arXiv:2304.03974v1 [cs.AR] 8 Apr 2023

BRAMAC：面向现场可编程门阵列乘加运算的存内计算 BRAM 架构

Yuzong Chen和MohamedS.Abdelfattah电气与计算机工程系，
康奈尔大学{yc2367, mohamed}@cornell.edu

摘要—**与全精度浮点相比，采用低整数精度的深度神经网络(DNN)推理已被证明能够在几乎不损失精度或完全不损失精度的情况下，显著提高内存利用率和计算吞吐量。现代基于现场可编程门阵列的深度神经网络推理严重依赖片上块随机存取存储器（BRAM）来存储模型，并依赖数字信号处理器（DSP）单元来实现乘加运算（MAC）操作，这是深度神经网络的一项基本运算。在本文中，我们通过提出BRAMAC（存内计算 BRAM 架构，用于现场可编程门阵列上的乘加运算），增强现有BRAM，使其也能够计算 MAC。BRAMAC采用位串行与位并行混合数据流，在小型虚拟BRAM 阵列中支持 2位补码的2位至8位MAC。不同于以往的BRAM内计算架构，BRAMAC允许在虚拟 BRAM 阵列中进行计算的同时，对主BRAM阵列执行读写访问，从而同时支持持久化和基于分块的深度神经网络推理。我们探索了两种BRAMAC 变体：BRAMAC-2SA（带有2个同步虚拟阵列）和BRAMAC-1DA（带有 1 个双倍泵频虚拟阵列）。对于 2位、4位和 8位精度，BRAMAC-2SA/BRAMAC-1DA 可分别将大型Arria-10 现场可编程门阵列的峰值 MAC吞吐量提升至 2.6×/2.1×、2.3×/2.0×和1.9×/1.7× ，代价是 FPGA核心面积增加 6.8%/3.4%。将BRAMAC-2SA/BRAMAC-1DA添加到最先进的基于分块的深度神经网络加速器后，在不同模型精度下，AlexNet 和ResNet-34 分别可实现平均2.05×/1.7× 和 1.33×/1.52× 的加速比。我们的代码可在以下地址获取：https://github.com/abdelfattah-lab/BRAMAC。**

I. 引言

深度神经网络（DNN）已广泛应用于计算机视觉、语音识别和自然语言处理等许多重要领域。然而，用于复杂任务且经过良好训练的深度神经网络模型，模型大小十分庞大，从数百兆字节（例如，用于 ImageNet 分类的 AlexNet）到数百吉字节（例如，生成类人文本的 GPT3）不等 [1]–[3]。因此，许多研究人员一直在探索使用较低的数值精度来表示深度神经网络模型权重和激活值，尤其是在推理过程中；与全精度浮点数（FP）相比，低精度算术造成的精度损失很小，甚至没有精度损失 [4], [5]。这种低精度特性有助于更好地利用片上存储器和计算资源，从而提升性能。例如，与 FP32 精度相比，英伟达 GPU 使用 INT8 精度可获得 4-8× 推理加速比 [6], ；与 INT8 [7] 相比，使用 INT4 精度还可额外获得 1.6× 的加速比。

与此同时，得益于硬件可编程性，现场可编程门阵列正日益成为深度神经网络加速的热门平台；这种可编程性支持定制数据通路和适用于低精度推理的数值位宽 [8]–[11]。基于 FPGA 的深度神经网络加速器高度依赖块随机存取存储器（BRAM）来存储模型，并依赖数字信号处理（DSP）单元来实现乘加运算（MAC）——这是深度神经网络中的基本运算单元。然而，大多数 FPGA 厂商的 DSP 模块原生不支持低于 18 位的精度，因此并不适合实现低精度乘加运算 [12]–[14]。为了让深度神经网络更好地利用 FPGA 的片上资源，研究人员提出了用于低精度乘加运算的新型 DSP 架构[15], [16]。最近，一些研究提出在块随机存取存储器中加入计算能力，使其能够执行各种布尔运算和算术运算 [17], [18]。这种存内计算（CIM）方法不会牺牲 FPGA 上现有逻辑资源的性能，反而能与之互补，进一步提升 FPGA 的计算吞吐量。此外，存内计算可以减少存储器与逻辑单元之间数据传输所需的布线，从而节省能耗和面积。对于深度神经网络加速器而言，这一点尤为重要，因为模型参数和激活值经常需要在块随机存取存储器与数字信号处理器之间传输，以执行大量计算。

本文提出 BRAMAC，这是一种高效的 BRAM 内乘加架构，借此进一步提升现场可编程门阵列对低精度深度神经网络的兼容性。与直接在主 BRAM 阵列 [17], [18], 上进行计算的既有存内计算架构不同，BRAMAC 会先将数据从主 BRAM 阵列复制到一个额外的独立存储器阵列，再在这个“虚拟”阵列上进行计算。该阵列是真双端口 BRAM，列数与主 BRAM 阵列相同，但只有 7 行。与主 BRAM 阵列通常具有的 >100 个物理行相比，这个 7 行虚拟阵列的位线寄生负载要小得多，因此能够以低功耗快速访问。此外，即使在存内计算期间，虚拟阵列也能让 BRAMAC 像普通 BRAM 一样运行——主 BRAM 阵列的读写端口仍可供应用逻辑使用。最后，BRAMAC 针对深度神经网络乘加运算进行了优化，采用共享输入乘法和原位累加。我们的贡献如下：

1)我们提出了新的外围电路，使BRAMAC 能够利用位串行与位并行混合数据流，同时计算两个乘累加器（或一个 MAC2），$P = (W_1 I_1 + W_2 I_2)$。2) 我们提出了两种具有不同面积-吞吐量权衡的 BRAMAC 变体：采用2个同步虚设阵列（2SA）的 BRAMAC，以及采用一个双倍泵送虚拟阵列（1DA）的 BRAMAC。

3) 我们设计了一种嵌入式有限状态机（eFSM），以在MAC2计-算期间释放主BRAM端口，并允许同时访问主BRAM，从而实现高效的基于分块的深度神经网络加速。

4) 我们量化了在分块式 FPGA 深度神经网络加速器中采用 BRAMAC 的优势，与不采用 BRAMAC 的基准加速器相比，该加速器在 AlexNet 和 ResNet-34 上的性能分别提升了最高 2.04× 倍和 1.52× 倍。

II. 相关工作

在本节中，我们讨论了以在现场可编程门阵列上实现高效乘加运算实现为目标的既有工作，包括逻辑块、数字信号处理器和块随机存取存储器增强方案。

A. 具有快速算术功能的逻辑块

为了在软逻辑中高效实现算术运算，现代现场可编程门阵列在其逻辑块（LB）中包含硬化加法器电路 [19]。这些加法器涵盖了从简单的脉动进位加法器到跳跃进位加法器和超前进位加法器等更复杂的变体。为了降低进位传播延迟，会使用专用布线在不同逻辑块之间传递进位信号。受深度神经网络中采用低精度所展现出的卓越效率启发，近期研究开始探索在逻辑块中增加更多硬化算术电路。例如，Boutros 等人 [20] 提出了三种逻辑块架构增强方案，以提升软逻辑中乘加运算的性能。其中最有前景的方案将乘加运算密度提高了 1.7× ，同时提升了乘加运算速度。

B. 低精度*DSP*架构

现代商用现场可编程门阵列包含 DSP 模块，可高效实现乘法，并提供预加法和累加等附加功能，这些功能常用于信号处理应用 [19]。然而，大多数现场可编程门阵列厂商的 DSP 乘法器最低精度为 18 位，因此在加速低精度深度神经网络方面竞争力较弱。为解决这一限制，研究人员提出了新的 DSP 架构，以支持低精度乘加运算。Boutros等人 [15] 推出了一种增强型英特尔 DSP（eDSP），无需使用额外的布线端口即可支持四次 9 位乘法或八次 4 位乘法。Rasoulinezhad等人 [16] 提出了一种改进的赛灵思 DSP，称为 PIR-DSP，可执行六次 9 位、十二次 4 位或二十四次 2 位乘法。关于行业 DSP 的发展趋势，最新的赛灵思 Versal 和英特尔 Agilex 器件在其 DSP 模块中增加了对 8 位乘法的支持 [21], [22]。此外，英特尔最新的 Stratix-10 NX器件新增了一个数字信号处理器（称为AI张量）模块，其中包含30个 INT8 乘法器，并且还可配置为 60 个 INT4 乘法器 [23]。

C. 在块随机存取存储器中计算

随着存内计算的出现，以克服冯·诺依曼瓶颈 [24]，一些 FPGA 研究人员建议为现有 BRAM 架构增添计算能力。Wang 等 [17] 提出了一种可计算 BRAM（CCB），使用位串行运算来实现高程度的计算

