# 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）：

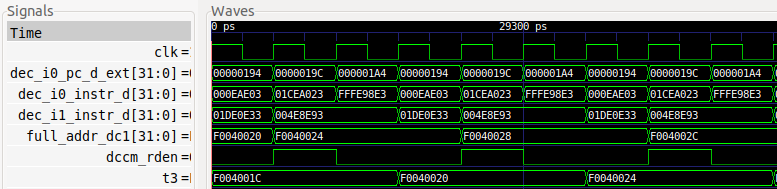
* 必要时生成仿真二进制文件（*Vrvfpgasim*）。
* 在PlatformIO中，打开在以下位置提供的项目：*[RVfpgaPath]/RVfpga/Labs/Lab19/LW-SW\_Instruction\_ExtMemory*。
* 在文件*platformio.ini*中建立到RVfpga仿真二进制文件（*Vrvfpgasim*）的正确路径。
* 使用Verilator生成仿真轨迹（生成轨迹）。
* 在GTKWave上打开轨迹。
* 使用文件*test\_Blocking\_Extended.tcl*（在为*[RVfpgaPath]/RVfpga/Labs/Lab19/LW-SW\_Instruction\_ExtMemory*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。

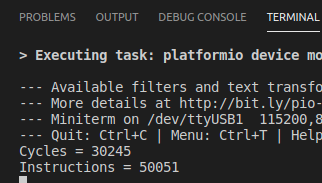
**DCCM**：

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。

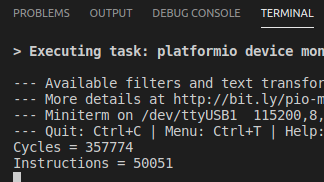
在开发板上执行：



每次迭代的周期数 = 3

**DDR存储器**：

在开发板上执行：



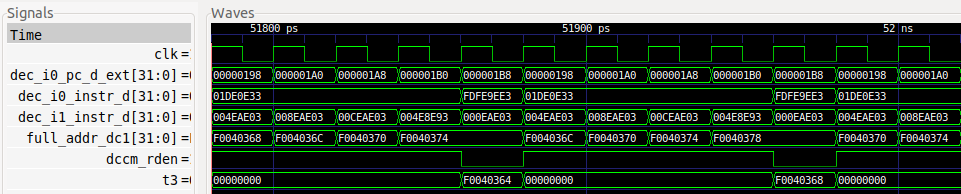
因为程序未发生变化，所以指令数相同。但此时执行所有迭代需要大约358000个周期，因此：

每次迭代中访问存储器所用的周期数 ≈ (358000 - 30000) / 10000 ≈ 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。

