

1. TAREFAS

TAREFA: Usando as instruções fornecidas no Lab 1, implemente um novo sistema RVfpga que inclua uma ICCM de 64 KiB.

Lembre-se de que a ICCM está desativada no sistema base. Portanto, conforme explicado na Seção 2.A do documento SweRVref, para ativar a ICCM, é necessário incluir a seguinte linha no ficheiro

[RVfpgaPath]/RVfpga/src/SweRVolfSoC/SweRVEh1CoreComplex/include/common_defines.vh:

```
`definir RV_ICCM_ENABLE 1
```

Além disso, os parâmetros fornecidos no sistema RVfpga padrão são para uma ICCM de 512 KiB. Portanto, para implementar uma ICCM de 64 KiB, deve modificar as seguintes linhas do mesmo ficheiro (ficheiro common_defines.vh):

- 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

Conforme explicado na Seção 2.A do documento SweRVref, em vez de modificar manualmente o ficheiro *common_defines.vh*, também pode modificar a configuração do processador SweRV EH1 usando o script *swerv.config*.


Solução não fornecida.

TAREFA: Desenhe uma figura semelhante à Figura 2 para a ICCM implementada na tarefa anterior.

Solução não fornecida.

TAREFA: Replicar a simulação da Figura 4 no seu computador. Para fazer isso, siga as próximas etapas (conforme descrito em detalhes na Seção 7 do GSG):

- Se necessário, gere o binário de simulação (*Vrvfpgasim*).

- No PlatformIO, abra o projeto fornecido em: `[RVfpgaPath]/RVfpga/Labs/Lab20/LW-SW_Instruction_DCCM`.
- Defina o caminho correto para o binário de simulação do RVfpga (`Vrvfpgasim`) no ficheiro `platformio.ini`.
- Gere o trace da simulação usando o Verilator (Generate Trace).
- Abra o trace no GTKWave.
- Use o ficheiro `scriptLoadStore.tcl` (fornecido em `[RVfpgaPath]/RVfpga/Labs/Lab20/LW-SW_Instruction_DCCM`) para abrir os mesmos sinais que os mostrados na Figura 4. Para isso, no GTKWave, clique em `File → Read Tcl Script File` e selecione o ficheiro `scriptLoadStore.tcl`.
- Clique em `Zoom In` () várias vezes e analise a região que começa em 43.900 ps.

Solução fornecida no documento principal do Lab 20.

TAREFA: Explique como os sinais `rden_bank`, `wren_bank` e `addr_bank` são obtidos nas linhas 103, 104 e 105 do módulo `lsu_dccm_mem`.

```

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]);

```

Sinal `wren_bank`

- O sinal `wren_bank[7:0]` contém 8 bits, um por banco. A escrita no banco *i* é ativada quando `wren_bank[i]==1`.
- Se a LSU definir o sinal `dccm_wren` (analisamos esse sinal no Lab 13), um banco será escrito, conforme determinado pelo campo `Bank` do endereço fornecido em: `dccm_wr_addr`.

