处理器设计报告

哈尔滨工业大学（深圳）春日影队
金正达、章晨冉、官佳文、董佳鹏

一、设计简介

本项目是第八届“龙芯杯”全国大学生计算机系统能力培养大赛（NSCSCC 2024）的参赛作品。项目使用 System Verilog 语言进行开发[1]，使用 Chiplab 框架进行仿真测试。

项目实现了一款基于龙芯架构 32 位精简版 (LoongArch32-Reduced, LA32R) 指令集的八级静态流水线的超标量顺序双发射 CPU。功能上，该 CPU 实现了大赛要求的指令，支持例外的精确处理，拥有一级动态分支预测单元，拥有 8KB 大小的一级指令缓存和 8KB 大小的一级数据缓存，通过 AXI 协议接口与外部进行通信。

二、设计方案

（一）总体设计思路

本处理器是基于 Loognarch-32r 指令集的超标量顺序双发射处理器。

1. 实现指令

按照指令类别，本处理器实现指令如下：

表格 1：指令支持

算数运算类指令	add.w, sub.w, addi.w, lu12i.w, slt, sltu, slti, sltui, pcaddu12i, and, or, nor, xor, andi, ori, xori, mul.w, mulh.w, mulh.wu, div.w, div.wu, mod.w, mod.wu
移位运算类指令	sll.w, srl.w, sra.w, slli.w, srli.w, srai.w
转移指令	beq, bne, blt, bltu, bge, bgeu, b, bl, jirl
普通访存指令	ld.b, ld.bu, ld.h, ld.hu, ld.w, st.b, st.h, st.w
原子访存指令	ll.w, sc.w
特权指令	csrrd, csrwr, csrxchg，ertn, idle
系统异常指令	syscall, break
计时器相关指令	rdcntvl.w, rdcntvh.w, rdcntid

2. CSR 实现

本处理器实现了 Loognarch-32r 指令集手册所要求的全部 CSR，具体如下表所示：

表格 2：CSR 支持

地址	名称	地址	名称
0x0	当前模式信息 CRMD	0x19	低半地址空间全局目录基址 PGDL
0x1	例外前模式信息 PRMD	0x1A	高半地址空间全局目录基址 PGDH
0x4	例外配置 ECFG	0x1B	全局目录基址 PGD
0x5	例外状态 ESTAT	0x20	处理器编号 CPUID
0x6	例外返回地址 ERA	0x30-33	数据保存 SAVE0-SAVE1
0x7	出错虚地址 BADV	0x40	定时器编号 TID
0xc	例外入口地址 EENTRY	0x41	定时器配置 TCFG
0x10	TLB 索引 TLBIDX	0x42	定时器值 TVAL
0x11	TLB 表项高位 TLBEHI	0x44	定时中断清除 TICLR
0x12	TLB 表项低位 0 TLBELO0	0x60	LLBit 控制 LLBCTL
0x13	TLB 表项低位 1 TLBELO1	0x88	TLB 重填例外入口地址 TLBEENTRY
0x18	地址空间标识符 ASID	0x180-181	直接映射配置窗口 DMW0-DMW1

3. 例外与中断

本处理器支持的例外类型如下表所示，处理器对所有例外的处理都是精确处理，即对于引发例外的指令，位于该指令前的所有指令都会被执行，其后的指令不会被执行。

表格 3：例外支持

Ecode	EsubCode	例外代号	例外类型
0x0	0	INT	中断
0x1	0	PIL	Load 操作页无效例外
0x2	0	PIS	Store 操作页无效例外
0x3	0	PIF	取指操作页无效例外
0x4	0	PME	页修改例外
0x7	0	PPI	页特权等级不合规例外
0x8	0	ADEF	取指地址错例外
	1	ADEM	访存指令地址错例外

0x9	0	ALE	地址非对齐例外
0xB	0	SYS	系统调用例外
0xC	0	BRK	断点例外
0xD	0	INE	指令不存在例外
0xE	0	IPE	指令特权等级错例外
0x3F	0	TLBR	TLB 重填例外

