

## 1. 任务

**任务：**使用实验1中提供的指令，实现一个包含64 KiB ICCM的新RVfpga系统。

请注意，默认系统中会禁止ICCM。因此，如SweRVref文档的第2.A部分所述，要使能ICCM，必须在文件 `[RVfpgaPath]/RVfpga/src/SweRVolfSoC/SweRVEh1CoreComplex/include/common_defines.vh` 中包含以下行：

```
`define RV_ICCM_ENABLE 1
```

此外，默认RVfpga系统中提供的参数适用于512 KiB ICCM。因此，要实现64 KiB ICCM，必须修改上述文件（文件`common_defines.vh`）的以下行：

- `RV_ICCM_DATA_CELL ram_16384x39` → `RV_ICCM_DATA_CELL ram_2048x39`
- `RV_ICCM_BITS 19` → `RV_ICCM_BITS 16`
- `RV_ICCM_ROWS 16384` → `RV_ICCM_ROWS 2048`
- `RV_ICCM_INDEX_BITS 14` → `RV_ICCM_INDEX_BITS 11`
- `RV_ICCM_SIZE_512` → `RV_ICCM_SIZE_64`
- `RV_ICCM_SIZE 512` → `RV_ICCM_SIZE 64`
- `RV_ICCM_EADR 32'hee07ffff` → `RV_ICCM_EADR 32'hee00ffff`

如SweRVref文档的第2.A部分所述，除手动修改文件`common_defines.vh`外，还可以使用`swerv.config`脚本修改SweRV EH1处理器的配置。


不提供解答。

**任务：**为上一任务中实现的ICCM绘制一张与图2类似的图。

不提供解答。

**任务：**在自己的计算机上重复图4中的仿真过程。为此，请按照以下步骤操作（在GSG的第7部分中详述）：

- 必要时生成仿真二进制文件（`Vrvfpgasim`）。

- 在PlatformIO中，打开在以下位置提供的项目：[RVfpgaPath]/RVfpga/Labs/Lab20/LW-SW\_Instruction\_DCCM。
- 在文件platformio.ini中建立到RVfpga仿真二进制文件（Vrvfpgasim）的正确路径。
- 使用Verilator生成仿真轨迹（生成轨迹）。
- 在GTKWave上打开轨迹。
- 使用文件scriptLoadStore.tcl（在[RVfpgaPath]/RVfpga/Labs/Lab20/LW-SW\_Instruction\_DCCM中提供）打开与图4所示信号相同的信号。为此，在GTKWave上，单击“File → Read Tcl Script File”（文件 → 读取Tcl脚本文件）并选择scriptLoadStore.tcl文件。
- 单击几次“Zoom In”（放大）（），然后分析自43900 ps起的区域。

解答请参见实验20的主文档。

**任务：**解释如何在模块lsu\_dccm\_mem的第103、104和105行中获取信号rden\_bank、wren\_bank和addr\_bank。

```
101 // 8 Banks, 16KB each (2048 x 72)
102 for (genvar i=0; i<DCCM_NUM_BANKS; i++) begin: mem_bank
103     assign wren_bank[i] = dccm_wren & (dccm_wr_addr[2+:DCCM_BANK_BITS] == i);
104     assign rden_bank[i] = dccm_rden & ((dccm_rd_addr_hi[2+:DCCM_BANK_BITS] == i) | (dccm_rd_addr_lo[2+:DCCM_BANK_BITS] == i));
105     assign addr_bank[i][(DCCM_BANK_BITS+DCCM_WIDTH_BITS)+:DCCM_INDEX_BITS] = wren_bank[i] ? dccm_wr_addr[(DCCM_BANK_BITS+DCCM_WIDTH_BITS)+:DCCM_INDEX_BITS] :
106                                     (((dccm_rd_addr_hi[2+:DCCM_BANK_BITS] == i) & rd_unaligned) ?
107                                     dccm_rd_addr_hi[(DCCM_BANK_BITS+DCCM_WIDTH_BITS)+:DCCM_INDEX_BITS] :
108                                     dccm_rd_addr_lo[(DCCM_BANK_BITS+DCCM_WIDTH_BITS)+:DCCM_INDEX_BITS]);
```

