光电技术及应用

DOI：10.16818/j.issn1001－5868.2023022601

一种高资源效率的 Fast-RCSC 极化码译码器

陶玉辉，杜高明，郭文杰，李桢旻，王晓蕾*

（合肥工业大学 微电子学院，合肥 230009）

摘　要： 针对现有极化码软输出译码器存在的高资源消耗与低资源效率，设计了一种快速低复杂度软取消(Fast Reduced Complexity Soft-Cancelation，Fast-RCSC)译码算法及其译码器硬件架构。Fast-RCSC 算法对内部特殊结点进行完整计算，在减少译码周期的同时仍有较好译码性能。基于不同特殊结点公式之间存在相似性，进而通过对引入的特殊结点模块进行计算结果复用以及计算模块分时复用，减少特殊结点模块资源消耗。通过共用存储单元以及对不足存储单元数据宽度的数据进行合并，降低存储资源消耗。在华润上华(Central Semiconductor Manufacturing Corporation，CSMC)180 nm 工艺下综合结果表明，设计的译码器在码长为 1 024 的情况下，面积为 2.92 mm^2，资源效率为 245.2 Mbps/mm^2，相比现有软输出译码器有不同程度的提升。

关键词： 极化码译码器；软输出；资源效率；资源复用；专用集成电路

中图分类号： TN47　**文章编号：** 1001－5868(2023)03－0436－08

A High Resource Efficiency Fast-RCSC Polar Decoder

TAO Yuhui，DU Gaoming，GUO Wenjie，LI Zhenmin，WANG Xiaolei

（School of Microelectronics，Hefei University of Technology，Hefei 230009，CHN）

Abstract： The existing polar code soft-output decoders suffer from high resource consumption and low resource efficiency of the hardware implementation. In this paper，a Fast-RCSC decoding algorithm and its decoder hardware architecture were designed. The Fast-RCSC algorithm performed complete calculations on internal special nodes，reducing the decoding cycle while maintaining high decoding performance. Based on the similarity between different special node formulas，the resource consumption of special node modules was reduced by reusing calculation results of the introduced special node modules and reusing time-division multiplexing scheme of calculation modules. The storage resource consumption was reduced by sharing storage units and merging data with bit width lower than the limit. The synthesis results in CSMC 180 nm process show that the designed decoder has an area size of 2.92 mm^2 and a resource efficiency of 245.2 Mbps/mm^2 when the code length is 1 024 bits，which demonstrates different degrees of improvement compared to existing polar code soft-output decoders.

Key words： polar code decoder；soft output；resource efficiency；resource reuse；ASIC

0　引言

极化码由 Arikan 在 2009 年首次提出[1]，由于其被证明能够达到香农极限[2]，因此得到极大的关注。对于现代通信系统，极化码的软输出作为外在信息，可应用于多输入多输出检测、信道估计和混合

收稿日期：2023－02－26.

基金项目：国家重点研发计划项目(2018YFB2202604)；安徽省高校协同创新项目(GXXT-2019-030).

* **通信作者：**王晓蕾　**E-mail：** wangxiaolei@hfut.edu.cn

自动重传请求方案[3-5]。极化码通常使用置信传播(Belief Propagation，BP)算法[6]和软取消(Soft Cancelation，SCAN)[7]算法来产生软输出。

极化码的 BP 译码算法最先在文献[6]中被提出，文献[8]为了提升运算并行度提出了一种泛洪 BP(Flooding BP，FO-BP)算法，但是增加了算法计算复杂度和资源消耗。文献[9]通过简化译码过程中的消息传递，减少了迭代中的运算数量。文献[10]通过将相邻两列消息合并为一列，减少对中间消息的存储，降低整体内存使用量。文献[11]通过提出一种流水线以及展开方案，减少关键路径长度，获得较高的系统频率和吞吐率。文献[12]通过机器学习技术对极化码进行译码，通过提出一种性能调度学习方法，提升短码下的极化码译码性能。

文献[7]提出了一种 SCAN 算法，其译码过程类似于文献[13]中的连续取消(Successive Cancellation，SC)算法，是一种基于串行消息更新的译码算法。相较于 BP 算法，SCAN 算法有较低复杂度以及更少存储单元的优势，并且由于传播信息更高效，SCAN 算法的收敛速度比 BP 算法快得多。在硬件方面，文献[14]将 SCAN 算法在现场可编程门阵列(Field Programmable Gate Array，FPGA)上实现，相比 BP 算法在硬件复杂度以及硬件资源效率上均存在优势，但是串行消息更新消耗的周期数仍比较高。文献[15]提出了两种基于 SCAN 的译码算法，一种是低复杂度软取消(Reduced Complexity Soft-Cancelation，RCSC)译码算法，RCSC 算法在对左子结点进行向左运算时简化了对输入数据的需求，减少了对所需返回值的存储。另一种是基于结点合并的低延迟软取消(Reduced Latency Soft-Cancelation，RLSC)算法，RLSC 算法通过对译码二叉树进行剪枝，并且采用一种修改结点激活(Modified Node Activation，MNA)的近似算法，在译码过程中能够遍历更少的译码结点，达到减少译码周期，降低延迟的目的。文献[16]提出一种 MRLSC 算法，消除 RLSC 算法中由于采用 MNA 近似造成的译码性能损失问题。