4. 地址翻译模式

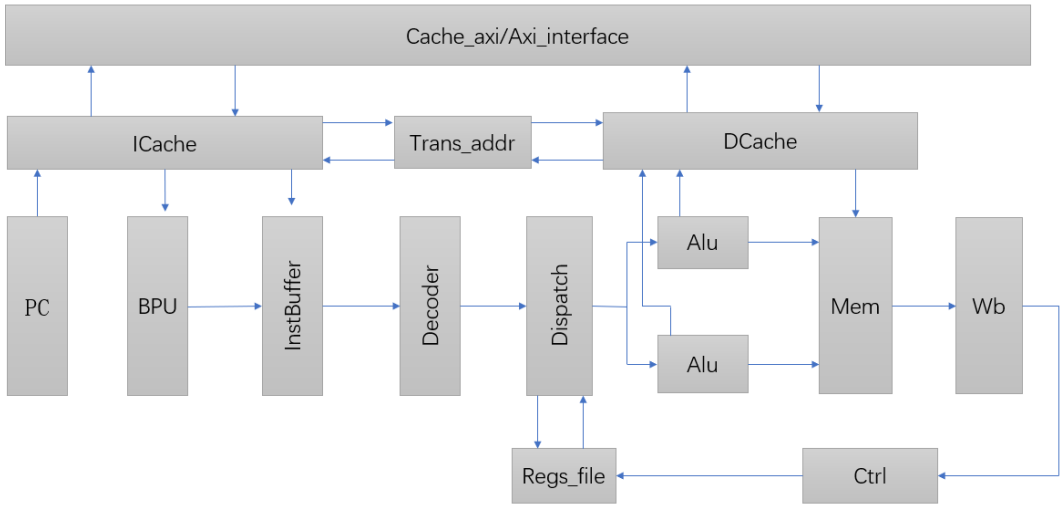
本处理器目前的地址翻译模块仅支持直接地址翻译模式和直接映射地址翻译模式

直接地址翻译模式：当 CSR.CRMD 的 DA=1 且 PG=0 时，为直接地址翻译模式。在这种映射模式下，物理地址等于虚拟地址。

直接映射地址翻译模式：当 CSR.CRMD 的 DA=0 且 PG=1 时，为映射地址翻译模式，通过直接映射配置窗口机制完成虚实地址的直接映射。当虚地址命中某个有效的直接映射配置窗口时，物理地址为虚拟地址的[28:0]位拼接上映射窗口所配置的物理地址高位。

5. 流水级划分

本处理器的整体流水级共 8 级，可分为前端与后端两个部分，两者通过指令缓冲队列（Instruction Buffer）进行解耦，前后端流水级具体划分如下图所示：



图表 1：CPU 架构图

各个流水线功能如下：

取指 1：将 PC 送入地址翻译模块进行地址翻译，并将其作为虚地址读取 ICache 的 Block Ram。

取指 2：根据 ICache 的命中结果返回取到的指令。

分支预测：对从 ICache 返回的 PC 和指令进行分支预测，给出跳转的方向和目标地址。

译码：对从指令缓存中读出的指令进行译码，将译码后的指令存入发射队列。

发射：一次读取发射队列队头的两条指令，判断是否发射并读取寄存器堆获得源操作数。

执行：根据指令类型进行运算，若是访存指令，则将访存虚地址送入地址翻译模块，并将其作为虚地址读取 DCache 的 Block Ram。

访存：根据访存指令类型和 DCache 的控制信号获取访存结果。

写回：根据指令类型及异常相关信息，控制写回信号将指令执行结果写入寄存器堆。

（一）指令缓存（ICache）模块设计

适配双发射内核的 ICache，每次读取 PC 和 PC + 4 对应的指令。存储采用 Block Ram 实现，Cache 大小为 8KB，使用 VIPT 的查询方式。ICache 的映射方式为二路组相联，主存字块标记 20 位，Cache 索引地址 7 位，字块内偏移 5 位，故一个 Cache 行存储 8 个字。设置刷新信号 branch_flush 和暂停信号 pause_icache。替换策略采用伪最近最少使用算法（PLRU）。

