龙芯中科
LOONGSON TECHNOLOGY

龙芯教学实验平台 LS-CPU-EXB-002
CPU 设计与体系结构实验指导手册

V1.0

北京市海淀区中关村环保科技示范园龙芯产业园2号楼
Building No. 2, Loongson Industrial Park, Zhongguancun Environmental Protection Park, Haidian District Beijing 100095, P. R. China
www.loongson.cn

目录

一、 LS-CPU-EXB-002 CPU 设计与体系结构教学实验系统介绍

LS-CPU-EXB-002 CPU 设计与体系结构教学实验系统是根据高等院校计算机专业本科生及研究生相关专业开设的《数字逻辑》、《计算机组成原理》、《计算机体系结构》等计算机基础软硬件课程的实验教学需要，自主研发，具有自主知识产权的集成实验设计、开发与教学平台。该实验系统围绕计算机系统的基础软硬件专业能力培养，可以满足不同层次院校开设上述课程实验的验证型、综合型、创新型实验教学要求。

实验平台的基本结构和软硬件结构图如下所示。

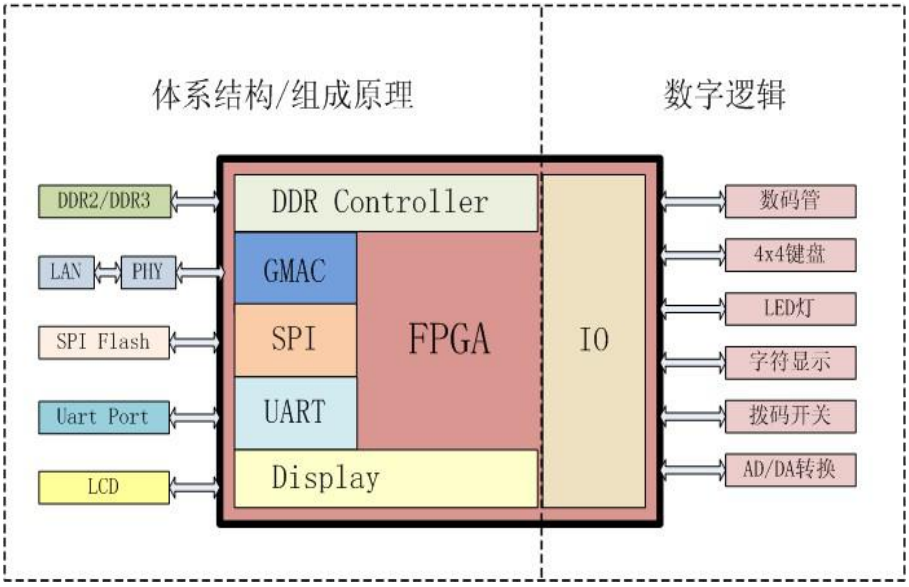


图 1.1　FPGA 实验平台框架图

实验箱里配套资源有

——FPGA 实验板；

——Xilinx 的下载线；

——串口线（含 USB 串口转化器）；

——电源线（含适配器）。

为了使在实验板上演示实验成为可能，LS-CPU-EXB-002 试验箱实现了 LCD 触摸屏的硬件驱动，使得不需要处理器核就能使用触摸屏的显示和输入功能，且设计了简单清晰的调用接口方便学生使用。

为降低学生在 FPGA 硬件平台上的实际操作难度，方便教师指导，特针对组成原理和体系结构课程编写了本实验指导手册。指导手册主要分为两部分，cpu 设计部分和 cpu 优化部分：前者主要为组成原理需要，包含实验一到实验七，以及课程设计；后者主要为体系结构需要，包含课程设计和课程设计拓展。

具体实验内容如下：

实验一 数据运算：定点加法(软硬件平台入门)

实验二 数据运算：定点乘法

实验三 寄存器堆实现

实验四 ALU 模块实现

实验五 存储器实现

实验六 单周期 CPU 实现

实验七 多周期 CPU 实现

课程设计 静态 5 级流水线 CPU 实现

课程设计拓展 优化 CPU 系统

二、 实验一 数据运算：定点加法

1 实验目的

1. 熟悉 LS-CPU-EXB-002 实验箱和软件平台。
2. 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。
3. 理解并掌握加法器的原理和设计。
4. 熟悉并运用 verilog 语言进行电路设计。
5. 为后续设计 CPU 的实验打下基础。

2 实验设备

1. 装有 Xilinx Vivado 的计算机一台。
2. LS-CPU-EXB-002 教学系统实验箱一套。

3 实验任务

1. 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。

2. 熟悉计算机中加法器的原理。

3. 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。

4. 根据设计的实验方案，使用 verilog 编写相应代码。

5. 对编写的代码进行仿真，得到正确的波形图。

6. 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，见图 2.1。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。

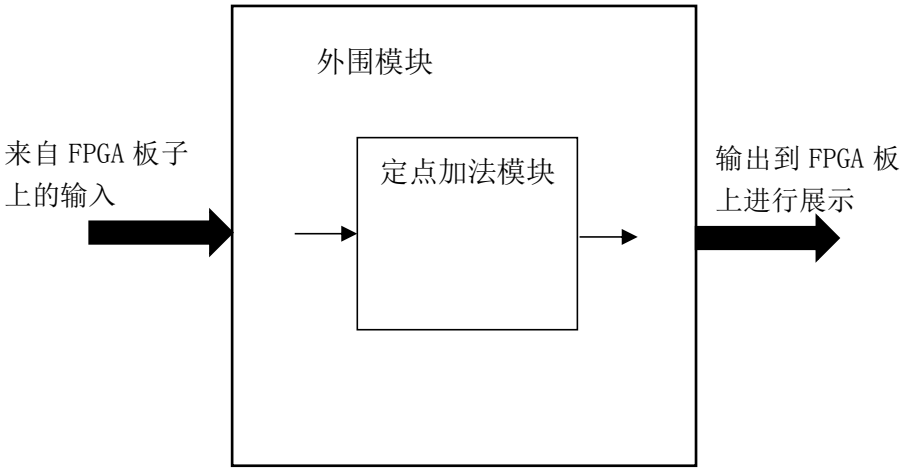


图 2.1 定点加法设计实验的顶层模块大致框图

7. 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。

4 实验要求

1. 做好预习：
   1） 了解软硬件平台；
   2） 掌握定点加法的工作原理；
   3） 确定定点加法的输入输出端口设计；
   4） 在课前画好设计框图或实验原理图；
   5） 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。
2. 实验实施：
   1） 确认定点加法的设计框图的正确性；
   2） 编写 verilog 代码；
   3） 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；
   4） 完成调用定点加法模块的外围模块的设计，并编写代码；
   5） 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。
3. 实验检查：
   1） 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。
4. 实验报告的撰写：
   1）实验结束后，需按照规定的格式完成实验报告的撰写。

5 参考实现

本部分给出了定点加法实验的参考设计方案，详细阐述了软硬件平台的使用方法，特别是 FPGA 实验板上的 LCD 触摸屏的调用模块的使用。

5.1 实验步骤

1) 新建工程

启动 vivado 软件，在菜单栏点击“File”->”New Project”，出现新建工程向导，选择“Next”，输入工程名称，选择工程的文件位置，然后选择“Next”：

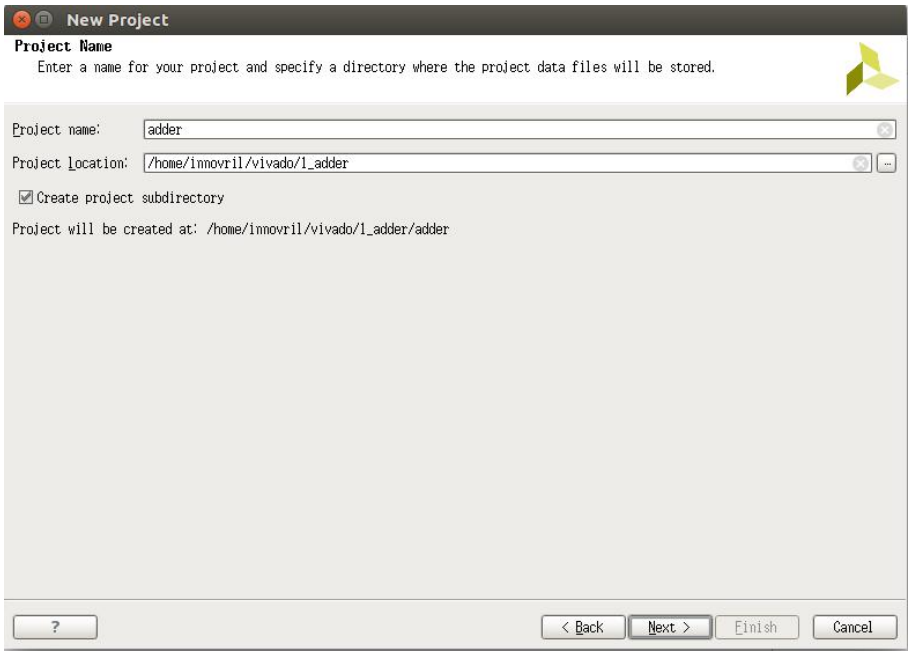


图 2.2 设置工程名称和位置

选择”RTL Project”，勾选”Do not specify sources at this time”，点击”Next”：

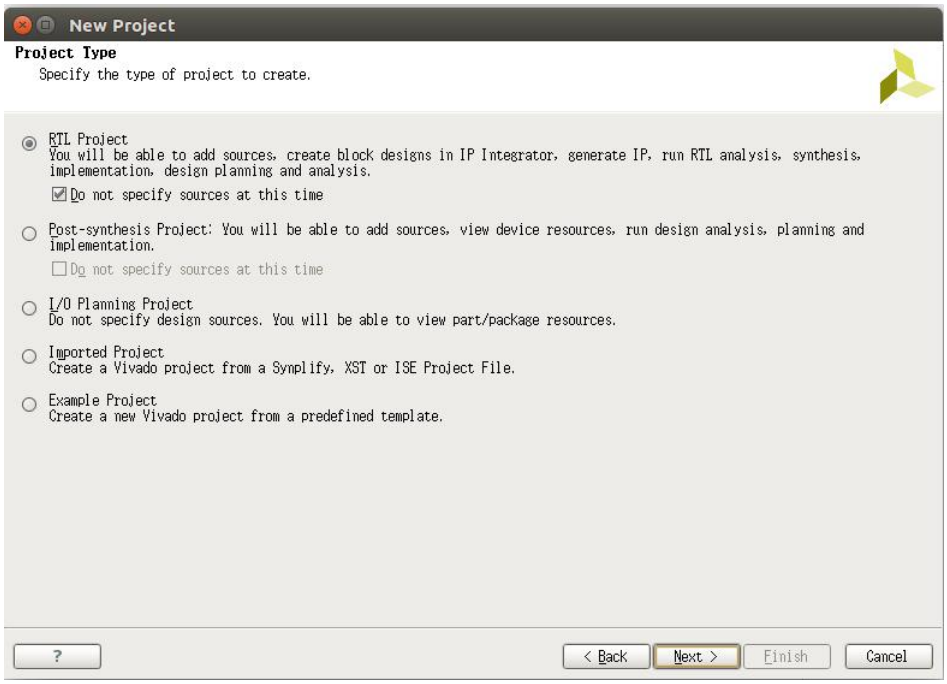


图 2.3 工程设置

在筛选器的”family”选择”Artix 7”，”package”选择”fbg676”，在筛选得到的型号里面选择”xc7a200tfbg676-2”：

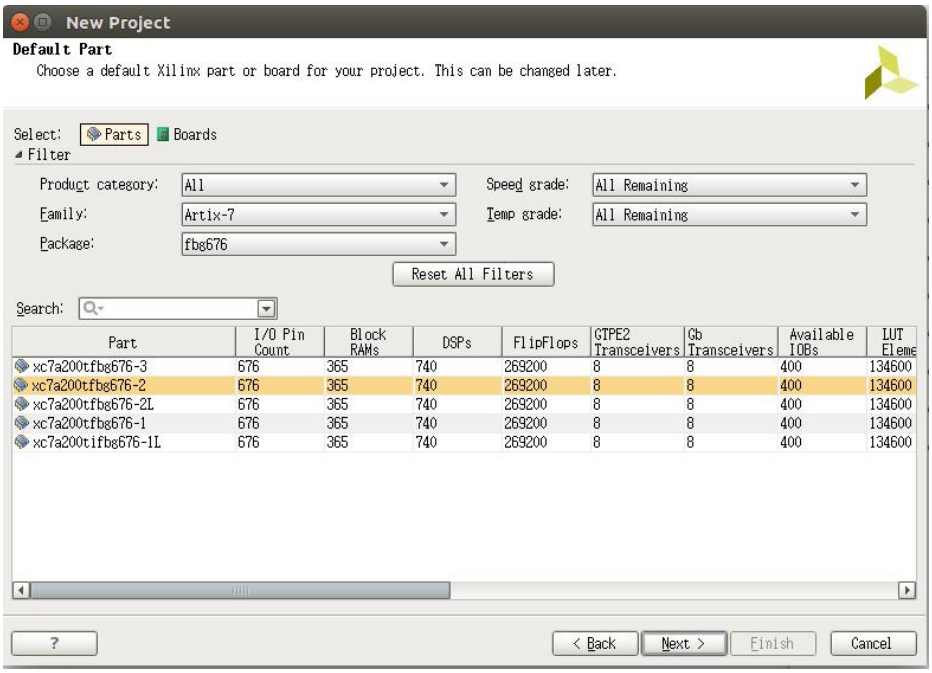


图 2.3 工程设置

然后选择“Next”，出现下图界面，点击“Finish”。

图 2.4 工程信息

2) 添加源文件

verilog 代码都是以“.v”为后缀名的文件，可以在其他文件编辑器里写好，再添加到新建的工程中，也可以在工程中新建一个再编辑。

添加已有 verilog 文件的方法如下：在”Project Manager”下点击”Add sources”，选择”Add or create design sources”

图 2.5 添加源文件

点击”Next”，可以看到如下的界面，可以按文件添加或者按文件夹添加，也可以自己创建：

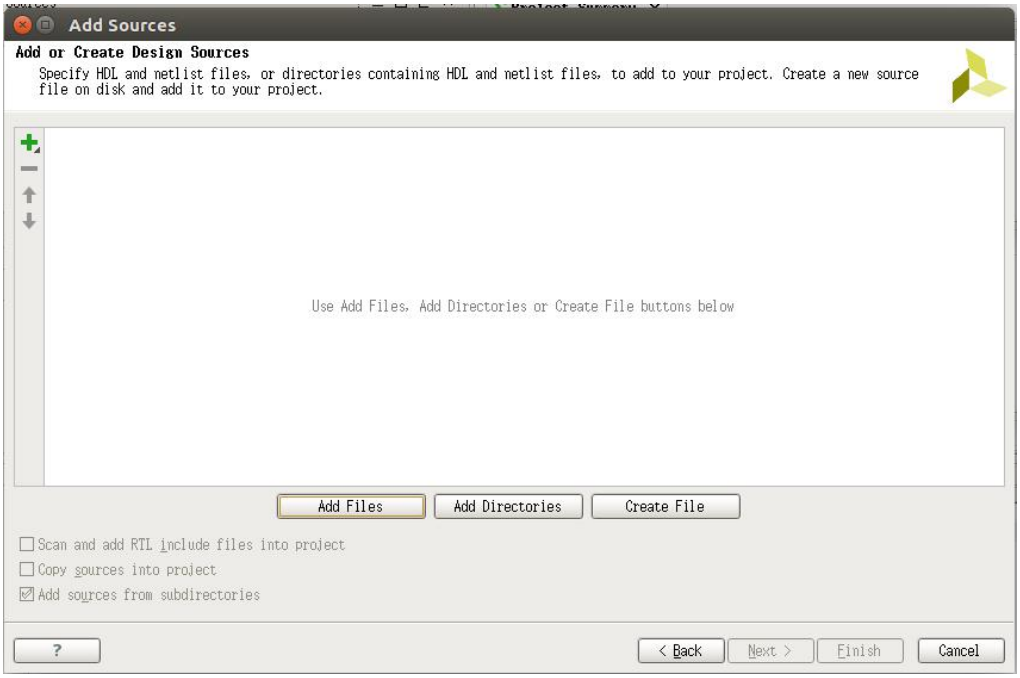


图 2.6 添加源文件

点击”Add Files”，选择 adder.v，点击”OK”:

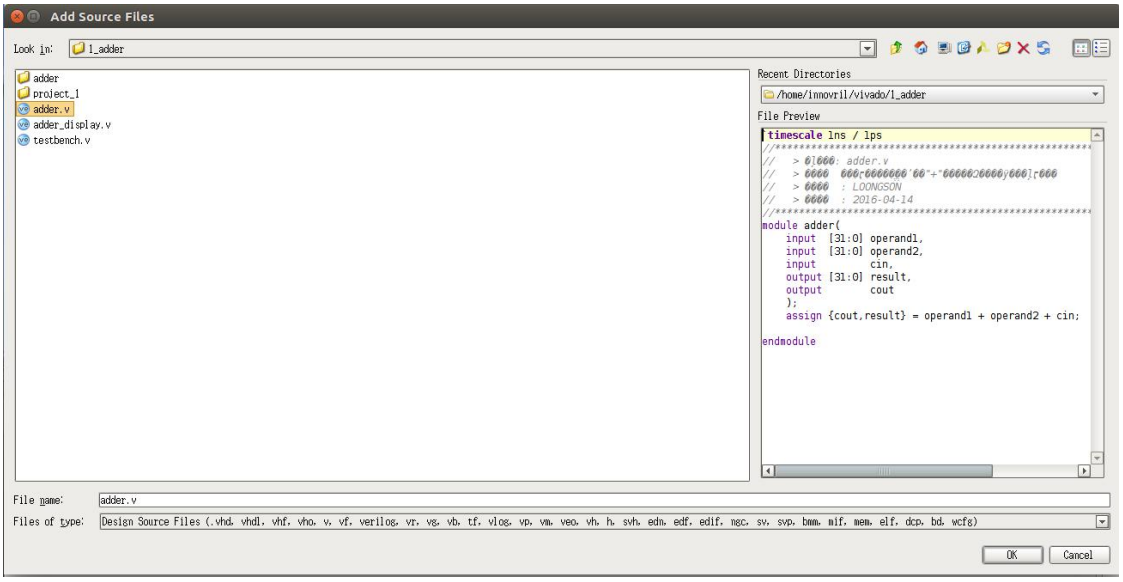


图 2.6 添加源文件

adder.v 的代码如下：

```
1  `timescale 1ns / 1ps
2  //*************************************************************
3  //    > 文件名: adder.v
4  //    > 描述   : 加法器，直接使用"+"，会自动调用库里的加法器
5  //    > 作者   : LOONGSON
6  //    > 日期   : 2016-04-14
7  //*************************************************************
8  module adder(
9      input  [31:0] operand1,
10     input  [31:0] operand2,
11     input         cin,
12     output [31:0] result,
13     output        cout
14     );
15     assign {cout,result} = operand1 + operand2 + cin;
16
17 endmodule
```

代码比较简单，有 2 个 32 位数的输入和 1 个进位输入，产生 1 个 32 位的加法和结果和 1 个向高位的进位。本实验提供的参考设计是直接写“+”号实现加法功能的，这样的写法综合工具会调用内部的模块库的加法器来实现，往往会比自行设计的加法模块更高效和省资源。

本例中，adder.v 为定点加法实验的主体代码，由于实验较简单，故只有这一个.v 文件，对于以后的 CPU 实验，则会有多个.v 文件，形成一定的调用层次。

3) 添加展示外围模块

在本例中，按照实验要求还需要一个外围模块，该外围模块调用 adder.v，且要调用触摸屏模块的以便在板上实验结果，本例中将该模块命名为 adder_display.v，添加该模块到工程里，结果如下图：

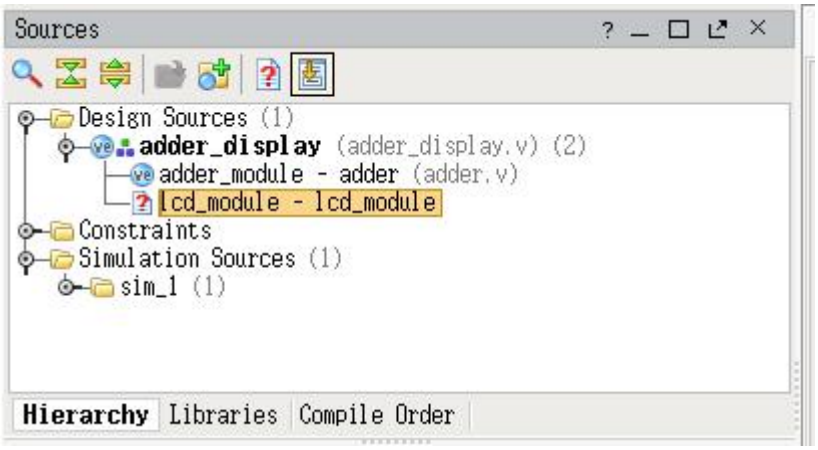


图 2.7 工程管理区

在工程管理区可以看到各模块间的层次关系，本例中顶层模块即为 adder_display，里面调用了两个子模块：一个为 adder，即定点加法主体代码；一个为 lcd_module，即 LCD 触摸屏的模块。外围展示模块 adder_display.v 具体代码解析见 5.2 小节，以后所有实验的外围展示模块都可以仿照 adder_display.v 编辑。

在图 2.7 中，lcd_module 前面打了问号，表示该模块还未添加。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，使得学生不会过分关注而导致迷失在 LCD 触摸屏实现中，只需要调用即可。继续选择“Add Source”，添加 lcd_module.dcp，使用 vivado2012.3 版本软件工具时，需先解压 dcp 文件后再添加，如下图：

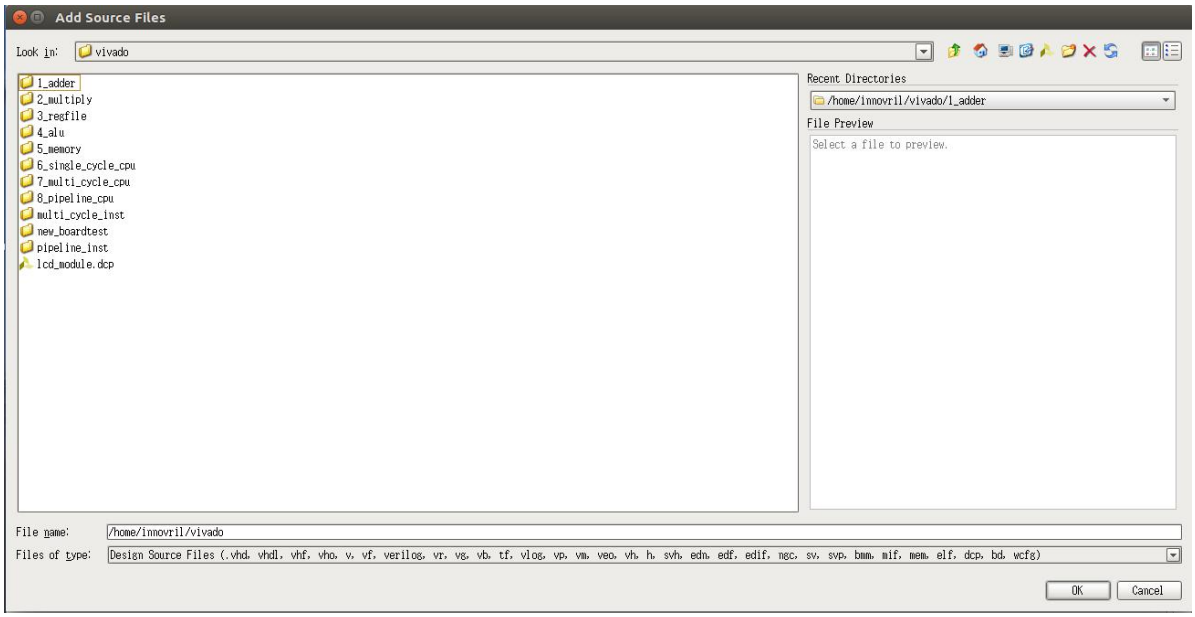


图 2.8 添加 lcd_module 文件

添加成功后，结果如下图：

图 2.9 添加 lcd_module 文件成功

至此，代码实现都已经完成，下面需要对代码功能进行仿真验证功能的正确性。

4) 功能仿真

在进行功能仿真时，需要先建立一个 testbench(测试平台)。一个比较完备的 testbench 就是产生输入激励，送入到要测试的功能模块里，然后读出功能模块的执行结果，与预期的结果进行比较，以此验证功能模块的正确性。但在目前的实验设计上，只需要一个最简单的 testbench——只产生输入激励即可。