文献[14]中的译码器所采用的 SCAN 译码算法所需译码周期较大，因此其吞吐率和资源效率都较低。文献[15]中的 RCSC 译码算法仅考虑 RATE0 与 RATE1 作为特殊结点，因此其所需译码周期仍较大。文献[15]中的 RLSC 译码算法虽能够大幅减少译码所需周期，但其引入的特殊结点模块没有充分考虑各类结点之间的关联关系，会在引入特殊结点模块时消耗较多资源，且 RLSC 译码算法在译码过程中采用了近似算法，破坏了结点之间的信息传递关系，导致在同等条件下相对 RCSC 译码算法会有约 0.5 dB 的译码性能下降。文献[16]提出的 MRLSC 算法相比 RLSC 算法虽在译码性能上有所提升，但其结点之间的分布关系也被改变，相较于 RCSC、SCAN 算法译码性能明显下降。除此之外，以上译码器在译码过程中没有充分利用各对数似然比(Likelihood Rate，LLR)值存取关系，在存储资源上消耗较大，使得译码器的资源效率较低，存在改进的空间。

通过研究发现，在计算 REP-SPC 结点时不采用近似算法，改为左右两部分计算，可以在加速译码迭代的同时相对 RCSC 译码不产生译码性能下降。不同的译码特殊结点在返回值计算过程中存在一些通用的部分，例如 011 结点的返回值属于 SPC 结点返回值的一部分，001 结点的返回值属于 REP 结点返回值的一部分，这使得在计算特殊结点的过程中存在计算结果复用以及计算单元分时复用的可能性。另外，由于信道输入的初始 LLR 值与译码得到的软输出值无需同时被使用，使得这两部分数据可以共用同一存储单元。

本文设计并实现了一种相对 RCSC 算法无译码性能下降的 Fast-RCSC 译码算法，以及一种高资源效率的极化码 Fast-RCSC 译码器。

(1)通过将 RLSC 译码算法中计算 REP-SPC 结点时造成译码性能下降的 MNA 近似算法舍去，将 REP-SPC 结点的计算分为左右两个运算部分进行完整计算，使得在加速 RCSC 译码迭代的过程中不产生译码性能下降。

(2)针对引入的特殊结点模块消耗资源过大，将在第一个特殊结点译码周期输出的 011 和 001 结点的计算结果复用于在第二个计算周期输出的 SPC 和 REP 结点以及在第三个译码周期输出的 REP1 和 0SPC 结点，将第三个周期输出的 REP1 计算结果用于后续 REP-SPC 结点，并在第五个周期输出 REP-SPC 结点的返回值。在不同的特殊结点译码周期中，分时复用 SPC4，REP4 等计算模块。

(3)针对存储单元模块消耗过多资源，将信道输入的 LLR 值与译码软输出值共用同一存储单元，根据两种数据所需存储的时间先后不同，对其进行分时存储，降低存储单元的使用数量。此外，将译码过

程中不足存储单元数据宽度的 LLR 阵列通过数据合并存储的方式共用同一存储单元地址，以此来降低存储资源消耗。

1 极化码基本理论

1.1 极化码定义以及构造

极化码(N,K)是大小为$N=2^n$的线性分组码，其中N为码长大小，K为信息位大小。经过信道极化后，从N个信道中会挑选K个信道用于传输信息比特，另外$N-K$个差信道用于传输冻结比特。

对于大小为N的极化码，编码需要使用大小为$N\times N$的生成矩阵$\boldsymbol{G}_N=\boldsymbol{B}_N\boldsymbol{F}^{\otimes n}$，其中$\boldsymbol{B}_N$为比特排序矩阵，$\boldsymbol{F}^{\otimes n}$中：

$$\boldsymbol{F}=\begin{bmatrix}1 & 0\\ 1 & 1\end{bmatrix} \tag{1}$$

$\otimes n$表示n次 Kronecker 内积。将生成矩阵$\boldsymbol{G}_N$与待传输信息U_0^{N-1}相乘，将得到的结果称为码字$\boldsymbol{X}$，并通过信道进行传输。

1.2 SCAN 译码算法

SCAN 译码算法可以看作是 SC 算法和 BP 算法的混合，其调度顺序类似于 SC 算法。然而，SCAN 算法在迭代性方面与 BP 算法很接近，并且传播的都是软值。第i行和第j级的左传播和右传播分别表示为$\lambda_{i,j}$和$\beta_{i,j}$，其中$i\in(0,N-1)$，$j\in(0,n)$。$\lambda_{i,n}$和$\beta_{i,0}$的 LLR 值不需要在译码过程中更新，$\lambda_{i,n}$的值为信道 LLR 值，$\beta_{i,0}$的值取决于结点是否为冻结比特结点，若是，则$\beta_{i,0}=+\infty$，否则$\beta_{i,0}=0$。

