

一、CPU 模块的总体设计

1. CPU 的主要性能指标和特性

(1) 基本技术指标

地址线宽度	32 位	数据线宽度	32 位
字节序	小端模式	是否采用流水线	否
存储字长	32 位	I/O 编址方式	与主存统一编址
指令系统	精简版 MIPS32 指令集 RISC 系统	数据表示	定点表示和浮点表示相结合 存在两套处理机构

(2) 寄存器组织：

本系统采用的寄存器分为通用寄存器和浮点寄存器两大模块，分别包含 32 个寄存器单元。寄存器的设计与使用规范符合 MIPS 标准。根据 MIPS 规范，通用寄存器中的 \$29(\$sp) 寄存器存有预设内容 0x00003ffc，表示栈地址在内存中的起始地址。

(3) 内存访问模式：

按字访问，按字节编址。内存地址的最后两位皆为 0。

2. CPU 的构成及各部分的简要说明

本 CPU 主要由主控制器(CU)、算术运算器(ALU)、浮点运算器(FLU)、乘商运算器(MLU)、乘商寄存器(Hi&Lo)，数据通路、通用寄存器、浮点寄存器组成。

CPU 指令系统通过主控制器与其他各模块之间的控制信号传输实现。数据通路模块负责协调与规划各模块之间的硬件连接。浮点运算器(FLU)采用浮点 IP 核实现，具有基本的浮点加减乘除运算功能。乘商运算器(MLU)实现了整数乘除法功能。

寄存器分为通用寄存器和浮点寄存器两类。它们的硬件结构基本一致，本质是从相同的通用寄存器模块实例化而来，而具体的功能区分由实例化参数决定。

当作为定点寄存器时，寄存器符合通用寄存器规范。具体表现为 \$0 寄存器输出常值 0，\$29 寄存器(\$SP)具有初始值 0x00003ffc，表示栈起始地址，即主存尾地址。

当作为浮点寄存器时，寄存器内部各存储单元不做特殊限制。

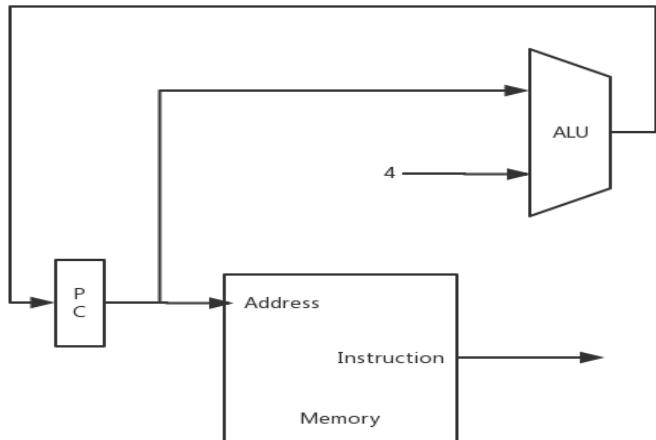
乘商寄存器为 MLU 专用寄存器。Hi 存储每次整数乘法指令结果的高位或者是除法指令结果的余数，Lo 存储整数乘法指令结果的低位或除法指令结果的商。其值不可通过指令操作直接写入值，也不可被直接访问。而是通过 mfhi 和 mflo 指令间接把值载入其他通用寄存器。

3. CPU 的指令系统

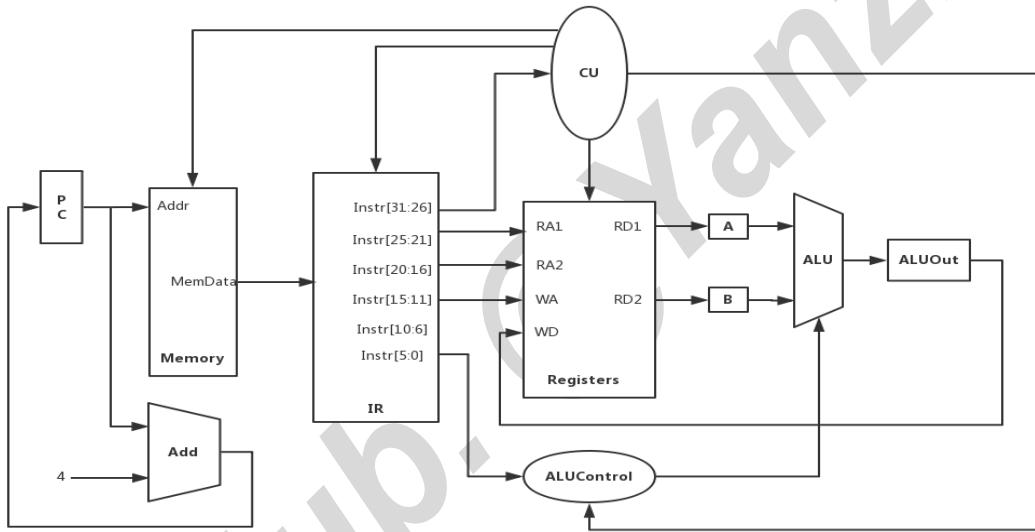
指令名称	调用格式	指令格式							指令功能																						
		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2
addiu	addiu rt, rs, imm	9		rs		rt		imm							无符号立即数相加																
addu	addu rd, rs, rt	0		rs		rt		rd		0		0x21		无符号整数相加																	
add.s	add.s fd, fs, ft	0x11		0x10		ft		fs		fd		0		单精度浮点数相加																	
andi	andi rt, rs, imm	0xC		rs		rt		imm							立即数与																
bc1t	bc1t cc, label	0x11		8		cc	1	offset							CC寄存器为真时跳转																
beq	beq rs, rt, label	4		rs		rt		offset							若rs与rt的值相等则跳转																
bgez	bgez rs, label	1		rs		1		offset							若rs的值大于等于0则跳转																
blez	blez rs, label	6		rs		0		offset							若rs的值小于等于0则跳转																
bne	bne rs, rt, label	5		rs		rt		offset							若rs与rt的值不相等则跳转																
cvt.w.s	cvt.w.s fd, fs	0x11		0x10		0		fs		fd		0x24		将单精度浮点数转化为整数																	
cvt.s.w	cvt.s.w fd, fs	0x11		0x14		0		fs		fd		0x20		将整数转化为单精度浮点数																	
c.le.s	c.le.s cc, fs, ft	0x11		0x10		ft		fs		cc	0	FC	0xE	若fs的值小于等于ft的值，则将CC寄存器置1																	
c.lt.s	c.lt.s cc, fs, ft	0x11		0x10		ft		fs		cc	0	FC	0xC	若fs的值大于ft的值，则将CC寄存器置1																	
div.s	div.s fd, fs, ft	0x11		0x10		ft		fs		fd		3		单精度浮点数除法																	
mtc1	mtc1 rt, fs	0x11		4		rt		fs		0		将通用寄存器rt的值存入浮点寄存器fs中																			
mfc1	mfc1 rt, fs	0x11		0		rt		fs		0		将浮点寄存器fs的值存入通用寄存器rt中																			
mul.s	mul.s fd, fs, ft	0x11		0x10		ft		fs		fd		2		单精度浮点数乘法																	
multu	multu rs, rt	0		rs		rt		0		无符号整数乘法																					
mfhi	mfhi rd	0		0				rd		0		0x10		将乘商寄存器Hi的值存入rd																	
mflo	mflo rd	0		0				rd		0		0x12		将乘商寄存器Lo的值存入rd																	
lw	lw rt, address	0x23		rs	0	offset							从指定的主存地址取字存入rt																		
lwc1	lwc1 ft, address	0x31		rs		ft		offset							从指定的主存地址取字存入浮点寄存器ft																
lui	lui rt, imm	0xF		0		rt		imm							立即数装入寄存器高16位																
j	j target	2		target							无条件跳转																				
jal	jal target	3		target							跳转并调用																				
jr	jr rs	0		rs	0			8			无条件跳转至rs的值表示的内存地址																				
sll	sll rd, rt, shamt	0		rs		rt		rd		shamt		0		逻辑左移																	
sllv	sllv rd, rt, rs	0		rs		rt		rd		0		40		逻辑左移rs的值																	
slt	slt rd, rs, rt	0		rs		rt		rd		0		0x2A		小于则置1																	
slti	slti rt, rs, imm	0xA		rs		rt		imm							立即数小于则置1																
srl	srl rd, rt, shamt	0		rs		rt		rd		shamt		2		逻辑右移																	
srlv	srlv rd, rt, rs	0		rs		rt		rd		0		6		逻辑右移rs的值																	
sw	sw rt, address	0x2B		rs	0	offset							将rt的值存入主存指定地址中																		
swc1	swc1 ft, address	0x39		rs		ft		offset							将ft的值存入指定的内存地址																
sub.s	sub.s fd, fs, ft	0x11		0x10		ft		fs		fd		1		单精度浮点数相减																	
subu	subu rd, rs, rt	0		rs		rt		rd		0		0x23		无符号整数相减																	