信号wren\_bank

- 信号wren\_bank[7:0]包含8位，每个存储区对应1位。wren\_bank[i]==1时，将使能存储区i的写操作。
- 如果LSU将信号dccm\_wren（我们已在实验13中分析过该信号）置1，则会根据dccm\_wr\_addr中提供的存储区字段地址，对某个存储区进行写操作。

信号rden\_bank

- 信号rden\_bank[7:0]包含8位，每个存储区对应1位。rden\_bank[i]==1时，将使能存储区i的读操作。
- 如果LSU将信号dccm\_rden（我们已在实验13中分析过该信号）置1，则会根据dccm\_rd\_addr\_lo和dccm\_rd\_addr\_hi中提供的存储区字段地址，对一到两个存储区进行读操作（取决于访问是对齐访问还是未对齐访问）。

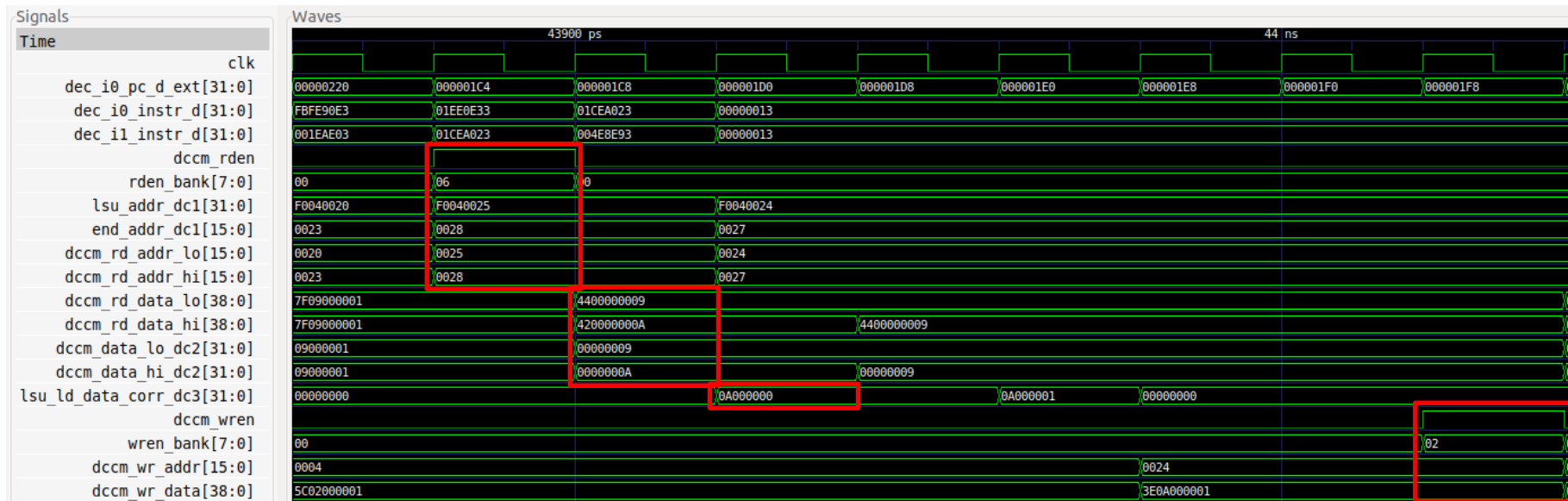
信号addr\_bank

- 信号wren\_bank[7:0][11]包含8个11位地址，每个存储区对应1个地址。

- 进行写操作时，地址从信号dccm\_wr\_addr处获取。
- 进行读访问时，地址可能位于信号dccm\_rd\_addr\_lo中（对齐读访问），也可能同时位于信号dccm\_rd\_addr\_lo和dccm\_rd\_addr\_hi中（未对齐读访问）。

