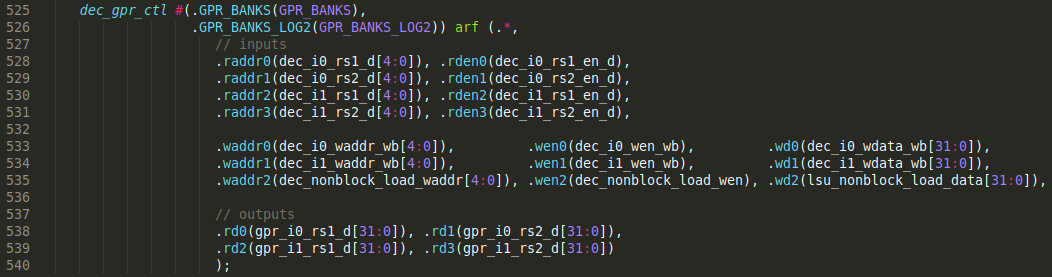
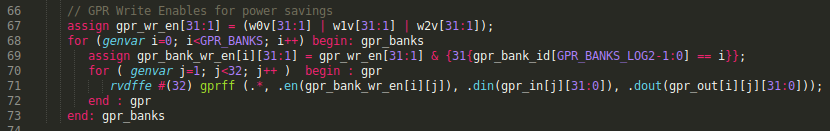
# 1. 任务

**任务：**在模块**dec\_gpr\_ctl**中实现寄存器文件，并在模块**dec**中将其实例化（参见图7）。分析模块**dec\_gpr\_ctl**的Verilog代码及主要信号的仿真（位于文件*[RVfpgaPath]/RVfpga/src/SweRVolfSoC/SweRVEh1CoreComplex/dec/dec\_gpr\_ctl.sv*中），以了解其工作方式。请注意，SweRV EH1处理器允许包含多个寄存器文件，但RVfpga系统中使用的配置仅使用一个寄存器文件（参见文件*dec.sv*的第402行：  
localparam GPR\_BANKS = 1;）。

**模块dec中的实例化**：

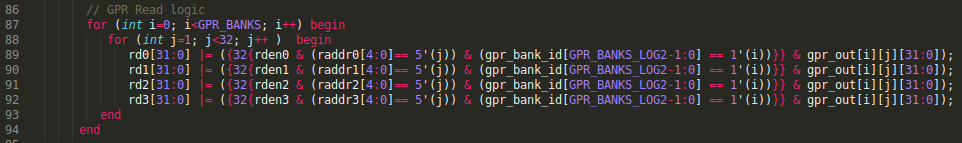


**模块dec\_gpr\_ctl中32个寄存器的实现**：



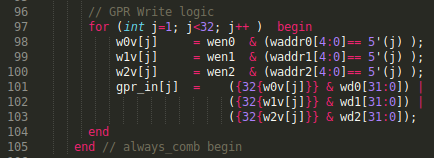
本例中仅实现了1个存储区。这一存储区通过将模块**rvdffe**（位于文件*[RVfpgaPath]/RVfpga/src/SweRVolfSoC/SweRVEh1CoreComplex/lib/beh\_lib.sv*中）实例化31次实现31个寄存器。注意每个**rvdffe**寄存器的宽度使用参数选择，本例中为32位🡪 rvdffe #(32)。寄存器0不是必需的，因为RISC-V架构强制其始终为0。

**寄存器读操作**：



实现了4个读端口。每一个读端口均分配有寄存器的值（用raddr0/raddr1/raddr2/raddr3信号表示）。rden0/rden1/rden2/rden3信号允许/禁止读操作。注意*j*的初始值为1，因此寄存器0的读操作始终返回值0。

**寄存器写操作**：



实现了3个写端口。每个寄存器写入信号wd0/wd1/wd2中提供的值，具体取决于寄存器地址waddr0/waddr1/waddr2。wen0/wen1/wen2信号使能/禁止写操作。注意*j*的初始值为1，因此没有写入寄存器0。

