附录 7 实验环境使用方法

1 下发资料介绍

在“计算机组成原理课程设计”课程下发的资料中，包含两个目录：“Appendix”目录存放了一些文档，作为对指导书内容的扩充和增补，供大家参考；而“CDE”目录则存放了课程实验所需的全部内容，如果需要快速开始实验，我们只需关注“CDE”目录的内容即可。

“CDE”实际上是“CPU Development Environment”的缩写，即“CPU 开发环境”，包括 TinyMIPS 处理器的源代码、功能测试环境的 Vivado 工程，以及运行于处理器上的功能测试程序源码等内容。我们使用这套环境，再配合下发的实验箱，可以方便地为 TinyMIPS 处理器，或者是我们自行实现的处理器进行功能测试的仿真和上板。

关于“CDE”目录的详细介绍，同学们可以参考《指导书上册》中第 3.3 节，“CPU 实验开发环境”部分，此处仅作简单介绍：

图 1-1 CDE 目录的内容

CDE 目录的内容如图 1-1 所示，其中：

- “cpu”目录：存放 TinyMIPS 处理器的完整实现；
- “cpu132_gettrace”目录：存放用于获取 trace 的 Vivado 工程，下文会详细说明；
- “soc_axi_func”目录：针对 AXI 接口 CPU 设计的功能测试 Vivado 工程。由于我们的 TinyMIPS 使用的是 AXI 总线接口，所以我们需要使用此目录来进行实验；
- “soc_sram_func”目录：针对 SRAM 接口 CPU 设计的功能测试 Vivado 工程。如果大家自行实现了 SRAM 接口的 CPU，可以使用此目录中的文件来完成功能测试；
- “soft”目录：存放功能测试用到的程序和源代码。

2 什么是功能测试

顾名思义，所谓“功能测试”，实际上就是一种针对被测对象的功能实现的测试。在实际的 CPU 开发流程中，为了确保我们实现的 CPU 在功能上尽可能符合预期，我们需要对已经实现完成的 CPU 进行功能测试，以期检测出 CPU 在功能上存在的问题。与之相对的是“性能测试”，即针对 CPU 性能进行的测试。不同 CPU 在同一性能测试下的结果可以大致反映这些 CPU 之间的性能高低。大家熟悉的各种跑分软件实际上就是某种性能测试，例如移动端的 Geekbench、安兔兔，PC 端的 Cinebench R20、3DMark 等。

简单来说，CPU 是一种负责解释执行机器指令的硬件。测试 CPU 的功能，实际上就是针对每一种机器指令编写一段测试程序，在程序中尽可能多地模拟指令的各种执行情况，然后让 CPU 执行所有的测试程序，最后检查执行结果的正确性。这实际上就引出了两个问题：如何模拟指令的各种执行情况？以及，如何有效的检测结果的正确性？

第一个问题其实比较好解决。大家在大三下学期会学习一门课程——“软件工程及课程设计”。在这门课程中，老师会为我们讲授许多在软件开发中很有用的方法论和工程实践，其中就包括了如何进行有效的软件测试。这些软件测试的方法在硬件上同样适用：我们可以合理运用“黑盒测试”的思想，从每条指令的功能出发设计测试用例程序。

例如“LB”指令要求 CPU 将指令中的立即数做符号扩展，和 base 寄存器的值相加得到虚拟地址，然后经过地址变换机构得到物理地址，向总线发起访存请求得到内存中对应字节的数据，再将数据做符号扩展放入目的寄存器。我们的测试用例就可以针对每个功能要求构造一些特殊值，例如：分别测试立即数最高位为 0 和为 1 的情况，检测 CPU 是否正确完成了立即数的符号扩展；分别测试不同 base 寄存器的情况，检测 CPU 读取寄存器时是否正常；分别构造不同地址段的虚拟地址，检测地址变换是否正常……等等。

第二个问题解决起来也不是那么复杂。我们知道，CPU 所执行的每一条指令，基本上都会转化为对寄存器做的操作。无论是做算术运算、从内存读取数据，还是执行跳转（如果把 PC 也算作一个寄存器的话）。最开始 CPU 寄存器堆中所有寄存器的值全为 0，随着指令的执行，CPU 向若干不同的寄存器内写入了各种各样的值。理想情况下，CPU 在每个周期都会执行一条指令，这些指令依赖上一个周期时某些寄存器的值，然后在当前周期结束后又会修改另一个寄存器的值。我们完全可以把寄存器堆和 PC 寄存器看作是 CPU 的内部状态，每执行一条指令，CPU 的状态就会从上一周期更新到下一个周期——这个概念可能比较抽象，但事实上的确如此。

