# 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.

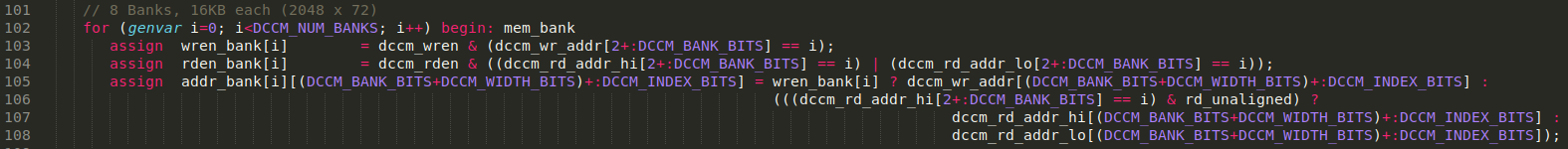
Solução não fornecida.

**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**.



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

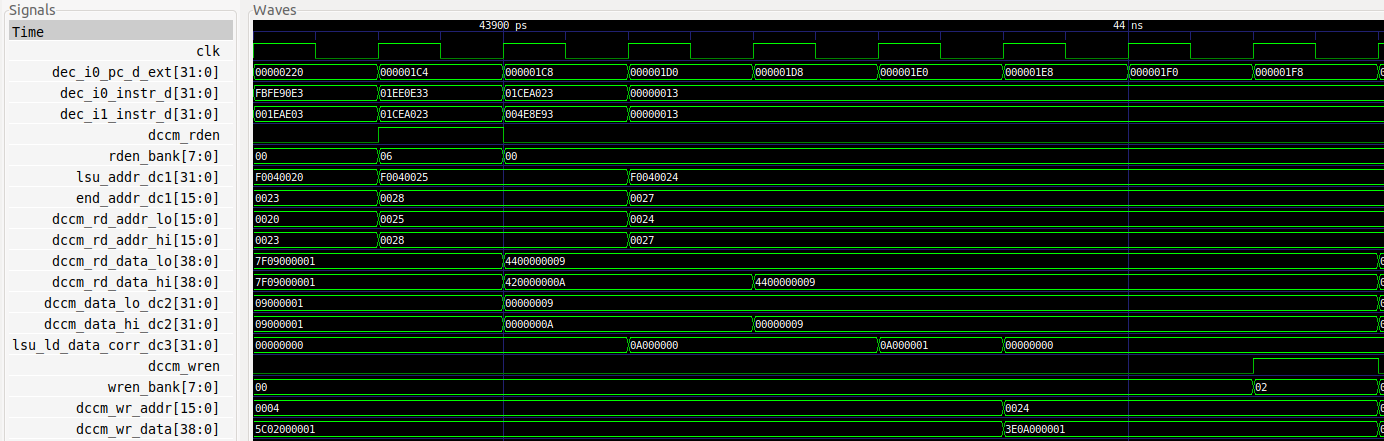
* O sinal rden\_bank[7:0] contém 8 bits, um por banco. A leitura do banco *i* é ativada quando rden\_bank[i]==1.
* Se a LSU definir o sinal dccm\_rden (analisamos esse sinal no Lab 13), um ou dois bancos serão lidos (dependendo de o acesso ser alinhado ou não alinhado), conforme determinado pelo campo 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.
  + No caso de uma escrita, o endereço é obtido do sinal dccm\_wr\_addr.
  + 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:
  + dccm\_data\_lo\_dc2 = 0x9
  + 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

