Flexible read-write delay memory with the ability of getting initialized or dumped from/into a file

In this guidance document we describe an RTL model for a memory that not only has the abilities of an ordinary sequential memory---storing and reading data---, but also supports adding flexible delay in read and write operations while give a chance to users to initialize the content of the memory from an ascii file or save its content whenever during a simulation.

- You can find the VHDL code of the memory in the “Main_Memory.vhd” file
- Also, you can find its test-bench in the “tb_Main_Memory.vhd”

Overview

The designed memory gives users some flexibilities that make it compatible with different applications. A user can define the size of the memory in two dimensions, height, which represents the number of existing words in the memory, and the width that determines the size of the memory-words in bits.

This memory is a synchronous one and needs a clk signal to work synchronously with other parts of the circuit. It is designed as a single port ram, can operates only a single command---read or write---at each clk edge and has only one port for data which decide to read from or write to due to its controlling signals.

This memory has the ability to insert some delays in front of reading or writing procedures to simulate the real existing delays in some kind of memories. Consequently, it generates some controlling signals to tell the user that his requested data is ready on the data-line or it has been sampled and written in the memory safely.

Another important ability of this memory is its content loading /saving. To be more specific, users can load the content of the memory or save it into a file by triggering corresponding signals and this could happen whenever during the simulation. This feature helps designers to analyze their memory continent outside the simulator and with other tools.

Definition of Memory signals and variables and their corresponding functionalities

You can find the memory block in a module, with the name of, **Main_Memory**. This module has some initialization variables that are represented in its generic section, refer to Table 1.

Moreover the memory block has some signals that provide proper needed interactions between memory and outside world. Their names and definitions are brought in Table 2.

Note that for the acceptable values of the signals and generics user has to take care and there would be nothing in the simulation that prevents you from assigning wrong values to this block's signals.

Table 1

Generic-variables of the Main_Memory block and their description, functionality and range of acceptable values

Variable name	Description	Functionality	Acceptable Values
File_Address_Read	Address of initializing file	Initializing memory values from this file address	Any valid address and name of file in the system
File_Address_Write	Address of a file for saving memory content	content of the memory will be saved in this pointing file	Any valid address and name of file in the system
Mem_Size_in_Word	Size of memory in word	Determines the height of the memory	[1, 2^{32}-1]
Num_Bits_in_Byte[1]	Determines the number of bits in each byte-block of memory	Determines the width of memory elements	[1, 2^{32}-1]
Num_Bytes_in_Word	Determines number of bytes in each word-block of memory	Determines the width of memory in terms of byte	[1, 2^{32}-1]
Read_Delay	Defines number of inserted delays for read operation	Number of delays, in clk, in front of read operation	[0, 2^{32}-1]
Write_Delay	Defines number of inserted delays for write operation	Number of delays, in clk, in front of write operation	[0, 2^{32}-1]

Table 2

Defined signals of the Main_Memory block and their description, functionality and range of acceptable values

Signal Name	Description	Functionality	Acceptable Values
clk	Clock, The synchronization signal	Synchronize memory with other modules.	This should be periodic signal [2 ps, …]
Address	Memory address	Defines the address of the memory that is being accessed	[0, $2^{\text{Mem\_Size}}$-1]

[1] Leave it to its initial value

Word_Byte	Word or Byte	Tell the memory that we are interacting with it in word or byte size	1= word size interaction[2] 0= byte size interaction[3]
we	Write Enable	Informs the memory that a write operation is performing	Logical 0 or 1 0 = deactivated 1 = activated
wr_done	Write done	Writing the data to the memory has been finished and now that data is accessible by a probable read operation	Logical 0 or 1 0 = not done 1 = done
re	Read Enable	Informs the memory that a read operation is performing	Logical 0 or 1 0 = deactivated 1 = activated
rd_ready	Read data ready	Reading the data from the memory has been finished and now that data is ready at the data output	Logical 0 or 1 0 = not ready 1 = ready
data[4]	Data port	The data gets ready on, or reads from this port, respect to the kind of operation (read or write)	$[0, 2^{\text{Num\_Bits\_in\_Word}}-1]$
initialize	Initialization of the memory	Reads binary data from “File_Address_Read” and puts them into the memory, in a sequential order	Logical 0 or 1 1, 0 , or 1→0 : do nothing! Only if 0→1: initialize
dump	Dumping the memory	Writes content of memory, in binary format, into “File_Address_Write”, in a sequential order	Logical 0 or 1 1, 0, or 1→0 : do nothing! Only if 0→1: dumping

[2] When it is set to '1', it means that you need to have access to the words so by changing the address you can have access to different words in the memory.

