**Ahmad Othman 2455921**

**Ahsan Mustafa 2328169**



# EEE 446

# Lab Module 4

# Memory Hierarchy Design, System Integration,

# and Evaluation

**OBJECTIVE**

The main objective of this lab module was to integrate Cache Hierarchy into our prebuilt CPU from the previous lab. Where some changes were made to the CPU as well to further tune it for more optimized performance. Firstly, based on the feedback from the Lab 3 and some other changes made to the CPU will be discussed. Secondly, the benchmark code provided in Section 4.3.2 will be ran with memory hierarchy and without, where analysis will be done in terms of speed and duration. These results will be supported by the verifications ran through Verilator and GTKWave.

Finally, the challenges faced in this lab will be discussed, where due to some issues the analysis could not be done to the fullest extent. For further clarity, GTKWave simulations will also be shown.

**4.3.1**

As per the feedback from the previous meeting, the Register File writing on the negative edge was fixed. As the Register File was switched to positive edge of the Clock, the forwarding unit also had to adjusted so that a cycle is not missed due to a Data hazard while Write-Back. Forwarding was done from Writeback stage to Decode stage, bypassing the Register File. This also helped the overly slow frequency problem which persisted In Lab module 3.

To further speedup the CPU, combinational multiplication was replaced in the design a modified “Wallace Tree” multiplier, which is based off lattice multiplication. Summation is done in layers called reduction layers. Computationally, it is a lot better than the combinational multiplication in Quartus, as the total number of reduction layers required for given 2 inputs of length L will be log(L). Fig. 1 from (Wikipedia, 2021) below shows an illustration of this Lattice Multiplication:

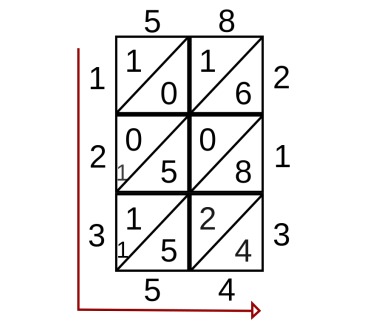


Fig. 1 shows Lattice Multiplication illustration (Wikipedia, 2021)

As it is complex, handling numbers of 16 bits or larger, we used BRAN to speed it up but rather doing a bit-wise lattice operation we can perform 4 bits multiplication at a time. This leads to the total number of reduction layers being halved. The stored memory of 15 x 15 quickly generates the lattice and with 3 layers, the expected frequency at which a single multiplication should be done is 100 MHz .

**4.3.2 & 4.3.3**

The given memory intensive benchmark was run with and without the memory hierarchy. Please refer to Appendix A to see the code in Assembly and HEX.

Verilator tests were also ran successfully to show the crucial parameters for measuring the performance along with confirmation for functionality. Fig. 2 below shows the frequency availed from Quartus Timequest Timing Analyzer without the memory hierarchy. Where Fig. 3 shows the verification ran through the Verilator with all the key parameters shown in the test.



Fig. 2

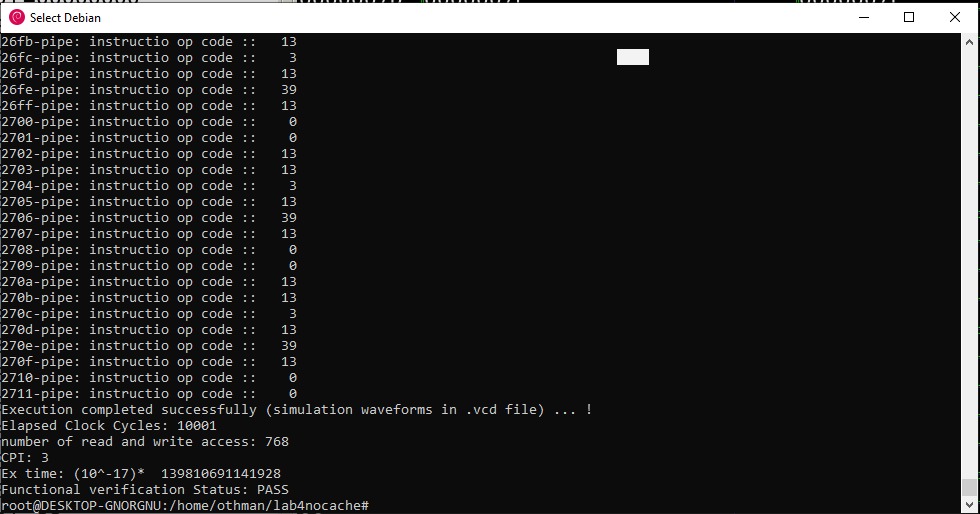


Fig. 3 shows Verilator verification, without the memory hierarchy.