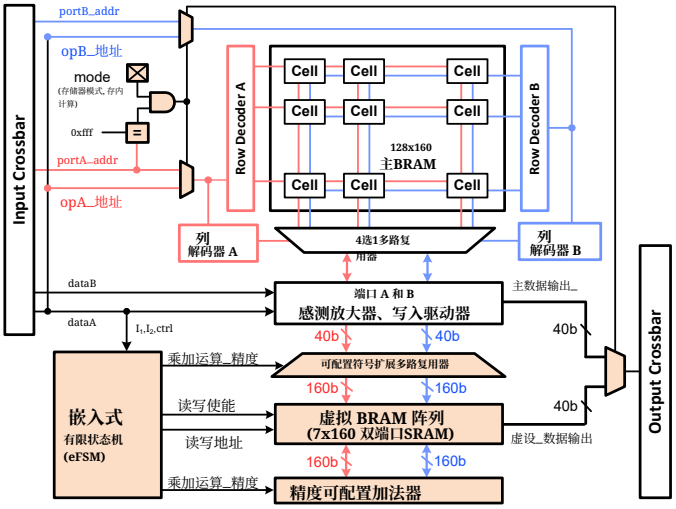


图 1：改自英特尔 M20K BRAM 的 BRAMAC 顶层模块图。新增电路模块以橙色阴影标示。

并行度。然而，CCB 的电路实现需要额外的电压电源，以缓解从一个 BRAM 端口激活两条字线所导致的读扰动问题，而这在实际应用中很难实现。Arora 等 [18]随后设计了一种名为 CoMeFa 的新型 BRAM 内计算架构，以克服 CCB 的一些局限性。CoMeFa 同样采用位串行运算，但利用 BRAM 的双端口特性，分别从两个端口读出两个操作数，而不是从一个端口激活两条字线，从而消除了读扰动问题。

CCB 和 CoMeFa 都要求位串行计算采用转置数据布局，即每个字占据一列和多行，而传统数据布局中每个字占据一行和多列。然而，在线执行时，转置数据会带来较高的延迟和额外的硬件成本（例如 CoMeFa 中的交织模块）。此外，这两种 BRAM 架构都直接在主 BRAM 阵列上计算，并通过 BRAM 写端口接收存内计算指令，这使得分块无法实现。因此，这两项工作仅限于加速持久型深度神经网络推理，其中模型权重会在离线时转置，并持久保存在片上存储器中。与 CCB 和 CoMeFa 不同，BRAMAC 采用位串行与位并行混合乘加数据流，无需转置数据布局。此外，BRAMAC 不在通常规模较大、因而速度较慢且功耗较高的主 BRAM 阵列上计算，而是将主 BRAM 中的数据复制到一个独立的专用虚拟 BRAM 阵列中进行计算。该虚拟阵列仅有 7 行，因此与主 BRAM 阵列相比，访问速度更快。在存内计算期间，它还可以释放主 BRAM 的读写端口，从而支持基于分块的深度神经网络加速。

III. BRAMAC 架构和数据流

A. 总体架构

图 1 展示了由英特尔的 M20K BRAM修改而来的BRAMAC 顶层模块图，其中添加的电路模块以橙色阴影标示。布线接口（即，输入和 [25] 并添加了橙色阴影的电路模块。布线接口（即，输入和

BRAMAC 的输出交叉开关）与 M20K 的相同。主 BRAM 阵列的维度为128行 × 160列，即20 kb的存储容量。M20K 的4:1列多路复用功能得以保留。额外添加了一个 SRAM单元，用于选择 BRAMAC 的两种操作模式之一：

1) 存储器模式： 在此存储器模式下，BRAMAC的行为与传统 M20K 完全相同。输入交叉开关将地址和数据发送至portA和portB。对于存储器读取操作，两个地址分别由行译码器和列译码器进行译码。来自感应放大器的40 位块随机存取存储器输出数据被发送至输出交叉开关。对于存储器写入操作，数据被发送至写入驱动器，以更新主BRAM。

*2)*存内计算：这是 BRAMAC 可以使用 2位、4位或 8位操作数精度计算 MAC2, $P = (W_1 I_1 + W_2 I_2)$ 的计算模式。在本文余下部分中，两组操作数（W_1, W_2）和（I_1, I_2）分别可以视为深度神经网络的权重和输入。从较高层次来看，BRAMAC 将权重保留在 BRAMAC 内部，同时从外部以数据流方式输入输入。

主BRAM 自动配置为简单双端口存储器，最大数据宽度为40 位，深度为512，以最大化读写吞吐量。保留一个特殊地址（0xfff），并将其与 portA地址进行比较；如果相等，则将 40 位 portA 数据视为存内计算指令。该存内计算指令包含两个地址，分别用于从主BRAM 读取两个40位数据。每个 40位数据都是一个向量，其中包含多个低精度 W_1/W_2 元素。可配置符号扩展多路复用器将 40 位向量进行符号扩展至 160 位，然后将其复制到虚拟 BRAM阵列中；该阵列是一个7 行 × 160 列的真双端口 BRAM，且不具备列复用功能。存内计算指令还包含两个输入，即 I_1 和 I_2，以及若干发送至 eFSM 的控制信号，用于触发和控制 MAC2 操作。精度可配置加法器可以从虚拟阵列读取两个160 位向量，执行单指令多数据（SIMD）加法，并将和写回虚拟阵列。由于虚拟阵列的列数与主 BRAM 阵列相同，因此它可以像主 BRAM一样读出 40位数据。添加了一个 2:1多路复用器，用于选择主 BRAM 与虚拟阵列之间的数据。

B. 位串行与位并行混合乘加数据流

BRAMAC 通过采用位串行与位并行混合数据流来计算二进制补码 MAC2 [26] ，如前所述。
在算法1中，第2至11行的for循环逐位遍历两个输入。每次迭代分别将完整的W_1 和 W_2 与 I_1 和 I_2中的一个比特相乘，然后进行位并行加法，以获得第3行所示的部分和（psum）。如果当前输入位是最高有效位（MSB），则由于在二进制补码表示中最高有效位为负，需从P中减去部分和（第5行）。如果当前输入位不是最低有效位（LSB），则在加上部分和后，还需要将P左移1位（第6、9行）。

混合位串行与位并行 MAC2 算法适用于高效计算矩阵-向量乘法（MVM）：其中， n^{th} 向量元素与所有元素相乘

算法1： 混合逐位串行与位并行MAC2

要求： 所有数字均采用二进制补码表示的整数
输入 ： $W \in \mathbb{Z}^2, I \in \mathbb{Z}^2$，精度 $n \geq 2$
输出： $P \in \mathbb{Z}$
初始化 $P = 0$
对于 $i = (n-1)$ *downto* 0 **执行**
  $psum = W_1 * I_1[i] + W_2 * I_2[i]$
  如果 $i == (n-1)$ **则**
    $P = P + inv(psum) + 1$
    $P = P << 1$
  否则如果 $i \neq 0$ **则**
    $P = P + psum$
    $P = P << 1$
  else
    $P = P + psum$
返回 P

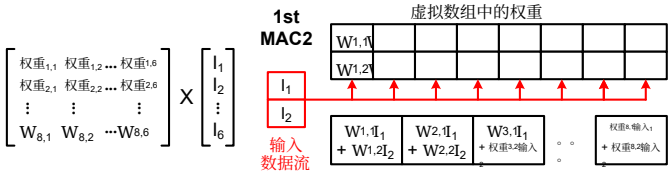


图 2：用于计算矩阵-向量乘法的 MAC2 示例。

这些元素属于 n^{th} 矩阵列。为了在 BRAMAC 中利用这种输入共享，两个输入被打包到发送给 BRAMAC 的存内计算指令中，然后分别与复制到虚拟阵列中的对应两列矩阵的所有元素相乘。复制矩阵列需要对权重矩阵进行转置，使矩阵列对应于块随机存取存储器的一行。对于深度神经网络，这可以轻松地离线完成。图 2 展示了一个使用 MAC2 计算矩阵-向量乘法的示例，其中矩阵维度为 8×6。对于第一次 MAC2 运算，第一列和第二列矩阵从主BRAM复制到虚拟阵列。两个向量元素 I_1 和 I_2 通过存内计算指令以数据流方式输入 BRAMAC，并分别与第一列和第二列矩阵的全部 8 个元素相乘，从而得到 8 个部分和。对于大型矩阵，可加载到虚拟阵列中的矩阵元素数量取决于 MAC 精度。由于主BRAM的两个读端口总数据宽度为 80 位，因此在一次 MAC2 运算中，它们可以向虚拟阵列复制十个 8位、二十个4位或四十个 2位权重，分别提供 10、20 或 40 个乘累加器的并行度。

C. 支持 *MAC2* 的电路设计

下面介绍 BRAMAC 中用于支持 MAC2 的新电路模块。这些电路模块如图 3 所示，包括一个双端口“虚设”BRAM阵列（图 3(a)）、一个可配置符号扩展多路复用器（图 3(b)）、一个由 1 位全加器实现的 160 位 SIMD加法器，以及读写电路（图 3(c)）。

1) 双端口虚拟块*RAM*阵列： 双端口虚拟块RAM 阵列为7行 × 160列，不采用列多路复用，如图3(a)所示。其SRAM单元与主BRAM中使用的相同。每列包含两个感应放大器和两个写入驱动器，以实现真正的双端口访问。其 1st 行被硬编码为始终存储 **0**。 2nd和 3rd 行分别存储**W₁** 和**W₂** 向量，

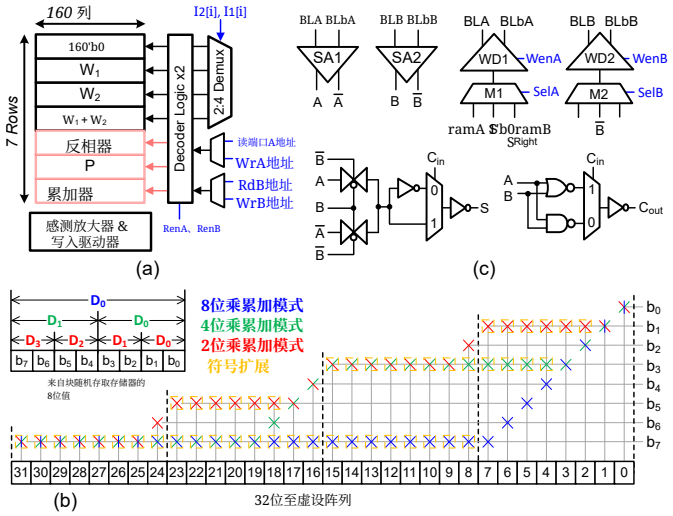


图 3：用于计算 MAC2 的 BRAMAC 电路模块：(a) 双端口虚拟块 RAM 阵列，(b) 可配置符号扩展多路复用器（此处展示五个相同模块中的一个），(c) 带读写电路的 1 位全加器。

