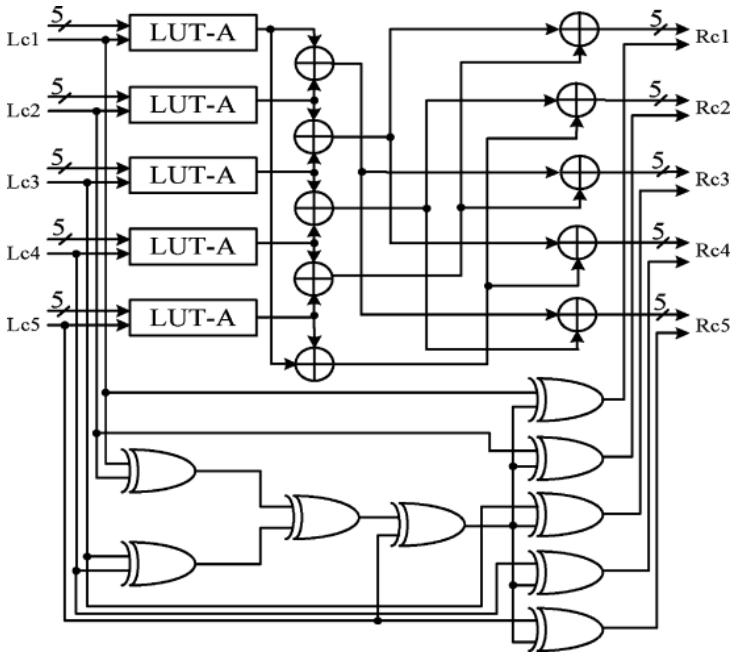
图 2.7　基于分层调度的 LDPC 译码器整体架构[4]

图 2.7 给出了基于分层调度的 LDPC 译码器架构。文献中提到的关于 LDPC 译码器的架构优化，可以大致分为几种：算法实现上的优化、调度上的优化、连接路径的优化和存储管理优化。其中连接路径的优化会在下章有专门的研究，所以在这先不详细讨论，其它的 3 个方面将一一展开。

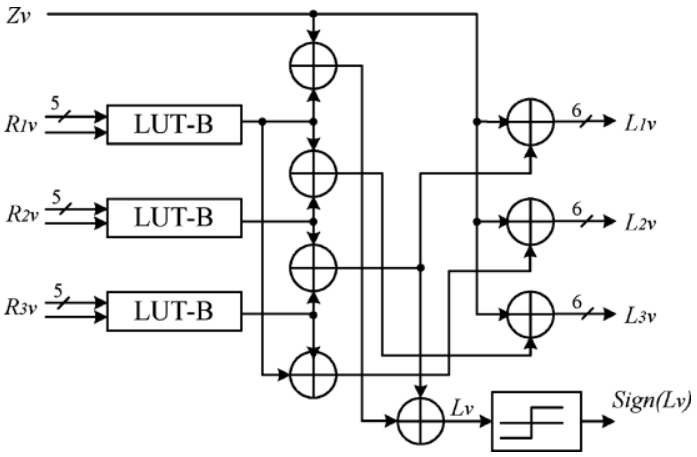
2.4.1 LDPC 译码器的算法实现优化

关于算法上的优化在 2.3.3 条已经提到过了，由 SPA 做近似简化成 MSA 可以在性能损失很小的同时大大降低硬件资源的消耗。此外也有文献提到其它形式的简化，它们一般都利用线性近似或者结合最值的计算来代替硬件代价较大的非线性 Gallager 函数，例如文献[4] 采用分段线性近似的方法逼近 SPA，做到性能损失在 0.1dB 左右。但是现在译码器算法的主流都选择硬件上最为经济的 MSA。所以，这里不会深入讨论算法上的优化，而是集中在算法的硬件实现优化上。

文献[15]和[13]提到了通过平衡 CNPU 和 VNPU 的计算负担来缩短关键路径。因为 LDPC 译码器的计算负担主要集中在 CNPU 上，对于 SPA，CNPU 需要完成两次查找表操作和一次加和，而 VNPU 只需要进行加和操作，这就大大延长了关键路径，使得最高时钟频率受到限制而约束了吞吐量，所以文献[15]把一次查找表操作转移到 VNPU 上，如下面图 2.8 所示，使得最高时钟频率提高。通过在 CNPU 和 VNPU 之间插入流水线还可以进行复用借此提高硬件的利用效率[15]。



(a)



(b)

图 2.8　计算负担平衡后的 CNPU (a) 和 VNPU (b)[15]

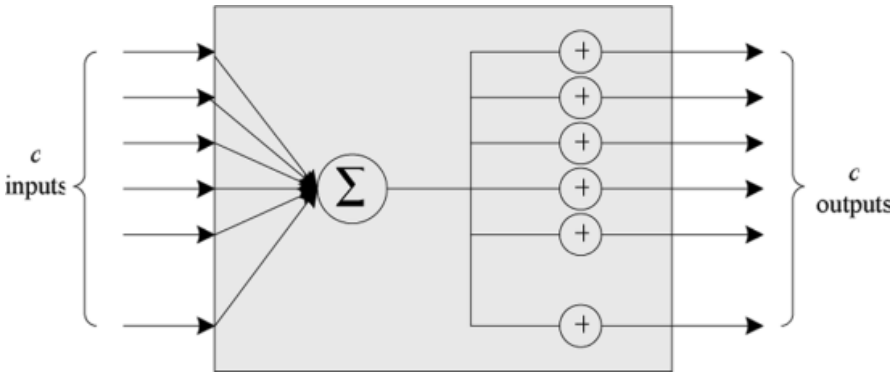
文献[16]和[6]都提到了在一相节点上恢复另一相节点的外部信息来降低硬件复杂度。对于洪水调度来说，文献[16]提到通过共用某些计算单元来使 VNPU 和 CNPU 重复对方的计算操作来分别恢复 CTV 信息和 VTC 信息，这样节点间的信息传递就可以由“单播”转变为“多播”，也就是通过增加节点的计算来简化连接网络的复杂度以减少信息传递的“拥堵”问题。对于分层调度来说，通过存储对应后验概率的 LLR 信息和 CTV 信息可使 VTC 信息通过前面两者的相减在 CNPU 上恢复，CNPU 只要再进行加和便可以使译码器省略 VNPU 模块，这一点在 2.3.2 描述分层调度过程的时候已经提及。

插入流水线贯穿于译码器的整个结构中，它对设计的影响主要有：把关键路径利用寄存器分割成若干时延相差不大的阶段，或者把整批处理利用流水线转变为分组处理，等等。文献[17]在行交叠的分层调度中通过对每一层的所有子矩阵块加上一个循环位移偏置使得相邻两层相同列开始更新的时间间隔在 5 个时钟周期以上，这样就可以利用流水线把每层的更新过程划分为延时相近的 5 个阶段，由此改善硬件利用效率和最高时钟频率。以 IEEE 802.16e 码率 1/2 码长 2304 比特的 LDPC 码基矩阵为例，偏置修正后的基矩阵如下面图 2.9 所示。上面提到平衡 CNPU 和 VNPU 的计算负担也是基于用流水线分割成时延相近的子过程的考虑。如下面图 2.10 所示，文献[5]提到通过把 VTC 信息分成数目相同的组，分批送到 CNPU 进行查找表操作和累加，批与批之间插入流水线，以此来改善洪水调度中 VNPU 和 CNPU 互相等待造成浪费资源的问题，提高了硬件复用的效率，也减小了处理延迟。文献[7]在 MSA 列分层调度中通过当前在处理的列的 VTC 信息最小及次小值估计相隔 p 列后的列的 VTC 信息最小及次小值，由此对每相隔 p 列的列同时进行 p 阶段的流水线更新，使一次迭代的处理速度提高了若干倍。可见流水线

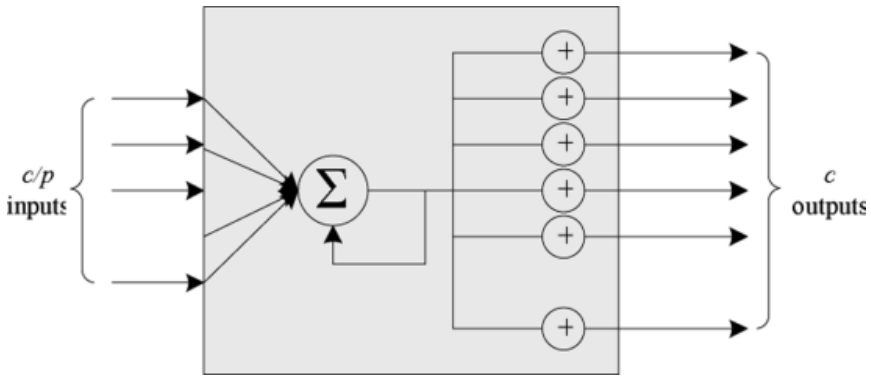
插入的设计适用于各种调度及各种算法的 LDPC 译码器。除此之外，流水线分割操作可以使相邻层[18]或相邻次迭代[19]的不同流水线阶段覆叠（Folding）从而提高硬件利用效率，分别如图 2.11 (a)和(b)所示。

Layer	Offset	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	+0		94	73						55	83			7	0										
2	+8		35				30	87	17				20		8	8									
3	+0				24	22	81		33				0			0	0								
4	+12	73		59						77	37						12	12							
5	+84			27				72			29	60						84	84						
6	+0					46	40		82				79	0					0	0					
7	+88			87	45						6	10								88	88				
8	+8		27	81				10			55										8	8			
9	+0	12				83	24		43				51									0	0		
10	+16						14		75			86	88										16	16	
11	+0			7	65					39	49													0	0
12	+80	27					50		25				10	87											80

