ECSE 425 Lab 2 Group 29

Patrick Juneau
260928476
patrick.juneau@mail.mcgill.ca

Kua Chen
260856888
kua.chen@mail.mcgill.ca

***Abstract— During this lab 2*, we designed and implemented a basic, direct-mapped cache circuit that can handle various cases of memory access. This report will discuss the cache implementation details and testing design.**

I. INTRODUCTION

This is a VHDL project aiming to design a direct-mapped cache, which adopts the write-back policy, 32-bit words, 128-bit blocks (4 words), 4096-bit of data storage, flag bits 'valid' and 'dirty'. There are two major components we need to implement: cache circuit design (cache.vhd), tests for validating cache circuit design (cache_td.vhd). The memory circuit is provided. It has 32-bit address mode and 32,768 bytes (32,768/4 = 8192 words) data storage. It should be noticed that the memory is byte-addressable, meaning that each individual address corresponds to one specific byte.

The dirty bit is a flag in each cache block to indicate whether the data in the cache is modified (different from the one in the memory). If the dirty bit is 1, then the cache data is modified and dirty. If the dirty bit is 0, then the cache data is unmodified and so-called clean.

The valid bit is a flag in each cache block to indicate whether the data is brought from the memory. In our project, the only case for cache data to be invalid is the situation where the system is just initialized.

The tag is a series of bits used to determine whether the cache block contains the data which the CPU requests.

A direct-mapped cache means each block from memory can only appear in one block in the cache. The mapping is achieved by the mod operation.

II. EXPLANATION

2.1 Tags and various offsets

The memory has 32,786 bytes and this number is equivalent to 2^15. As mentioned in the project description, only the lower 15 bits of the address will be used. The remaining bits are always 0 and can be ignored. In the system, reading and writing words are word-aligned and 1 word has 4 bytes. Therefore, 2 bits are needed for the byte offset. In terms of the word offset, each block has 4 words so 2 bits are needed for the word offset. The cache has 4096-bit data storage and each block is 128-bit. Therefore, there are 32 blocks in the cache, which is equivalent to 2^5. Therefore, 5 bits are needed for the block index. The remaining bits contribute to the tag. As mentioned above, only the lower 15 bits of the address contain useful information. Therefore, the tag will have 6 bits (15-5-2-2=6). It is definitely a feasible solution to have 23 bits for the tag (32-5-2-2=23) but some space is wasted in this design. Therefore, we adopted the 6-bit tag implementation to optimize the performance. Addressing fields are summarized in the table below and there is a visual representation of the cache design in Figure 1.

	Length	Position in address
Tag	6	9th to 14th bits
Block Index	5	4th to 8th bits
Word offset	2	2nd and 3rd bits
Byte offset	2	0th and 1st bits

Table 1: Addressing fields

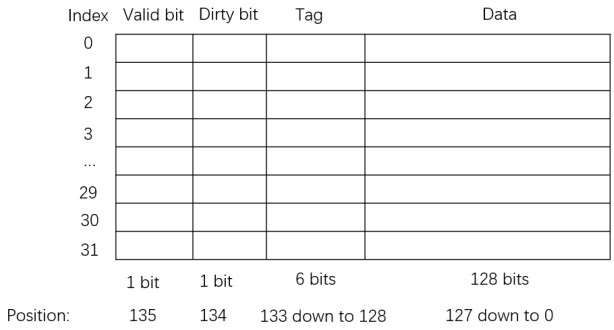


Figure 1: Visualization of the cache design

2.2 State machine for the cache

The most important design to fulfill the cache functionality is to implement a state machine that represents the behaviour of the cache. At first, the cache is initialized with the “initial” state. Then read and write operation will trigger the cache to enter the “read” or “write” state respectively.

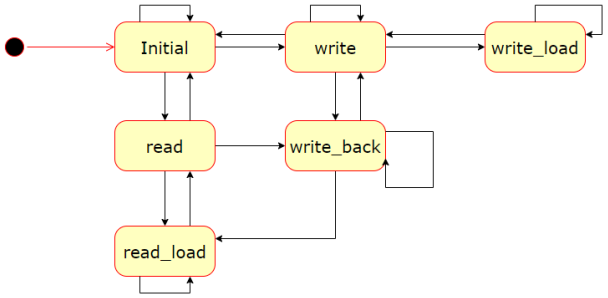


Figure 2: State machine for the cache

In the “read” state, if the system reads the valid data with a tag hit, then the cache directly outputs the word without accessing the memory and the cache returns to the “initial” state. Notably, the dirty bit is not important in this case. From the “read” state, the cache will move to the “read_load” state if the cache block has a miss and the dirty bit is 0. In this case, no write-back will be triggered because the cache data is clean. Meanwhile, the cache will enter the “write-back” state, if the cache block has a miss and the dirty bit is 1.

In the “write” state, if the system writes to valid data with a tag hit, then the cache data will be updated without accessing the memory. Also, the dirty bit is not important in this case. Next, The cache will go to the “write_load” state if the data is clean and (miss or invalid). At least one of the “miss” condition or “invalid” condition needs to be true. It will also go to the “write_load” state if the cache just returns from the “wirte_back” state. This is because after finishing writing the old data back into memory, the new data also needs to be loaded from the memory to the cache. Therefore, another load is needed after returning from the “write_back”. To enter the “write_back” state, the dirty bit should be 1 and there is a miss. The cache can loop at the “write” state due to a wait request.

