

串行 SCL 极化码译码器

刘丽华^{1,2}, 管 武¹, 梁利平¹

(1 中国科学院 微电子研究所, 北京 100029; 2 中国科学院大学, 北京 100029)

摘 要: 基于列表的极化码串行抵消译码算法(SCL 算法)可以改善中短码长的误码性能,但其递归结构大大降低了译码吞吐率,但同时也带来了大的硬件复杂度和硬件资源消耗. 本文提出了非递归结构的基于似然比的列表串行抵消译码算法(LLR-SCL 算法),设计了码长为 1 024 比特、搜索路径为 2 的 LLR-SCL 译码器. 仿真测试表明,该译码器具有较好的误码性能,且在 Xilinx XC7V2000 FPGA 上主频可以达到 227 MHz,占用硬件资源较低,复杂度小.

关键词: 极化码 SCL 译码器; 串行; 资源消耗低; FPGA 实现

中图分类号: TP302

文献标识码: A

文章编号: 1000—7180(2018)12—0064—06

A Serial Successive-Cancellation List Decoder of Polar Codes

LIU Li-hua^{1,2}, GUAN Wu¹, LIANG Li-ping¹

(1 Institute of Microelectronics of the Chinese Academy of Sciences, Beijing 100029, China;

2 University of Chinese Academy of Sciences, Beijing 100029, China)

Abstract: The successive cancellation list (SCL) algorithm can improve the decoding performance of polar codes with short and moderate code lengths. However the recursive structure results in the lower throughput, large hardware complexity and cost. A LLR-SCL arithmetic based non-recursive structure is proposed. And a LLR-SCL decoder of length 1 024 and list size $L = 2$ is designed. The simulation result shows the error performance is good. Also, the proposed LLR-SCL decoder is implemented under Xilinx XC7V2000 FPGA. The synthesis result shows that the utilization rate of hardware resources is low and the frequency is up to 227 MHz.

Key words: polar codes SCL decoder; serialization; less hardware cost; FPGA implementation

1 引言

自香农提出信道容量的概念及好码的存在性定理以后,不同可以接近信道容量的码相继被提出^[1-2]. 由 Arikan 在 2009 年提出的极化码是唯一一种被理论证明可以达到信道容量的纠错码,它具有简单的编译码结构^[3],受到广泛研究学者的青睐. 由于极化码的译码远比编码复杂,因此,国内外学者主要对极化码的译码算法及实现进行研究. Arikan 最早将 BP 算法应用于极化码译码^[4]中,但 BP 算法计算过于复杂且不易硬件实现. 2009 年, Arikan 在提出极化码的同时提出串行抵消 SC 译码算法^[3],这

种算法比 BP 算法计算简单,能获得较好的误码性能,易于硬件实现,但该算法只在码长趋于无限长时可达香农极限,在码长为中等长度或有限长时性能并不理想. 针对 SC 算法的不足,学者后续提出多种改进算法,如基于列表的 SC 算法(SCL)^[5-6],基于堆栈的 SC 算法(SCS)^[7]等. 基于 SCL 算法和 SCS 算法,又提出混合的 SC 算法(SCH)^[8],这种算法可通过灵活配置实现 SCL 算法和 SCS 算法的灵活切换. 为了进一步提高极化码中短码长的译码性能,在这些改进算法的基础上,又出现了 CRC 辅助的 SCL 算法(CRC-SCL)以及 CRC 辅助的 SCS 算法(CRC-SCS)^[9]. 极化码译码算法的硬件实现一直以来都是

一大难点,目前主要有基于 SC 算法的 SC 译码器,基于 SSC 算法的 SSC 译码器,基于 SCL 算法的 SCL 译码器以及 CRC-SCL 译码器. 2011 年, Leroux 等人最先提出 SC 译码算法的硬件实现,提出极化码 SC 译码器的 FFT 结构,流水线树形结构,线型 SC 结构以及基于向量重叠的 SC 结构^[10],但这篇文章只是介绍了 SC 算法实现的多种结构,并没有给出硬件实现结果. 2012 年, Bin Li 等人提出一种自适应的 CRC-SCL 译码器^[11]. 这种自适应的 SCL 译码器通过迭代计算增加搜索路径 L 直到至少一条路径可以通过 CRC 校验,可以显著降低复杂度,并且性能不会受到大的影响. 2013 年, Leroux 等人提出半并行的极化码 SC 译码器结构^[12],该结构大大降低了硬件处理复杂度,但内存复杂度没有改善. 2014 年,东南大学和南京大学实现 $L=2, N=1024$ 比特基于 SCL 算法的极化码译码器^[13],与传统译码器相比,该译码器性能提高,硬件消耗更少,并且延时更低. 同年, Jun Lin 等人提出 CRC 辅助的 SCL 算法的硬件实现结构并进行 ASIC 实现,提升了硬件效率^[14]. 2015 年, Alexios 等人提出基于 LLR 的极化码 SCL 译码器^[15],该结构电路更加规整且需要更少的内存资源,并且大大提高了吞吐率.

