EECE573: Final Project Report

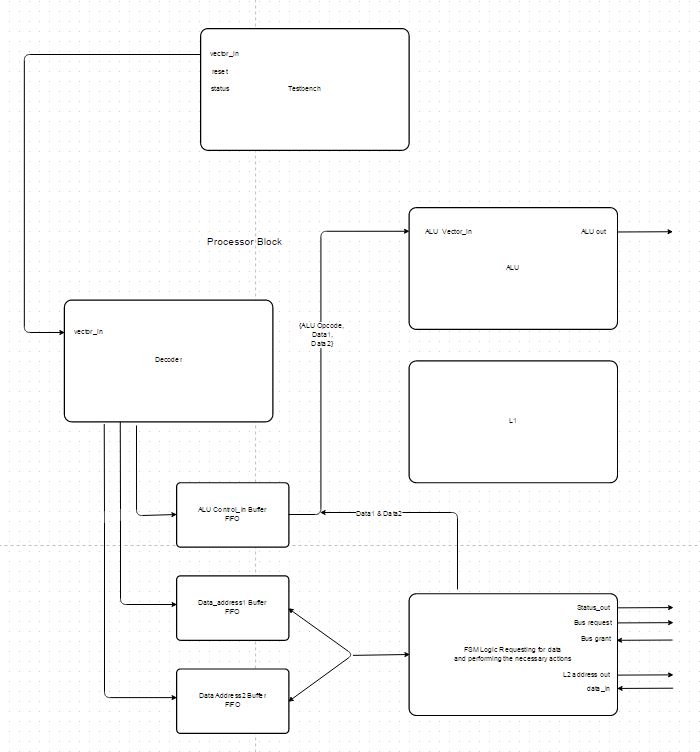
Dual Core Processor

Team Composition

Ilia Bautista

Jose Duque

Aniket Gurkhe

Tejas Randeria

# **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**

Contents

[Objective:- 4](#_Toc418546107)

[Assumptions/Features:- 4](#_Toc418546108)

[Description:- 4](#_Toc418546109)

[Working:- 7](#_Toc418546110)

[Waveforms:- 7](#_Toc418546111)

[Conclusion:- 11](#_Toc418546112)

[Tests:- 11](#_Toc418546113)

[Feedback:- 11](#_Toc418546114)

**Table of Figures**

[Figure 1 Dualcore Intial 7](file:///C:\Users\ANIKET\Downloads\Final%20Project%20Report%20V1.2%20(1).docx#_Toc418706210)

[Figure 2 Dual Core Complete 8](file:///C:\Users\ANIKET\Downloads\Final%20Project%20Report%20V1.2%20(1).docx#_Toc418706211)

[Figure 3 FIFO Working 8](file:///C:\Users\ANIKET\Downloads\Final%20Project%20Report%20V1.2%20(1).docx#_Toc418706212)

[Figure 4 ALU Operation 9](file:///C:\Users\ANIKET\Downloads\Final%20Project%20Report%20V1.2%20(1).docx#_Toc418706213)

[Figure 5 Cache Full Reads 9](file:///C:\Users\ANIKET\Downloads\Final%20Project%20Report%20V1.2%20(1).docx#_Toc418706214)

[Figure 6 Cache Full Writes 10](file:///C:\Users\ANIKET\Downloads\Final%20Project%20Report%20V1.2%20(1).docx#_Toc418706215)

[Figure 7 Cache Flash 10](file:///C:\Users\ANIKET\Downloads\Final%20Project%20Report%20V1.2%20(1).docx#_Toc418706216)

# Objective:-

To develop/code a simple multiprocessor environment (block diagram on cover page). Both cores are composed of: a simple decoder (with a FIFO buffer), a L1 cache, two decoders, and a Arithmetic Logic Unit (ALU). A L2 cache is implemented to hold the data that will be processed. A fixed arbitration scheme is used to assign the communication to the L2 cache. Two unidirectional buses are implemented for communication to the L2

Eight modules are implemented in the multiprocessor environment:

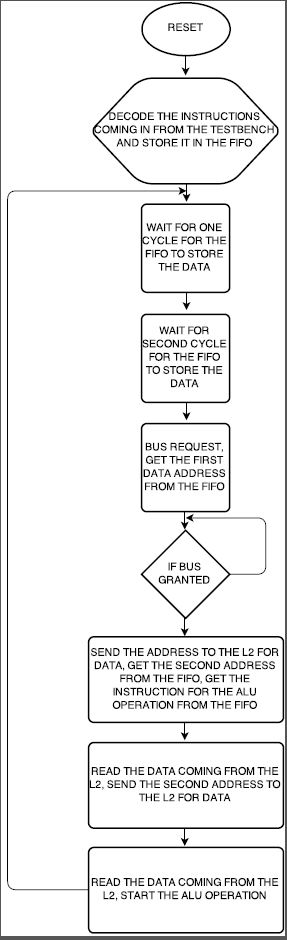
1. dualcore\_tb
2. dualcore
3. arbiter
4. cacheL2
5. processor
   1. alu
   2. cache
   3. fifo

# Assumptions/Features:-

* dualcore\_tb
  + Takes a feedback from the dualcore module to feed in the next instruction
  + The FIFO in the dualcore module gives the feedback mentioned above
* dualcore
  + The bus is implemented in the dualcore module
* arbiter
  + Asynchronous design
  + In a race condition, processor P1 will always receive the bus
  + When none of the processors are requesting for the bus, the arbiter is disabled
* cache - L2
  + Asynchronous design
  + On reset the cache lines are initialized with the required data
  + Synthesized this module taking initial values, which increased our area and power due to extra latches introduced
* processor
  + Synchronous design
  + Implemented the state machine in this module
* alu
  + Synchronous design
* cache - L1
  + Synchronous design
  + Not being used in our dual core implementation – exploiting the simplicity of the design
* fifo
  + Asynchronous design
  + FIFO is empty unless completely full

# Description:-

* dualcore\_tb: It is compose of three procedural blocks (one ‘initial’ and two ‘always’)
  + Initial:
* Prepares all the instructions and addresses that will be fed in the dualcore
* Also initializes all the variables (clock, reset, vector\_in)
  + First ‘always’ block feeds in instructions for the first core/processor
  + Second ‘always’ block: feeds in instructions for the second core/processor
* dualcore: Contains the instantiation of both processors P1 and P2, the arbiter and the L2 cache
  + There are two unidirectional buses, one for connecting the address lines and the other connecting the data lines for the selected processor by the arbiter
* The grant signals from the arbiter is used to connect the address lines from the processor to the L2 cache
* The grant signals from the arbiter are also used to connect the data lines from the the L2 cache to the processor
  + The enable signals for the arbiter and L2 cache are obtained by using the grant and request signals
* enable for the L2 is XOR of the grants signals
* enable for the arbiter is OR of request signals
* arbiter: The arbiter is composed of a case statement that checks two signals (request signal from each core).
  + If both processors try to access the bus, the processor P1 will receive a grant, fixed priority
  + The arbiter is disabled while none processor is communicating to it, which save power
* cacheL2: It takes as input a vector (that holds the tag and the data), an enable signal, and a clock
  + On the reset of the whole system the cache will be filled with the data assigned by the user
  + If there is data that needs to be written in the L2 cache the user will have to re-declare such information
* processor: is composed by a state machine which is used to communicate to the arbiter using the request signals and to the L2 cache using address and data lines
  + decoder: takes an instruction from the test bench and stores it in the fifo
  + fifo: is a congregation of buffers that store the ALU opcodes and data addresses
    - ALU Opcode (instruction) FIFO
    - Data Address 1 FIFO
    - Data Address 2 FIFO
  + cache: It takes as input a vector (that holds the tag and the data), an enable signal, and a clock
    - But in the implementation of this project it will never be used
    - In order for the user to access any of the L1 caches, he/she will have to write a test bench of that module, and use it independently/separately
  + alu is a Arithmetic Logic Unit containing the following instructions:
    - No-Op
    - Add
    - Subtract
    - And
    - Or
    - Zero Test
    - Greater Than
    - Equal
    - Less Than



# Working:-

* We want to feed the instructions to the dual core processors as soon as the internal buffers get empty. To ensure this we have a feedback signal that will interrupt pausing test bench
* The single core processor consists of a state machine as described/showed above. The instructions coming-in from the test bench are fed into the processors P1 and P2
* The processor then decodes the instruction and segregates them. The segregated portion of the instruction is then stored in the fifo for further use
* The state machine uses the data stored in the fifo to obtain the required data from the lower level cache
* The access to the lower level cache will be granted one processor at each given time. The arbitration scheme is simple and explained above
* Once the access is granted to the bus, the processor then sends the address to the lower level cache which then responds with the data for that particular address in the next clock cycle. In our case, it takes two clock cycles to obtain the required data
* The state machine is the entity which ensures the bus ownership. once all the data is obtained for an ALU operation, the bus is freed
* Then the next processor asking for the bus is given access and the whole process is repeated until all the instructions have been exhausted from the testbench

