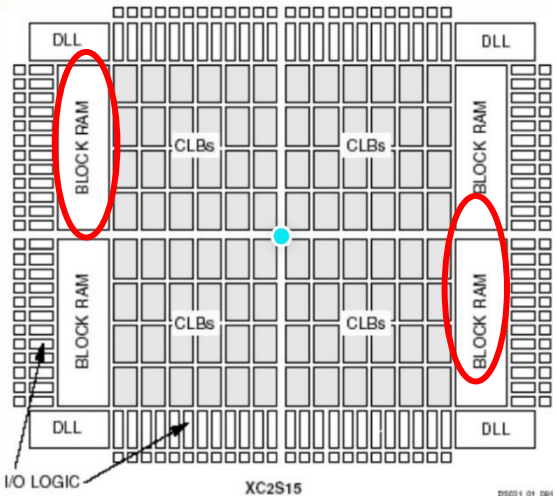
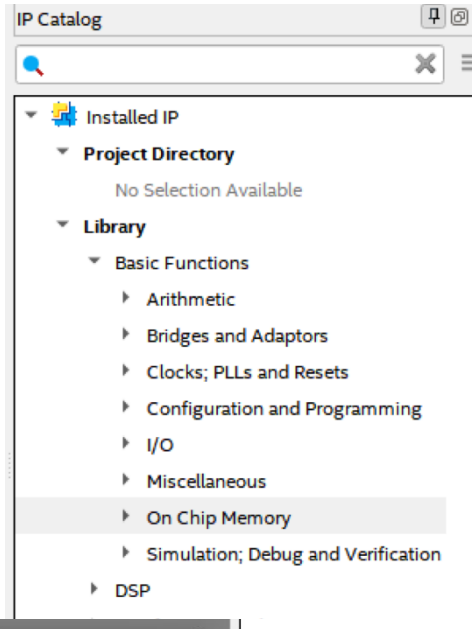
Tutorial: Using the Block RAM on the chip

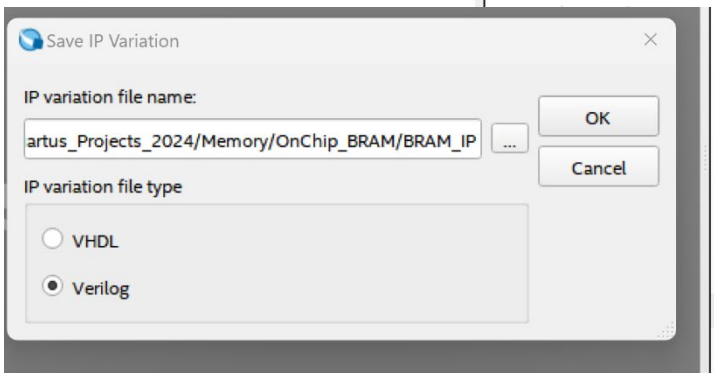


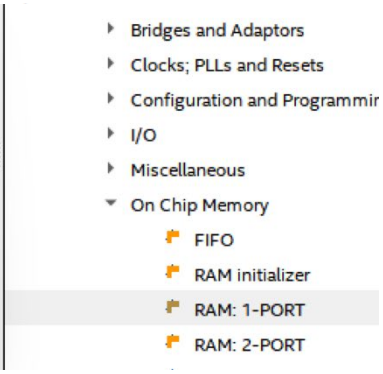
Some FPGAs have internal on-chip hardwired (ASIC) **Block RAMs**. These IP cores can be accessed by creating a wrapper file around them to connect to their ports.

One of the easy methods to create the wrapper files around the on chip BRAMs is by using the Memory design wizards.

