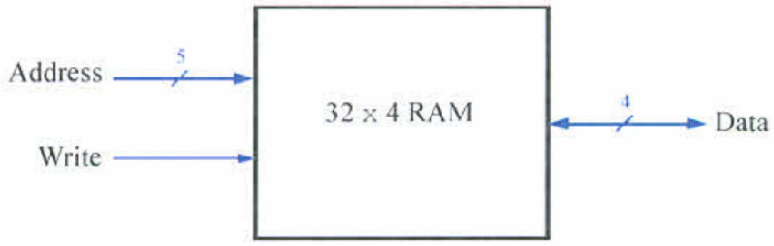
Laboratory Exercise 9

Memory Blocks

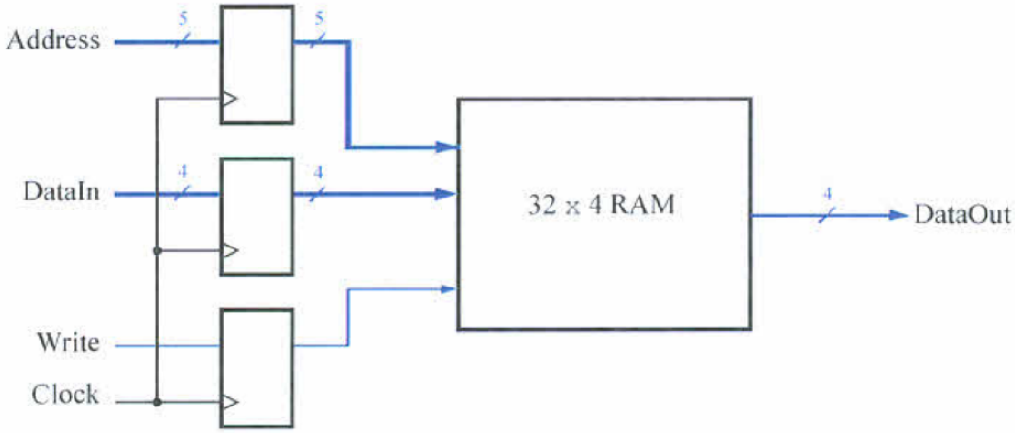
In computer systems it is necessary to provide a substantial amount of memory. If a system is implemented using FPGA technology it is possible to provide some amount of memory by using the memory resources that exist in the FPGA device. In this exercise we will examine the general issues involved in implementing such memory.

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.

FPGAs generally provide some internal dedicated memory resources. The DE10-Lite boards have M9K memory blocks for memory resources. Each M9K block contains 8192 memory bits. M9K 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 M9K 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 that have to be mentioned. First, they include registers that can be used to synchronize all of 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.

Bonus Opportunity:

You may do research and figure out how to “infer” these RAMs in pure VHDL rather than using the IP Catalog/Megawizard, skipping the wizard-generated steps and writing your own RAM. This is my preferred method since it removes difficult to manage megawizard flows and makes your design more portable and testable. If you do this method, you still need to do both kinds of RAM, and initialize the 2nd kind per the lab manual (though without using the .mif). Note that this is a place where to meet the inference requirements you may need to use a shared variable.