Sinal `rden_bank`

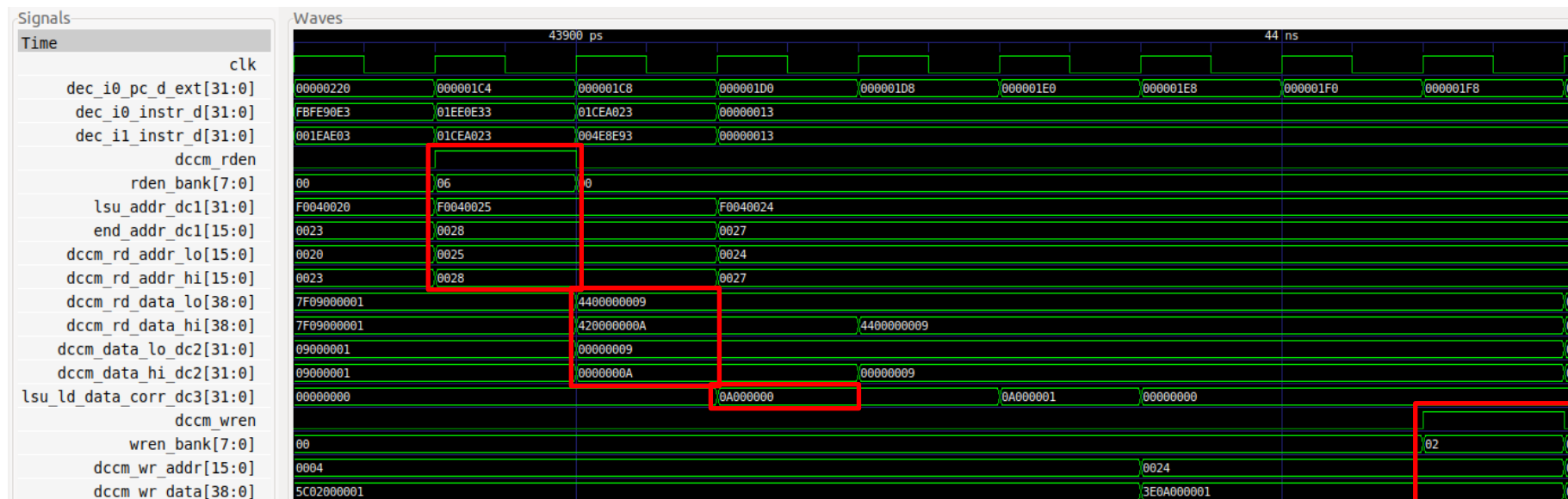
- O sinal `rden_bank[7:0]` contém 8 bits, um por banco. A leitura do banco *i* é ativada quando `rden_bank[i]==1`.
- Se a LSU definir o sinal `dccm_rden` (analisamos esse sinal no Lab 13), um ou dois bancos serão lidos (dependendo de o acesso ser alinhado ou não alinhado), conforme determinado pelo campo `Bank` dos endereços fornecidos em: `dccm_rd_addr_lo` e `dccm_rd_addr_hi`.

Sinal `addr_bank`

- O sinal `wren_bank[7:0][11]` contém 8 endereços de 11 bits, um por banco.
 - o No caso de uma escrita, o endereço é obtido do sinal `dccm_wr_addr`.
 - o No caso de uma leitura, o endereço está no sinal `dccm_rd_addr_lo` (em uma leitura alinhada) ou nos sinais `dccm_rd_addr_lo` e `dccm_rd_addr_hi` (em uma leitura não alinhada).

TAREFA: Simular uma leitura não alinhada para a DCCM e analisar como ela é tratada dentro da DCCM. Pode usar o programa acima (`[RVfpgaPath]/RVfpga/Labs/Lab20/LW-SW_Instruction_DCCM/`) e simplesmente substituir a instrução de leitura da seguinte forma:

```
lw t3, (t4) → lw t3, 1(t4)
```



- O sinal `dccm_rden` = 0x06, portanto, dois bancos estão habilitados para leitura.
- Dois valores são fornecidos ao núcleo:
 - o `dccm_data_lo_dc2` = 0x9
 - o `dccm_data_hi_dc2` = 0xA

- O núcleo alinha o valor conforme explicado no Lab 13: `lsu_ld_data_corr_dc3 = 0x0A000000`
- 5 ciclos depois, o valor, mais um, é gravado no DCCM: `dccm_wr_data = 0x3E0A000001`

TAREFA: Simule um conflito de banco DCCM modificando o programa da Figura 4 (*[RVfpgaPath]/RVfpga/Labs/Lab20/LW-SW_Instruction_DCCM*).

