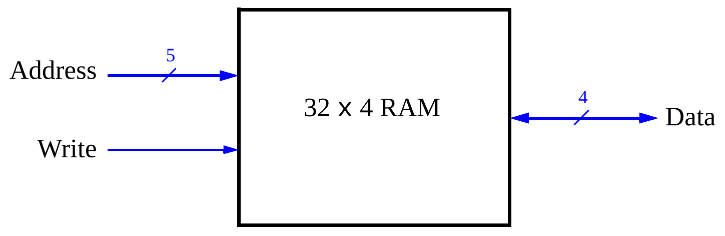
Laboratory Exercise 8

Memory Blocks

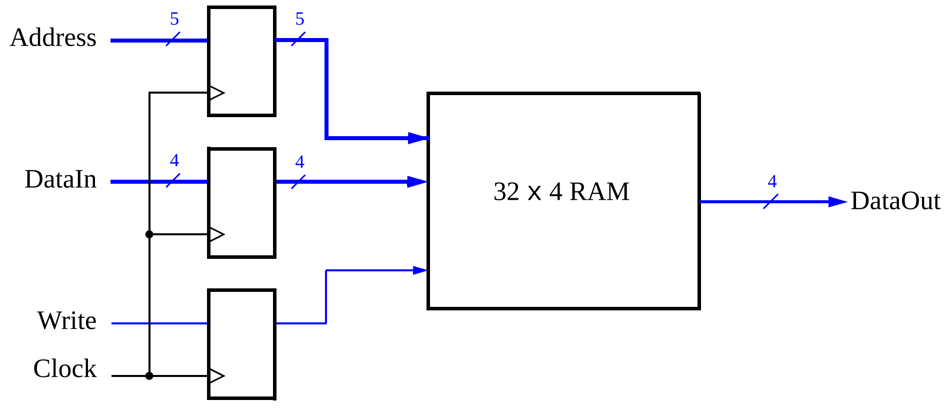
In computer systems it is necessary to provide a substantial amount of memory. If a system is implemented using FPGA technology it is possible to provide some amount of memory by using the memory resources that exist in the FPGA device. In this exercise we will examine the general issues involved in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

A diagram of the random access memory (RAM) module that we will implement is shown in Figure 1*a*. 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 Cyclone series of FPGAs that are included on the DE0-CV, DE1-SoC, and DE2-115 boards provide dedicated memory resources. The Cyclone V FPGA on the DE0-CV and DE1-SoC boards contain dedicated memory resources called *M10K blocks*, and the Cyclone IV on the DE2-115 contain dedicated memory resources called *M9K blocks*. Each M10K block contains 10240 memory bits, and each M9K block contains 9216 memory bits. Both M10K and 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). The aspect ratios common to both blocks are 8K x 1 and 2K x 4. We will utilize the 2K x 4 mode in this exercise, using only the first 32 words in the memory. We should also mention that M10K and M9K blocks support many other modes of operation, but 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 M10K and M9K blocks that have to be mentioned. First, they includes 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 1*b*. It includes registers for the *address*, *data input*, and *write* ports, and uses a separate unregistered *data output* port.