ICache 状态机共有 5 个状态：IDLE，ASKMEM1，ASKMEM2，RETURN，UNCACHE。

- IDLE：初始状态。在 Cache 命中时会持续保持 IDLE 状态。即 IDLE 状态下，若 Cache 命中，可在一个周期内接收新的取指令请求并返回上一个取指令的结果；若未命中，则根据 PC 和 PC + 4 的具体命中情况进入 ASKMEM1 或 ASKMEM2 状态中。
- ASKMEM1：PC 对应的指令在 ICache 中缺失则会进入该状态。在 ASKMEM1 状态下会向主存发起读请求，直到主存返回时接受一个 Cache 行的数据并写入 ICache。再根据 PC + 4 是否命中决定进入 ASKMEM2 或 RETURN 状态。
- ASKMEM2：PC + 4 对应的指令在 ICache 中缺失则会进入该状态。在 ASKMEM2 状态下会向主存发起读请求，直到主存返回时接受一个 Cache 的数据并并写入 ICache。随后进入 RETURN 状态
- RETURN：ICache 中缺失的数据从主存都取回后会进入该状态，在 RETURN 状态下会向前端返回取到的指令。随后进入 IDLE 状态。
- UNCACHE：Uncache 模式下会进入该状态。进行对应的 Uncache 读操作，并将读取到的结果返回给前端。随后进入 IDLE 状态。

ICache 的流水级：

- P0：得到输入的 PC，将 PC 送往 BRAM 进行读，同时也将虚拟地址给地址翻译模块进行地址转换。
- P1：得到 BRAM 的读取结果，也得到了物理地址，将 BRAM 中存储的 TAG 信息和物理地址的 TAG 位进行比较，判断命中。若命中则向前端返回取到的指令，否则停顿并向主存读。

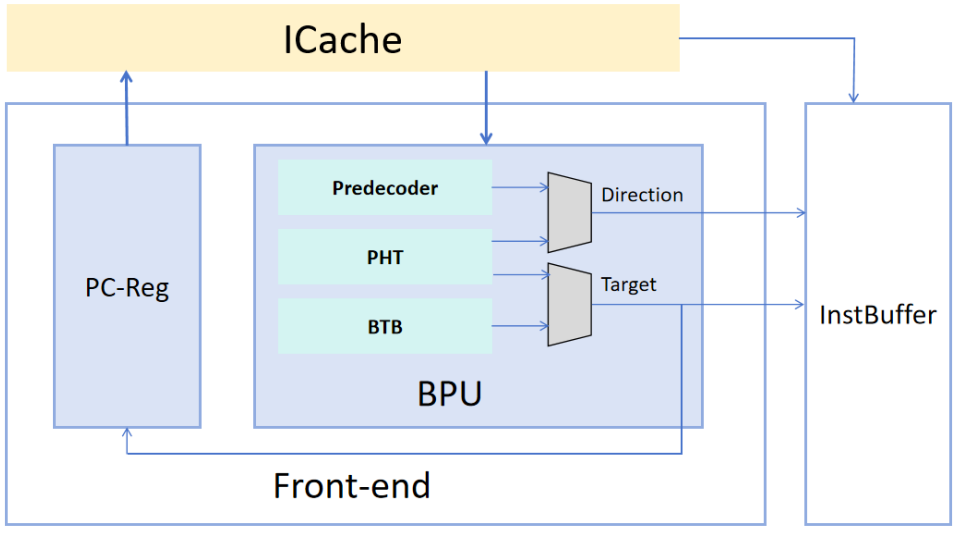
ICache 对特殊情况的处理：

Flush 信号的实现：由于可能 flush 时向主存读的请求已经发出，故这种情况下忽略主存的一次返回。引入 real_ret_valid 等信号作为经过忽略选择之后真实有效的主存返回信号。

考虑到情况：当 PC 和 PC+4 位于 Cache 中的同一路时，虽然两个都命中缺失，但不需要向主存读两次，故引入信号 same_way 来进行控制。

（二）分支预测（BPU）模块设计

由于本处理器分支预测失败会导致流水线产生 6 个周期的惩罚，而且分支指令在全指令中占比较多，所以提高分支预测的准确度对于提高处理器性能至关重要。但是过于复杂的分支预测逻辑的设计又会增加前端的组合逻辑延迟。所以为了权衡分支预测准确率和前端结构的简洁性，本处理器的分支预测模块（BPU）采用了“Predecoder（预译码）+ PHT（方向预测）+ BTB（目标预测）”的结构，既能保证分支预测的高准确率，又能保证前端逻辑结构的简洁性。带有分支预测的前端结构如下：



