Department of Electrical Engineering

ASIC/FPGA Chip Design

Laboratory Manual

ASIC/FPGA Chip Design Laboratory Manual

Dr. Mahdi Shabany

Spring 2025

Contents

Department of Electrical Engineering

ASIC/FPGA Chip Design

Lab 3 - Complex State Machines and Video Graphics Array (VGA) Display

Lab 3 - Complex State Machines and Video Graphics Array (VGA) Display

1.1 Objective

The purpose of this laboratory is to further expand your understanding of finite state machines (FSMs) and to learn how one can use a "Video Graphics Array" (VGA) Adapter to create pictures and animation on a computer screen. You will create FSMs to interact with a VGA Adapter "core" that has been implemented for you to control and generate images and animations on the screen.

1.2 Instructions

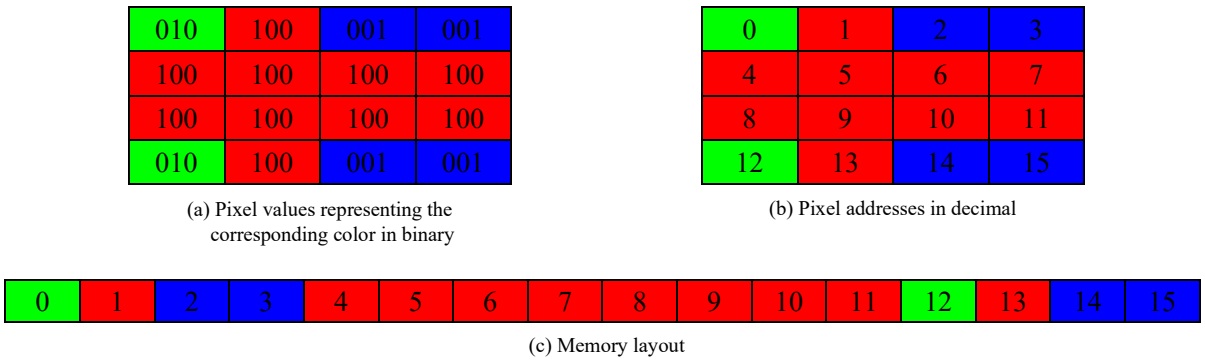
1.2.1 Section 1: Background & Pixel Control

1.2.1.1 Part a: Image Setup & Editing

To familiarize yourself with the VGA Adapter module you will perform a simple exercise to display a custom image on the screen. To do this you will need to use the bmp2mif converter provided in the starter kit. It is a program that converts a bitmap image into a stream of bits that can be programmed into the memory on an FPGA. To be able to display a picture you must first draw a picture using a graphics editing tool that can save files in a BMP format. The Microsoft Windows "Paint" program is one such tool. The image you will draw will cover the entire screen and thus has to be created correctly. Perform the following steps to draw an image:

1. Start the Paint program, typically available from the Start Menu:
   Start –> Programs –> Accessories –> Paint
2. Select the menu item Image –> Attributes. In the dialog box set "width" equal to 160 and "height" equal to 120, as this is the resolution of the monitor that the adapter uses. Select "pixels" as the unit. Select "colors" as well.
3. Draw a picture of your own design; use simple colors like Red, Green and Blue.
4. Save the file, using File –> Save As and save it as a 24-colour bitmap image.bmp.
5. Run the bmp2mif.exe converter program (a windows program described on the VGA website and available from the download directory on that site) to convert your BMP file to a Memory Initialization File (MIF) we will use next. (To do that, start up a DOS command shell on windows using Start –> Run and type "cmd" into the Open: box that pops up. This will create a window you can type commands into. Change into a folder that contains both

bmp2mif.exe and the image.bmp you wish to convert. Then type "bmp2mif image.bmp", This will produce two .mif files (image.mono.mif and image.colour.mif).



(a) Pixel values representing the corresponding color in binary

(b) Pixel addresses in decimal

(c) Memory layout

Figure 1.1: In-memory representation of the 4x4 pixel image. Each box holds a pixel color, a 3-bit value in our case

To ensure that you understand how the MIF file is used and what it represents, you are required to add a cyan 2x2 square with the top left corner placed at coordinates (x, y), where x corresponds to the last two digits of your student number and y corresponds to the second last two digits of your student number. (Which location in the VGA Adapter memory would you have to change to alter to color of the specified pixel?) You must use the MIF editor to do this. The MIF editor can be launched by simply opening the .mif file in Quartus. Refer to figure 1.1 for an illustration of how the VGA framebuffer is organized. Recall from the description of the VGA Adapter that it uses memory to store the current color of each pixel on the screen. Usually, this memory is initialized to 0 at first and hence you only see a black background. However, we can change the initial state of the VGA Adapter memory, causing it to display an image. You will use the image you created earlier with paint as that background.

Perform the following steps to change the initial image displayed by the Adapter:

1. The project for this part is provided in the starter kit. Open the project named background in the part1 subdirectory to begin your work.
2. The bmp2mif converter created a file called image.colour.mif, where your background image is stored. Copy this file to your working directory and change its name to display.mif.

   Note: The choice of the file name is not accidental. If you look carefully at the implementation of the VGA Adapter you will see that it has a parameter called BACKGROUND_IMAGE. This parameter is set to "display.mif" by default and signifies that the Quartus II software should use display.mif file to initialize the memory for the VGA Adapter. Note also that there can only be one display file which is programmed into the memory when the FPGA is configured (when you 'download' your design). It is a common mistake to think that you can

create many of these files and somehow cause the FPGA to switch through them; this isn't possible as the download only happens once.

3. Assign pins to your project and compile it.

DE2: Program the circuit onto the Altera DE2 board. When you program the DE2 board and connect a monitor to its graphics port, you should be able to see the image you have drawn.

⚠ **Important Note:**
Make sure you understand how this initialization process works: the memory (in the case of the VGA adapter, this memory is the framebuffer) is initialized with the contents of the MIF file (which stands for "Memory Initialization File") only when the FPGA is programmed. The MIF file is just the stream of raw data, and is not specific to the VGA Adapter - it can be used to initialize any kind of memory. The memory initialized this way can be changed – by modifying individual pixels on the screen as discussed in the VGA Adapter documentation. Note that as soon as you draw a pixel using the VGA Adapter, the contents of this memory will be altered. Thus, if you used a background image as we have shown above, the background image will be permanently altered. Resetting the VGA Adapter will NOT restore the "background image".

1.2.1.2 Part b: Interactive Pixel Plotting

In this part you are asked to design a very simple circuit using the VGA adapter. The circuit has to perform the following functions:

1. Accept the X and Y coordinate inputs and the color input from the switches on the DE2 board.
2. Set the given color of the pixel at the given coordinates when a push-button is pressed.

