

1. 任務

任務：使用實驗1中提供的指令，實作一個包含64 KiB ICCM的新RVfpga系統。

請注意，預設系統中會停用ICCM。因此，如SweRVref文件的第2.A部分所述，要啟用ICCM，必須在檔案 `[RVfpgaPath]/RVfpga/src/SweRVolfSoC/SweRVEh1CoreComplex/include/common_defines.vh` 中包含以下行：

```
`define RV_ICCM_ENABLE 1
```

此外，預設RVfpga系統中提供的參數適用於512 KiB ICCM。因此，要實作64 KiB ICCM，必須修改上述檔案（檔案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文件的第2.A部分所述，除手動修改檔案common_defines.vh外，還可以使用swerv.config指令碼修改SweRV EH1處理器的組態。


不提供解答。

任務：為上一任務中實作的ICCM繪製一張與圖2類似的圖。

不提供解答。

任務：在自己的電腦上重複圖4中的模擬過程。為此，請按照以下步驟操作（在GSG的第7部分中詳述）：

- 必要時產生模擬二進位檔案（Vrvfpgasim）。

- 在PlatformIO中，開啟在以下位置提供的專案：`[RVfpgaPath]/RVfpga/Labs/Lab20/LW-SW_Instruction_DCCM`。
- 在檔案`platformio.ini`中建立到RVfpga模擬二進位檔案（`Vrvfpgasim`）的正確路徑。
- 使用Verilator產生模擬軌跡（產生軌跡）。
- 在GTKWave上開啟軌跡。
- 使用檔案`scriptLoadStore.tcl`（在`[RVfpgaPath]/RVfpga/Labs/Lab20/LW-SW_Instruction_DCCM`中提供）開啟與圖4所示訊號相同的訊號。為此，在GTKWave上，按一下「**File** → **Read Tcl Script File**」（檔案 → 讀取Tcl指令碼檔案）並選擇`scriptLoadStore.tcl`檔案。
- 按幾次「**Zoom In**」（放大）（），然後分析自43900 ps起的區域。

解答請參見實驗20的主文件。

任務：解釋訊號`rden_bank`、`wren_bank`與`addr_bank`如何在模組`lsu_dccm_mem`的第103、104與105行中取得。

```

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]);

```

訊號`wren_bank`

- 訊號`wren_bank[7:0]`包含8位元，每個記憶庫對應1位元。當`wren_bank[i]==1`時，啟用記憶庫`i`的寫入。
- 如果LSU將訊號`dccm_wren`（我們已在實驗13中分析過該訊號）置1，則會根據`dccm_wr_addr`中提供的記憶庫欄位位址，對某個記憶庫進行寫入操作。

訊號`rden_bank`

- 訊號`rden_bank[7:0]`包含8位元，每個記憶庫對應1位元。當`rden_bank[i]==1`時，啟用記憶庫`i`的讀取。
- 如果LSU將訊號`dccm_rden`（我們已在實驗13中分析過該訊號）置1，則會根據`dccm_rd_addr_lo`和`dccm_rd_addr_hi`中提供的記憶庫欄位位址，對一到兩個記憶庫進行讀取操作（取決於存取是對齊存取還是未對齊存取）。

