# 任務

**任務：**在自己的電腦上重複圖3中的模擬過程。為此，請按照以下步驟操作（在GSG的第7部分中詳述）：

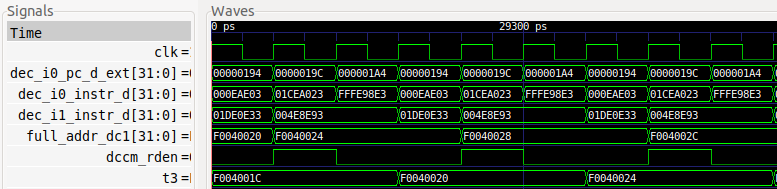
* 必要時產生模擬二進位檔案（*Vrvfpgasim*）。
* 在PlatformIO中，開啟在以下位置提供的專案：*[RVfpgaPath]/RVfpga/Labs/Lab19/LW-SW\_Instruction\_ExtMemory*。
* 在檔案*platformio.ini*中建立到RVfpga模擬二進位檔案（*Vrvfpgasim*）的正確路徑。
* 使用Verilator產生模擬軌跡（產生軌跡）。
* 在GTKWave上開啟軌跡。
* 使用檔案*test\_Blocking\_Extended.tcl*（在*[RVfpgaPath]/RVfpga/Labs/Lab19/LW-SW\_Instruction\_ExtMemory*中提供）開啟與圖6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）  
  並選擇*test\_Blocking\_Extended.tcl*檔案。
* 按幾次「*Zoom In*」（放大）（），然後分析自42500 ps起的區域。

解答請參見實驗19的主文件。

**任務：**使用硬體計數器測量圖2中程式的週期數、指令數、載入次數和儲存次數。存取DDR外部記憶體所用的總時間是多少（包括讀取存取和寫入存取）？可以比較使用圖3中的DDR記憶體與使用DCCM時的執行情況（*[RVfpgaPath]/RVfpga/Labs/Lab19/LW-SW\_Instruction\_DCCM/*下提供另一個PlatformIO專案，其中包含用於對DCCM進行讀/寫操作的相同程式）。請注意，用於模擬的記憶體不是Nexys A7開發板上實際使用的DDR記憶體。

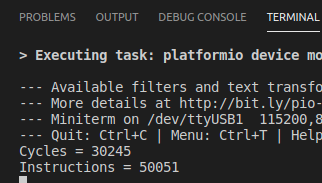
**DCCM**：

在Verilator中模擬：



每次迭代需要3個週期，執行5條指令。每次迭代僅遺失半個週期。

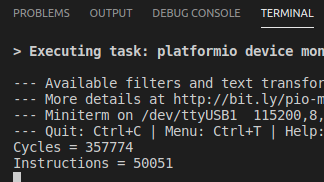
在開發板上執行：



每次迭代的週期數 = 3

**DDR記憶體**：

在開發板上執行：



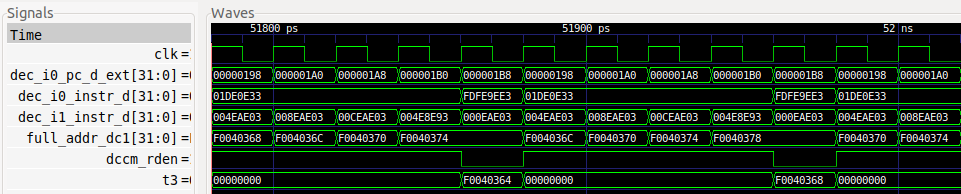
因為程式未發生變化，所以指令數相同。但此時執行所有迭代需要大約358000個週期，因此：

每次迭代中存取記憶體所用的週期數 ≈ (358000 - 30000) / 10000 ≈ 33

**任務：**使用*[RVfpgaPath]/RVfpga/Labs/Lab19/LW\_Instruction\_ExtMem*中提供的範例，藉助硬體計數器估算DDR外部記憶體的讀取延遲。與上一任務一樣，可以使用*[RVfpgaPath]/RVfpga/Labs/Lab19/LW\_Instruction\_DCCM*中的範例，將現有程式與因記憶體存取而不存在暫停的程式進行比較。請注意，用於模擬的記憶體不是Nexys A7開發板上實際使用的DDR記憶體。