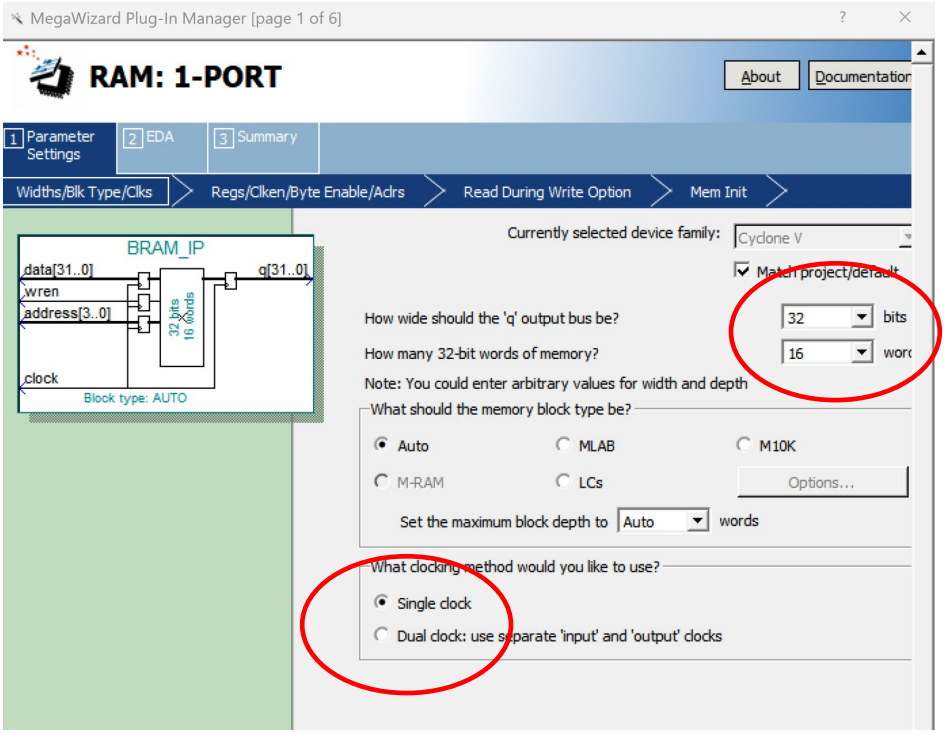
1. Create a project file and use DE1_SoC as your board. This board has an FPGA with 64 BRAMs. For this example, the project name was selected to be *BRAM_IP*.
2. From the **IP catalog** window select the **On Chip Memory** under the **Basic Functions Library.**
3. Under the On Chip Memory, select **RAM: 1-Port.**
4. The popup window will inquire about the Verilog name you want to chose for the wrapper file and the language you want. Within the same project folder, select the wrapper file name. If the BRAM is your top module, assign it the same name as the project. In this example, it was called *BRAM_IP*. The language used is Verilog as shown.







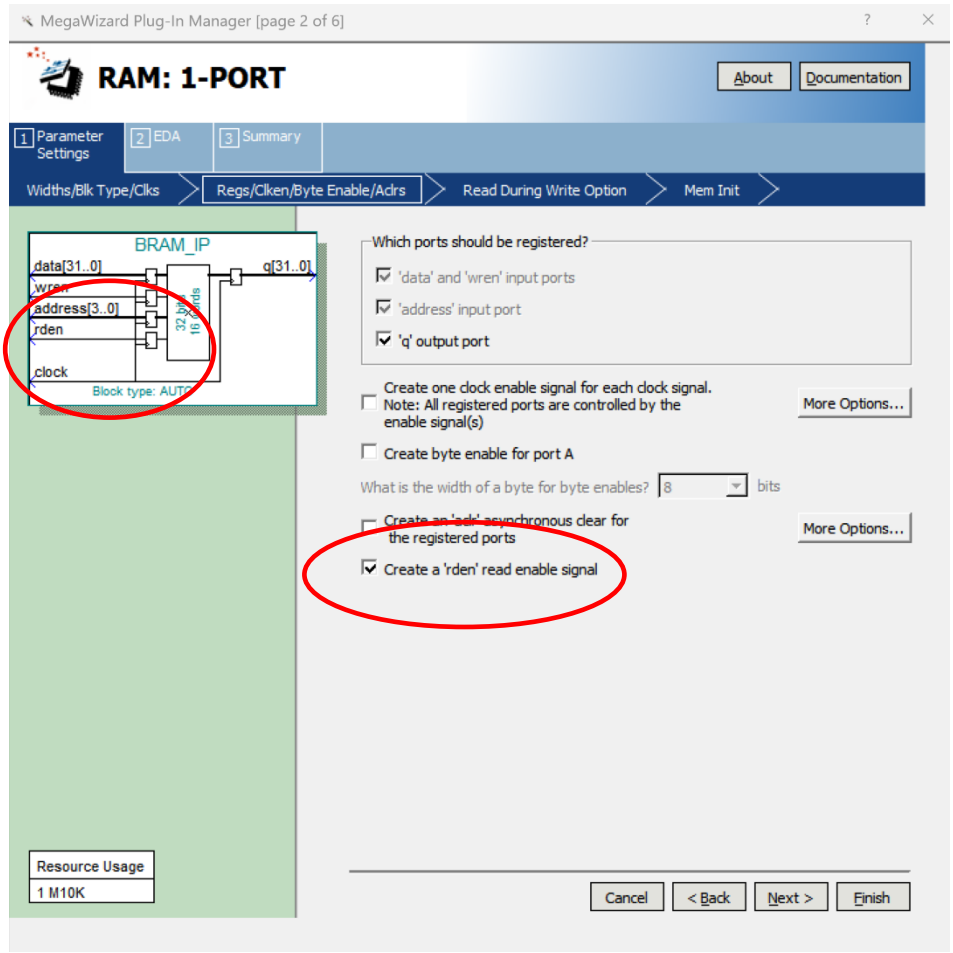
5. Now the wizard will start to guide you through the selection of the RAM parameters:



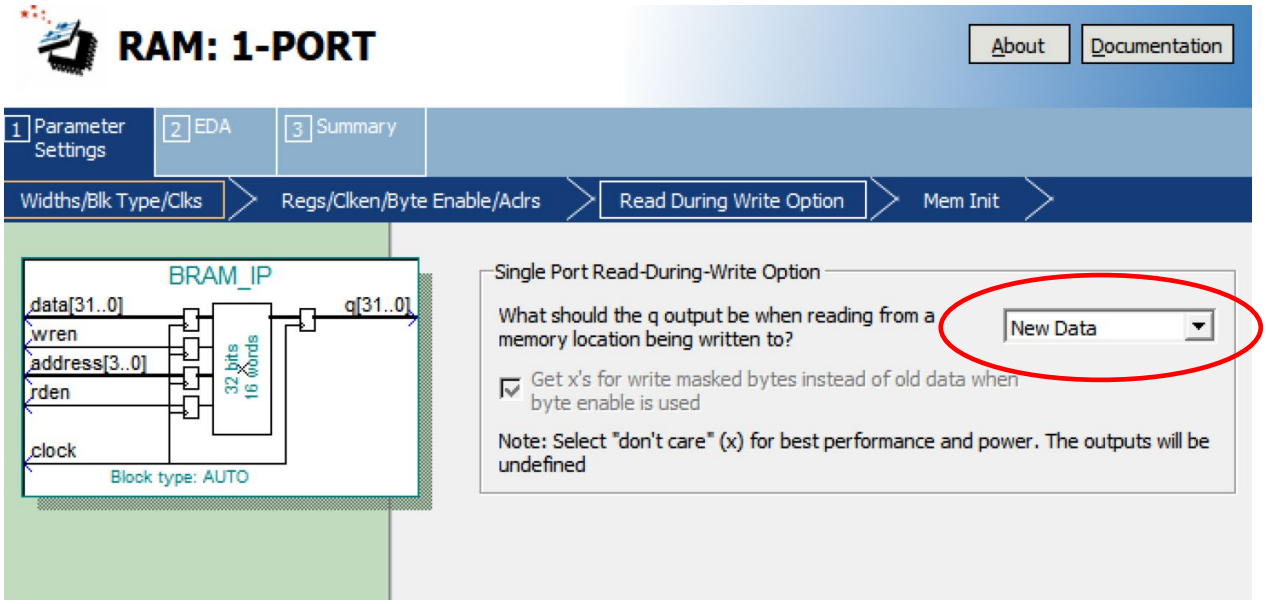
Selected are 16 words depth and 32bit word width. A single clock was selected for the reading from and writing to the RAM. There is an option of having two separate clocks for the two different operations. Then hit **Next**.

6. For this step, you will get options for enable signals. If you check the box for the **rden** (read_enable) a corresponsing port will be added to the RAM block diagram.