**任务：**仿真DCCM的未对齐读访问，分析DCCM内部的处理方式。仍可使用上述程序（`[RVfpgaPath]/RVfpga/Labs/Lab20/LW-SW_Instruction_DCCM/`），只需将装载指令进行如下替换：

```
lw t3, (t4) → lw t3, 1(t4)
```



- 信号dccm\_rden = 0x06，因此为两个存储区使能读访问。
- 向内核提供两个值：
  - dccm\_data\_lo\_dc2 = 0x9
  - dccm\_data\_hi\_dc2 = 0xA

- 如实验13中所述，内核会对齐值：lsu\_ld\_data\_corr\_dc3 = 0x0A000000
- 5个周期后，将该值加1，然后写入DCCM：dccm\_wr\_data = 0x3E0A000001

**任务：**通过修改图4中的程序（*[RVfpgaPath]/RVfpga/Labs/Lab20/LW-SW\_Instruction\_DCCM/*），仿真DCCM存储区的冲突。

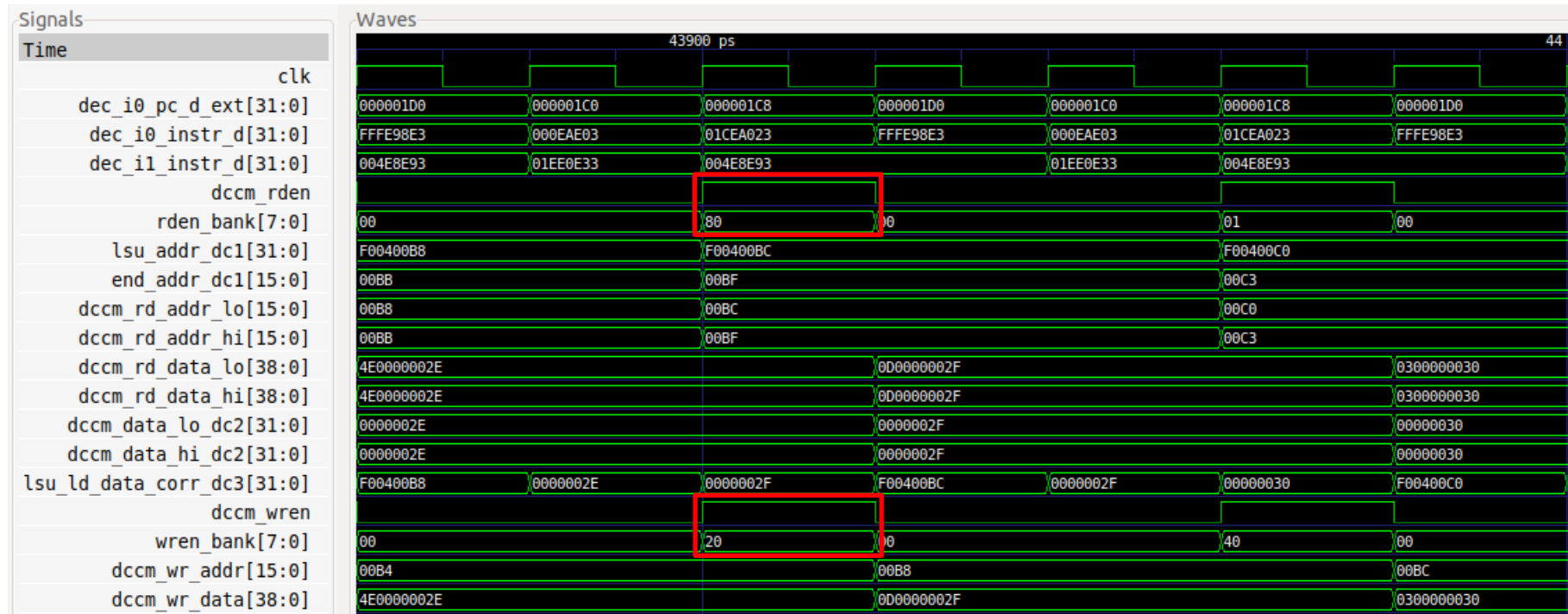
**第1项修改：**删除20条nop指令，重新生成仿真，并随机选取循环的某次迭代以分析lw和sw。