4. 按指令类型分类的 CPU 数据通路图

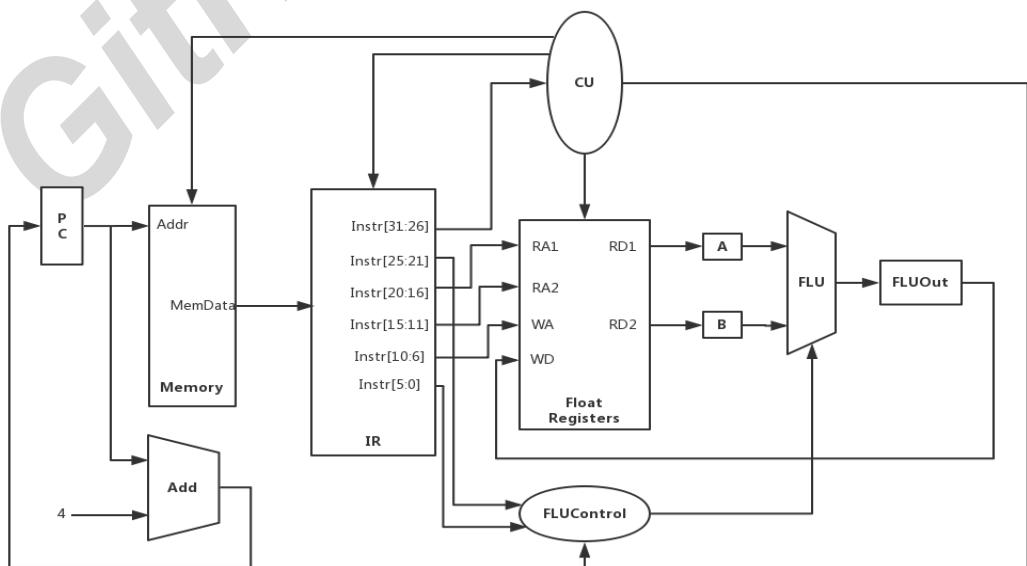
(1) 取指操作



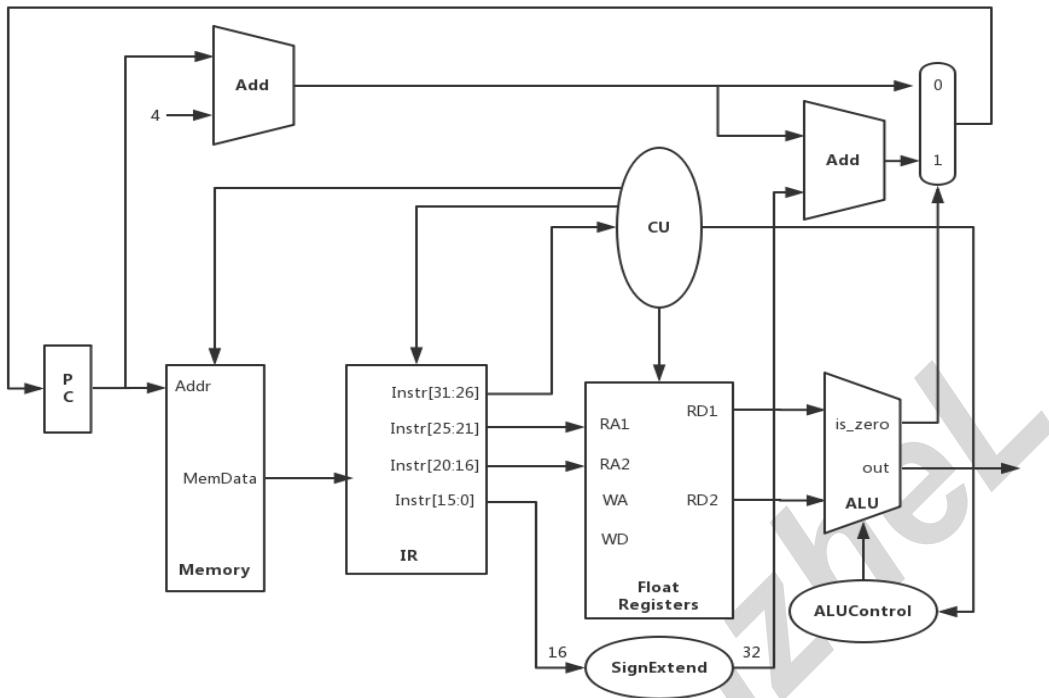
(2) R 型指令



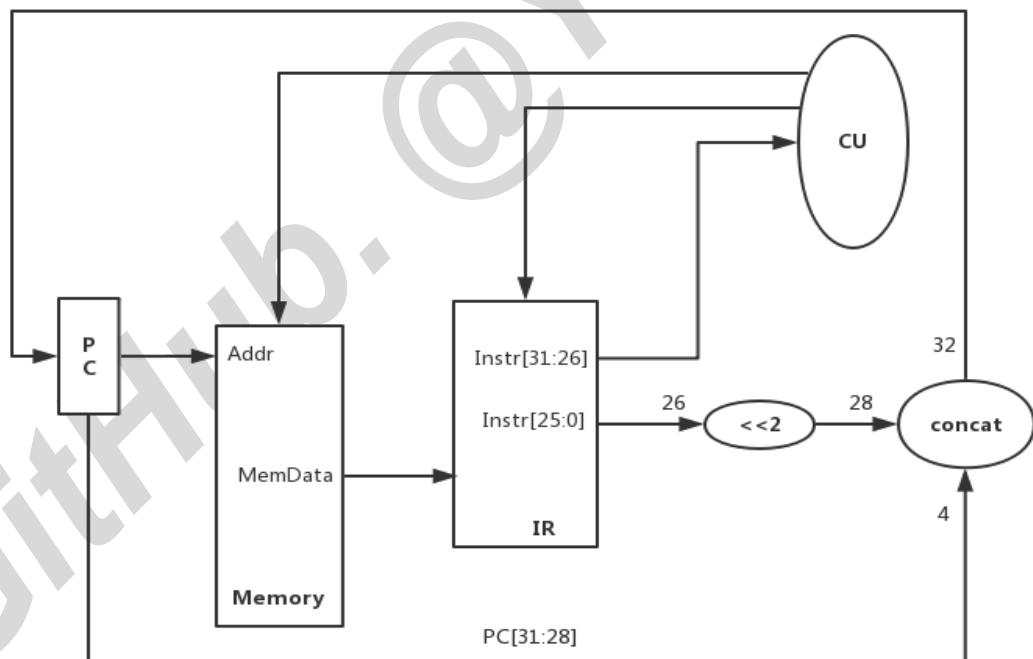
(3) FR 型指令



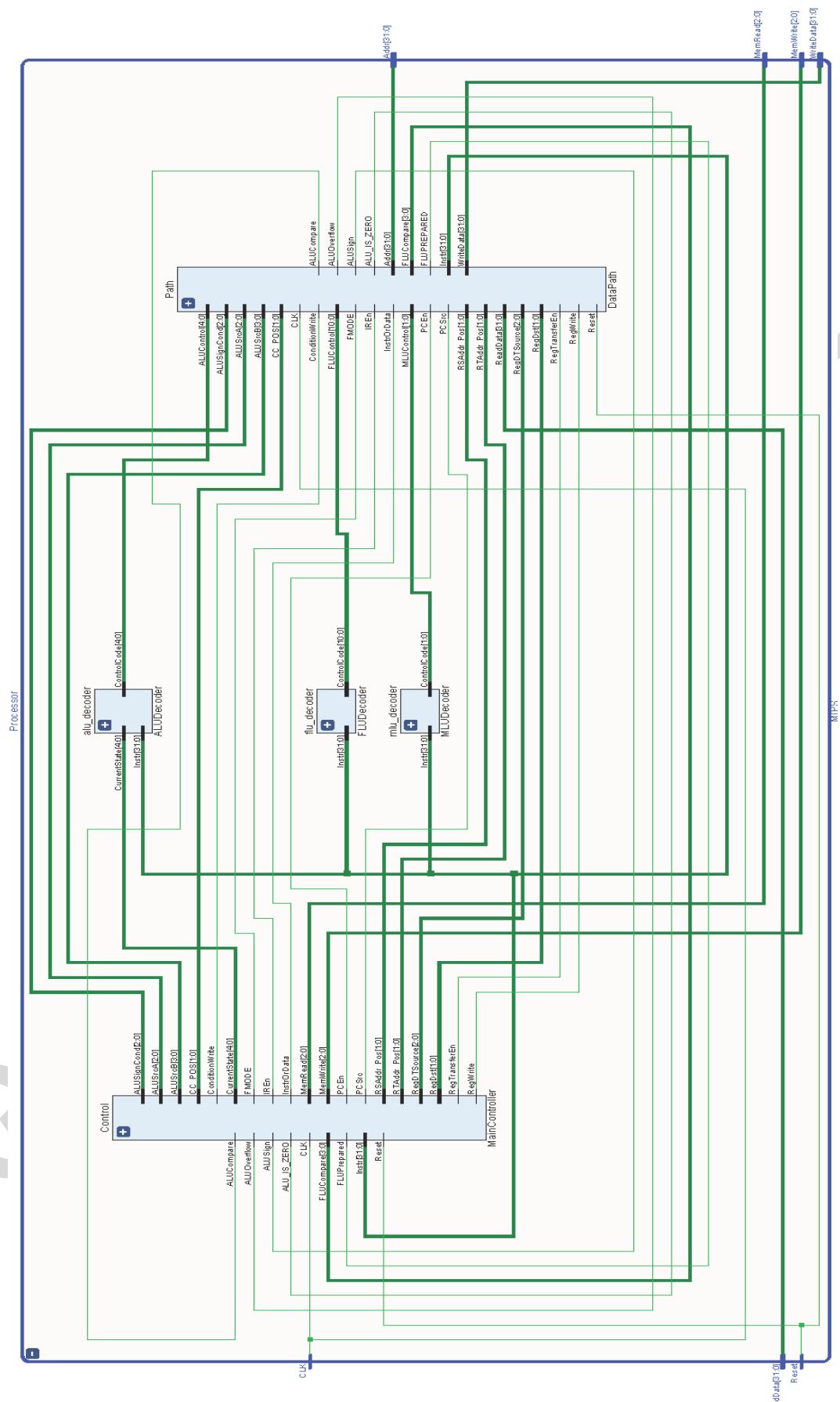
(4) I型指令



(5) J型指令



5. 总体的 CPU 数据通路图



6. CPU 主要模块的控制信号分析

输入信号

信号名称	信号功能描述	有效信号
CLK	驱动时钟	上升沿
Reset	重置信号，用于重置全局状态	上升沿
FLUPrepared	浮点运算器准备就绪标志	1
Instr[31:0]	取得的完整指令	X
ALU_IS_ZERO	ALU 计算结果是否为 0	1
ALUSign	ALU 计算结果符号位	0/1
ALUOverflow	ALU 计算结果是否溢出	0/1
ALUCompare	ALU 比较结果。其值由 ALUSignCond 给出的比较条件决定	0/1
FLUCompare[3:0]	FLU 比较结果	X

输出信号

信号名称	信号功能描述	有效信号
InstrOrData	指令或数据	0/1
IREn	指令寄存器使能	1
PCSrc	PC 来源选择	0/1
FMODE	浮点模式标志	0/1
RegTransferEn	通用寄存器与浮点寄存器直接数据交换使能	1
ConditionWrite	CC 寄存器写使能	1
PCEn	PC 值更新使能	1
RegWrite	寄存器写使能	1
MemWrite	主存写使能	1
MemRead	主存读使能	1
ALUSignCond[2:0]	ALU 比较模式信号，决定在何种情况下 ALUCompare 为真	000~110
RegDTSource[2:0]	寄存器写入数据来源选择信号	000~110
ALUSrcA[2:0]	加法器 A 路数据来源选择信号	000~100
ALUSrcB[3:0]	加法器 B 路数据来源选择信号	0000~1001
RegDst[1:0]	寄存器写入地址来源选择信号	00~11
RSAddr_Pos[1:0]	RS 数据在指令中的位置选择信号	00~11
RTAddr_Pos[1:0]	RT 数据在指令中的位置选择信号	00~11
CC_POS[1:0]	CC 数据在指令中的位置选择信号	00~11
CurrentState[4:0]	当前控制器状态码	00001~10000

7. 多周期操作设计(每个周期完成功能及控制信号电平分析)

(1) 取指周期 (FETCH)

该周期从存储器中取出一条指令并计算下一条指令的地址

```
IR <= Memory[PC];  
PC <= PC + 4;
```

执行操作：

将 MemRead 置 1，使内存进入读取状态；将 IREn 置 1，使 IR 可写；此时从内存读取到的数据已存于 IR 中；将 PCWrite 置 1，使 PC 寄存器可写；将 ALUSrcA 置 000，ALUSrcB 置 0001，使 ALU 分别输入 PC 和 4，此时 $PC + 4$ 的值通过 ALU 输出到 PC 寄存器的输入端，在下一个周期将完成 $PC \leftarrow PC + 4$ 的更新。

(2) 译码周期 (DECODE)

- 当 IR 寄存器获取到了完整的指令后，便需要通过译码步骤来实现指令的分析。由于本 MIPS 系统实现了多种类型的指令，特别是浮点指令集，因此对不同格式的指令进行译码是一个难点。

以普通 R 型指令和浮点数 FR 指令格式为例：

R 型指令

0	RS	RT	RD	0	FUNC
---	----	----	----	---	------

FR 型指令

010001	FMT	FS	FT	FD	FUNC
--------	-----	----	----	----	------

