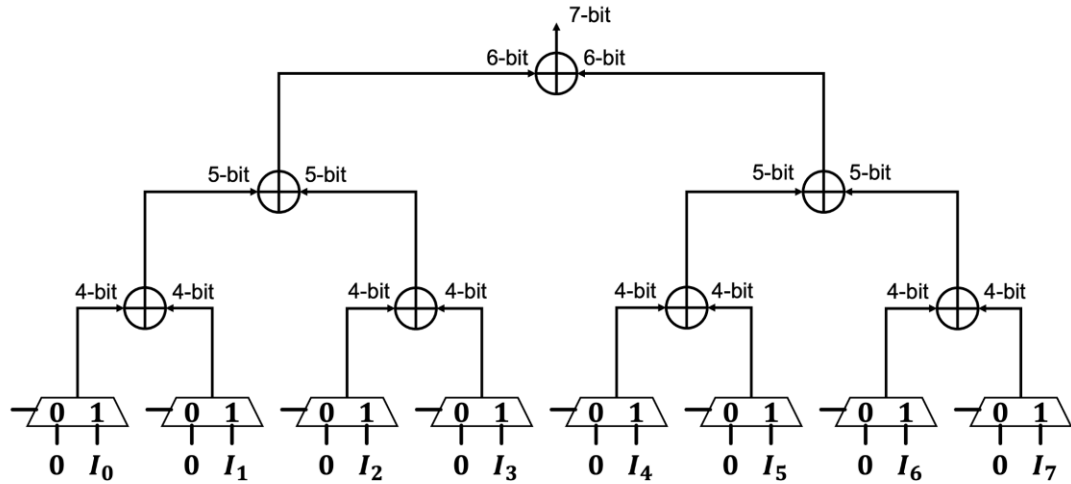


实验报告

实验内容一

设计思路



关键：如何确定每个输入MUX的控制信号

使用提示中的八输入加法树设计，根据 MUX 输入编号为 MUX0~MUX7，MUX 控制信号设置如下：

MUX0: M 和 m 有一个是 0 则选择 I_0 ，否则选择 0，因此 $C_0 = (!M_2!M_1!M_0) + (!m_2!m_1!m_0)$

MUX1: M 和 m 一个小于等于 1 一个大于 1，或者有一个是 1 时，选 I_1 ，因此 $C_1 = ((!M_2!M_1) \oplus (!m_2!m_1)) + (!M_2!M_1M_0) + (!m_2!m_1m_0)$

MUX2: M 和 m 一个小于等于 2 一个大于 2，或者有一个是 2 时，选 I_2 ，因此 $C_2 = (((!M_2!M_1) + (!M_2!M_0)) \oplus ((!m_2!m_1) + (!m_2!m_0))) + (!M_2M_1!M_0) + (!m_2m_1!m_0)$

MUX3: M 和 m 一个小于等于 3 一个大于 3，或者有一个是 3 时，选 I_3 ，因此 $C_3 = ((!M_2) \oplus (!m_2)) + (!M_2M_1M_0) + (!m_2m_1m_0)$

MUX4: M 和 m 一个小于等于 4 一个大于 4，或者有一个是 4 时，选 I_4 ，因此 $C_4 = (((M_2M_1) + (M_2M_0)) \oplus ((m_2m_1) + (m_2m_0))) + (M_2!M_1!M_0) + (m_2!m_1!m_0)$

MUX5: M 和 m 一个小于等于 5 一个大于 5，或者有一个是 5 时，选 I_5 ，因此 $C_5 = ((M_2M_1) \oplus (m_2m_1)) + (M_2!M_1M_0) + (m_2!m_1m_0)$