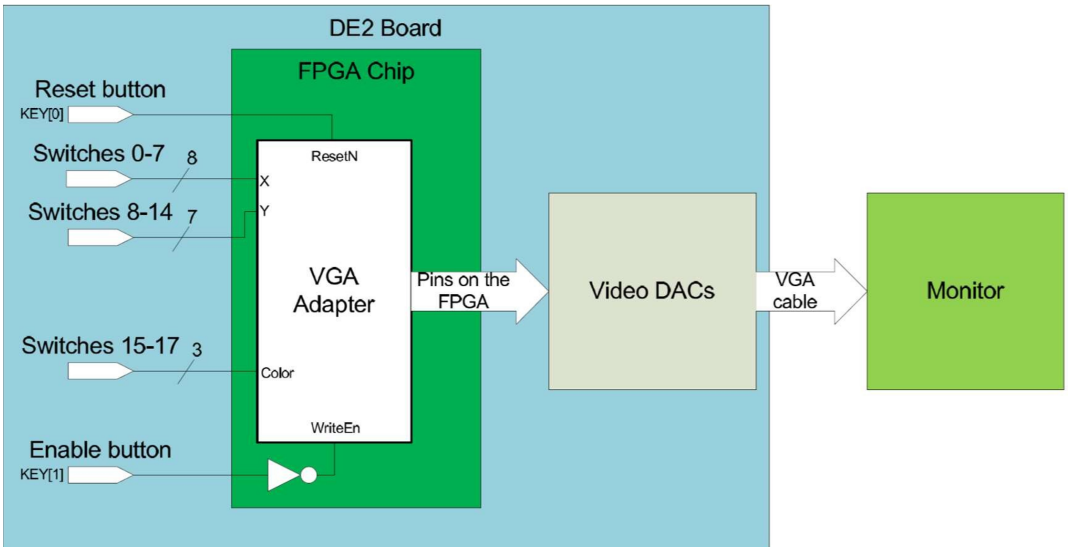


Figure 1.2: Full schematic of the circuit

The schematic of the circuit is given in figure 1.2. You should reuse the project file from Part a and design a circuit that connects up the switches on the DE2 board as given in the figure. After completing this circuit, you will be able to manually (and tediously) draw any picture on the screen. You will do this by choosing a (binary) value for the X and Y location of each pixel to change, choose a value for the color, and then pushing the "enable" button to cause the specific pixel to change. Observe that any image which you loaded in Part a is over-written as you change the colors of the pixels. This is because the VGA adapter has only one block of memory to store the value of each pixel (this block is called a **framebuffer** in the graphics world). This buffer was initialized with your image, but as you change the colors of the pixels, the old values are overwritten and lost.

1.2.2 Section 2: Shape Motion Control

1.2.2.1 Part a: Rectangle Motion Control

In this section, you draw a rectangle with arbitrary dimensions, and then modify the code in such a way that the rectangle moves in four different directions using four keys. Please note that when the rectangle reaches the edges of the display, it should stop moving and not go beyond them.

1. ResetN - an active low input to reset the system. Reset should cause the square to go to middle of the screen.
2. CLOCK_50 - the clock input to drive the finite state machine of the system as well as the VGA adapter.
3. Four switches labeled left, right, up, down - to indicate the direction to move the rectangle in. Each time an input, for example left, is set to 1, the drawing system should move the rectangle left by exactly 1 pixel. (It shouldn't wait until the Left signal returns to 0 before moving the cursor again).

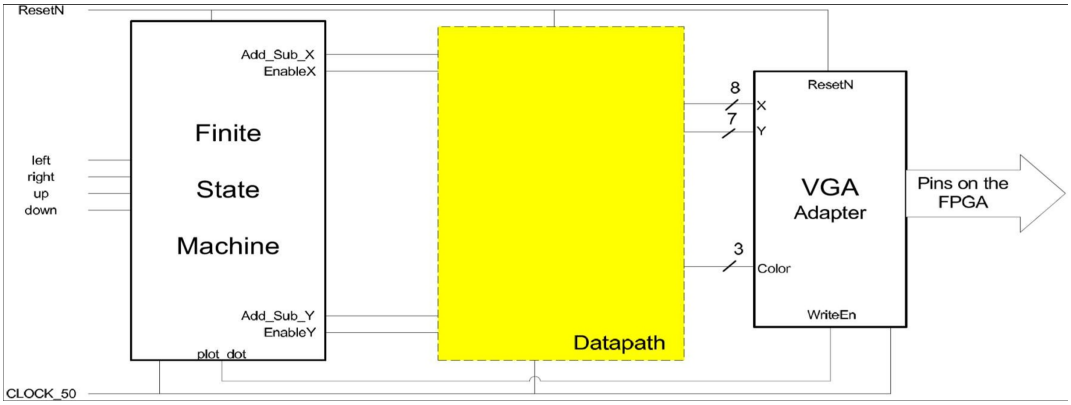


Figure 1.3: Design Overview - State Machine, Datapath and VGA Adapter. Although not shown, ResetN signal should be connected to all the registers in the circuit (including FSM state register).

The high-level design of the circuit for the etch-a-sketch system is given in figure 1.3. It contains 3 major blocks:

1. The VGA adapter responsible for the drawing of pixels on the screen, which you have been learning about it in section 1.
2. The "datapath" that controls the position of the square, providing the VGA adapter with the (x, y) (i.e., column and row) location where a pixel should be drawn.
3. A finite state machine that receives the input from a user and directs the datapath to change the position of the cursor accordingly, by adding and/or subtracting from the (x, y) position of the cursor.

You may use the circuit from Part b in section 1 as a base for your design.

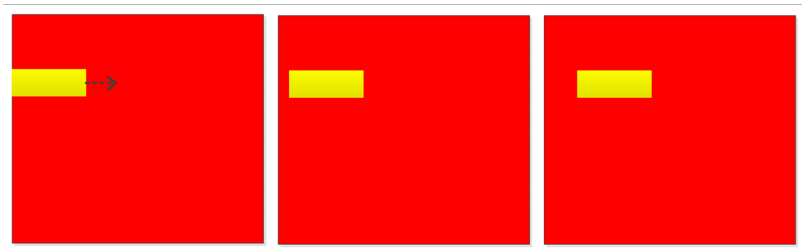


Figure 1.4: Movement of the rectangle when pushing right key (switch)

1.2.2.2 Part b (Bonus): Orientation & Scrolling

Modify your code in previous section to change the orientation of the rectangle when changing the orientation of its original movement. The scrolling graphic have to respond to key pushes in the following way:

Original Orientation	Key	Change
Horizontal	Up	Flip vertical and scroll up
	Down	Flip vertical and scroll down
	Left	No change
	Right	No change
Vertical	Up	No change
	Down	No change
	Left	Flip horizontal and scroll left
	Right	Flip horizontal and scroll right

Table 1.1: Orientation Changes Based on Key Input

The following figures show the change in the graphic due to a few key pushes:

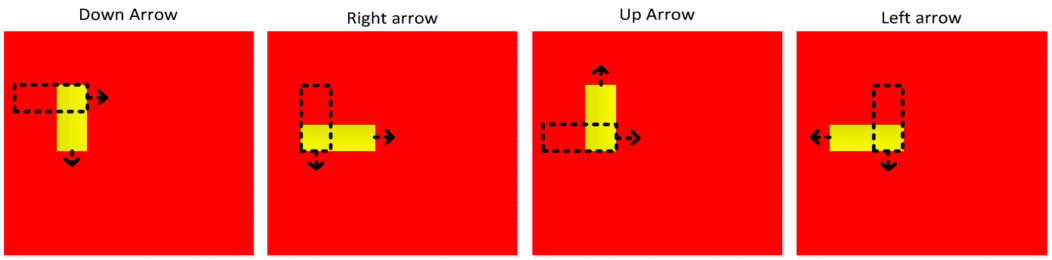


Figure 1.5: Change in the graphic due to a few key pushes

1.2.2.3 Part c: Circle Resizing Control

Repeat **Part a** using a circle.
Bonus: Modify your code in a way that allows changing the radius of the circle or the dimensions of the square using keys. Please note that the dimensions should be limited to a minimum and maximum value.

