用于 Tbps 吞吐率通信的快速极化码

童佳杰[1]，王献斌[1]，张其蕃[2]，张华滋[2]，李榕[1]，王俊[1]

[1] 无线技术实验室

[2] 渥太华无线先进系统能力中心

摘要

面向高吞吐、低功耗通信，我们实现了两种用于极化码的串行抵消（Successive Cancellation，SC）译码器——展开型和递归型译码器。按16 nm ASIC技术换算，展开型译码器的面积和能耗效率分别为4 Tbps/mm^{2}和0.63 pJ/bit，递归型译码器的面积和能耗效率分别为561 Gbps/mm^{2}和1.21 pJ/bit。为了实现高吞吐率的目标，我们提出了一种全新的码构造方法，即快速极化码，与高度并行的SC译码架构共同优化。我们首先重用现有的模块，快速对额外的外码块进行译码。然后，通过修改码构造，加速对所有外码块译码，使并行度达到16。此外，针对译码器的硬件实现，我们对并行比较电路和比特量化方案进行了定制。总体而言，相对目前最先进的技术，我们的方案可将面积效率提高2.66倍，能耗降低33%。

关键词

快速极化码，Tbps 通信，快速译码，递归型译码器，展开型译码器

1 引言

1.1 研究动机和背景

在移动通信发展的过程中，提高吞吐率一直都是主要目标。虚拟现实（Virtual Reality，VR）和增强现实（Augmented Reality，AR）等应用有着很高的速率要求，为第六代移动通信系统（6G）提出了1 Tbps峰值吞吐率的要求[3]，这一数值是5G标准10~20 Gbps吞吐率的50~100倍。

要满足如此之高的速率要求，需要我们对物理层进行全新的设计，进一步降低复杂度和降低能耗，提升频谱效率，这对资源受限（处理能力、存储或能源等受限）的设备尤为重要。众所周知，信道编码需要消耗大量计算资源，是实现超高吞吐率的一大瓶颈。因此，6G要实现1 Tbps峰值吞吐率，就必须要在信道编码这一物理层技术上取得突破。

Erdal Arikan在[4]中提出的极化码是一种线性分组码。码长为N的极化码生成矩阵G_N，$G_N \triangleq F^{\otimes n}$。其中，码长$N = 2^n$，$F^{\otimes n}$表示矩阵$F = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$的第n个克罗内克积。串行抵消（Successive Cancellation，SC）是极化码的基本译码算法。

虽然SC译码算法因其串行特点可能并不适合高吞吐应用，但先进的SC译码器[1, 5–8] 还是成功地简化了译码过程，实现了并行译码。正因如此，SC译码在面积效率上遥遥领先于采用置信传播（Belief Propagation，BP）译码的低密度奇偶检查码（Low-Density Parity-Check Code，LDPC）。如图1a所示，这些译码器将SC译码表示为二叉树遍历[5]，每一个子树代表一个较短的极化码。原始的SC译码算法需要遍历树上所有的边和节点，因此译码时延高。简化的SC译码器可以对某些子树（较短的极化码）实现快速译码，因此实现了对子树的“修剪”。这样一来，译码时延就很大程度上由修剪后的二叉树上的边和节点的数量决定。[5, 9–10]提出了一些子树修剪的技术。为了实现1 Tbps的吞吐率目标，译码端和编码端都需要更强大的技术。

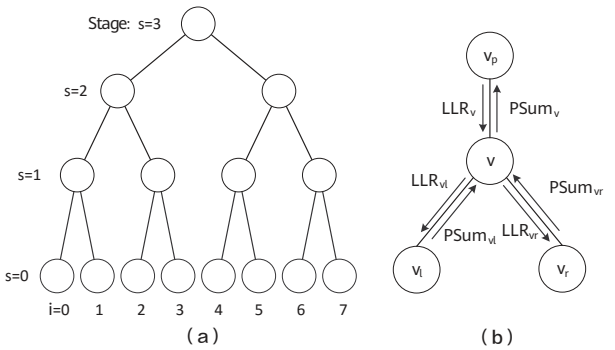

图1（a）二叉树译码结构；（b）节点v接收和响应信息

1.2 研究贡献

本文提出了一种名为“快速极化码”的新型极化码构造方法，便于SC译码器并行译码。我们的技术采用了编码和译码双端整体优化的策略，这点有别于现有技术仅关注译码端优化的策略。和现有的方法类似，从二叉树遍历的角度可以更好地理解我们方法的核心思想，包括以下几方面：（a）尽可能修剪子树；（b）将不可修剪的子树用其他相同码率且可快速译码的短码替代（嫁接），然后修剪这些“嫁接”来的子树；（c）通过改变码率，消除剩余的不可修剪的子树。可以看出，（b）和（c）都涉及对码构造的修改。因此，不论子树大小，我们都可以在不牺牲并行性的前提下对子树（短码）进行快速译码。

