

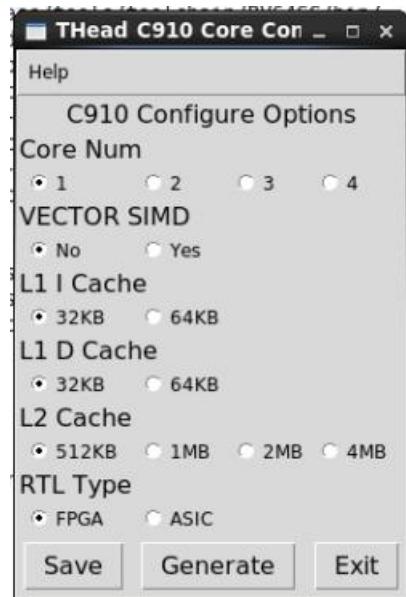
# Cache 操作试验

## 1、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。

## 2、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C910 rtl。



```
admin:/home/ECDesign/ecd11/C910_R1S2P19>[43]source setup/setup.csh
admin:/home/ECDesign/ecd11/C910_R1S2P19>[44]./THead_C910_Core_Config
...
.....
Start Generate, Waiting about 10+ Minutes.....
.....
C910 rtl generate completed!
get the rtl in gen_rtl directory
```

ct\_mp\_top\_merged.v + (~/C910\_R1S2P19/gen\_rtl) - GVIM (on admin)

```
File Edit Tools Syntax Buffers Window Help
```

```
'define SYSMAP_FLG3      5'b01101
#define SYSMAP_BASE_ADDR4 28'hfffff
#define SYSMAP_FLG4      5'b01111
#define SYSMAP_BASE_ADDR5 28'h3fffff
#define SYSMAP_FLG5      5'b01111
#define SYSMAP_BASE_ADDR6 28'h4fffff
#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_64K
`define L2_CACHE_512K
`define FPGA

:%s/ct_mp_top/C910MP/
```

ct\_mp\_top\_merged.v + (~/C910\_R1S2P19/gen\_rtl) - GVIM (on admin)

```
File Edit Tools Syntax Buffers Window Help
```

```
// &Depend("cpu_cfig.h"); @24
// &Depend("mp_top_golden_port.vp"); @25
// &Depend("ct_plic_top.v"); @26
// &Depend("ct_plic_hart_arb.v"); @27
// &Depend("ct_plic_hreq_busif.v"); @28
// &Depend("ct_plic_kid_busif.v"); @29
// &Depend("csky_apb_ltox_matrix.v"); @30
// &Depend("ct_plic_arb_ctrl.v"); @31
// &Depend("ct_plic_ctrl.v"); @32
// &Depend("ct_plic_32tol_arb.v"); @33
// &Depend("ct_plic_int_kid.v"); @34
// &Depend("ct_plic_granu_arb.v"); @35
// &Depend("ct_plic_granu2_arb.v"); @36
// &ModuleBeg; @37
module C910MP(
    axi_clk_en,
    biu_pad_araddr,
    biu_pad_arburst,
    biu_pad_arcache,
    biu_pad_arid,
    biu_pad_arlen,
    biu_pad_arlock,
    biu_pad_arprot,
    biu_pad_arsize,
    biu_pad_arvalid,
    biu_pad_awaddr,
    biu_pad_awburst,
    biu_pad_awcache,
    biu_pad_awid,
    biu_pad_awlen,
    biu_pad_awlock,
    biu_pad_awprot,
    biu_pad_awsize,
    biu_pad_awvalid,
```

