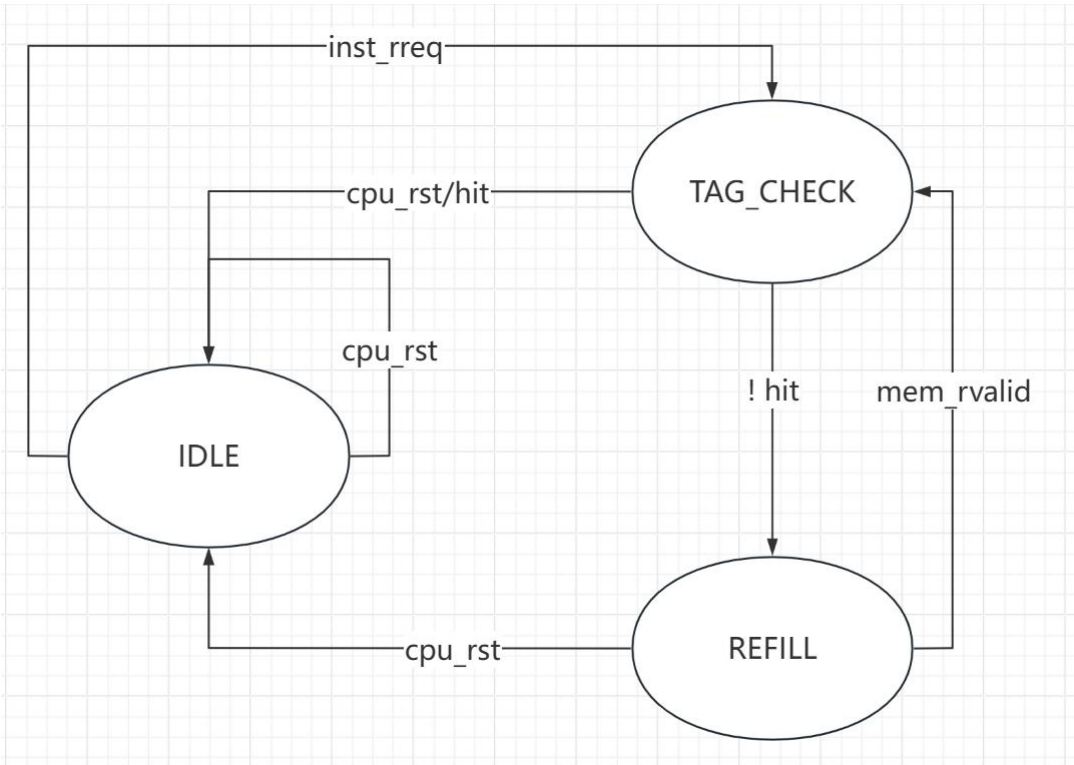


1、Cache 详细设计

要求：绘制 ICache 的状态转换图，并详细描述状态转移关系、转移条件、各状态的输入输出信号以及需要完成的操作。**若完成了附加题，则分别绘制 DCache 的读、写状态转换图，并配以文字详细描述相应的内容。*

ICache 的状态转移图如下：



当 `cpu_rst` 信号到来时，无论 ICache 当前处于任意状态，次态 `next_state` 都会更新为 IDLE，下一个时钟上升沿信号到来时 ICache 将进入 IDLE 状态。

1. IDLE 状态：

当 ICache 处于 IDLE 状态时，会将输出给主存的读使能信号 `mem_ren` 和记录是否已发送过读请求的信号 `mem_ren_pulse` 都置为 0（`mem_ren_pulse` 用于避免在 REFILL 状态中重复发送读请求以防止超时）。

ICache 处于 IDLE 状态时，当来自 CPU 的取指请求信号 `inst_req` 为高电平时，次态 `next_state` 更新为 TAG_CHECK，ICache 将在下一个时钟上升沿信号到来时进入 TAG_CHECK 状态；

否则（`inst_req` 为低电平时），次态 `next_state` 仍为 IDLE，下一个时钟上升沿信号到来时 ICache 仍将处于 IDLE 状态，`mem_ren` 和 `mem_ren_pulse` 都置为 0，表明当前状态下未收到读请求。