通过 Arikan 在文献[1]中提出的因子图，可以有效地进行消息的传播计算，图 1 给出了$N=8$极化码的因子图，其中因子图左侧浅色比特位表示为冻结比特位，深色则表示为信息比特位。SCAN 译码算法通过公式(2)连续地计算λ和β来实现信息传递，如图 2 所示。

$$\begin{aligned}
\lambda_{i,j}&=f(\lambda_{i+1,2j-1},\lambda_{i+1,2j}+\beta_{i,j+N/2})\\
\lambda_{i,j+N/2}&=f(\beta_{i,j},\lambda_{i+1,2j-1})+\lambda_{i+1,2j}\\
\beta_{i+1,2j-1}&=f(\beta_{i,j},\lambda_{i+1,2j}+\beta_{i,j+N/2})\\
\beta_{i+1,2j}&=f(\beta_{i,j},\lambda_{i+1,2j-1})+\beta_{i,j+N/2}
\end{aligned} \tag{2}$$

其中f运算的公式为：

$$f(a,b)=2\tanh^{-1}\left[\tanh\left(\frac{a}{2}\right)\tanh\left(\frac{b}{2}\right)\right] \tag{3}$$

为了降低f运算的运算复杂度，文献[3]提出了一种硬件友好型的近似公式：$f(a,b)=\mathrm{sgn}(ab)\times\min(|a|,|b|)$。当迭代完成后，对得到的 LLR 值进行比特判决，得到译码后的比特判决序列。

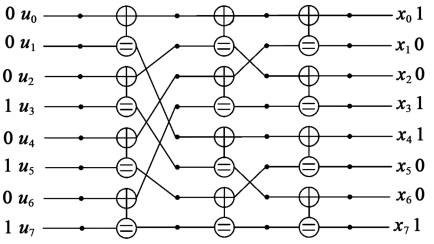


图 1 $N=8$因子图

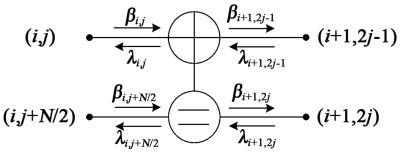


图 2 SCAN 向左与向右信息传递示意图

1.3 结点合并算法

文献[15]提出一种 RLSC 算法，对 SCAN 算法译码延迟较大问题做出改进，通过对译码二叉树的较低层数的结点进行合并，达到对译码二叉树进行剪枝的目的。如图 3 所示，将码长为 4 的左子节点合并为一个 REP 结点，右子结点合并为一个 SPC 结点，从而减少对译码二叉树的遍历过程，减少译码周期的消耗。

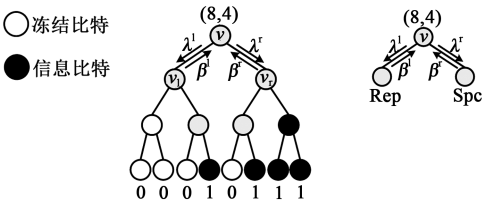


图 3 剪枝前后译码二叉树对比

根据经过极化后的信道分布特性，对于一个码长大小$P_v=8$的结点，其信息比特和冻结比特分布仅有九种可能：FFFFFFFF，FFFFFFFD，FDDDDDDD，FFFFFFDD，FFFFFDDD，FFFDFDDD，FFFDDDDD，FFDDDDDD，DDDDDDDD，这九种结点分别被称为 RATE0 结点、REP 结点、SPC 结点、001 结点、0SPC 结点、REP-SPC 结点、REP1 结点、011 结点、RATE1 结点，各个结点的计算步骤如表 1 所示。

表 1　RLSC 算法中特殊结点计算

Algorithm 1:RLSC LLC8	
Input:λ_t^i,P_v	
Output:β_t^i	
1 The function of the $h(a)$ operation is to take the sign a.	
2 if P_v=FFFFFFF(RATE0)	18 else if P_v=FFFDDDDD (REP1)
3　$\beta_t^i[k]=+\infty$	19 $\beta_t^i[k]=f(\sum_m f(\lambda_t^i[m],\lambda_t^i[m+4]),\lambda_t^i[k]),m=\{0,1,2,3\}/k$;
4 else if P_v=DDDDDDDD(RATE1)	20 else if P_v=FFFFFFFD(REP)
5 $\beta_t^i[k]=0$	21 $\beta_t^i[k]=\sum_{j=0}^{2^t-1}\lambda_t^i[j]-\lambda_t^i[k]$
6 else if P_v= FFFFFFDD(001)	22 else if P_v= FFFDDDDD (0SPC)
7 if(k=even)	23 $\beta_t^i[k]=(\prod h[(\lambda_t^i[m]+\lambda_t^i[m+4])]\times\min\{\lvert(\lambda_t^i[m]+\lambda_t^i[m+4])\rvert\})+$
8　$m=\{0,2,4,6\}/k$	24　$\lambda_t^i[(k+4)\bmod 8],m=\{0,1,2,3\}/k \bmod 4$
9　else	25 else if P_v=FDDDDDDD(SPC)
10 $m=\{1,3,5,7\}/k$	26　$\beta_t^i[k]=(\prod h[\lambda_t^i[n]])\times\min\{\lvert\lambda_t^i[n]\rvert\}$
11 $\beta_t^i[k]=\sum_m\lambda_t^i[m]$	27　$n=(0,1,\cdots,7)/k$
12　else if P_v=FFDFFFDD(011)	28 else if P_v=FFFDFDDD(REP-SPC)
13 if(k=even)	29　for t=0 to 3 do
14 $m=\{0,2,4,6\}/k$	30　$\lambda_1^i[k]=f(\lambda_t^i[t],\lambda_t^i[t+4]),\lambda_r^i[k]=\xi\times\lambda_t^i[t+4]$;
15　else	31　$\beta_1^i[k]=\mathrm{REP}(\lambda_1^i[k]),\beta_1^i[k]=\mathrm{SPC}(\lambda_r^i[k])$
16 $m=\{1,3,5,7\}/k$	32 for t=0 to 3 do
17 $\beta_t^i[k]=\prod h[\lambda_t^i[m]]\times\min\{\lvert\lambda_t^i[m]\rvert\}$	33　$\beta_t^i[k]=f(\beta_1^i[t],\lambda_t^i[t+4]);\beta_t^i[k+4]=f(\beta_1^i[t],\lambda_t^i[t])$