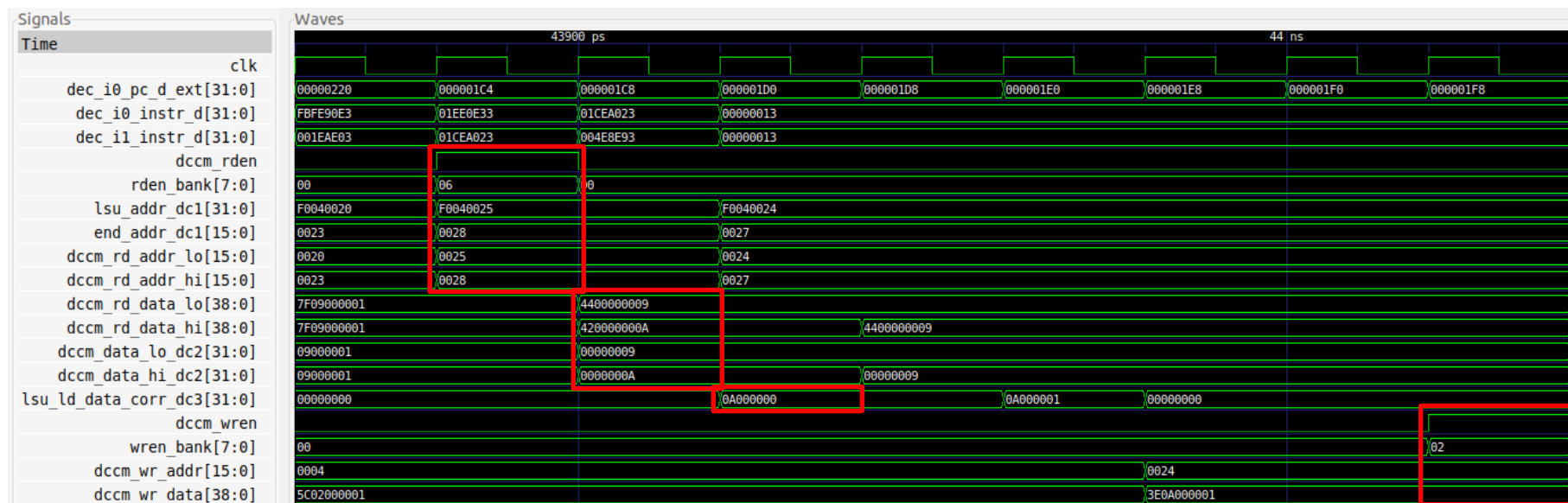
訊號`addr_bank`

- 訊號`wren_bank[7:0][11]`包含8個11位元位址，每個記憶庫對應1個位址。

- 進行寫入操作時，位址從訊號dccm_wr_addr處取得。
- 進行讀取存取時，位址可能位於訊號dccm_rd_addr_lo中（對齊讀取存取），也可能同時位於訊號dccm_rd_addr_lo和dccm_rd_addr_hi中（未對齊讀取存取）。

任務：模擬DCCM的未對齊讀取存取，分析DCCM內部的處理方式。仍可使用上述程式（`[RVfpgaPath]/RVfpga/Labs/Lab20/LW-SW_Instruction_DCCM/`），只需將載入指令進行如下替換：

```
lw t3, (t4) → lw t3, 1(t4)
```



- 訊號dccm_rden = 0x06，因此為兩個記憶體庫啟用讀取存取。
- 向核心提供兩個值：
 - dccm_data_lo_dc2 = 0x9
 - dccm_data_hi_dc2 = 0xA

- 如實驗13中所述，核心會對齊值：lsu_ld_data_corr_dc3 = 0x0A000000
- 5個週期後，將該值加1，然後寫入DCCM：dccm_wr_data = 0x3E0A000001

任務：修改圖4（*[RVfpgaPath]/RVfpga/Labs/Lab20/LW-SW_Instruction_DCCM/*）中的程式來模擬DCCM記憶體庫衝突。

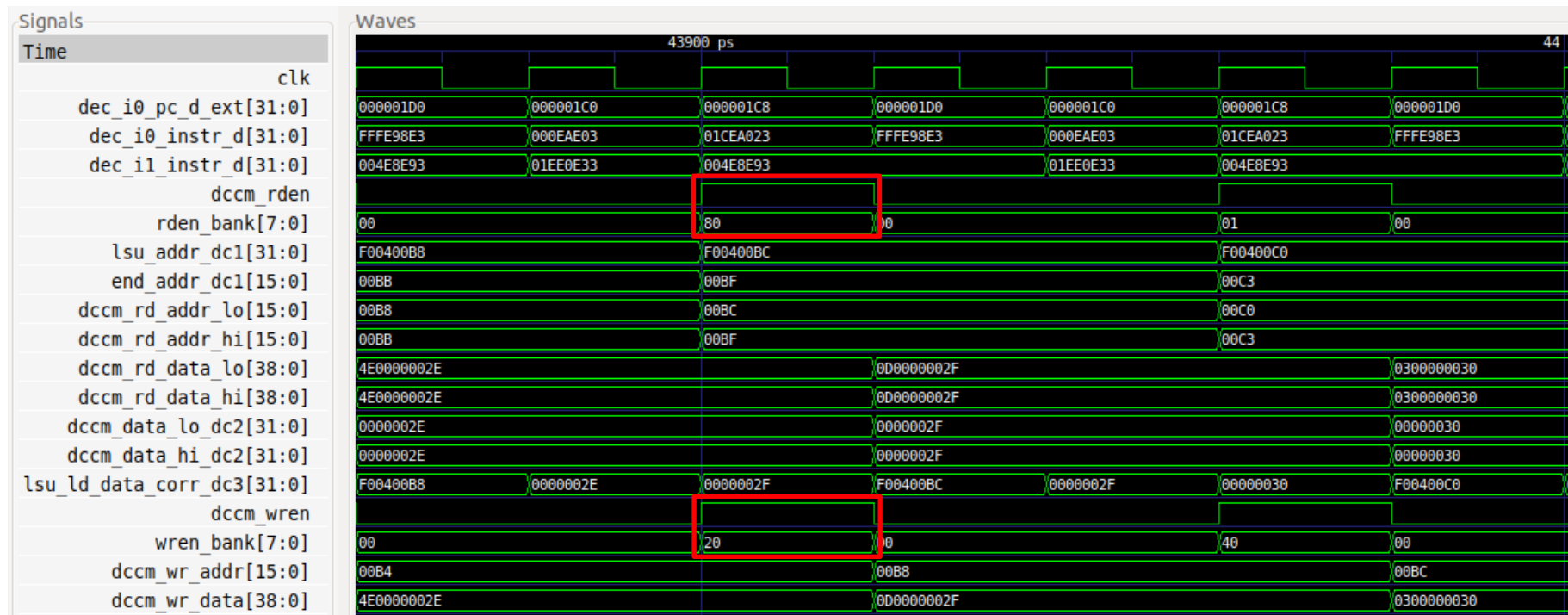
第1項修改：刪除20條nop指令，重新產生模擬，並隨機選取迴圈的某次迭代以分析lw和sw。

