面向教学的 RISC_SPM 处理器设计与验证

解鹏越[1] 卫建华[1] 郝子坤[1] 罗鑫迪[1]
XIE Pengyue WEI Jianhua HAO Zikun LUO Xindi

摘 要 针对当前国内微处理器实验教学的需求，文章设计了一款面向教学用途的 RISC_SPM 微处理器，优化资源利用和执行效率。处理器采用内部存储器布局，资源占用少，执行速度快。通过状态机控制方法，使得运行过程更加直观清晰，克服了传统实验中灵活性不足的问题，为实验者提供了创新空间。设计过程中使用 Verilog 语言，支持多种寻址方式，具备完善的指令系统，能够执行一般微处理器的功能操作，满足实验教学中的实际需求并提升学生的实践能力和创新意识，为科研工作奠定了良好的基础。

关键词 微处理器；精简指令集；现场可编程逻辑门阵列；存储器位于芯片内部；FPGA

doi：10.3969/j.issn.1672-9528.2025.01.004

0 引言

当前，各高校在计算机和电子相关课程中，已将微处理器设计列为重要学习内容之一，鼓励学生发挥创新能力。然而，国内在该课程的实践教学环节，多采用拨动开关或连接模块的简洁实验方法。这种方式虽操作简便，但在一定程度上，限制了学生的学习深度和创造空间，导致整体设计思想相对滞后。鉴于此，选择 FPGA 芯片进行设计，具备系统功能灵活性和低成本优势。具体而言，FPGA 具备可重构性，类似软件核心，可根据不同需求调整内部结构，提供更多实验可能性 [1]。而采用微程序设计方法，微处理器内部数据流清晰可见，方便学生进行功能扩展和改造，为其提供更好的创新空间 [2]。

1 精简指令集 RISC 处理器设计基础

在微处理器设计过程中，首先需要确定其控制管理模式，该处理器控制管理模式如图 1 所示。

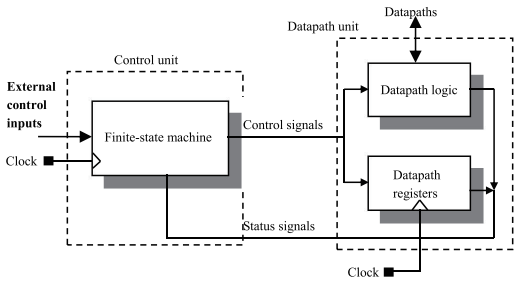

图 1 控制管理模式 FSMD

从图 1 中不难发现，为实现高效且精准的控制逻辑，选择有限状态机 FSM 作为控制器（Controller）。（1）实现逻辑称为 Datapath[3]；（2）实现逻辑 Datapath 接受控制器的指挥（接受命令信号 Control Signals），执行自动机动作；（3）实现逻辑 Datapath 同时向控制器反馈当前状态信号；（4）控制器（Controller）根据外部信息 input 和 Datapath 的反馈信息，及特定算法，发出特定命令。

2 RISC_SPM（Store-Program Machine）设计

2.1 理器顶层框图

处理器顶层框图如图 2 所示，该 8 位处理器具有加、减、与、非的无符号运算功能，并支持无条件转移指令，采用间接寻址方式。同时，它具有零标志转移指令，以及 4 个内部寄存器（R0、R1、R2、R3），可进行内部寄存器的寻址，寻址范围为 2。此外，其内部存储器 SPM[4] 位于芯片（FPGA）内部，宽度为 8，深度为 256，同时支持中断操作。