2　高资源效率 Fast-RCSC 译码器的设计

2.1　Fast-RCSC 译码算法

RLSC 译码算法相对于 SCAN 译码算法在译码周期上大幅降低，但是由于在计算 REP-SPC 结点时使用了近似算法改变了结点之间的信息传递关系，造成了较大的译码性能下降。本文设计了一种 Fast-RCSC 算法，在计算特殊结点时保持结点之间的信息传递关系，特别是在计算 REP-SPC 结点时，将其按照左右两部分进行计算，将左子结点作为 $P_v/2$ 大小的 REP 结点，右子结点作为 $P_v/2$ 大小的 SPC 结点，按照 SCAN 译码的运算顺序关系，首先对左侧 REP 结点进行运算，后对右侧 SPC 结点进行运算，其运算过程如表 2 所示。值得注意的是，在对 REP-SPC结点进行计算时，其运算过程中的前半部分与 REP1 结点返回值的上半部分运算过程完全相同，故可直接使用 REP1 结点的返回值。

表 2　Fast-RCSC 算法中 REP-SPC 结点计算

Algorithm 2:Fast-RCSC REP-SPC node calculation
Input: λ_t^i,P_v
Output: β_t^i
1　if P_v= FFFDFDDD,then
2　Take $\beta_t^i[4]\sim\beta_t^i[7]$ of REP1 node as $a_t^i[4]\sim a_t^i[7]$ of REP-SPC node;
3　Set $\eta_t^i[k \bmod 4]=(\prod h[a_t^i[m]])\times\min\{\lvert a_t^i[m]\rvert\},m=\{0,1,2,3\}/k \bmod 4$;
4　if(k<4)
5　$\beta_t^i[k]=f((\eta_t^i[k \bmod 4]+\lambda_t^i[(k+4)\bmod 4]),(\prod(h[\lambda_t^i[m]+\lambda_t^i[m+4]])\times\min\{\lvert\lambda_t^i[m]+\lambda_t^i[m+4]\rvert\}))$;
6　else
7　$\beta_t^i[k]=f(\lambda_t^i[(k+4)\bmod 4]),(\prod(h[\lambda_t^i[m]+\lambda_t^i[m+4]])\times\min\{\lvert\lambda_t^i[m]+\lambda_t^i[m+4]\rvert\}))+\eta_t^i[k \bmod 4]$;

2.2　整体架构

本设计的整体硬件架构如图 4 所示。

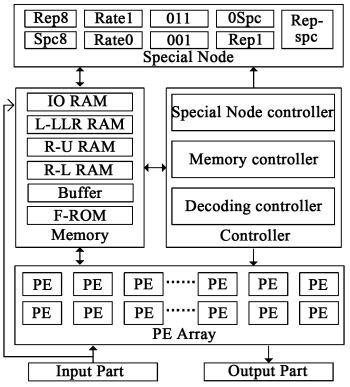


图 4　Fast-RCSC 译码器整体硬件架构

该硬件电路主要由六个模块组成：数据输入模块、数据输出模块、计算阵列模块、数据存储模块、控制模块以及特殊结点模块。计算阵列模块用来执行译码过程中所需的 $f(a,b+c)$和 $f(a,c)+b$ 运算，计算得到的结果送至数据存储模块中的 RAM 或 Buffer 中。特殊结点模块对符合特殊结点设定的 LLR 输入进行运算，将得到的结果送至数据存储模块中的 RAM 与 Buffer 中。数据存储模块用于存储计算模块得到的计算结果，通过控制模块给出的使能信号以及地址信号进行控制。控制模块输出选择信号给特殊结点模块以及计算阵列模块，输出使能信号以及地址信号给数据存储模块，从而控制译码数据的流动。数据输出模块将最后一次迭代计算得

出的 LLR 值与信道初始 LLR 之和进行比特判决，将判决结果和对应冻结比特进行与运算，得到对应的译码结果。

2.2.1 计算阵列模块

计算阵列模块包含 P 个 PE 计算单元，每个 PE 计算单元的框图如图 5 所示，用于执行 $f(a,b+d)$ 与 $f(a,c)+b$ 运算，其中 f 运算执行 $f(a,b)=\mathrm{sgn}(ab)\times\min(|a|,|b|)$。计算阵列模块将输入的四个大小为 PQ 的 LLR 执行运算，并返回两个大小为 PQ 的 LLR 值送至存储单元模块中进行选择存储，其中 Q 为 LLR 值量化位大小。