在本例中，需要产生的输入激励就是 2 个加数和 1 个低位进位信号，在该激励输入到加法功能模块中后，会输出加法结果和向高位的进位信号。仿真的过程中会产生波形文件，可以通过观察波形文件确定功能的正确性，在出错的情况下可以定位错误位置。

点击“Add Source”，选择”Add or create simulation sources”，添加“testbench.v”。

testbench.v 的代码如下：

```
1   `timescale 1ns / 1ps  //仿真单位时间为 1ns，精度为 1ps
2   module testbench;
3
4       // Inputs
5       reg [31:0] operand1;
6       reg [31:0] operand2;
7       reg cin;
8
9       // Outputs
10      wire [31:0] result;
11      wire cout;
12      // Instantiate the Unit Under Test (UUT)
13      adder uut (
14          .operand1(operand1),
15          .operand2(operand2),
16          .cin(cin),
17          .result(result),
18          .cout(cout)
19      );
```

```
20      initial begin
21          // Initialize Inputs
22          operand1 = 0;
23          operand2 = 0;
24          cin = 0;
25          // Wait 100 ns for global reset to finish
26          #100;
27          // Add stimulus here
28      end
29      always #10 operand1 = $random;  //$random 为系统任务，产生一个随机的 32 位数
30      always #10 operand2 = $random;  //#10 表示等待 10 个单位时间(10ns)，即每过 10ns，赋值一个随机的 32 位数
31      always #10 cin = {$random} % 2; //加了拼接符，{$random}产生一个非负数，除 2 取余得到 0 或 1
32  endmodule
```

添加 testbench.v 后，工程管理区如图所示，如果 testbench.v 前面没有三个方形的 top 标志，则右键点击 testbench.v，选择”Set as Top”：

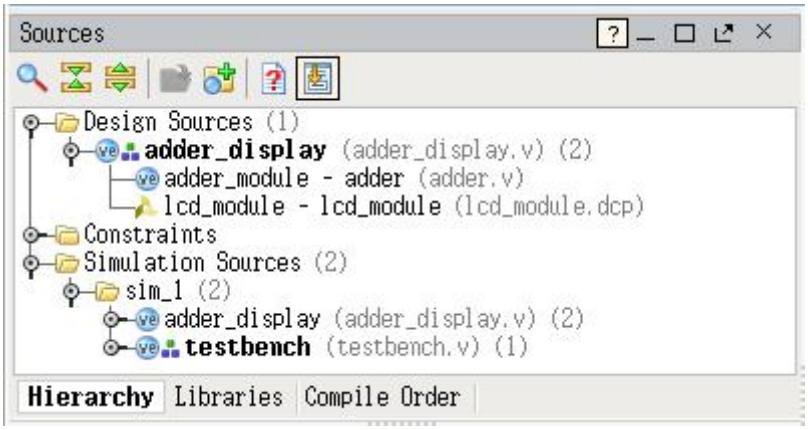


图 2.10 添加 testbench.v 成功

在左侧的导航栏中点击”Run Simulation”，选择”Run Behavioral Simulation”：

图 2.11 运行仿真

如果没有语法错误，会弹出如下的界面：

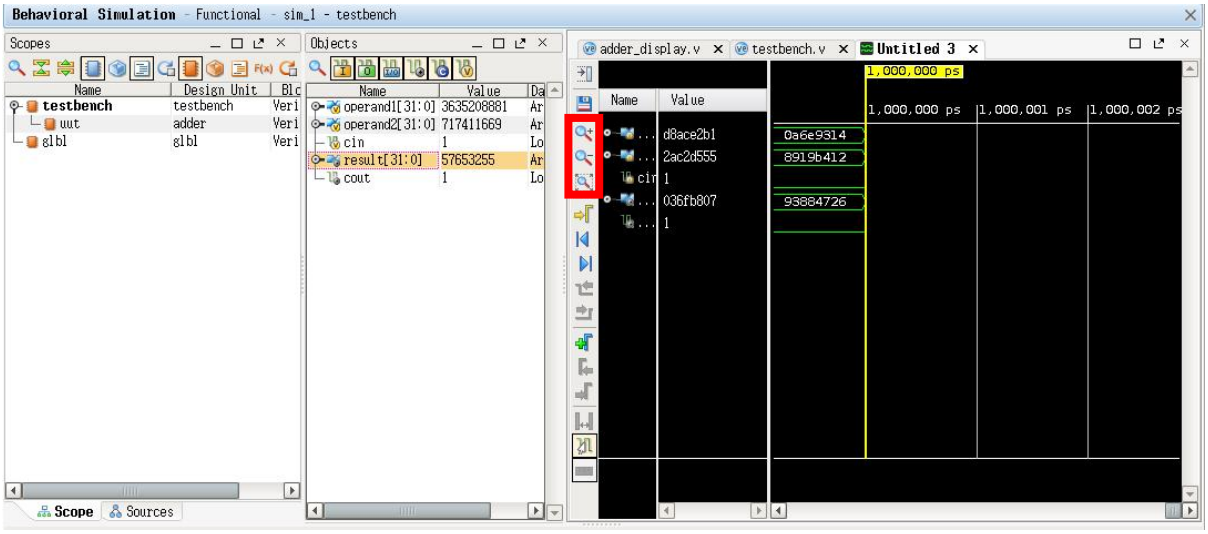


图 2.12 仿真界面

可以通过图 2.12 中红色圈出的放大、缩小、缩放到全屏三个按钮以及鼠标左键点击波形界面(出现图中黄线，缩放会以黄线为中心)观察特定波形区域的信息。

图 2.13 仿真工具栏

上图是仿真工具栏，从左到右依次是”从 0 时刻开始仿真”、”运行仿真”、”运行特定时长的仿真”、”仿真时长”、”时间单位”、”单步仿真”、”暂停”、”重新载入仿真”。

在仿真界面左侧的“Scopes”(图 2.12 的最左列)选择要观察信号所在源文件对象，然后在“Objects”窗口(图 2.12 的第二列)可以看到该对象里所有的信号，可以左键选中要观察的信号，然后右键会出现很多可选项，选择“Add To Wave Window”添加该信号到波形窗口，如下图。

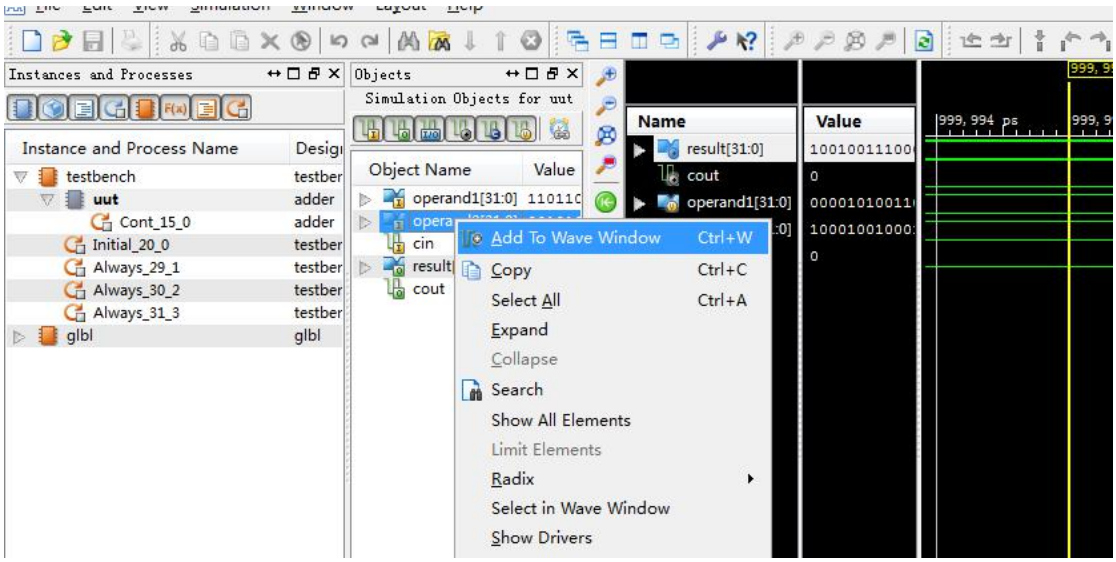


图 2.14 添加要观察的信号到波形界面

在添加新的信号到波形窗口后，需要点击 Relaunch 才能看到该信号的数据。由于定点加法模块内部所有信号均为输入输出信号，故此处没有新的信号可以添加。但在以后的实验中，会经常需要添加新的信号到波形窗口。

另外，当前波形窗口显示的数据均为 2 进制数，对于 32 位的数不便于观察，可以在波形窗口的 Name 列左键选择信号，然后右键会出现多很多可选项，选择“Radix”可以选择要数据显示的进制。

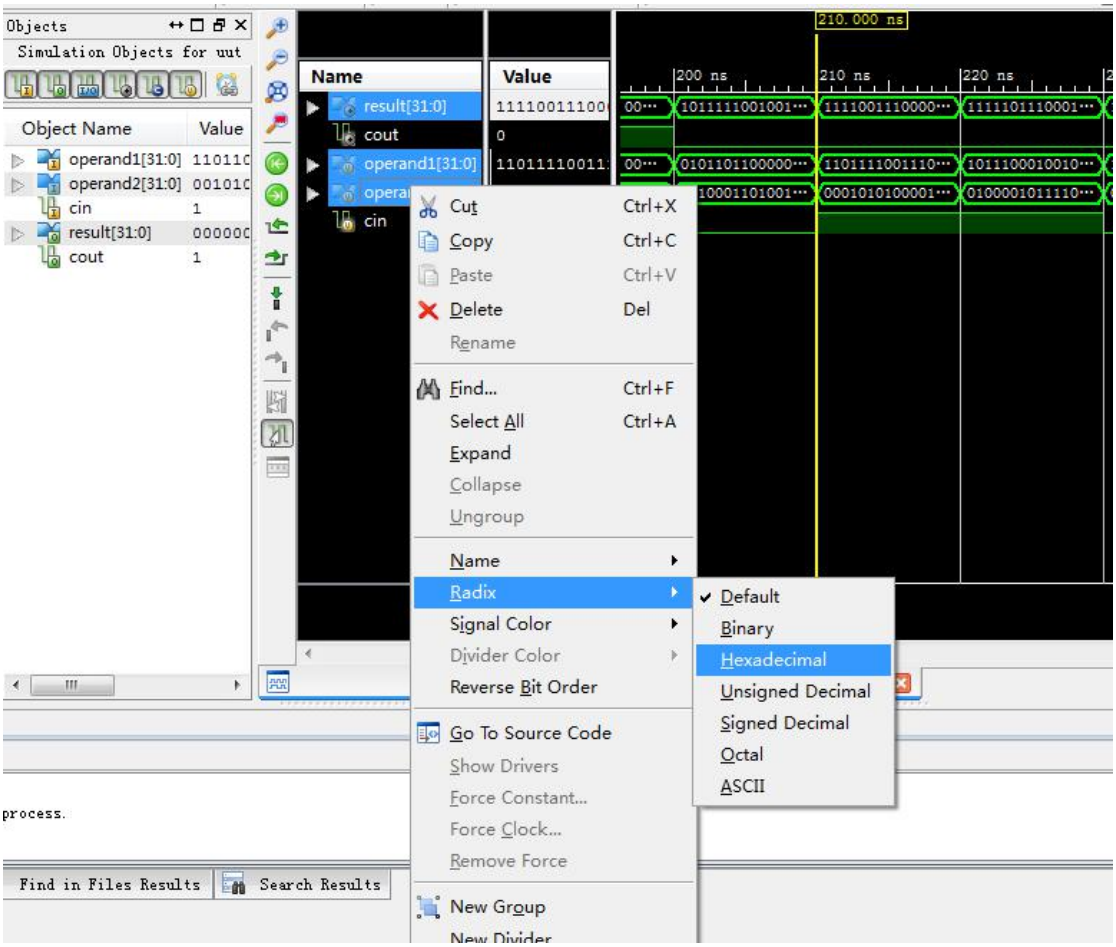


图 2.15 更改波形窗口信号的显示进制

此处选择 16 进制，结果如下图：

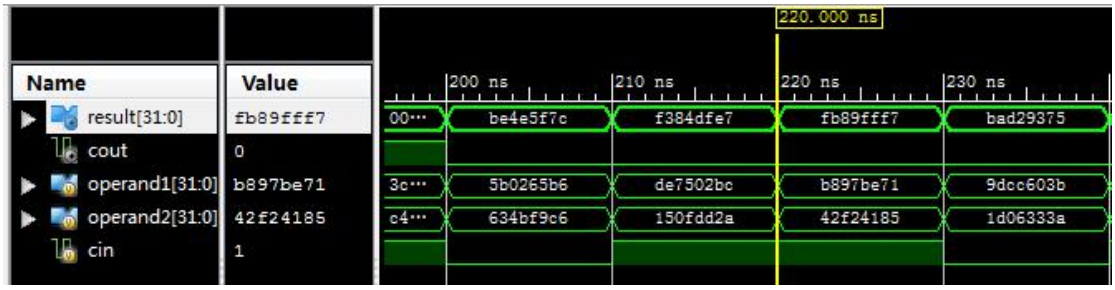


图 2.16 数据以 16 进制形式显示

可以检查几组数据，比如：b897be71+42f24186+1=fb89fff7，正确。类似的，通过观察波形可以看到加法功能模块随机测试并没有出错，我们认为功能趋于稳定，可以认为是正确的。

至此，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。

5) 上板验证

所谓上板验证是指将功能代码进行综合和布局布线后下载到 FPGA 板上运行，在板上检查运行的正确性。

因此，需要设定一套在板上检查结果的机制。比如，对于本例中的定点加法，可以设定使用拨码开关作为加数的输入，使用 led 灯作为加法结果的输出，这样就能在 FPGA 实验板上观察加法模块的运行结果了。但遗憾的是，对于 32 位加法，需要 64 个拨码开关和 32 个 led 灯，但显然实验板上提供不了如此之多的 IO 接口，当然也可以通过精巧的设计使用实验板上有限的资源来完成加数输入和结果显示的，但使用起来不够方便。

对于 LS-CPU-EXB-002 试验箱并没有上述的问题，因为可以通过 LCD 触摸屏输入 32 位加数，并能显示加法结果。之前提到的外围展示模块 adder_display.v 就是调用 LCD 触摸屏来完成上板验证的机制的设计的，具体验证机制的解析见本章的 5.2 小节。

在有了板上验证机制后，需要添加引脚绑定的约束文件。所谓约束文件就是将顶层模块(本例中为 adder_display)的输入输出端口与 FPGA 板上的 IO 接口引脚绑定，以完成在板上的输入输出。

约束文件后缀名为.xdc，有两种方法可以添加约束文件，其一是用“Add or create constraints”添加或创建约束文件，如下图：

图 2.17 添加约束文件

之后选择添加 adder.xdc。

adder.xdc 内容如下：

```
set_property PACKAGE_PIN AC19 [get_ports clk]
set_property PACKAGE_PIN A3 [get_ports led_cout]
set_property PACKAGE_PIN Y3 [get_ports resetn]
set_property PACKAGE_PIN AC21 [get_ports input_sel]
set_property PACKAGE_PIN Y6 [get_ports sw_cin]

set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports led_cout]
set_property IOSTANDARD LVCMOS33 [get_ports resetn]
set_property IOSTANDARD LVCMOS33 [get_ports input_sel]
set_property IOSTANDARD LVCMOS33 [get_ports sw_cin]

#lcd
set_property PACKAGE_PIN J25 [get_ports lcd_rst]
set_property PACKAGE_PIN H18 [get_ports lcd_cs]
set_property PACKAGE_PIN K16 [get_ports lcd_rs]
set_property PACKAGE_PIN L8 [get_ports lcd_wr]
set_property PACKAGE_PIN K8 [get_ports lcd_rd]
set_property PACKAGE_PIN J15 [get_ports lcd_bl_ctr]
set_property PACKAGE_PIN H9 [get_ports {lcd_data_io[0]}]
set_property PACKAGE_PIN K17 [get_ports {lcd_data_io[1]}]
set_property PACKAGE_PIN J20 [get_ports {lcd_data_io[2]}]
set_property PACKAGE_PIN M17 [get_ports {lcd_data_io[3]}]
set_property PACKAGE_PIN L17 [get_ports {lcd_data_io[4]}]
set_property PACKAGE_PIN L18 [get_ports {lcd_data_io[5]}]
set_property PACKAGE_PIN L15 [get_ports {lcd_data_io[6]}]
set_property PACKAGE_PIN M15 [get_ports {lcd_data_io[7]}]
set_property PACKAGE_PIN M16 [get_ports {lcd_data_io[8]}]
set_property PACKAGE_PIN L14 [get_ports {lcd_data_io[9]}]
set_property PACKAGE_PIN M14 [get_ports {lcd_data_io[10]}]
set_property PACKAGE_PIN F22 [get_ports {lcd_data_io[11]}]
set_property PACKAGE_PIN G22 [get_ports {lcd_data_io[12]}]
set_property PACKAGE_PIN G21 [get_ports {lcd_data_io[13]}]
set_property PACKAGE_PIN H24 [get_ports {lcd_data_io[14]}]
set_property PACKAGE_PIN J16 [get_ports {lcd_data_io[15]}]
set_property PACKAGE_PIN L19 [get_ports ct_int]
```

```
37    set_property PACKAGE_PIN J24 [get_ports ct_sda]
38    set_property PACKAGE_PIN H21 [get_ports ct_scl]
39    set_property PACKAGE_PIN G24 [get_ports ct_rstn]
40
41    set_property IOSTANDARD LVCMOS33 [get_ports lcd_rst]
42    set_property IOSTANDARD LVCMOS33 [get_ports lcd_cs]
43    set_property IOSTANDARD LVCMOS33 [get_ports lcd_rs]
44    set_property IOSTANDARD LVCMOS33 [get_ports lcd_wr]
45    set_property IOSTANDARD LVCMOS33 [get_ports lcd_rd]
46    set_property IOSTANDARD LVCMOS33 [get_ports lcd_bl_ctr]
47    set_property IOSTANDARD LVCMOS33 [get_ports {lcd_data_io[0]}]
48    set_property IOSTANDARD LVCMOS33 [get_ports {lcd_data_io[1]}]
49    set_property IOSTANDARD LVCMOS33 [get_ports {lcd_data_io[2]}]
50    set_property IOSTANDARD LVCMOS33 [get_ports {lcd_data_io[3]}]
51    set_property IOSTANDARD LVCMOS33 [get_ports {lcd_data_io[4]}]
52    set_property IOSTANDARD LVCMOS33 [get_ports {lcd_data_io[5]}]
53    set_property IOSTANDARD LVCMOS33 [get_ports {lcd_data_io[6]}]
54    set_property IOSTANDARD LVCMOS33 [get_ports {lcd_data_io[7]}]
55    set_property IOSTANDARD LVCMOS33 [get_ports {lcd_data_io[8]}]
56    set_property IOSTANDARD LVCMOS33 [get_ports {lcd_data_io[9]}]
57    set_property IOSTANDARD LVCMOS33 [get_ports {lcd_data_io[10]}]
58    set_property IOSTANDARD LVCMOS33 [get_ports {lcd_data_io[11]}]
59    set_property IOSTANDARD LVCMOS33 [get_ports {lcd_data_io[12]}]
60    set_property IOSTANDARD LVCMOS33 [get_ports {lcd_data_io[13]}]
61    set_property IOSTANDARD LVCMOS33 [get_ports {lcd_data_io[14]}]
62    set_property IOSTANDARD LVCMOS33 [get_ports {lcd_data_io[15]}]
63    set_property IOSTANDARD LVCMOS33 [get_ports ct_int]
64    set_property IOSTANDARD LVCMOS33 [get_ports ct_sda]
65    set_property IOSTANDARD LVCMOS33 [get_ports ct_scl]
66    set_property IOSTANDARD LVCMOS33 [get_ports ct_rstn]
```

可以看到主要有时钟与复位信号的引脚连接，led 灯和拨码开关的引脚连接，以及 LCD 触摸屏引脚的连接。此处时钟与复位信号和 LCD 触摸屏引脚都是 lcd_module 需要用到的，可以不用管。

对于 led 灯的连接（led_cout 绑定到 A3 引脚上），可以看到是用来显示加法向高位的进位的。

对于拨码开关的连接，“sw_cin”是用来输入低位的进位的，而“input_sel”是用来选择通过触摸屏输入的 32 位数据为加数 1 还是加数 2。

由于以后的实验都需要用到 LCD 触摸屏，故 LCD 触摸屏相关引脚的绑定是固定不变的，故建议以后实验都通过添加已有 xdc 文件，然后再根据需求修改 led 和拨码开关等引脚的绑定。

