嵌入式处理器与芯片设计基础实验报告
Cache 操作实验

陈之逸 21307130003

一、实验目的

在 SMART 平台上通过调整 Cache 参数观察示例程序的 CPI、Cache 缺失率等，了解 Cache 大小对 CPU 性能的影响。

二、实验步骤（包括实验结果，数据记录，截图等）

1. 通过配置生成工具，生成不同配置的 C910rtl。

   a) 按照讲义步骤启动 C910，打开配置工具界面

   b) 对生成的 RTL 代码文件进行修改（RTL generate completed）

```
/*
`ifdef FPGA
`define SYSMAP_BASE_ADDR0  28'h1f9ff
`define SYSMAP_FLG0        5'b01111

`define SYSMAP_BASE_ADDR1  28'h1fbd7
`define SYSMAP_FLG1        5'b10111

`define SYSMAP_BASE_ADDR2  28'h1fc1f
`define SYSMAP_FLG2        5'b01111

`define SYSMAP_BASE_ADDR3  28'h1fdff
`define SYSMAP_FLG3        5'b10111

`define SYSMAP_BASE_ADDR4  28'h28000
`define SYSMAP_FLG4        5'b10111

`define SYSMAP_BASE_ADDR5  28'h3fffff
`define SYSMAP_FLG5        5'b01111

`define SYSMAP_BASE_ADDR6  28'h7ffffff
`define SYSMAP_FLG6        5'b01111

`define SYSMAP_BASE_ADDR7  28'h8ffffff
`define SYSMAP_FLG7        5'b01111
`else
`define SYSMAP_BASE_ADDR0  28'h8ffff
`define SYSMAP_FLG0        5'b01111

`define SYSMAP_BASE_ADDR1  28'hbffff
`define SYSMAP_FLG1        5'b10011

`define SYSMAP_BASE_ADDR2  28'hcffff
`define SYSMAP_FLG2        5'b00011

`define SYSMAP_BASE_ADDR3  28'heffff
`define SYSMAP_FLG3        5'b01101
```

```
`define SYSMAP_FLG0        5'b01111

`define SYSMAP_BASE_ADDR1  28'hbffff
`define SYSMAP_FLG1        5'b10011

`define SYSMAP_BASE_ADDR2  28'hcffff
`define SYSMAP_FLG2        5'b00011

`define SYSMAP_BASE_ADDR3  28'heffff
`define SYSMAP_FLG3        5'b01101

`define SYSMAP_BASE_ADDR4  28'hffffff
`define SYSMAP_FLG4        5'b01111

`define SYSMAP_BASE_ADDR5  28'h3ffffff
`define SYSMAP_FLG5        5'b01111

`define SYSMAP_BASE_ADDR6  28'h4ffffff
`define SYSMAP_FLG6        5'b10010

`define SYSMAP_BASE_ADDR7  28'hfffffff
`define SYSMAP_FLG7        5'b01111

`endif*/
`define MULTI_PROCESSING
`define PROCESSOR_0
`define ICACHE_32K
`define DCACHE_32K
`define L2_CACHE_512K
`define FPGA
```