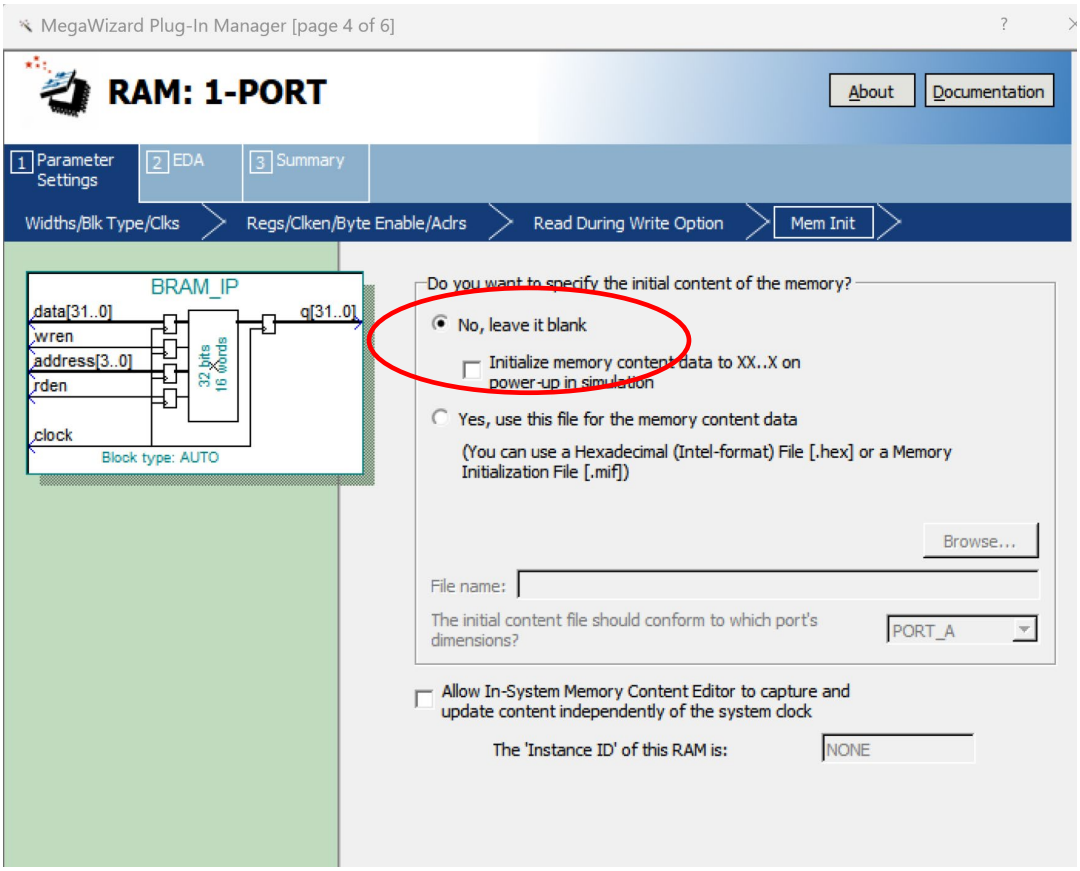
   After you make your selections for this window click on Next.