复制自主 BRAM 阵列。 4th 行存储一个**($W_1 + W_2$)** 向量。 5th **反相器**行用于存储二进制减法所需的临时求逆部分和（算法1的第5行）。 6th 行存储MAC2结果**P**。 7th 行是一个宽型**累加器**，用于累加构成大点积的多个 MAC2 结果。

虚拟阵列的读写操作由 eFSM 发送的地址和使能信号控制（图 3(a) 中的蓝色信号），如第 III-A2 节所述。MAC2 期间对 1st – 4th 行的访问由译码逻辑和一个 2 选 4 分路器共同管理。分路器的 2 位选择信号分别来自两个输入 I_1 和 I_2, 当前的两个处理位。这使得可以使用查找表 [27]计算部分和（算法1 第 3 行）。如果 **{I_2[i], I_1[i]}** 为 2'b00，则将读出 1st 零行并与 6th 行 P 相加。如果 **{I_2[i], I_1[i]}** 为 2'b11，则将读出4th 行 **($W_1 + W_2$)** 并与 P 相加。如果 **{I_2[i],I_1[i]}** 为 2'b01 或 2'b10，则将读出 2nd 行 W_1 或 3rd行 W_2 并与 P 相加。

由于虚拟阵列会从主 BRAM 阵列复制数据以进行计算，因此可能出现一致性问题：主 BRAM 正在更新，而虚拟阵列仍在使用过时数据进行计算。我们将此问题留给程序员/编译器显式确保存储器一致性，这类似于显式处理英特尔 BRAM [28] 的读期间写行为。

2) 可配置符号扩展多路复用器：虽然算法1中未体现，但主BRAM中的W_1 和 W_2 向量在复制到虚拟阵列之前需要进行符号扩展，以防止乘加运算MAC2期间发生溢出。为此，在主BRAM与虚拟阵列之间添加了两个可配置符号扩展多路复用器。每个多路复用器包含五个相同的模块，其中一个如图3(b)所示。由于主BRAM的数据宽度为40位，因此可同时向虚拟阵列复制五个8位、十个4位或二十个2位元素。五个相同的多路复用器模块中的每一个都可以将一个8位元素符号扩展为一个32位元素（图3(b)中的蓝色叉号），或将两个4位元素符号扩展为两个16位元素（图3(b)中的绿色叉号），或将四个2位元素符号扩展为四个8位元素（图3(b)中的红色叉号）。此外，由于2/4/8位MAC2存储结果最多只需要5/9/17位，所提出的符号扩展多路复用器可以提供MAC2所需的更高位宽。这样一来，通过将虚拟阵列的行（存储MAC2结果P）与行（存储累加器）相加，即可累加多个连续的MAC2结果。

3) 带读写电路的位并行*SIMD*加法器：BRAMAC 中的 160 位 SIMD 加法器采用传统的 1位全加器设计，如图 3(c) 所示。它支持位并行SIMD加法，可配置为二十个 8位加法器、十个 16 位加法器和五个 32 位加法器，分别用于 2位、4位和 8位 MAC2，其最坏情况下的延迟等于 32 位加法的延迟。SIMD 加法器的两个操作数 **A** 和**B**来自两个灵敏放大器 **SA1** 和**SA2**，它们用于比较两对位线对的电压差分，分别为（**BLA**，**BLbA**）和（**BLB**，**BLbB**）。为支持加法后执行1位左移操作（算法1第 6 行和第 9 行所需），在写驱动器 **WD1** 之前使用写回多路复用器**M1**，以选择当前全加器的和 **S** 或右侧全加器的和**S_{Right}**。**M1** 还可选择**ramA**，以从主 BRAM 复制第一份数据 W_1。类似地，在写驱动器 **WD2** 之前使用写回多路复用器**M2**，以在三个信号之间进行选择：**B-bar**（用于求反）、**ramB**（用于从主 BRAM 复制第二份数据 W_2），以及 1'b0（用于初始化**P**（第 1 行）或**累加器**）。**M1**和**M2**均由 eFSM 控制。

IV. BRAMAC 变体

A. 采用两个同步虚设阵列（*2SA*）的 *BRAMAC*

此变体称为 BRAMAC-2SA，包含两个与主BRAM处于同一时钟域的同步虚拟阵列。在此架构中，权重复制期间，主BRAM的两个端口分别为一个虚拟阵列提供数据。由于如第 III-B 节所述，BRAMAC 本身支持将同一输入与多个权重相乘，因此此变体采用输入共享方法来平衡数据复用。具体而言，在每次 MAC2 迭代中，两个虚拟阵列复制相同的权重，但处理不同的输入。第一个虚拟阵列接收两个输入 I_1，I_2并计算 $W_1I_1 + W_2I_2$，而第二个虚拟阵列接收另外两个输入 I_3，I_4并计算 $W_1I_3 + W_2I_4$。

BRAMAC-2SA 的一个虚拟阵列的 4 位 MAC2 运算示例如图 4所示。请注意，由于篇幅限制，我们展示了10 个数据通道中的2个，并采用10 位符号扩展（而不是第 III-C2 节所述的 16 位符号扩展）。在 **周期1** 和 **周期 2** 中，W_1 和W_2 进行符号扩展并复制到虚拟阵列中。在这两个周期内，每个虚拟阵列的两个输入也通过存内计算指令发送到 BRAMAC-2SA，并锁存以供进一步处理。在 **周期3** 中，W_1 和W_2 被读出并相加。其和被写回 4th 行，以存储 **(W_1 +W_2)**。同时， 6th 行 **P** 也可以初始化为零。在 **周期4** 中，两个输入的最高有效位被流入虚拟阵列。选定的行 **W1** 被求逆以准备进行

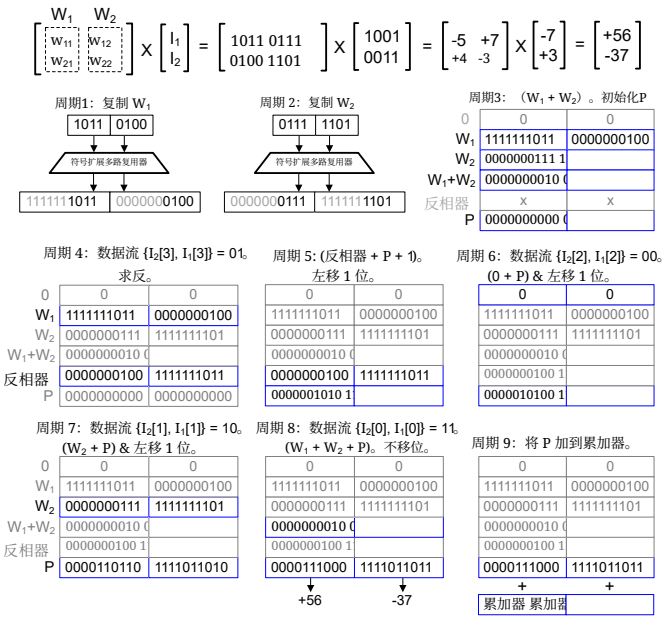


图 4：BRAMAC-2SA 中一个虚拟阵列执行 4位 MAC2 的示例。由于空间限制，我们展示了 10 个数据通道中的 2 个，并采用 10 位符号扩展，而非 16 位。

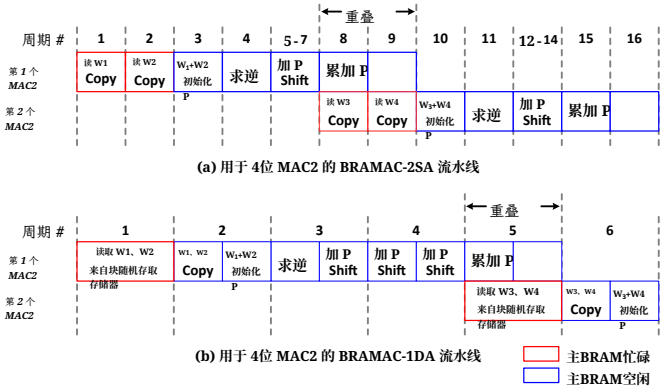


图 5：4位 MAC2 在 (a) BRAMAC-2SA 和 (b) BRAMAC-1DA 中的流水线图。

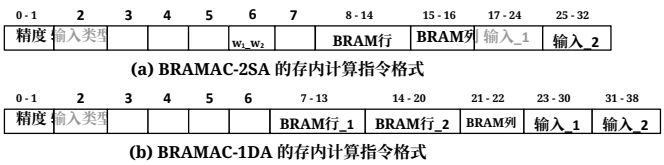


图 6：用于 (a) BRAMAC-2SA 和 (b) BRAMAC-1DA 的存内计算指令格式。

二进制减法。在 **周期 5**，将**反相器** 加到**P**。总和左移 1 位后写回**P**。输入流传输持续到**周期 8**，此时处理两个输入的最低有效位，并得到正确的 MAC2 结果**P** 。在**周期 9**，将**P** 加到 7th **累加器** 行。然后可将其初始化，以用于后续的 MAC2。