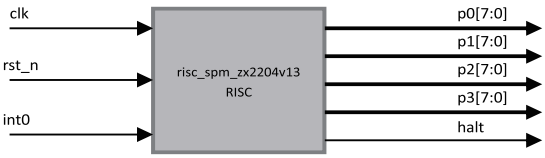

图 2 处理器顶层框图

2.2 指令集

2.2.1 指令格式

将寻址内部寄存器的指令称为短指（8 位），如表 1 所示，将寻址内部存储器的指令称为长指（16 位），如表 2 所示。

表 1 短指

Opcode				Source		Destination	
0	0	1	0	0	1	1	0

表 2 长指

Opcode				Source		Destination	
0	0	1	0	0	1	×	×
Address							
0	0	0	1	1	1	0	1

1. 西安工程大学电子信息学院 陕西西安 710600

2.2.2 指令码表（指令集）

指令码表[5-6]如表 3 所示，对表中指令作简单介绍。

表 3 指令集码表

指令	指令字				动作 Action
	操作码 opcode	源操作数 source	目标操作数 dest	地址码 addr	
NOP	0000	X	X	—	
ADD	0001	src	dest	—	dest+src=>dest
SUB	0010	src	dest	—	dest-src=>dest
AND	0011	src	dest	—	dest&src=>dest
NOT	0100	src	dest	—	~src=>dest
INC*	0101	src	dest	—	src+1=>dest
DEC*	0110	src	dest	—	src-1=>dest
IMM*	0111	X	dest	addr	addr=>dest
RD	1000	X	dest	addr	mem[addr]=>dest
WR	1001	src	X	addr	src=>mem[addr]
OUT*	1010	src	port	—	src=>port
BR	1011	X	X	addr	[addr]=>PC
BRZ	1100	X	X	addr	if z_flag [addr]=>PC else NOP
RTI	1101	X	X		
HLT	1111	X	X	—	