图表 2：前端结构图

BPU 为分支预测的顶层模块，该模块获取 ICache 传来的 PC 和指令，并利用上述模块根据信息进行预测，最后将预测结果和其他信息一起传给指令缓冲（Instruction Buffer）模块。由于 BTB 和 Predecoder 模块可能都会给出目标地址，因此需要对多个目标地址进行取舍。故而 BPU 模块中包含了一个多选器，如果预解码出的结果表明，该分支指令属于无条件直接跳转指令，则会优先将跳转地址的译码结果作为跳转地址，并且设置方向预测为跳转，否则会使用 PHT 和 BTB 的预测结果。

BPU 中实例化了以下子模块：

1.预译码（Predecoder）模块

预译码模块完成两部分工作。第一部分工作是对从 icache 取得的指令进行判断，对其是否是分支指令以及分支的类型进行判断。第二部分工作是对于无条件直接跳转的指令进行直接译码，将跳转的地址计算出来，从而减少了预测失败的风险。

2.方向预测（PHT）模块

方向预测模块采用二位饱和计数器表格存储分支预测的方向信息，使用 PC 的 2-8 位作为索引值。该结构为了简化设计，没有使用分支的历史信息，这样一方面可以使得方向预测的训练在较短的时间内完成，对于最常见的循环结构、调用类型具有很好的预测效果，极大提高了预测的精度。另一方面，对于那些规律难以捕捉的分支指令，预测的效果较差，但是由于该类指令的数目较少，所以对整体预测准确率的影响不大。因此整体来看，该设计可以保证较高的预测精度，同时也简化了方向预测的逻辑。

3.目标预测（BTB）模块

目标预测模块采用了直接映射的 Cache 结构存储分支预测的目标地址信息，每一行包括了 32 位的地址信息和 1 位的有效位，采用 PC 的 2-8 位作为索引值。通过观察，如果设置 tag 位进行检验，并不会显著提高预测目标的准确率，反而使得逻辑延迟过大，故取消了该部分的设计。、

（三）指令缓冲（InstBuffer）模块设计

为了实现前后端的解耦，使得前端取指和后端执行互不干扰，设置了 Instbuffer 作为缓冲区。其中实例化了两个 FIFO 作为双发射指令缓冲队列，从前端取得的两条指令及其他相关信息分别存入两个 FIFO 中，发射时同时发出。FIFO 中的一条信息包含了：指令 PC、指令的二进制码、分支预测的信息以及例外信息。当队列满时，会暂停前端取指，当队列空时，会向后端发射空指令，以此来保证前后端工作的独立性。

（四）译码（Decoder）模块设计

译码模块每周期至多从指令缓冲队列中读取两条指令进行译码，该模块会将读取的指令同时送入 6 个译码单元并行译码，并最终选择译码有效的输出作为该指令的译码结果。若各个译码单元均未给出译码有效信号，则会触发指令不存在例外，若存在系统调用例外，也会在该阶段被附在指令上。同时，该阶段会将前端传递的分支预测信息以及异常信息附在译码结果中传递至下一个模块。

译码模块中实例化了一个指令发射队列，最大容纳 8 条待发射的指令，译码结束后会将指令加入队列尾部，等待发射阶段仲裁发射。

当发射队列满时，译码模块会发出流水线暂停请求，以暂停从指令缓冲的读取指令的过程。

（五）发射（Dispatch）模块设计

发射模块每周期至多接受发射队列传来的两条指令，并对指令的是否发射进行仲裁。当两条指令存在数据相关、两条指令中存在访存指令、两条指令中存在特权指令时，该周期仅发射第一条指令，第二条指令仍然留在发射队列中等待下一次仲裁；其余情况下，指令均为双发射。

发射阶段在仲裁发射的同时，会根据指令的译码结果，进行读取寄存器堆（GPR）和读取控制状态寄存器（CSR）的操作。其中，执行、访存和写回阶段的寄存器堆读写信号会被前递至发射模块，供当前指令读取寄存器堆时进行选择，选择优先级为执行>访存>写回，控制状态寄存器的读取没有实现数据前递。

根据执行阶段返回的指令类型，若判断当前存在 Load to use 冒险，则发射模块会发出暂停请求，等待上一条 Load 指令的执行完毕。

（五）执行（Execute）模块设计

执行模块实例化了两个相同的 ALU 执行单元，每个 ALU 执行单元均可进行算数运算、分支判断与计算以及访存信息计算，两个 ALU 均实例化了一个乘法器和除法器。

