实验 1：数据运算定点加法

1.1 实验目的

1. 通过 Verilog HDL 方式实现加法器程序设计，通过 Vivado 软件进行开发调试，实现定点加法功能。
2. 为后续 CPU 设计的实验打下基础。

1.2 实验设备

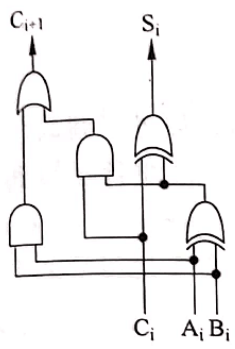
1. 装有 Xilinx Vivado 的计算机一台。

1.3 实验要求

1. 学习 Vivado 软件平台和设计流程。
2. 熟悉计算机加法器的原理。
3. 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。

1.4 实验原理

全加器的原理图、逻辑表达式及对应的真值表如图 3-1-2 所示，定点加法在全加器基础上构建。



(a) 原理图

$S_i=A_i\oplus B_i\oplus C_i$

$C_{i+1}=A_iB_i+A_iC_i+B_iC_i$

$=A_iB_i+(A_i\oplus B_i)C_i$

(b) 逻辑表达式

输入			输出	
A_i	B_i	C_i	S_i	C_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(c) 真值表

图 3-1-2 一位全加器

1.5 实验步骤

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。

1. 创建工程

启动 Vivado，在菜单栏点击“File”->“New Project”，根据新建工程向导，输入工程名称，选择工程的文件位置，如图 3-1-4 所示。注意路径中不要使用中文字符，且路径长度不要超过 260 字节。工程路径默认勾选“Create project subdirectory”用于创建工程子目录，勾

