面向多核极化码的高吞吐率灵活组合译码器

Hossein Rezaei，*IEEE* 学生会员，Nandana Rajatheva，*IEEE* 高级会员，以及 Matti Latva-aho，*IEEE* 高级会员

arXiv:2301.10445v1 [cs.IT] 25 Jan 2023

摘要—**过去十年间，极化码受到越来越多的关注，并已被选为第五代（5G）无线通信系统控制信道的编码方案。然而，传统极化码仅采用二元（2×2）核构造，因此码块长度受限于2的幂。为实现更灵活的码块长度，研究者提出了多核极化码。本文基于连续消除译码算法，提出了一种适用于多核极化码的高吞吐量组合架构。所提方案能够译码纯二元、纯三元（3×3）以及二元-三元混合极化码。该译码器架构具有速率灵活性，即每个时钟周期都可以为译码器指定新的码率。通过现场可编程门阵列实现对所提架构进行了验证，结果表明，码长为 $N=81$ 的编码可实现 1664.5 兆比特每秒的编码吞吐量。本文还提出了一种新颖的基于 Python 的极化码编译器，可自动为目标译码器生成硬件描述语言模块。设计人员输入目标极化码的码块长度和核排序后，即可自动获取所需的 VHDL 文件。仿真结果表明，大多数所需的硬件描述语言文件可在0.4 秒内生成。**

索引词—**极化码、连续消除译码器、多核、纠错码、硬件实现、极化码编译器。**

I. 引言

极化码由 Arikan [1] 提出，是一类纠错码。当码长趋于无穷时，这类码能够达到二进制输入离散无记忆信道的对称信道容量。通过递归构造，可利用二元矩阵$T_2=\left[\begin{smallmatrix}1&1\\1&0\end{smallmatrix}\right]$的 n次克罗内克幂构造大小为 $N=2^n$ 的极化码，该矩阵也称为 Arikan核。这种构造将物理信道转换为 N 个虚拟合成信道；随着码长增长，这些信道的可靠性趋近于零或无穷大。

Arikan [1]还证明，极化码可以通过连续消除（SC）译码算法达到对称信道容量。此后，研究人员一直致力于改进有限长度极化码在 SC 译码下的延迟、复杂度、功耗和纠错性能。将循环冗余校验（CRC） [3] 与连续消除列表（SCL） [2]译码级联，可提升极化码的纠错性能，使其能够与低密度奇偶校验（LDPC）码等其他信道编码方法相竞争。这些努力为极化码适用于 3GPP 第五代新空口（5G-NR）无线通信标准 [4]奠定了基础。

然而，目前大多数研究都聚焦于采用 Arikan核 [1]构造的极化码。使用 2×2极化矩阵会将极化码的码长限制为 2 的幂，因此无法满足超越5G网络中各种码长和速率的需求。为覆盖非二进制码长，研究人员提出了打孔和缩短等速率匹配方案 [5], [6] 。然而，打孔码和缩短码的先验性能及最优性难以评估。由 2 的克罗内克积实现的极化现象也可以扩展到其他核。尤其是三元核（3×3），因其复杂度低且具有极化最优性而日益受到关注。利用这种方法，多核（MK）极化码 [7]–[9] 通过采用维度不同的核，实现了灵活的码长。其计算复杂度与 Arikan 极化码相同，并且在纠错性能和复杂度方面优于类似的打孔码和缩短码 [8], [10]。

在硬件实现方面，已有多种译码器被提出，用于译码 Arikan 极化码。文献 [11] 将 Arikan 极化码 [12] 的架构应用于由 Arikan核和三元核构造的 MK码。文献 [10] 提出了一种可降低延迟的 MK 极化码译码架构。然而，文献 [10] 和 [11] 中提到的两种 MK译码器，在二元和三元级进行存储器读写时采用了复杂的存储器接口，导致编码吞吐量降低。

本文旨在开发一种灵活的架构，基于 SC 算法实现低功耗、高吞吐量的 MK 极化译码器。我们采用纯组合架构来实现这一目标。SC算法的递归和前馈结构便于将纯组合逻辑应用于极化译码器。基于 SC 的组合译码器具有完全可扩展性，与其时序对应架构相比，工作频率显著更低。不过，它们能够在一个较长的时钟周期内解码整个码字，从而大幅降低动态功耗。所提出架构的一个关键特性是针对给定码块长度进行在线速率分配，这意味着每个时钟周期都可以为译码器分配新的码率。

在 [13]中，我们提出了首个 MK 组合译码器。然而，它有两个局限。首先，它不支持纯三元极化码。其次，它未提供所需的

0000–0000/00$00.00 © 2023 IEEE

存储器，用于载入下一帧的对数似然比（LLR）和冻结位指示符，并输出估计的码字。缺少这些存储器会导致吞吐量受限，因为上一帧尚未解码完成时，输入数据尚未就绪。本研究在详细分析性能与复杂度的同时，也解决了这些局限。所提出的架构支持 55 种不同的码块长度，最大码块长度为 $N_{max} = 4096$ ，可由纯二元核、纯三元核及二元-三元混合核构成。为验证该架构，我们实现了现场可编程门阵列（FPGA），并将其与当前先进的 MK译码器进行了比较。

最后，我们提出一种基于 Python 的硬件编译器，用于自动生成所提出译码器在现场可编程门阵列上的实现所需的 VHDL 文件。其动机在于，改变码块长度或核排序都会导致整个 VHDL 模块发生变化。使用所提出的极化码编译器，用户只需输入码块长度和核排序，编译器便会自动输出所有必要的 VHDL 文件。如果用户未输入核顺序，编译器则会自动分配纠错性能最高的核排序。

本文其余部分组织如下：第二节介绍 Arikan 极化码和 MK 极化码的背景。第三节阐述码构造方法、所提出的 MK 译码器架构以及复杂度分析。第四节讨论如何使用所提出的极化码编译器，为目标译码器自动生成 VHDL 文件。第五节详细介绍实现结果及与先前工作的比较。最后，第六节给出结论。

II. 预备知识

本节在介绍极化码的背景知识之外，还将介绍 Arikan 极化码和 MK 极化码的码构造方法。随后将概述 Arikan 极化码和 MK 码的连续消除译码算法。

A. Arikan 极化码

$\mathcal{PC}(N,\ K)$ 表示一个大小为 N=2^n 、含有 K 个信息位的极化码，其码率可计算为$\mathcal{R} = K/N$。Arikan 在 [1] 中证明了二元极化码的信道极化现象，该现象可用于将物理信道 W 转换为 N 个独立的虚拟信道 W_i^N （$1 \leq i \leq N$）。划分后的虚拟信道具有相对更高或更低的可靠性；当$N \to \infty$时，每个信道的可靠性趋近于 1（完全可靠）或 0（完全不可靠）。可使用 Bhattacharya 参数 [1] 或高斯近似 [14] 来确定各个可靠信道。最可靠的 K 个位位置组成的集合称为信息集，以 $\mathcal{I}$ 表示；其余 N-K 个位位置称为冻结集，以 $\mathcal{F}$ 表示。冻结集位置上的位值设为零。

线性变换可用于构造极化码。它表示为 $x = uG$其中 x 表示编码后的比特流， u 表示输入编码器的 N 位输入向量，G 为生成矩阵。输入向量 u 的构造方式是分别将信息数据和冻结数据插入可靠位置和不可靠位置。生成矩阵 $G = T_2^{\otimes n}$由 Arıkan核 $T_2 = \left[\begin{smallmatrix}1 & 1\\ 1 & 0\end{smallmatrix}\right]$的 n 阶克罗内克积构造而成。从定义中可以看出， G是以递归方式构造的。因此，可通过级联两个大小为$N/2$的编码来构造一个大小为 N 的极化码。

B. MK 极化码

将 T_2 作为生成矩阵的唯一核，可知较大编码的码块长度将受限于 2 的幂。然而，以 LDPC WiMAX [15] 码长为参考，我们可以发现，还需要由 T_2 以外的核构造的码长。不过，仅使用一个或少数几个非二元核，就能提供大部分所需的码长。通过对各种核进行一系列克罗内克积，可以构造出生成矩阵：

$$G \triangleq T_{n_0} \otimes T_{n_1} \otimes \ldots \otimes T_{n_s} \tag{1}$$

对于大小为 $N = n_0 \times n_1 \times \ldots \times n_s$ 的编码，其中 n_i （$0 \leq i \leq s$）不一定是各自的素数，且 T_{n_i} 为方阵。每个不同的素数都可以用作核的维数。不过，复杂度最低且最理想的非二元核——三元核——定义为 $T_3 = \left[\begin{smallmatrix}1 & 1 & 1\\ 1 & 0 & 1\\ 0 & 1 & 1\end{smallmatrix}\right]$ [8]。文献 [16], 证明了 T_3 的极化最优性，尽管其极化指数低于 T_2。本文研究由二元核和三元核任意组合构造的编码，即纯二元（Arikan）极化码、纯三元极化码以及二元-三元混合极化码。

