计算机组成原理实验指导书

马世禹

目录

实验三 直接相连 Cache 设计

1、实验目的

- 掌握直接相联 Cache 的基本结构及其设计方法。
- 实现直接相连地址映射的机制
- 实现读命中和读缺失两种情况的处理。

2、实验内容

为主存容量 8KB 的存储系统，设计直接相联 Cache，参数如下：

- Cache 的总容量为 512B
- 每块 4 个字节
- 包括 128 个 Cache 块

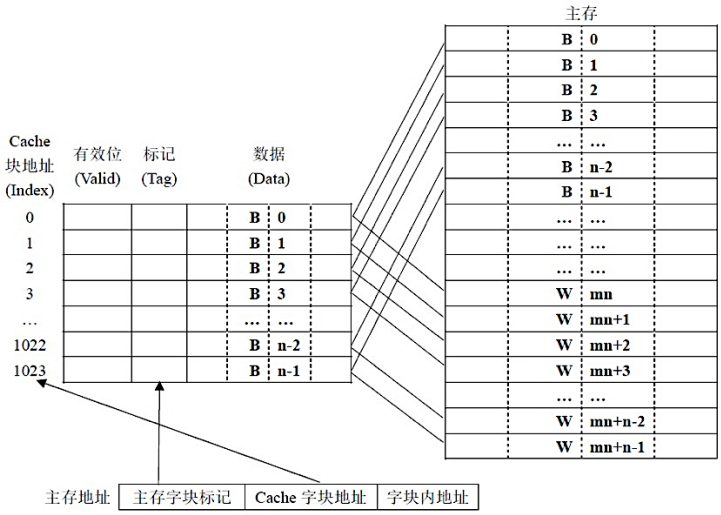
实现在 CPU 发出读操作请求后的取数据操作，包括读命中和读缺失两种情况。

3、实验原理

3.1 Cache 的结构

Cache 由存储体和控制机构两大主要部分构成。

Cache 的存储体

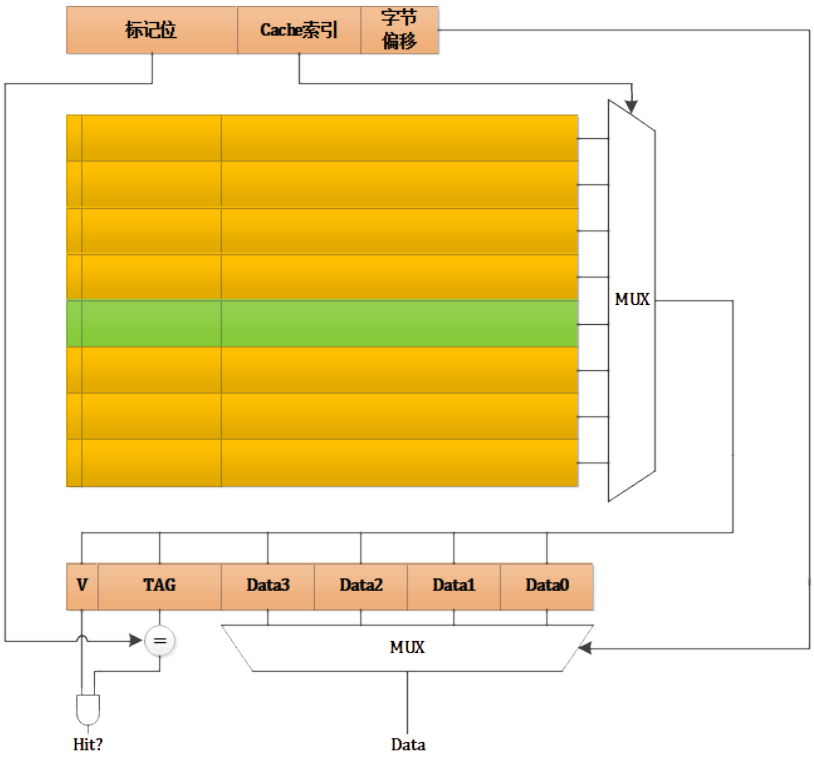


- 有效位（Valid）：标识该行内存储的数据是否有效，未装入任何数据之前无效。
- 标记（Tag）：用以唯一地标识某个地址，判断是否访问命中。
- 数据域（Data）：连续存储的 k 个相邻的在主存中连续的数据。