1.2.3 Section 3: Image Animation Control

Now that we covered drawing pixels on the screen, we can proceed to create a simple animation. We will create a circuit that takes a small image (16x16 pixels) and moves it around the screen. To accomplish this, your circuit will have to make it seem as though the image is seamlessly moving around the screen. You will implement the circuit in two steps. First, you will design a module that is able to draw (or erase) the image at a given location. Then you will design another module that moves the image around the screen by quickly redrawing it at the different locations.

1.2.3.1 Part a: Bitmap Draw & Erase

To implement this part, you will have to create a circuit that takes as input the (screen_X, screen_Y) coordinate of where the top left corner of the image is to be drawn or erased. The circuit will then either draw the image from the memory (LPM_RAM_DQ module) or erase the image starting at position (screen_X, screen_Y). Images are drawn or erased pixel by pixel. To draw an image, read a single entry from memory and pass that value to the "Color" line on the VGA adapter. During this process, you must set the (X, Y) values on the VGA adapter to the appropriate values. Erasing an image is a similar process, however, instead of passing the values from the "Image RAM" to "Color", you will set "Color" to black (constant 000). To accomplish these steps, you will need to create a state machine that performs the following:

1. Set counter_X and counter_Y to 0.
2. While counter_X is less than 16, either load a pixel value from memory containing image.mif, or set the pixel value to black (if erasing). Then draw that pixel at location (screen_X + counter_X, screen_Y + counter_Y) on the screen. Increment counter_X.
3. If counter_Y is less than 15, then increment counter_Y and set counter_X to 0. Go to step 2.
4. Stop when counter_Y reaches 16.

The suggested circuit diagram is shown in figure 1.6. The "Blank" and "Plot" inputs may be used in several different ways. One way is to have "Blank" input select whether the image should be drawn or erased (i.e., if the "Blank" is high when "Plot" is asserted, the image is erased). Another way is to have two separate inputs, one to start drawing ("Plot"), and another to start erasing ("Blank"). You are free to choose whichever method you want. Implement the circuit by completing the following steps:

1. The project for this part is provided in the starter kit. Open the project named part3 in the part3 subdirectory to begin your work.
2. Create a 16x16 bitmap image that is to move around the screen. Make sure to set the image width and height to 16 pixels.
3. Use the bmp2mif.exe converter to convert the image into an MIF file. Call it image.mif.

4. In your design instantiate a memory using an LPM_RAM_DQ and use image.mif as its memory initialization file.
5. Create a circuit to draw an image at a specified location on the screen as discussed above.
6. Compile the circuit and program it onto the DE2 board. When your circuit starts you should be able to see the image you have drawn somewhere on the screen.

Sample instantiation of LPM_RAM_DQ with "animation.mif" as an input file:

```
parameter IMAGE_FILE = "animation.mif"; assign black_color = 3'b000;
assign gnd = 1'b0;

lpm_ram_dq my_ram(.inclock(CLOCK_50), .outclock(CLOCK_50), data(black_color),
.address(mem_address), .we(gnd), .q(mem_out) );

defparam my_ram.LPM_FILE = IMAGE_FILE;
defparam my_ram.LPM_WIDTH = 3;
defparam my_ram.LPM_WIDTHAD = BITS_TO_ADDRESS_IMAGE + IMAGE_ID_BITS;
defparam my_ram.LPM_INDATA = "REGISTERED";
defparam my_ram.LPM_ADDRESS_CONTROL = "REGISTERED";
defparam my_ram.LPM_OUTDATA = "REGISTERED";
```

Snippet 1.1: Sample instantiation of LPM_RAM_DQ

- mem_out is the RAM output, which should be connected to the VGAAdapter.
- Mem_address is the RAM address (input to this module)
- WIDTH is defined to be 3, which is the color codes (e.g., 000 for black)

Note that this can be done using Tools –> MegaWizard Plug-In Manager, which is described in Tutorial II on the course's website.

Note:
If the LPM_RAM_DQ module does not work for you, follow the steps given in the Appendix A to build your own RAM IP core.

