arXiv:2304.04995v1 [cs.AR] 11 Apr 2023

定制存储器设计：逻辑–输入–存储器相较于传统存储器的缺点与改进

Fabrizio Ottati*、**Giovanna Turvani**、**Marco Vacca**、**Guido Masera**意大利都灵，都灵理工大学电子与电信系

摘要

现代数字系统的速度受到存储器延迟（“存储墙”问题）的严重限制。逻辑与存储器之间的数据交换也导致了系统能耗的很大一部分。存内逻辑（LiM）代表了这一问题的一种有吸引力的解决方案。通过直接在存储器内部执行部分计算，可以提高系统速度，同时降低其能耗。能够带来显著性能提升的存内逻辑解决方案，基于对存储单元的修改。然而，这类修改的代价是什么？它们会如何影响存储器阵列的性能？

在本工作中，通过分析一种实现最大值/最小值计算算法的存内逻辑存储器阵列，对这一问题进行了研究。该存储器阵列采用 FreePDK45nm CMOS 工艺在物理层面进行设计，包含三种存储单元变体，并将其性能与静态随机存取存储器和内容可寻址存储器进行了比较。结果表明，读取和写入操作的性能有所下降，但存内操作非常高效：与静态随机存取存储器的读取操作相比，与操作的能量–延迟积降低了55.26%；因此，存内逻辑方案代表了低密度、高性能存储器的一种极具前景的解决方案。

***K*关键词** 存内逻辑 (存内逻辑) · 存内计算 (IMC) · 存储器墙。

1 引言

现代数字架构基于冯·诺依曼原理：系统被划分为两个主要单元，即中央处理单元和存储器。CPU 从存储器中提取数据，对其进行处理，并将结果写回。这种结构代表了现代计算系统的主要性能瓶颈：事实上，存储器无法以接近处理速度的速率向CPU 提供数据，从而限制了整个系统的吞吐量；此外，CPU与存储器之间的高速数据交换会导致较大的功耗。这个问题通常被称为“存储墙”问题或“冯·诺依曼瓶颈”。为部分弥补这一问题，系统采用了复杂的存储器层次结构，但这并不能完全解决问题：系统仍然受限于无法同时拥有容量大且速度非常快的存储器。

基于这些原因，企业和研究人员正在寻找克服“存储墙”问题的方法：存内逻辑（LIM），也称为存内计算（IMC） [17], , 是一种为此目的而研究的计算范式。在这种模型中，部分计算在存储器内部执行。通过修改存储器架构并向其中添加逻辑电路，即可实现这一目标。由于部分计算直接在存储器内部进行，因此在需要执行某些操作时，CPU 不会受到存储器延迟的限制。此外，CPU 与存储器之间的数据交换速率也会降低，从而减少功耗。

文献中可以找到许多存内逻辑的方法；不过，可以区分出两种主要方法。第一种可归类为近存计算（NMC） [8, 9, 12, 14, 16, 20, 23, 24, 26, 27, 11, 13, 25, 31,32, 33, 7], , 因为存储器内部的阵列未经过修改，而是在其外围电路中添加了逻辑电路；第二种

*通信邮箱：fabrizio.ottati@polito.it。

则可称为存内逻辑（LiM）[10, 18, 19, 21, 22, 15, 28, 29, 36, 30], , 因为通过向存储单元中添加逻辑电路，直接对存储单元进行了修改。
在近存计算（NMC）架构中，逻辑和算术电路被添加到存储器阵列的外围电路中，有时还会利用三维结构；因此，计算电路与存储器电路之间的距离缩短，从而降低两者之间数据交换的功耗和延迟。例如：在 [8], 中，逻辑和算术电路被添加到静态随机存取存储器（SRAM）阵列的底部，数据从不同的存储器块传输而来，经处理后再写回阵列；在 [7], 中，动态随机存取存储器（DRAM）经过改造，可在位线上执行逻辑位运算，并将感应放大器配置为可编程逻辑门。近存计算能够以最小的存储器阵列本身改动最大化存储密度，而存储器阵列是存储器设计中最关键的部分；因此，与基于传统存储器的计算系统相比，其性能提升有限。

在存内逻辑（LiM）架构中，存储单元和外围电路经过改造，在其中添加逻辑和算术电路，从而实现真正的存内处理，数据也能够在每个存储单元内部进行处理。例如：在[36], 中，每个存储单元都添加了一个异或逻辑门，以便直接在存储器中实现二值神经网络（BNN）；在 [28], 中，静态随机存取存储器（SRAM）在存储单元层面经过改造，可直接在存储单元中执行逻辑运算，然后由阵列外围专门设计的感应放大器对运算结果进行合并。这种方法会导致存储密度降低，因为存储单元的面积增大；然而，由于存储器中存储的所有数据都可以由内部阵列一次性处理，最终带来的性能提升非常显著。

许多应用都能从存内计算（IMC）方法中受益，例如机器学习和深度学习算法 [9, 12, 14, 16, 20, 23, 24, 26, 27, 10, 18, 19, 21, 22], 以及通用算法 [11, 13, 25, 31, 32, 33, 7, 15, 28, 29]。例如：在 [10], 一个 6T SRAM 单元中，通过增加两个晶体管和一个电容对其进行改造，从而能够对整个存储器执行模拟计算，这使得可以为机器学习算法实现近似算术运算；在 [33], SRAM 阵列中，由锁存器和查找表（LUT）组成的逻辑层与存储层交错排列，以便直接在阵列内部执行不同类型的逻辑运算；在 [29], 对 6T SRAM 单元的传输晶体管进行改造，以便直接在存储单元中执行逻辑运算，这使存储器能够作为 SRAM、CAM（内容可寻址存储器）或逻辑内存架构运行。总的来说，任何处理高度并行数据并并行执行大量逐元素运算的算法（例如神经网络），在采用 IMC 解决方案时都有可能获得性能提升。
另一个有趣的应用领域是基于超越 CMOS 技术（例如忆阻器技术）的神经形态计算 [5, 6] 。这类器件具有非易失性，而且存储单元占用面积小，因此非常适合用于存内计算或存内逻辑应用。例如，在 [4] 中，为神经形态应用设计了一种 VRRAM 阵列，通过实现存储器内 XNOR 运算来处理突触权重。

通过添加计算元件来修改存储单元电路，是一种有风险的方案：存储器电路经过了高度优化，因此即使是微小的改动，也可能对其行为和性能产生重大影响；此外，这种方案还会降低存储密度。与此同时，由于所有存储的数据都能一次性处理，系统整体性能可以大幅提升。因此，对于缓存等低密度、高性能存储器而言，存内逻辑方案颇具吸引力。重要的是，要确定修改存储单元电路对标准存储器操作（读取和写入）以及存储器内逻辑运算的影响，并客观评估这一方案的优缺点。

本工作的目标是理解并量化这一影响。作为案例研究，采用了一种基于按位逻辑与操作的最大值/最小值计算算法 [30] 。该阵列使用FreePDK 45nm CMOS工艺，在Cadence Virtuoso中于晶体管级进行设计和表征。研究了三种实现相同逻辑功能的不同存储单元电路方案，随后通过考察每种存储器操作的延迟和能耗，将阵列性能与两种传统存储器——6T SRAM和NOR内容寻址存储器——进行了比较。结果表明，修改存储器确实会以不可忽略的方式影响读取和写入操作的性能，但通过适当设计和优化存储单元，可以大幅降低这种影响；然而，存内逻辑操作在能耗方面表现得非常高效。事实上，与SRAM读取操作相比，与操作的能量–延迟积降低了44%。所得结果表明，存内逻辑架构是实现算法加速器的一种非常理想的替代方案，这些加速器可用作二级存储器，在此类存储器中，读取和写入操作的执行速率低于存内逻辑操作的执行速率。

论文结构如下：

• 在第2节中，讨论了作为性能参考的传统存储器（SRAM和CAM）实现方案的设计。

• 在第 3 节中，分析 LiM 阵列的设计以及三种存储单元类型。
• 第 4 节介绍了用于表征所生成存储器阵列的测试平台。
• 输入第5节，讨论了所采用的仿真框架。
• 输入第6节，介绍并分析所得结果。
• 第 7 节对结果和架构进行了讨论。

本文的主要贡献如下：

• 一个 LiM 阵列，以特定算法 [30] 作为案例研究，采用 FreePDK 45nm CMOS 工艺在物理层进行设计，并通过广泛的 SPICE 仿真进行表征。
• 设计并表征了存内逻辑存储单元的三种变体。
• 将 LiM 阵列性能与传统存储器的性能进行比较；具体而言，采用与 LiM 阵列相同的参数设计并仿真静态随机存取存储器和 CAM 阵列。
• 为表征大规模存储器大小下的设计，提出并采用了一种可大幅缩减电路网表规模的电路模型，以尽可能缩短大型阵列的仿真时间。
• 为加快存内逻辑等定制存储器阵列的设计，提出并采用了一种脚本方法。

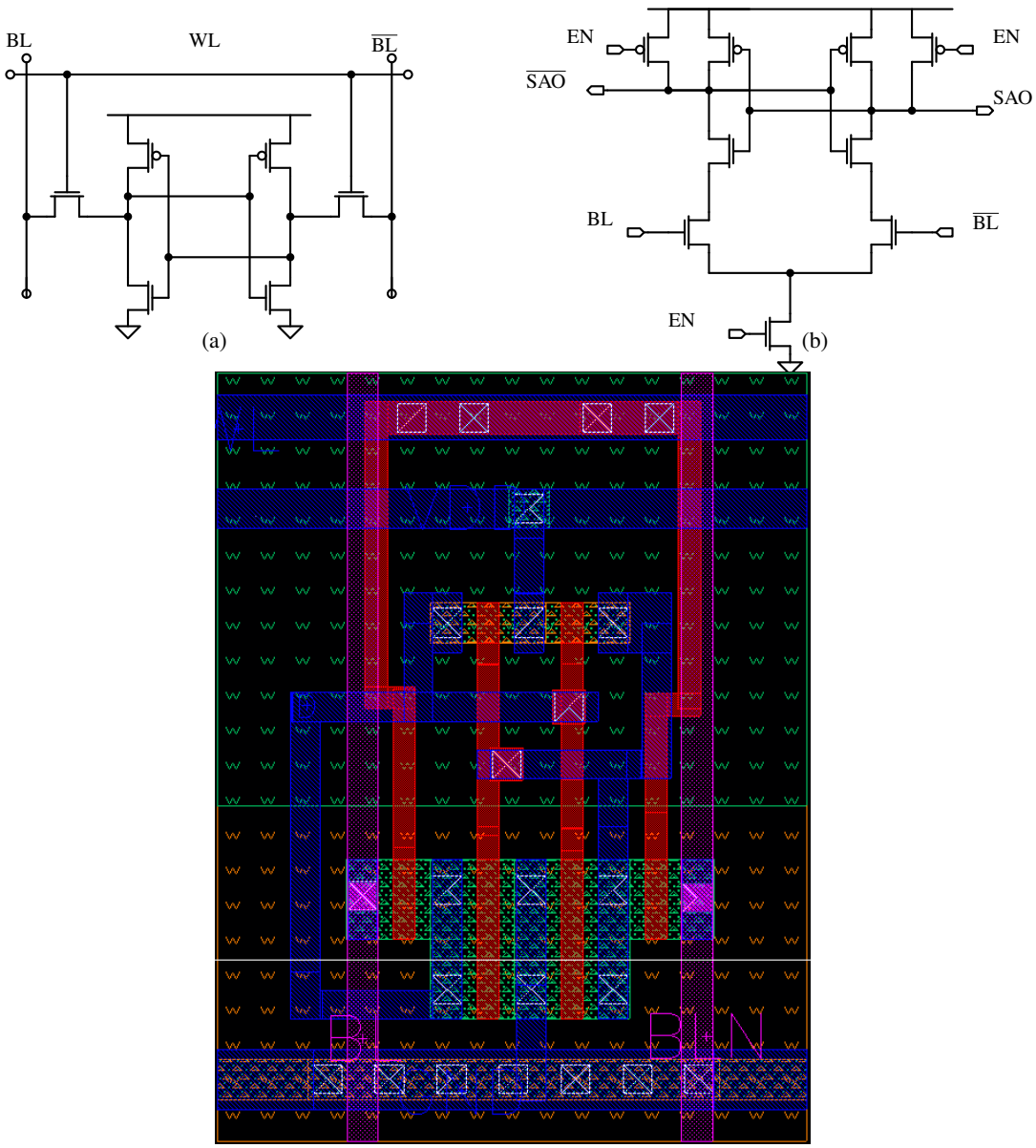
2 参考架构

为了恰当地表征存内逻辑架构设计，我们在 Cadence Virtuoso 中设计了两种标准存储器阵列——SRAM 和 CAM，作为参考电路：选择 SRAM 阵列，是因为它为存储单元电路的复杂度提供了一个较低的基准，其他存储器架构可以此为参考；而选择 CAM 阵列，则是因为它是如今广泛使用的逻辑内存架构实例（每个存储单元执行一次 XNOR 操作）。这些存储器架构选用的存储单元拓扑分别为 6T SRAM 和 NOR内容寻址存储器 [34]。

对于 SRAM 阵列，选用标准的 6T 存储单元（图 1a）：由于本研究旨在设计一种能够执行逻辑运算的存储器架构，因此选择占用 SRAM 核心面积最小的设计，以尽可能减小用于存储功能的存储单元面积。对于读取感测电路，选用传统的电压锁存型感应放大器（SA） [37] ，其电路如图 1b 所示。为比较各存储器的读操作性能，并了解新增的复杂性对阵列标准存储器操作的影响，选用一种常用的感应放大器电路拓扑。该电路具有较高的感测速度和较低的静态功耗，这得益于其用作锁存器的双稳态环的非线性特性。

对于 CAM，采用传统的 NOR 拓扑结构 [34] （图 2a）。在 CAM 感测电路方面，从各种可能的方案中选用了一种电流节省方案 [35] [34]。相应的匹配线感应放大器（MLSA）电路如图 2b 所示。在内容可寻址存储器中，该电路用于降低搜索操作的能耗。与标准感测方案相比，其优势在于：匹配时对匹配线（ML）充电，而发生失配时则不对其放电。事实上，众所周知，在 NOR 内容寻址存储器阵列中，搜索操作的结果最常见的是失配（存储器阵列中只有一个或少数几个字与待搜索字匹配）。将匹配线电压的切换与匹配结果而非失配结果相关联，可大幅降低搜索操作的能耗，因为只有少数线路的电势会发生变化。

