

Digital Academy

Lab 6

Generating images in a VGA monitor

In this lab, you will use a VGA controller implemented in a FPGA to generate images in a VGA monitor. Also, you will create a very simple encryption algorithm with your own code.

Figure 1 shows the system implemented controlling a VGA monitor. The image on the left is the original image with original colors. The image on the right is the encrypted image. The user has to find the 3-bit code that produces the original colors in the image in the right. The user's code is inputted to your design using the push buttons 3, 2, and 1. Every time the user fails with the code, the colors of the encrypted image will change its colors.



Figure 1. Generating images in the VGA monitor lab.

You will need to design the encryption block (described below) and connect all the necessary blocks each other to generate the complete system.

Follow the next steps for the design. Use the same names for all the inputs, outputs and signals.

1. Download the files:

- reader.vhd
- vga_controller.vhd
- mux.vhd
- memory_coe.coe
- main_ucf.ucf

And copy them in your local hard drive.

2. Create a new project **lab_vga**. In the window for creating a **New Source**, click on **Next**. In the next window, click on **Add Source** and look for the 3 previous VHDL codes (.vhd files). Check on the option **Copy to Project**.

3. Create a new VHDL text file and call it **encryption**. In this file, you will design your encrypting system for the image. The goal is to choose a code of 3 bits that will be fixed, e.g. “**101**”. This code will be compared with the user input and, for each bit, if the user code is right, the value of the output pixel will be the same as the value of the input pixel, otherwise, the output pixel will be the negative of the input pixel. The next truth table depicts this function:

Fixed code – only one bit (pw)	User input (U)	Input pixel (P)	Output pixel (O)
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Design the encrypting system for the 3 bits using **sum-of-products**. Use the next names for the ports:

- U – User 3-bit input. Since it has 3 bits, it is a **std_logic_vector**.
- P – Pixel 3-bit input. Again, since it has 3 bits, it is a **std_logic_vector**.
- O – Pixel 3-bit output.

Since the code/password is fixed, it is not an external port. You must define it inside your VHDL code using '1' or '0'. For this, create a **signal** called **pw**:

```
signal pw : std_logic_vector(2 downto 0) := "101";
```

After you finish with your VHDL code, add it to your project and click on **Check Syntax** in the **Processes** window.

4. Creating a memory file.

In this step, you will create a memory block. This memory will be placed in the internal memory of the FPGA.

a. Click on **Create New Source**.

b. Highlight on **IP (Coregen & Architecture Wizard)** in your left and call your source **mymemory** (Figure 2). Click on **Next**.

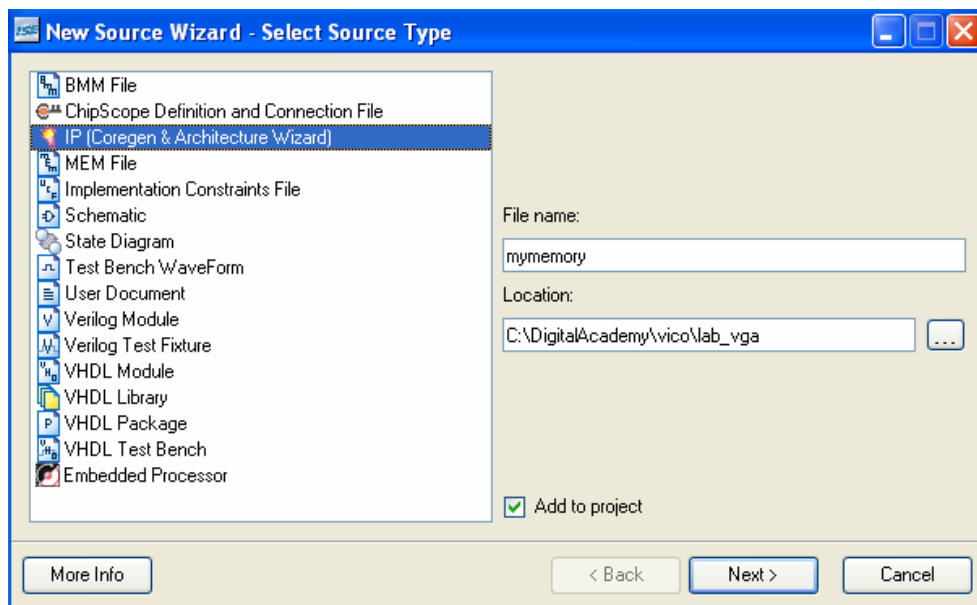


Figure 2. New Source Wizard.

c. Select **Single Port Block Memory v6.2** in the submenu **RAMs & ROMs** under the **Memories & Storage Elements** option (Figure 3). Select **Next**, then **Finish**.

