Lab 4 – Memory
Spring 2019
EE457

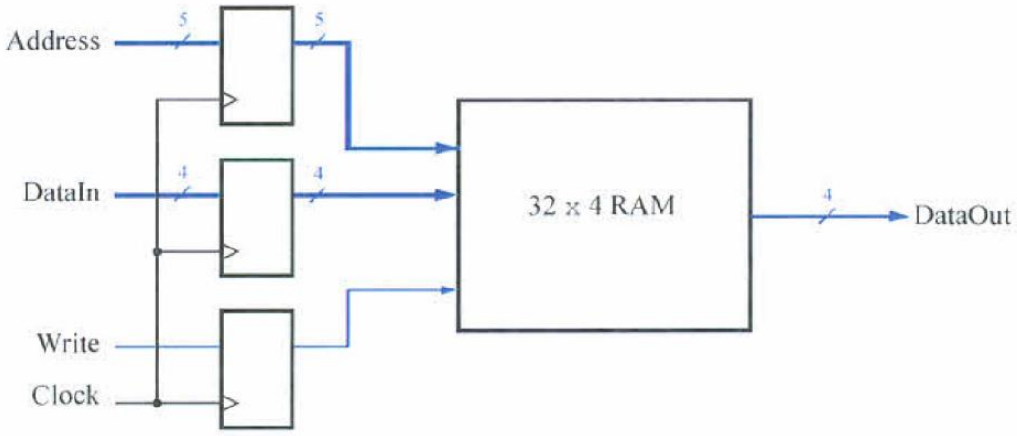
Memory Blocks

In computer systems it is necessary to provide a substantial amount of memory. If a system is implemented using FPGA technology it is possible to provide some amount of memory by using the memory resources that exist in the FPGA device.

A diagram of the random access memory (RAM) module that we will implement is shown in Figure 1a. It contains 32 four-bit words (rows), which are accessed using a five-bit address port, a four-bit data port, and a write control input.

The FPGAs that are included on the DE1-SoC board provide dedicated memory resources. The DE1-SoC boards have M10K memory blocks for memory resources. Each M10K block contains 10240 memory bits. M10k blocks can be configured to implement memories of various sizes. A common term used to specify the size of a memory is its aspect ratio, which gives the depth in words and the width in bits (depth x width). In this exercise we will use an aspect ratio that is four bits wide, and we will use only the first 32 words in the memory. Although the M10K blocks support many other modes of operation, we will not discuss them here.

(a) RAM organization



(b) RAM implementation

Figure 1: A 32 x 4 RAM module.

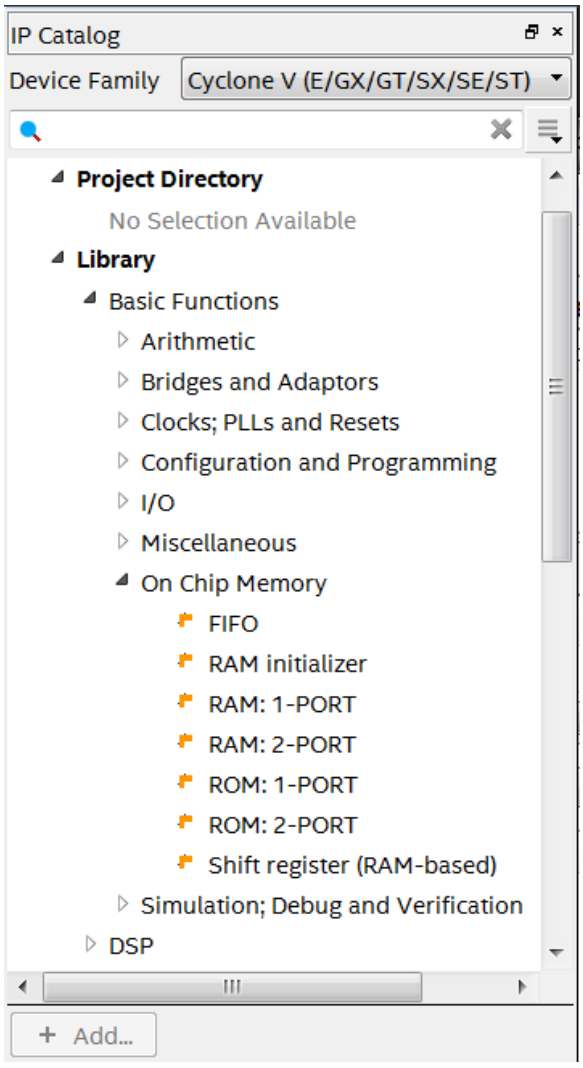
There are two important features of the M9K and M10K blocks that have to be mentioned. First, they include registers that can be used to synchronize all the input and output signals to a clock input. The registers on the input ports must always be used, and the registers on the output ports are optional. Second, the blocks have separate ports for data being written to the memory and data being read from the memory. Given these requirements, we will implement the modified 32 x 4 RAM module shown in Figure 1b. It includes registers for the address, data input, and write ports, and uses a separate unregistered data output port.