既然是这样，我们完全可以通过记录 CPU 中寄存器堆和 PC 的变动，来检测 CPU 状态的变化，进而得到 CPU 执行指令的结果序列。为了验证结果序列的正确性，我们可以借助另一个在功能实现上已经基本无误的 CPU，让它执行同样的程序，再用相同的方法得到结果序列。如果我们 CPU 的实现是正确的，那么我们的结果序列应该和正确结果序列完全一致——这种测试的思想叫做“差分测试”，CDE 中的功能测试程序就采用了这样的思想。

功能测试的实际过程如下：

1. 使用工具链编译“soft”目录中的功能测试程序，得到需要 CPU 执行的二进制机器指令序列；
2. 使用“cpu132_gettrace”目录中的 Vivado 工程获得基准结果序列，即 trace。这个目录中的 Vivado 工程会使用龙芯公司开源的 GS132 处理器核运行编译好的功能测试程序，然后抓取执行结果，保存在“golden_trace.txt”中；
3. 使用“soc_axi_func”目录中的 Vivado 工程导入被测 CPU，然后运行仿真。该工程同样会使用被测 CPU 执行功能测试程序，只不过会将执行结果和 trace 中的结果进行逐条比对。如果不符，直接停止测试并报错。

详情可以参考《指导书上册》第 3.1 节“CPU 实验开发环境快速上手”。我们为大家提供的环境中已经完成了第 1、2 步，接下来的文档将指导大家进行第 3 步，来实现对 TinyMIPS 处理器的功能测试。

3 准备工作

在开始之前，你首先要确认：

1. Vivado 版本必须为 2018.3；
2. “CDE”目录所在的路径中不能包含中文字符，否则 Vivado 在仿真和综合实现时会报错。

接下来进入“soc_axi_func/run_vivado/mycpu_prj1”目录，双击“mycpu.xpr”来启动 Vivado。加载完毕后如图 3-1，我们可以注意到左侧“Sources”窗格内有一个被标记为问号的模块，这是因为我们还没有导入 CPU。