选后会创建一个文件名与工程名形同的文件夹，用于存放整个工程。如果选择不勾选，将在当前文件夹下创建工程。

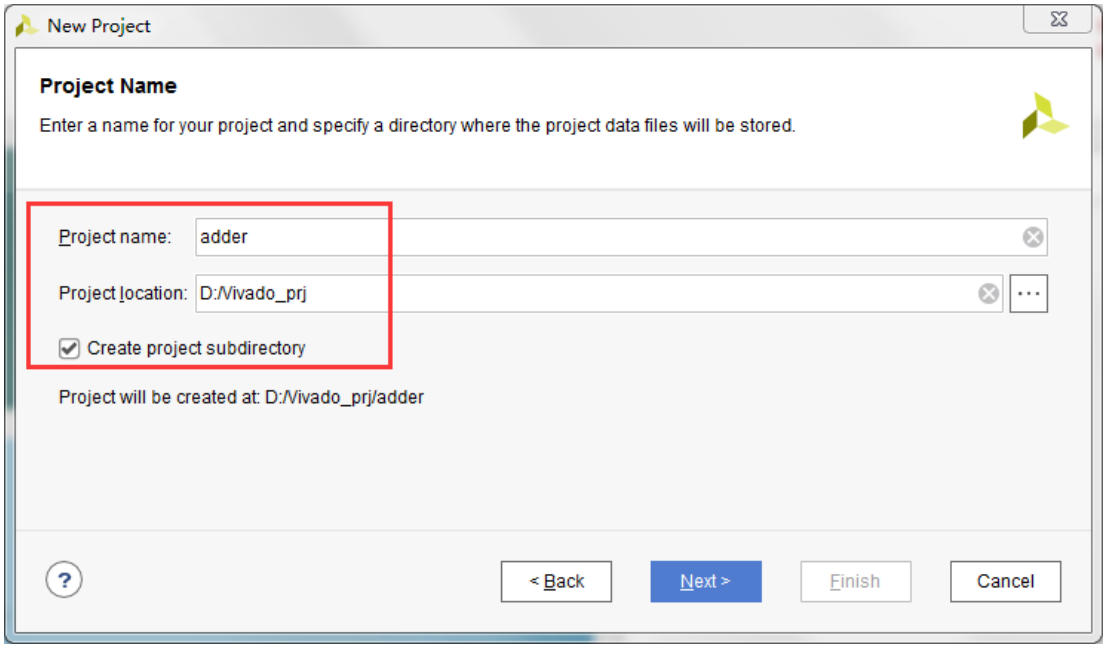


图 3-1-4 设置工程名称和位置

在工程类型“Project Type”选择界面，选择“RTL Project”，勾选“Do not specify sources at this time”。在“Default Part”FPGA 器件选择界面，指定所用的 FPGA 器件或开发板。“Parts”指 FPGA 芯片，包含了 Vivado2019.2 所支持的芯片型号；“Boards”指的是 Vivado2019.2 所支持的开发板。这里选“Parts”，然后在筛选器的“Family”中选择“Artix-7”，在“Package”中选择“fbg676”，在筛选得到的型号里面选择“xc7a200tfbg676-2”，点击“Next”，根据向导，单击“Finish”按钮即完成整个工程的创建。

2. 添加设计文件

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”，根据向导，点击“Add Files”，选择“adder.v”，点击“OK”，再点击“Finish”按钮即完成设计源文件的添加。

adder.v 的代码如下:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//    > 文件名: adder.v
//    > 描述   : 加法器
//    > 作者   : LOONGSON
//    > 日期   : 2016-04-14
//////////////////////////////////////////////////////////////////////////////////
module adder(
    input  [31:0] operand1,
    input  [31:0] operand2,
    input         cin,
    output [31:0] result,
```

```
    output              cout
    );
    assign {cout,result} = operand1 + operand2 + cin;
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，则会由多个.v 文件，形成一定的调用层次。

3. 添加展示外围模块

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以便在板上完成实验，本实验中将该模块命名为 adder_display.v，添加该设计源文件到工程里，结果如图 3-1-5 所示。

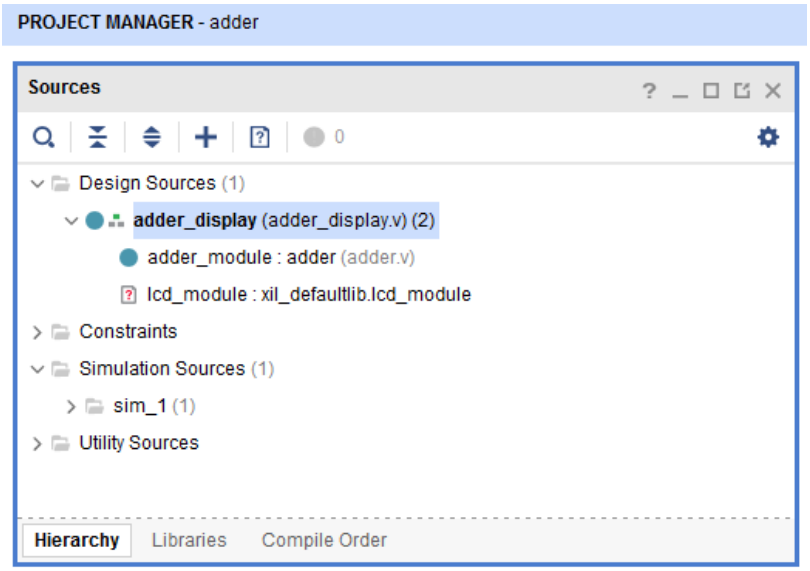


图 3-1-5 触摸屏模块工程管理区

在工程管理区可以看到各模块间的层次关系，本实验中顶层模块即为 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 触摸屏模块。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 3-1-5 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，只需要调用即可。继续选择“Add Sources”， 选择 “Add or create design sources”，添加 lcd_module.dcp。添加成功后，结果如图 3-1-6 所示。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