图 2.9　偏置修正后的 IEEE 802.16e 码率 1/2 码长 2304 比特的 LDPC 码基矩阵[6]

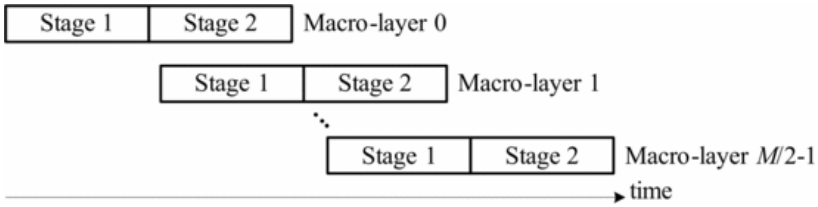


(a)

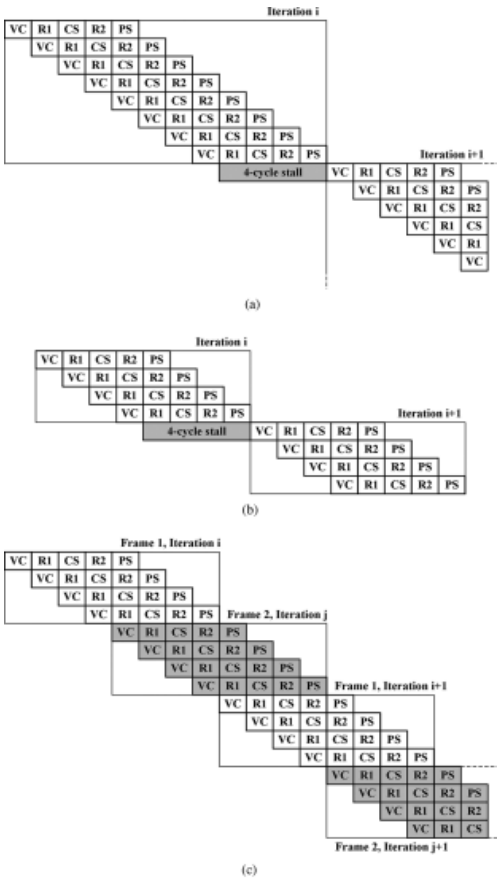


(b)

图 2.10　CTV 信息分组前(a) 送入 CNPU 和分组后(b)送入 CNPU 处理[5]



(a)



(b)

图 2.11 利用流水线分割对相邻更新的层(a)和迭代(b)进行不同流水线阶段的覆叠[18, 19]

2.4.2 LDPC 译码器的调度优化

2.3.1 节概述了两种主要的调度算法，洪水调度和分层调度，其中分层调度因为具有双倍的收敛速度，而且易于利用半并行架构和进行存储器的优化，所以成为了现在研究的主流。其中，基于行的分层和基于列的分层具有相似的收敛速度，但列分层之前一直被认为计算复杂度较高而没有受到重视，但最近有研究对列分层更新算法进行了优化使得列分层更具竞争力，包括更短的关键路径和译码延迟，所以逐渐受到更多的关注。

2.3.2 概述洪水调度的时候曾经指出 CNPU 和 VNPU 互相等待造成硬件资源闲置的问题，这个在上一条的算法实现优化中已经提到通过平衡 CNPU 和 VNPU 以及把 VTC

信息分组加流水线来改善。但这样的方法毕竟只适用于 SPA，而且也无法做到硬件利用效率的 100%。文献[20]提出了在基矩阵一行的各子矩阵块同时更新的半并行架构中，当码矩阵满足一定条件时可以使 CNPU 和 VNPU 的更新做到完全的交叠，如下面图 2.12 所示，也就是把吞吐量提高了一倍，把硬件利用效率近似地提高到 100%。同时还给出了对应吞吐量最大增益时 CNPU 和 VNPU 在子矩阵块中开始更新的位置，并避免了存储器访问冲突的产生。

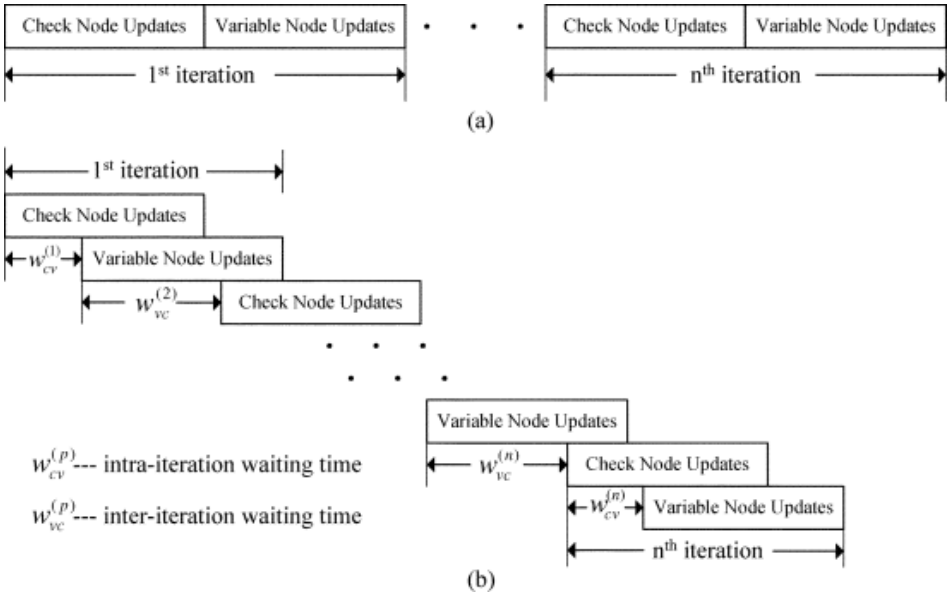


图 2.12 CNPU 与 VNPU 非交叠的更新顺序(a) 和交叠的更新顺序(b)[20]

2.3.2 节曾提到分层调度可以把任意 “1”位置不相同的层进行简并，这一点在很多讨论分层调度译码的文献中都有提及，例如文献[19]。但后来的研究指出，“1”位置重叠的层也可以通过算法的改进来进行简并，这在文献[18, 21, 22]有集中的讨论。其中，文献[22]针对大循环码重 K 的 QC-LDPC 码（如下面图 2.13 所示），对应后验概率的 LLR 存储器通过当前 CTV 信息和上一轮 CTV 信息的差的 K 次累加进行更新，以此来解决 LLR 更新时的冲突。文献[18]基于相同的原理合并两层相邻的 4 个子矩阵块构成大子矩阵块进行串行更新（如下面图 2.14 所示），文献[21]讨论了利用单一码重的标准 QC-LDPC 码 H 矩阵和覆叠矩阵的叠加来增强码的纠错能力，实际上这两种码结构跟上面的大循环码重 QC-LDPC 码是类似的，所以译码原理也是共通的。

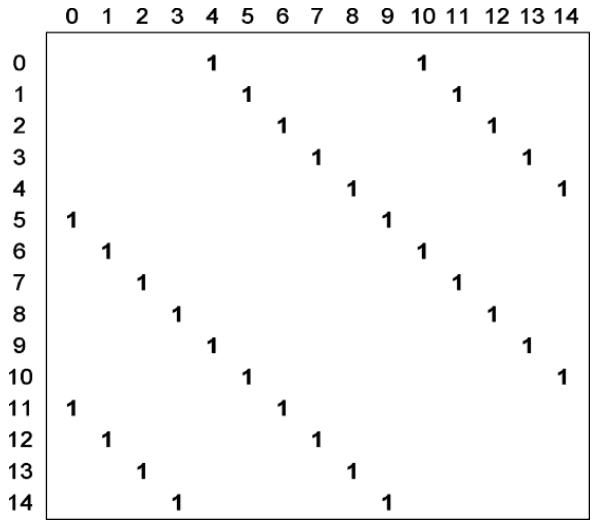


图 2.13　大循环码重 QC-LDPC 码的子矩阵块（$K = 2$）[15]

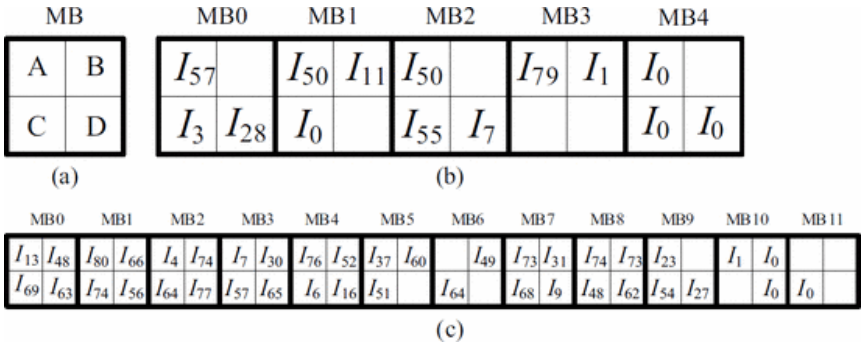


图 2.14　(a) 大矩阵块的构造 (b) IEEE802.11n 码率 1/2 的基矩阵前两行的合并
(c) IEEE802.11n 码率 5/6 码长 1944 比特的基矩阵的合并[18]

