基于 Chisel 的 RISC-V 处理器设计*

秦国锋** 周涛

同济大学电子与信息工程学院计算机科学与技术系，上海 200092

摘 要 随着物联网和人工智能技术的迅速发展，基于特定应用场景的处理器核得到广泛研究，芯片产业迫切需要具备相关知识的人才。RISC-V 是新兴的开源指令集架构，简洁易用，具备广阔的市场潜力，是理想的教学研究对象。采用敏捷开发语言 Chisel 设计了 4 级流水线的 RISC-V 处理器核。处理器核采用 AXI 接口，支持 IM 指令集和中断功能。该处理器核在基于 DiffTest 原理搭建仿真平台上进行了测试，并通过 FPGA 进行下板验证。验证结果表明处理器核能在 Nexys 4 开发板上运行 RT-Thread Nano 系统，频率为 50MHz，IPC 为 0.5。该研究成果可以为嵌入式设计和计算机体系结构相关实验的教学提供一定参考。

关键字 RISC-V，Chisel，处理器，FPGA，Verilator

Design of RISC-V Processor Based on Chisel

Guofeng Qin Tao Zhou

College of Electronics & Information Engineering of Tongji University
Shanghai 200092, China
gfqing@tongji.edu.cn

Abstract—With the rapid development of the Internet of Things and artificial intelligence technology, processor cores based on specific application scenarios have been widely studied, and the chip industry urgently needs talents with relevant knowledge. RISC-V is an emerging open-source instruction set architecture that is concise and easy to use, with broad market potential, and is an ideal teaching and research object. This paper completes the design of a 4-stage pipeline processor based on the agile design language Chisel. The designed processor core uses AXI interface, and support IM instruction set as well as interrupt functionality. This paper also uses Verilator open-source simulation software to establish a simulation framework based on the DiffTest principle for testing, and builds SoC based on the designed processor core on FPGA, runs operating system to verify the core design. The verification results show the designed processor core can run the RT-Thread Nano system on the Nexys 4 board, with a frequency of 50MHz. The IPC of the processor core is estimated to be approximately 0.5 based on the simulated waveform of the testing program. The research of this paper can provide some references for the teaching of embedded system design and computer architecture related experiments.

Keywords—RISC-V, chisel, processo, FPGA, verilator

1 引 言

RISC-V是由UC Berkeley大学提出的精简指令集架构，具备高可扩展性和优异的性能。与x86、ARM架构不同，RISC-V指令集是开源的且不带有沉重的历史包袱。新兴的RISC-V发展十分迅猛，得到了学术界和工业界的广泛关注。目前RISC-V国际基金会已拥有来自70个国家的3000多名会员。国内平头哥、赛昉、芯来等多家知名企业都推出了自己的商用RISC-V处理器。开源处理器国内主要有中科院计算所推出的高性能香山处理器芯来公司的E203处理器。教育教学方面主要有国科大和北京开源芯片研究院等机构共同推动的一生一芯计划，以及众多高校开展的基于RISC-V的教学实验[2-5]。

Chise 是由 UC Berkeley 大学设计的面向敏捷开发的语言。该语言由 Scala 构建，能够对硬件进行高度的抽象，进行函数式编程。有研究表明采用 Chisel 能够加快处理器的设计速度，并且编写的代码质量不弱于 Verilog 语言。Chisel 作为新兴的用于敏捷开发的硬件编程语言，具备许多优点和应用潜力。本文尝试采用 Chisel 语言设计 RISC-V 四级流水线处理器。设计中我们基于香山处理器设计所提的 DiffTest 原理[6]搭建仿真框架，进行处理器核测试。仿真框架中使用的参考模拟器 NEMU 来自南京大学。最后我们将利用 Xilinx 的 IP 核在 FPGA 上搭建简易 SoC，分析并设计了 RT-Thread Nano 在 SoC 上的启动，完成下板验证。

*基金资助：本文得到上海市教委 2022∽2024 年上海市高校本科重点教学改革项目资助，项目名称：计算机关键技术专业人才的系统能力培养核心课程链迭代重构。

**通讯作者：秦国锋 gfqing@tongji.edu.cn

2 RISC-V 处理器设计

本文设计的处理器目前实现了 RISC-V 的机器模式，指令集实现了 I、M。测试程序在 GCC 编译时不能产生浮点指令和原子指令。在编译 RISC-V 工具链时指定指令集配置来避免上述情况发生。本文设计的处理器结构如图 1 所示。

