


任务

任务： 在自己的计算机上重复图3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）：

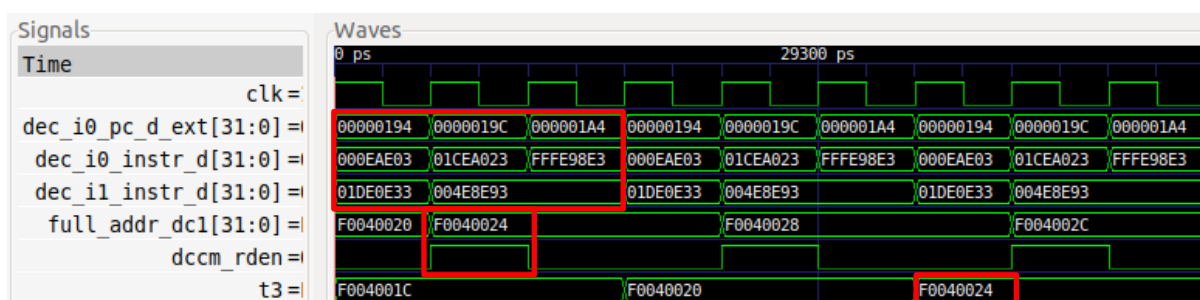
- 必要时生成仿真二进制文件（*Vrvfpgasim*）。
- 在PlatformIO中，打开在以下位置提供的项目：*[RVfpgaPath]/RVfpga/Labs/Lab19/LW-SW_Instruction_ExtMemory*。
- 在文件*platformio.ini*中建立到RVfpga仿真二进制文件（*Vrvfpgasim*）的正确路径。
- 使用Verilator生成仿真轨迹（生成轨迹）。
- 在GTKWave上打开轨迹。
- 使用文件*test_Blocking_Extended.tcl*（在为*[RVfpgaPath]/RVfpga/Labs/Lab19/LW-SW_Instruction_ExtMemory*中提供）打开与图6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*test_Blocking_Extended.tcl*文件。
- 单击几次“Zoom In”（放大）（），然后分析自42500 ps起的区域。

解答请参见实验19的主文档。

任务： 使用硬件计数器测量图2中程序的周期数、指令数、装载次数和存储次数。访问DDR外部存储器所用的总时间是多少（包括读访问和写访问）？可以比较使用图3中的DDR存储器与使用DCCM时的执行情况（*[RVfpgaPath]/RVfpga/Labs/Lab19/LW-SW_Instruction_DCCM*下提供另一个PlatformIO项目，其中包含用于对DCCM进行读/写操作的相同程序）。请注意，用于仿真的存储器不是Nexys A7开发板上实际使用的DDR存储器。

DCCM:

在Verilator中仿真：



每次迭代需要3个周期，执行5条指令。每次迭代仅丢失半个周期。

在开发板上执行：

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

> Executing task: platformio device mon

--- Available filters and text transform
--- More details at http://bit.ly/pio-
--- Miniterm on /dev/ttyUSB1 115200,8
--- Quit: Ctrl+C | Menu: Ctrl+T | Help:

Cycles = 30245
Instructions = 50051

```

每次迭代的周期数 = 3

DDR存储器:

在开发板上执行:

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

> Executing task: platformio device mon

--- Available filters and text transform
--- More details at http://bit.ly/pio-
--- Miniterm on /dev/ttyUSB1 115200,8
--- Quit: Ctrl+C | Menu: Ctrl+T | Help:

Cycles = 357774
Instructions = 50051

```

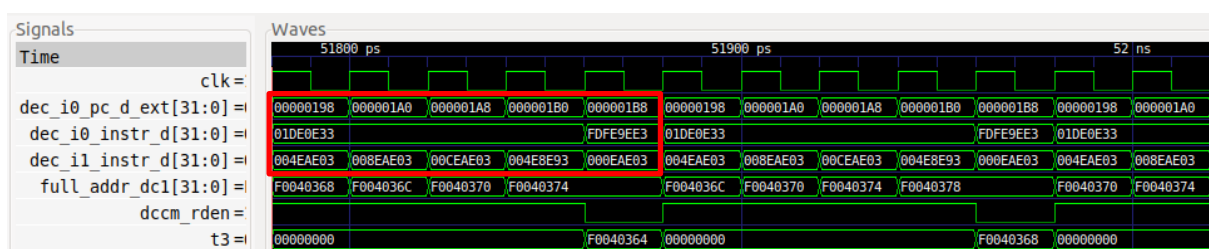
因为程序未发生变化, 所以指令数相同。但此时执行所有迭代需要大约358000个周期, 因此:

每次迭代中访问存储器所用的周期数 $\approx (358000 - 30000) / 10000 \approx 33$

任务: 使用[RVfpgaPath]/RVfpga/Labs/Lab19/LW_Instruction_ExtMem中提供的示例, 借助硬件计数器估算DDR外部存储器的读延时。与上一任务一样, 可以使用[RVfpgaPath]/RVfpga/Labs/Lab19/LW_Instruction_DCCM中的示例, 将现有程序与因存储器访问而不存在暂停的程序进行比较。请注意, 用于仿真的存储器不是Nexys A7开发板上实际使用的DDR存储器。