**第2项修改：**修改sw指令的立即数，使lw和sw尝试在同一周期内访问同一存储区：

sw t3, (t4) → sw t3, 8(t4)

### 第1项修改：

```
// Access array
la t4, D
li t5, 50
li t0, 1000
la t6, D
add t6, t6, t0
li t5, 1
REPEAT_Access:
    lw t3, (t4)
    add t3, t3, t5
    sw t3, (t4)
    add t4, t4, 4
    bne t4, t6, REPEAT_Access    # Repeat the loop
```

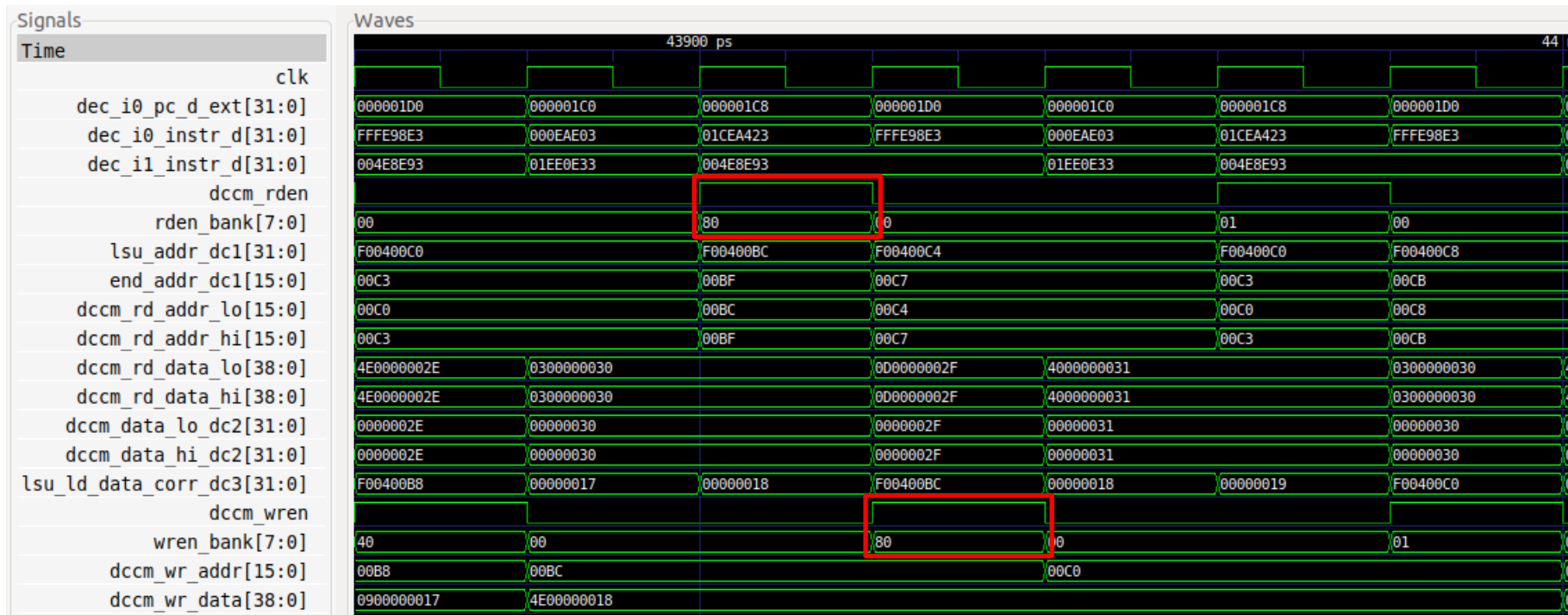


在这种情况下，将在同一周期中发出DCCM读操作和DCCM写操作请求。因为两种操作访问的存储区不同，所以可以在同一周期中执行。

## 第2项修改：

```
// Access array
la t4, D
li t5, 50
li t0, 1000
la t6, D
add t6, t6, t0
```

```
li t5, 1
REPEAT_Access:
    lw t3, (t4)
    add t3, t3, t5
    sw t3, 8(t4)
    add t4, t4, 4
    bne t4, t6, REPEAT_Access    # Repeat the loop
```



同样，将在同一周期中发出DCCM读操作和DCCM写操作请求。但是，不同于上一示例，此时读操作和写操作的对象为同一存储区，因此写操作会延迟一个周期。

**任务：** 在文件`platformio.ini`中（参见图10），注释掉第18行并取消注释第19行，以便程序使用以下路径中的链接器脚本：  
`[RVfpgaPath]/RVfpga/Labs/Lab20/RealBenchmarks/CoreMark_HwCounters/ld/link_DCCM-ICCM.ld`。分析新的链接器脚本，该脚本使用DCCM存储大部分数据，使用ICCM存储指令。执行CoreMark基准测试，将结果与本部分提供的结果进行比较。在本例中，由于我们的默认RVfpga系统不包括ICCM，请使用您在本实验的第一个任务中创建的比特流或以下路径中的比特流：  
`[RVfpgaPath]/RVfpga/Labs/Lab20/RealBenchmarks/rvfpganexys_DCCM-ICCM.bit`。

```
58 while(1);
59
60
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL

```
Invert any Switch to execute CoreMark
Invert any Switch to execute CoreMark
Invert any Switch to execute CoreMark
Invert any Switch to execute CoreMark