Note: address in this memory points to the byte numbers in the memory. In other words, when you want to have access to two consecutive words in the memory you have to add the address by 4, not by 1, since next word is 4 bytes far from the current word. For example, when word_byte='1', we intend to have word access, address=0 points to the first word in the memory while address=12 points to the 4th word.

Note: If you use address values other than multiple of 4, the memory module interpreter your address to a number which is nearest multiple of 4 to your presented address while it must be less than or equal to your presented address.

[3] When word_byte='0', in this case, you intend to work with the bytes in the memory and you asked byte access. So, you have to be more careful about your address while each address is pointing to a specific byte in the memory not a word. For example, if you have defined a 256 word size memory (Mem_Size_in_Word =>256) you have a 4*256=1024B memory, which starts from address=0 and ends to the address=1023.

In the case of byte access to memory module, whenever you ask for a byte, memory module will provide that byte for you, respect to the observed address, on its bus and it will put that byte's value on the least signification byte of data-bus. For example, you read from memory in address=3, while the memory content in the first word (word0) is : 0X"FFAABBCC", then you have to expect to see AA (third byte of the first word) on the least signification byte of data bus: 0X"ZZZZZZAA", which in this example "Z" represent high impedance value.

This behavior is exactly the same when you want to write to the memory in byte order. For example, you want to write 0X"3D" to Byte# 151 and word_byte='0', in this case, you have to put address =151 and put "3D" value on the least significant byte of data-bus(0X"ZZZZZZ3D"). Then you have to expect to see that Byte value ("3D") in the memory content if you follow the write signalling process, presented in the memory documentation

[4] Be aware, this is an INOUT port so whenever you do not use it assign it to ‘Z’ (like the code for the test-bench).

Simulation of the memory

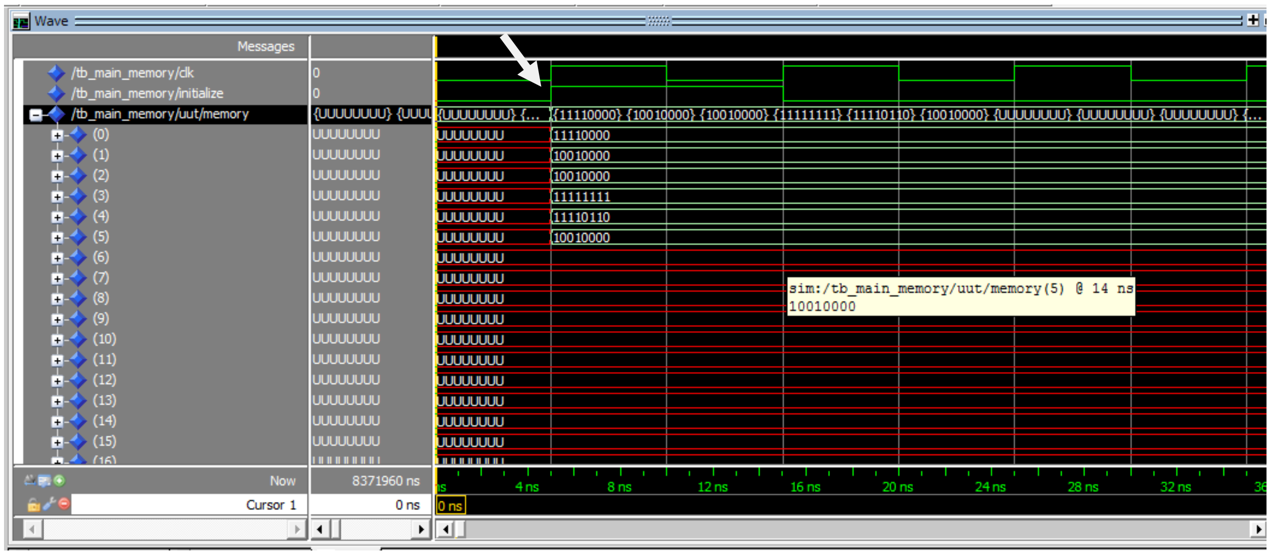
In this section some examples for working with the Main_Memory module are being presented and meanwhile the needed signaling for interacting with this memory is being explained.

You can find corresponding coding and signaling examples of the corresponding below subsections in the "tb_Main_Memory.vhd" file.

1. Initializing the memory from a file

For this procedure there is only a need for triggering the "initialize" signal.