2. TAG_CHECK 状态：

当 ICache 处于 IDLE 状态时，会将输出给主存的读使能信号 `mem_ren` 置为 0。

ICache 处于 IDLE 状态时，当读命中信号 `hit` 为高电平时，次态 `next_state` 更新为 IDLE，ICache 将在下一个时钟上升沿信号到来时进入 IDLE 状态。进入 IDLE 状态后，将输出给主存的读使能信号 `mem_ren` 和记录是否已发送过读请求的信号 `mem_ren_pulse` 都置为 0，等待下一次读请求信号的到来；

否则（`hit` 为低电平，未命中时），次态 `next_state` 更新为 REFILL，ICache 将在下一个时钟上升沿信号到来时进入 REFILL 状态。

3. REFILL 状态：

当 ICache 处于 REFILL 状态时，若主存就绪信号 `mem_rdy` 为高电平且记录是否已发送过读请求的信号 `mem_ren_pulse` 为低电平（表示内存已经准备好接受读请求，并且给主存的读请求还未发出），设置内存读取地址 `mem_raddr`（`mem_raddr = {inst_addr[31:4], 4'b0000}`），将输出给主存的读使能信号 `mem_ren` 和记录是否已发送过读请求的信号 `mem_ren_pulse` 都置为 1（表示读请求已经发出且可以有效避免重复发出读请求）；否则，将输出给主存的读使能信号 `mem_ren` 置 0。

ICache 处于 REFILL 状态时，当来自主存的数据有效信号 `mem_rvalid` 为高电平时，次态 `next_state` 更新为 TAG_CHECK，ICache 将在下一个时钟上升沿信号到来时进入 TAG_CHECK 状态，表示已经从主存中读取完毕，数据可用。

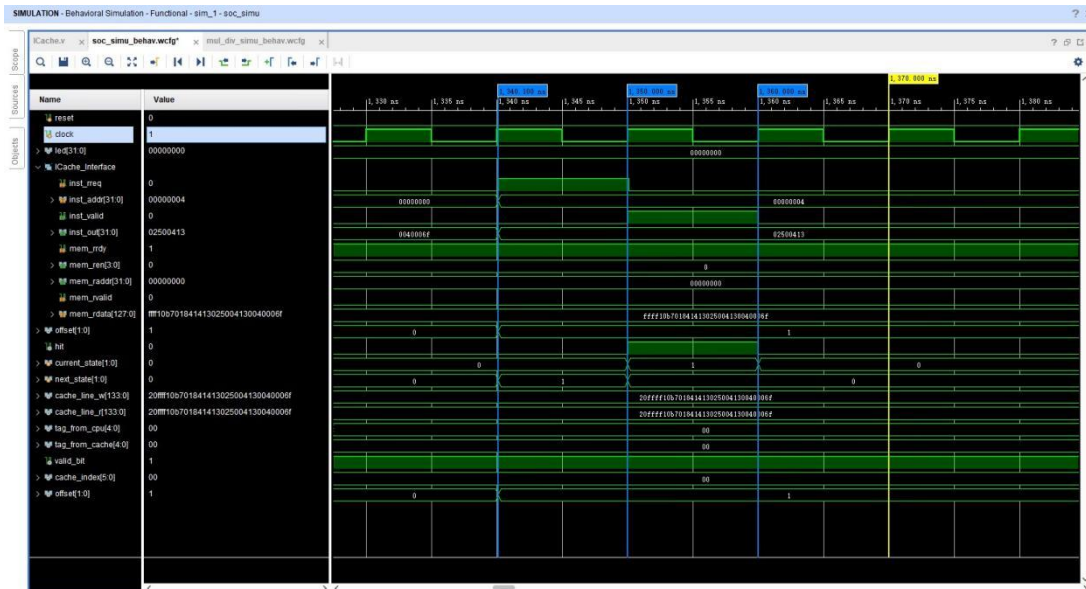
否则（`mem_rvalid` 为低电平时），次态 `next_state` 仍为 REFILL，下一个时钟上升沿信号到来时 ICache 仍将处于 REFILL 状态，继续等待读取主存任务的完成。

2、调试报告

要求：结合仿真波形截图对 ICache 作详细的时序分析，要求包含读命中、读缺失 2 种情形，且每种情形列举 2 个测试用例。**若完成了附加题，则需额外给出 DCache 的仿真波形截图及其详细文字分析，要求包含写命中、写缺失和 Uncached 访问 3 种情形。*

