TASKS

<u>TASK</u>: Replicate the simulation from Figure 3 on your own computer. To do so, follow the next steps (as described in detail in Section 7 of the GSG):

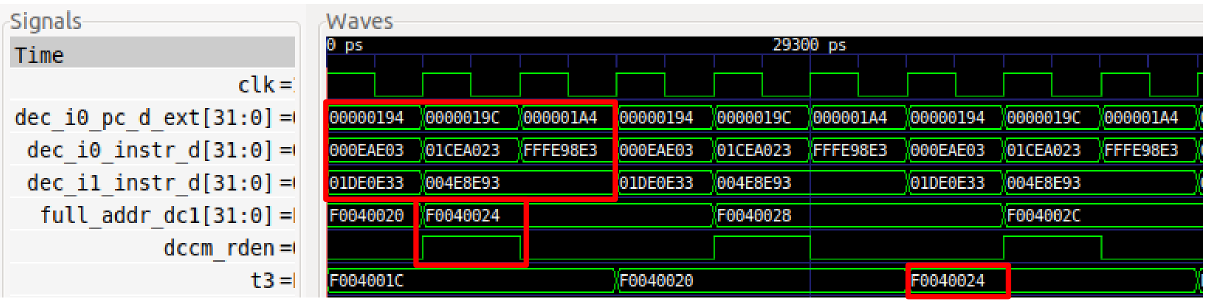
- If necessary, generate the simulation binary (*Vrvfpgasim*).
- In PlatformIO, open the project provided at: *[RVfpgaPath]/RVfpga/Labs/Lab19/LW-SW_Instruction_ExtMemory*.
- Establish the correct path to the RVfpga simulation binary (*Vrvfpgasim*) in file *platformio.ini*.
- Generate the simulation trace using Verilator (Generate Trace).
- Open the trace on GTKWave.
- Use file *test_Blocking_Extended.tcl* (provided at *[RVfpgaPath]/RVfpga/Labs/Lab19/LW-SW_Instruction_ExtMemory*) for opening the same signals as the ones shown in Figure 6. For that purpose, on GTKWave, click on *File → Read Tcl Script File* and select the *test_Blocking_Extended.tcl* file.
- Click on *Zoom In* () several times and analyse the region starting at 42500 ps.

Solution provided in the main document of Lab 19.

<u>TASK</u>: Using the HW Counters, measure the number of cycles, instructions, loads and stores in the program from Figure 2. How much time in total (both for reading and writing) does it take to access the DDR External Memory? You can compare the execution when using the DDR memory as in Figure 3 and when using the DCCM (another PlatformIO project is provided at *[RVfpgaPath]/RVfpga/Labs/Lab19/LW-SW_Instruction_DCCM/*, which contains the same program prepared for reading from / writing to the DCCM). Remember that the simulated memory is not the same as the actual DDR memory on the Nexys A7 board.

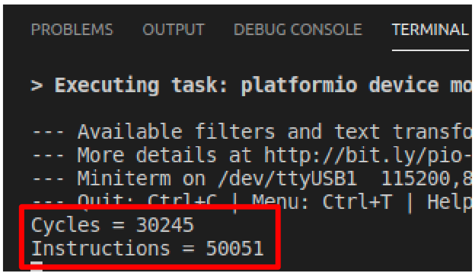
DCCM:

<u>Simulation in Verilator</u>:



Each iteration executes 5 instructions in 3 cycles. Only half a cycle is lost per iteration.

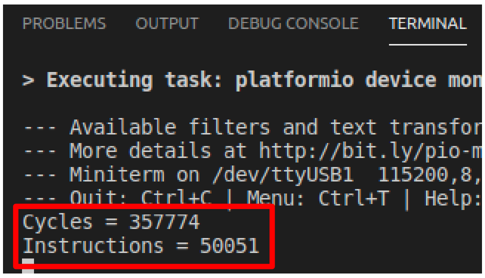
<u>Execution on the Board</u>:



Cycles per iteration = 3

DDR Memory:

<u>Execution on the Board</u>:



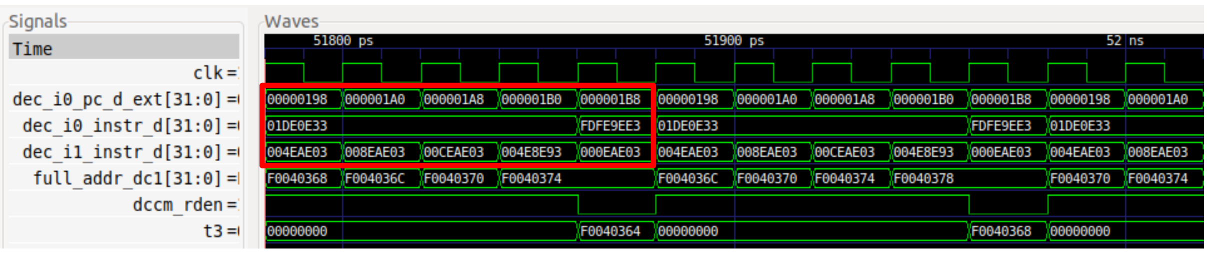
The number of instructions is the same, since the program is the same. However, now around 358000 cycles are necessary for executing all the iterations, thus:

Number of cycles spent accessing memory per iteration ≈ (358000 - 30000) / 10000 ≈ 33

<u>TASK</u>: Use the example from *[RVfpgaPath]/RVfpga/Labs/Lab19/LW_Instruction_ExtMem* to estimate the DDR External Memory read latency using the HW Counters. As in the previous task, you can use the example from *[RVfpgaPath]/RVfpga/Labs/Lab19/LW_Instruction_DCCM* to compare with a program with no stalls due to the memory accesses. Remember that the simulated memory is not the same as the actual DDR memory on the Nexys A7 board.

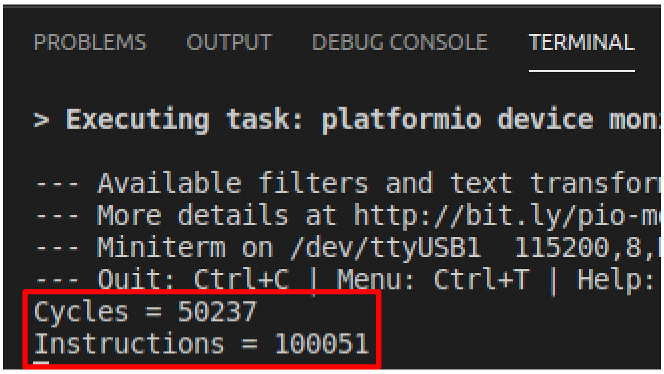
DCCM:

<u>Simulation in Verilator</u>:



Each iteration executes 10 instructions in 5 cycles, so it executes with the ideal IPC.

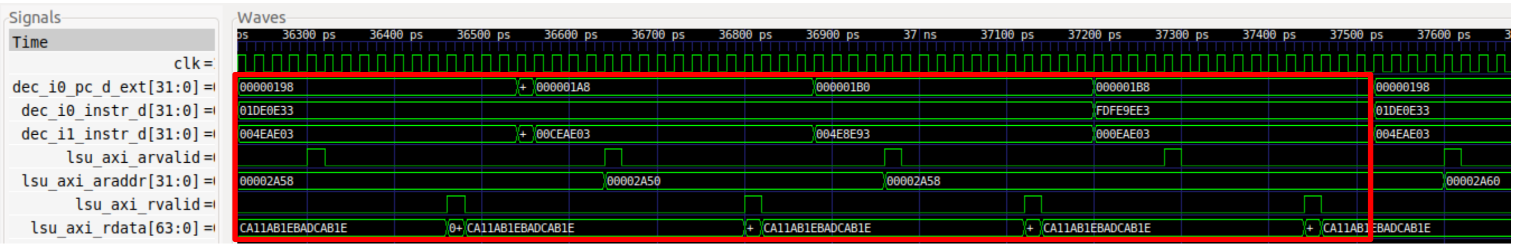
Execution on the Board:



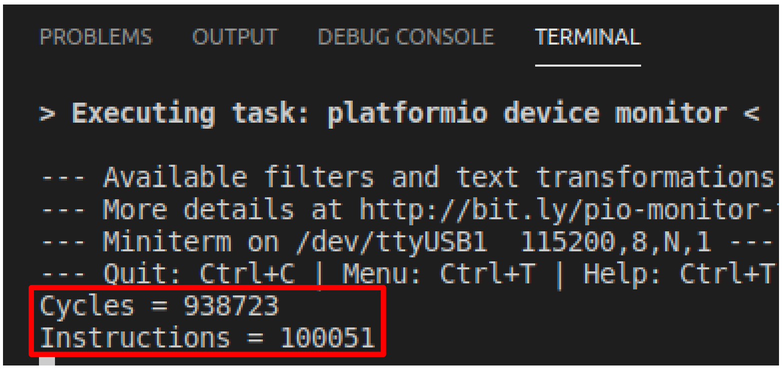
Cycles per iteration = 5

DDR Memory:

Simulation in Verilator:



Execution on the Board:

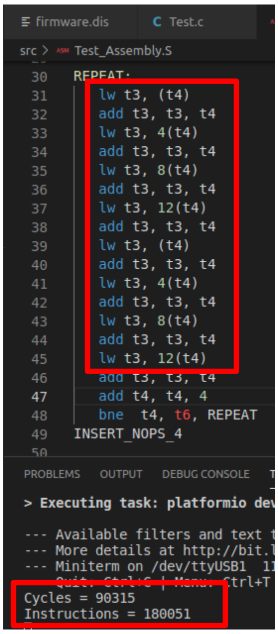


The number of instructions is the same, since the program is the same. However, now around 939000 cycles are necessary for executing all the iterations, thus:

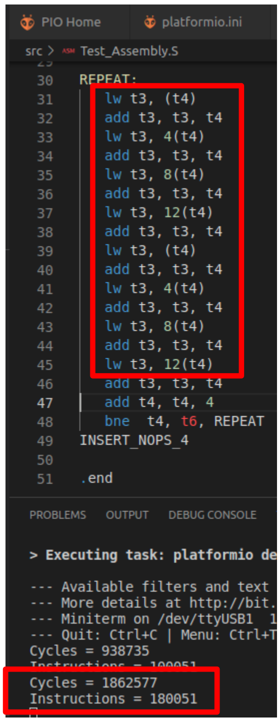
Latency of a DDR memory read ≈ (939000 - 50000) / (10000 * 4) ≈ 22

To check if it is correct, we double the number of load instructions and execute the program again:

DCCM:



DDR Memory:



Latency of a DDR memory read ≈ (1862000 - 90000) / (10000 * 8) ≈ 22

<u>TASK</u>: A quite complex but very interesting exercise is to analyse the Memory Controller used in the RVfpga System. Remember that you can find the modules that make up this controller in folder *[RVfpgaPath]/RVfpga/src/LiteDRAM*, and that the top module is implemented in file *litedram_top.v* inside that folder. You can start with the simulation from Figure 3 and add and analyse some signals from the LiteDRAM controller.

Solution not provided.

<u>TASK</u>: Analyse module **ifu_ic_mem** to understand how the elements in Figure 4 are implemented.

Module **ifu_ic_mem**:

Data Array and Tag Array instantiation:

```
IC_TAG #( .ICACHE_TAG_HIGH(ICACHE_TAG_HIGH) ,
          .ICACHE_TAG_LOW(ICACHE_TAG_LOW) ,
          .ICACHE_TAG_DEPTH(ICACHE_TAG_DEPTH)
          ) ic_tag_inst
        (
         .*,
         .ic_wr_en     (ic_wr_en[3:0]),
         .ic_debug_addr(ic_debug_addr[ICACHE_TAG_HIGH-1:2]),
         .ic_rw_addr   (ic_rw_addr[31:3])
         ) ;

IC_DATA #( .ICACHE_TAG_HIGH(ICACHE_TAG_HIGH) ,
           .ICACHE_TAG_LOW(ICACHE_TAG_LOW) ,
           .ICACHE_IC_DEPTH(ICACHE_IC_DEPTH)
          ) ic_data_inst
        (
         .*,
         .ic_wr_en     (ic_wr_en[3:0]),
         .ic_debug_addr(ic_debug_addr[ICACHE_TAG_HIGH-1:2]),
         .ic_rw_addr   (ic_rw_addr[ICACHE_TAG_HIGH-1:2])
         ) ;
```

