

北京航空航天大学学报
Journal of Beijing University of Aeronautics and Astronautics
ISSN 1001-5965,CN 11-2625/V

《北京航空航天大学学报》网络首发论文

题目：基于混合映射的存算一体电路运算速度优化
作者：孙秀楠，吴乾火，王伦耀，查晓婧，储著飞，夏银水
DOI：10.13700/j.bh.1001-5965.2026.0315
收稿日期：2026-05-26
网络首发日期：2026-09-02
引用格式：孙秀楠，吴乾火，王伦耀，查晓婧，储著飞，夏银水．基于混合映射的存算一体电路运算速度优化[J/OL]．北京航空航天大学学报.
https://doi.org/10.13700/j.bh.1001-5965.2026.0315



网络首发：在编辑部工作流程中，稿件从录用到出版要经历录用定稿、排版定稿、整期汇编定稿等阶段。录用定稿指内容已经确定，且通过同行评议、主编终审同意刊用的稿件。排版定稿指录用定稿按照期刊特定版式（包括网络呈现版式）排版后的稿件，可暂不确定出版年、卷、期和页码。整期汇编定稿指出版年、卷、期、页码均已确定的印刷或数字出版的整期汇编稿件。录用定稿网络首发稿件内容必须符合《出版管理条例》和《期刊出版管理规定》的有关规定；学术研究成果具有创新性、科学性和先进性，符合编辑部对刊文的录用要求，不存在学术不端行为及其他侵权行为；稿件内容应基本符合国家有关书刊编辑、出版的技术标准，正确使用和统一规范语言文字、符号、数字、外文字母、法定计量单位及地图标注等。为确保录用定稿网络首发的严肃性，录用定稿一经发布，不得修改论文题目、作者、机构名称和学术内容，只可基于编辑规范进行少量文字的修改。

出版确认：纸质期刊编辑部通过与《中国学术期刊（光盘版）》电子杂志社有限公司签约，在《中国学术期刊（网络版）》出版传播平台上创办与纸质期刊内容一致的网络版，以单篇或整期出版形式，在印刷出版之前刊发论文的录用定稿、排版定稿、整期汇编定稿。因为《中国学术期刊（网络版）》是国家新闻出版广电总局批准的网络连续型出版物（ISSN 2096-4188，CN 11-6037/Z），所以签约期刊的网络版上网络首发论文视为正式出版。

基于混合映射的存算一体电路运算速度优化

孙秀楠，吴乾火，王伦耀✉，查晓婧，储著飞，夏银水

（宁波大学 集成电路学院，宁波 315211）

✉通信作者 E-mail： wanglunyao@nbu.edu.cn

摘要 在以铁电晶体管数据选择器（Fe-MUX）为基础实现存算一体逻辑电路时，为了兼顾电路运算速度与铁电晶体管使用数量，提出了一种同时使用二选一和四选一 Fe-MUX 的混合映射方法。该方法主要包含二方面内容：提出了一种 Fe-MUX 电路结构，并在该电路结构基础上提出了用于映射的二选一和四选一 Fe-MUX 的电路设计；提出了适合二选一和四选一 Fe-MUX 混合映射的逻辑函数分割方法。通过将待实现的逻辑函数表示成二元决策图（BDD），然后将 BDD 分割成适合二选一和四选一 Fe-MUX 混合映射的子 BDD 集合，最后完成逻辑函数基于二选一和四选一 Fe-MUX 的混合映射。所提 Fe-MUX 电路的逻辑功能用 Preisach 模型进行仿真验证，所提 BDD 分割算法用 C++实现。实验结果表明：相比采用单一二选一 Fe-MUX 的映射结果，采用所提混合映射方法可以使电路的计算周期减少约 50%；相比采用单一四选一 Fe-MUX 的映射结果，采用所提混合映射方法可以使电路的铁电晶体管使用数量节省约 12.3%。

关键词 铁电晶体管，数据选择器，存算一体，逻辑函数分割，二元决策图分割

中图分类号 TN384; TN4; TP331.1 文献标志码 A DOI: 10.13700/j.bh.1001-5965.2026.0315

Optimizing the Operation Speed of Computing-in-Memory Circuits Based on Hybrid Mapping

Sun Xiunan, Wu Qianhuo, Wang Lunyao✉, Zha Xiaojing, Chu Zhufei, Xia Yinshui

（School of Integrated Circuits,Ningbo University,Ningbo 315211,China）

✉E-mail: wanglunyao@nbu.edu.cn

Abstract When implementing compute-in-memory logic circuits based on ferroelectric field-effect transistor multiplexers (Fe-MUX), a hybrid mapping method that simultaneously utilizes 2-to-1 and 4-to-1 Fe-MUXes is proposed to balance circuit processing speed with the number of ferroelectric transistors used. This method consists of two main components: the proposal of a Fe-MUX circuit architecture, along with circuit designs for the 2-to-1 and 4-to-1 Fe-MUXes based on this architecture; and the proposal of a logic function partitioning method suitable for the hybrid mapping of 2-to-1 and 4-to-1 Fe-MUXes. By representing the logic function to be implemented as a Binary Decision Diagram (BDD), then partitioning the BDD into a set of sub-BDDs suitable for hybrid mapping using 2-to-1 and 4-to-1 Fe-MUXes, the logic function is finally mapped using a hybrid approach based on these Fe-MUXes. The logical functionality of the proposed Fe-MUX circuit was verified through simulation using the Preisach model, and the proposed BDD partitioning algorithm was implemented in C++. Experimental results show that, compared to the mapping results using a single 2-to-1 Fe-MUX, the proposed hybrid mapping method reduces the circuit's cycle time by approximately 50%; compared to the mapping results using a single 4-to-1 Fe-MUX, the proposed hybrid mapping method reduces the number of ferroelectric transistors used by approximately 12.3%.

收稿日期：2026-05-26

基金项目：国家自然科学基金（U23A20351，62304115），浙江省自然科学基金创新群体项目（LDT23F04021F04）

英文基金项目：National Natural Science Foundation of China (U23A20351， 62304115); Natural Science Foundation of Zhejiang pvovince,China (LDT23F04021F04)

网络首发时间：2026-09-02 13:46:09 网络首发地址：https://link.cnki.net/urlid/11.2625.V.20260902.1041.002

Key words ferroelectric field-effect transistor; multiplexer; computing-in-memory; logic function partitioning; binary decision diagram (BDD) partitioning

随着物联网、人工智能、机器人和边缘计算等新兴应用的迅速发展，数据规模呈指数级增长。传统的冯·诺依曼架构是处理器和存储器分离的架构，二者性能的差异和数据的逐级搬运导致了系统延迟和能量损耗。存算一体(Computing in Memory，CiM)架构通过在存储单元内部或邻近的区域执行计算，成为解决传统架构问题的一个有效途径。