映射机制

如上所示的地址映射机制，低 2 位为字块内偏移，代表一个 Cache 行存储有 4 个字节。中间 10 位为 Cache 字块地址，代表 Cache 有 1024 个 Cache 行。高 20 位为 Tag，用以唯一地标识某个主存地址，与 Cache 中的标记位比对，判断是否访问命中。

地址分解



例如，我们系统的主存容量为 256KB，则主存地址宽度为 18 位；我们设计 Cache 每个字块中存储 4 个字节，则每个块中需要用 2 位去寻址相应的字节；Cache 容量为 256 行（256*4B=1024B），且设计的方式是直接相联，则 Cache 字块地址为 8 位；剩余的 18-8-2=8 位，则作为 Tag 使用。此时，我们即完成了地址的分解工作。

如上图所示，一个典型的直接映射 Cache 访问主要完成几项工作：

- Cache 寻址：根据主存地址中的 Cache 索引，选出对应的 Cache 块（行）
- 标记比对：主存中的标记位和取出行中的标记位对比，加上 Valid 位，判断是否命中
- 选出数据：如果命中，根据地址的低位偏移，在读出的 Cache Line 的数据域中，选择相应的数据，回送给主设备，并采取相应的机制通知主设备，在本次实验中，我们要求拉高 Cache 模块的 hit 信号，同时将数据输出到 data_out 上。
- 缺失处理：如果不命中的话，还需要进行缺失的处理，在下一部分将会提到。

3.2 控制机构

缺失处理机制

当 Cache 的访问发生缺失时，应要有相应的机制，确定缺失的位置，并向下一级存储器发出读字块的请求。

缺失动作

在缺失时，Cache 向下一级存储器（本实验中为 mem_wrap）发起读请求，请求读取 4 个连续存储的字，待下级存储器响应后，将返回的数据采集、打上相应的标签，然后写入自身的存储体中。

实现原理

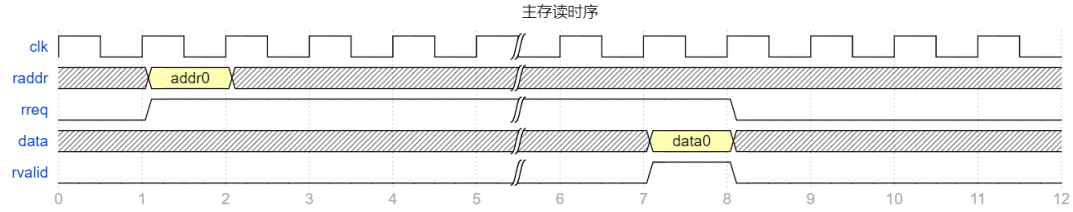
在时序电路中，如果要执行顺序发生的动作，并带有条件跳转，状态机是通用的做法。我们使用状态机生成控制信号，对数据通路进行操纵，从而达到处理各种情况的目的。

4、实验准备

4.1 熟悉存储系统模型

本次实验采用的存储系统大小为 8K，地址宽度为 13，字长为 1 字节。

存储器模块为 mem_wrap.v，为简化实验流程，我们已将 Cache 缺失时连续读取 4 个字并拼接的工作完成，具体时序图如下：



在主存空闲阶段，Cache 将地址放在 raddr 口上，并将 rreq 信号拉高，代表一次的读请求。等待若干个周期，主存将返回 4 个字拼接而成的数据，并将 rvalid 信号拉高，代表数据已经准备好，此时可以取走数据。

