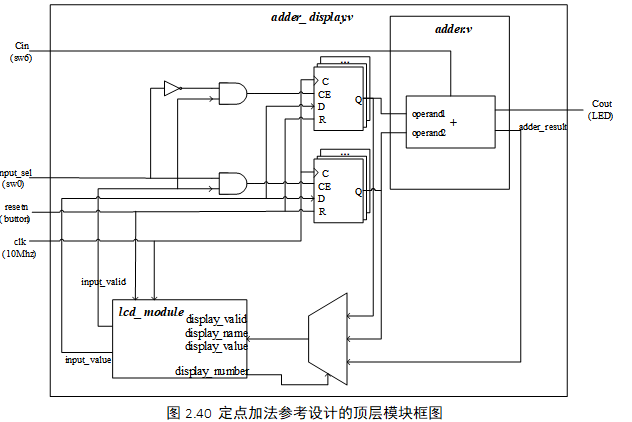
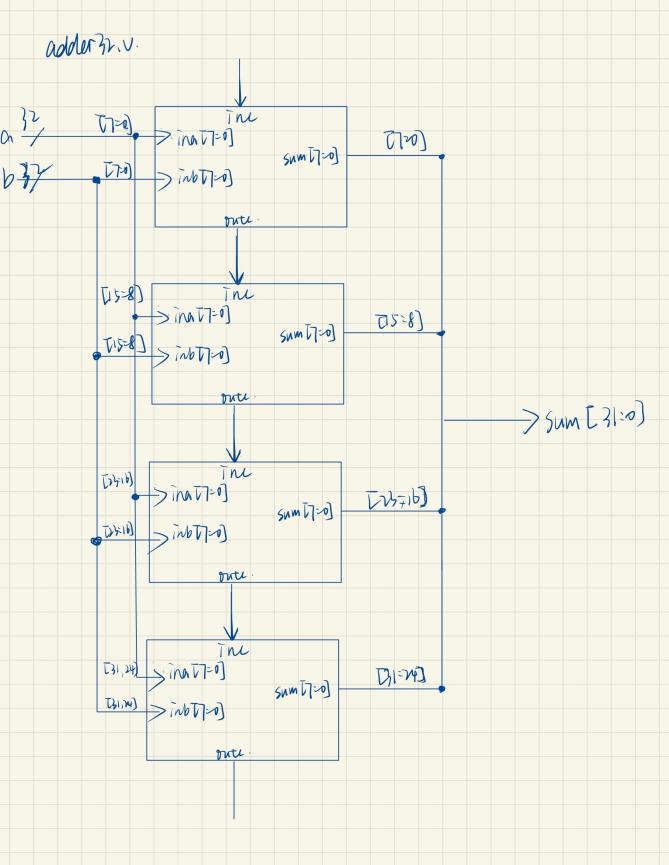
**组成原理实验课程第 1 次实验报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实验名称 | 数据运算：定点加法 | | | 班级 | 李涛老师 |
| 学生姓名 | 孙蕗 | 学号 | 2112060 | 指导老师 | 董前琨 |
| 实验地点 | A306 | | 实验时间 | 2023.3.21 | |