本文基于非递归结构的 LLR-SCL 算法,实现了串行结构的极化码 SCL 译码器.

2 极化码译码原理

极化码译码算法本质上是树搜索算法. 极化码最原始的译码算法为接续删除(SC)算法,其他算法由 SC 算法改进而来. 本章将介绍 SC 译码算法^[3]以及基于 LLR 的 SCL 译码算法^[15]原理.

SC 译码算法的主要原理是根据 (y_1^N, u_{A^c}) 来得到 u_1^N 的一个估计 $\hat{u}_1^N$. 其中, y_1^N 为信道输出; u_{A^c} 为固定比特集; u_1^N 为传输的信息比特集; N 为码长. 该算法的译码过程可以看成由 N 个决策元(DEs) 构成,若当前比特为固定比特,即为已知比特,则直接赋值 $\hat{u}_i = u_i$,并将此估计值传送给其他所有决定元,以备使用;若当前比特为信息比特,则需要根据信道输出以及之前估计出的所有比特值来计算该比特的似然比 LR ,从而对该比特进行估计,计算似然比如公式(1)所示.

$$L_N^{(i)}(y_1^N, \hat{u}_1^{i-1}) \triangleq \frac{W_N^{(i)}(y_1^N, \hat{u}_1^{i-1} | 0)}{W_N^{(i)}(y_1^N, \hat{u}_1^{i-1} | 1)} \quad (1)$$

式中, $W_N^{(i)}(y_1^N, \hat{u}_1^{i-1} | 0)$ 表示当前比特为 0,经过信道传输得到 $(y_1^N, \hat{u}_1^{i-1})$ 的概率. 则所有比特的决策法

则如公式(2)所示.

$$\hat{u}_i = \begin{cases} 0, & \text{if } L_N^{(i)}(y_1^N, \hat{u}_1^{i-1}) \geq 1 \\ 1, & \text{otherwise} \end{cases} \quad (2)$$

似然比的计算可以通过递归公式(3)和(4)直接计算得到, $N=8$ 时的译码迭代结构图如图 1 所示.

$$L_N^{(2i-1)}(y_1^N, \hat{u}_1^{2i-2}) = \frac{L_{N/2}^{(i)}(y_1^{N/2}, \hat{u}_{1,o}^{2i-2} \oplus \hat{u}_{1,e}^{2i-2}) L_{N/2}^{(i)}(y_{N/2+1}^N, \hat{u}_{1,e}^{2i-2}) + 1}{L_{N/2}^{(i)}(y_1^{N/2}, \hat{u}_{1,o}^{2i-2} \oplus \hat{u}_{1,e}^{2i-2}) + L_{N/2}^{(i)}(y_{N/2+1}^N, \hat{u}_{1,e}^{2i-2})} \quad (3)$$

$$L_N^{(2i)}(y_1^N, \hat{u}_1^{2i-1}) = [L_{N/2}^{(i)}(y_1^{N/2}, \hat{u}_{1,o}^{2i-2} \oplus \hat{u}_{1,e}^{2i-2})]^{1-2\hat{u}_{2i-1}} \cdot L_{N/2}^{(i)}(y_{N/2+1}^N, \hat{u}_{1,e}^{2i-2}) \quad (4)$$