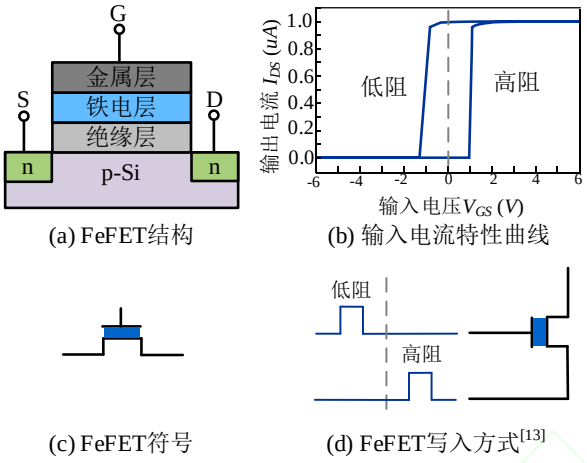
目前可实现存算一体的器件主要有静态随机存储器、闪存以及新型非易失存储器件等。新型非易失存储器件主要有旋转移力矩磁存储器[1]、相变存储器[2]、阻变存储器[3]以及铁电场效应晶体管[4-6](Ferroelectric Field-Effect Transistor，FeFET)等。相比于其他非易失存储器件，FeFET 具有电压控制写入、非易失存储、写入能效高以及与互补金属氧化物半导体(Complementary Metal Oxide Semiconductor，CMOS)工艺高度兼容[4]等优势，使得 FeFET 在 CiM 领域具有广阔的应用前景。

在 FeFET 的电路设计方面，Huang 等人[7]使用互补的 FeFET 设计了二选一铁电晶体管-数据选择器(Fe-MUX)，并且基于 Fe-MUX 实现可重构的与门、或门和异或门；Breyer 等人[8]提出了基于 FeFET 的 CiM 阵列架构，可以实现查找表和 MUX 的复合逻辑功能；Ramanujam 等人[9]设计了二选一 Fe-MUX，并且以此开展了仲裁器物理不可克隆函数方面的研究。Zha 等人[10]利用 FeFET 的存算特性设计了 CiM 单元电路，并且基于所提的单元电路和基本逻辑门实现了 CiM 全加器。

在逻辑综合领域，目前还没有可用于映射的基于 FeFET 的逻辑电路单元库。因此，基于通用的具有存算功能的逻辑门映射实现复杂存算电路的设计是一个有效途径。文献[11]提出了一种具有共享结构、数据输入端可扩展的二选一 Fe-MUX，并且借助二元决策图(Binary Decision Diagram, BDD)分割实现复杂逻辑电路用所提的 Fe-MUX 进行映射。但受到二选一 Fe-MUX 的电路功能与映射策略的限制，由文献[11]得到的电路的运算速度取决于 BDD 的层数，基于 k 层的 BDD 得到的电路需要 k 个计算周期完成计算。为了减少电路的计算周期，一个有效途径就是采用四选一或具有更多选择通路的数据选择器，但此类更为复杂的选址电路会引入更多的控制电路，且实际电路在用数据选择器实现方面往往采用多种类型的组合，而不是单一类型数据选择器。为此，本文在综合考虑电路复杂性基础上提出了一种基于二选一和四选一 Fe-MUX 的混合映射策略，以减少电路的计算周期，提高电路的运算速度。

1 FeFET 及 Fe-MUX 设计

图 1(a)为 FeFET 的基本结构，该结构在晶体管的栅极介质引入铁电材料作为铁电层。其铁电层可以通过栅源电压(V_{GS})写入极化状态，使 FeFET 表现为不同的阻态，从而调节输出电流(I_{DS})，且该极化状态具有非易失性。图 1(b)为 Preisach 模型[12]仿真得到的 N 型 FeFET 的输出电流特性曲线。由图 1(b)可得，当 V_{GS} 大于某个正阈值时，FeFET 写入正极化状态，此刻 FeFET 为低阻，对应逻辑“1”；反之当 V_{GS} 小于某个负阈值时，FeFET 写入负极化状态，此刻 FeFET 为高阻，对应逻辑“0”；且写入的状态在 $V_{GS}=0$ 时刻也保持不变。FeFET 的符号如图 1(c)所示。本文 FeFET 写入方式如图 1(d)所示[13]。



(a) FeFET结构　(b) 输入电流特性曲线

(c) FeFET符号　(d) FeFET写入方式[13]

图 1 FeFET 器件结构和电路特性。
Fig.1 FeFET device structure and circuit characteristics.

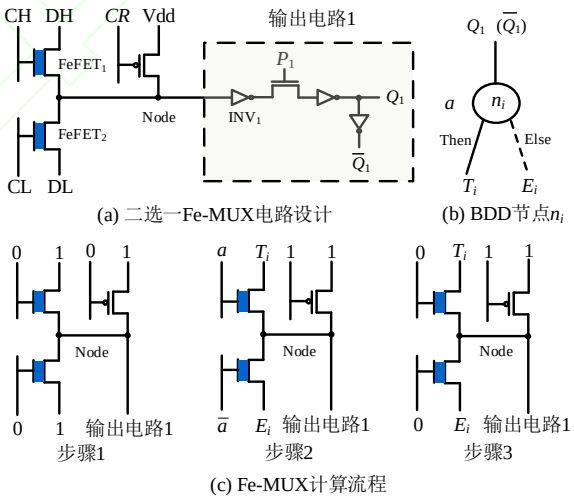
图 2(a)为本文提出的二选一 Fe-MUX 的电路结构，该电路的逻辑功能对应图 2(b)所示的一个 BDD 节点，假设该节点层级变量为 a，Then 边输入为 T_i，Else 边输入为 E_i。现以图 2(c)所示的 3 个步骤说明图 2(a)的电路如何实现一个 BDD 节点的逻辑功能。其中，逻辑信号为逻辑“1”时(Vdd)，电压为 5 V；为逻辑“0”时(Gnd)，电压为 0 V。

步骤 1：预充阶段。CH 和 CL 输入 1，DH 和 DL 输入 1，FeFET 初始化为高阻，*CR* 信号为 1，电路的 Node 节点预充至 Vdd。而在其它阶段，*CR* 信号为 1，预充通路处于关断状态。

步骤 2：计算阶段。CH 和 CL 分别以脉冲形式输入变量 a 和 $\overline{a}$，DH 和 DL 分别输入节点的 Then 边输入 T_i 和 Else 边输入 E_i，实现 FeFET 的状态写入。写入的状态可以改变 Node 的电位。以变量 $a=0$ 为例，若 $T_i=1$，$FeFET_1$ 保持高阻状态，Node 的电位保持，对应逻辑“1”；若 $T_i=0$，$FeFET_1$ 写入低阻状态，Node 的电位下降，对应逻辑“0”。因此，当变量 $a=1$ 时，Node 的电位对应 Then 边输入 T_i 的逻辑值，且 $FeFET_2$ 在上述计算过程始终保持高阻。最终，反相器 INV_1 输出 T_i 的逻辑非。

步骤 3：输出阶段。若 P_1 信号为 1，反相器 INV_1 的输出传输至级联反相器形成互补输出 Q_1 和 $\overline{Q_1}$；若 P_1 信号为 0，级联反相器的输出保持不变。

通过上述步骤，二选一 Fe-MUX 可以实现如图 2(b)所示的一个 BDD 节点的逻辑功能。

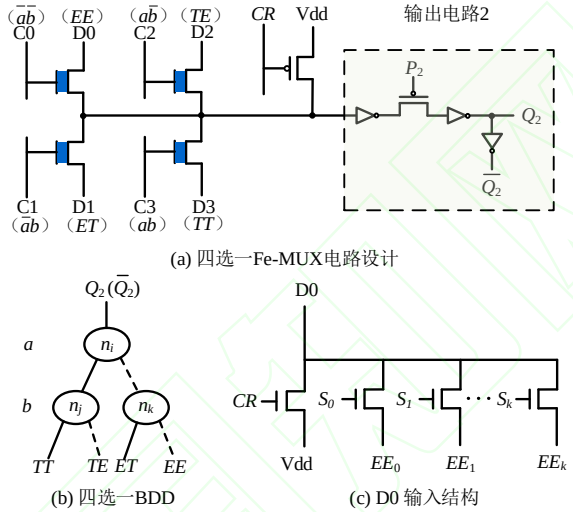


(a) 二选一Fe-MUX电路设计　(b) BDD节点n_i

(c) Fe-MUX计算流程

图 2　二选一 Fe-MUX 电路结构以及逻辑功能。
Fig.2 2-to-1 Fe-MUX circuit structure and logic function.