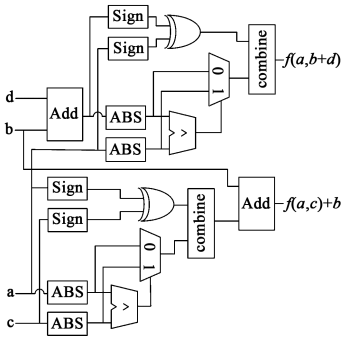


图 5 PE 计算单元

2.2.2 特殊结点模块

特殊结点模块中使用 2.1 节所设计的 Fast-RCSC 算法对特殊结点的计算，其中 RATE0 和 RATE1 结点最大支持的码长为 32，其余七类结点考虑到译码资源消耗，支持的结点码长都为 8。特殊结点模块对于不同的译码结点所需译码周期是不同的，对于 RATE0 和 RATE1 结点，特殊结点模块在输入数据周期就更新返回值 β；对于 011，001 结点，在将 λ 值输入模块的一个周期后将更新返回值 β；对于 REP，SPC 结点，返回值 β 的更新计算则需要两个周期；对于 REP1 及 0SPC 模块，β 值的更新计算则需要三个周期；对于 REP-SPC 结点，为了相对 RCSC 算法译码性能不下降，通过保持结点之间的信息传递关系，将 REP-SPC 结点分为左右两个部分进行顺序运算，共需要五个周期对返回值 β 进行更新计算。

特殊结点模块按照图 6 进行展开译码，可将译码过程分为两部分进行，一部分基于 f 运算，如 SPC4 与 SPC8 模块，另一部分则基于 g 运算（加法运算），如 REP4 与 REP8 模块。由于不同的译码结点中含有相同的基本计算单元，且不同的译码结点得到返回值所需周期也不同，因此可对这些基本计算单元进行分时复用。如图 6 所示，SPC4 模块在第一个周期计算 011 结点，第二个周期计算 0SPC 结点，第四个周期计算 REP-SPC 结点。REP4 模块则在第一个周期计算 001 结点，第二个周期计算 REP1 结点。由八个 f 运算组成的计算单元 $F*8$ 模块，可在第三个周期进行 REP1 结点计算，第五个周期进行 REP-SPC 结点计算。

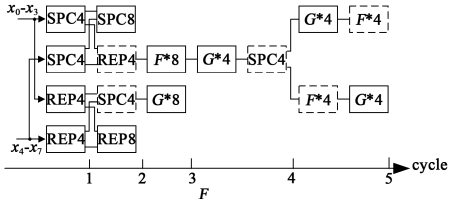


图 6 特殊结点模块框图与周期图

如表 1 和表 2 所示，由于不同的译码结点之间的计算公式存在相似部分，因此可以复用这些相似部分计算结果。如图 7 与图 8 所示，SPC4 模块与 REP4 模块的第一级运算结果分别可被用于计算 REP1 结点以及 0SPC 结点。

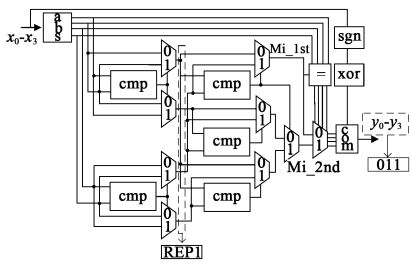


图 7 SPC4 模块

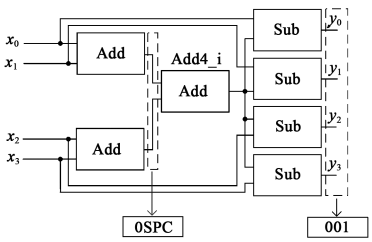


图 8 REP4 模块

SPC4 模块的第二级运算得到的最小值 Mi_1st 以及次小值 Mi_2nd 可被 SPC8 模块用于计算 SPC 结点，如图 9 所示。REP4 模块的第二级运算得到的输入值之和 Add4_i 可被 REP8 模块用于计算 REP 结点，如图 10 所示。SPC4 模块第三级运算结果 $y_0\sim y_3$ 可直接作为 011 结点的输出结果，REP4 模块第三级运算结果 $y_0\sim y_3$ 可直接作为 001 结点的输出结果。而在计算 REP-SPC 结点时，由 Fast-RCSC 译码算法可知，可直接复用 REP1 结点的计算结果，无需重新计算这部分计算结果。