(2) 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。

```

cache_test.c + (~/smart9_release/case/cache_test) - GVIM (on admin)
File Edit Tools Syntax Buffers Window Help
# include "stdio.h"
# include "stdlib.h"
# include "pmu.h"

int main(void)
{
    int i;
    int len1 = 1024 * 12; //48KB
    int len2 = 1024 * 768; //3MB
    int *arr1 = (int*)malloc(sizeof(int)*len1);
    int *arr2 = (int*)malloc(sizeof(int)*len2);

    int tmp = 1;

    int N = 1;

    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

    // test L1 Cache capacity
    // L1 Cache: 2-way , 64B-Cache line , strategy-fifo
    // access per Cache line(totally 768 lines) in arr1 N times , 16 int (64B) gap
    // unable Cache pre-fetch
    // cold miss first time
#endif L1_TEST

```

32,2

Top

表一

configure	L1_Dcache size	32K						
	L2_Dcache	512K	512K	512K	512K	1M	1M	1M
	N(1/2/3/4)	1	2	3	4	1	2	3
result	cycle	7478	13845	20136	26466	7478	13845	20136
	insts	1996	5502	8238	10974	1996	5502	8238
	CPI	3.746493	2.516358	2.444283	2.4117	3.746493	2.516358	2.444283
	L1_Dread access	385	768	1152	1536	385	768	1152
	L1_Dread miss	384	767	1151	1535	384	767	1151
	L1_Dread miss_	0.997403	0.998698	0.999132	0.999349	0.997403	0.998698	0.999132
	L1_Dwrite access	390	774	1158	1542	390	774	1158
	L1_Dwrite miss	386	770	1154	1530	386	770	1154
	L1_Dwrite miss_	0.989744	0.994832	0.996546	0.992218	0.989744	0.994832	0.996546

configure	L1_Dcache	64K						
	L2_Dcache size	512K	512K	512K	512K	1M	1M	1M
	N(1/2/3/4)	1	2	3	4	1	2	3
result	cycle	7331	12608	17984	23360	7331	12608	17984
	insts	1996	5502	8238	10974	1996	5502	8238
	CPI	3.672846	2.29153	2.183054	2.128668	3.672846	2.29153	2.183054
	L1_Dread access	385	768	1152	1536	385	768	1152
	L1_Dread miss	383	383	383	383	383	383	383
	L1_Dread miss_rate	0.994805	0.498698	0.332465	0.249349	0.994805	0.498698	0.332465
	L1_Dwrite access	390	774	1158	1542	390	774	1158
	L1_Dwrite miss	384	384	384	384	384	384	384
	L1_Dwrite miss_	0.984615	0.496124	0.331606	0.249027	0.984615	0.496124	0.331606

表二

configure	L1_Dcache	32K							
	L2_Dcache size	512K	512K	1M	1M	2M	2M	4M	4M
	N(1/2)	1	2	1	2	1	2	1	2
result	cycle	484115	975647	476012	981420	459621	951215	442891	793160
	insts	153635	405541	153635	405541	153635	405541	153635	405541
	CPI	3.151072	2.405791	3.09833	2.420027	2.991643	2.345546	2.882748	1.955807
	L1_Dread access	49156	98304	49152	98304	49156	98304	49152	98304
	L1_Dread miss	49155	98303	49151	98303	49155	98303	49151	98303
	L1_Dread miss_rate	0.99998	0.99999	0.99998	0.99999	0.99998	0.99999	0.99998	0.99999
	L2_Dread access	49168	98321	49164	98321	49168	98321	49164	98321
	L2_Dread miss	49162	98310	49152	98304	49156	98304	49148	49148
	L2_Dread	0.999878	0.999888	0.999756	0.999827	0.999756	0.999827	0.999675	0.499873

configure	L1_Dcache	64K							
	L2_Dcache	512K	512K	1M	1M	2M	2M	4M	4M
	N(1/2)	1	2	1	2	1	2	1	2
result	cycle	484115	975647	476012	981420	459621	951215	442891	793160
	insts	153635	405541	153635	405541	153635	405541	153635	405541
	CPI	3.151072	2.405791	3.09833	2.420027	2.991643	2.345546	2.882748	1.955807
	L1_Dread access	49156	98304	49152	98304	49156	98304	49152	98304
	L1_Dread miss	49155	98303	49151	98303	49155	98303	49151	98303
	L1_Dread miss_rate	0.99998	0.99999	0.99998	0.99999	0.99998	0.99999	0.99998	0.99999
	L2_Dread access	49168	98321	49164	98321	49168	98321	49164	98321
	L2_Dread miss	49162	98310	49152	98304	49156	98304	49148	49148
	L2_Dread miss_rate	0.999878	0.999888	0.999756	0.999827	0.999756	0.999827	0.999675	0.499873