为了及时快捷地验证所设计模块的正确性，通常在下板之前先要对系统进行仿真。尤其是功能仿真，是计算机组成原理实验中经常用到的手段。

在进行仿真时，通常的做法是设计一个测试文件，通过该文件产生模块所需要的激励输入信号，并观察模块的输出。该测试文件就是 testbench（测试平台）。

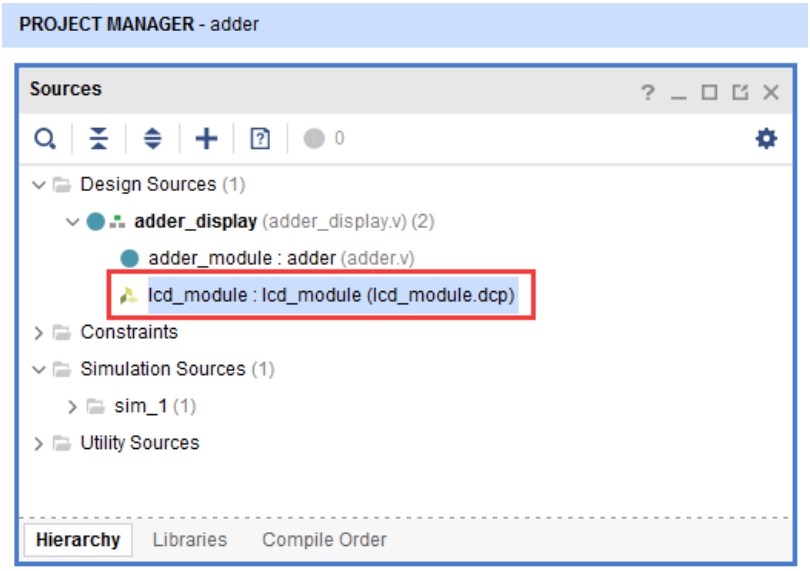


图 3-1-6 添加 lcd_module 文件成功

编写 testbench 的目的就是为待仿真模块提供输入并获取输出。在本实验中，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后，会输出加法结果和向高位的进位信号。

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

testbench.v 的代码如下:

```
`timescale 1ns / 1ps    //仿真单位时间为 1ns，精度为 1ps
module testbench;

    // Inputs
    reg [31:0] operand1;
    reg [31:0] operand2;
    reg cin;

    // Outputs
    wire [31:0] result;
    wire cout;
    // Instantiate the Unit Under Test (UUT)
    adder uut (
        .operand1(operand1),
        .operand2(operand2),
        .cin(cin),
        .result(result),
        .cout(cout)
    );
    initial begin
        // Initialize Inputs
        operand1 = 0;
        operand2 = 0;
```

```
            cin = 0;
            // Wait 100 ns for global reset to finish
            #100;
            // Add stimulus here
        end
        always #10 operand1 = $random;   //$random 为系统任务，产生一个随机的 32 位数
        always #10 operand2 = $random;   //#10 表示等待 10 个单位时间(10ns),即每过 10ns,
赋值一个随机的 32 位数
        always #10 cin = {$random} % 2; //加了拼接符，{$random}产生一个非负数，除 2 取
余得到 0 或 1
endmodule
```

添加 testbench.v 后，工程管理区如图 3-1-7 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”。