As it is shown in the Figure 1, when the "initialize" is getting triggered (0→1 edge) the memory content is being initialized by the values from the "File_Address_Read" file, which in this case there are only 6 values and rest of the file is empty which leads to memory not having determined values, "U" (undefined), for the rest of the words.



```
Messages
/tb_main_memory/clk          0
/tb_main_memory/initialize   0
/tb_main_memory/uut/memory   {UUUUUUUU} {UUUU   {UUUUUUUU} {...   {11110000} {10010000} {10010000} {11111111} {11110110} {10010000} {UUUUUUUU} {UUUUUUUU} {UUUUUUUU} {...
  (0)                        UUUUUUUU           UUUUUUUU          11110000
  (1)                        UUUUUUUU           UUUUUUUU          10010000
  (2)                        UUUUUUUU           UUUUUUUU          10010000
  (3)                        UUUUUUUU           UUUUUUUU          11111111
  (4)                        UUUUUUUU           UUUUUUUU          11110110
  (5)                        UUUUUUUU           UUUUUUUU          10010000
  (6)                        UUUUUUUU           UUUUUUUU
  (7)                        UUUUUUUU           UUUUUUUU
  (8)                        UUUUUUUU           UUUUUUUU
  (9)                        UUUUUUUU           UUUUUUUU
  (10)                       UUUUUUUU           UUUUUUUU
  (11)                       UUUUUUUU           UUUUUUUU
  (12)                       UUUUUUUU           UUUUUUUU
  (13)                       UUUUUUUU           UUUUUUUU
  (14)                       UUUUUUUU           UUUUUUUU
  (15)                       UUUUUUUU           UUUUUUUU

sim:/tb_main_memory/uut/memory(5) @ 14 ns
10010000

Now       8371960 ns
Cursor 1  0 ns
```

Figure 1

Initializing process and its corresponding signaling

2. Reading from a specific address of the memory

In this procedure, first you have to define the address and activate the "re" signal not forget to determin the type of interaction that you need to have with the memory (word or byte) set Word_Byte signal to appropriate value, then, in the next clks (due to defined # of read delays), the "data" signal sets to the corresponding memory's content and "rd_ready" signal asserts to '1' while lasts high for one clk cycle for each memory block read. So, you need to sample and save the data value when the "rd_ready" is high.

