

基于 SystemC 的周期精确级 DSP 处理器建模

何卫强, 杨 靓, 卢 强

(西安微电子技术研究所, 陕西 西安 710054)

摘 要: 针对高级语言做处理器建模在模型精度方面的不足, 本文探讨了一种基于 SystemC 的周期精确级 DSP 处理器建模方法. 在分析各流水段功能的基础上, 结合 SystemC 的语言结构特点, 对流水级内各功能模块进行了抽象建模. 该模型能够精确地模拟处理器指令的执行情况, 对软件算法的设计优化和处理器微结构的探索具有一定的参考价值.

关键词: 周期精确; SystemC; 处理器建模

中图分类号: TP302

文献标识码: A

文章编号: 1000-7180(2013)04-0107-04

Cycle-Accurate DSP Processor Modeling with SystemC

HE Wei-qiang, YANG Liang, LU Qiang

(Xi'an Microelectronic Technology Institute, Xi'an 710054, China)

Abstract: With respect to the accurate shortage of modeling CPU with high-level language, a cycle accurate DSP processor modeling method with SystemC is explored in this paper. Based on the function analysis of pipeline's each stage, and combining the characteristic of SystemC, pipeline's function modules are modeled. This model can accurately simulate the implementation of the processor instruction, which has a certain reference value for the optimization of software algorithms and the exploration of processor micro-structure.

Key words: cycle accurate; SystemC; processor modeling

1 引言

直接采用硬件描述语言对电路进行 RTL 级 (Register Transfer Level) 设计, 不仅使设计工作繁重耗时^[1], 而且如果发现综合后的电路不满足时序要求, 就需要从结构层次去更改电路设计, 这一反复验证修改的过程延长了芯片设计的周期, 增加了芯片设计的成本, 所以当前处理器微结构的设计需要借助处理器的高级模型来进行分析和优化.

SystemC 作为一种建模语言, 能够很好地支持对硬件电路的描述, 已经受到处理器结构研究者的广泛关注^[2-4]. SystemC 是一种 IEEE 标准的系统级设计语言, 能够在处理器设计不同抽象层次 (系统级、行为级、RTL 级) 对处理器进行建模. 严格地讲, SystemC 不是一种新的语言, 其本质是在 C++ 语言的基础上增加了硬件类库和仿真库, 将并发、时

钟、端口等概念引入了 C++, 使其可以对硬件进行建模描述^[5-6].

2 DSP 处理器的建模

文中基于 SystemC 实现了图 1 所示的周期精确级的定点 DSP 处理器内核模型. 该 DSP 内核采用自主设计的指令集, 是一款 32 位字长的四级流水 RISC 处理器, 该内核包括一个片上程序存储器 (Program Memory, PM) 和两个片上数据存储器 (Data Memory, DM). 指令集包括 55 条基本指令, 分为寄存器赋值指令、存储器赋值指令、算术逻辑指令、控制类指令 4 大类. 所建模型按照流水周期 (一个时钟周期) 划分内核模块, 能够精确地模拟处理器各部分在每个时钟周期的行为, 该模型主要用于软件算法的设计优化和处理器微结构的研究.

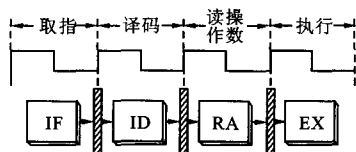


图1 DSP处理器流水模型

2.1 总体设计

本文在一般硬件设计方法的基础上,结合 SystemC 的语言结构特点,提出了一种针对处理器建模的设计文件结构,如图 2 所示.按功能和用途可划分为:Stimulus、DUT、Monitor 三部分.其中 Stimulus 主要用于生成测试用指令序列,该部分可以采用标准的 C++ 语言完成图形化界面,实现指令集仿真器;DUT 为所设计的 CPU 抽象模型,用于描述 CPU 的行为功能;Monitor 主要用于监视指令执行情况,控制仿真时间,打印执行结果信息等. Main 函数模块主要是负责完成模块例化,控制整个程序运行和停止.正是由于 SystemC 对 C++ 的继承与扩展保证了二者间的无开销连接.

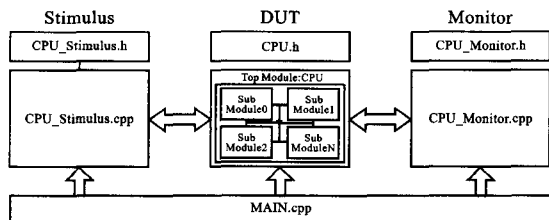


图2 处理器建模设计文件结构

所有内核模型模块分为组合逻辑模块和时序逻辑模块两类.组合逻辑模块,即其输出仅与输入信号有关,当输入发生变化时,其输出就立即更新,组合逻辑的实现可以看成是函数的实现,输入输出采用端口(sc_port)类型,且所有的输入都要作为敏感信

号添加到敏感信号列表中.