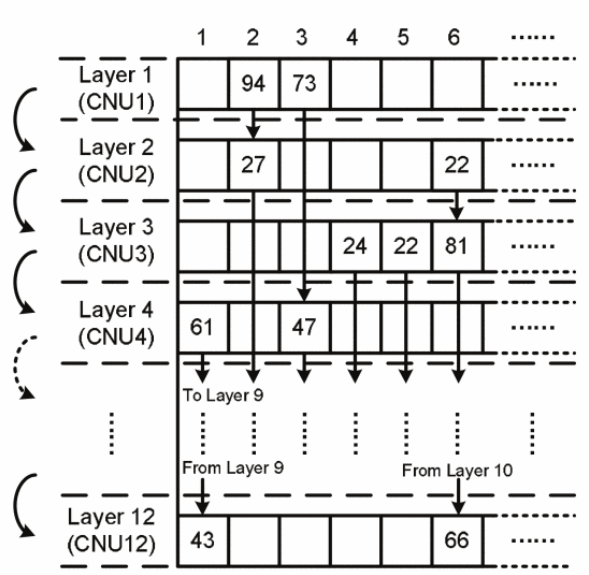


图 2.15　PLDA 中不同层同时更新时 LLR 信息传递示意图[6]

对于非空子矩阵块有交叠的不同层的同时更新，文献[17]提出了并行分层调度算法（Parallel Layered Decoding Algorithm, or PLDA），也就是通过调整层更新的顺序，使子矩阵块循环位移数比较大的层把对应变量节点更新完后的 LLR 及时地传递到循环位移数较小的层在它之后的更新时被使用，如上面图 2.15 所示。加上 2.4.1 刚提到的对每一层子矩阵块加上某个特定的偏置来人为地预先设计好更新顺序，这种 PLDA 架构在吞吐量和硬件利用效率上都可以做到较高的水平，但是需要预先对每一种支持的码结构做好调度规划和偏置优化。

2.4.3 LDPC 译码器的存储管理优化

计算数据如何进行存储管理是 LDPC 译码器设计中重要的一环。像上面提到的，分层调度中一般需要一个对应后验概率的 LLR 存储器以及一个 CTV 信息存储器，洪水调度中需要一个 CTV 信息存储器和一个 VTC 信息存储器，来保存迭代过程中持续更新的中间数据。

在全串行的架构中，CNPU 和 VNPU 只要依次地对 **H** 矩阵的“1”位置进行更新，不容易发生存储器访问冲突；全并行的架构中，CNPU 和 VNPU 可以各自对应某个存储单元，也不容易发生冲突。但在设计比较自由的半并行架构中，由于 CNPU 和 VNPU 或者多个 CNPU 可能会同时在某个时钟周期访问一个存储单元，这就造成了潜在的访问冲突。所以，存储管理优化解决的一个问题是，如何把存储器配置成合适的位宽及深度，并按照一定的合适的顺序来保存数据，以保证在等待时间和缓存硬件尽可能少的情况下使处理单元不会因为同时访问一个存储单元而读出错误的数据影响译码的进行，以及保证 VNPU 和 CNPU 读取数据是顺序交替的，不会重复读出还没有被更新的数据。

文献[15]探讨了把一个子矩阵块内的多条信息存储在同一个物理单元的可能性，给出了避免访问冲突的管理方案，并对一个子矩阵块内的两个 CNPU 及 VNPU 同时进行更新时的存储管理进行了优化。

文献[23]在这基础上进一步研究了存储管理优化的问题，提出了两种内存优化设计，即矢量化（Vectorization）和折叠化（Folding）。矢量化是指在 Block RAM 中配置足够宽的内存宽度从而把 N 条信息写入到同一个存储单元中，可以让译码的吞吐量提高将近 N 倍，还可以跟 2.4.2 中提到的 CNPU 和 VNPU 交叠相结合，但同时它也伴随着 CNPU 和 VNPU 的数量相应地增加，连接网络的复杂度随之上升，并可能因为增加的延迟而降低

时钟频率。因此内存矢量化应该结合具体的需求并考虑到整个译码器的设计。折叠化实际上是一种虚拟内存技术，也就是把多个子矩阵块的信息映射到同一个物理内存块中，从而通过 FPGA 上的少量嵌入内存来实现大码长大子矩阵块 QC-LDPC 码的译码，达到节省硬件资源的目的。同时，矢量化和折叠化可以同时使用，从而进一步提到单位硬件资源的吞吐量增益。

此外，文献[19]研究发现 LDPC 译码器里面存储器的访问频率足够 DRAM 的数据维持，提出了以非刷新的嵌入式 DRAM 代替原来的分布式 SRAM 来节省空间资源以及降低能耗，是存储管理优化的一个很好的思路。

3 可重配置移位网络 (Reconfigurable Permutation Network)

这一章将集中讨论 LDPC 译码器的一个重要组成模块——可重配置移位网络。由于移位网络耗费资源相对较多，关键路径较长，将直接影响整个 LDPC 译码器的硬件复杂度及吞吐量等性能。下面首先陈述移位网络的原理，然后根据几篇主流论文提出的算法进行验证并比较，在这基础上进行优化，最后得出适用于 WLAN（IEEE 802.11n）及 WiMAX（IEEE 802.16e）的 LDPC 码标准的最优设计。

在 LDPC 迭代译码的过程中，Message-Passing 在 VNPU 和 CNPU 之间交替进行。由于校验矩阵 **H** 中每一行中“1”的位置各不相同，每处理完一行转到下一行时，VNPU 和 CNPU 之间的连接都会发生变化。对于某一种给定 **H** 矩阵的 LDPC 码，我们可以预先确定 VNPU 和 CNPU 之间的连接关系并进行优化。但实际的无线通信系统中使用的 LDPC 码往往需要根据链路环境调整码率 R 及子矩阵块的大小 p，这就需要一个可重配置的移位网络（Permutation Network，以下简称 PN）实现 VNPU 和 CNPU 之间连接的动态变化，如下面图3.1所示。由于每次迭代或者子迭代中都需要两次信息传递，从CNPU传递 CTV 信息到 VNPU（Shuffle），以及从 VNPU 传递 VTC 信息到 CNPU（Re-shuffle），通过可重配置移位网络的时分复用就可以在不重复消耗硬件资源的情况下实现两次信息传递的功能[22]。

由于实际使用的 LDPC 码标准都具有准循环结构，每个子矩阵块都由单位矩阵 $\boldsymbol{I}$ 通过循环位移得到，所以我们设计的移位网络实际上只需要实现输入和输出在各个循环位移值 c 下的连接就可以了。当然，为了适应不同的子矩阵块大小，移位网络还须支持任意比它端口数 S 小的子矩阵块大小 p。其中 p，c，S之间满足关系式 (3.1)。

$$0 \le c \le p \le S \tag{3.1}$$

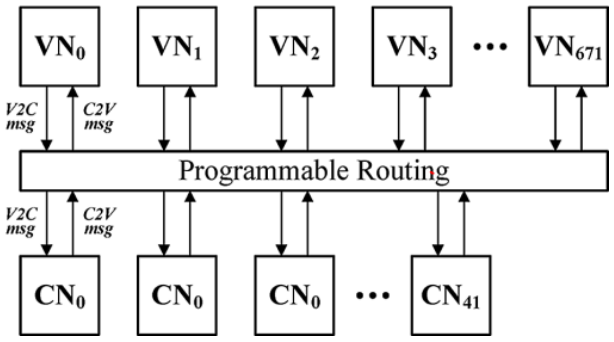


图 3.1 检验节点和变量节点通过可重配置移位网络实现动态连接

3.1 网络结构

可实现输入输出之间任意连接的网络结构早在上世纪 50 年代电话网普及的时候就开始研究了。数学上可以证明，对于$S \times S$的移位网络（S 是 2 的某次幂），一个非阻塞的网络至少包含（2n-1）阶，便可不用改变输入的排序而实现输出的任意排序。

Benes 网络和 Banyan 网络是讨论得最多的两种结构。$S \times S$的 Benes 网络可以实现 S 输入或比 S 更少的输入端口之间任意的非阻塞连接，其中每一阶都由2×2的开关单元构成，可动态设置为 BAR 状态（输入端口与对应位置的输出端口连接）或者 CROSS 状态（输入端口与输出端口交叉连接）。Banyan 网络实际上相当于 Benes 网络的后 n 阶，并不是严格意义上的非阻塞，对于 p 输出实现循环位移下的排序时还需要增加多工器（MUX）进行选择，所以并不适用于 p 不是 2 的某次幂以及 p 可变的情况，不能满足实际应用的需要。两种网络及开关单元的结构分别如下图 3.2 所示。

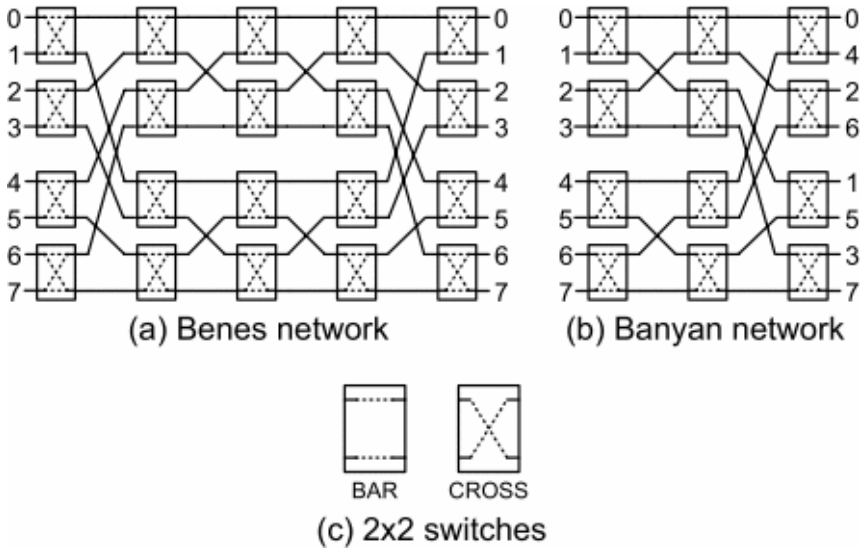


图 3.2 Benes 网络、Banyan 网络及开关单元[24]