根据译码周期常采用的寄存器值预读取的设计，译码周期应该完成对寄存器 RS(FS)和 RT(FT)内容预读取的功能。而 R 型指令和 FR 型指令中的寄存器地址位置并不相同，两者还涉及到通用和浮点两类寄存器。因此在根据指令操作码区分两类指令后，增加 FMODE 标志来表明此时使用的是何种寄存器，并将合适的 RSAddr_Pos 和 RTAddr_Pos 信号输入到寄存器地址数据选择器中，选择出合适的 RS(FS)和 RT(FT)的值，从而正确预读取出寄存器地址对应的内容。

对于其他非 R 或 FR 型指令，采用了类似的指令译码方式，寄存器预读取的值若未使用则被丢弃。

- 浮点指令集存在特殊的两个浮点指令 mtc1 和 mfc1，其作用是将通用寄存器中的值传输到浮点寄存器中(或者是反方向传输)。而根据 FMODE 浮点模式标志的功能限制，通用寄存器和浮点寄存器并不能进行数据直接交换。因此需要设置一个暂存寄存器来临时存放中转途中的数据。

对于 mtc1 指令，其传输方向是通用寄存器 -> 浮点寄存器。因此在本译码周期将 RegTransferEn 置 1，即开启寄存器中转，中转寄存器处于可写状态；将 FMODE 置 0，即当前使用通用寄存器。此操作使通用寄存器对应地址的值暂存到 TransferData 中转寄存器中。

对于 mfc1 指令，其传输方向是浮点寄存器 -> 通用寄存器。于是将 RegTransferEn 置 1；将 FMODE 置 1，即当前使用浮点寄存器。此操作使浮点寄存器对应地址的值暂存到 TransferData 中转寄存器中。将中转数据写入过程提前到译码阶段能缩减 mtc1 和 mfc1 指令所需的总指令周期。

(3) 执行周期 (EXECUTE)

之前译码阶段得出的 RS 和 RT 预读取的值已经存入 RS 和 RT 寄存器，本阶段只需根据不同的指令将正确的数据与合适的控制信号传入各类运算器单元即可。

- 对于操作码为 00 0000(多数 R 型)和 01 0001(FR)的指令，因为其功能由指令末 6 位的 FUNC 代码决定。因此在执行阶段，这两大类特殊指令的 FUNC 部分将转交到第三方解码器 ALUDecoder、MLUDecoder 和 FLUDecoder 来处理，用以区分具体的指令功能，不同的解码器发出不同的控制信号给运算器。独立解码器的设计使运算器专用的控制结构从主控制器中独立出来，精简主控制器的实现。

此处 FUNC 指令的转交、控制信号的发出和运算器执行是同时发生的。在执行周期的结束后的下一个或多个周期之后，运算器输出端将产生运算结果。

- 对于整数运算，运算器(ALU 和 MLU)能够在一个周期内产生运算结果。因此执行周期的下一个周期便可以得到运算器输出。
- 对于浮点运算，运算器(FLU)需要多个时钟周期才能得出正确的运算结果。因此主控制器需要运算器传来的 FLUPrepared 就绪信号来得知运算器何时运算结束，进而进入后续的控制周期。

此处就绪信号的获取采用了程序查询方式，在浮点数据送入 FLU 之后的每个时钟周期，主控制器对 FLUPrepared 信号进行轮询，若 FLUPrepared 为假，则主控制器重新进入 EXECUTE 执行状态，即进行循环等待，否则进入后续控制周期。

采用程序查询方式也有其固有的弊端，在 FLU 运算过程中主控制器处于阻塞状态，对整体性能有很大的影响。若能采用异步通信机制或使用流水线设计，便能提高整体性能，但由于时间所限，本处理器并未实现。

- 对于 mtc1 指令，由于在译码阶段已经使中转数据存入 TransferData 中转寄存器，因此在本执行周期将 RegTransferEn 置 0，即关闭寄存器中转，中转寄存器处于保持

状态；FMODE 置 1，即当前使用浮点寄存器；将 RegSet 置 1，开启寄存器写使能；RegDTSource 选择 TransferData 中转寄存器中的数据作为浮点寄存器数据输入；RegDst 选择 Instr[15:11]作为寄存器写入地址。从而完成中转数据的写入。