也可以通过 Vivado 中的 I/O Planning 功能来产生约束文件，具体过程如下：

点击 Flow Navigator 中 Synthesis 下 Run Synthesis，进行综合：

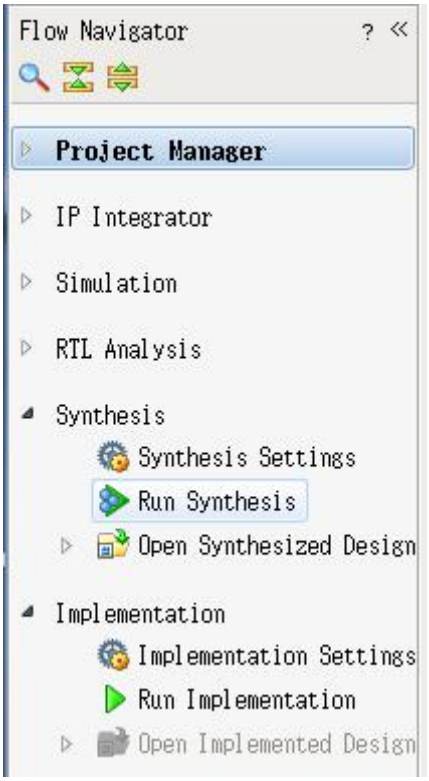


图 2.18 运行综合

综合完成后，选择”Open Syunthesized Design”，查看综合结果：

图 2.19 综合完成

得到如下界面：

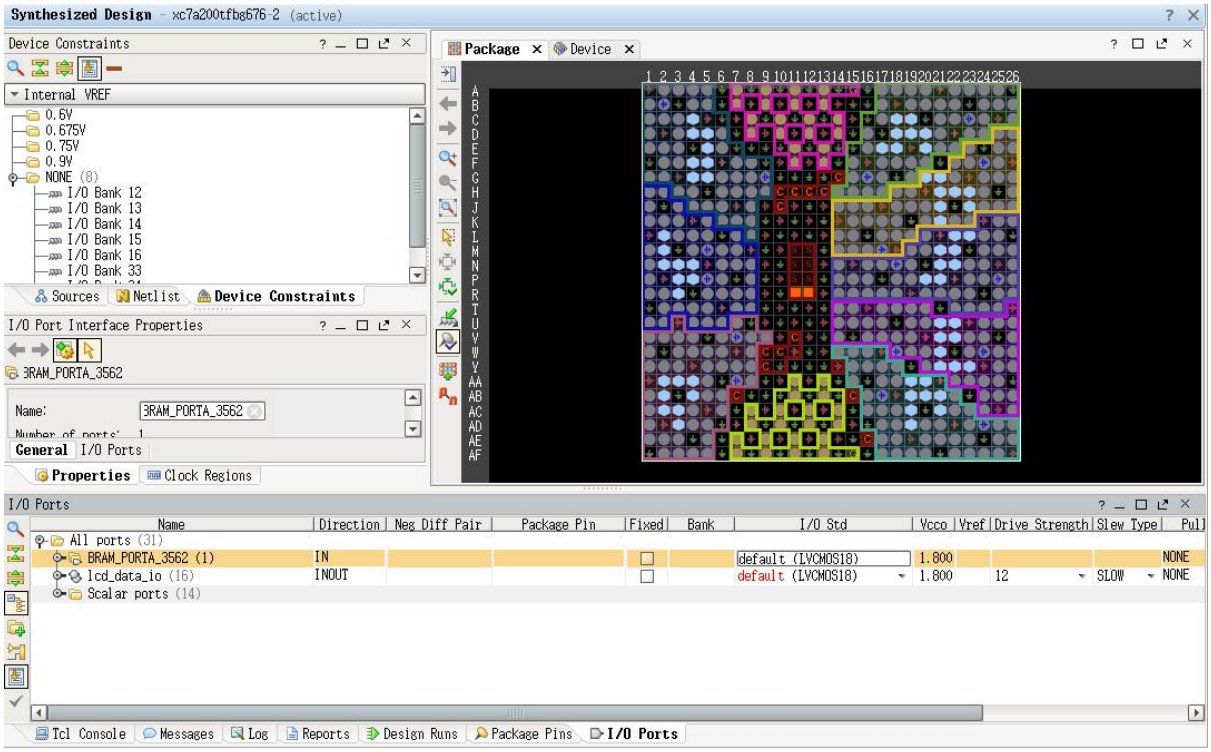


图 2.20 综合结果显示

之后参照实验板原理图和引脚对应关系表，在 I/O Ports 栏下填入正确的 Package Pin，针对本实验板，I/O Std 要统一设置为 LVCMOS33。

所有约束条件填写完成之后效果如下：

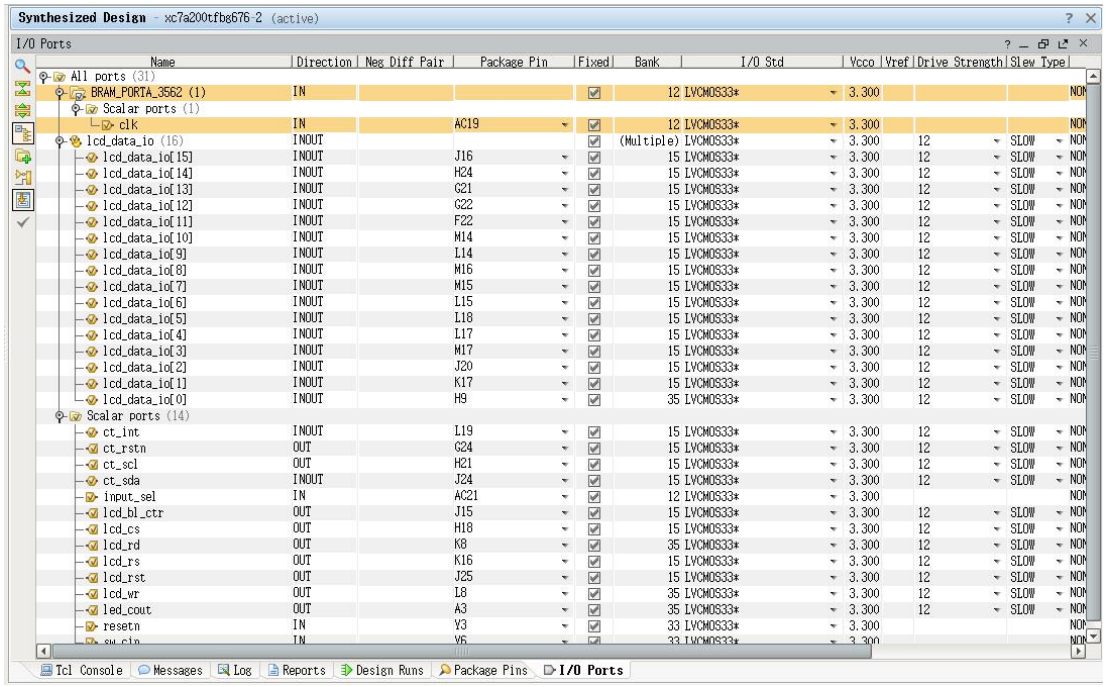


图 2.21 填写完成的 I/O Planning

点击右上角的”保存”键即可保存为 xdc 文件。

后续的流程就是综合、布局布线和产生可烧写文件，可以依次双击运行，也可以只双击“Generate Bitsteam”会自动运行这三步，运行结果如下，可以选择 Open Implemented Design 来查看实现结果：

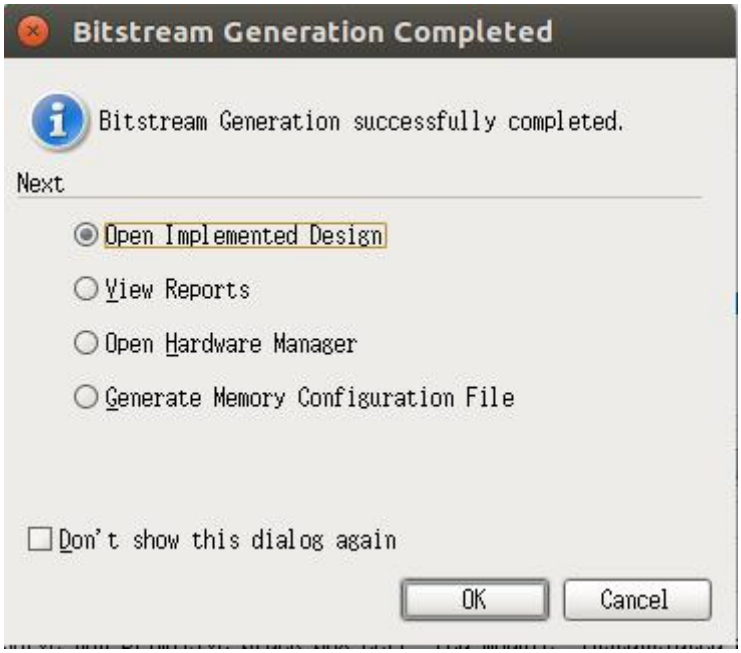


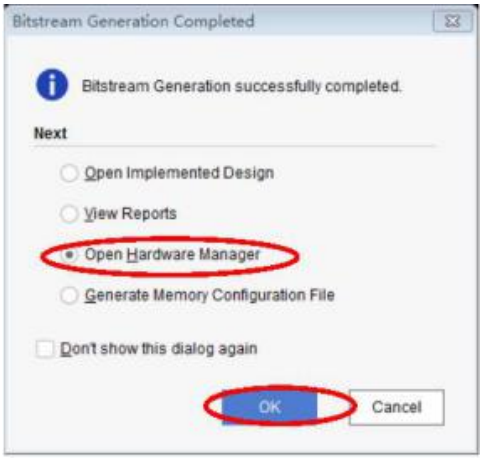
图 2.22 综合、布局布线和产生可烧写文件运行结果

这时可烧写的文件已经产生成功了，后缀名为.bit。

在打开 FPGA 实验板，上电，并将下载线与电脑相连后，打开电源，FPGA 板如下图：

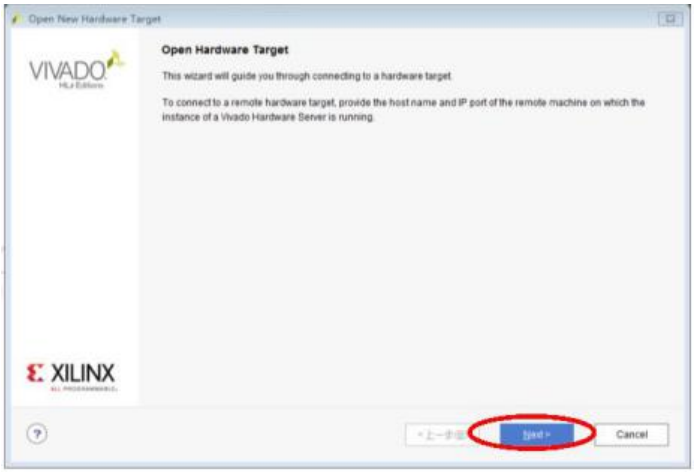
图 2.23 上电后初始的 FPGA 板

在比特流文件生成完成的窗口选择“Open Hardware Manager”，进入硬件管理界面。连接 FPGA 开发板的电源线和与电脑的下载线，打开 FPGA 电源。如下图：

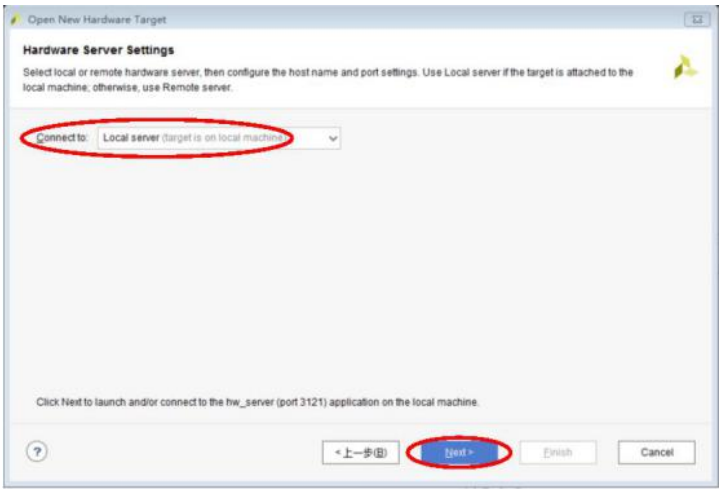


在“Hardware Manager”窗口的提示信息中，点击“ Open target”的下拉菜单的“ Open New Target”（或在“Flow Navigator”下“Program and Debug”中展开“Open Hardware Manager”，点击“Open Target”->“Open NewTarget”）。也可以选择“Auto Connect”自动连接器件。

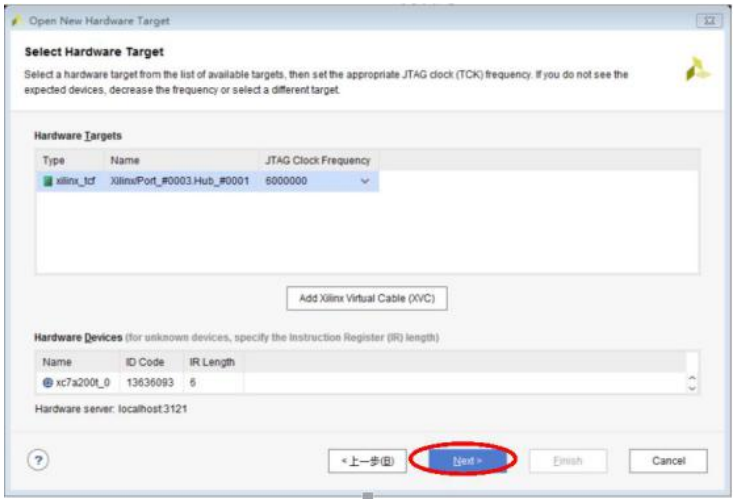
在“Open Hardware Target”向导中，先点击“Next”，进入 Server 选择向导。如下图：



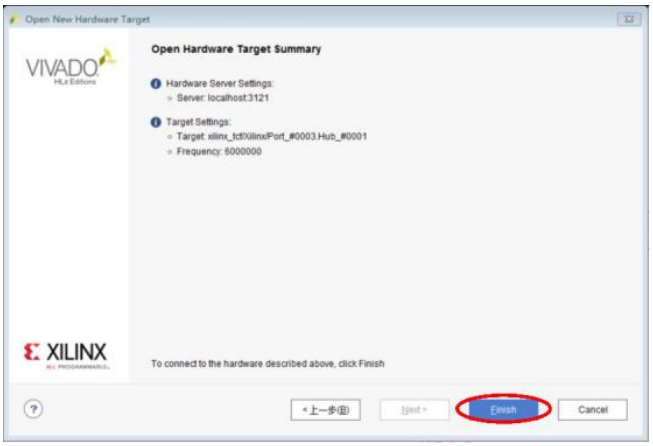
选择连接到“Local server”，点击“Next”。出现以下图：



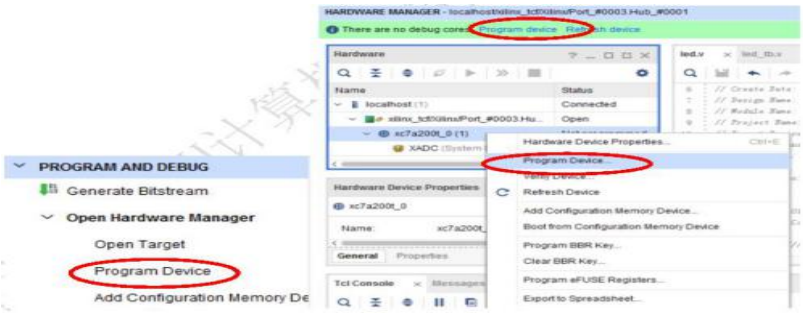
选择目标硬件，点击“Next”。



完成目标硬件打开。点击“Finish”。

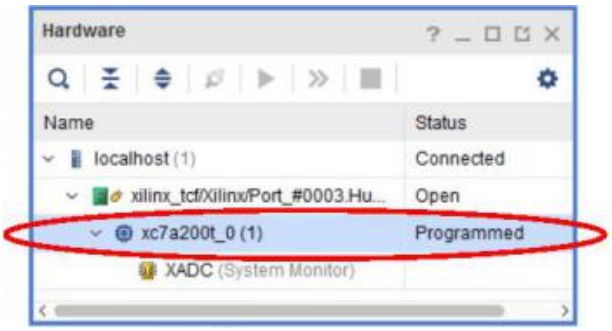


对目标硬件编程。在“ Hardware”窗口右键单击目标器件“ xc7a200t_0”，选择“Program Device…”。或者“Flow Navigator”窗口中“Program and Debug”->“Hardware Manager”->“Program Device”。



选择下载的 bit 流文件，点击“Program”。

完成下载后，“Hardware”窗口下的“xc7a200t_0”的状态变成“Programmed”。



烧写 bit 文件，烧写完成后的 FPGA 板显示如下：

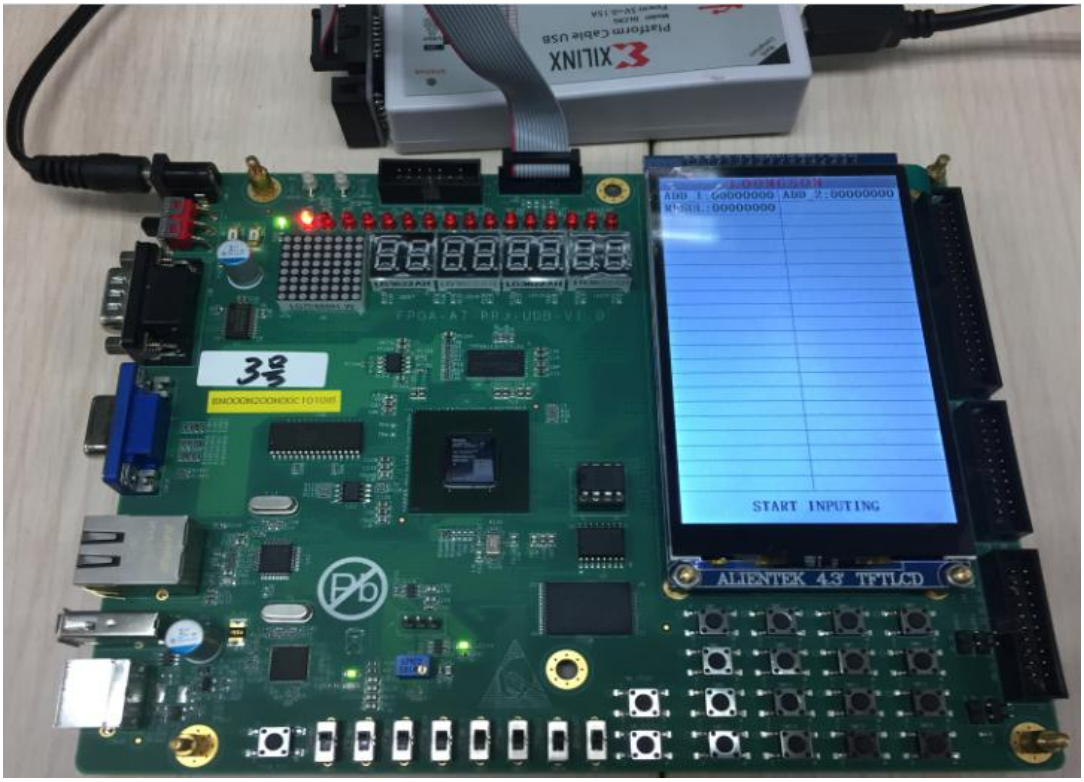


图 2.24 加法模块下载成功后的 FPGA 板

从图 2.24 中可以看到 LCD 触摸屏上分别显示了 2 个加数和加法结果，最右侧的 led 灯为向高位的进位，该 led 灯为共阴极的，即输入为 0 是亮，为 1 是不亮，由于当前进位为 0，故 led 灯为亮。拨码开关最右侧的开关用来选择触摸屏输入的数据为加数 1 还是加数 2，当前该开关置为 1，说明触摸屏输入的 32 位数位加数 2。

在需要使用触摸屏的输入功能时，手指点击屏上最后一列的“START INPUTING”即可进入输入模式，如下图：

图 2.25 LCD 屏进入输入模式

从图 2.25 中可看到，触摸屏的输入模式为小键盘，包含 0~F，回退键(BACK)和确认键(OK)。输入的数据为 32 位的 16 进制数，当输出错误的时候，可以按 BACK 键回退一格，当输入完成时按 OK 键完成输入，同时会退出输入模式。当按 OK 时，输入未满 32 位，则会高位补 0，比如：只输入了 123，则最终屏调用模块输入时，输入数据为 0x00000123。

在本例中，给加数 2 输入 0x11111111 后，可看到 FPGA 板上结果如下图：

图 2.26 加数 2 输入 0x11111111

可看到加法结果页变为 0x11111111，正确。可以将拨码开关最右侧的置为 0，此时表示输入数据为加数 1。因此，通过触摸屏可以给加数 1 和加数 2 输入任意 32 位数，并能实时看到加法结果，完成上板验证。

5.2 LCD 触摸屏调用方法

正如前面所述，adder_display.v 作为顶层模块，里面实例化了 adder.v，同时也调用了 lcd_module 用来上板演示。此小节着重讲解 adder_display.v 里调用 LCD 触摸屏的方法。

1) LCD 触摸屏调用接口

先来看 lcd_module.v 的代码：

```
1  //*************************************************************
2  //    > 文件名: lcd_module.v
3  //    > 描述   : lcd 触摸屏模块，为黑盒文件
4  //    > 作者   : LOONGSON
5  //    > 日期   : 2016-04-14
6  //*************************************************************
7  //synthesis attribute box_type <lcd_module> "black_box"
8  module lcd_module(
9      input  clk,         //10Mhz
10     input  resetn,    //低使能
11
12     //调用触摸屏的接口
13     input          display_valid,
14     input  [39:0] display_name,
15     input  [31:0] display_value,
16     output [ 5:0] display_number,
17     output         input_valid,
18     output [31:0] input_value,
19
20     //lcd 触摸屏相关接口，不需要更改
21     output reg    lcd_rst,
22     output        lcd_cs,
23     output        lcd_rs,
24     output        lcd_wr,
25     output        lcd_rd,
26     inout [15:0] lcd_data_io,
27     output        lcd_bl_ctr,
28     inout         ct_int,
29     inout         ct_sda,
30     output        ct_scl,
31     output        ct_rstn
32     );
33 endmodule
```

可以看到该模块为黑盒文件，其中时钟和复位信号以及 LCD 触摸屏引脚接口不需要关注，下面介绍调用触摸屏的 6 个接口。

2) 调用 LCD 触摸屏显示

```
input          display_valid,
input  [39:0] display_name,
input  [31:0] display_value,
output [ 5:0] display_number,
```

display 的 4 个接口用于在屏上显示数据。从图 2.24 FPGA 板的实物图中可以看到，LCD 屏用于显示的区域块共有 2 列，22 行，故共可显示 44 组数据。

display_number 就是输出到外部说明当前需要显示的区域块为第几块，有效编号从 1~44，指示 44 块显示区域块。

每块显示区域块可显示 14 个字符，其中头五个字符为块名，指示当前块显示的数据的意义，由 display_name 输入指定。display_name 输入的为要显示字符的 ASCII 码，5 个 ASCII 码共 40 位。块名可显示字符为大写的 26 个字母，下划线”_”，0~9 数字以及空格。

每块显示区域块的第 6 个字符为冒号，用于区分块名和块数据段。后 8 个字符显示该块的数值，显示的为 32 位 16 进制数，故占用 8 个字符。该段由 display_value 输入指定，display_value 输入的为 32 位 2 进制数，lcd_module 内部会自动转换为 8 个字符显示。

最后还有 display_valid 输入，用于指示是否需要在当前显示区域块(由 display_number)显示数据，为 1 有效。

在本例中，顶层模块 adder_display.v 调用 LCD 触摸屏显示功能的代码如下：

```
117  //-----{输出到触摸屏显示}begin
118  //根据需要显示的数修改此小节，
119  //触摸屏上共有 44 块显示区域，可显示 44 组 32 位数据
120  //44 块显示区域从 1 开始编号，编号为 1~44，
121      always @(posedge clk)
122      begin
123          case(display_number)
124              6'd1 :
125              begin
126                  display_valid <= 1'b1;
127                  display_name  <= "ADD_1";
128                  display_value <= adder_operand1;
129              end
130              6'd2 :
131              begin
132                  display_valid <= 1'b1;
133                  display_name  <= "ADD_2";
134                  display_value <= adder_operand2;
135              end
136              6'd3 :
137              begin
138                  display_valid <= 1'b1;
139                  display_name  <= "RESUL";
140                  display_value <= adder_result;
141              end
142              default :
143              begin
144                  display_valid <= 1'b0;
145                  display_name  <= 40'd0;
146                  display_value <= 32'd0;
147              end
```

```
148          endcase
149      end
150  //-----{输出到触摸屏显示}end
151  //----------------------{调用触摸屏模块}end--------------------//
```