我们的算法贡献如下：

- 针对码率为$\{\frac{2}{M}, \frac{3}{M}, \frac{M-3}{M}, \frac{M-2}{M}\}$的节点，我们提出了四种快速译码模块，其中，$M = 2^s$是子树上叶节点的数量，$s$则是阶编号。这些叶节点分别称为双重复（Dual Repetition，REP-2）节点、重复的奇偶校验（Repeated Parity Check，RPC）节点、奇偶校验的重复（Parity Checked Reptition，PCR）节点、和双单向奇偶校验（Dual Single Parity Check，SPC-2）节点。更重要的一点是，这些模块重用了现有的用于重复（Repetition）和单向奇偶校验（Single Parity Check）节点的译码电路。
- 对于那些码率适中但无法原生支持快速译码的节点，嫁接两个扩展BCH码节点以替换原始的外极化码。由于BCH码具有良好的最小距离、原生支持高效的硬输入译码算法等优点，BCH码在性能和时延之间取得了很好的平衡。同时，扩展方法也进行了相应定制，进一步提升性能。
- 可以在全局范围内重新分配码率，让达到一定大小的节点都支持上述快速译码算法，避免遍历时遇到某些“慢”节点。

对于码长$N = 1024$、码率$R = 0.875$的码，我们所提出的快速极化码可以对所有长度为16的节点并行译码。与原始极化码相比，我们提出的译码算法减少了55%的节点访问、43.5%的边访问，性能损耗低于0.3 dB。我们同时设计了两种译码硬件，用于评估面积效率和能耗效率。

硬件实现的贡献概述如下：

- 设计了一种可以灵活支持任意码率以及码长不超过1024的递归型译码器。该译码器的估算芯片版图面积仅为0.045 mm^2。对于码长$N = 1024$、码率$R = 0.875$的码，该译码器可实现25.6 Gbps的编码比特吞吐率，面积效率为561 $Gbps/mm^2$。
- 设计了一种只支持单一码率、单一码长的展开型译码器。

该译码器的估算芯片版图面积仅为0.3 mm^2，对于码长$N = 1024$、码率$R = 0.875$的码，该译码器可实现1229 Gbps的编码比特吞吐率，面积效率为4096 $Gbps/mm^2$。

2 从简化SC译码到快速极化码

沿用[5]的表示方法，我们将树上的节点用v表示，其父节点用p_v表示，左子节点用v_l表示，右子节点用v_r表示。v与p_v、v_l、v_r直接相连[1]。节点v的阶数由和它最近的叶节点之间的边数决定，所有叶节点的阶数$s = 0$。以节点v为根的子树的节点集用V_v表示，那么V_{root}就表示满二叉译码树。所有叶节点的集合用U表示，叶节点u的索引值[5]用$i(u)$表示，U的索引值用$i(U)$表示。同时，子树V_v上的叶节点集用U_v表示，U_v的索引值用$i(U_v)$表示。

所有信息比特的位置集合用I表示，所有冻结比特的位置集合用I^C表示。子树V_v上的信息比特的位置集合用I_V表示，其他冻结比特位的集合则用I_V^C表示。

2.1 简化SC译码

如果I_V^C匹配码型，则可以触发对于该叶节点基于码型的简化译码，译码可以并行进行，无需逐比特进行。从二叉树遍历的角度来看，v的所有子节点都不在遍历范围中，译码时延因此降低。

现有的基于码型的简化译码包括4种不同类型。如果子树V_v上的所有叶节点都是信息比特，则节点v是Rate-1节点[5]，如果子树V_v中的所有叶节点都是冻结比特，则节点是Rate-0节点[5]。为了提高译码器的效率，[9]定义了SPC和REP节点，我们可以根据各节点的码型采用相应的方法，实现并行处理。但是，我们还需要识别并利用其他特殊节点或码型以进一步降低延迟。

我们在本文中定义了四种新节点：

- 如果V_v只有两个冻结比特，且两个冻结比特的索引值在$i(U_v)$中最小，则将节点v定义为SPC-2节点。
- 如果V_v只有两个信息比特，且两个信息比特的索引值在$i(U_v)$中最大，则将节点v定义为REP-2节点。
- 如果V_v只有三个冻结比特，且三个冻结比特的索引值在$i(U_v)$中最小，则将节点v定义为RPC节点。
- 如果V_v只有三个信息比特，且三个信息比特的索引值在$i(U_v)$中最大，则将节点v定义为PCR节点。

上述类型节点相应的快速译码方法将在第3节中介绍。

遍历子树时，如果节点匹配上述码型，基于码型的简化译码处理会跳过这些节点。

目前，有8种码型类型覆盖子树的8种码率：$\{0, \frac{1}{M}, \frac{2}{M}, \frac{3}{M}, \frac{M-3}{M}, \frac{M-2}{M}, \frac{M-1}{M}, 1\}$。换句话说，如果节点的码率不在上述范围，则不支持快速译码，我们需要在两个参数上下功夫：

- 简化节点的比例：目前M+1种码率中有8种码率支持简化译码，比例为$\frac{8}{M+1}$。请注意，只有最低和最高的码率可以简化，这意味着$\frac{3}{M}$和$\frac{M-3}{M}$之间的码率不支持快速译码。而中短长度的码由于极化不足，许多节点属于此范围。我们希望通过引入更多支持快速译码的码型来覆盖更多的码率，进一步降低时延。
- 并行度：简化节点中有M个比特是并行译码的，可以用M表示并行度。M值越大，二叉树可以通过简化译码修剪的比例就越大。我们希望增大M的值，提高吞吐量率。

当$M = 8$时，简化节点的比例为8/9，只有一个不支持的码率$(\frac{4}{8})$，但并行度只有8。当$M = 16$时，简化节点的比例降低到8/17，剩下了9个不支持的码率$(\frac{4}{16}, \cdots, \frac{12}{16})$，但并行度翻了一番。