3.2 网络控制

虽然 Benes 网络可以提供多达 $S!$ 种输出的排列组合，但对于只需要循环位移的情况排列数可减至 $\left[\sum_{i=1}^{S} i + (1-S)\right]$种。传统情况下使用 Benes 网络时会把控制信号都提前保存在一个专用的查找表（LUT）里[16]，但 LUT 的位长将随 S 的平方项成比例增长，对于 S 较大的 Benes 网络，这种方法的效率和可行性都不高，特别是当网络规模变大时对于 LUT 的资源消耗会很大，例如，查找控制信号所需的 LUT 位长是 $\left[\sum_{i=1}^{S} i + (1-S)\right] \times [\frac{p}{2}(2log_2 p - 1)]$，对于$32 \times 32$ 的网络是 71,568 bits，对于64×64 的网络则增

加到了 709,984 bits。

文献[25]提到了一种动态连接方法，由主从两个 Benes 网络构成可重配置移位网络，其中一个实现输入输出连接，而另一个辅助网络产生第一个 Benes 网络所需的控制信号组，具体来说，先根据 p 和 c 对输入信号进行重排列，然后通过控制从 Benes 网络产生特定排列的输出，把对应的控制信号组加到主 Benes 网络就可以得到希望的输出排列。虽然这种方法比起直接对每个输入端口各分配一个 M - 1 多工器方案要高效，但它包含两个 Benes 网络，硬件复杂度依然相对较高，辅助网络产生控制信号的计算负担也不小。移位网络产生控制信号比较实用的方法是通过组合逻辑实时产生[26]。文献[27]进一步研究指出，为了支持译码器的并行处理，一个 VNPU 可能把信息同时传给多个 CNPU，这时候就可以把要实现的循环位移值分解为为共同位移和各自位移的加和，分别通过一组固定的循环位移器和一个更简单的移位网络级联实现（因为各自位移变小所以移位网络的控制会变得简单）。虽然这种方法对于一个 VNPU 与一个 CNPU 之间每次信息传递的平均资源消耗是最小的，但大量使用多工器会使控制算法变得非常复杂，而且层层位移的叠加使得关键路径大大延长，不利于时钟频率的提高，所以这种方法暂时不作考虑。下面我们将基于几篇引用数比较多的文章提出的方案，通过一个 Benes 网络加上组合逻辑产生控制信号组来具体实现支持循环位移的可重配置移位网络，适用于 IEEE 802.11n（WLAN）及 IEEE 802.16e（WiMAX）的各种 LDPC 码标准（两种标准的码率和子矩阵块大小可见下表 3.1）。

表 3.1 IEEE 802.11n（WLAN）和 IEEE 802.16e（WiMAX）的 LDPC 码标准

	码率 R	子矩阵块大小 p	码长 n
IEEE 802.11n	1/2, 2/3, 3/4, 5/6	27, 54, 81	648, 1296, 1944
IEEE 802.16e	1/2, 2/3, 3/4, 5/6	24+4n,n={0,1,…,18}	576+96n,n={0,1,…,18}

3.3 网络实现

一个具有 S 个输入输出端口的 Benes 网络可分解为中间的两个$S/2 \times S/2$的子网络和直接由开关单元构成的输入输出两阶。同样地，两个子网络又可以类似地迭代分解，最终得到各阶的2×2开关单元，如图 3.3 所示。每个开关单元实际上就相当于两个 2 选 1 选择器的组合，可由对应的控制信号设置为 BAR（控制信号为 0）或者 CROSS（控制信号为 1）。下面将基于 Benes 网络这种迭代分解的结构特性，从硬件资源消耗和关键路

径长短等方面对一个可重配置移位网络的几种实现方式进行比较和优化。

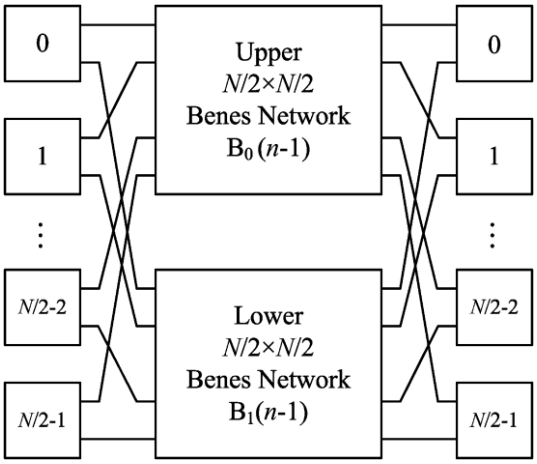


图 3.3 Benes 网络的迭代分解

3.3.1 控制信号产生算法

利用 Benes 网络的对称性质，通过解布尔方程组可以得到以下的实时控制算法[26]。用函数(p, c, S)表示通过$S \times S$的 Benes 网络实现 p 位输入的循环位移 c，这里的a_i表示输入阶第 i 个开关单元的控制信号，b_i表示输出阶第 i 个开关单元的控制信号，B_0和B_1是分解得到的两个子 Benes 网络。函数(p, c, S)可表示成如下的算法 A：

Initialization: $k = [p/2] + 1$, $m = [(p - c)/2]+1$
if (p[0] == 0) and (c[0] == 0) do begin
$a_0 = 0, a_1 = 0,\ldots, a_{k-1} = 0$
$b_0 = 0, b_1 = 0,\ldots, b_{k-1} = 0$
pass (p/2, c/2, S/2) to B_0
pass (p/2, c/2, S/2) to B_1
end
if (p[0] == 0) and (c[0] != 0) do begin
$a_0 = 1, a_1 = 1,\ldots, a_{k-1} = 1$
$b_0 = 0, b_1 = 0,\ldots, b_{k-1} = 0$
pass (p/2, (c+1)/2, S/2) to B_0
pass (p/2, (c-1)/2, S/2) to B_1
end
if (p[0] != 0) and (c[0] == 0) do begin
$a_0 = 0, a_1 = 0,\ldots, a_{m-2} = 0,\ a_{m-1} = 1,\ldots, a_{k-1} = 1$
$b_0 = 0, b_1 = 0,\ldots, b_{k-2} = 0, b_{k-1} = 1$
pass ((p-1)/2, c/2, S/2) to B_0
pass ((p+1)/2, c/2, S/2) to B_1
end
if (p[0] != 0) and (c[0] != 0) do begin
$a_0 = 1, a_1 = 1,\ldots, a_{m-1} = 1,\ a_m = 0,\ldots, a_{k-1} = 0$
$b_0 = 0, b_1 = 0,\ldots, b_{k-1} = 0,$
pass ((p+1)/2, (c+1)/2, S/2) to B_0

pass ((p-1)/2, (c-1)/2, S/2) to B_1
end

从硬件实现上看，基于 Benes 网络的可重配置移位网络如下面图 3.4 所示（用 IN 表示子矩阵块大小 p，SN 表示循环位移数 c），其中控制信号组的产生可分为两大部分，一个是参数计算模块（Parameter Generator，以下简称 PG），用于计算传递给子网络的两个参数(p, c)；另一个是控制信号产生模块（Control Signal Generator，以下简称 CSG），利用传递进来的两个参数计算输入阶和输出阶每一个开关单元对应的控制信号。其中，基于上述算法的 PG 可用下面图 3.5 的逻辑门电路实现。每一阶 CSG 分别计算第 i 阶和第$(2n\text{-}2\text{-}i)$阶开关单元的控制信号，可由常规的译码器（如 4 线-16 线译码器，把二进制编码变成某端口的高电平输出）加上一些门的组合逻辑产生控制信号。而最后一个 CSG 只用计算中间阶（第 n 阶）的控制信号，当输入的 p、c 分别等于 2 和 1 时控制信号取 1，否则取 0。CSG 的硬件结构如下面图 3.6 所示。这样，非阻塞 Benes 网络工作时每个开关单元所需的控制信号都可以通过迭代分解和层层传递得到。