本文中MK码的码块长度可表示为 $N = 2^n \cdot 3^m$ ，其生成矩阵为 $G = \otimes_{i=0}^{n+m} T_{n_i}$，其中 n，$m \in \mathbb{N}$ 且 $0 \leq i \leq n + m$。例如，我们考虑尺寸为 $N = 6$的最简单MK极化码。 $T_2 \otimes T_3$ 和$T_3 \otimes T_2$ 是两种可能的核序列，由于克罗内克积不满足交换律，因此会得到两个不同的生成矩阵。因此，不同的核排序会形成各具特色、性能特征不同的极化码。尺寸为 $N = 6$ 且 $G = T_2 \otimes T_3$ 的MK码的Tanner图如图1(a)所示。

C. Arikan 与 *MK* 逐次消除译码

SC译码算法最初由Arikan在 [1]中提出。该算法可以扩展用于译码MK极化码。图1(b)中的译码树对应于图1(a)中的Tanner图。软信息（α_c）以对数似然比的形式输入树的根节点。在先访问左节点的条件下，对数似然比需要沿树传递并依次访问所有叶节点，从而估计出一个码字。为此，给定节点 ν需要用到三个函数。第一个函数是 α_{v_l} ，它是传递至左分支的对数似然比集合，可按如下方式计算：

$$\alpha_{v_l}^b[i] = sgn(\alpha_v[i] \cdot \alpha_v[i+2^{(\lambda\text{-}1)}])min(|\alpha_v[i]|, |\alpha_v[i+2^{(\lambda\text{-}1)}]|) \tag{2}$$

其中 $i \in [0\colon 2^{(\lambda\text{-}1)} - 1]$， $\lambda \in [0, n]$ 是二叉树中节点 ν的层级。在估计出

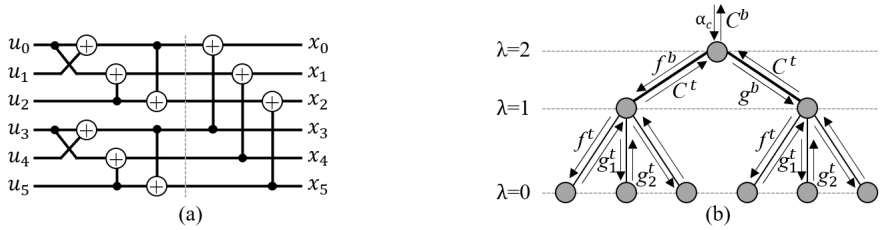


图1. a) 大小为 $N=6$、具有 $G=T_2\otimes T_3$ 的 MK 极化码的 Tanner图和 b) 译码树。

节点 ν ($\beta_{v_l}^b$)处左分支的硬判决后，我们可以使用以下方法计算要发送到右分支的对数似然比集合

$$\alpha_{v_r}^b[i]=(1-2\beta_{v_l}^b[i])\alpha_v[2i]+\alpha_v[2i+1] \text{ for } i\in[0:2^{(\lambda\text{-}1)}-1] \tag{3}$$

其中，$\alpha_{v_r}^b$ 是右分支的对数似然比。在估计左、右分支的硬判决后，我们可以将它们合并，计算对应于节点 ν 的硬判决:

$$[\beta_i^{\nu b},\beta_{i+2^{(\lambda\text{-}1)}}^{\nu b}]=[\beta_i^{\nu bl}\oplus\beta_i^{\nu br},\beta_i^{\nu br}], \tag{4}$$

其中，$\oplus$ 表示对 $\mathbb{F}_2$ 求和。若 ν 是叶节点，则可估计其硬判决为:

$$\beta_v=\begin{cases}h(\alpha_v), & \text{if } v\in\mathcal{I},\\ 0, & \text{if } v\in\mathcal{F}\end{cases},\ h(x)=\begin{cases}0, & \text{if } x\geq 0,\\ 1, & \text{otherwise.}\end{cases} \tag{5}$$

本文中，如图1(b)所示，我们分别将式 (2)、(3) 和 (4) 称为 f^b、g^b 和 C^b。此外，还需要定义两个通用函数：二进制符号函数 $s(x)$ 和冻结比特指示向量 a，其定义如下:

$$s(x)=\begin{cases}0, & \text{if } l\geq 0,\\ 1, & \text{otherwise}\end{cases},\ a_i=\begin{cases}0, & \text{if } i\in\mathcal{F},\\ 1, & \text{if } i\in\mathcal{I}.\end{cases} \tag{6}$$

二进制 SC 译码算法的精确伪代码见算法1。

算法 1 β^b = 使用二进制 SC 解码(α, a)

$N=lengh(\alpha)$
如果 $N==2$ **则**
  $\beta_0\leftarrow s(f^b(\alpha))\cdot a(0)$
  $\beta_1\leftarrow s(g^b(\alpha,\ \beta_0))\cdot a(1)$
  返回 $\beta\leftarrow(\beta_0,\ \beta_1)$
else
  $\alpha'\leftarrow f_{N/2}^b(\alpha)$
  $a'\leftarrow a(0\ to\ N/2\text{-}1)$
  $\beta'\leftarrow Decode(\alpha',\ a')$
  $v'\leftarrow C^b(\beta'(0\ to\ N/4\text{-}1)\ \beta'(N/4\ to\ N/2\text{-}1))$
  $\alpha''\leftarrow g_{N/2}^b(\alpha,\ v')$
  $a''\leftarrow a(N/2\ to\ N\text{-}1)$
  $\beta''\leftarrow Decode(\alpha''\ a'')$
  $v''\leftarrow C^b(\beta''(0\ to\ N/4\text{-}1)\ \beta''(N/4\ to\ N/2\text{-}1))$
  返回 $\beta^b\leftarrow C^b(v',\ v'')$
end

对于位于层级 λ 的纯三元节点 ν，需要定义四个函数以满足消息传递准则。左、中、右分支对应的译码函数分别表示为 $\alpha_{v_l}^t$、$\alpha_{v_c}^t$ 和 $\alpha_{v_r}^t$。对于 $i\in[0,3^{\lambda\text{-}1}-1]$，$\alpha_{v_l}^t$ 的计算方式为

$$\alpha_{v_l}^t[i]=\text{sgn}(\alpha_v[i]\cdot\alpha_v[i+3^{(\lambda\text{-}1)}]\cdot\alpha_v[i+2\times3^{(\lambda\text{-}1)}])\ \min(|\alpha_v[i]|,|\alpha_v[i+3^{(\lambda\text{-}1)}]|,|\alpha_v[i+2\times3^{(\lambda\text{-}1)}]|). \tag{7}$$

计算出左分支的硬判决($\beta_{v_l}^t$)后，对数似然比可以通过以下方式传递到中间分支

$$\alpha_{v_c}^t[i]=(1\text{-}2\beta_{v_l}^t[i])\alpha[i]+f^b(\alpha[i+3^{(\lambda\text{-}1)}],\alpha[i+2\times3^{(\lambda\text{-}1)}]). \tag{8}$$

最后，利用左分支和中间分支的硬判决，对数似然比向量可以通过以下方式传递到右分支

$$\alpha_{v_r}^t[i]=(1\text{-}2\beta_{v_l}^t[i])\alpha[i+3^{(\lambda\text{-}1)}]+(1\text{-}2\beta_{v_l}^t[i]\oplus\beta_{v_c}^t[i])\alpha[i+2\times3^{(\lambda\text{-}1)}]. \tag{9}$$

现在，可以通过以下方式在节点 ν 处合并硬判决

$$[\beta_i^{\nu t},\beta_{i+3^{(\lambda\text{-}1)}}^{\nu t},\beta_{i+2\times3^{(\lambda\text{-}1)}}^{\nu t}]=[\beta_i^{\nu t_l}\oplus\beta_i^{\nu t_c},\beta_i^{\nu t_l}\oplus\beta_i^{\nu t_r},\beta_i^{\nu t_l}\oplus\beta_i^{\nu t_c}\oplus\beta_i^{\nu t_r}]. \tag{10}$$

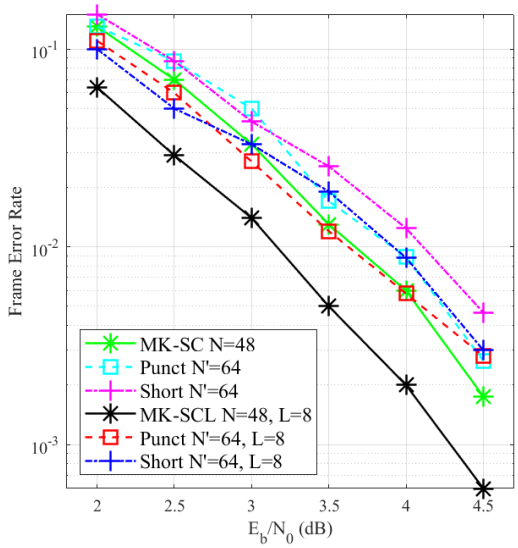
如图1 (b)所示，方程(7)、(8)、(9)和(10)分别称为 f^t、g_1^t、g_2^t 和 C^t。三元连续消除译码的准确描述见算法2。