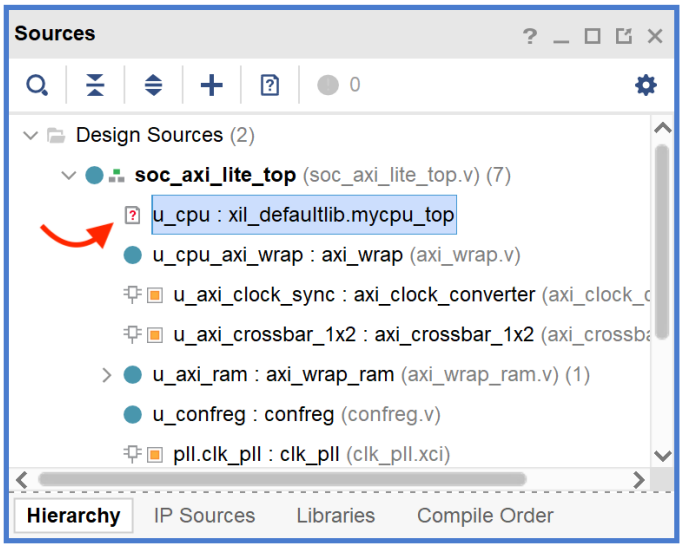

图 3-1 未导入 CPU 时的功能测试工程结构

点击“Sources”窗格中的加号图标，在弹出的窗口中选择第二项，如图 3-2。

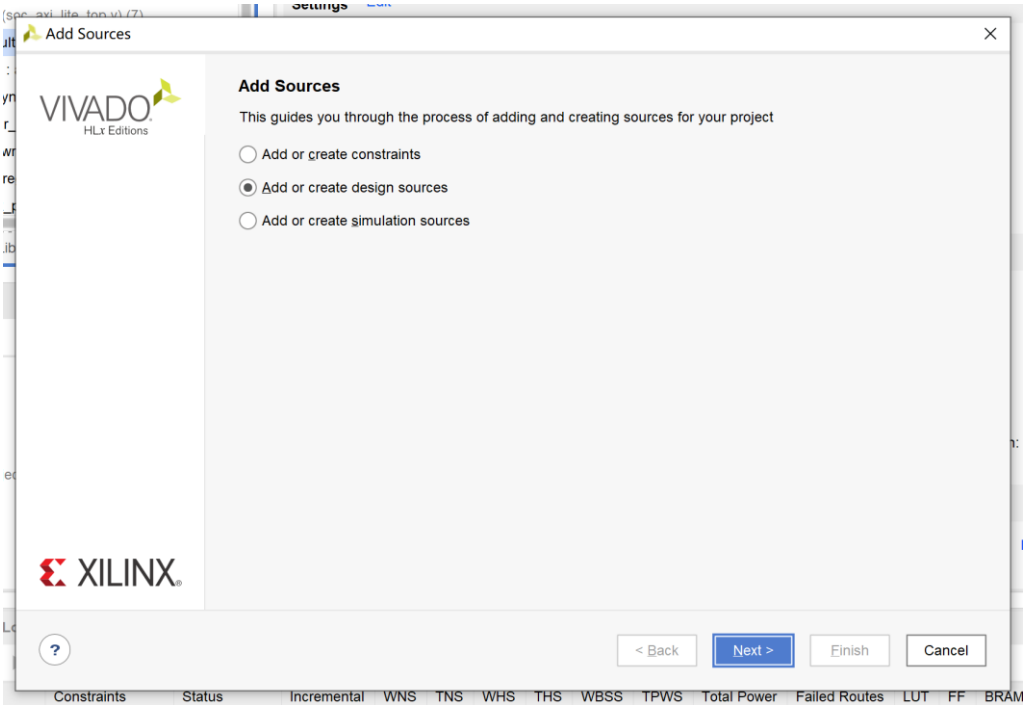

图 3-2 导入 design sources

点击下一步，然后点击“Add Directories”按钮，在弹出的窗口中选择 CDE 中的“cpu”目录，导入 TinyMIPS 处理器的实现，如图 3-3。

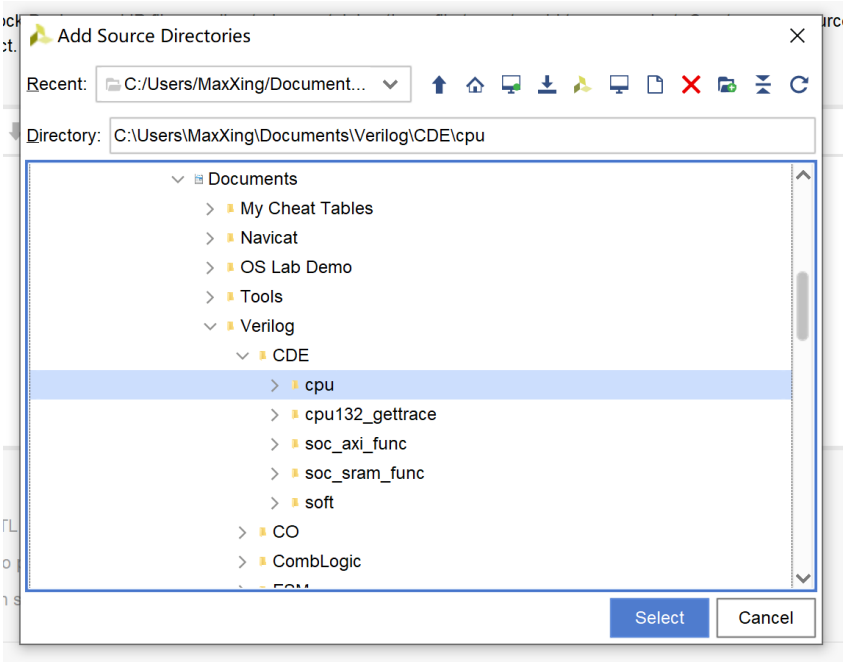

图 3-3 选中“cpu”目录

选择之后，我们还需要选中下方复选框中的第一项和第三项，如图 3-4。

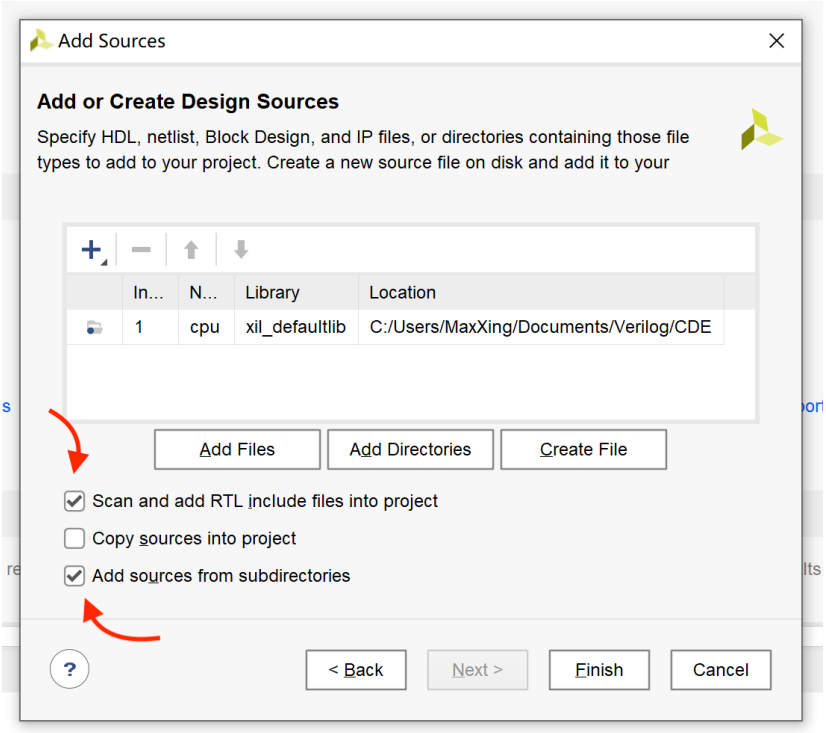

图 3-4 选择第一项和第三项

此时点击完成，稍等片刻，Vivado 就会自动导入所需的文件。但是我们很快发现，Vivado 将其中的 15 个文件都标记为了语法错误，如图 3-5。这是为什么呢？

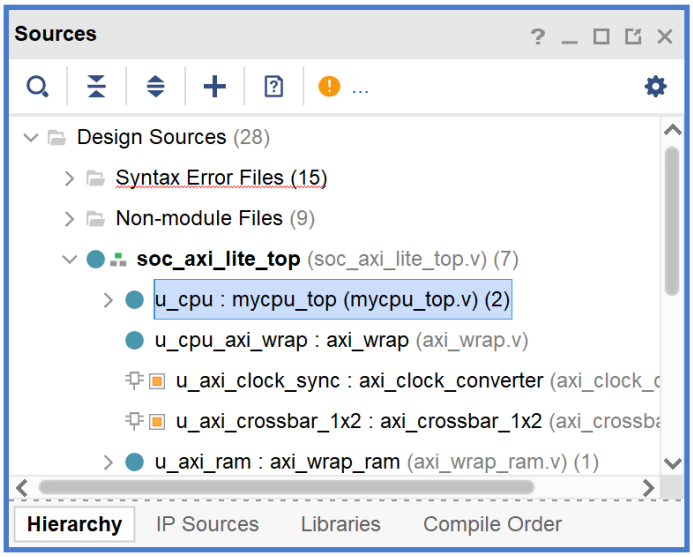

图 3-5 Vivado 检测到语法错误