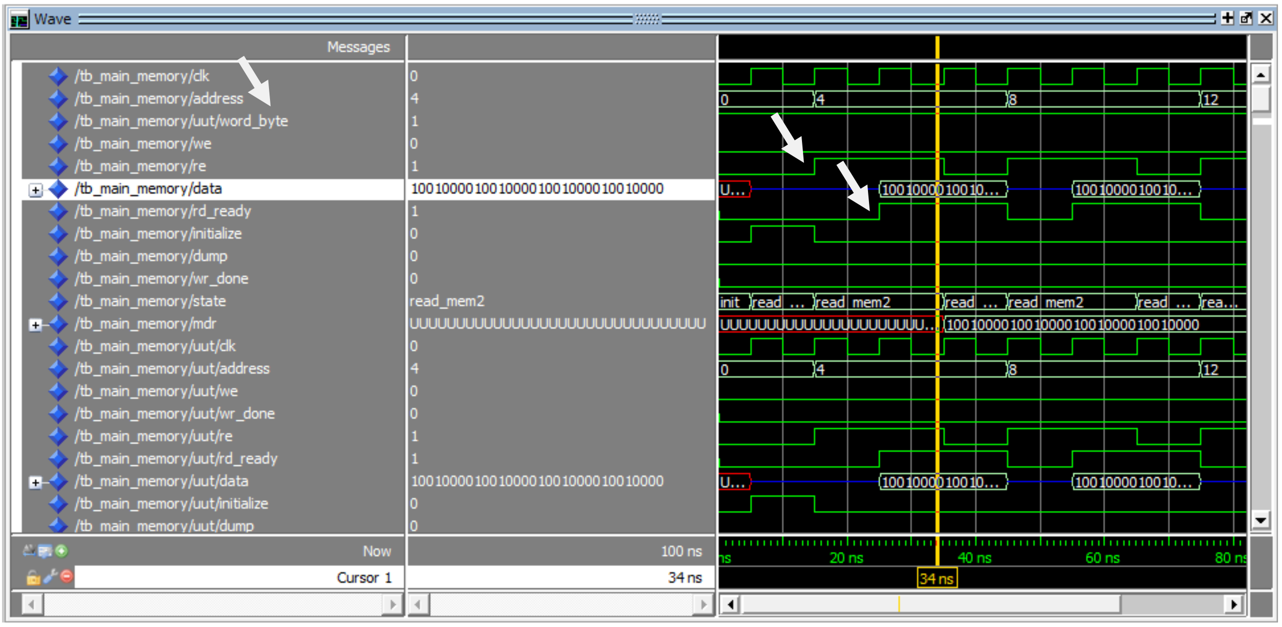
1) Reading a single entry from memory

   a) Zero delay for memory read interaction

   In this case, we have to provide proper address and activate the “re” signal, as depicted in the Figure 2, then we have to wait for the “rd_ready” signal to get asserted and indicates that our requested data is available on the bus. After that “rd_ready” is asserted, we can change the address and deactivate the “re”. For the next read operation, we have to enable “re” signal again and expect to see the data in the consecutive clock cycle. You can see some consecutive single read operations in the Figure 2(word read) and Figure 3 (byte read) while “Read_Delay” is set to zero. This means that in the best case (fastest mode), it takes one clock cycle for memory to provide the requested data and by this method of reading from memory we lose one clock cycle.

   b) More than zero delay for memory read interaction

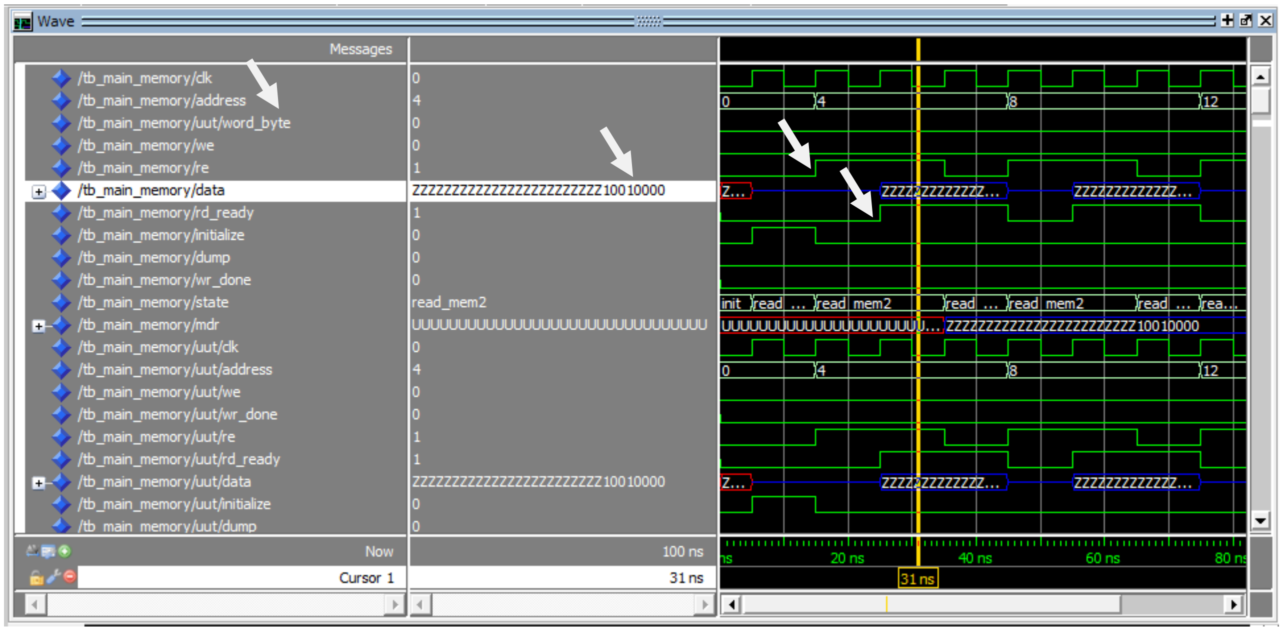
   In this case, there is nothing more than previous case that you have to consider for requesting a data from memory. You have to put your address on the bus and keep the “re” signal activated till you see “rd_ready”. When you see “rd_data” asserted, it is time to latch your data and deactivate the “re” signal. You can put the new address after seeing “rd_ready” signal and wait for the consecutive activation of “rd_data”, still keep in mind whenever you put new address you have to activate “re” signal and keep it activated till you see the “rd_ready”. This procedure is show in Figure 4 for a byte read from address=4 and read_delay=3 clock cycles[5].



