Lab3 Cache 设计

实验目的：

1.权衡 cache size 增大带来的命中率提升收益和存储资源电路面积的开销
2.权衡选择合适的组相连度（相连度增大 cache size 也会增大，但是冲突 miss 会减低）
3.体会使用复杂电路实现复杂替换策略带来的收益和简单替换策略的优势（有时候简单策略比复杂策略效果不差很多甚至可能更好）
4.理解写回法的优劣

实验时间安排：

Lab4 Cache 实验分为 2 周，分两个阶段，但**只需验收一次，阶段一和阶段二的最后验收的时间都在 cache 实验的第二周结束时**，并写一份实验报告
实验结束后的一周内需要提交 Lab3 的实验报告
实验方式：Vivado 自带的波形仿真工具

实验内容概述：

阶段一：理解我们提供的直接映射策略的 cache，将它修改为 N 路组相连的 cache，并通过我们提供的 cache 读写测试。
阶段二：使用阶段一编写的 N 路组相连 cache，正确运行我们提供的几个程序。
阶段三：对不同 cache 策略和参数进行性能和资源的测试评估，编写实验报告。
提示：在 cache 的第一周实验开始之前，由于有些学生还未验收 Lab2 的 CPU 代码，因此在此之前我们不能公布阶段二所需要的 CPU+cache 的代码。大家可以先使用公布出的资料进行阶段一的实验。

阶段一(35%)

我们提供：

1） 一个简单的直接映射、写回带写分配的 cache 代码(cache.sv)
2） 一个 python 脚本(generate_cache_tb.py)，该脚本用于生成不同规模的 testbench 文件。这些 testbench 文件对 cache 进行多次随机读写，最后从 cache 中读取数据并验证 cache 的正确性。同时，我们也提供一个已经生成的 testbench。
3） 一份文档《cache 编写指导.docx》，解读我们提供的 cache 的结构和时序，并简单的介绍 N 路组相连 cache 的修改建议。

要求：阅读并理解我们提供的简单 cache 的代码，将它修改为 N 路组相连的（要求组相连度使用宏定义可调）、写回并带写分配的 cache。要求实现 FIFO、LRU 两种替换策略。
验收要求：验收时，当场使用我们提供的 python 脚本生成一个新的 testbench，并对自己的 cache 进行验证（要求 FIFO 和 LRU 策略都要验证，并修改组相连度等参数进行多次验证），验证正确后向助教讲解你所编写的代码。

阶段二(15%)

我们提供:

1） 一个 CPU+简单的 cache。并具有 cache 缺失率统计的功能。(在仿真波形中查看缺失率)
2） 快速排序的汇编代码和二进制代码（大概进行 1000 个数的快速排序）

要求：要求在 cpu 中将原本的 cache 替换成你所编写的 N 路组相连的 cache，并要求能成功运行这个排序算法（所谓成功运行，是指运行后的结果符合预期）

实验报告(50%)

我们提供：更多的针对 cache 的汇编代码以及其对应的二进制代码。(待定)
使用我们提供的几个 benchmark 进行实验,鼓励自己编写更多的汇编 benchmark 进行测试，体会 cache size、组相连度、替换策略针对不同程序的优化效果，以及策略改变带来的电路面积的变化。针对不同程序，权衡性能和电路面积给出一个较优的 cache 参数和策略。其中“性能“参数使用运行 benchmark 时的时钟周期性进行评估,”资源占用“参数使用 vivado 或其它综合工具给出的综合报告进行评估。进行这一步时需要用阶段二的结果进行一些实验，不能仅仅进行理论分析，实验报告中需要给出实验结果（例如仿真波形的截图、vivado 综合报告等)。
提示：为了方便进行性能评估，建议用上阶段二中我们提供的缺失率统计功能

其他问题:

1. 暂时只做 dcache，icache 默认不缺失，仍然使用原有代码充当 icache
2. 在进行 cache 实验时，为了方便 Verilog 编写，一律不需要处理读写的独热码，只需考虑 sw 和 lw 这两种“整字读写“的指令。我们提供的相关 benchmark 中的所有 load、store 指令也将只有 sw 和 lw。
3. 我们的代码利用封装的 Bram 模拟 DDR，Cache 命中时间为 1 cycle，模拟 DDR 命中时间设置为 50 cycle（因为真实情况下 cache 命中时间为 1ns，DDR 为 50-100ns）