在图 2b 中，给出了一个电流节省方案 [34] 。该方案由一个用于为匹配线充电的电流源组成；当发生匹配时，匹配线表现为一个电容；因此，电容得到充电，从而导致匹配线电压发生变化，并在输出端记录匹配结果。相反，在不匹配的情况下，机器学习将电流源连接到地，因而不会被充电，从而避免匹配线电势发生变化；这种变化会导致额外的能耗。在不匹配情况下，采用反馈控制电路来限制注入地的电流，以便在搜索操作期间节省功耗；该电路能够向不匹配线提供尽可能小的电流，同时向匹配线提供尽可能大的电流，以加快匹配感测。



(c)

图 1：SRAM 单元和感应放大器。（**a**）6T 单元。（**b**）SRAM 感应放大器（SA）[37]。（**c**）SRAM 单元版图。

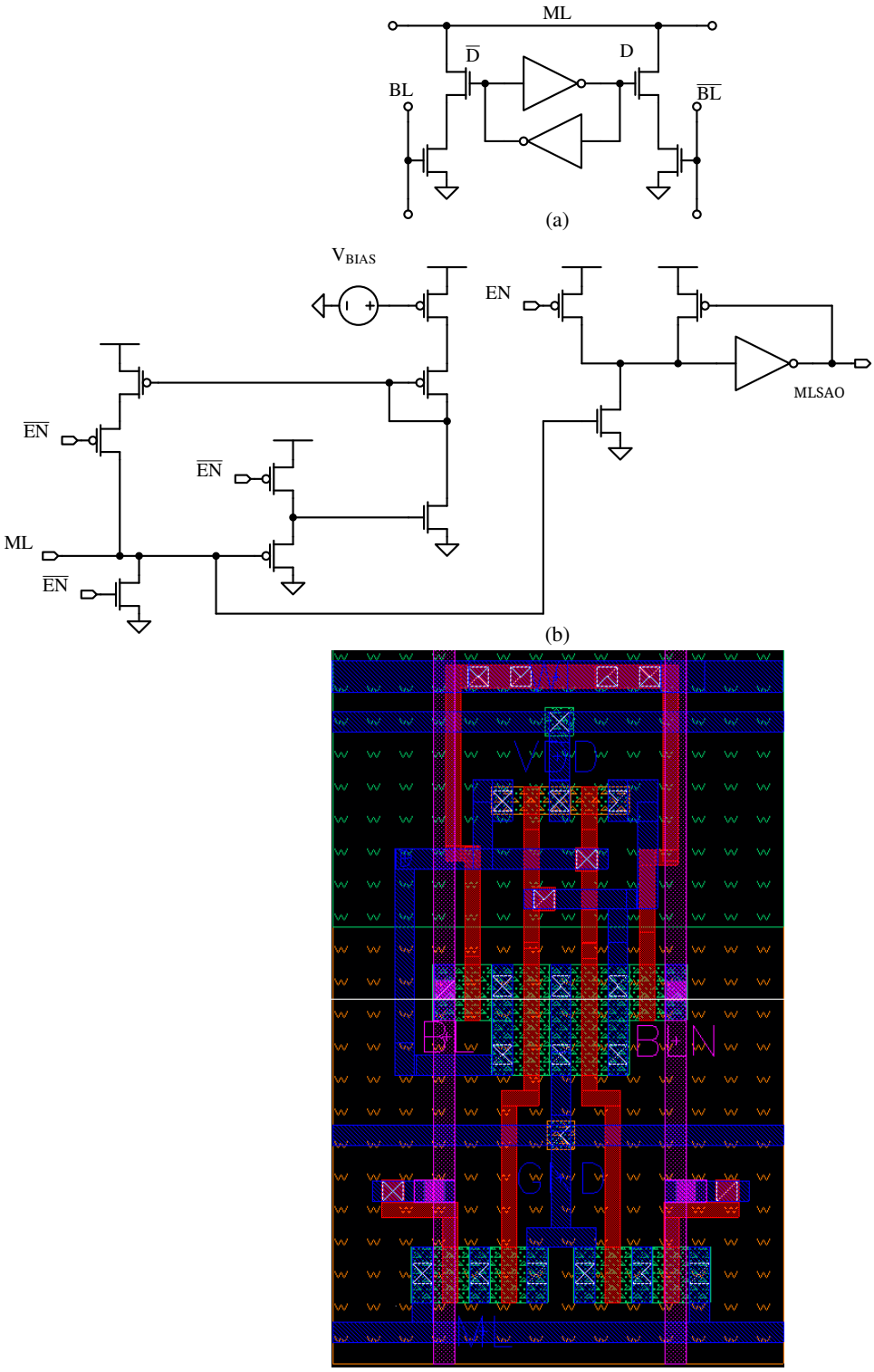


图 2：CAM 单元和 MLSA。**(a)** 简化的 10T NOR 内容寻址存储器单元示意图 [34]。省略了静态随机存取存储器核心的访问晶体管。**(b)** 匹配线感应放大器（MLSA） [35],, 采用电流保存感测方案来执行 CAM 阵列的搜索操作。**(c)** CAM 单元版图。

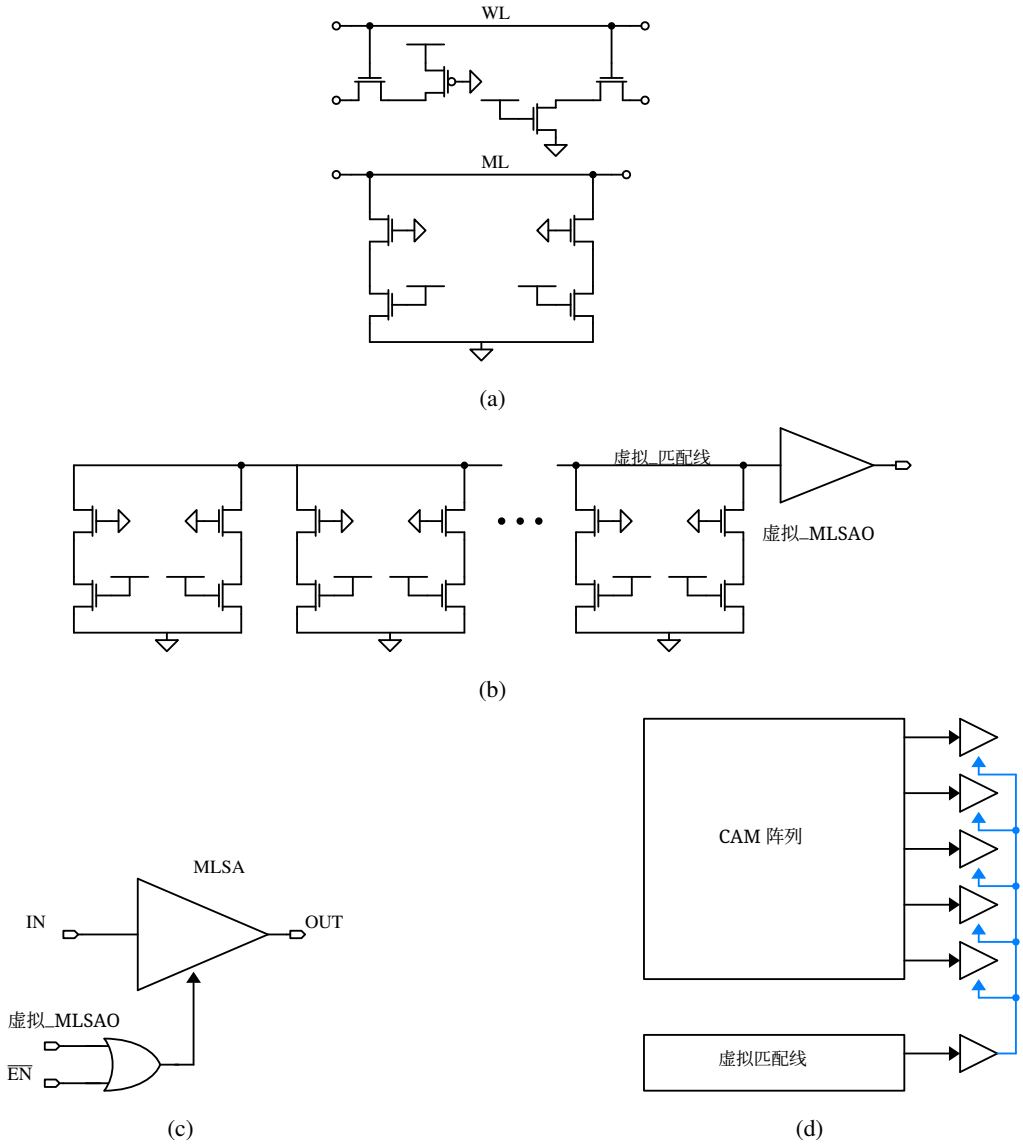


图3：虚拟匹配线方案。使用虚拟MLSA来关闭真实MLSA的电流源，以便在搜索操作期间节省功耗。**(a)** CAM 的虚拟存储单元。电路中仅保留匹配线和字线晶体管，以及存储逻辑“1”的虚拟静态随机存取存储器核心。**(b)** 虚拟匹配线。虚拟存储单元排列成一条匹配线，其长度等于存储器宽度，并连接到虚拟MLSA。为清晰起见，省略了部分虚拟CAM单元。**(c)** 虚拟MLSA的输出用于关闭其他MLSA：虚拟MLSA的输出一旦发生变化，就表示匹配感测所需的时间已过，此时可以关闭真实MLSA的电流源。为此，每个MLSA内部都添加了一个或门，其输出用作内部使能信号。**(d)** 虚拟MLSA的输出连接到所有其他MLSA。虚拟匹配线的位置至关重要：由于虚拟MLSA决定了存储器的时序，其线路位置必须对应于感测延迟的最坏情况。

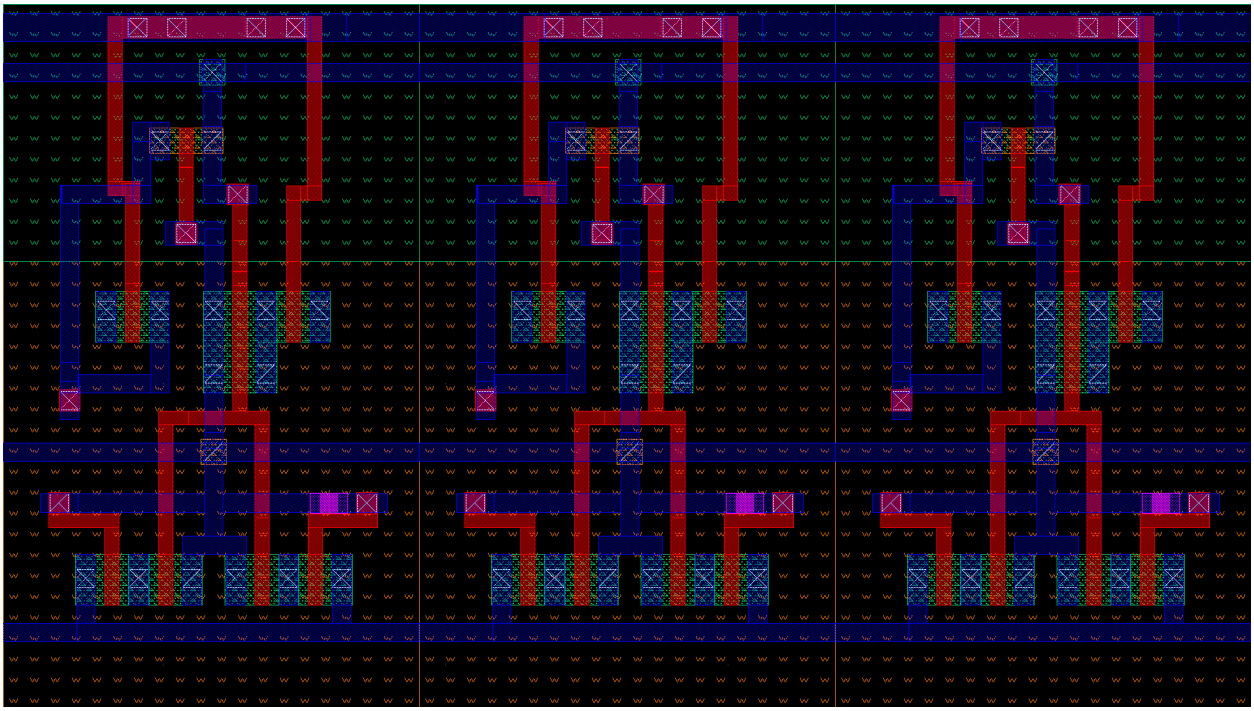


图4：CAM架构中虚拟线的版图局部。

为了限制 MLSA 电流源的导通时间，对 MLSA 电路及其架构进行了修改。为在所有匹配线的值均被正确感测到后立即关闭所有电流源（即所有匹配线均充电至 MLSA 的输入阈值），避免在不匹配线上浪费电流，采用了图3所示的“虚拟匹配线”方案。

图3a展示了一个虚拟CAM单元。它由一个 CAM 单元构成，其中所有未连接到匹配线的晶体管均已移除。剩余 MOSFET 的栅极电势经过选择，使该存储单元始终产生匹配结果，即表现为电容。事实上，由于只有匹配结果会在线路上引起电压变化，因此匹配结果决定了搜索操作的性能。

图3b展示了一条虚拟匹配线。哑单元排列成一行，并连接到一个 MLSA；该 MLSA 在每次搜索操作中都会输出一个“虚拟匹配结果”，记为 $Dummy_MLSAO$。在该架构中，一旦虚拟匹配线上检测到匹配，该信号便用于禁用所有实际的 MLSA。

图3c展示了MLSA的电路。每个MLSA都增加了一个或门，其输出在该电路内部用作使能信号。具体而言，由于使能信号为低电平有效，因此一旦 $Dummy_MLSAO$ 切换为“1”，即在匹配线上检测到匹配，或门的输出就应切换为“1”，以关闭MLSA的电流源。因此，全局使能信号 EN 通过逻辑或与 $Dummy_MLSAO$ 相连。

图3d展示了整个CAM架构。如上所述，虚拟MLSA的输出与全局使能信号一起连接到所有MLSA。由于虚拟匹配线感测延迟决定了每个MLSA正确感测匹配线电位的可用时间，因此它在存储器阵列中的位置对电路时序至关重要。这意味着最坏情况延迟必须与虚拟匹配线的位置相关联，即应将其放置在电路中距离使能信号驱动器尽可能远的位置。

图4展示了虚拟线版图的一部分。可以看出，图2c所示的原始版图中有些晶体管被移除了：实际上，SRAM 核心经过了修改，使存储单元无需显式地向虚拟线中的每个存储单元写入该值，就能存储逻辑“1”。