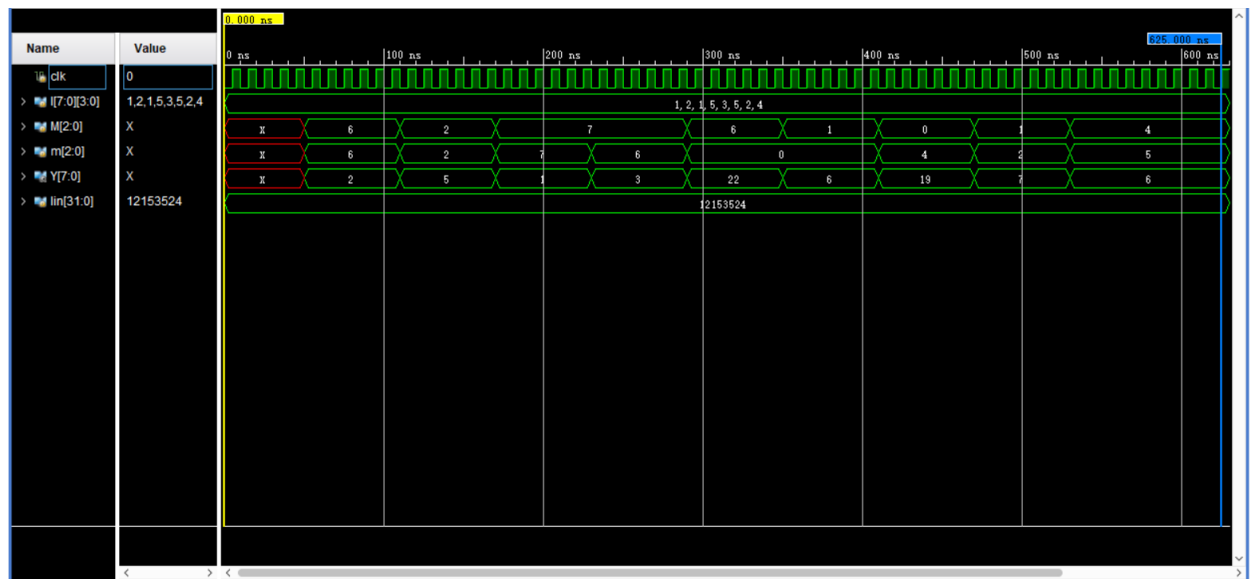
MUX6: M 和 m 一个小于等于 6 一个大于 6，或者有一个是 6 时，选 I_6 ，因此 $C_6 = ((M_2M_1M_0) \oplus (m_2m_1m_0)) + (M_2M_1!M_0) + (m_2m_1!m_0)$

MUX7: M 和 m 有一个是 7 则选择 I_7 ，否则选择 0，因此 $C_7 = (M_2M_1M_0) + (m_2m_1m_0)$

代码逻辑

32 位输入 lin，两个 3 位控制信号 M 和 m，首先将输入的 32 位 lin 拆分成 8 个四位输入，接入到以上述信号控制的 8 个 MUX 中，在 always 语句中完成第一级 4bit 加法得到 MUX0，MUX1 输出的和 s01，MUX2，MUX3 输出的和 s23，MUX4，MUX5 输出的和 s45，MUX6，MUX7 输出的和 s67，之后输入第二级加法器得到结果 s0123 和 s4567，用 assign 语句赋值 $Y=s0123+s4567$ 输出。

验证结果



仿真波形图

```
Tcl Console  x  Messages  Log
Time resolution is 1 ps
Stimulation Start.
Adder Correct.
Adder Correct.
Adder Correct.
Adder Correct.
Adder Correct.
Adder Correct.
Adder Correct.
Adder Correct.
Adder Correct.
$finish called at time : 630 ns : File "E:/vive
```

仿真输出结果，对 3×3 个测试样例设计的加法器都能输出正确结果

实验内容二

设计思路

设计的关键是有限状态机的状态和状态转移，共设计了 1+7+7 个状态，其中包含一个复位态，7 个 $M < m$ 的状态，7 个 $M \geq m$ 的状态，考虑到控制信号 3 位，加上一位表示 M 和 m 大小关系，所以用四位寄存器 `current_state` 记录现态，用四位寄存器 `next_state` 记录次态。复位之后根据 M 和 m 大小，决定次态的最高位是 0 还是 1，次态的后三位由 M 和 m 的最小值提供。特殊情况 $M=m=0$ ，则输出 `I[0]` 结束运算， M 和 m 中只有一个为 0，说明结果一定包含 `I[1]`。根据 M 和 m 大小跳到 `4'b0001` ($M < m$) 或 `4'b1001` ($M \geq m$)，之后继续进行运算。

