CoNM 开发者手册

Ver 0.8-draft

CoNM 介绍

概述

本说明所设计的处理器内核名为 CoNM（core of normal microarchitecture，规范架构核心）。该核心是一个使用 Verilog HDL 撰写，支持 RISC-V RV32I 指令集，采用经典四级流水线、经典静态分支预测、经典功能模块划分与经典微架构的处理器软核。其主要面向 RISC-V 指令集架构学习者与数字集成电路设计学习者设计，预期在教学中起到良好的范例作用。

CoNM 目前支持 RV32I 的 37 条指令，尚未支持 Zifencei 存储器读写屏蔽架构与 Zicsr 特权架构。

CoNM 以 Apache 2.0 协议维护在 GitHub[1]上。

指令集版本

RV32I Ver 2.1

The RISC-V Instruction Set Manual Volume I: Unprivileged ISA Document, Version 20191214-draft

代码层次

```
|--conm   //CoNM 根目录
    |--fpga   //FPGA 约束文件、仿真 testbench 源代码与初始化文件
        |--constrs   //FPGA 约束文件
        |--rtl   //CoNM 源代码
            |--core   //内核
            |--perips   //外设，目前仅包括了 IMEM 与 DMEM
            |--soc   //CoNM SoC 顶层代码，分为 ModelSim 测试版本（CoNM_soc_top）与
                //FPGA 版本（soc_top）
            |--utils   //标准 D 触发器模块
        |--tb   //模块 testbench 与 soc testbench，分为.verilog 文件读入版本（CoNM_soc_tb）
            //与手动单条指令测试版本（tb）
            |--module_tbs   //模块 testbench，目前大部分无法正常运行
        |--tests   //测试文件夹，包括测试记录（test.log）与 ModelSim 仿真脚本文件（tb1.do）
            |--isa   //rv32ui 定向指令测试集，引自 tinyriscv[2]
```

设计思想与代码规范

经典：从流水线到微架构，CoNM 都采用了较为经典的设计，使得仿真与验证结果更具有典型性。

[1] https://github.com/YJY1029/conm
[2] https://github.com/liangkangnan/tinyriscv

精简：CoNM 遵循了 RISC-V 与 RISC 精简的设计思想，严格控制使能信号数量，在保证功能分布明确的前提下尽可能地融合与简化了各个功能模块。

可拓展：CoNM 严格按照 RISC-V 的模块化标准设计，因而也保有了相当强的可拓展性，可以为持续性的更新提供保障。

（1） 优先使用纯组合逻辑电路而非时序电路。

（2） 尽可能使用 assign 语法代替 if 语句及 case 语句作为条件语句。

（3） 以 wire 型中间变量替代重复的条件判断。

（4） 使用规范的 DFF（D 触发器）与 MEM（存储器）模块例化级间锁存器、IMEM 及 DMEM。

（5） 尽量少使用使能信号。

环境与工具配置

开发工具环境：
Ubuntu 20.4

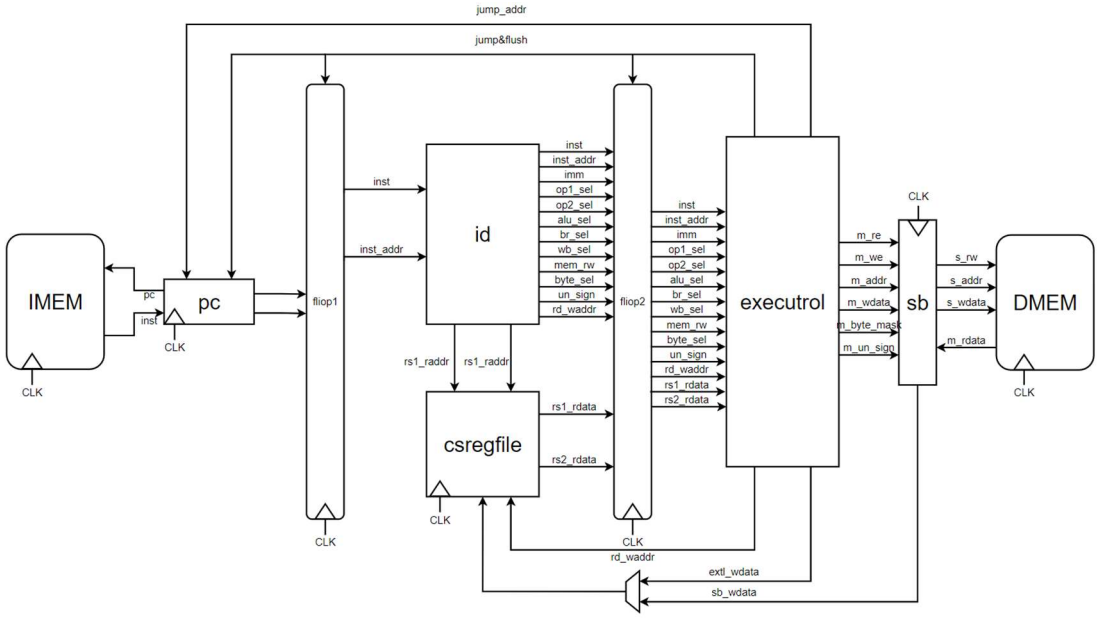
开发工具版本：
riscv32-unknown-elf-gcc version 10.2.0
Spike RISC-V ISA Simulator 1.0.1-dev