3 存内逻辑阵列

作为一个案例研究，本文选择了作者设计的一种用于存内最大值/最小值计算的架构 [30]，因为它结合了通用型改进（按位存内与逻辑运算）和用于最大值/最小值计算的专用近存逻辑电路。

因此，它是量化这种特定存内计算方法影响的良好研究案例，而这正是本研究的目标。该架构并非旨在取代CPU，而是作为特定任务的硬件加速器，例如最大值 / 最小值计算或按位存储器操作。

存储器内最大值/最小值搜索算法基于按位与操作。存储器中存储的所有字均与一个称为“掩码向量”的外部字进行按位与运算，该向量逐次按位施加到存储器位线上，直到扫描完整个字宽；然后由近存逻辑对这些与运算的结果进行处理，在每一步选择要舍弃的字，直到只剩下最大值/最小值。

考虑将无符号字存储在存储器中，并且必须找出其中的最大值：在这种情况下，在每次迭代中，仅从最高有效位（MSB）开始将掩码的一位设置为“1”，并丢弃与运算结果等于“0”的所有字。实际上，如果字 *A* 的该位等于“0”，而字*B* 的同一位等于“1”，那么 *B* 大于*A*；因此，从搜索中丢弃*A* 。

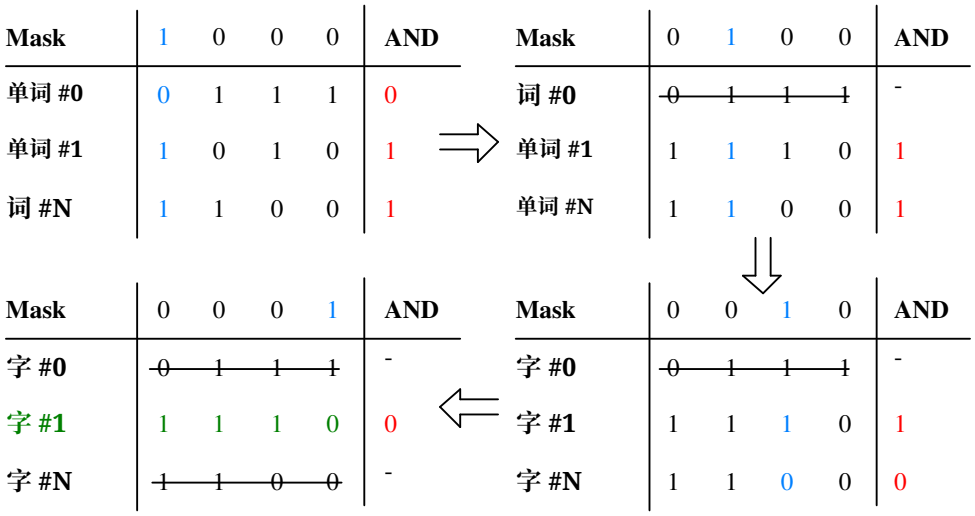


图5：所有字都通过与一个称为“掩码向量”的外部字进行按位与扫描。对于结果为逻辑“0”的字，将其丢弃；最后剩余的字被选为最大值，并可应用优先级机制从中进行选择。在该示例中，所选字以绿色突出显示。

图5给出了无符号字的最大值搜索示例。在每一步中，根据与操作的结果丢弃一个字，直到处理完整个存储器宽度，或只剩下一个字。对于最小值搜索和/或有符号字，以及其他类型的数据编码，只需更改掩码的位并对近存逻辑进行编程即可。

该存储器架构由一种标准 NOR内容寻址存储器组成，正如第2节中介绍的那样，并在此基础上增加了执行与操作的能力；该电路如图6所示。需要指出的是，在本文中仅介绍存内计算阵列示意图，不包括 [30]中所描述的近存逻辑电路。

如前所述，存储器内容与掩码之间的与操作会在所有行上并行执行，每次处理一位；随后，感应放大器通过对各行执行或操作来汇总这些结果，并将结果提供给外围逻辑电路。因此，单个存储单元增加了两种功能：与和或。与功能由插入存储单元中的专用逻辑栅极实现，而或功能则通过贯穿整行的线或线路实现，其结果由外围电路中的感应放大器处理，该放大器称为“ANDSA”。与线示意图如图7所示。

要选择必须执行与操作的列，掩码向量的所有位都必须设置为“0”，除了与选中列对应的那一位：这样，其他列上的所有与运算

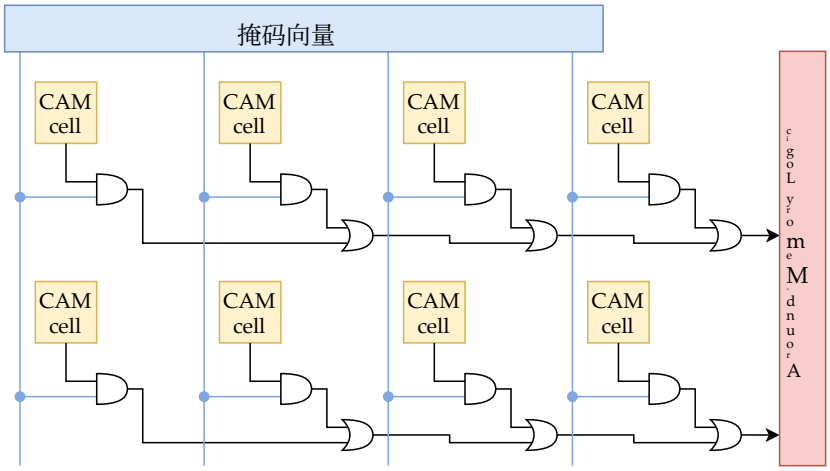


图6：LiM 阵列。它由内容可寻址存储器单元组成，并增加了与逻辑功能；与运算的结果在各行上进行或运算，并提供给近存逻辑。

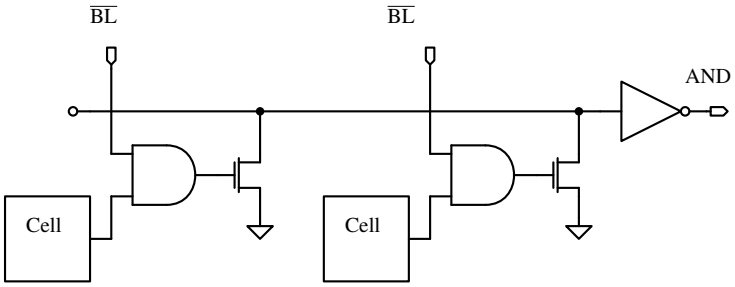


图7：与线。与门的输出通过下拉晶体管连接到线或线路。线路上的信号随后被反相为与结果。与门的结果通过 BL 由掩码选择。

给出‘0’作为结果，从而禁用相应的下拉晶体管，而在线路上感测到的逻辑值完全取决于所选存储单元的与运算输出。下面通过一个示例对此进行说明。用 D_i 表示第 i列存储单元的内容，用 M_i 表示同一位置的掩码位，并用 O 表示在输出端获得的结果；当考虑在该行上实现的按位与时，可得到公式 1：

$$O = \sum_{i=0}^{N-1} D_i \cdot M_i \tag{1}$$

该公式采用了一种不严谨的记法，将求和符号‘+’与或操作相关联，将乘积符号‘·’与与操作相关联。用索引 j 表示选中列，公式可改写如下：

$$M_i = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

$$
\begin{aligned}
O &= \sum_{i=0}^{N-1} D_i \cdot M_i \\
&= \sum_{i=0, i\neq j}^{N-1} D_i \cdot M_i + D_j \cdot M_j \\
&= \sum_{i=0, i\neq j}^{N-1} D_i \cdot 0 + D_j \cdot 1 \\
&= D_j
\end{aligned}
$$

因此，或操作的输出仅由选中列的单元内容决定。

与逻辑功能通过在存储单元内部嵌入一个逻辑门来实现。本文介绍了这种实现方式的三种变体：

- 一个如图 8a 所示的动态 CMOS 逻辑与门。
- 一个如图10a所示的静态CMOS逻辑与门。
- 一种专用与门，专门为待实现的算法设计，旨在尽可能减小存储单元面积，如图11a所示。

3.1 动态CMOS逻辑 与

在图 8a 中，展示了与门的电路。它输入存储单元内容的取反值 D、位线上的掩码位 BL，以及一个用于对与门的输出节点 $\overline{O}$ 进行预充电的额外外部信号 $\overline{PRE}$。可以注意到，无需在内部栅极的输出端增加反相级即可实现与功能：由于存储单元内容和掩码位的取反值可用，可以利用德摩根定律来避免反相级。事实上，由于图 8a 中的栅极输入为 D 和 BL，由该逻辑门实现的二者之间的逻辑或非可以按以下方式重写：

$$
\overline{\overline{D} + \overline{BL}} = D \cdot BL
$$

因此，实现与功能不再需要反相级。该逻辑门嵌入存储单元中，得到图8b所示的电路。可以注意到，在与门的输出端增加了一个下拉晶体管，并将其连接到行线路。

与线是动态 CMOS 逻辑的一种实现：先将线路预充电至逻辑“1”，然后，如果连接到该线路的某个下拉晶体管被使能，就会在评估阶段放电。为正确执行预充电阶段，必须禁用所有下拉晶体管。通常的做法是在每个存储单元的下拉电路源极处添加一个页脚晶体管，并通过专用的行信号在预充电阶段将其禁用，从而防止下拉电路不受与门输出值影响而使线路放电。图 9a 展示了一种可能的电路。

在本工作中，采用了一种不同的方法，在预充电阶段禁用下拉晶体管：将CAM的相同节省电流的感测方案用于与线。这样，由于该线路是预–放电且非预–充电，无需禁用下拉晶体管，因此不需要额外的晶体管和信号，从而可实现更小的存储单元和行占用面积。

图9b提出了一种电路。该逻辑门的真值表考虑了节省电流方案的实现方式，如表1所示。

3.2 静态CMOS逻辑与

提出了第二种嵌入静态CMOS逻辑与门的存储单元。该与门及存储单元的电路如图10所示。

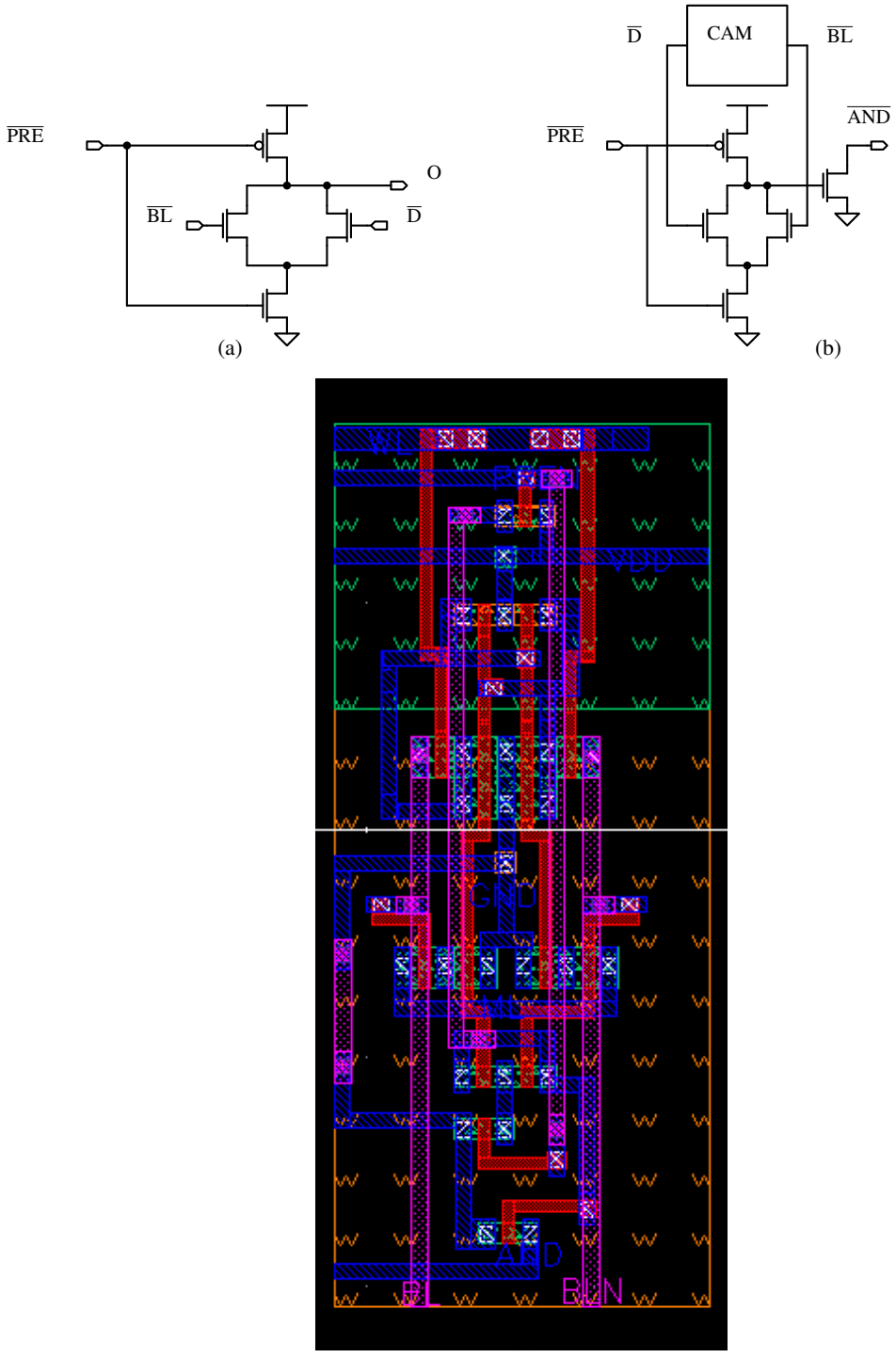


图 8：动态与门及其集成所在的存储单元。**(a)** 动态与门。通过使用单元内容的取反值 D 和掩码位的取反值 BL，可以利用布尔逻辑定律获得与门，而无需在输出端增加反相级。**(b)** 嵌入动态与门和与线下拉晶体管的存储单元。可以看到，与门的输出通过一个下拉晶体管取反，其输出对应于与该线路相关联的 AND 信号。**(c)** 存储单元版图。