在图 2(a)电路的基础上，通过增加数据输入通路可以得到四选一 Fe-MUX，电路结构如图 3(a)所示。四选一 Fe-MUX 的工作过程和二选一 Fe-MUX 相同。在计算阶段，电路通过 BDD 相邻 2 个层级对应的 2 个变量进行寻址以选定一个 FeFET 施加写入脉冲，实现数据选择。而未被选中的 FeFET 始终保持高阻。以图 3(b)的 BDD 为例，图中 BDD 的 2 个层级对应变量分别为 a 和 b。当变量 $\{a,b\}=\{0,1\}$ 时，该 BDD 对应的输出为 ET；而当 $\{a,b\}=\{0,1\}$ 时，图 3(a)电路选中 D1 的数据，也是 ET。同理，在变量$\{a,b\}$取不同组合时，图 3(b)的 BDD 和图 3(a)的电路选中的数据是相同的。因此，图 3(b)的 BDD 可以用图 3(a)的四选一电路实现。Fe-MUX 电路的数据输入采用文献[11]的结构，以图 3(c)所示的 D0 为例：在预充阶段，CR 信号为 1，$f(S_0,S_1,\cdots,S_k)$ 均置为 0。在计算阶段，CR 信号为 0，选定一组数据进入 Fe-MUX 计算，以 $S_1=1$ 为例，数据组合 $\{EE_1,ET_1,TE_1,TT_1\}$ 进入计算。



(a) 四选一Fe-MUX电路设计

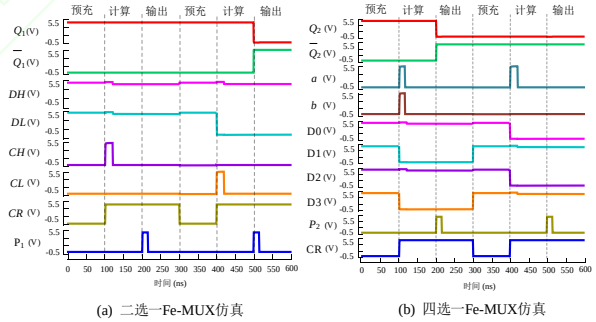
(b) 四选一BDD

(c) D0 输入结构

图 3 四选一 Fe-MUX 的电路结构及逻辑功能。

Fig.3 4-to-1 Fe-MUX circuit structure and logic function.

二选一 Fe-MUX 有 2^3 种输入组合，而四选一 Fe-MUX 有 2^6 种输入组合，图 4 为部分输入组合的 Fe-MUX 电路逻辑仿真结果。仿真平台为 Virtuoso，FeFET 使用 Preisach 模型[12]。其中二选一 Fe-MUX 的 2 组数据组合分别设为{1,1}与{0,1}，地址变量分别设为{0,1}；四选一 Fe-MUX 的 2 组数据组合分别设为{1,0,1,0}与{0,1,0,1}，地址变量{a,b}分别设为{11,10}。仿真结果表明，输出信号能够正确反映数据组合与地址的变化。



(a) 二选一Fe-MUX仿真

(b) 四选一Fe-MUX仿真

图 4 Fe-MUX 仿真验证结果

Fig.4 Simulation and verification results of Fe-MUX

2 基于 BDD 的混合 Fe-MUX 映射

本文 Fe-MUX 电路的映射借助 BDD 实现。BDD 是有向无环图，广泛应用于逻辑综合[14][15]。对于

任意一个具有 k 个变量的布尔函数 $f(x_1,x_2,\cdots,x_i,\cdots,x_k)$，其关于变量 x_i 的香农分解可表示为：

$$f = x_i \cdot f(x_i=1) + \overline{x_i} \cdot f(x_i=0) \quad (1)$$

当 $x_i=1$，输出为 $f(x_1,x_2,\cdots,1,\cdots,x_k)$；当 $x_i=0$，输出为 $f(x_1,x_2,\cdots,0,\cdots,x_k)$。公式(1)说明布尔函数的逻辑功能可以用数据选择电路实现。本文采用了简约有序的 BDD 形式(ROBDD)，记为 $\mathcal{B}=(V,H,\prec,E,lev,eva)$，其中：

V 表示由 BDD 的内部节点 V_{int} 和叶子节点 V_{leaf} 构成的节点集合，H 表示 BDD 的变量集合，其中 $V=V_{\text{int}}\cup V_{\text{leaf}}$。以图 5(a)中的 $\mathcal{B}_1$ 为例，其有 $V_{\text{int1}}=\{n_0,n_1,n_2,n_3,n_4,n_5\}$，$V_{\text{leaf1}}=\{1\}$，$H_1=\{a,b,c\}$。

$\prec$ 表示从根到叶子的层级变量顺序。以图 5(a)的 $\mathcal{B}_1$ 为例，有 $\prec_1=a<b<c$。

lev 表示层级映射。假设 k 个变量的布尔函数构建的 ROBDD 有 $k+1$ 层，其叶子节点位于第 0 层，根节点位于第 k 层，层级自叶子向根节点逐层递增，若节点 $n_i\in V$，则 $lev(n_i)\in\{0,1,\cdots,n\}$。

E 表示边的集合，$E(n_i,n_j)$表示节点 n_i 与 n_j 之间的边类型，且 n_i 为 n_j 扇入。本文使用 $E_t(n_i,n_j)$，$E_e(n_i,n_j)$，$E_c(n_i,n_j)$表示 Then、Else 以及 Complement 的边类型。

eva 表示节点的布尔值。假设节点 $n_i\in V_{leaf}$，x_k 为 n_i 所在层级的变量，则 $eva(n_i)=\{0,1\}$。对于边类型为 $E_t(n_1,n_i)$，$E_e(n_0,n_i)$ 的节点，n_i 的值可以表示为 $eva(n_i)=(\overline{x_k}\cdot eva(n_0)+x_k eva(n_1))$，对边类型为 $E_c(n_0,n_i)$，$E_t(n_1,n_i)$ 的节点，n_i 的值可以表示为 $eva(n_i)=(\overline{x_k}\cdot\overline{eva(n_0)}+x_k eva(n_1))$。

BDD 分割算法是 Fe-MUX 映射的基础，算法得到的每一个子 BDD 的逻辑功能都可以用一个对应的 Fe-MUX 电路实现。为描述分割策略，本文引入如下定义：

层级间距。$d(n_i,n_k)$ 表示节点 n_i 与 n_k 的层级间距，若 $lev(n_i)\geq lev(n_k)$，则 $d(n_i,n_k)=lev(n_i)-lev(n_k)$，若 $lev(n_i)<lev(n_k)$，则 $d(n_i,n_k)=\varnothing$。