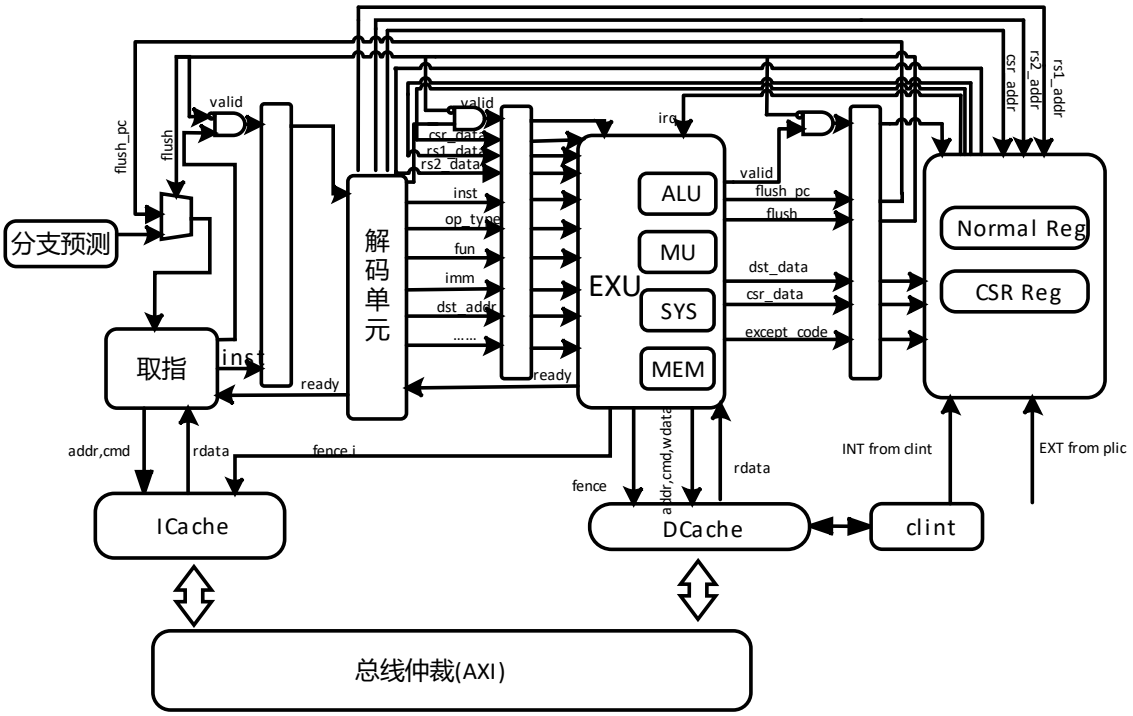

图 1 处理器的结构图

处理器的流水线结构分为取指、译码、执行、写回 4 级。取指单元根据指令地址从 ICache 中取出指令数据传递给下一级译码单元。指令地址产生受冲刷信号和分支预测器影响，如果冲刷信号存在，则指令地址为跳转地址，否则为分支预测器产生地址。译码单元主要将指令译码为所需要的操作码、指令类型、立即数、目的地址，并从寄存器中取出源操作数。译码单元会将这些信息传递给执行单元。执行单元会根据译码单元传递的指令类型选择对应的执行子模块进行执行。分支指令的跳转和中断异常也都由执行单元处理。中断的到来会改变特殊寄存器 mip 的相应位，执行单元将根据 mstatus、mie、mip 相应值来判断是否处理中断。写回单元主要负责将执行单元的结果写回寄存器。因为流水线中的写回阶段与译码阶段、执行阶段存在数据相关可能，因此要在译码和执行阶段进行数据相关性判断。另外通过仲裁方式处理 ICache 和 DCache 同时读写总线的问题。

2．1 流水线控制与模块接口

在 RTL 代码中会涉及许多模块与模块之间的连接。Verilog 是通过 wire 线将各模块的信号连接在一起的，该方法在连接大量信号线的时候比较繁琐且不便于代码的阅读。Chisel 能够将各个模块的接口信号进行封装，因此在进行信号连接的时候更为简单且便于代码的阅读。我们在进行模块连接的时候一般会采用 Chisel 的 Bundle 类对信号进行封装，如图 2 所示。

图 2 中我们展示译码模块和提交模块相关信号在最顶层中的连接，并且采用 Bundle 将多个信号打包成单一对象。例如 normal_rd 实际包含寄存器地址和寄存器数据信号，op_datas 包含多个解码相关的信号。处理器不会一直顺序执行，也不会总是每个周期执行完一条指令，这就需要控制流水线来保证处理器正常运行。流水线的控制主要涉及到流水线的暂停和冲刷，在 Chisel 中我们可以采用 Decoupled 接口进行相关逻辑的设计。UC Berkeley 大学的 Boom 处理器 **Error! Reference source not found.** 和中科院的高性能香山处理器都采用了 Decoupled 接口进行模块间的通信。Decoupled 会在包裹的传输信号上额外添加 valid-ready 握手信号，valid 是输出信号表示传输内容有效，ready 是输入信号表示相关设备已经准备好接受传输。模块与模块之间的握手过程如下图 3 所示。

译码模块

```
class Decode extends Module{
  val io = IO(new Bundle{
    ……
    val normal_rd = Flipped(new Normal_Read_IO)
    val csr_rd    = Flipped(new Csr_Read_IO)
    val op_datas  = Decoupled(new Op_Datas_IO)
    val flush     = Input(Bool())
  })
……
}
```

提交模块

```
class Commit extends Module{
  val io = IO(new Bundle{
    val normal_rd  = new Normal_Read_IO
    val normal_wb  = Flipped(new Normal_Wb_IO)
    val csr_rd     = new Csr_Read_IO
    val csr_wb     = Flipped(new CSR_Wb_IO)
    ……
})
……
}
```

连接

顶层模块

```
……
val decode   = Module(new Decode)
val commit   = Module(new Commit)
……
decode.io.flush         := execute.io.flush
decode.io.normal_rd <> commit.io.normal_rd
decode.io.csr_rd    <> commit.io.csr_rd
……
```