Part I

Commonly used logic structures, such as adders, registers, counters and memories, can be implemented in an FPGA chip by using prebuilt modules that are provided in libraries. In this exercise we will use such a module to implement the memory shown in Figure 1b.

1. Use the zip file on D2L – as this has all the correct pinouts needed.

2. To open the IP Catalog in the Quartus software click on Tools > IP Catalog. In the IP Catalog window choose the RAM: 1-PORT module, which is found under the Basic Functions > On Chip Memory category. Select VHDL as the type of output file to create, give the file the name ram32x4.vhd, and click OK. As shown in Figure 2 specify a memory size of 32 four-bit words.

3. 

4. Select M10K memory if you are using the Cyclone V, which we are. Also on this screen accept the default setting to use a single clock for the memory's registers, and then advance to the page shown in Figure 3. On this page deselect the setting called 'q' output port under the category Which ports should be registered? This setting creates a RAM module that matches the structure in Figure 1b, with registered input ports and unregistered output ports. Accept defaults for the rest of the settings in the Wizard, and click the Finish button to exit from this tool. Examine the ram32x4.vhd VHDL file which defines the following subcircuit:

```
ENTITY ram32x4 IS
        PORT ( address : IN STD_LOGIC_VECTOR (4 DOWNTO 0);
                clock : IN STD_LOGIC := '1';
                data : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
                wren : IN STD_LOGIC ;
                q : OUT STD_LOGIC_VECTOR (3 DOWNTO 0) );
        END ram32x4;
```