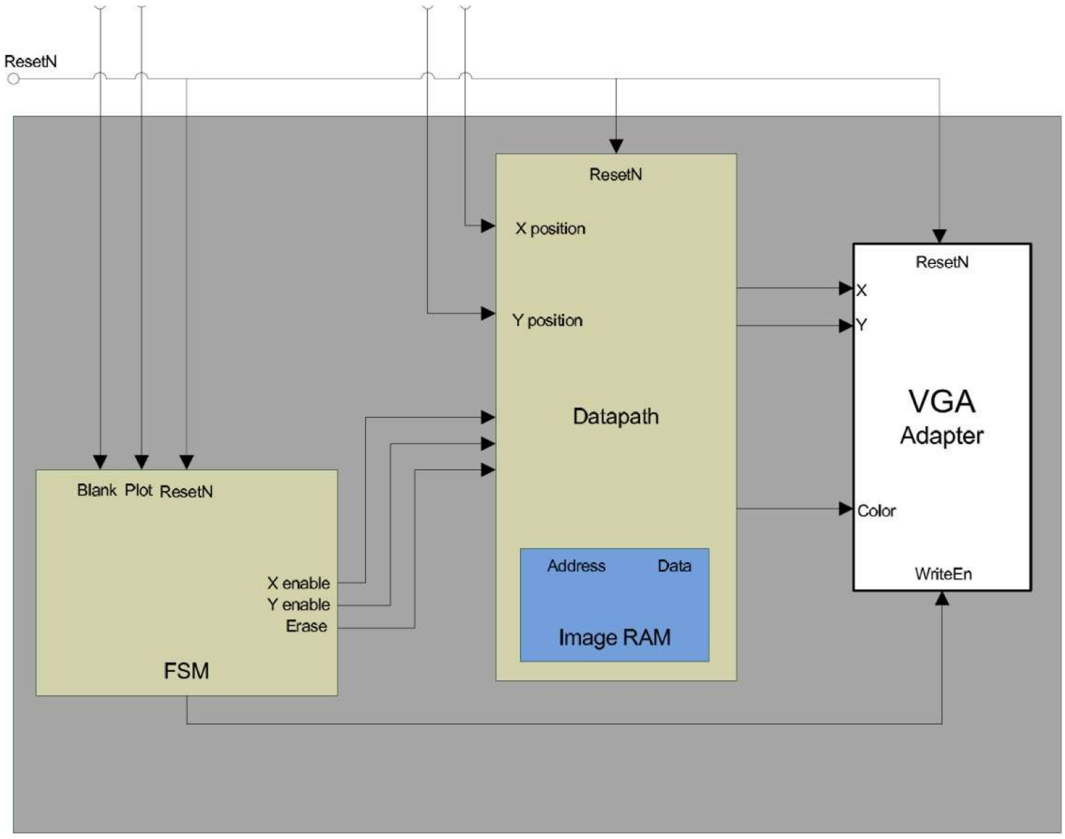


Figure 1.6: Suggested circuit layout for drawing the image

1.2.3.2 Part b: Image Bouncing Animation

The circuit from step 1 can be used to draw an image at any location (X, Y). To move the image, you will have to first erase it from the location it is currently at and then draw it again at an alternate location. There is a simple way to do that when the screen background is black. First, we draw an image at location (X, Y). Then to move the image we simply draw a black box on top of the image (using the erase function of the circuit from step 1) and redraw the image somewhere else. A suggested circuit is shown in figure 1.7, and you may use it as a starting point.

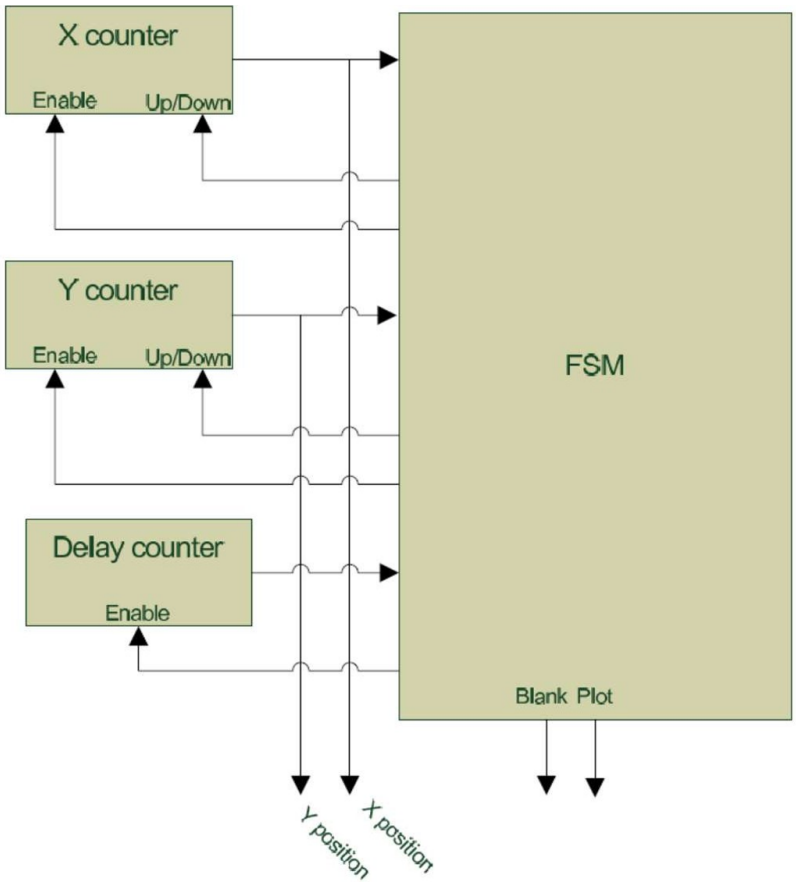


Figure 1.7: Suggested circuit layout for creating an animation

To complete the circuit, perform the following:

1. Create a circuit to change the location of the top left corner of where the image is to be drawn, once every 60th of a second. You may use circuit shown in figure 1.7 as a reference.
2. Put the circuits together to see if your image moves around the screen. Compile the circuit and program it onto the FPGA to see if it works.
3. If the circuit works, think of a way to get the image to bounce of the sides of the screen as it moves about. Implement the enhancement and show the circuit to your TA.

Appendix - RAM IP Core Construction

In this appendix, you will be guided to build your RAM IP core. To do this, follow the steps given in the figures below.