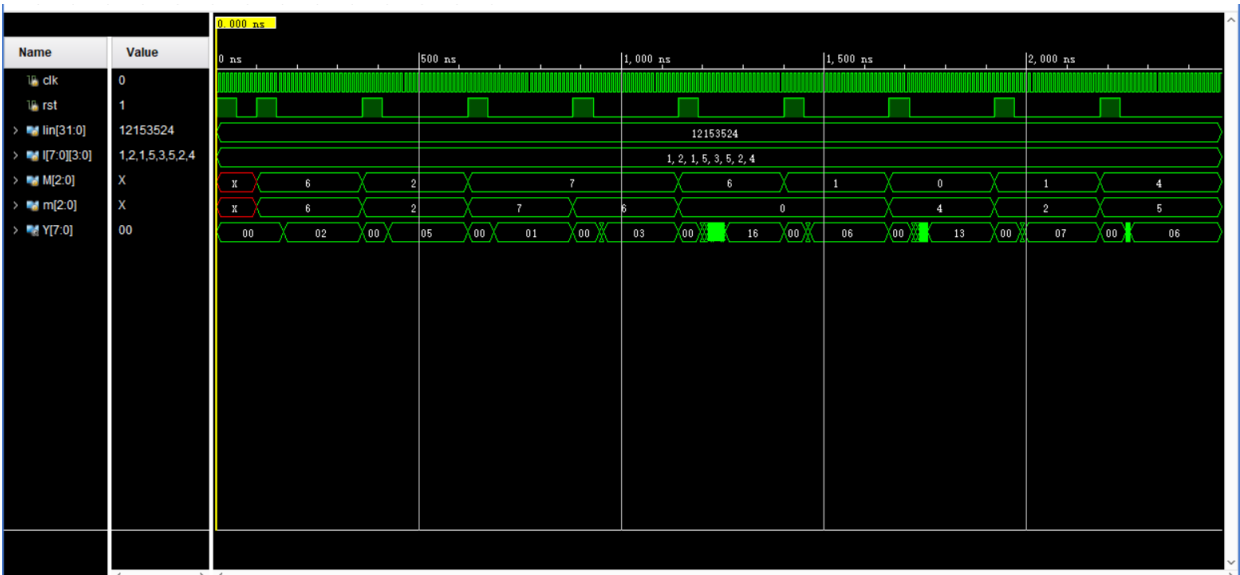
每个状态都要判断 $\max\{M, m\}$ 和当前 state 的后三位是否相等也就是运算是否已经结束决定

次态回到复位态还是向 state+1 转移继续运算。

代码逻辑

32 位输入 lin, 两个 3 位控制信号 M 和 m, 一位 reg add_done 指示运算是否已经结束, 用宏定义将输入的 32 位 lin 拆分成 8 个四位输入, 两个 4bit 指示状态转移变量 current_state 和 next_state. 1bit 时钟输入以及 1bit 异步复位信号输入。如果复位信号为 1, 则现态和次态都赋 4'b0000, 输出 Y 赋 0, add_done 赋 0。使用 case 语句实现状态转移, 如果复位信号为 0 并且 add_done 也是 0 则进入状态转移语句, 复位态 4'b0000, 进入之后先按一般情况比较 M 和 m 大小, $M < m$ 则次态设置为 {1'b0, M}, 否则次态设置为 {1'b1, m}, 之后检查 M 和 m 中是否有 0, $M = 0$ 说明输出结果要加 I[0], 直接把 I[0]赋给 Y, 之后判断 m 是不是 0, 若是 0 则 add_done 置 1, 直接结束运算, 否则说明 $m \geq 1$, 次态设置为 4'b0001; 如果 M 不是 0, 检查 m 是不是 0, 是 0 则把 I[0]赋给 Y, 次态设置为 4'b1001。其余状态编码的最高位指示 M 和 m 的大小关系, 此状态应该加的 I 由后三位决定, 每个状态在加和之后都要检查 $\max\{M, m\}$ 是否等于当前加和的 I 编号, 以决定是否结束运算。Case 语句之后将 next_state 的值赋给 current_state, 完成状态转移。

验证结果



仿真波形图

```
# run 1000ns
Stimulation Start.
Adder Correct.
Adder Correct.
Adder Correct.
INFO: [USF-XSim-96] XSim completed. Design snapshot 'adder_temp_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
run 100 ms
Adder Correct.
Adder Correct.
Adder Correct.
Adder Correct.
Adder Correct.
Adder Correct.
```