The cache will loop at the “read_load” or the “write_load” or the “write_back” state respectively because the memory can only output 1 byte (8-bit) at a time and a complete block contains 16 bytes (4 words). Therefore, the loading process will be repeated 16 times. For example, in the write-back, the cache will write the old data to the memory 1 byte a time and 16 times in total.

2.3 Testing

These parameters are: valid/invalid, dirty/not dirty, read/write, tag equal/tag not equal. There are 16 different possible combinations of tests to run while using the parameters Clean, Dirty, Valid, Invalid, Hit or Miss. The first 6 tests are impossible cases.

- Test 1 Invalid Dirty Read Hit
- Test 2 Invalid Dirty Write Hit
- Test 3 Invalid Dirty Read Miss
- Test 4 Invalid Dirty Write Miss
- Test 5 Invalid Clean Read Hit
- Test 6 Invalid Clean Write Hit

The tests with invalid and hit are impossible since there is nothing stored in cache yet to be hit. Also, the tests with invalid and dirty are impossible since for it to be dirty, it has to have been written to cache which would make it valid.

- Test 7 Invalid Clean Read Miss

This test tries to read the address "00000000000000000000000000111111", but since there is nothing there at that cache block yet, there is a

read miss as seen in the readdata in test 7 in the appendix section

- Test 8 Invalid Clean Write Miss

This test tries to access the address "00000000000000000000000000001111" and write "BBBBBBBB" to perform a write, but since there is nothing in that block yet, the data is stored in cache making it clean and the dirty bit is changed to 1.

- Test 9 Valid Dirty Read Hit based on test 8

Using the same address as the previous test and knowing that it is now Valid Dirty, we read the data from the cache and get a hit as shown to be “BBBBBBBB”

- Test 10 Valid Dirty Read Miss based on test 9

Making sure the address used maps to the same block in the cache as before "00000000000000000000000100001111", but has a different tag, we attempt to read from that address and since it misses, the previously stored data is sent to main memory and the new tag from the read is stored. This makes the dirty bit set back to 0 making it clean.

- Test 11 Valid Clean Read miss based on test 10

From the previous test, that stored from address "00000000000000000000000100001111" making it clean, we now try to read from address "00000000000000000000000000001111" which results in a miss since the tags do not match and readdata is erroneous in the screenshot.

- Test 12 Valid Clean Read Hit based on 11

From the last test, since we tried to read from "00000000000000000000000000001111" and it missed, it got stored in the cache block and so if we try to read from the same address it now results in a hit.

- Test 13 Valid Clean Write Hit based on 12

We now write to the same address as the last test since we know that the cache block is Valid and Clean and when writing we use the same address "00000000000000000000000000001111" and write "BCBCBCBC" so it results in a hit. This makes the dirty bit go back to 1.

- Test 14 Valid Dirty Write Miss based on 13

Since it is dirty from the last test, we write using the address "00000000000000000000000100001111" and write "BDBDBDBD" which will result in a miss since the tag that is stored corresponds to that of the last test. The dirty bit is still 1 after this test and for the next test we need it to be clean; therefore, we read using the address "00000000000000000000000000001111" which will result in a miss causing the old data to be stored in main memory and the new tag form the read to be updated in the cache block which will make it clean.

- Test 15 Valid Clean Write Miss based on 14

Since the last address that was sent to the cache was "00000000000000000000000000001111", we try to write to "00000000000000000000000100001111" and write "CDCDCDCD"which will result in a miss, but the dirty bit will be set to 1 as a result of this write.

- Test 16 Valid Dirty Write Hit based on 15

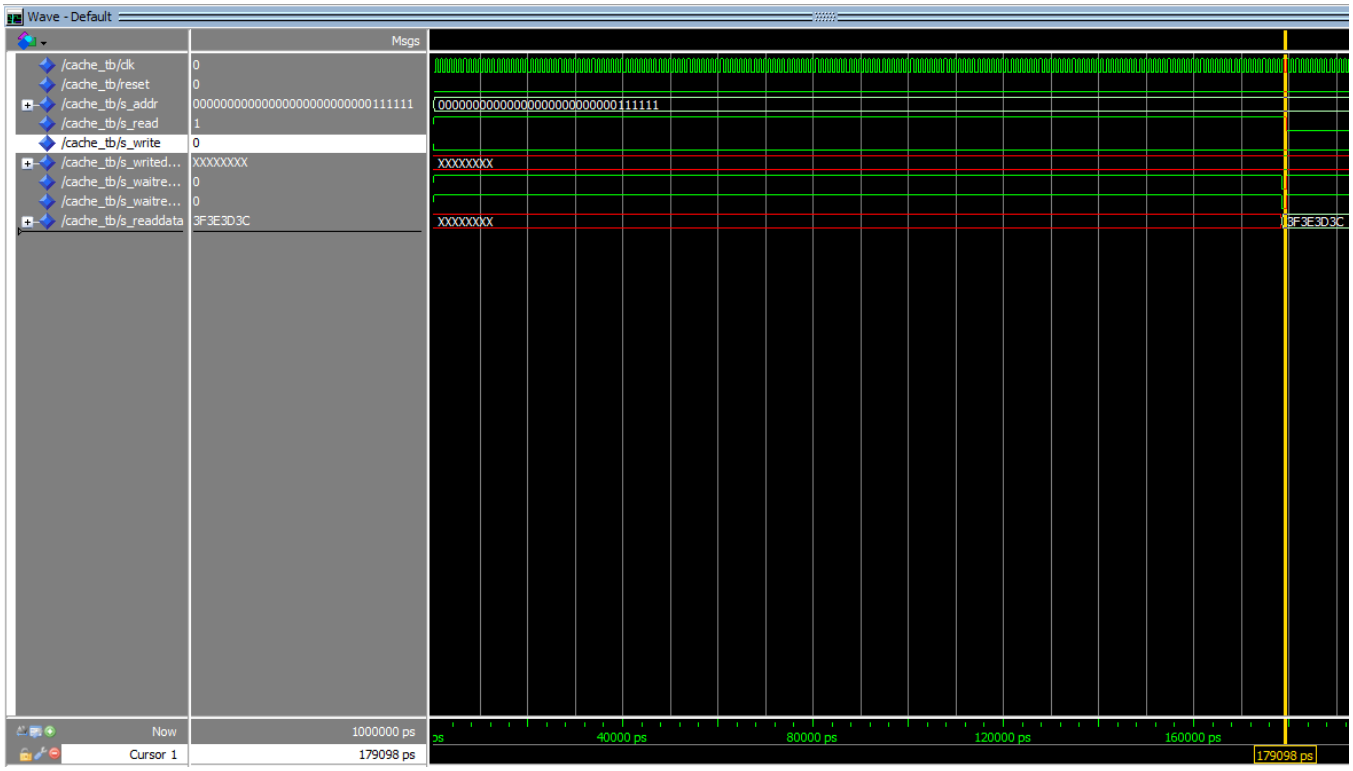
We write to the same address as the last test but with write data "ACBDABCD" which will result in a hit. Also, it was not necessary, but we read at the end to verify if the writedata was stored and it was since the readdata is also "ACBDABCD".