可以看到加法模块只需要区域块 1~3 用于显示，其他块(default 分支)的 display_valid 为 0。另外，display_name 接口赋值 5 位字符组成的字符串即可，字符串里应当只出现大写的 26 个字母，下划线"_"，0~9 数字或空格，其他字符在 LCD 触摸屏上暂不支持显示。而 display_value 直接赋值 32 位的数据即可。

3) 调用 LCD 触摸屏输入

```
output         input_valid,
output [31:0] input_value,
```

input 的两个接口用于使用触摸屏的输入功能。从图 2.24 可以看到，当需要使用输入功能时，触摸屏底部的"START INPUTING"栏即可进入输入模式，见图 2.25。点击屏小键盘上的 OK 键完成输入，会退出输入模式，同时 lcd_module 会拉高 input_valid 信号一拍，表示有数据要输出，而输出数据 input_value 会依据之前的输入确定，当输入不足 32 位时，会自动高位补 0，比如：只输入了 123，就按 OK 键，则最终 input_valid 的值为 0x00000123。当输入有误时，可以按 BACK 键回退一格。

在本例中，顶层模块 adder_display.v 调用 LCD 触摸屏输入功能的代码如下：

```
87  //-----{从触摸屏获取输入}begin
88  //根据实际需要输入的数修改此小节，
89  //建议对每一个数的输入，编写单独一个 always 块
90      //当 input_sel 为 0 时，表示输入数为加数 1，即 operand1
91      always @(posedge clk)
92      begin
93          if (!resetn)
94          begin
95              adder_operand1 <= 32'd0;
96          end
97          else if (input_valid && !input_sel)
98          begin
99              adder_operand1 <= input_value;
100         end
101     end
102
103     //当 input_sel 为 1 时，表示输入数为加数 2，即 operand2
104     always @(posedge clk)
105     begin
106         if (!resetn)
107         begin
108             adder_operand2 <= 32'd0;
109         end
110         else if (input_valid && input_sel)
```

```
111          begin
112              adder_operand2 <= input_value;
113          end
114      end
115  //-----{从触摸屏获取输入}end
```

可以看到加法模块需要使用触摸屏输入 2 个加数，故需要一个拨码开关指示输入的数据为加数 1 还是加数 2。故理论上，触摸屏加上外部的选择信号，可以给任意信号输入 32 位的数，比如可以输入一条 32 位的指令，可以输入内存的 32 位地址等等，这些在后续的 CPU 实验中可能会经常用到。当需要使用触摸屏输入多个数据时，比如本例中输入 2 个加数，建议每个数据单独用一个 always 块采集输入，这样的写法更接近电路实际情况，示例中的代码就是如此写的。

4) adder_display.v 完整代码

```
1  //************************************************************
   *
2  //    > 文件名: adder_display.v
3  //    > 描述   : 加法器显示模块，调用 FPGA 板上的 IO 接口和触摸屏
4  //    > 作者   : LOONGSON
5  //    > 日期   : 2016-04-14
6  //************************************************************
   *
7  module adder_display(
8      //时钟与复位信号
9      input clk,
10     input resetn,     //后缀"n"代表低电平有效
11
12     //拨码开关，用于选择输入数和产生 cin
13     input input_sel,//0:输入为加数 1(add_operand1);1:为加数 2(add_operand2)
14     input sw_cin,
15
16     //led 灯，用于显示 cout
17     output led_cout,
18
19     //触摸屏相关接口，不需要更改
20     output lcd_rst,
21     output lcd_cs,
22     output lcd_rs,
23     output lcd_wr,
24     output lcd_rd,
25     inout[15:0] lcd_data_io,
26     output lcd_bl_ctr,
27     inout  ct_int,
28     Inout  ct_sda,
29     output ct_scl,
```

```
30      output ct_rstn
31      );
32
33  //-----{调用加法模块}begin
34      reg  [31:0] adder_operand1;
35      reg  [31:0] adder_operand2;
36      wire        adder_cin;
37      wire [31:0] adder_result;
38      wire        adder_cout;
39      adder adder_module(
40          .operand1(adder_operand1),
41          .operand2(adder_operand2),
42          .cin     (adder_cin     ),
43          .result  (adder_result  ),
44          .cout    (adder_cout    )
45      );
46      assign adder_cin = sw_cin;
47      assign led_cout  = adder_cout;
48  //-----{调用加法模块}end
49
50  //----------------------{调用触摸屏模块}begin--------------------//
51  //-----{实例化触摸屏}begin
52  //此小节不需要更改
53      reg         display_valid;
54      reg  [39:0] display_name;
55      reg  [31:0] display_value;
56      wire [5 :0] display_number;
57      wire        input_valid;
58      wire [31:0] input_value;
59
60      lcd_module lcd_module(
61          .clk            (clk           ),   //10Mhz
62          .resetn         (resetn        ),
63
64          //调用触摸屏的接口
65          .display_valid  (display_valid ),
66          .display_name   (display_name  ),
67          .display_value  (display_value ),
68          .display_number (display_number),
69          .input_valid    (input_valid   ),
70          .input_value    (input_value   ),
71
72          //lcd 触摸屏相关接口，不需要更改
73          .lcd_rst        (lcd_rst       ),
74          .lcd_cs         (lcd_cs        ),
```

```
75          .lcd_rs          (lcd_rs        ),
76          .lcd_wr          (lcd_wr        ),
77          .lcd_rd          (lcd_rd        ),
78          .lcd_data_io     (lcd_data_io   ),
79          .lcd_bl_ctr      (lcd_bl_ctr    ),
80          .ct_int          (ct_int        ),
81          .ct_sda          (ct_sda        ),
82          .ct_scl          (ct_scl        ),
83          .ct_rstn         (ct_rstn       )
84      );
85  //-----{实例化触摸屏}end
86
87  //-----{从触摸屏获取输入}begin
88  //根据实际需要输入的数修改此小节，
89  //建议对每一个数的输入，编写单独一个 always 块
90      //当 input_sel 为 0 时，表示输入数为加数 1，即 operand1
91      always @(posedge clk)
92      begin
93          if (!resetn)
94          begin
95              adder_operand1 <= 32'd0;
96          end
97          else if (input_valid && !input_sel)
98          begin
99              adder_operand1 <= input_value;
100         end
101     end
102
103     //当 input_sel 为 1 时，表示输入数为加数 2，即 operand2
104     always @(posedge clk)
105     begin
106         if (!resetn)
107         begin
108             adder_operand2 <= 32'd0;
109         end
110         else if (input_valid && input_sel)
111         begin
112             adder_operand2 <= input_value;
113         end
114     end
115 //-----{从触摸屏获取输入}end
116
117 //-----{输出到触摸屏显示}begin
118 //根据需要显示的数修改此小节，
119 //触摸屏上共有 44 块显示区域，可显示 44 组 32 位数据
```

```
120  //44 块显示区域从 1 开始编号，编号为 1~44，
121      always @(posedge clk)
122      begin
123          case(display_number)
124              6'd1 :
125              begin
126                  display_valid <= 1'b1;
127                  display_name  <= "ADD_1";
128                  display_value <= adder_operand1;
129              end
130              6'd2 :
131              begin
132                  display_valid <= 1'b1;
133                  display_name  <= "ADD_2";
134                  display_value <= adder_operand2;
135              end
136              6'd3 :
137              begin
138                  display_valid <= 1'b1;
139                  display_name  <= "RESUL";
140                  display_value <= adder_result;
141              end
142              default :
143              begin
144                  display_valid <= 1'b0;
145                  display_name  <= 40'd0;
146                  display_value <= 32'd0;
147              end
148          endcase
149      end
150  //-----{输出到触摸屏显示}end
151  //----------------------{调用触摸屏模块}end---------------------//
152  endmodule
```

6 定点加法实验拓展

根据之前所述，可以画出定点加法实验的顶层模块框图，如下图：

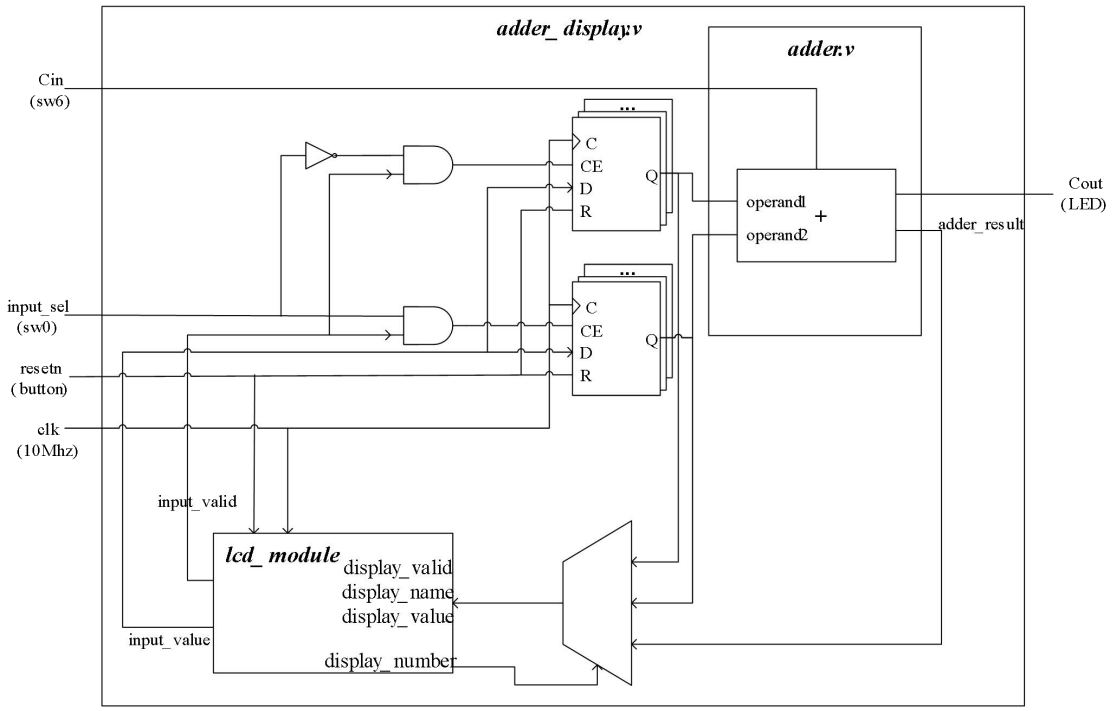


图 2.30 定点加法参考设计的顶层模块框图

从图中可以看到，外围模块 adder_display 内部调用了 adder 和 lcd_module 模块。在外部除了时钟和复位信号外，还有 cin 和 input_sel 通过拨码开关输入，以及 cout 输出到 led 灯上。

从实验二开始，针对每个实验需要画出类似图 2.40 的顶层模块。并且对内部功能模块(此处为 adder.v)需要画出原理图或实现框图，此处由于 adder.v 就用“+”实现，比较简单，故未给出加法模块的原理图。

本例中提供的实验参考设计主用用于熟悉软硬件平台的，加法也是直接使用加号实现的。

学生可以在走通软硬件流程后，建议尝试修改加法的实现，可以使用 1 位全加器搭建出 32 位的加法，搭建的方法就有很多了，可以根据已学定点加法实现方法选择性地搭建串行进位加法器或先行进位加法器等，此时需要画出对应的原理图。

三、 实验二 数据运算：定点乘法

1 实验目的

1. 理解定点乘法的不同实现算法的原理，掌握基本实现算法。
2. 熟悉并运用 verilog 语言进行电路设计。
3. 为后续设计 cpu 的实验打下基础。

2 实验设备

1. 装有 Xilinx Vivado 的计算机一台。
2. LS-CPU-EXB-002 教学系统实验箱一套。

3 实验任务

1. 学习并理解计算机中定点乘法器的多种实现算法的原理，重点掌握迭代乘法的实现算法。

2. 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，本次实验的乘法器建议采用迭代的方式实现，如果能力有余的，也可以采用其他效率更高的算法实现。本次实验要求实现的乘法为有符号乘法，因此需要注意计算机存储的有符号数都是补码的形式，设计方案传递进来的数也需是补码。

3. 根据设计的实验方案，使用 verilog 编写相应代码。

4. 对编写的代码进行仿真，得到正确的波形图。

5. 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，见图 3.1。外围模块中需调用封装好的 LCD 触摸屏模块，显示两个乘数和乘法结果，且需要利用触摸功能输入两个乘数。

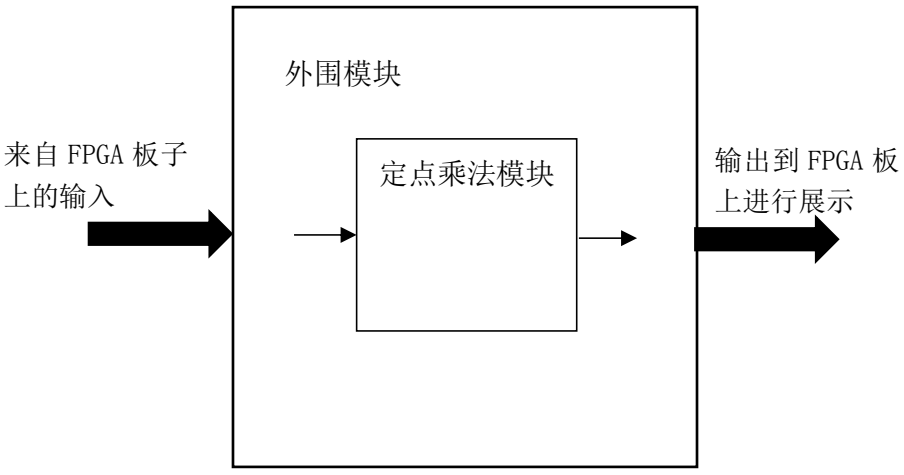


图 3.1 定点乘法设计实验的顶层模块大致框图

6. 将编写的代码进行综合布局布线，并下载到实验箱中的 FPGA 板子上进行演示。

4 实验要求

1. 做好预习：

   1） 掌握定点乘法的多种实现算法的原理；

   2） 确定定点乘法的输入输出端口设计；

   3） 在课前画好设计框图或实验原理图；

   4） 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 3.1。

2. 实验实施：

   1） 确认定点乘法的设计框图的正确性；

   2） 编写 verilog 代码；

   3） 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；

   4） 完成调用定点乘法模块的外围模块的设计，并编写代码；

   5） 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。

3. 实验检查：

   1） 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。

4. 实验报告的撰写：

   1） 实验结束后，需按照规定的格式完成实验报告的撰写。

四、 实验三 寄存器堆实现

1 实验目的

1. 熟悉并掌握 MIPS 计算机中寄存器堆的原理和设计方法。
2. 初步了解 MIPS 指令结构和源操作数/目的操作数的概念。
3. 熟悉并运用 verilog 语言进行电路设计。
4. 为后续设计 cpu 的实验打下基础。

2 实验设备

1. 装有 Xilinx Vivado 的计算机一台。
2. LS-CPU-EXB-002 教学系统实验箱一套。

3 实验任务

1. 学习 MIPS 计算机中寄存器堆的设计及原理，如：有多少个寄存器，有无特殊设置的寄存器，mips 指令如何去索引寄存器的等。

2. 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，本次实验建议设计为异步读同步写的寄存器堆，即读寄存器不需要时钟控制，但写寄存器需时钟控制。

3. 本次实验建议寄存器堆设计为 1 个写端口和 2 个读端口，后续 CPU 实验用到的寄存器堆需要 1 个写端口和 2 个读端口。

4. 根据设计的实验方案，使用 verilog 编写相应代码。

5. 对编写的代码进行仿真，得到正确的波形图。

6. 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，见图 4.1。外围模块中需调用封装好的 LCD 触摸屏模块，显示寄存器堆的读写端口地址和数据，最好能扫描出所有寄存器的值显示在 LCD 触摸屏上，并且需要利用触摸功能输入寄存器堆的读写地址和写数据。

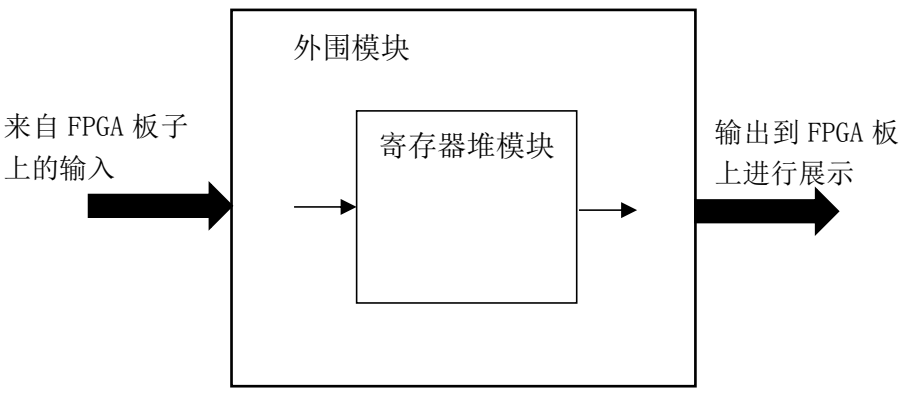


图 4.2 寄存器堆设计实验的顶层模块大致框图

7. 将编写的代码进行综合布局布线，并下载到实验箱中的 FPGA 板子上进行演示。

4 实验要求

1. 做好预习：
   1） 掌握寄存器堆的工作原理；
   2） 确定寄存器堆的输入输出端口设计；
   3） 在课前画好寄存器堆的设计框图或实验原理图；
   4） 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 4.1。
2. 实验实施：
   1） 确认寄存器堆的设计框图的正确性；
   2） 编写 verilog 代码；
   3） 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；
   4） 完成调用寄存器堆模块的外围模块的设计，并编写代码；
   5） 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。
3. 实验检查：
   1） 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，按照检查人员的要求，对特定寄存器读写，可对演示结果进行拍照作为实验报告结果一项的材料。
4. 实验报告的撰写：
   1） 实验结束后，需按照规定的格式完成实验报告的撰写。

5 调用 LCD 触摸屏

在寄存器堆实验中，调用 LCD 触摸屏的显示和输入功能时，有几点需要注意的地方。

5.1 显示功能

在本实验中需要扫描出寄存器堆里的所有寄存器的值显示在 LCD 触摸屏上，比如我们在 LCD 屏上 1~32 号显示块上对应显示 0~31 号寄存器的显示地址。由于 LCD 屏在显示时，是根据显示块号“display_number”来选择要显示的数据的，故可以给寄存器堆再增加一个调试读端口(此时寄存器堆有 3 个读端口)，该读端口的地址为 display_number-1，对应 0~31,读出的数据即为要显示的模块，代码可以如下编写：

```
1  assign test_addr = display_number-5'd1;
2  //-----{输出到触摸屏显示}begin
3  //根据需要显示的数修改此小节，
4  //触摸屏上共有 44 块显示区域，可显示 44 组 32 位数据
```

```
5  //44 块显示区域从 1 开始编号，编号为 1~44，
6      always @(posedge clk)
7      begin
8          if (display_number >6'd0 && display_number <6'd33 )
9          begin //块号 1~32 显示 32 个通用寄存器的值
10             display_valid <= 1'b1;
11             display_name[39:16] <= "REG";
12             display_name[15: 8] <= {4'b0011,3'b000,test_addr[4]};
13             display_name[7 : 0] <= {4'b0011,test_addr[3:0]};
14             display_value       <= test_data;
15         end
16         else
17         begin
18             case(display_number)
19                 6'd33: //显示读端口 1 的地址
20                 begin
21                     display_valid <= 1'b1;
22                     display_name  <= "RADD1";
23                     display_value <= raddr1;
24                 end
25       ……
26       ……
```

test_addr 为寄存器堆新增的调试读端口，而 test_data 为通过该端口读出的数据

可以看到在显示 32 个寄存器时，其“display_name”域有特殊处理，其头 3 个字符显示“REG”，后两个字符则显示 16 进制的寄存器号“00~1F”，由于 0~9 字符的 ASCII 码为 48~57(8’b0011_0000~8’b0011_1001)，故“display_name”第 4 位字符(0 或 1)赋值{7’b0011_000,test_addr[4]}即可。

但对于第 5 位字符(0~F)的赋值则有稍许麻烦，因为 A~F 字符的 ASCII 编码不是随后的 58~63，即与 0~9 的编码不连续。但我们在设计 LCD 屏显示字符时，将内部的 ASCII 编码 58~63 也作为 A~F 字符的编码，即 ASCII 编码 48~63(8’b0011_0000～8’b0011_1111)对应字符 0~F，，故我们可以直接给第 5 位字符赋值{4'b0011,test_addr[3:0]}。

5.2 输入功能

在调用 LCD 触摸屏的输入功能时，其输入的数据为 32 位的数据，但在寄存器堆的实验中，读写地址为 5 位的，但依然可以使用屏输入读写地址，比如，可以取输入的 32 位数据的低 5 位作为读写地址。

五、 实验四 ALU 模块实现

1 实验目的

1. 熟悉 MIPS 指令集中的运算指令，学会对这些指令进行归纳分类。
2. 了解 MIPS 指令结构。
3. 熟悉并掌握 ALU 的原理、功能和设计。
4. 进一步加强运用 verilog 语言进行电路设计的能力。
5. 为后续设计 cpu 的实验打下基础。

2 实验设备

1. 装有 Xilinx Vivado 的计算机一台。
2. LS-CPU-EXB-002 教学系统实验箱一套。

3 实验任务

1. 学习 MIPS 指令集，熟知指令类型，了解指令功能和编码，归纳基础的 ALU 运算指令。

2. 归纳确定自己本次实验中准备实现的 ALU 运算，要求不实现定点乘除指令和浮点运算指令，要求至少实现 5 种 ALU 运算，其中要包含加减运算，其中减法在内部要转换为加法，与加法运算共同调用实验一里自己完成的加法模块去做。

3. 自行设计本次实验的方案，画出结构框图，大致结构框图如图 5.1。图 5.1 中的操作码位数和类型请自行设计，可以设计为独热码（一位有效编码）或二进制编码。比如，设计方案中预定实现 7 种 ALU 运算，则操作码采用独热码，则需 7bit 数据，每位单独指示一种运算；若采用二进制编码，则只用 3bit 数据位即可，但在需 ALU 内部先进行解码，才能确定 ALU 作何种运算。

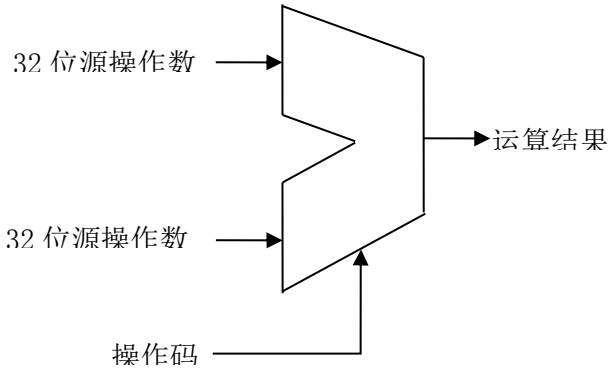


图 5.1 ALU 模块的大致框图

4. 根据设计的实验方案，使用 verilog 编写相应代码。
5. 对编写的代码进行仿真，得到正确的波形图。

8. 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，见图5.2。外围模块中需调用封装好的 LCD 触摸屏模块，显示 ALU 的两个源操作数、操作码和运算结果，并且需要利用触摸功能输入源操作数。操作码可以考虑用 LCD 触摸屏输入，也可以用拨码开关输入。