观察出现语法错误的文件我们不难发现，这些文件都使用了“include”语句来引用其他 Verilog 源文件。Verilog 中，“include”的用法和 C/C++ 中基本一致，我们可以通过类似的方法引用一个已经存在的头文件。而在 TinyMIPS 中，为了提高代码的可读性，增加代码的可维护性，几乎所有的常量定义都写在了头文件内，包括总线宽度、ALU 操作码等。

如果你曾经在命令行中直接调用编译器编译过 C/C++ 项目，你就会知道：编译器在编译过程中能找到我们自行编写的头文件的位置，是因为我们手动为其指定了“-I”参数。编译器遇到“include”语句时，会在“-I”参数之后跟随的目录中查找对应的头文件，否则同样会报错。这一切在 Verilog 中同理，所以我们需要在 Vivado 的设置中指定头文件的搜索路径。

在菜单栏中找到“Tools – Settings… – Project Settings – General – Verilog options”，点击右侧的“…”，将 CDE 目录下的“cpu/include”目录加入搜索路径中。如图 3-6 所示。确认后我们发现所有的语法错误提示都消失了，但是 Vivado 依然会显示黄色的惊叹号表示警告。这是因为 Vivado 虽然能够正确找到头文件，但是它还没有意识到工程中已经添加的一部分文件就是头文件。