图 2 模块之间信号连接

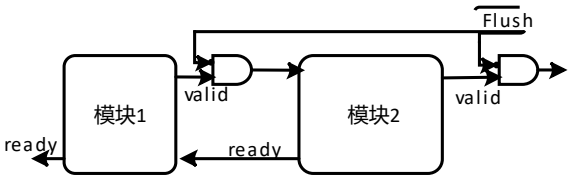

图 3 模块间的握手逻辑

流水线的暂停就是阻塞流水线的前级电路直到本级电路执行完毕。用图 3 中的电路表示，就是模块 2 不会发出 ready 信号，模块 1 因为接收不到 ready 信号会保持输出信号不变。同理因为模块 1 需要保持输出不变说明模块 1 操作也未完成，则模块 1 也不会发出 ready 信号，达到了层层阻塞的目的。流水线冲刷说明当前指令执行在错误路线上，需要将各级的输出 valid 信号与取反后的 flush 信号相与。下一次的取值地址由产生冲刷信号的单元传递给取指单元。流水线冲刷一般发生在指令跳转、中断异常、返回指令出现的情况下。

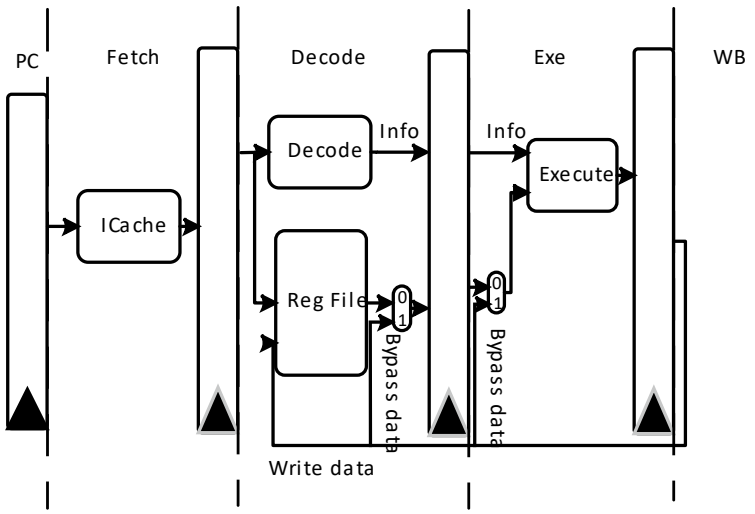

图 4 数据旁路结构

2．2 数据旁路

指令执行之间存在数据相关性，后续执行指令的源操作数据可能依赖于先前执行指令的结果。流水线中通过阻塞的方式来消除指令间的数据相关性，会引入较多空泡影响处理器性能。数据旁路直接从指令结果产生处将数据传递给需要的阶段，能够处理多数指令间的数据相关性问题，减少流水线空泡。本文处理器结构的数据旁路如图 4 所示。如果解码阶段读寄存器地址与写回阶段的地址相同，则选用写回阶段的数据，否则选用通用寄存器堆的数据。执行阶段类似，优先选择写回阶段的数据作为源操作数的数据。多周期执行的指令会在执行阶段产生阻塞，阻塞期间前级传递的数据会被忽略，从而保证旁路传递的数据是最新令结果。

2．3 两位饱和预测器标题

处理器运行过程中受分支指令和中断异常的影响，会打断 PC 寄存器自增过程，使得流水线前期阶段的数据无效，引入空泡。预测器能够在 PC 地址产生阶段预测指令的执行地址，减少流水线冲刷频率，提高效率。本文处理器采用两位数饱和预测器[7]，预测器结构如图 5 所示。分支指令的预测主要通过 PC 值，PC 的低位作为 PHT 和 BTB 的索引，PC 高位用于比较 BTB 对应项的 Tag。PHT 用于预测跳转方向，BTB 用于预测转移地址。PHT 中的每一项由两 bit 构成，2’b00 代表强不跳，2’b01 代表弱不跳，2’b10 代表弱跳转，2’b11 代表强跳转。当 PHT 预测为跳转，且 BTB 中能找到当前 PC 对应的转移地址 target_pc，则下一周期的 PC 值为 target_pc。函数返回指令的转移地址通过 RAS 预测。每当执行函数调用指令时，对应的 PC+4 值将压入 RAS 中。每当执行函数返回指令时，RAS 栈顶的值被弹出送给预测器。预测器中 PHT 和 BTB 的数据在写回阶段进行更新。

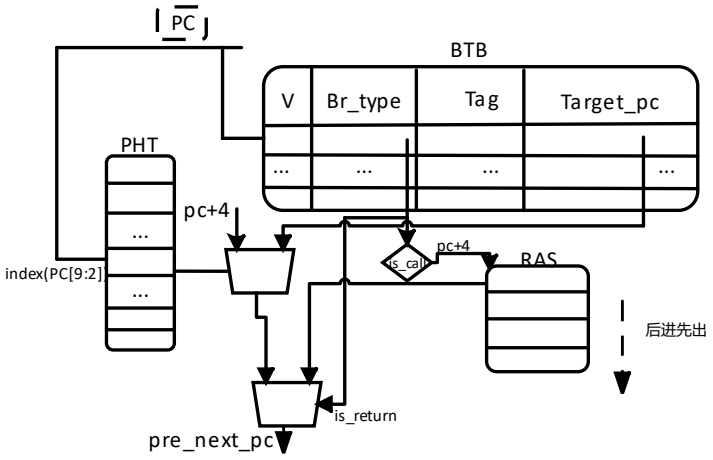

图 5 分支预测器结构

2．4 译码模块

指令在送入执行阶段之前需要进行译码。译码阶段主要负责解析指令的操作数、指令类型、指令操作信息等。解析操作数包括从通用寄存器、特殊寄存器中取值以及立即数的拼接。在 Verilog 中，我们需要先根据指令的操作码识别出指令的类型，再根据指令中的 fun 位识别出对应的执行功能。立即数的拼接也需要根据指令的类型进行手动拼接。使用 Verilog 编写译码过程会比较繁琐且不易于修改。在 Chisel 中有较为方便的处理方法，如下图 6 所示。在 Chisel 中，我们可以先编写一张 ISA 的 table，这张表定义了每条指令对应的译码信息。然后在译码模块中通过 Chisel 的 ListLookup 函数将指令的操作信息解析出来。这种方法在设计上更为直观，便于修改且不容易出错。