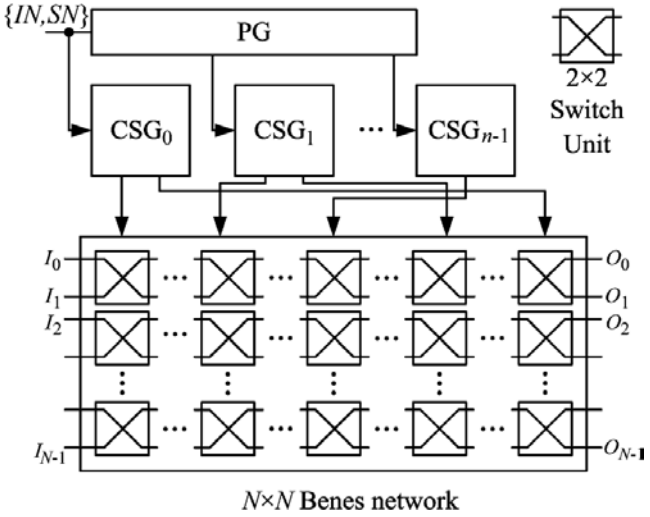


图 3.4　基于 Benes 网络的可重配置移位网络结构

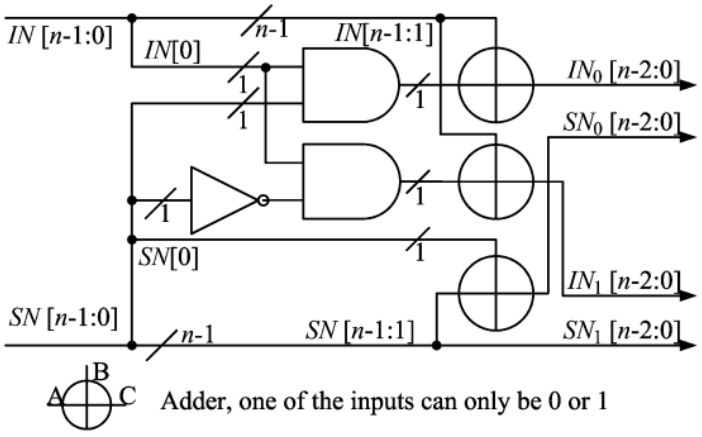


图 3.5　PG 模块的结构

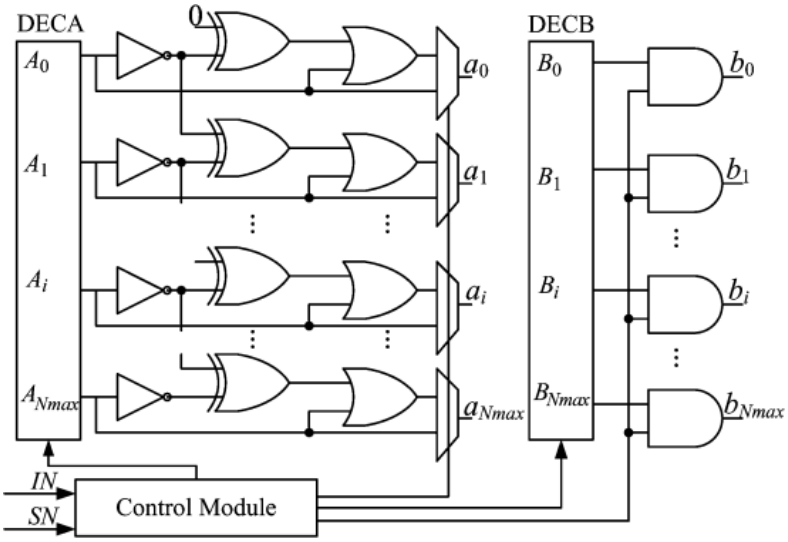
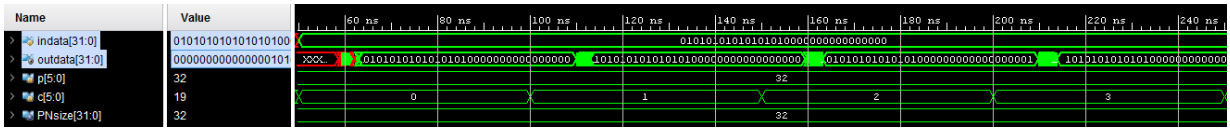


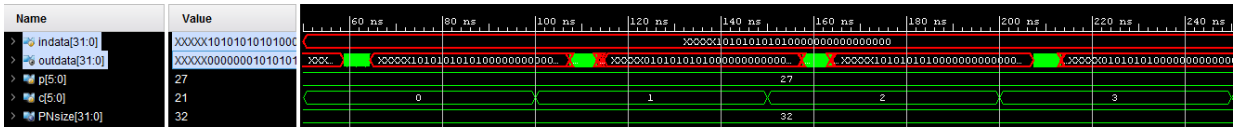
图 3.6　CSG 模块的结构

CSG 模块通过两个常规译码器 DECA 和 DECB 跟一些逻辑门的组合实现。DECA 和 DECB 分别对 IN 和 SN 进行译码。DECA 输出的低 (m-1) 位置低，高位置高；DECB 输出把对应二进制编码的位置高，其余置低。实际上，两个常规译码器可以合并为一个，进一步节省查找表资源，即把 DECB 往高扩展一位成为 DECB_EXTEND，取其除最高位外的其它位作为原来的 DECB，在 DECB_EXTEND 的每一个输出口接上或门，把或门输出接到相邻低位的或门的另一个输入，取除最低位的其它位的或门输出即可作为原来的 DECA，这样 DECB_EXTEND 便可同时完成两个常规译码器的功能。

利用上述结构实现32×32的 Benes 移位网络，经过布局布线后仿真及测试，当输入端口数 p 小于等于 32 时对于任意的循环位移 c 都可满足，如下面图 3.7 所示（以 p 分别为 32 和 27 时为例）。同理其它大小的 Benes 移位网络经验证亦可实现所要求的功能。



(a)



(b)

图 3.7　32×32可重配置移位网络经过布局布线后仿真验证可实现循环位移，(a) p = 32，(b) p = 27

对于算法 A，无论 p 和 c 如何取值，$b_0, b_1, \ldots, b_{k-2}$都恒定为 0。虽然 k 是可变的，但对于 IEEE 802.11n 及 IEEE 802.16e 这两个标准，p 的取值由 24 到 96，最后一阶（第

13 阶）的b_0~b_{10}必然为 0。同理可得第 13 阶的b_0~b_4、第 11 阶的b_0~b_1、第 10 阶的b_0也都一直保持为 0，对应的开关单元一直处于 BAR 状态，这时候我们就可以利用导线代替选择器达到节省硬件资源的目的。下面把算法 A 经过这样优化得到的算法称为算法 A++。

对于$S \times S$的 Benes 网络，S一般是 2 的某次幂。为了支持 IEEE 802.11n 及 IEEE 802.16e，S必须大于等于码标准中最大的子矩阵块大小p_{max}即 96，所以对于传统的 Benes 网络 S 应当至少取为 128。这就造成了硬件资源的极大浪费，因为译码过程最多只能使用其中的 96 个端口作为输入和输出，使得 Benes 网络中的很多开关单元或开关单元中的其中一个选择器处于闲置状态，有用的输入信号并不会从这些硬件上通过。文献[26]提出了一种改进方法，直接把这些闲置的硬件资源省去。

例如，对于128×128的 Benes 移位网络，一共有 13 阶，每阶有 64 个开关单元。只使用其中的 96 个端口的话，在输入阶（第 1 阶）和输出阶（第 13 阶）就可以各节省 16 个开关单元达 25%。实际上，在第 1~5 阶、第 9~13 阶的每一阶都可以节省 25%的开关单元。此外，处于中间的第 6~8 阶同样可以进行一定程度的优化。原本第 6~8 阶可视为 32 个并列的4×4的子 Benes 网络，但最下面的端口I_3和O_3实际上是没有信号通过的，这样就可以把I_2和I_3合并，把对应O_2和O_3的开关单元简化为一个选择器。这样的优化对于每个4×4子 Benes 网络可节省 30%的选择器资源，如下图 3.8 所示。省去了所有冗余的选择器后便可以得到96×96的可重配置移位网络。经过布局布线仿真及测试，该网络可实现 96 或 96 位以下输入的任意循环位移，如下面图 3.9 所示（以 $p = 81, c = 10$ 为例）。

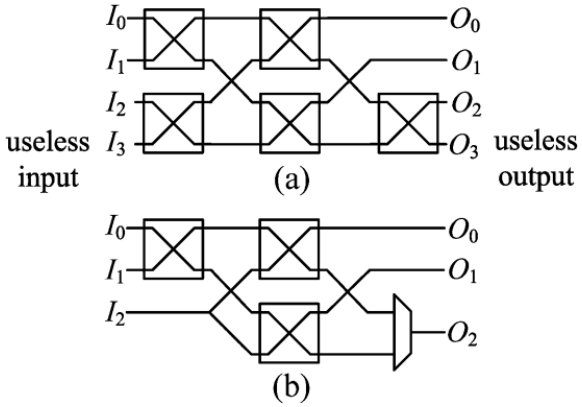


图 3.8 第 6~8 阶的4×4子网络优化前（a）和优化后（b）对比

图 3.9 128×128可重配置移位网络经过布局布线后仿真验证可实现循环位移($p = 81$， $c = 10$)

3.3.2 改进的控制信号产生算法

上述的控制信号产生算法可以实现位数比 S 小的输入信号的任意循环位移，但实际上算法的复杂度较高，尤其是当 IN 和 SN 分别是奇数和偶数和都是奇数的两种情况，需要复杂的组合逻辑才能产生对应的控制信号。文献[24]提出了另一种控制信号产生算法(p, c, S)如下面算法 B 所示。

Initialization: k = [p/2] , m = [(p - c)/2]
if (p[0] == 0) and (c[0] == 0) do begin
$a_0 = 0, a_1 = 0,\ldots, a_{k-1} = 0$
$b_0 = 0, b_1 = 0,\ldots, b_{k-1} = 0$
pass (p/2, c/2, S/2) to B_0
pass (p/2, c/2, S/2) to B_1
end
if (p[0] == 0) and (c[0] != 0) do begin
$a_0 = 0,\ldots, a_{k-1} = 0$
$b_0 = 1, b_1 = 1,\ldots, b_{k-1} = 0$
pass (p/2, c/2, S/2) to B_0
pass (p/2, (c+1)/2, S/2) to B_1
end
if (p[0] != 0) and (c[0] == 0) do begin
$a_0 = 0, a_1 = 0,\ldots, a_{m-1} = 0,\ a_m = 1,\ldots, a_{k-1} = 1$
$b_0 = 1, b_1 = 1,\ldots, b_{m-1} = 1, b_m = 0,\ldots, b_{k-1} = 0$
pass ((p-1)/2, c/2, S/2) to B_0
pass ((p+1)/2, c/2, S/2) to B_1
end
if (p[0] != 0) and (c[0] != 0) do begin
$a_0 = 0, a_1 = 0,\ldots, a_{m-1} = 0,\ a_m = 1,\ldots, a_{k-1} = 1$
$b_0 = 1, b_1 = 1,\ldots, b_{k-1} = 1$
pass ((p-1)/2, (c-1)/2, S/2) to B_0
pass ((p+1)/2, (c+1)/2, S/2) to B_1
end