2.2 BCH码

为了覆盖中等码率，我们需要找到支持快速译码并具有良好BLER性能的码型。然而据我们所知，对于码率在$\frac{3}{M}$和$\frac{M-3}{M}$之间的极化码，还没有并行译码的方法。此前许多研究，如[11-13]，提供了一个好的思路——可用任意码替换由子树表示的外码。将码率落入上述区间的极化节点移除，替换为其他支持快速译码的码，这是一种比较可行的方案。

BCH码由于其良好的最小距离和快速硬输入译码算法等特点，可作为理想的替代码。如果纠错能力为t，则很容易设计出最小汉明距离大于$2 \times t$的BCH码，提供良好的BLER性能。同时，伯利坎普－梅西（Berlekamp-Massey，BM）算法可以在极少的时钟周期内完成$t = 1$或$t = 2$的BCH码译码。将BCH码作为支持快速译码节点嫁接至极化码时，将内极化码（父节点）的对数似然比（Log-Likelihood Ratio，LLR）发送至外BCH码（子节点）之前，需要进行硬判决。在这里，我们将BCH码称为“BCH节点”。

但是仅靠BCH码不足以解决问题。BCH码只支持少数码率和码长，意味着它不能覆盖区间内的所有码率。当并行度$M = 16$时，目标码长为2^4，则BCH最近的码长为15。同时，BCH码只支持区间内$\frac{7}{15}$和$\frac{11}{15}$两种码率，对应的信息比特数为$k = 7$和$k = 11$。

[1] 叶节点v_{leaf}没有子节点，根节点v_{root}没有父节点。

为了解决这些问题，必须将码长增加到16比特。k = 7和t = 2的原始BCH码可以纠正两个错误比特，我们增加一个额外的比特作为所有BCH码的奇偶校验比特。我们所提出的两步硬译码的原理如下：硬判决产生三个误码，可以先用SPC比特纠正幅度最小的一个误码。对于余下两个误码，采用BM算法纠正。但是，相同的SPC扩展方法并不适用于k = 11、t = 1的BCH码，因为如果节点中存在两个或两个以上的误码，SPC函数和BM算法都无能为力。如果只有一个误码，SPC译码失败将进一步导致BM译码出错。为了提高可靠性，我们可以重复一个BCH码比特（而不是用SPC码扩展）。

现在，已经嫁接了两种BCH节点，基于码型的译码可以支持的码率达到10个。简化节点的比例提高至10/17，不支持快速译码的比特长度缩小到$\frac{4}{16}$。图3显示了基于码型的译码方法在并行度M = 16时支持的码率。

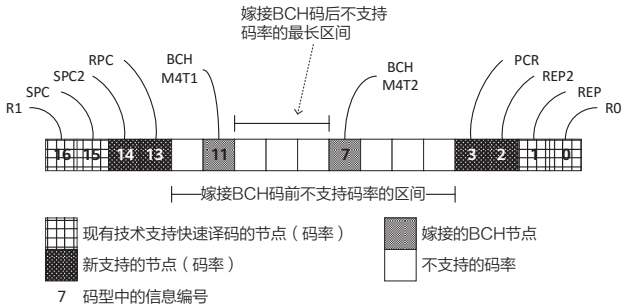

图 2 支持并行度 M = 16 快速译码的节点（码率）

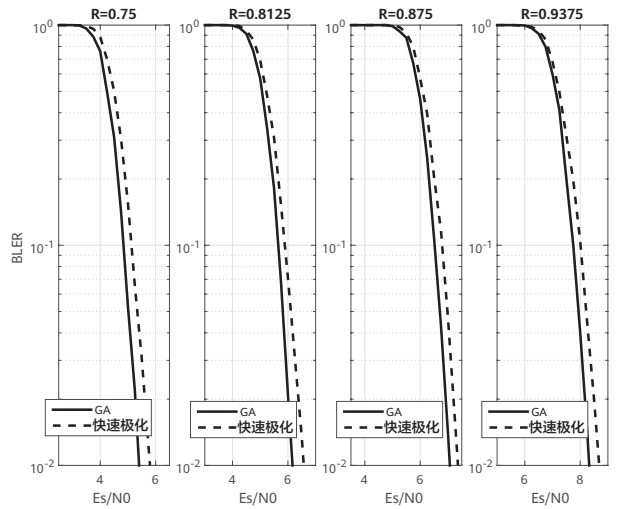

图 3 GA 与快速极化码构造法的 BLER 性能对比

2.3 重新分配码率实现快速极化码

即使嫁接了BCH节点，快速译码算法也不能支持长度为16的子树上的所有码率。作为解决方案的第二部分，我们提出快速极化码的构造，以规避“慢”节点，并且只使用现有的十种码型。这里所说的“快速”与快速SC译码速度相当，但是是通过调整码构造，而不是通过译码实现的。我们的实验证明，快速极化码以轻微的性能损失为代价，极大地降低了译码时延、提高了吞吐率。

以下步骤将说明如何仅使用非连续码率节点码型构造快速极化码：

1. 采用高斯近似（Gaussian Approximation，GA）或极化权重（Polarization Weight，PW）等传统方法构造码长为N、码率为R的极化码。
2. 将所有N个合成子信道拆分为N/16段。每段构成一个长度为16比特的分组码，相当于一个具有16个叶节点的子树。
3. 识别所有与受支持码率或码型不匹配的“慢”分段。在段与段之间重新分配码率，使其与最接近的受支持码率或码型（具有K个信息比特）匹配。
4. 如果当前分段中的信息比特数超过或少于K个，我们根据可靠性移除或增加相应数量的信息比特。在其他分段重复此操作，直到所有分段均支持快速译码。