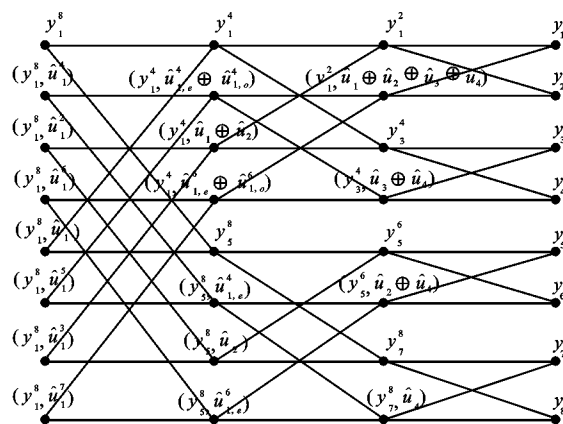


图 1 SC 译码迭代结构图

基于 LLR 的列表译码算法是在 SCL 算法的基础上增加了路径衡量值的计算,其原理如下. SCL 算法的译码可以看成是对一个满二叉树按广度进行搜索,是在原始的 SC 算法的基础上把每个估计比特考虑 0 和 1 两种情况,而不是单单考虑 0 或 1. 抽象地讲, LLR-SCL 算法包括构造译码树,路径扩展,路径衡量值计算三大部分. 首先,构造译码树即为构造一个满二叉树,如图 2 所示;树的每一层对应一个要估计的比特,当该比特为信息比特时将同时考虑 0 和 1 两种情况,即进行路径扩展,图中红线对应为一条路径. 随着码长不断增加,译码树将随码长成指数增加,为了降低计算复杂度,于是提出树的剪枝算法. LLR-SCL 算法提出对每条路径计算衡量值,选取其中衡量值最大的 L 条路径作为候选路径,从而节约空间,降低计算复杂度. 路径衡量值(PM)定义为该路径所对应的译码序列的概率,实现时往往采用其对数形式,计算如公式(5)所示.

$$\text{PM}(u_1^i) = \ln(\text{Pr}\{u_1^i | y_1^N\}) \quad (5)$$

路径衡量值 PM 按以下公式递归地计算得到:

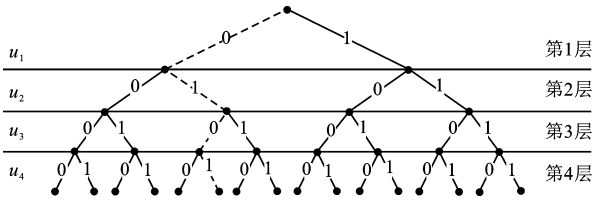


图 2 递归 SCL 算法译码过程图

$PM(u_i^t) =$

$PM(u_i^{t-1})$, 若 u_i 为信息比特或正确固定比特,

且 $(1-2u_i) = \text{sign}(L_N^{(i)}(y_1^N, \hat{u}_1^{t-1}))$

$PM(u_i^{t-1}) - |L_N^{(i)}(y_1^N, \hat{u}_1^{t-1})|$, 若 u_i 为信息比特或正

确固定比特, 且 $(1-2u_i) \neq \text{sign}(L_N^{(i)}(y_1^N, \hat{u}_1^{t-1})) - \infty$, 若

u_i 为固定比特, 且取值错误

(6)

由于这种结构, 需要递归的进行译码路径搜索, 复杂度大, 吞吐率低, 不适合硬件实现.

3 改进的极化码 SCL 译码器

3.1 译码器算法改进

由于原始的 SCL 算法为递归结构, 不利于硬件实现, 因此, 本文首先将递归算法改进为非递归算法, 同时保持空间复杂度为 $O(N)$ 不变. 非递归算法的改进主要针对 LLR 和部分和的递归计算, 将由外向里递推再向外一层层递归回来的计算模式改为直接由里向外计算, 节约计算时间, 更利于硬件实现. 同时, 由于计算 LLR 的递推公式(3)和(4)太过抽象, 且不适合硬件实现, 因此将其转换到对数域, 利用最小和近似算法^[10]对公式简化. 其简化公式如下:

$$f(L_a, L_b) = 2 \tanh^{-1} \left(\tanh \left(\frac{L_a}{2} \right) \tanh \left(\frac{L_b}{2} \right) \right) \quad (7)$$

$$g_{\hat{u}_s}(L_a, L_b) = L_a (-1)^{\hat{u}_s} + L_b \quad (8)$$

式中, $L_a = L_{N/2}^{(i)}(y_1^{N/2}, \hat{u}_{1,o}^{2i-2} \oplus \hat{u}_{1,e}^{2i-2})$, $L_b = L_{N/2}^{(i)}(y_{N/2+1}^N, \hat{u}_{N/2+1,o}^{2i-2})$, 式(3)就被简化为式(7), 式(4)简化为式(8), 相应的迭代关系也变成了 f 和 g 函数. 由于式(7)中的 f 函数为超越函数, 不易硬件实现, 因此进一步将其简化为

$$f(L_a, L_b) \approx \text{sign}(L_a) \text{sign}(L_b) \min(|L_a|, |L_b|) \quad (9)$$

