

1 性能测试

运行性能测试前，请先充分运行功能测试并确保通过。

性能测试程序共分为 10 个小程序。

性能测试位于发布包 *perf_test*/目录，目录组织结构如下：

-soc_axi_perf_demo/	目录，gs132 运行性能测试的环境（无延时），与 soc_axi_perf 目录组织类似
-soc_axi_perf/	目录，自实现 CPU 的性能测试环境（固定延时）
--rtl/	目录，SoC_lite 的源码。
--soc_axi_lite_top.v	SoC_lite 的顶层。
 --myCPU /	目录，自实现 CPU 源码。
--CONFREG/	目录，confreg 模块，连接 CPU 与开发板上数码管、拨码开关等 GPIO 类设备。
--ram_wrap/	目录，axi ram 的封装层，增加固定延迟设置。
--xilinx_ip/	目录，Xilinx IP，包含 clk_pll、axi_ram、axi_crossbar_1x2。
--testbench/	目录，仿真文件。
--mycpu_tb.v	仿真顶层。
--run_vivado/	目录，运行 Vivado 工程。
--soc_lite.xdc	Vivado 工程设计的约束文件
--mycpu_prj1/	目录，Vivado2018.1 创建的 Vivado 工程 1
--run_allbench.tcl	仿真依次运行 10 个性能测试程序的脚本
--*.xpr	Vivado2018.1 创建的 Vivado 工程，可直接打开
-soft/	目录，性能测试程序。
--perf_func/	目录，性能测试程序。
--include/	目录，mips 编译所有头文件
--lib/	目录，性能测试程序需使用的 C 库
--obj/	目录，性能测试编译结果
--Makefile	编译脚本 Makefile
--start.S	func 的主函数
--bin.lds.S	交叉编译的链接脚本源码
--convert.c	生成 coe 和 mif 文件的本地执行程序源码
--rules.make	子编译脚本，被 Makefile 调用
--Readme_First.txt	soft/perf_func/目录简单介绍
-Readme_First.txt	本目录简单介绍

1.1 快速上手

- (1) 【准备环境】确认大赛发布包的性能测试目录 *perf_test*/的位置路径中没有中文字符。
- (2) 【确认功能测试】确认 myCPU 接口封装为 AXI，已经充分运行 soc_axi_func 里的功能测试并通过。

-
- (3) **【myCPU 加入】**将 myCPU 代码拷贝到 *perf_test/soc_axi_perf/rtl/myCPU/*目录下；
 - (4) **【编译 Func】**步骤和功能测试类似，在 *perf_test/soft/perf_func* 里进行编译，编译后会该目录中得到 *obj/*目录（发布包中默认含有已生成好的 *obj* 目录）。（*perf_func* 编译结果的介绍详见本章 1.4 节）。
 - (5) **【myCPU 添加进工程】**进入 *perf_test/soc_axi_perf/run_vivado/mycpu_prj1/*目录，打开 Vivado 工程 mycpu，将你在第（2）步新加的文件添加到工程中。
 - (6) **【stream_copy 仿真】**stream_copy 运行时间最短，故推荐首先运行该测试，运行方法为：mycpu 工程中的 axi_ram 加载 *perf_test/soft/perf_func/obj/stream_copy/axi_ram.coe*（参考本章 1.5.1 节），进行仿真，参考本章 1.5.2 节的方法，看仿真输出 log 是否正确。如果结果异常，则进行调试直至通过。
 - (7) **【stream_copy 上板】**mycpu 工程中的 axi_ram 必须加载 *perf_test/soft/perf_func/obj/stream_copy/axi_ram.coe*，确认 CLK_PLL IP 的输出时钟 timer_clk 为 100MHz，cpu_clk 为 myCPU 可支持的频率，综合实现并生成 bit 流文件，确认 implementation 后 WNS 非负值。下载 bit 流文件，进行上板验证，观察实验箱上数码管、双色 LED 灯、单色 LED 灯显示结果，参考本章 1.6.2 节的方法，判断是否正确。如果结果与要求的一致，则 stream_copy 上板运行成功；否则转到第（11）步进行问题排查。
 - (8) **【其他性能测试】**可按照（6）~（7）步，依次对其他性能测试程序 bitcount、dhrystone、stringsearch、select_sort、quick_sort、sha、bubble_sort、crc32、coremark 进行仿真和上板。这些测试程序可以同步进行。
 - (9) **【联合性能测试】**mycpu 工程中的 axi_ram 加载 *perf_test/soft/perf_func/obj/allbench/axi_ram.coe*，进行综合实现并生成 bit 流文件。下载 bit 流文件，参考本章 1.6.3 节的方法，依次设定拨码开关，随后按实验板上复位键，选择待观察的性能测试。
 - (10) **【计算性能分】**联合性能测试通过后，记录各性能测试程序的上板运行时间，参考本章 1.8 节的方法，计算性能测试最终得分。
 - (11) **【反思】**仿真和上板行为不一致时，请按照下列步骤依次排查。
 - a) 复核生成、下载的 bit 文件是否正确。
 - b) 复核仿真结果是否正确。
 - c) 检查实现时的时序报告（Vivado 界面左侧“IMPLEMENTATION”→“Open Implemented Design”→“Report Timing Summary”），确保时序满足。
 - d) 认真排查综合和实现时的 Warning，特别是 Critical Warning，尽量修正。
 - e) 排查 RTL 代码规范，避免多驱动、阻塞赋值乱用、模块端口乱接、时钟复位信号接错。
 - f) 将 *perf_test/soc_axi_perf/rtl/soc_axi_lite_top.v* 里的 parameter 参数 SIMULATION 设定为 1，宏定义 SIMU_USE_PLL 设为 1，再进行仿真、综合实现上板验证，确认是否有错：如果有错，则再次复核 a)~e)，最后转 g)；如果无错，再尝试 SIMULATION 为 0 时是否有错，有错则转 g)。
 - g) 如果你会使用 Vivado 的逻辑分析仪进行板上在线调试，那么就去调试吧；如果调试了半天仍然无法解决问题，转 h)。
 - h) 反思。真的，现在除了反思还能干什么？