1. **实验目的**
2. 熟悉LS-CPU-EXB-002实验箱和软件平台。
3. 掌握利用该实验箱各项功能开发组成原理和体系结构实验的方法。
4. 理解并掌握加法器的原理和设计。
5. 熟悉并运用verilog语言进行电路设计。
6. 为后续设计cpu的实验打下基础。
7. **实验内容说明**
8. 实验设备
9. 装有Xilinx Vivado的计算机一台。
10. LS-CPU-EXB-002教学系统实验箱一套。
11. 实验任务
12. 阅读LS-CPU-EXB-002实验箱相关文档，熟悉硬件平台,特别需要掌握利用显示屏观察特定信号的方法。学习软件平台和设计流程。
13. 熟悉计算机中加法器的原理。
14. 自行设计本次实验的方案，画出结构框图，详细标出输入输出端口，本次实验的加法器可以使用全加器自己搭建加法模块，也可以在verilog中直接使用“+”（系统是自动调用库里加法IP，且面积时序更优），依据教师要求选择一种方法实现。
15. 根据设计的实验方案，使用verilog编写相应代码。
16. 对编写的代码进行仿真，得到正确的波形图。
17. 将以上设计作为一个单独的模块，设计一个外围模块去调用该模块。外围模块中需调用封装好的触摸屏模块，显示两个加数和加法结果，且需要利用触摸功能输入两个加数。
18. 将编写的代码进行综合布局布线，并下载到实验箱中的FPGA板上进行演示。
19. 不要在前三格显示。
20. 实验要求
21. 了解软硬件平台；
22. 掌握定点加法的工作原理；
23. 确定定点加法的输入输出端口设计；
24. 在课前画好设计框图或实验原理图；
25. 如果对FPGA板了解的话，可确定设计中与FPGA板上交互的接口，画出包含外围模块的整体设计框图，即补充完善图2.1。
26. 实验实施：
27. 确认定点加法的设计框图的正确性；
28. 编写verilog代码；
29. 对该模块进行仿真，得出正确的波形，截图作为实验报告结果一项的材料；
30. 完成调用定点加法模块的外围模块的设计，并编写代码；
31. 对代码进行综合布局布线下载到实验箱里FPGA板上，进行上板验证。
32. 实验检查：
33. 完成上板验证后，让指导老师或助教进行检查，进行现场演示，可对演示结果进行拍照作为实验报告结果一项的材料。
34. 实验报告的撰写：
35. 实验结束后，需按照规定的格式完成实验报告的撰写。
36. 本次实验
37. 设计两个模块，8位加法器和32位加法器，其中32位加法器通过调用8位加法器实现。
38. 针对32位加法器进行仿真验证
39. 针对32位加法器进行上实验箱验证
40. 要求不从前三格输出
41. **实验原理图**





1. **实验步骤**
2. 编写adder8模块，8位全加器

|  |
| --- |
| module adder8( |
| input [7:0] ina,//一个八位的数字输入 |
| input [7:0] inb,//一个八位的数字输入 |
| input inc,//一个进位输入 |
| output [7:0] sum,//一个八位的加法结果 |
| output outc//一个向高位的进位 |
| ); |
| assign {outc,sum}= ina+inb+inc;//8位全加器 |
| endmodule |

有2个8位数的输入和1个进位输入，产生1个8位的加法和结果和1个向高位的进位。

1. 编写8位全加器功能仿真testbench

|  |
| --- |
| module testbench1; |
| reg[7:0] op1,op2; |
| reg op; |
| wire[7:0] sum; |
| wire flag; |
| adder8 uut(op1,op2,op,sum,flag);//op1是第一个8位数，op2是第二个8位数，op是进位，sum是8位加法运算和，flag是进位标志 |
| initial//可类比C++的构造函数 |
| begin |
| op1=8'b0;op2=8'b0;op=1'b0; |
| end |
| always #3 op1=$random%9'b1\_0000\_0000;//always语句对后面的语句敏感，#3代表延长3个时间单位，合用意味着每隔3个时间单位，…。32位数，取模的作用是取后8位 |
| always #5 op2=$random%9'b1\_0000\_0000;//每隔5个时间单位，得到一个8位随机数 |
| always #7 op=$random%2'b1\_0; |
| endmodule |

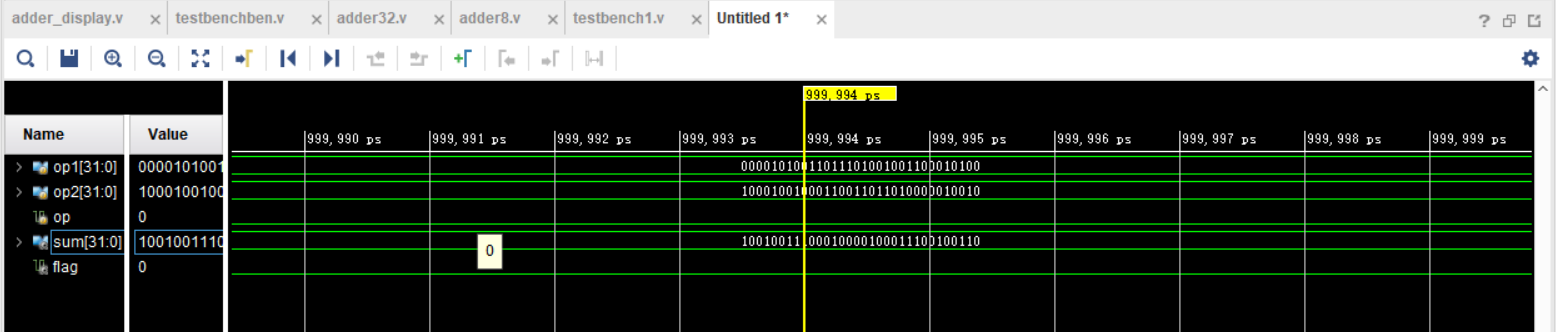
1. 编写adder32模块，32位全加器，32位加法器通过调用8位加法器实现。