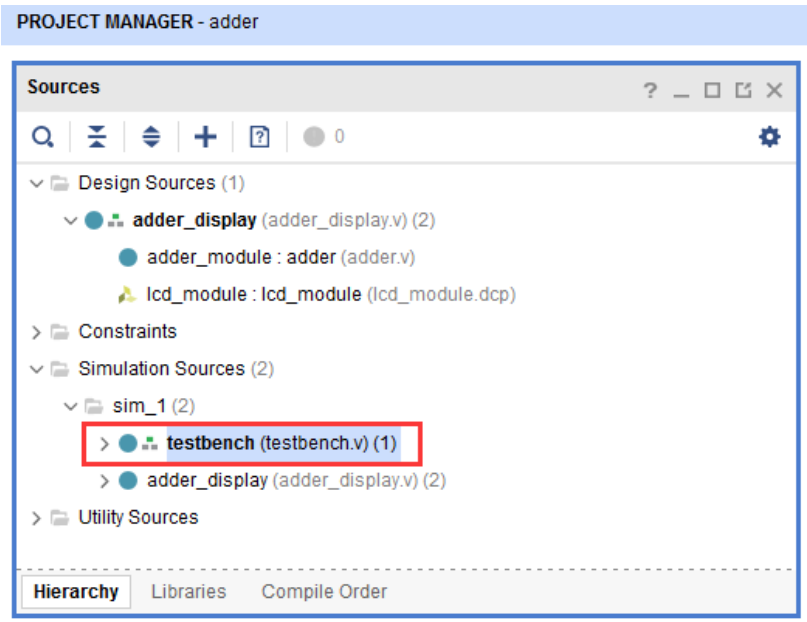


图 3-1-7 添加 testbench.v 成功

添加 testbench.v 成功后，在左侧的导航栏中点击“Run Simulation”，选择“Run Behavioral Simulation”可进行仿真验证。如果没有语法错误，弹出界面如图 3-1-8 所示。

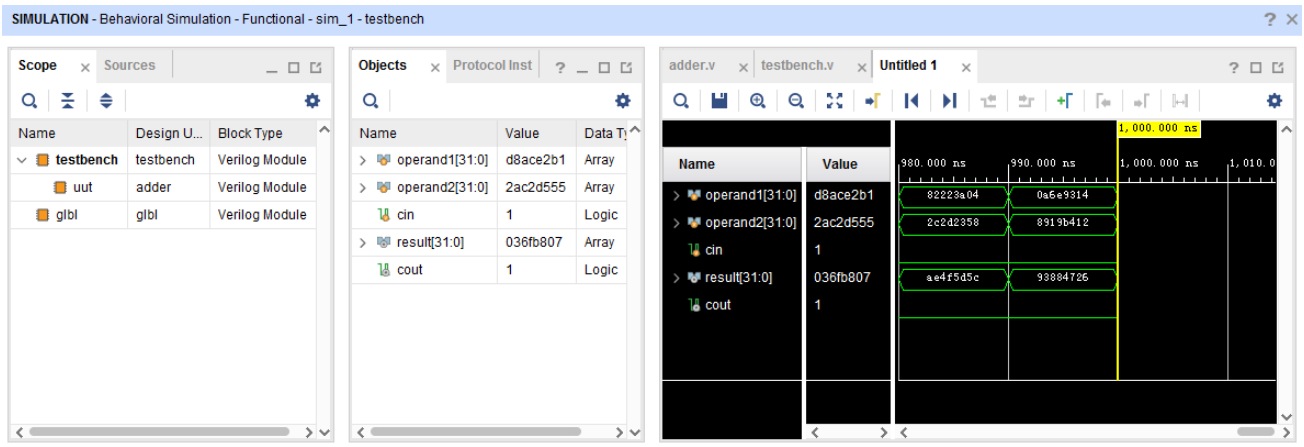


图 3-1-8 定点加法仿真界面

在仿真界面左侧的“Scope”面板选择要仿真的模块，选择后仿真的输入/输出便会出现在“Objects”面板中，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”添加需要观察的信号到仿真窗口。仿真波形的观察可以通过窗口的推拉进行缩放。在波形窗口的 Name 列右键选择信号，然后会出现很多可选项，选择“Radix”，然后选择数据显示的进制，如 16 进制“Hexadecimal”。

可以检查几组数据，比如：0b940917+d0f578a1+0=dc8981b8，正确。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，如图 3-1-9 所示。