- 对于 mfc1 指令，本执行周期将 RegTransferEn 置 0，即关闭寄存器中转，中转寄存器处于保持状态；FMODE 置 0，即当前使用通用寄存器；将 RegSet 置 1；RegDTSource 选择 TransferData；RegDst 选择 Instr[20:16]作为寄存器写入地址。从而完成中转数据的写入。

(4) 浮点运算结果转存寄存器周期(FLUTOREG)

大多数涉及浮点运算的 FR 型浮点指令在执行周期过后会进入本周期。FMODE 标志在前序周期已经开启，本周期进行的操作是将 RegSet 置 1，开启寄存器写使能；RegDTSource 选择 FLU 输出数据作为通用寄存器数据输入；RegDst 选择 Instr[10:6]作为寄存器写入地址。从而完成浮点数据的写入。

(5) 整数运算结果转存寄存器周期(ALUTOREG 和 MLUTOREG)

本周期功能与 FLUTOREG 周期类似，此处不再赘述。

(6) 内存寻址周期 (MEMADDR)

ALUOut <= (\$rs)+SignExtend(Instr[15:0])

执行操作：

ALUSrcA 选择 RS 寄存器值作为 ALU 运算器 A 端输入；ALUSrcB 选择按符号扩展的立即数 Instr[15:0]的值作为 ALU 运算器 B 端输入。从而在本周期结束时 ALU 输出端将产生计算后的内存地址。

(7) 内存读写周期 (MEMREAD 和 MEMWRITE)

执行操作：

将 InstrOrData 置 1，即声明此时 Addr 的值表示数据；将 MemRead 或 MemWrite 置 1，开启内存读或写使能。对于 lwc1 和 swc1 指令，需要在本周期开启 FMODE。

(8) 跳转并链接周期 (JUMPLINK)

本周期的功能为 jal 指令计算跳转地址，并将当前 PC 值存入\$ra (\$31)寄存器中。

执行操作：

RegDst 选择 11111(31)作为寄存器写入地址；RegDTSource 选择当前 PC 值作为寄存器写入数据；ALUSrcA 和 ALUSrcB 分别选择当前 PC 值和 Instr[25:0]按 0 前后扩展的值作为 ALU 的 A、B 端输入。从而 PC 的值将在下个周期被更新为运算结果。

(9) 跳转周期 (JUMP)

本周期的功能为把 PC 值更新为之前计算好的跳转地址

执行操作：将 PCWrite 置 1；PCSrc 选择 ALUOut 寄存器的值作为新的 PC

(10) 分支周期 (BRANCH)

本周期的功能是根据不同的指令，为运算器传入不同的比较成立条件 ALUSignCond。

当运算器传出的 ALUCompare 信号有效时，表示比较条件 ALUSignCond 成立，从而允许进行跳转。

执行操作：

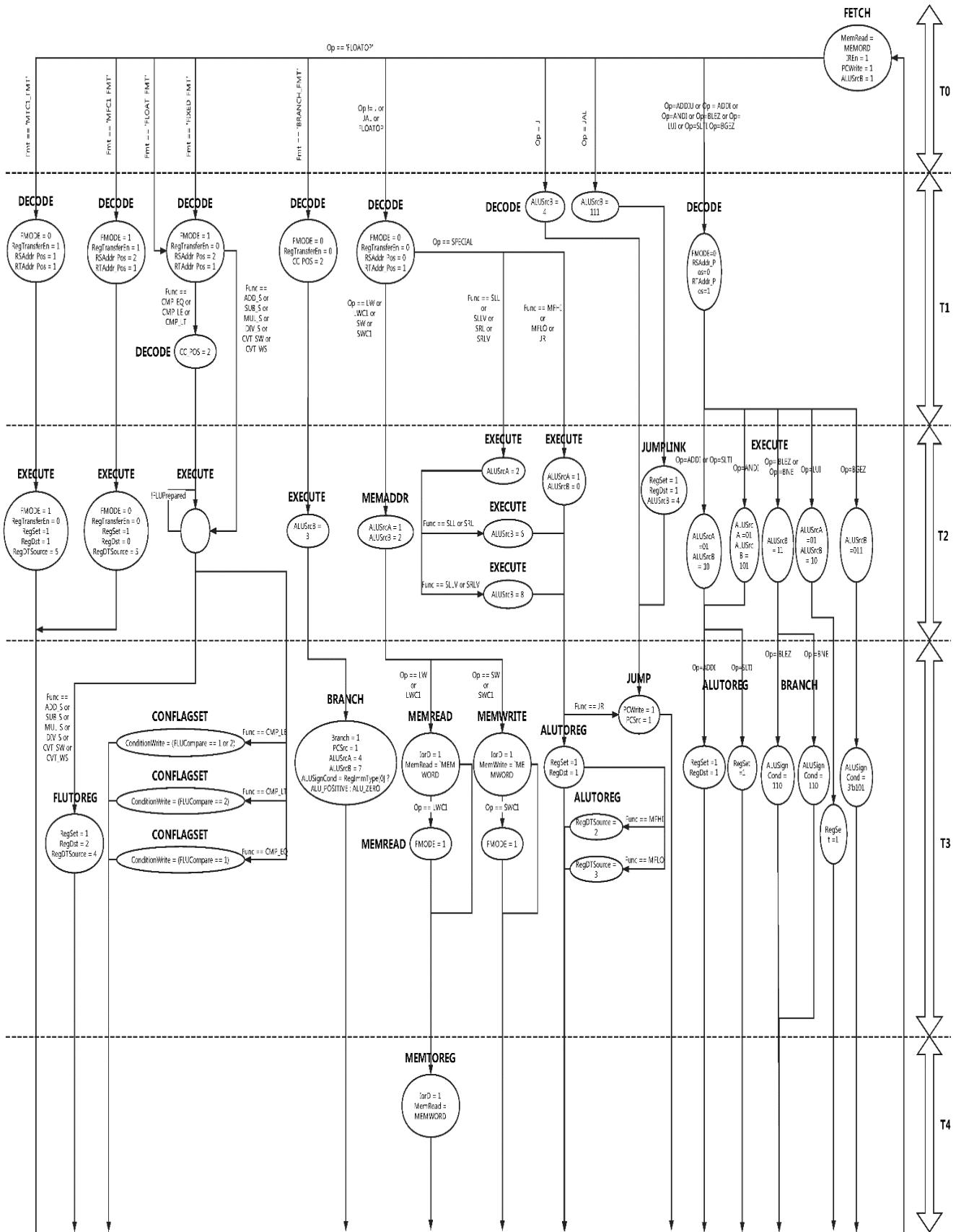
PCSrc 选择 ALUOut 作为 PC 更新值来源；将 Branch 置 1，开启 Branch 功能；

ALUSignCond 的内容由具体的跳转指令决定。

- 对于浮点指令的跳转操作，ALUSrcA 和 ALUSrcB 分别选择 CC 寄存器的值和 0 送入 ALU；
- 对于其他指令的跳转操作，ALUSrcA 和 ALUSrcB 分别选择合适的数据。

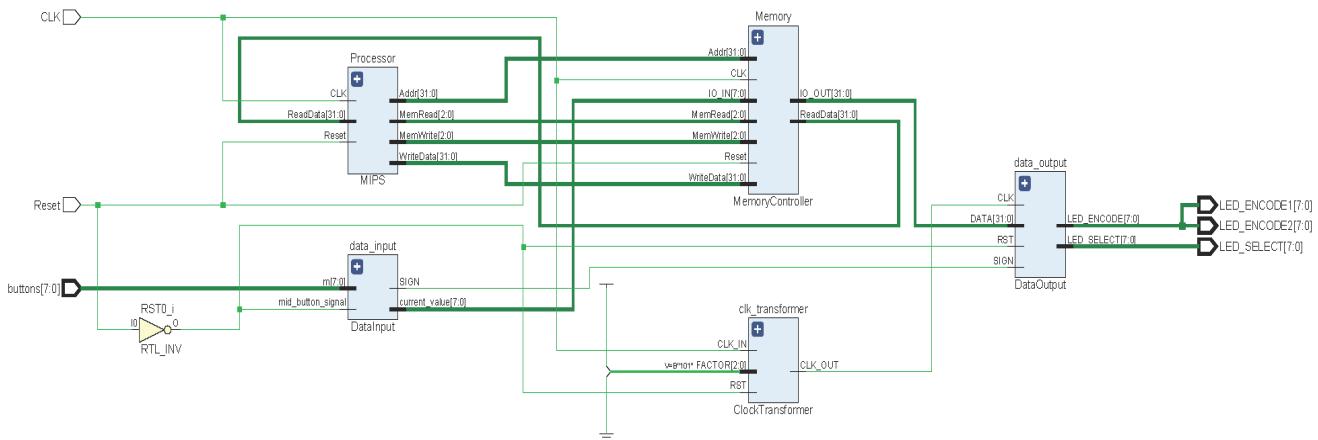
当且仅当 Branch 和 ALUCompare 同时为真时才将 PCEn 置 1，使 PC 的值在本周期立即更新为上周期 ALUOut 寄存器保持的值。