7. Make no changes to the next window.



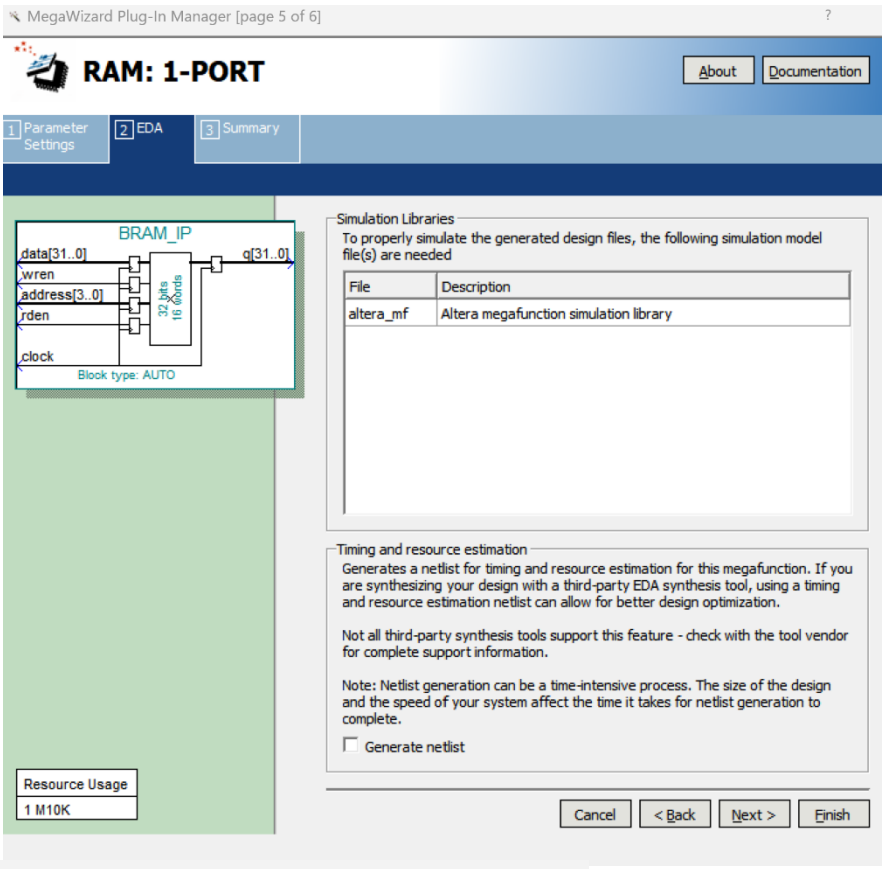
8. Here the window is asking about the initial values saved to the RAM. You have an option of setting all the initial values in the memory to **zero (blank)**, or to **unknown x**, or you can use an initialization file with the required values. The initialization fille may be a **HEX file** as the one we created for the ROM, and testbench examples. For this example, we will just set it to zero.