式(8)和式(9)就被称为极化码译码的最小和算法公式, 其可以直接由比较器和加法器实现, 避免了使用乘法器和除法器比较复杂的电路.

3.2 译码器结构

该译码器主要包括 LLR 计算模块, 路径选择与更新模块, 部分和更新模块, RAM 模块, 寻址与地址更新

模块以及控制模块, 其中, LLR 计算模块, 路径选择与更新模块和部分和更新模块为三大主模块.

LLR 计算模块主要是计算每个要译码比特的对数似然比; 部分和更新模块的功能为更新 LLR 计算模块所需要的部分和比特; 路径选择与更新模块主要实现路径的扩展、删除和复制, 并更新译码后的比特估计和路径衡量值; RAM 模块主要存储每层中 LLR 和部分和的计算结果, 以及存放已经完成译码的信息比特和路径衡量值; 寻址与地址更新模块的功能为根据当前的活跃路径和层数寻找对应寄存器的地址, 并更新活跃路径个数及路径与地址的对应关系, 以及决定 RAM 中的信息是否需要复制; 控制模块主要控制 LLR 计算模块、部分和更新模块以及路径选择与更新模块的工作顺序, 以及 RAM 模块内部信息的交互, 保证译码器有条不紊的运行. 整个译码器的结构框图如图 3 所示.

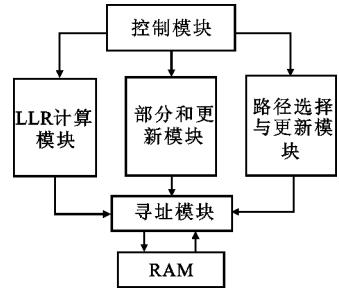


图 3 串行 SCL 译码器结构框图

3.2.1 LLR 计算模块

LLR 计算模块主要分为参数更新模块, llr_lambda 计算模块, 循环控制单元, 以及计数器单元, 其中循环控制单元需要触发寻址与地址更新模块, 从而对 RAM 模块进行读写数据; 同时寻址与地址更新模块又将触发控制模块决定是否进行复制 RAM 模块中的信息, 从而保证 llr_lambda 计算结果的正确性. LLR 计算模块的结构框图如图 4 所示.

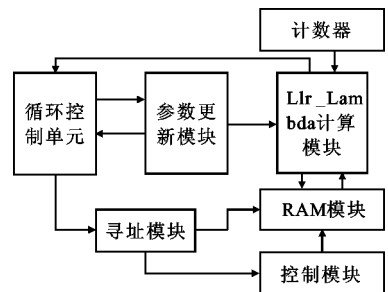


图 4 LLR 计算模块结构框图

参数更新模块主要包括层数 lambda, 分支数

phi 以及当前活跃路径 l 等参数的更新. 由循环控制单元根据当前译码比特 phi_i 以及活跃层数 r 更新需要计算的当前层数 lambda 及分支数 phi 的值, 由寻址与地址更新模块给出当前活跃路径 l. 由于本文提到的译码器只涉及两条路径, 因此这里计算 LLR 时采用串行方式计算, 节约了寄存器资源并使得电路可重复利用.

