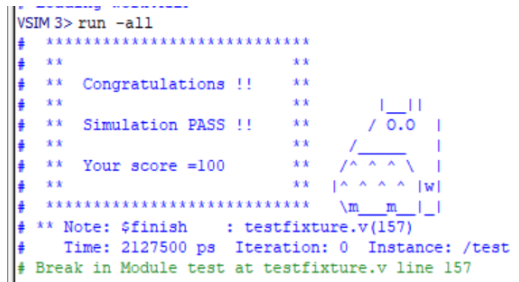
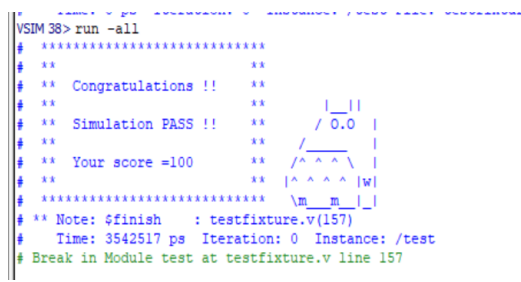
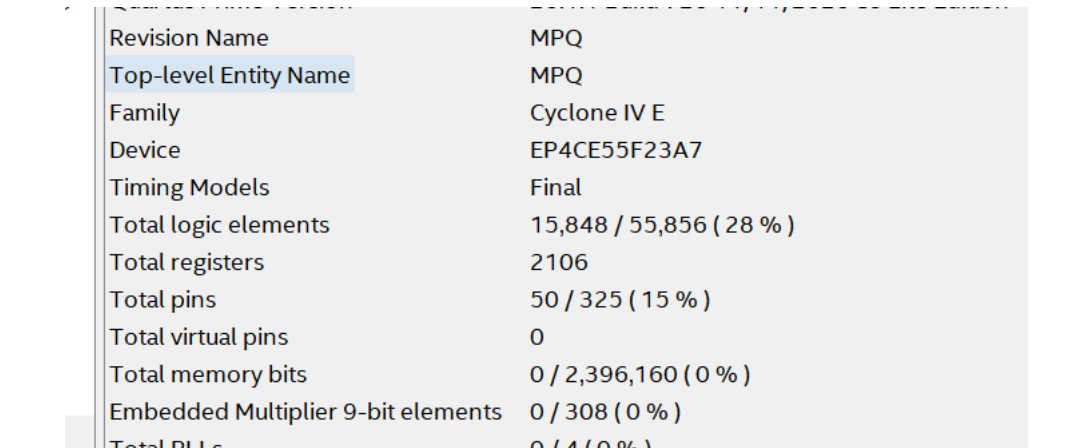


Homework 4: Max-Priority Queue

NAME	陳育政						
Student ID	E24094198						
Simulation Result							
Functional simulation	100	Gate-level simulation	100	Clock width	30 ns	Gate-level simulation time	3543 ns
							
Synthesis Result							
Total logic elements				15848			
Total memory bit				0			
Embedded multiplier 9-bit element				0			
							
Description of your design							
<p>這次的作業一開始我是直接把 BUILD_QUEUE 寫在同一個 state，但是在合成的時候發現似乎是 BUILD_QUEUE 的邏輯太複雜，會讓合成的時間拉長很久，而且出來的 critical path 也會變長，因此後來我把 BUILD_QUEUE 分成 BUILD_QUEUE_1 和 BUILD_QUEUE_2 兩部分實作。BUILD_QUEUE_1 是決定 largest 的 index，而 BUILD_QUEUE_2 則更新 QUEUE 的排序，這樣也的確大幅降低合成所需的時間。</p> <p>至於最初 register 的宣告，我是在各個 state 各自宣告 register，但後來我發現整個 module 中的單一個 state 最多只需要 3 個，因此便使用教授上課教</p>							

過的 resource sharing 技巧把 register 共用，而我需要做的就只是在進入 state 之前，根據 state 所需先更新 register 的值。

在 Post-Simulation 的時候，我最初也沒有通過，而 RAM_D 的輸出結果都是 0，但是在 Pre-Simulation 又可以正常通過。後來我才發現不知為何我沒在 reset 階段初始化 RAM_A 和 RAM_D 會致使 Quartus 在合成的時候有一個警告，就是 output stuck at VCC and GND，後果是合成的電路會讓 RAM_A 和 RAM_D 永遠卡在 0 或 1，於是我把 RAM_A 和 RAM_D 初始化後，Post-Simulation 的結果便成功通過了。

在實作 Max-Priority Queue 的演算法，我都是以助教提供的 pseudo-code 為主，而 EXTRACT_MAX 和 INSERT_DATA 因為在 pseudo-code 分別是直接呼叫 BUILD_QUEUE 和 INCREASE_VALUE，於是在 Verilog 實作中，我是直接讓 currState 跳轉到對應的 state，而非重複寫一次同樣的 code。

*Scoring = (Total logic elements + total memory bit + 9*embedded multiplier 9-bit element) × (Total cycle used*clock width)*