III. CONCLUSION

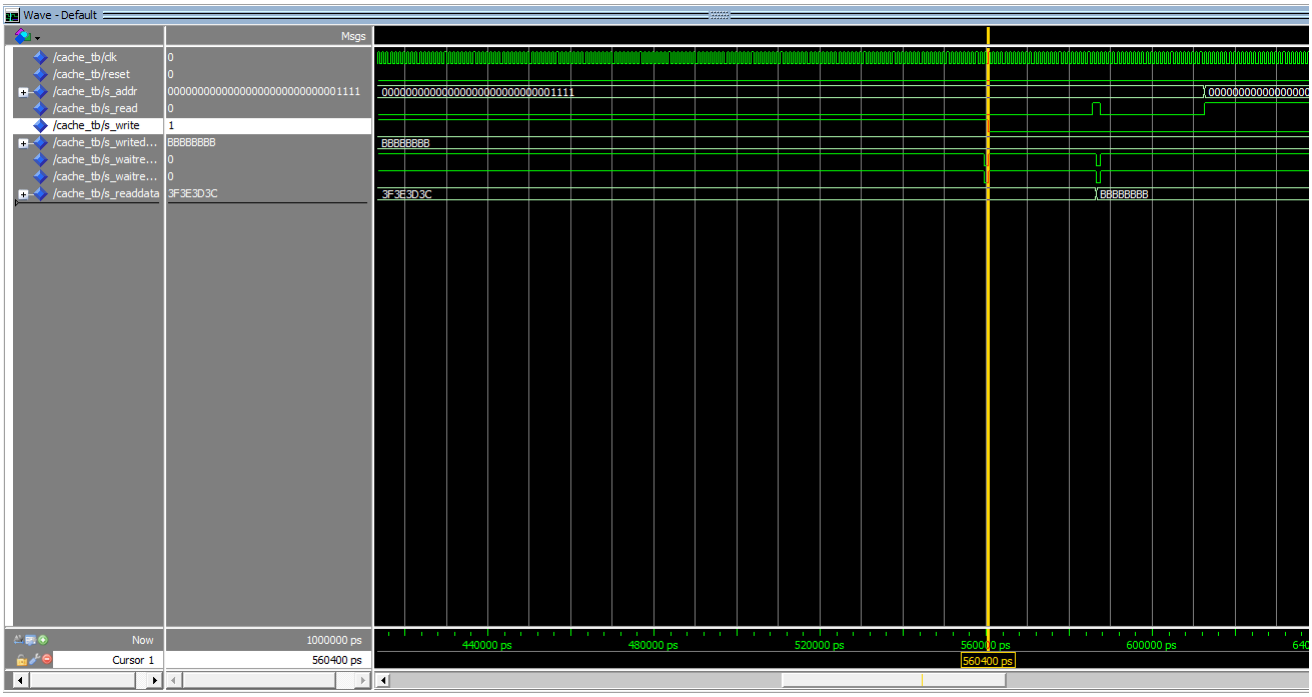
During this lab, we implemented a direct-mapped cache which helps the CPU to achieve read and write operations. A finite state machine is designed for the cache in order to achieve the corresponding

functionalities. A test suit is developed to validate our cache design, taking several parameters into account.