Data Array plus Parity bits (In our case RV_ICACHE_ECC is not defined):

```
for (genvar i=0; i<NUM_WAYS; i++) begin: WAYS


   for (genvar k=0; k<NUM_SUBBANKS; k++) begin: SUBBANKS   // 16B subbank

      // way3-bank3, way3-bank2, ... way0-bank0
      assign  ic_bank_way_clken[i][k]   = ic_bank_read[k] |  ic_b_sb_wren[k][i];

      rvoclkhdr bank_way_bank_c1_cgc  ( .en(ic_bank_way_clken[i][k] | clk_override), .l1clk(ic_bank_way_clk[i][k]), .* );

   `ifdef RV_ICACHE_ECC
    `RV_ICACHE_DATA_CELL  ic_bank_sb_way_data (
                                .CLK(ic_bank_way_clk[i][k]),
                                .WE (ic_b_sb_wren[k][i]),
                                .D  (ic_sb_wr_data[k][41:0]),
                                .ADR(ic_rw_addr_q[ICACHE_TAG_HIGH-1:4]),
                                .Q  (wb_dout[i][(k+1)*42-1:k*42])
                               );
   `else
    `RV_ICACHE_DATA_CELL  ic_bank_sb_way_data (
                                .CLK(ic_bank_way_clk[i][k]),
                                .WE (ic_b_sb_wren[k][i]),
                                .D  (ic_sb_wr_data[k][33:0]),
                                .ADR(ic_rw_addr_q[ICACHE_TAG_HIGH-1:4]),
                                .Q  (wb_dout[i][(k+1)*34-1:k*34])
                               );
   `endif
   end // block: SUBBANKS

 end
```

4-1 Multiplexer:

```
`else
   logic       [135:0] ic_premux_data_ext;
   logic [3:0] [135:0] wb_dout_way;
   logic [3:0] [135:0] wb_dout_way_with_premux;

   assign ic_premux_data_ext[135:0]   = {2'b0,ic_premux_data[127:96],2'b0,ic_premux_data[95:64] ,2'b0,ic_premux_data[63:32],2'b0,ic_premux_data[31:0]};
   assign wb_dout_way[0][135:0]       = wb_dout[0][135:0] &  {  {34{ic_bank_read_ff[3]}} ,  {34{ic_bank_read_ff[2]}}  ,  {34{ic_bank_read_ff[1]}}  ,  {34{ic_bank_read_ff[0]}} };
   assign wb_dout_way[1][135:0]       = wb_dout[1][135:0] &  {  {34{ic_bank_read_ff[3]}} ,  {34{ic_bank_read_ff[2]}}  ,  {34{ic_bank_read_ff[1]}}  ,  {34{ic_bank_read_ff[0]}} };
   assign wb_dout_way[2][135:0]       = wb_dout[2][135:0] &  {  {34{ic_bank_read_ff[3]}} ,  {34{ic_bank_read_ff[2]}}  ,  {34{ic_bank_read_ff[1]}}  ,  {34{ic_bank_read_ff[0]}} };
   assign wb_dout_way[3][135:0]       = wb_dout[3][135:0] &  {  {34{ic_bank_read_ff[3]}} ,  {34{ic_bank_read_ff[2]}}  ,  {34{ic_bank_read_ff[1]}}  ,  {34{ic_bank_read_ff[0]}} };

   assign wb_dout_way_with_premux[0][135:0]  =  ic_sel_premux_data ? ic_premux_data_ext[135:0] : wb_dout_way[0][135:0] ;
   assign wb_dout_way_with_premux[1][135:0]  =  ic_sel_premux_data ? ic_premux_data_ext[135:0] : wb_dout_way[1][135:0] ;
   assign wb_dout_way_with_premux[2][135:0]  =  ic_sel_premux_data ? ic_premux_data_ext[135:0] : wb_dout_way[2][135:0] ;
   assign wb_dout_way_with_premux[3][135:0]  =  ic_sel_premux_data ? ic_premux_data_ext[135:0] : wb_dout_way[3][135:0] ;

   assign ic_rd_data[135:0]       = ({136{ic_rd_hit_q[0] | ic_sel_premux_data}} &  wb_dout_way_with_premux[0][135:0]) |
                                    ({136{ic_rd_hit_q[1] | ic_sel_premux_data}} &  wb_dout_way_with_premux[1][135:0]) |
                                    ({136{ic_rd_hit_q[2] | ic_sel_premux_data}} &  wb_dout_way_with_premux[2][135:0]) |
                                    ({136{ic_rd_hit_q[3] | ic_sel_premux_data}} &  wb_dout_way_with_premux[3][135:0]) ;

`endif
```