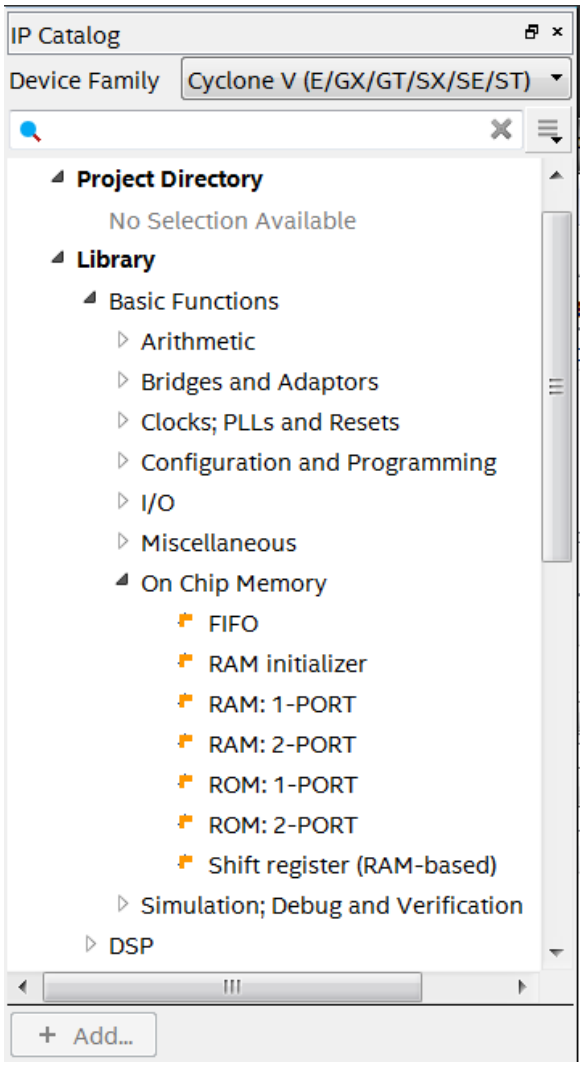
Both Part A and Part B NEED you to create an .sdc to define the clock. This is a new requirement for labs. You should not have a missed timing requirements warning in quartus if you have done this correctly. You'll need to add this .sdc file to your Quartus Project.

Part A

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. As in previous labs, use the template project provided.