我们需要在“Sources”窗格中选中“Verilog”分类下的所有文件，如图 3-7。然后点击右键，选择“Set Global Include”，将其设置为全局的头文件，完成后如图 3-8。

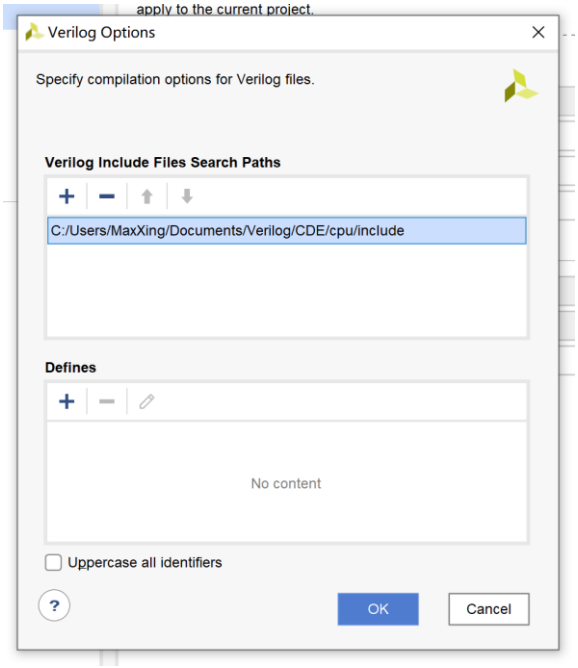

图 3-6 添加头文件搜索路径

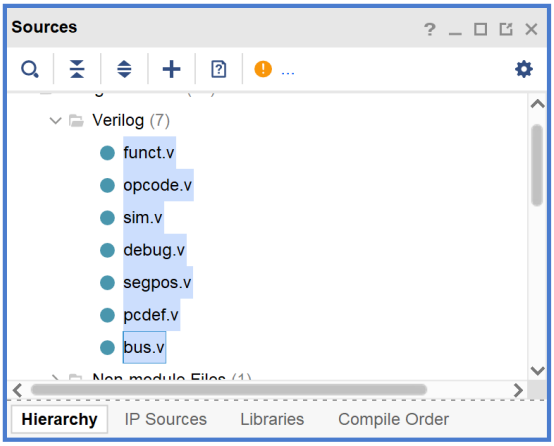

图 3-7 选择“Verilog”分类下的所有文件

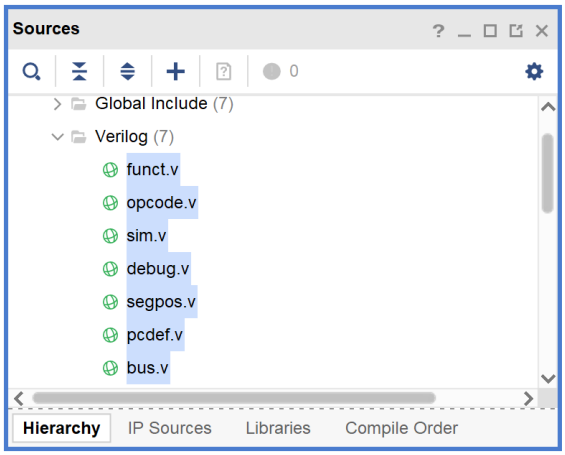

图 3-8 设为 Global Include

此时 Vivado 就不会提示任何错误或警告了，TinyMIPS 的导入工作就此完成。

4 功能测试仿真

导入 CPU 之后，点击 Vivado 界面左侧的“Run Simulation – Run Behavioral Simulation”即可进入仿真模式，如图 4-1。

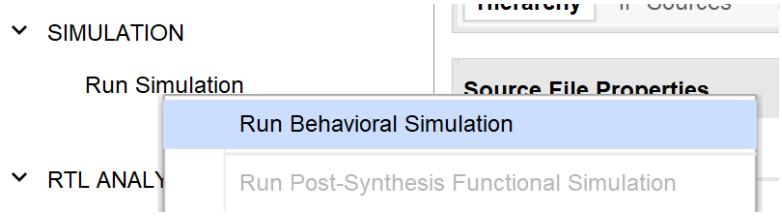

图 4-1 运行行为仿真

需要注意，在初次仿真时，Vivado 必须生成项目中所有的 IP 核，这步操作可能会执行很长时间。由于计算机之间存在性能差异，该操作可能需要花费 5~15 分钟。