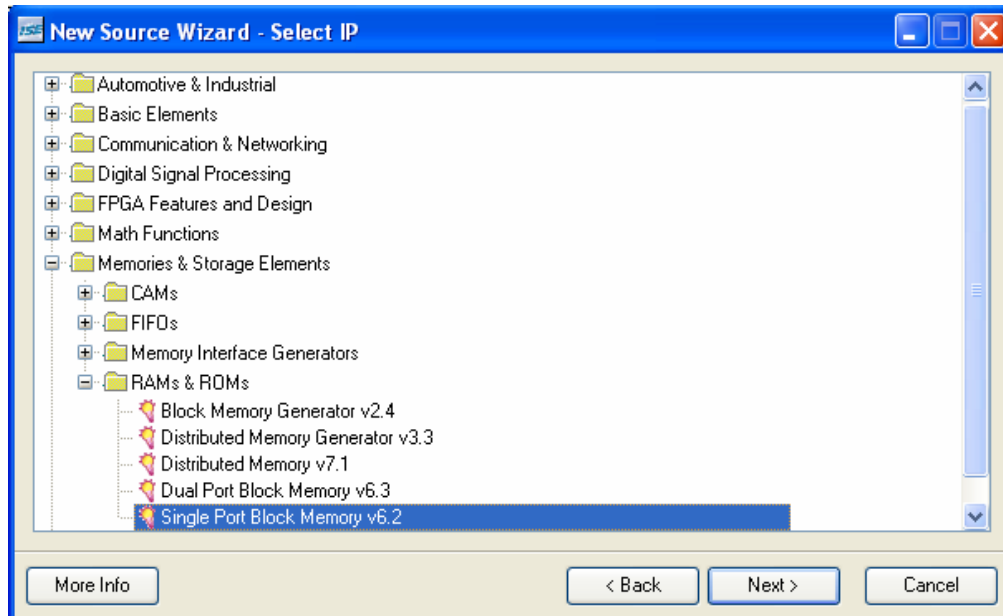


Figure 3. IP selection.

d. In the next window you will select the characteristics of your memory (Figure 4). Since you will not store data in your memory, choose the option **Read Only** under **Port Configuration**. The image that we will use has 224 rows and 224 columns (224x224 image) and we will use only 1 bit per color. Since the VGA controller use three main colors (Red, Green and Blue), the **width** of our memory is **3** and the **depth** is $224 \times 224 = 50176$ (50176 possible locations). Click on **Next** three times until you get the **Page 4 of 4** (Figure 5).