映射深度。假设 $\mathcal{B}_i$ 的最大层级数为 Max_lev_i，则某一节点 n_i 的映射深度 $Depth(n_i)$与其所在层级的关系为：$Depth(n_i)=Max_lev_i-lev(n_i)$。由映射深度定义可知，$\mathcal{B}_i$ 的根节点的映射深度为 0，叶子节点的映射深度为 Max_lev_i。

时序长度(Time Length)。时序长度表示借助 BDD 分割得到的电路完成从叶子节点到根节点的计算并最终输出所需的时间，它一般包含多个 Fe-MUX 计算周期。

为了实现 Fe-MUX 电路的混合映射，本文对 BDD 进行层级合并和节点分类。

层级合并。以图 3(b)为例，$eva(n_i)$可以由变量 a 和 b 以及节点 n_j 和 n_k 的输入 $\{EE,ET,TE,TT\}$ 计算得到，所以位于 $lev(n_i)$与 $lev(n_i)-1$ 层级的节点可以并行计算。为此，本文自根向叶地划分 BDD 层级，除叶子之外相邻的 2 个层级视为一个同时计算的合并层级。以图 5(d)的 $\mathcal{B}_1$ 为例，Lv3/Lv2 为一个合并层。合并层级存在如下规律：若节点 n_i 位于合并层的上层，则 $Depth(n_i)$为偶数；若节点 n_i 位于合并层的下层，则 $Depth(n_i)$为奇数。

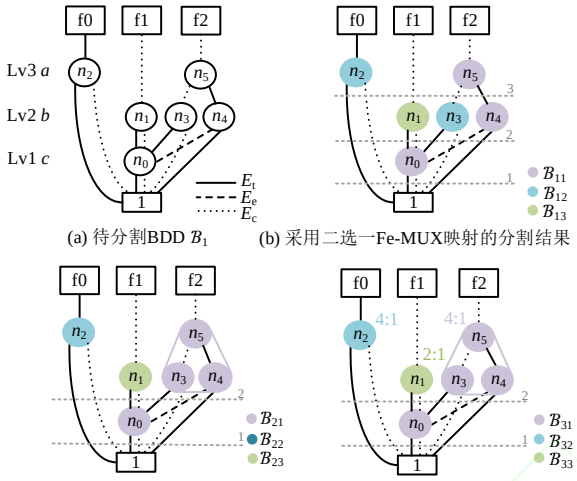
假设节点 n_i 为某一个合并层级中的一个节点，n_i 为以下节点类型之一：

四选一节点。若 $Depth(n_i)$为偶数，则节点 n_i 为四选一节点，该节点作为四选一电路的输出节点。

二选一节点。若 $Depth(n_i)$为奇数，且节点 n_i 的一个扇出节点 n_k 满足 $lev(n_k)>lev(n_i)+1$，则节点 n_i 为二选一节点。该节点可作为二选一或四选一电路的输出节点。

隐节点。若 n_i 不为上述节点，则 n_i 为隐节点，该节点无独立的电路输出。

以图 5(d)的 $\mathcal{B}_Q$ 分割为例，$\mathcal{B}_1$ 的 Lv3 和 Lv2 视为一个合并层级。其中 n_5、n_2 和 n_0 均为四选一节点，n_3 和 n_4 为隐节点，n_1 为二选一节点。在第 1 个计算周期，n_0 的值由一个四选一电路计算并输出。在第 2 个计算周期，$\{n_3,n_4,n_5\}$ 作为一个节点集合由一个四选一电路计算并输出节点 n_5 的值，同时 n_2 的值由另一个四选一电路计算并输出，n_1 的值由一个二选一电路计算并输出。



(a) 待分割BDD $\mathcal{B}_1$ (b) 采用二选一Fe-MUX映射的分割结果

(c) 采用四选一Fe-MUX映射的分割结果 (d) 采用混合映射的分割结果

图 5 不同 Fe-MUX 和映射策略分割 $\mathcal{B}_1$ 所得结果。

Fig.5 Results of $\mathcal{B}_Q$ partitioning using different Fe-MUX types and mapping strategies

图 5 为采用不同 Fe-MUX 和映射策略分割 $\mathcal{B}_1$ 得到的电路和时序长度。相同颜色节点构成一个子 BDD，其中灰色横虚线用于区分在不同的计算周期计算的节点。

图 5(a)为待分割的 $\mathcal{B}_1$，图 5(b)为采用二选一 Fe-MUX 映射算法[11]对 $\mathcal{B}_1$ 的分割结果，分割得到子 BDD 集合为$\{\mathcal{B}_{11},\mathcal{B}_{12},\mathcal{B}_{13}\}$。电路运行时，数据沿着叶子节点到根节点的方向逐层计算。因此该电路的计算周期数等于 BDD 的层数，时序长度为 3。该映射方法所得电路完成 k 层 BDD 的计算所需的时序长度为 k。

图 5(c)为采用单一四选一 Fe-MUX 映射算法对 $\mathcal{B}_1$ 的分割结果，分割得到子 BDD 集合为$\{\mathcal{B}_{21},\mathcal{B}_{22},\mathcal{B}_{23}\}$，子 BDD 包含的节点分别为$\{n_1\}$，$\{n_2\}$和$\{n_0,n_3,n_4,n_5\}$。不同于二选一电路采用的逐层计算方式，该电路的 Lv3 和 Lv2 进行层级合并。由于合并层级可以在同一个计算周期完成计算，采用四选一 Fe-MUX 映射可以有效减少时序长度。以图 5(b)(c)的分割为例，$\mathcal{B}_{11}$ 在 3 个计算周期依次计算节点 n_0，n_4 以及 n_5，而 $\mathcal{B}_{21}$ 在 2 个计算周期分别计算节点 n_0 和节点集合$\{n_3,n_4,n_5\}$，所以图 5(c)的时序长度减少为 2。

相比于采用单一的二选一或者四选一电路的映射，同时使用二选一和四选一 Fe-MUX 的混合映射可以同时兼顾时序长度和 FeFET 数量的优化。图 5(d)为采用混合映射的方法对 $\mathcal{B}_1$ 的分割结果。整个电路用 2 个四选一 Fe-MUX($\mathcal{B}_{31},\mathcal{B}_{32}$)以及 1 个二选一 Fe-MUX($\mathcal{B}_{33}$)实现，在减少了 FeFET 数量的同时时序长度仍为 2。

图 6 为图 5(d)混合映射得到的电路，其中 $\mathcal{B}_{31}$，$\mathcal{B}_{32}$ 以及 $\mathcal{B}_{33}$ 分别由 Fe-MUX 单元 FM1，FM2 和 FM3 进行映射。其中，FM1 和 FM2 为四选一 Fe-MUX，而 FM3 为二选一 Fe-MUX。四选一 Fe-MUX 的地址变量从 VG_1 和 VG_2 输入，而二选一 Fe-MUX 的地址变量仅从 VG_2 输入。FM1、FM2 和 FM3 的最终输出分别对应 BDD 的输出 f2、f0 和 f1。