完成之后，Vivado 会进入仿真界面，此时我们只需要关注“Tcl Console”中的内容，而非波形的内容。我们可以将下方的“Tcl Console”窗格最大化，然后点击工具栏中的蓝色三角形按钮即可开始运行仿真，如图 4-2。

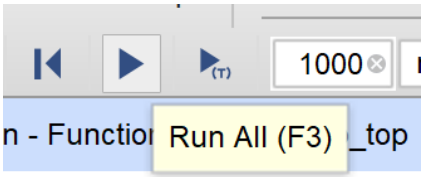

图 4-2 点击按钮运行仿真

运行仿真时，每隔 10000ns，“Tcl Console”会打印调试信息。如果中途通过了某个功能测试点，控制台中同样会输出对应的信息。所有测试结束并通过后，控制台会输出“Test end!”和“PASS!!!”。详细信息如图 4-3。

在 CDE 环境的功能测试程序中，我们一共设置了 26 个功能测试点。因为 TinyMIPS 处理器在设计时考虑到简化实现难度，只实现了 MIPS 指令集当中最重要的 22 种指令，每种指令设置 1 个功能测试点，就会有 22 个测试点。而除此之外，MIPS 指令集规定了分支和跳转指令需要具备“延迟槽”的特性（详情请自行查阅相关资料），这 22 种指令中有 4 种分支跳转指令，于是必须设置额外 4 个针对延迟槽特性的测试点。所有的测试点总和为 26 个。在仿真测试结束后，你应当看到 26 条测试点通过的信息，否则表示测试出错。