(1) 读命中

①读命中测试用例 1 分析（取指地址：0x00000004，应取指令：0x02500413）



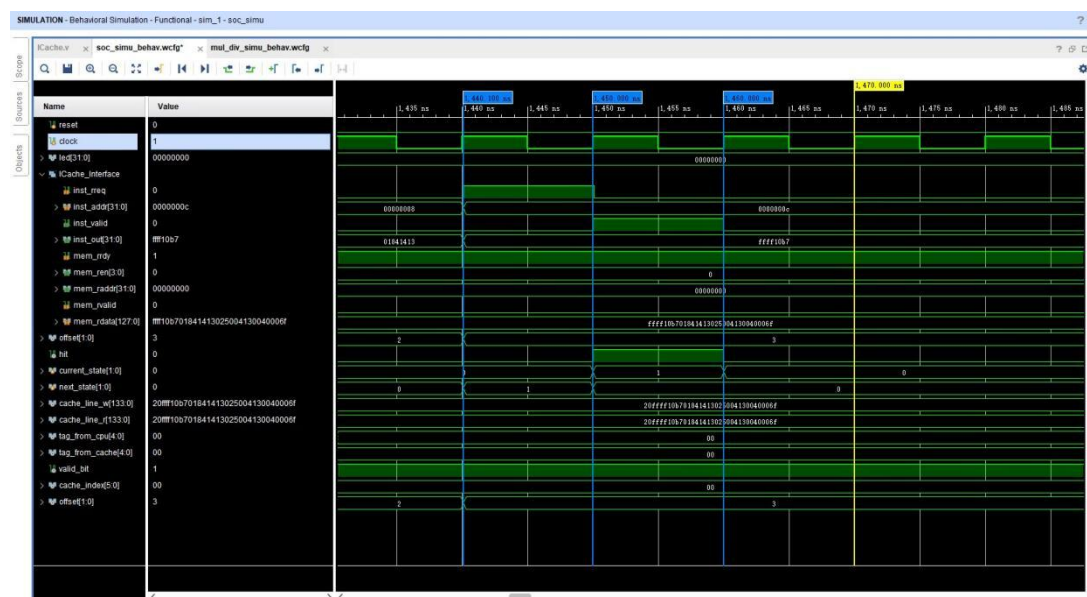
1340.100ns 时，CPU 发来取值请求，inst_req 信号变为高电平，进行读操作，并且传入来自 CPU 的取指地址 inst_addr: 00000004，由此将得到的偏移量 offset 更新为 1 (offset = inst_addr[3:2])。同时，根据 offset 和从 ICache 读出的 Cache 行 cache_line_r 的数据将输出给 CPU 的指令 inst_out 置为 02500413（此种情况下 inst_out = cache_line_r[63:32]），与预期相同，取指正确。

1350.000ns 时，时钟上升沿到来，hit 信号更新为高电平，inst_valid 信号也因此变为高电平，说明读命中。current_state 由 0 更新为 1，状态机现态变为 TAG_CHECK。

1350.100ns 时，inst_rreq 信号恢复低电平。

1360.000ns 时，时钟上升沿到来，hit 信号恢复低电平，inst_valid 信号也因此恢复为低电平。current_state 更新为 0，状态机现态恢复为初始的 IDLE。inst_rreq 信号也已经恢复低电平，此轮读数据操作结束，状态机次态 next_state = 0（即 IDLE 状态）。

②读命中测试用例 2 分析（取指地址：0x0000000c，应取指令：0xffff10b7）



1440.100ns 时，CPU 发来取值请求，`inst_req` 信号变为高电平，进行读操作，并且传入来自 CPU 的取指地址 `inst_addr`: 0000000c，由此将得到的偏移量 `offset` 更新为 3 (`offset = inst_addr[3:2]`)。同时，根据 `offset` 和从 ICache 读出的 Cache 行 `cache_line_r` 的数据将输出给 CPU 的指令 `inst_out` 置为 ffff10b7（此种情况下 `inst_out = cache_line_r[127:96]`），与预期相同，取指正确。

1450.000ns 时，时钟上升沿到来，`hit` 信号更新为高电平，`inst_valid` 信号也因此变为高电平，说明读命中。`current_state` 由 0 更新为 1，状态机现态变为 TAG_CHECK。

