EECE573: Assignment #3

## Group Assignment:- Tejas Randeria Ilia Bautista

# CAUTION: Please use Xilinx to run the files, it’ll not work in Modelsim All the files included have to be present in the same folder

# Objective:-

Design and test a memory/cache with the following functions:

* Flash clear – 2’b00
* Read – 2’b01
* Write – 2’b10
* Search – 2’b11

All the operations take a single cycle to access the cache, and can be selected using `define in the file “cache\_defines.vh”. “Swap” is implemented in the multilevel cache and will consist of using the functions defined above, when necessary.

# Description:-

All the variables that are present in the code have been named to be self-explanatory and consist of two blocks:

1. Cache (module used for both L1 and L2)

* data\_out: is the output of the internal cache block
* data\_out\_miss: necessary only to hold the data of L1 that has been overwritten
* tag\_out\_miss: necessary only to hold the tag of L1 that has been overwritten
* tag\_out\_miss: outputs the hit/miss of the internal cache blocks
* vector\_in: cache input, internally carries (function to operate, tag and data)
* clk: clock which synchronizes the cache internally and is the same as the multilevel clock
* enable: it enables the cache module

1. Multilevel Cache (overall system module)

* data\_out: final output of the multilevel cache
* hit\_miss\_out: outputs the hit/miss of the block
* vector\_in: multilevel cache input, carries (function to operate, tag and data)
* clk: is the clock of the multilevel cache synchronizing all internal operations
* enable: it enables the multilevel cache to be operational

The opcodes are defined using `define in a different file named “cache\_defines.vh” and it’s included using `include at the start of the files “cache.v” and “multilevel\_cache\_top.v”.

The files “cacheblock1\_tb.v”, “cacheblock2\_tb.v”, and “multilevel\_cache\_top\_tb.v” contain the tests for the level 1 cache, level 2 cache, and the multilevel cache respectively.

# Working:-

All the tasks have been included in the “cache.v” file.

Files:

* cache.v – This is the main file for part A of the assignment. It takes as input, a vector (that holds the tag and the data), an enable signal, and a clock.

Then it is structured by the tasks defined in the objectives above. This task will be executed when the proper opcode is selected and it outputs the data, if it is a hit/miss, the data that is being replaced, and the tag of such data

* multilevel\_cache\_top.v – This is the main file for part B of the assignment. It takes the same inputs as the cache module described above

Through the proper opcode it selects the cache which will be used which will then execute in the lower level the tasks from the cache.v file. When is done accessing the cache it will output the data, and if it is a hit or miss

* cacheblock1\_tb.v and cacheblock2\_tb.v – Tests the functionality of the cache block 1 (L1) and cache block 2 (L2) respectively, by giving different inputs testing corner cases and instances specified in the assignment
* multilevel\_cache\_top\_tb.v – Tests the functionality of the multilevel cache (L1 & L2 interacting with each other)
* cache\_defines.vh – Holds the predefined values needed in the files cache.v and multilevel\_cache\_top.v

Each single part of the assignment (cache and multilevel cache) were designed using strategic display statements that monitored every single case, helping both to debug, and encounter potential issues.

# Conclusion:-

* All the functions are implemented using tasks
* The replacement policy implemented is Most Recently Used (MRU). This policy was carried out by saving the entry of the last read/search data, since it will need both to search and read in order to write an instruction.
* When having multiple writes to the same location it will overwrite the data.
* The write function took the more thought and the more alterations once anything was added. It implementation is such that it keeps track of the MRU entry such that it can be replaced properly when the cache is full. It also outputs the proper data and tag for a miss in order to have this data available when needed for swapping between the L1 and the L2.\*\*\*Please read disclaimer about swap\*\*\* The whole design is no exclusive, nor inclusive, so when writing something in the L1 will not be copied to the L2.
* An initialization adding data to the L2 is required in order to be able to test the caches properly at the multilevel. When there is a read miss in L1 (full or empty) it will access L2, when accessing L2 (full or empty) if there is a miss it will have to access the main memory.
* Data will be written to the L2 only at initialization or when read-miss in L2, which will entail accessing the main memory.
* If there is a write into L1 (empty or full) it will always overwrite the data in the L1, without saving old data in the L2.
* The swap function will only happen if there is a read miss in L1, and read hit in L2. Also there is a cache swap wait needed in order to give time for the swap to be done without losing any data.\*\*\*Swap\*\*\*
* Overall there are extra cycles wasted when accessing any of the caches that will be studied and eliminated for the final project.

