

1 功能测试

功能测试分为两部分：89 个功能点测试和记忆游戏测试。

1.1 89 个功能点测试

89 个功能点的测试方法、功能测试程序和环境的介绍参考发布包目录 *doc/All_Trace 比对机制使用说明.pdf*。

测试环境依据 myCPU 的实现接口分为两个环境：SRAM 接口的 myCPU 使用 soc_sram_func 目录下的 SoC_SRAM_Lite 环境；AXI 接口的 myCPU 使用 soc_axi_func 下的 SoC_AXI_Lite 环境。

1.1.1 测试结果判断

(1) 仿真测试结果

仿真结果正确判断有两种方法。

第一种方法，也是最简单的，就是看 Vivado 控制台打印 Error 还是 PASS，这是依据参考模型比较验证而打印的。如果仿真过程中打印了 Error，会同时打印出 myCPU 执行时的 debug 信息和参考的 debug 信号，随后可以很快的定位 bug，bug 就在打印 Error 的 PC 处，或者在其前几条指令处，这时就需要观察波形进行 debug 了。

第二种方法，是通过波形窗口观察程序执行结果。func 正确的执行行为是：

- (1) 开始，双色 LED 灯一红一绿（数值 0x1 和 0x2），数码管显示全 0；
- (2) 执行过程中，双色 LED 灯一红一绿，数码管高 8 位和低 8 位同步累加；
- (3) 结束时，双色 LED 灯亮两绿（数值 0x1 和 0x1），数码管高 8 位和低 8 位数值相同，对应测试功能点数目 89（16 进制 0x59）。

如果 func 执行过程中出错了，则数码管高 8 位和低 8 位第一次不同处即为测试出错的功能点编号，且最后的结果是双色 LED 灯亮两红，16 个单色 LED 灯全亮，数码管高 8 位和低 8 位数值不同。

最后值得一提的是，实验箱上数码管显示是采样扫描形式实现的。对于仿真而言，由于数码管要显示的数值不停的累加，且其累加的频率高于其扫描显示的频率，因而在 testbench 顶层观察不到数码管片选信号 num_csn 和数据位 num_a_g 的变化。只有观察 confreg 模块里的 num_data(其为数码管要显示的 32 位数值)，才能观察到数码管的变化。

如果需要在波形中的观察的单色 LED 灯、双色 LED 灯 0 和 1、数码管，请抓取 confreg 模块的信号 led_data、led_rg0_data、led_rgl_data、num_data。其中单色 LED 灯低 16 位写 0xffff 表示全灭；双色 LED 灯写 0x1 表示亮绿灯，写 0x2 表示亮红灯。

(2) FPGA 测试结果

在 FPGA 上验证时其结果正确与否的判断只有一种方法，func 正确的执行行为是：

- (1) 开始，16 个单色 LED 灯显示随机种子，双色 LED 灯一红一绿，数码管显示全 0；
- (2) 执行过程中，双色 LED 灯一红一绿，数码管高 8 位和低 8 位同步累加，累加频率由 8 个拨码开关控制；
- (3) 结束时，双色 LED 灯亮两绿，数码管高 8 位和低 8 位数值相同，对应测试功能点数目 89（16 进制 0x59）。

如果大家想看动态的 FPGA 运行过程，可以对 cpu132_gettrace 下的 Vivado 工程进行综合实现并上板运行，其将显示正确的 FPGA 验证运行过程。

1.1.2 Trace 比对机制使用

Trace 比对机制，对 soc_sram_func 和 soc_axi_func 都适用。如果需要使用该机制，需要 CPU 封装为相应的接口，如下表展示了 SRAM 接口下对 myCPU 接口的实验要求，参考发布包 func_test/soc_sram_func/rtl/soc_lite_top.v 里对 myCPU 的调用。如果 CPU 实现为 AXI 接口，则将下表的取指/数据 SRAM 接口统一封装成 AXI 接口，参考发布包 func_test/soc_axi_func/rtl/soc_axi_lite_top.v 里对 myCPU 的调用。