III. MK码：构造与架构

A. MK码构造

在 [8] 中，提出了一种用于 MK 极化码的码构造方法。与打孔 [17]和缩短 [18] 方法相比，该方法显著提升了纠错性能。此方法的编码复杂度仍然较低，且译码时可采用与 Arikan 极化码相同的通用结构。图2展示了尺寸为 $N=72$ 和 $\mathcal{R}=1/2$、生成矩阵为 $G=T_3\otimes T_2\otimes T_2\otimes T_2\otimes T_3$ 的 MK 极化码的纠错性能。仿真采用加性高斯白噪声（AWGN）信道上的二进制相移键控（BPSK）调制。显然，MK码的性能优于采用母码 $N'=128$构造的打孔码和缩短码。

可以采用任意核排序来构造 MK极化码。然而，由于克罗内克积不满足交换律，不同的核排序会呈现出不同的纠错性能表现 [8][9]。目前还没有理论方法能够找到最佳核排序。因此，对于给定的编码



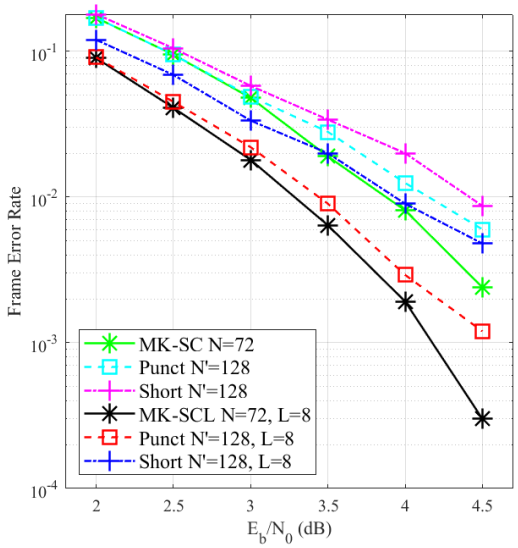


图2。 $\mathcal{PC}$(72，36)的MK码与打孔和缩短方法的纠错性能比较。

算法 2 $\beta^t = Decode(\alpha,\ a)$，使用三元SC

$N = lengh(\alpha)$
如果 $N == 3$ **则**
  $\beta_0 \leftarrow s(f^t(\alpha)) \cdot a(0)$
  $\beta_1 \leftarrow s(g_1^t(\alpha,\ \beta_0)) \cdot a(1)$
  $\beta_2 \leftarrow s(g_2^t(\alpha,\ \beta_0,\ \beta_1)) \cdot a(2)$
  返回 $\beta \leftarrow (\beta_0, \beta_1, \beta_2)$
else
  $\alpha' \leftarrow f_{N/3}^t(\alpha)$
  $a' \leftarrow a(0\ to\ N/3\text{-}1)$
  $\beta' \leftarrow Decode(\alpha',\ a')$
  $v' \leftarrow C^t(\beta'(0\ to\ N/3\text{-}1)\ \beta'(N/3\ to\ 2N/3\text{-}1)\ \beta'(2N/3\ to\ N\text{-}1))$

  $\alpha'' \leftarrow g_{1N/3}^t(\alpha,\ v')$
  $a'' \leftarrow a(N/3\ to\ 2N/3\text{-}1)$
  $\beta'' \leftarrow Decode(\alpha''\ a'')$
  $v'' \leftarrow C^t(\beta''(0\ to\ N/3\text{-}1)\ \beta''(N/3\ to\ 2N/3\text{-}1)\ \beta''(2N/3\ to\ N\text{-}1))$

  $\alpha''' \leftarrow g_{2N/3}^t(\alpha, v', v'')$
  $a''' \leftarrow a(2N/3\ to\ N\text{-}1)$
  $\beta''' \leftarrow Decode(\alpha'''\ a''')$
  $v''' \leftarrow C^t(\beta'''(0\ to\ N/3\text{-}1)\ \beta'''(N/3\ to\ 2N/3\text{-}1)\ \beta'''(2N/3\ to\ N\text{-}1))$
  返回 $\beta^t \leftarrow C^t(v',\ v''\ v''')$
end

我们需要通过仿真找到纠错性能最佳的核顺序。本文采用 [9] 中提出的方法来寻找核排序，因为如图 3 所示，该方法在纠错性能方面优于 [8]。如前所述，我们以 LDPC WiMAX 码块长度 [15] 为指导。因此，我们主要关注该标准所需的编码。

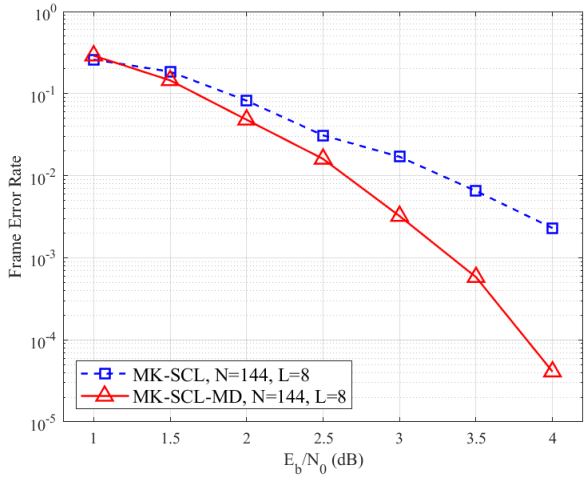


图 3. 采用 [8] (MK-SCL) 和 [9] (MK-SCL-MD) 方法构造的大小为 $\mathcal{PC}$(144, 72) 的 MK码的纠错性能。

在复杂度方面，MK码相较于打孔和缩短方法具有显著更低的复杂度。这是因为，与采用大小为$N' = 2^{\lceil \log_2 N \rceil}$的母码的打孔和缩短方法相比，MK极化码的Tanner图更小。母码决定了打孔码和缩短码的复杂度。可以将每种方法所需的对数似然比计算总次数定义为复杂度度量。我们假设码构造中使用的核数为 s，这等同于该码Tanner图中的阶段数。利用 $N \times s$ 和$N'\log_2 N'$，我们可以分别计算MK码和打孔/缩短码的复杂度度量。图4展示了在多种码块长度下，MK码相对于打孔码和缩短码的复杂度增益。显然，

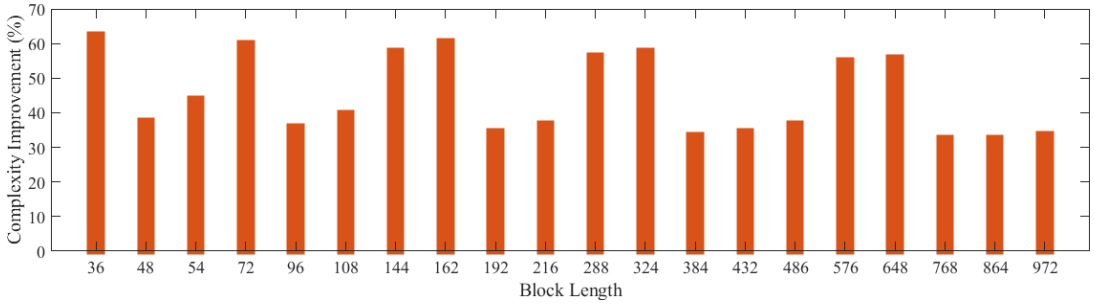


图 4. 使用所定义的复杂度度量，MK码相对于打孔码和缩短码的复杂度增益。

与打孔码和缩短码相比，MK码的对数似然比计算复杂度更低，降幅为32.5%至62.5%。

B. 所提出的译码器架构

鉴于 SC 算法不包含循环，我们可以设计一种纯组合架构，在输入级和输出级之间不包含存储元件。首要目标是实现高吞吐量。本节主要介绍置信传播函数的实现方法。随后，将介绍针对 Arikan 极化码、纯三元极化码以及二元-三元混合极化码所提出的组合架构。

1) 置信传播函数：为设计组合函数，与 [19], [20] 类似，并为避免不同表示形式之间的转换，我们使用 Q 位，以符号-数值格式表示信道观测的对数似然比。直接采用公式 (2)，即可利用一个比较器和一个多路复用器实现 f^b 函数。预计算前瞻方法可用于任意深度，以降低 SC 算法 [21] 的延迟，代价是增加硬件复杂度。采用该技术，所有可能的输出候选项均可在一个时钟周期内预计算，然后再选出正确的候选项。本文所有 Arıkan 码均采用大小为 $N = 4$ 的极化码作为基本构建块。核中的 g_2^b 函数通过预计算方法实现。图 5 展示了所提出的预计算电路逻辑，该电路用于实现大小为 $N = 4$ 的二元基本构建块。此处未展示冻结比特指示器（a）。图 6 展示了针对大小为 $N = 2$ 的极化码所提出的二元判决逻辑电路，该电路仅使用比较器、多路复用器和逻辑门实现。为进一步降低 Arikan 极化码的延迟和复杂度，图 6 中的判决逻辑以 $2i$ 和 $2i$+1 （$0 \le i < N/2$）的形式估计索引，

$$\beta_{2i} = (s(\alpha_{2i}) \oplus s(\alpha_{2i+1})) \cdot a_{2i}, \tag{11}$$

$$\beta_{2i+1} = \begin{cases} s(\alpha_{2i+1}) \cdot a_{2i+1} & \text{if } |\alpha_{2i+1}| \ge |\alpha_{2i}| \\ (s(\alpha_{2i}) \oplus \beta_{2i}) \cdot a_{2i+1} & \text{otherwise,} \end{cases} \tag{12}$$