**4.3.4**

Integration of the CPU and the provided Cache Hierarchy was done successfully. This was verified using the provided benchmark. Unfortunately, we did run into a problem when writing the block back to the DRAM from the Data Cache, the DRAM Interface was not sending the signal to the DRAM.

Reading to the Instruction Cache and loading to the Data Cache was done successfully. This is shown below from the GTKWave simulation in Fig. 4, showing the values being correctly written to the Data Cache while running the provided benchmark.

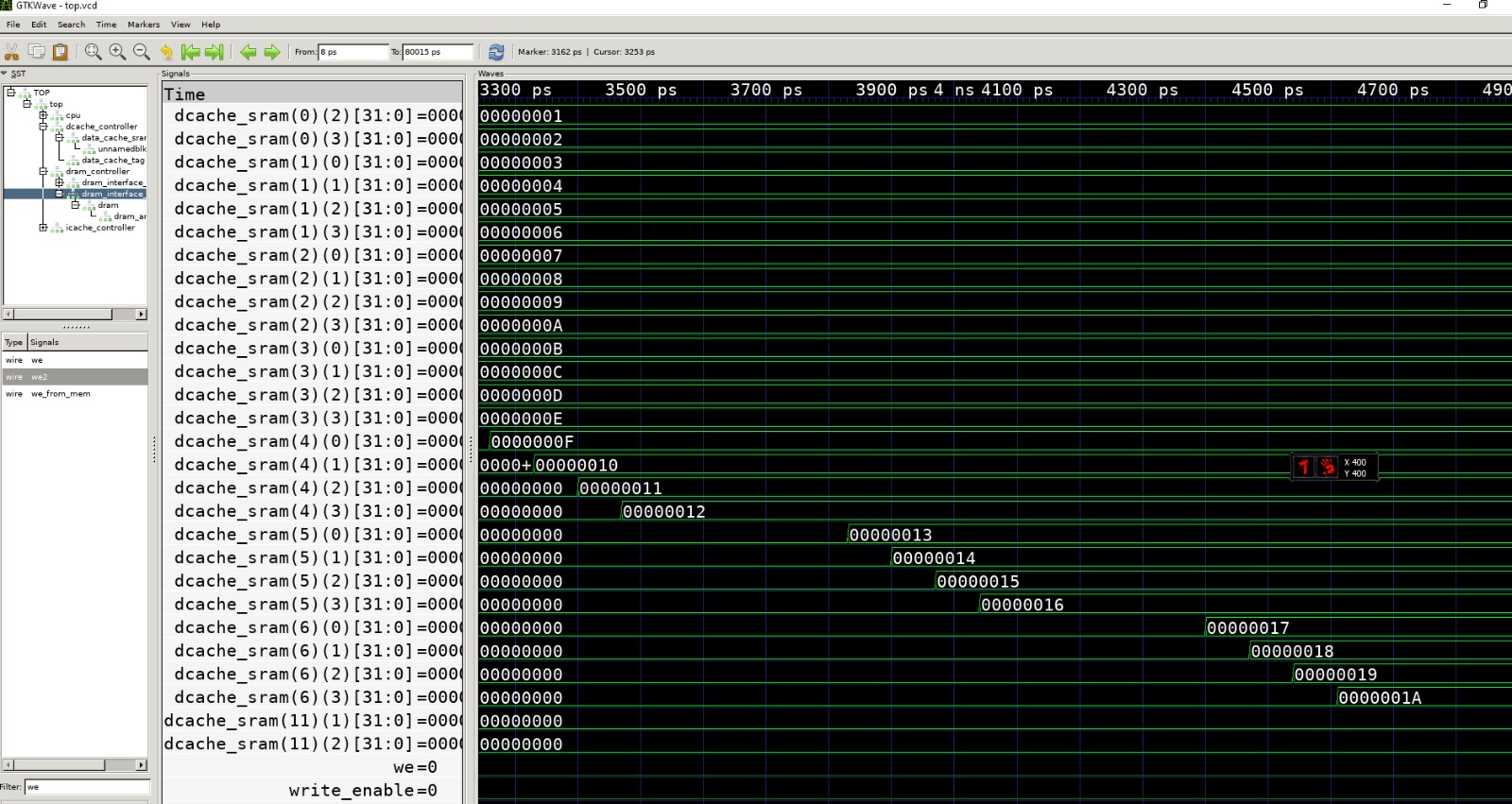


Fig. 4 shows the part of the benchmark, storing integer i to its respective address location.

**4.3.5 – 4.3.7**

Fig. 5 below shows the frequency at which our integrated CPU with Memory Hierarchy is running:



Fig. 5

As we could not write to the DRAM hence, we could not load the useful data from it for the second loop in the benchmark. Apart from this, we confirmed that everything was functional.

What was expected to be seen from the Verilator tests for with the memory hierarchy was a longer execution time as the benchmark is focused on memory transactions, which would incur stall cycles during the access time for read and write to the DRAM, as well as now we would have to take in to account the cache misses, when the data is being accessed by the caches from the DRAM. This resultantly would increase the CPI of our program.

Furthermore, changing the number and size of the blocks in the caches would result in various execution time. As larger the blocks going from 4 to a single 32 block size, we would be accessing the DRAM a lot less, however it would take a significant toll while accessing DRAM when compared to writing or reading a block size of 4.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Frequency | CPI | Execution Time | Func. verif | Clock Cycles | Number of memory accesses |
| Without Mem | 94.2 MHz | 3 | See figure 3 | Pass | 10001 | 768 |
| With Mem | 83.8 MHz | - | - | PASS | 8001 | - |

Please refer to Appendix C to see the DRAM, and data cache in detail

# Appendix A

Assembly Code:

Addi x8, x8, -1

Addi x9, x9, 255

L2: beq x8, x9, L1

Addi x8, x8, 1

Slli x10, x8, 2

Swj x8, 0(x10), L2

L1 addi x8, x8, -1

L4 beq x8, x9, L3

Addi x8, x8, 1

Slli x12, x8, 2

Lw x11, 0(x12)