1450.100ns 时，`inst_req` 信号恢复低电平。

1460.000ns 时，时钟上升沿到来，`hit` 信号恢复低电平，`inst_valid` 信号也因此恢复为低电平。`current_state` 更新为 0，状态机现态恢复为初始的 IDLE。`inst_req` 信号也已经恢复低电平，此轮读数据操作结束，状态机次态 `next_state` = 0（即 IDLE 状态）。

(2) 读缺失

①读缺失测试用例 1 分析（取指地址：0x00000290，应取指令：0x00000093）

其总体波形如下：



具体分析:

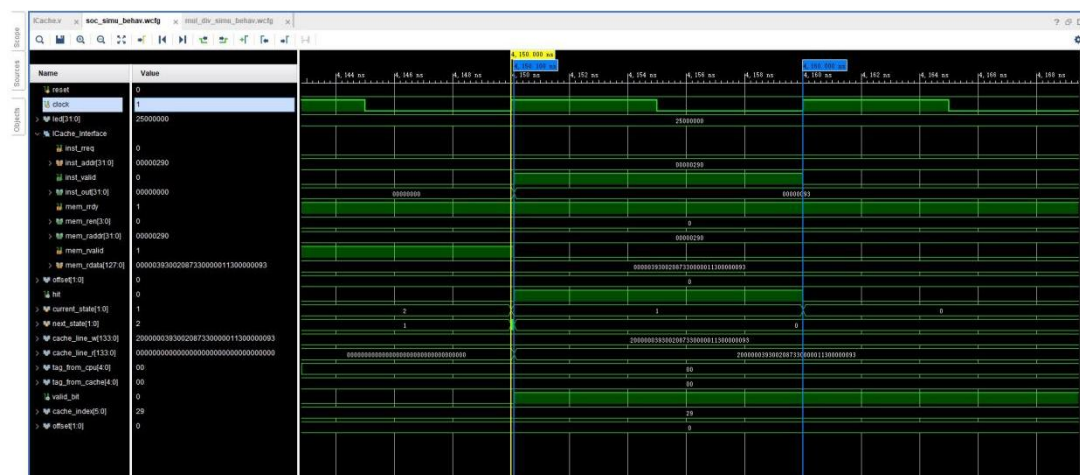


2910.100ns 时, CPU 发来取值请求, inst_req 信号变为高电平, 进行读操作, 并且传来自 CPU 的取指地址 inst_addr: 00000290, 由此将得到的偏移量 offset 更新为 0 (offset = inst_addr[3:2])。同时, 根据 offset 和从 ICache 读出的 Cache 行 cache_line_r 的数据将输出给 CPU 的指令 inst_out 置为 0080a023 (此种情况下 inst_out = cache_line_r[31:0])。cache_index 更新为 29。

2920.000ns 时, 时钟上升沿到来, hit 信号仍为低电平, inst_valid 信号也因此维持低电平, 说明读缺失, 此情况下 cache_line_r 的数据恢复为 0。current_state 由 0 更新为 1, 状态机现态变为 TAG_CHECK。由于读缺失, next_state 更新为 2, 状态机次态为 REFILL。

2920.100ns 时, inst_req 信号恢复低电平。由于读缺失, inst_out 在此情况下无作用, 恢复为 00000000。

2930.000ns 时, 时钟上升沿到来, 由于读缺失, mem_ren 由 0 变为 f, 标志需要从主存中读取数据。同时, current_state 更新为 2, 状态机进入 REFILL 状态。mem_raddr 接收来自 inst_addr 的数据 (mem_raddr = {inst_addr[31:4], 4'b0000}) 作为输出给主存的读地址。



从 2930.000ns 至 4150.000ns 期间, current_state 始终为 2, 状态机一直处于 REFILL 状态, 在此期间未能完成读操作。