由于 Fe-MUX 的数据选择以 FeFET 阻值状态的形式实现以及存储，并且电路的输出采用时分复用的结构。因此，图 6 中 2 个标注为 FM1 的电路实际上是同一个 Fe-MUX 时分复用的结果[11]。图 6 中 Fe-MUX 的具体连接方式如下：在计算周期 1，Fe-MUX 的地址输入 VG_1 输入变量 c，VG_2 为任意值，同时 FM1 的输出“Q”分别接 FM3 的“DH”输入端和 FM1 的“D2”输入端；“$\overline{Q}$”分别接 FM3 的“DL”输入端和 FM1 的“D1”输入端。当电路运行到计算周期 2 时，Fe-MUX 的地址输入 VG_1 和 VG_2 分别输入变量 a 和 b。

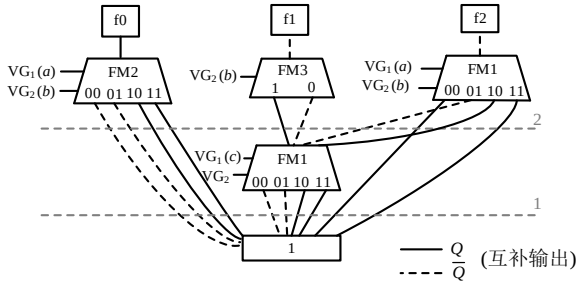


图 6 图 5(d)的电路映射。
Fig.6 Circuit partitioning mapping of Fig. 5(d).

假设待分割的 BDD 为 $\mathcal{B}$，$\mathcal{B}$ 的节点集合为 V；$\mathcal{B}_\alpha$ 为 $\mathcal{B}$ 的一个子 BDD，$\mathcal{B}_\alpha$ 的节点集合为 V_α。则本文用于混合映射的 BDD 分割应当满足如下约束：

(1)假设 $V_\alpha=\{n_1,n_2,\cdots,n_n,n_i\}$，且 V_u 为节点 n_i 的扇出节点集合，对于非隐节点 $n_k\in V_\alpha$，若存在 $d(n_k,n_i)=min\{d(n_\alpha,n_i)\,|\,n_\alpha\in V_\alpha\}$，则要求 n_k 满足 $d(n_k,n_i)\geq max\{d(n_u,n_i)\,|\,n_u\in V_u\}$。该约束规定了子 BDD 内部节点的层级间距要求。

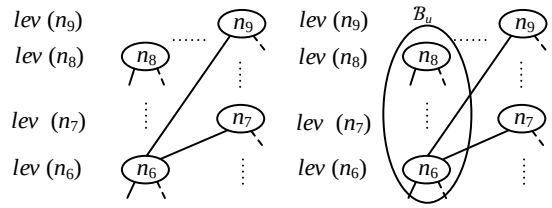
(2)假设节点 $n_i\in V$，若节点 n_i 与 BDD 的输出相连，为使 n_i 的输出不被覆盖，要求 n_i 应为某个子 BDD 的根节点。

(3)假设 $\mathcal{B}_\alpha$ 的根节点为二选一节点，对于 $n_k\in V_\alpha$，要求 n_k 为二选一节点。

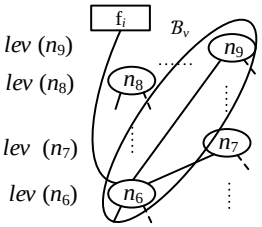
现以图 7 中 BDD 分割的例子说明上述约束的作用。图 7(a)中 BDD 的 4 个节点之间的层级关系为 $lev(n_6)<lev(n_7)<lev(n_8)<lev(n_9)$，且图 7(a)中节点均不为隐节点。

约束(1)(2)是为解决 Fe-MUX 输出电路的时分复用引发的数据寄存冲突的问题。以图 7(b)为例，$\mathcal{B}_u$ 为一个子 BDD，节点 $\{n_6,n_8\}\in\mathcal{B}_u$，$\{n_7,n_9\}\notin\mathcal{B}_u$。假设 $\mathcal{B}_u$ 被映射到一个 Fe-MUX 单元 FM1。当计算层级 $lev(n_6)$时，FM1 的输出是 n_6 的输出。由于节点 n_6 和 n_8 共用 FM1 的输出电路，当计算 n_8 节点时，FM1 的输出已经被替换为 n_8 的输出。因此，当计算 n_9 节点时，n_9 的输入数据已经被改写。为避免节点数据在失去作用之前被改写，分割得到的子 BDD 应当满足约束(1)。同理，以图 7(c)为例，节点 n_6 为 BDD 的一个输出节点，$\mathcal{B}_v$ 为一个子 BDD，$\{n_6,n_9\}\in\mathcal{B}_v$，$\{n_7,n_8\}\notin\mathcal{B}_v$。若 $\mathcal{B}_v$ 被映射到一个 Fe-MUX 单元 FM2。当计算 n_6 节点时，FM2 的输出是 n_6 的输出。由于 n_6 和 n_9 共用 FM2 的输出电路，因此，当计算 n_9 节点时，FM2 的输出已经被替换为 n_9 的输出。由于每一个子 BDD 均会被映射为一个 Fe-MUX，并且每一个 Fe-MUX 的最终输出为子 BDD 的根节点的输出，且不会被改写。因此，与 BDD 输出相连的节点一定是某一个子 BDD 的根节点，这就是引入约束(2)的原因。

为了在不改变电路时序长度的同时构建可以映射至二选一 Fe-MUX 的子 BDD，一个简单途径就是构建一个完全由二选一节点构成的子 BDD，所以引入约束(3)。



(a) 待分割BDD的部分节点 (b) 不满足约束(1)的分割



(c) 不满足约束(2)的分割

图 7 不满足约束的 BDD 分割。
Fig.7 Non-constraint-satisfying BDD partitioning.

3 混合优化映射算法

算法 1 为混合映射算法的伪代码，待分割的 BDD 以 $\mathcal{B}$ 表示。算法采用自定义数据结构管理 BDD 节点信息，记录节点的变量、层级、映射深度、节点类型、子 BDD 编号、复制编号以及节点扇入/扇出关系。算法输出为完成分割的 $\mathcal{B}$。节点的子 BDD 编号、节点类型和复制编号初始为 -1。

子程序 Classify_BDD($\mathcal{B}$)用于设置 BDD 节点的节点类型。该子程序从 BDD 的根节点开始遍历所有节点，通过检查节点的映射深度和节点扇入/扇出关系设置节点的类型为 0、1 和 -1，分别对应二选一节点、四选一节点和隐节点。

子程序 Find_Start_Node($\mathcal{B}$)用于找到第 i 个子 BDD 的起始节点。该子程序从 BDD 的根节点开始遍历所有节点，通过检查节点子 BDD 编号和节点类型，找到并返回一个尚未被分割的非隐节点，设置该节点的子 BDD 编号为 i，将其加入到第 i 个子 BDD，同时定义该节点为 Ns；若没有符合条件的节点，则返回空节点，说明遍历完成。

子程序 Extend_Node($\mathcal{B}$,Ns,i)用于实现第 i 个子 BDD 的扩展。功能是以为 Ns 起点向下搜索符合分割约束的非隐节点，将其加入到第 i 个子 BDD，伪代码见算法 3。

子程序 Node_Copy($\mathcal{B}$,SubBDDs)用于节点的检查与复制，SubBDDs 表示 $\mathcal{B}$ 分割所得的子 BDD 集合。该子程序检查与 BDD 输出相连的节点是否为其所在子 BDD 的根节点。如果存在输出节点不为根节点的情况，则复制该节点作为新的子 BDD，同时设置该节点的复制编号为新的子 BDD 编号。然后检查四选一节点是否存在与其层级间距为 1 的扇入节点，若存在，则复制该节点加入四选一节点所在的子 BDD。