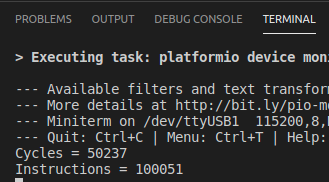
**DCCM**：

在Verilator中模擬：



每次迭代需要5個週期，執行10條指令，因此IPC為理想值。

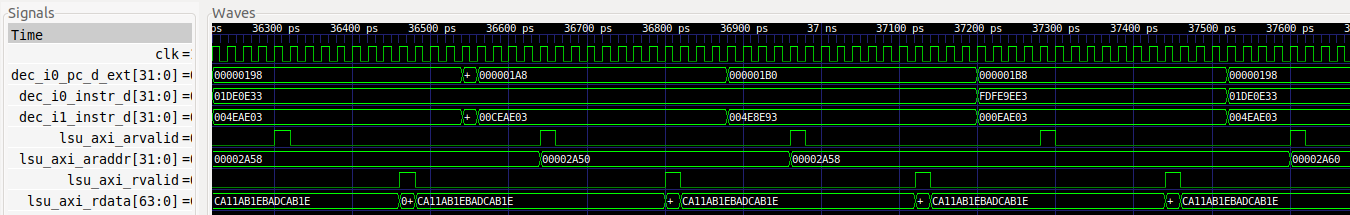
在開發板上執行：



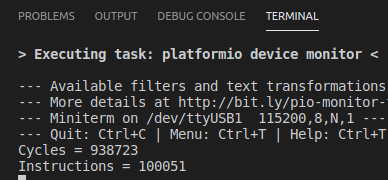
每次迭代的週期數 = 5

**DDR記憶體**：

在Verilator中模擬：



在開發板上執行：

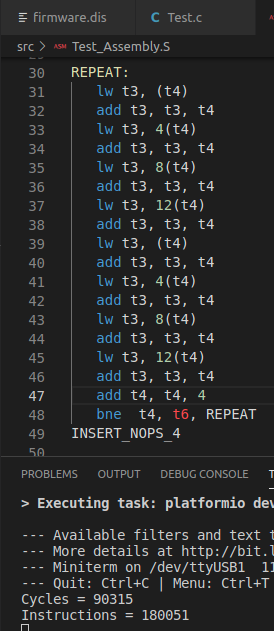


因為程式未發生變化，所以指令數相同。但此時執行所有迭代需要大約939000個週期，因此：

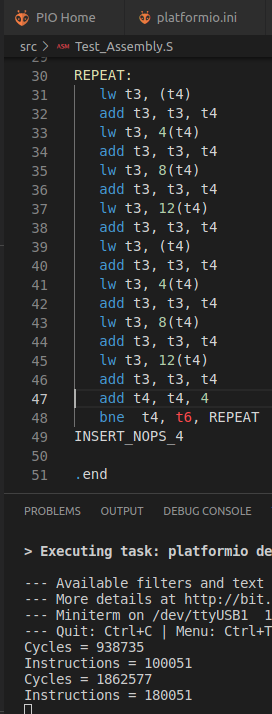
DDR記憶體的讀取延遲 ≈ (939000 - 50000) / (10000 \* 4) ≈ 22