4.2 项目框架概览

■ 设计文件

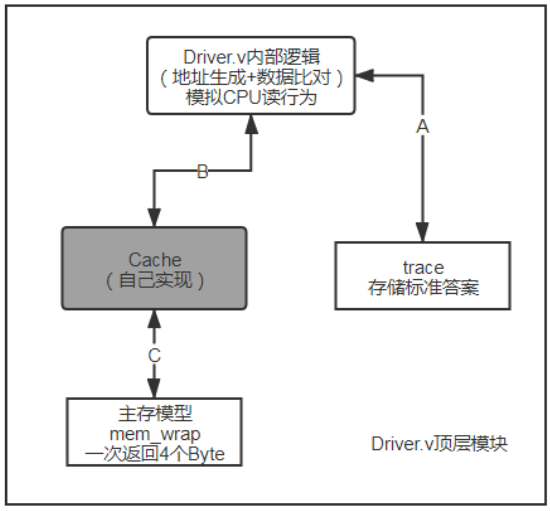
driver (driver.sv) ----- 顶层模块，模拟 CPU 的访存行为，并进行数据正确性检查，可以看作是 CPU

trace (IP 核) ----- 存储标准答案

cache (cache.v) ----- Cache 模块（需要完成）

mem_wrap(mem_wrap.v) ----- 主存存储器模型

模块间的关系如下图所示：



■ 仿真文件

使用说明

完成设计后，可以运行 all_sim.v 文件中的仿真，运行仿真时，下方的 Tcl Console 也会打印相应的调试信息，帮助你定位出错点。

如果测试全部通过，控制台会显示相应字样，同时仿真将会停止在对应位置。

```
34      if(g0.next_state == 8'b0000_0001)      $display("该地址测试正确，将测试下一个地址\n -
35  end
36  always begin
37      #10 clk = ~clk;
38  end
39  always @(posedge clk)   begin
40      if(test_success)    begin
41          [redacted]
42          [redacted]
43          [redacted]
44          $display("======================测试全部通过=========================");
45          $stop;
```

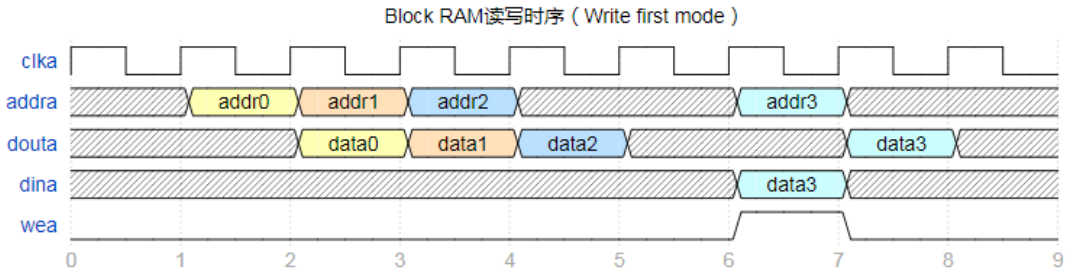
4.3 模块接口规范

属性	名称	含义	位宽
输入	clk	时钟	1
输入	reset	复位（高电平有效）	1
输入	raddr_from_cpu	CPU 的读地址	13
输入	rreq_from_cpu	CPU 的读请求	1
输出	rdata_to_cpu	Cache 读出的数据	8
输出	hit_to_cpu	命中标记	1
输入	rdata_from_mem	主存模块读取的连续 4 字节	32
输入	rvalid_from_mem	主存读取完毕标记	1
输出	rreq_to_mem	读主存请求	1
输出	raddr_to_mem	读主存首地址	13

4.4 Cache 模块时序规范

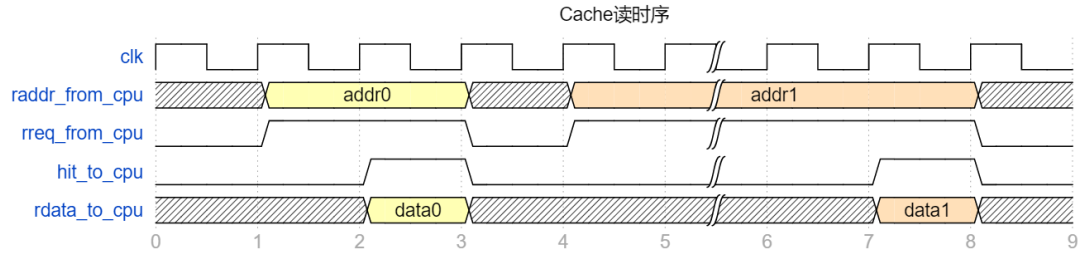
Cache 存储体调用 Block RAM 的 IP 核实现，因此，我们需要了解 Block RAM 的读时序。IP 核的使用见后文附录。