关于 Trace 比对机制的详细使用参考发布包目录 *doc/A11_Trace 比对机制使用说明.pdf*

表 1-1 myCPU 实现为 SRAM 接口的信号描述

名称	宽度	方向	描述
时钟/复位与中断			
clk	1	input	时钟信号，来自 clk_pll 的输出时钟
resetsn	1	input	复位信号，低电平同步复位
int	6	input	硬件中断，高电平有效
取指端访存接口			
inst_sram_en	1	output	ram 使能信号，高电平有效
inst_sram_wen	4	output	ram 字节写使能信号，高电平有效
inst_sram_addr	32	output	ram 读写地址，字节寻址
inst_sram_wdata	32	output	ram 写数据
inst_sram_rdata	32	input	ram 读数据
数据端访存接口			
data_sram_en	1	output	ram 使能信号，高电平有效
data_sram_wen	4	output	ram 字节写使能信号，高电平有效
data_sram_addr	32	output	ram 读写地址，字节寻址
data_sram_wdata	32	output	ram 写数据
data_sram_rdata	32	input	ram 读数据
debug 信号，供验证平台使用			
debug_wb_pc	32	output	写回级（多周期最后一级）的 PC，因而需要 mycpu 里将 PC 一路带到写回级
debug_wb_rf_wen	4	output	写回级写寄存器堆(regfiles)的写使能，为字节写使能，如果 mycpu 写 regfiles 为单字节写使能，则将写使能扩展成 4 位即可。
debug_wb_rf_wnum	5	output	写回级写 regfiles 的目的寄存器号
debug_wb_rf_wdata	32	output	写回级写 regfiles 的写数据

1.1.3 拨码开关控制 wait_1s

上板时，拨码开关有两个作用，第一个作用是：**复位后，拨码开关控制 wait_1s 的循环次数**，也就是控制数码管累加的速度。

89 个功能点测试中，每两个功能点之间会穿插一个 wait_1s 函数，wait_1s 通过一段循环完成计时的功能：在

上板时，wait_1s 循环次数由拨码开关控制，可设置循环次数为 $(0 \sim 0xaaaa) * 2^9$ 。请在复位后，通过拨码开关选择合理的 wait_1s 延时。

1.1.4 Soc_axi_fun 的随机种子控制

上板时，拨码开关有两个作用，第二个作用是：**复位期间，拨码开关控制随机种子**（只对 soc_axi_func 环境有用），也就是 axi ram 访问随机延迟的初始种子。

为尽可能验证 myCPU 的功能，AXI 接口的 CPU 支持随机延时，随机延时通过一个 23 位的伪随机数生成；在仿真时，初始随机种子由 confreg.v 里的 RANDOM_SEED 宏定义；在上板时，初始随机种子由拨码开关控制。

实验箱上共有 8 个拨码开关，实际电平是：拨上为 0，拨下为 1；但为便于以下描述，我们记作：拨上为 1，拨下为 0。16 个 LED 单色灯，实际电平是：驱动 0 亮，驱动 1 灭；但为便于以下描述，我们记作：驱动 1 亮，驱动 0 灭。

上板时，**按下复位键**，会自动采样 8 个拨码开关的值，传为初始随机种子，且会显示初始随机种子低 16 位到单色 LED 灯上。上板时随机种子与拨码开关对应关系如下表，需要注意的时延迟类型依据拨码开关的值分为三大类：长延迟、短延迟和无延迟类型。在上板运行时都应当覆盖到这三类延迟类型。

表 1-2 上板时随机种子设定

拨码开关状态	LED 灯显示	实际初始种子 seed_init
约定，8 个拨码开关：拨上为 1，拨下为 0，记作 switch[7:0] 约定，16 个单色 LED 灯：驱动 1 亮，驱动 0 灭，记作 led[15:0]； 对应关系：led[15:0]={2{switch[7]}}, {2{switch[6]}}, {2{switch[5]}}, {2{switch[4]}}, {2{switch[3]}}, {2{switch[2]}}, {2{switch[1]}}, {2{switch[0]}}		
随机延迟类型分为 3 中类型： (1) 长延迟类型：随机种子低 8 位不为 8'hff，即 seed_init[7:0]!=8'hff (2) 短延迟类型：随机种子低 8 位为 8'hff，即 seed_init[7:0]==8'hff，（排除无延迟类型） (3) 无延迟类型：随机种子低 16 位为 16'h00ff，即 seed_init[15:0]==16'h00ff		
8'h00	16'h0000	{7'b1010101, 16'h0000}
8'h01	16'h0003	{7'b1010101, 16'h0003}
8'h02	16'h000c	{7'b1010101, 16'h000c}
8'h03	16'h000f	{7'b1010101, 16'h000f}
...
8'hff	16'hffff	{7'b1010101, 16'hffff}

1.1.5 仿真通过、上板出错的调试方法

如果上板发现一个功能点都不通过（比如数码管显示全 0），可能是以下问题之一导致的：

- (1) 时序违约；
- (2) 仿真时控制信号有“X”。仿真时，有“X”调“X”，有“Z”调“Z”。AXI 接口的 valid 和 ready 信号千万不能出现“X”或“Z”。
- (3) 模块里的控制路径上的信号未进行复位。
- (4) 多驱动。

