采用差分时间表示的低功耗脉冲神经网络加速：模块化硬件框架

Daniel Wi ndhager*¶ Lothar Ratschbacher† Bernhard A. Moser**‡ 、Michael Lunglmayr§

arXiv:2611.32758v1 [cs.NE] 24 Nov 2026

摘要—脉冲神经网络（SNN）具有良好的能效，非常适合能量受限型应用。其大部分节能效果源于脉冲序列的稀疏编码。我们表明，使用差分时间表示（DTR）可以更高效地表示脉冲序列；与地址事件表示（AER）或等间隔采样的1位脉冲序列相比，DTR所需的位数更少。为支持多个输入脉冲序列，我们严格证明了DTR序列可以以较低复杂度进行合并。我们还提供了一组采用该表示的模块化硬件流式组件，这些组件能够灵活连接，以针对特定SNN构建定制加速器架构。通过一个将模块化硬件组件组合成完整SNN实现的Python框架，可以简化针对所选SNN定制硬件架构的过程。我们使用该框架为海德堡脉冲数字数据集（SHD）、脉冲语音命令（SSC）和PSMNIST基准生成了三个加速器，每个加速器均提供吞吐量优化并行变体和资源缩减时分复用变体。我们报告了现场可编程门阵列（FPGA）和ASIC实现的结果；后者采用完全开源的流程，并以预测型ASAP7工艺设计套件（PDK）为目标。在SHD数据集上，ASIC实现达到92.46% 的分类准确率，同时以 20.37μJ的单次推理能耗维持每秒 ≈ 12,027次推理，在吞吐量和能效方面分别超过神经形态Loihi处理器 109× 和 1.8×。相应的FPGA实现无需使用DSP 模块，即可达到每秒 ≈ 6,023 次推理。在全部三个数据集上，合成的ASIC相比其FPGA 对应实现将单次推理能耗降低27–37× ，表明完全在开源设计流程中也能够实现具有竞争力的SNN加速器。

索引词—脉冲神经网络、差分时间表示、模块化硬件、流式处理、现场可编程门阵列、专用集成电路

一、引言

THE 人工智能（AI），更具体地说是机器学习（ML）领域正在不断发展，其明显趋势是网络架构规模越来越大、越来越复杂，进而导致数据集和参数空间不断扩大。

*智能无线系统，Silicon Austria Labs，奥地利林茨，daniel.windhager@silicon-austria.com ¶双重隶属：信号处理研究所，约翰内斯·开普勒大学，奥地利林茨

†智能无线系统，Silicon Austria Labs，奥地利林茨，lothar.ratschbacher@silicon-austria.com ‡信号处理研究所，约翰内斯·开普勒大学，约翰内斯·开普勒大学奥地利硅实验室智能无线系统实验室，奥地利林茨，bernhard.moser@jku.at

**双重隶属：哈根贝格软件能力中心（SCCH），4232 哈根贝格，奥地利 §信号处理研究所，约翰内斯·开普勒大学，奥地利林茨，michael.lunglmayr@jku.at

尽管图形处理器、更快的存储器和更高的带宽可以缓解服务器端应用对高存储器容量的需求所带来的影响，但大型模型对嵌入式设备和边缘应用构成了重大挑战，因为这些应用高度重视能效并受到计算能力的限制。因此，近年来，计算成本更低的架构逐渐受到学术界和业界的关注。其中一种方法是使用脉冲神经网络（SNNs）代替传统的人工神经网络（ANNs），这是因为脉冲神经网络具有更高的能效 [1],稀疏特性 [2]，并且根据神经元模型和脉冲编码的选择，还能降低计算量。这一点对于嵌入式设备和边缘应用尤为重要，因为它们通常对能效和推理时间设有严格限制。

值得注意的是，脉冲神经网络相较于人工神经网络所具备的显著效率优势，很大一部分源于其输入的稀疏性，以及用于这些脉冲序列的高效数据表示方案 [3]。这种表示尤为重要，因为它不仅影响编码脉冲序列所使用的符号数量，还直接决定处理这些脉冲所需的计算。因此，我们主张采用差分时间表示（DTR） [4]，而不是其他常用方案，例如地址事件表示（AER）经常采用的绝对时间表示，以及等距采样的1位脉冲序列。我们证明，就所需总位数而言，差分时间表示优于上述两种表示。此外，我们还强调这样一个事实：

DTR 所使用的差分时间可以直接用于计算神经元动力学。

与绝对时间表示不同，在跨不同脉冲序列比较脉冲时，DTR 需要考虑脉冲历史。我们提出了一种高效算法，无需转换为其他表示形式即可合并不同脉冲序列中的 DTR 脉冲。此外，我们还展示了该算法在数字硬件中的实现，证明了其实用可行性。

基于差分时间表示，我们提出了一套资源高效、可扩展且可配置的硬件组件。这些组件可以灵活连接，从而实现针对最先进脉冲神经网络定制的硬件加速器架构。硬件组件包括全连接模块、循环路径模块和延迟模块，以及一种用于高效合并不同脉冲序列中脉冲的实现—

在不同脉冲序列之间高效合并脉冲的机制。

借助这些组件，可以构建任意拓扑的前馈和循环脉冲神经网络，并在循环路径或前向路径中设置轴突延迟。此外，虽然我们的实现采用标准的漏积分发放（LIF）神经元模型，在经过线性层后进行激活，但也可以改用其他神经元模型（只需替换一个 VHDL 实体）。这种模块化方法应用于网络中的每一层，还带来了另一项优势：能够对各层分别进行量化，即使采用较低位宽，也能将量化引入的噪声降至最低。这使得多种网络架构成为可能，例如在脉冲语音命令（SSC）数据集上取得最先进准确率的网络 [5]。

为便于快速配置多个网络，我们提供了一个 Python框架，可生成实例化所有组件所需的相应顶层 VHDL 文件。随后，可通过 JSON 文件定义网络，其中指定脉冲神经网络的各个层及其相应特性，例如量化位宽。

我们对 FPGA 和 ASIC实现进行了广泛的综合。综合设计的性能指标与当前最先进的硬件加速器进行了基准比较，证明了这些设计及整体方法的可行性。

总而言之，本工作的主要贡献如下：

- **DTR 的理论与实证论证：**通过比较分析（第三节），证明了 DTR 相较于 AER 和采样脉冲在模块化流式架构中的优越性。
- **数学基础：** 提供了严格的证明，从而能够直接比较差分时间域脉冲序列（第三节、附录）。
- **模块化硬件套件：** 提供可配置的 VHDL 组件，支持任意 SNN 拓扑结构（包括循环连接和轴突延迟）、逐层量化以及高效的本地 SRAM 利用（第四节）。
- **自动化设计流程：** 一种 Python 工具可根据 JSON 配置自动生成顶层 VHDL，从而简化 SNN 到硬件的实现过程（第五节）。
- **基准测试与验证：**在 SHD、SSC 和 PSMNIST 数据集上评估性能和准确率，展示了在多种脉冲基准测试中的最先进结果（第 VI 节）。
- **综合结果：**使用开源工具报告了 ZCU102 现场可编程门阵列和初步 7纳米 ASAP7 专用集成电路实现的面积、功耗和性能指标（第 VI 节）。

II. 背景与相关工作

近年来，在图形处理器因功耗或外形尺寸限制而无法使用的情况下，研究人员和工程师采用现场可编程门阵列来加速神经网络。大量出版物提出了各种基于现场可编程门阵列的脉冲神经网络加速器架构，这些架构或针对特定应用，或设计为通用加速器 [6]–[11]。然而，其中许多设计与现有的中央处理器或GPU架构相似。它们由一个负责协调加速过程的中央控制器，以及许多较小的分布式处理单元（PE）组成，后者负责执行计算 [12]–[16]。这些处理单元通常排列成晶格状结构，类似于现代图形处理器中的脉动阵列，因此非常适合执行稠密矩阵运算，但对于脉冲神经网络固有的时间稀疏性而言，效果则不够理想。

然而，由于设计具有这种灵活性，如何将数据和输入（即脉冲）路由至各自的处理单元（PE）成为一项关键挑战。值得注意的是，现代加速器的大部分功耗都消耗在数据传输上，而非计算上 [17], [18]。因此，为了使新一代边缘设备能够使用功能更强大的模型，有必要限制所涉及的数据传输量。为优化效率，定制加速器可以通过合理利用本地存储器来缓解这一问题。

在尽量减少数据传输以及由此产生的处理功耗（包括传输和计算所产生的功耗）时，另一个重要考量是加速器输入的表示方式，即进入系统的脉冲和脉冲序列。数学上，脉冲是狄拉克δ脉冲，延迟特定时间（即若干个周期）。这些脉冲的序列构成一个脉冲序列。在硬件中，这些脉冲序列存在多种不同的表示方式，每种方式都有其相应的优点和缺点。请注意，这与频率编码或延迟编码等编码方法完全不同 [19], ，后者仅用于将模拟数据或自然向量转换为脉冲序列。脉冲表示关注的是如何高效地表示给定的脉冲序列，即尽量减少面向目标硬件的位宽开销。现有文献经常使用地址事件表示（AER）[20]–[23], ，但这种方法可能效率不高，因为它通常使用绝对时间来存储每个脉冲的延迟。 [4]中介绍的 DTR方法是一种更高效的脉冲表示方法，下一节将对此进行说明。

III. 面向数字硬件的高效脉冲表示

本节评估差分时间表示（DTR）作为一种资源高效的替代方案，用于在数字硬件中表示单个脉冲和脉冲序列。请注意，差分时间表示这一步是在数据初始采样之后、数据编码（例如使用频率编码或延迟编码）之后进行的，因此在很大程度上独立于这两个步骤。虽然有人可能会认为，单独表示脉冲是浪费资源的，因为许多研究采用频率编码，因此可以用一个表示发放率的标量来表示整个脉冲序列，但也有研究展示了采用其他方法的优势：这些方法依赖单个脉冲的时间特征来编码信息 [24], [25]。

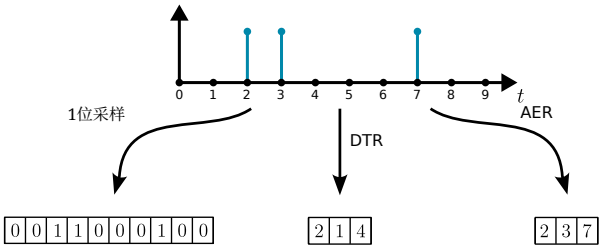

图1。使用1位直接采样、差分时间表示（DTR）和地址事件表示（AER）表示的示例脉冲序列对比。对于1位采样，整个脉冲序列以等间隔进行采样，生成一系列表示该脉冲序列的1位值。AER存储每个脉冲的发生时间（从某个预定义的起始点开始）。DTR仅存储脉冲之间的时间差，与AER相比可能需要更少的位数，与1位采样相比则需要更少的值；如本例所示，1位采样需要10位，DTR需要9位，而AER需要12位。

