# 实验报告模板

华南理工大学

《计算机组成原理》课程实验报告

实验题目： 存储器

姓名： 陈映松 学号： 202130442563

班级： 21计科(2)班 组别：

合作者：

指导教师： 张齐

|  |
| --- |
| **实验概述** |
| 【实验目的及要求】  实验目的：   1. 掌握存储器扩展方法和存储器与CPU连接的设计技巧 2. 理解堆栈与FIFO的电路特点及两者特点 3. 了解cache的三种映射方式及其电路   实验要求：   1. 学会如何进行存储器容量扩展。 2. 学会如何进行存储器与CPU连接。 3. 学习如何使用堆栈，FIFO，Cache结构组织。   【实验环境】  Proteus 8.1  Digiblock  Java配置环境 |
| **实验内容**  **一、实验步骤：**  **实验1：存储器容量扩展**  (1) 存储器容量扩展有位扩展和字扩展两种形式，分别如下图 3.1.1 左和右所示。存储器  位扩展是从字长方向扩展，CPU 的地址线 A、片选信号 CS 分别连在一起，芯片的数据线 D  分别对应于所搭建存储器的高若干位和低若干位；存储器字扩展则从字数方向扩展，CPU  的数据线 D 连在一起，存储器地址线的低若干位（如下图右的 A0~A17）连接各芯片的地址  线；高若干位（如下图右的 A20~A18）通过译码器生成各芯片的片选信号 CS。  (2) 与上图 3.1.1 对应的存储器容量位扩展电路（4×8 位 ROM 芯片◊4×16 位存储器）和  字扩展电路（8×8 位 ROM 芯片◊32×8 位存储器）分别如下图 3.1.2 左和右所示。图中的  Test 测试用例可以通过设置存储器 Addr 输入端口/Data 输出端口的数据为 ROM 赋值，但是是任何时刻只有一个 ROM 有效。因此，同时多个 ROM 不能通过 Test 测试用例赋值。注意:所有端口和元件（包括存储器）都必须命名！否则转换 Verilog 代码会报错。  **(3)** 根据图 3.1.2 的存储器容量位扩展和字扩展电路图，设置输入端的相应数据或控制相  应输入端的电平；观察输出端的电平或数据，并将结果填入表 3.1 中  **实验2：存储器与CPU连接**   1. 存储器与 CPU 连接的方法并不唯一，一般先根据 CPU 芯片提供的地址线数目确定   CPU 访问存储器的地址范围，并写出相应的二进制地址码，如下图 3.2.1 所示。  **(2)** 根据上图 3.2.1 的 CPU 地址空间范围，分配 CPU 地址线的低位直接连接存储芯片的  地址线，CPU 地址线的高位形成存储芯片的片选信号，然后连接数据线、读写控制线（R/W#）87以及 MREQ#信号线（地址译码使能信号）。注意：不允许多个存储器的输出端口直接连在一起，必须使用选择器，否则转换 Verilog 代码会报错。  **(3)** 根据图 3.2.2 的 CPU 与存储器连接电路图，设置输入端的相应数据或控制相应输入端  的电平；观察输出端的电平或数据，并将结果填入表 3.2 中  **实验3：堆栈**  **(1)** 堆栈是一种隐含地址的先进后出 (FILO) 存储电路，最简单的位堆栈（4bit 容量）如  下图 3.3.1 所示：的位堆栈，端口 push/#pop=1，则 clock 上升沿时刻，输入端口 D 的值入栈，stack 右移(→Q3Q2Q1Q0)；端口 push/#pop=0，则 clock 上升沿时刻，输出端口 S 显示出栈的值，stack 左移(←Q3Q2Q1Q0)。注意：该堆栈是否已满，由软件判断。  **(2)** 字节堆栈（4byte 容量）如下图 3.3.2 所示，分别展示了由寄存器堆元件（图 3.3.2 左  下方）和独立寄存器电路构成的字节堆栈。若 push/#pop=1，则 clock 上升沿时刻，输入端口 Data 的值推入堆栈：→stack0/1/2/3 ，堆栈指针 sp+1；若 push/#pop=0，则 clock 上升沿时刻，堆栈指针 sp-1，输出端口 Out 和 Dout 分别弹出各自堆栈的值：stack0/1/2/3→。同样,这两个堆栈是否已满，由软件判断。  **(3)** 根据图 3.2.2 的 4byte 字节堆栈图，依据上述的描述，设置输入端的相应数据或控制  相应输入端的电平；观察输出端的电平或数据，并将结果填入表 3.3 中  **实验4：FIFO**  **(1)** 与堆栈(FILO)电路不同，FIFO 电路的特点是“先进先出”。其中，同步 FIFO（输入  和输出端共用时钟信号）又称为队列，其电路如下图 3.4.1 所示：端口 Enq=1，则 clock 上  升沿，数据 data 写入队列，且写指针 tp+1；端口#Deq=0，则 clock 上升沿，数据 data 读出队列，且读指针 hp+1。写指针 tp 和读指针 hp 均由统一的时钟信号 clk 生成。  **(2)** 异步 FIFO 的原理图如下图 3.4.2 所示，一般用于两个独立的时钟系统的接口。因为  异步 FIFO 输入端的写时钟和输出端的读时钟不同步，读写操作是各自独立的。所以异步  FIFO 考虑的情况比较多，可能会出现读操作已经把 FIFO 中的数据读完（即 FIFO“空”状  态）和写操作已经填满 FIFO（即 FIFO“满”状态）的情况。  **(3)** 异步 FIFO 电路如下图 3.4.3 所示：clock 上升沿，若 WEN=1，则数据 data 写入，同时写指针 wp+1；若 REN=1， 则数据 data 读出 FIFO， 同时读指针 rp+1。写指针 wp 和读90指针 rp 分别由各自独立的写时钟 clk\_w 和读时钟 clk\_r 生成。注意：异步 FIFO 满或空的状态必须由硬件电路判断（端口信号 full 和 empty）！  **(4)**根据图 3.4.2 的异步 FIFO 电路图，设置输入端的相应数据或控制相应输入端的电平；  观察输出端的电平或数据，并将结果填入表 3.4 中  **实验5：Cache**  **(1)** 如下图 3.5.1 左所示，Cache 是位于 CPU 和主存之间的高速小容量的存储器，一般由  SRAM 构成，主要用来弥补 CPU 和主存之间速度差异，提高 CPU 访问主存平均速度。设置cache 的理论基础是程序访问的局部性原理，即 CPU 执行程序所使用的存储单元相对集中或小批簇聚于相邻单元中。当 CPU 读取主存中一个字时，首先发出此字的内存地址到 Cache和主存。此时 Cache 控制逻辑依据地址判断此字当前是否在 cache 中：若是，此字立即传送给 CPU；若非，则用主存读周期把此字从主存读出送到 CPU，与此同时，把含有这个字的整个数据块从主存读出送到 Cache 中。因为需要根据主存地址判断 cache 有无命中,所以并且按某种规则把主存地址定位到 cache 中，即地址映射。数据按照地址映射装入 cache 后，CPU 访存时，由 cache 控制逻辑按照地址映射将主存地址变换成 cache 地址。如下图 3.5.1右所示，地址映射方式有三种：全相联映射、直接映射和组相联映射。  **(2)** 如下图 3.5.2 所示，全相联映射方式：主存中一个块的地址与块的内容一起存于 cache  的行中，其中块地址存于 cache 行的标记部分。全相联映射是多对多的映射关系：对于主存的任何一块均可以映射到 Cache 的任何一行。全相联映射的优点是机制灵活，命中率高，  Cache 的行利用率高；缺点是速度较慢，成本较高，比较器电路难于设计和实现。因此，全  相联映射适合小容量的 cache。  **(3)** 全相联映射电路如下图所示，其读写数据时序如下：初始化时，1 次 clk 输入第一个地址，使能 EN=1; 根据地址 Addr 和 Cache 所有行的行地址逐一比较，判断是否命中：若命中，则从 Cache 输出数据， 1 次 clk 输入下一个地址，命中行计数器清 0，其余行计数器+1，结束；若不命中，则提示 miss=1 ，使能 Ready=1，然后连续 4 次 clk 令 Bulkdata 端口输出 4 字节数据块写入 Cache 中行计数器值最大的行（或多个最大行中行号 x 最大的行）同时， Ready=1 直接从 Bulkdata 端口输出地址 Addr 对应的数据，且在第一次 clk 时刻将未命中行的地址写入新的行地址 Lx，令 miss=0；使能 Ready=0，1 次 clk 输入下一个地址，新写入行计数器清 0，其余行计数器+1，结束下图 3.5.3 中，cache 每行四个字节，行地址 L0~7，行有效标志 V0~7，行命中标志 h0~7,行写入标志 w0~7，行计数器标志 C0~7(8 位: 0~255)；Vx=1 表示第 x 行有数据，hx=1 表示第 x 行命中，则 Cache 直接输出数据；若没有 hx 置位，则表示 cache 没有命中，所有计数器中最大值对应的行被替换（若计数器中有多个最大值，则其中行号 x 最高的行被替换)， 即 LRU 替换算法。