Tag Array plus Parity bits (In our case RV_ICACHE_ECC is not defined):

```
for (genvar i=0; i<NUM_WAYS; i++) begin: WAYS

   rvoclkhdr ic_tag_c1_cgc  ( .en(ic_tag_clken[i]), .l1clk(ic_tag_clk[i]), .* );

  if (ICACHE_TAG_DEPTH == 64 ) begin : ICACHE_SZ_16
   `ifdef RV_ICACHE_ECC
      ram_64x25  ic_way_tag (
                             .CLK(ic_tag_clk[i]),
                             .WE (ic_tag_wren_q[i]),
                             .D  (ic_tag_wr_data[24:0]),
                             .ADR(ic_rw_addr_q[ICACHE_TAG_HIGH-1:ICACHE_TAG_LOW]),
                             .Q  (ic_tag_data_raw[i][24:0])
                            );


      assign w_tout[i][31:ICACHE_TAG_HIGH] = ic_tag_data_raw[i][31-ICACHE_TAG_HIGH:0] ;
      assign w_tout[i][36:32]              = ic_tag_data_raw[i][24:20] ;

      rvecc_decode  ecc_decode (
                   .en(~dec_tlu_core_ecc_disable),
                   .sed_ded ( 1'b1 ),    // 1 : means only detection
                   .din({12'b0,ic_tag_data_raw[i][19:0]}),
                   .ecc_in({2'b0, ic_tag_data_raw[i][24:20]}),
                   .dout(ic_tag_corrected_data_unc[i][31:0]),
                   .ecc_out(ic_tag_corrected_ecc_unc[i][6:0]),
                   .single_ecc_error(ic_tag_single_ecc_error[i]),
                   .double_ecc_error(ic_tag_double_ecc_error[i]));

       assign ic_tag_way_perr[i]= ic_tag_single_ecc_error[i] | ic_tag_double_ecc_error[i]  ;
   `else
      ram_64x21  ic_way_tag (
                             .CLK(ic_tag_clk[i]),
                             .WE (ic_tag_wren_q[i]),
                             .D  (ic_tag_wr_data[20:0]),
                             .ADR(ic_rw_addr_q[ICACHE_TAG_HIGH-1:ICACHE_TAG_LOW]),
                             .Q  (ic_tag_data_raw[i][20:0])
                            );

      assign w_tout[i][31:ICACHE_TAG_HIGH] = ic_tag_data_raw[i][31-ICACHE_TAG_HIGH:0] ;
      assign w_tout[i][32]                 = ic_tag_data_raw[i][20] ;

      rveven_paritycheck #(32-ICACHE_TAG_HIGH) parcheck(.data_in   (w_tout[i][31:ICACHE_TAG_HIGH]),
                                                        .parity_in (w_tout[i][32]),
                                                        .parity_err(ic_tag_way_perr[i]));
   `endif

end // block: ICACHE_SZ_16
```

Comparators:

```
assign ic_rd_hit[0] = (w_tout[0][31:ICACHE_TAG_HIGH] == ic_rw_addr_ff[31:ICACHE_TAG_HIGH]) & ic_tag_valid[0];
assign ic_rd_hit[1] = (w_tout[1][31:ICACHE_TAG_HIGH] == ic_rw_addr_ff[31:ICACHE_TAG_HIGH]) & ic_tag_valid[1];
assign ic_rd_hit[2] = (w_tout[2][31:ICACHE_TAG_HIGH] == ic_rw_addr_ff[31:ICACHE_TAG_HIGH]) & ic_tag_valid[2];
assign ic_rd_hit[3] = (w_tout[3][31:ICACHE_TAG_HIGH] == ic_rw_addr_ff[31:ICACHE_TAG_HIGH]) & ic_tag_valid[3];
```

TASK: Replicate the simulation from Figure 6 on your own computer. To do so, follow the next steps (as described in detail in Section 7 of the GSG):

- If necessary, generate the simulation binary (*Vrvfpgasim*).
- In PlatformIO, open the project provided at: *[RVfpgaPath]/RVfpga/Labs/Lab19/InstructionMemory_Example*.
- Update the path to the RVfpga simulation binary (*Vrvfpgasim*) in file *platformio.ini*.
- Generate the simulation trace with Verilator (Generate Trace).
- Open the trace on GTKWave.
- Use file *test1_Miss.tcl* (provided at *[RVfpgaPath]/RVfpga/Labs/Lab19/InstructionMemory_Example*) for opening the same signals as the ones shown in Figure 6. For that purpose, on GTKWave, click on *File → Read Tcl Script File* and select the *test1_Miss.tcl* file.
- Click on *Zoom In* ( ) several times and analyse the region from 28900 ps to 30220

ps.