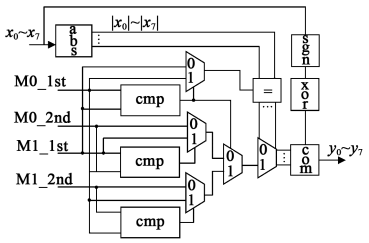


图 9　SPC8 模块

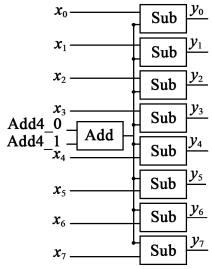


图 10　REP8 模块

2.2.3　存储单元模块

存储单元模块中包含输入输出存储单元(IO RAM)、λ 值存储单元(L-LLR RAM)、混合 β 值存储单元(R_U RAM)、根结点的右子结点 β 值存储单元(R_L RAM)、结点信息存储单元(F-ROM)以及 Buffer 单元。除了 Buffer 与 F-ROM 外每个存储单元均由上下两部分组成，每个存储部分的数据宽度大小为 PQ。其中 IO RAM 用于存储来自信道的初始 LLR 值和译码得到的软输出值；L-LLR RAM 用于存储向左运算过程中得到的 λ 值；R_U RAM 用于存储向右运算过程中除根结点的右子结点外的所有 β 值；R_L RAM 则用于存储向右运算过程中根结点的右子结点 β 值；F-ROM 单元用于存储各个结点信息，包括结点所在层数、左右结点信息、冻结比特信息以及结点执行的运算类型；若当前译码结点的 LLR 值个数不超过 P 个，则来自 PE 阵列模块或特殊结点模块的 LLR 值可通过 Buffer 进行缓存，在下一个周期直接送入 PE 阵列模块或特殊结点模块进行计算。

在存储单元中，译码二叉树上的不同结点所占深度是不同的，若码长 N 为 1 024，计算并行度 P 的大小为 64，则第九层(dv9)结点的 λ 值在 L-LLR RAM 中上下两个部分所占深度都为 4，第八层(dv8)结点的 λ 值在 L-LLR RAM 中上下两个部分所占深度都为 2。为了降低存储单元的所需深度，采用了一种拼接存储的方式来对不足 PQ 大小的 LLR 阵列进行存储，如图 11 所示。每个结点对应的 LLR 阵列分为两部分送往同一存储单元的上下部分中，对于计算并行度 P 大小为 64 的译码器，可将译码二叉树的第六层(dv6)中的 32 个 LLR 值、第五层(dv5)中的 16 个 LLR 值、第四层(dv4)中的 8 个 LLR 值以及第三层(dv3)中的 4 个 LLR 值拼接在同一存储单元的地址中，减少存储单元所需深度，避免将各层 LLR 数据分别存于不同的地址单元中，造成存储资源的过度消耗。

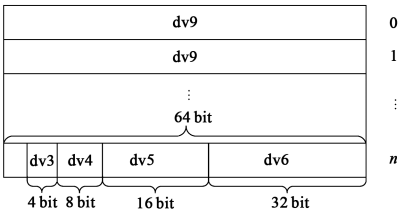


图 11　不足位宽大小的 LLR 值合并存储

由于在得到软输出值后信道输入值不再需要存储，可以使用一个共用的存储单元去存储信道输入值与软输出值。IO RAM 在译码开始后将信道输入 LLR 值进行暂存，在达到译码终止的条件后则将得到的软输出值覆盖原先存储的信道输入 LLR 值进行存储，从而降低存储 LLR 值所需的存储资源。

若存储信道输入初始 LLR 值以及软输出值所需存储总数为 N，存储向左运算得到的 LLR 值所需的存储总数为 $N-8$(当结点码长为 8 时输入到特殊结点模块直接返回 β 值，故总数需要减去 8)，存储向右运算过程中左子结点的 β 值所需存储总数为 $N-8$。因 Fast-RCSC 算法与 RCSC 算法类似，在向左运算中计算根结点的左子节点时，除根结点外的其余结点计算都无需 β 值，故在向右运算过程对右子结点 β 值存储时仅需存储译码二叉树中根结点的右子结点 β 值即可，其余右子结点的 β 值可以直接与向右运算过程中的左子结点 β 值共用同一存储单元，所以在向右运算中右子结点的 β 值所需存储总数为 $N/2$ 即可。

综上所述，若译码器的码长为 N、计算并行度大小为 P，则存储模块在迭代运算过程中所需存储 LLR 总数为 $7/2N-16$，其中 L-LLR RAM 与 R_U RAM 存储单元都有 $W=\sum_{t=\log_2 P}^{n}(2^{n-t}/P)$ 个字、IO RAM 存储单元共有 N/P 个字、R_L RAM 存储单元共有 $N/2P$ 个字，每个字对应存储 P 个 LLR 值。

3　实验与结果分析

3.1　实验目的与配置

在本节中，分别对 Fast-RCSC 算法以及高资源

效率 Fast-RCSC 译码器硬件架构性能进行评估，通过误帧率（Frame Error Rate，FER）来评估 Fast-RCSC 算法译码性能，从最高工作频率、面积、周期、延迟、吞吐率以及资源效率等指标来对高资源效率 Fast-RCSC 译码器硬件架构进行评估。

对 Fast-RCSC 算法评估部分采用的 PC 型号为 Intel(R) Core(TM) i5-9300H CPU@2.40 GHz，内存大小为 24 GB，仿真采用软件 MATLAB2014b，在 AWGN 信道下进行仿真，使用高斯近似法对信道可靠性进行排序，调制方式使用 BPSK 调制，仿真采用的码长 $N=1\,024$，信息比特长度 $K=512$，所采用的 SCAN 算法、Fast-RCSC 算法、RLSC 算法的迭代次数都为 2 次，BP 算法迭代次数为 50 次。