乘法器使用了 FPGA 板上的 DSP 资源，从执行阶段接受到乘法指令到乘法指令结果得出共需 3 个周期。除法器使用基于前导零的除法器[2]，在输入延迟一个周期之后，通常约 5 个周期可得出结果。

分支执行单元会将分支实际执行结果传递回前端分支预测单元进行更新，对于分支预测

错误的情况，会发出流水线重定向信号，清空流水线从正确的分支地址重新取指执行。

若指令进入执行阶段后已经携带异常、执行完未进入下个阶段时已经产生异常、该指令为 ertn 指令，执行阶段也会发出流水线重定向信号，清除此时处于译码和发射阶段的指令信息，防止当前指令提交前后面的指令进行访存行为。

执行乘法指令、除法指令或者访存指令但 DCache 处于忙碌状态时，执行阶段会发出暂停请求，等待乘除法指令执行完毕或 DCache 成功接受访存请求。

（六）访存（Mem）模块设计

访存模块根据执行阶段的指令信息判断接受 Load 指令的访存结果，当前为访存指令且 DCache 返回有效时，访存模块将返回数据记录，传递至写回模块；若 Dcache 未返回有效，则访存模块会发出暂停请求，等待 DCache 结果的返回。

（七）数据缓存（DCache）模块设计

DCache 大小为 8KB，映射方式为二路组相联，使用 VIPT 的查询方式。主存字块标记 20 位，Cache 索引地址 7 位，字块内偏移 5 位，故一个 Cache 行存储 8 个字。DCache 存储采用 Block Ram。Tag 和 valid（数据有效）信号也用 Block Ram 进行存储，Dirty（数据为脏）信息用寄存器存储。替换策略采用伪最近最少使用算法（PLRU）。

DCache 状态机共有 6 个状态：IDLE, ASKMEM, WRITE_DIRTY, REFILL, RETURN, UNCACHE。

- IDLE：初始状态，在 Cache 命中时会持续保持 IDLE 状态。即 IDLE 状态下，若 Cache 命中，可在一个周期内接收新的访存请求并返回上一次访存的结果；否则未命中，根据被替换的 Cache 行是否脏决定下一状态进入 WRITE_DIRTY 还是 ASKMEM 。特别的，当 Cache 命中且为写操作时，下一状态会进入 RETURN 来进行一拍延迟，使 BRAM 输出结果稳定。
- ASKMEM：Cache 缺失则会进入该状态。在 ASKMEM 状态下会向主存发起读请求，直到主存返回时接受一个 Cache 行的数据并写入 ICache。下一状态进入 REFILL。
- WRITE_DIRTY：Cache 缺失且被替换的 cache 行为脏则会进入该状态。在 WRITE_DIRTY 状态下会向主存发起写请求，将脏数据写回主存，直到得到写完成信号，下一状态进入 ASKMEM。
- REFILL：重填 Cache 行的周期，下一状态进入 RETURN。
- RETURN：Cache 未命中情况下返回访存结果的状态，也是写 BRAM 之后延一拍使 BRAM

输出稳定的一个周期，每当对 BRAM 进行写之后会进入。下一状态直接进入 IDLE。

- UNCACHE：Uncache 模式下会进入该状态。进行对应的 Uncache 读操作。完成之后进入 IDLE 状态。

DCache 的流水级：

- P0：得到输入的访存信号，将虚拟地址送往 BRAM 进行读，同时也将虚拟地址给地址翻译模块进行地址转换。
- P1：得到 BRAM 的读取结果，也得到了物理地址，将 BRAM 中存储的 TAG 信息和物理地址的 TAG 位进行比较，判断命中。若命中则进行对应的读写操作，否则停顿并开始向主存读写。
- P2：将读操作的结果返回给后端。

（八）控制（Ctrl）模块设计

控制模块控制流水线重定向信号，流水线暂停信号的生成。

流水线传递至此的例外信息会被按照优先级统一处理，中断请求会在该阶段被附在指令上，例外处理结果会决定该指令的执行结果是否被提交，以及其结果是否被写入寄存器堆，以此实现例外的精确处理。