算法 1：BDD_Mapping_Optimization
输入： ROBDD $\mathcal{B}$
输出： $\mathcal{B}$ with SubBDDs marks

```
1.  Classify_Node(B);
2.  i=1;
3.  Ns=Find_Start_Node(B);
4.  WHILE(Ns!=Null_Node) {
5.    Extend_Node(B, Ns, i);
6.    i++;
7.    Ns=Find_Start_Node(B); }
8.  Node_Copy(B, SubBDDs);
```

图 8 算法 1 伪代码
Fig.8 Pseudocode of Algorithm 1

算法 2 为子程序 Classify_BDD($\mathcal{B}$)的伪代码。Const_Node 表示 $\mathcal{B}$ 的常数节点；$Ns.type$ 为节点 Ns

的节点类型。该子程序的执行过程如下：从 BDD 的根节点开始遍历所有节点，当遍历至节点 *Ns* 时，如果 *Depth*(*Ns*)为偶数，则调用子程序 Process_Even_Node(*Ns*)设置 $Ns.type=1$。如果 *Depth*(*Ns*)为奇数，则调用另一个子程序 Process_Odd_Node(*Ns*)。该子程序的执行过程如下：检查节点 *Ns* 是否存在层级大于 $lev(Ns)+1$ 的扇出节点。若不存在，则设置 $Ns.type=-1$；若存在，则设置 $Ns.type=0$，直到遇到常数节点，说明遍历完成。

算法 2：**Classify_BDD**(*B*, *Ns*, *i*)
输入：ROBDD *B*, *Ns*, *i*
输出：*B* with node type
1. **WHILE**(*Ns*!=Const_Node) {
2. **IF** (*Depth*(*Ns*) mod 2 = 0) **THEN**
3. Process_Even_Node(*Ns*);
4. **ELSE**
5. Process_Odd_Node(*Ns*); } }

图 9 算法 2 伪代码
Fig.9 Pseudocode of Algorithm Extend_Node

算法 3 为子程序 Extend_Node(*B*,*Ns*,*i*)的伪代码。其中子程序 Select(*B*,*Ns*,*i*)用于检查节点 *Ns* 的输入节点是否存在符合分割约束的非隐节点。若存在，则调用子程序 Add_Node(*B*,*Ns*,*i*)。该子程序以 *Ns* 的扇入节点与 *Ns* 之间的层级间距作为分割依据，优先选择层级间距最大的扇入节点，将其加入到第 *i* 个子 BDD，设置该节点的子 BDD 编号为 *i*，同时定义该节点为新的 *Ns*；若不存在，则调用子程序 Add_Other_Node(*B*,*lev*(*Ns*),*i*)。该子程序从 *lev*(*Ns*)-1 层级开始向下搜索满足分割约束的非隐节点，将其加入到第 *i* 个子 BDD，设置该节点的子 BDD 编号为 *i*，同时定义该节点为新的 *Ns*。对新的 *Ns* 继续调用子程序 Extend_Node 进行拓展，直到遇到常数节点，结束第 *i* 个子 BDD 的构建。

算法 3：**Extend_Node**(*B*, *Ns*, *i*)
输入：ROBDD *B*, *Ns*, *i*
输出：*B* with SubBDDs marks
1. **WHILE**(*Ns*!=const_node) {
2. **IF** Select(*B*, *Ns*, *i*) **THEN**
3. *Ns*=Add_Node(*B*, *Ns*, *i*);
4. **ELSE** {
5. **ELSE** *Ns* =Add_Other_Node(*B*, *lev*(*Ns*), *i*); } }

图 10 算法 3 伪代码
Fig.10 Pseudocode of Algorithm Extend_Node

现以图 5(d)的 $\mathcal{B}_Q$ 为例说明本文算法执行过程。首先调用子程序 Classify_BDD，以设置 $\mathcal{B}_1$ 的节点类型，$\{n_0,n_1,n_2,n_3,n_4,n_5\}$ 的节点类型分别设为 $\{1,0,1,-1,-1,1\}$。再调用子程序 Find_Start_Node 搜索得到第 1 个子 BDD 记为 $\mathcal{B}_{31}$ 与其起始节点 n_5，然后调用子程序 Extend_Node 对 $\mathcal{B}_{31}$ 进行拓展，优先检查 n_5 的扇入节点 n_3 和 n_4，由于二者均为隐节点，放弃 n_3 和 n_4。调用子程序 Add_Other_Node 向下搜索节点。对于节点 n_1，由于其为 BDD 的输出节点，放弃 n_1。而节点 n_0 符合约束条件，因此加入 $\mathcal{B}_{31}$。子程序 Extend_Node 再对 n_0 进行拓展，由于下一层级仅有常数节点，于是得到了第一个子 BDD 为 $\{n_0,n_5\}$。再次由子程序 Find_Start_Node 返回第 2 个子 BDD 记为 $\mathcal{B}_{32}$ 与其初始节点 n_2，调用子程序 Extend_Node 对 $\mathcal{B}_{32}$ 进行拓展，由于不存在符合条件的节点，于是得到第 2 个子 BDD 为$\{n_2\}$。按照相同的分割规则可以得到第 3 个子 BDD 为$\{n_1\}$记为 $\mathcal{B}_{33}$。子程序 Find_Start_Node 遍历完成。调用子程序 Node_Copy 检查与输出相连的节点，其中 n_1，n_2 和 n_5 均为所在子 BDD 的根节点，检查四选一节点 n_0，n_1 和 n_5 与其扇入节点的层级间距，由于 $d(n_5,n_3)=d(n_5,n_4)=1$，分别复制节点 n_3 和 n_4 加入 n_5 所在的 $\mathcal{B}_{31}$，最终构成子 BDD 集合为$\{\mathcal{B}_{31},\mathcal{B}_{32},\mathcal{B}_{33}\}$分别包含节点$\{n_0,n_3,n_4,n_5\}$、$\{n_2\}$和$\{n_1\}$。最后，由于$\{\mathcal{B}_{31},\mathcal{B}_{32},\mathcal{B}_{33}\}$的根节点类型为$\{1,1,0\}$，$\mathcal{B}_{31}$ 和 $\mathcal{B}_{32}$ 用四选一 Fe-MUX 实现逻辑功能，$\mathcal{B}_{33}$ 用二选一 Fe-MUX 对应实现逻辑功能。

4 实验结果和分析

本文的算法用C++语言实现，算法运行平台为Ubuntu 24.04，处理器为Intel Ultra 7，内存32 GB，使用 g++ 13.3.0 编译器编译。测试电路为开源逻辑综合基准电路 Iwls 93。测试流程如下：首先利用CUDD 将电路的 Verilog 描述转化为 ROBDD，构建过程使用 GROUP_SIFT 的动态变量重排策略优化BDD 结构；最后利用 C++容器存储 ROBDD 的节点信息和节点连接关系作为算法的输入。