对高效率 Fast-RCSC 译码器硬件架构评估部分设计的码长为 $N=1\,024$，信息比特长度为 $K=512$，计算并行度即 PE 计算单元个数为 64，量化位宽 Q 的大小为 6，用 1 bit 表示小数部分，译码器的输入 LLR 数据来自 AWGN 信道，对应信噪比大小为 4 dB，在 CSMC（Central Semiconductor Manufacturing Corporation）180 nm 工艺下，使用 Synopsys 的 DC(Design Compiler)对电路进行综合。

3.2 Fast-RCSC 算法译码性能实验结果

本文所使用的 Fast-RCSC 算法得到的 FER 结果如图 12 所示。由于本文采用的 Fast-RCSC 算法相对于 RLSC 算法在迭代过程中没有采用 MNA 近似算法，因此相对于 RLSC 算法有着最高达到 0.5 dB 的译码性能增益。相较于 SCAN 算法，由于 Fast-RCSC 算法在向左子结点迭代运算时简化了运算过程，因此其译码性能会产生约 0.1 dB 的译码损耗。相较于文献[17]的 BP 译码，Fast-RCSC 算法在信噪比水平较高时会有更好的译码性能。相较于 RCSC 算法只加速了其迭代过程，没有改变迭代中数据大小，因此与 RCSC 拥有相同的译码性能。

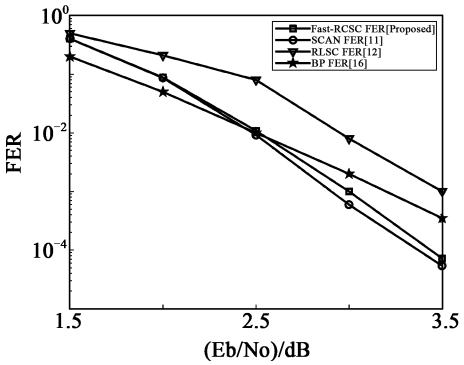


图 12 （1 024，512）极化码软输出算法性能对比

3.3 高效率 Fast-RCSC 译码器硬件架构综合结果

本文所设计的高效率 Fast-RCSC 译码器硬件架构综合结果如表 3 所示，表中对比对象均为能够输出软输出的译码器。其中，文献[14]提出的传统 SCAN 译码架构需要完整遍历全部译码二叉树，因此所需译码周期较长。文献[15]提出的 RCSC 译码架构的译码二叉树仍比较大，需要遍历的周期仍比较长，且资源效率比较低。文献[15]提出的 RLSC 译码架构对译码二叉树进行了剪枝，并通过 MNA 近似算法减少了译码周期数，但是仍存在译码性能较差以及资源消耗较高的问题。本文设计的译码器硬件架构通过一种基于计算结果复用和计算单元分时复用的特殊结点模块以及对译码器中存储单元进行存储优化的方式来减少面积上的消耗，将不同的特殊结点返回值在不同的特殊结点译码周期中输出，节约时钟周期上的消耗。综合结果与转换为 180 nm 的参考文献[14-15，18-19]相比，在面积上分别减少了 53.4%，24.7%（RCSC），35.8%（RLSC），67.2%，81.6%。与文献[14-15，18]吞吐率分别提升了 529.1%，51.5%（RCSC），53.9%。与文献[14-15，18-19]相比资源效率分别提升了 1 251%，101.3%（RCSC），10.7%（RLSC），369.8%，147.2%。

表 3 各软输出译码器在 180 nm 工艺下综合结果对比

参数	本文	DASIP'15[14]	TVLSI'16[15]		ISCAS'21[18]	TCASII'20[19]
Algorithm	Fast-RCSC	SCAN	RCSC	RLSC	Fast-SCANF	BP
Technology/nm	180	180(40) *◇	180(90)	180(90)	180(40) *	180(65)
Max. Freq/MHz	294.7	236.4(1 064)	285.5(571)	225(520)	217.8(980)	134.7(373)
Area/mm^2	2.92	6.27(0.31)	3.88(0.97)	4.55(1.138)	8.91(0.44)	15.87(2.07)
Cycle	422	2 112	610	250	454	—
Latency/μs	1.43	8.93(1.98)	1.08(2.16)	0.51(1.02)	—	—
Coded (T/P)/Mbps	716	113.8(521)	472.5(945)	1 007.5(2 015)	465.1(2 093)	1 574(4 360)
AE/(Mbps/mm^2)	245.2	18.14(1 680)	121.77(947)	221.4(1 770)	52.19(4 756)	99.18(2 106)

* 统一化至 180 nm 工艺基于以下标准：Area$\propto 1/Z^2$，Max. Freq$\propto Z$，其中 Z 为 180 nm 与相应被统一化工艺的比值大小。

◇ 文献[14]给出 FPGA 综合结果，文献[14]给出其在 40 nm 工艺下的 DC 结果。

通过以上对本文所设计的高资源效率 Fast-RCSC 译码器的实验结果分析可知，对比其他软输出译码器，本文所设计的译码器在频率、吞吐率没有显著降低的情况下，降低了资源使用量，提升了资源