# Waveforms:-

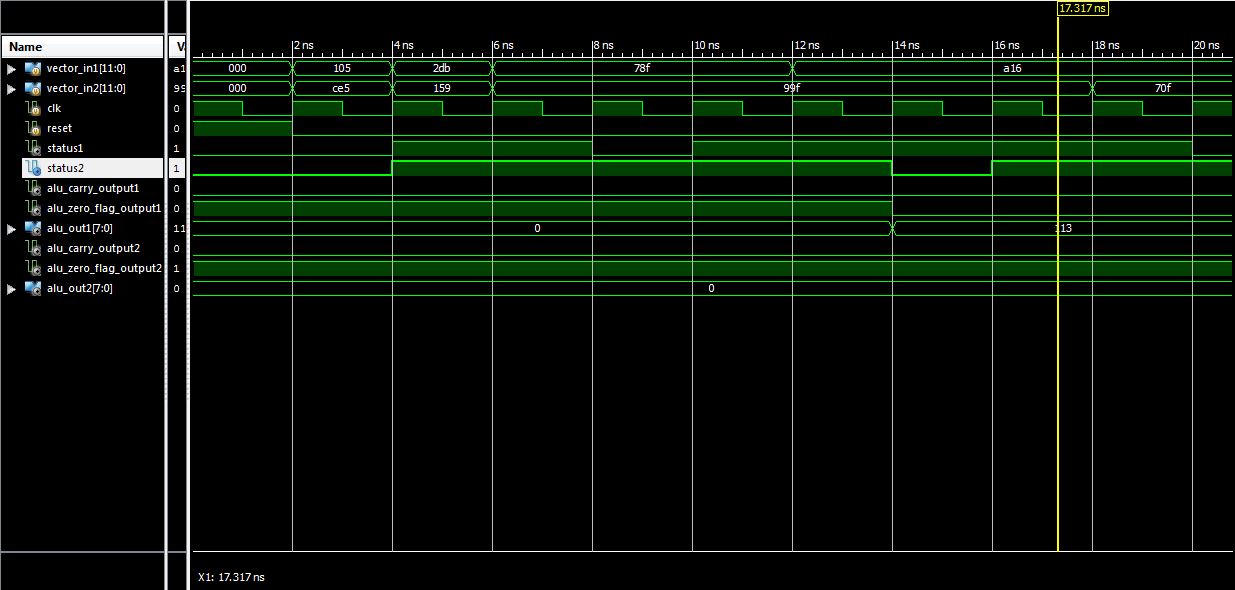


Figure Dualcore Intial

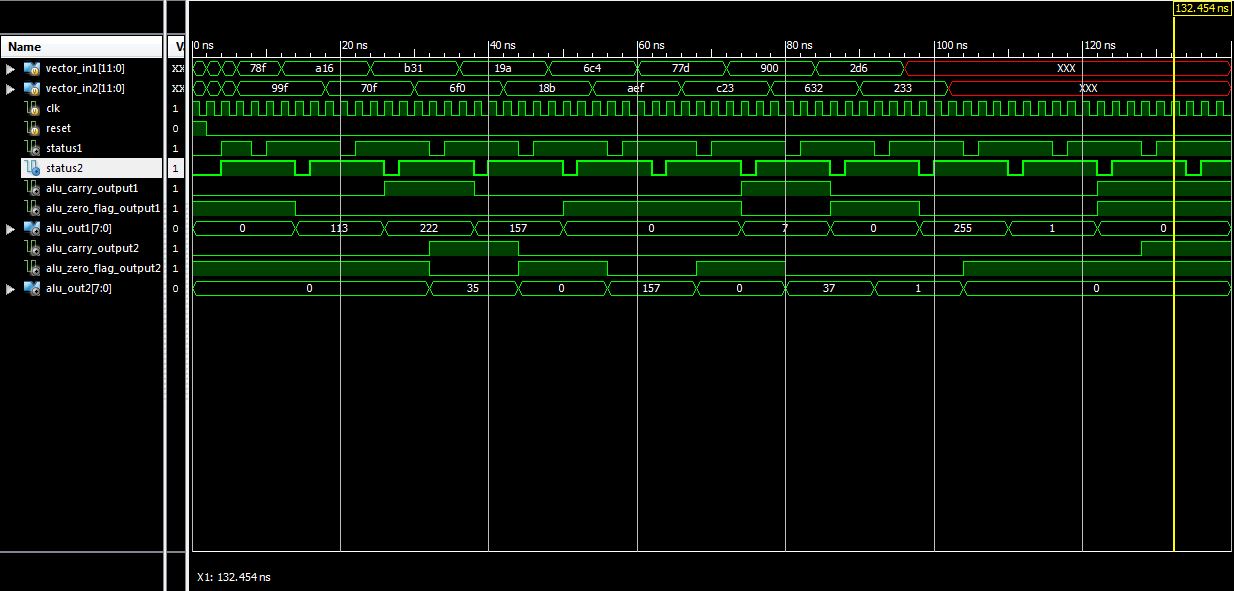


Figure Dual Core Complete



Figure FIFO Working

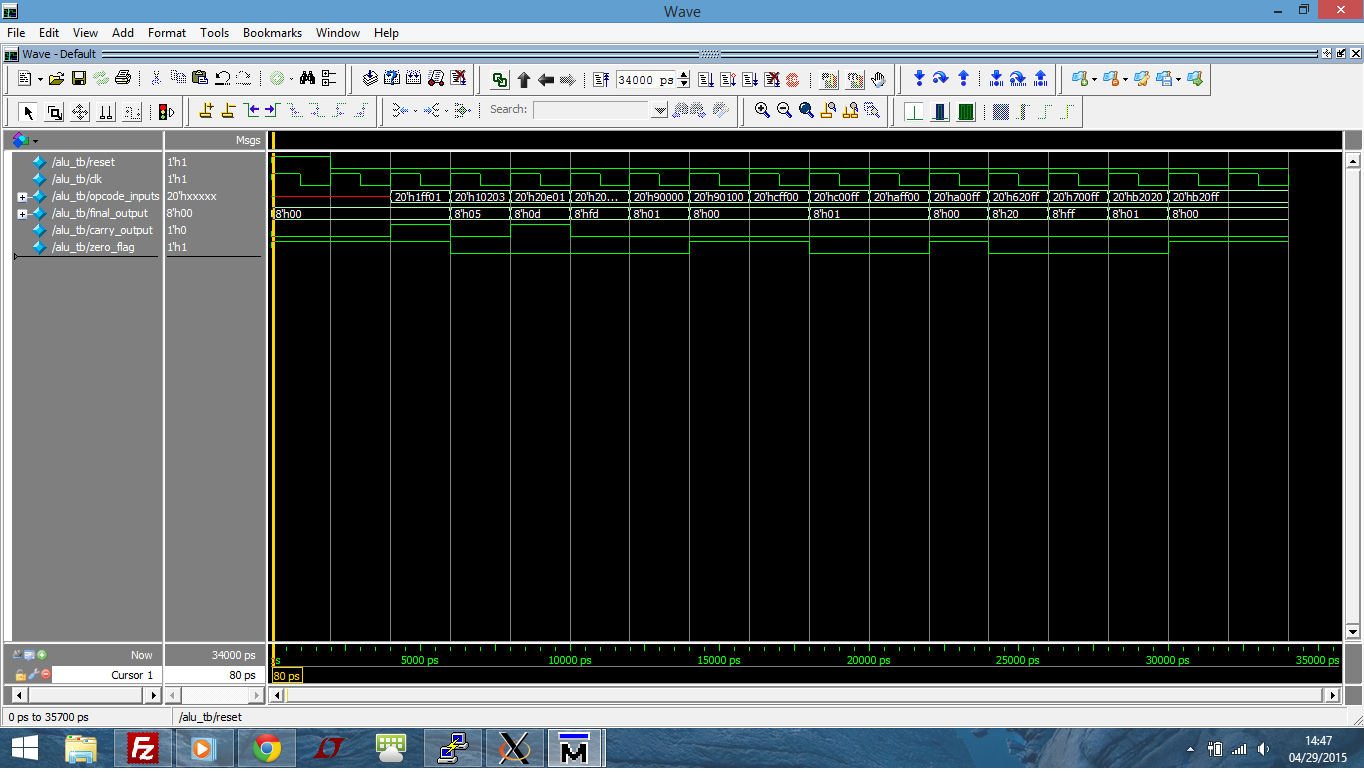


Figure ALU Operation

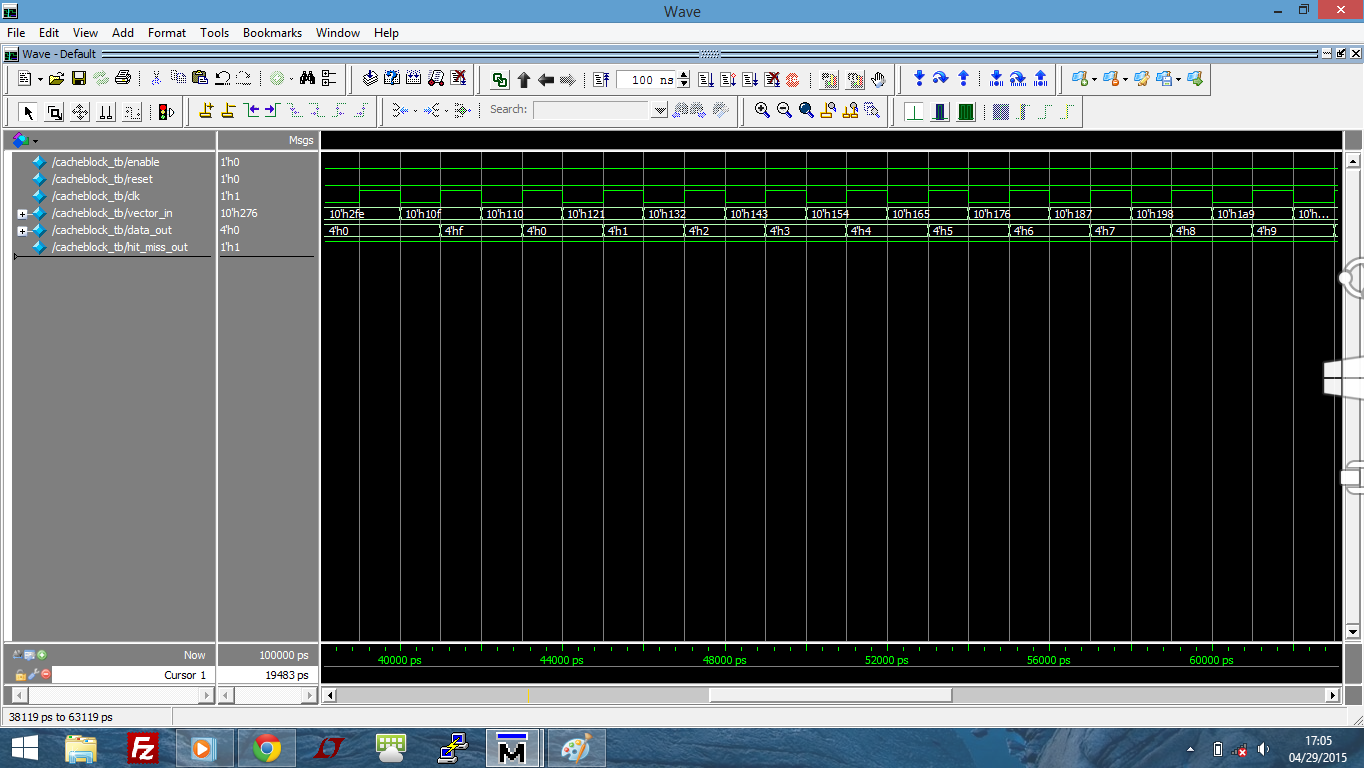


Figure Cache Full Reads

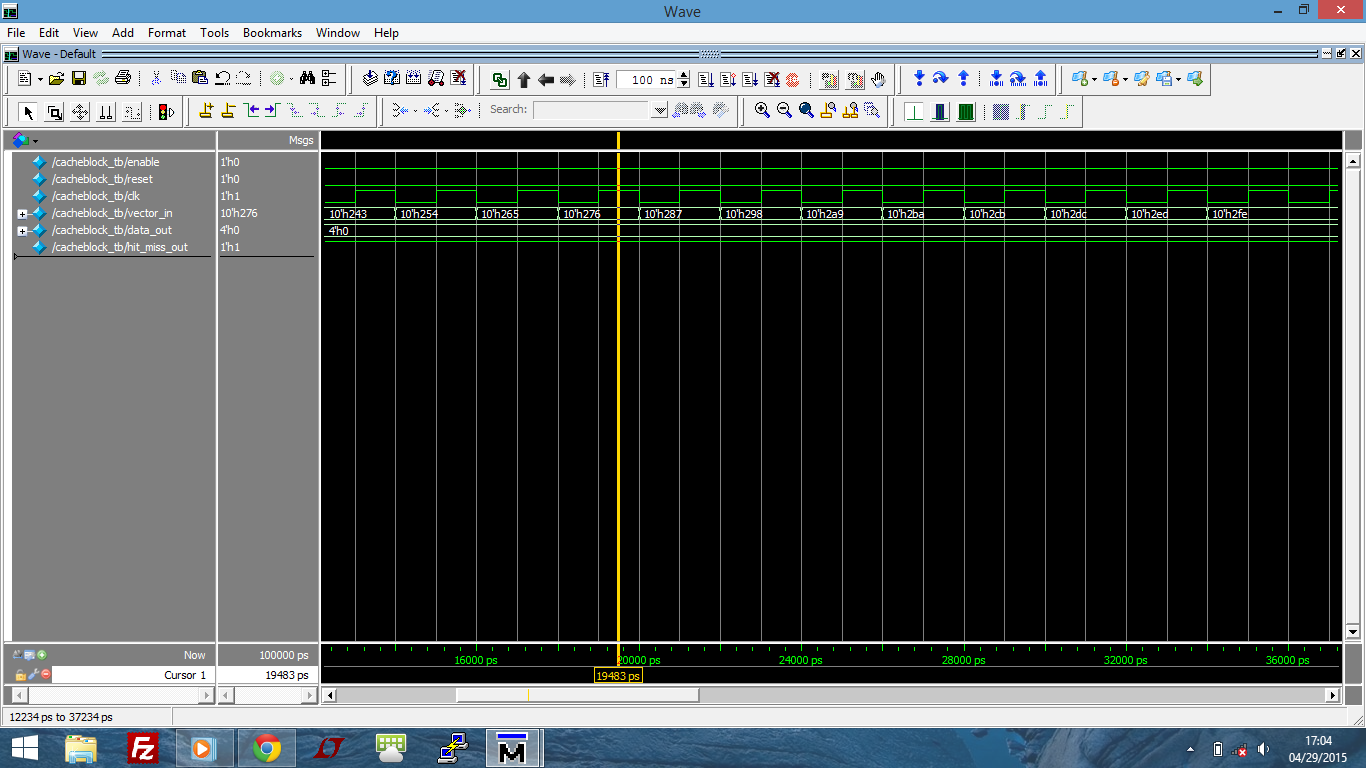


Figure Cache Full Writes

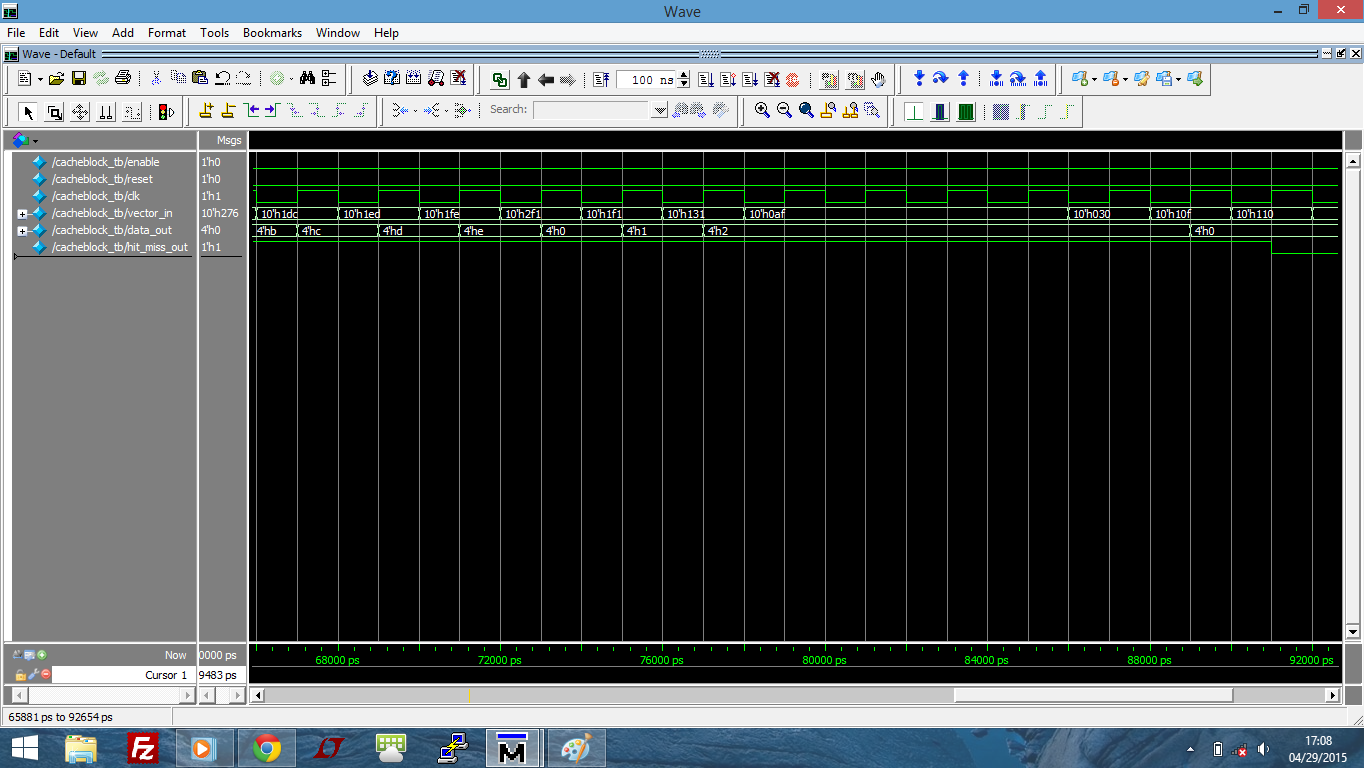


Figure Cache Flash