8. 控制单元的状态转换图

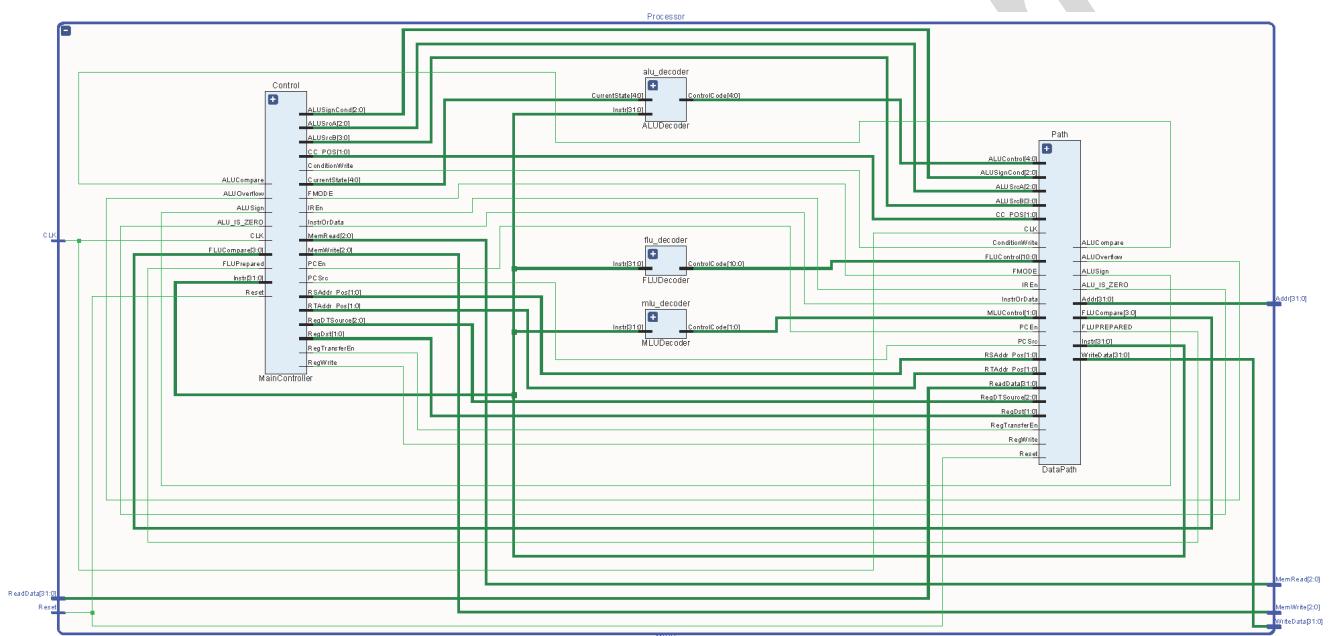


9. CPU 的 Verilog 描述分析(模块构成及层次图)

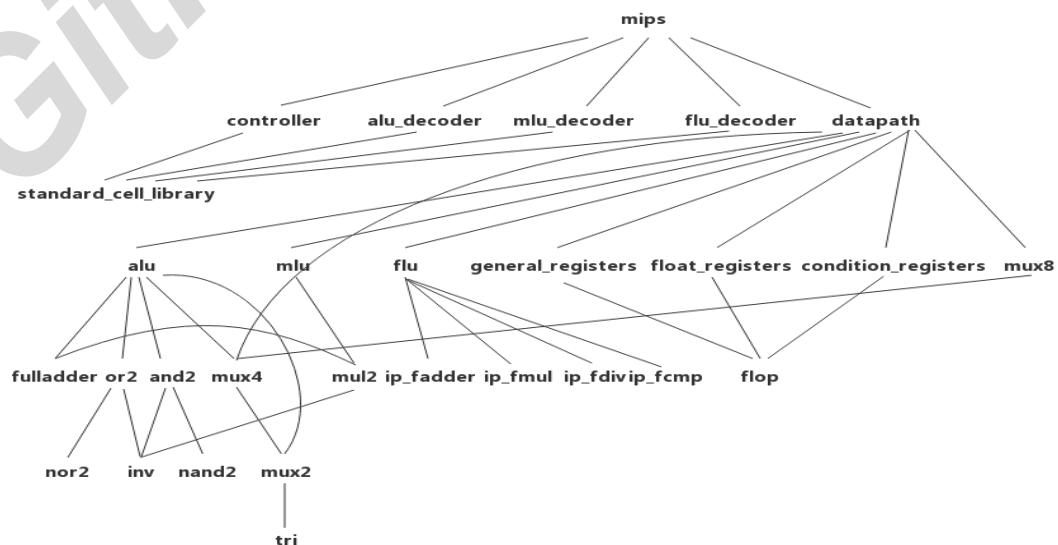
● 顶层模块



- 内部模块



● 层次结构



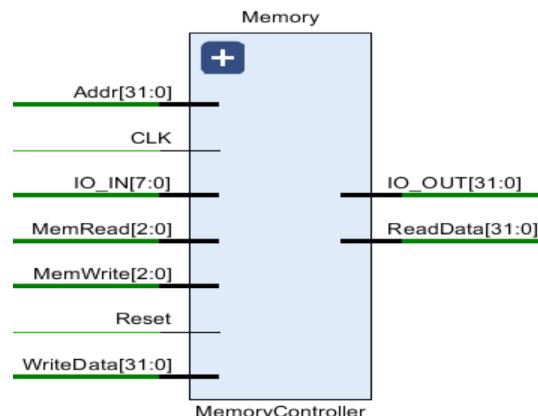
二、外围模块的设计

1. 时钟模块的设计

本设计的时钟信号直接由 EGO1 板载晶振(P17 引脚)提供

2. 存储器模块的设计

(1) 封装图



(2) 存储器划分

通用 RAM 存储器

(3) 信号分析

输入信号

信号名称	信号功能描述	有效信号
CLK	驱动时钟	上升沿
Reset	重置信号，用于重置全局状态	1
Addr[31:0]	访存地址	X
IO_IN[7:0]	外设数据输入	X
MemRead	主存读使能	1
MemWrite	主存写使能	1
WriteData[31:0]	写入数据	X

输出信号

信号名称	信号功能描述	有效信号
IO_OUT[31:0]	输出给外设的数据	X
ReadData[31:0]	读取出的数据	1

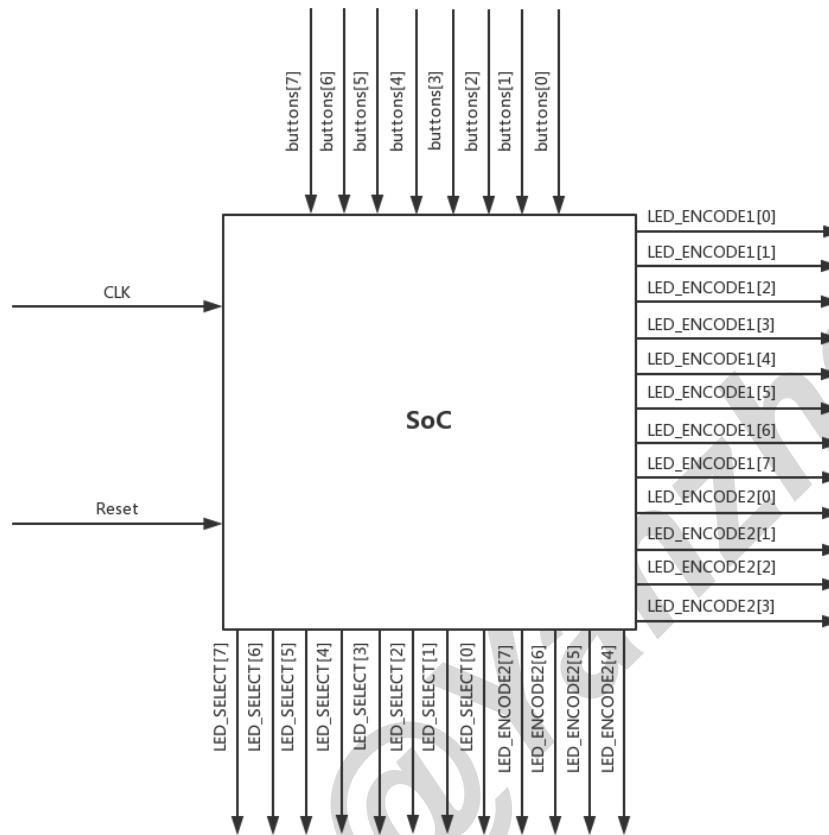
(4) 主存地址划分

内存地址	用途
0x0000 ~ 0x0ffc	.text 程序段地址
0x1000 ~ 0x2ff7	.data 数据段地址
0x2ff8	IO 输入设备地址
0x2ffc	IO 输出设备地址
0x3000 ~ 0x3ffc	栈地址