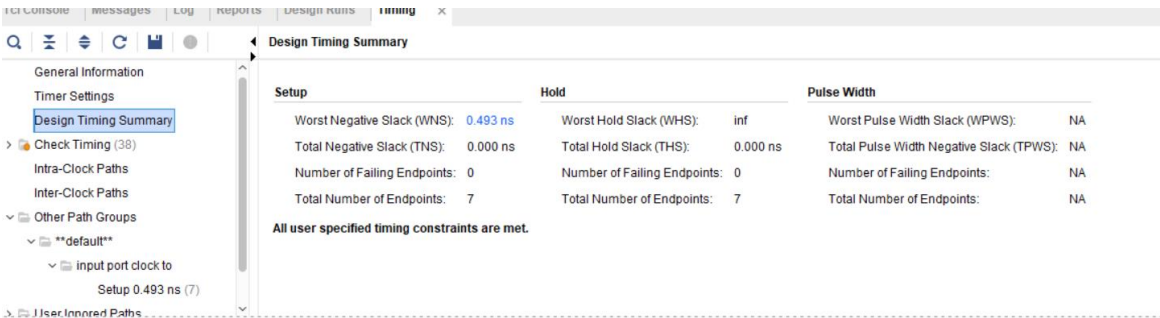
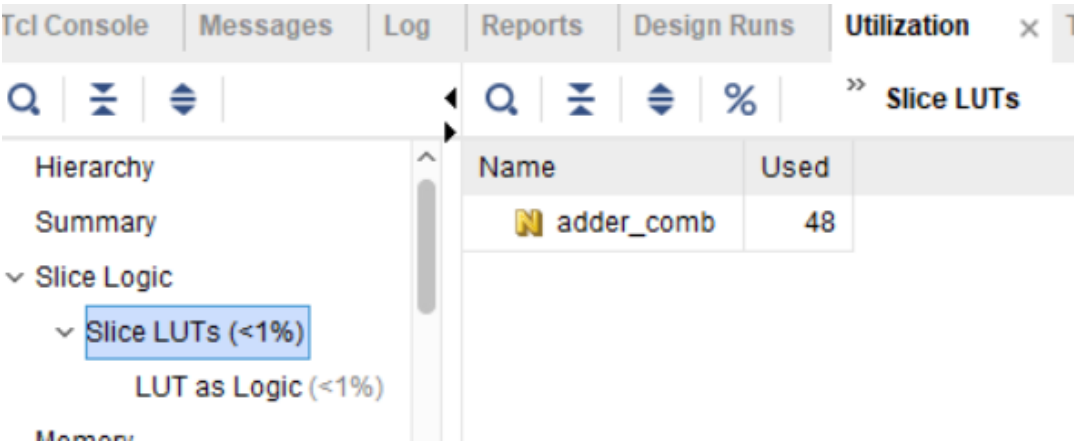
仿真输出结果, 对 3×3 个测试样例时序逻辑设计的加法器都能输出正确结果

对两种方式设计加法器的 STA

1. 组合逻辑

资源占用

逻辑资源:



设置 M, m 和 lin 到 Y 的 max_delay 为 10ns 后的时序性能

Name	Slack	Levels	Routes	High Fanout	From	To	Total ...	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Excep
Path 7	2.115	7	8	8	M[2]	Y[0]	7.885	4.426	3.459	10.0	input port clock		MaxC
Path 6	1.462	7	8	8	M[2]	Y[1]	8.538	4.410	4.128	10.0	input port clock		MaxC
Path 5	1.070	6	7	8	M[2]	Y[2]	8.930	4.612	4.318	10.0	input port clock		MaxC
Path 4	1.006	6	7	8	M[2]	Y[3]	8.994	4.676	4.318	10.0	input port clock		MaxC
Path 3	0.804	8	9	8	M[1]	Y[4]	9.196	4.815	4.381	10.0	input port clock		MaxC
Path 2	0.687	8	9	8	M[1]	Y[5]	9.313	4.932	4.381	10.0	input port clock		MaxC
Path 1	0.493	7	8	8	M[1]	Y[6]	9.507	4.740	4.767	10.0	input port clock		MaxC

单次计算最低延时7.885ns

t_{clk-q} 最大9.507ns, $t_{logic(max)}$ 最大4.740ns, $t_{su} = 0.493ns$, 由建立时间约束, 最大时

间频率 $f_{clk} = \frac{1}{t_{clk-q} + t_{logic(max)} + t_{su}} = \frac{1}{14.740n} = 67.84MHz$

2. 时序逻辑

资源占用

逻辑资源：

Tcl Console	Messages	Log	Reports	Design Runs	Utilization	Timing
			» LUT as Logic			
Hierarchy			Name			
Summary			Used			
▼ Slice Logic			adder_temp 47			
▼ Slice LUTs (<1%)						
LUT as Logic (<1%)						
▼ Slice Registers (<1%)						
Register as Flip Flop (
F8 Muxes (<1%)						
F7 Muxes (<1%)						
Memory						