（1）NOP 指令：空操作指令，短指，操作码为 0000。

（2）ADD 指令：加指令，短指，操作码为 0001，执行的动作为将源寄存器中数据和目标寄存器中数据相加，并将结果送至目标寄存器。

（3）SUB 指令：减指令，短指，操作码为 0010，执行的动作为目标寄存器中数据减去源寄存器中数据，并将结果送至目标寄存器。

（4）AND 指令：与指令，短指，操作码为 0011，执行的动作为将目标寄存器中数据和源寄存器中数据相与，并将结果送至目标寄存器。

（5）NOT 指令：非指令，短指，操作码为 0100，执行的动作为将源寄存器中数据取非，并将结果送至目标寄存器。

（6）INC* 指令：增指令，短指，操作码为 0101，执行的动作为将源寄存器中的数据减 1，并将结果送至目标寄存器。

（7）DEC* 指令：减指令，短指，操作码为 0110，执行的动作为将源寄存器中的数据加 1，并将结果送至目标寄存器。

（8）IMM* 指令：立即数指令，长指，操作码为 0111，执行的动作为将地址码作为立即数，并将立即数送至目标寄存器。

（9）RD 指令：读指令，长指，操作码为 1000，执行的动作为将存储器中地址码所代表地址的数据读出来，并将数据送至目标寄存器。

（10）WR 指令：写指令，长指，操作码为 1001，执行的动作为将源寄存器中的数据，写入存储器中地址码所对应的地址。

（11）OUT* 指令：输出指令，短指，操作码为 1010，执行的动作为将源寄存器中的数据，送至目标端口。

（12）BR 指令：无条件转移指令，长指，操作码为 1011，执行的动作为将地址码作为新的 PC 地址发送给 PC，这里的地址为间接地址。

（13）BRZ 指令：条件转移指令，长指，操作码为 1100，用于当零标志寄存器有效时才进行跳转。

（14）RTI 指令：中断返回指令，中断完成后返回此指令，表示中断结束，继续执行中断前的指令。

（15）HLT 指令：停机指令，短指，操作码为 1111，表示暂停 CPU 的执行，直到中断或复位信号被触发。

2.3 处理器架构

顶层架构如图 3 所示，对各指令的路径作详细分析介绍[7]。

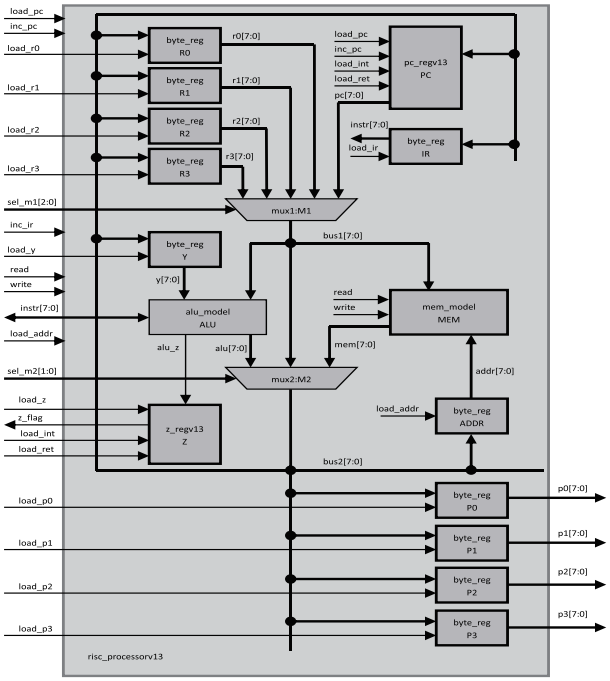

图 3 处理器架构图

（1）取指过程：第 0 拍控制器通过 sel_m1 选择 pc、通过 sel_m2 选择 bus1，发出 load_addr；第 1 拍控制器通过 load_addr，使 ADDR 寄存器输出一个等于 bus2 的地址 addr，发出 read；第 2 拍控制器通过 read 信号，将存储器（MEM）对应 addr 上的数据 mem 读出来，并发出 sel_m2、load_ir；第 3 拍控制器再通过 sel_m2 将 mem 送到 bus2 上，通过 load_ir，使指令寄存器 IR 输出等于 bus2 的指令 instruction。

（2）运算指令分析：第 4 拍控制器通过 sel_m1 选择 src 所代表寄存器，将寄存器中数据作为源操作数通过 sel_m2 送至 bus2，并发出 load_y 信号；第 5 拍 Y 寄存器收到 load_y 信号发出数据 y，送至 ALU，第 5 拍控制器通过 sel_m1 选择 dest 所代表寄存器，将寄存器中数据作为目标操作数送至 ALU，并发出 load_z 信号；第 6 拍 Z 寄存器若收到 load_z 信号则输出 z_flag 信号，将 ALU 运算结果通过 load_ri 送至目

标寄存器。

（3）立即数指令分析：第 4 拍控制器通过 sel_m1 选择 pc，通过 sel_m2 选择 bus1，并发出 load_addr；第 5 拍地址寄存器收到 load_addr 发出 addr，并发出 read 信号；第 6 拍控制器通过 read 信号，将存储器（MEM）对应 addr 上的数据 mem 读出来，并发出 sel_m2 和 load_ri 信号；第 7 拍通过 load_ri 信号送至目标寄存器。

（4）读访问指令分析：若前 3 拍取指为读指令，第 4 拍控制器通过 sel_m1 选择 pc，通过 sel_m2 选择 bus1，并发出 load_addr；第 5 拍地址寄存器收到 load_addr 发出 addr，并发出 read 信号；第 6 拍控制器通过 read 信号，将存储器（MEM）对应 addr 上的数据 mem 读出来，送至 AADR，发出 load_addr；第 7 拍通过 load_addr 输出数据地址，发出 read 信号；第 8 拍通过 read 和数据地址读出 mem 中的数据；第 9 拍将数据送至目标寄存器。

（5）写访问指令分析：写指令的前 6 拍与读指令一致，第 7 拍通过 load_addr 输出数据地址，同时，控制器通过 sel_m1 使选择器选择源寄存器的数据，发出写信号 write，第 8 拍写到存储器中。

（6）输出指令分析：第 4 拍通过 sel_m1、sel_m2 将数据送至 bus2，并发出 load_<port> 信号，第 5 拍输出到端口。

（7）无条件转移指令分析：无条件转移指令与读指令前 7 拍一致，送至 ADDR 的为转移的间接地址，第 8 拍通过 read 信号和间接地址读出存储器中的直接地址，通过 sel_m2 发送到 bus2，同时发出 load_pc 信号；第 9 拍通过 load_pc 将 bus2 上的直接地址放入 PC。

2.4 加入外部中断

2.4.1 中断进入过程

（1）外部设备发出中断请求 int，CPU 在取指周期的第一拍判断当前中断标志位 int_flag 是否有效，若有效，发出 load_int 命令，关中断（int_mask=1）；（2）PC 模块，收到 load_int 命令后，将当前的 PC 指针，保存到返回寄存器 ret_pc 中，将 pc 指向中断向量 [8-9]；（3）Z 模块，收到 load_int 命令后，将当前 z_flag 保存到 ret_z 寄存器；（4）从下一个取指周期开始，进入中断服务程序；（5）中断服务程序的第一条指令，保护现场，将中断服务程序中使用的资源写入数据区。