看上去算法 B 似乎要比上一节提到的算法 A 复杂度要高，$b_0\ldots, b_{k-1}$不恒为零，因此不能如上述用导线代替化简某些开关单元。但是相比上述算法 A ，p 和 c 分别是奇数和偶数及都是奇数这两种比较复杂的情况，算法 B 可以大大简化 CSG 模块常规译码器及外围逻辑门的设计，如下面图 3.10 所示。

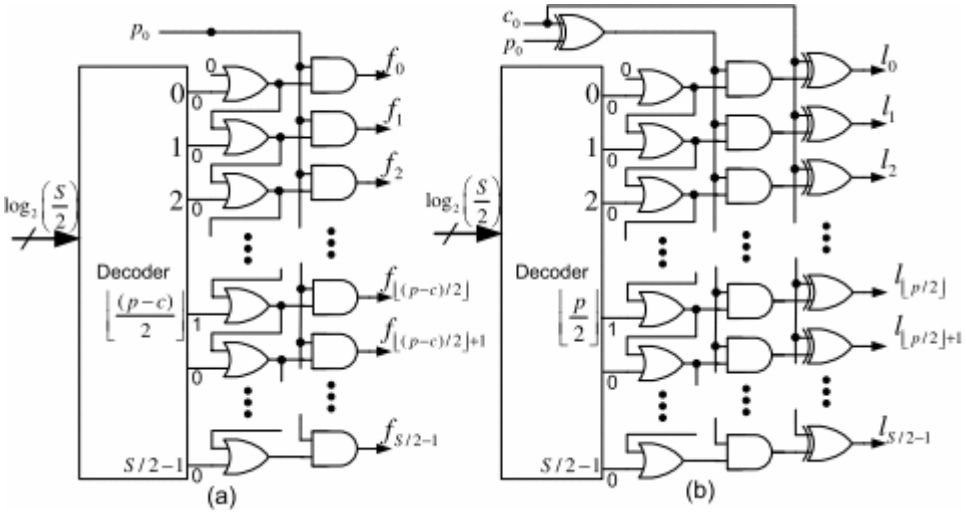


图 3.10　改进后的 CSG 模块（a、b 两个常规译码器可复用）

文献[28]提到，可重配置移位网络并一定使用传统的$S \times S$的 Benes 网络结构(S 是 2 的某次幂)，而可以是任意的$P_M \times P_M$，其中$P_M = JK, J \in \{2^i : i = 1,2,\ldots\}$, $K = \{2,3,5\}$，网络结构如下面图 3.11 所示。以下把这种改进的算法称为算法 B++。 按照这种新的结构，我们可以直接构造96×96的移位网络。当$P_M = 96$时，网络结构与上节所述的改进结构是类似的，除了中间 3 阶原来的 4×4 子 Benes 网络由新的 3×3 开关单元代替以外可看作完全相同，结构如下面图 3.12 所示。与图 5 优化前后的 4×4 子网络相比，在原来已节省 30%选择器的基础上还可以再节省 10%。，进一步优化了硬件利用效率。对于其它的码标准，文献[28]提到中间阶也可由 5×5 开关单元构造，原理类似故不再赘述。

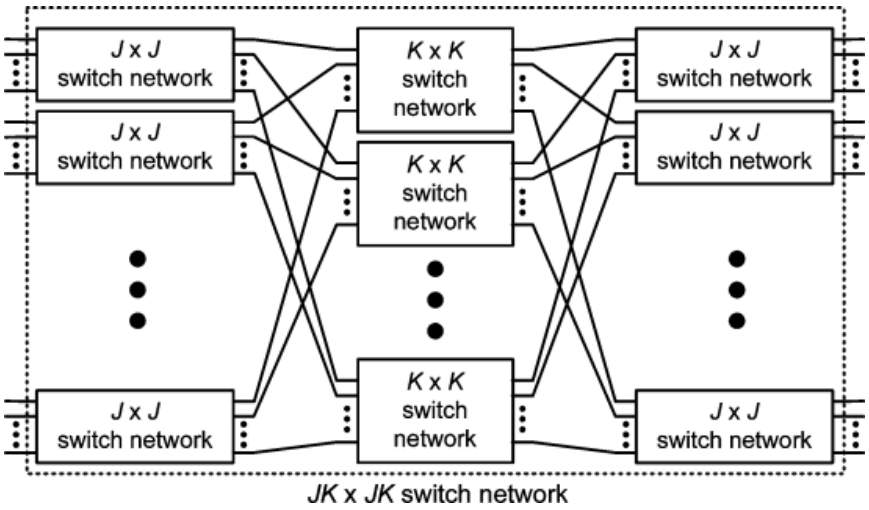


图 3.11　$P_M \times P_M$移位网络的结构[28]

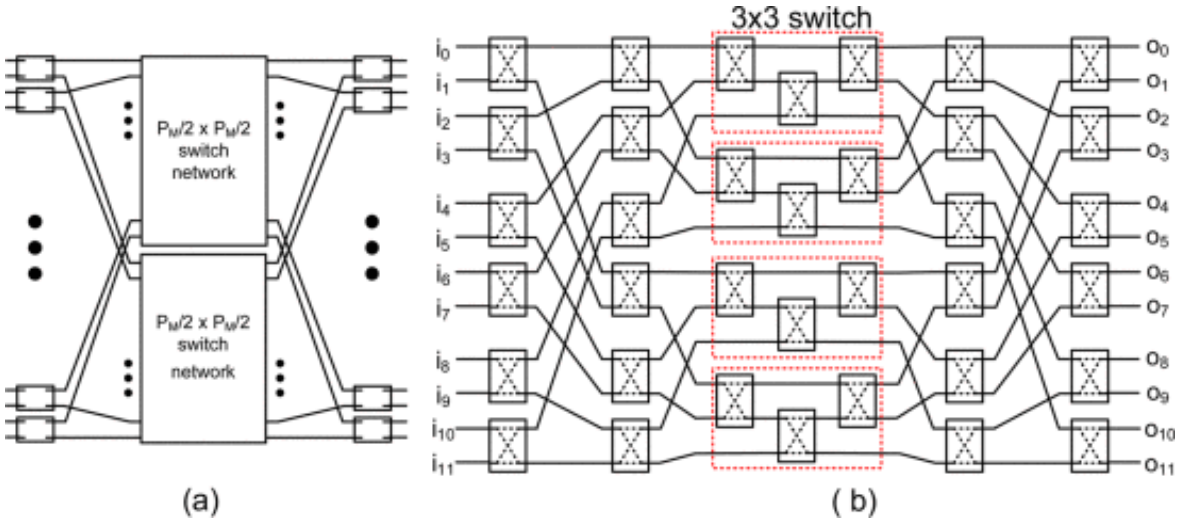


图 3.12　$P_M \times P_M$移位网络中间阶以3 × 3开关单元代替4 × 4 子 Benes 网络

每个3 × 3开关单元有三个2 × 2开关单元组成。把三个2 × 2开关单元由低位输入顺时针依次编号为（1）、（2）、（3）。当函数 (p, c, 3) 传递进来时，三个开关单元的控制信号可由下面表 3.2 得到：

表 3.2　函数(p, c, 3)对应的控制信号

开关 \ 参数	(3, 1, 3)	(3, 2, 3)	(2, 1, 3)	其它
（1）	0	1	0	0
（2）	1	0	1	0
（3）	1	1	0	0

令三个2 × 2 开关单元的控制信号分别为 *ctrl1*，*ctrl2*，*ctrl3*。参数为(2, 1, 3)时，控制信号亦可分别取为 1，0，0，但为了逻辑方便，统一取为 0，1，0。p 和 c 分别用两位表示时，控制信号可表达为:

$$\begin{cases} ctrl1 = p[1] * p[0] * c[1] * \overline{c[0]} \\ ctrl2 = p[1] * \overline{c[1]} * c[0] \\ ctrl3 = p[1] * p[0] * (c[0] \hat{} c[1]) \end{cases} \quad (4.2)$$

进而可通过组合逻辑电路产生三个2 × 2开关单元的控制信号从而使对应的3 × 3开关单元产生对应的循环位移。

简单地总结一下，3.3.1 和 3.3.2 两条一共提到了 3 类优化方法，可在传统的$S \times S$ Benes 网络结构的基础上节省大量硬件资源。第一类是对于控制信号保持不变的开关单元，可以直接用导线代替选择器进行信号的传递，以下把这类优化称为“可简化”，不过只适用于算法 A 而不适用于算法 B。第二类是对于没有信号流过的开关单元，可以直接

不进行实例化避免硬件资源的浪费，以下把这类优化称为“可省略”。第三类优化主要针对中间的第 6~8 阶，中间阶的每个4×4 子网络实际上只使用了 3 个输入和输出端口，所以可以引入3×3 开关单元进行中间阶的信号传递，以下把这类优化称为“可代替”。