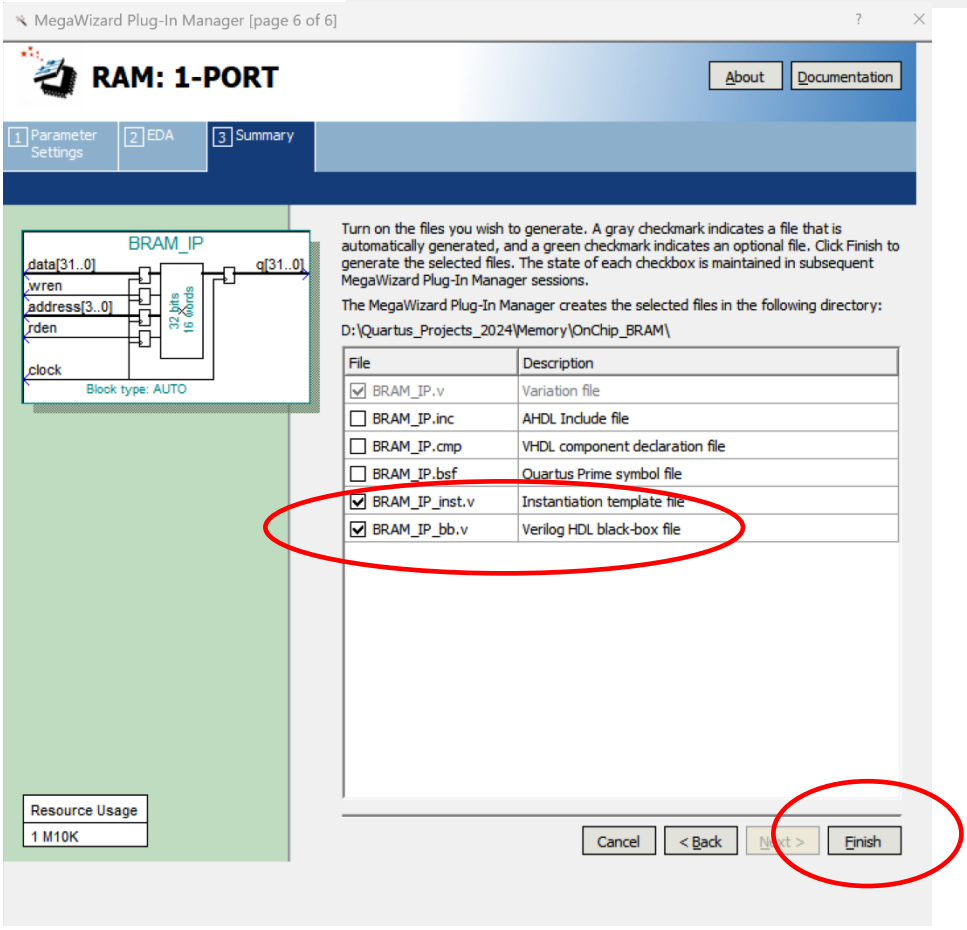


9. Click on Next.

10. For the final step, the wizard will ask about the files you would like to generate. By default we have the BlackBox bb file (the wrapper file assigned around the hardwired logic to access the BRAM.)

11. Select the instantiation template so that you can directly use it for the testbench or in the top module depending on your design. Then click on **finish**.