我们将DTR与当前文献中应用最广泛的两种方法进行基准比较：地址事件表示（AER）和等间隔1位采样。在传统AER中，脉冲通常由其源地址或目的地址表示，且常附加绝对时间戳和极性信息。尽管这种方法稳健，但在脉冲密集的序列中可能产生显著的数据开销。相比之下，1位采样将脉冲序列表示为连续的比特流，其中每一位对应一个固定的时间步（“1”表示存在脉冲，“0”表示不存在脉冲）。这种方法虽然易于实现，但在时间分辨率较高时扩展性较差。如图1所示，DTR仅对连续脉冲之间的时间间隔（时间增量）进行编码，从而优化了这一过程。如果位宽不足以编码时间差，则可将可编码的最大时间差定义为溢出符号，表示经过的时间已超过最大可表示时长。当遇到小于最大值的时间差时，将该最大时间增量与后续时间增量累加，即可还原这种情况下的时间增量。我们表明，通过关注相对时间而非绝对时间，DTR显著降低了所需位宽（相较于AER），并减少了1位采样固有的冗余数据。

*A. DTR*与*AER*对比

AER由于存在多方面的不足，对于持续运行的系统可能导致同步失败，因为在经过较长时间的运行后，用于表示当前时间的计数器通常会发生溢出。作为替代方案，我们在 [4] 中提出了DTR；就整个脉冲序列所需的位长度而言，在最坏情况下，DTR与绝对时间表示渐近等价。对于大多数其他情况，DTR可显著降低表示整个脉冲序列所需的位宽。

考虑一个包含 $K \in \mathbb{N}$ 个脉冲的脉冲序列，其中上一个脉冲发生在某个时间 $T \in \mathbb{N}_0$。采用绝对时间（AER 中通常采用这种方式，例如在 [26], [27] 中），每个表示单个脉冲的符号都需要 $\lceil \log_2(T+1) \rceil$ 位，而 DTR 中符号所需的位宽取决于脉冲之间经过的周期数的分布（通常小于最大时间）[1]。在不使用溢出符号的情况下，最坏情形是所有 K 个脉冲都在最晚可能的时刻发生，此时同样需要 $\lceil \log_2(T+1) \rceil$ 的位宽。如果使用溢出符号，那么 DTR 的最坏情形出现在长时间没有脉冲的情况下；这可能导致 DTR 为表示整个脉冲序列所需的位数超过绝对时间，前提是插入了许多溢出符号。实际应用中，可以根据预期的脉冲分布选择位宽，从而避免这种情况，将使用的溢出符号数量保持在最低水平，同时显著减少存储单个脉冲所需的硬件资源。

DTR 的最优情形出现在所有 K 脉冲都严格均匀（即等距）分布时，此时所需的位宽可降至 $\lceil \log_2(\binom{T+1}{K}) \rceil$。因此，位宽方面的确切节省量取决于脉冲的分布。然而，即使在最坏情况下，差分时间表示所需的位数最多也与绝对时间表示相同。后续分析将纳入溢出符号，从而进一步提高 DTR 的效率。

B. DTR 与采样的*1*位脉冲序列的比较

表示脉冲序列的另一种方法是对脉冲序列进行等距采样，并使用由此得到的全1和全0序列（假设仅使用幅度为一的正脉冲），其可表示为

$$S = (s_1, s_2, ..., s_{n-1}), s_i \in \{0, 1\}. \tag{1}$$

当位宽限制为单个位时，这种脉冲序列的采样版本与差分时间表示类似。在相应的 DTR 表示中，1 将被解释为溢出符号，表示在没有触发脉冲的情况下已经过去了一个采样时间，而 0 则表示当前时间步发生了脉冲。

由于脉冲序列本质上具有稀疏性，与1位采样相比，差分时间表示通常只需更少的位即可表示相同的脉冲序列。因此，就硬件需求而言，差分时间表示在采用绝对时间和等间隔1位采样之间实现了有利的平衡。

C. 为差分时间表示选择理想的位宽

在尖峰密度较低的场景中，如果根据尖峰间时间差的预期分布选择位宽，则采用较大位宽的差分时间表示，相较于1位采样时间表示更具优势。对于给定的脉冲序列，位宽不足会产生大量溢出符号，从而降低效率。此外，频繁生成溢出符号不仅会造成位利用率不佳，还会因需要

[1]当考虑没有专用时间限制的脉冲序列时，例如对于在流式数据上执行推理的架构，绝对时间表示概念的局限性会立即显现出来。

额外的处理周期而增加硬件延迟。相反，过大的位宽虽然可以消除溢出符号，但在编码稠密脉冲序列时会带来不必要的开销。因此，最高效的位宽必须取决于脉冲序列中脉冲的分布。

假设 b 是单个符号的位宽，那么时间差 $d \in \mathbb{N}_0$（相对于前一个脉冲）小于 $2^b - 1$ 的每个脉冲，都可以用单个符号表示。之所以采用严格不等式，是因为符号的位组合中有一种配置被保留为指定溢出符号。类似地，对于满足 $2^b - 1 \leq d < 2^{b+1} - 1$ 的、差值为 d 的每个脉冲，都可以用两个符号表示（其中一个为溢出符号）。按照相同的逻辑，对于更大的时间间隔，给定时间差 d 和位宽 b 时所需的符号数量为

$$N_b(d) = \left\lceil \frac{d+1}{2^b - 1} \right\rceil. \tag{2}$$

分子中的 $+1$ 项是保持不等式关系所必需的，适用于 $d = 0$ 或 $2^b - 1$ 能够整除 d 且无余数的边界情况，这是因为 d 和 $2^b - 1$ 都是整数。

现在令 $\mathcal{D}$ 表示整个数据集中所有脉冲间时间差的集合。那么，符号总数关于 b 的函数为

$$N_{\text{total}}(b) = \sum_{d \in \mathcal{D}} N_b(d) = \sum_{d \in \mathcal{D}} \left\lceil \frac{d+1}{2^b - 1} \right\rceil. \tag{3}$$

将其与 b 相乘，即可得到表示整个数据集中的所有脉冲所需的总位数，表示为

$$B_{\text{total}}(b) = bN_{\text{total}}(b) = b \sum_{d \in \mathcal{D}} \left\lceil \frac{d+1}{2^b - 1} \right\rceil. \tag{4}$$

理想情况下，我们希望找到使 (4) 最小的最优位宽 b^*，记为

$$b^* = \arg\min_{b \in \mathbb{N}^+} B_{\text{total}}(b). \tag{5}$$

由于向上取整函数的存在，(5) 不存在解析解。不过，可以通过在有限的、具有实际意义的 b 值范围内进行经验评估，确定 $B_{\text{total}}(b)$ 的最小值。图 2 展示了在我们评估模型的三个数据集（海德堡脉冲数字数据集（SHD）、SSC 和置换序列MNIST（PSMNIST））中出现的所有差分时间表示脉冲的分布。为生成该分布，我们使用经过充分训练的模型处理每个数据集的训练集、测试集和验证集划分，同时记录整个网络中的所有脉冲。这也表明，理想位宽的选择取决于数据集，进而取决于目标应用。例如，较高的位宽 $b = 4$ 使 PSMNIST 的总位成本最小，这表明其中存在较长时间段

在不发生脉冲的脉冲序列中，对于 SHD，$b = 2$ 的位宽最优。理想位宽较小，说明大多数脉冲都密集地连续出现，这进一步凸显了使用绝对时间表示脉冲的低效。

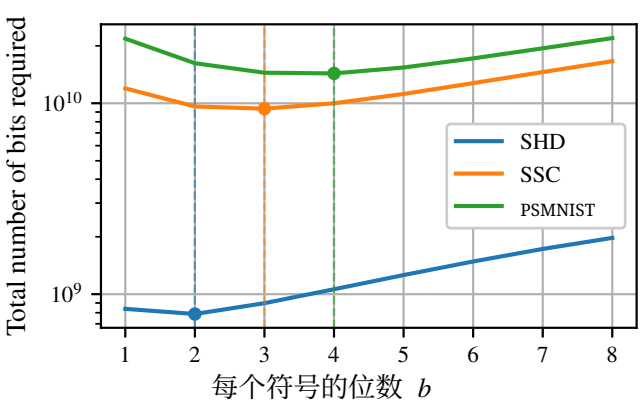

图 2。在推理整个数据集期间，对于完全训练的网络，根据为差分时间选择的位宽 b，对所有脉冲序列进行编码所需的总位数 B_{total}。横轴表示用于编码差分时间表示符号的位数，纵轴表示三个数据集 SHD、SSC 和 PSMNIST 中符号总数（包括溢出符号）乘以所选位宽后的结果。结果基于在训练集、验证集和测试集上进行推理期间，汇总所有网络层的总脉冲数。垂直线表示使每个数据集所需总位数最小的最佳位宽 b^*。

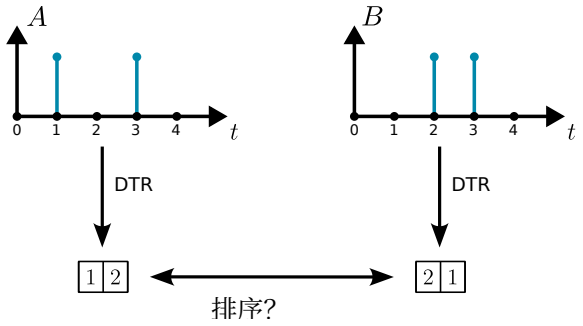

图3。以差分时间表示（DTR）给出的两个独立脉冲序列。仅从该表示中，有时无法立即判断（跨两个脉冲序列）下一个脉冲是哪一个。虽然第一个脉冲（来自A）很容易确定，但对于 A和 B，下一个脉冲都是 2；然而，脉冲序列 B 中该脉冲的实际（绝对）位置在时间 $t = 2$，因此早于脉冲序列 A 中时间为 $t = 3$ 的脉冲。

D. 比较差分时间中的脉冲序列

分析上述讨论的三种表示方式——AER、1 位采样和 DTR 后，结果表明 DTR 更具优势。不过，迄今为止，分析仅限于单个脉冲序列中的脉冲。在现实世界应用中，存在许多由多个独立源产生脉冲序列的场景，例如多个传感器或模数转换器会产生以差分时间表示的脉冲。因此，一个根本性问题随之而来：处理多个 DTR 脉冲序列时，如何识别下一个出现的脉冲?