参数更新完成以后, 由寻址与地址更新模块返回需要读写的 RAM 地址, 并将 RAM 模块中的数据输出到 llr_lambda 计算模块. 同时 llr_lambda 计算模块根据当前的层数 lambda 确定该层需要计数的 beta 值, 计数器开始计数. 该层计算完成后, 通知循环控制单元进行新一轮的参数更新或结束计算. 由于路径的删除及复制, RAM 模块中的值应该被更新, 因此, 由寻址与地址更新模块触发控制模块确定是否进行更新 RAM 中的值. 所谓数据更新, 即是数据的赋值, 这里采用 lazy-copy 算法, 只复制当前层的前一层 lambda-1 的地址, 然后寻址进制复制数据. 若需要更新, 则待更新完成后再输出到 llr_lambda 计算模块中进行计算.

llr_lambda 计算模块主要完成 f 和 g 函数的选择和计算, 由分支数的最后一位 phi[0] 来控制 f 和 g 函数的切换. 当 phi[0] 为 1 使选择 g 函数, 否则选择 f 函数. f 和 g 函数由对数域的最小和近似算法实现, 在不降低极化码译码性能的基础上简化了电路, 降低了硬件复杂度.

3. 2. 2 路径选择与更新模块

包括 prob 计算与更新模块、比较模块、以及 kill_clone 模块, 并由寻址与路径更新模块对 RAM 模块中的 info, c_lambda10 以及 PM 进行更新. 其结构框架图如图 5 所示.

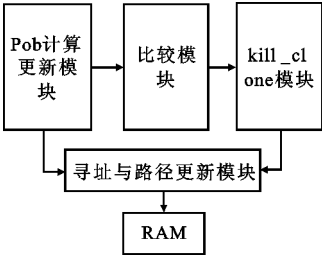


图 5 路径选择与更新模块

如图 5 所示, 在 LLR 计算模块完成后, 路径选择与更新模块首先进行路径扩展, 即根据得到的 llr 值及之前得到的 PM 值计算当前状态下每条扩展路径的概率值 prob. 然后通过一个比较器对该

比特是否为固定比特进行判断, 若该比特为固定比特, 则直接进入寻址与路径更新模块, 对 RAM 模块中的 info、c_lambda10、和 PM 进行更新; 若为信息比特, 则进入比较模块, 比较得到的 prob 值计算阈值 threshold, 由于本文实现的是 2 条路径的极化码 SCL 译码器, 因此阈值 threshold 即为 4 个 prob 值中第二大的值. 显而易见, 大于等于阈值的 prob 值对应的路径即为存活路径, 其他路径则删除. 若存活的两条路径为同一条路径扩展来的, 则将其其中一条扩展路径应该转移到另一条路径下面, 同时将这条扩展路径的所有相关信息复制到另一条路径中, 以上描述即为 kill_clone 模块的功能. 若存活的两条路径为两条不同路径扩展得到的, 则直接进入寻址与路径更新模块, 对 RAM 模块中的 info、c_lambda10、和 PM 进行更新.

3. 2. 3 部分和更新模块

部分和更新模块采用与 LLR 计算模块相同的结构, 只是其中的 llr_lambda 计算模块替换成 c_lambda 计算模块, 并且参数更新模块中的参数又增加了参数 psi, 用于控制部分和结果的输出地址. 该模块因为要对已经估计出的比特进行部分求和, 因此在计算时, 先从最高层进行读取数据计算, 然后向低层进行计算, 由循环控制单元控制参数的更新及计算结束. 并且, 该模块只更新除最高层外其它层的部分和结果, 最高层中的部分和数据由路径选择与更新模块进行更新. 该模块的结构框图参照图 4.

3. 2. 4 RAM 模块

RAM 模块存储两条路径所有层中的 LLR 值, 部分和 c 值, 以及两条路径的比特估计值 info 和 PM 值. 对于存放 LLR 和 c 的 RAM 块, 每条路径每一层都对应一个 RAM. 经量化处理, 接收的信道 LLR 为 6 比特, 中间 LLR 为 8 比特, 路径衡量值为 16 比特. 为了方便计算, 存放中间 LLR 和 c 的 RAM 位宽分别为 16 比特和 4 比特, 即一个地址包括 2 个 LLR 和 2 个 c . 由于采用空间复杂度低的算法, 随着层数 lambda 的增加, RAM 的长度在以 2 的倍数缩减. 例如在本文提到的 SCL 译码器中, 当 lambda 为 0 时, 存放 LLR 的 RAM 长度为 512; 当 lambda 为 1 时, 存放 LLR 的 RAM 长度为 256; 以此类推, 当 lambda 为 8 时, 存放 LLR 的 RAM 长度为 2, 因此, 该 RAM 模块不包括 llr_lambda9 和 llr_lambda10. 由此可以看出, 本文提出的译码器空间复杂度仅为 $O(N)$, 相比于原始的 SC 算法的空间复杂度 $O(N \cdot \log_2 N)$ 大大降低, 有效的节

约了硬件资源.

