LaTex 文档模版

付杰

2023 年 12 月 7 日

目录

1 研究背景

5G 通信相较于 4G 通信，具有更高的数据速率、更低的延迟、更大的容量跟连接密度等优点。5G 基带芯片需要在有限的计算时间 (TBD: 多少 ms?) 内完成复杂的计算，如信号处理和调制解调、FFT 计算等，这些计算为了保证时效性被硬化成了对应的加速器以保证对应的计算任务可以在规定的时隙内完成。

5G 基带芯片通过引入硬件加速器来满足时间严苛的计算需求，但又因此引入了加速器的调度配置问题。基带芯片内部有数十个硬件减速器来处理不同的复杂计算任务，各个加速器需要被正确地配置才能工作、各个加速器完成计算任务之后会发送中断信号给微控制器，微控制器需要及时处理器加速器的中断，才能保证 5G 通信的实施性。

5G 设备需要在更高的频谱和更大的数据速率下运行，这可能导致设备的能源消耗增加；而且搭载基带芯片的设备可能会采用电池进行供电，因此基带芯片内部的微控制器设计的时候需要考虑低功耗设计、并且支持在低负载的情况下进入睡眠模式。

本研究课题主要的内容是设计一款低功耗微控制器对 5G 基带芯片内部的加速器进行调度，要求微控制器本身的计算性能满足加速器调度的需求、并且满足低功耗的特性；在此基础上该微处理器需要针对微控制器需要对数十个加速器调度的应用场景做专门的优化。MCU 在基带芯片中的位置如图1粉色虚线框部分所示。

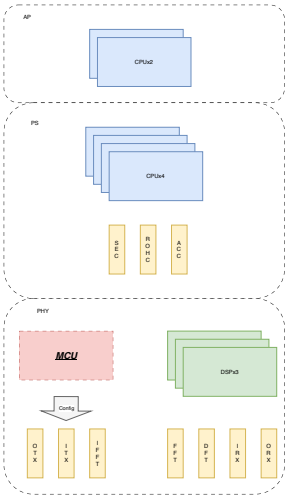

图 1: 所设计微控制器在 5G 基带芯片中的位置

2 研究内容及当前进度

本课题主要目标是基于 RISC-V 指令集开发 MCU，用于控制 5G 基带芯片内部的加速器。需要满足快速响应及低功耗的特征。本课题的研究内容主要有三个方面，如图2所示。

目前本课题主要研究内容及进度如表2所示

cell1	cell2	cell3
cell4	cell5	cell6
cell7	cell8	cell9

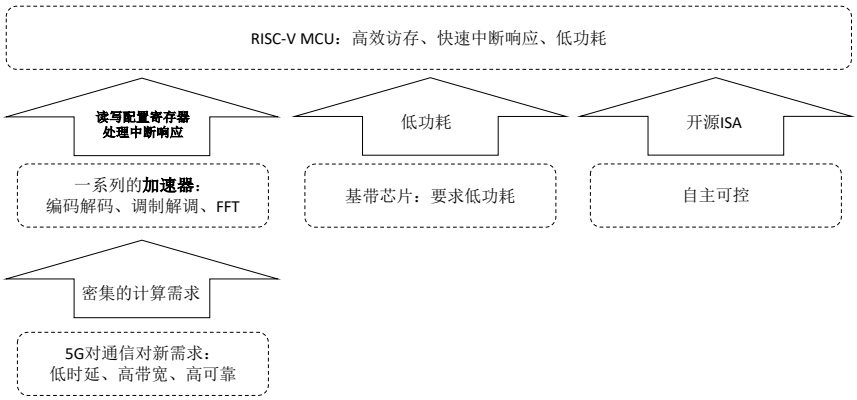

图 2: 本课题研究内容

3 已完成内容

3.1 技术路线

本课题的技术路线采用**先实现再优化**的思想。首先根据当前项目的资料分析了需要解决的问题；确定加速器调度问题的主要应用场景之后确定了微控制器的主要任务，并且在此基础上确定了微控制器采用顺序单发射架构、支持 RISC-V 32IMC 指令集，其中乘法指令集 (M) 是为了保证微控制器一定的计算性能、压缩指令集 (C) 则是压缩程序所占用的存储面积，保证应用程序能够存储。

微控制器功能特性确定之后，按照五级流水线架构对微处理器进行了实现，其中模块集级验证采用了 testbench 进行简单验证；微控制器系统级验证采用了 Difftest 框架进行验证，验证通过了所有的 RISC-V TESTS 测试集，在一定程度上保证了微控制器是正确实现了 RISC-V 手册规范的。