对于采用差分时间表示的脉冲而言，确定多个脉冲序列之间的时间排序，比对地址事件表示（AER）而言更具挑战性。为说明这一点，考虑来自两个独立脉冲序列的两个采用 AER 编码的脉冲 $a \in \mathbb{N}_0$ 和 $b \in \mathbb{N}_0$。无需任何其他信息，我们便可确定哪个脉冲先发生：如果满足 $a < b$，则为 a；否则为 b。

然而，对于使用差分时间表示表示的相同两个脉冲，

无法进行直接比较，因为它们的全局时间参考取决于各自的历史记录，如图3所示。因此，对于差分时间表示，脉冲序列内部的顺序得以保留，但不同序列之间的顺序并不保证。如下一节所示，无需转换为其他表示形式，即可通过一种硬件高效算法恢复这种顺序。此外，该算法可以在线计算，即无需对输入脉冲序列进行时间缓冲。

下一节将说明，多个以差分时间表示的脉冲序列可以合并，而无需显式计算累积和，也无需随后在绝对时间下合并这些序列。我们将给出两个序列的证明，因为根据归纳法，该证明可以扩展到任意数量的序列。

设 $(A = (a_i)_{i=1}^{n},\ \leq)$ 为一个脉冲序列中脉冲绝对时间的有序序列（有序序列最常用的排序方式是按时间值自然排序）而 $(B = (b_j)_{j=1}^{m},\ \leq)$ 为以相同方式定义的另一个脉冲序列中脉冲绝对时间的有序序列。为清晰起见，本节中以差分时间表示的所有脉冲和脉冲序列前面都加上 Δ ，以避免混淆。除非另有说明，其他所有脉冲均假定以绝对时间表示。

差分时间的有序序列定义如下：$(\Delta A = (\Delta a_i = a_i - a_{i-1})_{i=1}^{n},\ a_{i-1} \leq a_i)$ 使用 $a_0 = 0$。

合并序列 $Y = A \cup B$ 由序关系实现。其差分时间的有序序列 ΔY 同样按照上文对 ΔA 所述的方式定义。对 ΔY 中的元素求累加和，将再次得到 Y。

命题 1。有序序列 ΔY 可以根据 ΔA 和 ΔB 通过以下过程直接生成：将输入序列 ΔA 和ΔB 的第一个元素相互比较。将较小值追加到输出序列ΔY中，从其所属序列中移除该最小值，并通过从另一个序列的第一个元素中减去该最小值来更新该元素（使用两个以上序列时：更新其他所有序列的第一个元素）。重复此过程，直到所有输入序列均为空。

证明概要。 上述命题可以基于以下观察来证明：由差分时间表示的序列的第一个元素彼此具有绝对关系，因为它们可以被解释为相对于某个共同

参考时间 $t_0 = 0$。图4阐释了这一概念。只有差分时间表示序列的首个元素，其相对顺序才能得到保证。因此，合并后的脉冲序列 ΔY 的第一个时间增量必须是各输入脉冲序列首个元素中的最小值。通过减去这个最小时间，更新其他首个元素（一般情况下，输入脉冲序列不止两个）可以理解为移动集合的零时间轴（即将时间轴重置为新的偏移零点）。相减后，所有差分时间表示的输入脉冲序列的首个元素

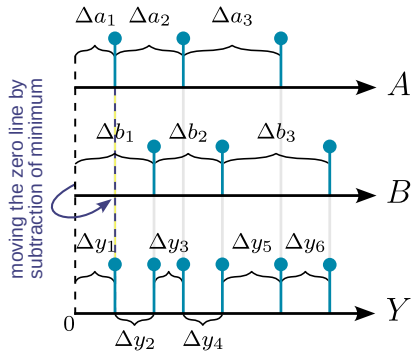

图4. 给定差分时间的脉冲序列合并。DTR序列$\Delta A = (\Delta a_1、\ \Delta a_2、\ \Delta a_3)$与 $\Delta B = (\Delta b_1、\ \Delta b_2、\ \Delta b_3)$合并为组合脉冲序列 $\Delta Y = (\Delta y_1、\ \Delta y_2、\ .....) = (\Delta a_1、\ \Delta b_1 - \Delta y_1、\ .....)$。

再次处于绝对关系，尽管此时采用的是偏移后的集合时间。随后，通过依次输出剩余元素中的最小值，生成序列ΔY。

□

有关更形式化的证明，请参阅附录。随机过程和网络取证文献中也出现了结构相似的论证 [28]–[30]。

E. 硬件中的脉冲序列合并

这种脉冲合并可以通过直接实现命题 1 来完成，从而得到图 5 所示的脉冲合并器˜元件，其中变量 $\Delta\hat{a}$和 $\Delta\hat{b}$分别表示持续更新的 Δa_i 和 Δb_j 。该更新通过减法实现，或用脉冲序列的下一个差分时间进行替换。

图 5 中的硬件模块将两条独立的 DTR 脉冲序列合并，生成一条新的 DTR 脉冲序列，其中包含来自两条输入脉冲序列的所有脉冲。通过级联脉冲合并单元，可以利用深度为 $\lceil \log_2(M) \rceil$ 、包含 $M-1$ 个脉冲合并单元的树结构，高效地将任意数量的脉冲序列合并为单条脉冲序列，其中 M 表示进入网络的独立脉冲序列数量。

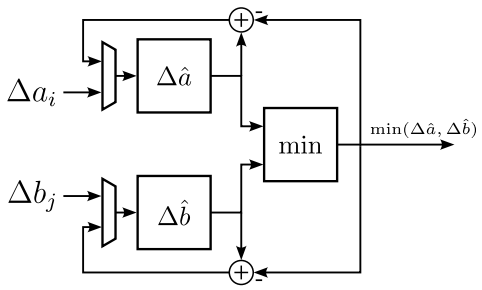

图 5. 能够合并两个以差分时间表示的独立脉冲序列的脉冲合并单元。比较输入脉冲的差分时间，将最小值传递至输出，同时从较大的剩余脉冲中减去该值。重复此过程，直到没有输入脉冲剩余。结果是一个将两个输入序列合并在一起的DTR序列。将该合并器元件级联成深度为 $\log_2(M)$ 的树（其中 M 为输入脉冲总数），即可合并任意数量的脉冲序列。

尽管该合并器元件能够找出接下来出现的脉冲，但它掩盖了产生该脉冲的脉冲序列信息。为恢复这一信息，我们引入一个独立的向量，在 $\log_2(M)$ 树的每个阶段将其扩展一位。根据每个脉冲合并单元中是上方还是下方的脉冲序列产生了最小值，向量会追加 1 或 0，从而生成一个用于标识脉冲来源脉冲序列的索引。表 I 列出了针对三个评估数据集的整个脉冲合并树所需的硬件资源。

F. DTR 脉冲合并器与绝对时间合并方法的比较评估

与其直接合并由差分时间表示（DTR）表示的脉冲，不如考虑先将脉冲转换回绝对时间，再按照绝对时间进行排序。正如我们将在此处展示的，与我们的方法相比，这种方法会在硬件资源方面产生额外开销。作为示例，假设我们再次希望从若干个不同的（但按时间排序的）脉冲序列中确定最早的脉冲，并且所有脉冲均以绝对时间给出。对所有首个脉冲进行简单的线性搜索即可实现这一点，但对于每个脉冲，其时间复杂度将达到 $\mathcal{O}(n)$。相反，人们很快会采用一种类似于前文所述的树结构，但其中的比较单元只需转发所接收两个脉冲中的最小值，即图 5 中去除减法机制后的结构。针对多种位宽综合这两种变体（即用于差分时间脉冲的脉冲合并单元，以及用于绝对时间脉冲的脉冲合并单元），即可得到图 6 所示的曲线。该曲线清楚表明，对于相同的位宽，用于地址事件表示（AER）的比较器所需资源（以查找表计）少于用于 DTR 的更复杂树结构单元（参见图 5）。由于 DTR 的表示位数少于 AER，按照数据集所选用的实际位宽可以看出，对于 DTR 而言，查找表资源使用情况基本相当，而寄存器方面的资源使用情况则低 2.5 倍。因此，即使假设输入和输出均采用 AER，使用 DTR 定位下一个发生的脉冲，与在绝对时间中进行搜索相比，仍可节省资源。

表 I 加速器中使用的脉冲合并器的资源利用率。

数据集	#查找表	#FF	功耗 [毫瓦]
SHD	7054	4158	137
SSC	7055	4158	156
PSMNIST	N/A§	N/A§	N/A§

§不适用，因为网络只有一个输入，因此无需脉冲合并器。

IV. 模块化硬件架构

我们提出了一套模块化基元库，可通过灵活组合来构建定制加速器，优先采用流式数据流，而非将数据路由至独立的计算单元。硬件模块包括：

全连接块 实现标准线性层，后接神经元激活层。

循环路径模块 将全连接块的输出连接回输入，从而支持循环神经元层。

延迟模块 可将脉冲延迟任意个周期，从而允许在循环路径和前向路径中使用任意学习型轴突延迟。

这些模块可以灵活地互连，也支持并行路径，以构建类似于 ResNet 架构 [31]中的残差连接。图 7 示意了一个用于 SHD 数据集的网络配置。该网络由两个带有 LIF 激活的全连接模块层组成，还包括一个中间层，其中包含循环路径以及在训练过程中定义的轴突延迟。

为实现这些模块之间的灵活连接，采用了带反压功能的 AXI4-Stream 接口 [32] 。使用 AXI4-Stream 协议中定义的有效-就绪握手，可使不同硬件模块彼此独立地工作，同时避免数据丢失。此外，为模块定义清晰的接口也使其能够独立开发，从而改进验证与开发。

硬件所使用的 DTR 脉冲由轴突索引和差分时间组成。轴突索引用于标识上一层中产生该脉冲的神经元；差分时间则表示自上一个脉冲以来经过的整数时间步数。假设网络第一层的输入来自网络之前设置的输入编码层。网络中的所有模块都会接收、处理并产生单脉冲，也就是说，网络中的所有脉冲都是按顺序处理的。与并行处理多个脉冲相比，这确实会影响性能，但能大幅简化

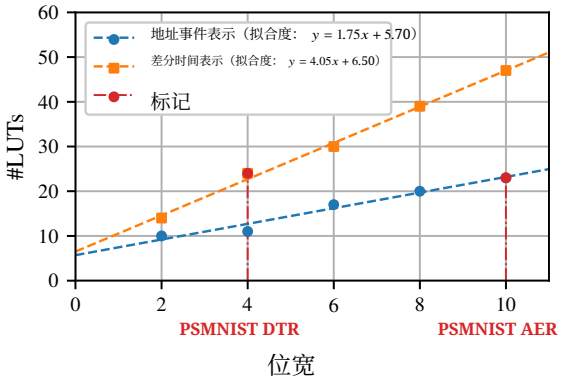

图 6. 用于差分时间表示（DTR）和绝对时间（AER）表示输入的脉冲合并树中的单个比较单元的 LUT 使用量（见图 5）。尽管对于相同数量的输入位，DTR 需要更多资源，但 DTR 实际所需的输入位数量更少；如两个标记所示，它们分别表示 PSMNIST 数据集使用的 DTR 和 AER 位宽。

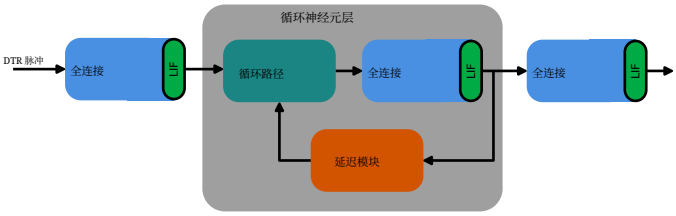

图 7. 用于 SHD 数据集的加速器硬件系统架构示意图。该网络由两个前馈（全连接）线性层组成，每层都带有一个 LIF 激活层，此外还包括一个循环神经元层，该层后接一个 LIF 激活层。循环神经元层还包含轴突延迟（即特定神经元的输出会延迟指定数量的时间步），这些延迟与权重在训练过程中一同学习。

硬件，并最大限度地利用稀疏脉冲序列，即跳过没有脉冲出现的时间步。此外，由于神经元的非线性阈值操作会使脉冲之间产生时间依赖性，因此并行处理多个发生在不同时间步的脉冲并无显著优势。

A. 全连接块

全连接块如图 7 所示，是这些架构的基本构建模块。它表示标准的矩阵-向量乘法

$$y = \mathbf{W}x, \tag{6}$$

其中 $\mathbf{W} \in \mathbb{R}^{n\times m}$ 是一个矩阵，$y \in \mathbb{R}^n$ 和 $x \in \{0,1\}^m$是向量。具体而言，$\mathbf{W}$ 是网络训练过程中学习得到的权重矩阵，而 x 是由脉冲组成的输入向量。由于乘数为 0/1，这种乘法在硬件中会形成累加过程；也就是说，块在接收到来自轴突 i 的事件后，会提取权重矩阵$\mathbf{W}$ 的第 i 列，并更新相应的神经元电位。矩阵-向量乘法完成后，会应用一个 LIF 激活层，其动力学如下：

$$\begin{aligned} P_k &= P_{k-1}\beta^{\Delta t} + y_k - r_{out} \\ r_{out} &= \begin{cases} \theta, & \text{if } P_k \geq \theta \\ 0, & \text{if } P_k < \theta, \end{cases} \end{aligned} \tag{7}$$