Block RAM 的读写时序：



- 读时序：上一周期给出地址，下一周期输出数据，可连续读取。（周期 1，2，3）
- 写时序：上一周期给数据、写地址，拉高 wea 信号，下一周期成功写入数据，刚刚写入的数据出现在 douta 口上。

Cache 的读时序



下面以 Cyc# 代表周期号，详细叙述 Cache 和 CPU 之间的通信约定。

- CPU 发来 rreq 信号，同时把地址放在 raddr 上(Cyc #1)，代表启动一次读取，在得到 Cache 的 hit 响应(Cyc #2)之前，CPU 会保证：rreq_from_cpu 信号不会

撤下地址线 raddr_from_cpu 上的地址不会改变.

- 在得到 Cache 的 hit 响应之后(Cyc #2)，CPU 会保证：在 hit 信号到来的下一个周期(Cyc #3)，rreq 信号马上撤下 Cache 需要做到：hit 响应信号和读出的数据只需持续一个周期(Cyc #2 - Cyc #3)，同时有效
- CPU 未发 rreq 信号的时候，Cache 需要做到：hit 信号始终为 0，不得置高，数据输出可以是任意值

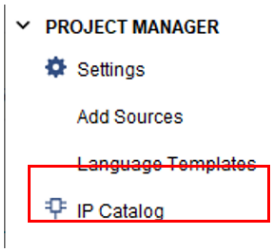
5、实验步骤

- 进行地址各个字段的分解
- 计算 Cache 各个参数，选择大小合理的 Block RAM，创建 IP 核并实例化
- 编写命中判断的组合逻辑
- 设计控制机制的状态机（不局限于 Moore 或 Mealy，也不局限于我们提供的几个状态，可以自由发挥）
- 自己编写 Testbench，验证相关功能正确性。
- 使用提供的 all_sim.v 测试模块进行测试、调试，根据输出的调试信息，定位错误点。

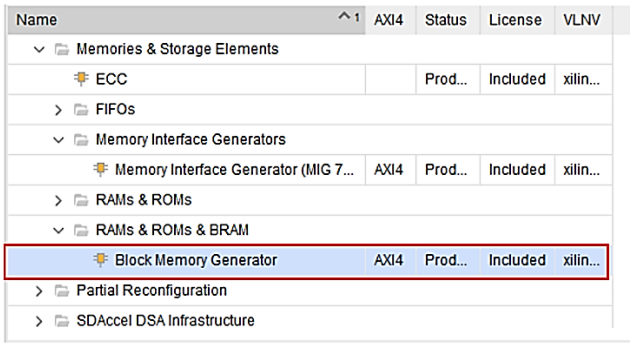
附录 1 IP 核的使用步骤

- **加入 IP：Block memory generator**

点击 Flow Navigator 中的 IP Catalog，打开窗口添加 IP 核。



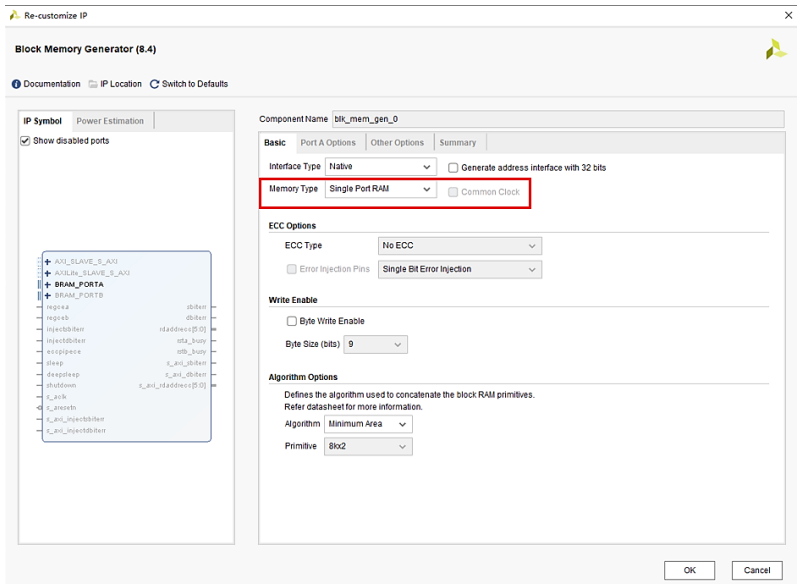
在 Memories & Strorage Elements –> RAM & ROMs & BRAM 文件夹，双击 Block Memory Generator。