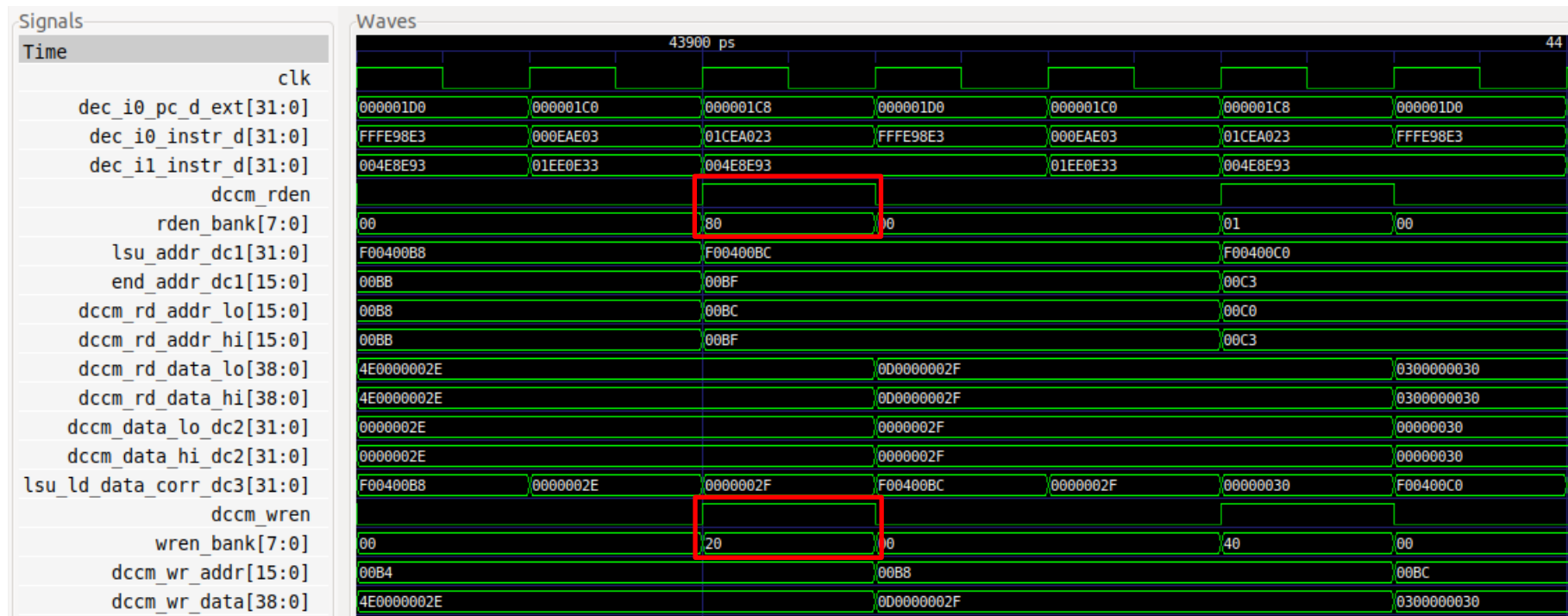
1st modificação: Remova as 20 instruções `nop`, gere novamente a simulação e analise o `lw` e o `sw` em uma iteração aleatória de o loop.

2nd modification: Modifique o imediato da instrução `sw` para fazer com que o `lw` e o `sw` tentem aceder ao mesmo banco no mesmo ciclo:

`sw t3, (t4) → sw t3, 8(t4)`

1ª modificação:

```
// 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
```



Nesse caso, a leitura e a escrita da DCCM ocorrem no mesmo ciclo. No entanto, como elas estão em bancos diferentes, podem ser executadas no mesmo ciclo.