时序逻辑模块,其输出是受到时钟信号触发,只有当时钟上升沿这一事件发生时,输出值才会更新,否则输出一直保持上一次的输出结果.在时序逻辑模块中,对于时钟信号模型的建立可以在所设计的模块中直接加入 SystemC 提供的 sc_clock 类,配置相应的参数完成,该类模块的敏感信号仅为时钟信号.

SystemC 提供了非常丰富的数据类型^[5-6],诸如 sc_bit(2 值单比特数据类型),sc_logic(4 值单比特数据类型),sc_uint(无符号整型),sc_lv(任意宽度的 4 值比特向量数据类型),sc_fixed(模板类有符号定点数据类型)等.对于处理器的建模,从可综合的角度考虑,则应选取 sc_bit、sc_logic、sc_lv 等相关类型,从仿真速度的角度考虑则应尽可能选取 C++ 原有的标准数据类型,如 bool、sc_int 等,来降低仿真时间.本文设计主要用 SystemC 语言实现指令集仿真器,因此选取 sc_uint、bool 为相应信号的数据类型,来提高仿真速度.

SystemC 引入 MODULE(模块)的概念,能够保证对所建模系统的层次化描述.本文正是基于这一特点,将处理器内核抽象为多个的功能模块,每一个模块看做一个硬件电路. SystemC 特殊的 Simulation Kernel,保证了这些功能模块的并行执行.

2.2 流水段设计

流水线设计能够保证每个时钟周期流出一条指令,从而提高系统的吞吐率^[7].对于单周期指令,每个流水段只占用一个时钟周期.本文所设计的周期精确级模型的关键在于模拟每个流水段不同模块的行为.图 3 为所设计的 DSP 内核 SystemC 模型及各模块连接关系.

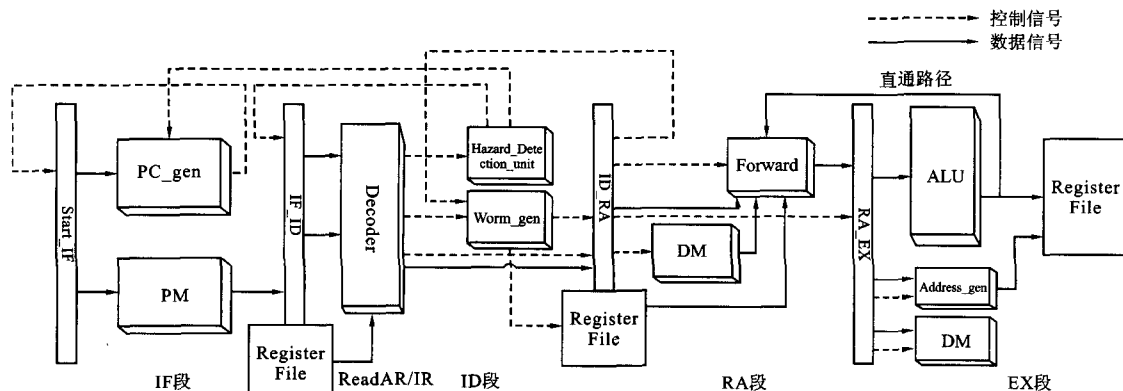


图3 DSP内核 SystemC 模型

取指级 (Instruction Fetch, IF), 主要完成 PC 的自加以及发生相关冲突时控制 PC 的暂停或对

PC 的重新计算,从程序存储器中取出指令.根据取指级功能可以将该级抽象为两个模块,PC_gen 模块

和 PM 模块. 对于 PC_gen 模块, 由其功能可以确定该模块的输入 sc_in 为当前 PC 以及来自译码级的相关控制信号, 输出 sc_out 为更新后 PC. PC 的更新由函数 pc_gen() 实现, 该函数中通过调用库函数 read() 和 write() 来实现对输入端口的读和对输出端口的写. PM 模块中采用数组类型完成对存储体的建模, 由于 PM 只存在读操作, 因此该模块的输入为当前 PC 值, 输出为被取出的指令, 将其设计为组合逻辑, 在不影响功能和时序的基础上, 简化了模型结构. 由图 3 可以看出这两个模块在 IF 级内是并行执行.