Tcl Console	Messages	Log	Reports	Design Runs	Utilization	Timing
			» Register as Flip Flop			
Hierarchy			Name			
Summary			Used			
▼ Slice Logic			adder_temp 17			
▼ Slice LUTs (<1%)						
LUT as Logic (<1%)						
▼ Slice Registers (<1%)						
Register as Flip Flop (
F8 Muxes (<1%)						
F7 Muxes (<1%)						
Memory						

寄存器使用：

Design Timing Summary			
General Information			
Timer Settings			
Design Timing Summary			
Clock Summary (1)			
Check Timing (47)			
Intra-Clock Paths			
Inter-Clock Paths			
Setup		Hold	
Worst Negative Slack (WNS): 2.419 ns		Worst Hold Slack (WHS): 0.233 ns	
Total Negative Slack (TNS): 0.000 ns		Total Hold Slack (THS): 0.000 ns	
Number of Failing Endpoints: 0		Number of Failing Endpoints: 0	
Total Number of Endpoints: 34		Total Number of Endpoints: 34	
		Pulse Width	
		Worst Pulse Width Slack (WPWS): 4.500 ns	
		Total Pulse Width Negative Slack (TPWS): 0.000 ns	
		Number of Failing Endpoints: 0	
		Total Number of Endpoints: 18	

设置 M, m 和 lin 到 Y 的 max_delay 为 5ns 后的时序性能

Intra-Clock Paths - ckin - Setup													
Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Dest	
Path 8	3.349	4	5	12	m[0]	current_state_reg[2]/D	3.457	1.409	2.049	10.0	ckin	ckin	
Path 9	3.349	4	5	12	m[0]	next_state_reg[2]/D	3.457	1.409	2.049	10.0	ckin	ckin	
Path 10	3.379	6	7	1	lin[20]	Y_reg[0]/D	3.694	2.132	1.563	10.0	ckin	ckin	
Path 7	3.190	6	7	1	lin[20]	Y_reg[1]/D	3.883	2.320	1.564	10.0	ckin	ckin	
Path 6	2.899	6	7	1	lin[21]	Y_reg[0]/D	4.174	2.471	1.704	10.0	ckin	ckin	
Path 5	2.832	7	8	1	lin[21]	Y_reg[4]/D	4.241	2.670	1.572	10.0	ckin	ckin	
Path 4	2.707	7	8	1	lin[21]	Y_reg[5]/D	4.366	2.794	1.573	10.0	ckin	ckin	
Path 3	2.667	6	7	1	lin[21]	Y_reg[3]/D	4.406	2.537	1.870	10.0	ckin	ckin	
Path 2	2.664	7	8	1	lin[21]	Y_reg[6]/D	4.409	2.697	1.713	10.0	ckin	ckin	
Path 1	2.419	7	8	1	lin[21]	Y_reg[7]/D	4.654	2.776	1.879	10.0	ckin	ckin	

单次计算最低延时3.457ns，最高4.654ns

Timing Summary - timing_1													
Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Dest	
Path 4	2.707	7	8	1	lin[21]	Y_reg[5]/D	4.366	2.794	1.573	10.0	ckin	ckin	
Path 1	2.419	7	8	1	lin[21]	Y_reg[7]/D	4.654	2.776	1.879	10.0	ckin	ckin	
Path 2	2.664	7	8	1	lin[21]	Y_reg[6]/D	4.409	2.697	1.713	10.0	ckin	ckin	
Path 5	2.832	7	8	1	lin[21]	Y_reg[4]/D	4.241	2.670	1.572	10.0	ckin	ckin	
Path 3	2.667	6	7	1	lin[21]	Y_reg[3]/D	4.406	2.537	1.870	10.0	ckin	ckin	
Path 6	2.899	6	7	1	lin[21]	Y_reg[2]/D	4.174	2.471	1.704	10.0	ckin	ckin	
Path 7	3.190	6	7	1	lin[20]	Y_reg[1]/D	3.883	2.320	1.564	10.0	ckin	ckin	
Path 10	3.379	6	7	1	lin[20]	Y_reg[0]/D	3.694	2.132	1.563	10.0	ckin	ckin	
Path 8	3.349	4	5	12	m[0]	current_state_reg[2]/D	3.457	1.409	2.049	10.0	ckin	ckin	
Path 9	3.349	4	5	12	m[0]	next_state_reg[2]/D	3.457	1.409	2.049	10.0	ckin	ckin	