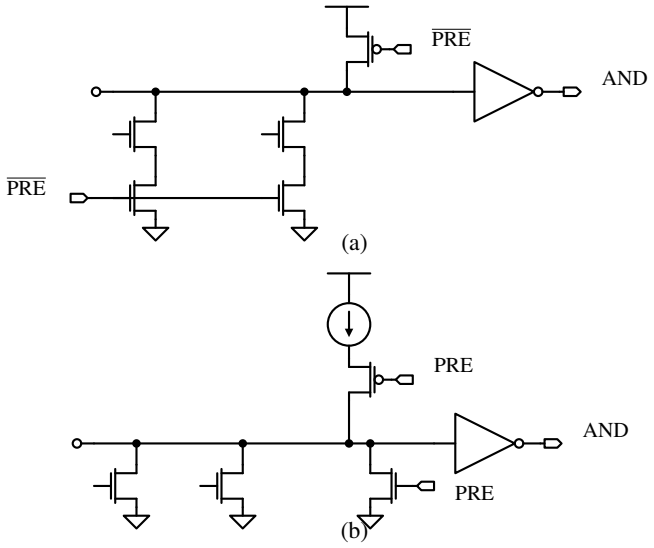


图9：标准方案和节流方案。**(a)** 标准预充电线。使用上拉晶体管将线路预充电至逻辑“1”，同时使用页脚晶体管禁用所有下拉电路；然后，在评估阶段通过停用 PRE 使能这些下拉电路。**(b)** 节流线路。在该方案中，由于线路采用预放电而非预充电，因此无需使用页脚晶体管来禁用下拉电路；随后，在评估阶段，如果没有导通的下拉电路，线路将被充电。

D 位线D	$\overline{D}$	BL	$\overline{BL}$	与	与
0	0	1	1	$1 \to 0$	$0 \to 1$
0	1	1	0	$1 \to 0$	$0 \to 1$
1	0	0	1	$1 \to 0$	$0 \to 1$
1	1	0	0	1	0

表 1：图 8b 中动态与单元的真值表。可以注意到，使用电流节省方案时，当与放电至“0”时，AND输出会充电至“1”；而当 AND=为“1”时，它保持在地电压。

静态与门如图10a所示。与动态与（第3.1节）相比，该门需要更大的存储单元面积，因为按照标准微电子设计规则 [38]，新增的 pMOS 晶体管宽度必须大于图8a中的预充电晶体管宽度。然而，添加这些晶体管后，就可以移除图8a中的预充电信号PRE，而该信号是动态逻辑正常工作的必要条件。如图10b所示，该门嵌入存储单元中，其输出连接到与线的下拉晶体管。该门的真值表与表1所示的动态与单元相同，唯一的区别是，对于静态存储单元，与输出信号是静态 CMOS 信号。

3.3 特殊用途与

本文提出了第三种存储单元变体。该存储单元设计旨在通过根据算法特性进行针对性设计，尽可能降低添加与门所带来的存储单元面积开销。图11展示了与门和存储单元的电路示意图。

如图5所示，每次迭代时，掩码向量通过将掩码中相应的位设为“1”来选择一列存储器，而所有其他存储单元均被禁用。由于与操作是在值为“1”的位与单元内容之间进行的，因此运算结果由存储单元决定，如公式1所示；所以，这与其说是与操作，不如说是选择操作。基于此，存储单元电路可以简化为仅通过承载掩码向量的位线实现存储单元选择，并使被选中存储单元的内容反映到与线上，而无需实现真正的与功能。如图11a所示，将一个下拉晶体管的输入端连接到单元内容、输出端连接到与线，即可实现这一点。

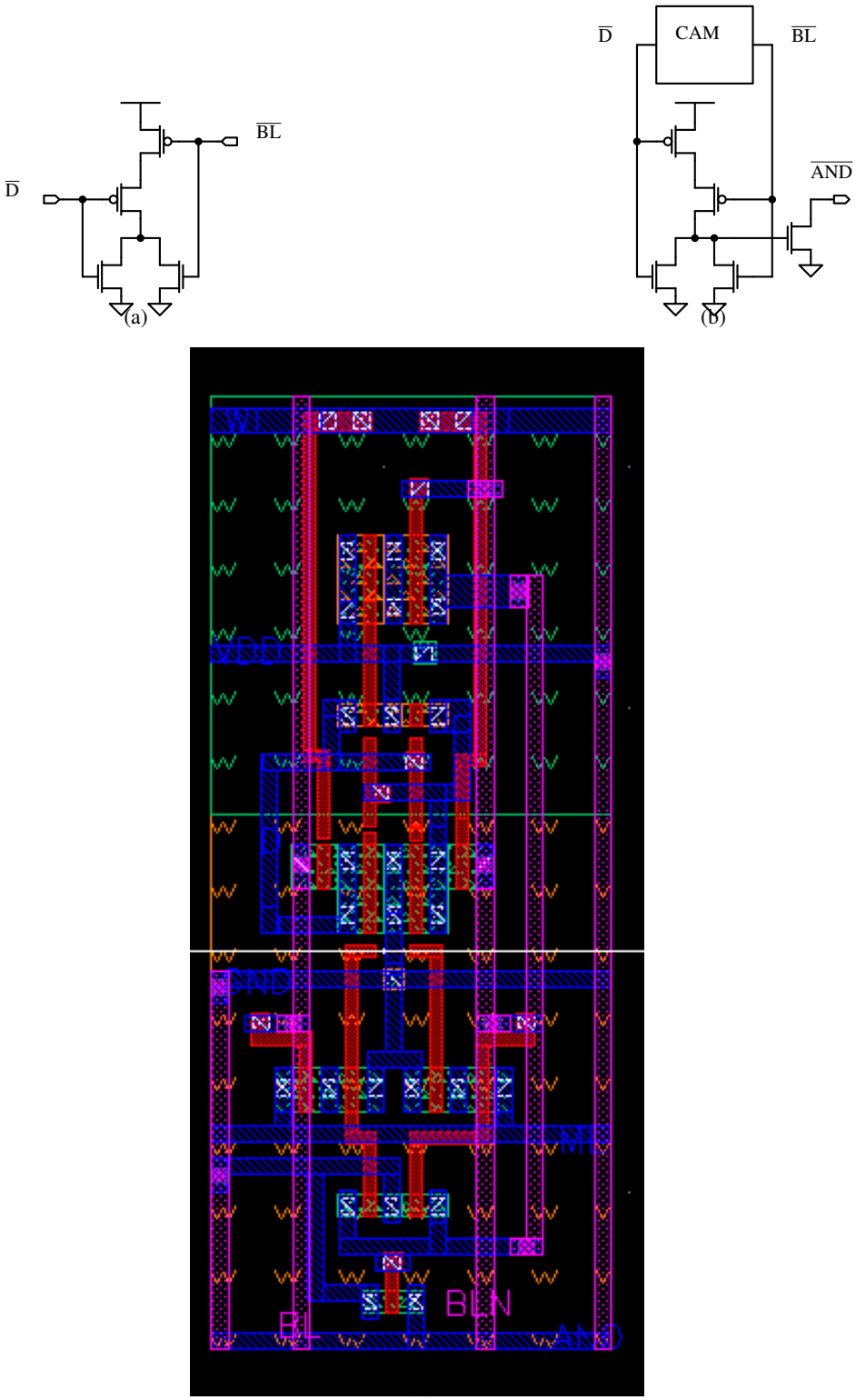


图10：静态与门和存储单元。**(a)**静态 CMOS 与门。**(b)** 嵌入静态与门的存储单元。**(c)**存储单元版图。

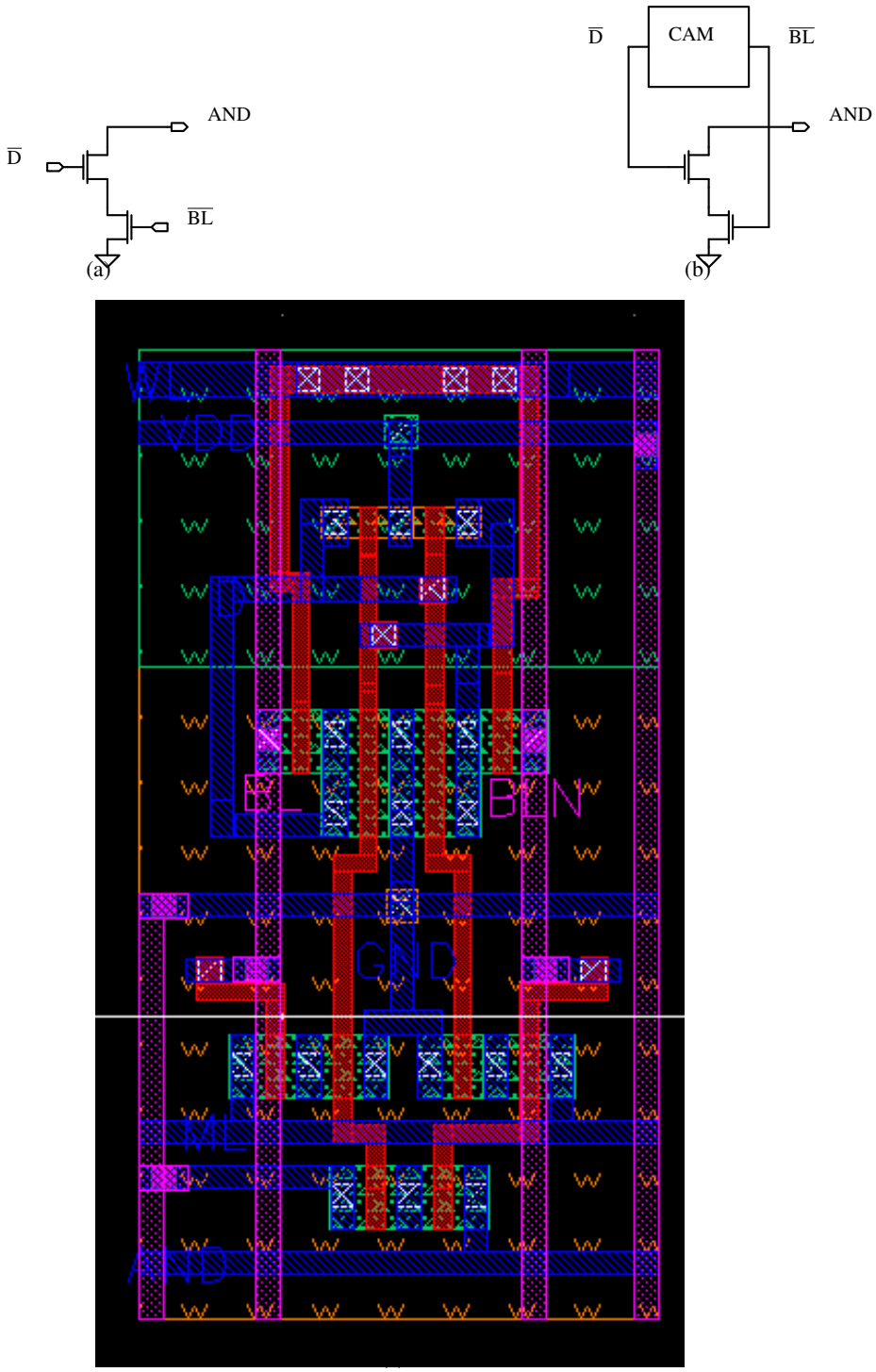


图11：专用与门和存储单元。考虑到算法特性，可以仅使用另外两个晶体管来实现与操作。**(a)**专用与门。**(b)** 嵌入逻辑门的存储单元。**(c)**单元布局。

由于存储单元只有在掩码位 M 等于逻辑“1”（即 BL=“1”、 BL=“0”）时才必须被选中，因此当 $\overline{M}$=为“0”（即 BL=“0”、 BL=“1”）时，应将其与与线断开；因此，只需在下拉晶体管的源极上增加一个栅极连接至 BL的页脚晶体管，即可将其禁用。然而，由于静态（图10a）和动态（图8a）栅极的其中一个输入连接至 BL 而不是 BL，因此在此情况下采用了不同的掩码向量编码，将逻辑“0”而非逻辑“1”用作掩码位的有效值；这样，图11a中的页脚晶体管便可连接至 BL；因此，就与存储器信号线的连接而言，三种变体是等效的，从而可以进行适当比较。

关于下拉晶体管，其栅极在静态（图10a）和动态（图8a）栅极中连接到与逻辑门的输出；而在图11a中，它连接到单元内容 D的取反值；事实上，一旦选中存储单元，算法只需知道单元内容是否等于‘0’或‘1’，后者可以直接连接到下拉晶体管的栅极。这样，当 D=‘1’（D=‘0’）时，与逻辑门被禁用，线路被充电至逻辑“1”；当 D=‘0’（D=‘1’）时，下拉晶体管被使能，线路不被充电，并感测到逻辑“0”。

可以注意到，在图 11b 中，存储单元的输出引脚标记为 AND 而非 $\overline{AND}$：这是因为与运算结果不会被下拉晶体管反相。事实上，未选中列的下拉晶体管通过图 9b 所示的机制禁用，因此，选中列上的与运算结果可以直接传输到线路上。如果选中存储单元的内容 D_i 等于“1”，线路就会充电，并将 $D_i \cdot M_i =$“1”（M_i 是有效掩码位）登记为输出；否则，线路不会充电，输出 $D_i \cdot M_i =$“0”。因此，存储单元核心与与线之间无需额外的隔离级；而图 8b 和图 10b 中的静态与和动态实现则需要隔离级，因为当相应的存储单元未被选中时，必须将逻辑门的输出与线路断开。图 11b 所示专用与单元的真值表见表2。

$\boldsymbol{D}$	BL	D	位线	与
0	0	1	1	0
-	1	-	0	$0 \rightarrow 1$
1	0	0	1	$0 \rightarrow 1$

表2：图 11b 中专用与存储单元的真值表。评估此函数时，需要记住， BL 并非真正的数据信号，而是一个选择信号，它允许将存储单元内容 D 报告到线路上。每当 BL=为‘0’时，与逻辑门便会禁用，线路被充电至‘1’（特别是，当 D=为‘0’时，会阻止下拉电路对线路放电）。

图 11b 中的专用存储单元具有最低的面积开销（额外晶体管数量最少）。然而，这些存储单元能够执行适当的与逻辑运算，这对于实现其他算法可能很有用；尽管如此，专用存储单元电路表明，通过适当的优化，可以大幅降低逻辑电路带来的面积开销。

图8b 和图10b中的动态单元和静态单元分别具有相同数量的晶体管，但静态单元占据的面积更大，这是由于逻辑栅极中的上拉 pMOS晶体管比动态单元中的预充电 pMOS 晶体管大得多；然而，静态单元的工作不需要 (PRE) 信号，因此单元和行面积更小。