所提出的二元极化码SC译码伪代码总结于算法3中。主要有两个

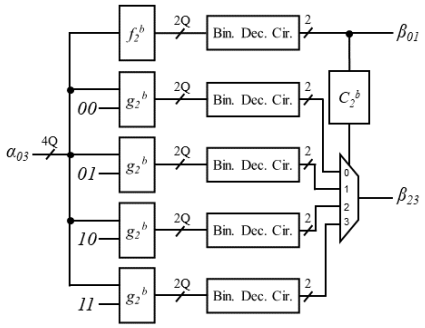


图 5. 尺寸为 $N = 4$ 的二进制基本构建块所采用的二进制预计算电路。

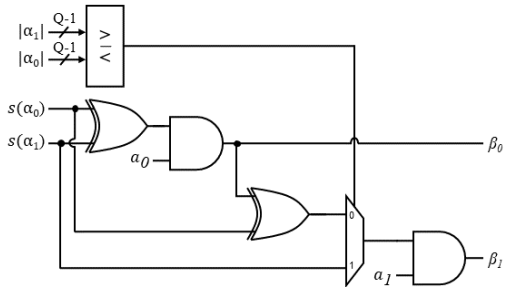


图 6. 与尺寸为 $N = 2$ 的极化码等效的二进制判决逻辑。

方面使所提出的算法与 [19]有所不同。首先是判决逻辑的实现方式。其次， [19] 的架构在每个阶段都利用一个大小为 $N/2$的编码器作为粘合逻辑的一部分，而我们提出的架构则采用 C^b 。这使得异或门的消耗大幅降低。由于 C^b 仅由一层异或门构成，其延迟远低于大小为 $N/2$的编码器，因此这一特点直接带来了更低的延迟。例如，对于大小为 $N = 32$的极化码，可节省整体异或门数量的42，3%。这一改动还减少了互连资源的消耗。节省互连资源十分重要，因为互连拥塞会限制在现场可编程门阵列上实现的大型组合电路的性能。

为了实现组合三进制解码器，需要实现三个函数 f^t、g_1^t 和 g_2^t 。与 f^b 类似，并利用公式 (7)，我们可以实现

算法 3 $\beta^b = Decode(\alpha, a)$ 使用所提出的方法

$N = lengh(\alpha)$
如果 $N == 2$ **那么**
  $\beta_0 \leftarrow (s(\alpha(0)) \oplus s(\alpha(1))) \cdot a(0)$
  如果 $abs(\alpha(1)) \geq abs(\alpha(0))$ **那么**
    $\beta_1 \leftarrow s(\alpha(1)) \cdot a(1)$
  else
    $\beta_1 \leftarrow (s(\alpha(0)) \oplus \beta(0)) \cdot a(1)$
  end
  返回 $\beta \leftarrow (\beta_0, \beta_1)$
else
  如果 $N == 4$ **则**
    $\alpha' \leftarrow f_2^b(\alpha)$
    $a' \leftarrow a(0\ to\ 1)$
    $\beta' \leftarrow Decode(\alpha', a')$
    $v' \leftarrow C^b(\beta'(0)\ \beta'(1))$

    $\alpha_0'' \leftarrow g_2^b(\alpha, (0, 0))$
    $\alpha_1'' \leftarrow g_2^b(\alpha, (0, 1))$
    $\alpha_2'' \leftarrow g_2^b(\alpha, (1, 0))$
    $\alpha_3'' \leftarrow g_2^b(\alpha, (1, 1))$
    $a'' \leftarrow a(2\ to\ 3)$

    $\beta_0'' \leftarrow Decode(\alpha_0'', a'')$
    $\beta_1'' \leftarrow Decode(\alpha_1'', a'')$
    $\beta_2'' \leftarrow Decode(\alpha_2'', a'')$
    $\beta_3'' \leftarrow Decode(\alpha_3'', a'')$

    如果 $v'(0) == 0$ **那么**
      如果 $v'(1) == 0$ **那么**
        $v'' \leftarrow C^b(\beta_0''(0)\ \beta_0''(1))$
      else
        $v'' \leftarrow C^b(\beta_1''(0)\ \beta_{1(1)}'')$
      end
    else
      如果 $v_1' == 0$ **，则**
        $v'' \leftarrow C^b(\beta_2''(0)\ \beta_2''(1)))$
      else
        $v'' \leftarrow C^b(\beta_3''(0)\ \beta_{3(1)}'')$
      end
    end
    返回 $\beta \leftarrow (v', v'')$
  else
    $\alpha' \leftarrow f_{N/2}^b(\alpha)$
    $a' \leftarrow a(0\ to\ N/2\text{-}1)$
    $\beta' \leftarrow Decode(\alpha', a')$
    $v' \leftarrow C^b(\beta'(0\ to\ N/2\text{-}1)\ \beta'(N/2\ to\ N\text{-}1))$

    $\alpha'' \leftarrow g_{N/2}^b(\alpha, v')$
    $a'' \leftarrow a(N/2\ to\ N\text{-}1)$
    $\beta'' \leftarrow Decode(\alpha'', a'')$
    $v'' \leftarrow C^b(\beta''(0\ to\ N/2\text{-}1)\ \beta''(N/2\ to\ N\text{-}1))$
    返回 $\beta \leftarrow C^b(v', v'')$
  end
end

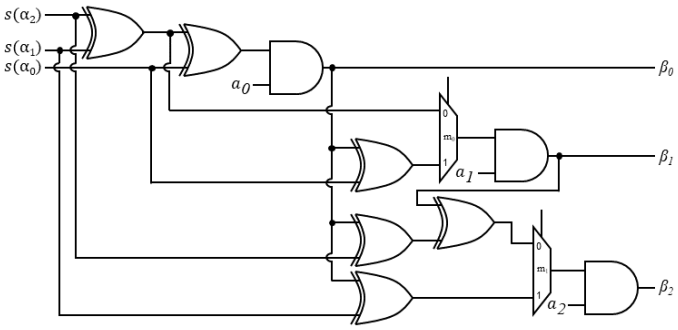


图7. 所提出的三元判决逻辑，等效于大小为 $N = 3$ 的极化码。

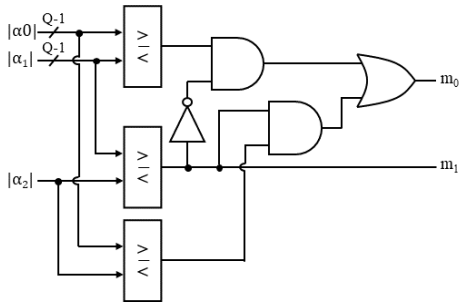


图8. 图7中三元判决逻辑所提出的控制逻辑。

f^t 函数，方法是使用两个比较器和一个多路复用器。为了进一步降低三元解码器的延迟和复杂度，我们将索引形式为 $3i$、 $3i$+1 和 $3i$+2 （$0 \leq i < N/3$）的码字估计为

$$\beta_{3i} = (s(\alpha_{3i}) \oplus s(\alpha_{3i+1}) \oplus s(\alpha_{3i+2})) \cdot a_{3i}, \tag{13}$$

$$\beta_{3i+1} = \begin{cases} (s(\alpha_{3i}) \oplus \beta_{3i}) \cdot a_{3i+1} & \text{if } (|\alpha_{3i}| \geq |\alpha_{3i+1}| \text{ and } |\alpha_{3i+2}| \geq |\alpha_{3i+1}|) \text{ or } (|\alpha_{3i}| \geq |\alpha_{3i+2}| \text{ and } |\alpha_{3i+1}| \geq |\alpha_{3i+2}|) \\ (s(\alpha_{3i+1}) \oplus s(\alpha_{3i+2})) \cdot a_{3i+1} & \text{otherwise,} \end{cases} \tag{14}$$

$$\beta_{3i+2} = \begin{cases} (s(\alpha_{3i+1}) \oplus \beta_{3i}) \cdot a_{3i+2} & \text{if } |\alpha_{3i+1}| \geq |\alpha_{3i+2}| \\ (s(\alpha_{3i+2}) \oplus \beta_{3i} \oplus \beta_{3i+1}) \cdot a_{3i+2} & \text{otherwise,} \end{cases} \tag{15}$$

其中， α_0、 α_1和 α_2 是输入至 g_1^t的输入对数似然比， g_2^t 为函数， a_0、 a_1 和 a_2 是与各输入对数似然比对应的冻结比特指示器。图7展示了所提出的三元判决逻辑，该逻辑由多路复用器和逻辑门实现。为避免拥塞，控制逻辑单独绘于图8中。控制逻辑生成所需信号，以控制图7中的两个多路复用器（m_0 和m_1）；该控制逻辑仅由比较器和逻辑门实现。