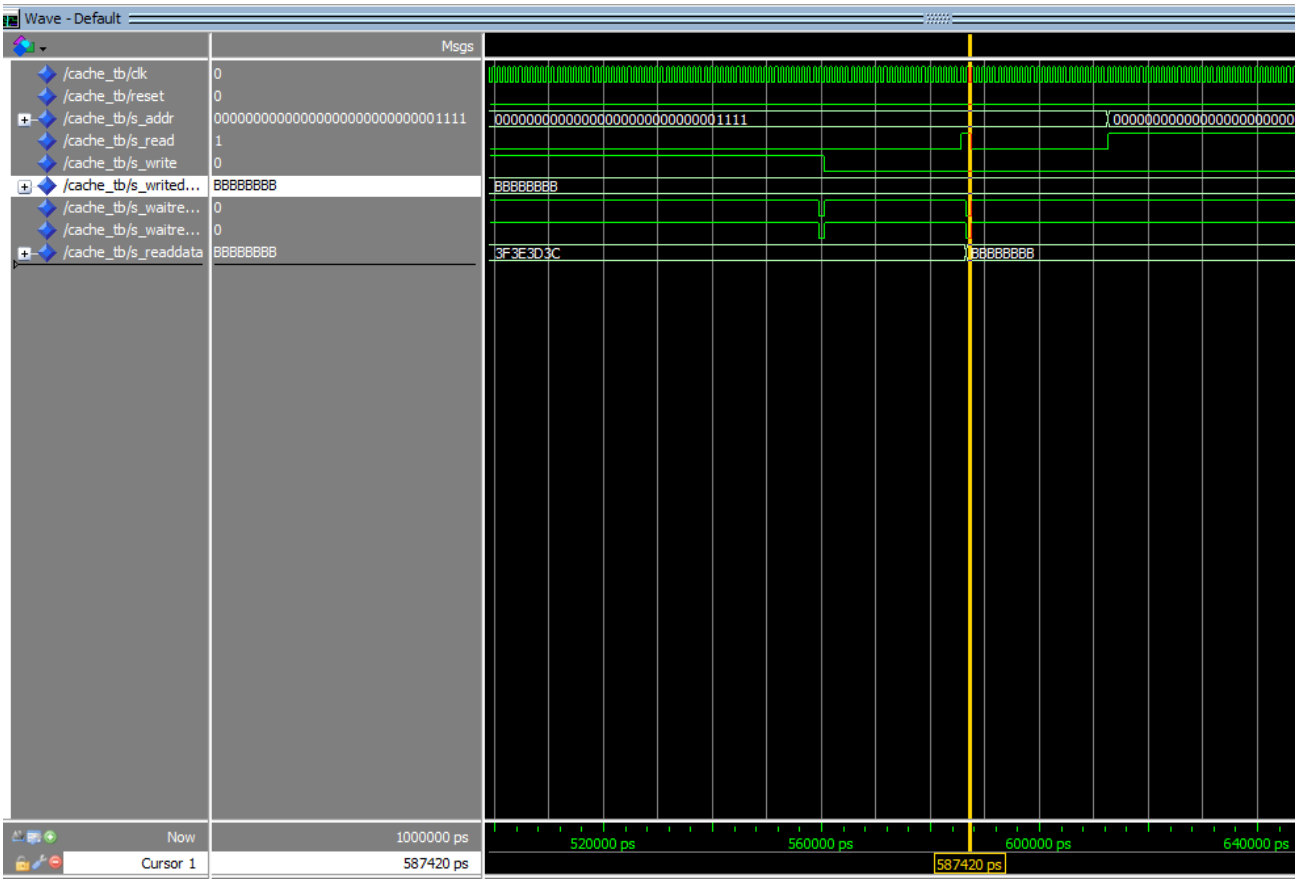
Appendix



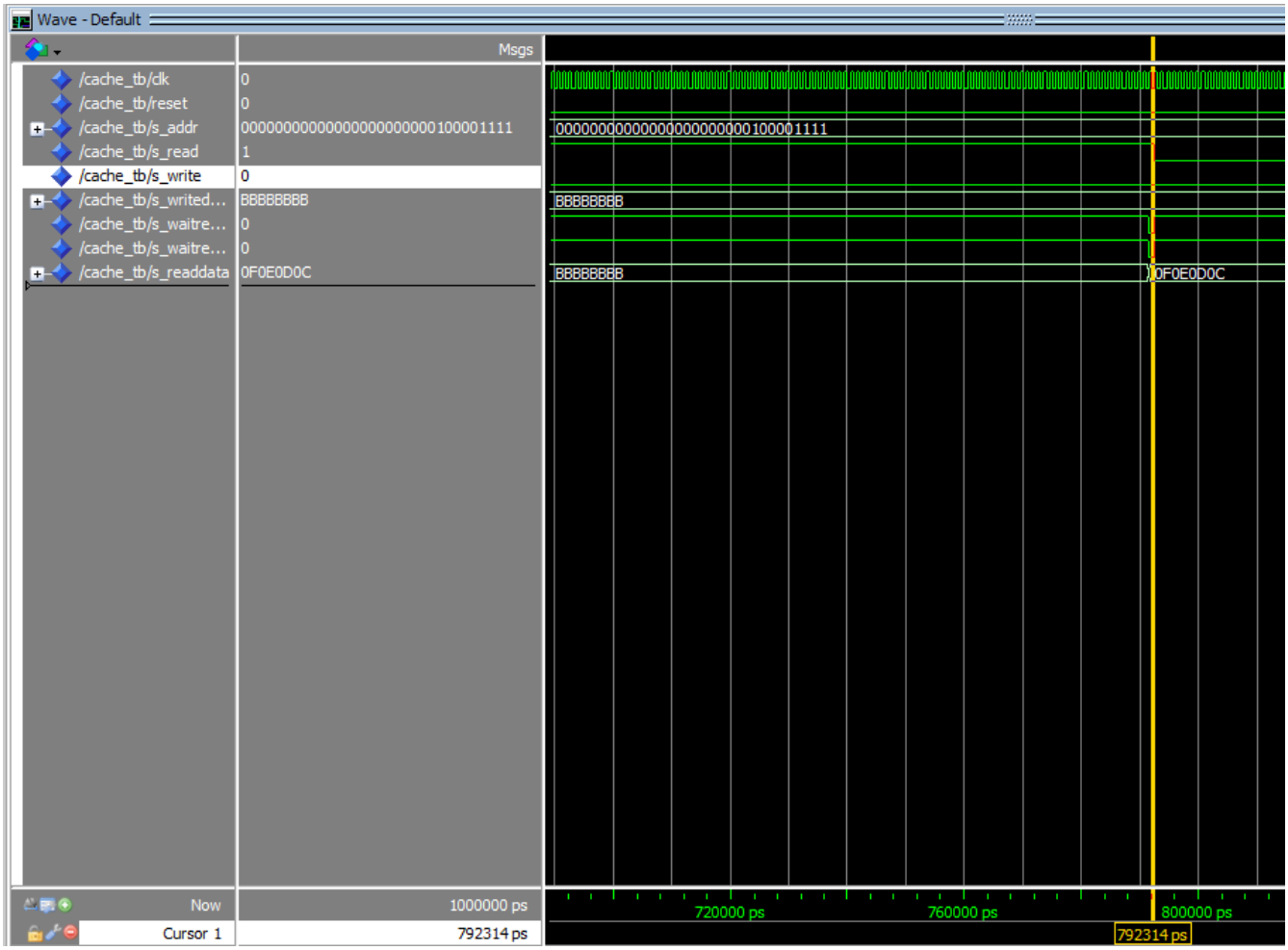
Test 7 Invalid Clean Read Miss



