

1. 任務

任務：在模組**dec_gpr_ctl**中實作暫存器檔案，並在模組**dec**中將其實例化（參見圖7）。分析模組**dec_gpr_ctl**的Verilog程式碼及主要訊號的模擬（位於檔案 *[RVfpgaPath]/RVfpga/src/SweRVolfSoC/SweRVEh1CoreComplex/dec/dec_gpr_ctl.sv* 中），以瞭解其工作方式。請注意，SweRV EH1處理器允許包含多個暫存器檔案，但RVfpga系統中使用的組態僅使用一個暫存器檔案（參見檔案**dec.sv**的第402行：
`localparam GPR_BANKS = 1;`）。

模組**dec**中的實例化：

```
525     dec_gpr_ctl #(.GPR_BANKS(GPR_BANKS),
526                  .GPR_BANKS_LOG2(GPR_BANKS_LOG2)) arf (.*,
527                  // inputs
528                  .raddr0(dec_i0_rs1_d[4:0]), .rden0(dec_i0_rs1_en_d),
529                  .raddr1(dec_i0_rs2_d[4:0]), .rden1(dec_i0_rs2_en_d),
530                  .raddr2(dec_i1_rs1_d[4:0]), .rden2(dec_i1_rs1_en_d),
531                  .raddr3(dec_i1_rs2_d[4:0]), .rden3(dec_i1_rs2_en_d),
532
533                  .waddr0(dec_i0_waddr_wb[4:0]), .wen0(dec_i0_wen_wb), .wd0(dec_i0_wdata_wb[31:0]),
534                  .waddr1(dec_i1_waddr_wb[4:0]), .wen1(dec_i1_wen_wb), .wd1(dec_i1_wdata_wb[31:0]),
535                  .waddr2(dec_nonblock_load_waddr[4:0]), .wen2(dec_nonblock_load_wen), .wd2(lsu_nonblock_load_data[31:0]),
536
537                  // outputs
538                  .rd0(gpr_i0_rs1_d[31:0]), .rd1(gpr_i0_rs2_d[31:0]),
539                  .rd2(gpr_i1_rs1_d[31:0]), .rd3(gpr_i1_rs2_d[31:0])
540                  );
```

模組**dec_gpr_ctl**中32個暫存器的實作：

```
66     // GPR Write Enables for power savings
67     assign gpr_wr_en[31:1] = (w0v[31:1] | w1v[31:1] | w2v[31:1]);
68     for (genvar i=0; i<GPR_BANKS; i++) begin: gpr_banks
69         assign gpr_bank_wr_en[i][31:1] = gpr_wr_en[31:1] & {31{gpr_bank_id[GPR_BANKS_LOG2-1:0] == i}};
70         for (genvar j=1; j<32; j++) begin: gpr
71             rvdffe #(32) gprff (.*, .en(gpr_bank_wr_en[i][j]), .din(gpr_in[j][31:0]), .dout(gpr_out[i][j][31:0]));
72         end: gpr
73     end: gpr_banks
74
```

本例中僅實作了1個記憶庫。這一記憶庫透過將模組**rvdffe**（位於檔案 *[RVfpgaPath]/RVfpga/src/SweRVolfSoC/SweRVEh1CoreComplex/lib/beh_lib.sv* 中）實例化31次實作31個暫存器。注意每個**rvdffe**暫存器的寬度使用參數選擇，本例中為32位元→**rvdffe** #(32)。暫存器0不是必需的，因為RISC-V架構強制其始終為0。

暫存器讀取操作：

```
86     // GPR Read logic
87     for (int i=0; i<GPR_BANKS; i++) begin
88         for (int j=1; j<32; j++) begin
89             rd0[31:0] |= ({32{rden0 & (raddr0[4:0] == 5'(j)) & (gpr_bank_id[GPR_BANKS_LOG2-1:0] == 1'(i))}} & gpr_out[i][j][31:0]);
90             rd1[31:0] |= ({32{rden1 & (raddr1[4:0] == 5'(j)) & (gpr_bank_id[GPR_BANKS_LOG2-1:0] == 1'(i))}} & gpr_out[i][j][31:0]);
91             rd2[31:0] |= ({32{rden2 & (raddr2[4:0] == 5'(j)) & (gpr_bank_id[GPR_BANKS_LOG2-1:0] == 1'(i))}} & gpr_out[i][j][31:0]);
92             rd3[31:0] |= ({32{rden3 & (raddr3[4:0] == 5'(j)) & (gpr_bank_id[GPR_BANKS_LOG2-1:0] == 1'(i))}} & gpr_out[i][j][31:0]);
93         end
94     end
95
```

實作了4個讀取連接埠。每一個讀取連接埠均分配有暫存器的值（用 **raddr0/raddr1/raddr2/raddr3** 訊號表示）。**rden0/rden1/rden2/rden3** 訊號啟用/停用讀取操作。注意的初始值為1，因此暫存器0的讀取操作始終傳回值0。