|  |
| --- |
| module adder32( |
| input [31:0] ina,//32位数输入 |
| input [31:0] inb,//32位数输入 |
| input inc,//进位输入 |
| output [31:0] sum,//一个32位加法结果 |
| output outc,//一个32位加法向高位的进位 |
| wire [2:0] c//每一步的进位 |
| ); |
| adder8 u1 (.ina(ina[7:0]),.inb(inb[7:0]),.inc(inc),.outc(c[0]),.sum(sum[7:0]));//前8位数加法，分别取两个输入的前8位数参与运算，进位是进位输入inc，输出的是前8位运算的加法结果和进位输出，进位输出记录在c[0]里，加法运算结果记录在sum的前8位 |
| adder8 u2 (.ina(ina[15:8]),.inb(inb[15:8]),.inc(c[0]),.outc(c[1]),.sum(sum[15:8]));//第9-16位数加法，分别取两个输入的第9-16位数参与运算，进位是c[0]，输出的是第9-16位运算的加法结果和进位输出，进位输出记录在c[1]里，加法运算结果记录在sum的第9-16位 |
| adder8 u3 (.ina(ina[23:16]),.inb(inb[23:16]),.inc(c[1]),.outc(c[2]),.sum(sum[24:16]));//第17-24位数加法，分别取两个输入的第17-24位数参与运算，进位是c[1]，输出的是第17-24位运算的加法结果和进位输出，进位输出记录在c[2]里，加法运算结果记录在sum的第17-24位 |
| adder8 u4 (.ina(ina[31:24]),.inb(inb[31:24]),.inc(c[2]),.outc(outc),.sum(sum[31:24]));//第25-32位数加法，分别取两个输入的第25-32位数参与运算，进位是c[2]，输出的是第25-32位运算的加法结果和进位输出，进位输出记录在outc里，加法运算结果记录在sum的第25-32位 |
| endmodule |

1. 32位全加器功能仿真testbench

|  |
| --- |
| module testbenchben; |
| reg[31:0] op1,op2;//两个32位数输入 |
| reg op;//进位输入 |
| wire[31:0] sum;//一个32位加法结果 |
| wire flag;//一个32位加法向高位的进位 |
| adder32 uut(op1,op2,op,sum,flag);//op1是第一个32位数，op2是第二个32位数，op是进位，sum是32位加法运算和，flag是进位标志 |
| initial//可类比C++的构造函数 |
| begin |
| op1 = 0; |
| op2 = 0; |
| op = 0; |
| // Wait 100 ns for global reset to finish |
| #100; |
| // Add stimulus here |
| end |
| always #10 op1 = $random; //$random为系统任务，产生一个随机的32位数 |
| always #10 op2 = $random; //#10 表示等待10个单位时间(10ns)，即每过10ns，赋值一个随机的32位数 |
| always #10 op = {$random} % 2; //加了拼接符，{$random}产生一个非负数，除2取余得到0或1 |
| endmodule |

1. adder32，32位全加器，仿真验证，调成二进制数便于观察



调成二进制数便于检验，第一个32位数op1是0000 1010 0110 1110 1001 0011 0001 0100，第二个32位数op2是1000 1001 0001 1001 1011 0100 0001 0010，进位op是0，32位运算结果sum是1001 0011 1000 1000 0100 0111 0010 0110，进位输出flag是0，结果是正确的。

