项目说明simple-version

项目借鉴 TSMC 在 2023 年 IEEE 国际固态电路会议（ISSCC 2023）中提出的基于 SRAM 的数字存算一体计算宏（DCIM macro）思想，需要实现一种支持可变位宽与并行操作的高效计算架构，可处理 12/24b 整数权重与 12/24b 整数输入。

电路架构

- 核心组成：电路由 SRAM 存储阵列、局部乘累加电路（LMAC）、全局 IO（Global IO）、读字线驱动（RWLDRV）和全局控制器（GCTRL）构成。其中，SRAM 阵列预加载权重；LMAC 包含按位乘法器与加法树，完成 1b 输入与 12b 权重的乘累加；Global IO 集成移位寄存器、27b 加法器与 51b 移位累加器，处理多周期结果累加；GCTRL 通过 WWIDTH 和 INWIDTH 信号配置权重与输入位宽。
- 处理流程：1）权重经 WBL 传输至 LMAC，与串行输入的 XIN 通过 OAI 执行乘法；2）加法树累加 8 个 1b×12b 乘积结果；3）Global IO 对多周期结果移位累加，最终得到 8×12/24b 输入与 8×12/24b 权重的乘累加结果；4）16 个相同模块并行工作，实现 16×8×12/24b 的全局计算。
- 位宽配置：支持 12/24b 权重（WWIDTH 信号控制）与 12/24b 输入（INWIDTH 信号控制），通过符号扩展机制兼容有符号运算：前 4 个周期执行 4b 有符号操作，其余周期为 4b 无符号操作，确保位宽切换时的精度一致性。
- RTL 设计：基于 Verilog 完成模块化设计，顶层模块（top）包含存算阵列（cim_array）、数字逻辑（digital_circuit）等子模块，其中 cim_array 负责存储与基础运算，global_io 处理累加与输出，rwldrv 实现输入信号串并转换，gctrl 生成控制时序与状态信号。
- 同时 MAC 与权重更新：采用 ping-pong 结构，阵列设计为2rows，通过行选择逻辑实现并行的权重写入与 MAC 操作（一个行用于写操作时，另一个行可同时进行计算）。

RTL架构设计

- 顶层结构：以 `top.v` 为核心，包含存算阵列（`cim_array.v`）、数字逻辑（`digital_circuit.v`）等子模块，层级关系如下：
- plaintext

代码块

```
top.v
├── cim_array.v（含cim_bank.v）
└── digital_circuit.v
```

```
    ├── cim_array_ctrl.v（阵列控制）
    ├── global_io.v（全局IO，含累加器）
    │   ├── add.v
    │   └── accumulator.v（含se_cla.v、s_cla.v）
    ├── local_mac.v（局部MAC，含oai_mult.v）
    ├── rwldrv.v（读字线驱动）
    └── gctrl.v（全局控制）
```

- 关键模块接口
  - 顶层模块（top.v）：定义输入输出信号，包括数据（D）、时钟（clk）、复位（rstn）、地址（WA）、位宽配置（inwidth/wwidth）、输入激活（xin0）、输出（nout）等：
  - verilog

代码块

```
module top(input [23:0] D,input clk, rstn, cima, acm_en,input [7:0] WA,input
 inwidth, wwidth, start,input [191:0] xin0,output [50:0] nout,output wire st
);
```

  - 全局 IO（global_io.v）：处理局部 MAC 输出的累加，接口如下：
  - verilog

代码块

```
module global_io(input [14:0] macout_a, macout_b,input clk, acm_en, rstn,
st, wwidth,output [50:0] nout
);
```

  - 局部 MAC（local_mac.v）：实现按位乘法与加法树累加，接收权重（wb0/wb1）与行选择信号（rwlb_row0/rwlb_row1），输出乘累加结果：
  - verilog

代码块

```
module local_mac(input [95:0] wb0, wb1,input [7:0] rwlb_row1,
rwlb_row0,input sus,output wire [14:0] mac_out
);
```

详细模块说明

top.v

- 功能: 系统的顶层模块，连接并协调所有子模块以完成 CIM 操作。
- 端口:
  - D[23:0]: 写入存储器的数据。
  - WA[7:0]: 存储器写地址。
  - xin0[95:0]: 输入的计算数据向量。
  - clk, rstn: 时钟和复位信号。
  - cima, acm_en, inwidth, wwidth: 控制信号。
  - nout[50:0]: 最终计算结果。
  - st: 累加器状态信号。
- 内部逻辑: 实例化所有主要的硬件模块，并负责它们之间的顶层连线。

---

cim_array_ctrl.v

- 功能: CIM 存储阵列的控制器。
- 端口:
  - D, WA: 输入的数据和地址。
  - cima: 片选信号，用于选择两个 cim_bank 中的一个。
  - D1, WA0, WA1: 输出给 cim_array 的数据和经过选择的地址。
- 内部逻辑: 根据 cima 信号，将输入的 WA 路由到 WA0 或 WA1，同时将 D 传递给 D1。