其中，$P_k \in \mathbb{R}$ 是在时间步 k时的神经元电位，$\beta = 0.5$是电位的衰减率，$\Delta t \in \mathbb{N}_0$ 是自上一个事件以来经过的时间，$k \in \mathbb{N}_0$ 是至少一个输入脉冲 $s_i \in \{0,1\}$ 发生的时间，y_k 是权重矩阵$\mathbf{W}$ 与时间步 k的输入脉冲向量 x_k 的贡献所得到的值，$r_{out} \in \mathbb{R}$ 表示复位值，$\theta \in \mathbb{R}$ 是阈值。值得注意的是，神经元电位的衰减需要自上一个脉冲以来的时间增量，这直接利用了差分时间表示（DTR）。全连接块的主要功能是实现式（6）中的矩阵向量乘积，以及基于式（7）的激活规则。由于该块的输入由单个

脉冲，这两个方程都不需要乘法运算。具体而言，通过选择 $\beta = 0.5$，电位衰减可以

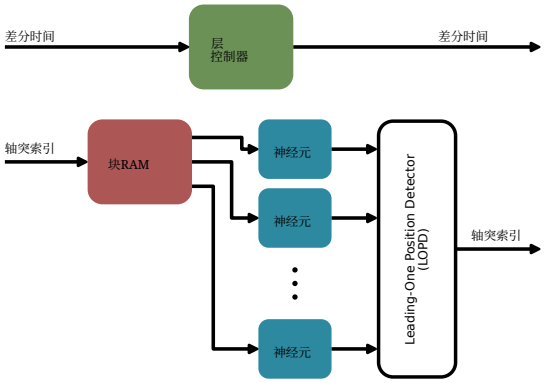

图8. 全连接块的内部结构，用于实现线性层，后接 LIF 等神经元激活函数。轴突索引直接用作块RAM的地址，块RAM中存储着网络的权重。该权重向量被提供给相应的神经元处理单元，由它们负责处理神经元动力学和脉冲生成。脉冲被送入 LOPD，确保当前处理的脉冲在传递到下一层后被重置。层控制器负责协调数据流，并向各个神经元发送衰减信号，以触发电位衰减过程。

实现为右移，从而无需使用成本高昂的 DSP切片。

整个全连接单元的框图如图8所示，其中综合后每个神经网络各层的完整资源使用情况列于表II。每个输入脉冲由差分时间和轴突索引组成，二者按照图8所示分别进行处理。轴突索引可直接用作地址，以访问包含权重的块RAM，随后将这些权重分别加到各个神经元电位上。如果神经元数量较大，则并行实例化多个块RAM，以确保获取权重时具有足够的带宽。

直接从块RAM接收这些权重的神经元是标准LIF神经元，其由一个用于存储神经元电位的寄存器，以及用于检测阈值穿越和实现电位衰减的逻辑组成。尽管这里仅使用LIF神经元，但模块化设计以及组件之间定义明确的接口，使得可以无缝替换神经元实现，例如替换为生物学上更合理的Izhikevich模型 [33]。

B. 动态神经元复用

由于该架构的硬件资源会随神经元数量和权重数量自然扩展，因此必须实例化所有神经元及其各自的神经元电位，这会导致大量的查找表和触发器占用。然而，可以通过对现有神经元进行时分复用，在性能与所需硬件之间实现权衡。我们无需同时将所有神经元电位保存在寄存器中，也无需在单个时钟周期内加载所有神经元的权重；相反，可以在设计中实例化数量更少的神经元（即“物理”神经元），然后依次为这些神经元加载所需的电位。不过，这需要一种加载和存储

表II 在 ZCU102（XCZU9EG）上未采用时分复用时全连接层的 FPGA 资源利用率和功耗。百分比相对于器件总量（274080 个查找表、548 160 个触发器、32 832 Kbit 块RAM）计算。

层	LUTs		FFs		BRAM		功耗
	[k]	[%]	[k]	[%]	[Kbit]	[%]	[mW]
SHD							
1	24.9	9.1	6.8	1.2	315	1.0	799
2	24.9	9.1	6.8	1.2	1152	3.5	682
3	1.5	0.6	0.6	0.1	45	0.1	74
SSC							
1	23.8	8.7	7.3	1.3	792	2.4	622
2	23.8	8.7	7.3	1.3	1024	3.1	876
3	23.8	8.7	7.3	1.3	1024	3.1	907
4	3.0	1.1	1.1	0.2	70	0.2	136
PSMNIST							
1	4.2	1.5	1.6	0.3	37	0.1	116
2	14.5	5.3	4.8	0.9	457	1.4	554
3	14.1	5.1	4.8	0.9	702	2.1	576
4	0.7	0.2	0.3	0.1	17	0.1	29

相应神经元电位的机制。图9以示意方式展示了这一策略的一个示例。通过采用这种具有动态神经元复用功能的时分复用全连接模块，如果整体脉冲吞吐量可以略有降低，所需的硬件将显著减少。

当希望减少实例化神经元的数量时，一个自然的问题是应为每层选择多少个神经元。图9所示的硬件以批次方式计算单层中的逻辑神经元；也就是说，硬件中并未实际实例化全部神经元，而是仅配置较小的物理神经元子集，然后通过时分复用提供与完整实例化的层等效的功能。这种动态重载只需一个时钟周期（因为所有神经元的神经元电位可以同时进行写入和读取），而衰减、激活以及将权重加到神经元电位上的处理，则与完整实例化的层相同，并且与实例化的神经元数量无关。因此，计算整个网络所需的时钟周期数必然取决于各层所需的批次数。由于各层之间直接连接，中间没有先进先出队列或缓冲区，因此当可用数据不足时，它们可能会相互停滞。此外，如果没有发生停滞或阻塞，则所有层都能够并行工作。综合这两点来看，均匀分配批次，即每层都计算相同数量的批次，是一种自然的选择。我们将这种分配称为成比例分配，因为每层的物理神经元数量与该层中存在的神经元数量成正比。

尽管这种比例分配无法保证
在所有数据集、神经元配置和超参数下均达到最优；仿真表明，其与可能实现的最佳分配非常接近。图10显示，最佳

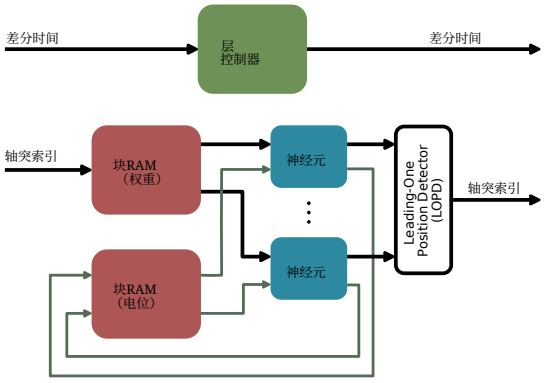

图9. 使用动态神经元复用且神经元数量受限（即少于神经网络中现有数量）时全连接块的内部结构。通过使用轴突索引直接对块RAM进行索引来获取权重，而另一个块RAM则存储相应的神经元电位。由于硬件中仅存在所需神经元数量的一个子集，因此还必须针对正在处理的每个神经元批次动态重新加载电位。LOPD负责对发送至下一层的脉冲进行串行化处理。当LOPD处理完所有当前脉冲后，便会切换至后续批次的神经元、权重和电位。

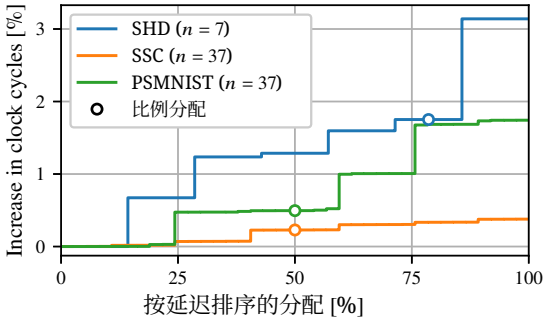

图10. 对固定的276个物理神经元预算进行所有评估分配时，总时钟周期的相对增幅（百分比）。每个数值均相对于相应数据集的最快分配进行测量。在每个数据集中，分配按照从最快到最慢的顺序，在 x轴上列出，每个分配对应一个数据点。 n 表示每个数据集评估的分配数量。圆圈标记与逻辑层大小成比例的分配。

分配与比例分配仅相差1–2个百分点。该结果是在三个数据集上，对比例分配附近的三种范围（神经元分配的所有变化，即比比例分配高三档或低三档）进行测试，并对40次随机运行取平均后得出的。结果表明，对于与比例分配相差更大的分配方案，这种接近最优分配的情况依然成立，如图11所示。此外，图11还表明，分配空间在最优点附近似乎近似凸形；比例分配接近该最优点，因此也可以通过迭代改进逐步将这一分配方案推向最优值。

对于希望尽量减少权重数据传输的应用，可以采用[34]中介绍的权重驻留方法，通过利用时间窗口大小优化技术（TWSO）来提高权重的复用率。借助TWSO，计算前馈路径所需的权重可以在多个时间步中重复使用，从而

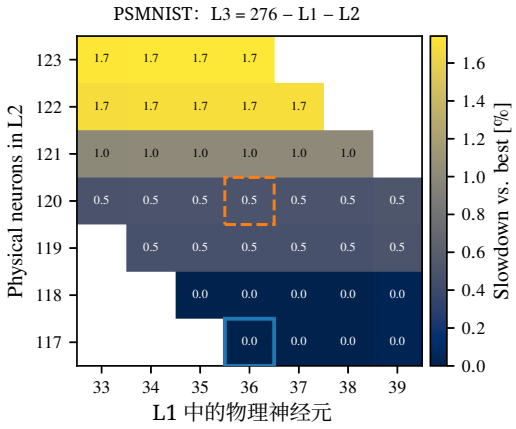

图11. PSMNIST网络的延迟与物理神经元分配的关系。(1–64–212–212–10)网络中的三个动态复用循环层L1–L3共享276个物理神经元的固定预算。最后一层采用完全实例化。每个单元格显示相对于最快分配方案的总时钟周期增加百分比，该百分比取40个测试样本的平均值，每个样本包含784个时间步，其中L3 = 276 –L1 – L2。实线和虚线轮廓分别标示最快分配方案（36/117/123）以及与逻辑层大小成比例的分配方案（36/120/120）。未对空白单元格进行仿真，因为L3将超出所评估的117–123个物理神经元范围。

直接减少RAM传输的次数。然而，该技术需要使用缓冲区存储多个时间步的前馈脉冲，因此本文未采用该技术。

C. 循环层与循环路径块

循环层可以通过扩展全连接块来构建。如前所述，全连接块计算方程 $\mathbf{y} = \mathbf{W}\mathbf{x}$，其中 $\mathbf{W} \in \mathbb{R}^{n\times m}$，这与循环全连接层所使用的方程类似

$$\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{W_r}\mathbf{x_r} \tag{8}$$