验证环境：
Microsoft Windows 10 家庭版

验证工具：
ModelSim 10.4 SE
Vivado 2020.2 Webpack

FPGA 板卡：
PYNQ-Z1

数据通路与基本机制



四级流水线分为取指、译码、执行与写回，包含的模块分别是 pc、id 与 csregfile、executrol、sb 与 csregfile。

采用默认分支失败的静态分支预测机制。

模块功能

程序计数器 pc

向 IMEM 请求指令内容并向后传递。此外，它会根据跳转信号置位与否从跳转地址取指，实现跳转操作。当全局复位信号使能，它会自程序起始地址重新开始取指。

译码器 id

纯组合逻辑的译码器。按照 opcode、funct3、funct7 的顺序一步步地向下直至得到具体的指令内容并处理立即数。

向 csregfile 发送源寄存器地址或 CSR 读地址请求寄存器数据。

向 executrol 发送的信号内容包括：inst 指令内容（后续暂未被使用）、inst_addr 当前指令地址、imm 立即数、op1_sel 操作数 1 选择、op2_sel 操作数 2 选择、alu_sel ALU 操作选择、br_sel 分支条件选择、wb_sel 写回目标选择、mem_rw 寄存器读写使能标志、byte_sel 比特片选、un_sign 符号位操作选择、rd_waddr 寄存器写地址。

这些信号能完整地描述任一 RV32I 指令操作。

寄存器堆 csregfile

功能包括寄存器读与寄存器写，分别工作在译码级与写回级。

对于读操作，它响应译码器发送来的读寄存器请求，并将对应的数据发送至级间锁存器；对于写操作，它接收寄存器写数据与写地址，并在时钟上升沿写入。

对于写数据，数据通路中的数据选择器实际上内含在了执行与控制模块中。如果写回级的指令是加载指令，那么 executrol 写回 0，反之 sb 写回 0。因此，csregfile 中只需要一个位或操作即可获得最终的写数据内容。

执行与控制模块 executrol

根据流水线前一级传递来的数据与操作信号，处理计算任务并发出控制信号。

执行部分运作核心是一个支持加法、减法、逻辑左移、逻辑右移、算术右移、位或、位与、位异或、小于时置位和无符号小于时置位的 ALU，其计算结果被存放于 alu_rslt 中，常用于存储器读写地址与跳转地址计算。

通过向 csregfile 发送 rd_waddr 目标寄存器写地址、extl_wdata 执行模块寄存器写数据，向总线发送 m_re 读使能、m_we 写使能、m_addr 存储器地址、m_wdata 存储器写数据、m_byte_mask 比特掩码、m_un_sign 无符号读标志，分别实现了对寄存器与存储器的写回级控制。

通过发送 extl_jump 跳转标志与 extl_jump_addr 跳转地址，实现了对分支与跳转的控制。

级间锁存器 fliop1 与 fliop2

通过大量例化可配置数据长度，复位时输出特定数据的标准 D 触发器模块实现。

存储流水线上一级传来的信号，在下一个时钟上升沿传递至下一级，实现流水线分割。

如果全局复位信号或跳转信号使能，其中数据会被替换为 NOP 对应的内容，实现流水线的冲刷。

总线 sb

sb 是一个多主单从的总线，当前仅用于 CoNM 与 DMEM 间的通讯。

目前 sb 分为 ModelSim 版本（sb）与 Vivado 版本（sb_vvd），对应不同的存储器接口。

默认情况下，sb 总是将读写数据与地址初始化为 0，避免对指令执行造成干扰。

与一般的存储器读操作不同，总线读数据直接连接到寄存器堆，使得内核读写路径是单向的，优化写回级时序。sb 具有一定的数据处理能力，可以根据符号、字节及小端格式的要求进行调整，使数据可以直接被目标模块使用。