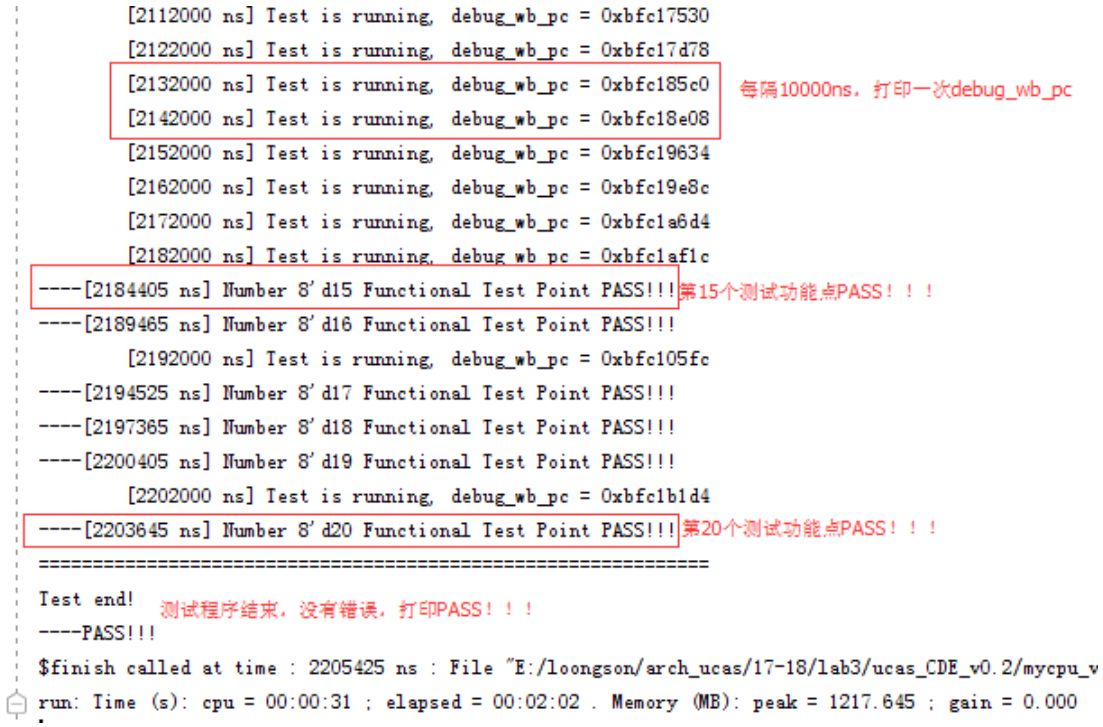

图 4-3 仿真时的 Tcl Console 输出（仅作示例）

关于功能测试仿真的其他详细内容，可以参考《指导书上册》第 4.10.9 节“CPU 验证结果”的第（1）部分。

5 功能测试上板

在功能测试仿真通过后，我们就可以对这个 Vivado 工程进行综合和实现了，最终我们可以得到一个用于给 FPGA 编程的 bitstream 文件。

点击 Vivado 界面左侧的“Generate Bitstream”即可直接生成 bitstream，如图 5-1 所示。由于我们之前并未进行过综合和实现，Vivado 会提示我们先进行综合实现的流程，我们直接确认即可。除此之外，我们并不需要对整个工程进行任何其他额外的设置，因为诸如 IO 分配、时序约束等操作已经预先完成了。

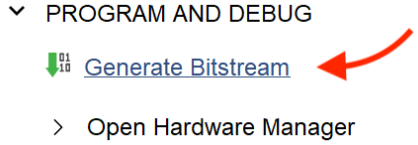

图 5-1 生成 bitstream

生成完成后我们点击左侧的“Open Hardware Manager”（同样在图 5-1 中可见），或者在

弹出的对话框中选择相同选项，进入“Hardware Manager”界面。此时我们需要打开实验箱，将开发板连接电源，然后将 USB 下载线连接电脑和下载器（在开发板下方），并打开开发板的电源开关。接着点击界面上的“Auto Connect”按钮，让 Vivado 自动连接 FPGA 设备，如图 5-2。

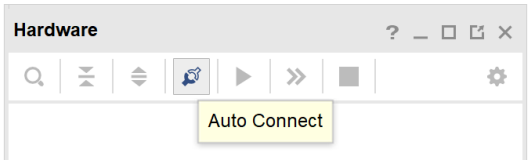

图 5-2 自动连接 FPGA

成功连接之后，界面上将会显示 FPGA 设备。我们可以点击“Program device”来对板子编程，如图 5-3。在编程之前，需要确保板子上的全部 8 个拨码开关（位于板子下方）都拨下，具体原因之后再进行解释。拨码开关的位置如图 5-4。

