加速器配置场景分析

付杰

2023 年 12 月 7 日

目录

1 ACC 配置信息写入时的预先设定

1. L1C 作为主处理器负责计算需要下发给加速器的配置信息
2. L1C 计算出的配置信息会被暂存入 PPM(Ping Pong Memory) 中
3. PPM 内部的配置信息按照 (地址，信息) 的格式进行存储
4. PPM 内部的配置信息需要具体由 DMA 或者 MCU 写入到 ACC 配置寄存器中
5. ACC 内部的配置寄存器，其地址无法保证连续

2 每个 ACC 都有一套 PPM，通过 DMA 进行配置

- 工作流：

  1. 在这种场景下，L1C 计算出每个 ACC 的配置信息，这些配置信息会被写入到每个 ACC 的 PPM 中。L1C 跟 ACC 的 AXI Bus 采用一主多从的拓扑结构。
  2. 每个 ACC 内部都有自己的 DMA 用于从 PPM 搬运配置信息到 ACC 内部的寄存器，DMA 每次搬运数据都会搬运 (地址，信息) 这样的而元组数据。
  3. DMA 搬运配置信息并且更新 ACC 配置寄存器的启动，需要由一个信号来控制，这个信号跟 ACC 以及 PPM 有关。
  4. ACC 内部的配置寄存器组会根据地址来译码选中某一个寄存器，写入的使能信号由 DMA 给出，将信息写入到该配置寄存器中。
  5. DMA 搬运的过程中，每个 Cycle 有 1 个 ACC 配置寄存器可以被更新。
  6. DMA、ACC 以及 PPM 读操作，属于同一个时钟域；L1C 跟 PPM 写属于同一个时钟域。

- 该设计优点：实现上比较简单

- 存在的问题：

  Ppm to share memory or config memory

  1. ACC 的 PPM 可能会 busy(PPM 里的配置信息没有及时被写入到 ACC)，此时 L1C 写入到 PPM 的动作会被阻塞。
  2. PPM 利用率不高，这种方式每个 ACC 的 PPM 是独享的，某些 ACC 的 PPM 可能写不满、另一些 ACC 的 PPM 可能不够用，导致了 PPM 空间的浪费。
  3. L1C 需要针对每一个 ACC 的 PPM 都有一个 AXI Bus 连线，需要更复杂的连线以及仲裁结构 (L1C 不仅需要跟 ACC 交互，还需要跟其他模块交互)。