1.2 性能测试 SoC 介绍

性能测试 SoC 位于 资源发布包 目录下 *perf_test/soc_axi_perf*。

该运行环境与功能测试的 SoC 环境 *soc_axi_func* 基本是一致的，如下图。

黄色部分为自实现 myCPU，最后要求封装为 AXI 接口，不一定需要使用类 SRAM-AXI 转换桥来转接。

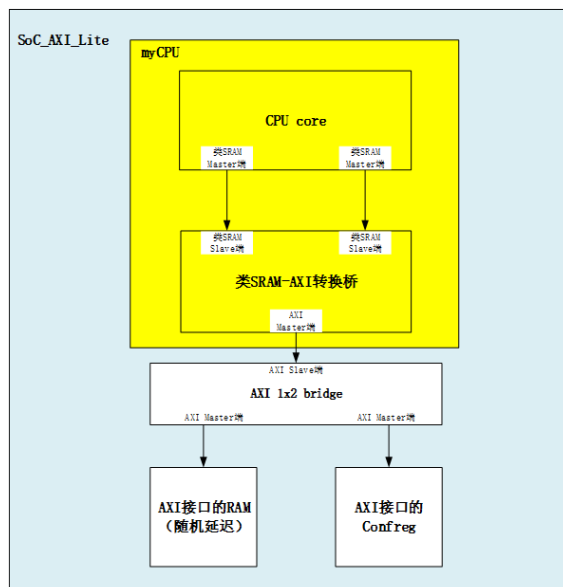


图 1-1 基于 mycpu 的简单硬件系统 SoC_AXI_Lite

1.2.1 AXI RAM 固定延时

依据在 vivado 里对 DDR3 控制器的仿真行为观察，在性能测试时，为尽量模拟实际 FPGA 上 DDR3 访存行为，我们对 SoC 中的 AXI RAM 设置了固定延时：写操作 *aw->w->b*（从写请求发出到写响应返回）有 3 拍延时，读操作 *ar->r*（从读请求发出到读数据返回）有 25 拍延时。这一机制的代码参见 *perf_test/soc_axi_perf/rtl/ram_wrap/axi_ram_wrap.v*。

这一机制使得 myCPU 大部分时间都花费在等待取回指令或数据上。因而性能测试仿真也会严重变慢。

1.2.2 PLL 设置

无论功能测试还是性能测试的 SoC 中，里面都调用了 xilinx 的 PLL IP 用来给 SoC 生成 2 个适当频率的时钟 *timer_clk*（外部计时器时钟，固定 100MHz）和 *cpu_clk*（SoC 全局时钟，根据 CPU 频率调整）。