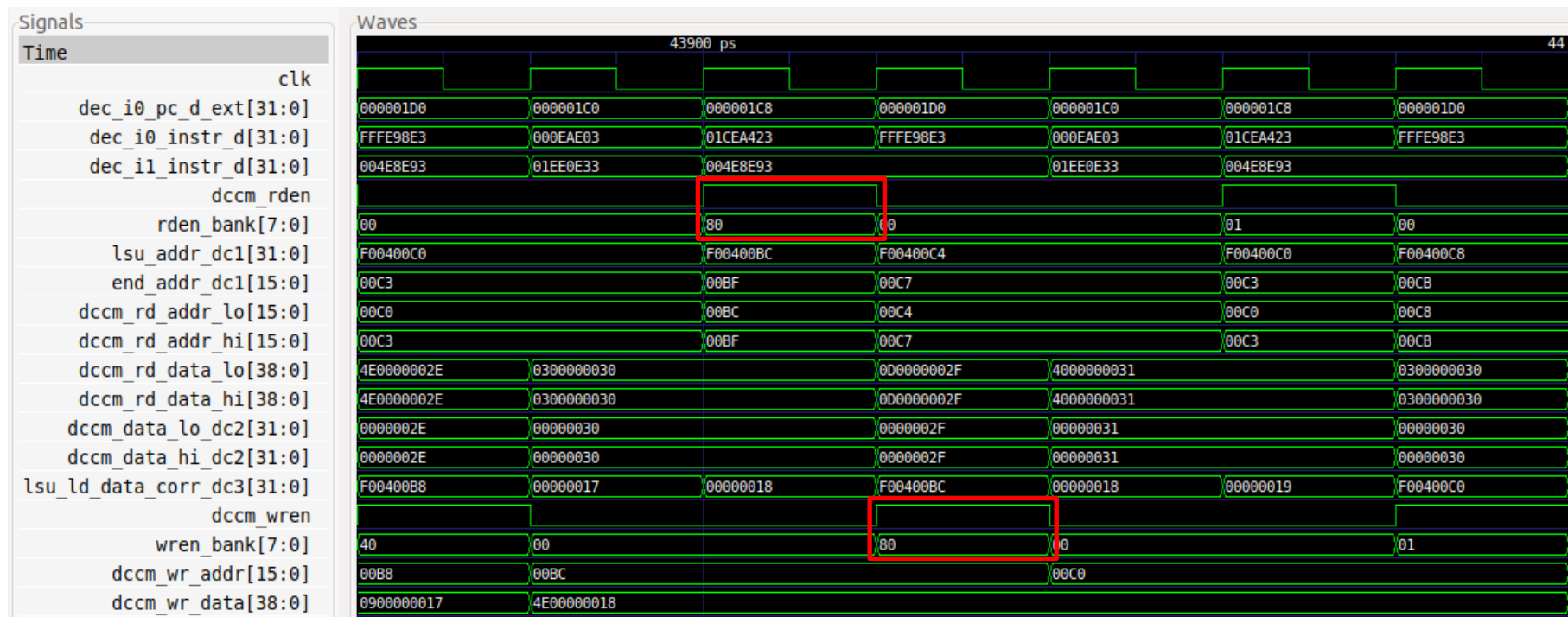
2ª modificação:

```
// 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

```



Novamente, a leitura e a escrita da DCCM ocorrem no mesmo ciclo. Entretanto, ao contrário do último exemplo, a leitura e a escrita são feitas no mesmo banco, de modo que a escrita é atrasada um ciclo.

TAREFA: No ficheiro *platformio.ini* (consulte a Figura 10), comente a linha 18 e descomente a linha 19 para que o programa use o *Linker Script* fornecido em: *[RVfpgaPath]/RVfpga/Labs/Lab20/RealBenchmarks/CoreMark_HwCounters/ld/link_DCCM-ICCM.ld*. Analise esse novo Linker Script, que usa a DCCM para armazenar a maioria dos dados e a ICCM para armazenar as instruções. Execute o benchmark CoreMark e compare os resultados com os obtidos nesta seção. Nesse caso, como nosso sistema RVfpga base não inclui uma ICCM, use o Bitstream que criou na primeira tarefa deste laboratório ou o Bitstream que fornecemos em: *[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