$t_{logic(max)}$ 最大 2.794ns, $t_{su} = 2.491ns$, 由建立时间约束, 最大时间频率 $f_{clk} =$

$$\frac{1}{t_{clk-q} + t_{logic(max)} + t_{su}} = \frac{1}{14.740n} = 100.61MHz$$

综上, 组合逻辑设计使用的 LUT 更少, 但是 wns 和最高时钟频率方面较差; 相对的, 时序逻辑用的 LUT 更多但是 wns 和最高时钟频率方面较组合逻辑设计优秀。

实验内容三

设计思路

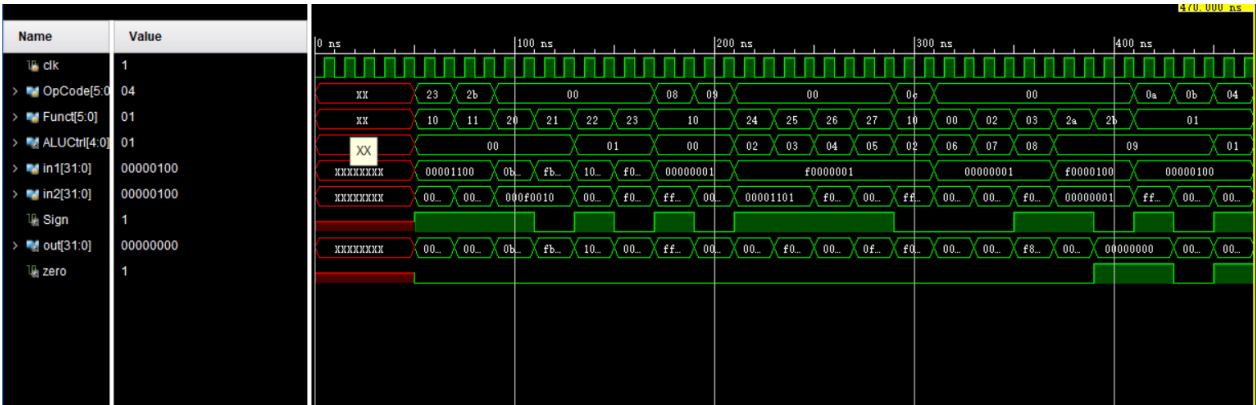
对经过 ALU 的指令进行了分类, 即 add、addu、sub、subu、and、er、xor、noe、sll、srl、sra、slt、sltu、lw、sw、addi、addiu、andislti、sltiu 和 beq。以及这些指令对 ALU 的功能需求, 包括加减法 (包括有无符号运算, 虽然不考虑溢出的情况有无符号数的加减法都是一样的, 区别主要在 i 形指令需要的立即数扩展上, 但为了 testbench 中区分有无符号数结果在输出时区分了有符号和无符号的情况); 按位与; 按位或; 按位异或; 按位或非; 左移; 逻辑右移; 算数右移和 slt 系列的四个语句, 根据输入的 Sign 信号 0 还是 1 决定有还是无符号比较。ALU 功能的选择依赖于上一级 ALUController 输出的控制信号 ALUCtrl, 决定 ALU 功能。ALUCtrl 由输入的 Opcode 和 Funct 决定, 根据指令类型决定控制信号赋值。

代码逻辑

ALUController 输入 6bit Opcode 和 6bit Funct, 输出 5bit ALUCtrl 和 1bit Sign。用 case 语句给 ALUCtrl 和 Sign 赋值。Opcode 等于零说明是 R 型指令, 具体功能要看 Funct 的值, 因此再加一个 case 语句选择 Funct。其他指令可以根据 Opcode 确定。进入下一级 ALU 根

据 ALUCtrl 选择对应的功能，根据 Sign 选择有无符号运算，用 assign 语句持续赋值 zero，输出 out=0 就赋给 zero 1'b1，否则赋 1'b0。

验证结果



仿真波形图

```
Stimulation Start.
lw Correct.
sw Correct.
add Correct.
addu Correct.
sub Correct.
subu Correct.
addi Correct.
addiu Correct.
and Correct.
or Correct.
xor Correct.
nor Correct.
andi Correct.
sll Correct.
srl Correct.
sra Correct.
slt Correct.
sltu Correct.
slti Correct.
sltiu Correct.
beq Correct.
```

仿真结果，对设计的各条指令能正确输出