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 M9K as the memory block type. 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;
```

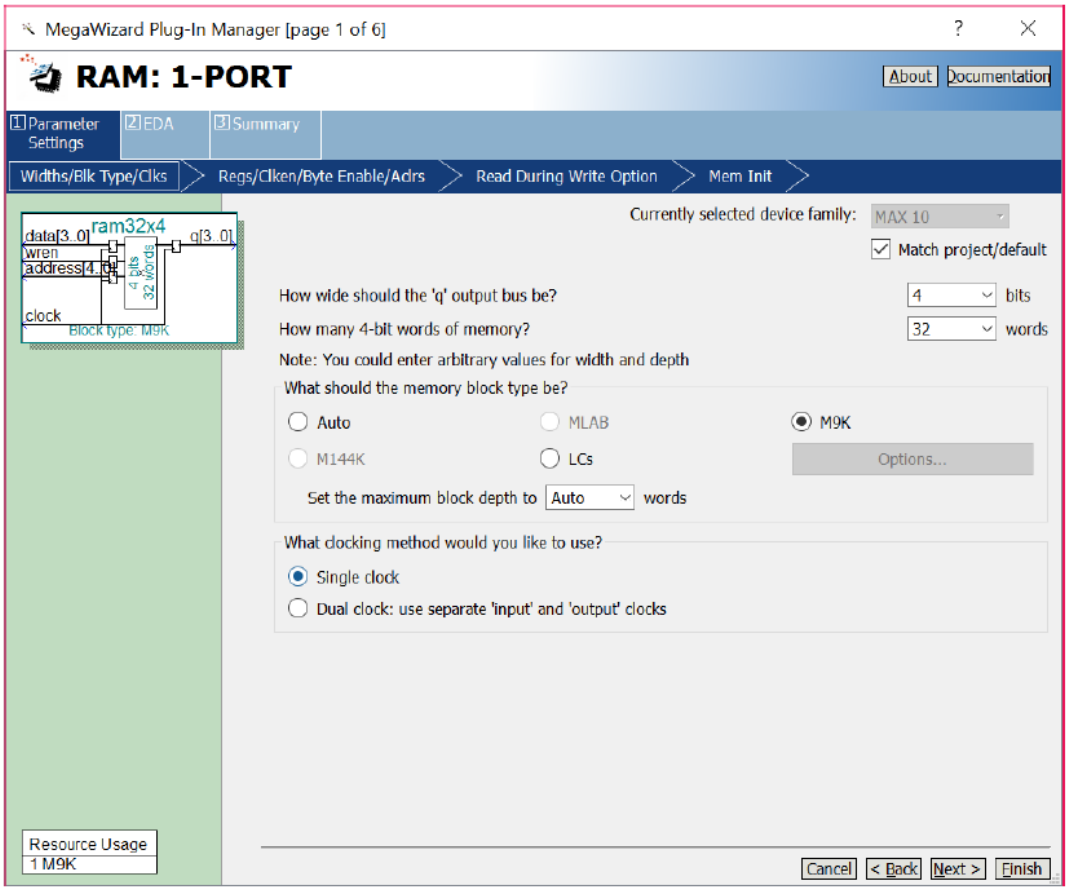


Figure 2 Configuring the size of the memory module.

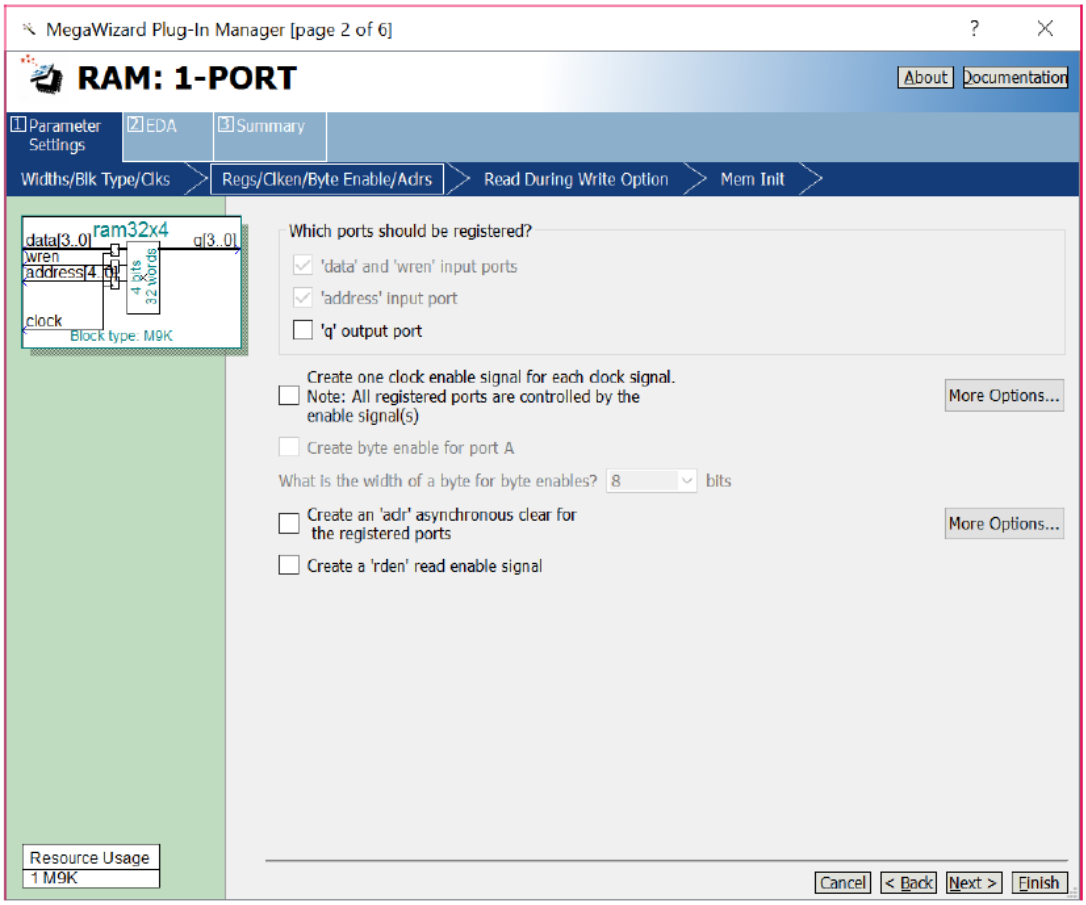


Figure 3 Configuring the Input and Outputs

Now, we want to realize the memory circuit in the FPGA on your DE-series 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. Instantiate the ram32x4 module and that includes the required input and output pins on your DE-series board. Use (synchronized) slide switches SW_{3-0} to provide input data for the RAM, and use switches SW_{8-4} to specify the address. After synchronizing SW_9, using rising edge detector output as the Write signal and the 50MHz clock as the input. Show the address value on the 7-segment displays HEX5 - 4, show the data being input to the memory on HEX2, and show the data read out of the memory on HEX0. Key(0) (synchronized) is your reset input.
2. Test (with a testbench) your circuit and make sure that data can be stored into the memory at various locations.

Save this folder as lab9a, for submission later.