**任务：**从图8分析多路开关的控制位。请注意，控制位在信号e3d中，该信号由信号dd经  
流水线处理得到，后一个信号由控制单元在译码阶段生成（有关控制位的说明，请参见SweRVref.docx）。

* 如果DC3阶段的指令有效（e3d.i0v == 1）且为load指令（e3d.i0load == 1），则选择来自LSU管道的值：i0\_result\_e3\_final = lsu\_result\_dc3。
* 如果EX3阶段的指令有效（e3d.i0v == 1）且为mul指令（e3d.i0mul == 1），则选择来自乘法器的值：i0\_result\_e3\_final = exu\_mul\_result\_e3。
* 否则，将选择来自I0管道的值：i0\_result\_e3\_final = i0\_result\_e3。

**任务：**从图9分析多路开关的控制位，这些控制位位于模块**dec\_decode\_ctl**中。

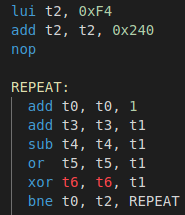
* 如果EX4阶段的结果必须从I0辅助ALU中选择（e4d.i0secondary == 1），则选择来自I0辅助ALU的值：i0\_result\_e4\_final = exu\_i0\_result\_e4。我们将在实验15中分析辅助ALU操作。
* 如果DC4阶段的指令有效（e4d.i0v == 1）且为load指令（e4d.i0load == 1），则选择来自LSU管道的值：i0\_result\_e4\_final = lsu\_result\_corr\_dc4。
* 否则，将选择来自I0管道的值：i0\_result\_e4\_final = i0\_result\_e4。

**任务：**按照以下步骤（如GSG第7部分中详述）在自己的计算机上重复图11和图12中的仿真过程：

* 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/Lab11/ExampleProgram*。
* 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）。
* 使用GTKWave打开轨迹。
* 使用文件*test\_1.tcl*和*test\_2.tcl*（在*[RVfpgaPath]/RVfpga/Labs/Lab11/ExampleProgram*中提供）打开与图11和图12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），然后选择*test\_1.tcl*或*test\_2.tcl*文件。
* 单击几次“*Zoom In*”（放大）（）移动至48500 ps（或循环的任何其他迭代，第一次迭代除外）。

解答请参见实验11的主文档。

**任务：**按照GSG所述在Nexys A7板上执行图13中的程序。对于测量的四个事件，应获得图14所示的结果。解释并证明结果。



程序由1000000次迭代的循环构成，该循环包含5条算术逻辑指令和一个条件分支。未发生由于冒险引起的暂停，因此：

* + 执行6 \* 1000000条指令
  + 每个周期执行2条指令，因此：(6/2) \* 1000000个周期
  + 执行1000000个分支，据预测几乎所有分支均命中。

**任务：**在图13所示程序的硬件计数器中测量其他事件。为此，必须使用pspPerformanceCounterSet函数在*Test.c*文件中更改待测量事件的配置。请注意，可以使用WD PSP文件中定义的宏引用不同的事件（如图1所示）：  
*.platformio/packages/framework-wd-riscv-sdk/psp/api\_inc/psp\_performance\_monitor\_eh1.h*。例如，如果要测量I$未命中数而不是分支未命中数，则必须在文件中将*Test.c*行：**pspPerformanceCounterSet**(D\_PSP\_COUNTER3, E\_BRANCHES\_MISPREDICTED);

替换为行：**pspPerformanceCounterSet**(D\_PSP\_COUNTER3, E\_I\_CACHE\_MISSES);

不提供解答。

**任务：**在Test\_Assembly函数中提供其他程序并检查不同的事件是否提供了预期的结果。可以尝试其他指令，例如装载、存储、乘法、除法...以及引发流水线暂停的冒险。

不提供解答。