Part I

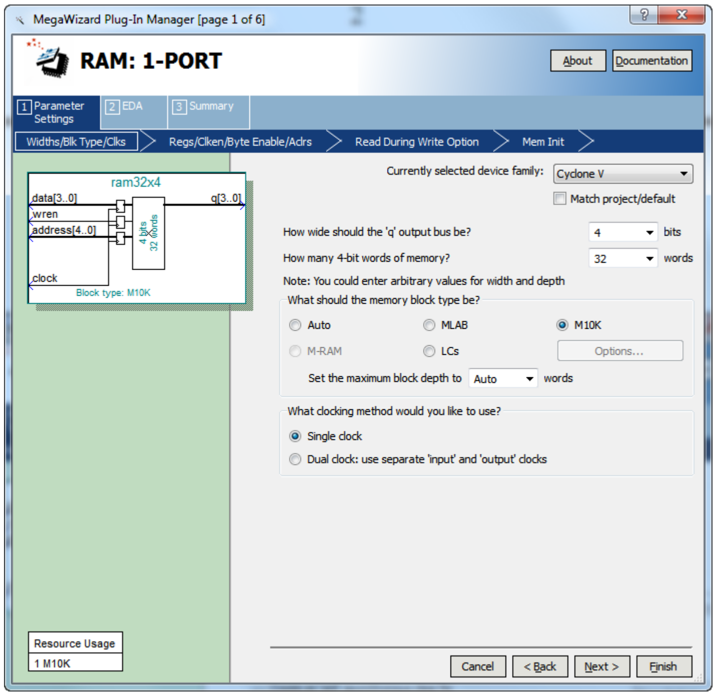
Commonly used logic structures, such as adders, registers, counters and memories, can be implemented in an FPGA chip by using prebuilt modules that are provided in *libraries*. In this exercise we will use a module called the *altsyncram* to implement the memory in Figure 1*b*.

1. Create a new Quartus II project to implement the memory module.

2. You can learn how the IP Catalog is used to generate a desired module by reading the tutorial *Using Library Modules in VHDL Designs*. This tutorial is provided in the University Program section of Altera's web site.

   In the IP Catalog 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 in Figure 2 specify a memory size of 32 four-bit words.Select M10K if your DE-series board has a Cyclone V FPGA or M9K if it has a Cyclone IV FPGA. 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 1*b*, 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;
   ```


MegaWizard Plug-In Manager [page 1 of 6]

RAM: 1-PORT

About | Documentation

1 Parameter Settings | 2 EDA | 3 Summary

Widths/Blk Type/Clks > Regs/Clken/Byte Enable/Adrs > Read During Write Option > Mem Init >

Currently selected device family: Cyclone V

☐ Match project/default

How wide should the 'q' output bus be? 4 bits

How many 4-bit words of memory? 32 words

Note: You could enter arbitrary values for width and depth

What should the memory block type be?

○ Auto ○ MLAB ● M10K

○ M-RAM ○ LCs Options...

Set the maximum block depth to Auto words

What clocking method would you like to use?

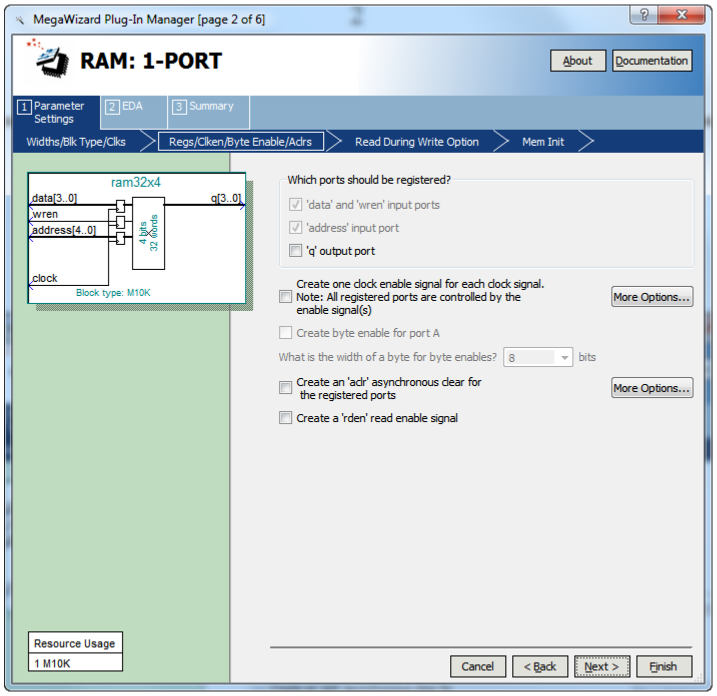
● Single clock

○ Dual clock: use separate 'input' and 'output' clocks

Resource Usage
1 M10K

Cancel | < Back | Next > | Finish

Figure 2: Configuring the size of the memory module.


MegaWizard Plug-In Manager [page 2 of 6]

RAM: 1-PORT

About | Documentation

1 Parameter Settings | 2 EDA | 3 Summary