其中，$\mathbf{W_r} \in \mathbb{R}^{n\times n}$ 和 $x_r \in \{0,1\}^n$ 分别是循环权重和循环输入。由此，我们可以构建一个新的权重矩阵

$$\mathbf{W}' = \begin{bmatrix} \mathbf{W} & \mathbf{W}_r \end{bmatrix} \in \mathbb{R}^{n\times(m+n)} \tag{9}$$

以及一个由组合输入构成的新向量

$$\mathbf{x}' = \begin{bmatrix} \mathbf{x} \\ \mathbf{x_r} \end{bmatrix} \in \{0,1\}^{m+n}. \tag{10}$$

用这些量改写（6）后，可得

$$\mathbf{y} = \mathbf{W}'\mathbf{x}' = \begin{bmatrix} \mathbf{W} & \mathbf{W}_r \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{x_r} \end{bmatrix} \tag{11}$$

该结果可以与现有的全连接块配合使用。权重矩阵$\mathbf{W}$ 和 $\mathbf{W_r}$ 可以在设计综合之前合并为矩阵$\mathbf{W}'$。然而，输入 $\mathbf{x}$ 和$\mathbf{x_r}$需要在运行时进行合并，这由循环路径块负责，该模块充当仲裁器。

在硬件中，两个输入 $\mathbf{x}$和 $\mathbf{x_r}$ 均为 DTR 脉冲流，其中既包含来自前一层的脉冲，也包含循环脉冲。最初，所有脉冲

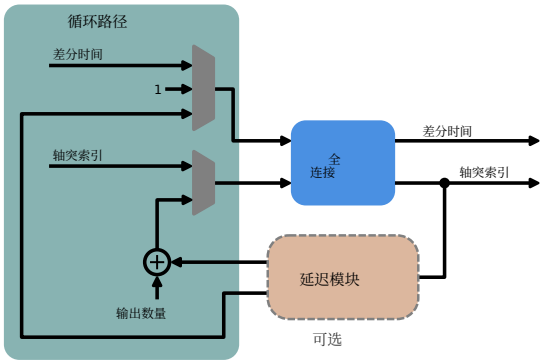

图12。循环层通过复用全连接块构建。循环路径块要么转发传入脉冲（来自前一层），要么转发由该层生成的循环脉冲。通过加上当前层的输出数量来调整轴突索引，以确保在全连接块中正确进行权重选择。差分时间由延迟模块确定；为向该层引入更长的循环轴突延迟，可以添加该模块。对于直接循环脉冲，即不带轴突延迟的脉冲，该时间设为1。

来自前一层的脉冲都会进入循环路径块，直到该时间步的上一个脉冲触发全连接块的脉冲生成。这些脉冲随后进入循环路径块，该块会处理所有循环脉冲，直至该时间步也处理完毕；此时，循环路径块将恢复处理来自前一层的脉冲。

为确保循环脉冲（$\mathbf{x_r}$）能够选择正确的权重（$\mathbf{W_r}$），必须调整其轴突索引。具体而言，将输入数量加到循环脉冲的轴突索引上，即加上循环权重在存储器中的位置偏移量。此外，由于在最简单的情况下，循环脉冲恰好在触发它的脉冲之后 1 个时间步产生，因此在不使用轴突延迟时，循环脉冲的时间差被设为 1。在使用学习型轴突延迟的情况下，循环脉冲的时间差由延迟模块直接确定。图12展示了包含循环路径块的循环层框图；综合后的完整资源使用情况列于表 III中。

D. 延迟模块

延迟模块通过将传入脉冲延迟指定数量的时间步来实现可编程延迟；该数量可针对每个轴突索引进行配置，并在正确的时间步输出先前接收到的脉冲。因此，需要一种机制来获取给定轴突索引对应脉冲所存储的延迟，并在其延迟大于 0 时存储传入脉冲。

延迟模块与全连接块一样，也接收一个脉冲作为输入，并产生一个脉冲作为输出。这可以无缝集成到基于DTR的架构中，因为轴突延迟天然是相对于触发脉冲以差分时间表示的。由于硬件模块根据脉冲的轴突索引检索存储的延迟，因此此操作无需输入脉冲的具体差分时间。

表 III 无时分复用情况下 ZCU102（XCZU9EG）上循环层的 FPGA资源利用率和功耗。百分比相对于器件总资源（274080 个查找表、548 160 个触发器、32 832 Kbit 块 RAM）计算。非循环层未列出。

层	LUTs		FFs		BRAM		功耗
	[k]	[%]	[k]	[%]	[千位]	[%]	[mW]
SHD							
2	25.2	9.2	7.0	1.3	1441	4.4	704
SSC							
1	24.1	8.8	7.6	1.4	1081	3.3	638
2	24.1	8.8	7.6	1.4	1313	4.0	892
3	24.1	8.8	7.6	1.4	1212	3.7	926
PSMNIST							
1	4.4	1.6	1.8	0.3	166	0.5	125
2	14.8	5.4	5.1	0.9	746	2.3	570
3	14.4	5.3	5.1	0.9	991	3.0	595

表IV 在无时分复用的情况下，ZCU102（XCZU9EG）上每个循环层的延迟模块的 FPGA 资源利用率和功耗。百分比相对于器件总量（274 080 个查找表、548 160 个触发器、32 832 Kbit 块RAM）计算。未列出非循环层。

层	LUTs		FFs		BRAM		功耗
	[#]	[%]	[#]	[%]	[Kbit]	[%]	[mW]
SHD							
2	220	0.08	168	0.03	289	0.88	14
SSC							
1	245	0.09	168	0.03	289	0.88	15
2	219	0.08	168	0.03	289	0.88	13
3	217	0.08	168	0.03	289	0.88	14
PSMNIST							
1	148	0.05	148	0.03	129	0.39	8
2	235	0.09	168	0.03	289	0.88	15
3	234	0.09	168	0.03	289	0.88	17

延迟模块中处理输入脉冲的各个阶段如下：

1) 首先，使用轴突索引访问延迟存储器，即访问块RAM，以检索产生该脉冲的神经元所分配的延迟。2) 如果延迟为 0，则直接将脉冲转发至输出阶段。

3) 如果延迟大于零，则脉冲会延迟相应的时间步数。为实现这一点，脉冲会存储在本地块RAM中，以便稍后读取。为简化读取过程，脉冲会根据其延迟存储在RAM中的相应位置，也就是说，将同一时间步发生的脉冲聚合到时间分箱中。具体而言，

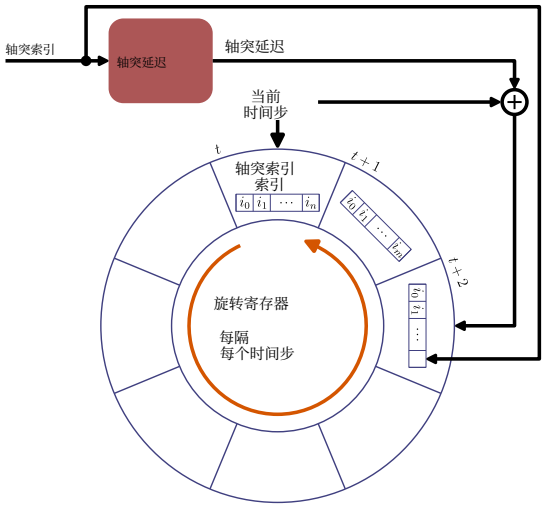

图13. 延迟模块内旋转存储器用于存储和检索脉冲的示意图。使用轴突索引检索先前存储的延迟，并结合偏移指针确定放置脉冲的地址，以便之后检索。旋转存储器置于块RAM中，以确保高效访问。每个时间步都会发送偏移指针所指位置存储的脉冲，随后偏移指针向前移动。

通过将这些分箱存储在旋转存储器中，可以高效地实现这一过程，如图13所示。根据延迟模块需要在未来哪个时间步转发脉冲，通过将延迟加到当前偏移指针上，对脉冲进行分组。随后，可将整个旋转存储器置于块RAM中，以确保高效的索引和访问。

每经过一个输入脉冲的时间步，偏移指针就会递增。因此，它始终指向包含当前时间步内产生的脉冲的存储桶。因此，该偏移指针用于检索先前存储、并将在当前周期发出的脉冲。

综合后，每个层和每个神经网络的延迟模块的完整资源使用情况均列于表IV中。

V. 可复现性与代码可用性

为了将我们的硬件与其他现有加速器进行评估比较，我们使用了三个标准神经形态基准数据集。这些数据集分别是海德堡脉冲数字数据集（SHD）、脉冲语音命令（SSC）和排列顺序 MNIST（PSMNIST）。SHD 和 SSC 是基于音频的数据集，而 PSMNIST 是基于图像的数据集，这表明本文使用的脉冲神经网络在短音频数据集以及长程序列模式上均表现良好。由于 SHD 和 SSC 是成熟的神经形态基准数据集，因此它们已直接提供脉冲和脉冲序列，即无需使用频率编码或延迟编码等编码方式。PSMNIST 数据集包含经过阈值处理以将输入二值化的原始像素数据。

针对这些数据集训练了三个不同的脉冲神经网络，其初始配置和超参数改编自 [35]。这些网络的结构和层大小见表 V。开发的 Python框架可自动为这些架构生成相应的硬件，也可应用于其他网络结构。

对网络进行了少量修改，以确保其与本文所述硬件兼容。具体而言，网络中省略了偏置参数和经过修改的批归一化（鉴于全连接块的设计不含偏置参数，批归一化无法融合到线性层的权重中）。此外， [35] 使用的 PSMNIST 数据集网络将原始像素数据输入第一个线性层，因此需要执行乘法运算。因此，在训练过程中引入了阈值参数 $0 \leq \theta \leq 1$ ，用于将像素值二值化为集合 $\{0, 1\}$。这种阈值处理使总体准确率降低约 0.8%，但完全消除了硬件执行乘法运算的需求，因此被认为适用于这一使用场景。虽然这种阈值处理足以应对 PSMNIST 数据集，但对于其他更多样化的数据集，采用这种方法可能会导致准确率显著下降。因此，可以采用其他技术（例如 [36] 中针对二值化神经网络（BNN）提出的随机二值化方法）来进一步提升性能。

网络的训练使用 [35]，公开提供的源代码，并采用其建议的超参数完成。本文所述结果通过分别训练三个网络各五次并对结果取平均获得。对于硬件结果，选取每个数据集上的最佳运行结果进行量化，并使用定制测试平台评估每个特定网络的硬件。这些测试平台加载真实的量化权重以及来自每个数据集的一个随机抽取的测试样本，将其输入加速器，然后将结果与 Python 结果进行比较。对于每个加速器，均执行了 1000 次此类仿真。报告的硬件延迟是这些运行的平均值，硬件准确率则是正确分类的样本所占的比例。

整个硬件设计均使用 VHDL 开发，并与网络的位精确 PyTorch 实现同步完成验证。硬件的 RTL 源代码以及基于 Python 的生成工具可在 https: // github.com/DanielWindhager/streaming-snn 找到。该仓库还包含上述数据集对应加速器的 JSON 配置文件。

VI. 实验结果

表 VI 汇总了 SHD、SSC 和 PSMNIST 测试集上的分类准确率（每项结果均为五次独立训练运行的平均值），以及相应专用集成电路和现场可编程门阵列（FPGA）实现的吞吐量、时钟频率、资源和功耗数据。对于每个数据集，均评估了两种配置：吞吐量优化的并行 版本和资源精简的时分复用 版本。SHD 数据集上达到的 $92.46 \pm 0.69\%$ 准确率，仅有一个先前的加速器超过了这一水平；但该加速器使用的 FPGA 规模大得多，所需查找表（LUT）约为我们的并行配置的四倍，功耗也约为其三倍（10.24 W 对比 3.325 W）。

