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.

By looking at the code you can find the type of each signal or variable. Use vhdl type-change-functions if your defined system's signals are different from the defined signals for this memory.

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	Size of memory in word	Determines the height of the memory	[1, 2^{32}-1]
Num_Bits_in_Word	Determines the number of bits in each word of memory	Determines the width of the memory	[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^{Mem_Size}-1]
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[1]	Data port	The data gets ready on, or reads from this port, respect to the kind of operation (read or write)	[0, $2^{Num\_Bits\_in\_Word}$-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

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.

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

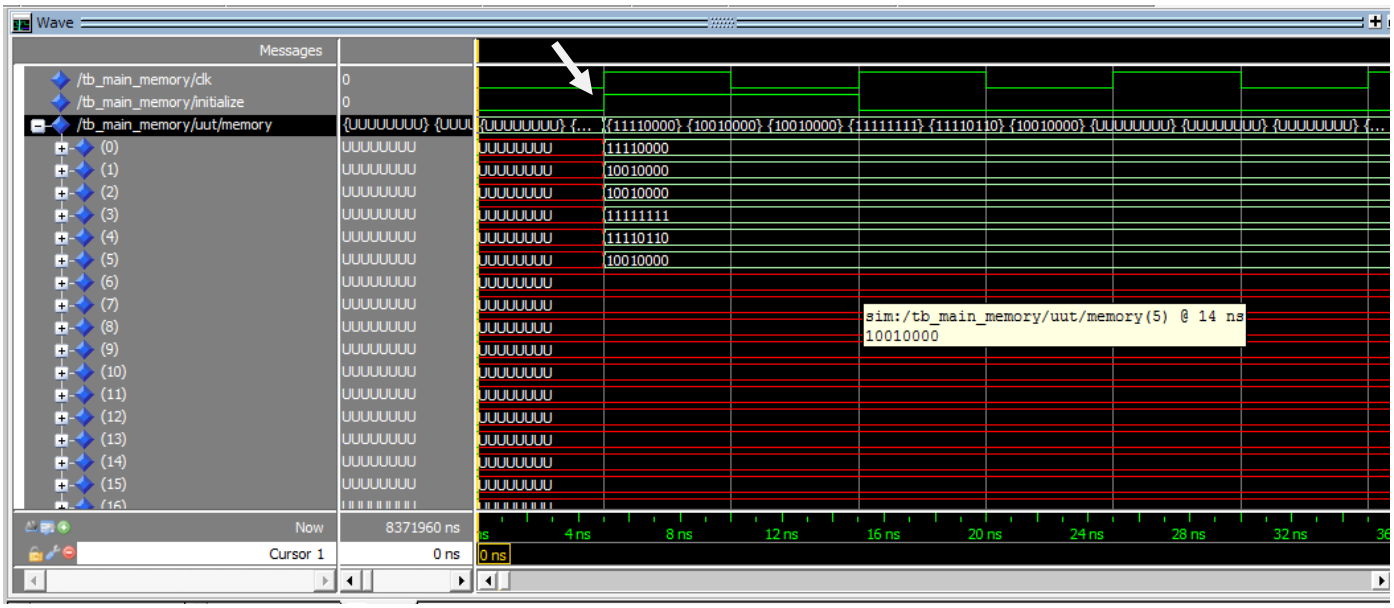

Figure 1

Initializing process and its corresponding signaling

2. Reading from specific addresses of the memory

In this procedure first you have to define the address and activate the “re” signal, then, in the next clks (due to defined # of read delays), the “data” signal gets the corresponding memory’s content and “rd_ready” signal gets to ‘1’ and lasts high for one clk. So, you need to save the data value when the “rd_ready” is high.

Reading timing	
Clk	Action
1st	Activate the “re”, put the correct Address “address” on the line (keep these values till you see rd_ready)
2nd	If rd_delay=0, the data will be read from the memory and “rd_ready” gets ‘1’
3rd	If rd_delay=1, the data will be read from the memory and “rd_ready” gets ‘1’
…	…

As it is depicted in Figure 2, 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 appeared in the consecutive clk and the “rd_ready” activates, showing that the data is ready to be sampled from user. 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 2.