   c) 更改以及替换 SMART 平台的文件

   d) 通过上述步骤，以便将不同配置的 RTL 代码放入 SMART 平台进行测试

和仿真。

2. 基于不同配置的 rtl，仿真运行示例的 cache_test.c 程序，并记录表格数据

a) 以 L1Dcache=32KB,L2Cache=512KB 中的 N=1、L1Dcachetest 为例，首先生成 RTL 代码文件后，存入已经准备好的文件夹。11.v、12.v、21.v、22.v 是分别对应四种 L1Dcache 和 L2Cache 的组合的文件。然后打开 smart9 文件夹中的 case/cache_test/cache_test.c 文件，进行相应修改。

```
#include "stdio.h"
#include "stdlib.h"
#include "pmu.h"

int main(void)
{
  int i;
  int len1 = 10
  int len2 = 10
  int *arr1 = (
  int *arr2 = (

  int tmp = 1;

  int N = 1; //Step4 Simulation is finished

  int num_cycle_start = get_cycle();
  int num_instret_start = get_instret();
  int num_L1_Icache_access_start = get_L1_Icache_access();
  int num_L1_Icache_miss_start = get_L1_Icache_miss();
  int num_L1_Dcache_read_access_start = get_L1_Dcache_read_access();
  int num_L1_Dcache_read_miss_start = get_L1_Dcache_read_miss();
  int num_L1_Dcache_write_access_start = get_L1_Dcache_write_access();
  int num_L1_Dcache_write_miss_start = get_L1_Dcache_write_miss();
  int num_L2_Dcache_read_access_start = get_L2_Dcache_read_access();
  int num_L2_Dcache_read_miss_start = get_L2_Dcache_read_miss();
  int num_L2_Dcache_write_access_start = get_L2_Dcache_write_access();
  int num_L2_Dcache_write_miss_start = get_L2_Dcache_write_miss();

  // select L1/L2 to test
   #define L1_TEST
  // #define L2_TEST
```

Step3 (Make) is finished!
vcs!
Job <17379> is submitted to default queue <normal>.
cache_test

b) 在 workdir 中使用 run ../case/cache_test/cache_test.c 进行仿真运行，并使用 bjobs 查看运行情况，随后在 workdir 下查看生成的 run.log

```
num_cycle is 7478

num_instret is 1996

num_L1_Icache_access is 1464

num_L1_Icache_miss is 4

num_L1_Dcache_read_access is 385

num_L1_Dcache_read_miss is 384

num_L1_Dcache_write_access is 390

num_L1_Dcache_write_miss is 386

num_L2_Dcache_read_access is 780

num_L2_Dcache_read_miss is 771

num_L2_Dcache_write_access is 271

num_L2_Dcache_write_miss is 0
*********************************************
*    simulation finished successfully       *
*********************************************
$finish called from file "../tb/tb.v", line 315.
$finish at simulation time             4176250
           V C S   S i m u l a t i o n   R e p o r t
Time: 417625000 ps
CPU Time:     31.190 seconds;       Data structure size: 1028.4Mb
```

c) 记录相应数据，两个数据记录表如下：

L1Dcachetest

（L1_Dcachesize=32K)

configure	L1_Dcachesize	32K	32K	32K	32K	32K	32K	32K	32K
	L2_Dcachesize	512K	512K	512K	512K	1M	1M	1M	1M
	N	1	2	3	4	1	2	3	4
result	cycle	7478	13845	20136	26466	7478	13845	20136	26466
	insts	1996	5502	8238	10974	1996	5502	10148	12884
	CPI	3.746	2.516	2.444	2.412	3.746	2.516	2.534	2.482
	L1_Dreadaccess	385	768	1152	1536	385	768	1152	1536
	L1_Dreadmiss	384	767	1151	1535	384	767	1151	1535
	L1_Dreadmiss_rate	0.9974	0.9987	0.9991	0.9993	0.9974	0.9987	0.9991	0.9993
	L1_Dwriteaccess	390	774	1158	1542	390	774	1158	1542
	L1_Dwritemiss	386	770	1154	1538	386	770	1154	1538
	L1_Dwritemiss_rate	0.9897	0.9948	0.9965	0.9974	0.9897	0.9948	0.9965	0.9974

configure	L1_Dcachesize	64K	64K	64K	64K	64K	64K	64K	64K
	L2_Dcachesize	512K	512K	512K	512K	1M	1M	1M	1M
	N	1	2	3	4	1	2	3	4
result	cycle	7331	12608	17984	23360	7331	12608	17984	23360
	insts	1996	5502	8238	10974	1996	5502	8238	10974
	CPI	3.673	2.292	2.183	2.129	3.673	2.292	2.183	2.129
	L1_Dreadaccess	385	768	1152	1536	385	768	1152	1536
	L1_Dreadmiss	383	383	383	383	383	383	383	383
	L1_Dreadmiss_rate	0.9948	0.4987	0.3325	0.2493	0.9948	0.4987	0.3325	0.2493
	L1_Dwriteaccess	390	774	1158	1542	390	774	1158	1542
	L1_Dwritemiss	384	384	384	384	384	384	384	384
	L1_Dwritemiss_rate	0.9846	0.4961	0.3316	0.2490	0.9846	0.4961	0.3316	0.2490