译码级 (Instruction Decode, ID), 主要用于生成寄存器/存储器读写使能信号、寄存器/存储器地址信号、多路选择器控制信号、立即数、运算模式信号等. 根据所建模的 DSP 的数据/控制通路, 将译码级功能抽象为 Decoder、HDU、Worm_gen 三个模块, 分别用来实现译码, 相关检测、直通路径控制信号生成的功能. 其中译码功能由 decode() 函数实现, 采用定长编码指令集的 CPU 利用选择语句来生成的. HDU 和 Worm_gen 模块是整个处理器建模的重点, 需要考虑所有的相关冲突以及解决方法. 如图 3 所示, Decoder 模块生成的部分控制信号作为 HDU 和 Worm_gen 模块的输入 (敏感信号), 这就表明了三者在 ID 级的执行存在先后顺序.

读操作数级 (Read Access), 主要是获得源操作数, 源操作数可能来源于寄存器文件, 数据存储器, 立即数, 也可能是由上一条指令的 EX 级直接得到的数据 (旁路逻辑). 因此将该级的功能抽象为 Forward 和 DM 两个模块. Forward 模块的输入 sc_in 包括 4 类数据信号以及由 Worm_gen 模块产生的控制信号, 通过所编写的 get_value() 函数实现数据选择. DM 模型与 PM 模型相比, 增加了读写使能信号, 地址信号以及读写函数. 如图 3 所示, 这两个模块在 RA 这个时钟周期的执行是存在先后顺序的.

执行级 (Execution) 完成算术运算、逻辑运算、寄存器、存储器赋值等操作. 根据其功能将其划分为 ALU 和 Address_gen 两个模块. ALU 模块, 用于完成执行级的主要操作, 其输入为 A、B 操作数, 计算模式信号等, 输出为计算结果数据以及需要写入目的寄存器的数据. Address_gen 模块主要是针对存储器赋值类指令对于地址寄存器和索引寄存器的更新.

以上所设计的各流水级中的模块均为组合逻辑模块, 任何一个输入的改变都会使输出立即发生变化. 要实现完整的周期精确级仿真模型, 就需要在每

一段中插入时序逻辑的级间寄存器模块, 这些模块的输入为上一流水级组合逻辑模块的输出, 输出为下一流水级组合逻辑模块的输入. 时序逻辑模块受到复位、全局运行使能信号的控制, 只在时钟上升沿更新输出数据, 保证整个模型按照时钟周期触发. 寄存器文件 (Register File) 也是时序逻辑模块, 保证在每个时钟周期可以观测到输入输出.

3 仿真波形分析

对内核建模完成后, 首先在 VC6.0 环境下调试, 并生成波形文件, 然后使用 Active HDL 7.3 仿真软件观测波形, 采用了不同的指令组合对所建模型进行测试, 验证了处理器模型中关键结构的功能和时序的正确性. 仿真时设定时钟周期为 10 ns, 下面对几种特殊情况进行分析:

输入指令为 LDI 0x55 R0(0x55→R0), LDI 0x5A R1(0x5A→R1), ADDI R0 R1(R0+R1→R1), LDI R1 R0(R1→R0), 对应的机器码为: 0x14000055, 0x1420005A, 0x80001040, 0x10008000, 指令执行仿真波形如图 4(a) 所示. 4 条指令执行完成共花费了 70 ns (7 个时钟周期), 由图可以看出每个时钟周期各信号及寄存器信息均正确, 指令得到正确执行, 流水线未发生停顿, 说明所设计的旁路逻辑有效解决了 RAW (Read after Write) 相关.

输入指令为 LDI 0x01 R0(0x01→R0, 第 5 条指令), LDI R0 PC(R0→PC, 第 6 条指令), 对应的机器码为: 0x14000001, 0x128C0000. 指令执行仿真波形如图 4(b) 所示. 由仿真图可以看出在第 70 ns 时控制相关信号、排空信号被拉高, 在下一个周期第 80 ns 时 (更新 PC 指令译码后) PC 被更新为 0x01, 紧接着的指令 0x0F0F0F→R0 由于存在控制相关, 只被取指, 未被译码, 在第 90 ns 时地址为 0x01 的指令 0x14000055 被取指, 说明了该种结构下发生控制相关的开销为 2 个时钟周期的. 指令执行结果正确, 控制相关处理正确.

输入指令为 LDI *AR0++(1) R0, 此条指令是将地址寄存器 AR0 中的值作为数据存储器地址, 将取出的值赋到 R0 中, 执行完后 AR0 的值自增 1. AR0 的初始值为 0, MEM[0] 的初始值为 0xFFFF. 对应机器码为: 0x201E4000, 指令执行仿真波形如图 4(c) 所示. 该条指令执行完花费了 40 ns, 在 50 ns 时通用寄存器 UR0 被更新为 0xffff 和地址寄存器 AR0 被更新为 0x01, 表明该存储器赋值指令得到正确执行.