- **适合的工作场景**

1. 对于灵活性要求不高
2. 允许 L1C 因为配置 PPM 而阻塞

系统架构图如图1所示。

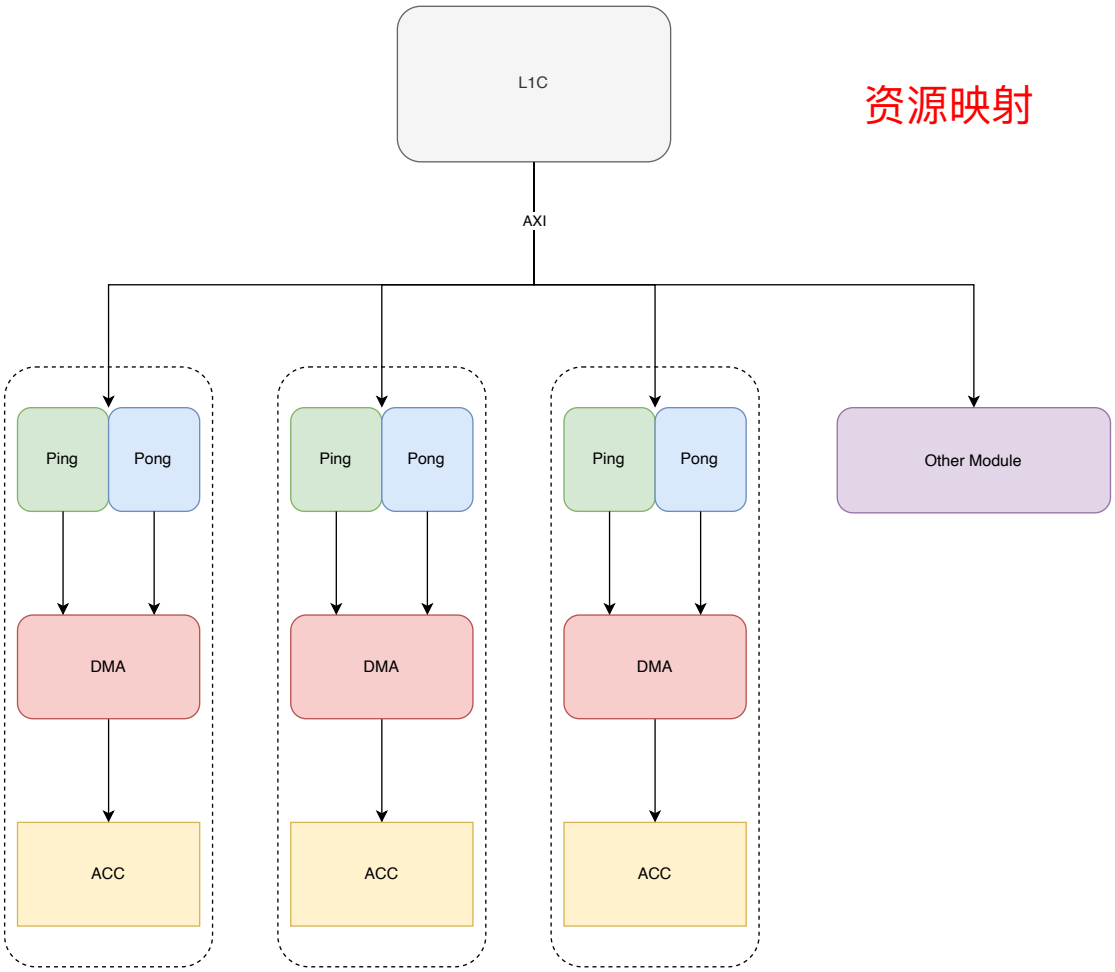

图 1: 每个 ACC 都有一套 PPM，通过各自的 DMA 进行配置

3 所有的 ACC 共享同一块 PPM，通过 DMA 搬运

- 工作流：
  1. 所有的 ACC 不再有自己的 PPM，而是共享一块更大的 PPM，该 PPM 可以容纳 2N 组配置寄存器信息 (N 为加速器个数)。
  2. 为了支持多个 ACC 同时从 PPM 读取配置信息，PPM 实际上有多 bank 构成，进而支持同时读取。
  3. PPM 只要有空间，L1C 就可以把配置信息写入，此时需要维护一些信息:
     (a) 这笔配置信息是针对哪个 ACC 的配置信息。
     (b) 修改该 ACC 对应的配置信息有效队列。
  4. 每个 ACC 通过其自己的 DMA 从统一的 PPM 搬运配置信息到 ACC 中。
- 该设计优点：
  1. 增加了 PPM 的利用率以及灵活性。
  2. **简化了 L1C 写入到 PPM 的过程**，L1C 不用跟每个 ACC 都有 AXI 总线连接，避免了总线 busy 的情况。
- 存在的问题：
  1. PPM 被 L1C 写入的时候，需要维护额外的与 ACC 的映射关系。
  2. PPM 需要支持多端口读，且 PPM 跟每个 ACC 都需要通过 AXI 总线连接。
- **使用的场景**：
  1. 加速器内部寄存器需要配置的情况比较简单，简单地就能判断是否需要发起对加速器的配置更新，此时通过 DMA 直接进行配置
  2. 某些加速器其内部 PPM 不够用导致加速器性能出现瓶颈、而其余加速器内部 PPM 空闲