Invert any Switch to execute CoreMark
2K performance run parameters for coremark.

CoreMark Size      : 666
Total ticks        : 515967
Total time (secs): 515
Iterat/Sec/MHz     : 1.94
Iterations         : 1
Compiler version   : GCC8.3.0
Compiler flags     : -O2
Memory location    : STATIC
seedcrc           : 0xe9f5
[0]crclist        : 0xe714
[0]crcmatrix      : 0x1fd7
[0]crcstate       : 0x8e3a
[0]crcfinal       : 0xe714

Correct operation validated. See readme.txt for run and reporting rules.

Cycles = 515855
Instructions = 496680
Data Bus Transactions = 0
Inst Bus Transactions = 0
```

在本例中，CM/MHz（即Iterat/Sec/MHz）的值为1.94。仅使用DCCM时，CM/MHz为1.88。与仅使用DCCM时相比，性能略微提升，这是由于周期次数稍有减少。SweRV EH1处理器包含一个IS，因此，使用ICCM只会产生很小的差异，性能提升的幅度并不大。最后，可以观察到数据和指令总线事务数均为0，因为数据和指令分别通过DCCM和ICCM访问。



**任务：**将编译器优化方式改为-O3，执行程序并解释结果。

```
Invert any Switch to execute CoreMark
Invert any Switch to execute CoreMark
2K performance run parameters for coremark.

CoreMark Size      : 666
Total ticks        : 268997
Total time (secs): 268
Iterat/Sec/MHz     : 3.73
Iterations         : 1
Compiler version   : GCC8.3.0
Compiler flags     : -O2
Memory location    : STATIC
seedcrc           : 0xe9f5
[0]crclist        : 0xe714
[0]crcmatrix      : 0x1fd7
[0]crcstate       : 0x8e3a
[0]crcfinal       : 0xe714

Correct operation validated. See readme.txt for run and reporting rules.