对于适用于 WLAN（IEEE 802.11n）及 WiMAX（IEEE 802.16e）LDPC 码标准的可重配置移位网络，使用传统的 Benes 网络结构和优化后的网络结构对比，硬件资源的变化可由下面的图 3.13 呈现。

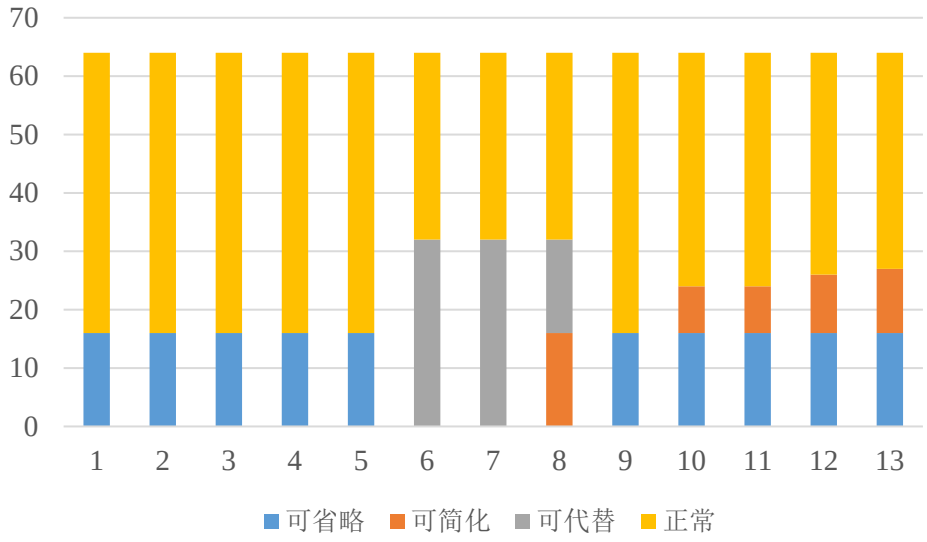


图 3.13 128×128 Benes 网络改进前后各阶开关单元数示意图

3.4 网络性能分析

上节陈述了 Benes 网络的实现并对网络结构进行了优化。这一节将从两个方面对网络性能进行分析，即硬件实现复杂度和关键路径长短，它们将直接影响译码器整体的硬件资源消耗和最高时钟频率。

FPGA 器件的内部结构如下面的图 3.14 所示[1]。基于 FPGA 实现，网络的复杂度可以通过 LUT 和 FF 使用的数量进行表征。由于信息经过可重配置移位网络传递只用一个时钟周期完成，所以网络设计时没有插入流水线，只在输入输出端口把信号寄存，所以网络不同实现所使用的 FF 数目实际上是一样的，那么只需要关注 LUT 的消耗就可以了。

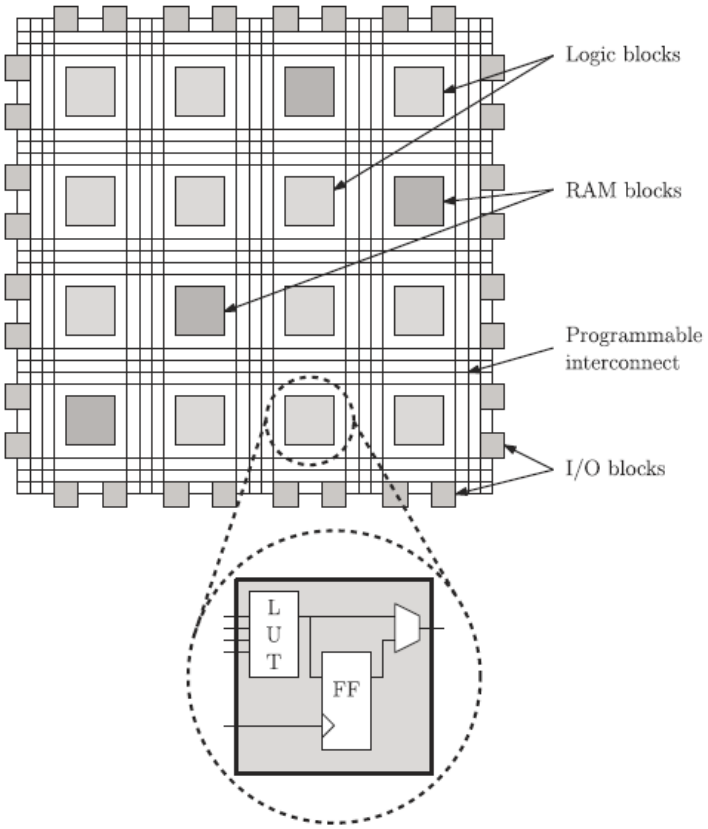


图 3.14　FPGA 器件的内部结构[1]

关键路径是信号到达对应的输出端口延时最长的一条路径，是系统提高时钟频率的瓶颈所在。实际上，路径的总延迟包括产生控制信号组的逻辑延迟和信号通过网络传递的传输延迟。这里用一个时钟周期完成网络传输的最高时钟频率f_{max}来表征关键路径的长短。系统在某个时钟频率下是否能够正常工作主要由两个参数来表征：最差负时序裕量（Worst Negative Slack, or WNS）以及最差保持时序裕量（Worst Hold Slack, or WHS）。当系统在最高时钟频率下工作时，所有的输入信号在一个时钟周期内恰好能够到达对应的输出端口或者只留下很短的松弛时间。如果时钟频率继续提高，两个裕量指标中至少一个会变成负值（瓶颈一般是 WNS），关键路径上的某些输入信号将可能无法在一个时钟周期内到达输出端口而出现错误的输出。当然，静态时序分析考虑的是比较极端的测试环境，但为了保险起见，系统工作频率不宜超过使 WNS 或 WHS 成为负值的阈值。

下面将会对上节提到的多种设计优化方案进行比较。这里使用的综合平台是 Vivado 2017.1 版本，综合和布局布线策略选择“Default”，器件是 Xilinx Virtex-7，型号 x7vx485tffg1157-1。

表 3.3 各个设计优化方案资源比较

设计	方案	使用 Slice LUT 数量
设计 1	算法 A 128 × 128	1973
设计 2	算法 B 128 × 128	1722
设计 3	算法 A 96 × 96	1616
设计 4	算法 A++ 96 × 96	1446
设计 5	算法 B 96 × 96	1298
设计 6	算法 B++ 96 × 96	1269

以上表格 3.3 中，128×128的可重配置移位网络可以完成任意位数不超过 128 的输入的任意循环位移，但对于适用 WiMAX 和 WLAN 标准的多模式 LDPC 译码器，网络只需要做到96×96就可以了。各个方案在上节都有提到，在此再做简单的说明。算法 A 和算法 B 分别在 3.3.1 和 3.3.2 中描述；从 128×128 到 96×96 节省了信号未流过的开关单元以及某些开关单元中信号未流过的选择器；算法 A++在算法 A 的基础上优化了“可代替”的开关单元；算法 B++在算法 B 的基础上引入了 3×3 开关单元优化了中间阶。

另外值得说明的一点是，Xilinx 近几代 FPGA 器件包括上面所用的 Virtex-7 使用的逻辑单元都是 Slice 6LUT，跟 FPGA 的另一主要提供商 Altera 使用的逻辑单元 LE 和 ALM 有所不同，不能直接进行比较。文献[1]中提到，为了方便与 Altera 器件常用的 4LUT 进行比较，可引入等效逻辑块（Equivalent Logic Block, or ELB）的概念，一个 4LUT 和一个 FF 的组合等效一个 ELB。对于 Xilinx 器件，一个 6LUT 可以近似地认为跟两个 4LUT 等效。

各个设计的关键路径长短也是有所不同的。以下表格 4.4 对消耗资源最少的 3 种设计 4~6 的最高时钟频率进行了比较。

表 3.4 各设计优化方案最高时钟频率比较

设计	方案	最高时钟频率f_{max}/Hz
设计 4	算法 A++ 96 × 96	140 M
设计 5	算法 B 96 × 96	147 M
设计 6	算法 B++ 96 × 96	144 M

通过比较可以发现，各设计的差异主要体现在硬件复杂度上。对于本文讨论的码标准，文献提出的$P_M \times P_M$可重配置移位网络可以达到最少的硬件资源消耗。节省的硬件

资源来自于对所有不必要的选择器的节省和优化（可参考图 3.12），以及控制信号组产生逻辑的简化。相比之下，这些设计在最高时钟频率上的差异并不大，可以支持高达 140MHz 的时钟频率。实际上各个设计在信号的网络传输延迟上是基本一样的，差异来自于控制信号组产生的逻辑延迟。设计 4 因为采用了逻辑较为繁琐的算法 A 所以关键路径稍长。

综合以上分析来看，对于 WiMAX 和 WLAN 码标准的多模式 LDPC 译码器，可以参考设计 6，既使所需的硬件资源最省，又可达到较高的时钟频率，取得性能和资源的最佳折衷。值得说明的是，假如码长增加、子矩阵块继续增大的话，网络和控制逻辑消耗的硬件资源将大致和端口数P_M的平方呈正相关。例如 5G 的 LDPC 中长码标准中最大的子矩阵块是512×512，用可重配置移位网络直接实现 CNPU 和 VNPU 连接的话，消耗的硬件资源是相当可观的。那么这种情况下，对可重配置移位网络的结构或者控制逻辑做出什么样的调整，或者通过其它方法如直接由存储器寻址来实现复杂的路由，这些都是值得进一步思考的问题。