4150.000ns 时, 时钟上升沿到来, current_state 由 2 变为 1, 状态机现态变为 TAG_CHECK。

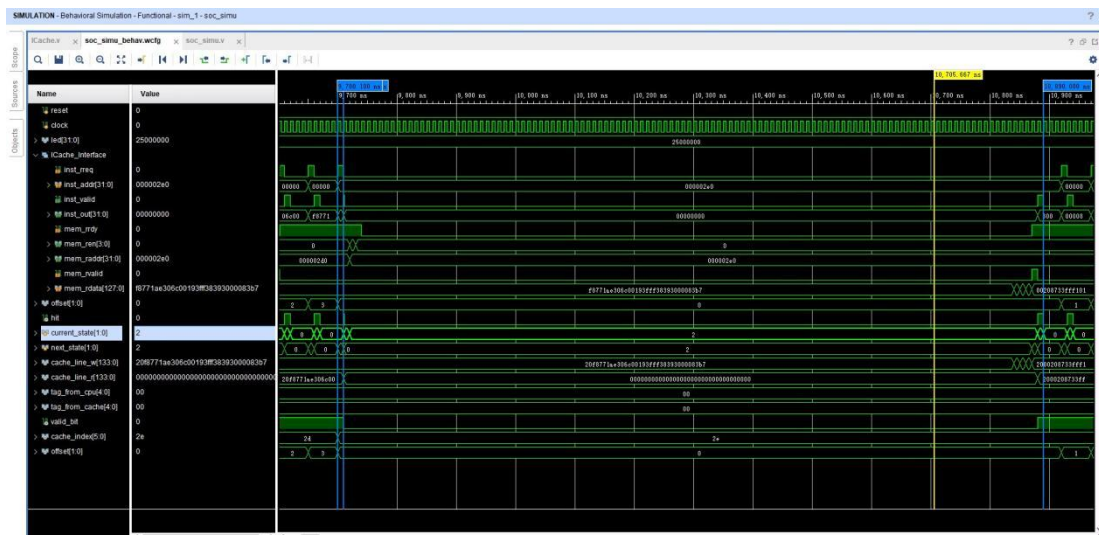
4150.100ns 时, hit 信号更新为高电平, inst_valid 信号也因此变为高电平, 成功从 cache

中读取数据，cache_line_r 的数据相应地更新，输出给 CPU 的指令 inst_out 因此更新为 00000093（offset 保持为 0，inst_out = cache_line_r[31:0]），与预期相同，取指正确。

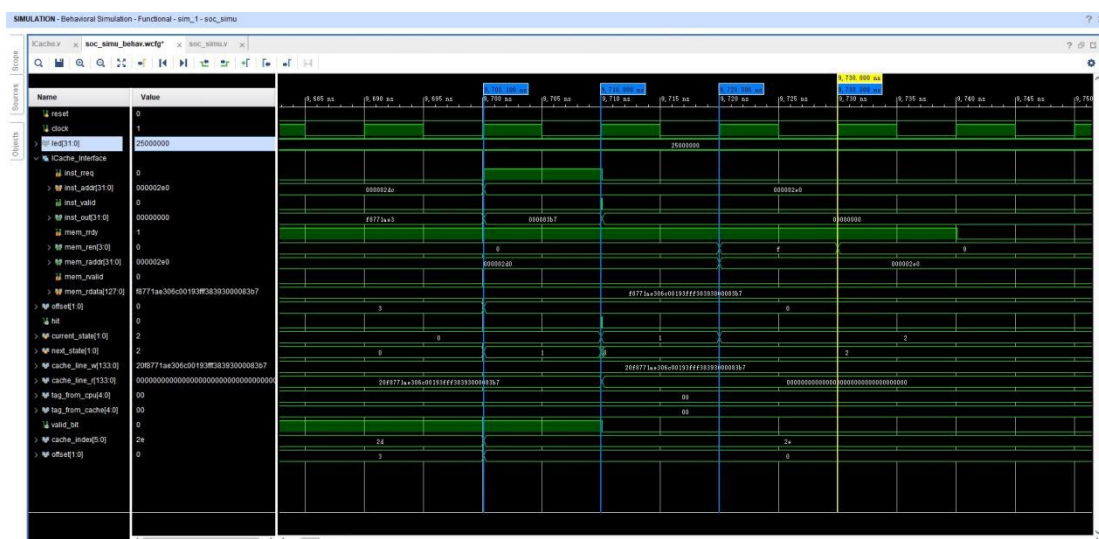
4160.000ns 时，时钟上升沿到来，hit 信号恢复低电平，inst_valid 信号也因此恢复为低电平。current_state 更新为 0，状态机现态恢复为初始的 IDLE。inst_rreq 信号也已经恢复低电平，此轮读数据操作结束，状态机次态 next_state = 0（即 IDLE 状态）。