表 V 面向 SHD、SSC 和 PSMNIST 数据集的加速器在硬件中实例化的网络规模和神经元数量。

加速器	In	隐藏层 1	隐藏层 2	隐藏层 3	Out
SHD（并行）	$140^{\S}$	256	256^{R}	–	20
SHD（时间复用）*	$140^{\S}$	64 (256)	64^{R} (256)	–	20
SSC（并行）	$140^{\S}$	256^{R}	256^{R}	256^{R}	$35^{\ddagger}$
SSC（时间复用）*	$140^{\S}$	64^{R} (256)	64^{R} (256)	64^{R} (256)	$35^{\ddagger}$
PSMNIST（并行）	1	64^{R}	212^{R}	212^{R}	$10^{\ddagger}$
PSMNIST（时分复用）*	1	32^{R} (64)	64^{R} (212)	64^{R} (212)	$10^{\ddagger}$

*采用动态神经元复用的时分复用（TM）：网络与并行（Par.）版本相同。数值表示硬件中实例化的物理神经元数量，括号内为其模拟的逻辑神经元数量。§ 输入通过以 5 为因子的分箱处理，从原始的 700 个输入减少而来。R带有轴突延迟的循环层，位于循环路径中。‡ 最后一个线性层的输出将直接使用，不经过 LIF 激活。

为剥离技术节点和器件规模差异对架构效率的影响，表 VII 和表 VIII 将表 VI 中的结果按逻辑资源、硅面积和能耗进行了归一化。在 FPGA 方面，并行配置在 SHD 数据集上实现了 112.1 每秒每千查找表推理次数，较准确率最接近的先前工作 [41] （68.5 每秒每千查找表推理次数）提升 64%；同时，该配置还具有更高的吞吐量（≈ 6,023 对比 5,586.59 每秒推理次数）、略高的准确率（92.46% 对比 92.30%）、少 34% 的查找表，且不使用数字信号处理模块。 [41]的低查找表配置达到名义上更高的 165.8 每秒每千查找表推理次数，但准确率仅为 88.78%（比本工作低 3.68 个百分点），因此不纳入比较（见表 VII）。在其余设计中，所提出的并行配置实现了最高的吞吐密度。能效方面，该配置达到 1.81×10^3 每焦耳推理次数，优于 [41]的两种配置。值得注意的是，所有提出的 FPGA 实现均完全不使用数字信号处理模块，从而将这些资源留作协处理之用；而时分复用的 SHD 配置占用的查找表少于 21,000 个，使其有望部署于中端器件。

Spiker+ 实现 [42] 在所比较的现场可编程门阵列设计中报告了最低的单次推理能耗（232.20 μJ/每秒推理次数），然而，如表 VI 所述，报告的 0.43 瓦最有可能仅计入了动态功耗，因为该数值与 XA7Z020 器件本身的静态功耗相当。此外，这种效率是以显著降低准确率（72.99%，相比之下为 92.46%）以及超过3× 更低的吞吐量（1,851.85，相比之下为 6,023 每秒推理次数）为代价实现的。

ASIC 综合使用开源软件 Open- ROAD [43]及其配套的 OpenROAD- Flow-Scripts (ORF) 执行，目标工艺为公开提供的 ASAP7 工艺设计套件 (PDK) [44]。针对三个数据集的加速器，均执行了完整的设计流程。为确保综合环境的一致性和可复现性，使用了由约翰内斯·开普勒大学集成电路与量子计算研究所托管的容器 [45]

表 VI 与其他硬件加速器架构的比较。

ASIC实现

Work	工艺	数据集	准确率 [%]	每秒推理次数	频率 [兆赫兹]	面积 [mm2]	#乘法次数	功耗 [瓦]
我们的方法（并行）	ASAP7	SHD	**92.46 ± 0.69**	**≈ 12027**	642	‡0.1073（0.49）	**0**	0.245
我们的方法（时分复用）*	ASAP7	SHD	**92.46 ± 0.69**	≈ 6768	**1103**	‡**0.0933（0.49）**	**0**	0.296
[37]	GF 22纳米	SHD	87	≈ 235	未报告	1.45	未报告	未报告
[38]	英特尔14纳米 (Loihi)	SHD	89.04	109.89	未报告	不适用	不适用	**0.004**
我们的方法（并行）	ASAP7	SSC	**82.08 ± 0.18**	**≈ 7032**	822	‡0.2033 (0.49)	**0**	0.506
我们的方法（时间复用）*	ASAP7	SSC	**82.08 ± 0.18**	≈ 2752	**1069**	‡**0.1789（0.49）**	**0**	**0.441**
我们的方法（并行）	ASAP7	PSMNIST	**94.86 ± 0.24**	**≈ 11937**	985	‡**0.0879（0.49）**	**0**	0.233
我们的方法（时间复用）*	ASAP7	PSMNIST	**94.86 ± 0.24**	≈ 6223	**1119**	‡0.0885 (0.49)	**0**	**0.222**

FPGA实现

Work	FPGA	数据集	准确率 [%]	每秒推理次数	频率 [兆赫兹]	#查找表	#DSP	功耗 [瓦]
我们的方法（并行）	ZCU102	SHD	92.46 ± 0.69	**≈ 6023**	328	53706	**0**	3.325
我们的方法（时间复用）*	ZCU102	SHD	92.46 ± 0.69	≈ 2056	**335**	**20488**	**0**	1.647
[39]	VCU128	SHD	**94.6**	未报告	未报告	†212900	未报告	¶10.24
[40]	Zynq UltraScale	SHD	87.8	未报告	未报告	230400	1728	3.349
[41]	ZCU104	SHD	92.30	5586.59	200	81567	318	3.94
[41]	ZCU104	SHD	88.78	5714.29	200	34474	106	3.79
[42]	XA7Z020	SHD	72.99	1851.85	100	未报告	未报告	§**0.43**
我们的方法（并行）	ZCU102	SSC	**82.08 ± 0.18**	**≈ 2644**	309	86636	**0**	6.013
我们的方法（时分复用）*	ZCU102	SSC	**82.08 ± 0.18**	≈ 857	**333**	**31639**	**0**	**2.538**
[39]	VCU128	SSC	65.6	未报告	未报告	†338700	未报告	¶10.24
我们的方法（并行）	ZCU102	PSMNIST	**94.86 ± 0.24**	**≈ 4049**	334	37571	**0**	2.936
我们的方法（时间复用）*	ZCU102	PSMNIST	**94.86 ± 0.24**	≈ 1930	**347**	**14150**	**0**	**1.646**

*使用动态神经元复用的时分复用（TM）。‡ 数值表示使用面积。括号中的数值表示包括未使用硅片在内的总体使用面积。† 原始报告中使用了 CLB（这会超出 FPGA 的资源容量）。¶ 由于使用了外部组件，功耗较高。§ 尽管未明确说明，这很可能仅指动态功耗，因为仅 FPGA 的静态功耗就与该数值相当。

。[2] 如表 VIII 所示，并行 ASIC 配置在 SHD 数据集上的能效达到 49.1×10^3 每焦耳推理次数，相当于 Loihi 的 $1.8\times$（ [38]），同时吞吐量高出$109\times$（≈ 12,027 对比 109.89 每秒推理次数）。尽管 Loihi 的绝对功耗较低（0.004 W，而并行配置为 0.245 W），但其单次推理能耗（36.40 μJ/推理）高于并行配置（20.37 μJ/推理）；时间复用变体则仍具有竞争力，能耗为43.74μJ/推理。时间复用配置在 SHD 数据集上的功耗较高（0.296 W，而并行配置为 0.245 W），这是由于其 $\approx 1.7\times$ 工作频率更高（1103 MHz，而并行配置为 642 MHz），在此情况下，动态功耗惩罚占主导。

在所有数据集上，时分复用配置的使用面积始终比并行变体最多仅高出13%（例如，在 SHD 数据集上为 0.0933 mm^2，而并行变体为 0.1073 mm^2 ），因为动态重载版本需要额外的 SRAM 来存储神经元膜电位。在 PSMNIST 数据集上，这甚至导致其占用面积略大（0.0885 mm^2 ，而并行变体为 0.0879 mm^2），进一步凸显了

[2]截至目前尚未完成流片。所有结果均仅来自综合和仿真。

时分复用版本相比 ASIC 更适用于现场可编程门阵列。所提出的 ASIC 设计在 PSMNIST 数据集上实现了最高达 135.8×10^3 每秒推理次数/mm^2 的吞吐密度，并且尽管采用了预测性 PDK，时钟频率仍最高可达 1.1 GHz。与其 FPGA 对应实现相比，ASIC实现将功耗降低了一个数量级以上（例如，在 SHD 数据集上为 0.245 W，而 FPGA 对应实现为 3.325 W），并将单次推理能耗降低了 27–37×，这表明完全开源的设计流程能够实现具有竞争力的脉冲神经网络加速器。

七、结论

本研究评估了三种用于数字神经形态硬件的脉冲序列表示方法，并证明，与绝对时间表示或 1 位等间隔采样脉冲序列相比，差分时间表示（DTR）可显著减少表示脉冲序列所需的总位宽。此外，我们还给出了合并独立 DTR 脉冲序列的数学证明，并提出了相应的低复杂度硬件实现。

基于这一表示，我们提出一组模块化硬件组件，可灵活组合成定制的脉冲神经网络

表 VII 归一化 FPGA 效率指标。

Work	数据集	每秒每千查找表推理次数	每焦耳推理次数 ($\cdot 10^3$)	μ焦耳/推理
我们的方法（并行）	SHD	**112.1**	**1.81**	**552.05**
我们的方法（时分复用）*	SHD	100.4	1.25	801.07
[41]	SHD（高精度）	68.5	1.42	705.26
[41]	SHD（低查找表）	†165.8	1.51	663.25
[42]	SHD	N/R	†4.31	†232.20
我们的方法（并行）	SSC	**30.5**	**0.44**	**2274.21**
我们的方法（时间复用）*	SSC	27.1	0.34	2961.49
我们的方法（并行）	PSMNIST	107.8	**1.38**	**725.12**
我们的方法（时间复用）*	PSMNIST	**136.4**	1.17	852.85

∗ 采用动态神经元复用的时分复用（TM）。N/R 原始研究中未报告。† 由于所引用研究的准确率显著较低，该结果不予考虑。

表 VIII 归一化 ASIC 效率指标。

Work	数据集	每秒推理次数/平方毫米² ($\cdot 10^3$)	每焦耳推理次数 ($\cdot 10^3$)	μ焦耳/推理次数 ($\cdot 10^3$)
我们的方法（并行）	SHD	**112.1**	**49.1**	**20.37**
我们的方法（时间复用）*	SHD	72.5	22.9	43.74
[38]	SHD	N/R	27.5	36.40
我们的方法（并行）	SSC	**34.6**	**13.9**	**71.96**
我们的方法（时间复用）*	SSC	15.4	6.2	160.25
我们的方法（并行）	PSMNIST	**135.8**	**51.2**	**19.52**
我们的方法（时间复用）*	PSMNIST	70.3	28.0	35.67

∗ 采用动态神经元复用的时分复用（TM）。N/R 表示原始研究中未报告。

加速器，支持具有学习型轴突延迟的循环和前馈网络，从而支持当前最先进的脉冲神经网络，并提供吞吐量优化和资源精简两种配置。定制的 Python 框架可根据 JSON 配置文件生成顶层 VHDL，从而简化针对新数据集构建加速器的过程。值得注意的是，生成的设计完全不涉及乘法运算，无需乘法器或数字信号处理（DSP）模块。