效率，也验证了本文优化的 Fast-RCSC 译码器设计方案的可行性。

4　结束语

本文通过分析 SCAN 类型算法，从 RLSC 译码算法、特殊结点模块设计以及存储单元模块的存储方案进行优化的角度出发，设计了一种 Fast-RCSC 译码算法及一种高资源效率的 Fast-RCSC 译码器。通过将 REP-SPC 结点的计算分为左右两部分去进行完整计算，使得在加速 RCSC 算法迭代的同时不产生译码性能下降。通过对特殊结点模块中的计算结果进行复用以及对计算模块进行分时复用，减少特殊结点模块的资源消耗。通过将存储单元中信道输入的 LLR 值与软输出值共用同一存储单元以及对不足存储单元数据宽度的 LLR 阵列进行合并存储的方式降低存储资源消耗。最终本文设计了码长为 1 024，信息位为 512 的译码器，并在 CSMC 180 nm 标准工艺下完成电路综合。结果表明该译码器对比现有软输出译码器有着较低的译码资源消耗和较高的资源效率，使得译码器更具有实用价值。

参考文献：

[1] Arikan E. Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels [J]. IEEE Trans. on Information Theory, 2009, 55(7): 3051-3073.

[2] Şaşoglu E, Telatar E, Arikan E. Polarization for arbitrary discrete memoryless channels[C]// 2009 IEEE Information Theory Workshop, 2009: 144-148.

[3] Zhang M, Jiang X Q, Huang Y. Secret key generation with signal-like noise in polar coded wireless MIMO systems[C]// 2019 Inter. Conf. on Information and Communication Technology Convergence (ICTC). IEEE, 2019: 204-206.

[4] Li L, Xu Z, Hu Y. Channel estimation with systematic polar codes[J]. IEEE Trans. on Vehicular Technology, 2018, 67(6): 4880-4889.

[5] Chen K, Niu K, Lin J. A hybrid ARQ scheme based on polar codes[J]. IEEE Commun. Lett., 2013, 17(10): 1996-1999.

[6] Yuan B, Parhi K K. Architectures for polar BP decoders using folding[C]// IEEE Inter. Symp. on Circuits and Systems (ISCAS), 2014: 205-208.

[7] Fayyaz U U, Barry J R. Low-complexity soft-output decoding of polar codes[J]. IEEE J. on Sel. Areas in Communications, 2014, 32(5): 958-966.

[8] Eslami A, Pishro-Nik H. On finite-length performance of polar codes: stopping sets, error floor, and concatenated design[J]. IEEE Trans. on Communications, 2013, 61(3): 919-929.

[9] Xu J, Che T, Choi G. XJ-BP: Express journey belief propagation decoding for polar codes[C]// 2015 IEEE Global Communications Conf. (GLOBECOM), 2015: 1-6.

[10] Park Y S, Tao Y, Sun S, et al. A 4.68 Gb/s belief propagation polar decoder with bit-splitting register file[C]// 2014 Symp. on VLSI Circuits Digest of Technical Papers. IEEE, 2014: 1-2.

[11] Lopacinski L, Hasani A, Panic G, et al. Ultra high-speed BP decoder for polar codes achieving 1.4 Tbps in 28 nm CMOS[C]// 2022 Joint European Conf. on Networks and Communications & 6G Summit (EuCNC/6G Summit). IEEE, 2022: 434-439.

[12] Ibrahim Z, Fahmy Y. Enhanced learning for recurrent neural network-based polar decoder[C]// 2022 13th Inter. Conf. on Electrical Engineering (ICEENG). IEEE, 2022: 105-109.

[13] 袁建国，方小倩，梁栩珩，等. 一种基于均匀量化的快速简化极化码 SC 译码算法[J]. 半导体光电，2021，42(1)：127-131.
Yuan Jianguo, Fang Xiaoqian, Liang Xuheng, et al. A fast simplified SC decoding algotithm of polar codes based on uniform quantization [J]. Semiconductor Optoelectronics, 2021, 42(1): 127-131.

[14] Berhault G, Leroux C, Jego C, et al. Hardware implementation of a soft cancellation decoder for polar codes [C]// 2015 Conf. on Design and Architectures for Signal and Image Processing (DASIP). IEEE, 2015: 1-8.

[15] Lin J, Yan Z, Wang Z. Efficient soft cancelation decoder architectures for polar codes[J]. IEEE Trans. on Very Large Scale Integration (VLSI) Systems, 2016, 25(1): 87-99.

[16] Wang X, Gu R, Li J, et al. Improved reduced latency soft-cancellation algorithm for polar decoding [J]. China Communications, 2020, 17(5): 65-77.

[17] Yuan B, Parhi K K. Early stopping criteria for energy-efficient low-latency belief propagation polar code decoders [J]. IEEE Trans. on Signal Processing, 2014, 62(24): 6496-6506.

[18] Zhang L, Sun Y, Shen Y, et al. Efficient fast-SCAN flip decoder for polar codes[C]// 2021 IEEE Inter. Symp. on Circuits and Systems (ISCAS), 2021: 1-5.

[19] Shen Y, Song W, Ren Y, et al. Enhanced belief propagation decoder for 5G polar codes with bit-flipping[J]. IEEE Trans. on Circuits and Systems Ⅱ: Express Briefs, 2020, 67(5): 901-905.

作者简介：

陶玉辉(1999—)，男，安徽省芜湖市人，硕士研究生，研究方向为极化码译码器设计；

王晓蕾(1978—)，女，安徽省合肥市人，副研究员，研究方向为集成电路设计。