这样得到的码就称为“快速极化码”。快速极化码构造算法请见如下算法1。

算法 1 构造快速极化码的方法

输入：

码长N，信息比特长度K，一组快速译码模式Θ。

输出：

逐节点重新分配码率，使所有节点都支持快速译码。
用GA或PW方法构造（N,K）极化码。
将码按长度16分段，段数用N_s表示。
逐步优化码结构。所有冻结比特位置都初始化为激活状态，并且“激活”比特位置可以在优化过程中转换为信息比特位置。

```
for t = 1··· N_s do
  while 第 t 段不属于Θ do
    用 i 表示第 t 段中最不可靠的信息比特位置。
    用 j 表示下一段中最可靠的激活状态的冻结比特位置，并用 k_j 表示该段中信息比特的数量。
    if k_j ⩾ 11且k_j < 16或 k_j < 3 then
      将 i 标记为冻结位位置，将 j 标记为信息位位置。
    else
      将 j 标记为非激活状态。
    end if
  end while
end for
```

表 1 GA 构造和快速极化码构造法遍历的节点、边和执行的 $f_{+/-}$ 函数数量对比

可快速译码节点分布							
GA				快速极化码			
Rate-1	4	SPC	20	Rate-1	2	SPC	9
SPC-2	2	RPC	0	SPC-2	1	RPC	0
PCR	1	REP-2	1	PCR	3	REP-2	1
REP	11	Rate-0	1	REP	1	Rate-0	1
BCH t = 1	0	BCH t = 2	0	BCH t = 1	3	BCH t = 2	2
以二叉树遍历维度统计的相关指标							
	GA		快速		减少比例（%）		
节点	40		22		55%		
$f_{+/-}$	4160		3792		8.9%		
边	76		43		43.5%		

以码长N = 1024、码率R = 0.875为例，统计需要访问的支持快速译码的节点的数量、需要执行的 $f_{+/-}$ 函数[14]、需要遍历的边的数量。基于这些数据，可以较为准确地估算SC译码的时延[5，9]，进而与本节所提出的GA构造方法进行对比（见表1）。可以看到，遍历的节点和边的数量分别减少了55%和43.5%，而 $f_{+/-}$ 函数数量仅减少了8.9%。需要注意的是，节点和边的遍历相对于 $f_{+/-}$ 函数受到的影响更大，因为两种函数无法以任何形式并行处理。

值得注意的是，我们所提出的快速极化码构造算法根据信道极化所得到的节点容量，将某些节点的码率进行了重新分配，这样必然会影响BLER性能。在码长N = 1024、码率R = {0.75, 0.8125, 0.875, 0.9375}的情况下，我们通过模拟得到了两种构造方法的BLER曲线，以评估BLER性能损失。采用QPSK调制时，GA极化码和快速极化码在BLER = 10^{-2}处的最大损耗为0.3 dB。

3 快速译码算法

本节我们将阐述用于支持新定义的SPC-2、REP-2、RPC和PCR节点快速译码的算法。对于BCH节点，我们采用经典的BM算法。该算法采用硬输入，支持硬件友好型快速译码。

阶数s处的每个可快速译码节点v可以看作是长度$M = 2^s$的外码。作为外码的节点v的编码比特用X_v表示，具有M个比特。

3.1 SPC-2

对于双SPC节点v，我们将其代码比特X_v分为两组，X_v^{even}的索引值均为偶数，X_v^{odd}的索引值均为奇数。根据SPC-2节点的定义，子树V_v上有两个奇偶校验比特，对应的奇偶校验函数$p[0]$和$p[1]$可以写为：

$$\begin{cases} p[0] : \oplus x = 0, x \in X_v \\ p[1] : \oplus x = 0, x \in X_v^{odd} \end{cases}$$

将两个奇偶校验函数相加，获得奇偶校验函数$p[2]$：

$$p[2] = p[0] \oplus p[1]: \oplus x = 0, x \in X_v^{even}$$

由于$p[1]$和$p[2]$两个奇偶校验函数涉及的编码比特没有交集，SPC-2节点的译码可以两个SPC节点并行，每个SPC节点可以继承X_v各一半元素。通过复用两个SPC译码模块，可对SPC-2节点快速译码。

3.2 REP-2

对于双REP节点v，我们将其编码比特X_v分为两组，X_v^{even}的索引值均为偶数，X_v^{odd}的索引值均为奇数。根据REP-2节点的定义，子树V_v上有两个信息比特，分别用u_{M-2}和u_{M-1}表示。

可以很容易得证，X_v^{odd}是u_{M-1}的重复，而X_v^{even}是$u_{M-2} \oplus u_{M-1}$的重复。因此，我们可以将长度为M的双REP节点划分为两个具有$M/2$个比特的REP节点，并且我们可以重用两个REP译码模块来快速对REP-2节点进行并行译码。

3.3 RPC

对于RPC节点v，我们将其编码比特X_v分为四组：

$$X_v^i = \{x \in X_v, \mathrm{mod}(l(x),4) = i\},\ i \in \{0,1,2,3\} \quad (1)$$

