

1. TAREFAS

TAREFA: O Register File é implementado no módulo **dec_gpr_ctl** e é instanciado no módulo **dec** (consulte a Figura 7). Analise o código Verilog e a simulação dos principais sinais do módulo **dec_gpr_ctl** (disponível no ficheiro `[RVfpgaPath]/RVfpga/src/SweRVolfSoC/SweRVEh1CoreComplex/dec/dec_gpr_ctl.sv`) para entender como ele funciona. Observe que o processador SweRV EH1 permite a inclusão de vários Register Files, mas a configuração usada no sistema RVfpga usa apenas um Register File (consulte a linha 402 do ficheiro `dec.sv`: `localparam GPR_BANKS = 1;`).

Instanciação no módulo dec:

```
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               );
```

Implementação dos 32 registos no módulo dec_gpr_ctl:

```
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
```

No nosso caso, apenas um banco é implementado. Para esse único banco, 31 registos são implementados instanciando 31 vezes o módulo **rvdffe** (que pode ser encontrado no ficheiro `[RVfpgaPath]/RVfpga/src/SweRVolfSoC/SweRVEh1CoreComplex/lib/beh_lib.sv`). Observe que a largura de cada registo **rvdffe** é seleccionada usando um parâmetro, que, no nosso caso, é de 32 bits→ `rvdffe #(32)`. O registo 0 não é necessário, pois a arquitetura RISC-V força-o a ser sempre 0.

Leitura de registos:

```
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
```

São implementados 4 portos de leitura. Cada uma é atribuída com o valor do registo indicado pelos sinais `raddr0/raddr1/raddr2/raddr3`. Os sinais `rden0/rden1/rden2/rden3` ativam/desativam a leitura. Observe que o valor inicial de `j` é 1, portanto, a leitura do registo 0 devolve sempre o valor 0.

Escrita de registos:

```

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

```

São implementados 3 portos de escrita. Cada registo é escrito com o valor fornecido nos sinais `wd0/wd1/wd2`, dependendo do endereço de registo `waddr0/waddr1/waddr2`. Os sinais `wen0/wen1/wen2` ativam/desativam a escrita. Observe que o valor inicial de `j` é 1, portanto, não há escrita do registo 0.

TAREFA: Analise os bits de controle do multiplexer da Figura 8. Observe que os bits de controle estão no sinal `e3d`, que foi registado (pipelined) a partir do sinal `dd`, que foi gerado no andar Decode pela Unidade de Controle (consulte `SweRVref.docx` para obter descrições dos bits de controle).


- Se a instrução no DC3 for válida (`e3d.i0v == 1`) e for uma instrução `load` (`e3d.i0load == 1`), o valor proveniente do Pipe LSU será selecionado:
`i0_result_e3_final = lsu_result_dc3.`
- Se a instrução em EX3 for válida (`e3d.i0v == 1`) e for uma instrução `mul` (`e3d.i0mul == 1`), o valor proveniente do Multiplicador será selecionado:
`i0_result_e3_final = exu_mul_result_e3.`
- Caso contrário, o valor proveniente do Pipe I0 é selecionado:
`i0_result_e3_final = i0_result_e3.`

TAREFA: Analise os bits de controle do multiplexer da Figura 9, que podem ser encontrados no módulo `dec_decode_ctl`.

- Se o resultado em EX4 precisar ser selecionado na ALU secundária I0 (`e4d.i0secondary == 1`), o valor proveniente da ALU secundária I0 será selecionado: `i0_result_e4_final = exu_i0_result_e4`. Analisaremos a operação da ALU secundária no Lab 15.
- Se a instrução no DC4 for válida (`e4d.i0v == 1`) e for uma instrução `load` (`e4d.i0load == 1`), o valor proveniente do Pipe LSU será selecionado:
`i0_result_e4_final = lsu_result_corr_dc4.`
- Caso contrário, o valor proveniente do Pipe I0 é selecionado:
`i0_result_e4_final = i0_result_e4.`

TAREFA: Replique a simulação da Figura 11 e da Figura 12 no seu computador seguindo

estas 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/Lab11/ExampleProgram.
- Defina o caminho correto para o binário de simulação do RVfpga (*Vrvfpgasim*) no ficheiro *platformio.ini*.
- Gere o trace da simulação com o Verilator (Generate Trace).
- Abra o trace usando o GTKWave.
- Use os ficheiros *test_1.tcl* e *test_2.tcl* (fornecidos em *[RVfpgaPath]/RVfpga/Labs/Lab11/ExampleProgram*) para abrir os mesmos sinais que os mostrados na Figura 11 e na Figura 12. Para isso, no GTKWave, clique em *File* → *Read Tcl Script File* e selecione o ficheiro *test_1.tcl* ou *test_2.tcl*.
- Clique em *Zoom In* () várias vezes e vá para 48500ps (ou qualquer outra iteração do loop, exceto a primeira).

Solução fornecida no documento principal do Lab 11.

TAREFA: Execute o programa da Figura 13 na placa Nexys A7, conforme explicado no GSG. Você deve obter os resultados mostrados na Figura 14 para os quatro eventos medidos. Explique e justifique os resultados.

```
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
```

O programa é feito por um loop de 1000000 iterações que inclui 5 instruções aritméticas-lógicas e um salto condicional. Portanto, não há paragens devido a conflitos:

- o 6 * 1000000 instruções são executadas
- o São executadas 2 instruções por ciclo, portanto: (6/2) * 1000000 ciclos
- o 1000000 saltos são executados e quase todos eles acertam na previsão.

TAREFA: Medir outros eventos nos contadores em hardware para o programa da Figura 13. Para isso, você deve alterar no ficheiro *Test.c* a configuração dos eventos a serem medidos com a função `pspPerformanceCounterSet`. Observe que os diferentes eventos (mostrados na Tabela 1) podem ser referenciados usando as macros definidas no ficheiro PSP da WD: *.platformio/packages/framework-wd-riscv-sdk/psp/api_inc/psp_performance_monitor_eh1.h*. Por exemplo, se quiser medir o número de erros da *I\$* em vez do número de erros de salto, você deve substituir no ficheiro *Test.c* a linha: `pspPerformanceCounterSet(D_PSP_COUNTER3, E_BRANCHES_MISPREDICTED);`

para a linha: `pspPerformanceCounterSet(D_PSP_COUNTER3, E_I_CACHE_MISSES);`

Solução não fornecida.

TAREFA: Proponha outros programas na função `Test_Assembly` e verifique se os diferentes eventos fornecem os resultados esperados. Pode tentar outras instruções tais como leituras, escritas, multiplicações, divisões... bem como conflitos que provocam paradas no pipeline.

Solução não fornecida.