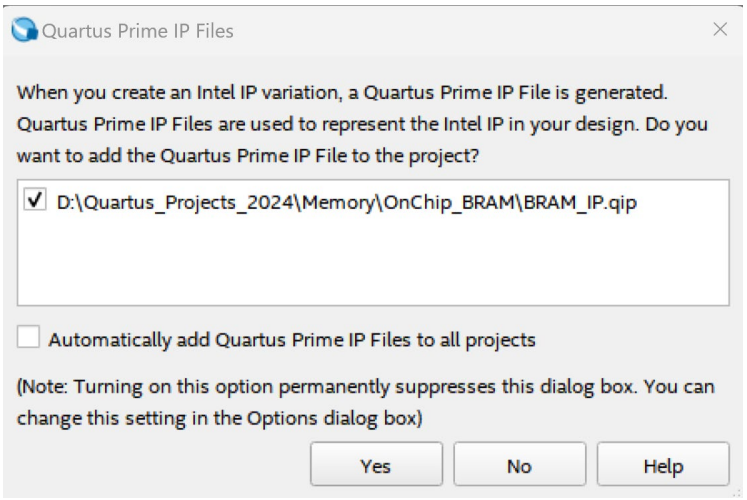


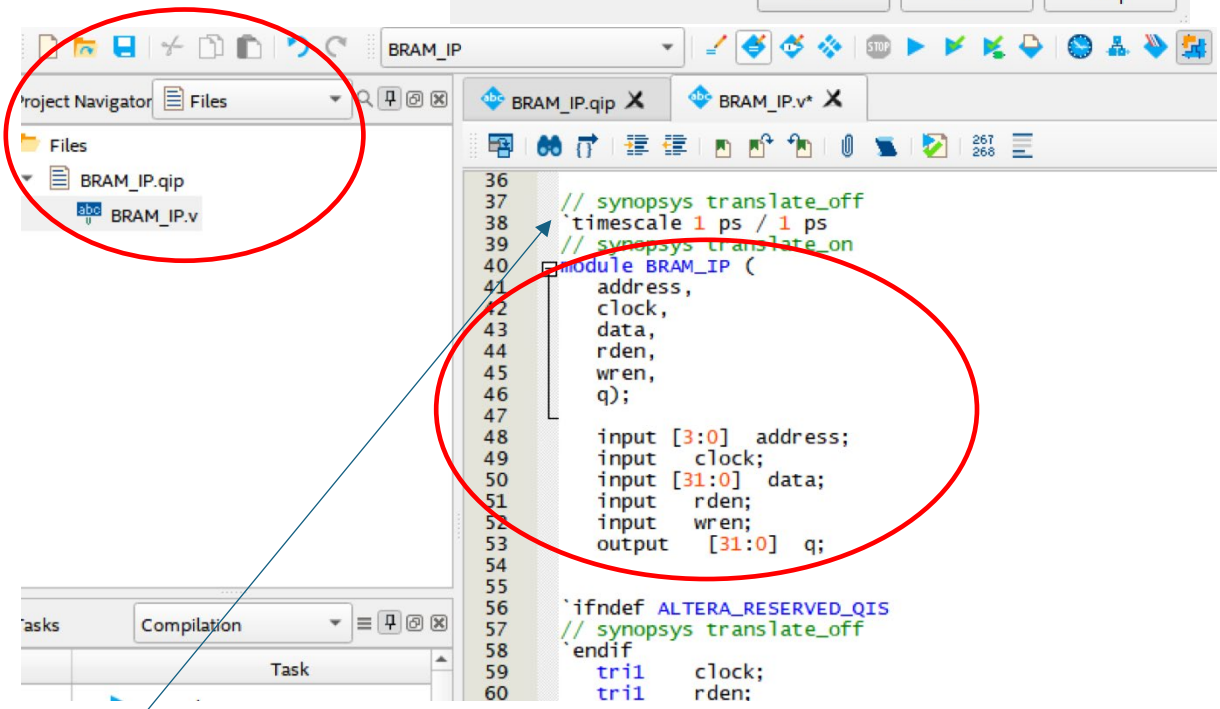


12. Click on Yes

13. The wrapper file will be generated and can be opened as follows. Open the files, and see the files under BRAM_IP.qip.

14. Double click on BRAM_IP.v



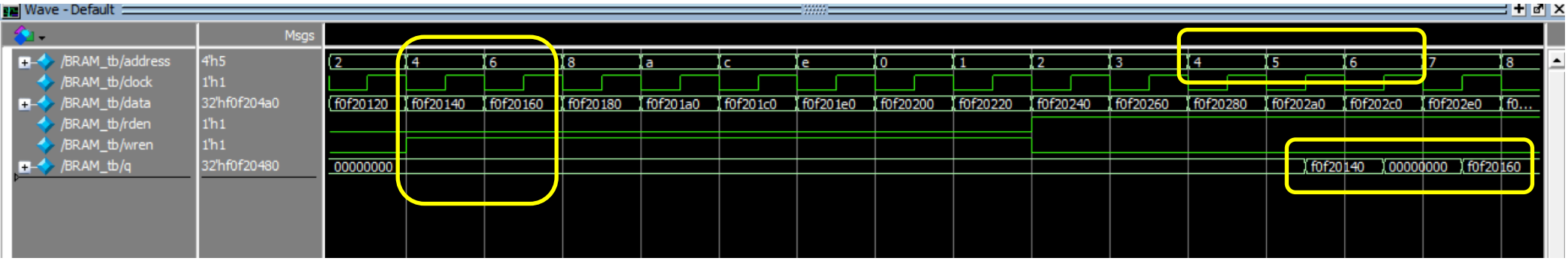


Notice that the Verilog file generated by Quartus has **time unit/time precision** information. This sets the time unit (in this example unit = 1ps). Any delay then # is represented in ps. #5 = 5ps. As long as one file in the project has this timing information, all the other modules must have this information.