②读缺失测试用例 2 分析（取指地址：0x000002e0，应取指令：0x800000b7）

其总体波形如下：



具体分析：



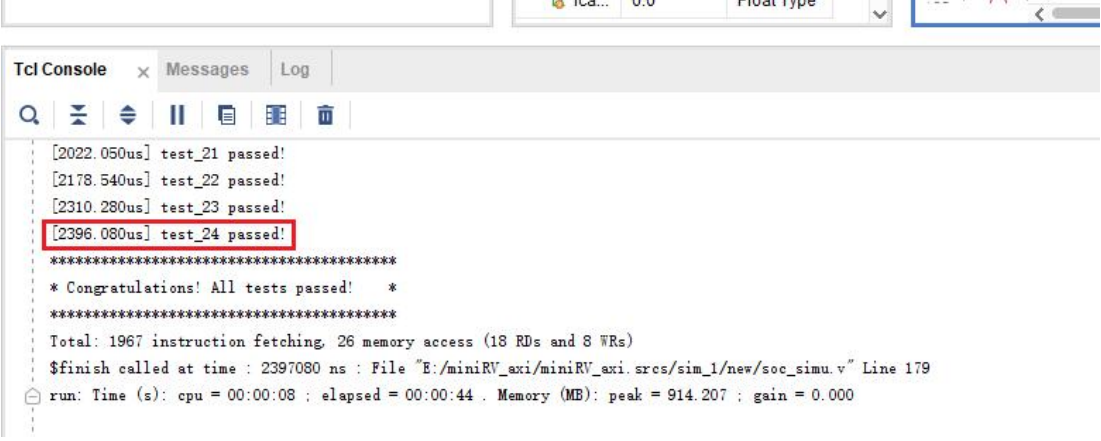
9700.100ns 时，CPU 发来取值请求，inst_rreq 信号变为高电平，进行读操作，并且传入来自 CPU 的取指地址 inst_addr: 000002e0，由此将得到的偏移量 offset 更新为 0（offset = inst_addr[3:2]）。同时，根据 offset 和从 ICache 读出的 Cache 行 cache_line_r 的数据将输出给 CPU 的指令 inst_out 置为 000083b7（此种情况下 inst_out = cache_line_r[31:0]）。cache_index 更新为 2e。

9710.000ns 时，时钟上升沿到来，hit 信号仍为低电平，inst_valid 信号也因此维持低电平，说明读缺失，此情况下 cache_line_r 的数据恢复为 0。current_state 由 0 更新为 1，状态机现态变为 TAG_CHECK。由于读缺失，next_state 更新为 2，状态机次态为 REFILL。

3、思考与讨论

(1) 分别给出无 ICache 时和有 ICache 时, SoC 运行测试程序的总时间的截图, 并谈谈你对该测试结果的理解。

无 ICache 时, SoC 运行测试程序的总时间的截图如下: (2396.080us)



```

Tcl Console x Messages Log
[2022.050us] test_21 passed!
[2178.540us] test_22 passed!
[2310.280us] test_23 passed!
[2396.080us] test_24 passed!
*****
* Congratulations! All tests passed! *
*****
Total: 1967 instruction fetching, 26 memory access (18 RDs and 8 WRs)
$finish called at time : 2397080 ns : File "E:/miniRV_axi/miniRV_axi.srcs/sim_1/new/soc_simu.v" Line 179
run: Time (s): cpu = 00:00:08 ; elapsed = 00:00:44 . Memory (MB): peak = 914.207 ; gain = 0.000

```

有 ICache 时, SoC 运行测试程序的总时间的截图如下: (607.340us)



```

Tcl Console x Messages Log
实际取到的指令: 0x0080a023
本次取指消耗的时钟数: 1
----- test 24 -----
[607.340us] test_24 passed!
*****
* Congratulations! All tests passed! *
*****
Total: 1967 instruction fetching, 26 memory access (18 RDs and 8 WRs)
ICache hit rate: 79.004%
$finish called at time : 608340 ns : File "E:/miniRV_axi/miniRV_axi.srcs/sim_1/new/soc_simu.v" Line 179
run: Time (s): cpu = 00:00:05 ; elapsed = 00:00:12 . Memory (MB): peak = 913.461 ; gain = 0.000