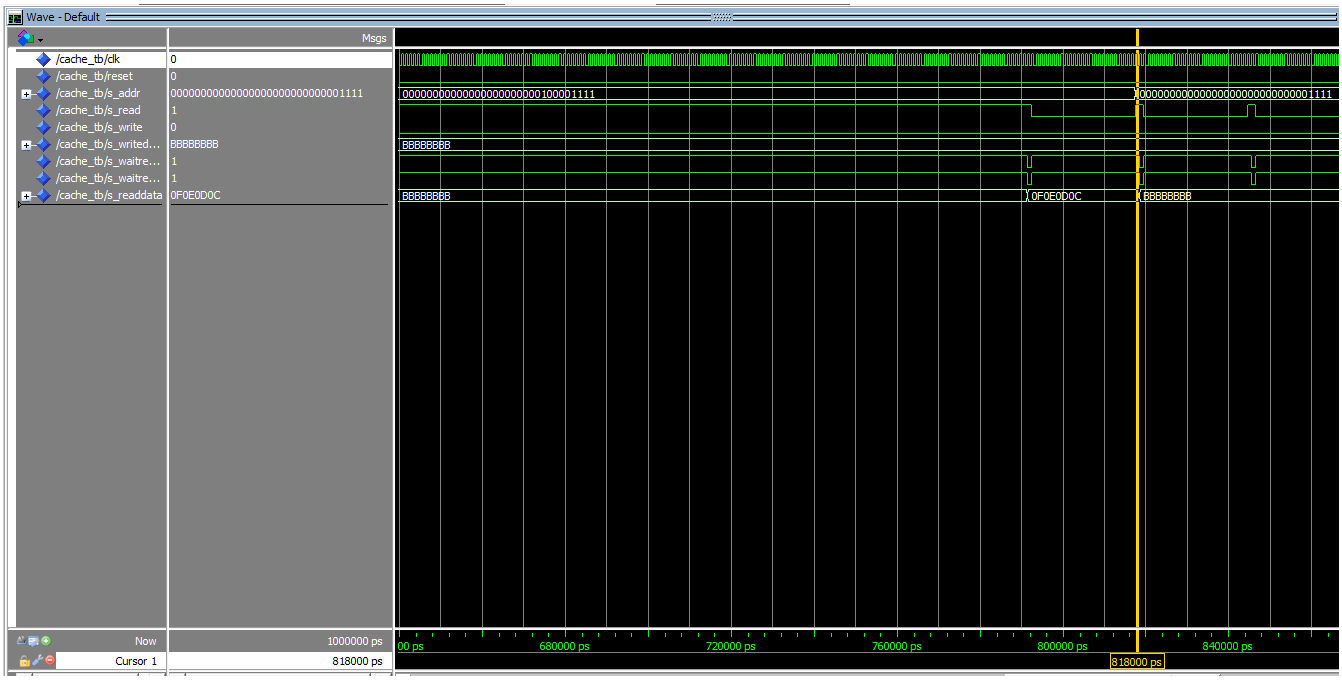
Test 8 Invalid clean write miss



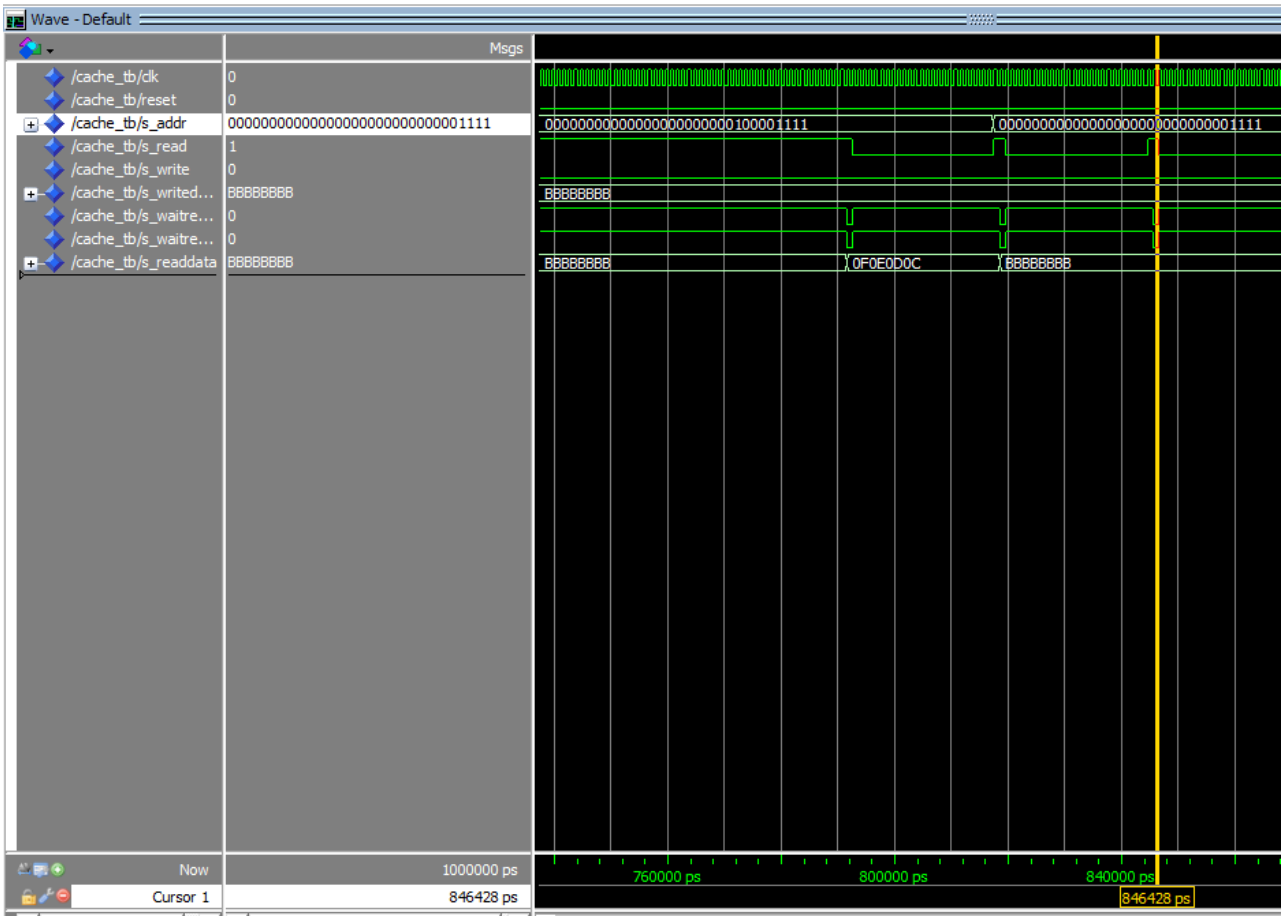