对于流水线暂停信号的生成，以指令缓冲（Instruction Buffer）为界，后端的暂停请求和前端的暂停请求分离，从而使后端的暂停不影响前端的取指。

（九）寄存器堆（Regs_File）模块设计

寄存器堆模块拥有 4 个读端口和 2 个写端口。内部实例化了 8 个 reg_lutram 模块，一个写端口对应 4 个 reg_lutram，一个读端口对应 2 个 reg_lutram。当有写请求时，对应的 4 个 reg_lutram 被同时写入；当有读请求时，会同时读出 2 个数据并根据 last_valid_table 的值进行选择输出。

寄存器堆模块拥有如下子模块：

1. Last_valid_table 模块

last_valid_table 用于一个读端口同时读取两个数据时选择哪个数据进行输出。该模块拥有 2 个写端口和 4 个读端口，内部有一个 32 位的 reg 数组，当有新的数据写入时，该数组对应的位置也会被更新，若为第一个端口写入，则更新为 0，若位第二个端口写入，则更新为 1。

2. Reg_lutram 模块

reg_lutram 模块是寄存器堆的储存模块，拥有 1 个读端口和 1 个写端口，内部有一个 32*32 的寄存器数组，仅有一个写端口的设计使其在 FPGA 上可以被综合为 Distributed Ram。

（十）Cache_axi 模块设计

Cache_axi 模块负责在缓存与 AXI 总线之间实现数据传输，支持对指令缓存（ICache）、数据缓存（DCache）和未缓存数据的读写操作。模块通过状态机控制数据流，并与 AXI 总线进行交互。其设计目的是在缓存与 AXI 总线之间提供桥梁，管理缓存和未缓存的读写请求，确保数据传输的有效性和可靠性。

- **读状态机：**读状态机负责处理各种读请求，包括 ICache、DCache 和未缓存的读请求。状态机的状态如下：
  - STATE_READ_FREE：空闲状态，等待读请求。
  - STATE_READ_ICACHE：处理 ICache 读请求。
  - STATE_READ_IUNCACHED：处理 ICache 未缓存读请求。
  - STATE_READ_DCACHE：处理 DCache 读请求。
  - STATE_READ_DUNCACHED：处理 DCache 未缓存读请求。
- **写状态机：**写状态机负责处理各种写请求，包括 DCache 和未缓存的写请求。状态机的状态如下：
  - STATE_WRITE_FREE：空闲状态，等待写请求。
  - STATE_WRITE_BUSY：处理 DCache 写请求。
  - STATE_WRITE_DUNCACHED：处理 DCache 未缓存写请求。

（十一）Axi_interface 模块设计

Axi_interface 模块的核心职责是桥接缓存系统与 AXI 总线，实现数据的有效传输。它处理从指令缓存（ICache）以及数据缓存（DCache）到 AXI 总线的读写操作，并且支持未缓存数据的传输。该模块通过状态机管理数据流，确保在处理不同读写请求时，AXI 总线的读写信号和数据通路得到正确配置。设计目标是通过精确的控制逻辑，确保数据传输的高效性与可靠性，从而实现缓存与 AXI 总线之间的无缝对接。

- **读状态机：**读状态机负责处理各种读请求，包括 ICache、DCache 和未缓存的读请求。状态机的状态如下：
  - AXI_IDLE: 空闲状态，等待读请求。

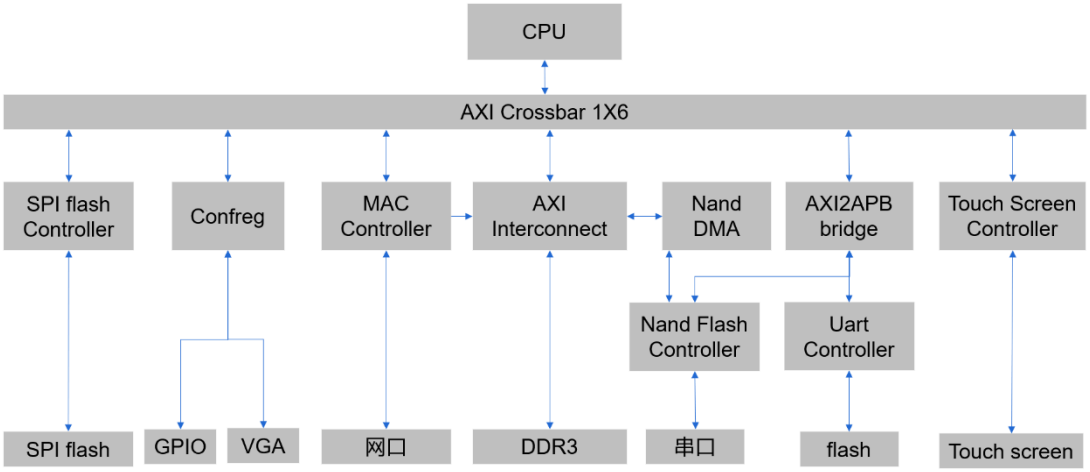
- ARREADY: 处理 AXI 总线的读地址握手。
- RVALID: 处理 AXI 总线的读数据传输。