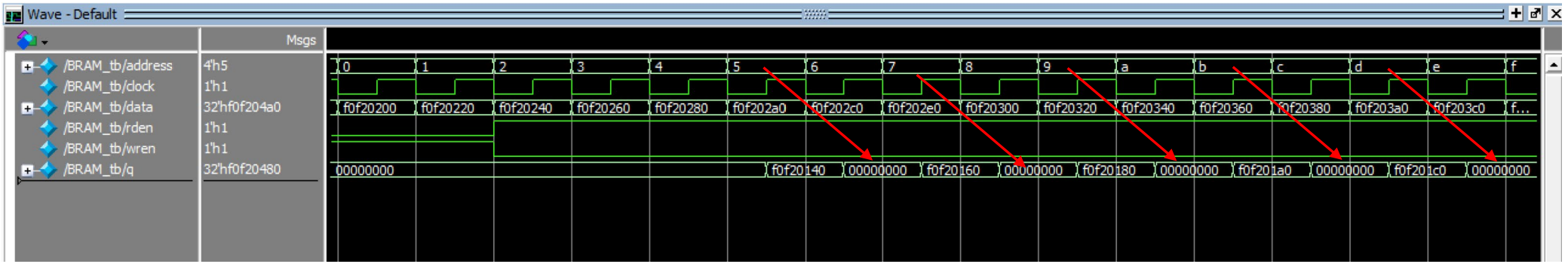
`timescale 1ps / 1ps

Testbench

```
`timescale 1 ps / 1 ps   //time unit/time precision specifies to match the RAM file generated
module BRAM_tb ();

 reg   [ 3:0]   address;
 reg            clock;
 reg   [31:0]   data;
 reg            rden;
 reg            wren;
 wire  [31:0]   q;

 BRAM_IP  UUT ( address, clock, data, rden, wren, q);

 initial
       begin              clock = 0;
       forever  begin  #5 clock = ~clock; end
       end

 initial
       begin
       address = 4'b0000;       #10;
       repeat(8)  begin
       address = address + 2;   #10;  end
       repeat(30) begin
       address = address + 1;   #10; end
       repeat(30) begin
       address = address + 5;   #10; end
       end

 initial
      begin
      data = 32'hF0F2_0100;
      forever
      #10 data = data + 32;
      end


 initial
     begin
        {wren, rden} = 2'b00; # 20;     // no read and no write
        {wren, rden} = 2'b10; # 80;     // write
        {wren, rden} = 2'b01; #160;     // read
        {wren, rden} = 2'b11; # 80;     // read and write at the same time
        end

 initial
 #300 $stop;

 endmodule
```

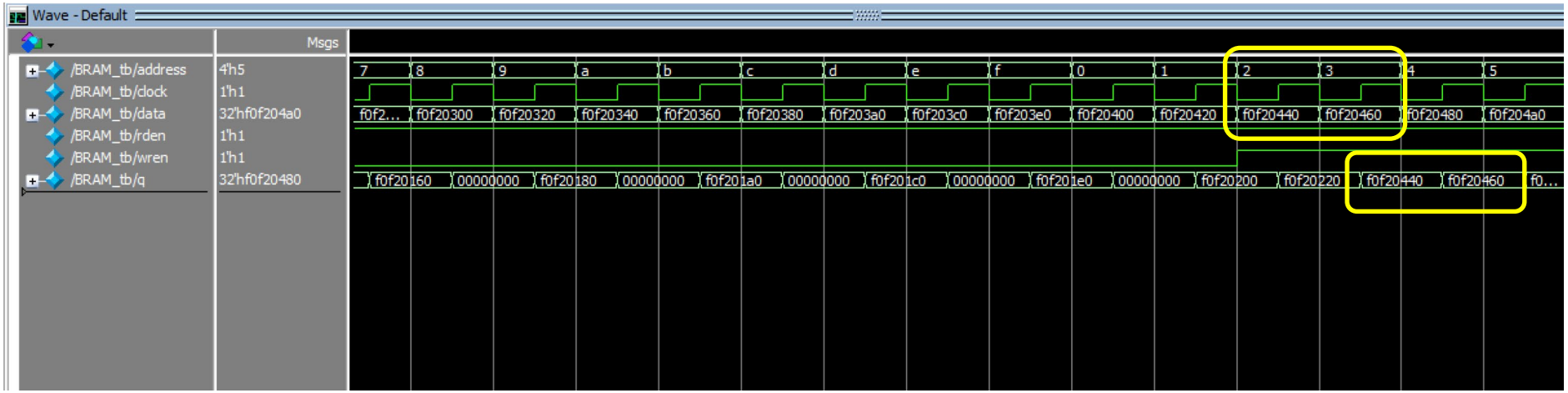


No new data was written in address 5, hence we see that the content is still 0.
The values written in address 4 and 6, appear when reading the content. There is a one clock delay showing in the timing diagram
During the writing process, if the rden is 0, the q outputs a 0.



All the addresses that were skipped in the writing process for this testbench show 0 content, as selected in the memory wizard. The memory was initialized with blank values. (shown using the red arrows)

Comparing the top and second figure, we see that the values saved at the even addresses are showing in the reading operation with one clock cycle delay. For the addresses selected without enabling the wren stayed blank and were not assigned values from the data_in.



When the rden and wren are both active, the output will show the new data. In the 2nd figure address 3 had blank content (00000000), but in the 3rd figure, we see that the new value is showing at the output as selected in the wizard.