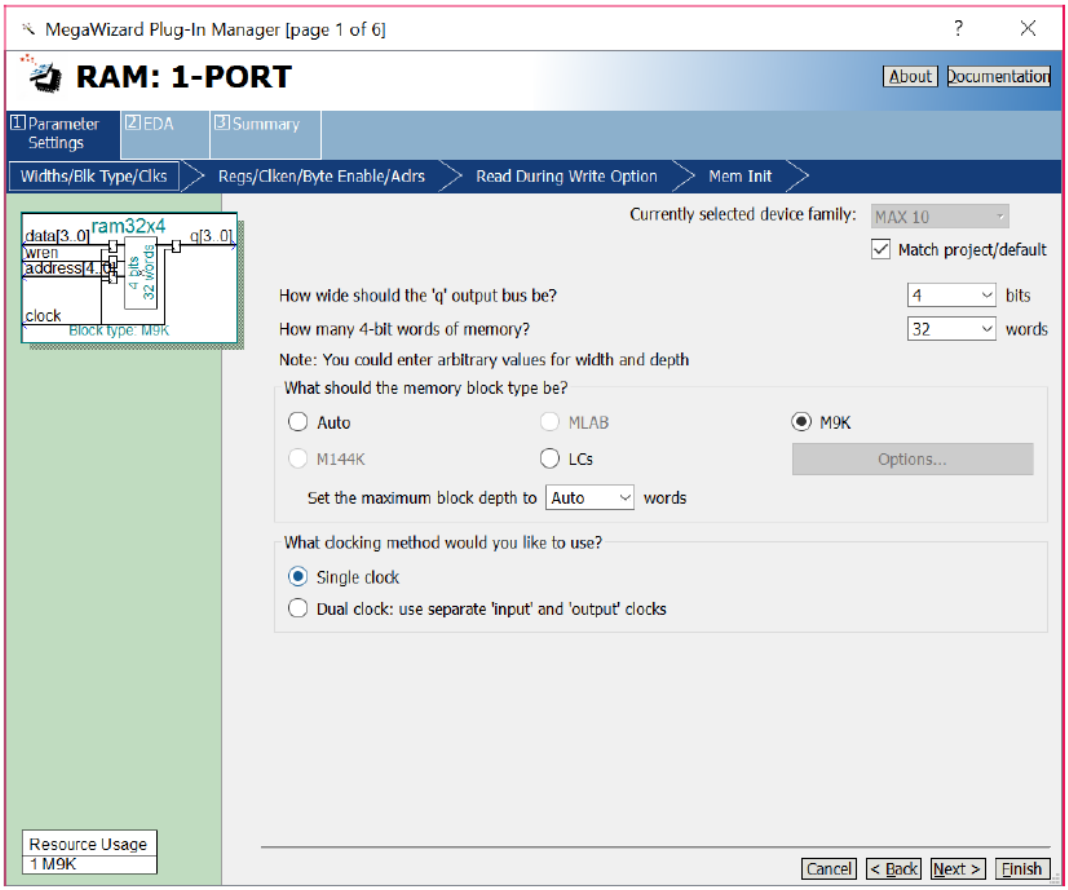


Figure 2 Configuring the size of the memory module.