Widths/Blk Type/Clks > Regs/Clken/Byte Enable/Adrs > Read During Write Option > Mem Init >

Which ports should be registered?

☑ 'data' and 'wren' input ports

☑ 'address' input port

☐ 'q' output port

☐ Create one clock enable signal for each clock signal. Note: All registered ports are controlled by the enable signal(s) More Options...

☐ Create byte enable for port A

What is the width of a byte for byte enables? 8 bits

☐ Create an 'aclr' asynchronous clear for the registered ports More Options...

☐ Create a 'rden' read enable signal

Resource Usage
1 M10K

Cancel | < Back | Next > | Finish

Figure 3: Configuring input and output ports.

3. Instantiate this subcircuit in a top-level VHDL file that includes appropriate input and output signals for the memory ports given in Figure 1*b*.

4. Compile the circuit. Observe in the Compilation Report that the Quartus II Compiler uses 128 bits in one of

the M10K memory blocks to implement the RAM circuit.

5. Simulate the behavior of your circuit and ensure that you can read and write data in the memory.

Part II

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. Make a new Quartus II project which will be used to implement the desired circuit on your DE-series board.
2. Create another VHDL file that instantiates the *ram32x4* module and that includes the required input and output pins on your DE-series board. Use slide switches SW_{3-0} to provide input data for the RAM, and use switches 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 $HEX5-4$, 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 and make sure that data can be stored into the memory at various locations.

Part III

Instead of creating a memory module subcircuit by using the IP Catalog, we can implement the required memory by specifying its structure in VHDL code. In a VHDL-specified design it is possible to define the memory as a multidimensional array. A 32 x 4 array, which has 32 words with 4 bits per word, can be declared by the statement