第2項修改：修改sw指令的立即數，使lw和sw嘗試在同一週期內存取同一記憶體庫：

sw t3, (t4) → sw t3, 8(t4)

第1項修改：

```
// 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
```



在這種情況下，將在同一週期中發出DCCM讀取操作和DCCM寫入操作要求。因為兩種操作存取的記憶體庫不同，所以可以在同一週期中執行。

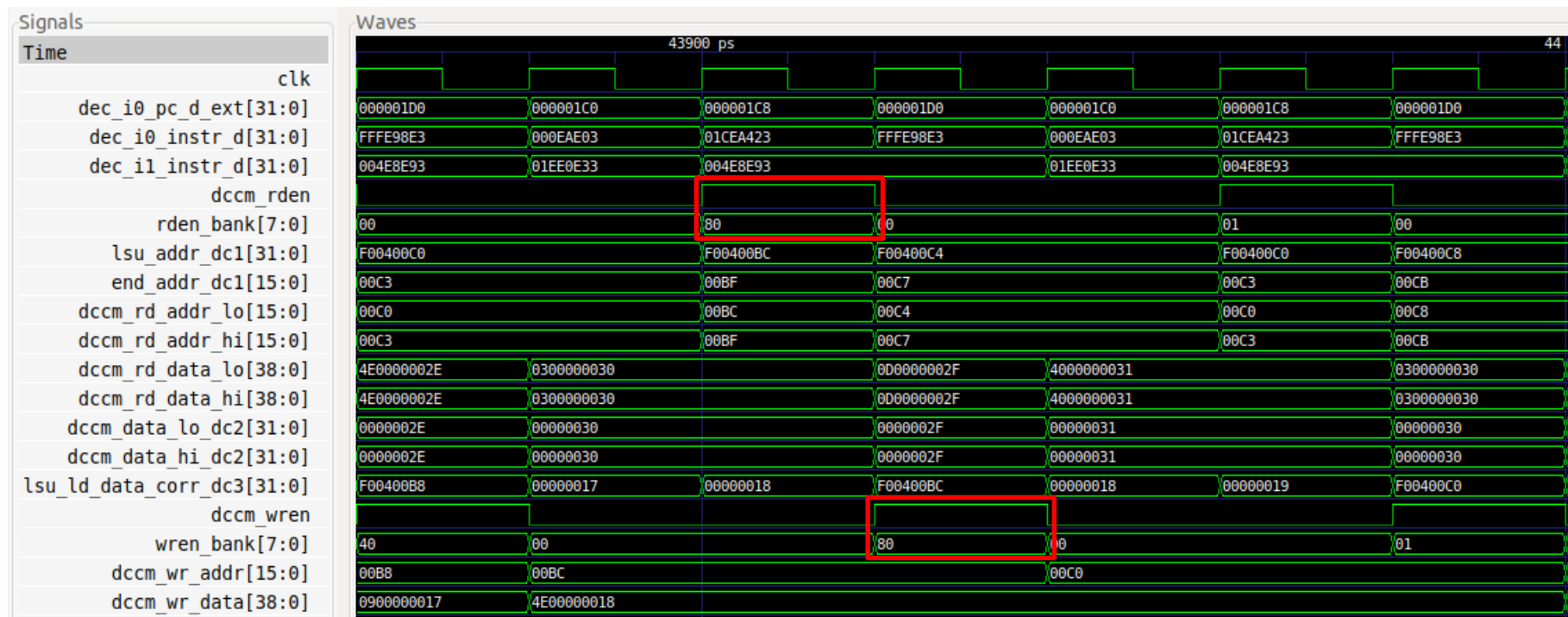
第2項修改：

```
// 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