在仿真时，该 PLL 单元会严重拖慢仿真速度。为避免这一问题，在性能测试环境里，对 RTL 进行了设置：PLL 单元只在综合实现时候调用，在仿真时不会调用 PLL，仿真时 *timer_clk* 和 *cpu_clk* 固定为 100MHz。这一设置的具体代码可以参考 *perf_test/soc_axi_perf/rtl/soc_axi_lite_top.v*。

上述设置使得仿真时候的设计和上板时候的设计不完全一致，但通常不会带来出错。如果碰到仿真和上板行为不一致的时候，担心是这一快速仿真机制的原因，可以在 *perf_test/soc_axi_perf/rtl/soc_axi_lite_top.v* 里将参数 *SIMU_USE_PLL* 设定为 1，表示仿真时使用 PLL 单元，这样仿真和上板时 SoC 环境时完全一致的。甚至可以在上板时将 *perf_test/soc_axi_perf/rtl/soc_axi_lite_top.v* 里的 parameter 参数 *SIMULATION* 默认值设为 1（该值会影响性能测试程序的循环次数），这样仿真和上板时软件行为也是一致的了。

1.3 性能测试程序介绍

性能测试的 10 个小程序源码位于 资源发布包目录下 `perf_test/soft/perf_func/bench` 目录下，`perf_test/soft/perf_func/start.S` 为共有的启动程序。

表 1-1 性能测试程序

序号	测试程序	性能测试程序介绍
1	bitcount	来自 Mibench 测试 automotive 集，统计一个整数数组包含的 bit 中 1 的个数
2	bubble_sort	冒泡排序算法
3	coremark	嵌入式系统中 CPU 性能测试程序，2009 年由 EEMBC 发布。程序包括查找和排序、矩阵操作、状态机和循环冗余操作四部分算法
4	crc32	来自 Mibench 测试 telecomm 集，CRC32 计算工具
5	dhrystone	程序的主要目标是测量处理器的整型运算性能
6	quick_sort	快速排序算法
7	select_sort	选择排序算法
8	sha	来自 Mibench 测试 security 集，SHA 散列算法
9	stream_copy	来自 Stream 测试集的 Copy 操作，访问一个内存单元读出其中的值，再将值写入到另一个内存单元
10	stringsearch	来自 Mibench 测试 office 集，字符串查找工具

1.4 编译性能测试程序

编译环境见 资源发布包目录下 `perf_test/soft/perf_func/`。

需要注意的是，每个性能测试程序中都有默认的 gcc 优化选项，该项不得更改。

编译后结果存放在 `perf_test/soft/perf_func/obj/` 各子目录下，

表 1-2 编译生成的子目录

子目录	解释
allbench	联合编译结果。包含 10 个测试程序的源码，用于综合实现并上板。
bitcount	独立编译结果。仅包含 bitcount 测试程序。
bubble_sort	独立编译结果。仅包含 bubble_sort 测试程序。
coremark	独立编译结果。仅包含 coremark 测试程序。
crc32	独立编译结果。仅包含 crc32 测试程序。
dhrystone	独立编译结果。仅包含 dhrystone 测试程序。
quick_sort	独立编译结果。仅包含 quick_sort 测试程序。
select_sort	独立编译结果。仅包含 select_sort 测试程序。
sha	独立编译结果。仅包含 sha 测试程序。
stream_copy	独立编译结果。仅包含 stream_copy 测试程序。
stringsearch	独立编译结果。仅包含 stringsearch 测试程序。

各子目录中具体编译得到的文件如下：

表 1-3 编译生成的文件

文件名	解释
axi_ram.coe	重新定制 axi ram 所需的 coe 文件

axi_ram.mif	仿真时 axi ram 读取的 mif 文件
inst_data.bin	编译后的代码、数据段，可以不用关注
main.elf	编译后的 elf 文件
test.s	对 main.elf 反汇编得到的文件

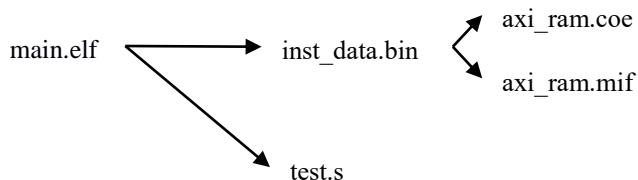


图 2 编译得到的文件生成关系

1.5 仿真

1.5.1 仿真指定程序

性能测试程序共分为 10 个测试程序，建议按照仿真运行时间从少到多，依次对 stream_copy、bitcount、dhrystone、stringsearch、select_sort、quick_sort、sha、bubble_sort、crc32、coremark 进行仿真。因而仿真时间越短，发现错误迭代会更快速，随着性能测试依次通过，myCPU 也趋于稳定，运行仿真很慢的 coremark 出错的可能也就越低。