根据RPC节点的定义，子树V_v上有三个奇偶校验比特，奇偶校验函数$p[0]$、$p[1]$、$p[2]$可以写为：

$$\begin{cases} p[0] : \oplus x = 0, x \in X_v^0 \cup X_v^1 \cup X_v^2 \cup X_v^3 \\ p[1] : \oplus x = 0, x \in X_v^1 \cup X_v^3 \\ p[2] : \oplus x = 0, x \in X_v^2 \cup X_v^3 \end{cases}$$

将后两个函数相加得到奇偶校验函数$p[3]$：

$$p[3] = p[1] \oplus p[2]: \ \oplus x = 0, x \in X_v^1 \cup X_v^2$$

然后将此奇偶校验函数与第一个函数相加，得到奇偶校验函数$p[4]$：

$$p[4] = p[0] \oplus p[3]: \oplus x = 0, x \in X_v^0 \cup X_v^3$$

定义$\hat{c}_i = \oplus x,\ x \in X_v^i,\ i \in [0,1,2,3]$。通过奇偶校验函数$p[1]$至$p[4]$，可以很容易得证以下关系成立：

$$\hat{c}_1 \oplus \hat{c}_3 = \hat{c}_2 \oplus \hat{c}_3 = \hat{c}_1 \oplus \hat{c}_2 = \hat{c}_0 \oplus \hat{c}_3 = 0 \quad (2)$$

从等式（2）可以看出存在一个码率为$\frac{1}{4}$的虚拟重复码，因为：

$$\hat{c}_0 = \hat{c}_1 = \hat{c}_2 = \hat{c}_3 = 0$$

或

$$\hat{c}_0 = \hat{c}_1 = \hat{c}_2 = \hat{c}_3 = 1$$

其中，$\hat{c}_0$，$\hat{c}_1$，$\hat{c}_2$，$\hat{c}_3$是虚拟重复码比特。

综上，当阶数$s \leqslant 2$时，可以很容易地导出RPC节点的译码算法2，其中

$$\mathrm{sig}(\alpha) \triangleq \begin{cases} 0, & \alpha \geq 0 \\ 1, & \alpha < 0。 \end{cases}$$

3.4 PCR

对于PCR节点v，我们将其编码比特X_v像（1）一样分为四组：根据PCR节点的定义，该节点中有三个信息比特，分别用u_{M-3}、u_{M-2}、u_{M-1}表示。

根据下面的等式，我们定义c_i，其中$i \in \{0, 1, 2, 3\}$

$$[c_0\ c_1\ c_2\ c_3] = [0\ u_{M-3}\ u_{M-2}\ u_{M-1}] \times G_4 \quad (3)$$

可以很容易得证，X_v^0是c_0的重复，X_v^1是c_1的重复，X_v^2是c_2的重复，X_v^3是c_3的重复。因此，我们根据索引值将输入信号α_v分为四组，并与REP节点一样将各组内的输入信号组合为四个增强信号Δ_i，其中$i \in \{0, 1, 2, 3\}$。

算法 2 重复奇偶校验（RPC）节点译码算法

输入：

接收到的信号 $\alpha_v = \{\alpha_{v_k}, k = 0 \cdots M-1\}$;

输出：

待恢复的码字：$\hat{x} = \{\hat{x}_k, k = 0 \cdots M-1\}$;

初始化：$\Delta_0 = 0;\ \Delta_1 = 0$

初始化：$\delta_i = \infty,\ c_i = 0,\ p_i = 0$ for $i = 0 \cdots 3$;

初始化：$\hat{x}_k = sig(\alpha_{v_k})$ for $k = 0 \cdots M-1$};

for $i = 0 \cdots 3$ **do**

  for $j = 0 \cdots M/4$ **do**

    $k = j \times 4 + i$;

    $c_i = c_i \oplus sig(\alpha_{v_k})$;

    if $|\alpha_{v_k}| < \delta_i$

      $p_i = k$;

      $\delta_i = |\alpha_{v_k}|$;

  end for

  if $c_i = 1$

    $\Delta_0 = \Delta_0 + \delta_i$

  else

    $\Delta_1 = \Delta_1 + \delta_i$

end for

for $i = 0 \cdots 3$ **do**

  if $((\Delta_0 > \Delta_1) \cap (c_i = 0)) \cup ((\Delta_0 < \Delta_1) \cap (c_i = 1))$

    $\hat{x}_{p_i} = \sim \hat{x}_{p_i}$

end for

从等式（3）可以看出存在码率为$\frac{3}{4}$的虚拟单向奇偶校验比特c_i（$i \in \{0, 1, 2, 3\}$），这样我们就可以重用SPC模块进行译码。PCR译码算法详见算法3。

算法 3 奇偶校验重复（PCR）节点译码算法

输入：

接收到的信号 $\alpha_v = \{\alpha_{v_k}, k = 0 \cdots N-1\}$;

输出：

待恢复的码字：$\hat{x} = \{\hat{x}_k, k = 0 \cdots N-1\}$;