算法4给出了所提出的三元连续消除译码算法的伪代码。

2) 总体架构： 本节提出与算法3和算法4相对应的硬件架构。

算法 4 β^t = 解码(α, a)，使用所提出的方法

$N = lengh(\alpha)$
如果 $N == 3$，**则**
  $\beta_0 \leftarrow (s(\alpha(0)) \oplus s(\alpha(1)) \oplus s(\alpha(2))).a(0)$
  如果 $abs(\alpha(0)) \geq abs(\alpha(1))$ 且 $abs(\alpha(2)) \geq abs(\alpha(1))$ 或 $abs(\alpha(0)) \geq abs(\alpha(2))$ 且 $abs(\alpha(1)) \geq abs(\alpha(2))$ **则**
    $\beta_1 \leftarrow (s(\alpha(0)) \oplus \beta_0) \cdot a(1)$
  else
    $\beta_1 \leftarrow (s(\alpha(1)) \oplus s(\alpha(2))) \cdot a(1)$
  end
  如果 $abs(\alpha(1)) \geq abs(\alpha(2))$ **则**
    $\beta_2 \leftarrow (s(\alpha(1)) \oplus \beta_0) \cdot a(2)$
  else
    $\beta_2 \leftarrow (s(\alpha(2)) \oplus \beta_0 \oplus \beta_1) \cdot a(2)$
  end
  返回 $\beta \leftarrow (\beta_0, \beta_1, \beta_2)$
else
  $\alpha' \leftarrow f^t_{N/3}(\alpha)$
  $a' \leftarrow a(0\ to\ N/3\text{-}1)$
  $\beta' \leftarrow Decode(\alpha', a')$
  $v' \leftarrow C^t(\beta'(0\ to\ N/3\text{-}1)\ \beta'(N/3\ to\ 2N/3\text{-}1)\ \beta'(2N/3\ to\ N\text{-}1))$

  $\alpha'' \leftarrow g^t_{1N/3}(\alpha,\ v')$
  $a'' \leftarrow a(N/3\ to\ 2N/3\text{-}1)$
  $\beta'' \leftarrow Decode(\alpha''\ a'')$
  $v'' \leftarrow C^t(\beta''(0\ to\ N/3\text{-}1)\ \beta''(N/3\ to\ 2N/3\text{-}1)\ \beta''(2N/3\ to\ N\text{-}1))$

  $\alpha''' \leftarrow g^t_{2N/3}(\alpha, v', v'')$
  $a''' \leftarrow a(2N/3\ to\ N\text{-}1)$
  $\beta''' \leftarrow Decode(\alpha'''\ ,\ a''')$
  $v''' \leftarrow C^t(\beta'''(0\ to\ N/3\text{-}1)\ \beta'''(N/3\ to\ 2N/3\text{-}1)\ \beta'''(2N/3\ to\ N\text{-}1))$
  返回 $\beta^t \leftarrow C^t(v', v'', v''')$
end

图9展示了使用 Arikan 核构建的、规模为 N 的译码器的通用组合架构。规模为 N 的 Arikan 译码器由两个规模为 $N/2$ 的译码器通过一个 f^b、一个 g^b 和一个 C^b 连接而成。规模为 $N = 4$ 的译码器用作 Arikan 译码器的基本构建块。与规模为 [19]，的译码器不同，所提出的译码器包含用于加载下一帧对数似然比和冻结比特指示集所需的存储器，以及用于输出先前估计码字所需的存储器。

图10展示了所提出的、规模为 N的纯三元极化码组合架构。规模为 N的译码器由三个规模为 $N/3$的译码器构成。粘合逻辑包括一个 f^t、一个 g^t_1、一个 g^t_2 ，以及两个规模为 $N/3$的 C^t 。在此情况下，规模为 $N = 3$ 的译码器用作基本构建块。其存储结构与所提出的 Arikan 方案类似。

由于SC算法具有递归特性，因此可以通过混合设计Arikan架构和三元架构来构造大小为 N 的MK码。核顺序决定使用 $N = 3$ 还是 $N = 4$ 作为判决逻辑

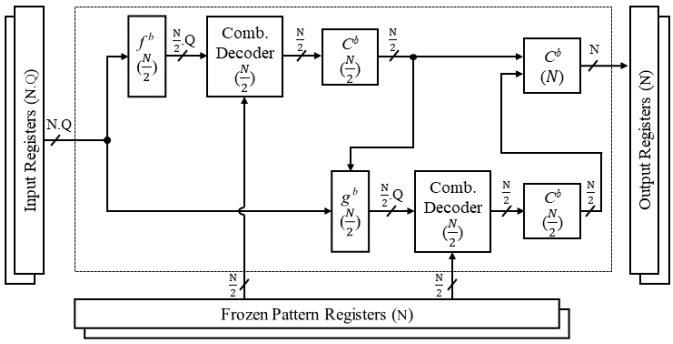


图9。 Arikan 译码器的组合架构 。

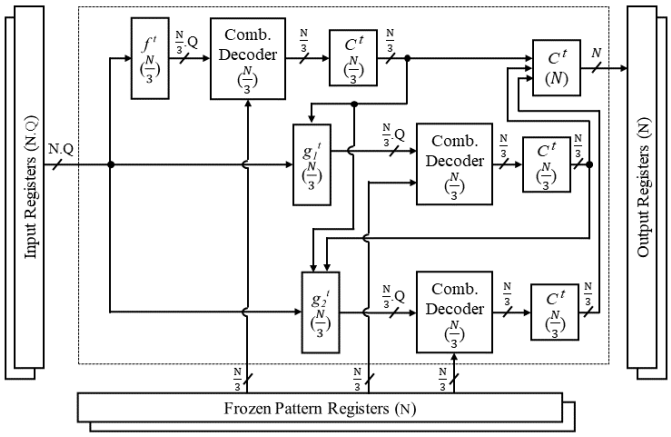


图10。所提出的纯三元译码器组合架构。

电路。在核顺序不满足将 $N = 4$ 用作判决电路的条件时，可以将其替换为 $N = 2$。唯一的区别是，不再使用预计算方法。然而，本文中的所有MK码都满足将 $N = 4$ 用作判决电路的条件。选择基本构建块后，核顺序决定顶层采用二元还是三元粘合电路。我们继续使用粘合逻辑级，直至构造出目标分组码。

图11展示了与图1(b)中的译码树相关联、规模为 $N = 6$且具有 $G = T_3 \otimes T_2$的 MK 极化码架构。为避免拥塞，此处未显示寄存器和冻结比特指示器。由于核序列中的最后一个核是 T_2，因此该译码器采用二元判决电路（$N = 2$）作为基本构建块。由于下一个核为 T_3 ，粘合逻辑由一个 f^t、一个 g^t_1、一个 g^t_2和三个二元合并逻辑（C^b）组成。此外，还采用三元合并函数（C^t），在将估计码字写入输出寄存器之前接收译码树根节点处的估计码字。在将估计码字写入输出寄存器之前，是使用二元还是三元合并逻辑，由核序列中的第一个核决定（此处为T_3 ）。

3）存储器架构：所提出的 Arikan、纯三元和 MK 组合架构占用 $N \times (Q+2)$ 个寄存器位。这些存储元件用于存储输入对数似然比（$N \times Q$ 位）、估计码字（N 位），

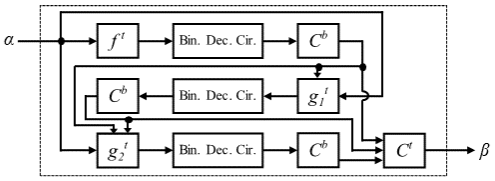


图11. 用于 $N = 6$ 和 $G = T_3 \otimes T_2$的组合式 MK 极化译码器。

以及冻结比特指示集（N 位）。从图9、图10和图11可以看出，输入寄存器与输出寄存器之间没有同步逻辑元件，即寄存器或 RAM 阵列。组合译码器的这一特性提高了功耗和处理时间效率。移除 RAM 路由器还降低了硬件复杂度，并消除了较长的读写延迟。译码器的延迟为一个时钟周期，因为它在接收输入对数似然比后，会在一个较长的时钟周期内生成估计码字。因此，输入寄存器与输出寄存器之间的逻辑电路决定了整体关键路径。

C. MK 译码器的复杂度分析

所提出架构的复杂度可表示为设计中基本构建模块的总数，即比较器、加法器和减法器的数量。假设c_N^b 表示实现 f^b时所用的比较器数量。从算法3可以看出，Arikan 译码器所消耗的比较器初始值为 $c_4^b = 2$。 [19] 表明，大小为 N 且具有 $c_4^b = 2$ 的基于组合逻辑的 Arikan 译码器所需的基本构建模块总数可估算为

$$c_N^b + s_N^b + r_N^b = N(\frac{3}{2}\log_2(N) - 1) \approx \frac{3}{2}N\log_2(N), \quad (16)$$

其中， s_N^b 和 r_N^b 分别表示判决逻辑所消耗的比较器数量，以及实现 g^b所用的加法器和减法器总数。公式（16）证明，Arikan 组合译码器的复杂度为 $\mathcal{O}(N\log_2(N))$量级。