仿真前请确认已生成 *perf_test/soft/perf_func/obj* 目录，发布包中默认含有已生成好的 obj 目录。

对单独一个程序进行仿真时，以仿真 stream_copy 程序为例，有两个方法：

- (1) 方法一：双击 vivado 工程里的 axi ram，重新定制，在 IP 定制界面->“Other Options”->“Memory Initialization”选择 *perf_test/soft/perf_func/obj/stream_copy/axi_ram.coe*，点击“OK”确认。这样就可以更新 axi ram 里的程序为 stream_copy，相应的仿真也是对该程序进行仿真，只是需要重新定制 axi ram 会耗费一些时间。
- (2) 方法二：先打开性能测试的仿真波形界面，然后在控制台“Tcl Console”输入“cd [get_property DIRECTORY [current_project]]”并回车切换到当前工程目录（如果已在工程目录则不需要切换了，可以输入“pwd”查看当前目录）；随后继续输入“file copy -force ../../soft/perf_func/obj/stream_copy/axi_ram.mif ./mycpu.sim/sim_1/behav/xsim/axi_ram.mif”，这一命令的作用是将仿真目录 mycpu.sim/sim_1/behav/xsim/ 里的 mif 文件替换为 stream_copy 对应的 mif；可以继续输入“restart”->“run all”命令（或者在仿真界面点击相应按键），即可对 stream_copy 进行仿真。

推荐方法二，因为不需要重新定制 axi ram，可以快速更换被仿真的程序。

仿真时，由 testbench 顶层 *perf_test/soc_axi_perf/testbench/mycpu_tb.v* 里传递 parameter 参数 SIMULATION 为 1，表示是仿真环境，此时运行的性能测试程序会自动识别并设置循环次数为 1。

1.5.2 仿真结果判断

如果性能仿真正确运行，在控制台“Tcl Console”里可以看到类似如下打印信息

```
Test begin!
...(不同程序有不同打印)

... PASS!... (不同程序有所不同)

...: Total Count =...(不同程序有所不同)
```

如果需要观察详细打印信息，可以运行 `soc_axi_perf_demo` 里的仿真，观察龙芯开源 `gs132` 运行性能测试程序的打印信息。

1.5.3 仿真所有程序

当所有性能测试程序的仿真都已经稳定无错后，想要再次确认，或者最后提交预赛分数前再次确认仿真结果，可以使用以下方法采用脚本的方式自动对 10 个性能测试程序进行仿真：

- (1) 打开性能测试的仿真波形界面；
- (2) 在控制台“Tcl Console”输入“`cd [get_property DIRECTORY [current_project]]`”并回车切换到当前工程目录（如果已在工程目录则不需要切换了，可以输入“`pwd`”查看当前目录）；
- (3) 继续输入“`source ./run_allbench.tcl`”并回车，这一命令的作用是依次对 10 个性能测试程序进行仿真，感兴趣可以查看该脚本文件 `perf_test/soc_axi_perf/run_vivado/mycpu_prj1/run_allbench.tcl`。

`soc_axi_perf_demo` 里也支持该命令进行所有性能测试程序的自动仿真。

1.5.4 快速仿真机制

对于初始运行性能测试，可以关闭 AXI RAM 的固定延时机制，以加快仿真。关闭的方法是：将 `perf_test/soc_axi_perf/rtl/ram_wrap/axi_ram_wrap.v` 里的宏定义 `RUN_PERF_NO_DELAY` 开启（``define RUN_PERF_NO_DELAY` 代表开启，``define RUN_PERF_NO_DELAY` 代表关闭，后者前面有一个下划线）。

需要注意的时，开启宏 `RUN_PERF_NO_DELAY` 后，由于访存行为不一致了，CPU 行为也会不一致，相应的出错的现象也会不一致。但一个正确的 CPU，应当无论是开启还是关闭宏 `RUN_PERF_NO_DELAY`，都应当能够正确执行。

注意 `perf_test/soc_axi_perf/rtl/ram_wrap/axi_ram_wrap.v` 里的宏 `RUN_PERF_TEST` 应该一直开启，不要修改。