3.4 虚设线感测方案

对于 LiM 阵列，采用与内容可寻址存储器相同的虚设线感测方案：利用哑单元生成一条虚设存储线，作为所有与感测放大器（ANDSA）的参考。

图12展示了 LiM 各种方案中的哑单元：

- 在图12a中，展示了动态逻辑版本的哑单元。在该栅极中，每个存储单元连接两条行信号：与线 AND信号和预充电信号 PRE；因此，必须包含连接到这些信号的晶体管。
- 在图12b和图12c中，展示了静态与专用变体。由于这些变体不需要额外的行信号，电路中仅存在与线引脚。

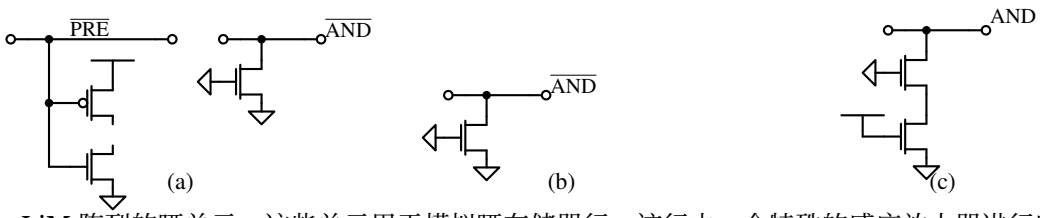


图12：LiM 阵列的哑单元。这些单元用于模拟哑存储器行，该行由一个特殊的感应放大器进行感测，并驱动其他感应放大器，以降低在整个阵列上执行与操作所涉及的总体能耗。**(a)** 动态与哑单元。**(b)** 静态与哑单元。**(c)**专用与哑单元。

4 存储器阵列表征

这些存储单元按存储器阵列的形式组织，以评估其性能。我们针对阵列的不同高度和宽度对存储器电路进行仿真，以获得适用于多种存储器大小的测量结果。所有仿真均在 Cadence Virtuoso 中以原理图级别进行，并使用 SPECTRE仿真引擎。

为了考虑互连寄生效应的影响，将哑行和列的版图生成并纳入仿真网表中。

具体而言，使用 32 位宽的行和列作为基本模块来构建阵列：提取其版图并转换为网表，然后将网表纳入测试平台。

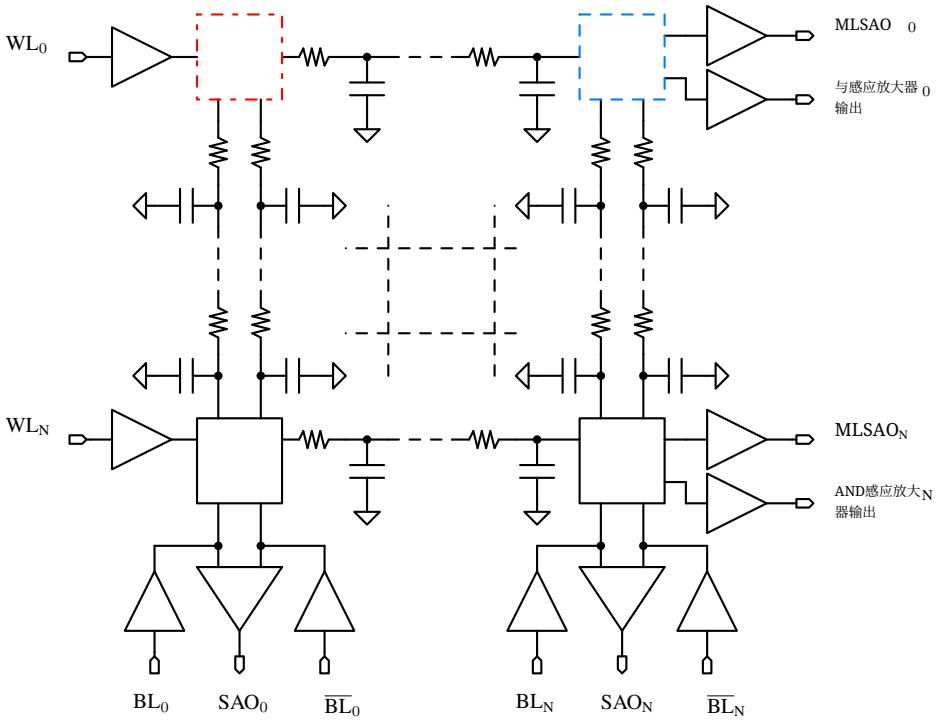


图13：每种存储器操作的最坏情况延迟。为清晰起见，大多数存储单元被省略，互连由 RC 电路表示，这些电路在测试平台中由提取的行/列网表替代。与读取和写操作相关的存储单元以蓝色突出显示，并以虚线标示；它距离字线和位线驱动器以及感应放大器最远；与搜索和与操作最坏情况相关的存储单元以红色突出显示，并以点划线标示；它距离MLSA和ANDSA最远。

在考虑读操作时，必须将待读取存储单元与字线驱动器及感应放大器之间的距离考虑在内，以衡量存储单元的位置对性能的影响。请看图 13 所示的示意图：

- 激活字线以选择存储单元时，存储单元距离驱动器越远（即位于图 13 中的最后一列），选择延迟就越大，这是因为驱动器需要提供更高的容性-电阻性负载；因此，与该存储单元相关的读延迟是阵列中可能出现的最大值。

- 使用感应放大器（SA）感测位线时，存储单元距离感应放大器输入端越远（即位于图 13 中的第一行），该存储单元在感应放大器引脚上产生电压变化所需的时间就越长。

因此，最坏情况下读延迟对应的存储单元是图 13 中位于第一行、最后一列的存储单元（以蓝色突出显示），读操作性能也据此存储单元进行评估。对于写操作的最坏情况，可以参考图 13 中的示意图进行类似分析：

- 对于字线激活和存储单元选择，读操作的考虑同样适用：选择延迟最大的是最后一列上的存储单元。

- 在将待写入的数据置于位线上时，为评估最坏情况，需要考虑距离位线驱动器输出端最远的存储单元。在图 13中，这些存储单元位于第一行。

基于这些原因，与写操作最坏情况感测延迟对应的存储单元位于第一行和最后一列（以蓝色突出显示），如图 13 所示。

关于与运算和搜索操作，请考虑图13 中的示意图：由于MLSA 和ANDSA都位于行的末端，距离它们最远的存储单元是第一列中的存储单元，图中以红色标出。因此，对于该存储单元，与运算和搜索操作均对应最坏情况。行位置不会影响搜索和与运算的性能，即使这些操作与位线驱动器相关联：这是由于该架构所采用的特殊感测方案。事实上，由于采用电流节省方案，存储单元的下拉晶体管在预放电阶段无需被禁用，因此可以在此周期内将掩码向量加载到位线上，从而使所有存储单元在评估阶段之前就已完成配置；这样，搜索和与运算的性能便不取决于该行与位线驱动器输出端之间的距离。

由于正确测试存储器阵列所需的存储单元非常少，因此没有必要在仿真测试平台中包含所有存储单元：根据之前的考虑，通过从阵列中移除所有未被测试的存储单元，将该阵列简化为最坏情况模型，从而缩短仿真时间，并因此加快设计参数的调试；随后，推导出图 14 所示的电路模型，并在仿真过程中使用。

模型中仅考虑两条存储器线路：第一行和最后一列。这是因为所有存储器操作的关键单元都位于这些线路上；此外，由于只测试两个存储单元，其余存储单元可以替换为虚拟版本，其电路如图 14 所示。

哑单元分为行单元和列单元：在哑行单元中，电路中仅包含与行信号（字线；匹配线；与线；仅对动态与单元而言还包括预充电线路）相连接的晶体管；而在哑列单元中，则仅保留与位线相连接的晶体管。这样，在计入线路信号上存在存储单元的同时，还能从电路中移除许多晶体管，从而大幅缩短大型存储器阵列的仿真时间。

在 Cadence Virtuoso 中，采用了图 15 所示的测试平台。该原理图适用于 LiM 阵列，但也可以通过移除某些模块并替换存储单元电路，将其简化并适配至 CAM 和 SRAM 架构，因为 LiM 存储器集成了这些功能。

从图 15 可以看出，位线驱动器仅设置在最后一列，因为只有该线路用于测试读取和写入操作；而对于第一列，则采用理想开关和电压发生器来修改单元内容，因为在该列上仅测试行操作，例如与和搜索操作。

在图15所示的原理图中，还可以注意到，每个虚拟感应放大器的输出端都增加了一个称为“虚拟负载”的模块：这些模块用于模拟实际存储器阵列中各行的所有感应放大器的存在。如第 3节所述，虚拟感应放大器必须驱动嵌入每个真实感应放大器中的所有或逻辑门；由于图 14 所示模型中只有一行配备了 MLSA和 ANDSA，因此必须对其他行的感应放大器进行建模，以考虑它们对实际存储器阵列中性能的影响。

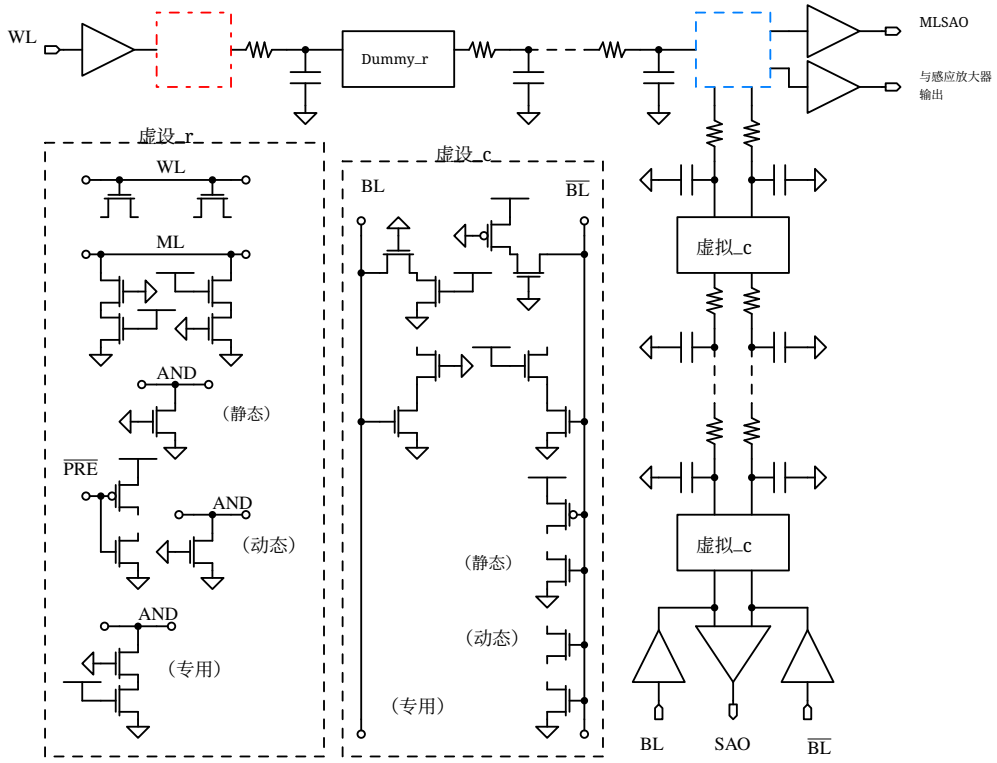


图 14：阵列模型。第一行对应最上方的一行，第一列对应最左侧的一列。阵列中仅包含用于读取、写入、搜索和与操作的关键存储单元，这些单元采用从版图中提取的电路；其余单元则由从版图中提取的哑行和哑列替代，其中仅包含重要的晶体管（即与原始阵列中的行/列信号相连的晶体管）。

存储器阵列。出于这一原因，由或门输入部分构成的虚拟负载连接到感应放大器的输出端。

虚拟负载模块的电路如图 16 所示。它由多个共享同一输入的或逻辑门组成，或门的数量与阵列中的行数一致。这些并非实际的栅极；为尽可能减少测试平台网表中的元件数量，仅包含与输入相连的晶体管。

图 15 展示了一些额外模块：预充电电路用于在读操作前对位线进行预充电；“延迟感应放大器”电路用于延迟感应放大器的使能信号，以测试读操作，因为采用了电压锁存感应放大器 [37] 。

5 仿真框架

为表征大型存储器阵列，我们采用脚本方法：先手动表征设计，再自动生成电路网表。

大型阵列的仿真方法如图 17 所示，包含以下步骤：

- 手动设计存储器阵列和感测电路，并通过仿真小型阵列（32x32 个存储单元）对其进行表征。
- 生成并提取存储单元和行的版图。采用 32 位宽的行和列作为基本模块，以构建最终阵列。
- 电路网表提取完成后，会按照精确的规范编写脚本，使电路参数化，以适应其尺寸（阵列的高度和宽度）。