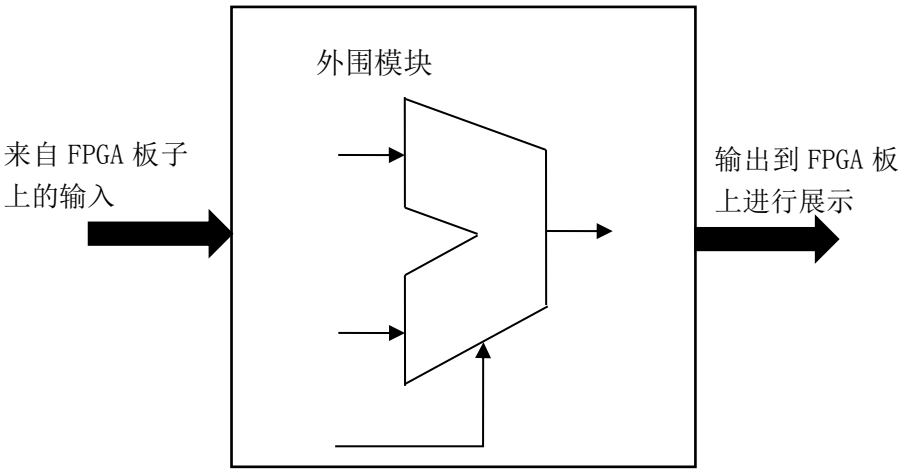


图 5.2 ALU 设计实验的顶层模块大致框图

6. 将编写的代码进行综合布局布线，并下载到试验箱中的 FPGA 板子上进行演示。

4 参考设计

1. 做好预习：
   1） 熟知指令类型，了解指令功能和编码；
   2） 归纳基础的 ALU 运算指令，确定自己准备实现的 ALU 运算；
   3） 设计本次实验的方案，列出准备实现的 ALU 运算和操作码的编码；
   4） 在课前画好实验方案的设计框图，即补充完善图 5.1；
   5） 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 5.2。
2. 实验实施：
   1） 确认 ALU 模块的设计框图的正确性；
   2） 编写 verilog 代码；
   3） 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；
   4） 完成调用 ALU 模块的外围模块的设计，并编写代码；
   5） 对代码进行综合布局布线下载到试验箱里 FPGA 板上，进行上板验证。
3. 实验检查：

1） 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。先说明自己实现的 ALU 运算类型，按照检查人员的要求，对特定源操作数进行特定运算操作，检查运算结果的正确性，可对演示结果进行拍照作为实验报告结果一项的材料。

4. 实验报告的撰写：

1） 实验结束后，需按照规定的格式完成实验报告的撰写。

5 可选 ALU 操作

本部分共给出 12 种 ALU 操作，可以选择全部或部分实现。

表 4.1 ALU 操作

ALU 操作
加法
减法
有符号比较，小于置位
无符号比较，小于置位
按位与
按位或非
按位或
按位异或
逻辑左移
逻辑右移
算术右移
高位加载

六、 实验五 存储器实现

1 实验目的

1. 了解只读存储器 ROM 和随机存取存储器 RAM 的原理。
2. 理解 ROM 读取数据及 RAM 读取、写入数据的过程。
3. 理解计算机中存储器地址编址和数据索引方法。
4. 理解同步 RAM 和异步 RAM 的区别。
5. 掌握调用 Xilinx 库 IP 实例化 RAM 的设计方法。
6. 熟悉并运用 verilog 语言进行电路设计。
7. 为后续设计 cpu 的实验打下基础。

2 实验设备

1. 装有 Xilinx Vivado 的计算机一台。
2. LS-CPU-EXB-002 教学系统实验箱一套。

3 实验任务

1. 学习存储器的设计及原理，如：ROM 读地址索引读取数据过程及时序，RAM 读写时序，同步和异步的区别等。

2. 学习计算机中内存地址编址和数据索引方法。

3. 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，确定存储器宽度、深度和写使能位数。

4. 学习 Vivado 工具中调用库 IP 的方法。

5. 本次实验要求调用 Xilinx 库 IP 实例化一块 RAM。实例化的 RAM 选择为同步 RAM。本次实验的 RAM 建议设置为两个端口，一个端口用来正常的读写，另一个端口作为调试端口只使用读功能用于观察存储器内部数据。

6. 调用 Xilinx 库 IP 实例化一块 RAM，并进行仿真，得到正确的波形图。

7. 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，见图 6.1。外围模块中需调用封装好的 LCD 触摸屏模块，显示 RAM 的正常端口的地址、待写入的数据和读出的数据，显示调试端口的地址和读出的数据。并且需要利用触摸功能输入正常端口的地址和写数据，以及调试端口的地址。

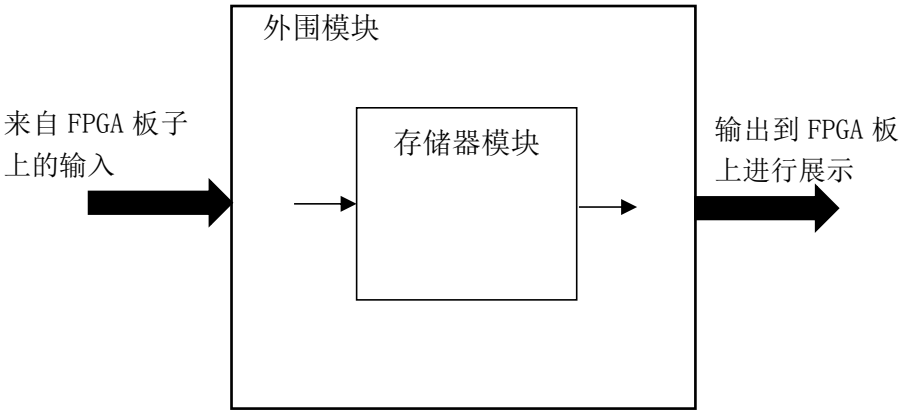


图 6.1 存储器设计实验的顶层模块大致框图

8. 将编写的代码进行综合布局布线，并下载到实验箱中的 FPGA 板子上进行演示。

注意：存储器深度不要过大，避免耗费过多的 FPGA 上的资源。本次实验要求实现同步的存储器。而异步存储器的搭建方法同寄存器堆的搭建，但不同的是，寄存器堆中读写端口是分开的，但对于异步 RAM 要求读写共用一个端口，只是会增加一个写使能信号。可以自行尝试搭建异步的 ROM 和 RAM，在单周期 CPU 实验中会用到异步的 ROM 作为指令存储器，而异步 RAM 作为数据存储器。

4 实验要求

1. 做好预习：
   1） 掌握存储器的工作原理，明白 ROM 和 RAM，同步和异步的区别；
   2） 学习并掌握调用 Xilinx 库 IP 进行设计的方法；
   3） 确定存储器的输入输出端口及宽度、深度和写使能设计；
   4） 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 6.1。
2. 实验实施：
   1） 确认存储器的设计方案的正确性；
   2） 编写 verilog 代码；
   3） 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；
   4） 完成调用存储器模块的外围模块的设计，并编写代码；
   5） 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。
3. 实验检查：
   1） 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，按照检查人员的要求，对特定存储器单元读/写，可对演示结果进行拍照作为实验报告结果一项的材料。

4. 实验报告的撰写：

1） 实验结束后，需按照规定的格式完成实验报告的撰写。

5 调用 Xilinx 库 IP 的方法

本部分以生成同步 RAM 和 ROM 为例说明调用 xilinx 库 IP 的方法。

5.1 生成 IP 核 RAM

1）新建工程

新建一个工程 data_ram:

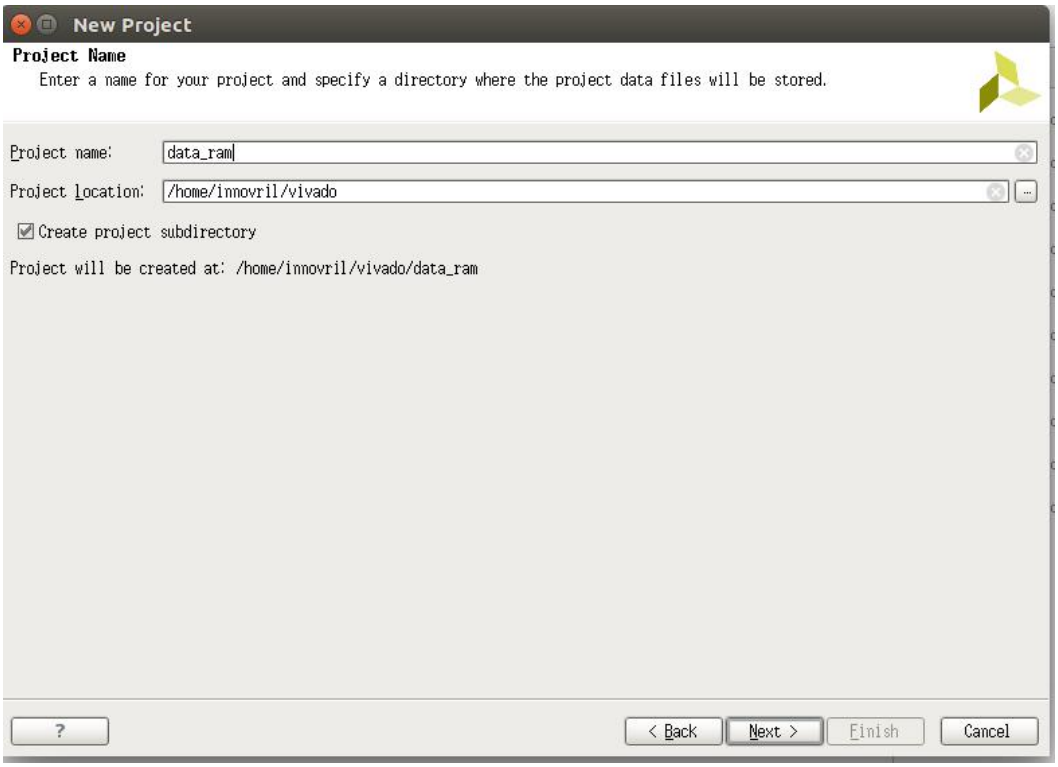


图 6.2 新建工程 data_ram

2）新建 IP

点击“IP Catalog”：

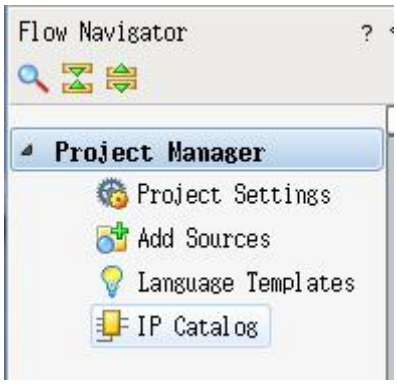


图 6.3 打开 IP 目录

在右侧列表中双击选择"Memories and Storage Elements"->"RAMs & ROMs & BRAM"中的"Block Memory Generator"：

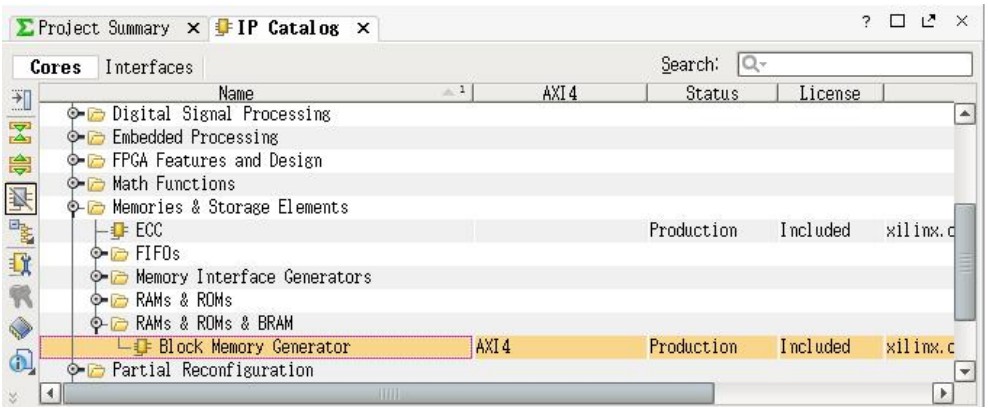


图 6.4 选择 IP 类型

3) 设置 RAM 参数

完成第二步后，会出现如下界面，需要依次选择 Memory 的参数：

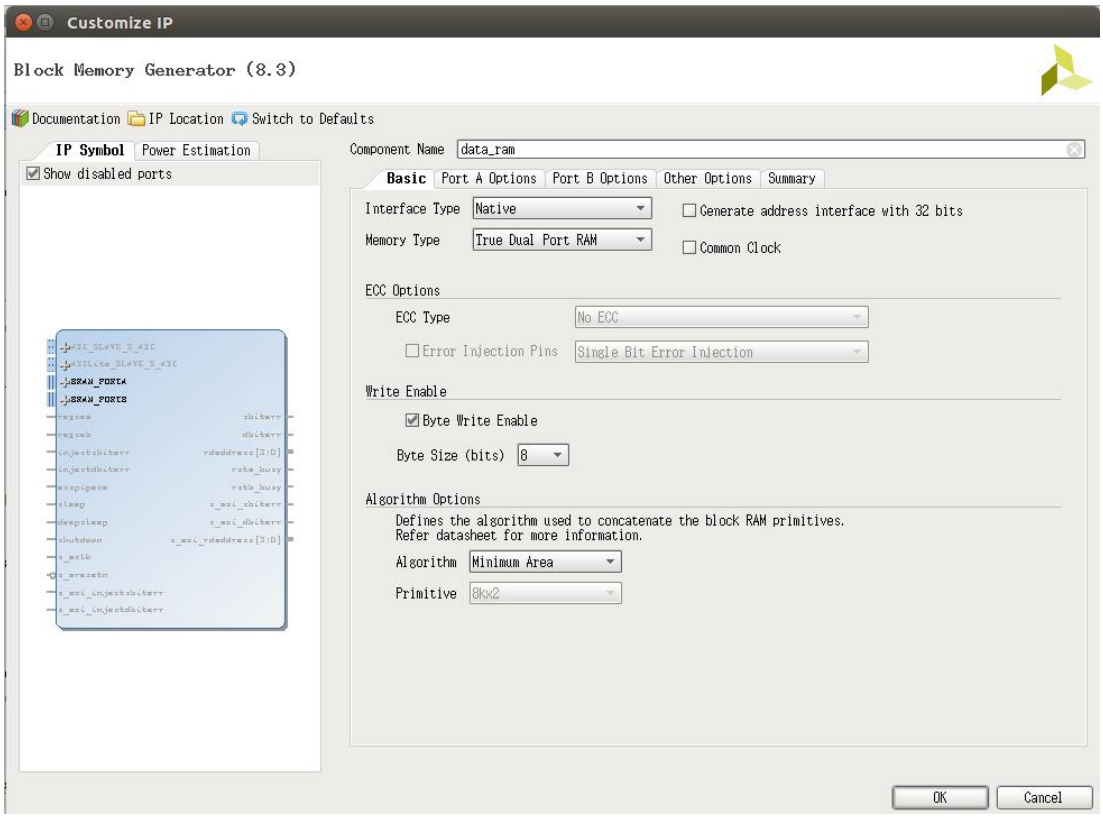


图 6.5 Memory 设定参数界面

在图 6.5 中，Component Name 输入"data_ram"，选择 Memory 类型为“True Dual Port RAM”，一个端口作为正常的读写端口，一个端口作为调试端口。勾选“Write Enable”下的“Byte Write Enable”，“Byte Size”选 8bits，因为后续 CPU 实验中存在写一个字节的 store 指令，故需要数据 RAM 为字节写使能。"Basic"部分设置完成后击“Port A options”选项卡：

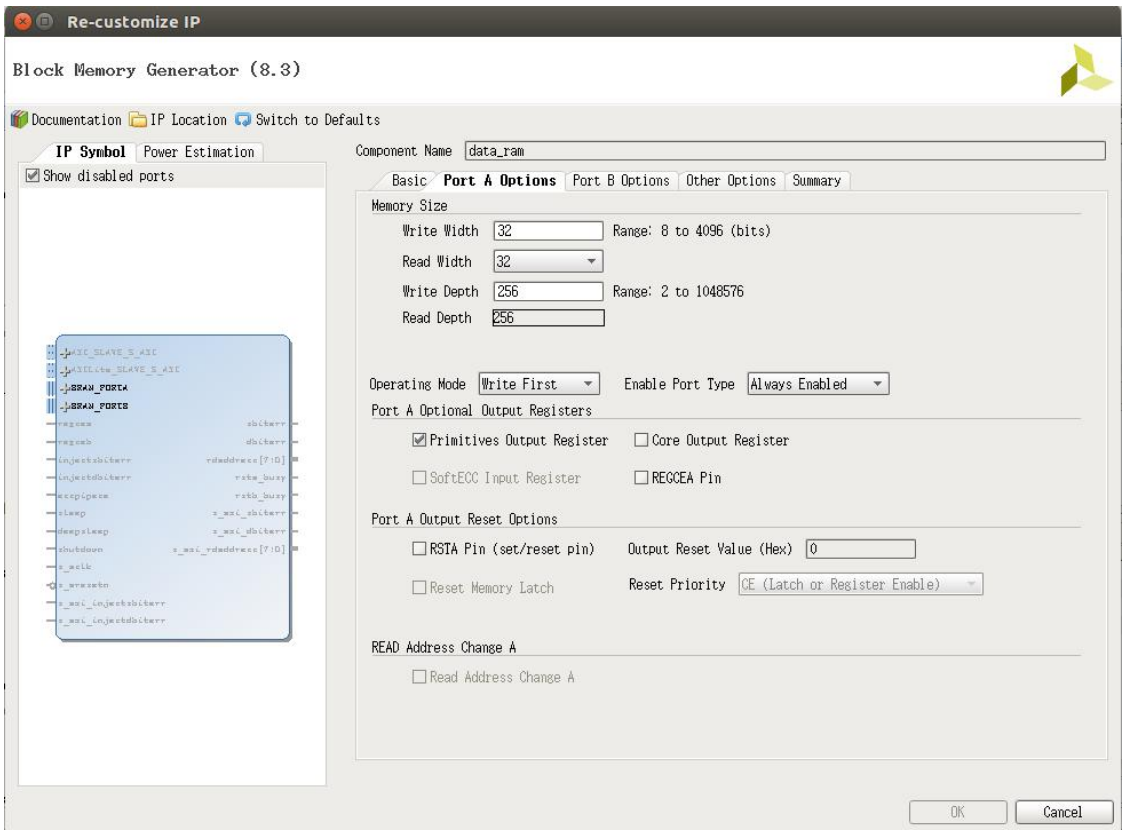


图 6.6 选择 Memory 类型

在图 6.6 中，RAM 宽度设置为 32 位，深度为 256，因为后续 CPU 实验是基于 32 位数据运算的。"Enalbe Port Type"选择"Always Enabled"。在“Port B options”选项卡下进行同样的设置。

对于一般的 RAM 生成，后续的步骤都不需要了，故此处直接点击“OK”，在弹出的窗口中点击“Generate”生成 IP 核即可。

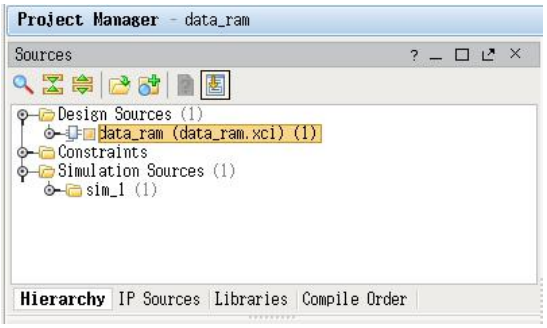


图 6.7 生成 RAM 成功

生成成功的工程管理界面如图 6.7，.xci 文件为存放 IP 核配置信息的文件，以后需要更改直接双击该文件即可重新配置 IP 核。

5.2 生成 IP 核 ROM

ROM 为只读存储器，需要初始化内部数据，可作为指令存储器。生成 ROM 的方法同 RAM 类似，首先新建一个 IP 核，取名为 inst_rom,后续步骤同 5.1 所述，但在图 6.5 那步时，Memory 类型需要选择为“Single Port ROM”，如下图：