性能测试分数提交应当是关闭宏 `RUN_PERF_NO_DELAY` 时候的分数。

1.5.5 参考仿真时间

在使用龙芯开源 `gs132` 运行性能测试程序仿真时，统计 `vivado` 工具的耗时如下表。表中无延时即开启宏 `RUN_PERF_NO_DELAY` 时的情况，固定延时即关闭宏 `RUN_PERF_NO_DELAY` 时的情况，0'50 表示 0 分 50 秒。

注意，下表数据不是发布包里 `perf_test/soc_axi_perf_demo` 里的运行结果，`soc_axi_perf_demo` 里的仿真环境时使用 `gs132` 的网表文件进行仿真的，网表文件进行仿真比 RTL 源码仿真要慢很多。`soc_axi_perf_demo` 里的仿真时间预计是下表时间的 5 倍。`soc_axi_perf_demo` 里默认设置 AXI RAM 为无延时的情况。

表 1-4 `gs132` 运行性能测试程序实际时间

运行环境	Win10, Vivado2018.1 处理器: Intel Core i5-6200U CPU @2.30GHz 内存: 8GB 运行 vivado 是以 4 线程并行运行, 且电脑不运行其他大型工程。		
序号	测试程序	无延时	固定延时
1	bitcount	0'50	2'06
2	bubble_sort	4'31	11'17
3	coremark	10'15	25'47
4	crc32	6'14	16'26
5	dhystone	1'14	3'04
6	quick_sort	4'18	10'55
7	select_sort	4'02	10'30
8	sha	4'19	11'06
9	stream_copy	00'21	0'54
10	stringsearch	3'01	7'15

1.5.6 仿真调试建议

性能测试时没有 trace 比对机制, 调试难度大大增加。因而建议大家按照从易到难依次对 10 个性能测试程序进行调试。调试过程中需要对性能测试程序源码有一定的理解, 特别是结合反汇编文件 test.s 进行调试。

最后, 大家可以考虑自己搭建一个简单的 Trace 比对机制: 可以先抓取龙芯开源 gs132 运行测试程序的写回级信息 debug_wb_pc、debug_wb_rf_wen、debug_wb_rf_wnum、debug_wb_rf_wdata, 输出到一个文件中(使用 \$fdisplay 打印信息到文件中), 然后抓取自实现 CPU 运行测试程序的写回寄存器堆的信号到另一个文件中, 对比这两个文件, 从前向后找寻第一个差异处, 通常 myCPU 出错就出错在这条指令附近。需要注意的是, 运行性能测试前后有读出计时器的指令, 显然不同 CPU 读回的计时器值也不同, 这会导致 Trace 比对在这附近会出错, 如果自己搭建 Trace 比对机制发现到差异处, 应当要注意排除这一情况。

1.6 上板

1.6.1 上板指定程序

通常所有测试程序仿真都稳定无错后, 可以参考 1.6.3 节对所有测试程序进行上板测试。

如果依然想对单独一个程序进行上板测试时, 可以使用以下方法:

- (1) 双击 vivado 工程里的 axi ram, 重新定制, 在 IP 定制界面->“Other Options”->“Memory Initialization”选择 *perf_test/soft/perf_func/obj* 里指定程序下的 axi_ram.coe, 点击“OK”确认。这样就可以更新 axi ram 里的程序为相应程序, 重新进行综合和布局布线即可。这一过程耗时比较长。

上板时, 无法使用 1.5.1 节里的方法二, 因为 mif 文件只能用于仿真, 需要更换上板时 bit 流文件里 RAM 的数据, 只有重新定制 RAM。

上板时, 由于 SoC 顶层 *perf_test/soc_axi_perf/rtl/soc_axi_lite_top.v* 里的 parameter 参数 SIMULATION 默认值为 0, 表示是上板环境, 此时运行的性能测试程序会自动识别并设置循环次数为 10。

1.6.2 上板结果判断

如果性能测试上板正确运行，双色 LED 灯全变为绿色，16 个单色 LED 全灭。如果性能测试的功能错误，双色 LED 灯会变成一绿一红，16 个单色 LED 灯全亮。无论功能是否正确都会在数码管上显示计数周期。