Addi x7, x12, 1

Swj x11, 0(x7), L4

**Hex:**

**4D 08 FF F0**

**4D 29 0F F0**

**10 08 48 C9**

**4D 08 00 10**

**4D 48 00 22**

**E7 CA 40 00**

**00 00 00 00**

**4D 08 FF F0**

**10 08 49 49**

**4D 08 00 10**

**4D 88 00 22**

**0D 6C 00 00**

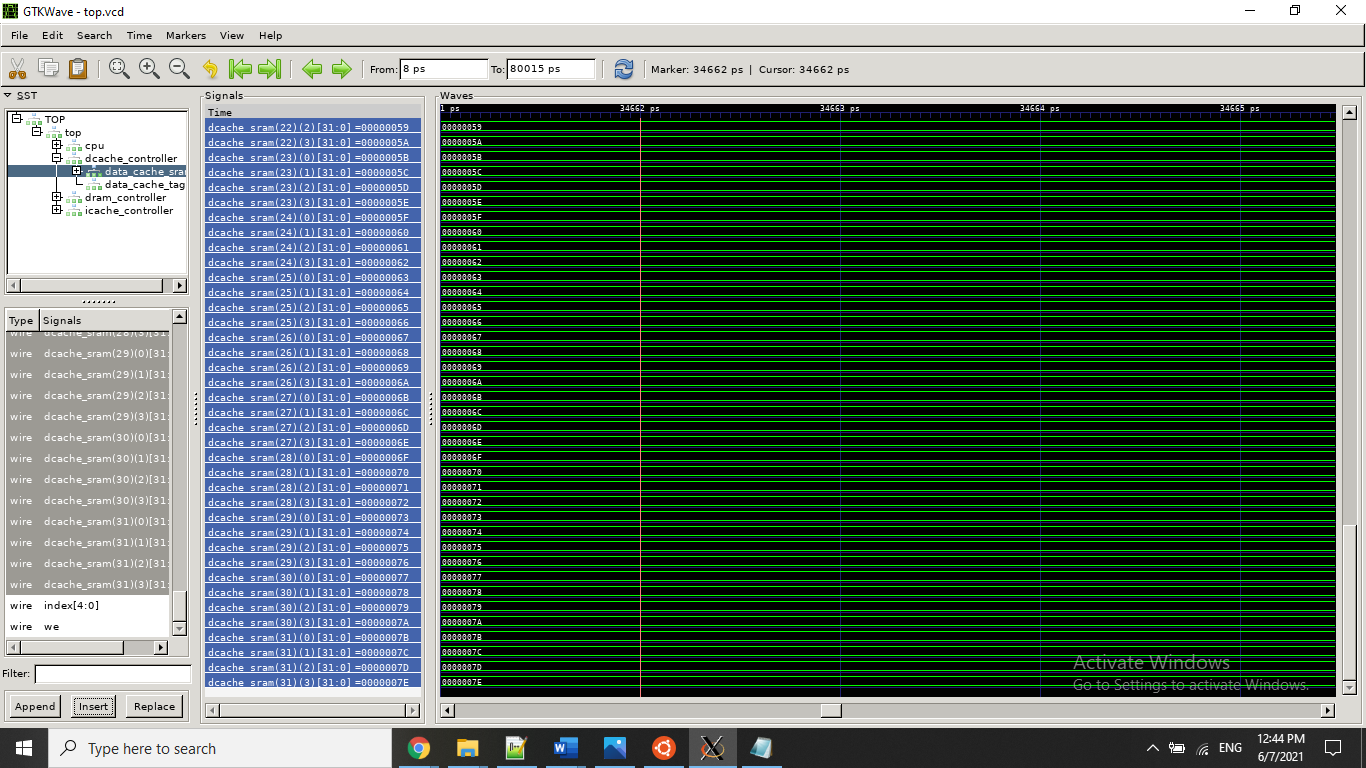
**4C EE 00 10**

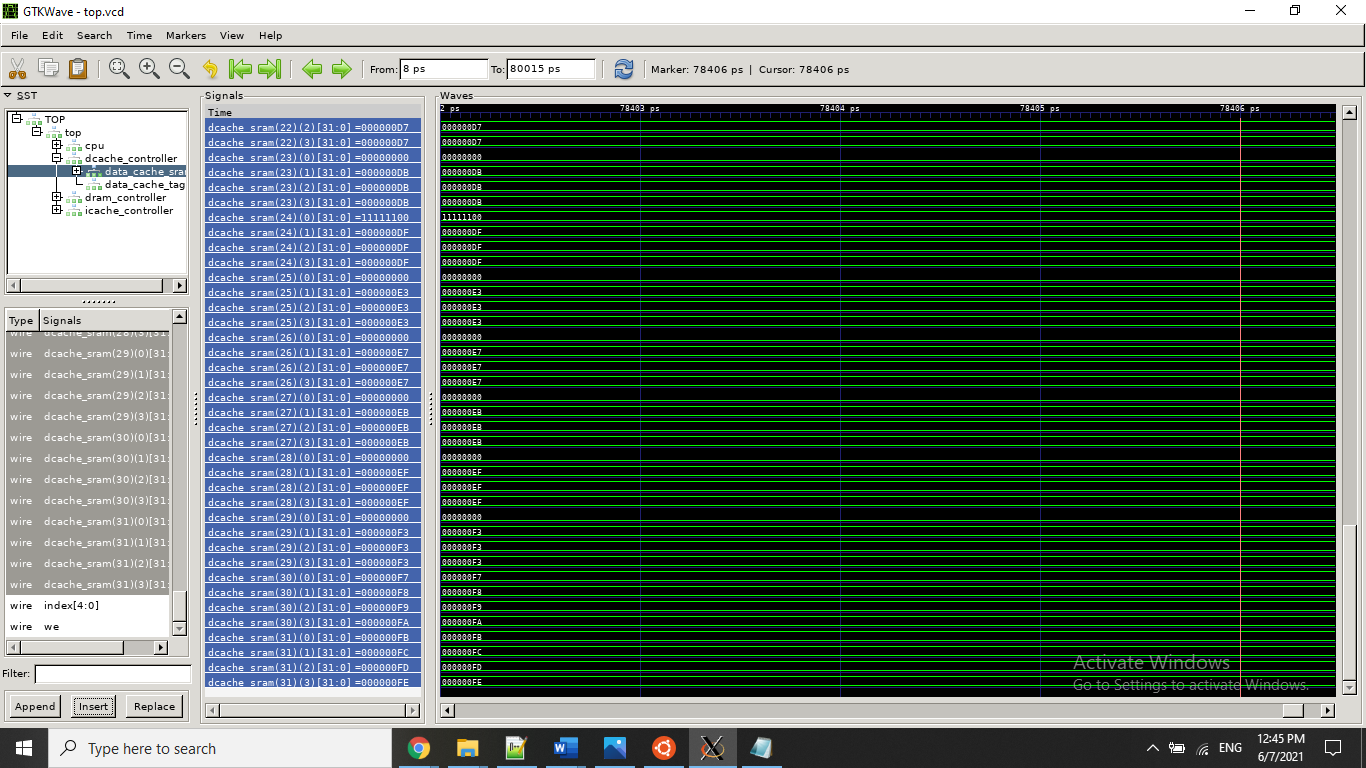
**E7 A7 5C 00**

# Appendix C

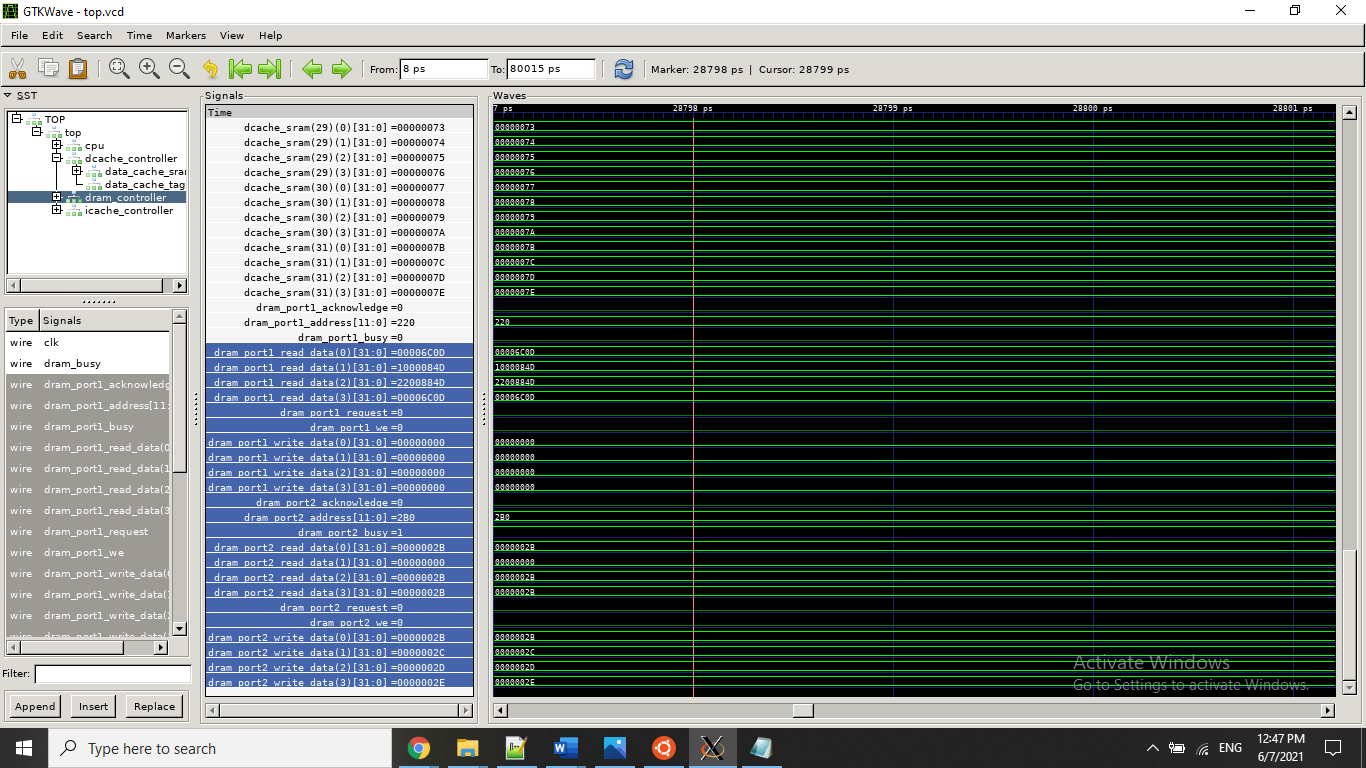