Messages	
/tb_main_memory/clk	0
/tb_main_memory/address	4
/tb_main_memory/uut/word_byte	1
/tb_main_memory/we	0
/tb_main_memory/re	1
/tb_main_memory/data	10010000100100001001000010010000
/tb_main_memory/rd_ready	1
/tb_main_memory/initialize	0
/tb_main_memory/dump	0
/tb_main_memory/wr_done	0
/tb_main_memory/state	read_mem2
/tb_main_memory/mdr	UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
/tb_main_memory/uut/clk	0
/tb_main_memory/uut/address	4
/tb_main_memory/uut/we	0
/tb_main_memory/uut/wr_done	0
/tb_main_memory/uut/re	1
/tb_main_memory/uut/rd_ready	1
/tb_main_memory/uut/data	10010000100100001001000010010000
/tb_main_memory/uut/initialize	0
/tb_main_memory/uut/dump	0
Now	100 ns
Cursor 1	34 ns

Figure 2

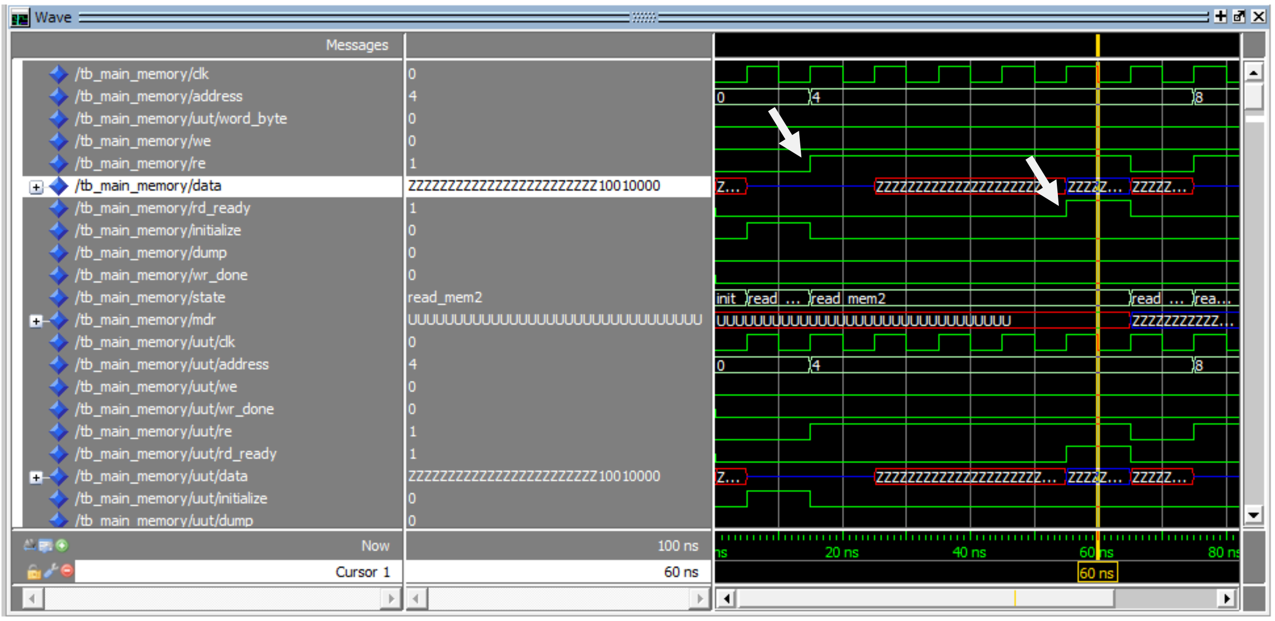
Reading from memory (single word) read when generic variable of “Read_Delay” is set to zero



```
Messages
/tb_main_memory/clk                 0
/tb_main_memory/address             4
/tb_main_memory/uut/word_byte       0
/tb_main_memory/we                  0
/tb_main_memory/re                  1
/tb_main_memory/data                ZZZZZZZZZZZZZZZZZZZZZZZZ10010000
/tb_main_memory/rd_ready            1
/tb_main_memory/initialize          0
/tb_main_memory/dump                0
/tb_main_memory/wr_done             0
/tb_main_memory/state               read_mem2
/tb_main_memory/mdr                 UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
/tb_main_memory/uut/clk             0
/tb_main_memory/uut/address         4
/tb_main_memory/uut/we              0
/tb_main_memory/uut/wr_done         0
/tb_main_memory/uut/re              1
/tb_main_memory/uut/rd_ready        1
/tb_main_memory/uut/data            ZZZZZZZZZZZZZZZZZZZZZZZZ10010000
/tb_main_memory/uut/initialize      0
/tb_main_memory/uut/dump            0
Now                                 100 ns
Cursor 1                            31 ns
```