➢ **写状态机：**写状态机负责处理各种写请求，包括 DCache 和未缓存的写请求。状态机的状态如下：

- AXI_IDLE: 空闲状态，等待写请求。
- AWREADY: 处理 AXI 总线的写地址握手。
- WREADY: 处理 AXI 总线的写数据传输。
- BVALID: 处理 AXI 总线的写响应。

三、外设

（一）概述



图表 3：SOC 架构图

春日影队的外设设计重点在于可视化。为此，我们实现了对 LCD 触摸屏和 VGA 的控制。此外，我们对比赛提供的 SoC 进行了简单的修改，使得 CPU 能够与 LCD、VGA 等外设实现联动，真正将其构建为一个完整的系统。

（二）LCD 控制器设计

我们精心设计了一款 LCD 控制器，专门用于处理显示和触控功能的初始化，并且具备一定的图形硬件加速能力。为了确保控制器与 CPU 之间的高效交互，我们在设计中采用了 AXI 协议对接口进行了封装，从而实现了系统间的无缝连接。

在显示功能上，由于在一个区域内逐点扫描所有像素点需要消耗大量的时钟周期，直接依赖 CPU 来完成每个像素点的绘制并不现实。为此，我们设计了一套专用的硬件加速模块。无论要显示什么图像，本质上都是在矩形区域内对像素寄存器进行颜色写入。基于这一原理，我们开发了一个矩形区域绘制模块。只需向该模块传入颜色和边界信息，它便能够自动完成整个区域的绘制工作。借助这一模块，CPU 只需通过三条 SW 指令即可完成一个矩形区域的绘制任务，大大降低了 CPU 的工作负担，使其能够专注于其他更多的操作。

在触控功能上，直接由 CPU 来控制电容触控芯片的寄存器读写不仅耗时较多，而且由于需要满足严格的时序要求，效率并不高。为了克服这一问题，我们设计了一个触摸控制器。该控制器能够持续监测电容触控芯片的寄存器，实时获取触控与坐标信息，并将其打包存储在 LCD 顶层模块的一个寄存器中。这样，CPU 只需通过一次读操作（执行 LW 指令）便可获取当前的所有触控信息，从而大幅提升了系统的效率。

除此之外，我们还开发了一款简洁且高效的用户界面，并将开机界面的 Logo 和主界面图形等信息预先存储在 ROM 中，以提升绘制效率。

（三）VGA 控制器设计

在 VGA 控制器的设计中，我们采用了一种简洁而有效的方案，重点实现了图像的稳定显示。具体而言，我们将一张预先生成的图像以 COE 文件的形式加载到 RAM 中，并通过 VGA 控制器将其输出到显示屏上。由于图像已固定存储在 RAM 内，因此系统只需进行简单的读取操作，便可将这张图像准确地展示在屏幕上。

虽然该方案的功能相对单一，仅能显示一张静态图像，且无法进行实时更改，但它在实现的过程中表现出极高的稳定性和可靠性。我们将这种设计方式应用于本次项目中，确保了图像显示的清晰度和一致性。这样的设计不仅简化了开发流程，同时也为系统提供了一个高效而坚实的显示解决方案。

四、设计结果

（一）设计交付物说明

|-score.xlsx	功能测试，性能测试成绩
|-design.pdf	设计文档
|--bit	比特流文件

|--show/	决赛展示 SOC 环境
|--src/	AXI 目录
| |--mycpu	CPU 源码
| |--xilinx_ip	CPU 使用的 IP 核
| |--perf_clk_pll.xci	性能测试时钟 IP 核