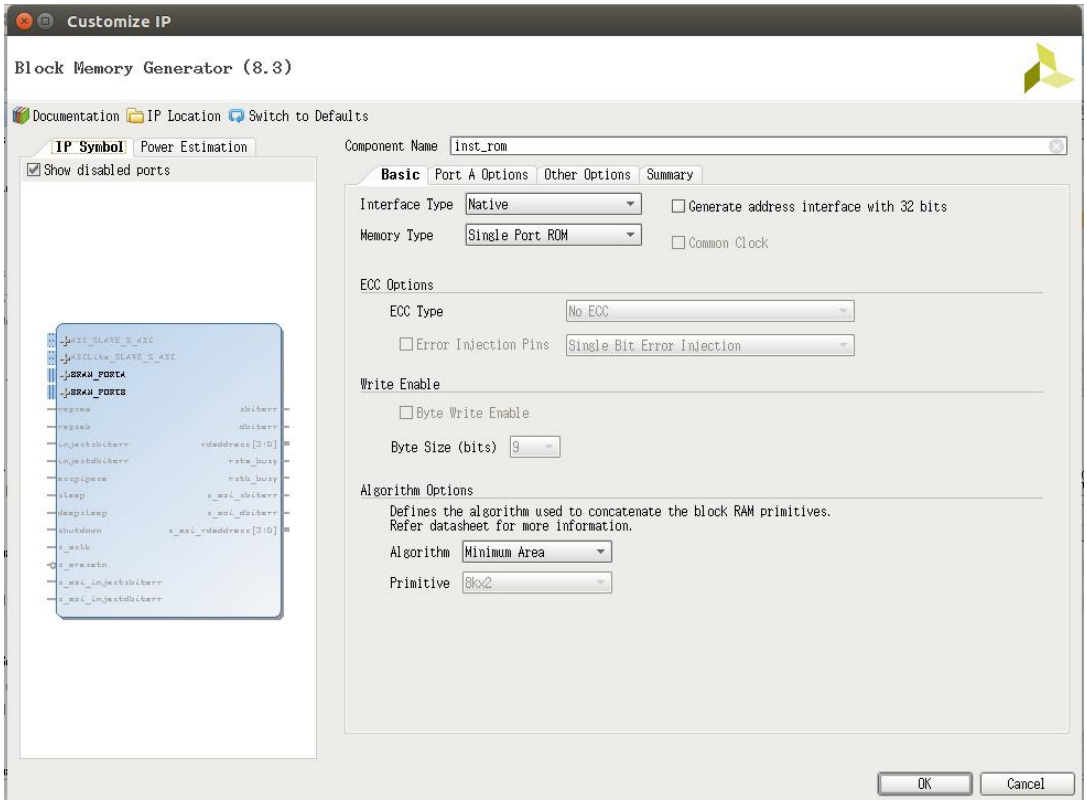


图 6.8 生成 ROM 时选择 Memory 类型

由于 ROM 为只读的，故我们不需要增加调试端口去观察内部数据的变化，故此处选择单端口即可。

点击“Port A Options”设置宽度和深度，"Enalbe Port Type"选择"Always Enabled"，如下图：

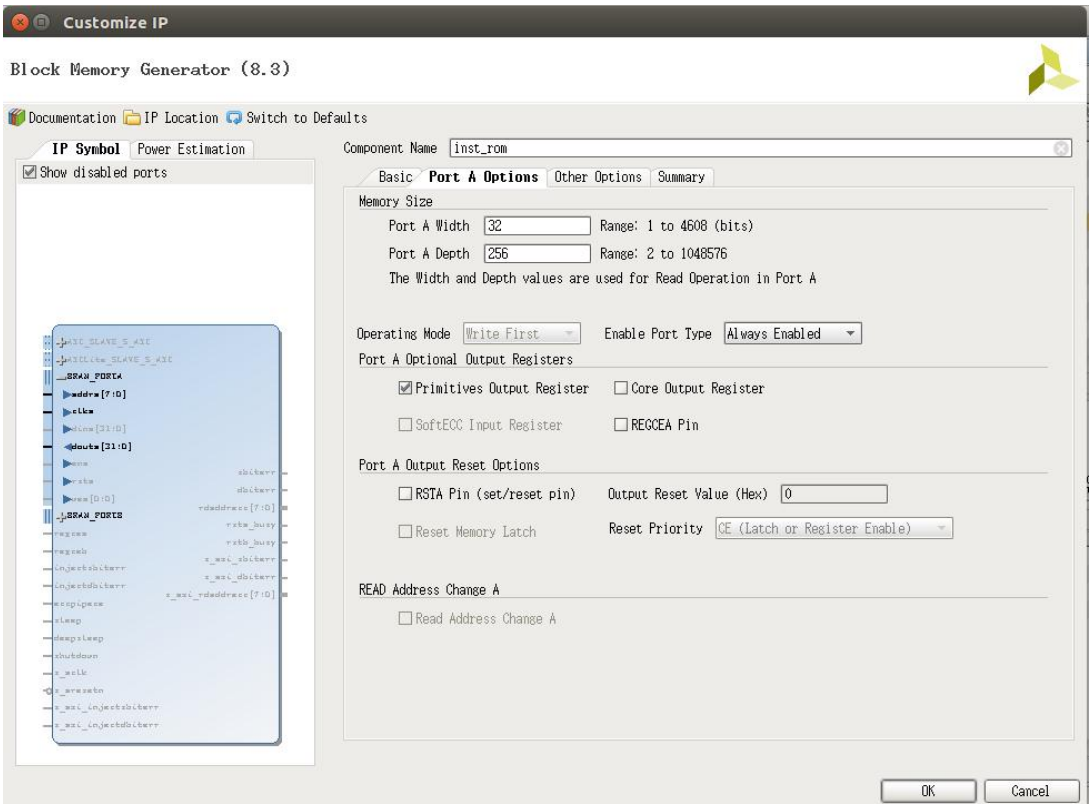


图 6.9 生成 ROM 时宽度和深度

宽度需要设置为 32 位，因为一条指令占用 32 位，深度可以依据自己要执行的指令数设定，此处先设定为 256。

点击“Other Options”选项卡：

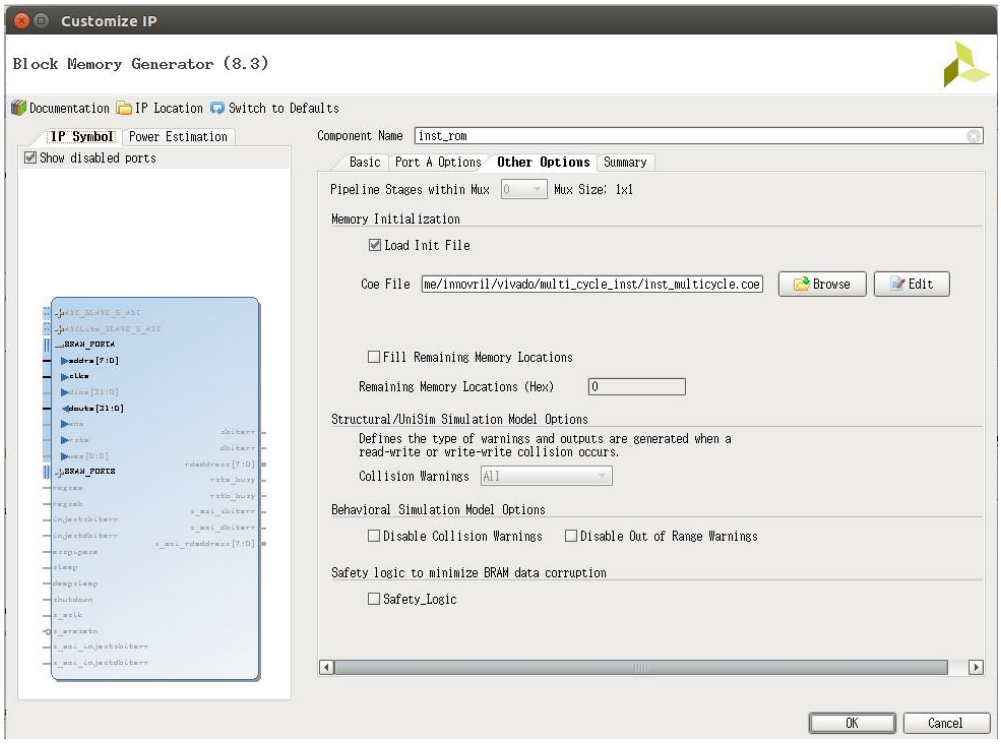


图 6.10 给 ROM 装载初始数据

在图 6.11 中，需要勾选“Load Init File”，并选中需要装载的初始化文件(.coe 文件)。.coe 文件为 Vivado 中 ROM 初始化文件，其格式如下：

```
1  memory_initialization_radix = 16;
2  memory_initialization_vector =
3  24010001
4  00011100
5  ……
```

第一行指定了初始化数据格式，此处为 16 进制，也可以设置为 2 进制。第二行说明从第三行开始为初始化的数据向量，由于 ROM 宽度为 32 位，故一个初始化向量为 32 位数据。初始化向量之间必须用空格或换行符隔开，此处使用换行符，故一行为一个初始化向量。初始化数据会从 ROM 中的 0 地址处开始依次填充。当初始化数据格式设置为 2 进制时，后续的初始化向量需要用二进制编写。

至此，生成 ROM 的所有参数都已设置完成，点击"OK"，“Generate”生成 ROM 即可。

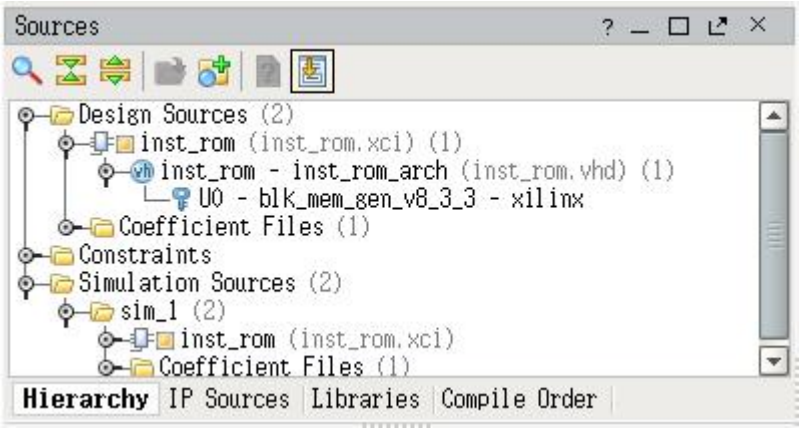


图 6.12 生成 ROM 成功

从图 6.12 中可以看到，生成结果和 RAM 的类似，以后需要更改参数设置或初始化文件时，双击中的 xci 文件即可。

七、 实验六 单周期 CPU 实现

1 实验目的

1. 理解 MIPS 指令结构，理解 MIPS 指令集中常用指令的功能和编码，学会对这些指令进行归纳分类。
2. 了解熟悉 MIPS 体系的处理器结构，如延迟槽，哈佛结构的概念。
3. 熟悉并掌握单周期 CPU 的原理和设计。
4. 进一步加强运用 verilog 语言进行电路设计的能力。
5. 为后续设计多周期 cpu 的实验打下基础。

2 实验设备

1. 装有 Xilinx Vivado 的计算机一台。
2. LS-CPU-EXB-002 教学系统实验箱一套。

3 实验任务

1. 学习 MIPS 指令集，深入理解常用指令的功能和编码，并进行归纳确定处理器各部件的控制码，比如使用何种 ALU 运算，是否写寄存器堆等。
2. 确定自己本次实验中的准备实现的 MIPS 指令，要求至少实现一条 load 指令、一条 store 指令、10 条基础运算指令、一条跳转指令。其中基础运算指令最好包含多种类型的操作，必须包含一条加法和一条减法指令。不考虑指令可能产生异常的情况。单周期 CPU 的实验重点是搭建出一个 CPU 架构，为避免被繁琐的指令所困惑，建议在单周期 CPU 实验中只实现十几条指令。
3. 对准备实现的指令进行分析，完成表 7.1 的填写。

表 7.1 mips 基础指令特性归纳表

指令类型	汇编指令	指令码	源操作数 1	源操作数 2	目的寄存器	功能描述
R 型指令	addu rd , rs , rt	000000_rs_|_rt_|rd_|00000|100001	[rs]	[rt]	rd	GPR[rd]= GPR [rs]+ GPR [rt]
I 型指令	addiu rt,rs,imm	001001_rs_|_rt|imm	[rs]	sign_ext(imm)	rt	GPR [rt]= GPR [rs]+sign_ext(imm)
J 型指令	j target	000010|target	PC	target		跳转, PC={PC[31:28],target,2'b00}

注：GPR 表示通用寄存器，[rs]表示寄存器 rs 里存储的值，PC 表示程序计数器；imm 为 16 位立即数，sign_ext(imm)表示对其进行符号扩展；target 为 26 位立即数。

4. 自行设计本次实验的方案，画出结构框图，大致结构框图如图 7.1。图 7.1 中粗线表示接口位数和种类不定，需要在自己的结构框图中详细给出。从图 7.1 中可以看出，本次实验是需要用到之前实验的结果的，比如 ALU 模块、寄存器堆模块、指令 ROM 模块和数据 RAM 模块，其中 ROM 和 RAM 要使用自行搭建的异步存储器。

单周期 CPU 是指一条指令的所有操作在一个时钟周期内执行完。设计中所有寄存器和存储器都是异步读同步写的，即读出数据不需要时钟控制，但写入数据需时钟控制。

故单周期 CPU 的运作即：在一个时钟周期内，根据 PC 值从指令 ROM 中读出相应的指令，将指令译码后从寄存器堆中读出需要的操作数，送往 ALU 模块，ALU 模块运算得到结果。

如果是 store 指令，则 ALU 运算结果为数据存储的地址，就向数据 RAM 发出写请求，在下一个时钟上升沿真正写入到数据存储器。

如果是 load 指令，则 ALU 运算结果为数据存储的地址，根据该值从数据存 RAM 中读出数据，送往寄存器堆根据目的寄存器发出写请求，在下一个时钟上升沿真正写入到寄存器堆中。

如果非 load/store 操作，若有写寄存器堆的操作，则直接将 ALU 运算结果送往寄存器堆根据目的寄存器发出写请求，在下一个时钟上升沿真正写入到寄存器堆中。

如果是分支跳转指令，则是需要将结果写入到 pc 寄存器中的。

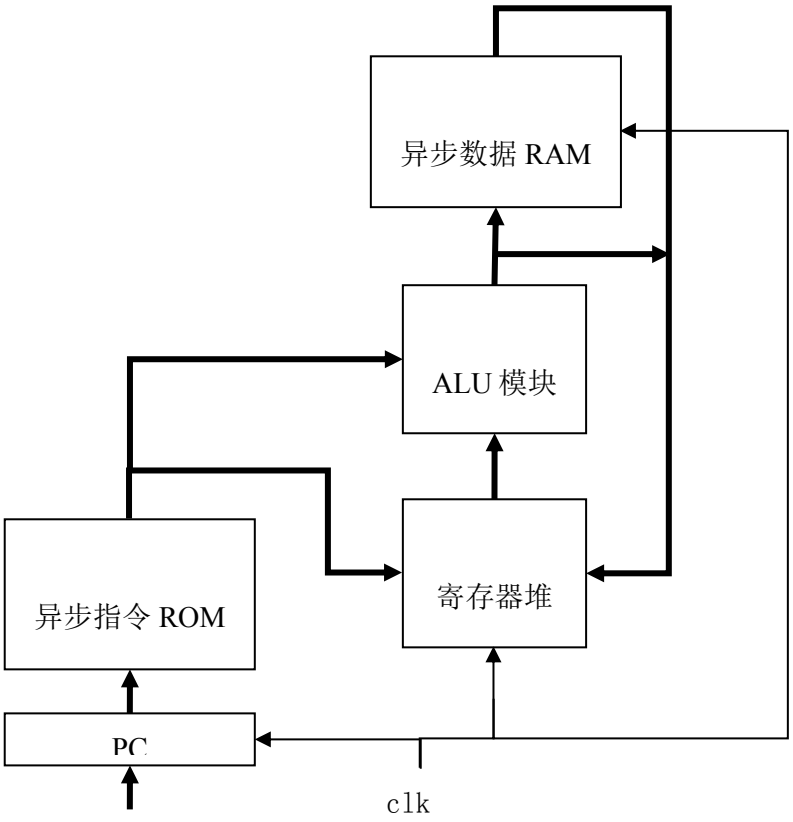


图 7.1 单周期 CPU 的大致框图

5. 根据设计的实验方案，使用 verilog 编写相应代码。

6. 依据自己设计中实现的指令，编写一段不少于 20 行的汇编程序，力求验证到所有实现的指令。该段汇编程序是需要内嵌到自行搭建的异步指令 ROM 中的。完成表 7.2 的填写。

表 7.2 测试所用汇编程序详述

指令地址	汇编指令	结 果描 述	机器指令的机器码	
			16 进制	二进制
00H	addiu ,$1, $0,#1	[$1] = 0000_0001H	24010001	0010_0100_0000_0001__0000_0000_0000_0001

7. 对编写的代码进行仿真，得到正确的波形图。

8. 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，见图 7.2。外围模块中需调用封装好的 LCD 触摸屏模块，观察单周期 CPU 的内部状态，比如 32 个寄存器的值，PC 的值等。并且需要利用触摸功能输入特定数据 RAM 地址，从该 RAM 的调试端口读出数据显示在屏上，以达到实时观察数据存储器内部数据变化的效果。通过这些手段，可以在板上充分验证 CPU 的正确性。

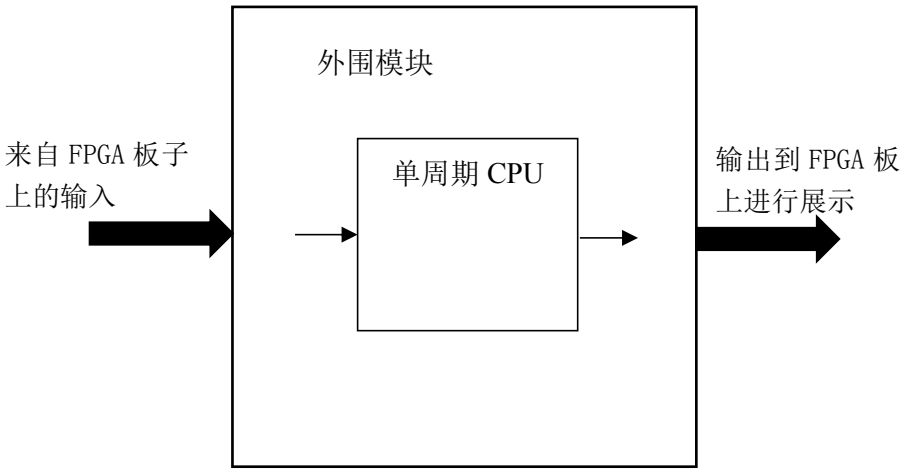


图 7.2 单周期 CPU 设计实验的顶层模块大致框图

9. 将编写的代码进行综合布局布线，并下载到实验箱中的 FPGA 板子上进行演示。

注意：

①：MIPS 架构中有延迟槽的设定，其本意是加快流水 CPU 的执行速度，故在单周期 CPU 中该设定毫无意义，反而带来了 CPU 实现上的麻烦，故建议在单周期 CPU 中不考虑延迟槽技术。

②：MIPS 架构中分支和跳转指令参与计算的 PC 值均为延迟槽指令对应的 PC(即分支跳转指令的 PC+4),而由于单周期不考虑延迟槽，故在实验中分支跳转指令参与计算使用本指令的 PC 值，故跳转的偏移量设置需要注意下。比如一条指令“beq,r0,r0,#2”在不考虑延迟槽的单周期 CPU 中，其跳转的目标地址为 beq 指令后面的第 2 条。而在考虑延迟槽的流水 CPU 中，其跳转的目标地址为 beq 指令后面的第 3 条（即延迟槽指令后面的第 2 条）。这里需要理解清楚。

4 实验要求

1. 做好预习：
   1） 熟知 MIPS 指令类型，深入理解常用指令的功能和编码；
   2） 归纳常用的 MIPS 指令，确定自己准备实现的 MIPS 指令；
   3） 对准备实现的指令进行分析，完成表 7.1 的填写；
   4） 设计本次实验的方案，画出实验方案的设计框图，即补充完善图 7.1；
   5） 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 7.2；
   6） 依据自己设计中实现的指令，编写一段不少于 20 行的汇编程序，要求包含所有实现的指令，完成表 7.2 的填写。
2. 实验实施：
   1） 确认单周期 CPU 的设计框图的正确性；
   2） 编写 verilog 代码，将表 7.2 中自己编写的汇编程序翻译为二进制，内嵌到指令 ROM 中；
   3） 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；
   4） 完成调用单周期 CPU 的外围模块的设计，并编写代码；
   5） 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。
3. 实验检查：
   1） 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。先解读表 7.2 中自己编写的汇编程序，然后采用手动输入时钟，每个周期查看 CPU 状态，按照检查人员的要求进行演示，检查指令运行结果的正确性，可对演示结果进行拍照作为实验报告结果一项的材料。
4. 实验报告的撰写：
   1） 实验结束后，需按照规定的格式完成实验报告的撰写。

八、 实验七 多周期 CPU 实现

1 实验目的

1. 在单周期 CPU 实验完成的提前下，理解多周期的概念。
2. 熟悉并掌握多周期 CPU 的原理和设计。
3. 进一步提升运用 verilog 语言进行电路设计的能力。
4. 为后续实现流水线 cpu 的课程设计打下基础。

2 实验设备

1. 装有 Xilinx Vivado 的计算机一台。
2. LS-CPU-EXB-002 教学系统实验箱一套。

3 实验任务

1. 本次实验是对单周期 CPU 实验的拔高，也是为流水线 CPU 打下基础。前期的实验准备同单周期 CPU 的实验，在单周期 CPU 中只要求实现了十几条指令，但此处要求扩展到 30 多条指令。

多周期 CPU 是指，一条指令需要花费多个周期才能完成所有操作，在每个周期内只做一部分操作，比如：取指、译码、执行、访存、写回，此时，一条指令执行完，共需 5 个周期，每个周期只做一部分操作。

将 CPU 划分为多周期的优势在于，每个时钟周期内 CPU 需要做的工作就变少，因此频率可以更高，且每个部件做的事情单一了，比如取指部件只负责从指令存储器中取出指令，因此 CPU 可以进行流水工作，也相当于一个时钟周期完成一条指令。频率更高，依然相当于是一个周期完成一条指令，因此 CPU 可以运行的更快。

本次实验就是将实验六所实现的单周期 CPU 划分为多周期的，并扩展指令到 30 多条。

2. 依据单周期实验的设计框图，将其划分为多个功能块，每个功能块占用一个周期完成，即每个功能块从上一个功能块获取信息，作相关动作，完成后将结果锁存到寄存器中，作为下一个功能块的输入。建议划分为教课书上的 5 个功能块：取指、译码、执行、访存、写回，将理论与实践相结合。

3. 画出划分后的多周期 CPU 的框图，大致框图如图 8.1。从图中可以看出指令每个周期走完一个功能块，进入下一个功能块。标注的 clk 箭头是去往相邻模块的中间锁存器，是因为每个模块的输出需要锁存到寄存器中，下一个模块会从该寄存器中读出数据作为自己的输入，寄存器的锁存是需要时钟控制的。值得注意的是，写回模块所做的就是从访存模块获取要写入寄存器堆的数据和目的寄存器，送往寄存器堆，其所需的 clk 信号是最终发生在寄存器堆的写操作上的。自己设计的框图中要求力求精细，可参考教科书上的 5 级流水的框图。

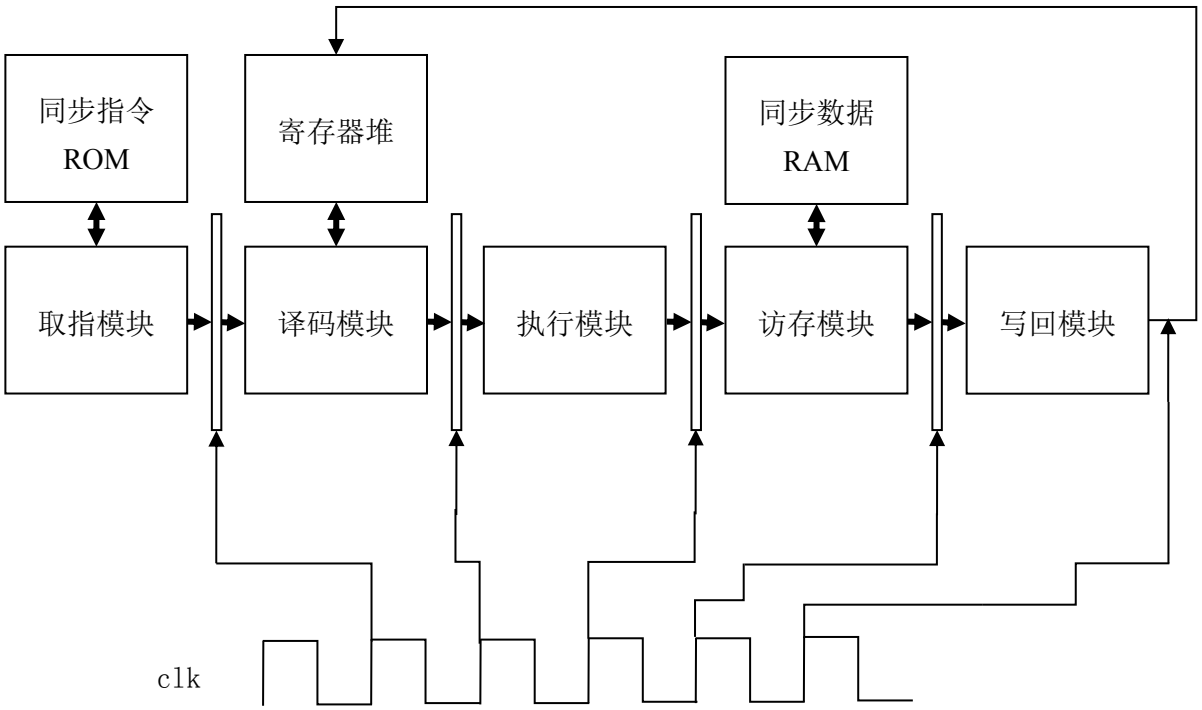


图 8.1 多周期 CPU 的大致框图

4. 本次实验是需要用到之前实验的结果的，比如 ALU 模块、寄存器堆模块、指令 ROM 模块和数据 RAM 模块，其中 ROM 和 RAM 建议使用调用库 IP 实例化的同步存储器，因为存储器在实际应用中基本都是同步读写的，为了更贴近真实情况，此处建议使用同步 RAM 和 ROM。