在处理器核功能正确的基础上，针对加速器调度的场景以及低功耗需求进行了优化设计，主要特现在：取指部分针对整数指令地址不对齐做了优化；译码部分实现了分支预测器以及地址重定向的仲裁；访存部分针对加速器配置信息下发做了优化。优化设计完成之后对微控制器再次进行 Difftest 测试，保证微控制器优化设计之后依然符合 RISC-V 手册规范、检查优化之后的微控制器设计是否取得了设计的性能提升。

3.2 研究方法

当前本课题已经实现了取指部分以及译码部分优化设计，针对访存部分的设计优化正在进行当中。

3.2.1 取指部分

(a) 问题提出：由于 5G 基带芯片对于实时性的要求很高，因此用于加速器调度的微控制器本身也需要满足实时性的设计。在这个前提下，微控制器不能使用 Cache 做指令存储，因为 Cache 会存在 Cache Miss 的问题，导致取指部分的时延不能保证，违反了实时性的要求，因此取指部分采用 **ITCM**(Instruction Tightly Coupuled Memory) 做指令存储。ITCM 由于需要匹配微控制器的速度，其容量不能做到很大，因此在设计微控制器架构的时候需要实现压缩指令集，RISC-V 手册规定其压缩指令 (16 bits) 跟整数指令 (32 bit) 在指令存储内部是混合存储的，因此引出了**整数指令跟压缩指令混合存储时整数指令的取指问题**，该问题具体表现为如下两点：

1. 当整数指令取指地址不是 2B 对齐的时候，如何访问一次 ITCM 在一个 cycle 内完成整数指令的取指。

2. 整数指令跟压缩指令在指令存储内混合存放，微控制器如何对从指令存储内取出的指令做边界判定，识别出每一条压缩指令跟整数指令。

(b) 研究目标：微控制器的取指部分需要实现在一个 cycle 内取出一条整数指令，并且完成整数指令跟压缩指令的识别并且在整数指令地址不对齐的时候完成对整数指令的拼接。

(c) 设计方案：取指部分采用 **odd even ITCM 跟取指 FIFO** 来实现上述的设计目标。其设计目标如图3所示。

- **odd even ITCM**：指令的取指地址可能是 4B 对齐也可能是 2B 对齐的，地址 2B 对齐的时候需要访问两次 ITCM 才可以才可以从 ITCM 里取出整数指令的高 16bits 跟低 16bits，再通过拼接电路完成整数指令的拼接。
  当地址是连续递增的时候可以采用 *leftover buffer* 存储上一次访问 ITCM 得到的高 16 bits 数据, 如图4所示。当地址 2B 对齐的时候可以利用 *leftover buffer* 里的内容实现整数指令的拼接。但是这种方法在地址出现跳转时会失

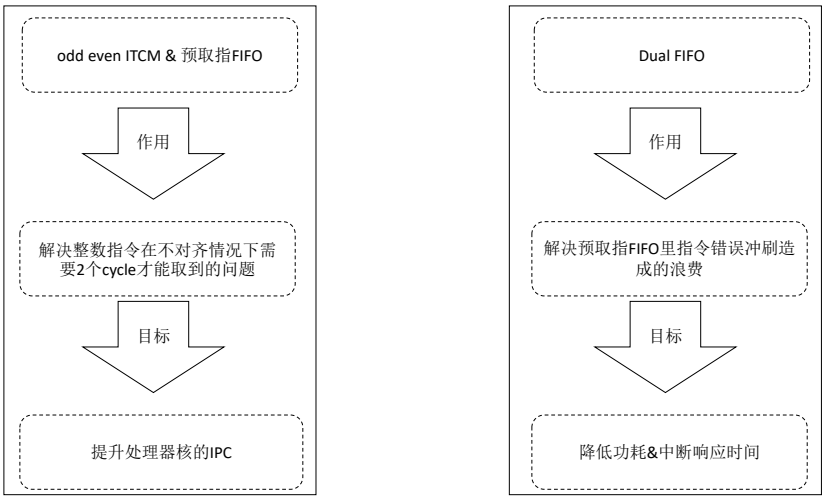

图 3: 采用 odd even ITCM 跟取指 FIFO 解决整数指令跟压缩指令混合存储时整数指令取指问题

效。本课题采用 *odd even ITCM* 可以解决指令跳转后出现地址不是 4B 对齐时的整数指令的取指问题。其核心思想在于**将 ITCM 分为 odd, even** 两部分，两部分可以独立的访问。当地址不是 4B 对齐的时候，*even ITCM* 访问地址比 *odd ITCM* 高一位，并且将 *even ITCM* 里的数据当作高 16 bits，*odd ITCM* 里的数据当低 16 bits 完成指令的拼接；当地址是 4B 对齐的时候，两块 ITCM 的访问地址一样，并且 *even ITCM* 的数据当作低 16 bits 完成指令的拼接。具体操作原理如图 TBD 所示。