```

如图, 无 ICache 时 SoC 运行测试程序的总时间为 2396.080us, 而有 ICache 时 SoC 运行测试程序的总时间为 607.340us。

显然, 有 ICache 时的运行时间显著少于无 ICache 时的运行时间。这是因为无 ICache 时, 每条指令都需要从主存中读取, 指令执行时间长; 而有 ICache 时, 缓存 (ICache) 能够存储最近使用的数据, 访问 ICache 的延迟比访问主存的延迟要低得多, 大部分指令都可以直接快速地从 ICache 中读取, 只有少部分未命中的指令才需要从主存中读取, 这显著降低了访问延迟, 提高了指令执行速度, 减少了运行测试程序所需的总时间。

此外, 由于缓存 (ICache) 减少了数据访问时间, CPU 能够更快地获取所需的数据并继续执行指令, CPU 的空闲等待时间因此减少, 运行测试程序所需的总时间也减少了。

总之, 使用 ICache 显著减少了访问延迟, 在有 ICache (缓存) 的情况下, 系统能更快地访问指令, 这显著减少了运行测试程序所需的总时间。

(2) 给出你的 ICache 命中率的截图，并尝试分析如何提高 ICache 命中率。

ICache 命中率的截图如下：



由图可知当前的 ICache 命中率为 79.004%，已经处于较高状态，但这也意味着有约 20.996% 的指令访问会导致缓存未命中。未命中会引发缓存回填（即 ICache 进入 REFILL 状态），从主存读取数据，导致性能下降（如 CPU 等待时间增加等）。

若要提高 ICache 命中率，可从以下几方面进行改进：

①增加缓存的关联度。本次实验中要求主存地址与 Cache 地址的映射关系为直接映射，可以将当前的映射关系改为组相联映射（如改为 2 路或 4 路组相联）。理论上全相联缓存具有最高的命中率，但其硬件实现复杂，故在此选择组相联缓存；

②选用更智能的替换策略。最近最少使用（LRU）、随机替换等替换策略可以保留更有可能被再次访问的数据块，从而提高缓存命中率；

③使用预取。预取可以隐藏存储延迟，在数据实际被访问之前将其加载到缓存中，从而减少为命中次数。在本实验中可以使用顺序预取策略——在缓存未命中时不仅加载当前请求的块，还提前加载下一块数据。

4、总结与反思

要求：总结完成本课程实验获得的收获，并给出合理的意见和建议。

计组实验进入尾声了，这一路收获颇多。上学期的数字逻辑设计实验结束后再没用过 Verilog 编程，以至于实验 2 刚开始时我打开 Vivado 时脑子一片空白，那时候关于 Verilog 的很多具体细节都忘了。幸好有实验指导书！实验须知里面关于 Verilog 的相关知识真的介绍得很详细也很全面，特别适合我这种人重拾丢失的记忆。回忆起 Verilog 的基础知识、意识到代码到底应该怎么写才正确才规范、从无到有一点点用 Verilog 实现实验要求的功能，一路上当然也会遇上困难，如实验 2，由于当时写出的代码时序有些混乱，虽侥幸通过了 8 位的测试，但当扩展到 32 位后就出现了问题，只盯着代码看了很久实在想不明白错哪了，最后还是在老师的提醒和帮助下打开仿真波形对照着一点点分析，这才终于发现了问题——我写的时序有问题。过程是曲折的，但这培养了我使用仿真波形分析从而找出代码中错误的能力，更意识到仿真波形不仅是写报告时要用，还对改正代码中的错误有莫大的帮助，学会分析仿真波形真的是个非常重要的技能。除了重拾 Verilog 相关知识、掌握根据分析仿真波形进行 debug 的能力，这门实验还帮我巩固了理论课上所学的知识（如除法原理、cache 映射），实验 1 也使我对于汇编语言有了更深的理解。

这门实验课有很多值得夸赞的地方：实验指导书写得特别好很全面很细致（特别是实验须知和每个实验的实验原理部分）、实验内容和理论课衔接得紧密、实验课时安排合理、课上讲解得详细等等，但我还是有一个小建议——可以建立一个共享文档集中解答一些出现频率较高的问题。