2．5 执行模块

典的 5 级流水线中会将访存单独划分一级，该方案可以复用执行阶段来计算存储地址，并且降低执行阶段的设计复杂度。本文设计的是 4 级流水线，出于简化控制逻辑的目的将访存合并到了执行阶段。实际上有不少处理器也将访存放在了执行阶段，如蜂鸟 E203、国科大的 NutShell 处理器。本文将执行模块划分为 4 个子模块分别是 ALU、MU、System、Mem。ALU 负责算术与逻辑计算，MU 负责乘除法计算，System 负责与特殊寄存器相关的系统计算，Mem 负责存储相关的计算。执行模块并不是总能在一个周期内执行完毕。

本文除法采用不恢复余数补码除法，一次运算需要 65 个周期。本文乘法采用基 4 Booth 算法，一次运算需要 32 个周期。存储需要与 Cache 和外设进行交互，其周期数不定。因此执行模块会在必要的情况下阻塞流水线并暂存信息直到得到计算结果。执行模块的控制逻辑如图 7 所示。

```
object ISA{
  def AND    = BitPat("b0000000_?????_?????_111_?????_0110011")
  ……
  val table = Array(
    //val opType :: exuType :: instType :: dest_is_reg :: rs1_is_reg :: rs2_is_reg
    AND  -> List(Op_type.op_alu,    ALUType.alu_and,    Inst_type.Type_R,Y,Y,Y),
    ……
  )
}
```

ListLookup根据inst值查找table表输出对应解码信号

“AND”代表and类型指令格式
“?”代表指令中无关位

```
class Decode extends Module{
  ……
  val opType :: exuType :: instType :: dest_is_reg :: rs1_is_reg :: rs2_is_reg :: Nil =
        ListLookup(inst,decodefault,ISA.table)
  ……
}
```

图 6 指令译码实现

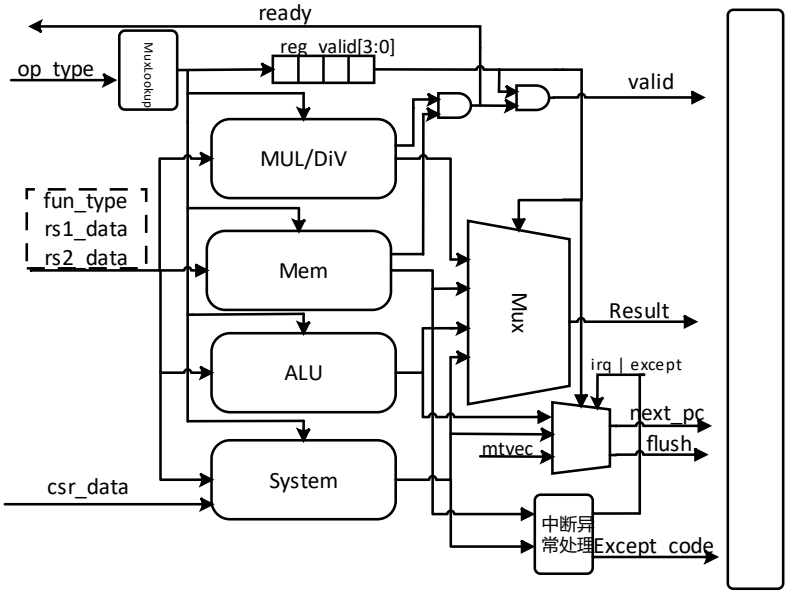

图 7 执行模块结构

执行阶段会根据指令类型信号 op_type 选择相应的子模块进行运算，并且将选择信号暂存到 reg_valid 中。由于执行模块不一定能够在一个周期内完成，需要借助 reg_valid 的值确定当前的那个子模块生效并将生效子模块的结果作为执行模块的输出。我们设计中采用 valid-ready 握手信号进行流水线的暂停。从图中我们可以看出，执行模块的 valid 信号由 reg_valid 与执行模块的 ready 信号相与产生。执行模块的

ready 信号由乘除法模块与存储模块的 ready 信号相与产生。这是因为执行模块只有乘除法和存储会造成流水线的暂停，当乘除法模块和存储模块的 ready 信号都为真时说明当前没有乘除法指令和存储指令需要执行，流水线正常流动。执行模块的 valid 信号只会在操作完成时产生，当 valid 为高则说明执行模块产生了有效输出。除 ALU 处理的跳转指令外，System 模块处理的 mret 指令以及中断异常也会引起流水线的冲刷。执行模块会对这些信号处理优先级进行排序，如果中断异常存在则跳转地址为特殊寄存器 mtvec 的值，如果中断异常不存在再根据 reg_valid 信号的值选择 ALU 模块或者 System 模块的 next_pc 输出作为下一条指令的地址。