对于纯三元极化码，用于实现 f^t和 g_1^t的大小为N 的译码器所需比较器的数量（c_N^t）满足以下递归关系：

$$c_N^t = 3c_{\frac{N}{3}}^t + N = 3(3c_{\frac{N}{9}}^t + \frac{N}{3}) + N = \dots. \quad (17)$$

使用算法4，我们可以通过 $c_3^t = 3$初始化 c_N^t。式（17）的递推方程可精确计算为 $N\log_3(N)$。已知 $N = 3$ 判决逻辑中使用的比较器数量为 $s_3^t = 3$，可得 $s_N^t = N$。

最后，将估算加法器和减法器的数量。函数 g_1^t 由一个加法器和一个减法器实现，函数 g_2^t 由两个加法器和两个减法器实现。因此，加法器和减法器的总数可计算为 $r_N^t = \frac{3}{2}c_N^t$。由此，纯三元译码器的基本逻辑块数量可估算为

$$c_N^t + s_N^t + r_N^t = N(\frac{5}{2}\log_3(N) + 1) \approx \frac{5}{2}N\log_3(N), \quad (18)$$

公式（18）验证了纯三元组合译码器的复杂度为 $\mathcal{O}(N\log_3(N))$ 量级。根据（16）和（18），显而易见，当 Arikan 码和纯三元极化码的码块长度处于同一范围内时（以 $N^b = 2048$ 和 $N^t = 2187$ 为例），Arikan 码的复杂度低于纯三元码。这是因为三元置信传播函数和判决逻辑电路比 Arikan 码中的对应部分更复杂。因此，我们可以得出结论：当 $N_{min} = 2$ 和 $N_{max} =$ 为 4096 时，所提出的混合核译码器的复杂度下界为 $\frac{3}{2}N\log_2(N)$，上界为$\frac{5}{2}N\log_3(N)$，即

$$\frac{3}{2}N\log_2(N) \le c_N^{MK} \le \frac{5}{2}N\log_3(N). \quad (19)$$

需要注意的是，无法为MK码的复杂度分析提供通用方程，因为它直接取决于核序列中不同核的数量和位置。

IV. 组合极化译码器的自动生成

A. 高级综合

将算法或行为的高层描述转换为寄存器传输级（RTL）的过程称为高级综合（HLS） [22]。HLS工具可将C/C++或Python等高级编程语言转换为硬件描述语言（HDL）。在MK组合架构中，更改码块长度或核顺序时，所有HDL子模块都需要修改。所提出的算法可自动转换为HDL，从而加快组合译码器的开发过程。

B. 生成过程

利用算法3和算法4，我们开发了一个使用 Python 编写的极化码编译器[23]，用于自动生成硬件描述语言文件，以实现具有不同核排序的各种码块长度译码器。用户需要输入目标极化码的码块长度和可选的核顺序。如果用户未输入核顺序，编译器会自动分配纠错性能最高的核排序。针对公式（2–10）、基本构建模块、顶层模块和子模块以及接口，编译器设有相应的函数。极化码编译器根据预定义规则调用相关函数。这些函数从编译器的顶层模块获取输入，并输出所需的 VHDL 文件。通常，编译过程类似于高级综合流程：

参数（由用户指定） → 函数（高级描述） → VHDL 文件（硬件描述语言）

C. 时间效率

使用运行频率为 1.90 GHz 的 AMD Ryzen 7 PRO 5850U x64中央处理器，对所提出的极化码编译器的时间效率进行评估。生成所有

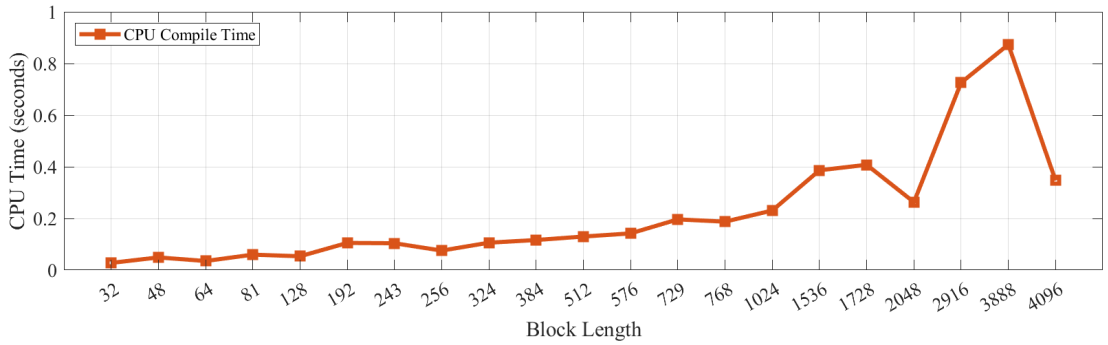


图12. 不同长度极化码的编译时间。

不同规模的极化译码器所需的全部 VHDL 文件所需时间如图12所示。每个数据值均通过运行所提出的编译器 20 次并计算平均值获得。CPU 时间会随所需子模块的数量和复杂度而变化，而这些因素直接受码长以及核的排列顺序影响。因此，正如图12所示，编译 MK译码器所需的时间预计会长于编译 Arıkan译码器所需的时间。不过，对于大多数极化码，编译时间不到 0.4 秒，而最长的 MK 极化码所需时间为 0.88 秒。因此，使用所提出的极化码编译器能够高效地生成组合极化译码器所需的全部 VHDL 文件。

V. 实现结果与比较

本文所有极化码均使用 VHDL 编码，并在 Xilinx Vivado 2019.1 环境中进行描述。为验证设计，针对 Xilinx Virtex-6 FPGA（40 纳米）进行逻辑综合、技术映射和布局布线。通过加性高斯白噪声信道进行 BPSK调制，软件程序生成随机码字并将其传输至 FPGA。通过实现具有不同核排序的不同编码，对所提出译码器的灵活性和可扩展性进行评估。在实验中，我们使用一组额外的寄存器来存储输入、输出和冻结模式数据。该方法使译码器能够根据给定的冻结模式对一帧进行译码，同时加载另一帧及其对应的冻结集，从而避免性能下降。同样，在另一次译码进行期间，也可以传输估计得到的码字。

为便于比较不同方案，译码时延定义为对一帧进行译码所需的时间。与二进制情形类似，MK极化码的编码吞吐量和信息吞吐量可分别计算为 $\mathcal{T}_C = N.f$和 $\mathcal{T}_C = N.f.\mathcal{R}$。

A. 纠错性能与量化

如前所述，LDPC WiMAX标准 [15] 指出，仅使用一个或少数几个非二元核即可构造相当多种码块长度。图13展示了仅利用一个非二元核、并以浮点格式表示对数似然比的编码的纠错性能，因为该标准特别关注此类编码。

我们将量化方案定义为 $Q(Q_i,\ Q_c)$，其中 Q_i 和 Q_c 分别表示用于表示内部对数似然比位和信道对数似然比位的总位数。图14比较了极化码 $\mathcal{PC}(1024,\ 512)$ 在 $Q(4,\ 4)$、$Q(5,\ 5)$ 和 $Q(6,\ 6)$ 量化方案下的性能损失。显然，与 $Q(6,\ 6)$ 相比，$Q(5,\ 5)$ 的纠错性能与浮点实现非常接近，差距可以忽略不计。因此，本研究选择 $Q(5,\ 5)$ 作为量化方案。

*B. FPGA*实现结果与比较

随着码长增加，组合译码器消耗的逻辑块数量增多，布线也变得更加复杂。因此，码长越长，互连延迟越大。由于采用预制布线资源，这种现象在现场可编程门阵列中尤为明显（与专用集成电路相比）。

表I总结了所提出的组合译码器与 [19]在多种码块长度下的FPGA利用率和时序性能。可以看出，在所有情况下，所提出的译码器的工作频率和编码吞吐量均大致翻倍。与 [19],类似，所提出的译码器性能会因互连延迟而随着码长增加而下降。不过，由于我们的实现消耗的逻辑资源较少（减少了16%至47%），这种影响较小；这通常是因为在译码器架构中以组合逻辑取代了编码器。需要注意的是，没有将查找表（LUT）用作存储器。我们的设计使用寄存器来保留输入对数似然比、冻结比特模式以及估计输出。寄存器也用于实现小型逻辑电路。与[19]相比，所提出的组合译码器使用的寄存器数量为其2.6至3.09倍。主要原因是 [19] 不支持在解码另一帧时加载下一帧（及其对应的冻结模式比特）。此外，它也没有所需的存储器来暂存先前估计的码字，同时解码另一帧。最后，所提出的译码器不消耗任何随机存取存储器，而 [19] 占用206至7168位随机存取存储器。从表I还可以看出，译码器的可扩展性也有所提升。资源