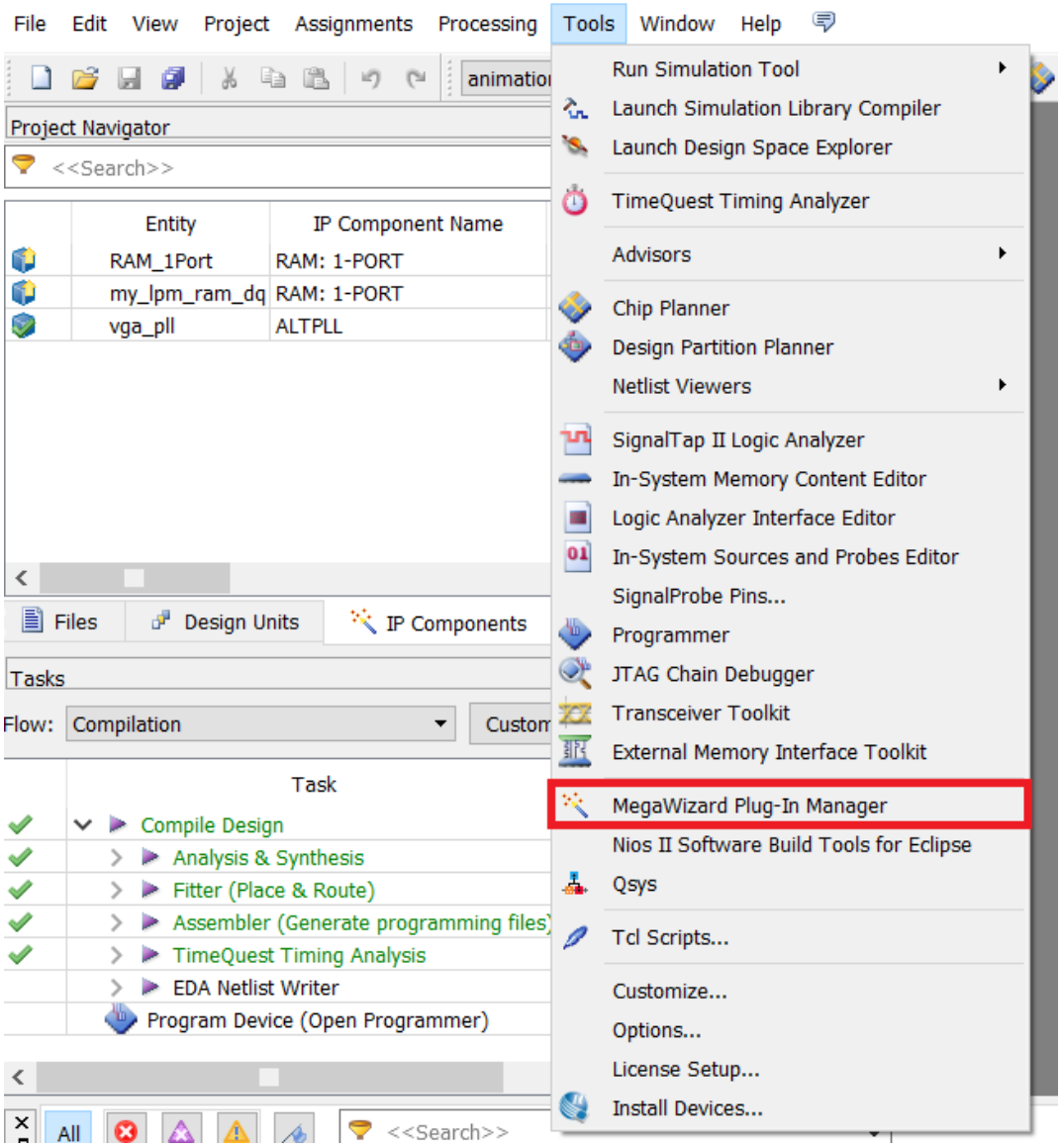


Figure A.1: Step1

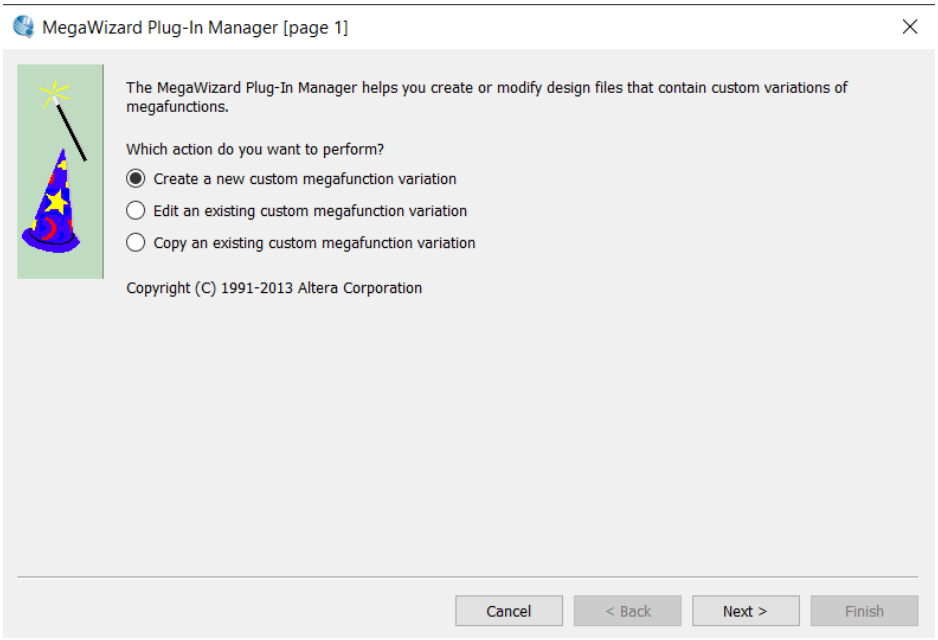


Figure A.2: Step2

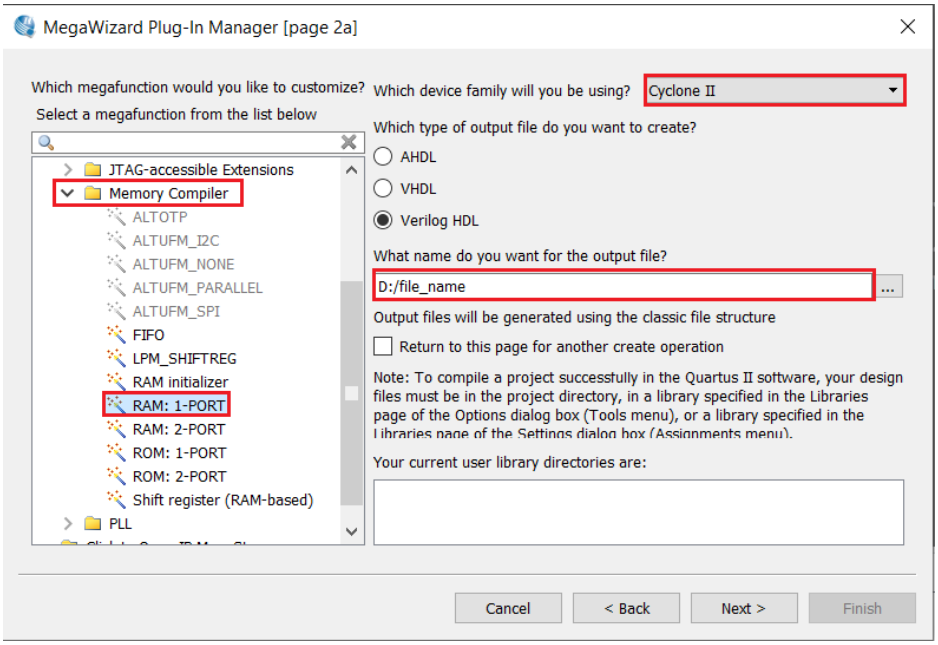


Figure A.3: Step3

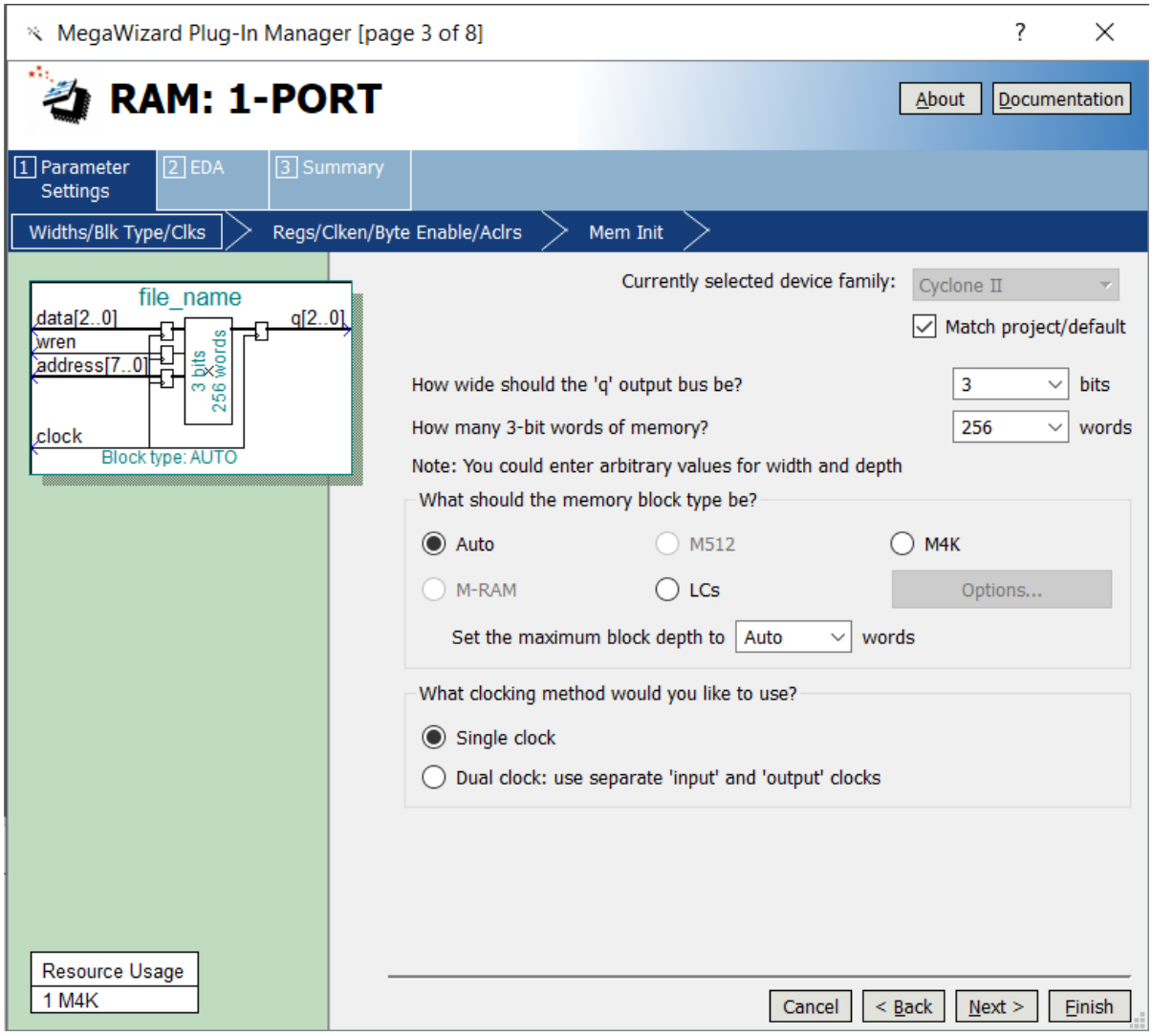