初始化：$\Delta_i = 0$ for $i = 0 \cdots 3$;

for $i = 0 \cdots 3$ **do**

  for $j = 0 \cdots N/4$ **do**

    $k = j \times 4 + i$

    $\Delta_i = \Delta_i + \alpha_{v_k}$

  end for

end for

$\{\hat{c}_0, \hat{c}_1, \hat{c}_2, \hat{c}_3\} = \mathrm{SPC\_DEC}(\{\Delta_0, \Delta_1, \Delta_2, \Delta_3\})$

for $i = 0 \cdots 3$ **do**

  for $j = 0 \cdots N/4$ **do**

    $k = j \times 4 + i$

    $\hat{x}_k = c_i$

  end for

end for

4 硬件实现

我们设计了两种硬件架构来验证性能、面积效率和能效。

- **递归型译码器：**此架构可灵活支持母码长N为2的幂（32到1024）的各种码长和码率，通过匹配码率，可支持$0 < N \leqslant 1024$的码长和$0 < R \leqslant 1$的码率。节点中的$f_{+/-}$函数由单个处理元（Processing Element，PE）逻辑处理，一个决策模块支持所有9种码型[1]。译码器一次处理一个数据包。
- **展开型译码器：**此架构仅支持固定码长和码率。在此架构支持的码长N固定为1024，码率R固定为0.875。这种完全展开的管道型设计包含了处理二叉树上各个$f_{+/-}$函数的专用PE。与判决模块相同，有21个专用节点的特定逻辑，用于支持21种节点码型。由于25个数据包同时译码，且充分利用了处理逻辑和存储资源，该译码器实现了超高吞吐率、高面积效率，以及低译码功耗。

上述两种译码器采用的串行抵消算法都由基于码型的快速译码加速。SPC和SPC−2节点的最大并行度为128，R1节点的最大并行度为256，其他所有节点的并行度为16。

4.1 并行比较电路

通过观察发现，二叉树右侧存在一些较大的SPC节点。如前所述，处理SPC节点需要较高的并行度才可以实现高吞吐。SPC译码算法非常简单，具体如下：首先，从SPC节点输入信号找到最小幅度并记录其位置。接下来，对输入信号的符号执行奇偶校验。如果校验通过，则将符号返回，否则，将记录到的最小幅度位置的符号反转，并将反转后的符号返回。

为了处理较大的SPC节点，需要一个从大量输入信号中定位最小幅度的电路。传统的成对比较方法需要一个深度为$log_2(M)$的电路，其中M是要比较的幅度的数量。例如，需要进行7次比较才能在128个幅度中找到最小的一个。考虑到时钟频率设置为1 GHz，满足时序约束并在一个时钟周期内完成所有比较难度较大。

为了解决这些问题，我们主张用并行比较架构来取代传统的比较架构。对于阶数s的节点v，其输入信号α_v包含$M=2^s$个元素，其幅度表示为$[A_0\ A_1 \cdots A_{M-1}]$。每个幅度都有$x$比特量化，我们将$x$比特量化的二进制向量填充到矩阵的列中，如右上图所示：

如果我们用行向量矩阵重写该矩阵，可以得到$[B_0 \cdots B_j \cdots B_{M-1}]^T$，其中$B_j = [b_0^j \cdots b_i^j \cdots b_{M-1}^j]$是行向量（$j \in \{0, 1 \cdots x-1\}$）。

$$[A_0 \ \cdots \ A_i \ \cdots \ A_{M-1}] = \begin{bmatrix} b_0^0 & \cdots & b_i^0 & \cdots & b_{M-1}^0 \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ b_0^j & \cdots & b_i^j & \cdots & b_{M-1}^j \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ b_0^{x-1} & \cdots & b_i^{x-1} & \cdots & b_{M-1}^{x-1} \end{bmatrix}$$

我们提出了算法4通过反向掩码D确定最小幅度的位置，其中比特“1”表示最小幅度位置。

算法 4 并行比较算法

输入：

接收到的信号$\alpha_v = \{\alpha_{v_k}, k = 0 \cdots M-1\}$;

输出：

反向掩码：D是M比特变量;

初始化：来自α_v的$[B_0 \cdots B_j \cdots B_{x-1}]^T$;

初始化：N比特变量$C = \mathbf{0}$。

for $j = x-1 \cdots 0$ **do**

  M比特变量$E = (C|B_j)$

  if（E中并非所有比特都为“1”）

    $C = E$

end for

反向掩码 $D = \sim C$

上述并行比较算法将比较逻辑的深度从$log_2(M)$降为1。然而，反向掩码D可能有两个或多个最小位置，这意味着输入信号α_v包括两个或更多个最小幅度，在这种情况下就肯定会出现误差。为此，我们另外增加了一个去重电路，用于确保所选最小位置的唯一性。

4.2 比特量化

极化码的一个显著优势在于，即便量化精度低（4~6比特），SC译码算法也不会受影响。低精度量化是提高吞吐率的关键，因为降低精度可以有效减少实现面积，同时可以提高时钟频率。