2.4.2 中断退出过程

（1）中断服务程序推出之前，需要恢复现场，从数据区中将保护的资源逐一读出；（2）中断服务程序最后一条指令为 RTI，CPU 在取指周期检测到 RTI，发出 load_ret 命令，开中断（int_mask=0），清中断（int_clear=1，下一拍反断言）；（3）PC 模块，收到 load_ret 命令后，从 ret_pc 中重新恢复 pc；（4）Z 模块，收到 load_ret 命令后，从 ret_z 中重新恢复 z_flag；（5）从下一个取指周期开始，恢复中断前的现场程序，继续运行。

2.4.3 控制器架构

中断控制器架构如图 4 所示，由一个中断寄存器和状态机组成，中断寄存器给状态机输送 int_flag 信号，状态机反馈给中断寄存器 int_mask 和 int_clear 信号 [10]。

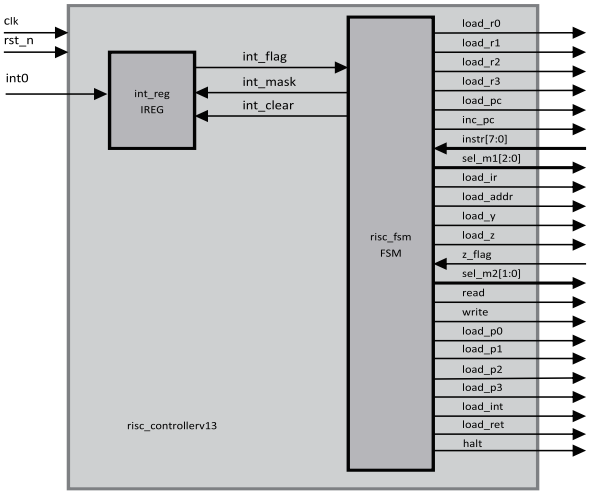

图 4 中断控制器架构

2.4.4 中断返回指令

中断返回指令状态转移图如图 5 所示，处理器收到中断返回指令时，拉高下载返回信号（load_ret=1），打开中断（int_mask=0），清中断（int_clear）。

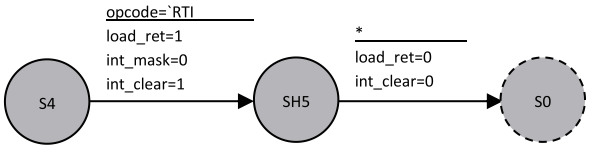

图 5 中断返回指令状态转移图

2.4.5 中断取指部分状态转移图

中断取指部分状态转移图如图 6 所示，若 S0 状态接收到中断信号（int_flag），则关中断（int_mask），下载中断（load_int =1）。

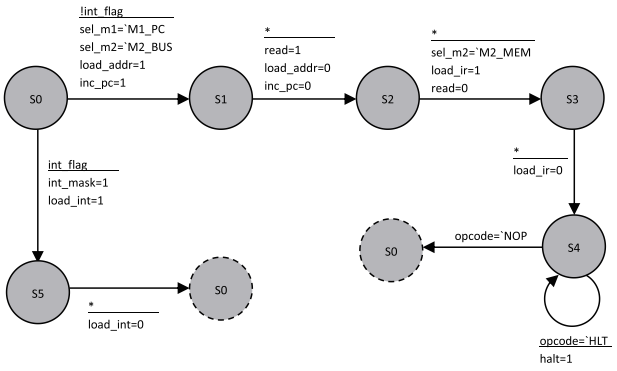

图 6 中断取指部分状态转移图

3 仿真验证

3.1 验证目标

使用设计的处理器，运行一组流水灯程序，当计数为 0 时，

第 1 盏灯亮；计数为 1 时，第 2 盏灯亮，依次亮 4 盏灯。中断程序运行一组 fibonacci 序列（1、1、2、3、5、8、13、21、34、55、89、144）。

3.2 算法流程图

主程序算法流程图如图 7 所示，从左至右依次为第一、二、三、四盏灯算法流程。考虑到人眼的反应时间为毫秒级，所以每一次计数都需要延迟一段时间。延迟子程序算法流程图如图 8 所示，延迟子程序采用了两级 256 计数延迟 65 536 个时钟周期，并将延迟子程序的退出地址写入到延迟子程序的返回地址，继续运行主程序。中断服务程序算法流程图如图 9 所示。