Figure 3

Reading from memory (single byte) read when generic variable of "Read_Delay" is set to zero



```
Messages
/tb_main_memory/clk                 0
/tb_main_memory/address             4
/tb_main_memory/uut/word_byte       0
/tb_main_memory/we                  0
/tb_main_memory/re                  1
/tb_main_memory/data                ZZZZZZZZZZZZZZZZZZZZZZZZ10010000
/tb_main_memory/rd_ready            1
/tb_main_memory/initialize          0
/tb_main_memory/dump                0
/tb_main_memory/wr_done             0
/tb_main_memory/state               read_mem2
/tb_main_memory/mdr                 UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
/tb_main_memory/uut/clk             0
/tb_main_memory/uut/address         4
/tb_main_memory/uut/we              0
/tb_main_memory/uut/wr_done         0
/tb_main_memory/uut/re              1
/tb_main_memory/uut/rd_ready        1
/tb_main_memory/uut/data            ZZZZZZZZZZZZZZZZZZZZZZZZ10010000
/tb_main_memory/uut/initialize      0
/tb_main_memory/uut/dump            0
Now                                 100 ns
Cursor 1                            60 ns
```

Figure 4

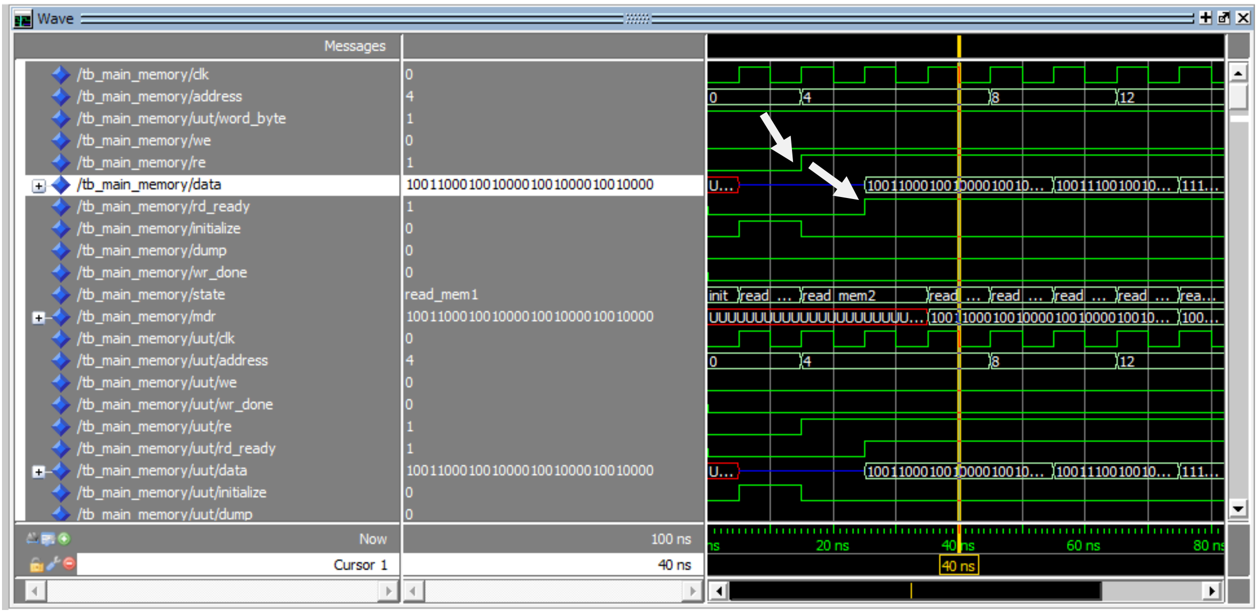
Reading from memory (single byte) read when generic variable of "Read_Delay" is set to three

1) Reading entries from memory in burst mode

a) Zero delay for memory read interaction

As it is depicted in Figure 5, first the “re” signal is activated while “we” is disabled, at the same time, an address is put on the address line (“address” signal). Consequently, data of the corresponding address is being appeared in the next rising edge of the clk and the “rd_ready” gets activated, showing that the data is ready to get sampled. Moreover, if we keep the “re” signal activated and change the address; we will see corresponding data values on the “data” signal every clock cycle, as shown in the Figure 5.

In this case, we look at the “rd_ready” and change the address every clock cycle, since it is always activate. In this case, only for the first read we would lose a clock cycle (we face two consequative reads from the same memory entry), however, for the next reads it happens in one clock cycle and we can latch the address in each rising edge.