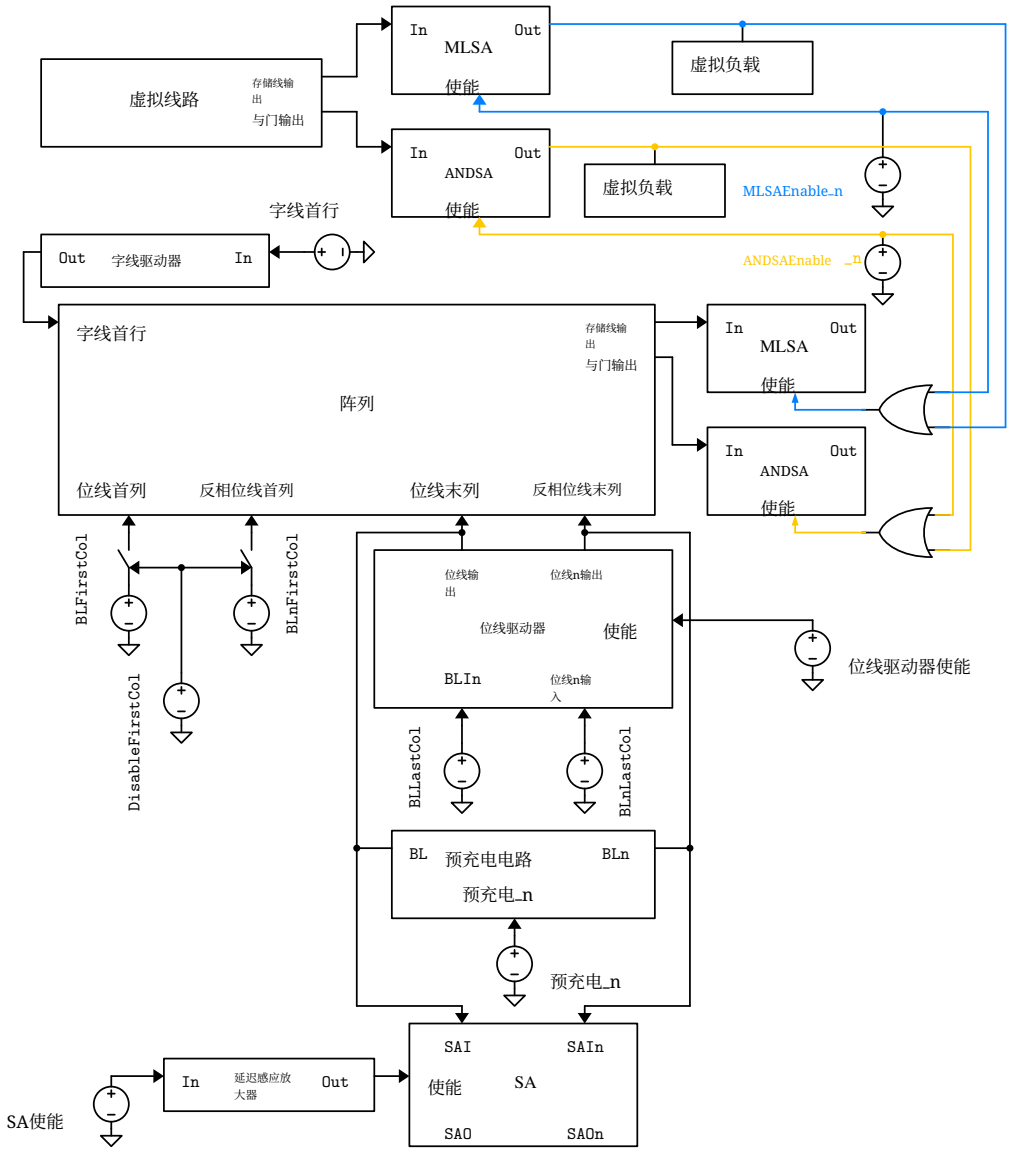


图 15：测试平台。与内容可寻址存储器和与功能相关的连线分别以蓝色和橙色突出显示，以便于查看。

- 使用脚本生成参数化的 Cadence Virtuoso 测试平台，通过 SPECTRE 仿真引擎，可针对任意宽度和高度对电路进行特性分析。
- 测试平台的输入激励会从用户提供的待仿真操作序列自动生成。

- 使用 Cadence Virtuoso 的 SPECTRE 引擎对电路进行仿真。
- 通过测量与每次存储器操作相关的能耗和延迟，提取阵列性能。

图 18 展示了名为 ALiAS（模拟逻辑存内阵列仿真）的脚本工作流程。ALiAS 接受输入：

- 必须手工设计的基本模块（即存储单元和感应放大器）的网表。

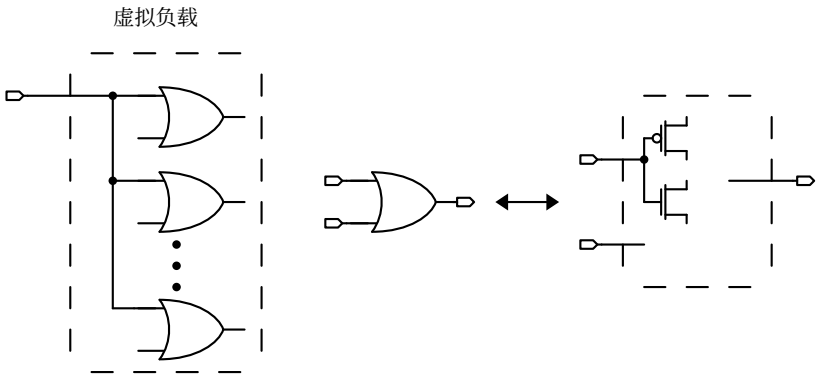


图 16：用于虚拟感应放大器的虚拟负载。该负载用于模拟多个或门的输入部分，这些或门嵌入每个实际的 MLSA/ANDSA 中，以便将其对阵列中感测性能的影响考虑在内。

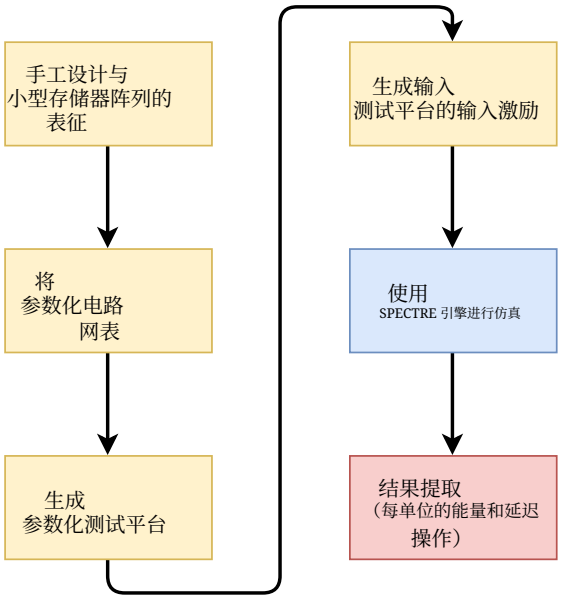


图 17：仿真流程。从手工设计的小型阵列电路出发，通过图中所示的算法实现大型阵列的仿真。

- 要模拟的阵列所需的特性：类型（静态随机存取存储器、内容可寻址存储器、三种存内逻辑变体）和大小（宽度和高度）。
- SPECTRE 的仿真参数（例如与仿真关联的最大计算线程数、图形界面模式等）。
- 为仿真选定的时钟周期，默认为 1 ns。

根据这些信息，生成阵列的网表和测试平台，运行 SPECTRE 仿真，提取性能测量结果（尤其是与每种存储器操作相关的能耗和延迟），并以不同格式（柱状图和 CSV 文件）保存。通过这种方法，ALiAS 能够加快采用定制存储单元拓扑的存储器阵列在原理图层级的设计与仿真。

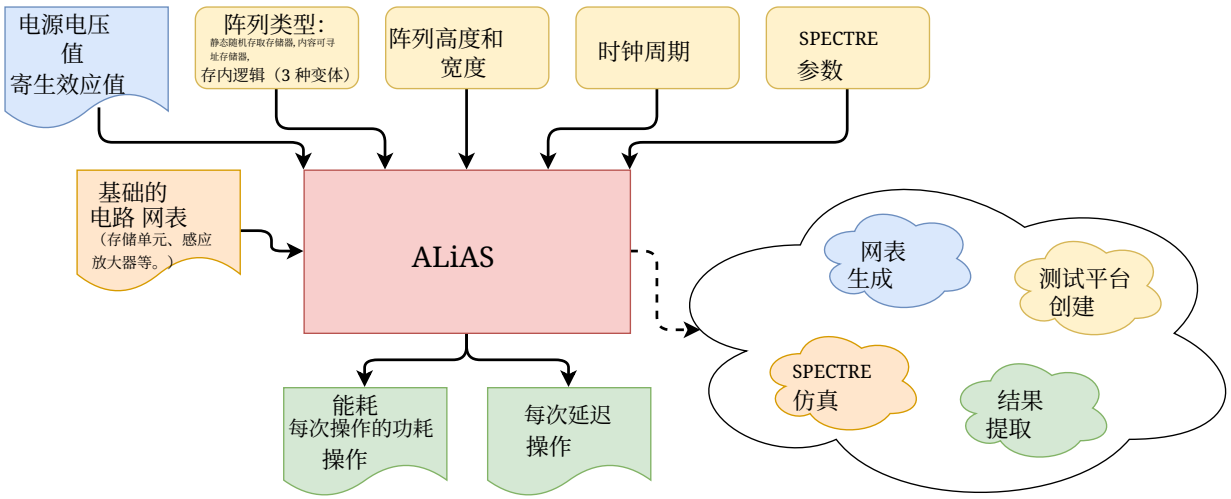


图 18：所采用的脚本方法称为 ALiAS（模拟逻辑存内阵列仿真）。从阵列特性（类型和尺寸）、仿真条件（电路参数、时钟周期、SPECTRE 配置）以及基本电路（存储单元、行和列）的版图提取网表开始，在 SPECTRE 中执行仿真并评估阵列性能。

6 结果与讨论

为评估存储器阵列的性能，从 SPECTRE 仿真中提取每种存储器操作的能耗和延迟。能耗通过对每个仿真周期内阵列的瞬时功耗进行积分来测量：

$$E_{operation} = \int_{cycle} p(t)dt$$

每个阵列均使用 Cadence Virtuoso 中的 SPECTRE 仿真器，在电源电压为 $V_{DD} = 1$V、时钟周期为 $t_{ck} = 4$ns 的条件下进行仿真。

能量-延迟积 [pJ · ps]				
存储器	**操作**			
	写入	**读取**	**搜索**	**AND**
SRAM	134	118	—	—
CAM	275	230	236	—
AND SP	309	260	717	98
AND DYN	596	451	1152	2008
AND ST	961	641	601	76

表3：每个存储器阵列中各项存储器操作对应的能量–乘积。数据提取自图19中的256x256阵列，该阵列用作案例研究。

输入图19中，展示了每个存储器阵列每次操作的能量–延迟乘积。考虑了四种不同的存储器大小：64x64、128x128、192x192和256x256，分别作为行和列。这些数值的选取旨在估计阵列性能如何随其大小变化，其中所采用的存储器大小通常为文献中测试芯片所采用的数值 [1, 2, 3, 34]。输入表3中，展示了能量-延迟积数值，并以图19中的256x256阵列作为参考案例。
下文将对每种操作进行分析并相互比较。

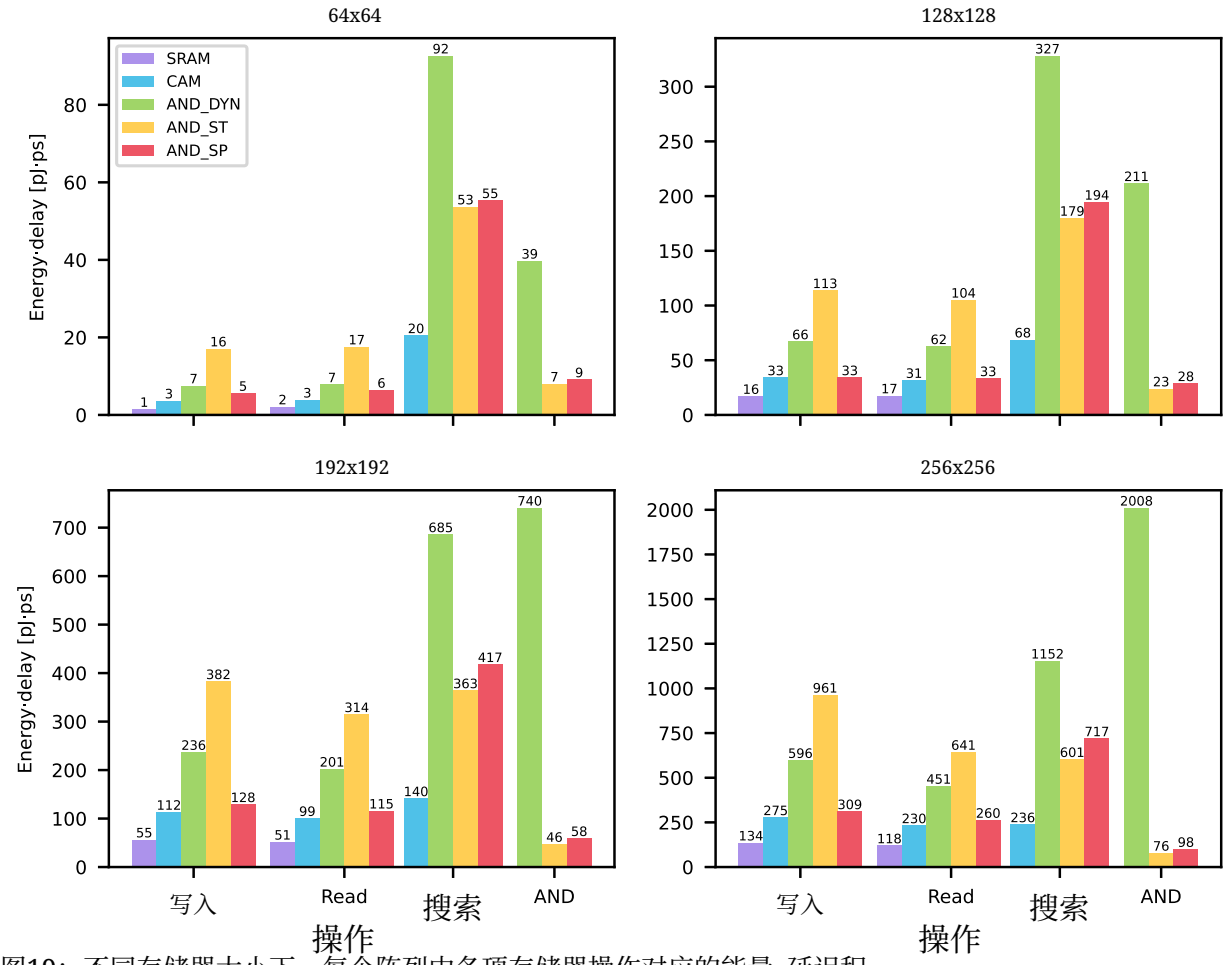


图19：不同存储器大小下，每个阵列中各项存储器操作对应的能量–延迟积。

6.1 读操作

从图19可以看出，存内逻辑（图中 AND_SP, AND_DYN, AND_ST分别表示专用、动态与和静态与单元）和内容可寻址存储器在所有存储器大小下的表现都不如 SRAM 阵列。这是因为这些架构所采用的单元电路比 SRAM 的单元电路复杂得多（即晶体管数量更多、晶体管尺寸更大、互连更多）。

输入表4中，阵列之间与读操作相关的能量–延迟积差异以百分比表示，如下所示。例如，对于内容可寻址存储器，测得其能量–延迟积值比静态随机存取存储器高 94.41%；对于静态与存储器，所得能量–延迟积值比专用与高 40.57%。这些数据提取自图19中的 256x256 阵列，该阵列在下文中用作案例研究。

通过研究各存储器的电路，可以解释它们在性能上的差异。图20展示了这些电路，其中仅绘出了连接到位线的存储单元晶体管。事实上，众所周知，要从类似静态随机存取存储器的存储单元中读取数据，必须通过字线访问该存储单元，并让它对其中一条位线放电，以确定其内容；在放电电流相同的情况下，位线对应的等效电容负载越大，放电时间就越长。由于位线电容由互连版图以及连接到位线的晶体管决定，因此，连接到位线的存储单元晶体管数量越多，读取性能就越差。