代码编辑和功能仿真都已完成，认为功能基本正确，后续流程就是上板验证了。

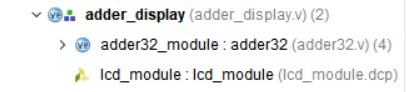
1. 添加外围展示模块，adder\_diaplay文件。对adder\_display文件进行修改,只展示修改部分的代码。

|  |
| --- |
| //-----{调用加法模块}begin |
| reg [31:0] adder32\_ina; |
| reg [31:0] adder32\_inb; |
| wire adder32\_inc; |
| wire [31:0] adder32\_sum; |
| wire adder32\_outc; |
| adder32 adder32\_module |
| ( |
| .ina(adder32\_ina), |
| .inb(adder32\_inb), |
| .inc(adder32\_inc), |
| .sum(adder32\_sum), |
| .outc(adder32\_outc) |
| ); |
| assign adder32\_inc = sw\_cin; |
| assign led\_cout = adder32\_outc; |
| //-----{调用加法模块}end |

|  |
| --- |
| //-----{从触摸屏获取输入}begin |
| //根据实际需要输入的数修改此小节， |
| //建议对每一个数的输入，编写单独一个always块 |
| //当input\_sel为0时，表示输入数为加数1，即adder32\_ina |
| always @(posedge clk) |
| begin |
| if (!resetn) |
| begin |
| adder32\_ina <= 32'd0; |
| end |
| else if (input\_valid && !input\_sel) |
| begin |
| adder32\_ina <= input\_value; |
| end |
| end |
|  |
| //当input\_sel为1时，表示输入数为加数2，即adder32\_inb |
| always @(posedge clk) |
| begin |
| if (!resetn) |
| begin |
| adder32\_inb <= 32'd0; |
| end |
| else if (input\_valid && input\_sel) |
| begin |
| adder32\_inb <= input\_value; |
| end |
| end |
| //-----{从触摸屏获取输入}end |

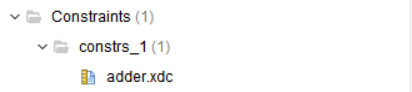
|  |
| --- |
| //-----{输出到触摸屏显示}begin |
| //根据需要显示的数修改此小节， |
| //触摸屏上共有44块显示区域，可显示44组32位数据 |
| //44块显示区域从1开始编号，编号为1~44， |
| always @(posedge clk) |
| begin |
| case(display\_number) |
| 6'd6 ://从第6格开始输出 |
| begin |
| display\_valid <= 1'b1; |
| display\_name <= "ADD\_1"; |
| display\_value <= adder32\_ina; |
| end |
| 6'd7 ://第7格 |
| begin |
| display\_valid <= 1'b1; |
| display\_name <= "ADD\_2"; |
| display\_value <= adder32\_inb; |
| end |
| 6'd8 ://第8格 |
| begin |
| display\_valid <= 1'b1; |
| display\_name <= "RESUL"; |
| display\_value <= adder32\_sum; |
| end |
| default : |
| begin |
| display\_valid <= 1'b0; |
| display\_name <= 40'd0; |
| display\_value <= 32'd0; |
| end |
| endcase |
| end |
| //-----{输出到触摸屏显示}end |
| //----------------------{调用触摸屏模块}end---------------------// |