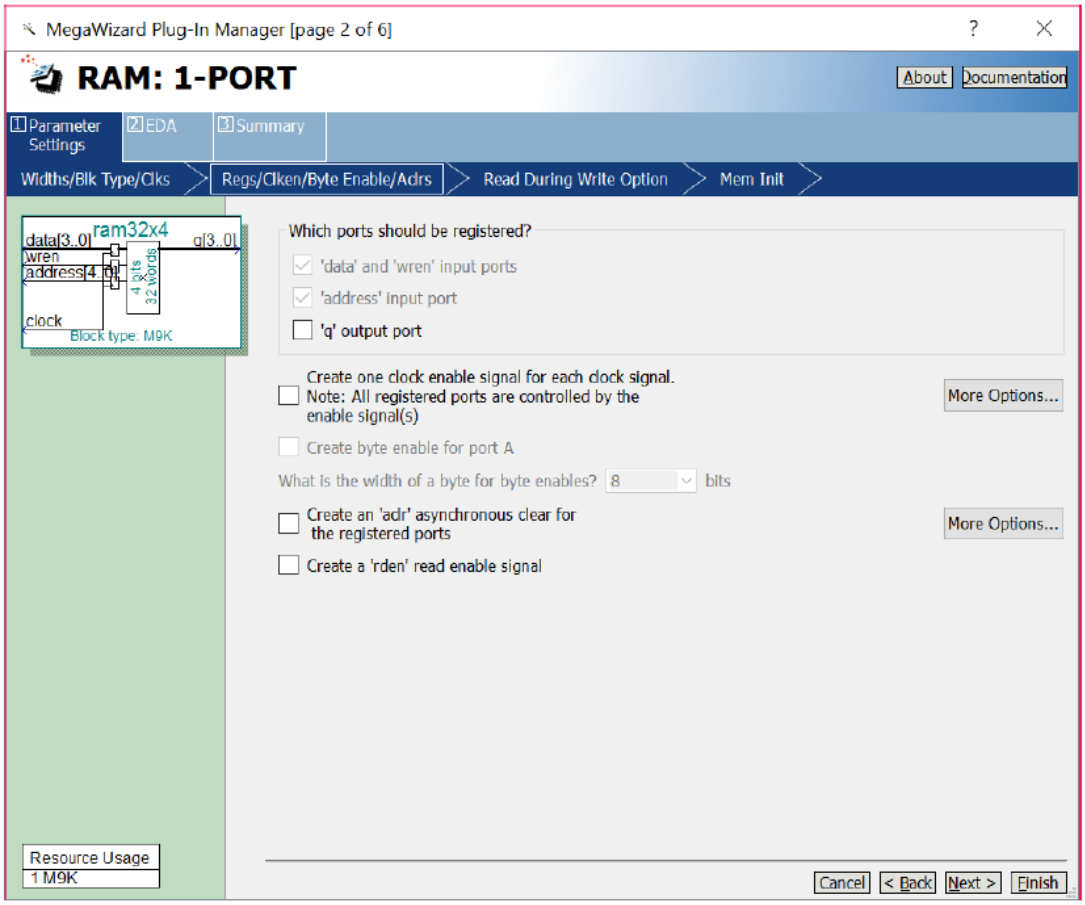


Figure 3 Configuring the Input and Outputs

Instantiate this sub circuit in the DE1_TOP.vhd file that includes the inputs and output signals for the memory ports given the figure. – move onto part II

Part II

Now, we want to realize the memory circuit in the FPGA on your DE1_SOC board, and use slide switches to load some data into the created memory. We also want to display the contents of the RAM on the 7-segment displays.

1. Using the Quartus project from Part I which will be used to implement the desired circuit on your DE-series board.
2. Use slide switches SW_{3-0} to provide input data for the RAM, and use switches SW_{8-4} to specify the address. Use SW_9 as the Write signal and use KEY_0 as the Clock input. Show the address value on the 7-segment displays HEX5 – 4 (use the upper address bit for HEX5, and the lower 4 bits for HEX4 and show the data being input to the memory on HEX2, and show the data read out of the memory on HEX0.
3. Test your circuit via simulation and the board and make sure that data can be stored into the memory at various locations.
4. Simulate your design with Modelsim using the de1_top_vhdl_tb.do file (this file expects the ram32x4.vhd file to be located at the root directory). – grab screen shots to show the simulation
    a. Simulate the following with one testbench
        i. reading from address decimal 3 / Hex “03”
        ii. reading from address decimal 14 / Hex “0E”
        iii. reading from address decimal 26 / Hex “1A”
        iv. reading from address decimal 27 / Hex “1B” then

    a) write “1100” to address decimal 3 / Hex “03”
    b) write “0011” top address decimal 14 / Hex “0E”
    c) write “1111” to address decimal 26 / Hex “1A”
    d) write “0101” to address decimal 27 / Hex “1B” then
        i. read from address decimal 3 / Hex “03”
        ii. read from address decimal 14 / Hex “0E”
        iii. read from address decimal 26 / Hex “1A”
        iv. read from address decimal 27 / Hex “1B”

5. Include simulation waveforms in your lab report. – describe in your report what this circuit is doing.

Zip up Part 2 and submit to D2L. – you can combine the lab report from part II and Part II together

Part III

The SRAM block in Figure 1 has a single port that provides the address for both read and write operations. For this part you will create a different type of memory module, in which there is one port for supplying the address for a read operation, and a separate port that gives the address for a write operation. Perform the following steps.
Note – in PartII – key(0) was acting as the clock input, so that you can set the switches and write data into the RAM – I this lab clock_50 is the clock and key(0) is reset signal.

1. Use the Quartus project from PartI/II as the starting point for Part III (make a copy of the directory, make sure Quartus and Modelsim are closed). To generate the desired memory module open the IP Catalog and select the RAM: 2-PORT module in the Basic Functions -> On Chip Memory category. As shown in Figure 4, choose with one read port and one write port in the category called How will you be using the dual port RAM? (call the file ram32x4.vhd, just like in Part I/II) you will be overwriting the ram32x4.vhd file from part I/II)

Configure the memory size, clocking method, and registered ports the same way as Part II. As shown in Figure 6 select I do not care (The outputs will be undefined) for Mixed Port Read-During-Write for Single Input Clock RAM. This setting specifies that it does not matter whether the memory outputs the new data being written, or the old data previously stored, in the case that the write and read addresses are the same during a write operation.