三、SoC 的设计

1. SoC 的外部封装图及信号分析(输入输出信号)

(1) 封装图



(2) 信号分析

输入信号

信号名称	信号功能描述	有效信号
CLK	系统驱动时钟	上升沿
Reset	重置信号，用于重置全局状态，由中键按钮触发	上升沿
buttons[7:0]	8 位拨码开关输入	X

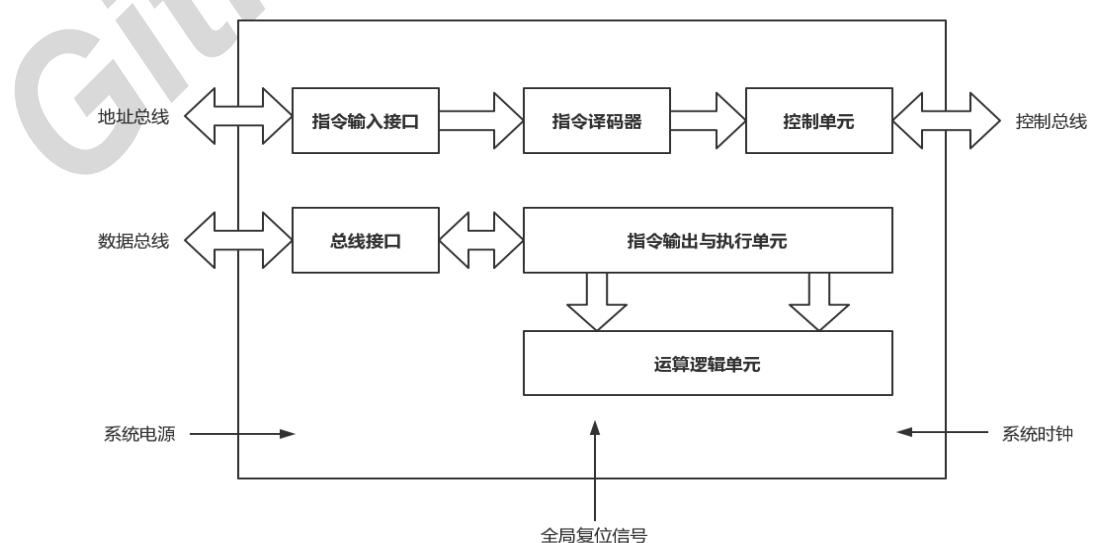
输出信号

信号名称	信号功能描述	有效信号
LED_ENCODE1 [7:0]	数码管第一组段选信号	X
LED_ENCODE2 [7:0]	数码管第二组段选信号	X
LED_SELECT[7:0]	数码管位选信号	X

2. SoC 的引脚约束表(信号与管脚的对应表)

Name	Direction	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type	Off-Chip Termination	IN_TERM
All ports (34)													
adk_intf_708 (1)	IN			✓	14	LVCMS18	✓	1.800			NONE	✓	NONE
Scalar ports (1)													
CLK	IN		P17	✓	✓	14	LVCMS18	✓	1.800		NONE	✓	NONE
aresetn_intf_708 (1)	IN			✓	14	LVCMS18	✓	1.800			NONE	✓	NONE
Scalar ports (1)													
Reset	IN		R15	✓	✓	14	LVCMS18	✓	1.800		NONE	✓	NONE
buttons (8)	IN			✓	34	LVCMS18	✓	1.800			NONE	✓	NONE
buttons[7]	IN		P5	✓	✓	34	LVCMS18	✓	1.800		NONE	✓	NONE
buttons[6]	IN		P4	✓	✓	34	LVCMS18	✓	1.800		NONE	✓	NONE
buttons[5]	IN		P3	✓	✓	34	LVCMS18	✓	1.800		NONE	✓	NONE
buttons[4]	IN		P2	✓	✓	34	LVCMS18	✓	1.800		NONE	✓	NONE
buttons[3]	IN		R2	✓	✓	34	LVCMS18	✓	1.800		NONE	✓	NONE
buttons[2]	IN		M4	✓	✓	34	LVCMS18	✓	1.800		NONE	✓	NONE
buttons[1]	IN		N4	✓	✓	34	LVCMS18	✓	1.800		NONE	✓	NONE
buttons[0]	IN		R1	✓	✓	34	LVCMS18	✓	1.800		NONE	✓	NONE
LED_ENCODE1 (8)	OUT			✓	35	LVCMS18	✓	1.800	12	✓ SLOW	✓	NONE	✓ FP_VTT_50
LED_ENCODE1[7]	OUT		B4	✓	✓	35	LVCMS18	✓	1.800	12	✓ SLOW	✓	NONE
LED_ENCODE1[6]	OUT		A4	✓	✓	35	LVCMS18	✓	1.800	12	✓ SLOW	✓	NONE
LED_ENCODE1[5]	OUT		A3	✓	✓	35	LVCMS18	✓	1.800	12	✓ SLOW	✓	NONE
LED_ENCODE1[4]	OUT		B1	✓	✓	35	LVCMS18	✓	1.800	12	✓ SLOW	✓	NONE
LED_ENCODE1[3]	OUT		A1	✓	✓	35	LVCMS18	✓	1.800	12	✓ SLOW	✓	NONE
LED_ENCODE1[2]	OUT		B3	✓	✓	35	LVCMS18	✓	1.800	12	✓ SLOW	✓	NONE
LED_ENCODE1[1]	OUT		B2	✓	✓	35	LVCMS18	✓	1.800	12	✓ SLOW	✓	NONE
LED_ENCODE1[0]	OUT		D5	✓	✓	35	LVCMS18	✓	1.800	12	✓ SLOW	✓	NONE
LED_ENCODE2 (8)	OUT			✓	35	LVCMS18	✓	1.800	12	✓ SLOW	✓	NONE	✓ FP_VTT_50
LED_ENCODE2[7]	OUT		D4	✓	✓	35	LVCMS18	✓	1.800	12	✓ SLOW	✓	NONE
LED_ENCODE2[6]	OUT		E3	✓	✓	35	LVCMS18	✓	1.800	12	✓ SLOW	✓	NONE
LED_ENCODE2[5]	OUT		D3	✓	✓	35	LVCMS18	✓	1.800	12	✓ SLOW	✓	NONE
LED_ENCODE2[4]	OUT		F4	✓	✓	35	LVCMS18	✓	1.800	12	✓ SLOW	✓	NONE
LED_ENCODE2[3]	OUT		F3	✓	✓	35	LVCMS18	✓	1.800	12	✓ SLOW	✓	NONE
LED_ENCODE2[2]	OUT		E2	✓	✓	35	LVCMS18	✓	1.800	12	✓ SLOW	✓	NONE
LED_ENCODE2[1]	OUT		D2	✓	✓	35	LVCMS18	✓	1.800	12	✓ SLOW	✓	NONE
LED_ENCODE2[0]	OUT		H2	✓	✓	35	LVCMS18	✓	1.800	12	✓ SLOW	✓	NONE
LED_SELECT (8)	OUT			✓	35	LVCMS18	✓	1.800	12	✓ SLOW	✓	NONE	✓ FP_VTT_50
LED_SELECT[7]	OUT		G2	✓	✓	35	LVCMS18	✓	1.800	12	✓ SLOW	✓	NONE
LED_SELECT[6]	OUT		C2	✓	✓	35	LVCMS18	✓	1.800	12	✓ SLOW	✓	NONE
LED_SELECT[5]	OUT		C1	✓	✓	35	LVCMS18	✓	1.800	12	✓ SLOW	✓	NONE
LED_SELECT[4]	OUT		H1	✓	✓	35	LVCMS18	✓	1.800	12	✓ SLOW	✓	NONE
LED_SELECT[3]	OUT		G1	✓	✓	35	LVCMS18	✓	1.800	12	✓ SLOW	✓	NONE
LED_SELECT[2]	OUT		F1	✓	✓	35	LVCMS18	✓	1.800	12	✓ SLOW	✓	NONE
LED_SELECT[1]	OUT		E1	✓	✓	35	LVCMS18	✓	1.800	12	✓ SLOW	✓	NONE
LED_SELECT[0]	OUT		G6	✓	✓	35	LVCMS18	✓	1.800	12	✓ SLOW	✓	NONE