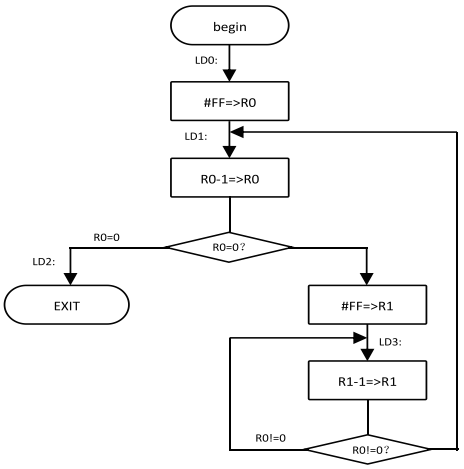

图 7 延迟子程序算法流程图

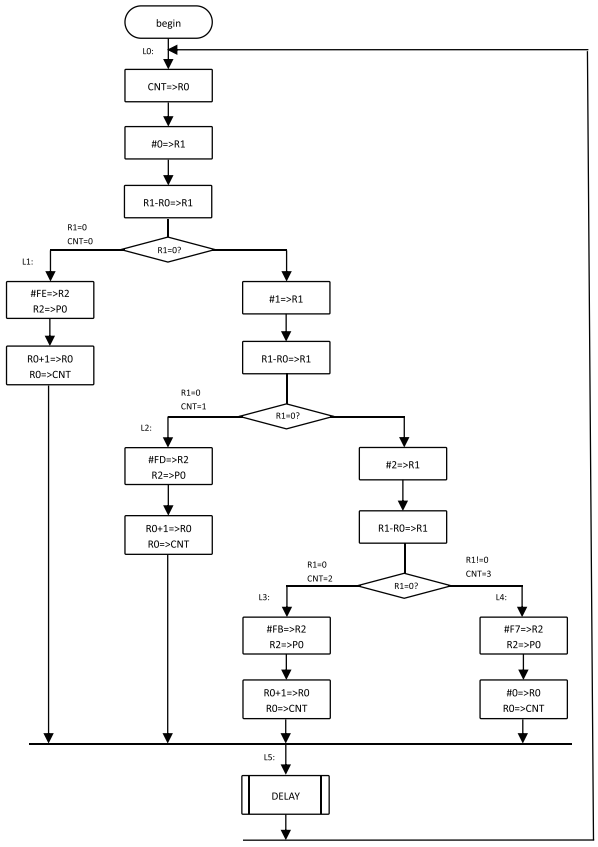

图 8 主程序算法流程图

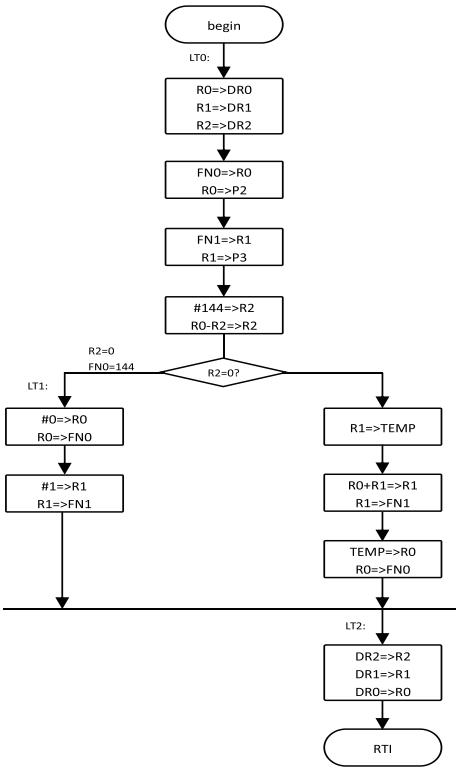

图 9 中断服务程序

3.3 手编机器码表

手编机器码表如表 4 所示。第一列地址表示存储器中的地址。第二列汇编程序，当为短指令时，表示相应的短指令、源寄存器、目标寄存器；当为长指令读时，表示读指令、存储器地址上的数、目标寄存器；当为立即数指令时，表示立即数指令，立即数、目标寄存器；当为写指令时，表示写指令、源寄存器、存储器上地址标签。