Figure 7 shows how the memory words can be initialized to specific values. It makes use of a feature that allows the memory module to be loaded with data when the circuit is programmed into the FPGA chip. As shown in the figure, choose the setting Yes, use this file for the memory content data, and specify the filename ram32x4.mif. An example of a MIF file is provided below Figure 7. You can also learn about the format of a memory initialization file (MIF) by using the Quartus Help. Finish the Wizard and then examine the generated memory module in the file ram32x4.vhd. (Mif file is provided ram32x4.mif).

2. Use a counter (gen_counter.vhd) as the read address so you can address the memory locations by displaying each word from the memory for one second, the read address is the output of the counter, and the address changes every second. As each word is being displayed, show its address (in hex format) on the 7-segment displays HEX3-2 (use the upper address bit for Hex3, and the lower 4 address bits for Hex 2). Use the supplied gen_counter.vhd for the counter and you will need two counters, one that pulses every second and one that will count to decimal 31/Hex 1F (Five bits) and then roll back to 00.

3.Use the 50 MHz clock, CLOCK_50 for the clocks, and use KEY_0 as a reset input. For the write address and corresponding data use switches SW_{8-4} and SW_{3-0}. Show the write address on HEX5-4 (use the upper address bit for Hex5, and the lower 4 address bits for Hex4) and show the write data on HEX1. Make sure that you properly synchronize the slide switch inputs to the 50 MHz clock, see the cascaded registers below figure 7 for the proper synchronization structure – remember metastability hardening?

4. Test your circuit and verify that the initial contents of the memory match your ram32x4.mif file. Make sure that you can independently write data to any address by using the slide switches.

5. Simulate your design using modelsim – you need to have this in your simulation (put screen shots in your lab report)

1) let the design run in read mode so that it just reads out all the contents from the memory decimal address 0->31 / Hex "00 ->"1F"

2) then simulate writing the following values

a) of "1100" to address decimal 3 / Hex "03"

b) of "0011" top address decimal 14 / Hex "0E"

c) of "1111" to address decimal 26 / Hex "1A"

d) of "0101" to address decimal 27 / Hex "1B"

3) then repeat step 1 above – read out the contents of the RAM, you should see your write data show up from reading the RAM

6. Zip up the directory and submit to - describe in your report what this circuit is doing – you can submit 1 lab report for PartII/PartIII