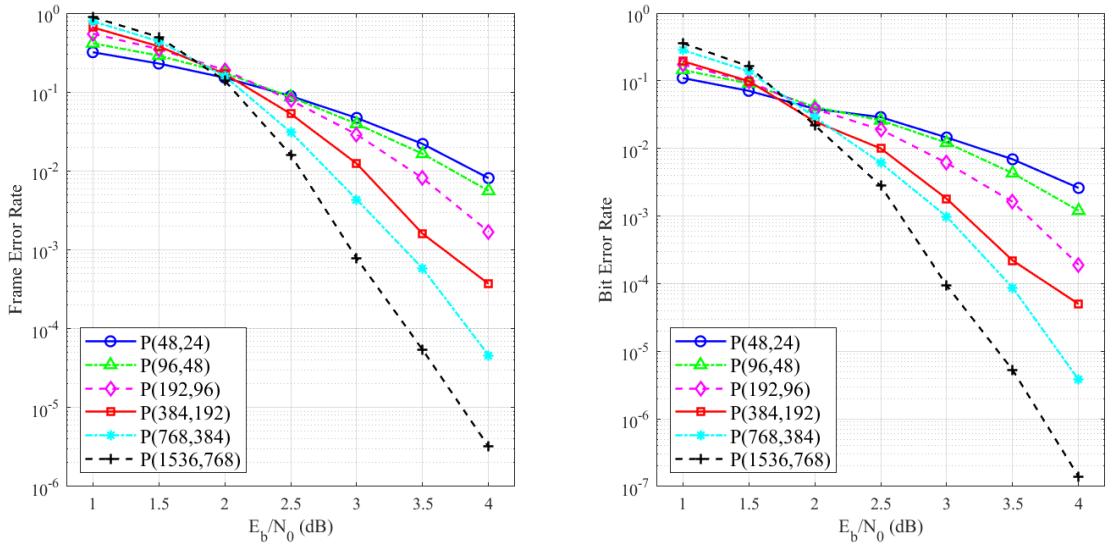


图13. 速率为 $R = \frac{1}{2}$ 的MK极化码的纠错性能。

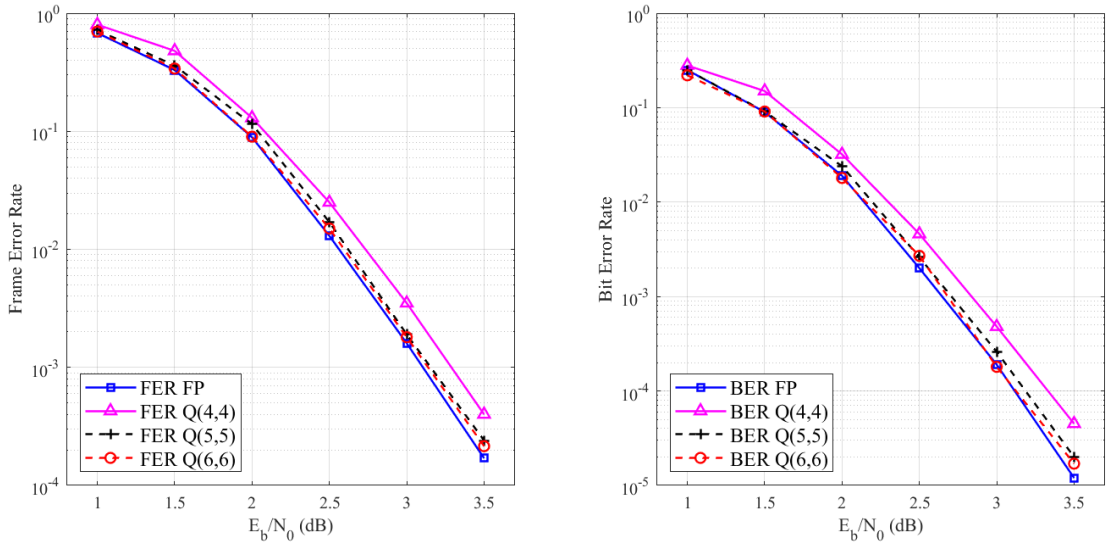


图14. LLR量化对 $\mathcal{PC}$(1024，512) MK码纠错性能的影响。

码长为 N 的编码所需资源是码长为 $N/2$ 的编码的2倍，此外还要加上连接两个 $N/2$ 编码的粘合逻辑所带来的额外开销。

表 II 汇总了各种纯三元码和 MK 码的 FPGA利用率及性能参数。我们尝试纳入所有可能的核排序。资源消耗取决于用于构建给定编码的核的数量。尽管三元层比二元层占用更多资源，但它们能够构建更大的编码。例如，表 I 中大小为 512（9 个核）的编码与表 II 中大小为 576（8 个核）的编码，所占用的查找表和寄存器数量几乎相同。因此，资源消耗与目标码块长度成正比，而与核顺序或基本构建块无关，这再次体现了该设计的可扩展性。表 II 还提供了

为便于比较， [11]中报告的相同码块长度的性能参数。所提出的方案将延迟降低了 63% 至 75%。在工作频率方面，我们的架构的工作频率比 [11]的架构低 2 至 3 个数量级。较低的工作频率直接转化为动态功耗的降低。最后，针对码块长度为 [11]的编码，所提出的译码器可提供 1.68× 至3.05× 的更高吞吐量。

表 III 总结了采用 SC [11], 展开式 SC[24],快速SSC [25] 和 MK 快速 SSC [10] 实现的各种先进译码器的现场可编程门阵列资源利用率和时序性能。需要注意的是，只有 [10]支持 MK 极化码。为确保比较公平，所有设计均采用 [27]中的缩放技术，在 40纳米工艺下实现或缩放。我们的方案消耗的存储器数量几乎相同

表I 各种 Arikan 极化码的拟合后结果

码块长度	本工作 LUTs	寄存器	RAM (位)	f (兆赫)	T/P (兆比特每秒)	[19] LUTs	寄存器	RAM (位)	f (兆赫)	T/P (兆比特每秒)
32	1611	742	0	51.875	1660	1918	206	224	27.5	880
64	3921	1480	0	25.625	1640	5126	392	448	13,28	850
128	8926	2985	0	11.3	1446.4	14517	783	896	6.4	820
256	20259	6058	0	5.34	1367.9	35152	1561	1792	2.93	750
512	45986	12334	0	2.49	1275.7	77154	3090	3584	1.43	730
1024	102740	25139	0	1.19	1214.2	193456	6151	7168	0.59	600

表 II 各类 MK 极化码的拟合后结果

码块长度	核顺序	本工作 LUTs	寄存器	RAM (位)	L (μs)	f (兆赫)	T/P (兆比特每秒)	[11] L (μs)	f (兆赫)	T/P (兆比特每秒)
48	{3,2,2,2,2}	3049	1088	0	0.029	34.17	1640	0.111	1230	430.9
81	{3,3,3,3}	5134	2129	0	0.049	20.55	1664.5	0.132	1230	615
192	{3,2,2,2,2,2,2}	15005	4426	0	0.12	8.3	1592.7	0.477	1230	402.3
243	{3,3,3,3,3}	1997	3443	0	0.147	6.78	1646.5	0.422	1230	578.1
324	{2,2,3,3,3,3}	27037	8719	0	0.218	4.59	1487.8	0.587	1110	555
384	{3,2,2,2,2,2,2,2}	32575	9003	0	0.288	3.47	1330.9	1.04	1110	368.7
576	{2,2,2,2,2,2,3,3}	49915	13933	0	0.361	2.77	1597.1	1.11	1110	521.7
729	{3,3,3,3,3,3}	72808	22242	0	0.465	2.15	1566.5	-	-	-
768	{2,2,3,2,2,2,2,2,2}	74822	18351	0	0.518	1.93	1483.6	2.096	1110	366.5

表 III 码长为 1024 的极化码的 FPGA利用率与性能比较。

Work	算法	Rate	P_e	集成电路类型	工艺 (nm)	Q (Q_i, Q_c)	LUTs	Reg.	RAM (位)	L (兆赫)	f (μs)	T/P （兆比特每秒）	支持数量 编码
[11]*	SC	$1/2$	60	ASIC	65	(6, 6)	-	-	29048	1.21	1803.75	588.2	15
[24]	展开式 SC	$1/2$	-	FPGA	40	(5, 5)	93225	43236	0	1.52	239	1500	15
[25]	快速SSC	$1/2$	128	FPGA	40	(7,6)	18982	3384	37700	1.3	94.36	393	15
[10]	MK 快速 SSC	$1/2$	120	FPGA	40	(5,4)	23126	4548	40200	1.18	86.14	432	55
本工作	MK 组合式连续消除	any	-	FPGA	40	(5,5)	102740	25139	0	0.84	1.19	1214.2	55

* 与 [26], 类似，性能参数根据 [27]中的缩放技术归一化至 40 纳米 CMOS技术。

于 [11] ，但延迟降低了 31%，工作频率降低了 3 个数量级，吞吐量则达到其 2.07 倍。与 [24], 相比，两种方案占用的逻辑电路（查找表）数量大致相同，而 [24]消耗的寄存器多 72%。在时序性能方面，[24] 的延迟和工作频率分别高出 81% 和 2 个数量级。不过，其吞吐量提高了 23.5%。与基于快速SSC [25]和 [10], 的设计相比，所提出的方案消耗的逻辑电路约为 $5\times$，并且需要多出近 20 K 位寄存器。不过需要注意的是，我们的译码器不使用随机存取存储器，而 [25]和 [10]分别需要 36.8 和 39.26 K 位随机存取存储器。此外，与 [25] 和 [10], 相比，所提出的译码器分别实现了 $2.09\times$ 和 $1.81\times$ 更高的吞吐量。最后，我们提出的译码器和 [10] 支持 55 种不同的码块长度，而其他译码器仅支持 15 种不同的编码。