上述示例表明，BRAMAC-2SA可以使用 9 个周期完成 4位 MAC2。然而，在最后两个周期的写回阶段，即 **周期 8**和**周期9**，当前的两个权重$\mathbf{W_1}$ 和$\mathbf{W_2}$ 在虚拟阵列中已不再需要，因为当前的MAC2结果**P** 已经在位并行加法器的输出端获得。因此，如图 5(a) 所示，这两个周期也可以分别用于复制接下来的两个权重$\mathbf{W_3}$ 和$\mathbf{W_4}$, 。因此，BRAMAC-2SA中的 4位MAC2只需 7 个周期即可完成。这种流水线处理也可以应用于 2位和8位 MAC2。2位、4位和 8位 MAC2之间的唯一差异，便是处理每个输入位所需的周期数，如算法1第 2–11 行所述。因此，2位和 8位 MAC2 分别需要 5 个周期和 11 个周期才能完成。

B. 采用一个双倍泵时虚设阵列（*1DA*）的 *BRAMAC*

该变体称为 BRAMAC-1DA，仅使用一个虚拟阵列，以减少面积开销。与 BRAMAC- 2SA相比，使用一个虚拟阵列会使乘加吞吐量降低 2×，但我们提出以 2× 主BRAM时钟频率对虚拟阵列进行双倍泵频。存储器多倍泵频是现场可编程门阵列设计中常用的提升系统吞吐量的技术 [29], [30]。与同步虚拟阵列相比，双倍泵频虚拟阵列不会增加任何额外的面积开销。相反，它只需在编译期间进行独立的时钟布线。

由于主BRAM和虚拟阵列仅在权重复制期间进行交互，因此它们之间的同步可以很容易处理。图 5(b)显示了 BRAMAC-1DA 中4位 MAC2的流水线图。在**周期1**中，主BRAM读出两个权重 $\mathbf{W_1}$ 和$\mathbf{W_2}$。在**周期 2**的前半个周期中，虚拟阵列使用其两个写端口复制 $\mathbf{W_1}$ 和$\mathbf{W_2}$ 。随后，虚拟阵列可以使用与BRAMAC-2SA 相同的操作流程来计算 MAC2，不同之处在于，BRAMAC-2SA 中的每个周期现在对应 BRAMAC-1DA 中的半个周期。与 BRAMAC-2SA 的流水线优化类似，主BRAM可以在**周期 5** 中启动读取接下来的两个权重$\mathbf{W_3}$ 和$\mathbf{W_4}$ ，同时虚拟阵列进行计算。因此，4位MAC2 可以用4个周期完成。这种流水线处理也可以应用于2位和8位 MAC2。因此，2位和 8位 MAC2 分别需要 3个和 6个周期才能完成。

C. 嵌入式有限状态机以释放 *BRAM*端口

由于虚拟阵列在执行 MAC2计算时的行为是确定的，我们提出使用 eFSM 对其进行控制。该eFSM接收存内计算指令，以触发MAC2计算并控制虚拟阵列的读写访问。仅当主BRAM需要向虚拟阵列发送数据时，才需要存内计算指令（如图 5 中的红色框所示）。因此，在 BRAMAC-2SA 中，主BRAM会忙于 2个周期；在BRAMAC-1DA 中，则会忙于1个周期。当主BRAM空闲时，它可以执行正常的读取操作，为查找表/数字信号处理器提供数据，或执行写入操作，从片外动态随机存取存储器加载下一个权重数据块，从而实现基于分块的深度神经网络加速。这与 CCB和CoMeFa 不同，后两者的 BRAM端口在存内计算期间始终处于忙碌状态。

图 6(a) 和 (b)分别展示了所提出的用于 BRAMAC-2SA 和 BRAMAC-1DA 的存内计算指令格式。对于 BRAMAC-2SA，在每次复制操作期间，**BRAM行** 和 **BRAM列** 组合形成一个块随机存取存储器地址。另一方面，BRAMAC-1DA 需要接收两个块随机存取存储器

表 I：基准 Arria10 GX900 FPGA 的资源数量和面积占比。

资源	计数器	面积占比
逻辑块（LB）	33920	70.4%
DSP 单元	1518	9.5%
块 RAM（M20K）	33920	20.1%

同时访问地址。这是通过使用两个 BRAM 行地址 **BRAM行1** 和 **BRAM行2** 以及一个共享列地址**BRAM列**实现的。

这两种 BRAMAC 变体共享一些通用控制信号。2 位 **精度** 指定三种受支持的 MAC2 精度之一。**输入类型** 用于指示两个输入是有符号还是无符号。如果输入为无符号，则可以跳过反相周期以提升性能。**复位** 会将虚拟阵列复位到初始状态，并将零写入其累加器。启用 **启动**后，BRAMAC 将被触发以执行 MAC2。**复制** 指示 BRAMAC 将从主BRAM读取的数据复制到虚拟阵列中；BRAMAC-2SA 还需要额外的 $\mathbf{w_1}$ $\mathbf{w_2}$ 信号，以指示当前复制的数据是 W_1 还是 W_2。这两个信号还支持图 5 中高效的流水线优化，使下一次 MAC2 的权重复制能够与当前 MAC2 的计算重叠。**完成** 指示是否读出虚拟阵列的累加器。启用后，每个周期都会使用 **BRAM列** 从虚拟阵列的累加器行中选择 40 位数据。因此，在每两个点积之间，主BRAM需要分别占用 8 个和 4 个周期，以读出 BRAMAC-2SA 和 BRAMAC-1DA 中的累加器。然而，由于虚拟阵列的累加器在 2/4/8 位 MAC 精度下的大小分别为 8/16/32 位，因此在读出累加器以摊销这项开销之前，它最多可以处理大小为 16/256/2048 的点积。

V. 电路级评估

A. 工具与基准现场可编程门阵列

我们使用 COFFE [31], 这一自动晶体管尺寸调整工具，对除 eFSM 之外的所有 BRAMAC组件进行建模并优化其面积和延迟；eFSM 采用 SystemVerilog 实现，以验证其功能。我们使用 Synopsys Design Compiler 和台积电 28纳米工艺对 eFSM 进行综合，并获得其面积；在缩放至 22 纳米后，BRAMAC-2SA 和 BRAMAC-1DA 的面积分别为 137 μm^2 和 81 μm^2。我们通过在 16 kb 和 32 kb 块随机存取存储器之间进行插值，利用 COFFE 获得一个 M20K模块的面积。对于延迟估算，COFFE 使用 22 纳米预测技术模型 [32]运行 Hspice 仿真。

在本文余下部分中，对于基准现场可编程门阵列，我们采用 Arria-10 GX900 器件 [33]，其速度等级为最高（10AX090H1F34E1SG），资源信息如表 I 所示。Arria-10 器件系列采用与 COFFE 仿真设置相似的 20 纳米工艺制造。根据 [34] 中的面积模型，估算每种资源类型的面积占比。所提出的 BRAMAC 架构通过将所有 M20K模块替换为 BRAMAC-2SA 或 BRAMAC-1DA 来增强基准现场可编程门阵列。

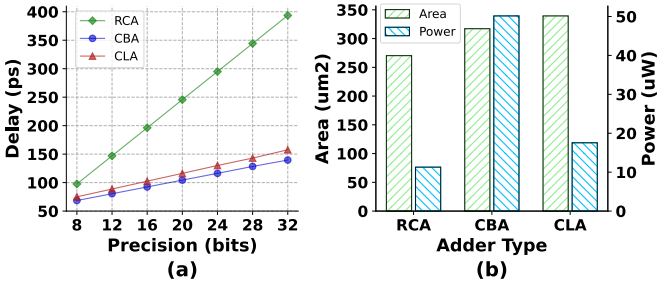


图 7：RCA、CBA 和 CLA 的比较：(a) 延迟与精度的关系。(b) 32位精度下的面积和功耗。

B. 加法器的设计选择

由于 BRAMAC 中的 SIMD加法器在 8位 MAC2 期间进行 32位加法时具有最坏情况延迟，脉动进位加法器（RCA）可能会显著增加虚拟阵列的关键路径延迟，并成为 BRAMAC 的频率瓶颈。因此，我们还研究了两种快速加法器变体 [35]：采用镜像实现、带有 4位超前进位生成器的超前进位加法器（CLA），以及采用动态逻辑、带有 4位曼彻斯特进位链的旁路进位加法器（CBA）。我们使用 COFFE 自动确定进位输出生成器、超前进位生成器和曼彻斯特进位链的尺寸，从而分别为 RCA、CLA 和 CBA 获得最佳的面积-延迟权衡。

图 7 展示了基于 COFFE 仿真得到的三种不同加法器——RCA、CBA 和 CLA——在性能、面积和功耗方面的表现。如图 7(a) 所示，随着加法器精度的提高，RCA 与另外两种快速加法器 CBA/CLA 之间的性能差距逐渐扩大。在最高加法器精度下，即 8位乘加运算期间进行 32 位累加时，RCA 的延迟为 393.6 ps，比 CBA（139.6 ps）慢 2.8× 倍，比 CLA（157.6 ps）慢 2.5× 倍。如图 7(b) 所示，三种加法器的面积相近，但 CBA 的功耗最高，为 50.2 μW，分别比 RCA（11.3 μW）和 CLA（17.6 μW）高 4.44× 倍和 2.86× 倍。这是因为 CBA 采用了动态曼彻斯特进位链；与静态 CMOS逻辑相比，这种结构速度更快，但功耗也更高。总的来说，CLA 在延迟、面积和功耗之间实现了最佳权衡。因此，在后续评估中，我们在 BRAMAC 中采用 CLA。