表 4 手编机器码表

地址	汇编程序	机器码			
		opcode	src	dest	addr
00H	BR L0	1011	00	00	1000_0001
02H	BR LT0	1011	00	00	1001_0001
10H	L0: RD CNT, R0	1000	00	00	1000_0000
12H	IMM #0, R1	0111	00	01	0000_0000
14H	SUB R0, R1	0010	00	01	
15H	BRZ L1	1100	00	00	1000_0010
17H	IMM #1,R1	0111	00	01	0000_0001
19H	SUB R0, R1	0010	00	01	
1AH	BRZ L2	1100	00	00	1000_0011
1CH	IMM #2,R1	0111	00	01	0000_0010
1EH	SUB R0,R1	0010	00	01	
1FH	BRZ L3	1100	00	00	1000_0100
21H	L4: IMM #F7,R2	0111	00	10	1111_0111
23H	OUT R2,P0	1010	10	00	
24H	IMM #0,R0	0111	00	00	0000_0000
26H	WR R0,CNT	1001	00	00	1000_0000
28H	BR L5	1011	00	00	1000_0110
2AH	L1: IMM #FE,R2	0111	00	10	1111_1110
2CH	OUT R2,P0	1010	10	00	
2DH	INC R0,R0	0101	00	00	
2EH	WR R0,CNT	1001	00	00	1000_0000
30H	BR L5	1011	00	00	1000_0110
32H	L2: IMM #FD,R2	0111	00	10	1111_1101
34H	OUT R2,P0	1010	10	00	
35H	INC R0,R0	0101	00	00	
36H	WR R0,CNT	1001	00	00	1000_0000
38H	BR L5	1011	00	00	1000_0110
3AH	L3: IMM #FB,R2	0111	00	10	1111_1011
3CH	OUT R2,P0	1010	10	00	
3DH	INC R0,R0	0101	00	00	

表 4（续）

地址	汇编程序			机器码			
				opcode	src	dest	addr
3EH		WR	R0,CNT	1001	00	00	1000_0000
40H		BR	L5	1011	00	00	1000_0110
42H	L5:	RD	L0,R3	1000	00	11	1000_0001
44H		WR	R3, LD2	1001	11	00	1000_1001
46H		BR	LD0	1011	00	00	1000_0111
48H	LD0:	IMM	#FF,R0	0111	00	00	1111_1111
4AH	LD1:	DEC	R0,R0	0110	00	00	
4BH		BRZ	LD2	1100	00	00	1000_1001
4DH		IMM	#FF,R1	0111	00	01	1111_1111
4FH	LD3:	DEC	R1,R1	0110	01	01	
50H		BRZ	LD1	1100	00	00	1000_1000
52H		BR	LD3	1011	00	00	1000_1010
54H	LT0:	WR	R0, DR0	1001	00	00	1000_1110
56H		WR	R1, DR1	1001	01	00	1000_1111
58H		WR	R2, DR2	1001	10	00	1001_0000
5AH		RD	FN0, R0	1000	00	00	1000_1011
5CH		OUT	R0, P2	1010	00	10	
5DH		RD	FN1, R1	1000	00	01	1000_1100
5FH		OUT	R1, P3	1010	01	11	
60H		IMM	#90H, R2	0111	00	10	1001_0000
62H		SUB	R0, R2	0010	00	10	
63H		BRZ	LT1	1100	00	00	1001_0010
65H		WR	R1, TEMP	1001	01	00	1000_1101
67H		ADD	R0, R1	0001	00	01	
68H		WR	R1, FN1	1001	01	00	1000_1100
6AH		RD	TEMP, R0	1000	00	00	1000_1101
6CH		WR	R0, FN0	1001	00	00	1000_1011
6EH		BR	LT2	1011	00	00	1001_0011
70H	LT1:	IMM	#0, R0	0111	00	00	0000_0000
72H		WR	R0, FN0	1001	00	00	1000_1011
74H		IMM	#1, R1	0111	00	01	0000_0001
76H		WR	R1, FN1	1001	01	00	1000_1100
78H	LT2:	RD	DR2, R2	1000	00	10	1001_0000
7AH		RD	DR1, R1	1000	00	01	1000_1111
7CH		RD	DR0, R0	1000	00	00	1000_1110
7EH		RTI		1101	00	00	