读取的能量-延迟积相对变化					
	SRAM	**CAM**	**ANDSP**	**ANDDYN**	**ANDST**
SRAM	—	—	—	—	—
CAM	+94.91%	—	—	—	—
AND 稀疏	+120.34%	+13.04%	—	—	—
AND DYN	+282.2%	+96.08%	+73.46%	—	—
AND ST	+443.22%	+178.69%	+146.54%	+42.13%	—

表 4：各阵列读取能量–延迟积的百分比差异。每个数值表示相应行中存储器的能量–延迟积相对于相应列中存储器的能量–延迟积的增幅，以百分比表示。为避免表格解读产生歧义，省略了部分数值（即每个百分比均以列中的存储器为基准计算，且每对存储器仅比较一次）。数据摘自表3。

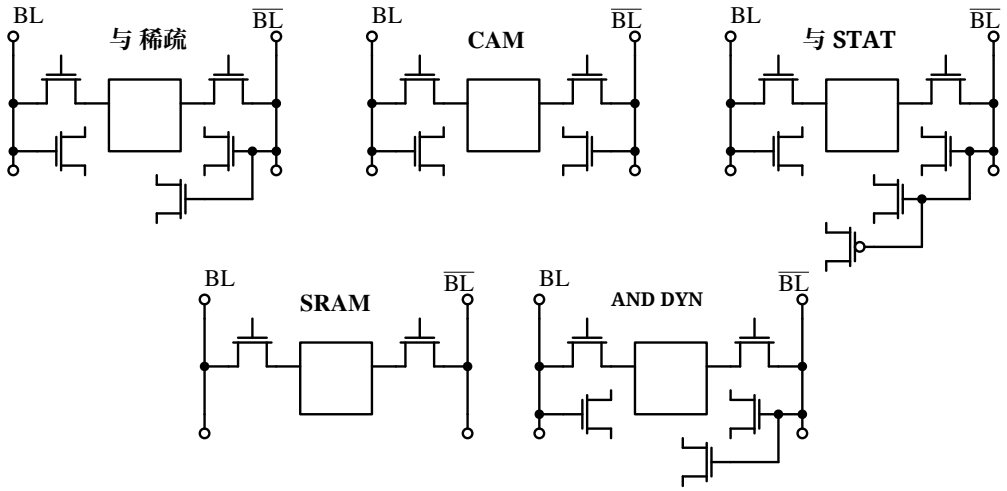


图20：每种存储单元类型在位线上的晶体管负载。

根据表 4 中的数据可以发现，性能最差的存储器是静态与存储器，它连接到位线上的晶体管数量也最多（图20）。这解释了为什么性能最佳的存储器是静态随机存取存储器（SRAM）：从电路角度看，它的结构最简单，因此相关的位线电容最低。类似的考虑也可以用来解释其他存储单元之间的差异。

从图20可以看出，尽管专用与存储单元和动态存储单元连接到位线（尤其是 BL）上的晶体管数量相同，但后者的性能不如前者；这是因为还必须考虑这些存储单元的版图，分别如图8c和图11c所示，其中图8c对应动态与存储单元，图11c对应专用与存储单元。可以观察到，动态与电路更为复杂，包含更多晶体管和互连，导致最终电路中的寄生效应更多，从而减慢单元读取操作，并增加相应的功耗。

6.2 写入操作

输入表5中，阵列之间以百分比表示的写入操作能量-延迟积差异如图所示。对读操作所作的相同考虑同样适用，因为写入和读取性能都大致由存储器电路和版图决定。

6.3 搜索操作

在表6中，阵列之间以百分比表示的搜索操作能量-延迟积差异如下所示。

写入的能量-延迟积相对变化					
	SRAM	CAM	AND SP	AND DYN	AND ST
SRAM	—	—	—	—	—
CAM	+105%	—	—	—	—
与 稀疏	+130.6%	+12.36%	—	—	—
与 DYN	+344.78%	+116.72%	+92.88%	—	—
与 静态	+617%	249.45%	+211%	+61.24%	—

表5：各阵列之间写入能量–延迟积的百分比差异。每个数值表示相应行中的存储器相对于相应列中的存储器，其写入能量–延迟积的增幅（以百分比表示）。数据摘自表3。

搜索的能量-延迟积相对变化				
	CAM	AND SP	AND DYN	AND ST
CAM	—	—	—	—
AND SP	+203.81%	—	—	—
AND DYN	+388.13%	+60.67%	—	—
AND ST	+154.66%	-19.30%	-91.68%	—

表6：各阵列搜索能量–延迟积的百分比差异。每个数值表示相应行中的存储器的能量–延迟积相对于相应列中的存储器的能量–延迟积增加的百分比。数据摘自表3。

可以发现，LiM 阵列在搜索操作中的表现不如 CAM。这可以通过考察存储单元的版图来解释：由于 LiM 存储单元更为复杂，其搜索功能会受到更多寄生效应的影响。

以动态与单元为例，其版图下部如图8c所示。可以看出，内容可寻址存储器电路与与电路紧邻布置；因此，与原始CAM单元相比，匹配线的寄生效应增大，导致搜索操作的延迟和功耗增加。专用与单元和静态与单元也有类似情况。

可以观察到，在 LiM 阵列中，搜索操作性能最佳的是静态 AND 阵列。这似乎有悖直觉，因为静态与门是 AND 单元中最复杂的；然而，通过考察存储单元的版图即可解释这一现象。

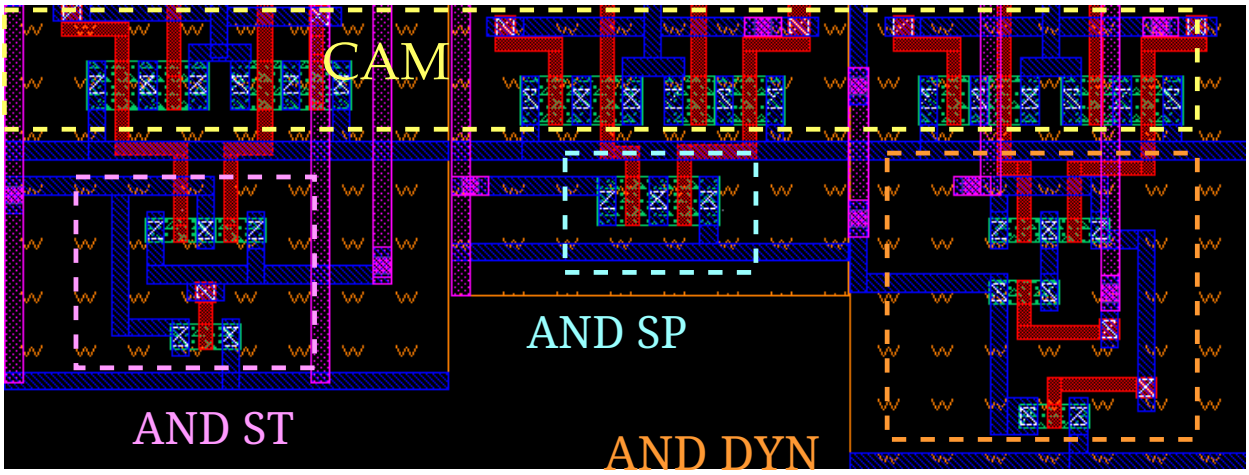


图21：静态与单元、专用与单元和动态与单元的版图下部。

图21并排展示了静态、专用与和动态与单元的下部区域。考察版图中以颜色标出的与门区域可以发现，版图最复杂（就晶体管数量和局部互连而言）的是动态与单元，以橙色标出；其次是专用与单元，晶体管数量较少但尺寸更宽，以青色标出；最后是静态与单元，以粉色标出。因此，该单元的性能最差。

就特殊用途存储单元而言，其电路似乎比静态存储单元的电路简单，但应注意，特殊用途电路中的晶体管比静态存储单元中的晶体管更宽；这会导致寄生电容增大，并使搜索操作的性能变差，因为这些晶体管通过栅极与实现 CAM 功能的晶体管相连。

6.4 与操作

与操作的能量-延迟积相对变化

	AND SP	AND DYN	AND ST
AND SP	—	—	—
AND DYN	+1948.98%	—	—
AND ST	-28.95%	-2542.1%	—

表7：各阵列之间与运算能量–延迟积的百分比差异。每个数值表示对应行的存储器与对应列的存储器相比，其能量–延迟积增加的百分比。数据摘自表3。

在表7中，显示了阵列之间与操作能量-延迟积（以百分比表示）的差异。

可以注意到，性能最佳的阵列是静态与阵列。这可以通过参考存储单元电路来解释。

由于其输出电路更简单（静态与见图10，专用与见图11），静态存储单元的性能优于专用与：静态门只有一个晶体管连接到与线，而专用与有两个串联的NMOSFET与其相连；这会导致更高的延迟和功耗。

静态与存储单元的性能也优于动态单元，因为后者采用动态 CMOS 逻辑实现，而前者采用静态CMOS逻辑实现。事实上，观察图 8 中动态与存储单元的电路可以发现，一旦通过 EN 使能对与线的感测，动态栅极需要一定时间才能将其输出（标记为 AND）放电，从而关闭下拉电路。在此期间，下拉电路处于导通状态，阻止与线（标记为 AND）由 ANDSA 充电。这会导致能耗和感测延迟均增加。

考虑图10中的静态与单元电路，可以发现，在使能感测之前，与门的输出已经处于地电压，这是第4节所讨论的原因所致。在与操作开始时，下拉电路已经禁用，这意味着线路会立即开始充电，而不会有电流流向地。此外，每次执行与操作时，所有与门都会反转其输出，以关闭连接到与线的下拉晶体管；这会导致能耗大幅增加，如表7所示。

6.5 不同操作之间的比较

本节对所执行的操作进行比较，并分析它们之间的关系。 r.

从图19可以发现，随着阵列尺寸增大，写入性能的恶化程度高于读取性能。这主要是因为，读操作并不意味着其中一条位线必须完全换向（一条位线只需放电到足以让感应放大器正确读取存储单元的值即可），而写操作则需要如此，因为必须在其中一条位线上施加强逻辑“0”，才能强制将期望值写入存储单元；因此，测得写操作的能耗高于读操作。
在表 8 中，对各存储器的读取和写入性能进行了比较，衡量指标为能量–延迟积。可以看出，读取和写入性能差异最大的是静态与存储器。这是因为随着阵列尺寸增大，相应位线的电容负载也会显著增加。

能量-延迟积相对变化：写入与读取						
				Read		
		SRAM	CAM	与 稀疏 与 静态 与 DYN		
写入	SRAM	+13.56%	—	—	—	—
	CAM	—	+19.56%	—	—	—
	与 稀疏	—	—	+18.85%	—	—
	AND ST	—	—	—	+49.92%	—
	AND DYN	—	—	—	—	+32.15%

表8：各存储器中写入和读取能量-延迟积的百分比差异。每个数值表示相应行中的存储器的能量-延迟积相对于相应列中的存储器的能量-延迟积的增加百分比。数据提取自表3。

而非线性增长；由于静态与单元最为复杂，因此对于大型阵列（例如图19中的256x256阵列），测得的写入和读取性能差异更大，而在其他阵列中，差异较小。事实上，在表8中，写入/读取差异值与存储单元电路的复杂度相对应：性能最佳的存储器是静态随机存取存储器，其次是内容可寻址存储器、专用存储器、动态与和静态与。

能量-延迟积相对变化：搜索与写入对比						
				写入		
		SRAM	CAM	ANDSP	ANDDYN	ANDST
搜索	CAM	+76.12%	-16.52%	—	—	—
	AND SP	+435.07%	+160.72%	+132.04%	—	—
	AND DYN	+759.7%	+318.91%	+272.81%	+93.29%	—
	AND ST	+348.51%	+118.54%	+94.5%	-91.68%	-59.9%

表 9：各阵列中读取与搜索的能量-延迟积百分比差异。每个数值表示对应行的存储器相对于对应列的存储器，其能量–延迟积增加的百分比。数据摘自表3。

能量-延迟积相对变化：搜索与读取						
				Read		
		SRAM	CAM	ANDSP	ANDDYN	ANDST
搜索	CAM	+100%	+2.61%	—	—	—
	AND SP	+507.63%	+211.74%	+175.77%	—	—
	AND DYN	+876.27%	+400.86%	+343.08%	+115.43%	—
	AND ST	+409.32%	+161.3%	+131.15%	+33.26%	–6.65%

表 10：各阵列的写入与搜索能量–延迟积之间的百分比差异。每个数值表示相应行所对应存储器的能量–延迟积相对于相应列所对应存储器的能量–延迟积的增幅，以百分比表示。数据摘自表3。

表 10 和表 9 分别报告了搜索操作与写入操作、读取操作之间的能量–延迟积比较。可以看出，在所有情况下，搜索操作的表现都不如 SRAM 阵列的读/写操作。不过，对于静态与阵列和 CAM 阵列，与同一阵列的写入操作相比，搜索操作的能量–延迟积分别低 16.52% 和 59.9%；至于读取操作，CAM 的搜索操作仅差 2.61%，而静态阵列则好 6.65%。

从图19可以看出，对于静态与和专用与阵列，与操作的表现优于搜索操作。这是因为与操作所涉及的硬件比搜索操作的硬件更简单：在 CAM 单元（图 2a）中，有两条由两个串联晶体管构成、连接至匹配线的下拉通路；而在与单元（图 10b、图 11b 和图 8b）中，只有一条下拉通路。因此，功耗和延迟都较低。

能量-延迟积相对变化：与搜索的比较					
			搜索		
		CAM	AND 稀疏	AND DYN	AND 静态
AND	AND SP	-140.81%	-631.63%	—	—
	AND DYN	+750.85%	+180.05%	+74.3%	—
	AND ST	-210.52%	-843.42%	-1415.79%	-690.79%

表11：阵列之间 AND 和搜索能量–延迟积的百分比差异。每个值表示对应行的存储器的能量–延迟积相对于对应列的存储器的能量–延迟积的增幅，以百分比表示。数据摘自表3。

表11比较了与操作和搜索操作的能量–延迟积。可以看出，除动态与的情况外，与操作的表现始终优于搜索操作。动态与的情况并非如此，这是因为栅极采用了动态 CMOS 逻辑实现，每次执行与操作时，都会使所有行单元中的与门发生切换。这会显著增加与功能相关的能耗。