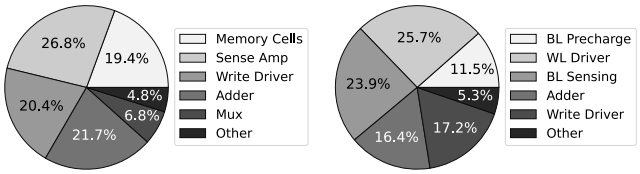
C. BRAMAC 面积与频率

图8(a)展示了BRAMAC虚拟阵列的面积分解。虚拟阵列的总面积为975.6 μm^2，，与基准M20K相比增加了16.9%。由于M20K占基准现场可编程门阵列面积的20.1%，因此这一面积开销仅相当于现场可编程门阵列核心面积增加3.4%。请注意，在后续评估中，我们忽略eFSM的面积开销，因为COFFE的面积模型不包含任何BRAM控制逻辑以及M20K的一些组件，例如纠错电路 [25]。鉴于BRAMAC-2SA/BRAMAC-1DA的eFSM面积分别仅相当于基准M20K面积的1.4%/2.4%，采用更准确的面积模型后，预计BRAMAC相对于基准M20K的面积开销不会改变。

TA表 II：BRAMAC 与先前最先进的现场可编程门阵列乘加运算架构的关键特性

架构		eDSP [15]	PIR-DSP [16]	CCB [17]	CoMeFa-D [18]	CoMeFa-A [18]	BRAMAC-2SA	BRAMAC-1DA
改进型 FPGA 模块		DSP	DSP	BRAM	BRAM	BRAM	BRAM	BRAM
支持的 MAC 精度（-位）		4, 8	2, 4, 8	任意的	任意	任意的	2、4、8	2、4、8
面积开销 (模块)		12%	28%	16.8%	25.4%	8.1%	33.8%	16.9%
面积开销（核心）		1.1%	2.7%	3.4%	5.1%	1.6%	6.8%	3.4%
时钟周期开销 相较于基准 FPGA 模块		0%	30%	60%	25%	150%	10%	46%
并行乘累加器数量 / 乘累加运算延迟（周期） [1]	2位	8 / 1	24 / 1	160 / 16	160 / 16	160 / 16	80 / 5	40 / 3
	4位	8 / 1	12 / 1	160 / 42	160 / 42	160 / 42	40 / 7	20 / 4
	8位	4 / 1	6 / 1	160 / 113	160 / 113	160 / 113	20 / 11	10 / 6
设计复杂度		极低	非常低	High	Low	中等	Low	中等

[1] 对于 DSP 架构，每种 MAC 精度对应的累加器位宽均与基准 DSP 相同。对于块随机存取存储器架构，2位、4位和8位 MAC 的累加器位宽分别为8位、16位和27位。CCB 和 CoMeFa 的乘累加运算延迟基于无符号乘法报告，而 BRAMAC 的乘累加运算延迟则基于二进制补码乘法报告。



(a) Dummy Array Area Breakdown (Total Area = 975.6um^2) **(b) Dummy Array Delay Breakdown (Total Delay = 958.5ps)**

图 8：（a）虚拟阵列的面积分解；（b）虚拟阵列的延迟分解。

图8(b)展示了BRAMAC虚拟阵列的关键路径延迟分解。虚拟阵列仅有7行，其位线寄生负载相比主BRAM显著降低。因此，它能够以快得多的速度完成预充电和放电，使关键路径延迟低于1 ns。这表明，虚拟阵列本身能够以1 GHz的最高频率（F_{max}）运行，不受M20K影响；在Arria-10 [28]中，M20K的F_{max} 为730 MHz。对于BRAMAC-1DA，这使M20K在存内计算模式下的F_{max} 被限制为500 MHz。虽然这低于典型BRAM的F_{max}，但实际的现场可编程门阵列延迟通常受软逻辑和布线限制，因此Arria-10上的设计不太可能达到高于500 MHz的频率。对于BRAMAC-2SA，关键路径出现在权重复制期间，因为只有在从主BRAM读出数据后，回写阶段才能开始。因此，BRAMAC-2SA的F_{max} 取决于M20K。具体而言，虚拟阵列的写驱动器延迟为165 ps，因此其F_{max} 比基准M20K低1.1× 。

D. FPGA 上与其他乘累加架构的比较

我们将 BRAMAC 与 FPGA 上其他最先进的乘累加架构进行了比较，包括 eDSP [15], PIR-DSP [16],CCB [17], 和 CoMeFa [18]。所有架构均采用第 V-A 节所述的相同基准 Arria-10 FPGA。每种架构都用其提出的新模块替换基准设计中对应的 FPGA 模块。表 II 总结了所研究架构的关键特性。

由于采用位串行运算，CCB 和 CoMeFa 在支持的精度方面具有最高的灵活性。然而，它们提出的用于定点乘法的位串行算法仅适用于无符号数，而 eDSP、PIR-DSP 和 BRAMAC 则支持二进制补码乘加。尽管如此。BRAMAC-2SA 的面积开销最高，但与其他 BRAM 架构相比，其频率也最高。两种 DSP 架构的设计复杂度最低，因为它们可以通过数字CAD流程实现，而 BRAM 设计通常涉及模拟元件和手动布局工作量 [31]。在所有 BRAM 架构中，CCB 的设计复杂度最高，因为它需要额外的电压电源。CoMeFa-A 和 BRAMAC-1DA 的设计复杂度中等，因为它们分别需要采用新颖的时序设计技术——感应放大器循环和双倍泵时钟。

VI. 应用级评估

A. 峰值 *MAC* 吞吐量对比

我们将基准现场可编程门阵列的峰值 MAC 吞吐量与采用 BRAMAC 及第 V-D 节所研究的其他乘累加架构的增强型现场可编程门阵列的峰值 MAC 吞吐量进行比较。我们考虑三种 MAC 精度：2位乘法（配备 8 位累加器）、4位乘法（配备 16 位累加器）和 8位乘法（配备 27 位累加器）。各类资源的峰值 MAC 吞吐量按如下方式确定：

*(1)*查找表：我们在Quartus中仅使用查找表对一个乘加单元进行综合、布局和布线，以获得其F_{max} 和资源利用率。随后，我们采用与 [17],[18]相同的方法，通过乐观地假设所有查找表都能以相同的 F_{max} 使用，来计算总乘加吞吐量。

(2) 数字信号处理器：Arria-10 数字信号处理器具有两个 18×19 乘法器，每个乘法器都可以使用 [36]中所述的DSP打包实现一个 8位乘加运算、两个 4位乘加运算或四个 2位乘加运算。我们运行 Quartus，在m18x18sumof2模式下生成一个数字信号处理器，并发现其F_{max}为 549兆赫。对于eDSP，我们使用相同的 F_{max} ，但根据其报告的F_{max}，对于 PIR-DSP 使用低 1.3×的 F_{max} 。

(3) 块随机存取存储器：我们使用 Quartus 以简单双端口模式生成基准M20K，并发现其 F_{max} 为 645 兆赫。BRAMAC-2SA 和 BRAMAC-1DA 的运行频率分别为 586 兆赫（低 1.1× ）和 500 兆赫，而根据报告的 F_{max} 性能下降情况，CCB、CoMeFa-D 和 CoMeFa-A 的运行速度分别会慢 1.6×、1.25× 和 2.5× 倍。

图9展示了不同架构和 MAC 精度下，以每秒万亿次 MAC 运算计的峰值 MAC 吞吐量细分。与基线 Arria-10 设备相比，BRAMAC-2SA/BRAMAC-1DA 可将峰值吞吐量提升

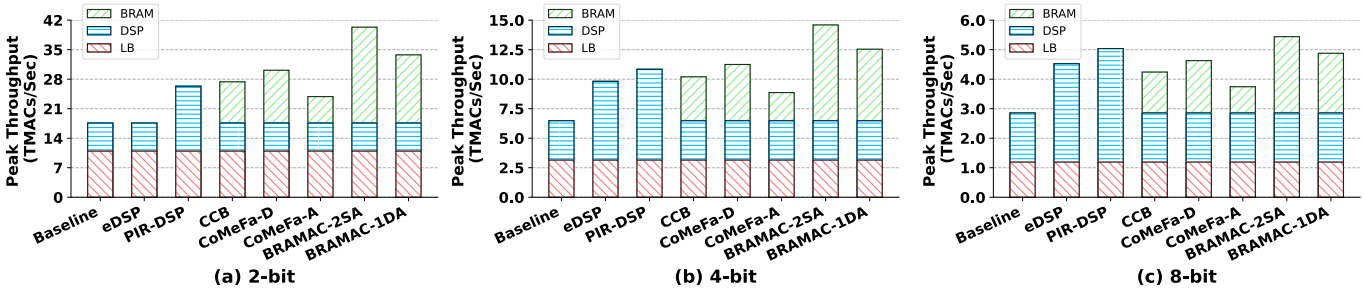


图9：Pea k不同架构在各种MAC 精度下的乘加吞吐量：(a) 2位，(b) 4位，(c) 8-bi t.