Test 9-->Valid Dirty Read Hit based on test8



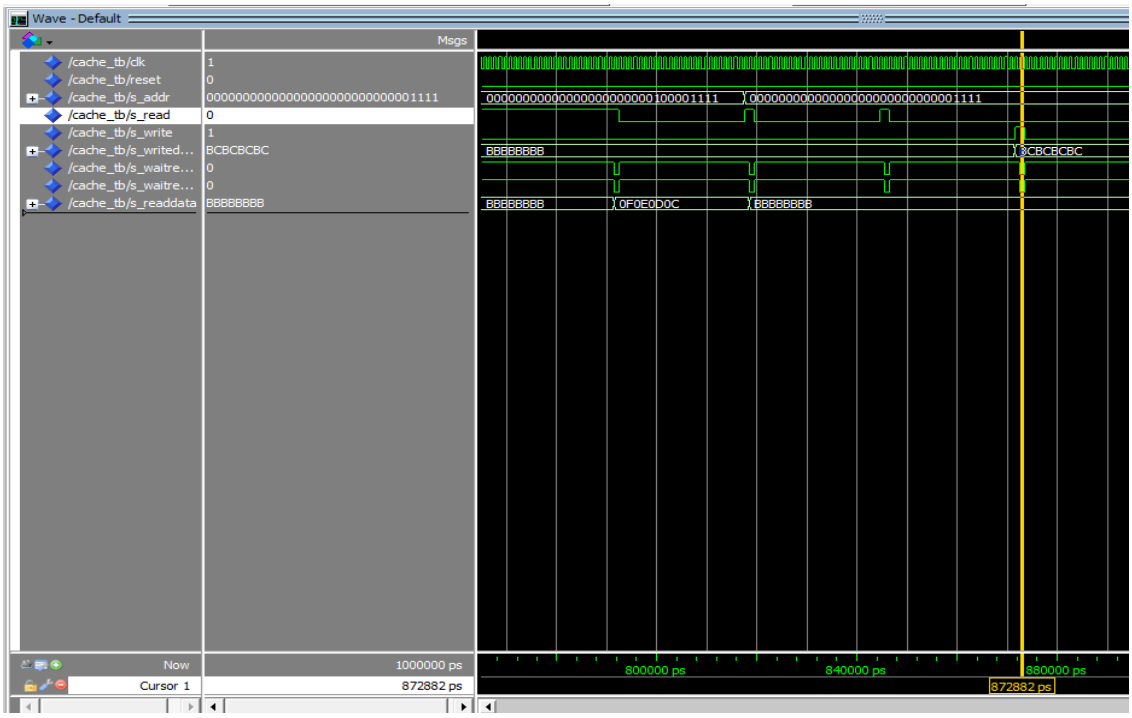
Test 10 -->Valid Dirty Read Miss based on test9