3.2.5 寻址与地址更新模块

该模块是三大主模块 LLR 计算模块,路径选择与更新模块和部分和更新模块以及控制模块与 RAM 连接的桥梁,由一系列与路径和地址相关的寄存器构成. 这些寄存器主要表示路径活跃情况 (active_path), 路径与地址的对应情况 (pathIndexToArrayIndex) 以及 RAM 中每个地址对应的数据被包含的路径个数 (arrayReferenceCount). 寻址功能是读取寄存器 pathIndexToArrayIndex 中的数值, 根据活跃路径的值得到需要的数据的地址的值; 数据的更新主要是路径选择与更新模块中路径的 kill 和 clone 都会导致寄存器中数据的变化.

3.2.6 控制模块

该模块主要控制三大主模块 LLR 计算模块,路径选择与更新模块和部分和更新模块的工作顺序, RAM 模块内部信息的传递以及 phi_i 更新, 由一系列的选择器和计数器构成, 如图 6 所示.

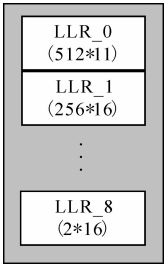


图 6 RAM_LL2R 结构图

4 性能分析

本文对码长为 1 024 比特, 码率为 0.5, 路径数目为 2 的译码器进行实现, 采用 6 比特量化信道 LLR, 8 比特量化中间 LLR, 16 比特量化路径衡量

值, 图 7 为量化前后的译码性能对比图. 其中, 上面两条线为量化后的误比特率和误码率, 下面两条线为浮点型的误比特率和误码率, 可以看出实现的译码器的性能与浮点型性能相差不多.

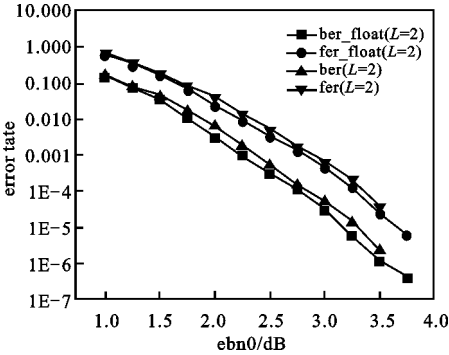


图 7 1 024 比特 SCL 译码性能图

4.1 FPGA 功能仿真结果

本文提出的串行 SCL 极化码译码器首先用 C++ 编程实现算法, 并在 vivado 软件中进行行为级仿真, 验证其结果的正确性. 其功能仿真结果如图 8 所示, 经验证, 该译码器的输出结果与 C++ 程序一致. 如图 8 所示, 置位信号 rst_n 为下降沿有效; start 信号为 1 时, 信道 LLR 开始输入; phi_i 为译码比特; en_calc、en_continue 和 en_update 信道分别为 LLR 计算模块、路径选择与更新模块以及部分和更新模块的有效控制信号. 当译码比特 phi_i 为 1 023, 且路径选择与更新模块更新完最后一个比特后开始输出译码比特. l_opt 为最佳路径, decoded_finished_1 控制译码比特的输出, 为高有效. decoded_codes 为译码比特, count_out_2 为译码比特输出控制计数器, 每个时钟周期输出 1 比特译码比特.

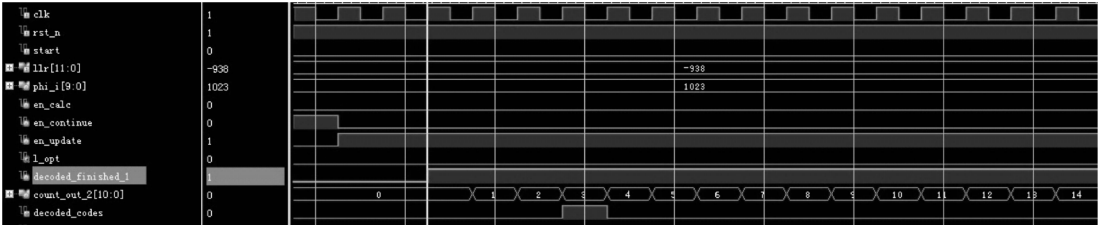


图 8 FPGA 功能仿真结果

4.2 FPGA 综合结果

该编码器在 Xilinx Virtex-7 XC7V2000-2 的 FPGA 上实现, 运行最高主频为 227 MHz, 资源占用相比于已有结构大大降低. 其资源占用结果如表

1 所示. 由于本文提出的 SCL 译码器为串行结构, 因此整体占用资源占用不多. 又由于巧妙地使用了 RAM 来存储数据量比较大的 LLR 和部分和数据, 使得查找表和寄存器资源使用较少.