Copy lab9a into a lab9b folder as a starting point for this part

Part B

The SRAM block in Figure 5 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.

1. 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 inFigure 4, choose “With one read port and one write port” in the category called “How will you be using the dual port ram?” (This generates a RAM typically called “simple dual port ram” vs a “true dual port ram” which allows reads and writes on both sides.)

Configure the memory size, clocking method, and registered ports the same way as Part A. As shown in Figure 5 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 6 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 in 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. Instantiate your dual-port memory at the top. To see the RAM contents, add to your design a capability to display the content of each four-bit word (in hexadecimal format) on the 7-segment display HEX0. Use a counter as a read address, and scroll through the memory locations by displaying each word for about one second. As each word is being displayed, show its address (in hex format) on the 7-segment displays HEX3-2. Use the 50 MHz clock, CLOCK_50, 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 and show the write data on HEX1. Make sure that you properly synchronize the slide switch inputs to the 50 MHz clock signal. After synchronizing SW_9, using rising edge detector output as the Write signal
3. Test with a testbench 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.
4. You will want a generic to speed up the 1sec tick rate for simulation like we've done in the previous labs.

Note: You'll need to add the following line to your .qsf to allow quartus to build for this device with internal configuration **and** a memory initialization: set_global_assignment -name INTERNAL_FLASH_UPDATE_MODE "SINGLE IMAGE WITH ERAM"

Submit both lab9a and lab9b via Canvas either as separate .zip files or zip a folder with both of them as subfolders with your lab report as normal.

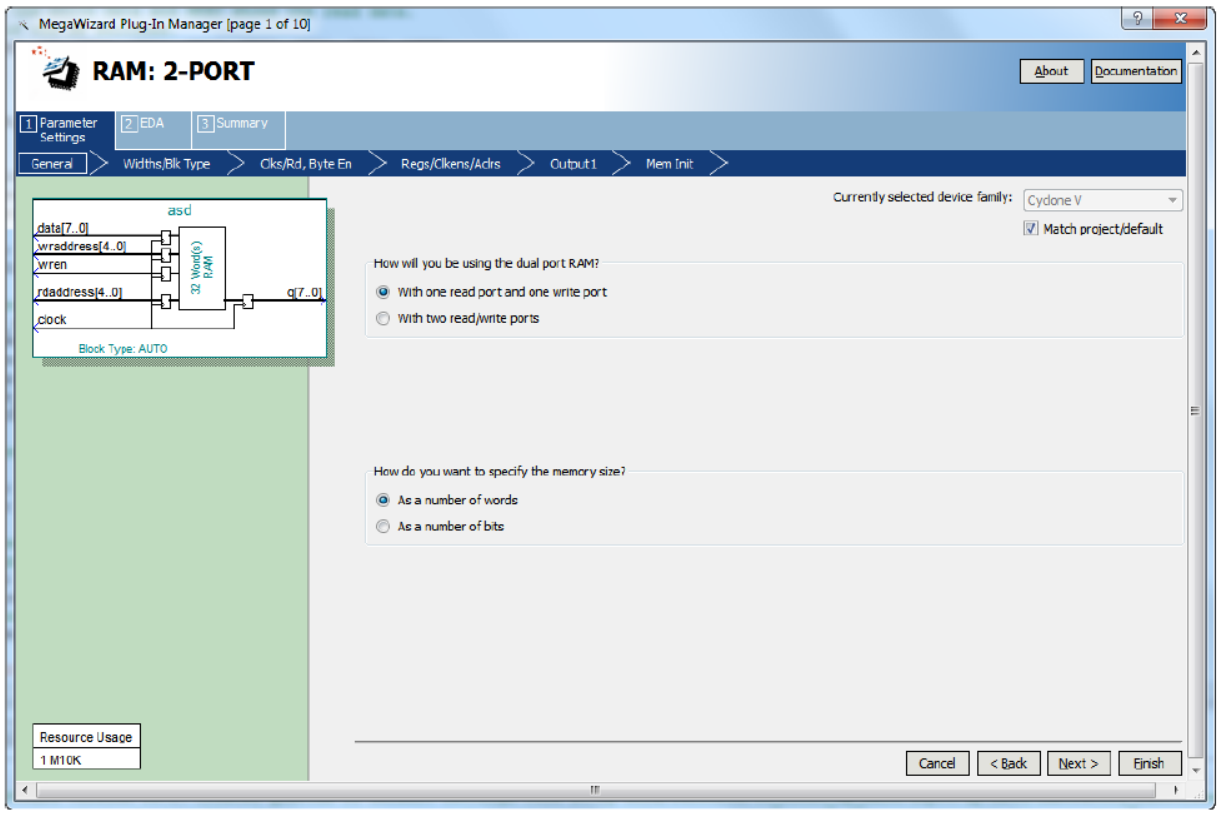


Figure 5: Configuring the two input ports of the RAM.

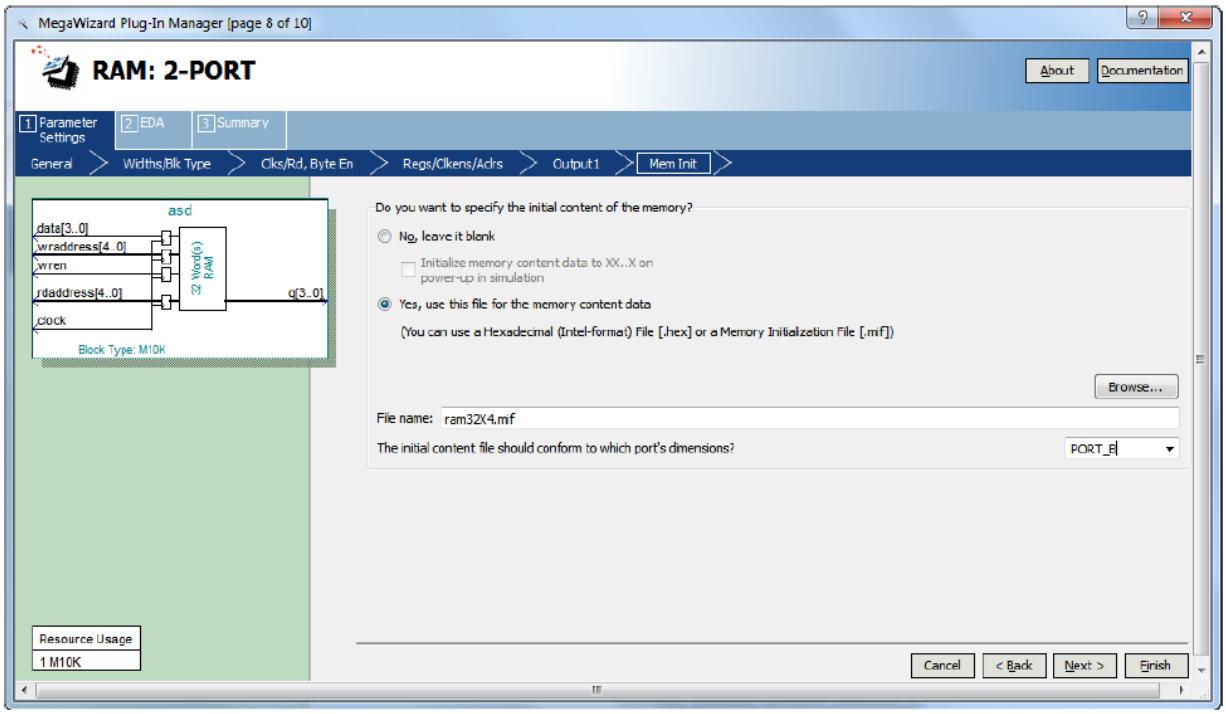


Figure 7: Specifying a memory initialization file (MIF).

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;