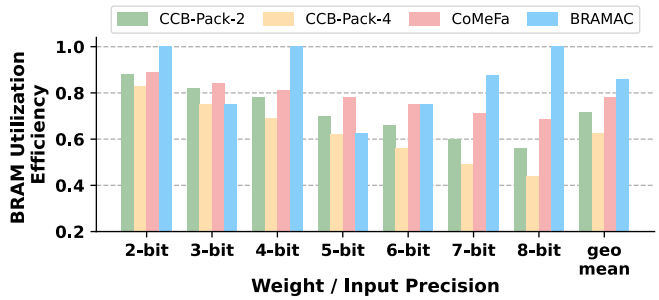


图 10：不同精度下DNN模型存储的BRAM利用效率比较。

2.6×/2.1×、2.3×/2.0× 和 1.9×/1.7× ，分别对应 2位、4位和 8位 MAC。尽管 CCB 和 CoMeFa 可以并行计算 160 个乘累加器，但它们受限于长延迟的位串行算术运算，因此吞吐量低于 BRAMAC。与低精度DSP架构相比，BRAMAC-2SA 在所有精度下都能提供更高的乘加吞吐量，而对于 8位 MAC，BRAMAC-1DA 的吞吐量仅略低于 PIR-DSP。需要注意的是，BRAMAC 是一种增强型 BRAM 架构，因此并不妨碍在同一 FPGA 上使用 eDSP 或 PIR-DSP。将 BRAMAC 与 eDSP/PIR-DSP 结合，可以进一步提升 FPGA 的乘加吞吐量。

*B. DNN*模型存储的*BRAM*利用效率

由于BRAMAC在与主BRAM完全解耦的独立虚拟阵列中执行乘加运算，因此能够高效存储深度神经网络模型。图 10比较了BRAMAC、CCB 和 CoMeFa 在存储精度从2 位到 8 位的不同深度神经网络模型时的BRAM 利用效率。这里，利用效率定义为 BRAM中可用于存储权重的有效容量比。更高的利用效率意味着可以使用更少的 BRAM块来存储深度神经网络模型，从而同时节省面积和功耗。对于 CCB，我们研究了两个变体CCB-Pack-2和 CCB-Pack- 4，它们分别将 2 个和 4 个顺序位串行MAC映射到同一 BRAM列。

BRAMAC 可在 2位、4位和 8位精度下实现 100% 的利用率。其他精度可以通过符号扩展至 4位或 8位，以较低的效率存储在 BRAMAC 中。尽管如此，BRAMAC 仍能实现最高的平均 BRAM利用率，分别比 CCB 和 CoMeFa 高 1.3× 和 1.1× 。这是因为 CCB 和 CoMeFa 使用额外的 BRAM 空间来存储临时值。乘积和部分和，而 BRAMAC 仅将临时结果存储在虚拟阵列中。对于 CCB，较高的打包因子可在速度较慢的存储器内归约之前执行更多顺序乘加运算，从而提升性能，但需要更多 BRAM使用量来保存输入向量的副本。另一方面，CoMeFa 提供单操作数在RAM外模式，可对输入向量进行数据流处理，避免将其复制到 BRAM；与 CCB 相比，这提高了利用效率。

C. 相较于 *CCB* 和 *CoMeFa* 的性能提升

我们使用通用矩阵-向量乘法（GEMV）对 BRAMAC、CCB 和 CoMeFa 的应用性能进行基准测试和比较。由于 BRAMAC-1DA 的 BRAM 面积和频率开销与 CCB/CoMeFa 相近，因此我们选择它进行本实验。我们假设只有一个块 RAM 可用于执行计算。此方法可评估按块 RAM 利用率归一化后的架构性能。我们同时考察持久化和非持久化（基于分块）的计算，前者不计入、后者计入将矩阵数据加载到单个块 RAM 所需的周期。由于这三种研究架构的数据映射和计算流程都是确定的，我们使用详细的分析模型，将给定的 GEMV 工作负载映射到各个架构，并统计所需的周期数。除乘加运算延迟外，我们的分析模型还会计入各架构中复制输入向量以及读出累加结果所需的延迟。

图 11 展示了在不同矩阵尺寸、精度（2位、4位、8位）和计算方式（持久型和非持久型）下执行通用矩阵-向量乘法时，BRAMAC-1DA 相对于 CCB 和 CoMeFa 的加速比。总体而言，对于 2/4/8 位的持久型（和非持久型）通用矩阵-向量乘法，BRAMAC-1DA 的加速比分别最高可达 3.3×/2.8×/2.4× （和 4.1×/3.4×/2.8×）。

在相同精度下，得益于 eFSM 能够在处理当前矩阵分块的同时加载下一个矩阵分块，BRAMAC-1DA 在非持久化计算中实现了更高的加速比。对于不同精度，BRAMAC-1DA 的加速比会随精度提高而降低。这是因为更高的精度会直接使 BRAMAC-1DA 的并行度降低 2×，并且处理更多输入位数需要更多周期。另一方面，精度提高时，CCB/CoMeFa 只会牺牲延迟，不会降低并行度。尽管如此，由于整体乘加吞吐量优于 CCB/CoMeFa，BRAMAC-1DA 在所有情况下仍能实现更好的性能，正如第

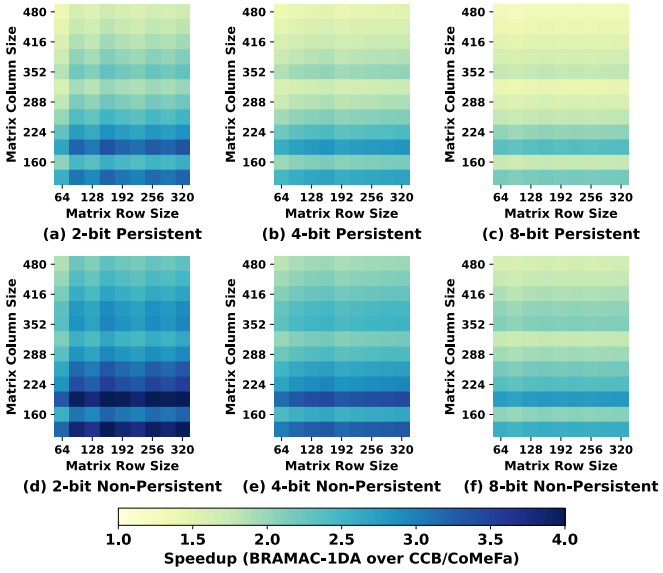


图 11：对于不同矩阵尺寸、精度和计算方式，BRAMAC-1DA 相较于 CCB/CoMeFa 在通用矩阵-向量乘法中的加速比（基于周期数）。

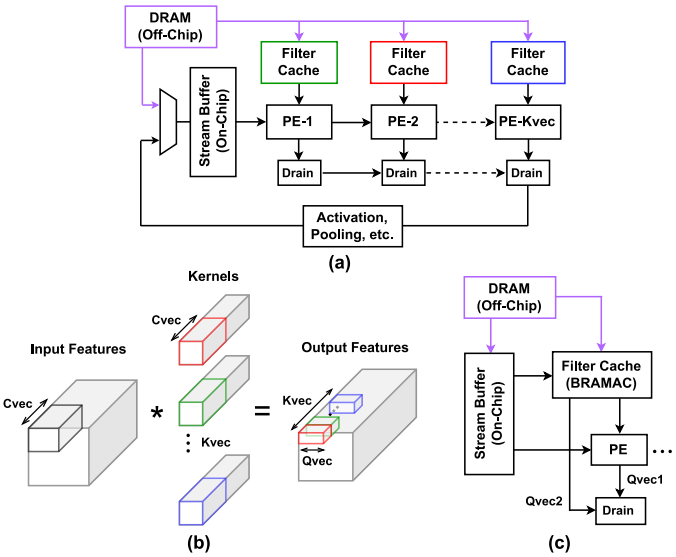


图12：DLA 在卷积神经网络中沿不同轴的 (a)架构和(b) 计算并行度。(c) DLA-BRAMAC 的架构（显示了一个 PE）。

VI-A 节所述。请注意，CCB/CoMeFa 用于定点乘法的位串行算法仅支持无符号数。预计在支持二进制补码乘加时，它们需要高得多的延迟。

随着矩阵行数的变化，BRAMAC-1DA 的加速比主要受向量化效率影响，而且精度越低，这种影响越明显。例如，考虑图 11(a) 中 2 位精度的持久化情形，此时 BRAMAC-1DA 可同时计算 20 个输出。如果矩阵行数为 64，即图 11(a) 中的第一列，那么至少需要 4 次迭代才能计算出大小为 64 的输出向量，而 BRAMAC-1DA 中只有 $64/80 = 80\%$ 的计算是有效的。另一方面，如果矩阵行数为 160，即图 11(a) 中的第四列，那么输出向量可以完美地划分为 8 次迭代，效率达到 100%，因此加速比更高；这也体现在第四列的颜色比第一列更深。4 位和 8 位精度的情形也存在类似趋势，但不那么明显。

沿着矩阵列数的变化，BRAMAC- 1DA 的加速比不仅取决于向量化效率，还取决于 CCB/CoMeFa 可实现的打包因子。例如，考虑图 11(f) 中的 8 位非持久化情形。如果矩阵列数为 480，即图 11(f) 的最上方一行，那么 CCB/CoMeFa 可以在同一 BRAM列上执行 3 次顺序乘加运算，然后再进行一次缓慢的存内归约，以摊销归约的延迟成本。另一方面，如果矩阵列数为 128，即图 11(f) 的最下方一行，那么 CCB/CoMeFa 在每次按位串行乘加运算之后都必须进行一次归约，从而导致长得多的延迟。相比之下，BRAMAC 的虚拟阵列不需要特殊的归约操作，而是在每次 MAC2 结束时执行原位累加。