5. 在实验五中生成的同步 RAM 和 ROM，都是在发送地址后的下一拍才能获得对应数据的。故在在使用同步存储器时，从指令和数据存储器中读取数据就需要等待一拍时钟了，即取指令需要两拍时间，load 操作也需要两拍时间。在真实的处理器系统中，取指令和访存其实都是需要多拍时钟的。

6. 本次实验，同样需要完成表 8.1 和表 8.2 的填写。

表 8.1 mips 基础指令特性归纳表

指令类型	汇编指令	指令码	源操作数 1	源操作数 2	目的寄存器	功能描述
R 型指令	addu rd , rs , rt	000000_rs_|_rt_|rd_|00000|100001	[rs]	[rt]	rd	GPR[rd]=GPR[rs]+GPR[rt]
I 型指令	addiu rt,rs,imm	001001_rs_|_rt_|imm	[rs]	sign_ext(imm)	rt	GPR[rt]=GPR[rs]+sign_ext(imm)
J 型指令	j target	000010|target	PC	target		跳转, PC={PC[31:28],target,2'b00}

表 8.2 测试所用汇编程序详述

指令地址	汇编指令	结 果描 述	机器指令的机器码	
			16 进制	二进制
00H	addiu ,$1, $0,#1	[$1] = 0000_0001H	24010001	0010_0100_0000_0001__0000_0000_0000_0001

7. 根据设计的实验方案，使用 verilog 编写相应代码。

8. 对编写的代码进行仿真，得到正确的波形图。

9. 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，见图 8.2。外围模块中需调用封装好的 LCD 触摸屏模块，观察多周期 CPU 的内部状态，比如 32 个寄存器的值，各模块 PC 的值等。并且需要利用触摸功能输入特定数据 RAM 地址，从该 RAM 的调试端口读出数据显示在屏上，以达到实时观察数据存储器内部数据变化的效果。通过这些手段，可以在板上充分验证 CPU 的正确性。

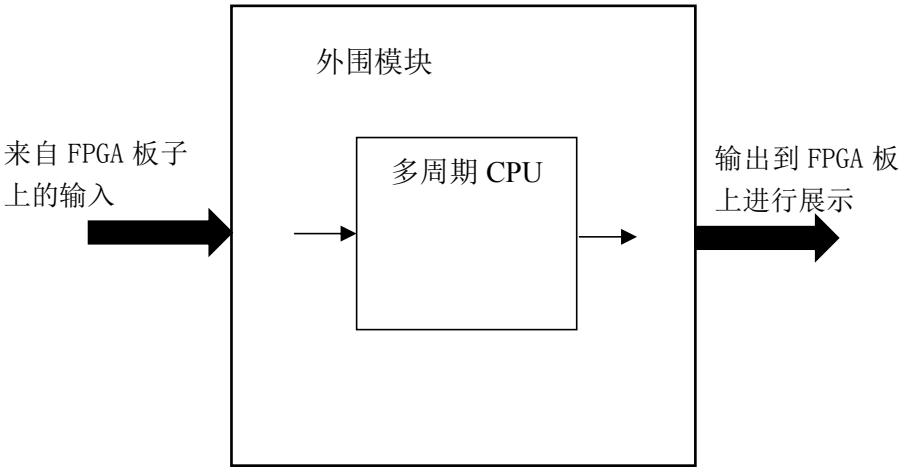


图 8.2 多周期 CPU 设计实验的顶层模块大致框图

10. 将编写的代码进行综合布局布线，并下载到实验箱中的 FPGA 板子上进行演示。

注意：

①：MIPS 架构中有延迟槽的设定，其本意是加快流水 CPU 的执行速度，故在多周期 CPU 中该设定无意义，反而带来了 CPU 实现上的麻烦，故建议在多周期 CPU 中不考虑延迟槽技术。

②：MIPS 架构中分支和跳转指令参与计算的 PC 值均为延迟槽指令对应的 PC(即分支跳转指令的 PC+4),而由于多周期不考虑延迟槽，故在实验中分支跳转指令参与计算使用本指令的 PC 值，故跳转的偏移量设置需要注意下。比如一条指令

“beq,r0,r0,#2”在不考虑延迟槽的多周期 CPU 中，其跳转的目标地址为 beq 指令后面的第 2 条。而在考虑延迟槽的流水 CPU 中，其跳转的目标地址为 beq 指令后面的第 3 条（即延迟槽指令后面的第 2 条）。这里需要理解清楚。

③：一般而言控制 CPU 运转的时钟是由 FPGA 板上的时钟输出提供的，但为了方便演示，我们需要在每一个时钟里查看一条指令的运算结果，故演示时理想的时钟是手动输入的，可以使用 FPGA 板上的脉冲开关代替时钟。

4 实验要求

1. 做好预习：

   1） 复习单周期 CPU 的实验内容，归纳常用的 MIPS 指令，确定自己准备实现的 MIPS 指令，对其进行分析，完成表 8.1 的填写；

   2） 依据自己设计中实现的指令，编写一段不少于 40 行的汇编程序，要求包含所有实现的指令，完成表 8.2 的填写；

   3） 认真学习多周期的概念，了解流水线的概念，明白划分为多周期的意义；

   4） 认真学习 CPU 各模块的功能，确认模块的划分。设计本次实验的方案，画出实验方案的设计框图，即补充完善图 8.1；

   5） 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 8.2。

2. 实验实施：

   1） 确认多周期 CPU 的设计框图的正确性；

   2） 编写 verilog 代码，将表 8.2 中自己编写的汇编程序翻译为二进制，以 coe 文件的方式初始化到指令 ROM 中；

   3） 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，在仿真时需要将生成指令 ROM 时产生的.mif 文件拷贝到工程目录下，才能仿真成功；

   4） 完成调用多周期 CPU 的外围模块的设计，并编写代码；

   5） 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。

3. 实验检查：

   1） 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。先解读表 8.2 中自己编写的汇编程序，然后采用手动输入时钟，每个周期查看 CPU 状态，按照检查人员的要求进行演示，检查指令运行结果的正确性，可对演示结果进行拍照作为实验报告结果一项的材料。

4. 实验报告的撰写：

   1） 实验结束后，需按照规定的格式完成实验报告的撰写。

九、 课程设计 静态 5 级流水线 CPU 实现

1 课程设计目的

1. 在多周期 CPU 实验完成的提前下，深入理解 CPU 流水线的概念。
2. 熟悉并掌握流水线 CPU 的原理和设计。
3. 最终检验运用 verilog 语言进行电路设计的能力。
4. 通过亲自设计实现静态 5 级流水线 CPU，加深对计算机组成原理和体系结构理论知识的理解。
5. 培养对 CPU 设计的兴趣，加深对 CPU 现有架构的理解和深思。

2 实验设备

1. 装有 Xilinx Vivado 的计算机一台。
2. LS-CPU-EXB-002 教学系统实验箱一套。

3 课程设计任务

1. 本课程设计是对之前课程实验的拔高。前期的课程设计准备同多周期 CPU 的实验，主体部分可以直接使用多周期 CPU 实验的设计方案，但在多周期 CPU 中只要求实现了 30 多条指令，此处要求扩展到 40 多条指令。

多周期 CPU 在单周期基础上提高了时钟频率，但并没有改善执行一条指令的时间，且存在资源闲置的问题，例如当指令在执行级有效时，译码级实际上在空转。若每一级都在执行有效的指令，将解决资源闲置的问题。最理想的情况是，当第一条指令从取指级转换到下一级译码时，第二条指令进入取指级，当第一条指令完成译码进入执行级时，第二条指令进入译码级，第三条指令进入取指级……静态 5 级流水 CPU 就是基于这样的设计思路。

在流水 CPU 中，当在一时钟周期内完成了某一条指令的全部执行时（写回级完成），则有望在下一时钟周期内完成下条指令的执行，因此依然相当于是一个周期完成一条指令，而时钟频率更高，因此 CPU 可以运行的更快。

本次课程就是将上次所实现的多周期 CPU 更改结构为静态 5 级流水线 CPU。

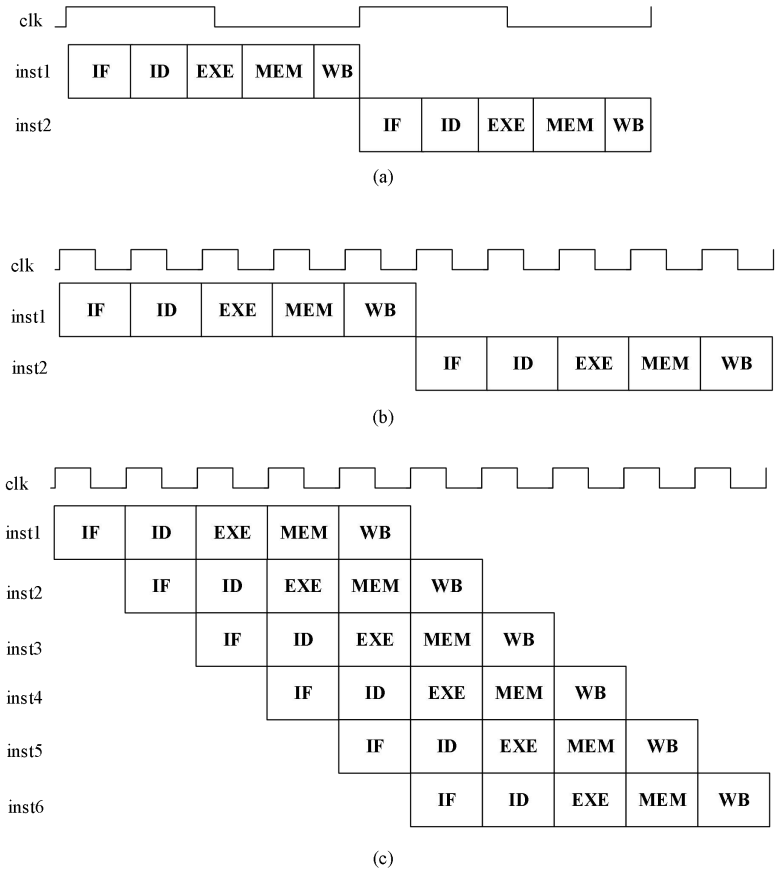


图 9.1　(a)单周期; (b)多周期; (c)5 级流水 CPU 时空图

2. 本次课程设计的设计框图基本同多周期 CPU 实验的设计框图。需要注意 5 个部件都是同时运转的，但对每条指令而言，依然是依次工作的，见图 9.2。

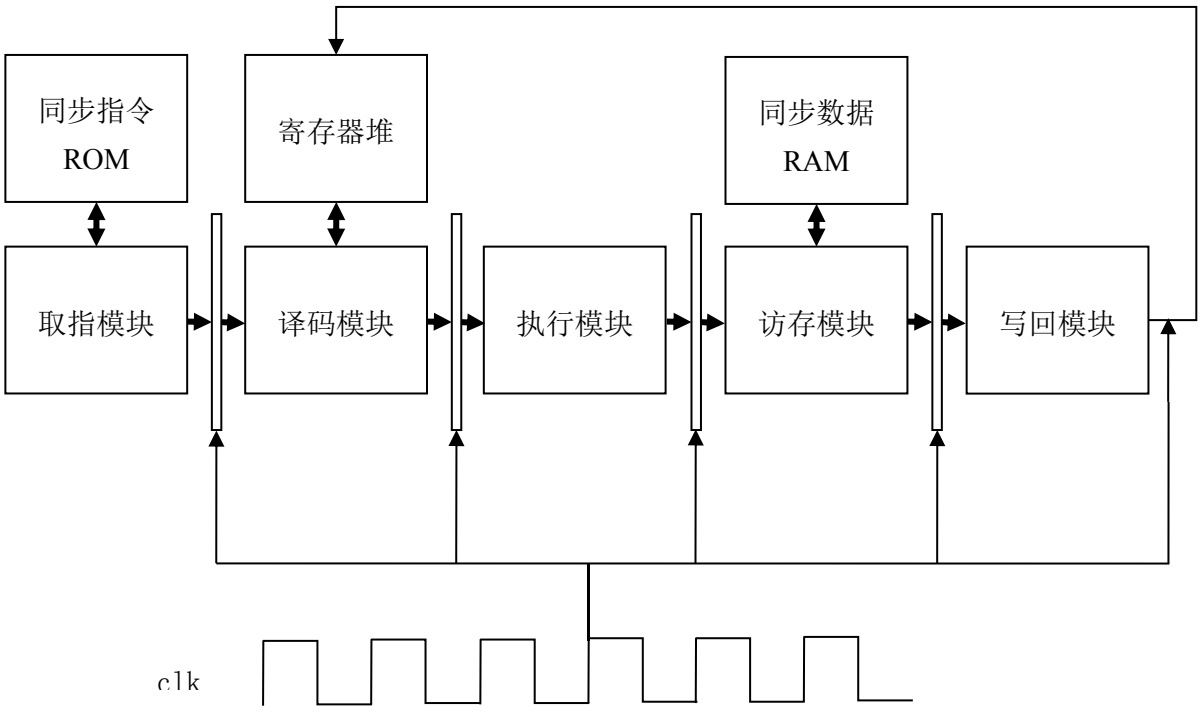


图 9.2　静态 5 级流水 CPU 的大致框图

3. 本次课程设计的关键和难点在于流水线的控制，比如一条指令何时可以从译码级进入执行级，一条指令何时需要堵在流水线中。本次课程设计暂不考虑前递技术，因此有数据相关时就需要堵在流水线中。

4. MIPS 架构中有延迟槽的设定，其本意是加快流水 CPU 的执行速度。在之前单周期和多周期 CPU 实验中未支持延迟槽，但在流水 CPU 中需要支持该设定，因为只有硬件支持了延迟槽技术，用通用编译器编译出来的二进制执行文件才能在自己设计的 CPU 中运行正确。

5. MIPS 架构中分支和跳转指令参与计算的 PC 值均为延迟槽指令对应的 PC(即分支跳转指令的 PC+4),在本课程设计中尤其需要注意这一点。比如一条指令“beq,r0,r0,#2”在不考虑延迟槽的多周期 CPU 中，其跳转的目标地址为 beq 指令后面的第 2 条。而在考虑延迟槽的流水 CPU 中，其跳转的目标地址为 beq 指令后面的第 3 条（即延迟槽指令后面的第 2 条）。在编写测试程序时就需要注意分支跳转指令的偏移量。

6. 本次实验，同样需要完成表 9.1 和表 9.2 的填写。

表 9.1 mips 基础指令特性归纳表

指令类型	汇编指令	指令码	源操作数 1	源操作数 2	目的寄存器	功能描述
R 型指令	addu rd , rs , rt	000000_rs_|_rt_|rd_|00000|100001	[rs]	[rt]	rd	GPR[rd]=GPR[rs]+GPR[rt]
I 型指令	addiu rt,rs,imm	001001_rs_|_rt_|imm	[rs]	sign_ext(imm)	rt	GPR[rt]=GPR[rs]+sign_ext(imm)
J 型指令	j target	000010|target	next_pc	target		跳转, PC={next_pc[31:28],target,2’b00}

表 9.2 测试所用汇编程序描述表

指令地址	汇编指令	结 果描 述	机器指令的机器码	
			16 进制	二进制
00H	addiu ,$1, $0,#1	[$1] = 0000_0001H	24010001	0010_0100_0000_0001__0000_0000_0000_0001

7. 根据设计的实验方案，使用 verilog 编写相应代码。

8. 对编写的代码进行仿真，得到正确的波形图。

9. 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，见图 9.3。外围模块中需调用封装好的 LCD 触摸屏模块，观察 CPU 的内部状态，比如 32 个寄存器的值，各级 PC 的值等。并且需要利用触摸功能输入特定数据 RAM 地址，从该

RAM 的调试端口读出数据显示在屏上，以达到实时观察数据存储器内部数据变化的效果。通过这些手段，可以在板上充分验证 CPU 的正确性。

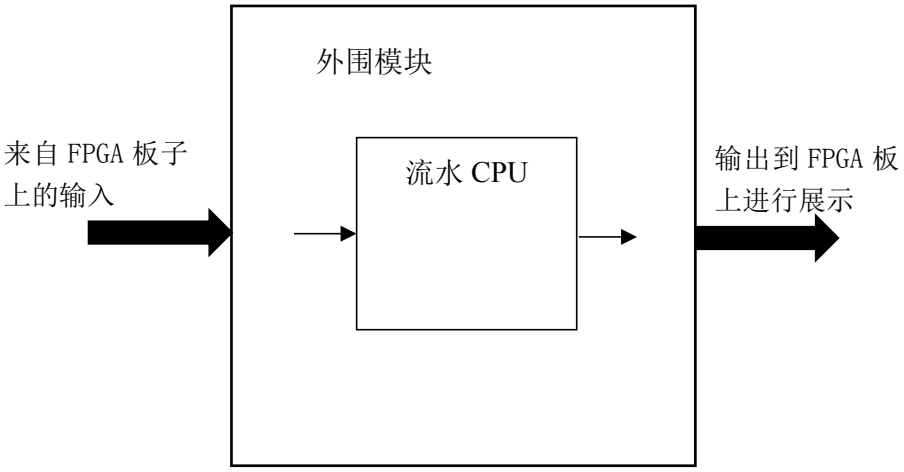


图 9.3 静态 5 级流水 CPU 设计实验的顶层模块大致框图

10. 将编写的代码进行综合布局布线，并下载到实验箱中的 FPGA 板子上进行演示。

注意：一般而言控制 CPU 运转的时钟是由 FPGA 板上的时钟输出提供的，但为了方便演示，我们需要在每一个时钟里查看一条指令的运算结果，故演示时理想的时钟是手动输入的，可以使用 FPGA 板上的脉冲开关代替时钟。

4 课程设计要求

1. 做好预习：

   1） 复习好上一次多周期 CPU 的实验，归纳常用的 MIPS 指令，确定自己准备实现的 MIPS 指令，对其进行分析，完成表 9.1 的填写；

   2） 认真学习流水线的概念，明白流水线的意义和架构，理解数据相关、控制相关和结构相关，尤其需要注意分支跳转指令及其延迟槽指令的处理；

   3） 依据自己设计中实现的指令，编写一段不少于 50 行的汇编程序，要求包含所有实现的指令，完成表 9.2 的填写。要求标注出指令间存在的相关，指出 CPU 可能存在的阻塞；

   4） 在多周期 CPU 实验的设计框图的基础上，完善课程设计的设计框图，即补充完善图 9.2；

   5） 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 9.3。

2. 实验实施：

   1） 确认流水 CPU 的设计框图的正确性；

2） 编写 verilog 代码，将表 9.2 中自己编写的汇编程序翻译为二进制，以 coe 文件的方式初始化到指令 ROM 中；

3） 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，在仿真时需要将生成指令 ROM 时产生的.mif 文件拷贝到工程目录下，才能仿真成功；

4） 完成调用流水 CPU 的外围模块的设计，并编写代码；

5） 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。

3. 实验检查：

1） 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。先解读表 9.2 中自己编写的汇编程序，然后采用手动输入时钟，每个周期查看 CPU 状态，按照检查人员的要求进行演示，检查指令运行结果的正确性，可对演示结果进行拍照作为实验报告结果一项的材料。

4. 实验报告的撰写：

1） 实验结束后，需按照规定的格式完成实验报告的撰写。

十、 课程设计拓展 优化 CPU 系统

1 课程设计拓展目的

1. 加深对计算机组成原理和体系结构理论知识的理解。
2. 培养对 CPU 设计的兴趣，在理解现有 CPU 架构的基础上，引发对体系结构的思考和创新。
3. 培养创新思维能力，并通过实践验证新想法。

2 实验设备

1. 装有 Xilinx Vivado 的计算机一台。
2. LS-CPU-EXB-002 教学系统实验箱一套。

3 课程设计拓展题说明

本拓展是对之前课程设计的拔高，是为大家在学有余力的时候设定的，旨在发挥大家的创新思维能力，为大家提供一个学习反思验证的环境，可以尽情地提出自己关于计算机组成原理和体系结构的想法，大胆地对书中的知识提出疑问，通过设计实验方案验证自己的想法，从而获得对课本知识的升华。

因此本拓展会提出几个拓展题，供大家思考，大家可以选择性地实现，也可以全部实现，当然更鼓励大家地实现自己的新想法。为充分发挥大家的主观能动性，也为了不限制大家的思维，各拓展题只会给出现有问题，并不会像以往的实验说明给出初步的实验方案。

4 课程设计拓展题

1. 分析静态 5 级流水 CPU 中的流水线阻塞情况，包括数据相关、控制相关、结构相关等，优化流水线设计，尽可能减少流水线阻塞情况，比如前递技术等。
2. 对于分支跳转指令，mips 架构中有延迟槽指令的设定，利用这一点，在静态 5 级流水 CPU 中，可实现分支指令永远不阻塞后续指令，大家可以检查自己的流水线设定，进行优化实现这一点。
3. 针对第 2 点，此时，分支跳转指令就不需要进行转移猜测了，但大家可以将流水线结构改为 x86 中无延迟槽技术的设定，此时分支跳转指令与后续真正需要执行的指令至少会堵塞一拍，此时可以考虑实现转移预测技术，提升流水线结构。
4. 在学习的过程中，大家一定会有很多自己的想法，比如，为什么取指、译码、执行、访存、写回称为经典的 5 级流水结构，可以实现 3 级、4 级、6 级流水结构的 CPU 吗?答案肯定是可以的，甚至在各类产品的 CPU 中采用经典 5 级流水结构的都很少。所以希望大家尽情地发挥自己的想法，大改流水结构，验证自己关于流水结构的

想法，您可以实现 3 级（比如：取指、译码、执行）、4 级、6 级等等各类流水结构的 CPU。

5. 课程设计要求大家实现的 mips 指令有限，大家可以分析其余 mips 指令，加以实现。

6. 目前课程设计实现的指令存储器和数据存储器是同步读的机制的，故在当前拍数（时钟周期）发出读数据的地址请求时，在下一拍才能获得读的数据，因此取值级和访存级的 load 都需要两拍时间。其实发地址请求在下一拍获得读的数据，明显也是一个可以流水做的工作，故可以考虑对流水线设计方案作稍微修改，使得取指级和访存级的 load 不需要多等一拍。

5 课程设计拓展要求

拓展题无特别要求，如果想做的话，需要自己花较多心思，在充分理解的基础上加以思考。在确定自己准备实现的想法后，请依然按照以往实验要求，画出设计框图，给出相应设计方案和验证方案。实现完后向老师充分阐述自己的改动，并进行演示。建议最后也形成一份实验报告的文档。文档还是很重要的，要养成一个好习惯。

附录 A 实现的 MIPS 指令集

附 A1 单周期 CPU 实现指令

（1） 无符号加法

ADDU rd, rs, rt　　R 型

31 26	25 21	20 16	15 11	10 6	5 0
000000	rs	rt	rd	00000	100001
6	5	5	5	5	6

GPR[rd] ← GPR[rs] + GPR[rt]

（2） 无符号减法

SUBU rd, rs, rt　　R 型

31 26	25 21	20 16	15 11	10 6	5 0
000000	rs	rt	rd	00000	100011
6	5	5	5	5	6

GPR[rd] ← GPR[rs] - GPR[rt]

（3） 有符号比较，小于置位

SLT rd, rs, rt　　R 型

31 26	25 21	20 16	15 11	10 6	5 0
000000	rs	rt	rd	00000	101010
6	5	5	5	5	6

GPR[rd] ← (sign(GPR[rs]) < sign(GPR[rt]))