外设与 SoC 设计

本项目含有指令存储器模块 imem 与数据存储器模块 dmem，运作原理与 csregfile 类似。imem 宽度为 32 位，深度为 4096，被读取时将地址逻辑右移 2 位，保证字对齐。dmem 模块深度是 imem 的两倍，采用字节对齐方式，读写时并不会除去最低 2 位。

CoNM_soc_top 采用哈佛结构设计，将 IMEM 与 DMEM 分开例化实现。前者的读使能信号总是使能，并且写数据端口悬空；后者通过 sb 与 CoNM 核心相连，而 sb 的另外一个主机端口全部悬空。

soc_top 中的数据存储器调用 Block RAM Generator IP 核进行了代替。

使用方法

ModelSim 仿真

ModelSim 仿真使用\conm\tb\CoNM_soc_tb 进行，这是一个行为部分包含了程序读入、测试进行与测试超时三个 initial 块的 testbench，实例化了 CoNM_soc_top 模块，时钟周期为 20ns。

对应 rv32ui 例程，该 testbench 引出了 x3(gp)作为 case 编号标志，x26(s10)作为测试结束标志，x27(s11)作为测试成功标志。

当测试成功时，transcript 中会输出测试成功提示、当前测试指令与运行时间；测试失败时，输出的变为测试成功提示、当前测试指令与失败现场内容。

通过修改 testbench 首行 CASE 宏定义修改为/conm/tests/isa/路径下的任一.verilog 文件，可以进行不同指令的定向测试。

简便起见，读者可以将/conm/tests/下的 tb1.do 文件复制至 ModelSim 项目根目录后，于 transcript 中输入 do tb1.do 即可获得当前指令测试的结果与经过笔者配置过后的波形查看窗

口。

FPGA 验证

FPGA 验证采用 PYNQ-Z1 开发板进行，目前可以完整运行加载与存储指令外的所有 rv32ui 测试。

Vivado 项目中包括的文件应当有：defines.v、pc.v、fliop1.v、id.v、csregfile.v、fliop2.v、executrol.v、CoNM.v、sb_vvd.v、soc_top.v，即除去 sb.v、CoNM_soc_top、imem.v、dmem.v 的所有\conm\rtl 路径下文件。如果需要在 Xsim 中进行仿真，则还需要添加 vvd_tb.v。

随后，参照原 IMEM 与 DMEM 规格，调用 Block RAM Generator IP 核生成两个存储器模块。

其中，IMEM 重命名为 imem_gen，单口 ROM，采用 32 位地址接口，可以添加至多 1 个额外的寄存器级，并且需要在此处指定.coe 初始化文件。若干初始化文件可以在\conm\fpga 路径下找到。

DMEM 重命名为 dmem_gen，单口 RAM，采用 32 位地址接口，可以添加至多 1 个额外的寄存器级，并且设置为读优先。

要实现时钟域分割以消除分支损耗，需要调用 clocking wizard IP 核，重命名为 clk_wiz 并生成 50MHz 衍生时钟，输入为板载 125MHz 时钟，输出端口名为 clk_50。

完成 Vivado 项目后，即可生成比特流文件并写入板卡。

以 PYNQ-Z1 开发板为例，板卡上 SW0 是全局复位开关；LD3 是处理器运行指示灯，当 rst 为高电平，即复位信号不使能时，处理器开始运行，LD3 亮起；LD0 是程序运行结束指示灯，通过判定 s10 寄存器是否为 1 实现，具体而言，它在复位时保持常亮，程序运行时熄灭，程序运行结束后重新亮起（由于 CoNM 运行频率较高，实际上无法观察到 LD0 熄灭）；LD2 是 BUG 指示灯，通过判定 s10 寄存器是否为 1 实现，具体而言，它在复位时保持常亮，在程序运行开始后熄灭，如果程序运行失败，则会亮起。

要运行不同的 rv32ui 测试，需要修改 IMEM 中的初始化文件，并重新生成比特流文件。