# Tests:-

* The cache controller and the caches work in synchronous manner, i.e. on positive clock edges, which introduces delay in its operation
* We are aware of the delays introduced due to the above implementation
* The cache test benches are single full tests
* The multilevel cache test bench is a single full test including but not limited to the three cases suggested in the assignment

# Feedback:-

* Tejas Randeria
  + Great lab assignment, gained more knowledge of cache implementation
  + Implementing this in Verilog was fun and exciting, used the concepts taught in the lectures
  + Yes, it was tedious but I understood a lot about cache operation and its implementation in Verilog
  + Could be implemented in different ways, but this looks like a good way of doing this assignment
* Ilia Bautista
  + It is a really tedious assignment but while working on it, I learned the behaviour of the cache, at first level and multilevel
  + Facing all the issues and making all the assumptions strengthen my cache knowledge from last year
  + Every single issue encountered while adding a new operation caused the addition of new pins or creating big assumptions.

# Figure Content:-

Note: The Figures are organized such that the top one (figure a) is the cache block 1 and the bottom one (figure b) is cache block 2.

Figure 1a and 1b. Show the cache block 1 and cache block 2 respectively, being filled the output will be the previous value which in this case is 0.

Figure 2a and 2b. Show the cache block 1 and cache block 2 reading the data written in both blocks.

Figure 3a and 3b. Show the writing and reading of random new data, then read entry 15 (block 1) or 1(block 2), and also it shows the reading of the same data (9 for cache block 1 and 2 for cache block 2) for consecutive cycles.

Figure 4a and 4b. Show the flashing of both caches and concecuently the intention of reading the caches.

Figure 5. Shows the read of the first data (15) that is already in the L1, then it reads data (12) from the L2. Then it has both a miss in L1 and L2. And ultimately it is shown how fast it reads the data (12) when it is already written in L1.



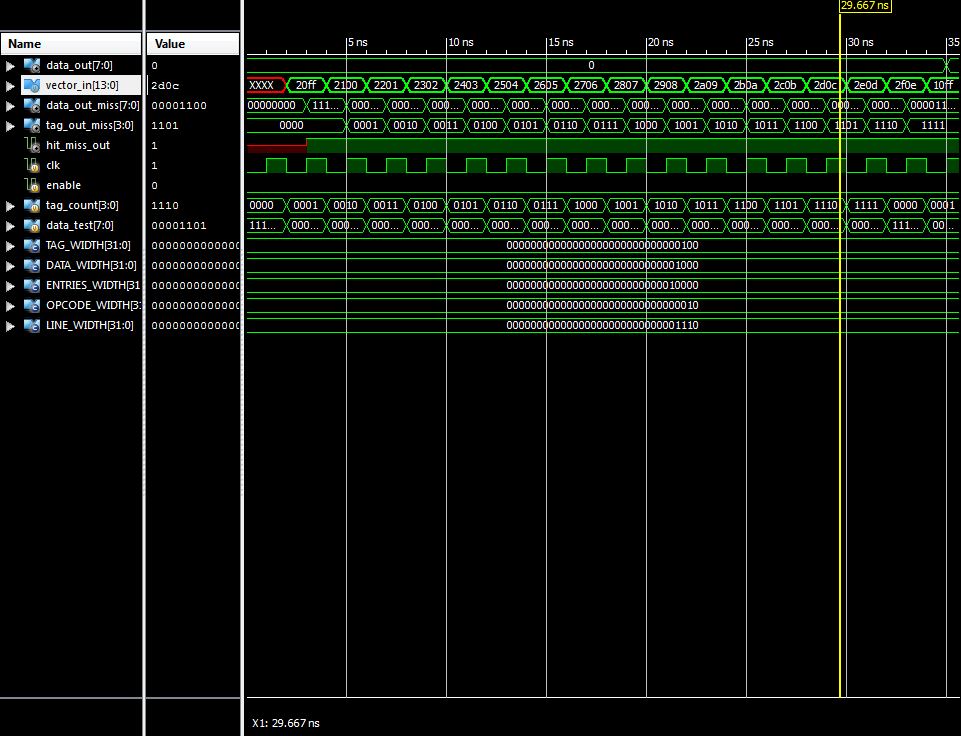


Figure 1a and 1b. Show the cache block 1 and cache block 2 respectively, being filled