為了檢查該值是否正確，我們將載入指令的數量加倍，然後再次執行程式：

**DCCM**：



**DDR記憶體**：



DDR記憶體的讀取延遲 ≈ (1862000 - 90000) / (10000 \* 8) ≈ 22

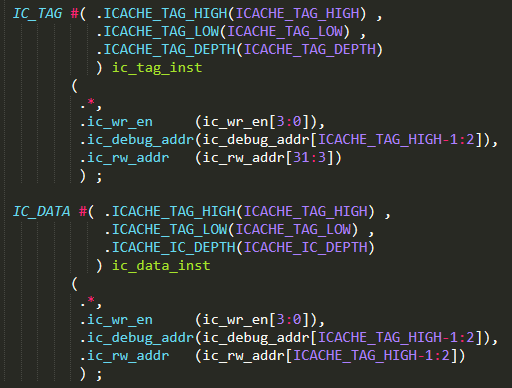
**任務：**分析RVfpga系統中使用的記憶體控制器，本練習頗為複雜但十分有趣。請記住，構成該控制器的模組位於*[RVfpgaPath]/RVfpga/src/LiteDRAM*中，頂層模組在該資料夾內的*litedram\_top.v*檔案中實作。可以先進行圖3所示的模擬，然後新增並分析來自LiteDRAM控制器的一些訊號。

不提供解答。

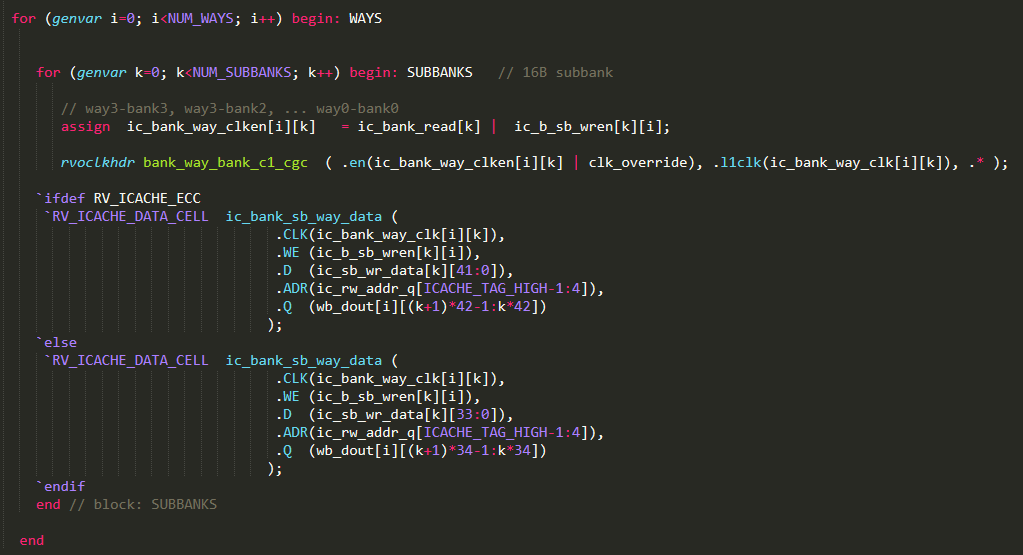
**任務：**分析模組**ifu\_ic\_mem**，瞭解如何實作圖4中的元素。