Cycles = 268869
Instructions = 292421
Data Bus Transactions = 0
Inst Bus Transactions = 1760
```

在本例中，CM/MHz（即Iterat/Sec/MHz）的值为3.73。相比于使用DCCM和-O2执行程序时，指令数量以及相应的周期数均有所减少。

## 2. 练习

1) 使用Dhrystone基准替代CoreMark基准，执行相同的分析。可访问以下路径获取包含Dhrystone基准的PlatformIO项目：  
`[RVfpgaPath]/RVfpga/Labs/Lab20/RealBenchmarks/Dhrystone_HwCounters`。根据所有基准的要求，我们已使用  
<https://github.com/chipsalliance/Cores-SweRV>中提供的源代码对该Dhrystone基准进行了修改，使其能够适用于特定系统（在本例中为RVfpga系统）。文件`Test.c`与CoreMark中的文件（图6）类似，但会调用函数`main_dhry()`，该函数包含Dhrystone基准本身。

- 无编译器优化，无DCCM，无ICCM

```
Cycles = 1838481
Instructions = 402057
Data Bus Transactions = 194011
Inst Bus Transactions = 232
```

- DCCM

```
Cycles = 475969
Instructions = 402057
Data Bus Transactions = 0
Inst Bus Transactions = 240
```

- DCCM和ICCM

```
Cycles = 475173
Instructions = 402057
Data Bus Transactions = 0
Inst Bus Transactions = 0
```

- 编译器优化（-O2）和DCCM

```
Cycles = 250481
Instructions = 274082
Data Bus Transactions = 0
Inst Bus Transactions = 176
```

- 编译器优化（-O3）和DCCM

```
Cycles = 236660  
Instructions = 264082  
Data Bus Transactions = 0  
Inst Bus Transactions = 160
```

- 2) 使用图像处理应用程序替代CoreMark，执行相同的分析。可访问以下路径获取包含图像处理应用程序的PlatformIO项目：  
[RVfpgaPath]/RVfpga/Labs/Lab20/RealBenchmarks/ImageProcessing\_HwCounters。我们在实验5中曾使用这些应用程序将RGB图像转换为灰度图像。文件Test.c与CoreMark中的文件（图6）类似，但会调用函数ImageTransformation()，该函数包含我们在实验5中分析的图像转换基准。默认RVfpga系统的DCCM没有足够的空间来存储图像，因此需使用具有128 KiB DCCM的RVfpga系统（比特流），该文件路径为：[RVfpgaPath]/RVfpga/Labs/Lab20/RealBenchmarks/Bitstreams/rvfpganexys\_DCCM-128.bit。

- 无编译器优化，无DCCM，无ICCM

```
Created Grey Image  
Cycles = 3218631  
Instructions = 951907  
Data Bus Transactions = 233659  
Inst Bus Transactions = 64
```

- DCCM

```
Created Grey Image  
Cycles = 735838  
Instructions = 952263  
Data Bus Transactions = 4119  
Inst Bus Transactions = 64
```

- 编译器优化（-O2）和DCCM

```
Created Grey Image  
Cycles = 358716  
Instructions = 456133  
Data Bus Transactions = 4132  
Inst Bus Transactions = 40
```

- 编译器优化（-O3）和DCCM

```
Created Grey Image  
Cycles = 285475  
Instructions = 346027  
Data Bus Transactions = 4134  
Inst Bus Transactions = 48
```

- 3) 根据本实验第2.C部分所述，允许/禁止各种内核功能。比较相应的性能结果（即在这些修改后的内核上执行程序时HW计数器的值）。在Nexys A7开发板上，使用这些修改后的RVfpga系统运行全部三个程序（CoreMark、Dhrystone和图像处理）。可能的变化包括：
- 使用不同的分支预测器配置和实现方案（如始终不采取分支、使用Gshare分支预测器或使用实验16的练习1中实现的双模分支预测器）。
  - 允许/禁止双发射功能。
  - 使用各种不同的I\$/DCCM/ICCM配置（例如不同的大小或不同的I\$替换策略）。

不提供解答。