2．6 clint 与 plic 模块

中断是处理器的重要组成部分，RISC-V 具有核内中断设备和外部中断设备。目前 RISC-V 定义的核内中断有 clint 和 aclint，核外中断有 plic 和 aplic。Aclint 兼容 clint，aplic 不兼容 plic。本文采用 clint 和 plic 作为处理器的中断处理设备，clint 的基地址是 0x2000_0000,plic 的基地址是 0xc00_0000。clint 寄存器的映射地址如表 1 所示。

表 1 核内中断 clint 寄存器地址映射

寄存器名	地址偏移	大小（Bits）
msip	0x0000	32
mtimecmp	0x4000	64
mtime	0xbff8	64

寄存器 msip 用于设置软件中断，只有最低位有效，写 0 代表清除软件中断，写 1 代表设置软件中断。寄存器 mtime 与 mtimecmp 用于设置定时器中断，当 mtime 值大于等于 mtimecmp 时产生定时器中断，否则定时器中断被清除。寄存器 msip、mtimecmp、mtime 都是可读写的，mtime 的值会随着时钟自动累加。Plic 寄存器的映射地址如表 2 所示。

在 plic 设计的时候，只用到了 2 个外部中断号，因此只需要有两个优先级寄存器 source_1_priority 和 source_2_priority。Source_0_priority 在 plic 中属于保留的寄存器，未被使用。Pending 用于暂存外部中断，当外部中断到来时，相应的中断暂存位会被置 1。本文只有 2 个外部中断，因此 peding 只需要关注低 3 位。Enable 用于设置外部中断的使能位与 pending 寄存器的暂存位一一对应。Threshold 用于设置优先级的阈值，只有优先级高于阈值的外部中断才会被响应。Claim 寄存器会保存当前响应的外部中断号，供处理器读取。当处理器写 claim 寄存器时意味着中断完成，plic 会根据写 claim 的值清除 pending 中对应的中断保留位。

表 2 核外中断 plic 寄存器映射地址

寄存器名	地址偏移	大小（Bits）
source_0_priority	0x000000	32
source_1_priority	0x000004	32
source_2_priority	0x000008	32
pending	0x001000	32
enable	0x002000	32
threshold	0x200000	32
claim	0x200004	32

RISC-V 机器模式下与中断相关的特殊寄存器有 mstatus、mie、mip、mtvec、mcause、mepc,如下图所示。中断信号由 mip 寄存器保存，mie 寄存器用于设置中断号使能。如果 mstatus 的全局中断位 mie 为 1，且 mip 寄存器与 mie 寄存器相与的结果不为 0，则说明有中断需要响应。中断响应时会将 mstatus 的 mpie 位置 1，mstatus 的 mie 位置 0，mcause 寄存器保存响应的中断号，mepc 寄存器保存当前指令的下一条地址，程序将会跳转到 mtvec 设置的中断程序处理地址运行。中断完成后将通过 mret 指令返回 mepc 寄存器所指的地址继续执行，同时 mstatus 的 mie 位将被恢复为 mstatus 的 mpie 位的值。需要注意的是在机器模式下 mip 寄存器中的 meie 位、mtie 位、msie 位是只读的，不能通过写 mip 寄存器进行清除，只能通过写 clint 和 plic 设备达到间接清除的目的。

2．7 总线仲裁

本文设计的处理器通过 AXI 总线与外围设备进行通信，AXI 分为写地址通道、写数据通道、写反馈通道、读地址通道、读反馈通道。ICache 和 DCache 都需要通过总线获得外围设备的数据，存在争用总线的情况。由于 ICache 只会进行读操作，ICache 和 DCache 的竞争只会发生在 AXI 的读通道上，DCache 的写过程能够直接与 AXI 的写通道相连。对于 AXI 读通道的争用，我们采用总线仲裁的方式进行处理，如下图 9 所示。