结论

本文针对硬件实现 LDPC 译码器的架构进行了深入的理论分析和广泛的调研，并着重研究了其中进行信息传递的低复杂度连接网络设计实现。评价一个实际的 LDPC 译码器好坏离不开 7 个性能指标：吞吐量、处理延迟、硬件资源、纠错能力、能耗效率、带宽利用率以及灵活性。因此 LDPC 译码器架构优化的方向就是围绕着这 7 个指标的折衷平衡整体地提升性能，或最大限度地满足设计的需求。

本文的主要工作可以归结为以下几点：

（1）对 LDPC 基础的译码算法和译码器结构进行了理论整理。

（2）对文献已提出的 LDPC 译码器架构优化方法进行了调研和综述，从中提取出最具有参考意义的若干点共性及特点。

（3）针对 LDPC 译码器中信息传递的需要，研究了低复杂度可重配置移位网络的实现。在文献现有的若干种实现上进行了资源优化，得出了适用于 WiMAX 和 WLAN 中 LDPC 码标准的多模式可重配置移位网络的最优设计。最后对网络的资源消耗和关键路径等性能做出了评估。

本文综述的文献覆盖面依然有限，提出的低复杂度可重配置移位网络在码长和子矩阵块更大的情况下可能会遇到资源消耗的瓶颈。因此未来的工作还需对广泛的文献进行更深入的调研，并针对 LDPC 中长码的低复杂度路由方法如直接的存储器寻址做更进一步的研究，通过性能比较得出更具普遍性的结论供设计者参考。

致谢

时光荏苒，光阴似箭，四年的本科生涯行将以毕业设计的结束告一段落。感谢北航带给我四年充实的经历和难忘的回忆，给我本科最后一个学期外派交换、体验海外科研学习的机会。

借着论文完成之际，我向毕业设计期间给予我莫大支持和帮助的原教授、学长杨雷博士和实验室其它热情的小伙伴表达诚挚的感谢！原老师非常热情地接纳了我进入他的组里完成毕设，对我日常的科研学习提供了非常耐心的指导，生活上也对我无微不至地关怀。在忙碌的工作之余，原老师经常抽出时间跟我们一起讨论、一起运动、一起吃午饭，他深厚的学术造诣和平易近人的形象给我留下了非常深刻的印象！杨雷博士也是一位知识面广阔、治学严谨的学者，手把手地带我在 LDPC 译码这个领域从零基础到慢慢产生兴趣，然后做出一些成果来。学长平时虽然也忙于教课和指导其它 PhD，但总是能及时地关心我毕设的进度，给我新的灵感和方向，非常的尽心尽责！还有其它关心我帮助我的小伙伴，像谈笑风生的 Shane，乐于助人的志强，一丝不苟的 Bryan，篇幅所限无法一一罗列，他们都让我在悉尼的生活感受到别样的温暖！

我在新南威尔士大学的科研生活虽然只有转瞬即逝的三个多月时间，但是无论在科研上还是生活上，都开启了我诸多新的技能。远离了本该喧嚣热闹的毕业季，静静地投心毕设让我提前进入了研究生的生活节奏。这里有着浓厚的不一样的科研氛围，大家都非常专注地研究学术问题。我慢慢学着自己去解决问题甚至寻找问题，自己去思考前进的方向，体验了一段非常纯粹的科研经历，真的受益良多。感谢这段交换经历带给我的满满的收获和一生的回忆！

最后，还要感谢在国内关心我的父母和朋友们，感谢一直以来给我帮助的辅导员和给我指导的老师们，感谢百忙之中给我评阅论文和参加答辩的评委们！

参考文献

[1] Hailes, P., et al., *A Survey of FPGA-Based LDPC Decoders*[J]. Ieee Communications Surveys and Tutorials, 2016. **18**(2): p. 1098-1122.

[2] Guilloud, F., et al., *Generic Description and Synthesis of LDPC Decoders*[J]. IEEE Transactions on Communications, 2007. **55**(11): p. 2084-2091.

[3] Orlitsky, A., K. Viswanathan, and Z. Junan, *Stopping set distribution of LDPC code ensembles*[J]. IEEE Transactions on Information Theory, 2005. **51**(3): p. 929-953.

[4] Bao, D., et al., *Programmable Architecture for Flexi-Mode QC-LDPC Decoder Supporting Wireless LAN/MAN Applications and Beyond*[J]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2010. **57**(1): p. 125-138.

[5] Liu, L. and C.J.R. Shi, *Sliced Message Passing: High Throughput Overlapped Decoding of High-Rate Low-Density Parity-Check Codes*[J]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2008. **55**(11): p. 3697-3710.

[6] Zhang, K., X. Huang, and Z. Wang, *High-throughput layered decoder implementation for quasi-cyclic LDPC codes*[J]. IEEE Journal on Selected Areas in Communications, 2009. **27**(6): p. 985-994.

[7] Cui, Z., Z. Wang, and X. Zhang, *Reduced-complexity column-layered decoding and implementation for LDPC codes*[J]. IET Communications, 2011. **5**(15): p. 2177-2186.

[8] Lin, Y.M., et al., *Byte-Reconfigurable LDPC Codec Design With Application to High-Performance ECC of NAND Flash Memory Systems*[J]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2015. **62**(7): p. 1794-1804.

[9] Kumawat, S., et al., *High-Throughput LDPC-Decoder Architecture Using Efficient Comparison Techniques & Dynamic Multi-Frame Processing Schedule*[J]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2015. **62**(5): p. 1421-1430.

[10] Zhang, C., et al., *Flexible LDPC Decoder Design for Multigigabit-per-Second Applications*[J]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2010. **57**(1): p. 116-124.

[11] Brink, S.t., *Convergence behavior of iteratively decoded parallel concatenated codes*[J]. IEEE Transactions on Communications, 2001. **49**(10): p. 1727-1737.

[12] Zhiqiang, C. and W. Zhongfeng. *A 170 Mbps (8176, 7156) quasi-cyclic LDPC decoder implementation with FPGA*[A]. in *2006 IEEE International Symposium on Circuits and Systems*. 2006.

[13] Chen, N., Y. Dai, and Z. Yan. *Partly Parallel Overlapped Sum-Product Decoder Architectures for Quasi-Cyclic LDPC Codes*[A]. in *2006 IEEE Workshop on Signal Processing Systems Design and Implementation*. 2006.

[14] Hocevar, D.E. *A reduced complexity decoder architecture via layered decoding of LDPC codes*[A]. in *IEEE Workshop onSignal Processing Systems, 2004. SIPS 2004.* 2004.

[15] Wang, Z. and Z. Cui, *Low-Complexity High-Speed Decoder Design for Quasi-Cyclic LDPC Codes*[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2007. **15**(1): p. 104-114.

[16] Masera, G., F. Quaglio, and F. Vacca, *Implementation of a Flexible LDPC Decoder*[J]. IEEE Transactions on Circuits and Systems II: Express Briefs, 2007. **54**(6): p. 542-546.

[17] Zhang, K., X.M. Huang, and Z.F. Wang, *High-Throughput Layered Decoder Implementation for Quasi-Cyclic LDPC Codes*[J]. Ieee Journal on Selected Areas in Communications, 2009. **27**(6): p. 985-994.

[18] Sun, Y., G. Wang, and J.R. Cavallaro. *Multi-layer parallel decoding algorithm and vlsi architecture for quasi-cyclic LDPC codes*[A]. in *2011 IEEE International Symposium of Circuits and Systems (ISCAS)*. 2011.

[19] Park, Y.S., et al., *Low-Power High-Throughput LDPC Decoder Using Non-Refresh Embedded DRAM*[J]. IEEE Journal of Solid-State Circuits, 2014. **49**(3): p. 783-794.

[20] Dai, Y., Z. Yan, and N. Chen, *Optimal Overlapped Message Passing Decoding of Quasi-Cyclic LDPC Codes*[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2008. **16**(5): p. 565-578.

[21] Shin, B., et al., *Quasi-cyclic LDPC codes using overlapping matrices and their layered decoders*[J]. Aeu-International Journal of Electronics and Communications, 2014. **68**(5): p. 379-383.

[22] Sun, Y. and J.R. Cavallaro, *VLSI Architecture for Layered Decoding of QC-LDPC Codes With High Circulant Weight*[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2013. **21**(10): p. 1960-1964.

[23] Chen, X., et al., *Memory System Optimization for FPGA-Based Implementation of Quasi-Cyclic LDPC Codes Decoders*[J]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2011. **58**(1): p. 98-111.

[24] Oh, D. and K.K. Parhi. *Area efficient controller design of barrel shifters for reconfigurable LDPC decoders*[A]. in *2008 IEEE International Symposium on Circuits and Systems*. 2008.

[25] Tang, J., et al. *Reconfigurable Shuffle Network Design in LDPC Decoders*[A]. in *IEEE 17th International Conference on Application-specific Systems, Architectures and Processors (ASAP'06)*. 2006.

[26] Lin, J., et al., *Efficient Shuffle Network Architecture and Application for WiMAX LDPC Decoders*[J]. IEEE Transactions on Circuits and Systems II: Express Briefs, 2009. **56**(3): p. 215-219.

[27] Jung, Y., et al., *Low-complexity multi-way and reconfigurable cyclic shift network of QC-LDPC decoder for Wi-Fi/WIMAX applications*[J]. IEEE Transactions on Consumer Electronics, 2013. **59**(3): p. 467-475.

[28] Oh, D. and K.K. Parhi, *Low-Complexity Switch Network for Reconfigurable LDPC Decoders*[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2010. **18**(1): p. 85-94.