---

cim_array.v

- 功能: CIM 存储阵列，由两个 cim_bank 组成。

- 内部逻辑: 简单地实例化两个 cim_bank，分别代表系统中的两个主要存储区域。

---

cim_bank.v

- 功能: 8x24-bit 的 SRAM 存储体。
- 内部逻辑:
  - 包含一个 8x24-bit 的寄存器数组 mem。
  - 通过 case 语句响应 WA (one-hot 编码) 来写入数据 D。
  - 组合逻辑，将 mem 中的数据整理后，通过 WB_a 和 WB_b 端口并行读出。

---

gctrl.v (Global Controller)

- 功能: 全局控制器，用于产生计算的步进控制信号。
- 端口:
  - inwidth: 输入位宽选择 (12-bit 或 24-bit)。
  - sel[5:0]: 输出的选择信号，用于驱动 rwldrv。
  - st: 累加器启停信号。
- 内部逻辑: 一个简单的计数器。根据 inwidth 决定计数上限（11 或 23）。在计数周期内，st 为低，sel

递增；计数完成后，st 拉高，sel 复位。

---

rwldrv.v (Row/Word Line Driver)

- 功能: 根据 sel 信号选择 xin 的特定位，并生成 CIM 计算所需的行驱动信号。
- 内部逻辑:

- 首先将 96-bit 的 xin 分解为 12 组 8-bit 的 xin_w。
- 使用 case 语句，根据 sel 的值，从 xin_w 中选择一组，并将其（取反后）输出到 rwlb_row1 或 rwlb_row0。cima

信号决定输出到哪一个。

---

local_mac.v

- 功能: 本地乘累加单元，是存内计算的核心。
- 内部逻辑:
  - 乘法: 实例化 8 个 oai_mult 模块，并行执行 8 次 12-bit 的位乘法操作。
  - 加法: 使用一个三级加法器树（由 add 模块构成），将 8 个乘法结果高效地相加，最终输出一个 15-bit 的 mac_out。

---

oai_mult.v

- 功能: 一个定制的 12-bit OAI (OR-AND-Invert) 乘法器。
- 逻辑: 实现 e = ~((a | c) & (b | d))。这里的 a, b 来自存储器，c, d 来自输入，e

是位乘法结果。这种设计是为了匹配 SRAM 单元的物理特性以高效实现计算。

---

global_io.v

- 功能: 全局输入输出模块，负责处理来自两个 local_mac 的结果并驱动最终的累加器。
- 内部逻辑:
  - 接收 macout_a 和 macout_b。
  - 对 macout_b 进行移位和逻辑运算。

- 使用一个 27-bit 的 add 模块将 macout_a 和处理后的 macout_b 相加。
- 将加法结果送入 accumulator。

---

accumulator.v

- 功能: 51-bit 累加器。
- 内部逻辑:
  - 在时钟驱动下，当 st 为低时，将 se_cla 加法器的输出 nout 左移一位并寄存到 nout_1，用于下一次累加。当 st

  为高时，清零累加结果。
  - 实例化 se_cla 模块执行核心的加法操作。

---

se_cla.v (Sign-Extend Carry-Lookahead Adder)

- 功能: 一个特殊的加法器，用于将一个 27-bit 数和一个 51-bit 数相加。
- 内部逻辑:
  - 对 27-bit 输入 a 进行符号位扩展。
  - 使用一个 add 模块计算低 27 位，并产生进位。
  - 使用一个 s_cla 模块计算高 24 位，并考虑来自低位的进位。
  - 拼接高位和低位结果，得到 51-bit 的和。

---

s_cla.v (Signed Carry-Lookahead Adder)

- 功能: 24-bit 有符号先行进位加法器。
- 内部逻辑: 实现了标准的 CLA 逻辑，包括生成 G (Generate) 和 P (Propagate)

信号，并以此快速计算所有位的进位，最终得到和。

---

add.v

- 功能: 参数化的通用加法器。
- 参数: width - 定义了加法器的位宽。
- 端口: sus - 控制执行有符号加法 (sus=1) 或无符号加法 (sus=0)。
- 内部逻辑: 根据 sus 的值，对输入进行相应的符号位扩展（或补零），然后执行加法。

处理流程实现

1. 权重加载: 外部通过 D 和 WA 端口将权重数据写入到 cim_array 中的 cim_bank。cim_array_ctrl负责将地址和数据路由到正确的 cim_bank。
2. 输入加载: 外部输入向量通过 xin0 端口送入系统。
3. 计算开始:
    - gctrl 模块开始计数，生成 sel 信号。
    - rwldrv 根据 sel 信号，从 xin0 中选择一小部分位，生成行驱动信号 rwlb_row0 和 rwlb_row1。
    - cim_array 中的 cim_bank 被 rwlb 信号激活，输出存储的权重位到 wb 总线。
    - local_mac 模块接收来自 cim_array 的权重位 (wb*) 和来自 rwldrv 的输入位 (rwlb_*)，通过内部的 oai_mult

    阵列和加法器树，计算出一个部分的乘累加结果 mac_out。
4. 结果处理:
    - global_io 模块接收两个 local_mac 的输出 macout_a 和 macout_b，进行一次加法/减法操作。
    - accumulator 模块将 global_io 的输出进行累加。gctrl 产生的 st 信号控制累加的开始和结束。
5. 输出: 最终的累加结果通过 nout 端口输出。这个过程会根据 inwidth 的设置，循环执行 12 或 24 次，以完成一次完整的向量-矩阵乘法。