Nesse caso, o CM/MHz (ou seja, o valor de Iterat/Sec/MHz) é 1,94. Ao usar somente a DCCM, o CM/MHz é de 1,88. Esse pequeno aumento no desempenho deve-se a uma pequena redução no número de ciclos, quando comparado ao uso apenas da DCCM. A melhoria é pequena porque o processador SweRV EH1 tem uma I\$ e, portanto, o uso da ICCM faz apenas uma pequena diferença. Por fim, é possível observar que agora as transações do barramento de dados e de instruções são 0, já que os dados e as instruções são acedidos pela DCCM e pela ICCM, respectivamente.

TAREFA: Modificar a otimização da compilação para -O3 e explicar os resultados.

```
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
```

Nesse caso, o CM/MHz (ou seja, o valor de Iterat/Sec/MHz) é 3,73. O número de instruções e, portanto, o número de ciclos, diminuiu um pouco em relação à execução em que a DCCM e o -O2 são usados.

2. EXERCÍCIOS

1) Faça a mesma análise que foi feita para o CoreMark, mas desta vez usando o benchmark Dhrystone. Um projeto PlatformIO que contém o benchmark Dhrystone está em: `[RVfpgaPath]/RVfpga/Labs/Lab20/RealBenchmarks/Dhrystone_HwCounters`. Conforme exigido por todos os benchmarks, esse benchmark Dhrystone foi adaptado ao sistema específico, nesse caso, o sistema RVfpga, usa os códigos-fonte fornecidos em <https://github.com/chipsalliance/Cores-SweRV>. O ficheiro `Test.c` é semelhante ao do CoreMark (Figura 6), mas invoca a função `main_dhry()`, que inclui o próprio benchmark Dhrystone.

- Sem otimizações de compilador, sem DCCM, sem ICCM

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

- Com DCCM

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

- Com DCCM e ICCM

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

- Com otimizações do compilador (-O2) e DCCM

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

- Com otimizações do compilador (-O3) e DCCM

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

2) Faça a mesma análise que foi feita para o CoreMark, mas desta vez para a aplicação ImageProcessing. Um projeto PlatformIO que contém a aplicação ImageProcessing está em: `[RVfpgaPath]/RVfpga/Labs/Lab20/RealBenchmarks/ImageProcessing_HwCounters`. Essas são as aplicações que usámos no Lab 5 para transformar uma imagem RGB em escala de cinza. O ficheiro `Test.c` é semelhante ao do CoreMark (Figura 6), mas invoca a função `ImageTransformation()`, que inclui o benchmark Image Transformation que analisámos no Lab 5. O DCCM do sistema RVfpga base não é grande o suficiente para armazenar a imagem, portanto, em vez disso, use o sistema RVfpga (bitstream) que tem uma DCCM de 128 KiB, que está em: `[RVfpgaPath]/RVfpga/Labs/Lab20/RealBenchmarks/Bitstreams/rvfpganexys_DCCM-128.bit`.

- Sem otimizações de compilador, sem DCCM, sem ICCM

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

- Com DCCM

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

- Com otimizações do compilador (-O2) e DCCM

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

- Com otimizações do compilador (-O3) e DCCM

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

- 3) Ative/desative vários recursos do núcleo, conforme descrito na Seção 2.C deste laboratório. Compare os resultados de desempenho, ou seja, os valores dos HW Counters ao executar os programas nesses núcleos modificados. Execute todos os três programas (CoreMark, Dhrystone e ImageProcessing) nesses sistemas RVfpga modificados na placa Nexys A7. As variações incluem:
- Usando diferentes configurações e implementações do preditor de saltos (como always not-taken, Gshare e o preditor bimodal implementado no Exercício 1 de Lab 16).
 - Ativação/desativação do recurso dual-issue.
 - Usando várias configurações de I\$/DCCM/ICCM (como diferentes tamanhos ou diferentes Políticas de substituição de I\$).

Solução não fornecida.