（L1_Dcachesize=64K)

L2cachetest

（L1_Dcachesize=32K)

configure	L1_Dcachesize	32K	32K	32K	32K	32K	32K	32K	32K
	L2_Dcachesize	512K	512K	1M	1M	2M	2M	4M	4M
	N	1	2	1	2	1	2	1	2
result	cycle	489800	989814	481516	987442	465232	957243	448371	798845
	insts	155581	407487	155581	407487	155581	407487	155581	407487
	CPI	3.148	2.429	3.095	2.423	2.99	2.349	2.882	1.96
	L1_Dreadaccess	49717	98860	49719	98860	49719	98860	49711	98853
	L1_Dreadmiss	49166	98318	49169	98318	49169	98318	49166	98317
	L1_Dreadmiss_rate	0.989	0.995	0.989	0.995	0.989	0.995	0.989	0.995
	L2_Dreadaccess	49303	98464	49306	98464	49306	98464	49306	98466
	L2_Dreadmiss	49266	98419	49263	98413	49263	98413	49235	49236
	L2_Dreadmiss_rate	0.999	1	0.999	0.999	0.999	0.999	0.999	0.5

（L1_Dcachesize=64K)

configure	L1_Dcachesize	64K	64K	64K	64K	64K	64K	64K	64K
	L2_Dcachesize	512K	512K	1M	1M	2M	2M	4M	4M
	N	1	2	1	2	1	2	1	2
Result	cycle	489800	989814	481516	987442	465232	957243	448371	798845
	insts	155581	407487	155581	407487	155581	407487	155581	407487
	CPI	3.148	2.429	3.095	2.423	2.99	2.349	2.882	1.96
	L1_Dreadaccess	49717	98860	49719	98860	49719	98860	49711	98853
	L1_Dreadmiss	49166	98318	49169	98318	49169	98318	49166	98317
	L1_Dreadmiss_rate	0.989	0.995	0.989	0.995	0.989	0.995	0.989	0.995
	L2_Dreadaccess	49303	98464	49306	98464	49306	98464	49306	98466
	L2_Dreadmiss	49266	98419	49263	98413	49263	98413	49235	49236
	L2_Dreadmiss_rate	0.999	1	0.999	0.999	0.999	0.999	0.999	0.5

3. 基于不同配置的 rtl，仿真运行示例的 cache_test.c 程序，完成前述的 2 个表格。

三、实验分析和总结

L1TEST 中，因为 int 为 4 字节，所以最原始给的 len1 实际对应了 48KB 的大小，运行 cache_test.c 并将数据全部存入需要大于 48KB 的空间，在这种空间不够的情况下，无论遍历多少次，每当 cache 被占满，多余的 16KB 会重新覆盖开始的 16KB。分割 cache 中所存的数据为三部分会更好理解：1-16KB、17-32KB 和 33-48KB，在 N>1，也就是多次遍历时，则会多次进行 16KB 的覆盖。所以循环的次数 N 越多，访问数据不在 cache 里的次数就越多，missrate 会一直接近 100%。

L1TEST 中，因为 int 为 4 字节，所以最原始给的 len1 实际对应了 48KB 的大小，做 L1Test 时需要大于 48KB 的空间，而 L2Cache 的 size 取值有 512KB、1M、2M、4M，远大于 48KB，而 L1TEST 的 size 取值和 48KB 为一个量级，L1Cache 的 size 才对命中率变化有显著影响。

四、实验收获、存在问题、改进措施或建议等

在本实验中，我较为完整的了解了 cache 的基本原理，运作方式，映射方式，替换策略，分配策略，读写策略，L1 和 L2 的 inclusive/exclusive 关系，相关性能指标等基础知识，并且通过在 SMART 平台上调整调整 Cache 参数观察示例程序的 CPI、Cache 缺失率等，了解 Cache 大小对 CPU 性能的影响，得到相应的结论，并且通过修改 len1 和 len2 对结论进行进一步的验证。

在需要 run 多次仿真时先对第一次的 log 文件进行检查，若结果出错就方便找出问题、节省时间，否则会耗费较多精力纠缠在实验上。