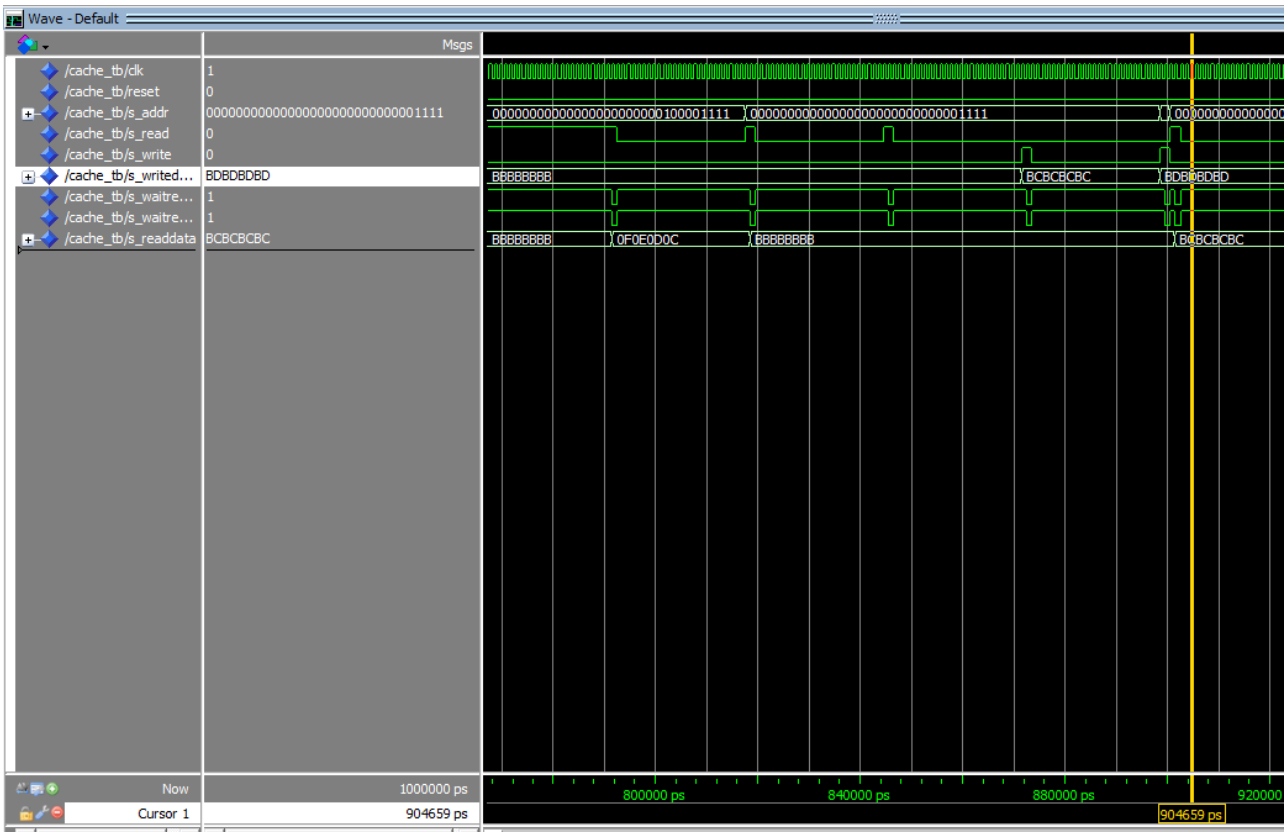
Test 11 ->Valid Clean Read miss based on test10