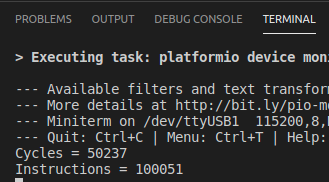
**DCCM**：

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。

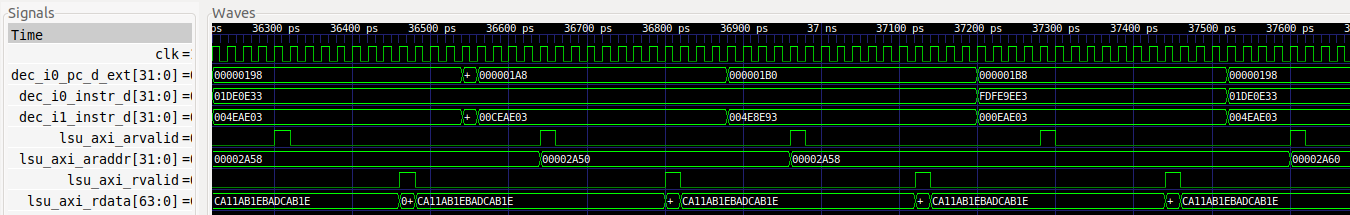
在开发板上执行：



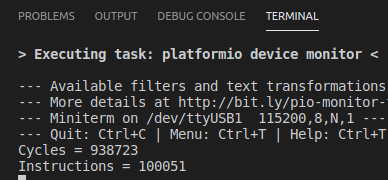
每次迭代的周期数 = 5

**DDR存储器**：

在Verilator中仿真：



在开发板上执行：

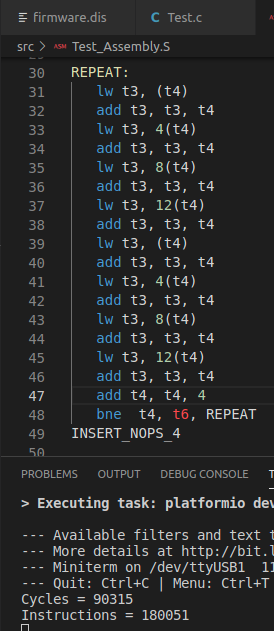


因为程序未发生变化，所以指令数相同。但此时执行所有迭代需要大约939000个周期，因此：

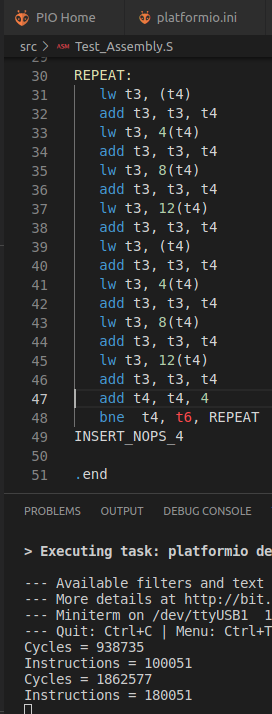
DDR存储器的读延时 ≈ (939000 - 50000) / (10000 \* 4) ≈ 22

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**：



**DDR存储器**：



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

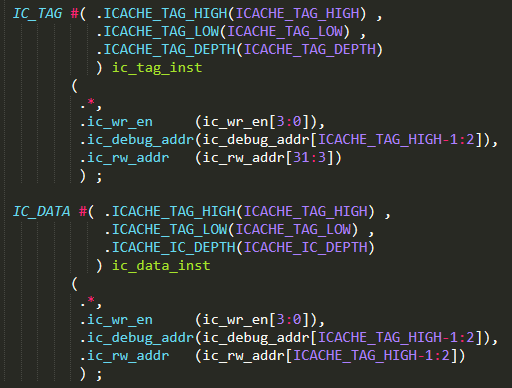
**任务：**分析RVfpga系统中使用的存储器控制器，本练习颇为复杂但十分有趣。请记住，构成该控制器的模块位于*[RVfpgaPath]/RVfpga/src/LiteDRAM*中，顶层模块在该文件夹内的*litedram\_top.v*文件中实现。可以先进行图3所示的仿真，然后添加并分析来自LiteDRAM控制器的一些信号。

不提供解答。

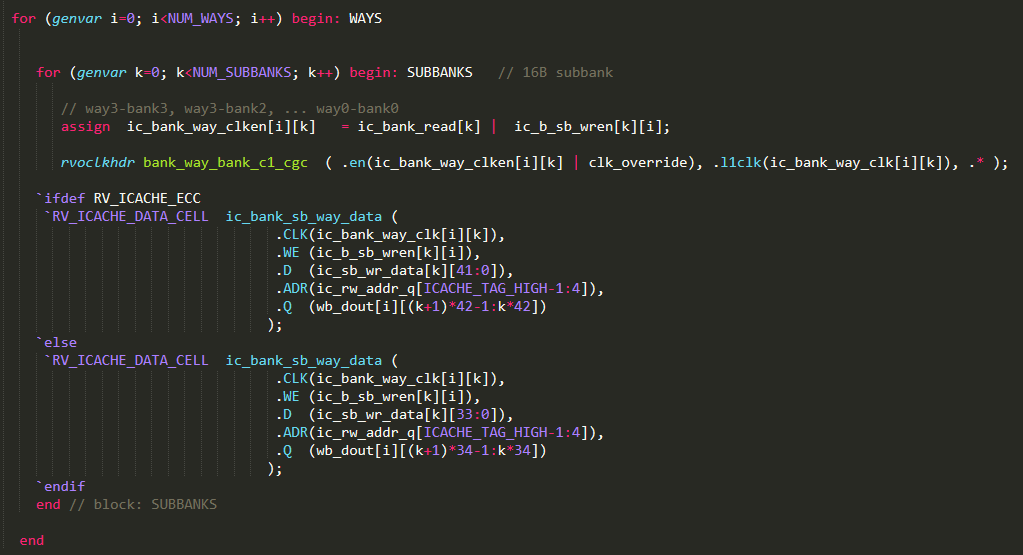
**任务：**分析模块**ifu\_ic\_mem**，了解如何实现图4中的元素。