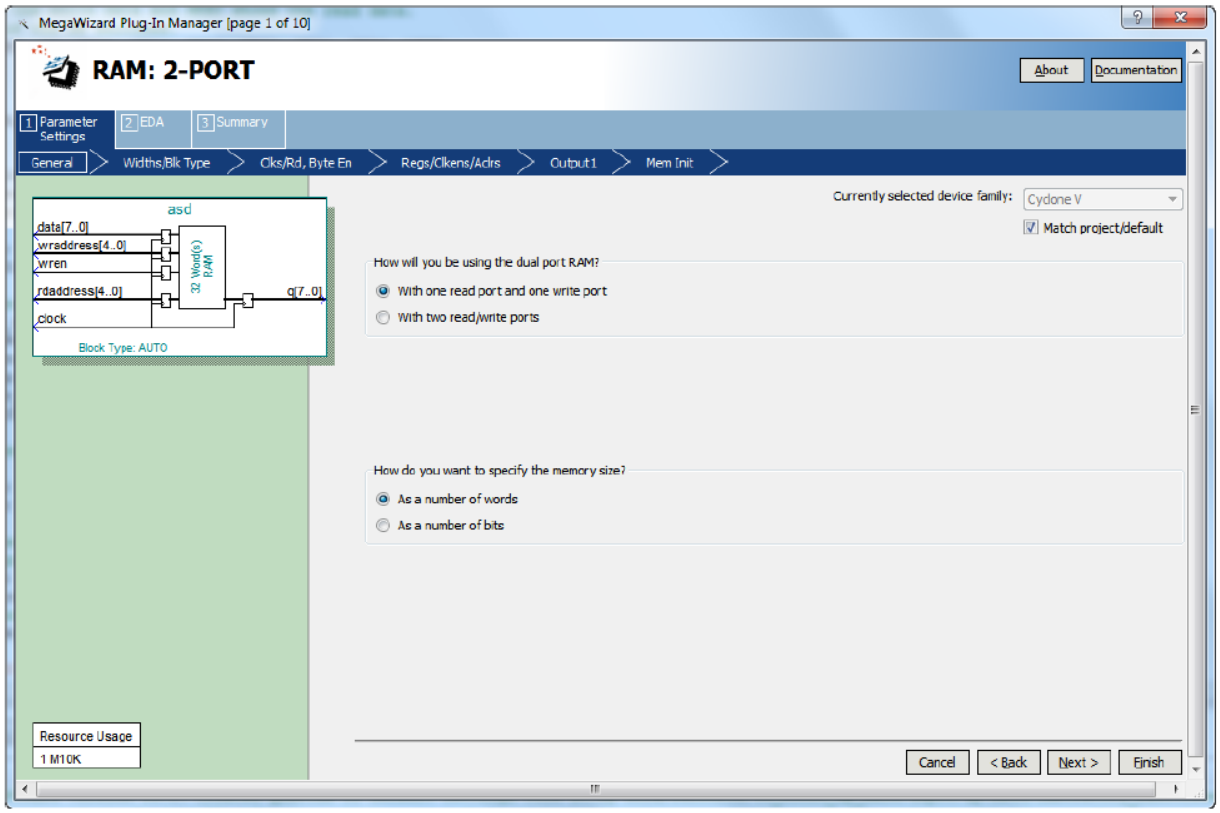


Figure 5: Configuring the two input ports of the RAM.