clk 输入下一个地址，命中行或被替换行的计数器清 0，其余行计数器+1。然后，8 位地址 addr 的 2~7 位与所有行的行地址逐一比较，若第 x 行的行地址 Lx 与其相同，且 Vx=1（即 Lx 有效），则 hit=1 命中；否则，hit=0 未命中。若命中，Cache 直接输出地址 addr 指定的数据；若未命中，Cache 禁止读出。此时，使能 Rd=1，令 We=1，Cache92允许写入5位地址由写入行的行号Whit和计数器决定的最低两位拼接而成；Whit选择Vx=0的行中最大行号 x，若 Vx 皆置位，则选计数器值最大行的行号 Chit。We=1 使 Bulkdata 端口输出地址 Addr 对应的数据到端口 Data；同时，连续 4 次 clk 将 Bulkdata 端口的数据写入Cache。最后，若命中，行命中标志 hx=1；若未命中，使能 Rd=1，令 We=1，由写入行的行号Whit 行写入标志 wx=1，在第一次 clk 时刻将地址 addr 的 2~7 位保存的新地址 Dx 写入对应行的行地址 Lx，且令其行有效标志 Vx=1，行命中标志 hx=1。接着，使能 Rd=0，clk 输入下一个地址时，命中行或新写入行的计数器 Cx 清 0，其余行计数器+1。注意：计数器值最大行的行号 Chit 由所有行计数器值两两比较，选其中最大者的行号 x。  **(4)** 如下图 3.5.4 所示，直接映射方式：主存的第 i 块一定映射到 Cache 的第 j 行，j 是对i 求模 c 的同余，c 是 cache 的全部行数。直接映射是多对一映射关系，一个主存块只能拷贝到 cache 的一个特定行位置上去。直接映射的优点是映射方式简单，易实现；缺点是机制不灵活，Cache 命中率低。存在 Cache 有空行而不能存数据块的问题，即造成替换频繁，效率下降。因此，直接映射适合需要大容量 Cache 的场合（更多行数可以减少冲突机会）。  **(5)** 直接映射电路如下图 3.5.5 所示，其读写数据时序如下：初始化时，1 次 clk 输入第一个地址，使能 EN=1; 根据地址 Addr 和 Cache 特定行地址 Lx(行号 x 模 8 同余)比较，判断是否命中：若命中，则从 Cache 输出数据， 1 次 clk 输入下一个地址，结束；若不命中，提示93miss=1，使能 Ready=1，然后连续 4 次 clk 令 Bulkdata 端口输出的 4 字节数据块写入 Cache特定的行(行号 x 模 8 同余)；同时，Ready=1 直接从 Bulkdata 端口输出地址 Addr 对应的数据，且在第一次 clk 时刻将未命中行的地址写入新的行地址 Lx，令 miss=0；使能 Ready=0，1 次 clk 输入下一个地址，新写入行计数器清 0，其余行计数器+1，结束。下图 3.5.5 中，Cache 每行四个字节，行地址 L0~7（相联存储器），行有效标志 V0~7;Vx=1 表示第 x 行有载入数据。然后，8 位地址 addr，其中 2~4 位选择行号 x，且取出对应的行有效标志 Vx：若第 x行的行地址 Lx 与地址 addr 的 5~7 位相同，且 Vx=1（即 Lx 有效），则 hit=1 命中，否则 hit=0未命中；同时，地址 addr 的 2~4 位生成行写入标志 wx，而 5~7 位保存成新地址 Dx。若未命中，在第一次 clk 上升沿时刻，行写入标志 wx=1 使新地址 Dx 写入对应行的行地址 Lx。  最后，若命中，Cache 直接输出地址 addr 的 0~4 位指定的数据；若未命中，则 Cache  禁止读取。