（4） 按位与

AND rd, rs, rt　　R 型

31 26	25 21	20 16	15 11	10 6	5 0
000000	rs	rt	rd	00000	100100
6	5	5	5	5	6

GPR[rd] ← GPR[rs] & GPR[rt]

（5） 按位或非

NOR rd, rs, rt　　R 型

31 26	25 21	20 16	15 11	10 6	5 0
000000	rs	rt	rd	00000	100111
6	5	5	5	5	6

GPR[rd] ← ~(GPR[rs] | GPR[rt])

（6） 按位或

OR rd, rs, rt　　R 型

31 26	25 21	20 16	15 11	10 6	5 0
000000	rs	rt	rd	00000	100101
6	5	5	5	5	6

GPR[rd] ← GPR[rs] | GPR[rt]

（7） 按位异或

XOR rd, rs, rt　　R 型

31 26	25 21	20 16	15 11	10 6	5 0
000000	rs	rt	rd	00000	100110
6	5	5	5	5	6

GPR[rd] ← GPR[rs] ^ GPR[rt]

（8） 逻辑左移

SLL rd, rt, shf　　R 型

31 26	25 21	20 16	15 11	10 6	5 0
000000	00000	rt	rd	shf	000000
6	5	5	5	5	6

GPR[rd] ← zero(GPR[rt]) << shf

（9） 逻辑右移

SRL rd, rt, shf　　R 型

31 26	25 21	20 16	15 11	10 6	5 0
000000	00000	rt	rd	shf	000010
6	5	5	5	5	6

GPR[rd] ← zero(GPR[rt]) >> shf

（10） 立即数、无符号加法

ADDIU rt, rs, imm　　I 型

31 26	25 21	20 16	15 0
001001	rs	rt	imm
6	5	5	16

GPR[rt] ← GPR[rs] + sign_ext(imm)

（11） 相等跳转

BEQ rs, rt, offset　　I 型

31 26	25 21	20 16	15 0
000100	rs	rt	offset

6	5	5	16

if GPR[rs] = GPR[rt] then PC ← B_PC + sign_ext(offset)<<2

B_PC：分支跳转参与运算的 PC，在不考虑延迟槽时为分支跳转指令的 PC，考虑延迟槽时为延迟槽指令的 PC，即分支跳转指令的 PC+4。

（12） 不等跳转

BNE rs, rt, offset　　I 型

31 26	25 21	20 16	15 0
000101	rs	rt	offset
6	5	5	16

if GPR[rs] ≠ GPR[rt] then PC ← B_PC + sign_ext(offset)<<2

B_PC：分支跳转参与运算的 PC，在不考虑延迟槽时为分支跳转指令的 PC，考虑延迟槽时为延迟槽指令的 PC，即分支跳转指令的 PC+4。

（13） 装载字

LW rt, offset(base)　　I 型

31 26	25 21	20 16	15 0
100011	base	rt	offset
6	5	5	16

GPR[rt] ← Mem[GPR[base] + sign_ext(offset)]

（14） 存储字

SW rt, offset(base)　　I 型

31 26	25 21	20 16	15 0
101011	base	rt	offset
6	5	5	16

Mem[GPR[base] + sign_ext(offset)] ← GPR[rt]

（15） 立即数装载高位

LUI rt, imm　　I 型

31 26	25 21	20 16	15 0
001111	00000	rt	imm
6	5	5	16

GPR[rt] ← {imm,16’d0}

（16） 直接跳转

J target　　J 型

31 26	25 0
000010	target
6	26

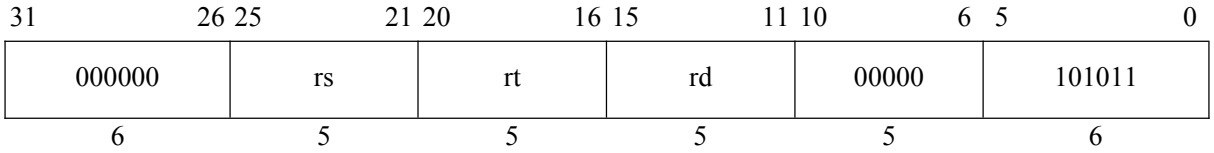
PC ← {B_PC[31:28], target << 2}

B_PC：分支跳转参与运算的 PC，在不考虑延迟槽时为分支跳转指令的 PC，考虑延迟槽时为延迟槽指令的 PC，即分支跳转指令的 PC+4。

附 A2 多周期 CPU 新增实现指令

（17） 无符号小于置位

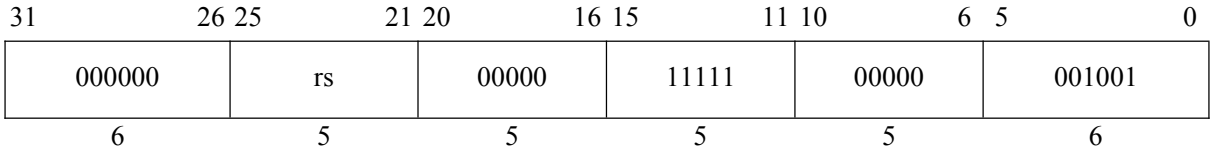
SLTU rd, rs, rt　　R 型



GPR[rd] ← (zero(GPR[rs]) < zero(GPR[rt]))

（18） 跳转寄存器并链接

JALR rs　　R 型

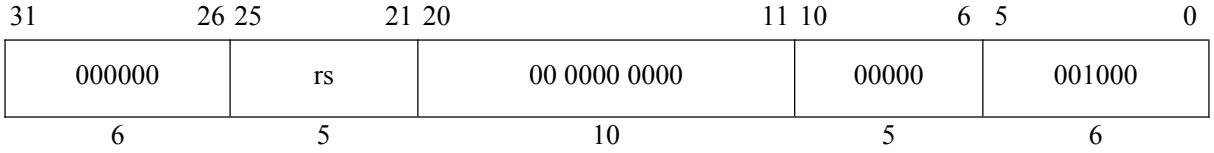


GPR[31] ← B_PC + 4, PC ← GPR[rs]

B_PC：分支跳转参与运算的 PC，在不考虑延迟槽时为分支跳转指令的 PC，考虑延迟槽时为延迟槽指令的 PC，即分支跳转指令的 PC+4。

（19） 跳转寄存器

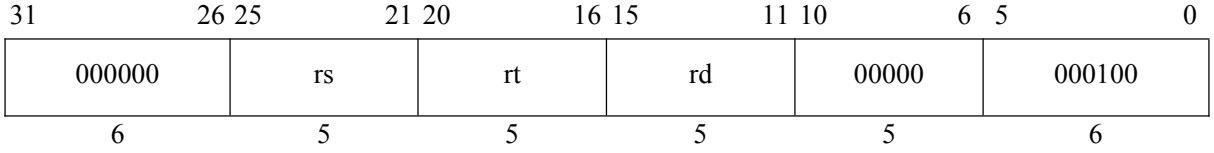
JR rs　　R 型



PC ← GPR[rs]

（20） 变量逻辑左移

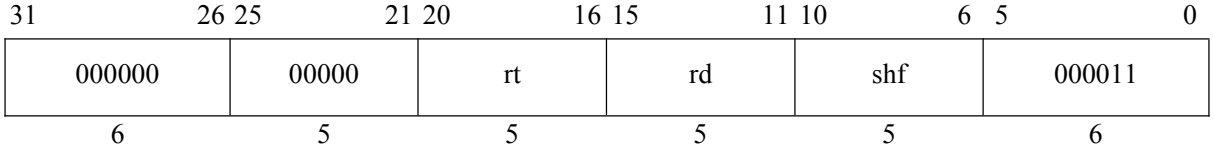
SLLV rd, rt, rs　　R 型



GPR[rd] ← zero(GPR[rt]) << GPR[rs]

（21） 算术右移

SRA rd, rt, shf　　R 型



GPR[rd] ← sign(GPR[rt]) >> shf

（22） 变量算术右移

SRAV rd, rt, rs　　R 型

31 26	25 21	20 16	15 11	10 6	5 0
000000	rs	rt	rd	00000	000111
6	5	5	5	5	6

GPR[rd] ← sign(GPR[rt]) >> GPR[rs]

（23） 变量逻辑右移

SRLV rd, rt, rs　　R 型

31 26	25 21	20 16	15 11	10 6	5 0
000000	rs	rt	rd	00000	000110
6	5	5	5	5	6

GPR[rd] ← zero(GPR[rt]) >> GPR[rs]

（24） 立即数有符号比较，小于置位

SLTI rt, rs, imm　　I 型

31 26	25 21	20 16	15 0
001010	rs	rt	imm
6	5	5	16

GPR[rt] ← (sign(GPR[rs]) < sign_ext(imm))

（25） 立即数无符号比较，小于置位

SLTIU rt, rs, imm　　I 型

31 26	25 21	20 16	15 0
001011	rs	rt	imm
6	5	5	16

GPR[rt] ← (zero(GPR[rs]) < sign_ext(imm))

（26） 大于或等于零跳转

BGEZ rs, offset　　I 型

31 26	25 21	20 16	15 0
000001	rs	00001	offset
6	5	5	16

if GPR[rs]≥0 then PC ← B_PC + sign_ext(offset)<<2

B_PC：分支跳转参与运算的 PC，在不考虑延迟槽时为分支跳转指令的 PC，考虑延迟槽时为延迟槽指令的 PC，即分支跳转指令的 PC+4。

（27） 大于零跳转

BGTZ rs, offset　　I 型

31 26	25 21	20 16	15 0

000111	rs	00000	offset
6	5	5	16

if GPR[rs] > 0 then PC ← B_PC + sign_ext(offset)<<2

B_PC：分支跳转参与运算的 PC，在不考虑延迟槽时为分支跳转指令的 PC，考虑延迟槽时为延迟槽指令的 PC，即分支跳转指令的 PC+4。

（28） 小于或等于零跳转

BLEZ rs, offset　　I 型

31 26	25 21	20 16	15 0
000110	rs	00000	offset
6	5	5	16

if GPR[rs] ≤ 0 then PC ← B_PC + sign_ext(offset)<<2

B_PC：分支跳转参与运算的 PC，在不考虑延迟槽时为分支跳转指令的 PC，考虑延迟槽时为延迟槽指令的 PC，即分支跳转指令的 PC+4。

（29） 小于零跳转

BLTZ rs, offset　　I 型

31 26	25 21	20 16	15 0
000001	rs	00000	offset
6	5	5	16

if GPR[rs] < 0 then PC ← B_PC + sign_ext(offset)<<2

B_PC：分支跳转参与运算的 PC，在不考虑延迟槽时为分支跳转指令的 PC，考虑延迟槽时为延迟槽指令的 PC，即分支跳转指令的 PC+4。

（30） 装载字节，并作符号扩展

LB rt, offset(base)　　I 型

31 26	25 21	20 16	15 0
100000	base	rt	offset
6	5	5	16

GPR[rt] ← sign(Mem[GPR[base] + sign_ext(offset)])

（31） 装载字节，并作无符号扩展

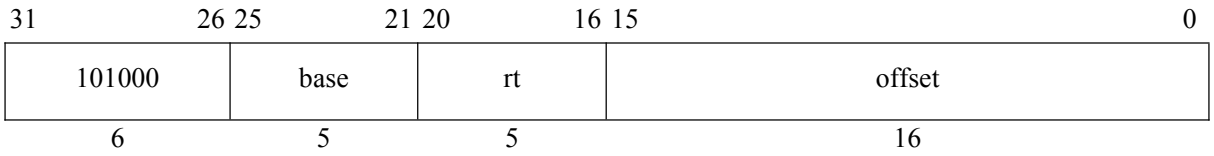
LBU rt, offset(base)　　I 型

31 26	25 21	20 16	15 0
100100	base	rt	offset
6	5	5	16

GPR[rt] ← zero(Mem[GPR[base] + sign_ext(offset)])

（32） 存储字节

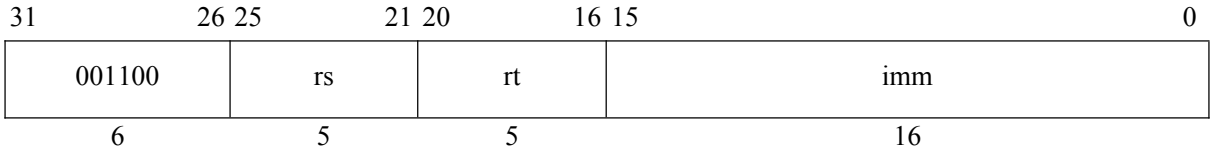
SB rt, offset(base)　　I 型



Mem[GPR[base] + sign_ext(offset)] ← GPR[rt]

（33） 立即数按位与

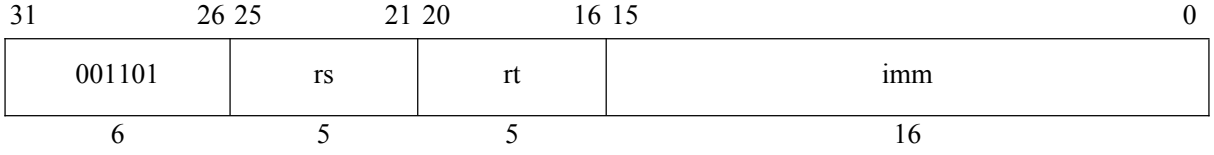
ANDI rt, rs, imm　　I 型



GPR[rt] ← GPR[rs] & zero_ext(imm)

（34） 立即数按位或

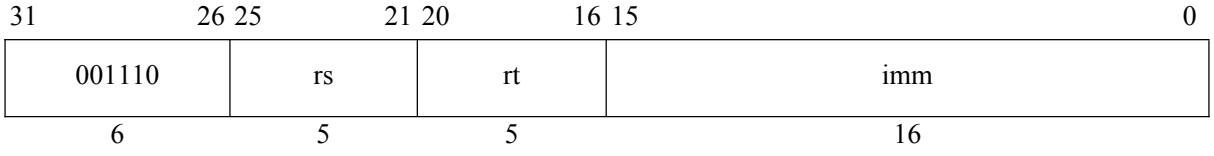
ORI rt, rs, imm　　I 型



GPR[rt] ← GPR[rs] | zero_ext(imm)

（35） 立即数按位异或

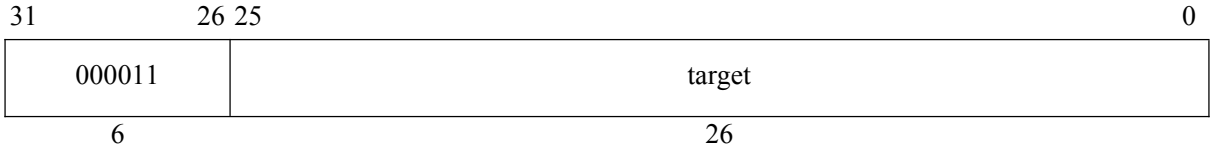
XORI rt, rs, imm　　I 型



GPR[rt] ← GPR[rs] ^ zero_ext(imm)

（36） 跳转和链接

JAL target　　J 型



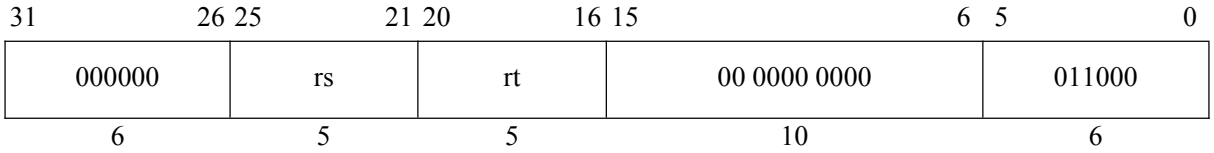
GPR[31] ← B_PC + 4, PC←{B_PC[31:28], target << 2}

B_PC：分支跳转参与运算的 PC，在不考虑延迟槽时为分支跳转指令的 PC，考虑延迟槽时为延迟槽指令的 PC，即分支跳转指令的 PC+4。

附 A3 静态 5 级流水 CPU 新增实现指令

（37） 有符号字乘法

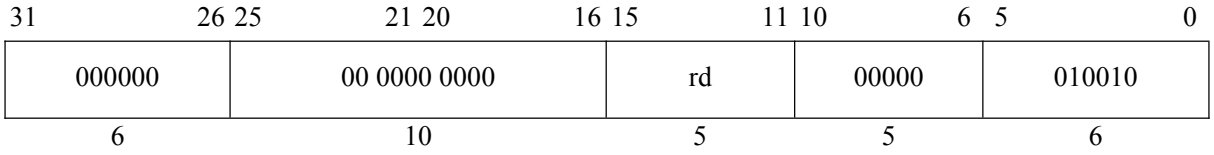
MULT rs, rt　　R 型



(HI, LO) ← sign(GPR[rs]) * sign(GPR[rt])

（38） 从 LO 寄存器取值

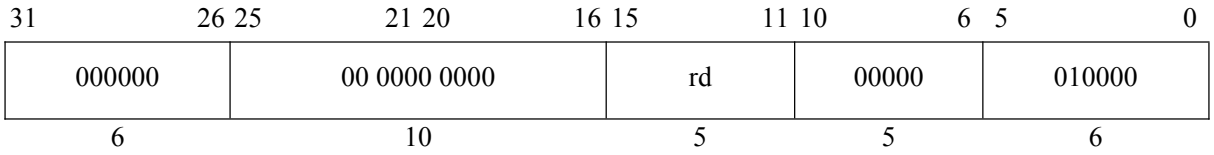
MFLO rd　　R 型



GPR[rd] ← [LO]

（39） 从 HI 寄存器取值

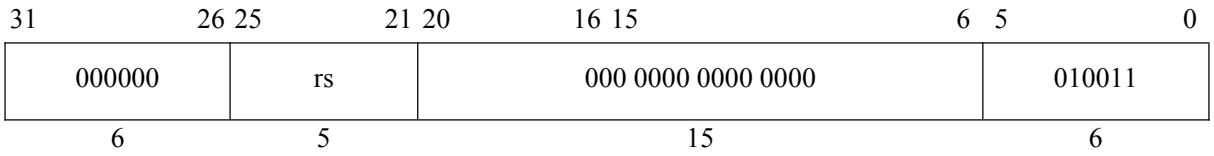
MFHI rd　　R 型



GPR[rd] ← [HI]

（40） 向 LO 寄存器存值

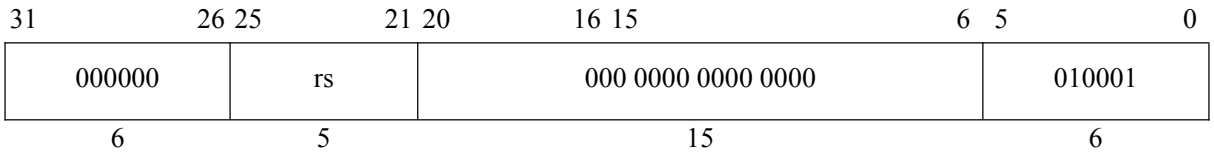
MTLO rs　　R 型



[LO] ← GPR[rs]

（41） 向 HI 寄存器存值

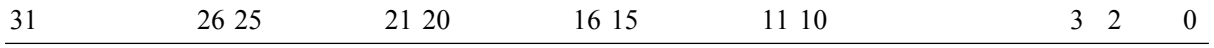
MTHI rs　　R 型



[HI] ← GPR[rs]

（42） 从协处理器 0 寄存器取值

MFC0 rt, cs.sel　　R 型



010000	00000	rt	cs	00000000	sel
6	5	5	5	8	3

GPR[rt] ← CPR[cs.sel]

（43） 向协处理器 0 寄存器存值

MTC0 rt, cd.sel　　R 型

31 26	25 21	20 16	15 11	10 3	2 0
010000	00100	rt	cd	00000000	sel
6	5	5	5	8	3

CPR[cd.sel] ← GPR[rt]

（44） 系统调用

SYSCALL

31 26	25 6	5 0
000000	code	001100
6	20	6

CPR[14.0] ← PC, CPR[13.0][6:2] ← 01000, CPR[12.0][1] ← 1, PC ←EXC_ENTER_ADDR

EXC_ENTER_ADDR 为例外入口地址，原本应为 CPR[15.1] + 0x180，但在课程设计中为方便编写测试程序，将 EXC_ENTER_ADDR 设置为 0。

（45） 异常返回

ERET

31 26	25	21 20 16 15 11 10 6	5 0
010000	1	000 0000 0000 0000 0000	011000
6	1	19	6

CPR[12.0][1] ← 0, PC ← CPR[14.0]

附录 B 实现的 cp0 寄存器

附 B1 Status 寄存器（CP0 寄存器 12，选择 0）

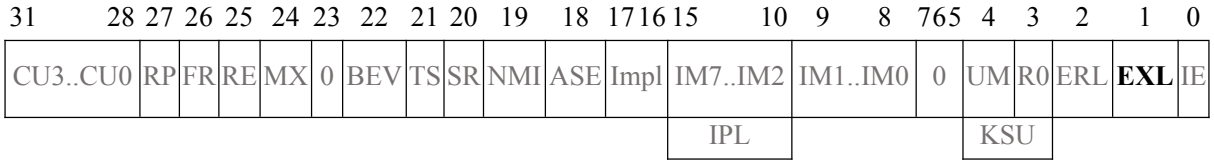


表 B-1 EXL（Exception Level）域

域		描述	读/写	复位状态	规则
名称	位				
EXL	1	Exception 级别；当出现任何除了复位、软复位、NMI 或缓存错误的例外时由处理器置位。	可读/写	Undefined	Required

表 B-2 EXL 编码表

编码	含义
0	正常级别
1	Exception 级别

附 B2 Cause 寄存器（CP0 寄存器 13，选择 0）

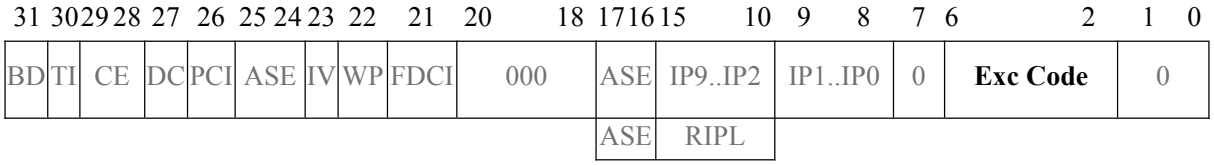


表 B-3 例外编码表

例外编码值		助记符	描述
十进制	十六进制		
0	0x00	Int	中断
1	0x01	Mod	TLB 修正例外
2	0x02	TLBL	TLB 例外（装载或取指）
3	0x03	TLBS	TLB 例外（存储）
4	0x04	AdEL	地址错误例外（装载或取指）
5	0x05	AdES	地址错误例外（存储）
6	0x06	IBE	总线错误例外（取指）
7	0x07	DBE	总线错误例外（数据相关：装载或存储）
8	0x08	Sys	系统调用例外
9	0x09	Bp	断点例外
10	0x0a	RI	保留指令例外

表 B-4 Exc Code（Exception Code）域

域		描述	读/写	复位状态	规则
名称	位				
ExcCode	6..2	例外编码	只读	Undefined	Required

附 B3 例外程序计数器 EPC（CP0 寄存器 14，选择 0）

31 0

EPC

表 B-5 EPC（Exception Program Counter）域

域		描述	读/写	复位状态	规则
Name	**Bits**				
EPC	31..0	例外程序计数器	可读/写	Undefined	Required