能量-延迟积相对变化：与写入的比较						
				写入		
		SRAM	CAM	ANDSP	ANDDYN	ANDST
AND	AND SP	-36.7%	-180.61%	-215%	—	
	AND DYN	+1398%	+630.2%	+549.84%	+236.91%	–
	AND ST	-76.31%	-261.84%	-306.57%	-684.21%	-1164%

表12：各阵列之间写入与与操作的能量–延迟积百分比差异。每个数值表示对应行中的存储器相对于对应列中的存储器，其能量–延迟积增加的百分比。数据摘自表3。

能量-延迟积相对变化：与操作与读取的比较						
				Read		
		SRAM	CAM	ANDSP	ANDDYN	ANDST
AND	与 稀疏	-20.41%	-134.69%	-165.31%	—	—
	AND DYN	+1601.7%	+773.04%	+672.31%	+345.23%	—
	AND ST	-55.26%	-202.63%	-242.1%	-1164.47%	-743.2%

表13：各阵列在读取和与操作能量-延迟积方面的百分比差异。每个数值表示相应行存储器的能量-延迟积相对于相应列存储器的能量-延迟积的增幅，以百分比表示。数据摘自表3。

关于与操作和传统操作，从图19可以看出，对于256x256的阵列尺寸，静态阵列和专用阵列中的与操作性能优于SRAM 阵列中的读取和写入操作。这是因为得益于额外的存储单元电路，执行与操作无需访问单元内容，从而实现更低的延迟和能耗；事实上，正如前文所述，SRAM 核心电路的效率很低。

表12中详细列出了与操作和写入性能之间的比较。可以注意到，除动态与情况外，与操作始终优于写入操作，即使与传统静态随机存取存储器架构进行比较也是如此：对于专用情况，与静态随机存取存储器写入相比，与操作的能量–延迟积降低了36.7%；而在静态与情况下，降幅为76.31%。

在表13中，对与和读取性能进行了比较分析。同样在这种情况下，与操作始终优于读取操作，动态与情况除外，即使与静态随机存取存储器相比也是如此：对于专用与，与静态随机存取存储器读取相比，测得与能量–延迟积降低了20.41%；对于静态与情况，降低了55.26%。

这意味着，从能耗和延迟的角度来看，执行内存中操作（例如与操作）更加便利，即使与传统SRAM存储器相比也是如此。需要强调的是，在此分析中，与从阵列提取数据相关的开销——即由于存储器与CPU之间的数据传输，以及由于处理过程中进行的数据处理而产生的能耗和延迟贡献——并未被纳入考量；因此，内存中方法所带来的优势被严重低估了。

7 结论

本研究设计了一种包含 3 种存储单元变体的 LiM 阵列，并在 Cadence Virtuoso 中完成物理层实现：绘制存储单元版图，并从中提取寄生网表。通过评估 LiM 硬件对标准存储器操作带来的开销，将所得电路与传统存储器阵列（如 SRAM 和 CAM）进行了比较。

结果显示，LiM 阵列的读写存储器操作能耗和延迟均有所增加（在最佳情况下，读操作相较于 SRAM 和 CAM 分别增加 +120.34% 和 +13.04%）。结果还表明，即使不考虑将数据传输至 CPU 所产生的片外传输能耗和延迟，在最佳情况下，存内处理的成本（以 LiM 操作的能量–延迟积表示）仍比 SRAM存储器读操作的能量–延迟积低 55.26%。这意味着，尽管存在前文所述的额外硬件复杂度等缺点，直接在存储器中处理数据也远比将数据从阵列中取出并在 CPU 中进行计算更为合算。

这些结果表明，逻辑存内阵列通过在存储单元中添加计算元件来对其进行改造，最适用于读取和写入操作较少、存内逻辑运算较多的应用。这类阵列可作为算法加速器设计的合适替代方案，也可用作低密度辅助传统存储器，以存储数据。

参考文献

[1] A. Fritsch、M. Kugel、R. Sautter、D. Wendel、J. Pille、O. Torreiter、S. Kalyanasundaram 和D. A. Dobson，“采用高性能电流检测放大器降低交流电流浪涌的14 nm SOI FinFET技术4 GHz低延迟三态内容可寻址存储器”，载于*ESSCIRC 2015*会议——第*41*届欧洲固态电路会议（*ESSCIRC*），第343–346页，2015年。[2] E. Seevinck、P. van Beers 和 H. Ontrop，“用于高速超大规模集成电路的电流模式技术及其在CMOS静态随机存取存储器电流检测放大器中的应用”，*IEEE*固态电路学报，第26卷，第4期，第525–536页，1991年。[3] N. Mohan、W. Fung、D. Wright 和 M. Sachdev，“低功耗三态内容可寻址存储器的设计技术与测试方法”，*IEEE*超大规模集成（*VLSI*）系统汇刊，第14卷，第6期，第573–586页，2006年。[4] B. Alimkhanuly、J. Sohn 和 I. e. a. Chang，“用于神经形态计算的基于石墨烯的三维异或非阈值可变电阻存储器与三值精度”，*npj*二维材料，第5卷，应用第55号，2021年。[5] M. Le、T. K. H. Pham 和 S. N. Truong，“用于神经形态图像识别的单交叉阵列架构的噪声与忆阻值变化容差”，*Micromachines*，第12卷，第6期，2021年。[6] H. Abbas、Y. Abbas、G. Hassan、A. S. Sokolov、Y.-R. Jeon、B. Ku、C. J. Kang 和 C. Choi，“ 原子层沉积氧化铪忆阻器突触阵列中阈值开关与存储开关特性的共存及其在高能效神经形态计算中的应用”，*Nanoscale*，第12卷，第14120–14134页，2020年。[7] S. Angizi 和 D. Fan，“ReDRAM：用于加速批量按位运算的可重构DRAM内处理平台”，载于*2019*年*IEEE/ACM*国际计算机辅助设计会议（*ICCAD*），第1–8页，2019年。[8] W. A. Simon、Y. M. Qureshi、M. Rios、A. Levisse、M. Zapater 和 D. Atienza，“BLADE：面向边缘设备的缓存内计算架构”，*IEEE*计算机汇刊，第69卷，第9期，第1349–1363页，2020年。[9] H. Jiang、S. Huang、X. Peng、J. W. Su、Y. C. Chou、W. H. Huang、T. W. Liu、R. Liu、M. F. Chang 和 S. Yu，“一种用于深度神经网络片上训练的双向SRAM阵列加速器”，载于*2020*年第*57*届*ACM/IEEE*设计自动化会议（*DAC*），第1–6页，2020年。[10] Z. Jiang、S. Yin、J. Seo 和 M. Seok，“C3SRAM：基于稳健电容耦合计算机制的存内计算SRAM宏单元”，*IEEE*固态电路学报，第55卷，第7期，第1888–1897页，2020年。

[11] K. Lee、J. Jeong、S. Cheon、W. Choi 和 J. Park，“具有可重配置位精度的位并行 6T 静态随机存取存储器存内计算”，载于*2020*年 第*57*届 *ACM/IEEE*设计自动化会议 *(DAC)*，第 1–6 页，2020年。[12] H. Jiang、X. Peng、S. Huang 和 S. Yu，“CIMAT：一种基于转置静态随机存取存储器阵列的片上训练存内计算架构”，*IEEE*计算机汇刊，第 69 卷，第 7 期，第 944–954 页，2020年。[13] A. K. Rajput 和 M. Pattanaik，“使用 8T 静态随机存取存储器存储单元实现用于存内计算的布尔和算术函数”，载于*2020*年新兴技术国际会议 *(INCET)*，第 1–5 页，2020年。[14] S. Yin、Z. Jiang、J. Seo 和 M. Seok，“XNOR-SRAM：用于二值/三值深度神经网络的存内计算静态随机存取存储器宏单元”，*IEEE*固态电路学报，第 55 卷，第 6 期，第 1733–1743 页，2020年。[15] A. Jaiswal、A. Agrawal、M. F. Ali、S. Sharmin 和 K. Roy，“i-SRAM：利用静态随机存取存储器实现向量布尔操作的交错字线”，*IEEE*电路与系统汇刊 *I*：常规论文，第 67 卷，第 12 期，第 4651–4659 页，2020年。[16] A. Agrawal、A. Kosta、S. Kodge、D. E. Kim 和 K. Roy，“CASH-RAM：利用标准 8T 静态随机存取存储器阵列中的电荷累积与共享实现边缘推理存内计算”，*IEEE*新兴与精选电路和系统主题期刊，第 10 卷，第 3 期，第 295–305 页，2020年。[17] G. Santoro、G. Turvani 和 M. Graziano，“新型存内逻辑范式：架构与技术视角”，*Micromachines*，第 10 卷，第 6 期，第 368 页，2019年。[18] G. Saha、Z. Jiang、S. Parihar、C. Xi、J. Higman 和 M. Ahosan Ul Karim，“一种具备卓越工艺变化鲁棒性的高能效、高吞吐量二值神经网络存内计算位单元”，*IEEE Access*，第 8 卷，第 91405–91414 页，2020年。[19] H. Kim、Q. Chen、T. Yoo、T. T. H. Kim 和 B. Kim，“一种用于人工神经网络能效处理的位精度可重配置数字存内计算宏单元”，载于*2019*年国际片上系统设计会议 *(ISOCC)*，第 166–167 页，2019年。[20] H. Shin、J. Sim、D. Lee 和 L. S. Kim，“一种用于加速二值神经网络、具备 PVT 鲁棒性的定制 4T 嵌入式动态随机存取存储器单元阵列”，载于*2019*年 *IEEE/ACM*国际计算机辅助设计会议 *(ICCAD)*，第 1–8 页，2019年。[21] A. Agrawal、A. Jaiswal、D. Roy、B. Han、G. Srinivasan、A. Ankit 和 K. Roy，“Xcel-RAM：在高吞吐量静态随机存取存储器计算阵列中加速二值神经网络”，*IEEE*电路与系统汇刊 *I*：常规论文，第 66 卷，第 8 期，第 3064–3076 页，2019年。[22] J. Saikia、S. Yin、Z. Jiang、M. Seok 和 J. Seo，“基于存内计算静态随机存取存储器的 K 近邻硬件加速器”，载于*2019*年 *IEEE/ACM*低功耗电子与设计国际研讨会 *(ISLPED)*，第 1–6 页，2019年。[23] H. Kim、H. Oh 和 J.-J. Kim，“采用输入分布正则化的能效型无 XNOR 存内二值神经网络加速器”，载于第*39*届计算机辅助设计国际会议论文集，ICCAD ’20，（美国纽约州纽约市），计算机协会，2020年。[24] S. Huang、H. Jiang、X. Peng、W. Li 和 S. Yu，“XOR-CIM：嵌入式 XOR 加密的存内计算静态随机存取存储器架构”，载于第*39*届计算机辅助设计国际会议论文集，ICCAD ’20，（美国纽约州纽约市），计算机协会，2020年。[25] M. Ali、A. Jaiswal、S. Kodge、A. Agrawal、I. Chakraborty 和 K. Roy，“IMAC：在 6T 静态随机存取存储器阵列中实现存内多位乘法与累加”，*IEEE*电路与系统汇刊 *I*：常规论文，第 67 卷，第 8 期，第 2521–2531 页，2020年。[26] H. Jiang、R. Liu 和 S. Yu，“基于 8T XNOR 静态随机存取存储器的深度神经网络加速器并行存内计算”，载于*2020*年 *IEEE*第*63*届国际中西部电路与系统研讨会 *(MWSCAS)*，第 257–260 页，2020年。[27] A. Biswas 和 A. P. Chandrakasan，“CONV-SRAM：一种用于低功耗卷积神经网络、支持存内点积计算的能效型静态随机存取存储器”，*IEEE*固态电路学报，第 54 卷，第 1 期，第 217–230 页，2019年。[28] A. Agrawal、A. Jaiswal、C. Lee 和 K. Roy，“X-SRAM：在 CMOS 静态随机存取存储器中实现存内布尔计算”，*IEEE*电路与系统汇刊 *I*：常规论文，第 65 卷，第 12 期，第 4219–4232 页，2018。[29] Q. Dong、S. Jeloka、M. Saligane、Y. Kim、M. Kawaminami、A. Harada、S. Miyoshi、M. Yasuda、D. Blaauw 和 D. Sylvester，“一种用于搜索和存内计算、工作电压为 0.3 V 的 4 + 2T 静态随机存取存储器 V_{DDmin}”，*IEEE*固态电路学报

，第53卷，第4号，pp. 1006–1015 ，2018年。

[30] M. Vacca、Y. Tavva、A. Chattopadhyay 和 A. Calimera，“用于最小值/最大值搜索的存内逻辑架构”，载于*2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*，第853–856页，2018年。[31] K. Yang、R. Karam 和 S. Bhunia，“用于高能效细粒度数据处理的交错式存内逻辑架构”，载于*2017 IEEE 60th International Midwest Symposium on Circuits and Systems(MWSCAS)*，第409–412页，2017年。[32] K. C. Akyel、H. Charles、J. Mottin、B. Giraud、G. Suraci、S. Thuries 和 J. Noel，“DRC2：基于存储器架构的动态可重构计算电路”，载于*2016 IEEEInternational Conference onRebooting Computing (ICRC)*，第1–8页，2016年。[33] S. Jeloka、N. B. Akesh、D. Sylvester 和 D. Blaauw，“采用推规则6T位单元、支持存内逻辑的28纳米可配置存储器（TCAM/BCAM/SRAM）”，*IEEE*固态电路学报，第51卷，第4期，第1009–1021页，2016年。[34] K. Pagiamtzis 和 A. Sheikholeslami，“内容可寻址存储器（CAM）电路与架构：教程与综述”，*IEEE*固态电路学报，第41卷，第712–727页，2006年3月。[35] I. Arsovski 和 A. Sheikholeslami，“用于内容可寻址存储器匹配线感测的失配相关功率分配技术”，*IEEE*固态电路学报，第38卷，第11期，第1958–1966页，2003年。[36] A. Coluccio、M. Vacca 和 G. Turvani，“存内逻辑计算：值得吗？二值神经网络案例研究”，《低功耗电子与应用期刊》，第10卷，第1期，2020年。[37] T. Kobayashi、K. Nogami、T. Shirotori 和 Y. Fujimoto，“用于低功耗架构的电流控制锁存式感测放大器和静态功耗节省型输入缓冲器”，*IEEE*固态电路学报，第28卷，第4期，第523–527页，1993年。[38] J. M. Rabaey、A. Chandrakasan 和 B. Nikolic，数字集成电路。美国：普伦蒂斯·霍尔出版社，第3版，2008年。