VI. 结论

本文提出了一种基于组合逻辑的硬件架构，用于基于SC算法解码MK极化码。所提出的架构支持在线码率分配机制，具有高吞吐量。它仅需一个时钟周期即可解码整个码字，从而降低了工作频率和动态功耗与基于同步SC的架构相比，功耗有所降低。本文报告了不同码块长度和核排序下的现场可编程门阵列利用率。根据实现结果，对于大小为 $N = 81$的编码，所提出的译码器可实现高达1664.5兆比特每秒的编码吞吐量。此外，本文还进行了复杂度分析，验证了实现结果。

最后，我们构建了一个基于 Python 的极化码编译器，可自动生成所提出译码器在现场可编程门阵列上的实现所需的 VHDL 文件。只需输入码块长度及其核顺序，极化码编译器便会自动输出所有必需的 VHDL 模块。此外，还提供了极化码编译器的编译时间。

致谢

本研究得到了芬兰科学院通过6G旗舰计划项目资助编号346208的支持。

参考文献

[1] E. Arikan, 《信道极化：一种为对称二进制输入无记忆信道构造达到容量的编码的方法》, *IEEE*信息论汇刊，第55卷，2009年，第7期，页码3051–3073。[2] I. Tal和A. Vardy, 《极化码的列表译码》, *IEEE*信息论汇刊，第61卷，第5期，页码2213–2226，2015年。

[3] K. Niu和K. Chen，《极化码的CRC辅助译码》，IEEE通信快报，第16卷，第10期，页码1668–1671，2012年。[4] 3第三代合作伙伴计划（3GPP），《5G；新空口；复用与信道编码》。3GPP文档38.212 V.15.3.0，2018年。[5] S. Han、B. Kim和J. Ha，《速率兼容打孔极化码》，IEEE通信快报，第26卷，第4期，页码753–757，2022年。[6] V. Bioglio、F. Gabry和I. Land，《极化码的低复杂度打孔与缩短》，载于2017年IEEE无线通信与网络会议研讨会（WCNCW），2017年，页码1–6。[7] C. Xia、C.-Y. Tsui和Y. Fan，《采用核替换构造多核极化码》，IEEE无线通信快报，第9卷，第11期，页码1879–1883，2020年。[8] F. Gabry、V. Bioglio、I. Land和J.-C. Belfiore，《极化码的多核构造》，载于2017年IEEE国际通信会议研讨会（ICC Workshops）。IEEE，2017年，页码761–765。[9] V. Bioglio、F. Gabry、I. Land和J.-C. Belfiore，《基于最小距离的多核极化码构造》，载于GLOBECOM 2017—2017年IEEE全球通信大会。IEEE，2017年，页码1–6。[10] H. Rezaei、N. Rajatheva和M. Latva-Aho，《低时延多核极化码译码器》，IEEE Access，第10卷，页码119460–119474，2022年。[11] G. Coppolino、C. Condo、G. Masera和W. J. Gross，《一种多核多码极化译码器架构》，IEEE电路与系统汇刊I：常规论文，第65卷，第12期，页码4413–4422，2018年。[12] G. Sarkis、P. Giard、A. Vardy、C. Thibeault和W. J. Gross，《快速极化码译码器：算法与实现》，IEEE通信领域精选期刊，第32卷，第5期，页码946–957，2014年。[13] H. Rezaei、N. Rajatheva和M. Latva-Aho，《一种用于极化码的组合多核译码器》，arXiv预印本arXiv:2211.08778，2022年。[14] R. Mori和T. Tanaka，《采用密度演化构造的极化码性能》，IEEE通信快报，第13卷，第7期，页码519–521，2009年。[15] K.-W. Shin和H. ju Kim，《一种用于IEEE 802.16e移动WiMAX的多模式LDPC译码器》，半导体技术与科学期刊，第12卷，页码24–33，2012年。[16] M. Benammar、V. Bioglio、F. Gabry和I. Land，《多核极化码：极化证明与错误指数》，载于2017年IEEE信息论研讨会（ITW）。IEEE，2017年，页码101–105。[17] K. Niu、K. Chen和J.-R. Lin，《超越Turbo码：速率兼容打孔极化码》，载于2013年IEEE国际通信会议（ICC），2013年，页码3423–3427。[18] R. Wang和R. Liu，《一种新型极化码打孔方案》，IEEE通信快报，第18卷，第12期，页码2081–2084，2014年。[19] O. Dizdar和E. Arıkan，《采用组合逻辑实现的极化码高吞吐量、高能效连续消除译码器》，IEEE电路与系统汇刊I：常规论文，第63卷，第3期，页码436–447，2016年。[20] C. Leroux、A. J. Raymond、G. Sarkis和W. J. Gross，《一种用于极化码的半并行连续消除译码器》，IEEE信号处理汇刊，第61卷，第2期，页码289–299，2013年。[21] C. Zhang和K. K. Parhi，《用于连续消除极化译码器的低时延串行与重叠架构》，IEEE信号处理汇刊，第61卷，第10期，页码2429–2441，2013年。[22] R. Nane、V.-M. Sima、C. Pilato、J. Choi、B. Fort、A. Canis、Y. T. Chen、H. Hsiao、S. Brown、F. Ferrandi等，《FPGA高级综合工具的调查与评估》，IEEE集成电路与系统计算机辅助设计汇刊，第35卷，第10期，页码1591–1604，2015年。[23] H. Rezaei、N. Rajatheva和M. Latva-Aho，《极化码编译器》。访问日期：2023年1月。[在线]。可用地址：https://github.com/hosseinrezaeii91/Polar-Compiler，2023年。[24] P. Giard、A. Balatsoukas-Stimming、G. Sarkis、C. Thibeault和W. J. Gross，《用于低码率极化码的快速低复杂度译码器》，信号处理系统期刊，第90卷，第5期，页码675–685，2018年。[25] H. Rezaei、V. Ranasinghe、N. Rajatheva、M. Latva-Aho、G. Park和O.-S. Park，《超快速极化译码器的实现》，载于2022年IEEE国际通信会议研讨会（ICC Workshops），2022年，页码235–241。[26] F. Ercan、T. Tonnellier、N. Doan和W. J. Gross，《实用动态SC翻转极化译码器：算法与实现》，IEEE信号处理汇刊，第68卷，页码5441–5456，2020年。[27] P. Giard、A. Balatsoukas-Stimming、T. C. M¨uller、A. Bonetti、C. Thibeault、W. J. Gross、P. Flatresse和A. Burg，《PolarBear：一款用于极化码译码的28纳米FD-SOI专用集成电路》，IEEE电路与系统新兴及精选主题期刊，第7卷，2017年，第4期，页码616–629。



Hossein Rezaei 于2016年获得伊朗德黑兰伊朗科技大学数字电子学理学硕士学位。目前，他正在芬兰奥卢大学攻读博士学位。他目前的研究兴趣包括纠错算法的设计与实现，尤其关注极化码、数字信号处理的超大规模集成电路设计，以及在嵌入式平台上实现通信系统。



Nandana Rajatheva （IEEE高级会员）于1987年获得斯里兰卡莫拉图瓦大学电子与电信工程理学学士（荣誉）学位，并分别于1991年和1995年获得加拿大马尼托巴省温尼伯马尼托巴大学的理学硕士和博士学位。他目前是芬兰奥卢大学无线通信中心的教授。研究生期间，他曾是马尼托巴省的加拿大英联邦学者。1995年至2010年，他担任教授／副教授曾在莫拉图瓦大学和泰国亚洲理工学院担任教授。他目前负责Hexa-X欧盟项目中的人工智能驱动的空中接口设计任务。他已在期刊和会议论文集中发表超过200篇经过同行评审的文章，并与他人合著。其研究兴趣包括超越5G的物理层、用于物理层和媒体访问控制的机器学习、集成感知与通信以及信道编码。



Matti Latva-aho （IEEE高级会员）于1992年、1996年和1998年分别获得芬兰奥卢大学电气工程理学硕士、技术执照和技术博士（荣誉）学位。1992年至1993年，他在芬兰奥卢的诺基亚移动电话担任研究工程师，随后加入奥卢大学的无线通信中心（CWC）。Latva-aho教授于1998年至2006年担任CWC主任，并担任通信工程系主任，直至2014年8月。目前，他担任学院的芬兰教授兼国家6G旗舰计划主任。他同时也是东京大学的全球研究员。他的研究兴趣涉及移动宽带通信系统，目前其团队专注于6G系统研究。Latva-aho教授在无线通信领域发表了500多篇会议或期刊论文。他于2015年因在移动通信研究方面的成就获得诺基亚基金会奖。