Figure A.4: Step4

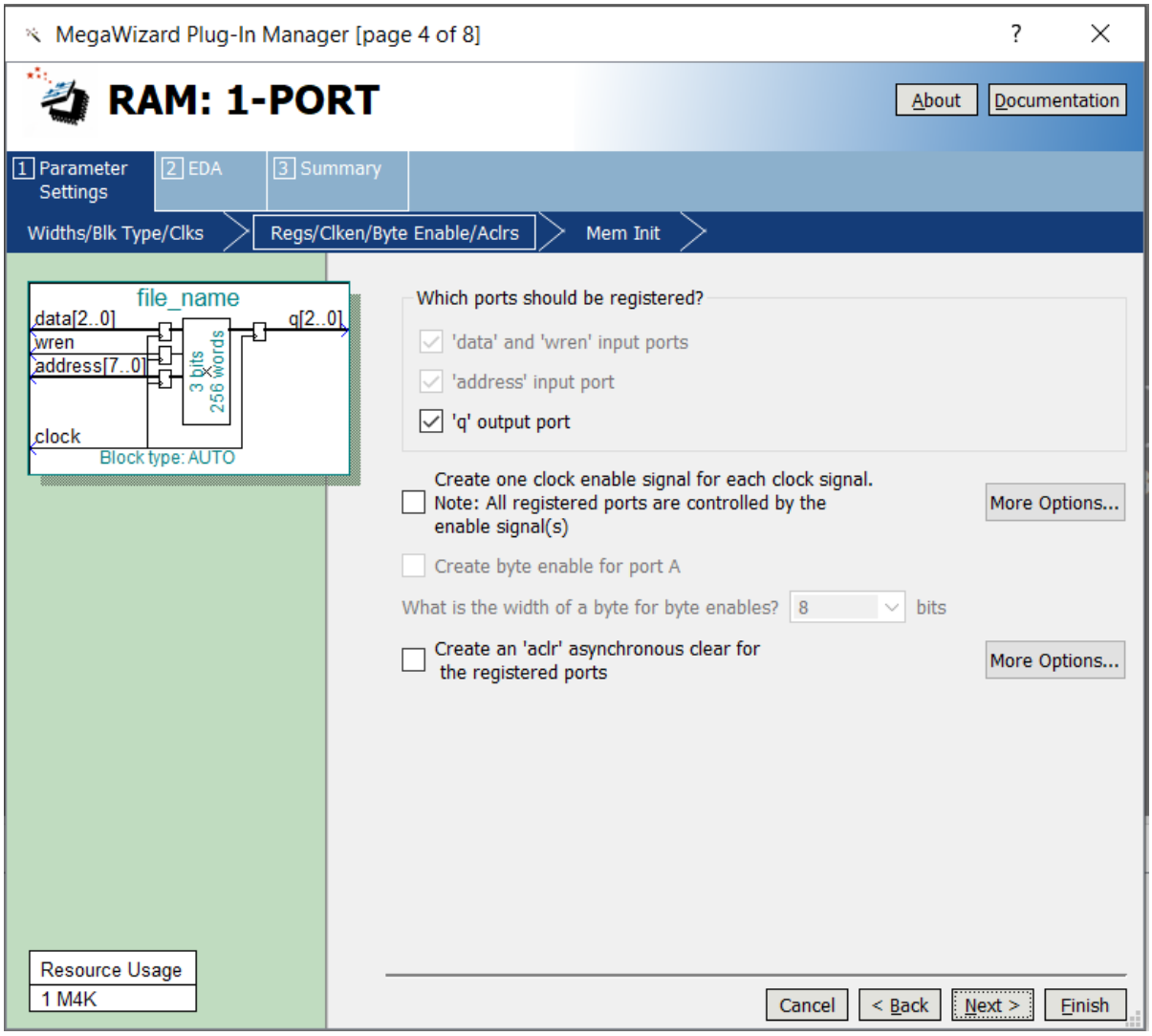


Figure A.5: Step5*

* It depends on your code style whether you register the output or not. In general, it's suggested to register the output.