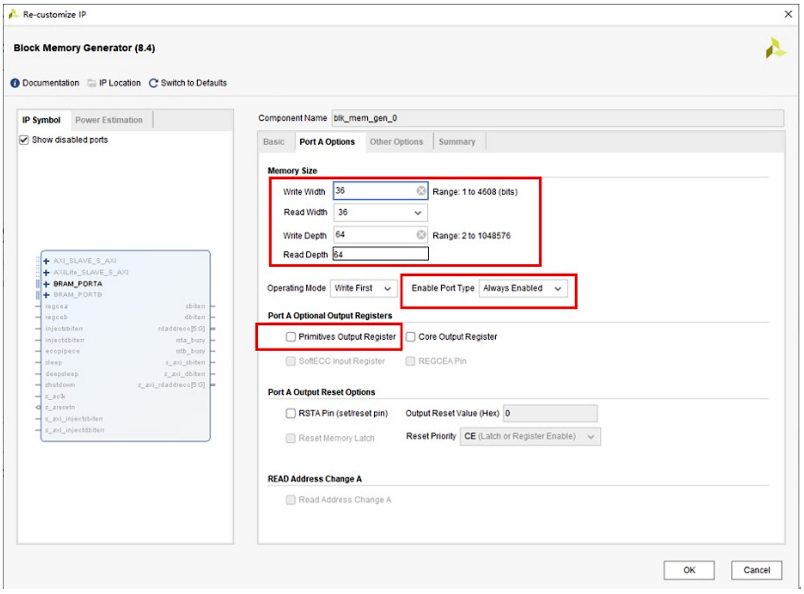


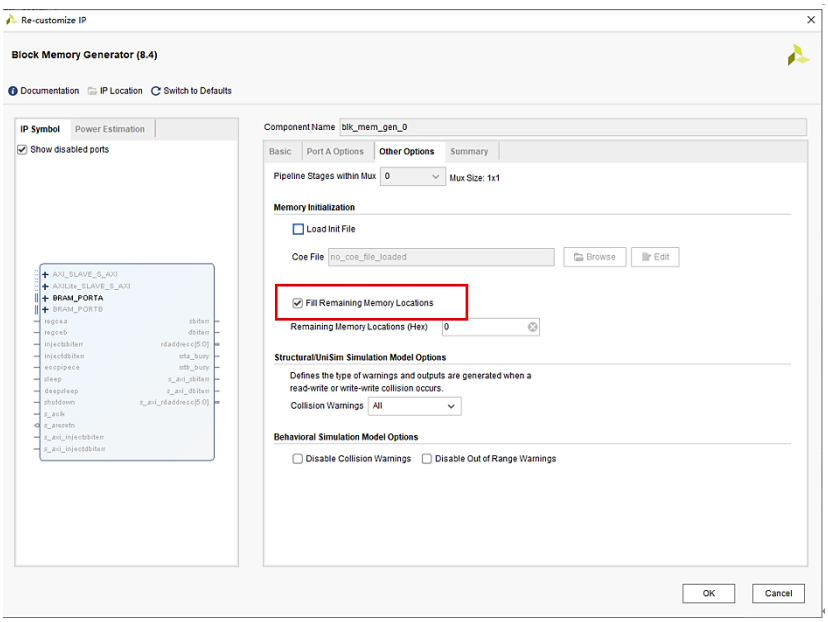
- **点击 Basic 选项卡配置 RAM 为简单单口 RAM**

点击 Port A Options 设置 Write Width 为 36，Read Width 为 36，Write Depth 为

64，Read Depth 为 64，端口使能选择 Always Enable，去掉勾选 Primitives Output Register。点击 Other Options，勾选 Fill Remaining Memory Locations。点击 OK 等待片刻即可生成 IP。