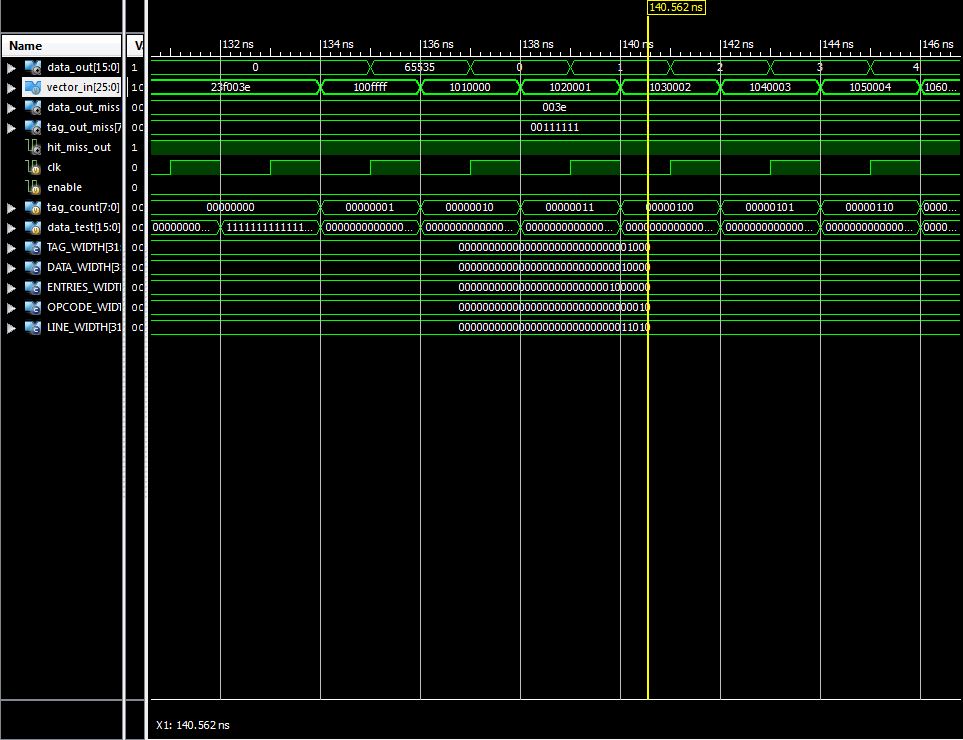
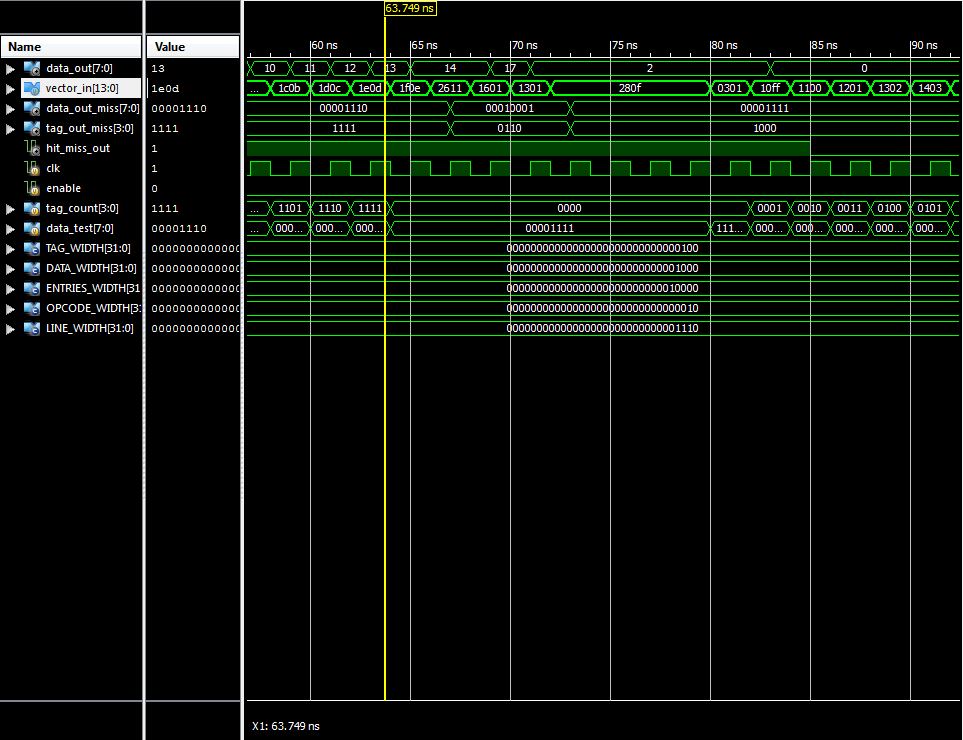


Figure 2a and 2b. Show the cache block 1 and cache block 2 reading the data written in both blocks.

Figure 3a and 3b. Show the writing and reading of random new data, then read entry 15 (block 1) or 1(block 2), and also it shows the reading of the same data for consecutive cycles.