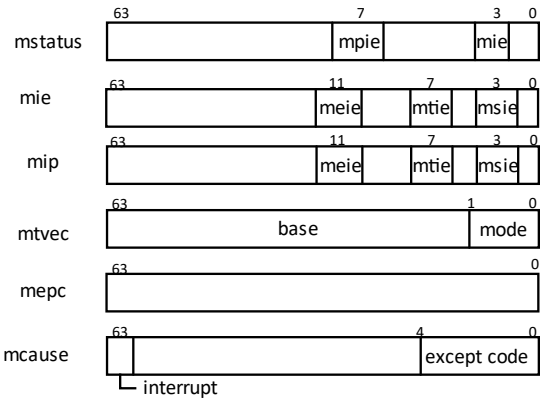

图 8 机器模式下中断处理相关特殊寄存器

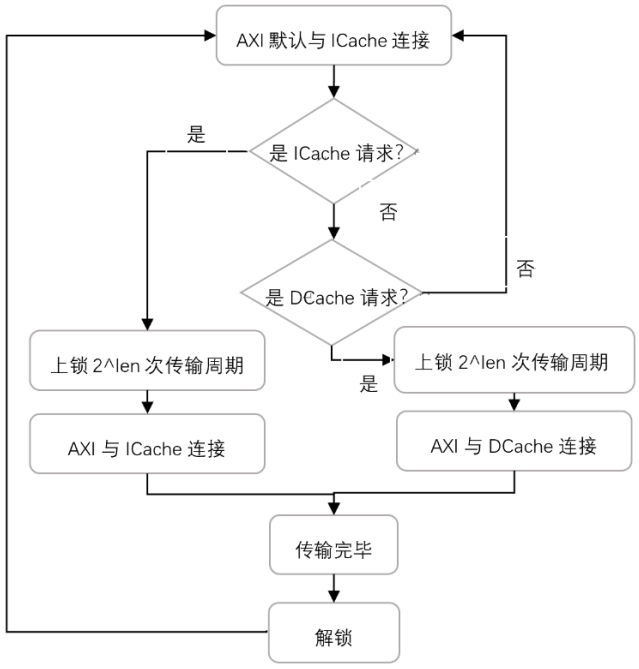

图 9 总线仲裁过程

AXI 的读通道默认与 ICache 相连，如果 ICache 和 DCache 同时发出读请求，仲裁模块会先处理 ICache 的请求。当仲裁模块接收到请求后，它会根据 Cache 发送的 len 值计算出需要上锁的传输次数并将 AXI 的读通道与相应的 Cache 连接。每当仲裁模块接收到一次 AXI 总线发送的 rready 信号，上锁的周期数会减一。上锁周期数为 0 代表传输结束，仲裁模块会进行解锁。锁定期间只有被锁定的 Cache 能够与 AXI 的读通道进行通信。

3 处理器核仿真

在处理器设计过程中仿真是一个重要的环节。通过仿真能够快速定位处理器核设计中的错误。仿真框架搭建部分主要是模拟处理器核运行时所需的外围设备，并提供调试工具。仿真测试部分主要是通过波形分析处理器核部件的执行状况。

3．1 仿真框架

本文目前基于中科院计算所香山处理器设计中所提的 DiffTest 原理搭建仿真测试框架[1]。其原理是比较模拟器与处理器 C++模型在测试程序中每条指令执行的寄存器结果。Verilator 具备将 RTL 代码转换为 C++模型的能力，能够快速的对 RTL 代码进行验证[8]。处理器模型由 Verilator 进行转化，模拟器使用的是南京大学的 NEMU，DiffTest 测试框架如图 10 所示。首先会将待测试程序加载入模拟器和处理器对应的 flash 中，并初始化模拟器的寄存器和 PC 状态使其与处理器一致。随后循环单步执行测试程序中的每条指令，并比较模拟器和处理器核中的寄存器结果。如果寄存器内容不同，仿真框架将停止指令执行并通过调试工具将寄存器、处理器状态、波形等信息输出。调试信息将辅助我们定位出错位置，排查错误原因。

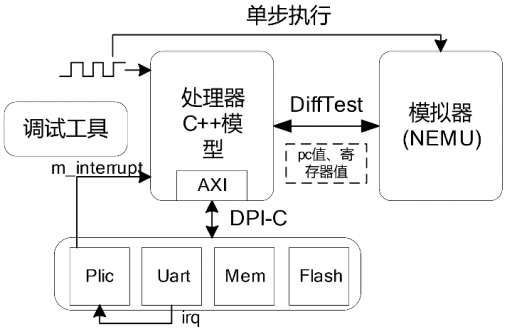

图 10 DiffTest 测试框架

框架中的调试工具移植于 NEMU，外围设备是对真实设备模拟的简化。处理器核通过 DPI-C 接口与 C 模型设备进行交互。处理器核不一定每周期执行一条指令，需要额外添加 commit 寄存器标记指令是否执行完成。Verilator 每检测到处理器核的 commit 置位，就会调用一次模拟器执行指令函数以达到两者之间的同步。

3．2 仿真测试

本文在设计中采用一生一芯的 cpu test 对单条指令的实现进行测试并通过 RT-Thread Nano 进行综合测试。图 11 展示的是加法测试程序的波形，图中 io_exuType 信号的值 0x3，0x22 分别代表 add 操作和 sub 操作，由写寄存器的结果可知处理器执行正确。

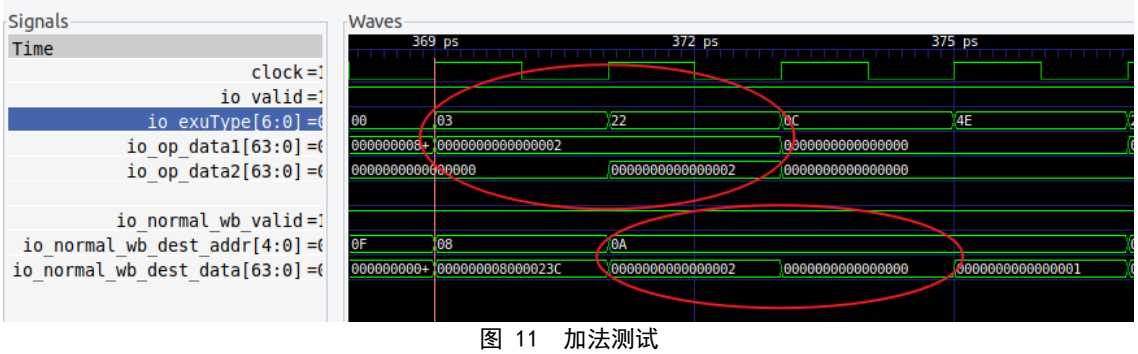

图 11 加法测试

图 12 中展示了取指信号和写通用寄存器信号，可以看出信号值在正常流动，指令可以正常提交。图中还是展示了 RISC-V 处理器性能测试相关的 mcycle 寄存器和 minstret 寄存器。寄存器 mcycle 代表执行的周期数，minstret 代表执行的指令数，可估算出处理器的 IPC 为 0.5。