基准电路的测试结果列于表 1。其中第 1 列为电路名称；第 2 列为电路输入/输出数量；第 3 列为ROBDD 的节点数量；第 4 列为采用不同映射方法电路的时序长度，分别显示采用二选一 Fe-MUX[11]、四选一 Fe-MUX 以及混合映射方法时的时序长度，分别以 TL_2、TL_4 和 TL_H 表示；第 5 列为采用不同映射算法得到的子 BDD 数量，分别显示采用二选一 Fe-MUX[11]、四选一 Fe-MUX 以及混合映射算法得到的子 BDD 数量，分别以 N_2、N_4 和 N_H 表示。其中，混合映射算法在得到 N_H 的基础上，分别给出适合二选一或四选一 Fe-MUX 映射的子 BDD 数量，分别以 N_T 和 N_F 表示，且 $N_H = N_T + N_F$；第 6 列为 FeFET 数量的优化情况，用 *FR* 表示，其为混合映射相对于四选一 Fe-MUX 映射所减少的FeFET 比例，其计算公式为：

$$FR = \frac{4N_4 - (2N_T + 4N_F)}{4N_4} \quad (2)$$

其中，4 倍的 N_4 表示四选一 Fe-MUX 映射所用的 FeFET 数量，2 倍的 N_T 和 4 倍的 N_F 之和表示混合映射所用的 FeFET 数量。

在时序长度方面，本文所提的四选一 Fe-MUX 映射和混合映射的时序长度相比文献[11]中所提的二选一 Fe-MUX 映射的时序长度显著减少。以电路 alu4 为例，采用二选一 Fe-MUX 映射的时序长度为 14，采用四选一 Fe-MUX 映射或混合映射的时序长度仅为 7。对于电路 alu4，混合映射可以减少50%的时序长度。测试结果说明，对于一个 k 层的 BDD，采用二选一 Fe-MUX 映射电路的时序长度为 k，而采用混合映射电路的时序长度与 k 的奇偶性相关：若 k 为偶数，则电路的时序长度为 $k/2$；若 k 为奇数，由于 BDD 的第 1 层不存在可以合并的相邻层级，电路的时序长度为$(k+1)/2$。总体而言，与二选一 Fe-MUX 映射相比，采用混合映射策略的电路在时序长度上平均减少约 50%。时序长度的减少可以提高电路的运行速度。

在 FeFET 数量方面，混合映射相比于四选一 Fe-MUX 映射使用更少的 FeFET。以电路 seq 为例，采用二选一 Fe-MUX 映射[11]所用的 FeFET 数量 $N_2 = 810$;采用四选一 Fe-MUX 映射所用的 FeFET 数量 $N_4 = 1620$；采用混合映射所用的 FeFET 数量 $N_H = 1180$。对于电路 seq，四选一 Fe-MUX 映射相比于二选一 Fe-MUX 映射的 FeFET 数量增加一倍，混合映射相比于四选一 Fe-MUX 映射节省 27.1%的FeFET 数量。此外，测试电路采用不同映射算法得到的子 BDD 数量基本接近，原因是本文的算法和文献[11]的 BDD 分割得到的子 BDD 数量的均受 BDD 宽度的制约。

表 1 数据表明，所有电路采用混合映射的时序长度与采用四选一 Fe-MUX 映射的时序长度相等。但由于二选一 Fe-MUX 电路比四选一 Fe-MUX 电路更简单，所以混合映射在电路面积上要优于四选一Fe-MUX 映射，因此也导致混合映射在电路面积上比二选一 Fe-MUX 映射的开销更大。但是表 1 数据表明，所有采用混合映射的时序长度约为采用二选一 Fe-MUX 映射的 50%，该结果说明混合映射可以显著提升电路运算速度。因此，本文所提出的混合映射策略兼顾了单一四选一电路速度更快和单一二选一电路面积更小的优点。

另一方面，由于 BDD 的结构会影响本文映射结果，而 BDD 的变量排序又会改变 BDD 的结构，表 2 给出了不同 BDD 变量排序算法优化得到的 ROBDD 的不同映射结果。表 2 数据表明，由排序算法 SIFT、GROUP_SIFT、SAME 以及 RANDOM 得到的 ROBDD，混合映射方法的 FeFET 优化情况均有所不同，平均能减少 12.3%的 FeFET 使用数量，即平均 $FR = 12.3\%$。说明混合映射方法在不同的BDD 结构下有良好的收益稳定性。

在动态功耗估算方面，本文采用电路节点的跳变密度(Transition Density，TD)总和衡量电路的动态功耗。对于一个有 k 个变量的逻辑函数 y_j，其信号的跳变密度之和可以由公式(3)计算，其中 $D(x_i)$表示信号 x_i 的跳变密度，而 $P(x_i)$表示信号 x_i 为逻辑 1 的概率[16]。

$$D(y_j)=\sum_{i=1}^{k}P(\frac{\partial y_j}{\partial x_i})\cdot D(x_i)\ (3)$$

根据公式(3)可以推导得到关于二选一 MUX 的跳变密度计算公式和四选一 MUX 的跳变密度计算公式，本文以电路节点的跳变密度总和衡量基准电路在不同映射方法下的动态功耗，其结果列于表 3。表 3 的第 1 列为电路名称；第 2 列为基于文献[11]映射方法的跳变密度总和，用 TD_2 表示；第 3 列为基于混合映射方法得到的跳变密度总和，用 TD_{H} 表示；第 5 列为 TD_{H} 相对于 TD_2 数据的变化情况，用 TR 表示。TR 的计算公式为：

$$TR=\frac{TD_2-TD_{\mathrm{H}}}{TD_2}\ (4)$$

表 3 数据表明，基于混合映射的跳变密度总和相对于基于二选一电路映射[11]平均增加约 12.0%，原因在于四选一 Fe-MUX 相对于二选一 Fe-MUX 具有更多输入信号，导致单个 Fe-MUX 电路节点产生的平均跳变密度增加。

表 1基准电路测试结果

Table 1　Benchmark Circuit Test Results

电路名称	I/O	节点数量	时序长度			子 BDD 数量			FR(%)
			TL_2[11]	TL_4	TL_{H}	N_2[11]	N_4	$N_{\mathrm{H}}(N_{\mathrm{T}}:N_{\mathrm{F}})$	
5xp1	7/10	55	7	4	4	18	18	19(5:14)	8.3
alu4	14/8	733	14	7	7	132	130	130(6:124)	2.3
b1	3/4	7	3	2	2	3	3	3(1:2)	16.7
b9	41/21	126	41	21	21	36	34	36(14:22)	14.7
clip	9/5	146	9	5	5	47	47	47(5:42)	6.0
cordic	23/2	95	23	12	12	8	10	10(2:8)	10.0
dalu	75/16	803	75	38	38	142	142	142(62:80)	21.8
e64	65/65	368	65	33	33	65	64	72(27:45)	8.6
ex4p	128/28	710	128	64	64	115	108	111(42:69)	16.7
frg2	143/139	1274	143	72	72	217	216	268(141:127)	8.6
pair	173/173	3872	173	87	87	580	573	627(294:333)	21.5
seq	41/35	2025	41	21	21	405	405	405(220:185)	27.1
squar5	5/8	39	5	3	3	15	15	15(3:12)	10.0
x3	17/39	697	135	68	68	148	147	157(68:89)	16.3
x4	94/71	544	94	47	47	124	118	122(49:73)	17.4
i2	201/1	584	201	101	101	36	36	36(12:24)	16.7
i9	88/63	925	88	44	44	265	259	277(135:142)	20.9
i10	257/224	33995	257	124	124	3455	3455	3609(1530:2079)	17.7
平均									14.3