表 1 FPGA 综合资源占用结果

资源	使用数目	总数目	使用率/%
LUT	7 009	1 221 600	0.57
LUTRAM	224	344 800	0.06
Register	9 142	2 443 200	0.37
BRAM	6	1 292	0.46
BUFG	1	128	0.78

5 结束语

本文基于非递归结构的 LLR-SCL 算法,开发了一种串行的 SCL 极化码译码器.使用最小和近似算法简化 LLR 的计算,并通过非递归算法简化电路,降低计算复杂度.在 LLR 计算模块和部分和更新模块采用串行结构对每条路径依次搜索,简化了电路,减少了硬件消耗.最后,该译码器在 Xillinx XC7V2000 FPGA 上的综合结果表明,在性能保持不变的情况下,硬件资源消耗较低,同时主频可达 227 MHz.

参考文献:

[1] 伏思昌,何小海,卿粼波,等.一种高速 LDPC 码译码器的设计及实现[J]. 微电子学与计算机, 2015, 32(6): 54-57.

[2] 仵林博,朱怡安,田杭沛.一种改进的 Turbo 码译码算法[J]. 微电子学与计算机, 2010, 27(12): 146-148.

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

[4] Arikan E. A performance comparison of polar codes and reed-muller codes[J]. IEEE Communications Letters, 2008, 12(6): 447-449.

[5] Tal I, Vardy A. List decoding of polar codes[J]. 2011 IEEE International Symposium on Information Theory Proceedings (Isit), 2011, 61(5): 1-5.

[6] Chen K, Niu K, Lin J R. List successive cancellation decoding of polar codes[J]. Electronics Letters, 2012, 48(9): 500-501.

[7] Niu K, Chen K. Stack decoding of polar codes[J]. E-

lectronics Letters, 2012, 48(12): 695-697.

[8] Chen K, Niu K, Lin J R. Improved successive cancellation decoding of polar codes[J]. IEEE Transactions on Communications, 2013, 61(8): 3100-3107.

[9] Niu K, Chen K. CRC-aided decoding of polar codes [J]. IEEE Communications Letters, 2012, 16(10): 1668-1671.

[10] Leroux C, Tal I, Vardy A, et al. Hardware architectures for successive cancellation decoding of polar codes[C]//IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP). Corfu, Greece, 2011: 1665-1668.

[11] Li B, Shen H, Tse D. An adaptive successive cancellation list decoder for polar codes with cyclic redundancy check[J]. Ieee Communications Letters, 2012, 16(12): 2044-2047.

[12] Leroux C, Raymond A J, Sarkis G, et al. A semi-parallel successive-cancellation decoder for polar codes [J]. Ieee Transactions on Signal Processing, 2013, 61(2): 289-299.

[13] Zhang C, You X H, Sha J, et al. Hardware architecture for list successive cancellation polar decoder[C]//2014 IEEE International Symposium on Circuits and Systems (ISCAS). Melbourne VIC, Australia, 2014: 209-212.

[14] Lin J, Yan Z Y. An efficient list decoder architecture for polar codes[J]. IEEE Transactions on Very Large Scale Integration (Vlsi) Systems, 2015, 23(11): 2508-2518.

[15] Balatsoukas-Stimming A, Parizi M B, Burg A. LLR-based successive cancellation list decoding of polar codes[J]. IEEE Transactions on Signal Processing, 2015, 63(19): 5165-5179.

作者简介:

刘丽华 女,(1993-),硕士研究生.研究方向为通信与信息处理. E-mail:liulihua@ime.ac.cn.

管武 男,(1981-),博士,副研究员.研究方向为信道编码与软件无线电.

梁利平 男,(1969-),博士,研究员.研究方向为高性能 DSP 和数字集成电路设计技术.