**Simulation Verification**

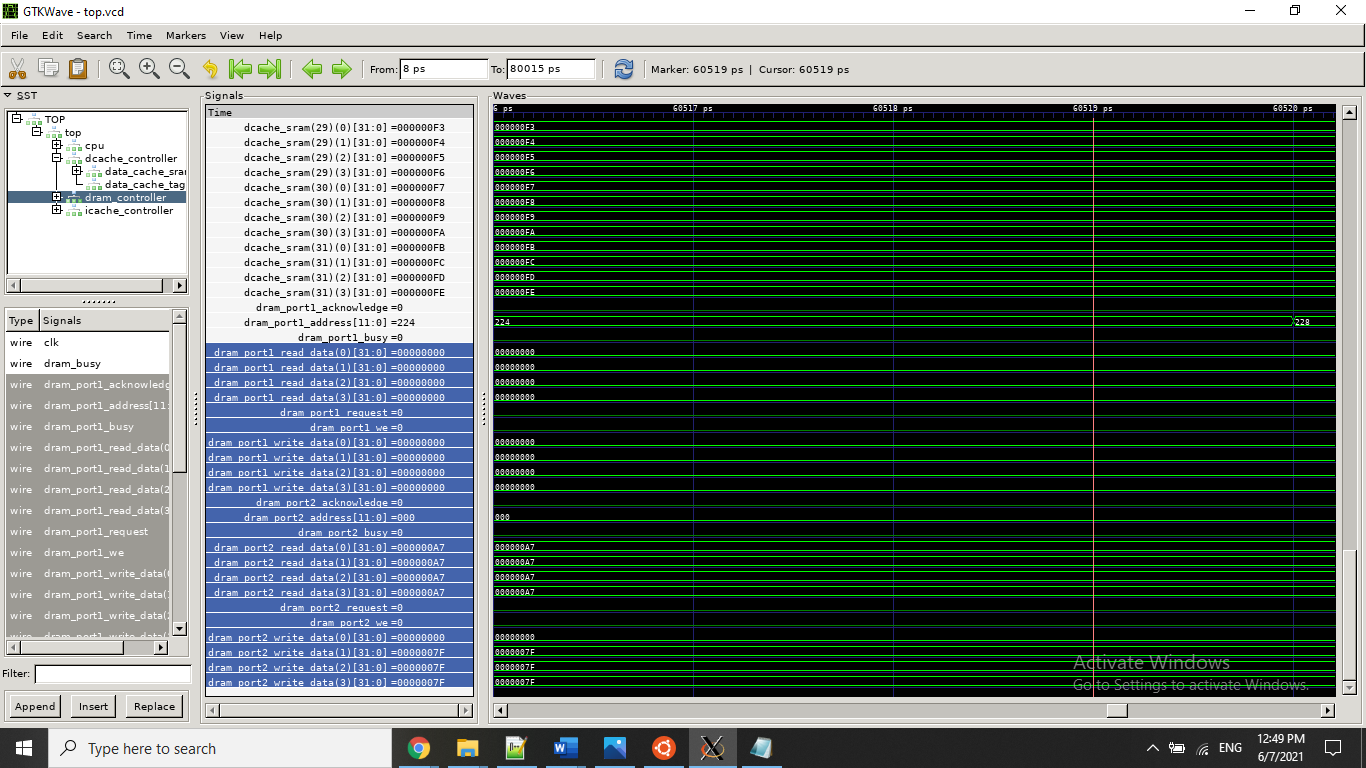
**Data Cache Being Written properly, loading properly**





**DRAM reading data properly**





# References

Wikipedia contributors. (2021b, March 30). *Lattice multiplication*. Wikipedia. https://en.wikipedia.org/wiki/Lattice\_multiplication#/media/File:Example\_of\_step\_3\_of\_lattice\_(shabakh)\_multiplication\_algorithm.sv