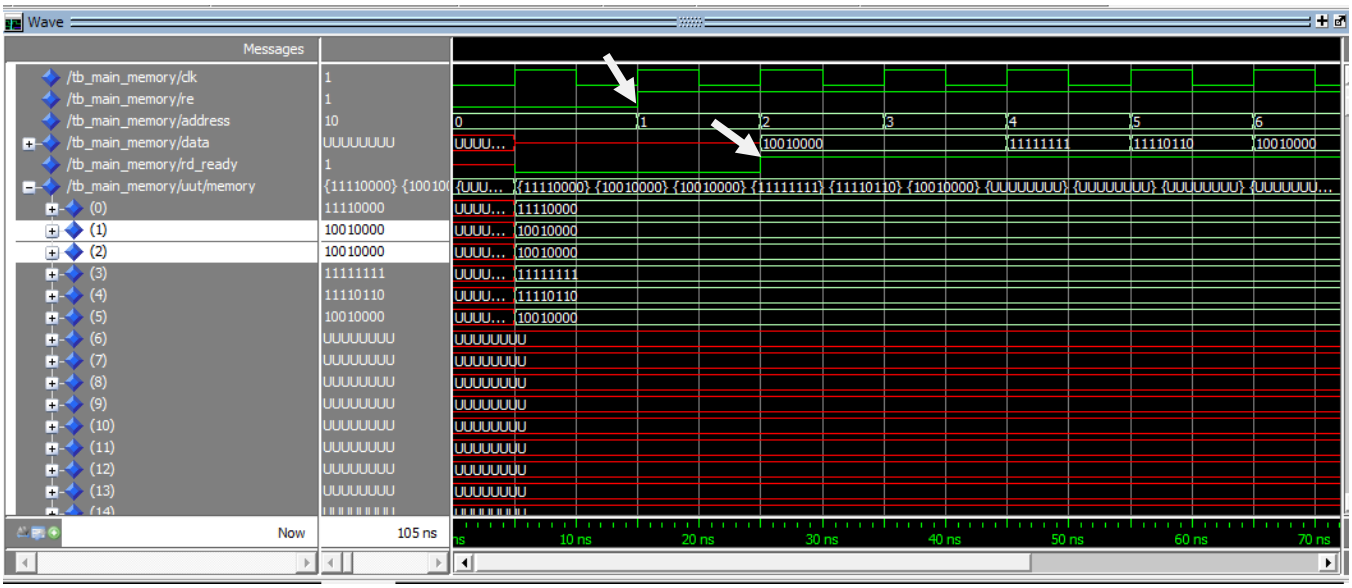

Figure 2

Reading from the memory procedure and its corresponding signaling

3. Writing to the memory

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

Writing timing	
Clk	Action
1st	Activate the “we”, put the correct Address and data on “address” and “data” signals
2nd	If wr_delay=0 the data will be written in the memory and “wr_done” is asserted to ‘1’
3rd	If wr_delay=1 the data will be written in the memory and “wr_done” is asserted to ‘1’
…	…

As it is depicted in Figure 3, first the “we” signal is activated while “re” is disabled, at the same time, an address is put on the address line (in this case, address=3). Consequently, the value of the data signal is written on the corresponding address in the consecutive clk and the “wr_done” asserts to ‘1’, showing that the data has been written on 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 into the corresponding memory addresses (Figure 4).