为支持可复现性和后续研究，完整的 VHDL 源代码已发布在公开的 GitHub 代码库中。针对 Xilinx ZCU102 FPGA 的综合结果表明，在 SHD 数据集上，吞吐量最高可达 $\approx 6,023$ Inf./s，吞吐密度为 112.1 Inf./s/kLUT，超过了准确率相当的既有加速器。使用开源 OpenROAD流程和预测型 ASAP7工艺设计套件综合的专用集成电路（ASIC）实现，吞吐量最高可达 $\approx 12,027$ Inf./s，时钟频率超过 1.1GHz，并且能效超过 Intel的Loihi [38]，高出 $1.8\times$（49.1×10^3 对比 27.5×10^3 Inf./J）。

附录

A. 问题陈述

目标是将 N 独立的时间间隔序列 $\mathcal{S}_m = \{\Delta S_{m,i}\}_{i=1}^{\infty}$ 合并为一个序列 ΔY_k，表示所有源点过程有序并集的事件间隔。

B. 文献分析

该算法将点过程叠加的递归实现形式化。

•**随机建模：** 其基础是泊松过程叠加。Kingman [28] 确立了这样一个结论：独立到达流的合并会形成一个点过程，其中到达间隔时间由当前残差中的最小值决定。

• **数字取证：** 当绝对时钟不可靠时，取证依赖于统一相对时间框架。Stevens[29] 和Dreier等人 [30] 强调整合隐式计时信息，以便在没有公共时钟的情况下重建连贯的时间轴。

定理。 合并后的事件间持续时间序列 ΔY_k可以通过对残差向量进行归纳递归唯一生成，其中残差向量 $\mathbf{R}_k = (R_{1,\,k},\ \ldots,\ R_{n,\,k})$表示每个活动流距离下一个事件的时间。

定理 A.1 （N流差分合并的正确性）。设 $\mathcal{S}_m = \{\Delta S_{m,i}\}_{i\geq 1}$ 为 N 个非负到达间隔时间序列（每个流 $m = 1, \ldots, N$ 各有一个）。定义每个流的绝对事件时间如下：

$$a_{m,1} := \Delta S_{m,1}, \quad a_{m,i+1} := a_{m,i} + \Delta S_{m,i+1} \quad \textit{for } i \geq 1,$$

并且，如果流 m 是有限的，则设定 $a_{m,\,i} := +\infty$ 对于其最后一个事件之后的所有 i 。

令 $\{y_k\}_{k\geq 1}$ 为将多重集 multiset $\biguplus_{m=1}^{N}\{a_{m,i}\}_{i\geq 1}$ 按重数排序后得到的非递减序列，并设定 $y_0 := 0$。考虑残余时间向量 $\mathbf{R}_k = (R_{1,\,k}, \ldots, R_{N,\,k})$ 以及按如下方式生成的差分输出 ΔY_k：

$$R_{m,1} := \Delta S_{m,1} = a_{m,1} - y_0, \qquad m = 1, \ldots, N, \quad (12)$$

$$\Delta Y_k := \min_{1\leq m\leq N} R_{m,k}, \qquad k \geq 1, \quad (13)$$

$$\mathcal{F}_k := \{m : R_{m,k} = \Delta Y_k\} \quad \textit{(firing set)}, \quad (14)$$

$$R_{j,k+1} := R_{j,k} - \Delta Y_k, \qquad j \notin \mathcal{F}_k, \quad (15)$$

$$R_{m,k+1} := \Delta S_{m,\mathrm{next}(m,k)}, \qquad m \in \mathcal{F}_k, \quad (16)$$

其中，$\mathrm{next}(m, k)$ 表示在第 k 步为流 m 触发的事件的（按流计）索引（如果该流已耗尽，则将其设定为使 $R_{m,\,k+1} = +\infty$）。

Then, for all $k \geq 1$, $\Delta Y_k := y_k - y_{k-1}$.

特别地，当且仅当 $y_k = y_{k-1}$ 时，$\Delta Y_k = 0$（流之间或同一流内发生同时事件）。

证明。 我们通过对 $k \geq 1$ 进行归纳法证明：

$$R_{m,k} := a_{m,\mathrm{next}(m,k)} - y_{k-1} \quad \text{for all } m = 1, \ldots, N, \quad (17)$$

并且，输出规则 (13) 会产生 $\Delta Y_k = y_k - y_{k-1}$。

我们首先明确 next(m, k)。对于每个流 m 和步$k \geq 1$，令 next(m, k) 为满足 i 使得$a_{m,\,i} \geq y_{k-1}$的最小索引，并约定：如果 $a_{m,\,i} = y_{k-1}$ 且流 m 在步 $k-1$触发，则 next(m, k) $= i + 1$（或已耗尽时

$+\infty$）。在操作上，更新规则（16）恰好实现了这一推进。

基本情况（$k=1$）。根据（12），$R_{m,1}=a_{m,1}-y_0$，因此（17）对 $k=1$ 成立，其中 next(m,1) $=1$。输出规则（13）给出

$$\begin{aligned}\Delta Y_1 &:= \min_m R_{m,1}\\ &= \min_m (a_{m,1}-y_0)\\ &= \big(\min_m a_{m,1}\big)-y_0\\ &= y_1-y_0.\end{aligned}$$

因此，该命题对 $k=1$成立。

归纳步骤。固定 $k\geq 1$ 并假设（17）在第 k步成立，即，

$$R_{m,k} := a_{m,\mathrm{next}(m,k)}-y_{k-1} \quad \text{for all } m.$$

根据输出规则（13）和不变量，

$$\begin{aligned}\Delta Y_k &= \min_m R_{m,k}\\ &= \min_m \big(a_{m,\mathrm{next}(m,k)}-y_{k-1}\big)\\ &= \big(\min_m a_{m,\mathrm{next}(m,k)}\big)-y_{k-1}\\ &= y_k-y_{k-1},\end{aligned}$$

其中，按定义，y_k 是多重集$\biguplus_m\{a_{m,i}\}$的非递减合并中的下一个元素。

还需证明式 (17) 在第 $k+1$ 步成立。

未触发流。对于任意 $j\notin\mathcal{F}_k$，根据式 (15) 和归纳假设，

$$\begin{aligned}R_{j,k+1} &= R_{j,k}-\Delta Y_k\\ &= \big(a_{j,\mathrm{next}(j,k)}-y_{k-1}\big)-(y_k-y_{k-1})\\ &= a_{j,\mathrm{next}(j,k)}-y_k.\end{aligned}$$

由于 j 在第 k 步未触发，其 next 事件索引保持不变，即 next(j, $k+1$) =next(j, k)，因此

$$R_{j,k+1}=a_{j,\mathrm{next}(j,k+1)}-y_k,$$

这就是流 j 在第 $k+1$ 步的式 (17)。

触发流。对于任意 $m\in\mathcal{F}_k$，有 $R_{m,\,k}=\Delta Y_k$，因此

$$\begin{aligned}a_{m,\mathrm{next}(m,k)}-y_{k-1} &= \Delta Y_k\\ &= y_k-y_{k-1} \quad\Longrightarrow\quad a_{m,\mathrm{next}(m,k)}\\ &= y_k.\end{aligned}$$

因此，由 next(m, k) 索引的事件恰好发生在 y_k。将每个流的索引推进到下一个事件（若已耗尽则推进到 $+\infty$），我们有

$$\mathrm{next}(m,k+1)=\mathrm{next}(m,k)+1,$$

而更新式（16）给出

$$\begin{aligned}R_{m,k+1} &= \Delta S_{m,\mathrm{next}(m,k)}\\ &= a_{m,\mathrm{next}(m,k)+1}-a_{m,\mathrm{next}(m,k)}\\ &= a_{m,\mathrm{next}(m,k+1)}-y_k,\end{aligned}$$

这就是流 m 在第 $k+1$ 步时的 (17)。

综上，这些情况表明 (17) 从 k 到 $k+1$ 始终成立，从而完成归纳法。

同时事件。如果多个流（或某个流中的多个事件）具有 $a_{m,\,i}=y_k$，那么所有这些流都处于$\mathcal{F}_k$。式（16）中的更新会使它们各自越过时间 y_k。如果在 y_k 仍有后续事件发生，那么下一步会得到 $\Delta Y_{k+1}=0$，依此类推，产生的零输出数量与 y_k 处的重数相同。因此，该算法通过差值 ΔY_k 输出绝对时间的非递减合并序列。

耗尽流。如果某个流是有限的，将 $a_{m,i}=+\infty$设为超过其最后一个事件，意味着该流耗尽后 $R_{m,k}=+\infty$，因此永远不会成为最小值。等价地，实现也可以将耗尽流从优先队列中移除；这两种处理方式都与上述代数关系一致。

因此，对所有 $k\geq 1$、$\Delta Y_k=y_k-y_{k-1}$，结论如上。 □

致谢

本研究获得了奥地利研究促进机构（FFG）和奥地利气候行动、环境、能源、交通、创新与技术部（BMK）的资助，项目编号为 FO999899263；同时还获得了芯片联合事业的资助，项目编号为 101112274（ISOLDE）。该事业及其成员包括奥地利、捷克、法国、德国、意大利、罗马尼亚、西班牙、瑞典和瑞士。

此外，这项工作还得到了林茨机电一体化中心（LCM）下属 COMET-K2“共生机电一体化中心”的支持，该中心由奥地利联邦政府和上奥地利州政府资助。

这项工作还得到了 Silicon Austria Labs（SAL）的支持。该机构由奥地利共和国、施蒂里亚州商业促进机构（SFG）、克恩顿州、上奥地利研究机构（UAR）和奥地利电气与电子工业协会（FEEI）所有。

参考文献

[1]Z. Yan、Z. Bai 和 W.-F. Wong，《重新审视脉冲神经网络的能效》，2026年。 [在线]。获取地址：https://arxiv.org/abs/2409.08290[2] B. A. Moser 和 M. Lunglmayr，《基于漏积分发放的模拟信号到脉冲转换的采样稀疏性》，神经形态计算与工程，第 5 卷，第 1 期，页码 014011，2025年3月。[在线]。获取地址：https://dx.doi.org/10.1088/2634-4386/adb884[3] M. Dampfhoffer、T. Mesquida、A. Valentian 和 L. Anghel，《脉冲神经网络的能效真的高于人工神经网络吗？一项深入的硬件感知研究》，《*IEEE* 计算智能新兴主题汇刊》。[4] M. Lunglmayr、G. Lindorfer 和 B. Moser，《用于时间序列分析的稳健高效仿生数据采样原型》，载于*DEXA 2021*。Springer International Publishing，2021年，页码 119–126。[5] B. Cramer、Y. Stradmann、J. Schemmel 和 F. Zenke，《用于系统评估脉冲神经网络的海德堡脉冲数据集》，*IEEE* 神经网络与学习系统汇刊，第 33 卷，第 7 期，页码 2744–2757，2022。[6] M. T. L. Aung、D. Gerlinghoff、C. Qu、L. Yang、T. Huang、R. S. M. Goh、T. Luo 和 W.-F. Wong，《DeepFire2：一种基于现场可编程门阵列的卷积脉冲神经网络加速器》，《*IEEE* 计算机汇刊》，第 72 卷，第 10 期，页码 2847–2857，2023年10月。 [在线]。获取地址：http://dx.doi.org/10.1109/TC.2023.3272284