1.6.3 上板所有程序

其实在所有性能测试程序的仿真都已经稳定无错后，可以直接进行上板所有程序，通常不需要进行单独测试程序的上板测试。

上板所有程序，需要使用 10 个性能测试程序的联合编译结果，参考 1.4 节。

装载所有程序进行上板测试时，可以使用以下方法：

- (1) 双击 vivado 工程里的 axi ram，重新定制，在 IP 定制界面->“Other Options”->“Memory Initialization”选择 `perf_test/soft/perf_func/obj/allbench/axi_ram.coe`，点击“OK”确认。这样就可以更新 axi ram 里的程序为联合编译的结果，其中包含了所有性能测试程序，重新进行综合和布局布线即可。这一过程耗时比较长。

下载 bit 流文件后，在实验板上使用 8 个拨码开关的右侧 4 个选择运行哪个测试，随后按复位键，开始运行由拨码开关指定的测试，数码管会显示当次运行所花费的 count 数，如图 1-2。约定拨码开关拨上为 1，拨下为 0，则 4 个拨码开关与性能测试程序的对应关系如表 1-5。



图 1-2 上板运行性能测试的操作

表 1-5 拨码开关与性能测试程序对应关系

约定序号	板上拨码开关，拨下为 1，拨下为 0	运行的测试程序	拨码开关状态
1		bitcount	4'b0001
2		bubble_sort	4'b0010
3		coremark	4'b0011
4		crc32	4'b0100
5		dhystone	4'b0101
6		quick_sort	4'b0110
7		select_sort	4'b0111

8	sha	4'b1000
9	stream_copy	4'b1001
10	stringsearch	4'b1010
其他	不运行性能测试	其他

1.6.4 上板与仿真行为不一致

通常上板测试，应当是仿真通过后再进行。我们建议 10 个性能测试程序仿真都通过后，再开始上板测试。通常，上板应该不会再出现错。但如果有同学碰到上板与仿真行为不一致的，可以参考 1.1 节 快速上手里的说明进行调试。

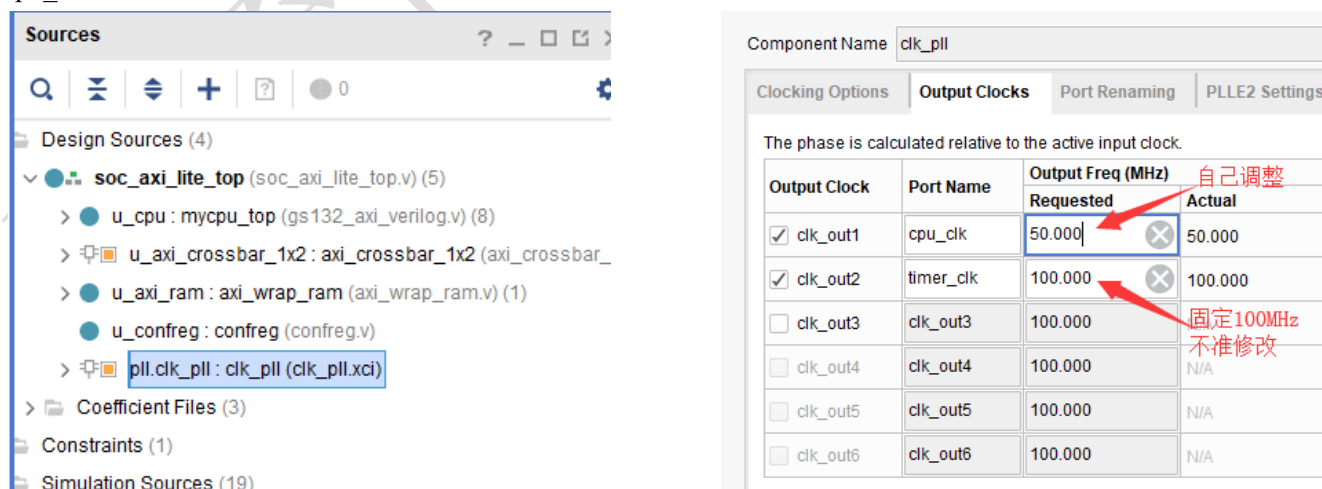
这里强调，目前我们设定的上板和仿真的环境是有所区别的：

- (1) SoC 环境不一致：PLL 单元，在仿真和上板时有所区别，参考 1.2.2 节。可以在 `perf_test/soc_axi_perf/rtl/soc_axi_lite_top.v` 里将参数 `SIMU_USE_PLL` 设定为 1，表示仿真时使用 PLL 单元，再进行仿真，确保仿真无错。
- (2) 软件运行环境不一致：parameter 参数 `SIMULATION` 指定了当前运行环境是仿真还是上板，可以将 `perf_test/soc_axi_perf/rtl/soc_axi_lite_top.v` 里的 parameter 参数 `SIMULATION` 默认值设为 1，表示上板使用仿真的环境，同时 AXI RAM 也有进行重新定制，选择与仿真时对应的测试程序的独立编译结果，进行综合布局布线生成 bit 流文件，下载到实验板上观察。