系统架构图如图2所示。

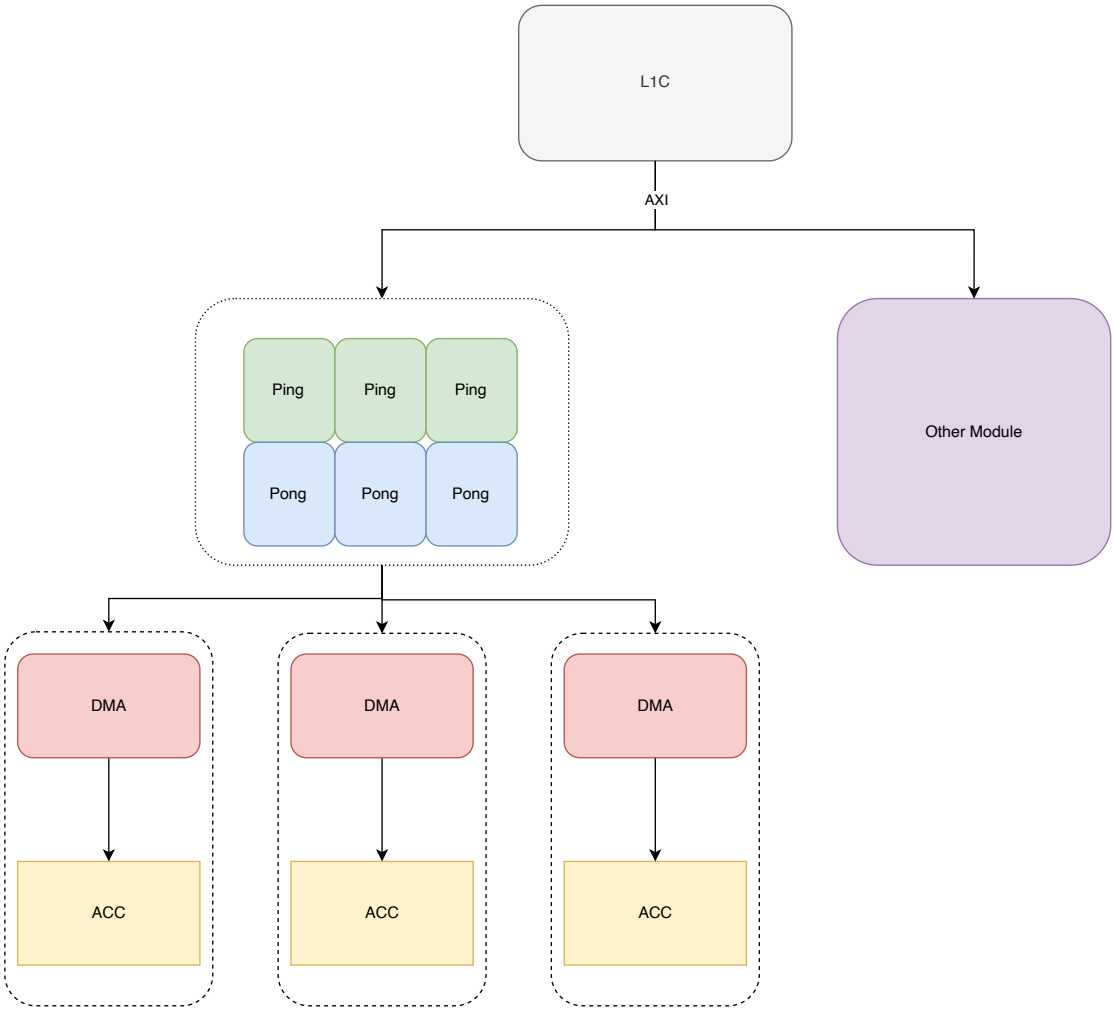

图 2: 每个 ACC 共享一套 PPM，通过各自的 DMA 进行配置

4 所有的 ACC 共享同一块 PPM，通过 MCU 搬运

- 工作流：
  1. 每个加速器内部没有 DMA，数据的搬运通过 MCU 从 PPM 搬运到每个 ACC 中
  2. 满足 ACC 配置的条件之后，可以发起一个中断，触发 MCU 进入到配置 ACC 的中断处理程序中去
  3. MCU 每一次配置 ACC，都可以由程序发起，程序可以指定：
     (a) PPM 读取地址的 base address
     (b) PPM 读取的配置信息的数量
     (c) 需要配置的 ACC
  4. 如果配置的过程被中断了，可以很好地在中断处理结束之后继续之前的配置流程
- 该设计优点：
  1. 灵活性最高，配置信息写入到 ACC 的过程甚至可以由软件进行控制
  2. MCU 跟 ACC 之间的 AXI 总线不需要仲裁结构，因为 MCU 一次只能配置一个 ACC
  3. PPM 只用提供一套读端口供 MCU 读取即可
- 存在的问题
  1. MCU 一次只能下发配置信息到一个 ACC
  2. 单笔配置信息包含两个动作 load, store，对比与 DMA 来说，效率不高
  3. MCU 需要运行其他任务，ACC 的配置动作可能会被 interrupt
- **适合的工作场景**：
  1. L1C 需要处理除 ACC 配置之外的其他很多事务，L1C 计算出的配置信息需要有简单的存储方式进行暂存，从而解放 L1C 去做其他的事务

2. 加速器需要配置信息更新的条件比较复杂，无法简单确定什么时候加速器的配置信息需要更新
3. 加速器需要配置的寄存器数量不是很多，若一个加速器的配置就需要占用 MCU 很长的时间，导致其他加速器配置暂停，就不适用
4. 加速器内部不包含 DMA 控制器的情况