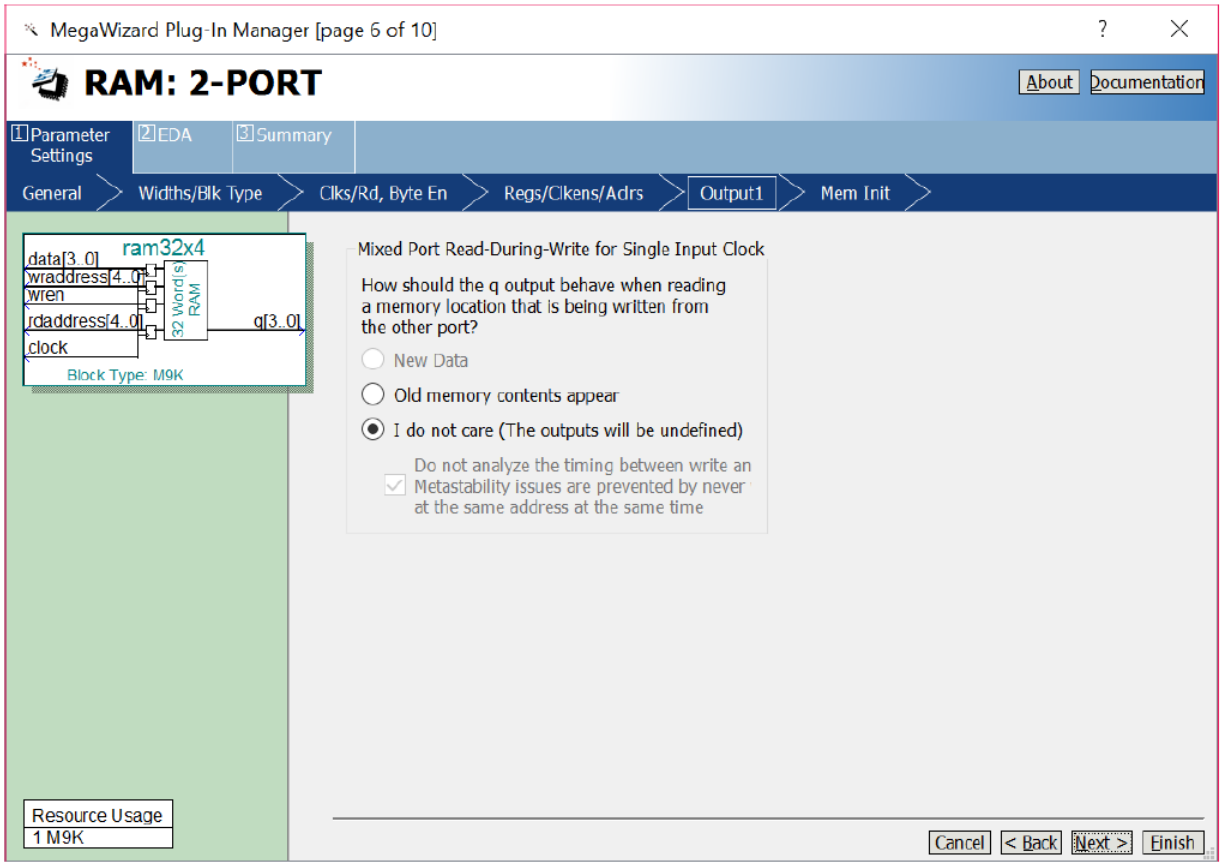


Figure 6: Configuring the output of the RAM when reading and writing to the same address.

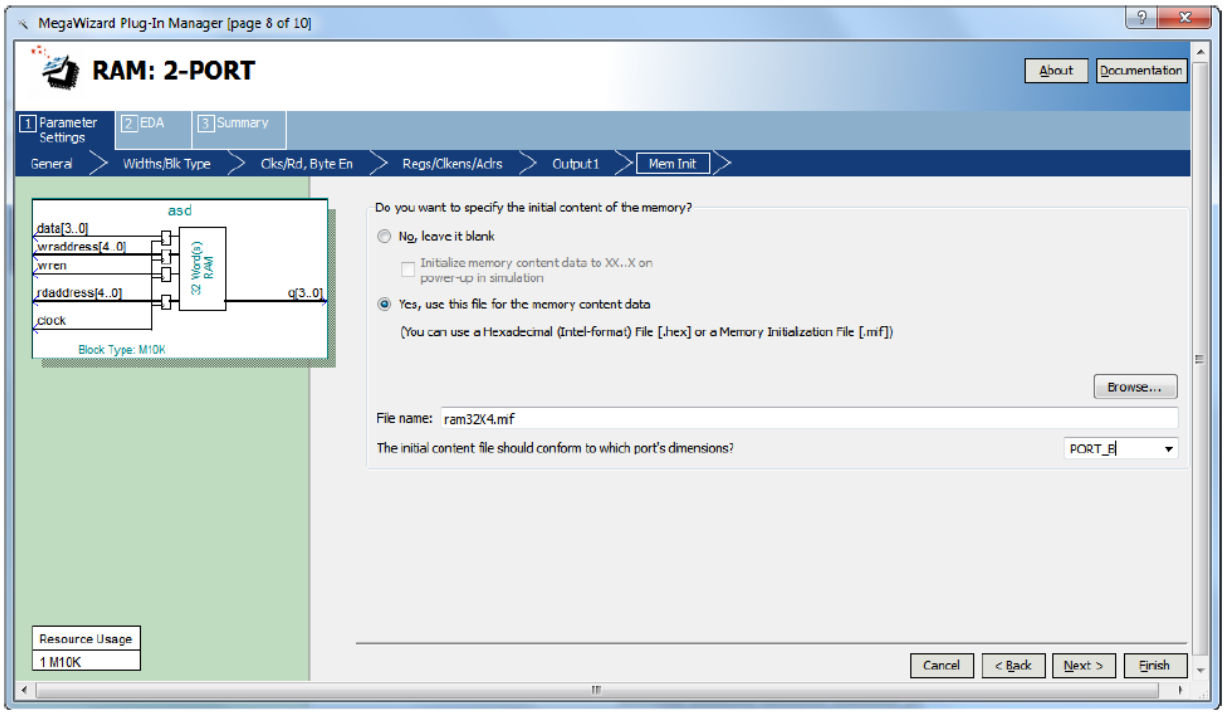


Figure 7: Specifying a memory initialization file (MIF).

```
DEPTH = 32;
WIDTH = 4;
ADDRESS_RADIX = HEX;
DATA_RADIX = BIN;
CONTENT
BEGIN

0 : 0000;
1 : 0001;
2 : 0010;
3 : 0011;
. . . (some lines not shown)
1E : 1110;
1F : 1111;

END;
```