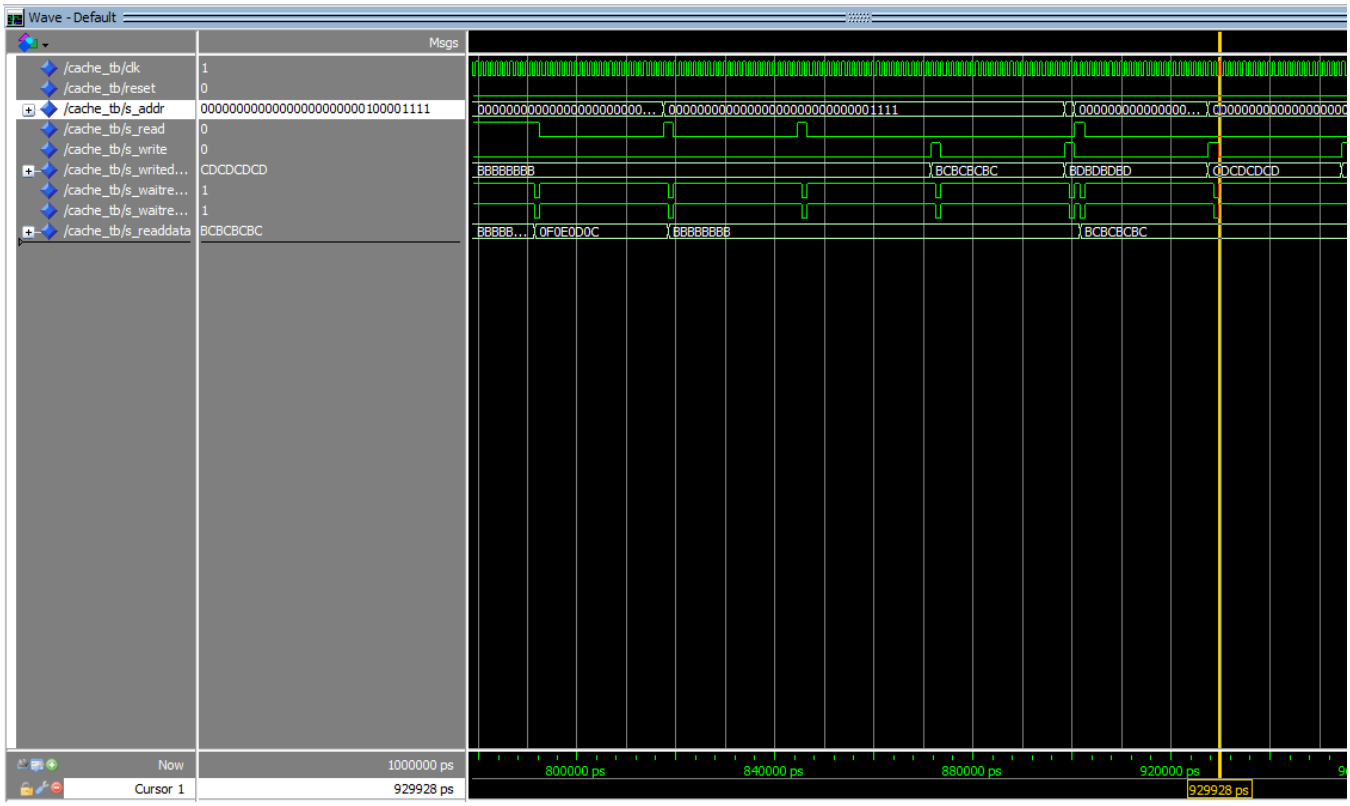
Test 12 ->Valid Clean Read Hit based on 11



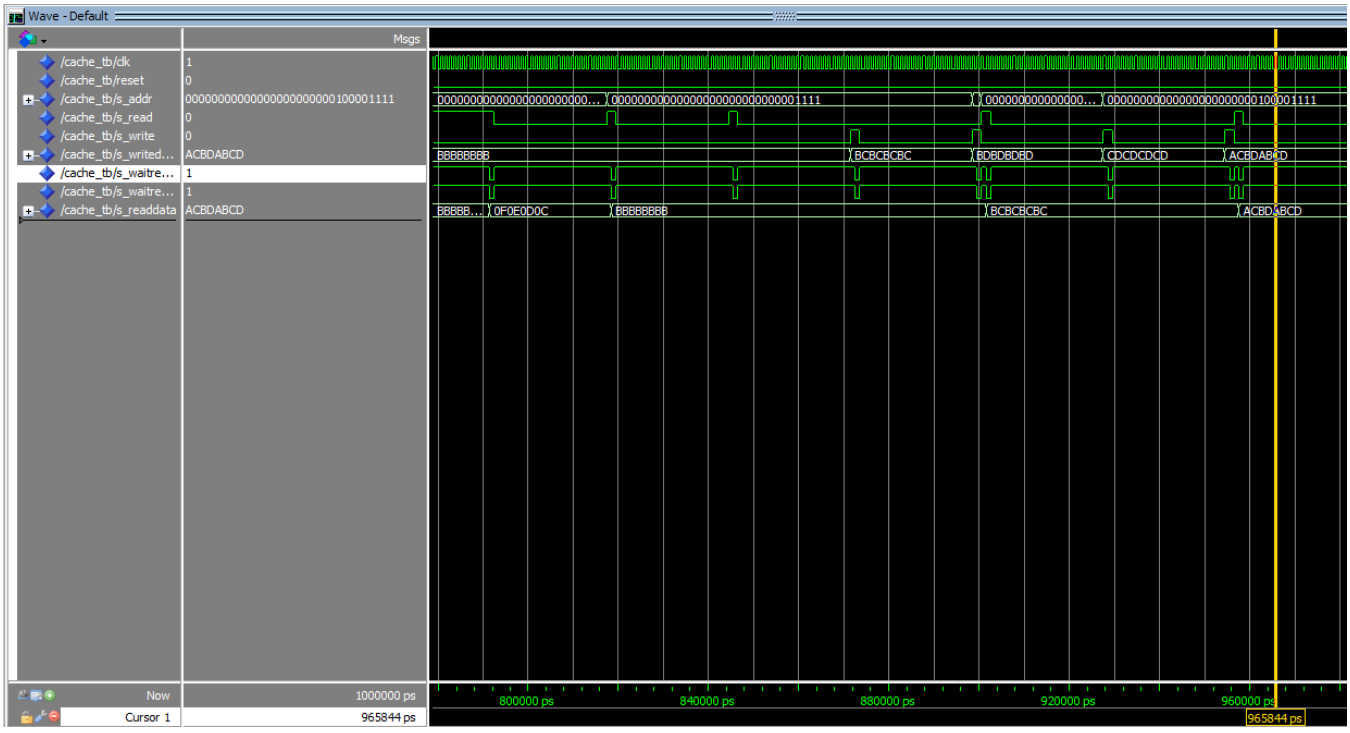
Test 13 ->Valid Clean Write Hit based on 12



Test 14 ->Valid Dirty Write Miss based on 13



--Test 15 ->Valid Clean Write Miss based on 14



Test 16 ->Valid Dirty Write Hit based on 15