模組**ifu\_ic\_mem**：

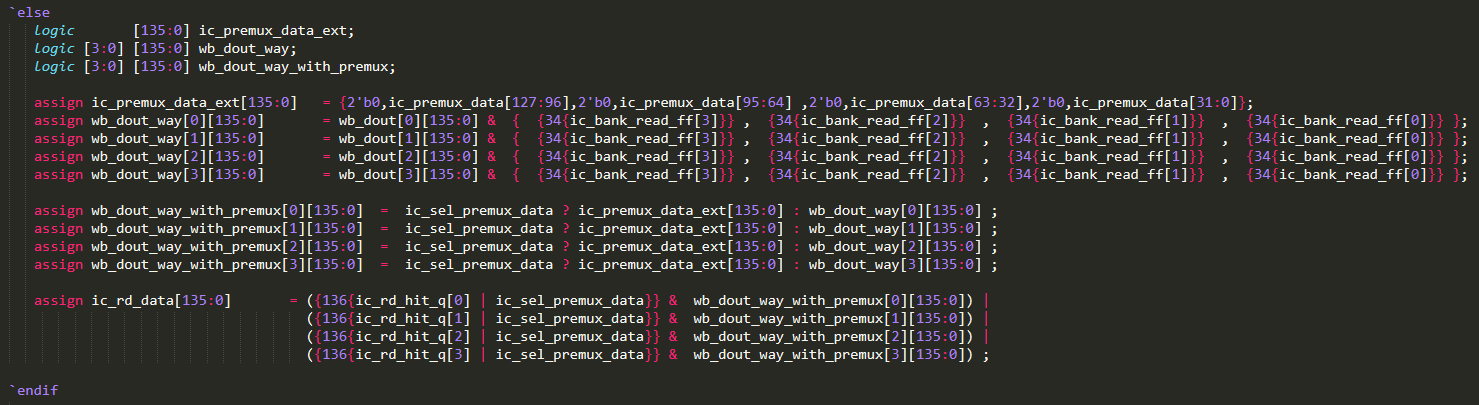
資料陣列和標籤陣列實例化：



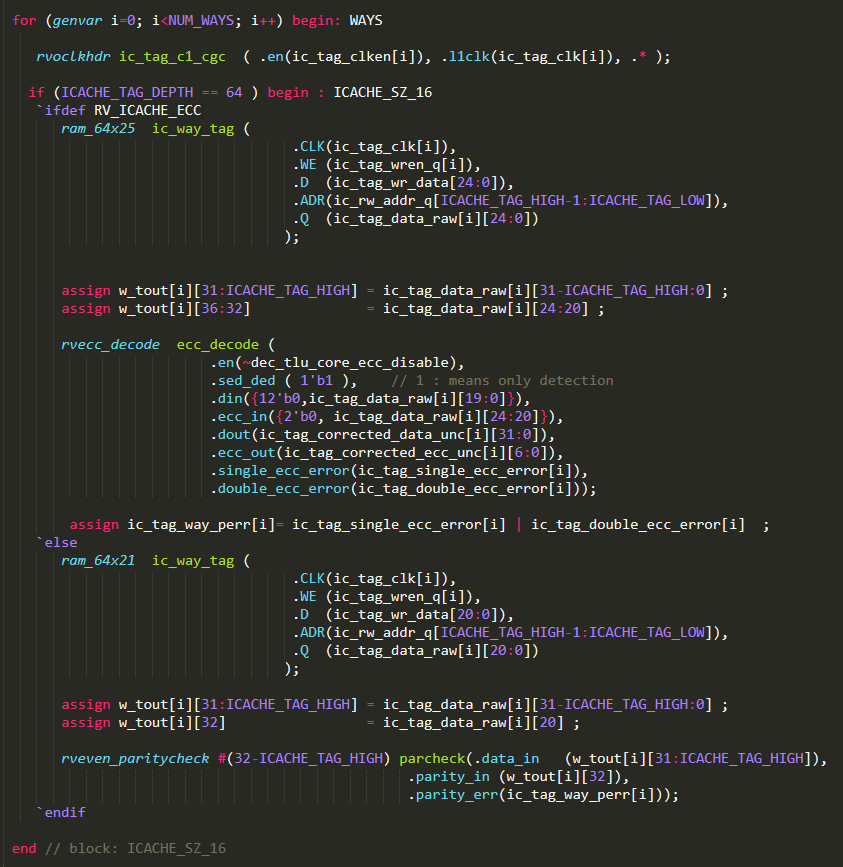
資料陣列加同位檢查（在本例中未定義RV\_ICACHE\_ECC）：



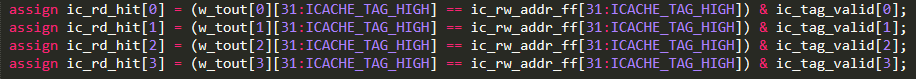
4-1多路開關：



標籤陣列加同位檢查（在本例中未定義RV\_ICACHE\_ECC）：



比較器：



**任務：**在自己的電腦上重複圖6中的模擬過程。為此，請按照以下步驟操作（在GSG的第7部分中詳述）：

* 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/Lab19/InstructionMemory\_Example*。
* 在檔案*platformio.ini*中更新到RVfpga模擬二進位檔案（*Vrvfpgasim*）的路徑。
* 使用Verilator產生模擬軌跡（產生軌跡）。
* 在GTKWave上開啟軌跡。
* 使用檔案*test1\_Miss.tcl*（在*[RVfpgaPath]/RVfpga/Labs/Lab19/InstructionMemory\_Example*中提供）開啟與圖6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）並選擇*test1\_Miss.tcl*檔案。
* 按幾次「*Zoom In*」（放大）（），然後分析28900 ps至30220 ps範圍內的區域。