表 2混合映射方法关于BDD变量排序算法的适应性测试结果

Table 2 BDD Variable Order Sensitivity Test Results of Hybrid Mapping Scheme

电路名称	基于排序算法得到的 ROBDD 采用混合映射的 FeFET 优化情况					
	SIFT 排序算法		SAME 排序算法		RANDOM 排序算法	
	节点数量	*FR*(%)	节点数量	*FR*(%)	节点数量	*FR*(%)
5xp1	51	16.7	55	8.3	47	14.3
alu4	720	1.3	733	2.3	733	2.3
b1	7	16.7	7	16.7	7	16.7
b9	128	2.9	126	17.2	191	1.0
clip	110	5.0	146	0.0	158	4.6
cordic	97	10.0	95	11.1	93	10.0
dalu	803	0.7	803	21.8	817	0.3
e64	376	6.4	368	8.6	661	5.6
ex4p	743	9.1	710	16.7	754	5.7
frg2	1317	13.6	1275	8.6	1320	11.1
pair	4498	0.2	3966	16.3	5441	20.5
seq	2029	26.3	2025	27.2	2067	13.3
squar5	39	6.7	39	6.7	39	6.7
x3	726	9.9	697	16.4	785	14.4
x4	468	17.0	544	17.4	836	16.7
i2	680	16.7	584	16.7	718	16.7
i9	929	13.1	925	19.1	926	18.4
i10	32964	12.5	33995	17.5	34155	9.7
平均		10.3		13.9		10.5

表 3不同映射方法的节点跳变密度总和对比

Table 3 Total Node Transition Density Comparison Across Mapping Methods

电路名称	节点跳变密度总和(Transition Density，TD)		
	TD_2[11]	TD_H	*TR*(%)
5xp1	130	118	9.5
alu4	1612	1265	21.5
b1	12	11	6.4
b9	192	280	-45.9
clip	364	299	17.8
cordic	96	90	6.5
dalu	1079	1245	-15.3
e64	28	32	-11.6
ex4p	1936	2023	-4.5
frg2	2082	3004	-44.3
pair	8308	10212	-22.9
seq	2187	3077	-40.7
squar5	93	74	20.7
x3	1405	1671	-18.9
x4	1055	1253	-18.8
i2	45	41	8.0
i9	2657	3590	-35.1
i10	80995	119892	-48.0
平均			-12.0

5 结 论

本文主要开展如下二方面工作：1)利用 FeFET 可以对阻态进行非易失存储的特性，结合数据选择

器在任何时刻不可能有多个数据被选中并输出的特点，提出一种具有存算功能的 Fe-MUX 电路结构，并在此基础上提出了用于映射的二选一和四选一 Fe-MUX 电路设计。2)提出适合所设计二选一和四选一 Fe-MUX 实现混合映射的 BDD 分割方法。该分割方法可以得到适用于映射不同逻辑功能 Fe-MUX 的子 BDD 集合，混合映射所得电路可以在一个计算周期并行计算2个层级的节点，从而减少电路的时序长度和 FeFET 使用数量，进而实现电路运算速度的提高和面积开销的减少。基准电路测试结果显示，该方法相比于二选一 Fe-MUX 映射平均减少约 50%的时序长度，相比于四选一 Fe-MUX 映射平均减少约 12.3%的 FeFET 使用数量。但在另一方面，由于二选一电路比四选一电路更简单，导致混合映射电路的跳变密度总和相比于文献[11]平均增加约 12.0%。

参考文献（References）

[1] LI Y T,BAI T S,XU X Y,et al.A survey of MRAM-centric computing:From near memory to in memory[J].IEEE Transactions on Emerging Topics in Computing,2023,11(2):318-330.

[2] ANTOLINI A,LICO A,SCARSELLI E F,et al.An embedded PCM peripheral unit adding analog MAC in-memory computing feature addressing non-linearity and time drift compensation[C]//ESSCIRC 2022-IEEE 48th European Solid State Circuits Conference (ESSCIRC).2022:109-112.

[3] DING Z T,LI X Y,JIN C J,et al.Experimental demonstration of non-volatile boolean logic with field configurable 1FeFET-1RRAM technology[J].IEEE Electron Device Letters,2024,45(6):1084-1087.

[4] BEYER S,DÜNKEL S,TRENTZSCH M,et al.FeFET:A versatile CMOS compatible device with game-changing potential[C]//2020 IEEE International Memory Workshop (IMW).2020:1-4.

[5] HAN G Q,PENG Y,LIU H,et al.Ferroelectric devices for intelligent computing[J].Intelligent Computing,2022,2022:2022/9859508.

[6] MARCHAND C,NICOLAS A,MATRANGOLO P A,et al.FeFET based logic-in-memory design methodologies,tools and open challenges[C]//2023 IFIP/IEEE 31st International Conference on Very Large Scale Integration(VLSI-SoC).2023:1-6.

[7] HUANG Y Y,HUANG P T T,LEE P Y,et al.A new approach for reconfigurable multifunction logic-in-memory Using complementary ferroelectric-FET (CFeFET)[J].IEEE Transactions on Electron Devices,2023,70(8):4497-4500.

[8] BREYER E T,MULAOSMANOVIC H,TROMMER J,et al.Compact FeFET circuit building blocks for fast and efficient nonvolatile logic-in-memory[J].IEEE Journal of the Electron Devices Society,2020,8:748-756.

[9] RAMANUJAM S,BURLESON W.Reconfiguring the mux-based arbiter PUF using FeFETs[C]//2021 22nd International Symposium on Quality Electronic Design (ISQED).Santa Clara,CA,USA:IEEE,2021:257-262.

[10] ZHA X, YE H.FeFET-based computing-in-memory unit circuit and its application[J].Nanomaterials,2025,15(4):319.

[11] 吴乾火，王伦耀，查晓婧，等。可重构铁电数据选择器设计及在映射中的应用 [J]. 电子与信息学报，2025，47 （9）：3321-3331.
WU Q H,WANG L Y,ZHA X J,et al.Design of reconfigurable FeFET-MUX and its application in mapping [J].Journal of Electronics & Information Technology,2025,47 (9):3321-3331 (in Chinese).

[12] DENG S,YIN G D,CHAKRABORTY W,et al.A comprehensive model for ferroelectric FET capturing the key behaviors:scalability,variation, stochasticity,and accumulation[C]//2020 IEEE Symposium on VLSI Technology Digest of Technical Papers.Honolulu:IEEE Press,2020:1-2.

[13] YIN X Z,CHEN X M,NIEMIER M, et al.Ferroelectric FETs-based nonvolatile logic-in-memory circuits[J].IEEE Transactions on Very Large Scale Integration(VLSI)Systems,2019,27(1):159-172.

[14] CHAKRABORTI S,CHOWDHARY P V,DATTA K,et al.BDD based synthesis of boolean functions using memristors[C]//2014 9th international design and test symposium (IDT).Algeries,Algeria:IEEE,2014:136-141.

[15] CHAKRABORTY A, GUPTA P S, SINGH R, et al. BDD-based synthesis approach for in-memory logic realization utilizing memristor aided loGIC (MAGIC)[J].Integration,2021,81:254-267.

[16] NAJM F N. Transition density: a new measure of activity in digital circuits [J].IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,1993,12 (2):310-323.