```



同樣，將在同一週期中發出DCCM讀取操作和DCCM寫入操作要求。但是，不同於上一範例，此時讀取操作和寫入操作的對象為同一記憶體，因此寫入操作會延遲一個週期。

任務：在檔案`platformio.ini`中（參見圖10），註解掉第18行並取消註解第19行，以便程式使用以下路徑中的連結器指令碼：
`[RVfpgaPath]/RVfpga/Labs/Lab20/RealBenchmarks/CoreMark_HwCounters/ld/link_DCCM-ICCM.ld`。分析新的連結器指令碼，該指令碼使用DCCM儲存大部分資料，使用ICCM儲存指令。執行CoreMark基準測試，將結果與本部分提供的結果進行比較。在本例中，由於我們的預設RVfpga系統不包括ICCM，請使用您在本實驗的第一個任務中建立的位元串流或以下路徑中的位元串流：
`[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）的值為1.94。僅使用DCCM時，CM/MHz為1.88。與僅使用DCCM時相比，效能略微提升，這是由於週期次數稍有減少。SweRV EH1處理器包含一個IS，因此，使用ICCM只會產生很小的差異，效能提升的幅度並不大。最後，可以觀察到資料和指令匯流排交易數均為0，因為資料和指令分別透過DCCM和ICCM存取。

任務：將編譯器最佳化方式改為-O3，執行程式並解釋結果。

```
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）的值為3.73。相較於使用DCCM和-O2執行程式時，指令數量以及相應的週期數均有所減少。

2. 練習

1) 使用Dhrystone基準替代CoreMark基準，執行相同的分析。可存取以下路徑取得包含Dhrystone基準的PlatformIO專案：
[\[RVfpgaPath\]/RVfpga/Labs/Lab20/RealBenchmarks/Dhrystone_HwCounters](#)。根據所有基準的要求，我們已使用
<https://github.com/chipsalliance/Cores-SweRV>中提供的原始程式碼對該Dhrystone基準進行了修改，使其能夠適用於特定系統（在本例中為RVfpga系統）。檔案Test.c與CoreMark中的檔案（圖6）類似，但會叫用函數main_dhry()，該函數包含Dhrystone基準本身。

- 無編譯器最佳化，無DCCM，無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
```

- 編譯器最佳化（-O2）和DCCM

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

- 編譯器最佳化（-O3）和DCCM

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

2) 使用影像處理應用程式替代CoreMark，執行相同的分析。可存取以下路徑取得包含影像處理應用程式的PlatformIO專案：
[RVfpgaPath]/RVfpga/Labs/Lab20/RealBenchmarks/ImageProcessing_HwCounters。我們在實驗5中曾使用這些應用程式將RGB影像轉換為灰階影像。檔案**Test.c**與CoreMark中的檔案（圖6）類似，但會叫用函數ImageTransformation()，該函數包含我們在實驗5中分析的影像轉換基準。預設RVfpga系統的DCCM沒有足夠的空間來儲存影像，因此需使用具有128 KiB DCCM的RVfpga系統（位元串流），該檔案路徑為：**[RVfpgaPath]/RVfpga/Labs/Lab20/RealBenchmarks/Bitstreams/rvfpganexys_DCCM-128.bit**。

- 無編譯器最佳化，無DCCM，無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
```

- 編譯器最佳化（-O2）和DCCM

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

- 編譯器最佳化（-O3）和DCCM

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

- 3) 根據本實驗第2.C部分所述，啟用/停用各種核心功能。比較相應的效能結果（即在這些修改後的核心上執行程式時HW計數器的值）。在Nexys A7開發板上，使用這些修改後的RVfpga系統執行全部三個程式（CoreMark、Dhrystone和影像處理）。可能的變化包括：
- 使用不同的分支預測器組態和實作方案（如始終不採取分支、使用Gshare分支預測器或使用實驗16的練習1中實作的雙模分支預測器）。
 - 啟用/停用雙指令功能。
 - 使用各種不同的IS/DCCM/ICCM組態（例如不同的大小或不同的IS替換策略）。

不提供解答。