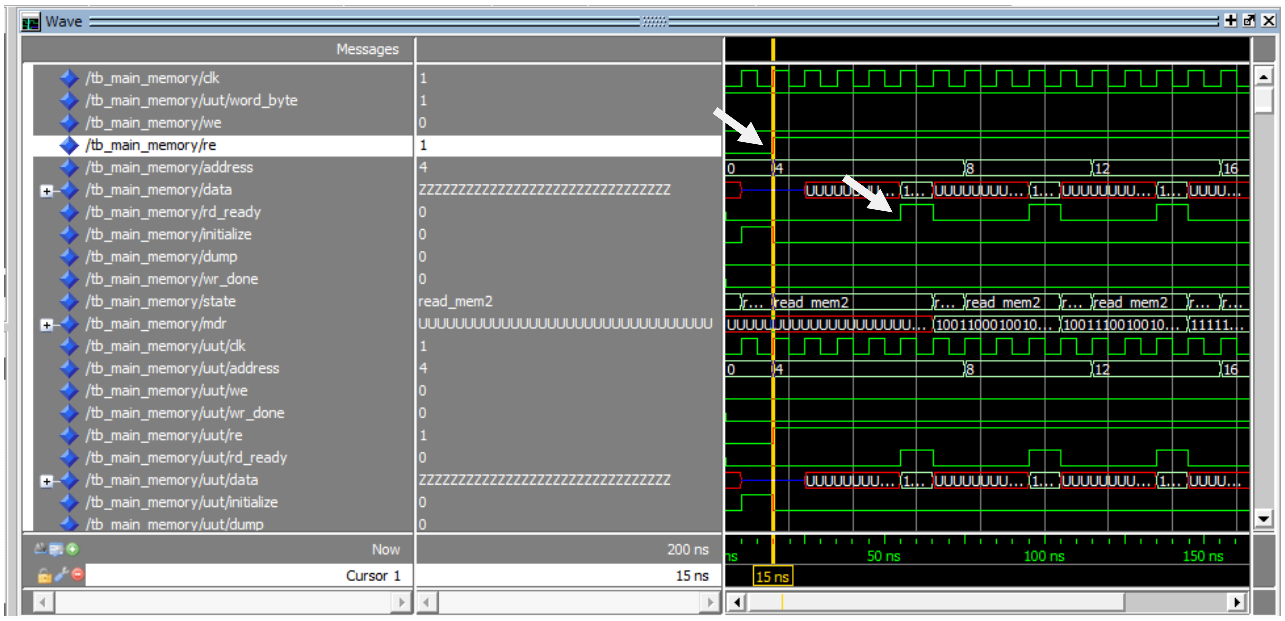
Messages	
/tb_main_memory/clk	0
/tb_main_memory/address	4
/tb_main_memory/uut/word_byte	1
/tb_main_memory/we	0
/tb_main_memory/re	1
/tb_main_memory/data	10011000100100001001000010010000
/tb_main_memory/rd_ready	1
/tb_main_memory/initialize	0
/tb_main_memory/dump	0
/tb_main_memory/wr_done	0
/tb_main_memory/state	read_mem1
/tb_main_memory/mdr	10011000100100001001000010010000
/tb_main_memory/uut/clk	0
/tb_main_memory/uut/address	4
/tb_main_memory/uut/we	0
/tb_main_memory/uut/wr_done	0
/tb_main_memory/uut/re	1
/tb_main_memory/uut/rd_ready	1
/tb_main_memory/uut/data	10011000100100001001000010010000
/tb_main_memory/uut/initialize	0
/tb_main_memory/uut/dump	0
Now	100 ns
Cursor 1	40 ns

Figure 5

Reading from the memory procedure (burst mode) and its corresponding signaling, when generic variable of “Read_Delay” is set to zero and “Word_Byte=1”

b) More than zero delay for memory read interaction

In this case, we have to activate “re” and change the address whenever we see “rd_ready” is asserted. Its timing diagram is depicted in Figure 6.



```
Messages
/tb_main_memory/clk                 1
/tb_main_memory/uut/word_byte       1
/tb_main_memory/we                  0
/tb_main_memory/re                  1
/tb_main_memory/address             4
/tb_main_memory/data                ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
/tb_main_memory/rd_ready            0
/tb_main_memory/initialize          0
/tb_main_memory/dump                0
/tb_main_memory/wr_done             0
/tb_main_memory/state               read_mem2
/tb_main_memory/mdr                 UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
/tb_main_memory/uut/clk             1
/tb_main_memory/uut/address         4
/tb_main_memory/uut/we              0
/tb_main_memory/uut/wr_done         0
/tb_main_memory/uut/re              1
/tb_main_memory/uut/rd_ready        0
/tb_main_memory/uut/data            ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ
/tb_main_memory/uut/initialize      0
/tb_main_memory/uut/dump            0
Now                                 200 ns
Cursor 1                            15 ns
```