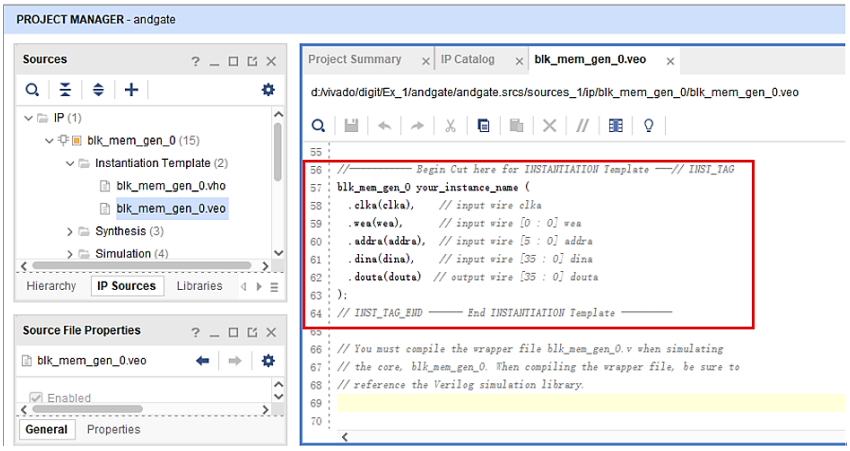






■ **例化 IP**

生成好 IP 后，点击 IP Sources 展开 IP，进入 Instantiation Template 下的 blk_mem_gen_0.veo 文件下复制实例化模板到自己的设计文件下将此 IP 实例化。

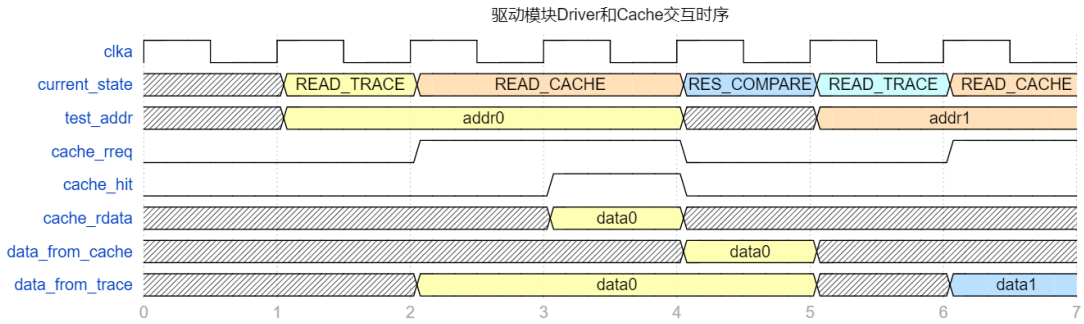


附录 2 Driver 模块设计说明

模拟 CPU 的读主存行为，与主存直接连接，与实现的 Cache 也直接连接。该模块生成一系列的测试地址，在对每一个测试地址进行测试时，先从主存中取出相应的数据(READ_TRACE)，然后再对 Cache 提出访问请求(READ_CACHE)，待 Cache 的数据可用后，将之前取出的数据和 Cache 给的数据进行比较(RES_COMPARE)，如果正确，则进行下一个地址的测试。如果错误，则会停留在出错的位置(TEST_FAIL)。

上层测试模块模拟 CPU 的取指令行为，其流程（状态转移）如下：

在对一个地址进行测试时，首先进入 READ_TRACE 状态（周期 1），一个周期后，trace 内读出的数据锁存到了 data_from_trace 寄存器中（周期 2），并进入 READ_CACHE（周期 2-3）状态，此时 Cache_rreq 信号拉高，CPU 向 Cache 发起读请求。



如图所示，可以看到，在 hit 信号到来的时候（周期 3），上层的 CPU 取走了数据(周期 4：data_from_Cache 发生了改变)，并转移到 RES_COMPARE 状态，对数据的正确性进行比较，如果正确，则进行下一轮的测试。