ECE4100/6100 Advanced Computer Architecture
Programming Assignment 4

Assigned: Nov. 1, 2010 **Due 12:35pm Nov. 24, 2010**

Score: _____ Date:_____ Name:_____

TA Signoff: _____

In this assignment, you will implement a **Blocking Instruction Cache** using **Verilog structural model programming**, which is different from your last assignment that allowed you to use behavioral model.

Difficulty Level: This project is non-trivial and could cause anxiety and panicking if you try to do it only in the last week. So start immediately.

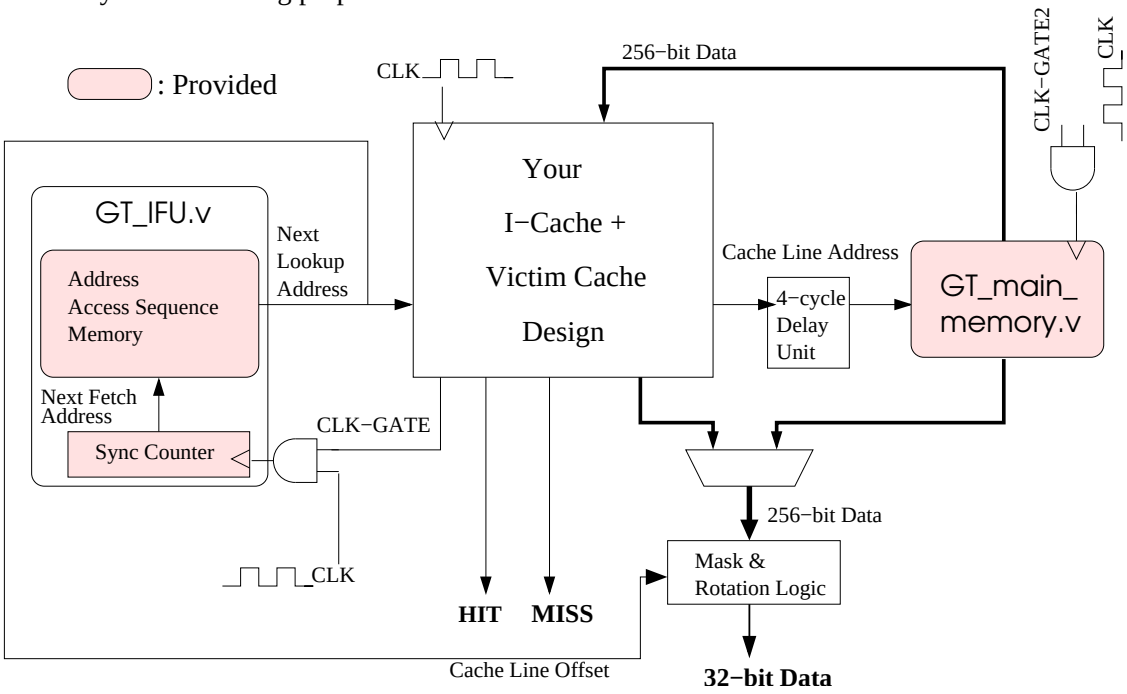
Problem Statement

Design a small 1KB direct-mapped blocking instruction cache with 32-byte cache lines for a 32-bit processor. The addresses for lookup and tags are all virtual (i.e., VIVT cache). This cache also consists of a 4-entry fully associative victim cache discussed in class. (Please refer to lecture slide or Norm Jouppi's ISCA-17 paper on our class webpage.) The replacement policy employed in the victim cache is a pseudo LRU algorithm. Since this is an instruction cache, the cache will be implemented read-only. You do not need to consider dirty writeback given self-modifying mode is prohibited. Note that you can **only use NAND gates** (you are allowed to use an arbitrary number of inputs for each NAND) and **Tristate Buffers** (tribuf) for your entire I-Cache module. "**No behavioral model**" means that you cannot use reg, arithmetic expressions, assignments, and any control flow structure such as if-then-else, for-loop. For testing purposes, the provided test modules are written in behavioral model in order to initialize specific data values at a specific given cycle. TA may use a different test module to check off your design.

Your Cache module will perform the lookup in one cycle (N^{th} cycle) in both the I-Cache and Victim Cache including the swap, if necessary, between the I-Cache and the Victim Cache. During a hit, your Cache module will assert a HIT signal and drive the corresponding 4-byte data out of your Cache module. (Assume the instruction width is 4 bytes.) If missing both caches, your Cache module will stall for 4 cycles and the missed cache line will be retrieved from the main memory and returned in the $(N+4)^{th}$ cycle. The line eviction (if any) to the Victim Cache as well as the LRU update in the Victim Cache also takes place within the same cycle. Afterward, your Cache module resumes to fetch the next address in the $(N+5)^{th}$ cycle. During the stall, nothing advances except the clock. You can either use a shift register or a Mod-by-4 counter to emulate the 4-cycle delay of main memory access. The LRU bits update in the Victim Cache uses the Pseudo LRU algorithm discussed in class. When the Victim Cache is full, the Pseudo LRU decision algorithm is used to pick the victim line. In this read-only implementation, an evicted victim line can simply disappear without updating the main memory.

The high-level schematic diagram is illustrated in the following figure. The address stream for the cache look comes from an instruction fetching unit called GT_IFU which is defined in the provided Verilog file: **GT_IFU.v**. The GT_IFU contains a sequence of instruction addresses to be

looked up in your cache module. It comes with an up-counter that points to one instruction for each cycle. As shown in the figure, you can perform clock-gating using an AND gate to stall instruction fetching. The provided **test_GT_IFU.v** presents an example of clock gating. Also provided is **GT_main_memory.v** which defines a main memory as the backup memory preloaded with some default values in the first 32KB memory. This memory is accessed whenever there is a cache miss. The provided **test_GT_main_memory.v** presents an example with respect to how to read the main memory. Note that the main memory is addressed based on the cache line size (256 bits), thus your I-Cache implementation only needs to send out cache line/block address (tag+index) to the GT_main_memory module rather than the entire 32-bit address. See the provided **test_GT_main_memory.v** file for details. You can modify these files for your own testing purposes.



Design Strategy. Due to the complexity, you are encouraged to form a study group with your peers to discuss your design approaches. However, you are not allowed to copy others' codes. Before you sit down in front of your keyboard and start coding, a good strategy is to perform a paper design first, namely, draw the logic diagram in a **hierarchical** manner of your I-Cache. Design each (smaller) module, test each module prior to connecting them altogether. For example, design an N-bit register using N D flip-flops and then design your tag array and data array using the same register.

How to receive credit? Check off by TA is needed. (We will give details later.) You need to prepare a one-page schematic diagram that illustrates a reasonable high-level logic design of your I-Cache implementation to help TA get a better understanding of your entire logic circuits.

Honor Code. This assignment must be done **individually**. For those who violate the rule, both the originator and the copier will receive zero credit and will be **immediately** reported to the Dean of Students' Affair for further action.