暫存器寫入操作：

```

96 // GPR Write logic
97 for (int j=1; j<32; j++ ) begin
98     w0v[j] = wen0 & (waddr0[4:0]== 5'(j) );
99     w1v[j] = wen1 & (waddr1[4:0]== 5'(j) );
100    w2v[j] = wen2 & (waddr2[4:0]== 5'(j) );
101    gpr_in[j] = ({32{w0v[j]}} & wd0[31:0]) |
102                ({32{w1v[j]}} & wd1[31:0]) |
103                ({32{w2v[j]}} & wd2[31:0]);
104 end
105 end // always_comb begin

```

實作了3個寫入連接埠。每個暫存器寫入訊號wd0/wd1/wd2中提供的值，具體取決於暫存器位址waddr0/waddr1/waddr2。wen0/wen1/wen2訊號啟用/停用寫入操作。注意j的初始值為1，因此沒有寫入暫存器0。

任務：從圖8分析多工器的控制位元。請注意，控制位元在訊號e3d中，該訊號由訊號dd經管線處理得到，後一個訊號由控制單元在解碼階段產生（有關控制位元的說明，請參見SweRVref.docx）。

- 如果DC3階段的指令有效（e3d.i0v == 1）且為load指令（e3d.i0load == 1），則選擇來自LSU管線的值：i0_result_e3_final = lsu_result_dc3。
- 如果EX3階段的指令有效（e3d.i0v == 1）且為mul指令（e3d.i0mul == 1），則選擇來自乘法器的值：i0_result_e3_final = exu_mul_result_e3。
- 否則，將選擇來自I0管線的值：i0_result_e3_final = i0_result_e3。

任務：從圖9分析多工器的控制位元，這些控制位元位於模組dec_decode_ctl中。

- 如果EX4階段的結果必須從I0輔助ALU中選擇（e4d.i0secondary == 1），則選擇來自I0輔助ALU的值：i0_result_e4_final = exu_i0_result_e4。我們將在實驗15中分析輔助ALU操作。
- 如果DC4階段的指令有效（e4d.i0v == 1）且為load指令（e4d.i0load == 1），則選擇來自LSU管線的值：i0_result_e4_final = lsu_result_corr_dc4。
- 否則，將選擇來自I0管線的值：i0_result_e4_final = i0_result_e4。

任務：按照以下步驟（如GSG第7部分中詳述）在自己的電腦上重複圖11和圖12中的模擬過程：

- 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/Lab11/ExampleProgram。

- 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）。
- 使用GTKWave開啟軌跡。
- 使用檔案`test_1.tcl`和`test_2.tcl`（在`[RVfpgaPath]/RVfpga/Labs/Lab11/ExampleProgram`中提供）開啟與圖11和圖12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），然後選擇`test_1.tcl`或`test_2.tcl`檔案。
- 按幾次「**Zoom In**」（放大）（）移動至48500 ps（或迴圈的任何其他迭代，第一次迭代除外）。

解答請參見實驗11的主文件。

任務：按照GSG所述在Nexys A7板上執行圖13中的程式。對於測量的四個事件，應獲得圖14所示的結果。解釋並證明結果。

```
lui t2, 0xF4
add t2, t2, 0x240
nop

REPEAT:
    add t0, t0, 1
    add t3, t3, t1
    sub t4, t4, t1
    or  t5, t5, t1
    xor t6, t6, t1
    bne t0, t2, REPEAT
```

程式由1000000次迭代的迴圈構成，該迴圈包含5條算術邏輯指令和一個條件分支。未發生由於冒險引起的暫停，因此：

- 執行 $6 * 1000000$ 條指令
- 每個週期執行2條指令，因此： $(6/2) * 1000000$ 個週期
- 執行1000000個分支，據預測幾乎所有分支均命中。

任務：在圖13所示程式的硬體計數器中測量其他事件。為此，必須使用`pspPerformanceCounterSet`函數在`Test.c`檔案中變更待測量事件的組態。請注意，可以使用WD PSP檔案中定義的巨集參考不同的事件（如圖1所示）：

`.platformio/packages/framework-wd-riscv-sdk/psp/api_inc/psp_performance_monitor_eh1.h`。

例如，如果要測量IS未命中數而不是分支未命中數，則必須在檔案中將`Test.c`行：

`pspPerformanceCounterSet(D_PSP_COUNTER3, E_BRANCHES_MISPREDICTED);`

替換為行：`pspPerformanceCounterSet(D_PSP_COUNTER3, E_I_CACHE_MISSES);`

不提供解答。

任務：在Test_Assembly函數中提供其他程式並檢查不同的事件是否提供了預期的結果。可以嘗試其他指令，例如載入、儲存、乘法、除法...以及引發管線暫停的冒險。

不提供解答。