At 500Mhz, it takes 70 cycles to complete all 20 instructions from the testbench input to the ALU outputs.

All the files are synthesized and then each of the modules/blocks are optimized for area. The synthesys reports are generated and they are all attached in the ZIP folder. The analysis of each report are in the previous table and analyzed below.

**Area:**

The total area of the dualcore processor should be the addition of 2 cores + Arbiter + L2 cache + bus logic (for the output data of L2 and the address input to the L2) which is:

(2 \* 12558) + 62 + 9516 = 34694.

When the dualcore is synthesized, the total area obtained is 34,915 units, which is much closer to the expected value of 34,694 units. The difference is due to the bus logic mentioned above.

The area of the processor, as calculated above, is the addition of ALU + Cache + 3 FIFO = 3028 + 3836 + (3\*759) = 9141; there is seven state machines will add up some area. So in our case the total area is 12558 which already takes into account the seven state machines previously mentioned.

**Power:**

The total dynamic power is the addition of the cell internal power and net switching power. The L2 cache holds the highest power consumption due to its size, furthermore there is data pre-written on it.

The cell leakage power for the whole system approximately adds to its individual internal modules.

**Timing:**

There is significant time mismatches between the data arrival of the top dualcore and the low level modules. This is simply because synopsys does not test the signals inside the lower level timing paths while synthesizing. Finally, the longest time of all the modules was tested/simulated to be around 2.78ns. Since every analysis is different the longest timing might be bigger depending on the specifics of the path selected.