（二）设计演示结果

本处理器通过全部 58 个功能测试，初赛性能测试指标如下：

序号	测试程序	myCPU 上板计时(16进制) 数码管显示 (SoC Count) (最左开关拨上)	openla500 上板(16进制) 数码管显示 (SoC Count)	$T_{openla500}/T_{mycpu}$
cpu_clk : sys_clk		80MHz : 100MHz	40MHz : 100MHz	-
1	bitcount	66480	d1612	2.0470946
2	bubble_sort	2bfc20	3ca393	1.378632495
3	coremark	5c0c20	aa6379	1.851096642
4	crc32	352d7e	7f8c71	2.398535288
5	dhrystone	b6c1c	15acff	1.897670498
6	quick_sort	24aa3f	41a98f	1.790871284
7	select_sort	141eae	275f21	1.956854147
8	sha	1f6d3b	4aa29a	2.374897977
9	stream_copy	232df	3e2bd	1.767257712
10	stringsearch	1d7748	32ce13	1.724193198

决赛性能测试指标如下：

序号	测试程序	myCPU		CPU count : SoC cout	openla500	$IPC_{openla500}/IPC_{mycpu}$
		上板计时(16进制)			上板(16进制)	
		数码管显示 (CPU count) (最左开关拨下)	数码管显示 (SoC count) (最左开关拨上)		数码管显示 (CPU count) (最左开关拨下)	
cpu_clk : sys_clk		80MHz : 100MHz		-	40MHz : 100MHz	-
1	bitcount	4bb97	5eabf	0.799863323	53ccc	1.106642551
2	bubble_sort	23c4b6	2cb5e9	0.799998498	184315	0.678309283
3	coremark	4af2c9	5db03f	0.799974495	44712a	0.913189152
4	crc32	2a0904	348bb3	0.799974446	330ff1	1.214749784
5	dhrystone	86857	a82ae	0.799924218	8b0ce	1.033667938
6	quick_sort	1d05a3	24475e	0.799972325	1a7e6d	0.912887943
7	select_sort	101ab9	1421a8	0.799960738	fd389	0.982732891
8	sha	1a2025	20a8b0	0.799951503	1df59a	1.146745787
9	stream_copy	1e780	261e3	0.799328769	1af7f	0.885120192
10	stringsearch	169b70	1c428e	0.799971491	146361	0.901846267
11	fireye_A0	217613	29d3e8	0.79997658	20cbf2	0.980139221
12	fireye_B2	720dc	8e954	0.799910962	737e0	1.012612273
13	fireye_C0	ca639	fd00a	0.79994847	9d625	0.777631682
14	fireye_D1	3dd495	4d49ff	0.799989142	397f05	0.929901333
15	fireye_I2	3febc6	4fe6fb	0.799989688	3a65a0	0.913580542
16	inner_product	a730f6	d0fd84	0.799995298	929cd9	0.876916005
17	lookup_table	270f8d	30d3b0	0.799984062	29f375	1.073993949
18	loop_induction	6b1f0a	85e713	0.799993573	62d195	0.922493746
19	my_memcmp	2a215a	34a9f8	0.799983427	23c215	0.848751381
20	minmax_sequence	393c61	478bbb	0.799988782	353e28	0.930235038

五、参考设计说明

ICache 存储使用 Xilinx IP 的 True Dual Port Ram 实现，DCache 的存储使用 Xilinx IP 的 Simple Dual Port Ram 实现。

分支预测的逻辑设计参考了 2020 年 UltraMIPS 队伍的实现。

指令缓冲队列（fifo.sv），除法器（div_alu.sv,clz.sv）和寄存器堆（regs_file.sv, last_valid_table.sv, reg_lutram.sv）模块参考了 2022 年 404 NOT FOUND 队伍的实现。

六、参考文献

[1] Sutherland S, Mills D. Synthesizing systemverilog busting the myth that systemverilog is only for verification[J]. SNUG Silicon Valley, 2013: 24.

[2] Eric Matthews, Alec Lu, Zhenman Fang, and Lesley Shannon. 2022. Quick-Div: Rethinking Integer Divider Design for FPGA-based Soft-processors. ACM Trans. Reconfigurable Technol. Syst. 15, 3, Article 32 (September 2022), 27 pages. https://doi.org/10.1145/3502492