1. 添加lcd\_module.dcp



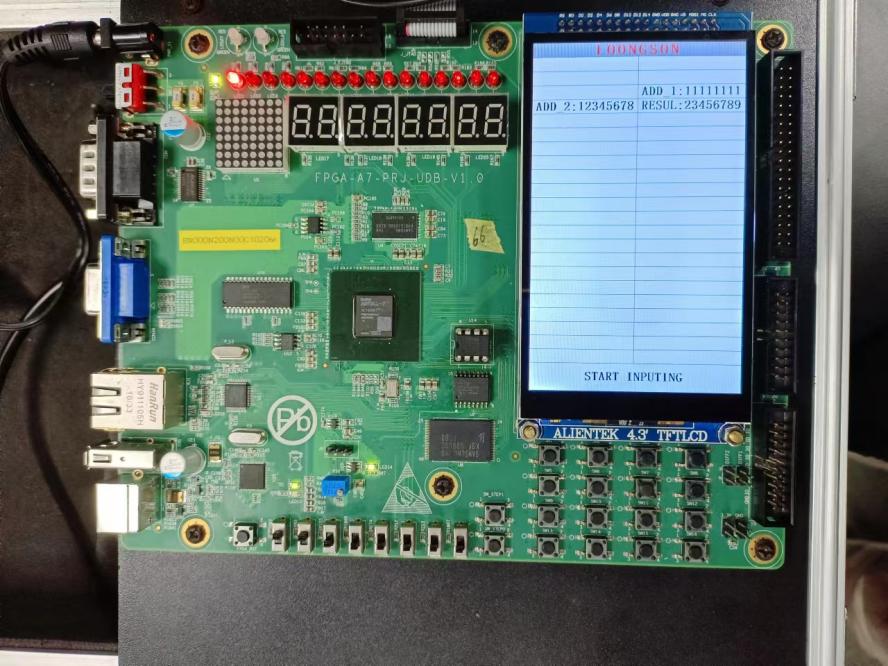
LS-CPU-EXB-002配套资源设计时，将lcd\_module模块封装为一个黑盒的网表文件，只需要调用即可。

1. 添加约束文件adder.xdc

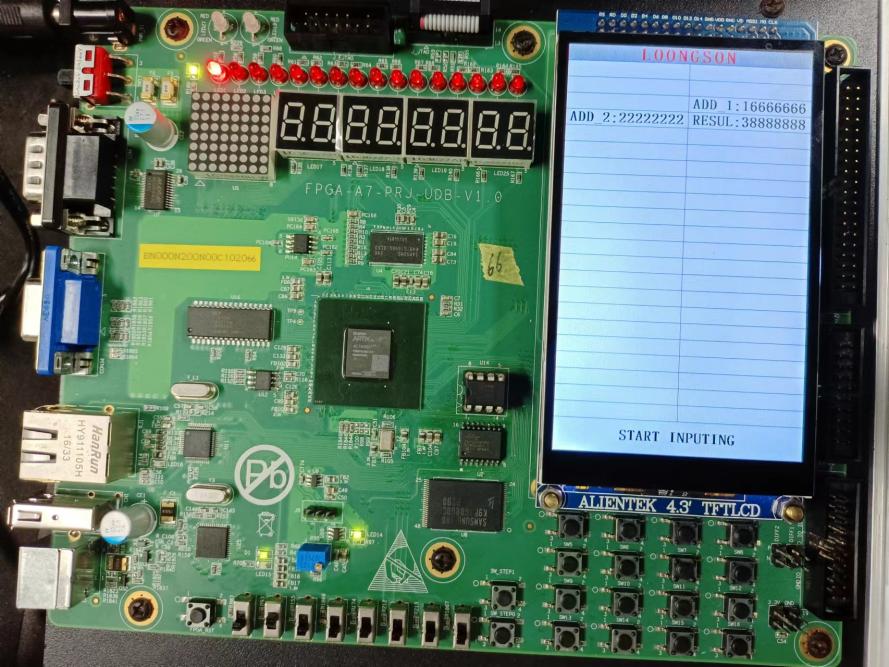


约束文件后缀名为.xdc，用“Add or create constraints”添加或创建约束文件。

1. **实验结果分析**

****

第一个输入为1111 1111，第二个输入为1234 5678，进位标志0，输出结果为2345 6789。



第一个输入为1666 6666，第二个输入为2222 2222，进位标志0，输出结果为3888 8888。

1. **总结感想**

本次实验主要是学习使用verilog语言的语法和结构，理解计算机硬件的基本构建块，并学习使用vivado软件。先正确编写8位全加器并验证功能，再之后使用4个8位全加器构建成一个32位全加器，实现更高位数的加法，这样可以减少后续检查错误需要消耗的时间。通过不断地测试和调试，最终实现了32位全加器的功能。verilog的每个模块都应该有清晰的输入和输出端口，并且应该有良好的模块化设计,学会了如何使用仿真工具来验证代码的正确性，确保代码可以按照预期运行。通过将 8 位全加器组合成 32 位全加器，我理解了如何将更复杂的电路设计分解为更简单的模块，并将它们组合成更大的系统.