# Synthesis:-

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Dualcore** | **Arbiter** | **L2 Cache** |
|  |  |  |  |
| **Area** |  |  |  |
|  |  |  |  |
| Number of ports | 46 | 7 | 26 |
| Number of nets | 90 | 8 | 696 |
| Number of cells | 30 | 5 | 679 |
| Number of references | 7 | 3 | 28 |
|  |  |  |  |
| Combinational area | 16132.60808 | 16.5888 | 3993.292838 |
| Noncombinational area | 16933.47849 | 44.236801 | 5124.09613 |
| Net Interconnect area | 1849.329565 | 0.866332 | 399.333479 |
|  |  |  |  |
| Total cell area | 33066.08656 | 60.825602 | 9117.388968 |
| Total area | 34915.41613 | 61.691933 | 9516.722447 |
|  |  |  |  |
| **Power** |  |  |  |
|  |  |  |  |
| Cell Internal Power | 273.2139 uW (61%) | 650.8351 nW (52%) | 315.1768 uW (70%) |
| Net Switching Power | 174.2537 uW (39%) | 590.4711 nW (48%) | 135.2258 uW (30%) |
|  |  |  |  |
| Total Dynamic Power | 447.4676 uW (100%) | 1.2413 uW (100%) | 450.4026 uW (100%) |
|  |  |  |  |
| Cell Leakage Power | 154.8283 uW | 277.1545 nW | 42.2318 uW |
|  |  |  |  |
| **Timing** |  |  |  |
|  |  |  |  |
| data arrival time (ns) | 1.84 | 0.22 | 2.78 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Processor** | **Cache** | **FIFO** | **ALU** |
|  |  |  |  |  |
| **Area** |  |  |  |  |
|  |  |  |  |  |
| Number of ports | 38 | 26 | 15 | 32 |
| Number of nets | 286 | 312 | 67 | 283 |
| Number of cells | 217 | 268 | 57 | 285 |
| Number of references | 21 | 21 | 12 | 27 |
|  |  |  |  |  |
| Combinational area | 6147.072009 | 1831.219192 | 265.420802 | 2152.857613 |
| Noncombinational area | 5779.353579 | 1825.689577 | 471.859211 | 721.612791 |
| Net Interconnect area | 632.546948 | 179.653243 | 22.506716 | 154.383752 |
|  |  |  |  |  |
| Total cell area | 11926.42559 | 3656.908769 | 737.280013 | 2874.470404 |
| Total area | 12558.97254 | 3836.562012 | 759.786729 | 3028.854155 |
|  |  |  |  |  |
| **Power** |  |  |  |  |
|  |  |  |  |  |
| Cell Internal Power | 59.8671 uW (77%) | 8.1153 uW (70%) | 22.7557 uW (63%) | 38.000 uW (76%) |
| Net Switching Power | 17.6042 uW (23%) | 3.4517 uW (30%) | 13.3024 uW (37%) | 11.7324 uW (24%) |
|  |  |  |  |  |
| Total Dynamic Power | 77.4713 uW (100%) | 11.5670 uW (100%) | 36.0581 uW (100%) | 49.7328 uW (100%) |
|  |  |  |  |  |
| Cell Leakage Power | 56.0394 uW | 16.9944 uW | 3.4540 uW | 13.2693 uW |
|  |  |  |  |  |
| **Timing** |  |  |  |  |
|  |  |  |  |  |
| data arrival time (ns) | 1.54 | 1.55 | 0.5 | 2.09 |

# Conclusion: -

* The project was a good challenge to design, simulate, and synthesize. Our final project was different from what we initially believed the dual core processor would look like, mostly due to the optimizations that we made in order to maximize the speed, size, and power.
* Initially we planned on possibly implementing several different designs which we learned about in Computer Design, such as memory forwarding, non-blocking caches, and there was even some discussion about Tomasulo's.
* Although we ended up not using any of these designs to ultimately simplify the functionality of the processor and to minimize the area of our design, we had many brainstorming sessions to create the best dual core processor possible.
* Our initial goal was to create a working system, and then and only then was our next goal to optimize an already working system, but throughout the design process we redesigned components and found ways to reduce the complexity and the area of the system.
* We took a minimalistic approach to the design of our system, by optimizing whatever we deemed too needlessly complex to the overall functionality of the system. Although this was a positive in our performance for this particular design, the processor is so specific to the design requirements that as a general purpose processor it would not be ideal.
* Part of the reason for this was because of the barebones ALU which did not have to perform multiplication, or division.
* Another example is our initial design for the L1 cache which had a static replacement policy that would only replace one designated slot in the cache. We computed a lower miss rate using a replacement policy that replaced only one slot once the first level cache after all other slots had been filled. If we had decided to use this particular optimization for our design we would have had less misses than a MRU, LRU, or round robin replacement policy.
* Realistically this approach of static replacement would have not been viable in a real world general purpose processor, but for our particular design we considered optimizing the system to achieve the best possible results.
* Through this project we learned how to optimize the design of a project to achieve the best possible results at the expense of general functionality. This can be good for systems that happen to be very specialized and therefore do not utilize all of the functionality of general purpose processors, and therefore would benefit from varied architectural approaches.
* We believe that just the fact that the project motivated us to apply some of the different computer design concepts that we have learned in the past, as well as to think outside of the box in order to better fulfill the needs of our design demonstrates that we can apply general purpose design optimizations as well as system specific optimizations.