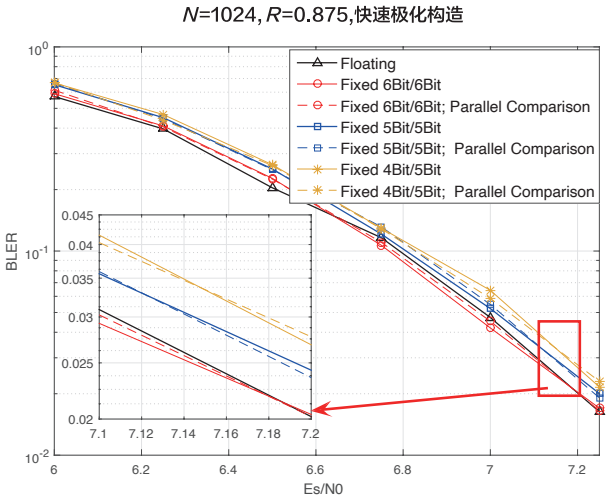

图 4 浮点与定点性能对比

[1] SC译码绕过了R0节点。

量化比特的数值分为两种：一种用于信道LLR，另一种用于内部LLR。我们首先测试6比特输入量化和6比特内部量化的情况。如图4所示，这一配置的性能与浮点相当。第二种情况是5比特量化和5比特内部量化，损耗小于0.1 dB。最后是4比特输入量化和5比特内部量化，损耗小于0.2 dB。本文我们评估了输入信号和内部信号在5比特量化条件下的物理实现结果，以寻找复杂度和吞吐率之间的平衡。

与此同时，我们还比较了原始SPC和并行SPC的BLER性能。所有量化方案产生的损耗，都在可接受范围。

4.3 版图面积估算

我们用FPGA方法实现了递归型和展开型两种架构，并将结果换算为物理实现。

根据FPGA综合结果，递归型译码器有10170个LUT和12772个FF，而展开型译码器有66192个LUT和55187个FF。两种译码器均未使用存储器，因此FPGA结果可以较为容易地换算为ASIC。按16 nm工艺换算，时钟频率1 GHz时，递归型译码器的综合面积和芯片版图面积分别为0.032 mm^2和0.045 mm^2。按16 nm工艺换算，时钟频率1.20 GHz时，递归型译码器的综合面积和芯片版图面积分别为0.17 mm^2和0.30 mm^2。

5 关键性能指标

本节将展示相关关键性能指标（Key Performance Indicator，KPI）。首先，我们用以下等式来评估面积效率：

$$\text{面积效率（Gbps/mm}^2\text{）} = \frac{\text{信息量（bit）}}{\text{时延（ns）} \times \text{面积（mm}^2\text{）}}$$

递归型译码器对一个码长N = 1024、码率R = 0.875的快速极化码构造的数据包进行译码需要40个时钟周期。因此，编码比特的吞吐率为(1024 bit × 1 GHz)/40个周期 = 25.6 Gbps，信息比特的吞吐率为((1024 × 0.875) bit × 1 GHz)/40个周期 = 22.4 Gbps。以16 nm工艺换算，编码比特的面积效率为561 Gbps/mm^2。

展开型译码器完成一个数据包译码需要25个时钟周期，其译码过程完全管道化，也就是说，第一个数据包的25个时钟周期之后的每一个时钟周期，都会产生一个新的译码结果数据包。因此，编码比特的吞吐率为1024 bit × 1.2 GHz = 1229 Gbps，信息比特的吞吐率为(1024 × 0.875) bit × 1 GHz = 1075 Gbps。以16 nm工艺换算，编码比特的面积效率为4096 Gbps/mm^2。

我们用200个数据包进行仿真分析，进一步评估能耗和每比特的译码能效。仿真的PVT（工艺、电压、温度）条件分别为TT、0.8 V和20℃。结果显示，递归型译码器的功率为30.9 mW，平均能效为1.21 pJ/bit；展开型译码器的功率为784 mW，平均能效为0.63 pJ/bit。

此外，我们还与其他一些高吞吐率译码器进行了面积效率、能耗比较，相关结果见表3。从KPI数据来看，展开型译码器更适合有超高吞吐率要求的场景，但只支持固定的码长和码率；递归型译码器尺寸更小，更适合资源受限的设备，且对码长和码率的支持也更灵活，这点对于无线通信很有吸引力。

6 结语

本文提出了一种新的极化码构造方式。该方法构造的极化码全部由可快速译码的特殊节点组成，码长为16。如果将译码过程视为二叉树遍历，在码长N = 1024、码率R = 0.875的条件下，与原始极化码构造方法相比，快速极化码只需牺牲轻微的BLER性能，就可以减少55%的节点访问、8.9%的$f_{+/-}$计算和43.5%的边遍历。

同时，我们为快速极化码实现了两种译码器。递归型译码器对码长和码率的支持灵活，所支持的码长最高可达1024。该译码器的估算芯片版图面积仅为0.045 mm^2，可以提供25.6 Gbps的编码比特吞吐率，面积效率为561 Gbps/mm^2。

展开型译码器只支持码长N = 1024、码率R = 0.875的码。得益于完全管道化的结构，展开型译码器硬件可实现超高的面积效率以及较低的译码能耗。展开型译码器的估算芯片版图面积为0.3 mm^2，可提供1229 Gbps的编码比特吞吐率，面积效率高达4096 Gbps/mm^2。

上述结果表明，快速极化码可以满足下一代无线通信系统高吞吐率的需求，递归型和展开型硬件设计可以满足不同的系统需求。

表 2 与高吞吐率极化码译码器的比较

方案	本研究（展开型）	本研究（递归型）	[1]	[2]	[8]
构造	快速极化码	快速极化码	极化码	乘积码[15]	极化码
译码算法	快速 SC	快速 SC	SC	PDF-SC [16]	OPSC
码长	1024	1024	32768	16384	1024
码率	0.875	0.875	0.864	0.864	0.83
工艺	FPGA 转换为 16 nm		台积电 16 nm		
时钟频率（GHz）	1.20	1.00	1.00	1.05	1.20
编码比特吞吐率（Gbps）	1229	25.6	5.27	139.7	1229
信息比特吞吐率（Gbps）	1075	22.4	4.56	120.73	1020
芯片版图面积（mm^2）	0.30	0.045	0.35	1.00	0.79
编码比特面积效率（$Gbps/mm^2$）	4096	561	15.1	139.7	1555
功率（mW）	784	30.9	–	94	1167
能效（pJ/bit）	0.63	1.21	–	0.67	0.95