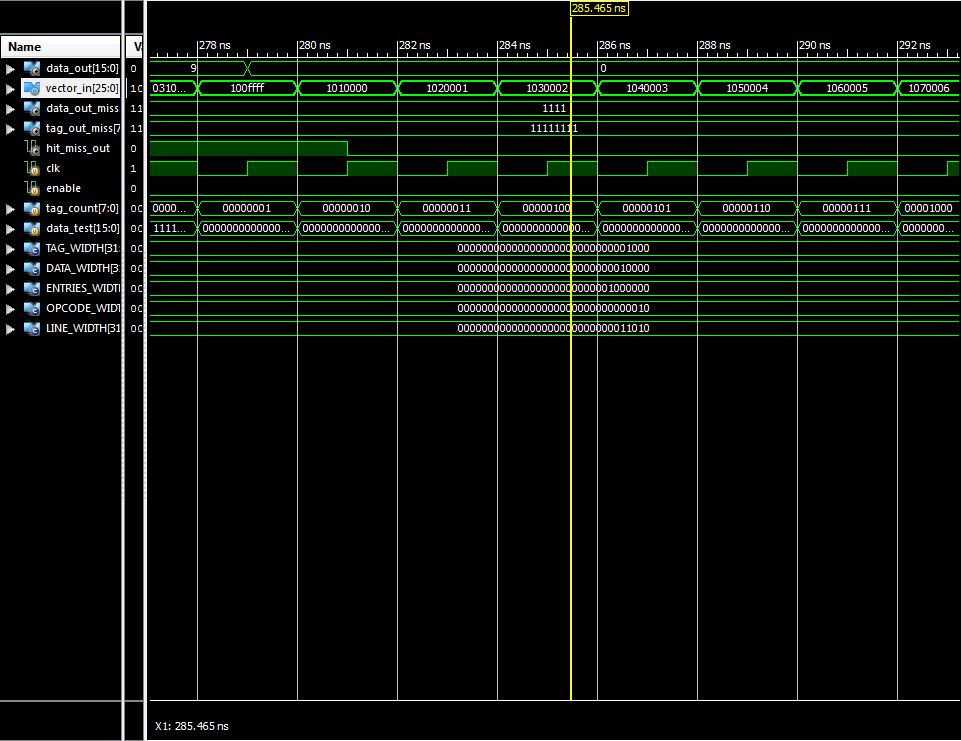




Figure 4a and 4b. Show the flashing of both caches and concecuently the intention of reading the caches.

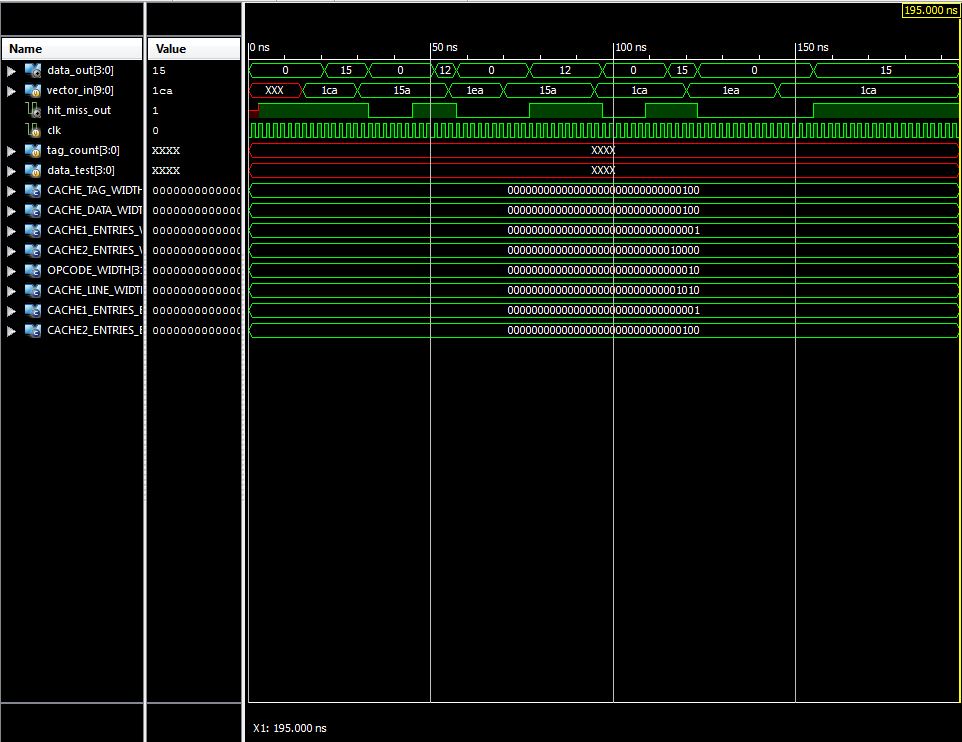


Figure 5. Shows the read of the first data (15) that is already in the L1, then it reads data (12) from the L2. Then it has both a miss in L1 and L2. And ultimately it is shown how fast it reads the data (12) when it is already written in L1.