D. 案例研究：将 *BRAMAC* 应用于英特尔的 *DLA*

为展示 BRAMAC 在采用非持久化权重存储的基于分块的深度神经网络推理中的可行性，我们将 BRAMAC 集成到英特尔的深度学习加速器（DLA） [9]，[10]，并开发了一个周期精确模拟器，用于对基准现场可编程门阵列中的 DLA 以及配备 BRAMAC 的增强型现场可编程门阵列中的 DLA（称为 DLA-BRAMAC）进行建模。原始 DLA 的设计旨在加速卷积神经网络（CNN），如图 12(a) 所示。它包含一个采用一维脉动结构的处理单元（PE）阵列、一个用于存储输入特征和输出特征的流缓冲区，以及一个用于存储权重的滤波器缓存。它可通过 Cvec、Qvec 和 Kvec 进行参数化，分别表示每个周期在输入深度、输出宽度维度和输出深度上的计算并行度，如图 12(b) 所示。对于 DLA-BRAMAC，流缓冲区可以同时向 PE 阵列和基于 BRAMAC 的滤波器缓存发送不同的输入特征，如图 12(c) 所示。这样，BRAMAC 可以与 PE 阵列协同，沿 Qvec 维度计算不同的输出。

与原始 DLA [9], 中采用的方法类似，我们进行设计空间探索，以为两个流行的卷积神经网络模型：AlexNet 和 ResNet- 34 找到最优的 DLA 和 DLA-BRAMAC配置（即 Cvec、Qvec和 Kvec）。我们的分析模型被设定为优化目标函数$perf * (perf/area)$，以平衡性能和面积成本。该模型假设所有乘法器均使用数字信号处理器实现，并且每个数字信号处理器可以使用 [36]中的 DSP打包技术执行一个8位、两个 4位或四个 2位乘法。在面积建模方面，我们使用 [9] 中的DLA面积模型来估算特定配置所需的数字信号处理器和块随机存取存储器数量。我们在面积建模中忽略自适应逻辑模块的数量，因为它们主要用于实现非计算密集型操作，并且预计在 DLA 和DLA-BRAMAC 中相近。为评估性能，我们的周期精确模拟器会考虑与 MAC2计算以及虚拟阵列累加器读出相关的延迟。需要注意的是，BRAMAC 的 eFSM 可以有效地对相邻的MAC2操作进行流水线处理，以隐藏权重复制的延迟，但每个卷积神经网络层的第一个 MAC2 除外。

T表 III：AlexNet 和 ResNet-34 的 DLA 与 DLA-BRAMAC 最优配置

模型	加速器	DLA			DLA-BRAMAC-2SA			DLA-BRAMAC-1DA		
		配置[1]	DSPs	块随机存取存储器	配置[2]	DSPs	块随机存取存储器	配置[2]	DSPs	块随机存取存储器
AlexNet	2位	(2, 16, 96)	1152	352	(1+2, 24, 140)	1260	1128	(2+2, 16, 100)	1200	816
	4位	(3, 16, 32)	1152	544	(1+2, 16, 100)	1200	1600	(1+1, 12, 130)	1170	1080
	8位	(3, 12, 24)	1296	868	(2+2, 10, 50)	1500	1740	(1+1, 8, 100)	1200	1664
ResNet-34	2位	(4, 12, 72)	1296	792	(1+2, 16, 140)	840	832	(2+2, 22, 80)	1320	924
	4位	(3, 8, 64)	1152	736	(2+2, 12, 70)	1260	972	(1+1, 16, 90)	1080	1056
	8位	(3, 4, 64)	1152	1452	(2+2, 6, 65)	1170	1530	(1+1, 12, 65)	1170	1788

[1]DLA 的配置值形式为（Qvec、Cvec、Kvec）。[2]DLA-BRAMAC 的配置值形式为（Qvec1+Qvec2、Cvec、Kvec），其中 Qvec1 和 Qvec2 分别表示由数字信号处理器和 BRAMAC 计算的输出特征数量。

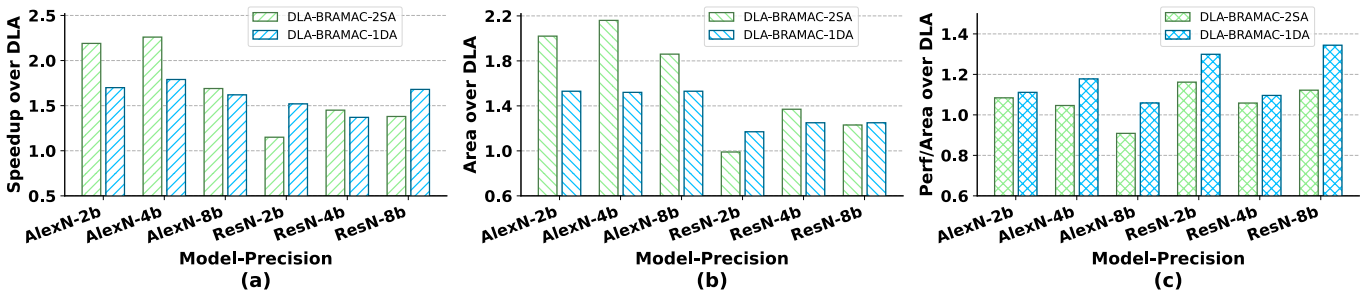


图 13：在不同精度下，DLA 与 DLA-BRAMAC 加速 AlexNet 和 ResNet 的比较：(a) 性能；(b) 所使用的 DSP 与 BRAM 面积；(c) 单位面积性能。

其中，启动初始权重复制还需要额外的 2 个周期。不过，鉴于每个卷积神经网络层都需要数千个周期才能完成，这一开销可以忽略不计。

表 III 总结了每种（加速器、模型、精度）组合的最优配置。图 13 展示了 DLA-BRAMAC 的性能和所使用的 DSP 与 BRAM 面积，并以 DLA 的相应指标进行归一化。所使用的 DSP 与 BRAM 面积根据 BRAMAC 的面积开销以及 [34] 中的面积模型计算得出。平均而言，与 AlexNet 的基准DLA 相比，采用 BRAMAC-2SA/BRAMAC-1DA 可实现 2.05×/1.7× 的加速比，但代价是 2.01×/1.52× 的 DSP 与 BRAM 面积，从而使单位使用面积的性能提升达到 1.01×/1.12×。对于 ResNet-34，采用 BRAMAC-2SA/BRAMAC-1DA 的平均加速比较低，为 1.33×/1.52× ，代价是 1.2×/1.22× 的 DSP 与 BRAM 面积，这对应于单位使用面积 1.11×/1.25× 的性能提升。更大的 DSP 与 BRAM 面积主要归因于用于计算的 BRAM 使用量增加以及 BRAMAC 的面积开销。

通常，如图13(a) 所示，与 ResNet-34 相比，BRAMAC-2SA和 BRAMAC-1DA 在 AlexNet 上实现了更高的加速比。这是因为BRAMAC更擅长支持较高的 Kvec，从而允许将同一个输入特-征与许多卷积核相乘。ResNet-34 早期且计算量最大的残差块的输出通道深度仅为64，而 AlexNet 的第一个卷积层的输出通道深度为 96。后者为 DLA-BRAMAC 提供了更大的自由度，使其能够以较高的向量化效率优化配置。然而，如图 13(b)所示，AlexNet更高的加速比伴随着更大的使用面积。比较这两种 BRAMAC 变体可以发现，对于所有模型-精度组合，BRAMAC-2SA 的单位使用面积性能增益均较低，如图 13(c) 所示。尽管 BRAMAC 的乘加吞吐量2SA略优于BRAMAC-1DA，其具有 2×块随机存取存储器面积开销，相较于BRAMAC-1DA。虽然我们的结果足以证明BRAMAC的面积开销是合理的，但我们预计，对于满足以下条件的深度神经网络加速器，可以获得更高的收益：(1)专门围绕BRAMAC的能力构建；(2) 用于加速具有更多矩阵乘法的深度神经网络，例如 Transformer [37]——未来我们将致力于这两个方面。

VII. 结论

本文提出了 BRAMAC，一种用于 FPGA 上乘加运算的 BRAM 内计算架构。据我们所知，BRAMAC 是首个具备以下特点的 BRAM 内计算架构：(1) 采用位串行与位并行混合数据流，支持使用二进制补码表示的可变精度乘加运算；(2) 在独立的虚拟阵列中进行计算，从而提高主 BRAM 阵列的利用效率；(3) 使用嵌入式有限状态机，在存内计算期间释放主 BRAM 端口。所提出的两种变体 BRAMAC-2SA/BRAMAC-1DA，在 FPGA 核心面积分别增加 6.8%/3.4% 的代价下，使大型 Arria 10 FPGA 的峰值乘加吞吐量在 2位、4位和 8位精度下分别提升 2.6×/2.1×、2.3×/2.0× 和 1.9×/1.7× 倍。与近期提出的两种 BRAM 内计算架构 CCB 和 CoMeFa 相比，BRAMAC 的 BRAM 利用效率也分别提高了 1.3× 和 1.1× 倍，同时在矩阵向量乘法方面显著优于这两种架构。将 BRAMAC-2SA/BRAMAC-1DA 与英特尔的 DLA（一种基于分块的深度神经网络加速器）结合后，AlexNet 和 ResNet-34 的平均加速比分别可达 2.05×/1.7× 和 1.33×/1.52× 。凭借同时支持持久型和基于分块的深度神经网络加速的能力，BRAMAC 有望成为未来面向 AI 优化的 FPGA 中极具实用价值的重要补充。