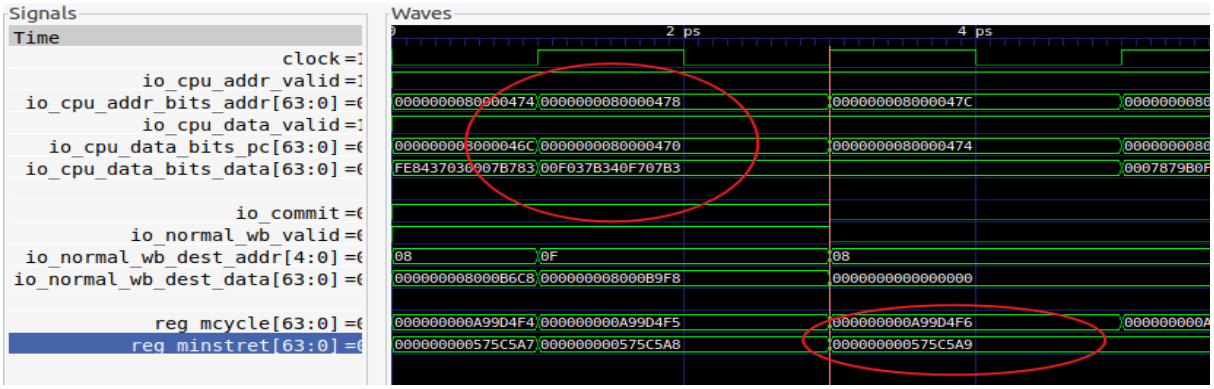

图 12 RT-Thread Nano 测试

4 FPGA 测试

采用软核和外设 IP 在 FPGA 上搭建 SoC 系统，具备灵活性大方便维护的特点[9]。其不仅可用于市场上的一些定制化需求，也可用于处理器原型的快速验证。目前仿真阶段未实现对外设的精确模拟，因此仿真阶段处理器核与外设的交互过程与真实情况存在些许差异。为了进一步验证处理器设计的正确性，我们搭建了简易的 SoC 进行 FPGA 原型验证，FPGA 采用的是 Nexys 4 开发板。测试程序是 RT-Thread Nano 系统，并通过串口进行人机交互。

4.1 SoC 结构

本文 SoC 如图 13 所示，整个 SoC 由处理器核、plic、DDR、Flash、UartLite、GPIO 构成。外围设备除 plic 和 Flash 由我们设计外，其余均采用 Xilinx 的 IP 核，构建起来十分方便。外围设备产生的中断将先经过外部中断平台 plic，再由 plic 转发给处理器核。UartLite 和 GPIO 采用的是数据总线为 32 位的 AXI-Lite 接口，不能与 AXI 直接相连。AXI-Lite 只支持单次传输，缺少 last 和 len 信号。本文设计 AXI to AXI-Lite 模块完成相应转换，写通道的 awlen 信号和 wlast 信号会被直接忽略，读通道的 rlast 信号将随着 rvalid 信号同时产生，读写数据根据地址低位进行偏移达到将 AXI 的 64 位数据总线上的数据放到 AXI-Lite 的 32 位数据总线上的目的。Flash 模块占用了 Nexys 4 开发板用于固化 FPGA 程序的 flash 芯片，测试程序可以通过下载线直接烧录到 flash 中。DDR 控制模块 MIG 的参考时钟采用的是 FPGA 的系统时钟为 100MHz，MIG 的输出时钟 ui clk 是该模块 AXI 接口交互时钟为 50MHz。时钟 ui clk 还会作为处理器核和其他外设的输入时钟，从而避开了对跨时钟域设计的考虑。

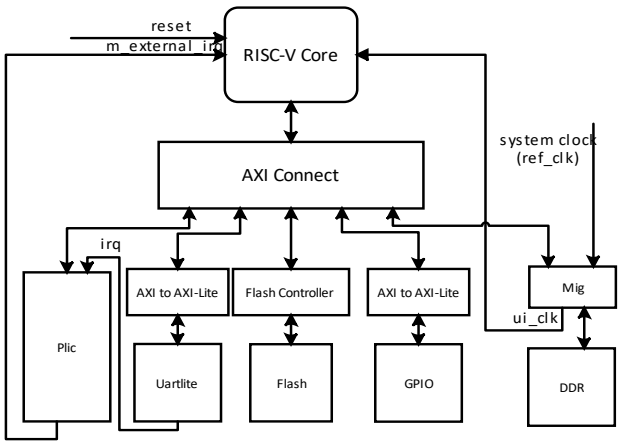

图 13 SoC 框架

AXI Connect 模块负责将处理器核的总线信息分发给不同外设，只有处理器核发送的总线地址信息符合外设的地址范围时，对应的外设才能与处理器核进行交互。总线的地址映射关系如表 3 所示，clint 位于处理器核内部不经过 AXI 总线进行访问，DDR 地址大于等于 0x8000_0000 被划分为内存，plic、UartLite、Flash、GPIO 的地址小于 0x8000_0000 被划分为 MMIO 设备。访问 MMIO 设备的数据不会被 Cache 暂存，且相应存储指令不会被推测执行。