```
TYPE mem IS ARRAY(0 TO 31) OF STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL memory_array : mem;
```

In the Cyclone series of FPGAs, such an array can be implemented either by using the flip-flops that each logic element contains or, more efficiently, by using the built-in memory blocks. The Quartus II Help provides other examples of VHDL code that show how memory can be specified (search in the Help for "Inferred memory").

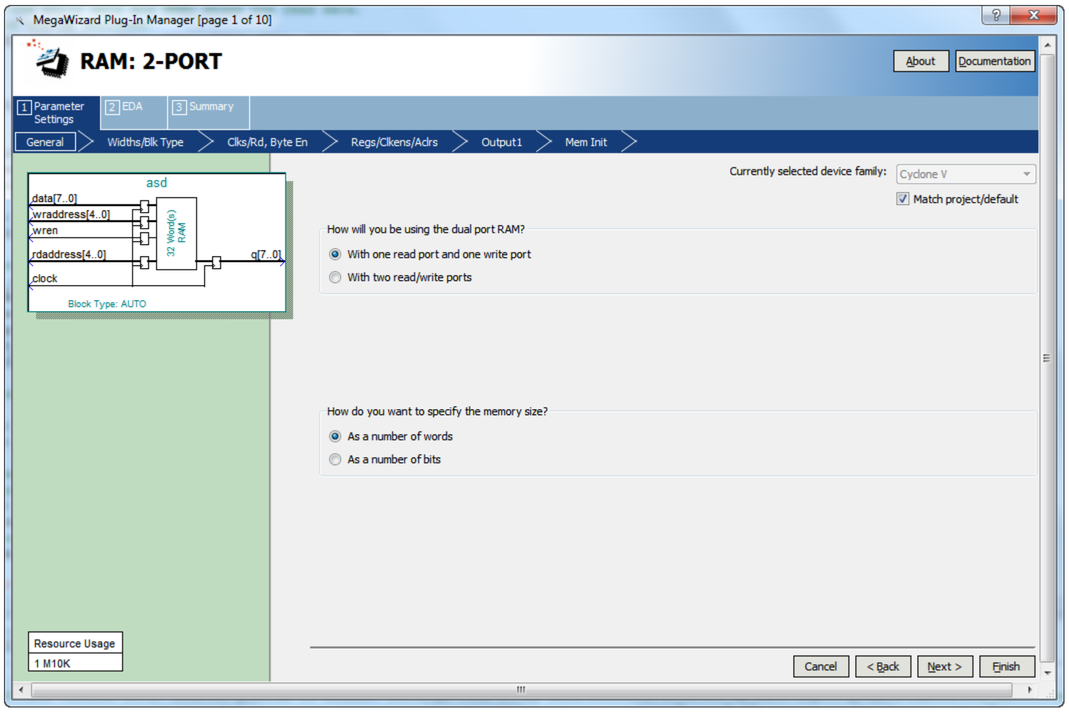
Perform the following steps:

1. Create a new project which will be used to implement the desired circuit on your DE-series board.
2. Write a VHDL file that provides the necessary functionality, including the ability to load the RAM and read its contents as was done in Part II.
3. Assign the pins on the FPGA to connect to the switches and the 7-segment displays.
4. Compile the circuit and download it into the FPGA chip.
5. Test the functionality of your design by applying some inputs and observing the output.

Part IV

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.

1. Create a new Quartus II project for your circuit. 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?


MegaWizard Plug-In Manager [page 1 of 10]

RAM: 2-PORT

Currently selected device family: Cyclone V

[✓] Match project/default

How will you be using the dual port RAM?

[●] With one read port and one write port

[○] With two read/write ports

How do you want to specify the memory size?

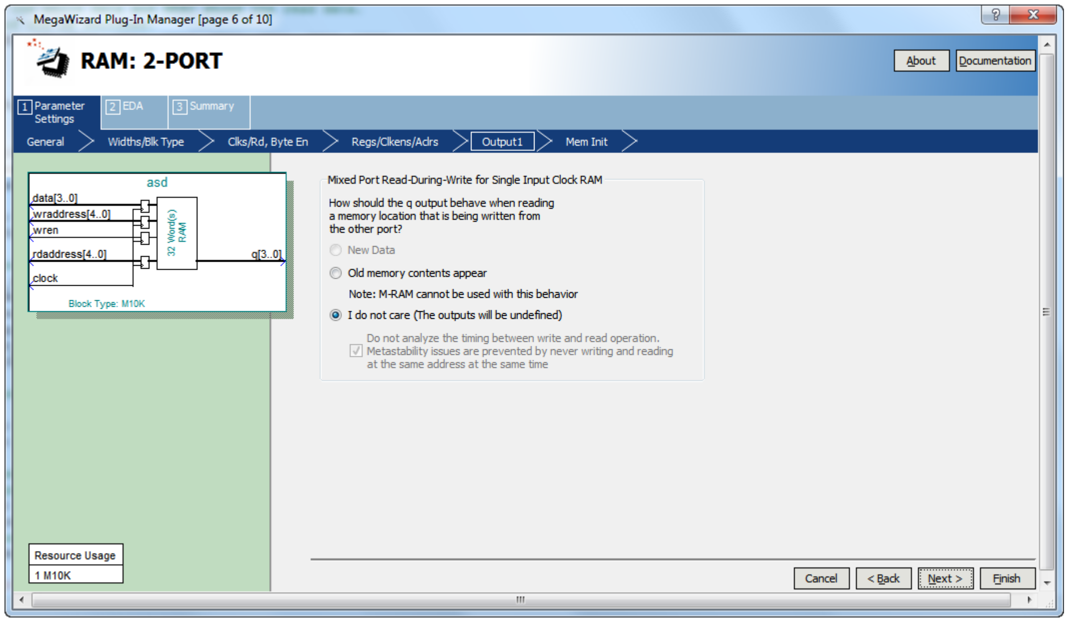
[●] As a number of words

[○] As a number of bits

Resource Usage: 1 M10K

Figure 4: Configuring the two input ports of the RAM.

Configure the memory size, clocking method, and registered ports the same way as Part II. 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.


MegaWizard Plug-In Manager [page 6 of 10]