5. MCU需要对L1C下发的配置信息做二次计算

系统架构图如图3所示。

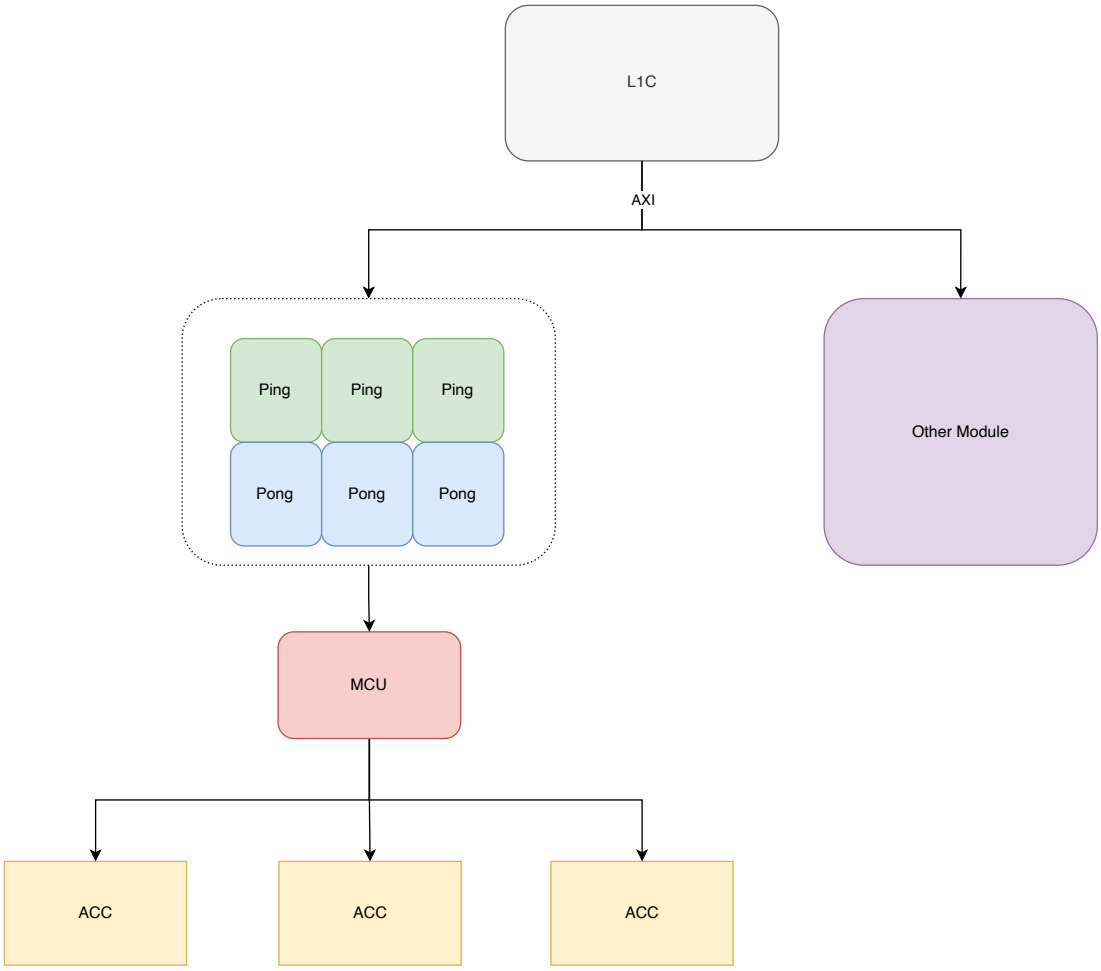

图 3: 每个 ACC 共享一套 PPM，通过 MCU 进行配置

5 一些想问的问题

- **问题一**：加速器是否需要等到所有配置信息都更新完毕了再开始计算?
  1. 在配置加速器配置寄存器的时候，加速器是不是不能开始计算?
  2. 如果是这样的话，那么加速器本身是否需要一个信号来告知它配置信息已经更新完毕了。
  3. 加速器内部是否有一个 **flag** 寄存器，用来表示寄存器配置是否完成?
- **问题二**：什么时候更新 ACC 的配置寄存器信息?
  1. L1C 计算出的一组配置信息写入到 PPM 的时候，就需要更新 ACC 的配置寄存器嘛? 此时 L1C 是否会拉高相应的信号?
  2. PPM 内部是否只能同时存在一组配置信息，这样每次更新 PPM 之后都表示需要更新 ACC 配置寄存器了。
  3. 配置信息的更新，应该需要加速器把当前正在进行的计算完毕再更新，那么计算器是否需要给出一个信号 (中断) 来告诉 MCU 或者 DMA 自己当前计算完毕了，可以更新配置寄存器?
- **问题三**：如何界定 L1C 一组配置信息的边界?
  假如 L1C 有两组给同一个 ACC 的配置信息，那么这两组配置信息的边界，是怎么给出的?
- **问题四**：ACC 的配置寄存器跟 ACC 需要计算的数据源是什么关系?
  1. 其配置信息是由 L1C 计算得出的，那么需要计算的数据由什么给出?
  2. 配置信息的更新跟数据源的更新，是否需要同步?
  3. 计算完多少组数据之后，需要更新配置寄存器?
- **问题五**：假设 PPM 的可以容纳 M 条配置信息，但是 L1C 只写入了 N 条配置信息 $N < M$
  1. 那么在写入配置信息的时候，如何处置 PPM 里无效的那些配置信息?
  2. DMA 搬运的时候，会只搬运有效的配置信息的部分然后就结束任务吗?