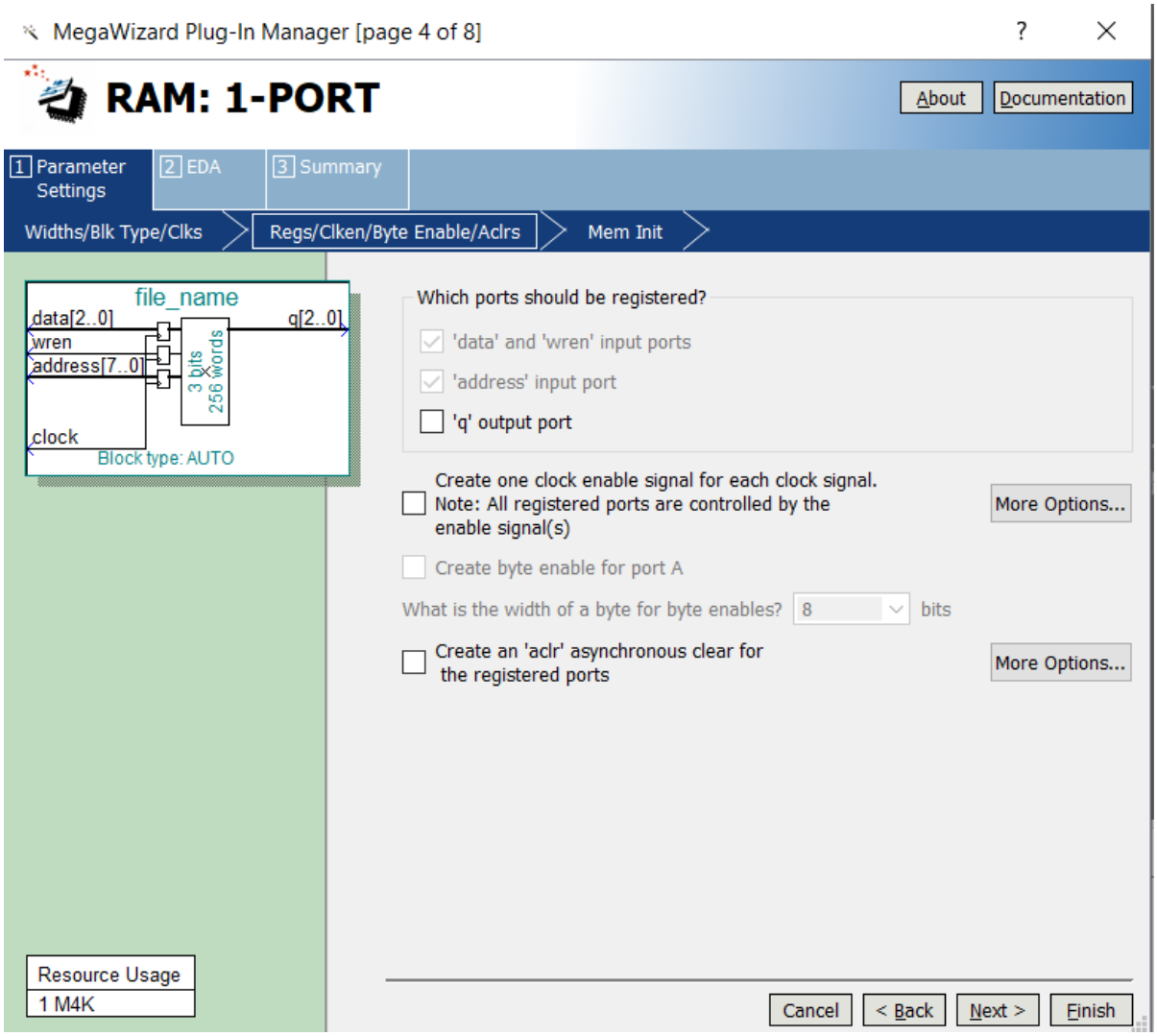


Figure A.6: Step6*

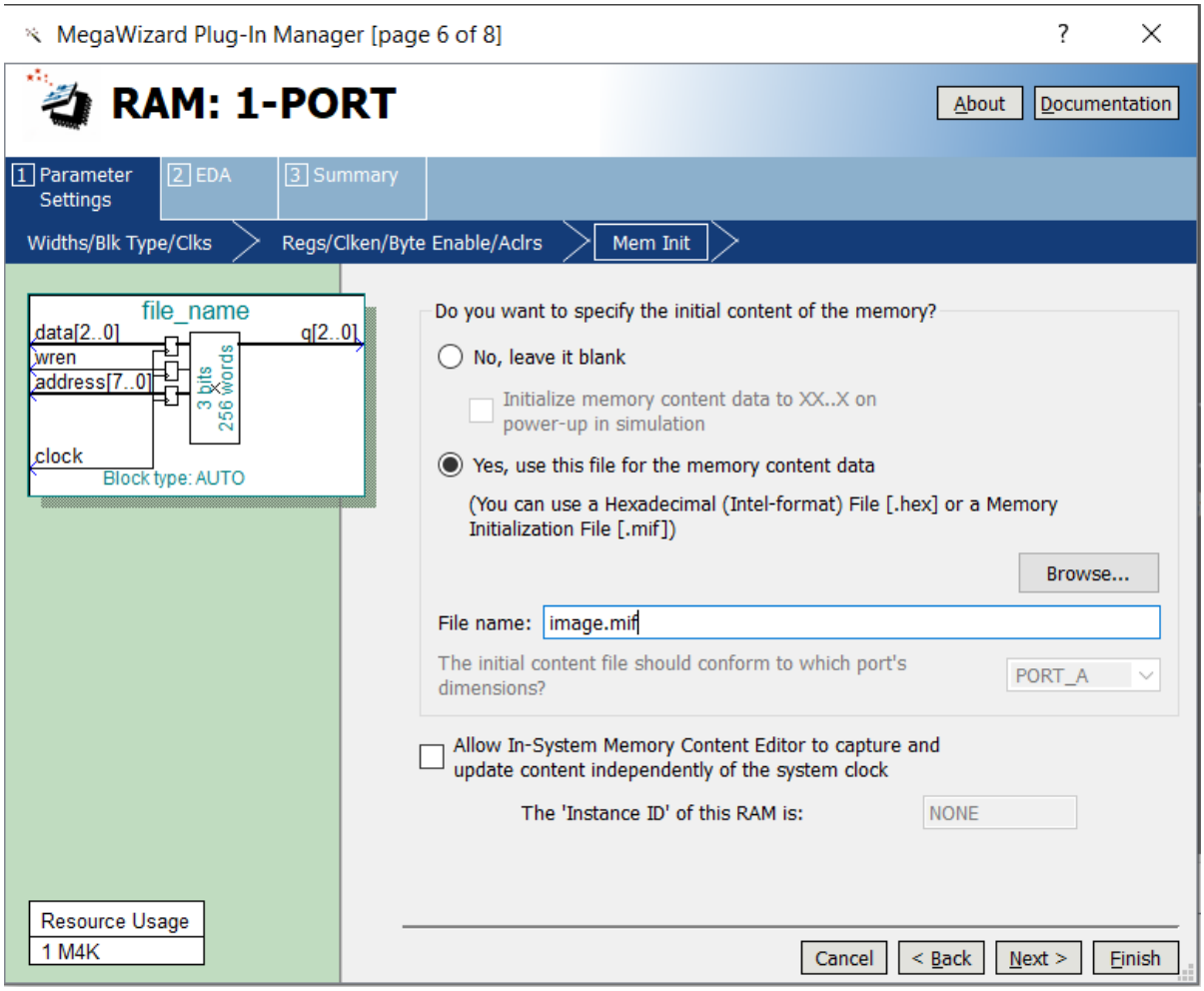


Figure A.7: Step7