-
- (5) 代码不规范，阻塞赋值乱用，always 语句随意使用。
 - (6) 时钟复位信号接错。
 - (7) 模块的 input/output 端口接入的信号方向不对。

如果上板时发现在某一些随机种子下测试通过，在另一些随机种子情况下出错，请按以下步骤进行调试：

- (1) 确认出错的随机种子，修改 ucas_CDE_axi/rtl/CONFREG/confreg.v 里的 RANDOM_SEED 的定义，改为出错时的随机种子，随后进行仿真：如果有错，则调试；如果发现仿真没错，则在上板时找寻下一个出错的随机种子，同样设定好 RANDOM_SEED 后进行仿真，如果还是没错则转(2)。
- (2) 当碰到，相同环境下仿真无法复现上板的错误时，请都转到本步骤：反思设计；也可以使用 Vivado 的逻辑分析仪进行在线调试，参考发布包 *doc/A10_FPGA 在线调试说明*。。
- (3) 最后的功能测试通过的要求是：上板后，随意切换拨码开关，均不会出错。此时拨码开关既控制 wait_1s 延时，也控制随机初始种子。

如果上板发现任意随机种子下，都只有部分功能点测试通过。则可能以上两种原因都有，请依次排查。

如果排查后都无法解决问题，可以使用 Vivado 的逻辑分析仪进行在线调试，参考发布包 *doc/A10_FPGA 在线调试说明*。

1.2 记忆游戏测试

只有在 myCPU 封装为 AXI 接口时才能运行记忆游戏。

只有在 89 个功能点通过随机延迟的充分测试后，才能开始运行记忆游戏。

记忆游戏生产 bit 文件并下载的步骤如下：

- (1) 在 linux 环境下，进入发布包 func_test/soft/memory_game/目录，执行 make 进行编译（发布包已经包含编译完成后的文件）。
- (2) 双击打开 func_test/soc_axi_func/run_vivado/mycpu_prj1/mycpu.xpr。
- (3) 重新定制(2)中的 axi_ram IP，并加载编译结果 func_test/soft/memory_game/obj/axi_ram.coe 作为初始化文件。
- (4) 综合、布局布线，下载到实验箱，下面就可以开始测试记忆游戏了。

记忆游戏的步骤如下：

- (1) 按实验箱上矩阵键盘的最下面一行的 4 个按键中的任一按键，开始游戏。
- (2) 单色 LED 等最右侧 4 个灯，会随机点亮，共点亮 8 次。
- (3) 努力回忆 8 次点亮顺序，使用阵键盘的最下面一行的 4 个按键复现 8 次点亮顺序。
- (4) 按完 8 次后，数码管左侧会展示 8（表示共 8 次点亮），右侧会展示你记忆对的次数。

(5) 如果 8 次都记忆对了，则数码管右侧会展示 8，且双色 LED 灯会亮 2 个绿色；如果不是 8 次都对，双色 LED 灯会亮一红一绿。

如果发现记忆游戏运行不正确，请按以下步骤调试：

- (1) 确认之前的 89 个功能点测试充分覆盖了各类随机种子，且均没有运行出错。
- (2) 可以尝试对记忆游戏进行仿真，但需要自己设计按键的激励，且需要能看懂记忆游戏源码并修改 `delay` 函数。
- (3) 可以尝试使用逻辑分析仪进行在线调试。
- (4) 反思自己的设计、代码规范、综合时序等。

1.3 功能测试分数计算

当 myCPU 仅实现为 SRAM 接口时，只能运行 `soc_sram_func`。

当 myCPU 封装为 AXI 接口时，可以运行 `soc_axi_func` 和记忆游戏。

可以参考大赛发布包 *Qualifiers_Submission/score.xls*，该 Excel 文档提供了自动计算。

功能测试分数计算规则：

- (1) 实现为 AXI 接口的 myCPU，在 89 个功能点测试全通过后，不需要运行 SRAM 接口的 myCPU 测试。得分为 $(89 + \text{记忆游戏} * 11)$ ；
- (2) SRAM 接口的 myCPU 测试，只有在 89 个功能点测试全通过后，才会计算 AXI 接口的 myCPU 运行 89 个功能点和记忆游戏的分数。得分为 $(44.5 + \text{AXI} * 0.5 + \text{记忆游戏} * 11 * 0.5)$ ；
- (3) SRAM 接口的 myCPU 测试，运行 89 个功能点测试不是全通过，且 AXI 接口 myCPU 也不是全通过，则只计算 SRAM 接口的 myCPU 得分。得分为 $(\text{SRAM} * 0.5)$ 。

1.4 功能测试分数提交

参考大赛发布包 *Qualifiers_Submission/预赛提交说明.pdf* 和 *Qualifiers_Submission/score.xls*。