- **取指 FIFO** 在访问 ITCM 的时候，只能根据地址是否是 4B 对齐控制访问 ITCM 的动作，但是从 ITCM 里取出的指令可能的组合有如下五种情况 TBD 增加论文 ref:

  1. 取出的 32 bits 数据是一条整数指令
  2. 取出的 32 bits 数据是两条压缩指令
  3. 取出的 32 bits 数据是一条压缩指令跟一条整数指令的低 16 bits
  4. 取出的 32 bits 数据是一条整数指令高 16 bits 跟一条压缩指令
  5. 取出的 32 bits 数据是一条整数指令高 16 bits 跟一条整数指令低 16 bits

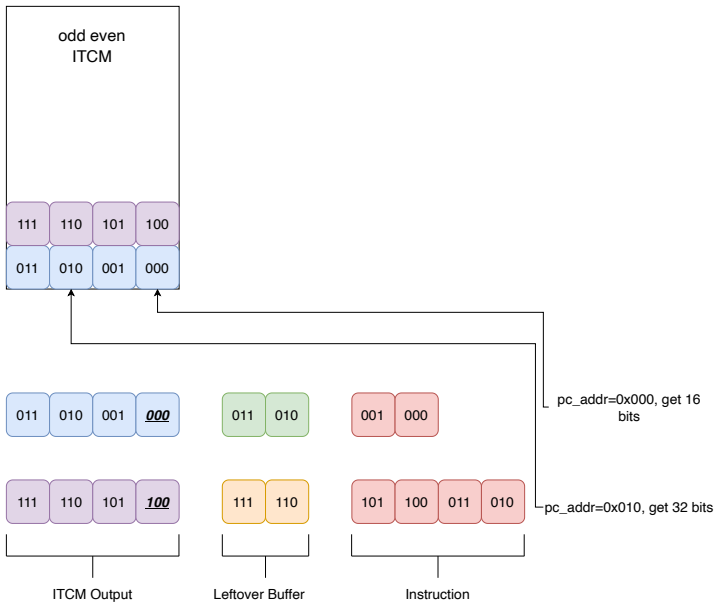

图 4: 采用 leftover buffer 来实现整数指令的拼接

为了判断具体指令的类型，需要复杂的硬件电路来从上述五种可能的情况里判断出取出的数据具体属于哪种类型，然后根据当前指令的类型来更新下一次取指地址的值。这会导致如下的缺陷：

1. 取指级的周期变长：取出来的数据需要复杂的硬件电路才可以判断出指令的类型，这会增减组合逻辑电路的延时；下一次取指的地址需要根据当前指令的类型选择加 4 或者加 2。
2. 在可能的组合 2、4 中，高 16 bits 的压缩指令会被重复取出，因为这两种情况下取指地址会加 2，导致了额外的访存功耗。

基于上述硬件电路的缺点，本课题提出了 *5*16* 的取指 FIFO，用于存储从 ITCM 中取出的数据、完成指令类型的判断以及整数指令的拼接，其结构如图 TBD 所示；其操作逻辑如下：

1. FIFO 总容量为 5，表示其内部的 5 个 16 bits 的寄存器。容量设计为 5 是为了避免写入到 FIFO 的时候出现 overflow。
2. 从 ITCM 中每次都会得到 32 bits 的数据，当 FIFO 容量 ≤ 3 的时候

FIFO 允许被写入，取出的 32 bits 数据会被 push 到 FIFO 中。

3. 当译码级没有被 stall 的时候，每个 cycle 会译码一条指令，译码的时候会判断 FIFO 头部寄存器的低 2 bits 是否是 *11*。根据 RISC-V 规定，指令低 2 bits 如果是 11 则表示当前指令是一条整数指令，此时会从 FIFO 头部 pop 出 2 个 16bits 的数据，即一条整数指令；反之如果低 2 bits 不是 11，则说明当前指令是一条压缩指令，此时只需要 pop 出 FIFO 头部一条指令即可。

- **Dual FIFO**

3.2.2 译码部分

4 未来工作计划