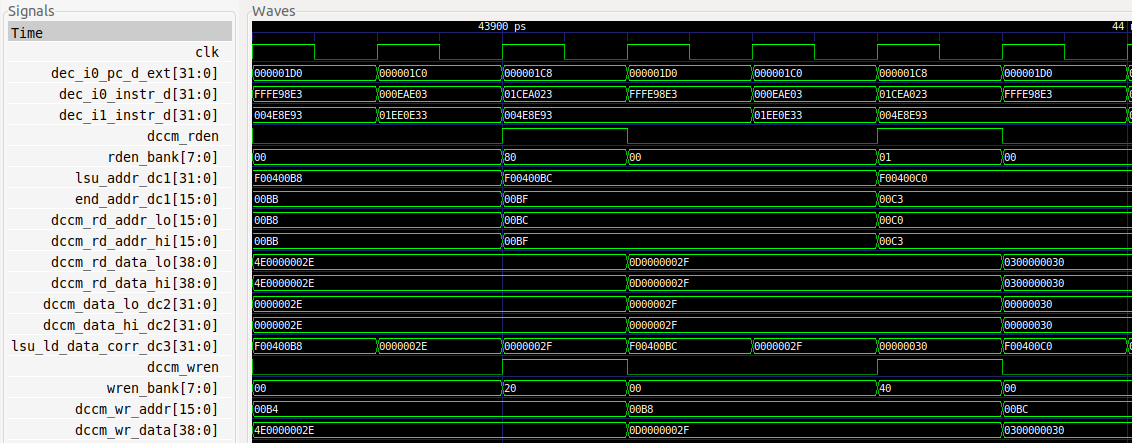
o loop.

**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:

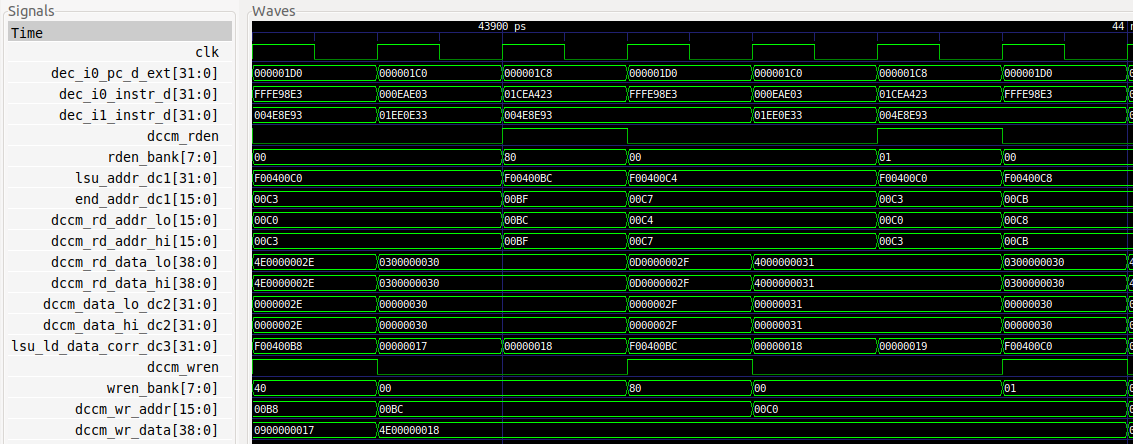
// Access arrayla t4, Dli t5, 50li t0, 1000la t6, Dadd t6, t6, t0li t5, 1REPEAT\_Access: lw t3, (t4) add t3, t3, t5 sw t3, (t4) add t4, t4, 4 bne t4, t6, REPEAT\_Access # Repeat the loop



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.

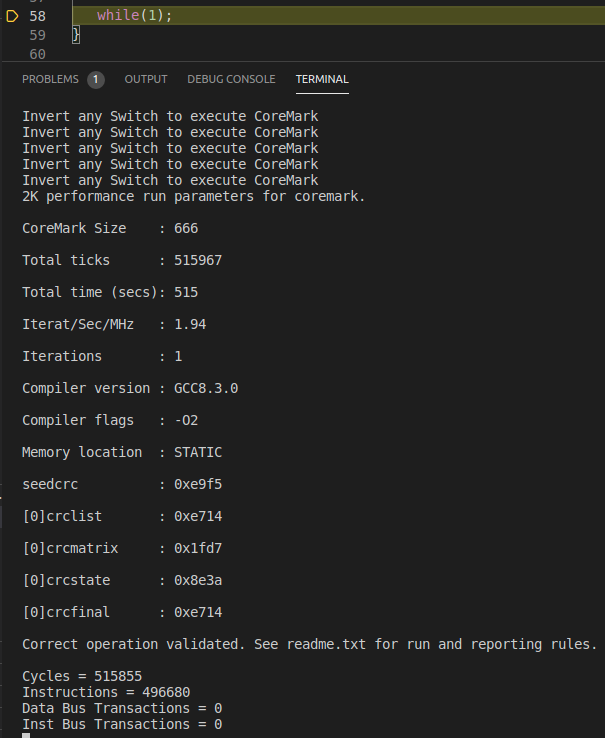
2a modificação:

// Access arrayla t4, Dli t5, 50li t0, 1000la t6, Dadd t6, t6, t0li t5, 1REPEAT\_Access: lw t3, (t4) add t3, t3, t5 sw t3, 8(t4) add t4, t4, 4 bne t4, t6, REPEAT\_Access # Repeat the loop



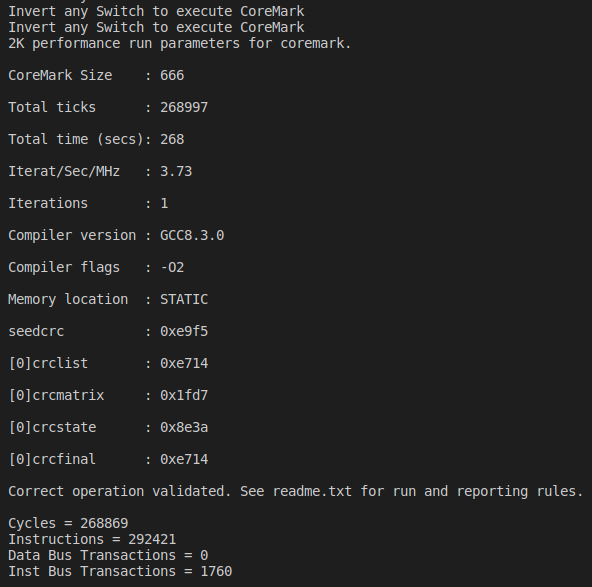
Novamente, a leitura e a escrita da DCCM ocorrem no mesmo ciclo. Entretanto, ao contrário do último exemplo, a leitura e a escrita são feitas no mesmo banco, de modo que a escrita é atrasada um ciclo.

**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*.



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.

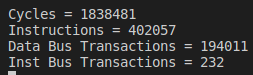


Nesse caso, o CM/MHz (ou seja, o valor de Iterat/Sec/MHz) é 3,73. O número de instruções e, portanto, o número de ciclos, diminuiu um pouco em relação à execução em que a DCCM e o -O2 são usados.

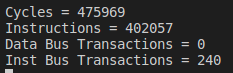
# EXERCÍCIOS

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.](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.

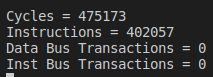
* Sem otimizações de compilador, sem DCCM, sem ICCM



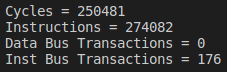
* Com DCCM



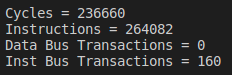
* Com DCCM e ICCM



* Com otimizações do compilador (-O2) e DCCM

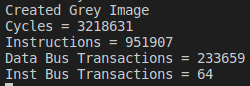


* Com otimizações do compilador (-O3) e DCCM

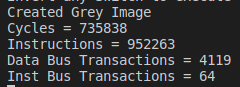


1. Faça a mesma análise que foi feita para o CoreMark, mas desta vez para a aplicação ImageProcessing. Um projeto PlatformIO que contém a aplicação ImageProcessing está em: *[RVfpgaPath]/RVfpga/Labs/Lab20/RealBenchmarks/ImageProcessing\_HwCounters.* Essas são as aplicações que usámos no Lab 5 para transformar 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*.

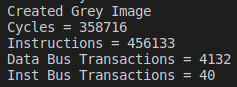
* Sem otimizações de compilador, sem DCCM, sem ICCM



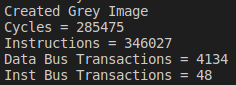
* Com DCCM



* Com otimizações do compilador (-O2) e DCCM



* Com otimizações do compilador (-O3) e DCCM



1. 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.