1.7 CPU 频率调整

性能测试统计的是 myCPU 运行性能测试程序实际花费的时间，其原理是：测试换 SoC_AXI_Lite 里设置了一个固定 100MHz 的计时器，在运行性能测试程序的前后读取该计时器，其差值就是运行这一性能测试程序的所花费的实际时间。

因而需要大家自行调整 SoC_AXI_Lite 里的 `cpu_clk`，使其为 myCPU 支持的最高频率，以获取最高性能分。调整 `cpu_clk` 的方法如下：



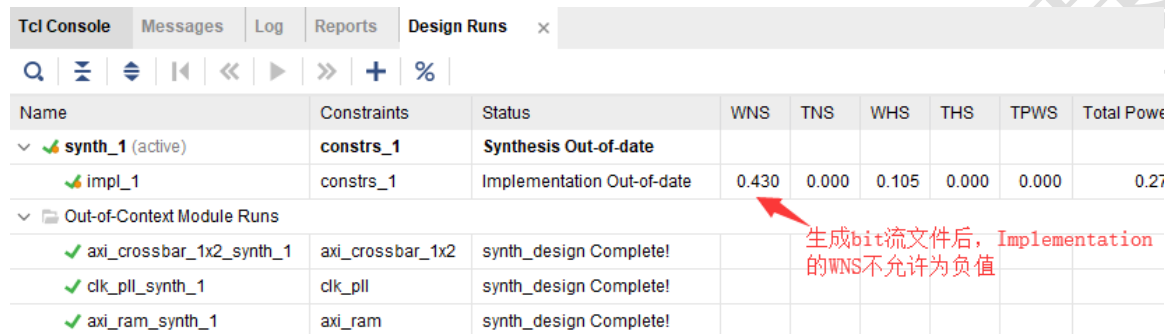
(1) 双击 PLL 进行 IP 重新定制

(2) 选择 Output Clocks 调整 `cpu_clk`

图 1-3 调试 SoC 里的 `cpu_clk`

调整 `cpu_clk` 后，一定要注意综合实现生成 bit 流文件后，Implementation 栏的 WNS 不允许为负值，见图 1-4。当 WNS 为负值时，表示设计中有违约路径，该栏也会显示为红色。在此提醒大家：

- (1) WNS 为负值，生成的 bit 流文件也可能是能够正确运行的。但是比赛统一约定，不允许 WNS 为负值。
- (2) 通常 SoC 里 uncore 部分不会成为违约路径，但如果发现 myCPU 没有违约路径，WNS 却为负值，这也是不被允许的。比赛统一约定，不允许 WNS 为负值。
- (3) myCPU 一定是嵌入到性能测试环境 SoC_AXI_Lite 里综合实现后 WNS 非负值。如果预赛提交作品不满足该项，性能测试分记为 0 分。



Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power
▼ synth_1 (active)	constrs_1	Synthesis Out-of-date						
▼ impl_1	constrs_1	Implementation Out-of-date	0.430	0.000	0.105	0.000	0.000	0.27
Out-of-Context Module Runs								
✓ axi_crossbar_1x2_synth_1	axi_crossbar_1x2	synth_design Complete!						
✓ clk_pll_synth_1	clk_pll	synth_design Complete!						
✓ axi_ram_synth_1	axi_ram	synth_design Complete!						

图 1-4 生成 bit 流文件后 Implementation 栏的 WNS 不允许为负值。

1.8 性能测试得分计算

1.8.1 GS132 性能测试计数

龙芯开源 gs132 运行于 50MHz 时，运行 10 个性能测试的计数结果见表 1-6。表中计数结果均为 16 进制。

仿真时，不采用 PLL 生成 `cpu_clk` 和 `timer_clk`，而是固定它们为 100MHz，所以仿真时 `cpu_clk : timer_clk = 100MHz : 100MHz`。仿真时，每个性能测试程序只循环一次。