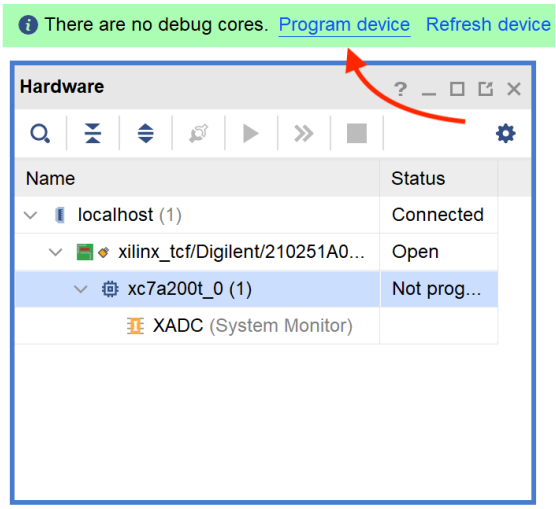

图 5-3 为 FPGA 编程

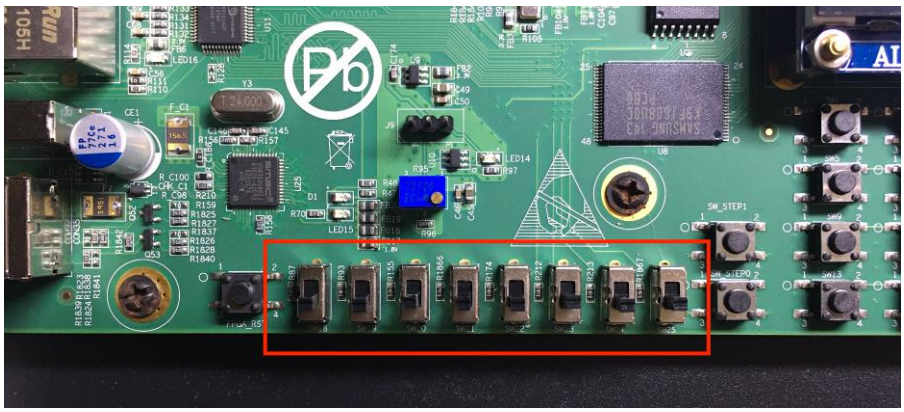

图 5-4 拨码开关的位置

编程完毕后，板子的情况应该如图 5-5：数码管显示“1A 00 00 1A”，左上角的两个 LED

灯亮绿色。

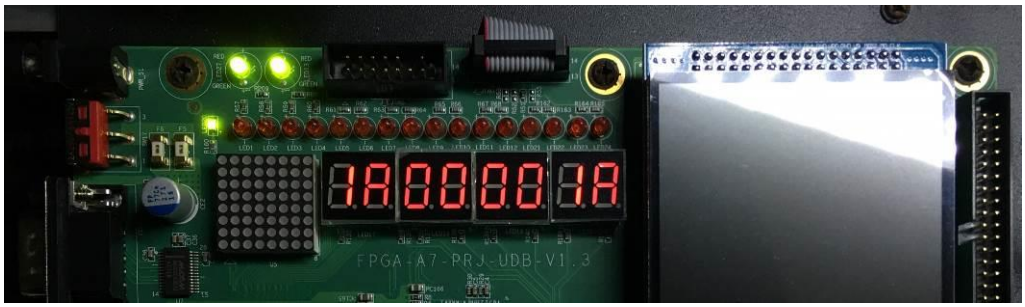

图 5-5 测试通过

数码管两侧显示的“1A”是什么意思呢？我们之前提到了，功能测试程序中一共设置了 26 个功能点，分别对 22 条指令和 4 个延迟槽做测试。这里的“1A”实际上就是 26 的 16 进制表示，说明我们的 CPU 已经完整执行了所有的功能测试。

那么前文所述的拨码开关控制的又是什么呢？如果我们查看“soft”目录下的功能测试程序源代码（start.S 文件），我们很容易发现：程序在完成每一个功能测试之后，都要执行一个“wait_1s”函数，这个函数会根据拨码开关设定的值来执行一段固定时长的延迟。如果将 8 个拨码开关视为一个 8 位二进制数，其数值越大，延迟的时间就越长。对于不同的 CPU 实现，上板验证时设定同一组拨码开关的值，延迟的实际执行的时间就可以从侧面反映出两个 CPU 性能的差异（但是不具备参考价值）。最主要的是，如果我们将延迟打开，就可以看到 CPU 在执行功能测试的过程中板子上发生的事情。如图 5-6 所示，在功能测试上板运行的过程中，数码管的值会不断累加，左上角的 LED 也会显示一红一绿。

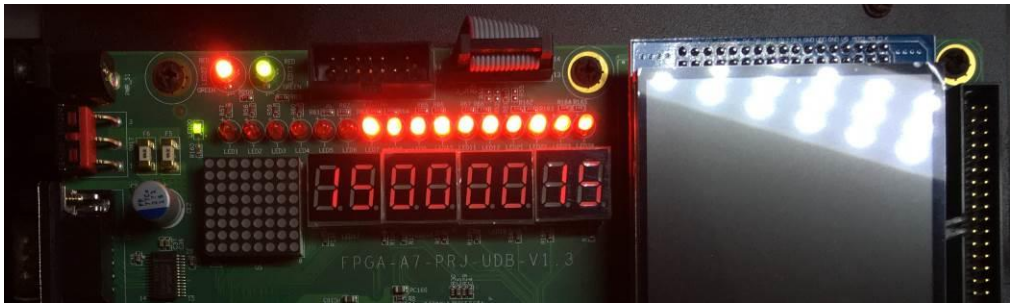

图 5-6 运行中的功能测试

另一方面，拨码开关还控制了 AXI 总线接口的访存延迟。TinyMIPS 处理器在实现上使用了 AXI 总线接口。当 CPU 需要向总线发出访存请求时，例如从内存取指令，CPU 会以 AXI 协议向内存设备发出请求，然后等待其响应。由于通常情况下 CPU 的速度要远快于访存速度，在不考虑缓存的情况下，CPU 必须等待一段时间，才能收到内存设备发来的响应信

号和返回的数据。而板子上的 8 个拨码开关，可以为 AXI 访存设定一个随机数种子，使得每次访存时，CPU 必须等待随机的一段时间。这样我们可以尽可能检测 CPU 在实现 AXI 协议时是否有误（例如只处理了特定延迟下的 AXI 请求和响应握手）。

关于功能测试上板的其他详细内容，可以参考《指导书上册》第 4.10.9 节“CPU 验证结果”的第（2）部分。