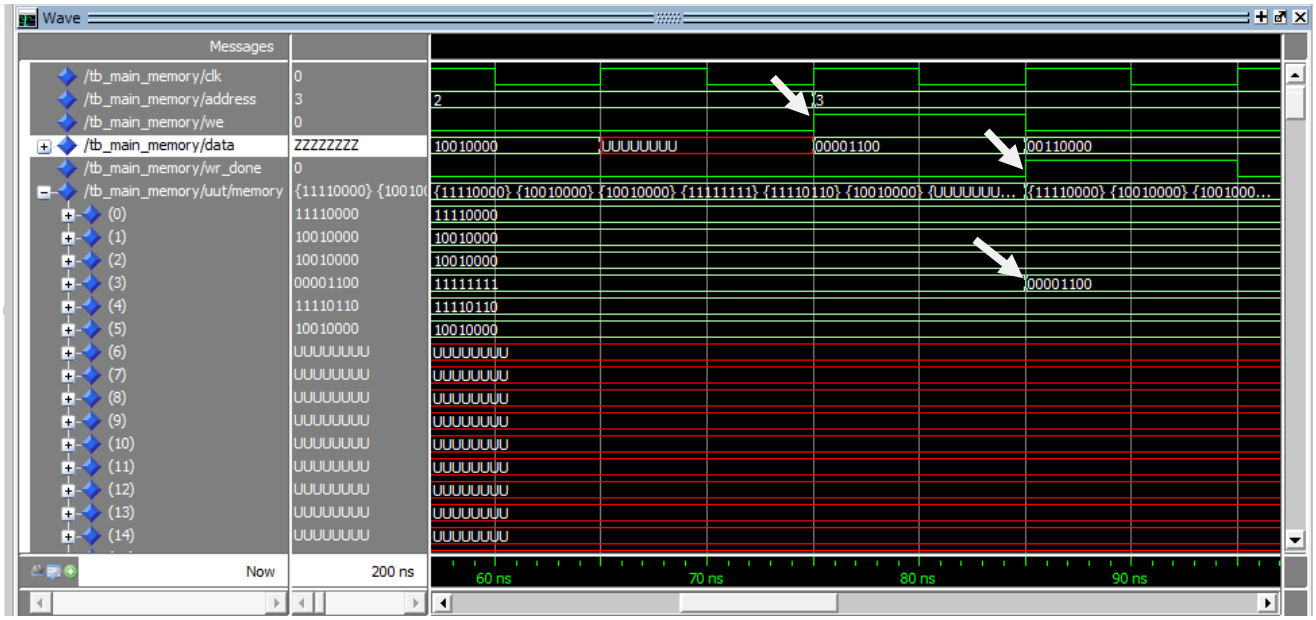

Figure 3

Writing to a specific memory address

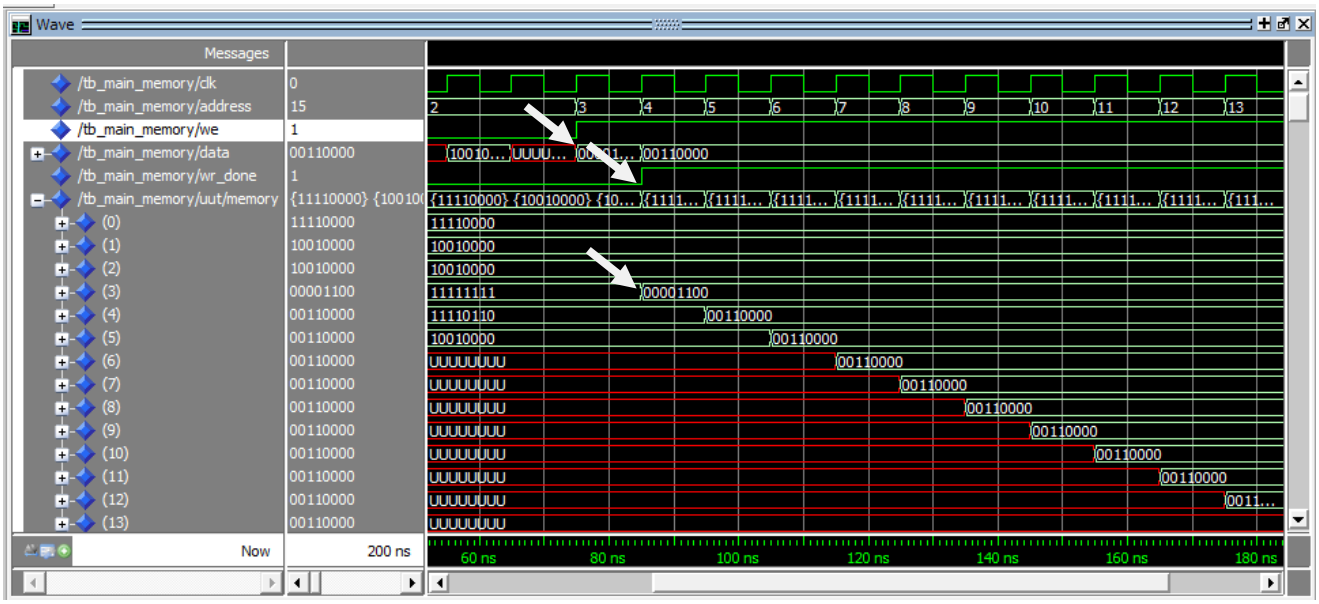

Figure 4

Writing to a series of addresses sequentially

4. Dumping memory content into a file

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

If you have any question ask me

SHM