還可以進行一些更深入的分析，例如對I$的寫入操作或初始指令的旁路。

解答請參見實驗19的主文件。

**任務：**在自己的電腦上重複圖7中的模擬過程。使用檔案*test1\_Hit.tcl*（在*[RVfpgaPath]/RVfpga/Labs/Lab19/InstructionMemory\_Example*中提供）。按幾次「*Zoom In*」（放大）（）移動至34680 ps。

解答請參見實驗19的主文件。

**任務：**分析圖9中的Verilog程式碼，並基於上述說明解釋程式碼如何執行。

不提供解答。

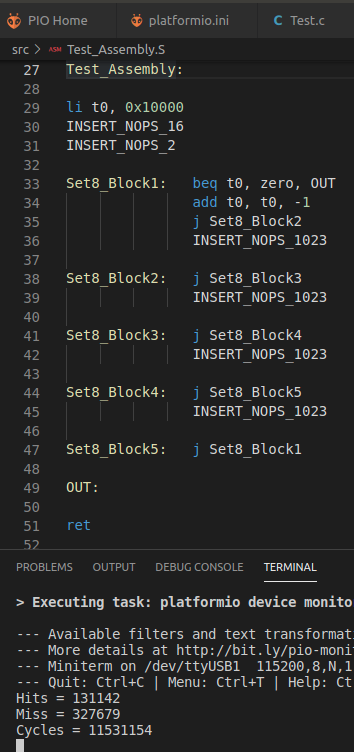
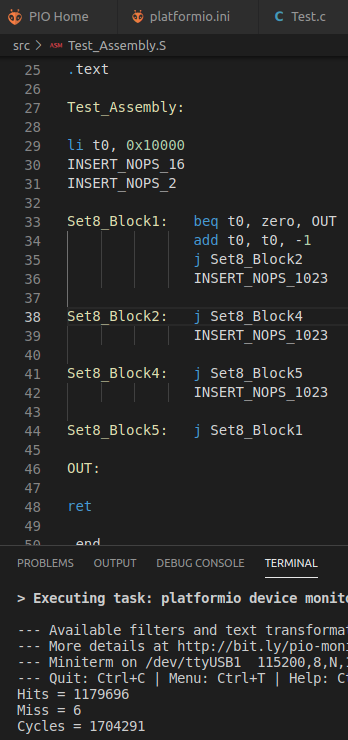
**任務：**分析圖10中的Verilog程式碼，並基於上述說明解釋程式碼如何執行。

不提供解答。

# 練習

1. 將圖11所示的迴圈轉換為0x10000次迭代的迴圈，但為j指令保持原有的位址。測量週期數以及I$命中和未命中的次數。然後刪除其中一條j指令，再次測量上述指標。比較測量結果，並做出解釋。

**5條跳轉指令： 4條跳轉指令：**

在採用4條j指令的程式中，I$未命中數和週期數顯著減少，因為此時區塊之間彼此不會發生衝突。與此同時，I$命中數大幅增加。

1. 使用圖5中的程式，從I$替換策略的角度分析I$命中。

不提供解答。

1. 展開圖6，詳細分析每個64位元區塊如何寫入I$。

不提供解答。

1. 透過模擬器和開發板分析其他I$組態，例如具有不同區塊大小的I$。請注意，無法修改通路的數量。

不提供解答。

1. 分析用於檢查資料陣列和標籤陣列同位檢查資訊正確性的邏輯。

不提供解答。