表 3 与高吞吐率极化译码器的比较

方案	本研究（展开型）	本研究（递归型）	极化码 -5G	LDPC-1K-5G	LDPC-1K-Unrolled	[8]
构造	快速极化码	快速极化码	极化码（5G）	LDPC（5G）	LDPC（展开型）	极化码
译码算法	快速 SC	快速 SC	SC	LOMS-3	LOMS-3	OPSC
码长	1024	1024	1024	1024	1024	1024
码率	0.875	0.875	0.875	0.875	0.875	0.83
工艺	FPGA 转换为 16 nm		台积电 16 nm			
时钟频率（GHz）	1.20	1.00	1.00	1.00	1.00	1.20
编码比特吞吐率（Gbps）	1229	25.6	10.8	12.44	1024	1229
信息比特吞吐率（Gbps）	1075	22.4	9.45	10.89	896	1020
芯片版图面积（mm^2）	0.30	0.045	0.069	0.154	1.02	0.79
编码比特面积效率（$Gbps/mm^2$）	4096	561	157	81	878	1555

参考文献

[1] X. Liu, Q. Zhang, P. Qiu, J. Tong, H. Zhang, C. Zhao, J. Wang, "A 5.16Gbps decoder ASIC for polar code in 16nm FinFET," 2018 *15th International Symposium on Wireless Communication Systems (ISWCS)*, Lisbon, 2018, pp. 1-5.

[2] J. Tong, X. Wang, Q. Zhang, H. Zhang, S. Dai, R. Li, and J. Wang, "Toward terabits-per-second communications: a high-throughput implementation of GN-Coset codes," *IEEE*

Wireless Communications and Networking Conference (WCNC), 2021, pp. 1–6.

[3] W. Saad, M. Bennis, and M. Chen, "A vision of 6G wireless systems: applications, trends, technologies, and open research problems," *IEEE Network*, 2019.

[4] E. Arikan, "Channel polarization: a method for constructing capacity achieving codes for symmetric binary-input memoryless channels," *IEEE Transactions on Information Theory*, vol. 55, no. 7, pp. 3051–3073, Jul. 2009.

[5] A. Alamdar-Yazdi and F. Kschischang, "A simplified successive cancellation decoder for polar codes," in *IEEE Communications Letters*, vol. 15, no. 12, pp. 1378–1380, December 2011.

[6] O. Dizdar and E. Arikan, "A high-throughput energy-efficient implementation of successive cancellation decoder for polar codes using combinational logic," in *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 3, pp. 436–447, March 2016.

[7] A. Sral, E. G. Sezer, Y. Ertugrul, O. Arikan, and E. Arikan, "Terabitsper-second throughput for polar codes," *2019 IEEE 30th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC Workshops)*, 2019, pp. 1–7.

[8] A. Sral, E. G. Sezer, E. Kolagasıoglu, V. Derudder, and K. Bertrand, "Tb/s polar successive cancellation decoder 16nm asic implementation," Available on http://www.polaran.com/documents/EPIC Polar Code Paper.pdf.

[9] S. A. Hashemi, C. Condo, and W. J. Gross, "Fast and flexible successive cancellation list decoders for polar codes," *IEEE Transactions on Signal Processing*, vol. 65, no. 21, pp. 5756–5769, Nov 2017.

[10] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, and W. J. Gross, "Fast polar decoders: algorithm and implementation," in *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 5, pp. 946–957, May 2014.

[11] Y. Wang and K. Narayanan, "Concatenations of polar codes with outer BCH codes and convolutional codes," *Communication Control and Computing (Allerton) 2014 52nd Annual Allerton Conference on*, pp. 813–819, 2014.

[12] H. Saber and I. Marsland, "Design of generalized concatenated codes based on polar codes with very short outer codes," *Vehicular Technology IEEE Transactions on*, vol. 66, no. 4, pp. 3103–3115, 2017.

[13] D. Goldin and D. Burshtein, "Performance bounds of concatenated polar coding schemes," *Information Theory IEEE Transactions on*, vol. 65, no. 11, pp. 7131–7148, 2019.

[14] A. Balatsoukas-Stimming, M. B. Parizi, and A. Burg, "LLR-based successive cancellation list decoding of polar codes," in *IEEE Transactions on Signal Processing*, vol. 63, no. 19, pp. 5165–5179, Oct. 2015.

[15] X. Wang, H. Zhang, R. Li, J. Tong, Y. Ge, and J. Wang, "On the construction of GN-coset codes for parallel decoding," *IEEE Wireless Communications and Networking Conference (WCNC)*, Seoul, Korea (South), 2020, pp. 1–6.

[16] X. Wang, J. Tong, H. Zhang, S. Dai, R. Li, and J. Wang, "Toward terabits-per-second communications: low-complexity parallel decoding of GN-coset codes," *IEEE Wireless Communications and Networking Conference (WCNC)*, 2021, pp. 1–5.