此时，若 Rd=1，则 We=1，Cache 允许写入，5 位地址由地址 addr 的 2~4 位（模8 同余的特定行号 x）和计数器决定的最低两位拼接而成。We=1 使 Bulkdata 端口输出地址Addr 对应的数据到端口 Data ；同时，连续 4 次 clk 将 Bulkdata 端口的数据写入 Cache。  **(6)**如下图 3.5.6 所示，组相联映射方式其实是全相联方式和直接映射方式的组合：将  Cache 的行分成 c 组，每组 r 行。根据直接映射方式，主存的字块只能存放到 Cache 中的某一个组，而映射到该组哪一行则是灵活的（组内全相联映射）。组相联映射的优点是增加了映射的灵活性，主存中的块可映射到 Cache 某一组内的任意 r 块之一的位置，提高了命中率.每次比较只进行 r 路比较，r 较小时,硬件开销不是很大（一般 r=2/4/8）。  **(7)** 组相联映射电路如下图 3.5.7 所示，其读写数据时序如下：初始化时，1 次 clk 输入第一个地址，使能 EN=1; 根据地址 Addr 和 Cache 特定组(组号模 4 同余)内的行地址比较，断是否命中；若命中，则从 Cache 直接输出数据， 1 次 clk 输入下一个地址，命中行计数器清 0，其余行计数器+1，返回，结束；若不命中，则则提示 miss=1 ，使能 Ready=1，然后连续 4 次 clk 令 Bulkdata 端口输出 4 字节数据块写入 Cache 特定组内的计数器值最大的行（或多个最大行中行号x 最大者）同时， Ready=1 直接从 Bulkdata 端口输出地址 Addr 对应的数据，且在第一次clk 时刻将未命中行的地址写入新的行地址 Lx，令 miss=0，结束。下图 3.5.7 中，cache 行间操作与直接映射相同，而行内操作与全相联映射相同，仅在组内判断命中和处理未命中。然后，8 位地址 addr，其中 3~4 位选择组号 set，5~7 位则与第 set 组内的行比较，若与其中某行地址 Lx 相同，且 Vx=1（即 Lx 有效），则 hit=1 命中，Cache 直接输出地址 addr指定的数据；否则，hit=0 未命中。则 Cache 禁止读出。此时，使能 Rd=1，令 We=1，Cache允许写入，5 位地址由写入行号 Whit 和计数器决定的最低两位拼接而成；Whit 由计数器值最大行号 Chit(第 set 组内行计数器值最大者的行号 x)和组号 set 拼接而成；We=1 使 Bulkdata端口输出地址 Addr 对应的数据到端口 Data，同时，连续 4 次 clk 将 Bulkdata 端口数据写入Cache。使能 Rd=0，clk 输入下一个地址时，命中行或新写入行的计数器 Cx 清 0，其余行计数器+1。  **(8)** 根据图 3.5.7 的组相联映射电路图，依据上述的描述，设置输入端的相应数据或控制  相应输入端的电平；观察输出端的电平或数据，并将结果填入表 3.5 中。  **二、实验数据：**  **实验1：存储器容量扩展**    **实验2：存储器与CPU连接**    **实验3：堆栈**    **实验4：FIFO**    **实验5：Cache** |
| **小结** |
| 这次是计组的第三次实验，内容是存储器，也是理论课程学习中很重要的一环，因此开展这次的实验课也是很有必要的。与前两次实验不同的是，这次的五个小实验内容在理论课上或多或少已经学习过了，所以操作起来也不是那么困难。但不知道是不是有段时间没复习了，刚开始看指导书时还有些陌生，于是我又回到书本复习了下第三章存储器的内容，这才顺利做完了实验。通过本次实验，我也再次熟悉了计算机字位扩展的硬件实现，如何设置扩展后的芯片存储，堆栈式结构管理和存放数据，FIFO形式传送数据，以及很重要的高速缓冲存储器—Cache的实现，同时也复习了如cache的三种映射方式、存储器扩展方式的计算等内容。不得不说做完以上实验我真是大开眼界，能够将理论学习的知识运用到实践设计中，是一件很成功的事情！ |
| **指导教师评语及成绩** |
| 评语：  成绩：           指导教师签名：                                                 批阅日期： |