3. SoC 的结构框图



四、测试用例的编写

1. 测试用例的功能描述

- 利用上文所述的 MIPS 指令集实现正弦函数值的计算。
- 输入角度值范围: 0~255 度
- 输入设备: 8 位拨码开关
- 输出设备: 8 位数码管

2. 测试用例的汇编语言代码

```
# -----
.globl main
main:
    addiu   $sp,$sp,-0x30
    sw      $ra, 0x2C($sp)
    sw      $fp, 0x28($sp)
    addu   $fp, $zero,$sp
    lw      $v0, 0x00002ff8($zero)
    # load degree from input device
    jal    deg_range_convert
    sw      $v0, 0x1C($fp)
    lw      $a0, 0x1C($fp)
    jal    sin_mips
    nop
    swc1   $f0, 0x20($fp)
    lwc1   $f12, 0x20($fp)
    jal   convert_digits
    nop
    sw     $v0, 0x24($fp)
    sw     $v0, 0x00002ffc($zero)
    # output result to output device
    addu   $sp,$zero, $fp
    lw      $ra, 0x2C($sp)
    lw      $fp, 0x28($sp)
    addiu  $sp,$sp, 0x30
wait_signal:
    lw      $t1,0x00002010($zero)
    blez   $t1,wait_signal
    jr     $ra
    nop
#-----
.globl sin_mips
sin_mips:
    addiu  $sp,$sp, -0x20
    sw     $fp, 0x1C($sp)
    addu   $fp,$zero, $sp
    sw     $a0, 0x20($fp)
    sw     $zero, 0x04($fp)
    lw      $v0, 0x20($fp)
    mtc1   $v0, $f0
    cvt.s.w $f2, $f0
    lui    $v0, 0x40
    lwc1   $f0, flt_400B20
    mul.s  $f2,$f2, $f0
    lui    $v0, 0x40
    lwc1   $f0, flt_400B24
    div.s  $f0, $f2, $f0
    swc1   $f0, 0x14($fp)
    addiu  $v0,$zero, 1
    sw     $v0, 0x08($fp)
    bgez   $zero,LL0C
    nop
# -----
CC:
    lwc1   $f0, 0x14($fp)
    swc1   $f0, 0x0C($fp)
    addiu  $v0,$zero, 2
    sw     $v0, 0x10($fp)
    bgez   $zero,LLA
    nop
# -----
LL0:
    lwc1   $f2, 0x0C($fp)
    lwc1   $f0, 0x14($fp)
    mul.s  $f0, $f2, $f0
    swc1   $f0, 0x0C($fp)
    lw      $v0, 0x10($fp)
    mtc1   $v0, $f0
    cvt.s.w $f0, $f0
    lwc1   $f2, 0x0C($fp)
    div.s  $f0, $f2, $f0
    swc1   $f0, 0x0C($fp)
    lw      $v0, 0x10($fp)
    addiu  $v0, $v0,1
    sw     $v0, 0x10($fp)
    LLA:
    lw      $v0, 0x08($fp)
    sll   $v0, $v0,1
    addiu $v0, $v0,-1
    lw      $v1, 0x10($fp)
    slt   $v0, $v0,$v1
    beq   $v0,$zero, LLo
    nop
    lw      $v0, 0x08($fp)
    andi  $v0, 1
    bne   $v0,$zero, LLE
    nop
    lw      $v1, 0x0C($fp)
    lui    $v0, 0x8000
    xor   $v0, $v1, $v0
    bgez  $zero,LLEC
    nop
# -----
LLE:
    lw      $v0, 0x0C($fp)
LLEC:
    sw     $v0, 0x0C($fp)
    lwc1   $f2, 0x04($fp)
```

```

lwc1    $f0, 0x0C($fp)
add.s   $f0, $f2, $f0
swc1    $f0, 0x04($fp)
lw      $v0, 0x08($fp)
addiu   $v0, $v0,1
sw      $v0, 0x08($fp)
LL0C:
lw      $v0, 0x08($fp)
slti   $v0, $v0,7
bne    $v0,$zero, CC
nop
lwc1    $f0, 0x04($fp)
addu   $sp,$zero, $fp
lw      $fp, 0x1C($sp)
addiu   $sp, $sp,0x20
jr     $ra
nop
# ====== S U B R O U T I N E ======
.globl convert_digits
convert_digits:
    addiu   $sp, $sp,-0x18
    sw      $fp, 0x14($sp)
    addu   $fp,$zero, $sp
    swc1    $f12, 0x18($fp)
    lwc1    $f2, 0x18($fp)
    lui     $v0, 0x40
    lwc1    $f0, flt_400B28
    mul.s   $f0, $f2, $f0
    lwc1    $f2, flt_400B2C
    c.le.s  $f2, $f0
    bc1t   LL8C
    nop
    cvt.w.s $f0,$f0
    mfc1    $v0, $f0
    bgez   $zero,LL88C
    nop
#
LL8C:
    sub.s   $f0, $f0,$f2
    lui     $v1, 0x8000
    cvt.w.s $f0,$f0
    mfc1    $v0, $f0
    or     $v0, $v0,$v1
LL88C:
    sw      $v0, 0x04($fp)
    sw      $zero, 0x08($fp)
    sw      $zero, 0x0C($fp)
    bgez   $zero, CB
    nop
#
LL8A:
    lw      $a0, 0x04($fp)
    addiu  $v0,$zero, 0xCCCCCCCCD
    multu  $a0, $v0
    mfhi   $v0
    srl    $v1, $v0, 3
    addu   $v0,$zero, $v1
    sll    $v0, $v0,2
    addu   $v0, $v0,$v1
    sll1   $v0, $v0,1
    subu   $v1, $a0, $v0
    lw      $v0, 0x0C($fp)
    sllv   $v1, $v1,$v0
    lw      $v0, 0x08($fp)
    addu   $v0, $v1, $v0
    sw      $v0, 0x08($fp)
    lw      $v1, 0x04($fp)
    addiu  $v0,$zero, 0xCCCCCCCCD
    multu  $v1, $v0
    mfhi   $v0
    srl    $v0, $v0,3
    sw      $v0, 0x04($fp)
    lw      $v0, 0x0C($fp)
    addiu  $v0, $v0,1
    sw      $v0, 0x0C($fp)
CB:
    lw      $v1, 0x0C($fp)
    addiu  $v0,$zero, 7
    bne   $v1, $v0, LL8A
    nop
    lw      $v0, 0x08($fp)
    addu   $sp,$zero, $fp
    lw      $fp, 0x14($sp)
    addiu  $sp, $sp,0x18
    jr     $ra
    nop
#-----convert v0 to 0-90deg-----
deg_range_convert:
    slti   $t0,$v0,360
    bne   $t0,$zero,D1
    addi   $v0,$v0,-360
    j      deg_range_convert
# judge if v0 gt 180 and lt 360, $t9 to 1
D1:
    slti   $t0,$v0,180
    bne   $t0,$zero,D2
    addi   $t0,$zero,360
    sub   $v0,$t0,$v0
    addi   $t9,$zero,1
D2:
    slti   $t0,$v0,90
    bne   $t0,$zero,D3
    addi   $t0,$zero,180
    sub   $v0,$t0,$v0
D3:
    jr     $ra
#
.data
flt_400B20: .float 3.1415927
flt_400B24: .float 180.0
flt_400B28: .float 1.0e7
flt_400B2C: .float 2.1474836e9

```

3. 测试用例的机器语言代码

27bdffd0	03e00008	e7c0000c	0043102a	28420007	44020000	00431021	1462ffe0
afb002c	00000000	24020002	1040ffed	1440ffd1	04010006	00021040	00000000
afbe0028	27bdffe0	afc20010	00000000	00000000	00000000	00821823	8fc20008
001df021	afbe001c	0401000e	8fc20008	c7c00004	46020001	8fc2000c	001ee821
8c022ff8	001df021	00000000	30420001	001ee821	3c038000	00021080	8fbe0014
0c0000a2	afc40020	c7c2000c	14400006	8fbe001c	46000024	00431804	27bd0018
afc2001c	afc00004	c7c00014	00000000	27bd0020	44020000	8fc20008	03e00008
8fc4001c	8fc20020	46001002	8fc3000c	03e00008	00431025	00621021	00000000
0c000018	44820000	e7c0000c	3c028000	00000000	afc20004	afc20008	28480168
00000000	468000a0	8fc20010	00621026	27bdffe8	afc00008	8fc30004	15000002
e7c00020	3c020040	44820000	04010002	afbe0014	afc0000c	3c01cccc	2042fe98
c7cc0020	c4002000	46800020	00000000	001df021	0401001e	3421cccc	080000a2
0c000061	46001082	c7c2000c	8fc2000c	e7cc0018	00000000	00011021	284800b4
00000000	3c020040	46001003	afc2000c	c7c20018	8fc40004	00620019	15000003
afc20024	c4002004	e7c0000c	c7c20004	3c020040	3c01cccc	00001010	20080168
ac022ffc	46001003	8fc20010	c7c0000c	c4002008	3421cccc	000210c2	01021022
001ee821	e7c00014	24420001	46001000	46001002	00011021	afc20004	20190001
8fbf002c	24020001	afc20010	e7c00004	c402200c	00820019	8fc2000c	2848005a
8fbe0028	afc20008	8fc20008	8fc20008	4600103e	00001010	24420001	15000002
27bd0030	0401002d	00021040	24420001	45010005	000218c2	afc2000c	200800b4
8c092010	00000000	2442ffff	afc20008	00000000	00031021	8fc3000c	01021022
1920ffff	c7c00014	8fc30010	8fc20008	46000024	00021080	24020007	03e00008