RAM: 2-PORT

Mixed Port Read-During-Write for Single Input Clock RAM

How should the q output behave when reading a memory location that is being written from the other port?

[○] New Data

[○] Old memory contents appear

Note: M-RAM cannot be used with this behavior

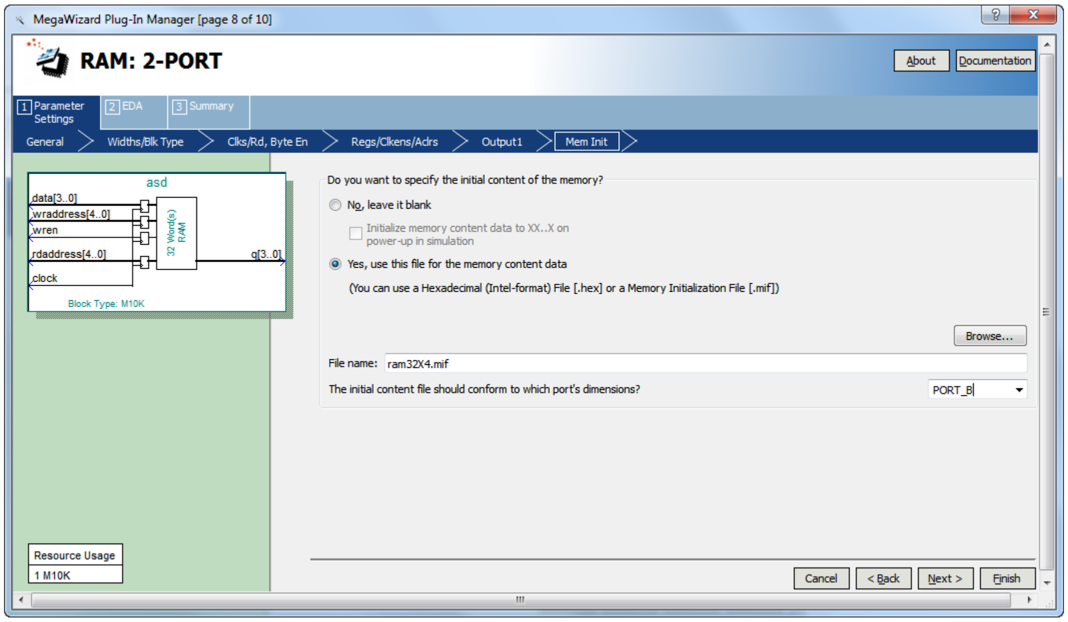
[●] I do not care (The outputs will be undefined)

[✓] Do not analyze the timing between write and read operation. Metastability issues are prevented by never writing and reading at the same address at the same time

Resource Usage: 1 M10K

Figure 5: Configuring the output of the RAM when reading and writing to the same address.

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 II Help. You will need to create a MIF file like the one in Figure 7 to test your circuit. Finish the Wizard and then examine the generated memory module in the file *ram32x4.vhd*.



Do you want to specify the initial content of the memory?

No, leave it blank

Initialize memory content data to XX..X on power-up in simulation

[✓ Yes, use this file for the memory content data]

(You can use a Hexadecimal (Intel-format) File [.hex] or a Memory Initialization File [.mif])

File name: ram32X4.mif

The initial content file should conform to which port's dimensions? PORT_B

Figure 6: 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;

Figure 7: An example memory initialization file (MIF).

2. Write a VHDL file that instantiates your dual-port memory. 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

*HEX*0. 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 *HEX*$3-2$. 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 *HEX*$5-4$ and show the write data on *HEX*1. Make sure that you properly synchronize the slide switch inputs to the 50 MHz clock signal.

3. 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.