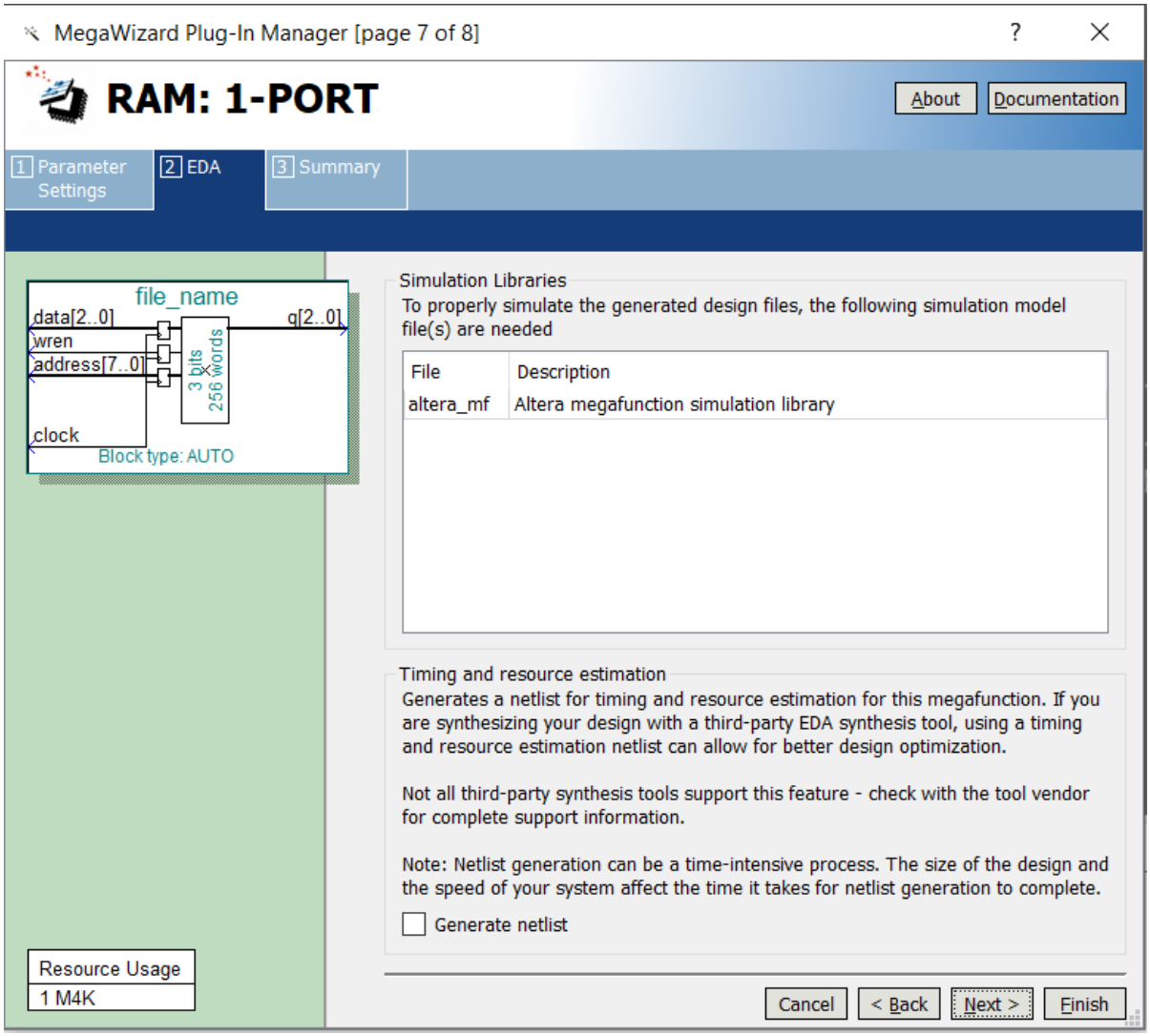


Figure A.8: Step8

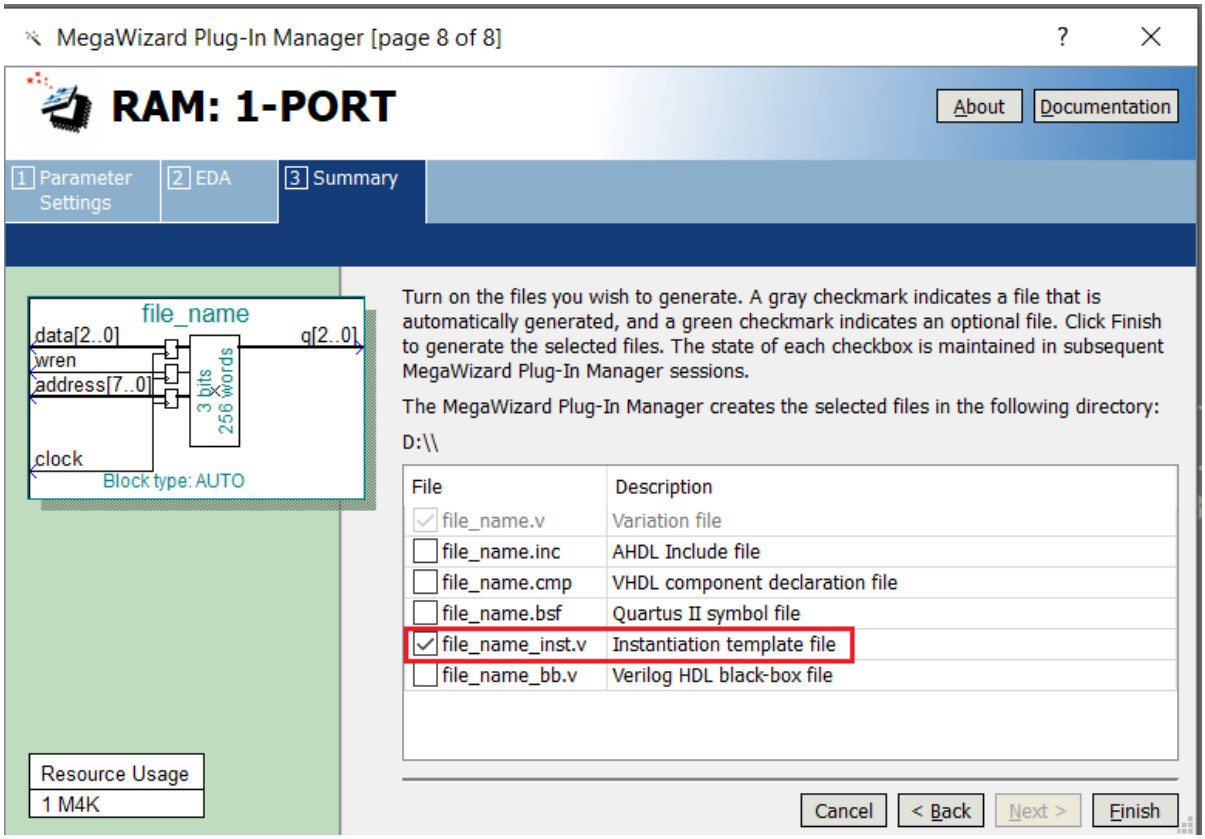


Figure A.9: Step9

Department of Electrical Engineering