4. 测试用例的运行结果

(1) MARS 汇编仿真软件运行结果

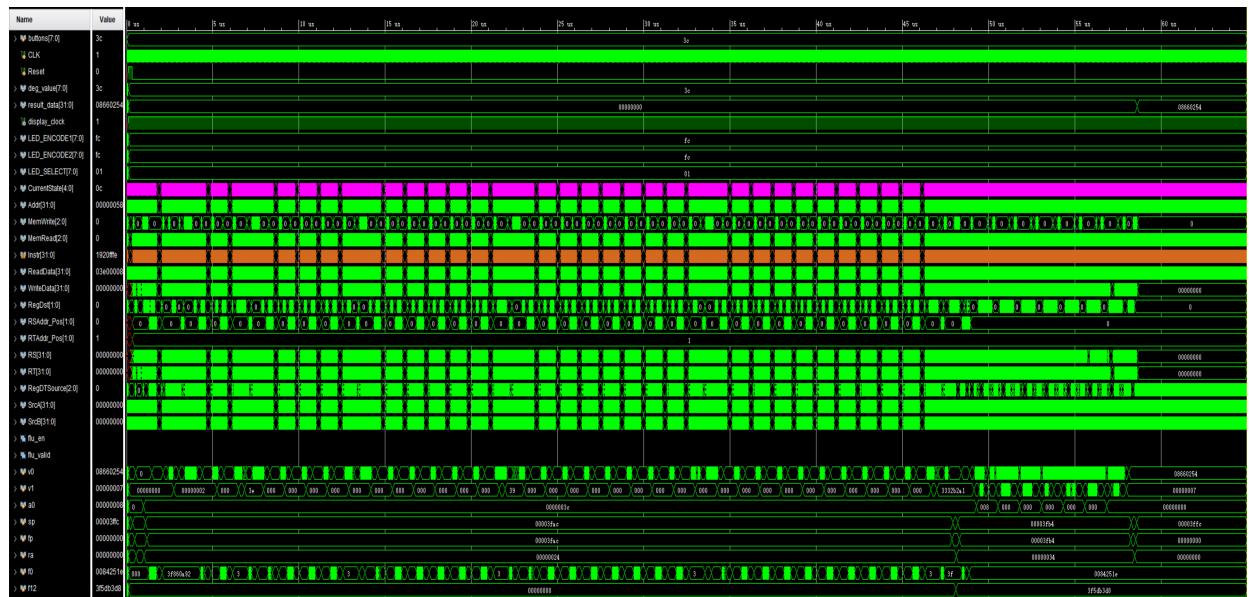
测试输入角度数据: 60 度 ; 预期输出数据: $\sin 60 = 0.8660254$

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0xc0000000
\$v0	2	0x08660254
\$v1	3	0x00000007
\$a0	4	0x00000008
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000001
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$s8	24	0x00000000
\$s9	25	0x00000000
\$s10	26	0x00000000
\$s11	27	0x00000000
\$sp	28	0x00001800
\$sp	29	0x00003ffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00000054
hi		0x00000006
lo		0x66666665
		\$f31

从上图可以看出，程序运行结束后\$v0 存有预期数据 0x08660254，此数据后续将通过 sw 指令存至输出设备中，每 4 位经过 4-10 译码器转化为数码管段选信号并配合位选信号扫描显示出来。浮点寄存器\$f12 存有预期浮点数据 0.8660254 的 IEEE 754 表示 0x3f5db3d8，作为中间运算结果。

(2) 硬件行为仿真结果

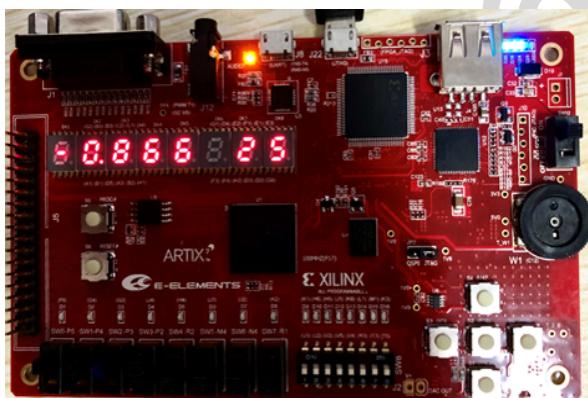
测试输入角度数据: 60 度 ; 预期输出数据: $\sin 60 = 0.8660254$



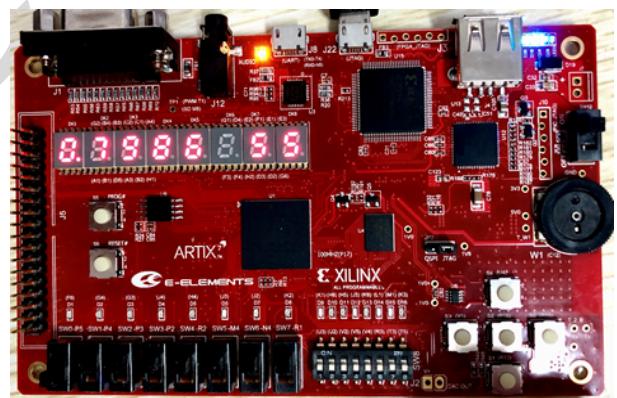
从 v0 和 result_data 的波形中可以看出，程序运行之后获得了稳定的输出 0x08660254

(3) 实际上机调试结果

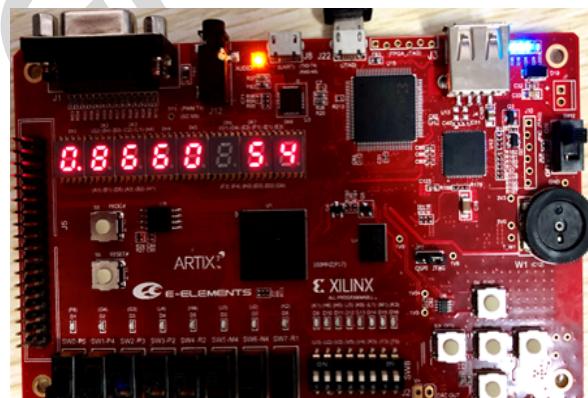
拨码开关输入 1111 0000 即 240 度



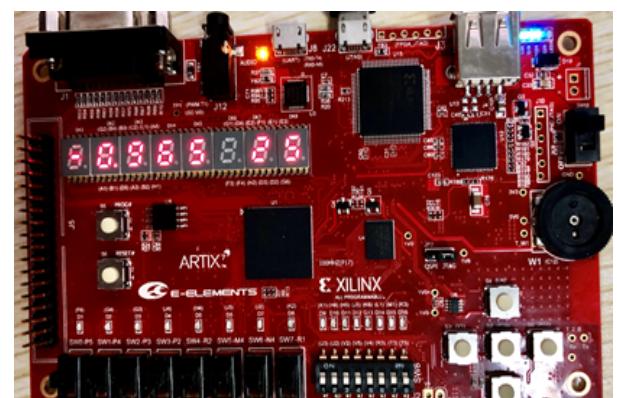
拨码开关输入 0111 1111 即 127 度



拨码开关输入 0011 1100 即 60 度



拨码开关输入 1111 1111 即 255 度



五、设计遇到的问题及解决方法

1. 指令时序问题

(1) 问题描述

在行为仿真的过程中，通过逐条指令的执行分析，会发现前期指令执行正确，而后期则出现大量指令时序混乱、错位等现象，进而导致结果错误。

(2) 问题分析

观察行为仿真的波形图后可以发现，指令执行的各个阶段的信号延迟现象在不断增加，这些延迟在大量累积之后会导致信号时序混乱。

推断出的原因是因为某些不需要边沿触发的信号在等待时钟边沿后才产生，导致其后续信号也延后产生，进而整体时序错位。

深入分析发现运算器解码器模块的 Verilog 代码中大量使用边沿触发的 always 块，而某些控制信号本应通过组合逻辑电路产生。当这些信号改为边沿触发后，本应在同一周期中立即收到控制信号的操作被推迟到下次边沿产生的时刻，从而不能够在本周期立即执行。而当等到控制信号被边沿触发，之前操作的使能信号却已关闭，导致控制信号不能被正确接收，最终指令执行错误。

(3) 解决方法

将 ALUDecoder、MLUDecoder、FLUDecoder 的控制信号改为通过组合逻辑电路产生。即把相应的 always @(posedge CLK) 块改为 always @(*)。

2. 硬件调试无效问题

(1) 问题描述

经过行为仿真成功的工程下载到硬件上后，硬件无法产生正确的结果。并且综合后功能仿真波形图出现大量无效信号。

(2) 问题分析

经过查找 Xilinx 官方论坛上的同类问题，了解到如果设计中使用了某些 IP 核，综合后功能仿真以及时序仿真输出值将可能出现错误的值，只有行为仿真结果正确。

硬件无法输出正确结果是因为在 IO 绑定管脚时，将不需要 Dedicated Clock Buffer 的 Reset 信号绑定在了默认具有 Buffer 功能的管脚上。

(3) 解决方法

在顶层模块端口声明语句之前添加额外的选项告知编译器关闭 Buffer 功能。

```
(* clock_buffer_type="none" *) input Reset
```

六、设计总结及设计心得

GitHub:@Yanzhel