DCCM:

在Verilator中仿真:



每次迭代需要5个周期, 执行10条指令, 因此IPC为理想值。

在开发板上执行：

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

> Executing task: platformio device mon

--- Available filters and text transform
--- More details at http://bit.ly/pio-m
--- Miniterm on /dev/ttyUSB1 115200,8,
--- Quit: Ctrl+C | Menu: Ctrl+T | Help:

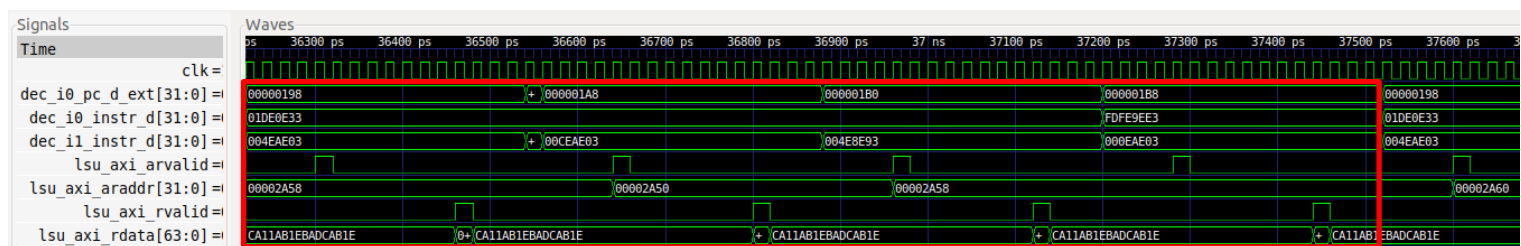
Cycles = 50237
Instructions = 100051

```

每次迭代的周期数 = 5

DDR存储器：

在Verilator中仿真：



在开发板上执行：

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

> Executing task: platformio device monitor <

--- Available filters and text transformations
--- More details at http://bit.ly/pio-monitor-
--- Miniterm on /dev/ttyUSB1 115200,8,N,1 ---
--- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T

Cycles = 938723
Instructions = 100051

```

因为程序未发生变化，所以指令数相同。但此时执行所有迭代需要大约939000个周期，因此：

DDR存储器的读延时 $\approx (939000 - 50000) / (10000 * 4) \approx 22$

为了检查该值是否正确，我们将装载指令的数量加倍，然后再次执行程序：

DCCM:

```

firmware.dis  Test.C
src > Test_Assembly.S
30 REPEAT:
31     lw t3, (t4)
32     add t3, t3, t4
33     lw t3, 4(t4)
34     add t3, t3, t4
35     lw t3, 8(t4)
36     add t3, t3, t4
37     lw t3, 12(t4)
38     add t3, t3, t4
39     lw t3, (t4)
40     add t3, t3, t4
41     lw t3, 4(t4)
42     add t3, t3, t4
43     lw t3, 8(t4)
44     add t3, t3, t4
45     lw t3, 12(t4)
46     add t3, t3, t4
47     add t4, t4, 4
48     bne t4, t6, REPEAT
49 INSERT_NOPS_4
50
PROBLEMS  OUTPUT  DEBUG CONSOLE  T
> Executing task: platformio de
--- Available filters and text t
--- More details at http://bit.l
--- Miniterm on /dev/ttyUSB1 1
Quit: Ctrl+C | Menu: Ctrl+T
Cycles = 90315
Instructions = 180051

```

DDR存储器:

```

PIO Home  platformio.ini
src > Test_Assembly.S
30 REPEAT:
31     lw t3, (t4)
32     add t3, t3, t4
33     lw t3, 4(t4)
34     add t3, t3, t4
35     lw t3, 8(t4)
36     add t3, t3, t4
37     lw t3, 12(t4)
38     add t3, t3, t4
39     lw t3, (t4)
40     add t3, t3, t4
41     lw t3, 4(t4)
42     add t3, t3, t4
43     lw t3, 8(t4)
44     add t3, t3, t4
45     lw t3, 12(t4)
46     add t3, t3, t4
47     add t4, t4, 4
48     bne t4, t6, REPEAT
49 INSERT_NOPS_4
50
PROBLEMS  OUTPUT  DEBUG CONSOLE
> Executing task: platformio de
--- Available filters and text
--- More details at http://bit.
--- Miniterm on /dev/ttyUSB1 1
--- Quit: Ctrl+C | Menu: Ctrl+T
Cycles = 938735
Instructions = 180051
Cycles = 1862577
Instructions = 180051