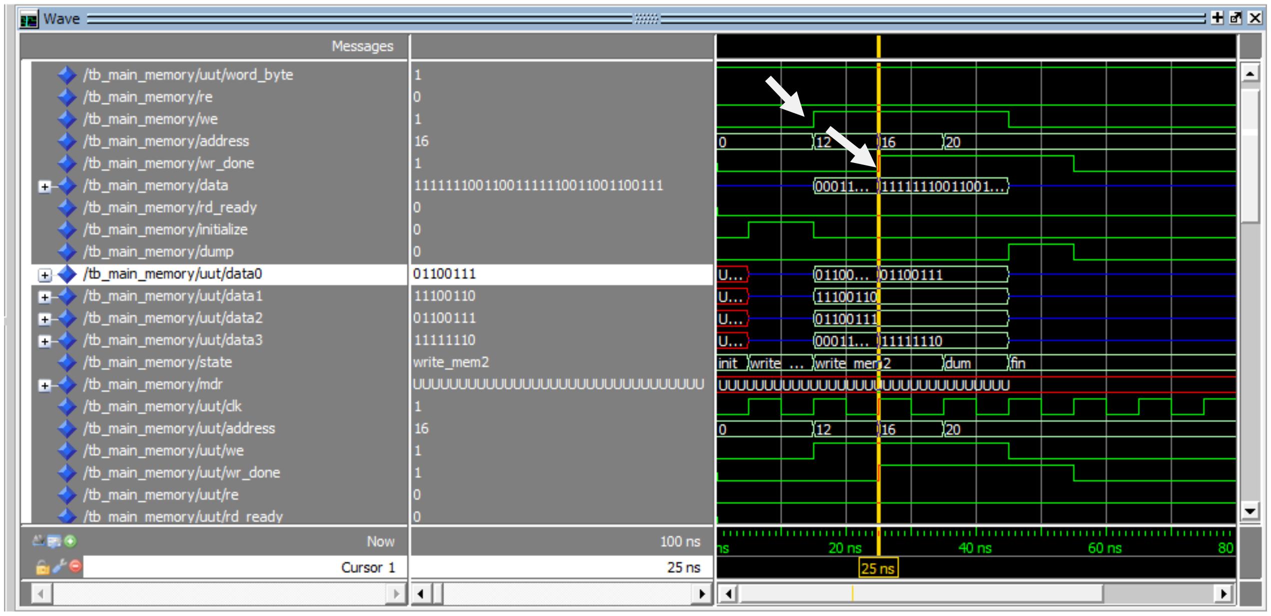
Figure 6

Reading from the memory procedure (burst mode) and its corresponding signaling, when generic variable of “Read_Delay” is set to three

3. Writing to a specific address of the memory

In this procedure first you have to define the address and activate the “we” signal simultaniously, then, in the next clks (respect to the # of write delays), the “data” signal vale will be written on the corresponding memory’s content and “wr_done” signal asserts to ‘1’ and lasts high for one clk cycle.

As it is depicted in Figure 7, first the “we” signal is activated while “re” is disabled, at the same time, an address is put on the address line (3). Consequently, the value of the data signal is written on the corresponding address in the consecutive clk and the “wr_done” activates, showing that the data has been written to the memory. Moreover, if we keep the “we” signal activated and change the address signal; we will see that corresponding data values on the “data” signal will be written on into the memory, sequentially.



```
/tb_main_memory/uut/word_byte   1
/tb_main_memory/re              0
/tb_main_memory/we              1
/tb_main_memory/address         16
/tb_main_memory/wr_done         1
/tb_main_memory/data            11111110011001111111001100110011 1
/tb_main_memory/rd_ready        0
/tb_main_memory/initialize      0
/tb_main_memory/dump            0
/tb_main_memory/uut/data0       01100111
/tb_main_memory/uut/data1       11100110
/tb_main_memory/uut/data2       01100111
/tb_main_memory/uut/data3       11111110
/tb_main_memory/state           write_mem2
/tb_main_memory/mdr             UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
/tb_main_memory/uut/clk         1
/tb_main_memory/uut/address     16
/tb_main_memory/uut/we          1
/tb_main_memory/uut/wr_done     1
/tb_main_memory/uut/re          0
/tb_main_memory/uut/rd_ready    0
Now                             100 ns
Cursor 1                        25 ns
```

Figure 7

Writing to a specific memory address with zero delay for write operation and in word size

4. Dumping memory into a file

This procedure is something like initializing and there is only needed to trigger dump signal (0→1). Then the memory content will be written to a file, which is indicted in “File_Address_Write”.

If you have any question, ask me

SHM