表 3 总线地址映射关系

设备	地址范围	大小
plic	0xc00_0000~0xcff_ffff	16M
clint	0x200_0000~0x200_ffff	64K
UartLite	0x1000_0000~0x1000_0fff	4K
Flash	0x3000_0000~0x3000_ffff	64K
GPIO	0x4000_0000~0x4000_ffff	64K
DDR	0x8000_0000~0x87ff_ffff	128M

4．2 烧录与测试

RT-Thread Nano 的启动会涉及到多个外设的交互[18]，能够充分的验证处理器核运行的多种情况。测试程序 RT-Thread Nano 会先存放于 flash 中，然后通过加载程序搬运到内存，并最终在内存上运行。因此测试程序虽然先存放于 flash，但运行时的地址依然是基于内存。加载程序需要确定测试程序在 flash 和内存中的位置才能实现正确搬运。测试程序的位置信息会在链接时确定，如图 14 所示。链接脚本中定义的符号 xing_data_start、_text_start 等记录着测试程序在 flash 和 DDR 上地址，启动代码中会引用链接脚本的这些符号，值暂存于寄存器中并传递给加载程序使用。

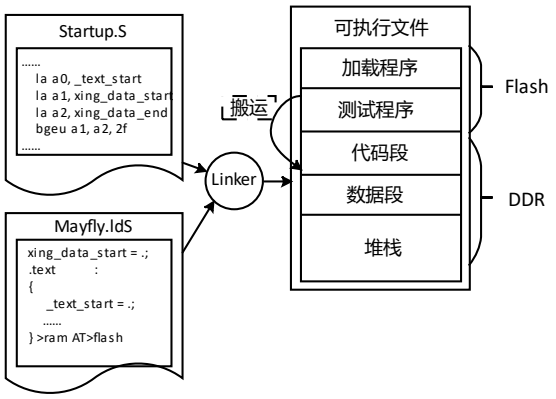

图 14 链接过程

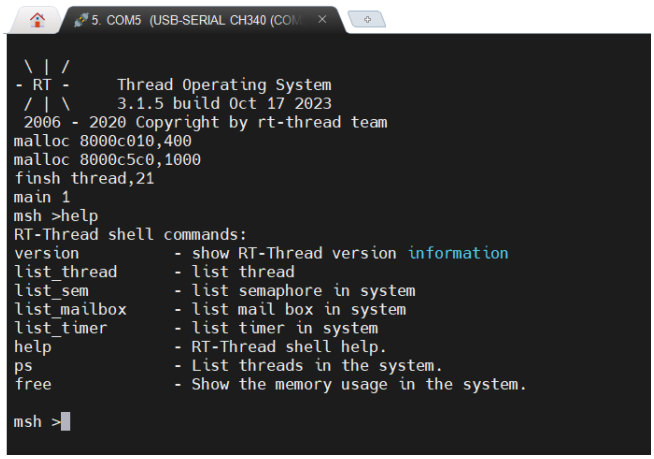

图 15 RT-Thread Nano 运行信息

本文主要介绍了采用 Chisel 进行设计的处理器结构，并通过 DiffTest 测试和 FPGA 对处理器核进行验证。处理器核的 IPC 为 0.5，在 Nexys 4 开发板上能稳定运行在 50MHz 频率。可用于嵌入式开发。本文内容涉及了处理器核设计，SoC 搭建以及 RTOS 系统的移植，具备连贯性，能够为计算机体系结构相关课程的实验设计提供参考。

后续计划充分利用 Nexys 4 资源添加 VGA 和 SD 卡外设，丰富 SoC 的应用场景和展示效果。还计划提供 AXI 接口模板，使学生能够自己设计组件。

参 考 文 献

[1] 王凯帆,徐易难,余子濠等.香山开源高性能 RISC-V 处理器设计与实现[J]. 计算机研究与发展,2023,60(03):476-493

[2] 陈志广，刘皓铧，卢宇彤.基于 RISC-V 的计算机组成原理实验教学改革与实践[J]. 计算机教育，2023, 338(2): 128-132.

[3] 冯建文.基于 RISC-V 架构的中断实验设计[J].实验室研究与探索，2022, 41(12): 34-38.

[4] 叶朝辉，华成英，张利伟，等.基于 FPGA 的 SOPC 电子系统设计实验研究[J].实验技术与管理，2018, 35(3): 161-164

[5] 秦国锋，李晨扬，林芃芃，等.重构计算机系统教学：从 MIPS 到 RISC-V 计算机原型系统实验[J].实验技术与管理，2022, 39(5):189-198,227.

[6] Celio C, Patterson DA, Asanovic K. The Berkeley out-of-order machine (BOOM): An industry-competitive, synthesizable, parameterized RISC-V processor [R]. EECS Department, University of California, Berkeley, 2015.

[7] Wang Huaqiang, Zhang Zifei, Zhang Linjuan, et al. OSCPU/NutShell:RISC-V SoC designed by students in UCAS GitHub repository[EB/OL]. [2022-12-28]. https://github.com/OSCPU/NutShell

[8] 孙恺,王田苗,魏洪兴等.嵌入式 CPU 软核综述[J].计算机工程,2006,(07):6-9.

[9] 陈瑞雪,王宜怀,王庭琛.实时操作系统 RT-Thread 启动流程剖析[J].现代电子技术,2022,45(12):36-42.