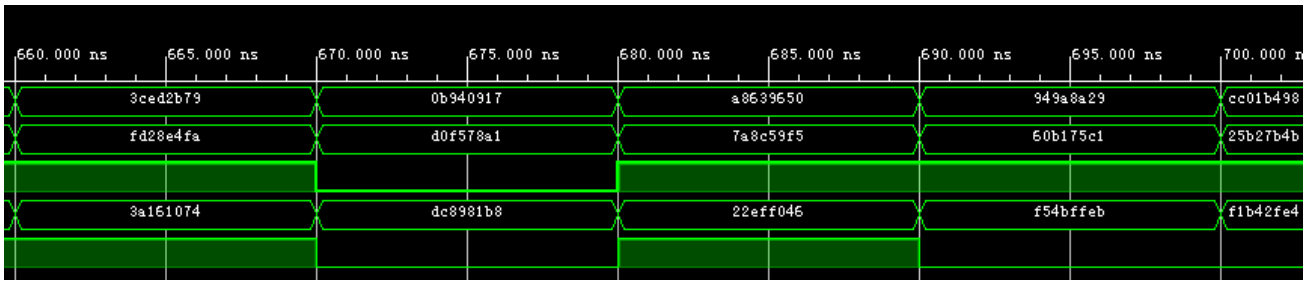


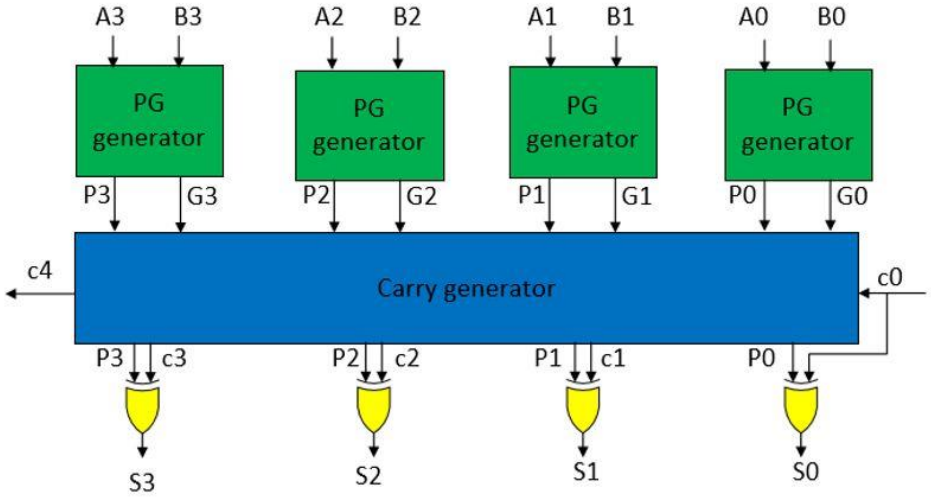
图 3-1-9 定点加法仿真数据以 16 进制形式显示

至此，代码编辑和功能仿真都已完成，认为功能基本正确。

在运算中，加法的效率和占用的资源一直是比较重要的因素，行波进位加法器在课本的 2.2.4 节有详细介绍，N-bit 加法器可以根据 1-bit 全加器组合而成。每个全加器的输出进位 cout 作为下一个全加器的输入进位 cin，行波进位加法器设计简单，只需要级联全加器即可，但它的缺点在于超长的进位链，限制了加法器的性能。课本 2.5.2 节还介绍了先行进位加法器(CLA)，请仔细阅读课本并设计一个 CLA 加法器。

1.6 请设计一个 4bit 先行进位加法器（Carry-lookahead adder，CLA）有关 CLA 的原理解释请参考课本或者阅读文献[1]：

超前进位加法器优化改进行波进位器的关键路径，通过采用并行计算进位的方法，解决了行波进位加法器的进位依赖问题。



[1] 简书. 加法器的优化——超前进位加法器（Carry-Lookahead Adder，CLA）（2017-07-27）：https://www.jianshu.com/p/6ce9cad8b467

1.7 请利用该 4bit CLA 组合设计一个 16bit 加法器

1.8 实验报告

1. 实验目的
2. 实验配置
3. 实验内容
   A. 模块代码（包括必要的注释）
   B. 完成仿真，并将仿真波形图像截图
4. 讨论与总结