[7] H. Liu, Y. Qu, “一种基于现场可编程门阵列的低功耗、低延迟脉冲神经网络加速器”，载于*2023*年 国际神经网络联合会议（*IJCNN*），2023年，第 1–8 页。

[8] J. Li, G. Shen, D. Zhao, Q. Zhang, 和 Y. Zeng, “FireFly：一种高 - 采用高效数字信号处理和内存优化的脉冲神经网络吞吐量硬件加速器，” *IEEE* 超大规模集成电路（*VLSI*）系统学报，第 31 卷，第 8 期，第 1178–1191 页，2023年。

[9] S. Panchapakesan、Z. Fang 和 J. Li，“SyncNN：在现场可编程门阵列上评估和加速脉冲神经网络”，载于 *2021* 年第*31*届现场可编程逻辑与应用国际会议（*FPL*），2021年，第 286–293 页。

[10] A. Khodamoradi, K. Denolf, 和 R. Kastner, “S2N2：用于深度脉冲神经网络的 FPGA 加速器”，载于 *2021* 年 *ACM/SIGDA* 国际现场可编程门阵列研讨会，FPGA ’21 系列。美国纽约州纽约市：计算机协会，2021年，第 194–205 页。 [在线]。可访问：https://doi.org/10.1145/3431920.3439283

[11] Y. Ortiz, E. Stromatias, M. Marinescu, G. Indiveri, 和 S. Furber, “利用降低定制浮点和对数占用位在现场可编程门阵列上加速逐脉冲神经网络”，*IEEE Access*，第9卷，第80 603–80620页，2021年。

[12] Y. Chen, W. Ye, Y. Liu, 和H. Zhou, “SiBrain：一种稀疏时空 - 并行神经形态架构，可低延迟加速脉冲卷积神经网络，”*IEEE*电路与系统学报*I*：常规论文，第71卷，第12期，第6482–6494页，2024年。

[13] J. Li 和 Y. Shu, “基于非结构化剪枝脉冲卷积神经网络的高效硬件加速器”，载于*2025*年 *IEEE* 第七届人工智能电路与系统国际会议（*AICAS*），2025年，第 1–5 页。

[14] B. Xu, J. Hwang, P. Vanna-iampikul, S. K. Lim, and P. Li, “脉冲 - 式 Transformer 在三维集成中的硬件加速器”，载于 *2024*年 *ACM/IEEE* 国际计算机辅助设计会议（*IC-CAD*），2024年，第 1–9 页。

[15] H. Nguyen-Dinh、L. Dang-Hoang、H. Tran-Quang、L. Nguyen-Phuong、C. Hoang-Phuong 和 M. Nguyen-Duc，“一种采用优化路由器的新型基于 NoC 的脉冲神经网络架构”，载于*2024*年第十届国际通信与电子会议（*ICCE*），2024年，第 119–124 页。[16] A. Nimbekar、P. Katti、C. Li、B. M. Al-Hashimi、A. Acharyya 和 B. Rajendran，“软硬件协同优化的快速、准确深度可重构脉冲推理加速器架构设计方法”，载于*2024*年第*37*届 *IEEE* 国际片上系统会议（*SOCC*），2024年，第 1–6 页。[17] M. Horowitz，“1.1 计算能耗问题（以及我们能采取的措施）”，载于*2014*年 *IEEE* 国际固态电路会议技术论文集（*ISSCC*），2014年，第 10–14 页。[18] A. D. Mauro、F. Conti、P. D. Schiavone、D. Rossi 和 L. Benini，“突破微功耗机器学习应用中片上存储器的可靠性边界”，载于*2019*年 *IEEE* 国际电子器件会议（*IEDM*），2019年，第 30.4.1–30.4.4 页。[19] J. K. Eshraghian、M. Ward、E. Neftci、X. Wang、G. Lenz、G. Dwivedi、M. Bennamoun、D. S. Jeong 和 W. D. Lu，“借鉴深度学习经验训练脉冲神经网络”，《IEEE 汇刊》，第 111 卷，第 9 期，第 1016–1054 页，2023年。[20] Q. Liu、G. Pan、H. Ruan、D. Xing、Q. Xu 和 H. Tang，“基于多尺度时空特征和脉冲神经元的无监督 AER 目标识别”，IEEE 神经网络与学习系统汇刊，第 31 卷，第 12 期，第 5300–5311 页，2020。[21] T. Mesquida、A. Valentian、D. Bol 和 E. Beigne，“AER 引起的时序失真对实现数字信号处理的脉冲神经网络的影响”，载于*2016*年第*12*届微电子与电子学博士研究会议（*PRIME*），2016年，第 1–4 页。[22] J. Kim、J. Park、S. Joo 和 S.-O. Jung，“面向基于 AER 的大规模脉冲神经网络神经形态系统的脉冲时序依赖可塑性高效硬件实现”，载于*2020*年第*35*届国际电路／系统、计算机与通信技术会议（*ITC-CSCC*），2020年，第 1–4 页。

[23] M. Thalakkottur, S. Chakravarthy, “Sens：通过事件时间裕度传播实现神经形态计算》，2023年。 [在线]。可访问：https://arxiv.org/abs/2304.13918

[24] H. Zheng, Y. Hou, Y. Li, Y. Zhang, 和 Z. Yu, “采用自适应相位编码与直接训练脉冲神经网络”，载于 *ICASSP 2026 - 2026 IEEE*国际声学、语音与信号处理会议（*ICASSP*），2026年，第4831–4835页。

[25] Y. Kim、H. Park、A. Moitra、A. Bhattacharjee、Y. Venkatesha 和 P. Panda，“频率编码还是直接编码：哪一种更适用于准确、鲁棒且节能的脉冲神经网络？”载于 *ICASSP 2022 - 2022 IEEE*国际声学、语音与信号处理会议（*ICASSP*），2022年，第71–75页。

[26] Q. Liu, H. Ruan, D. Xing, H. Tang, 和 G. Pan, “使用分段概率最大化学习脉冲神经网络中的地址事件表示目标进行有效分类”，2020年。 [在线]。网址：https://arxiv.org/abs/2002.06199

[27] Q. Liu, G. Pan, 和 H. Tang, “基于多尺度时空特征和脉冲神经元的无监督空中目标识别”，*IEEE Transactions onNeural Networks and Learning Systems*，第31卷，第12期，第5300–5311页，2020年。

[28] J. F. C. Kingman，泊松过程，ser .《牛津概率研究》. 纽约：克拉伦登出版社、牛津大学出版社，1993年，第3卷，牛津科学出版物。

[29] M. Stevens，“数字取证的相对时间框架的统一”，《数字调查》，第1卷，第225–239页，2004年9月。

[30] L. M. Dreier, C. Vanini, C. J. Hargreaves, F. Breitinger, and F. Freiling, “超越时间戳：将隐式时间信息整合到数字取证时间线中”，《国际法证科学：数字调查》，第49卷，第301755页，2024年，dFRWS USA 2024 - 第24届美国年度数字取证研究会议精选论文集。 [在线]。可访问：https://www.sciencedirect.com/science/article/pii/S266628172400074X

[31] K. He, X. Zhang, S. Ren, 和 J. Sun, “用于图像识别的深度残差学习”，2015 年。 [在线]。可访问：https://arxiv.org/abs/1512.03385

[32] ARM，《AMBA 4 AXI4-Stream 协议规范》，ARM Limited，2021年。 [在线]。可访问：https://developer.arm.com/documentation/ ihi0051/b/[33] E. Izhikevich，“脉冲神经元的简单模型”，《IEEE 神经网络汇刊》，第14卷第6期，第1569–1572页，2003年。[34] J.-J. Lee、W. Zhang、Y. Xie和P. Li，“Saarsp：用于循环脉冲神经网络脉动阵列加速的架构”，《J. Emerg. Technol. Comput. Syst.》，第18卷第4期，2022年10月。 [在线]。可访问：https://doi.org/10.1145/3510854[35] A. Queant、U. Ranc¸on、B. R. Cottereau和T. Masquelier，“DelRec：在循环脉冲神经网络中学习延迟”，2025年。 [在线]。可访问：https://arxiv.org/abs/2509.24852[36] T. Hirtzlin、B. Penkovsky、M. Bocquet、J.-O. Klein、J.-M. Portal和D. Querlioz，“用于二值化神经网络硬件实现的随机计算”，《IEEE Access》，第7卷，76 403，2019年76394pp. – 。[37] R. Meijer、P. Detterer、A. Yousefzadeh、A. Pati˜no-Saucedo、G. Tang、K. Vadivel、Y. Xu、M. Gomony、F. Corradi、B. Linares-Barranco和M. Sifalakis，“数字事件驱动人工智能加速器中的高效突触延迟实现”，arXiv.org，工作论文，2025年1月。[38] A. Pati˜no-Saucedo、R. Meijer、P. Detteter、A. Yousefzadeh、L. Garrido- Regife、B. Linares-Barranco和M. Sifalakis，“针对数字神经形态加速器的带突触延迟模型协同优化训练”，载于《2024 IEEE 国际电路与系统研讨会（ISCAS）》，2024年，第1–5页。[39] Y. Zhao、Y. Bouhadjar和E. Neftci，“集成轴突延迟的脉冲神经网络高效FPGA实现”，载于《2025 IEEE 生物医学电路与系统会议（BioCAS）》，2025年，第562–566页。

[40] S. Matinizadeh、N. Pacik-Nelson、I. Polykretis、K. Tishbi、S. Kumar、M. L. Varshika、A. Mohammadhassani、A. Mishra、N. Kandasamy、J. Shackleford、E. Gallo和A. Das，“完全可配置的开源软件定义数字量化脉冲神经核心架构”，2024年。 [在线]。可访问：https://arxiv.org/abs/2404.02248[41] H. Nakano、K. Blachut、K. Jeziorek、P. Wzorek、M. Dampfhoffer、T. Mesquida、H. Nishi、T. Kryjak和T. Dalgaty，“面向片上系统 FPGA 上低延迟时间序列分类的硬件加速事件图神经网络”，2025年。 [在线]。可访问：https://arxiv.org/abs/2503.06629[42] A. Carpegna、A. Savino和S. D. Carlo，“Spiker+：用于生成高效脉冲神经网络 FPGA 加速器、以实现边缘端推理的框架”，2024年。 [在线]。可访问：https://arxiv.org/abs/2401.01141

[43] T. Ajayi、V. A. Chhabria、M. Fogaça、S. Hashemi、A. Hosny、A. B. Kahng、M. Kim、J. Lee、U. Mallappa、M. Neseem等，“迈向开源数字流程：来自 OpenROAD 项目的初步经验”，载于第 *56* 届 *2019* 年，2019 年第 1 至 4届年度设计自动化会议，第 – 页。[44] L. T. Clark、V. Vashishtha、L. Shifren、A. Gujja、S. Sinha、B. Cline、C. Ramamurthy 和 G. Yeric，“ASAP7：7 纳米 finFET 预测性工艺设计套件”，《微电子学杂志》，第 53 卷，第 105–115 页，2016年。 [在线]。网址：https://www.sciencedirect.com/science/article/pii/S002626921630026X[45]H. Pretl 和 G. Zachl，“IIC-OSIC-TOOLS 的 GitHub 仓库”，2023 年 9 月。 [在线]。网址：https://github.com/iic-jku/ IIC-OSIC-TOOLS