参考文献

[1] A. Krizhevsky、I. Sutskever 和 G. E. Hinton，《使用深度卷积神经网络进行 ImageNet 分类》，载于《神经信息处理系统进展》，2012年。[2] J. Deng、W. Dong、R. Socher、L.-J. Li、K. Li 和 L. Fei-Fei，《ImageNet：大型分层图像数据库》，载于《计算机视觉与模式识别》，2009年。CVPR 2009。IEEE 会议，2009年，第 248–255 页。[3] T. Brown、B. Mann、N. Ryder、M. Subbiah、J. D. Kaplan、P. Dhariwal、A. Neelakantan、P. Shyam、G. Sastry、A. Askell、S. Agarwal、A. Herbert-Voss、G. Krueger、T. Henighan、R. Child、A. Ramesh、D. Ziegler、J. Wu、C. Winter、C. Hesse、M. Chen、E. Sigler、M. Litwin、S. Gray、B. Chess、J. Clark、C. Berner、S. McCandlish、A. Radford、I. Sutskever 和 D. Amodei，《语言模型是少样本学习者》，载于《神经信息处理系统进展》，2020年，第 1877–1901 页。[4] M. Nagel、M. Fournarakis、R. A. Amjad、Y. Bondarenko、M. van Baalen 和 T. Blankevoort，《神经网络量化白皮书》，arxiv:abs/2106.08295，2021年。[5] B. Jacob、S. Kligys、B. Chen、M. Zhu、M. Tang、A. G. Howard、H. Adam 和 D. Kalenichenko，《面向高效纯整数运算推理的神经网络量化与训练》，载于《计算机视觉与模式识别会议》，2018年，第 2704–2713 页。[6] H. Wu，《GPU 上的低精度推理》，2019年。 [在线]。网址：https://developer.download.nvidia.com/video/gputechconf/gtc/2019/presentation/s9659-inference-at-reduced-precision-on-gpus.pdf [7] 英伟达，《用于 AI 推理的 INT4 精度》，2019年。 [在线]。网址：https://developer.nvidia.com/blog/int4-for-ai-inference/ [8] J. Fowers、K. Ovtcharov、M. Papamichael、T. Massengill、M. Liu、D. Lo、S. Alkalay、M. Haselman、L. Adams、M. Ghandi、S. Heil、P. Patel、A. Sapek、G. Weisz、L. Woods、S. Lanka、S. K. Reinhardt、A. M. Caulfield、E. S. Chung 和 D. Burger，《用于实时 AI 的可配置云端规模深度神经网络处理器》，ACM/IEEE 第45届年度国际计算机体系结构研讨会（ISCA），第 1–14 页，2018年。[9] U. Aydonat、S. O'Connell、D. Capalija、A. C. Ling 和 G. R. Chiu，《Arria 10 上基于 OpenCL 的深度学习加速器》，ACM/SIGDA 国际现场可编程门阵列研讨会（FPGA），2017年。[10] M. S. Abdelfattah、D. Han、A. Bitar、R. Dicecco、S. O'Connell、N. Shanker、J. Chu、I. Prins、J. Fender、A. C. Ling 和 G. R. Chiu，《DLA：用于神经网络推理加速的编译器与 FPGA 覆盖层》，第28届国际现场可编程逻辑与应用会议（FPL），第 411–4117 页，2018年。[11] 英特尔，《英特尔 Stratix 10 NX FPGA 概述》，2020年。 [在线]。网址：https://www.intel.com/content/www/us/en/products/details/fpga/stratix/10/nx.html[12] Xilinx，《UltraScale 架构 DSP 切片用户指南（UG579 v1.11）》，2021年。 [在线]。网址：https://docs.xilinx.com/v/u/en-US/ug579-ultrascale-dsp[13] 英特尔，《英特尔 Stratix 10 可变精度 DSP 模块用户指南（UG-S10-DSP）》，2021年。 [在线]。网址：https://www.intel.com/programmable/technical-pdfs/683832.pdf [14] Achronix，《Speedcore eFPGA》。[在线]。网址：https://www.achronix.com/sites/default/files/docs/Speedcore eFPGA Product Brief PB028.pdf [15] A. Boutros、S. Yazdanshenas 和 V. Betz，《拥抱多样性：面向 FPGA 低精度深度学习的增强型 DSP 模块》，第28届国际现场可编程逻辑与应用会议（FPL），第 35–42 页，2018年。[16] S. Rasoulinezhad、H. Zhou、L. Wang 和 P. H. W. Leong，《PIR-DSP：面向多精度深度神经网络的 FPGA DSP 模块架构》，IEEE 第27届年度国际现场可编程定制计算机研讨会（FCCM），第 35–44 页，2019年。[17] X. Wang、V. Goyal、J. Yu、V. Bertacco、A. Boutros、E. Nurvitadhi、C. Augustine、R. R. Iyer 和 R. Das，《用于 FPGA 高效深度学习加速的可计算块 RAM》，IEEE 第29届年度国际现场可编程定制计算机研讨会（FCCM），第 88–96 页，2021年。[18] A. Arora、T. Anand、A. Borda、R. Sehgal、B. Hanindhito、J. Kulkarni 和 L. K. John，《CoMeFa：面向 FPGA 的存内计算模块》，IEEE 第30届年度国际现场可编程定制计算机研讨会（FCCM），第 1–9 页，2022年。[19] A. Boutros 和 V. Betz，《FPGA 架构：原理与演进》，《IEEE 电路与系统杂志》，第 21 卷，第 4–29 页，2021年。[20] M. Eldafrawy、A. Boutros、S. Yazdanshenas 和 V. Betz，《面向高效深度学习推理的FPGA逻辑块架构》，ACM可重构技术与系统汇刊（TRETS），第13卷，第1–34页，2020年。[21] Xilinx，《DSP58架构》，2022年。 [在线]。可访问：https://docs.xilinx.com/r/en-US/am004-versal-dsp-engine/DSP58-Architecture[22] 英特尔，《英特尔Agilex可变精度DSP模块用户指南》，2021年。 [在线]。可访问：https://www.intel.com/programmable/ technical-pdfs/683037.pdf [23] M. Langhammer、E. Nurvitadhi、S. Gribok 和 B. M. Pasca，《Stratix 10 NX架构》，ACM可重构技术与系统汇刊（TRETS），第15卷，第1–32页，2022年。[24] M. Horowitz，《计算的能耗问题（以及我们能采取哪些措施）》，IEEE国际固态电路会议技术论文集（ISSCC），第10–14页，2014年。[25] D. M. Lewis、D. Cashman、M. Chan、J. Chromczak、G. Lai、A. Lee、T. Vanderhoek 和 H. Yu，《Stratix V架构增强》，载于ACM/SIGDA国际现场可编程门阵列研讨会（FPGA），2013年。[26] P. Judd、J. Albericio 和 A. Moshovos，《Stripes：位串行深度神经网络计算》，第49届IEEE/ACM国际微架构研讨会（MICRO），第1–12页，2016年。[27] C.-F. Lee、C. Lu、C.-E. Lee、H. Mori、H. Fujiwara、Y.-C. Shih、T.- L. Chou、Y. D. Chih 和 T.-Y. J. Chang，《一种用于人工智能边缘应用的12纳米、121-TOPS/W、41.6-TOPS/mm2全数字全精度SRAM计算存储器，支持可配置位宽》，IEEE超大规模集成电路技术与电路研讨会，第24–25页，2022年。[28] 英特尔，《英特尔Arria 10核心逻辑结构与通用输入/输出手册》，2022年。 [在线]。可访问：https://www.intel.com/ programmable/technical-pdfs/683461.pdf [29] J. Choi、K. Nam、A. Canis、J. H. Anderson、S. D. Brown 和 T. S. Czajkowski，《缓存架构与接口对基于FPGA的处理器/并行加速器系统性能和面积的影响》，第20届IEEE国际现场可编程定制计算机研讨会（FCCM），第17–24页，2012年。[30] R. Shi、Y. Ding、X. Wei、H. Li、H. Liu、H. K.-H. So 和 C. Ding，《FTDL：面向深度学习、具备高可扩展性的定制FPGA覆盖层》，第57届ACM/IEEE设计自动化会议（DAC），第1–6页，2020年。[31] S. Yazdanshenas、K. Tatsumura 和 V. Betz，《别忘了存储器：块RAM自动建模、优化与架构探索》，ACM/SIGDA国际现场可编程门阵列研讨会（FPGA），第115–124页，2017年。[32] 亚利桑那州立大学，《预测技术模型》，2012年。 [在线]。可访问：http://ptm.asu.edu/ [33] 英特尔，《Arria 10器件概述》，2022年。 [在线]。可访问：https://www.intel.com/programmable/technical-pdfs/683332.pdf [34] R. Rashid、J. G. Steffan 和 V. Betz，《比较TILT覆盖层处理器与OpenCL高层次综合的性能、生产力和可扩展性》，国际现场可编程技术会议（FPT），第20–27页，2014年。[35] 加利福尼亚大学伯克利分校，《ECE241，第18讲：加法器》，2003年。[在线]。可访问：http://bwrcs.eecs.berkeley.edu/Classes/icdesign/ee241 s03/Lectures/lecture18-adders.pdf [36] J. Sommer、M. A. ¨Ozkan、O. Keszocze 和 J. Teich，《DSP打包：将低精度算术运算塞入FPGA DSP模块》，arxiv.org/abs/2203.11028，2022年。[37] A. Vaswani、N. M. Shazeer、N. Parmar、J. Uszkoreit、L. Jones、A. N. Gomez、L. Kaiser 和 I. Polosukhin，《Attention Is All You Need》，arxiv:abs/1706.03762，2017年。