You can also analyse some things in more detail, such as the write to the I$ or the bypass of the initial instructions.

Solution provided in the main document of Lab 19.

TASK: Replicate the simulation from Figure 7 on your own computer. Use file *test1_Hit.tcl* (provided at *[RVfpgaPath]/RVfpga/Labs/Lab19/InstructionMemory_Example*). *Zoom In* () several times and move to 34680ps.

Solution provided in the main document of Lab 19.

TASK: Analyse the Verilog code from Figure 9 and explain how it operates based on the above explanations.

Solution not provided.

TASK: Analyse the Verilog code from Figure 10 and explain how it operates based on the above explanations.

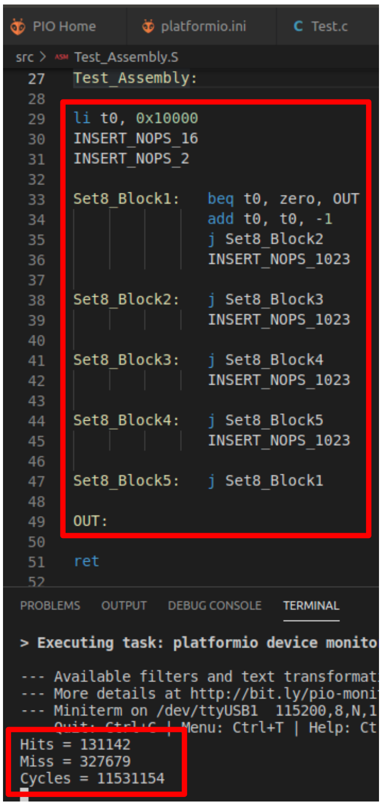
Solution not provided.

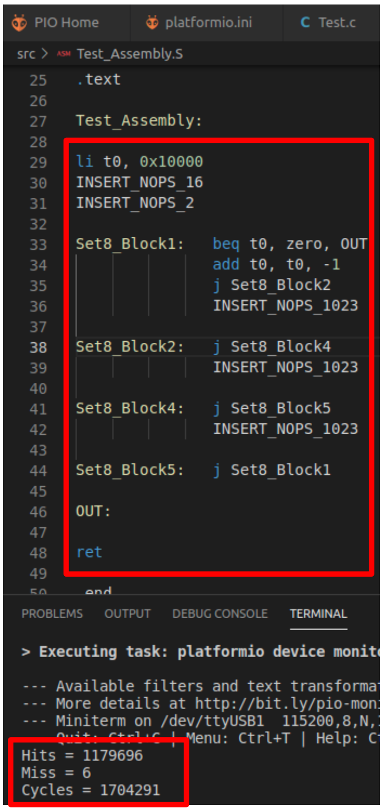
1. EXERCISES

1) Transform the infinite loop from Figure 11 into a loop with 0x10000 iterations, but keep the `j` instructions at the same addresses. Measure the number of cycles and I$ hits and misses. Then remove one of the `j` instructions and measure the same metrics. Compare and explain the results.

5 jump instructions:

4 jump instructions:





In the program with 4 `j` instructions the number of I$ misses and the number of cycles decrease drastically, as now only the blocks do not conflict with each other. At the same time, the number of I$ hits increases a lot.

2) Use the program from Figure 5 to analyse an I$ hit from the point of view of the I$ Replacement Policy.

Solution not provided.

3) Extend Figure 6 to analyse in detail how each 64-bit chunk is written in the I$.

Solution not provided.

4) Analyse in simulation and on the board other I$ configurations, such as an I$ with a different block size. Recall that the number of ways cannot be modified.

Solution not provided.

5) Analyse the logic that checks the correctness of the parity information from the Data Array and from the Tag Array.

Solution not provided.