### 3、实验分析和总结（结合测试程序 cache\_test.c 分析 cache 命中率变化的原因，注意控制变量比较）

仿真环境下一些 C910 Cache 的主要配置情况：

Cache 块大小：64Byte

Cache 块替换策略：L1 和 L2 的替换策略均为先入先出（FIFO）

Cache 的映射：L1 Data Cache 为 2 路组相联，L2 Cache 为 16 路组相联

Cache 的数据预取：禁止数据预取，数据缺失时只会取出该数据所在的 Cache 块

Cache 的写策略：写直（write-through）+写分配（write allocate）

做 L1 TEST 时，改变 L2 cache 的 size 对 L1 cache 的命中率变化没有影响。是因为二级缓存器存储的是一级缓存放不下的数据，容量大，但是调取速度慢。在 L1 的测试中，L2 大小的改变并不会对 L1 的命中率产生影响，因为 L2 缓冲区较大，可以容纳 len1 数组的数据。

做 L1 TEST 时，在 L1 Dcache size = 32KB 时，L1 Dcache 的缺失率一直接近 100% 是因为 FIFO 算法按调入 Cache 的先后决定淘汰的顺序，选择最早调入 Cache 的字块进行替换，它不需要记录各字块的使用情况，比较容易实现，系统开销小，其缺点是可能会把一些需要经常使用的程序块（如循环程序）也作为最早进入 Cache 的块替换掉，而且没有根据访存的局部性原理，故不能提高 Cache 的命中率。

最早调入的信息可能以后还要用到，或者经常要用到，如循环程序。这种方法简单、方便，利用了主存的“历史信息”，但并不能说最先进入的就不经常使用。缺点是不能正常反映程

序局部性原理，命中率不高，可能出现一种异常现象。Len1 共有 12K 个整型数据，会占用 48KB 的内存，每次读都会跨越 32 个整型。循环将迭代  $384 \times N$  次。由于 L1 缓存一路可以放 16 个整型（如上图所示），每次迭代的访问位置都

不在当前的页内。当总页数没分配满的时候，每次访存：从 arr1(32k) 读数据，写数据进入 arr1(32k+16) 都会触发缺页，并占用两个新的页面，因此在 256 个周期后，页面将会全部分配完。当  $N > 1$  时，由于数组是循环读写的，下一次循环时，之前保存的相关数据已经丢弃了，因此将会一直不命中。

在做 L1 TEST 时，在 L1 Dcache size = 64KB 时， $N=1, 2, 3, 4$  情况下的 L1 Dcache 的缺失率呈现约 100%、约 50%、约 33%、约 25% 这种等比下降的趋势。是因为当 L1 的大小变为 64KB 时，可以容得下所有的数组存储空间，不会因为  $N$  的增大而将之前分配的页面给替换掉，缺失率会下降。等比下降是因为遍历次数变多，步长相同的情况下，命中的概率变大。

在做 L2 TEST 时，适当修改 cache\_test.c 中两个数组 arr1、arr2 的大小 len1、len2，再基于不同 cache 配置的 CPU 进行仿真，由于只影响了总时钟周期，对缺页率以及 CPI 无明显影响。原因应该是仿真了 32K 的，修改前后，都是超过存储空间的，没有明显效果。如果是 64K 应与原实验结果有明显不同。

#### 4、实验收获、存在问题，改进措施或建议等

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。