```

DDR存储器的读延时 $\approx (1862000 - 90000) / (10000 * 8) \approx 22$

任务：分析RVfpga系统中使用的存储器控制器，本练习颇为复杂但十分有趣。请记住，构成该控制器的模块位于 *[RVfpgaPath]/RVfpga/src/LiteDRAM* 中，顶层模块在该文件夹内的 *litedram_top.v* 文件中实现。可以先进行图3所示的仿真，然后添加并分析来自LiteDRAM控制器的一些信号。

不提供解答。

任务：分析模块 *ifu_ic_mem*，了解如何实现图4中的元素。

模块 *ifu_ic_mem*：

数据数组和标记数组实例化：

```
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])
    ) ;
```

数据数组加奇偶校验位（在本例中未定义RV_ICACHE_ECC）：

```

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_cgic ( .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多路开关:

```

`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

```

标记数组加奇偶校验位（在本例中未定义RV_ICACHE_ECC）:

```

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

    rvoclkhdr ic_tag_c1_cgic ( .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

```

比较器:


```

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];

```

任务： 在自己的计算机上重复图6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/Lab19/InstructionMemory_Example。
- 在文件*platformio.ini*中更新到RVfpga仿真二进制文件（*Vrvfpgasim*）的路径。
- 使用Verilator生成仿真跟踪（生成跟踪）。
- 在GTKWave上打开跟踪。
- 使用文件*test1_Miss.tcl*（在
*[RVfpgaPath]/RVfpga/Labs/Lab19/InstructionMemory_Example*中提供）打开与图6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*test1_Miss.tcl*文件。

- 单击几次“**Zoom In**”（放大）（），然后分析28900 ps至30220 ps范围内的区域。还可以进行一些更深入的分析，例如对I\$的写操作或初始指令的旁路。

解答请参见实验19的主文档。

任务：在自己的计算机上重复图7中的仿真过程。使用文件`test1_Hit.tcl`（在[RVfpgaPath]/RVfpga/Labs/Lab19/InstructionMemory_Example中提供）。单击几次“**Zoom In**”（放大）（）移动至34680 ps。

解答请参见实验19的主文档。

任务：分析图9中的Verilog代码，并基于上述说明解释代码如何运行。

不提供解答。

任务：分析图10中的Verilog代码，并基于上述说明解释代码如何运行。

不提供解答。

1. 练习

- 1) 将图11所示的循环转换为0x10000次迭代的循环，但为j指令保持原有的地址。测量周期数以及I\$命中和未命中的次数。然后删除其中一条j指令，再次测量上述指标。比较测量结果，并给出解释。

5条跳转指令：

4条跳转指令：


```

PIO Home platformio.ini Test.c
src > Test_Assembly.S
27 Test_Assembly:
28
29 li t0, 0x10000
30 INSERT_NOPS_16
31 INSERT_NOPS_2
32
33 Set8_Block1: beq t0, zero, OUT
34              add t0, t0, -1
35              j Set8_Block2
36              INSERT_NOPS_1023
37
38 Set8_Block2: j Set8_Block3
39              INSERT_NOPS_1023
40
41 Set8_Block3: j Set8_Block4
42              INSERT_NOPS_1023
43
44 Set8_Block4: j Set8_Block5
45              INSERT_NOPS_1023
46
47 Set8_Block5: j Set8_Block1
48
49 OUT:
50
51 ret
52
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
> Executing task: platformio device monitor
--- Available filters and text transformations
--- More details at http://bit.ly/pio-monitor
--- Miniterm on /dev/ttyUSB1 115200,8,N,1
Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+?
Hits = 131142
Miss = 327679
Cycles = 11531154

```

```

PIO Home platformio.ini Test.c
src > Test_Assembly.S
25 .text
26
27 Test_Assembly:
28
29 li t0, 0x10000
30 INSERT_NOPS_16
31 INSERT_NOPS_2
32
33 Set8_Block1: beq t0, zero, OUT
34              add t0, t0, -1
35              j Set8_Block2
36              INSERT_NOPS_1023
37
38 Set8_Block2: j Set8_Block4
39              INSERT_NOPS_1023
40
41 Set8_Block4: j Set8_Block5
42              INSERT_NOPS_1023
43
44 Set8_Block5: j Set8_Block1
45
46 OUT:
47
48 ret
49
50 .end
51
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
> Executing task: platformio device monitor
--- Available filters and text transformations
--- More details at http://bit.ly/pio-monitor
--- Miniterm on /dev/ttyUSB1 115200,8,N,1
Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+?
Hits = 1179696
Miss = 6
Cycles = 1704291

```

在采用4条j指令的程序中，IS未命中数和周期数显著减少，因为此时块之间彼此不会发生冲突。与此同时，IS命中数大幅增加。

2) 使用图5中的程序，从IS替换策略的角度分析IS命中。

不提供解答。

3) 扩展图6，详细分析每个64位块如何写入IS。

不提供解答。

4) 通过仿真器和开发板分析其他IS配置，例如具有不同块大小的IS。请注意，无法修改通路的数量。

不提供解答。

5) 分析用于检查数据数组和标记数组奇偶校验信息正确性的逻辑。

不提供解答。