模块**ifu\_ic\_mem**：

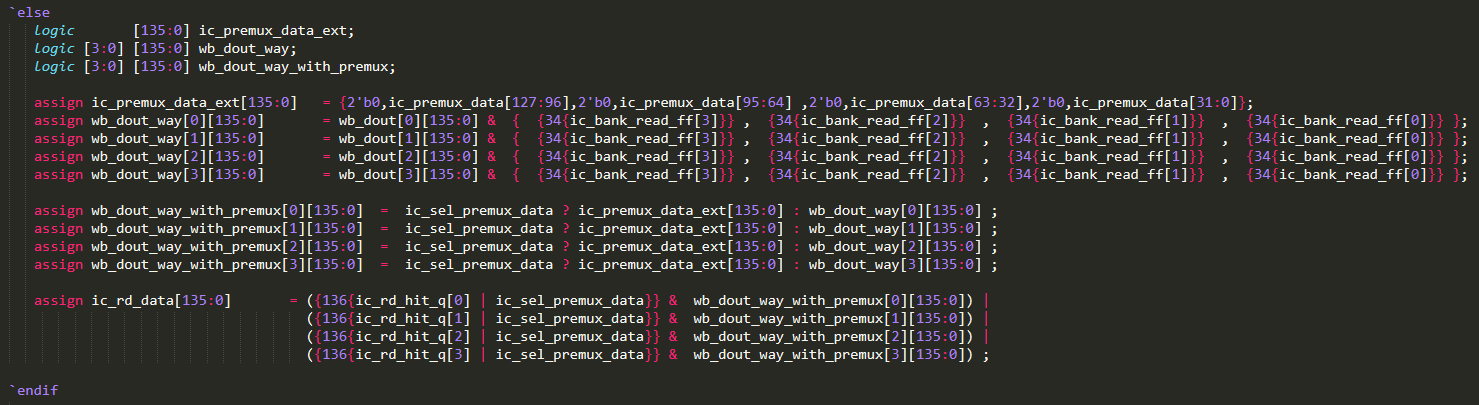
数据数组和标记数组实例化：



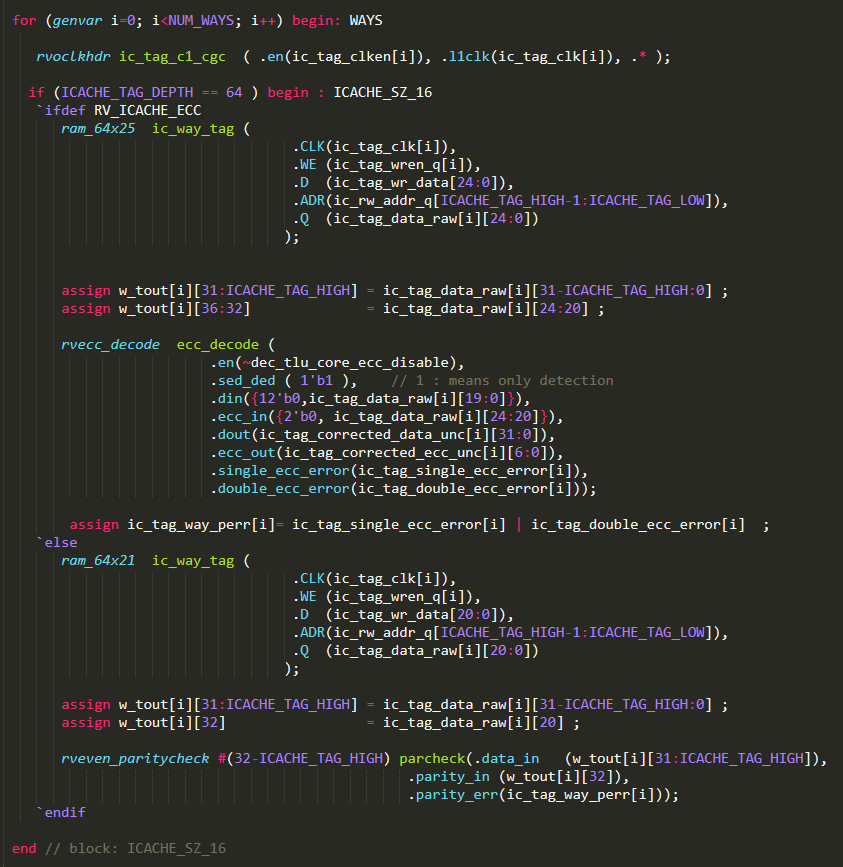
数据数组加奇偶校验位（在本例中未定义RV\_ICACHE\_ECC）：



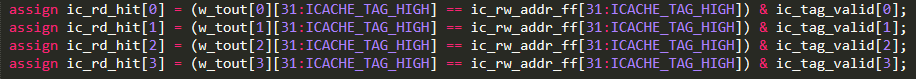
4-1多路开关：



标记数组加奇偶校验位（在本例中未定义RV\_ICACHE\_ECC）：



比较器：



**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。

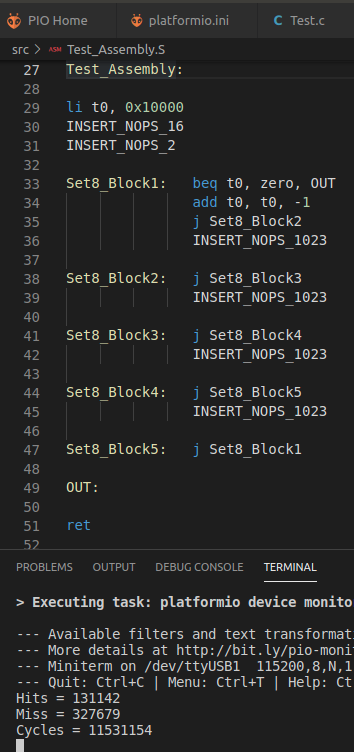
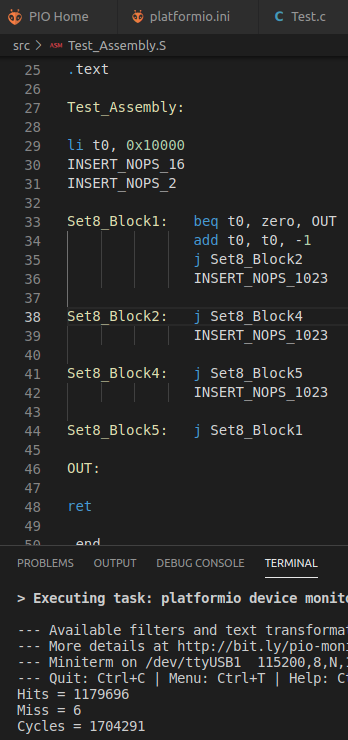
**任务：**分析图10中的Verilog代码，并基于上述说明解释代码如何运行。

不提供解答。

# 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：**

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

1. 使用图5中的程序，从I$替换策略的角度分析I$命中。

不提供解答。

1. 扩展图6，详细分析每个64位块如何写入I$。

不提供解答。

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

不提供解答。

1. 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。