上板时，`cpu_clk` 设为 50MHz，`timer_clk` 不准修改，为 100MHz，所以上板时 `cpu_clk : timer_clk = 50MHz : 100MHz`。上板时，每个性能测试程序循环十次，所以增加了一栏显示为“数码管显示/10”。

由于仿真 `cpu_clk` 与 `timer_clk` 为 1:1，而上板时为 1:2，所以可以看到“数码管显示/10”栏和“仿真”栏数值比大约也是 2:1。

注意表中的数据统计都是在设置 AXI RAM 为固定延时时候的运行结果。大赛发布包 `perf_test/soc_axi_perf_demo` 为默认设为无延时的情况。如果想要使用 `perf_test/soc_axi_perf_demo` 来运行 gs132 的性能测试，以得到下表数据，需要设置 AXI RAM 为固定延时，设置方法是：将 `perf_test/soc_axi_perf_demo/rtl/ram_wrap/axi_ram_wrap.v` 里的宏定义 `RUN_PERF_NO_DELAY` 关闭（将该行宏定义注释掉，或者在宏前加下划线“_”）。此时仿真会特别慢（比无延时大约慢 5 倍），建议直接综合上板查看结果。

表 1-6 gs132 运行性能测试计数

序号	测试程序	仿真(16 进制)	上板(16 进制)		数码管显示/10 : 仿真
			数码管显示	数码管显示/10	
cpu_clk : timer_clk		100MHz : 100MHz	50MHz : 100MHz		-
1	bitcount	1040B0	144FF46	207FED	2.000
2	bubble_sort	632030	7BE6D9E	C63E29	2.000
3	coremark	DB630C	11241A4E	1B6CF6E	2.000
4	crc32	8B384E	AE0DE84	1167CA6	2.000
5	dhrystone	188BFE	1EC6EF8	313E4C	2.006
6	quick_sort	5C53F6	7367916	B8A5B5	2.000
7	select_sort	5AF112	71ABF46	B5DFED	2.000
8	sha	5EE1DE	769AA9C	BDC442	2.000
9	stream_copy	6BB22	86947A	D753F	1.999
10	stringsearch	3D5258	4CC49B2	7AD42B	2.003

1.8.2 性能计算公式

性能测试得分以龙芯开源的 gs132 上板运行结果为基准分，进行归一化后，求几何平均。

假设 myCPU 上板运行性能测试都通过的情况系啊，COUNT 数依次为：T_{bitcount}、T_{bubble_sort}、T_{coremark}、T_{crc32}、T_{dhrystone}、T_{quick_sort}、T_{select_sort}、T_{sha}、T_{stream_copy}、T_{stringsearch}。如果某一测试项未通过，则该项运行时间 T 记为 T_{gs132}*10。

则 myCPU 运行性能测试花费的时间与 gs132 运行时间的比值求积：

$$\frac{T_{bitcount}}{0x144FF46} \times \frac{T_{bubble_sort}}{0x7BE6D9E} \times \frac{T_{coremark}}{0x11241A4E} \times \frac{T_{crc32}}{0xAE0DE84} \times \frac{T_{dhrystone}}{0x1EC6EF8} \times \frac{T_{quick_sort}}{0x7367916} \times \frac{T_{select_sort}}{0x71ABF46} \\ \times \frac{T_{sha}}{0x769AA9C} \times \frac{T_{stream_copy}}{0x86947A} \times \frac{T_{stringsearch}}{0x4CC49B2}$$

对上式求倒数并求 10 次根所得即为自实现 CPU 的最终性能分数，记为 P_{mycpu}：

$$P_{mycpu} = \sqrt[10]{\left(\frac{0x144FF46}{T_{bitcount}} \times \frac{0x7BE6D9E}{T_{bubble_sort}} \times \frac{0x11241A4E}{T_{coremark}} \times \frac{0xAE0DE84}{T_{crc32}} \times \frac{0x1EC6EF8}{T_{dhrystone}} \times \frac{0x7367916}{T_{quick_sort}} \times \frac{0x71ABF46}{T_{select_sort}} \times \frac{0x769AA9C}{T_{sha}} \times \frac{0x86947A}{T_{stream_copy}} \times \frac{0x4CC49B2}{T_{stringsearch}} \right)}$$

自实现 CPU 运行时间与 gs132 做比值，即为归一化，求积取根即为几何平均。

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

1.9 性能测试分数提交

参考大赛发布包 [Qualifiers_Submission/预赛提交说明.pdf](#) 和 [Qualifiers_Submission/score.xls](#)。