3.4 数据映射表

地址映射表如表 5 所示，表中，助记符表示存储器对应表格第一列地址的标签名。

表 5 数据映射表

地址	助记符	初始值	备注
80H	CNT	0000_0000	LED 灯计数
81H	L0	0001_0000	
82H	L1	0010_1010	2AH
83H	L2	0011_0010	32H
84H	L3	0011_1010	3AH
85H	L4	0010_0001	21H
86H	L5	0100_0010	42H
87H	LD0	0100_1000	48H
88H	LD1	0100_1010	4AH
89H	LD2	0000_0000	动态
8AH	LD3	0100_1111	4FH
8BH	FN0	0000_0000	P2
8CH	FN1	0000_0001	P3
8DH	TEMP	0000_0000	
8EH	DR0	0000_0000	
8FH	DR1	0000_0000	
90H	DR2	0000_0000	
91H	LT0	0101_0100	54H
92H	LT1	0111_0000	70H
93H	LT2	0111_1000	78H

3.5 验证结果

仿真验证结果如图 10 所示，成功使用设计的处理器执行所需的流水灯程序和中断程序。流水灯程序按照要求，在计数为 0 时点亮第 1 盏灯，计数为 1 时点亮第 2 盏灯，依次点亮 4 盏灯。

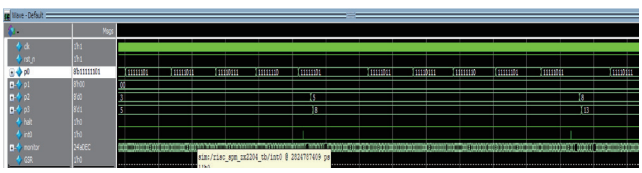

图 10 验证结果

中断程序也按照要求生成了 Fibonacci 序列，并成功在适当的时机中断流水灯程序执行。整个验证过程包括了设计集成、程序编写、仿真执行和功能测试等多个步骤。通过仿真验证，处理器表现出了预期的正确行为，满足设计需求。

4 结论

本文介绍了一种基于 RISC 架构和 SPM 型设计理念的处理器，并详细阐述了其指令格式、寻址方式、处理器核心以及控制器部分，旨在为后续研究提供一种处理器设计方法和验证思路。基于 FPGA 的特性，该处理器功能的优化和升级十分便捷。实验者可根据需求对微处理器的部分进行修改，实现自身设计理念，为创新思想提供了实践平台。因此，该处理器具有高度的灵活性，改善了微处理器实验教学的局限，充分展示了使用 FPGA 和 Verilog 语言进行数字系统设计的优势。此外，该微处理器还可作为片上系统设计的一个模块，具有一定的拓展性。

参考文献：

[1] 姜贺亮 . 基于 FPGA 的 32 位 RISC-V 处理器设计 [D]. 兰州 : 兰州大学 , 2023.

[2] 于洋 , 肖铁军 , 丁伟 . 面向教学的 16 位 CISC 微处理器的设计 [J]. 计算机工程与设计 , 2010,31(16):3584-3587.

[3] 魏忠军 . RISC-V 精简指令架构的研究及基于 RV32I 指令集的处理器内核设计 [D]. 成都 : 电子科技大学 , 2021.

[4] 杜岚 , 王裕 , 刘向峰 , 等 . 一种基于 RISC-V 架构的高性能嵌入式处理器设计 [J]. 小型微型计算机系统 , 2023, 44(12): 2865-2871.

[5] 赵博涵 . 基于 RISC-V 指令集的处理器核与 SoC 设计 [D]. 杭州 : 杭州电子科技大学 ,2023.

[6] 谢华 , 肖青 , 朱泽睿 , 等 . 基于 RISC-V 架构的向量指令集和通信扩展指令集在 5G Redcap 基带处理器中的开发和应用 [J]. 中国信息化 ,2024(1):89-90.

[7] 刘鑫 . 面向 RISC- V 架构处理器的集成开发环境设计与实现 [D]. 苏州 : 苏州大学 ,2023.

[8] 高嘉轩 , 刘鸿瑾 , 施博 , 等 . 基于向量表的 RISC-V 处理器普通中断与 NMI 优化设计 [J]. 微电子学与计算机 , 2024, 41(4): 112-122.

[9] 徐可凡 . 基于 RISC-V 的中断系统的研究与设计 [D]. 西安 : 西安电子科技大学 , 2020.

[10] 王亮 , 张盛兵 , 谭永亮 , 等 .8254 的 Verilog 实现和 FPGA 验证 [J]. 电子测量技术 , 2008,31(1): 150-152.

【作者简介】

解鹏越（1996—），男，陕西咸阳人，硕士研究生，研究方向：电子信息技术研究，email:1015770496@qq.com。

（收稿日期：2024-10-08）