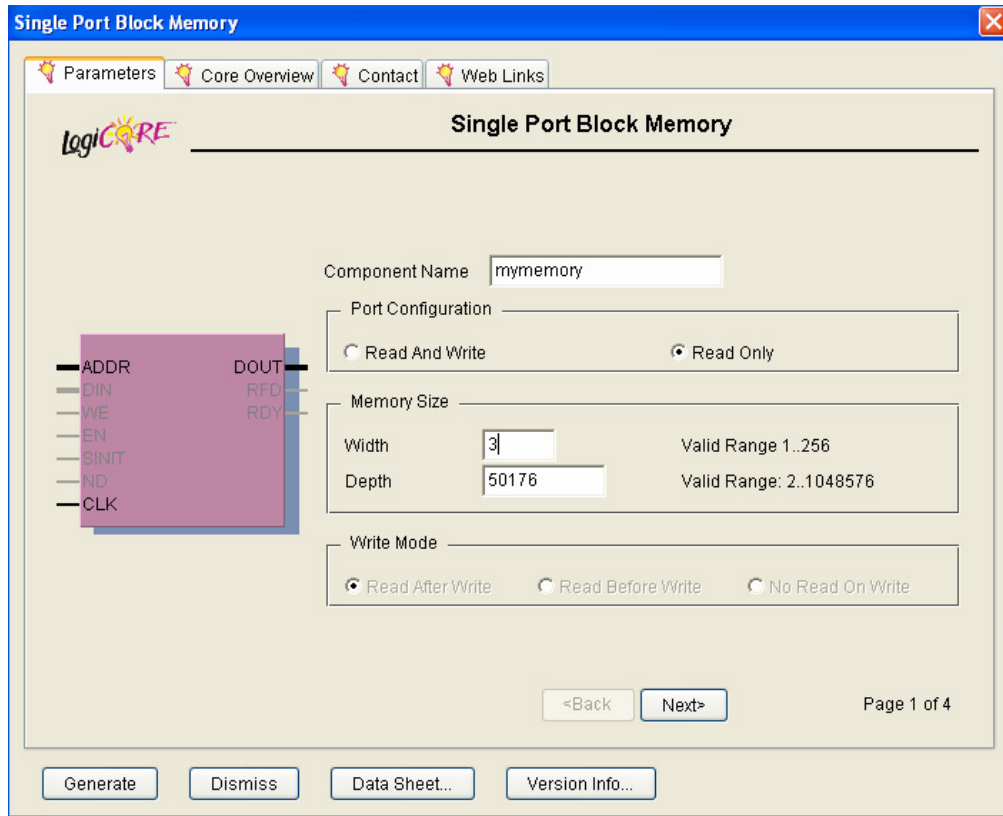


Figure 4. Single Port Block Memory.

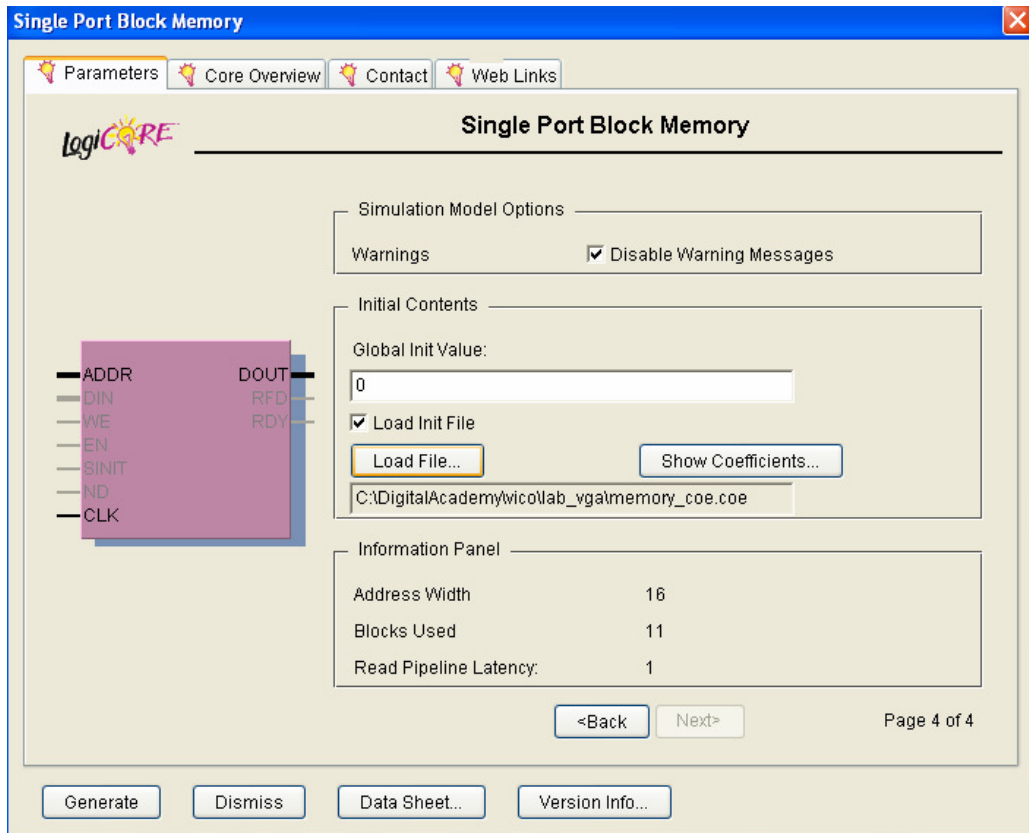


Figure 5. Single Port Block Memory. Page 4 of 4.

e. In this window, we will load the memory with the values of our image (Figure 5). The file **memory_coe.coe** has all the values stored in the format that Xilinx uses for its memories. Check the option **Load Init File** and then click on **Load File**. Look for the file **memory_coe.coe** and open it. You will have loaded the memory values in your memory (Figure 5).

f. Click on **Generate** to create the block. This process could take some minutes. You will know that Xilinx have finished with the block generation when you have the source added to your project (Figure 6).

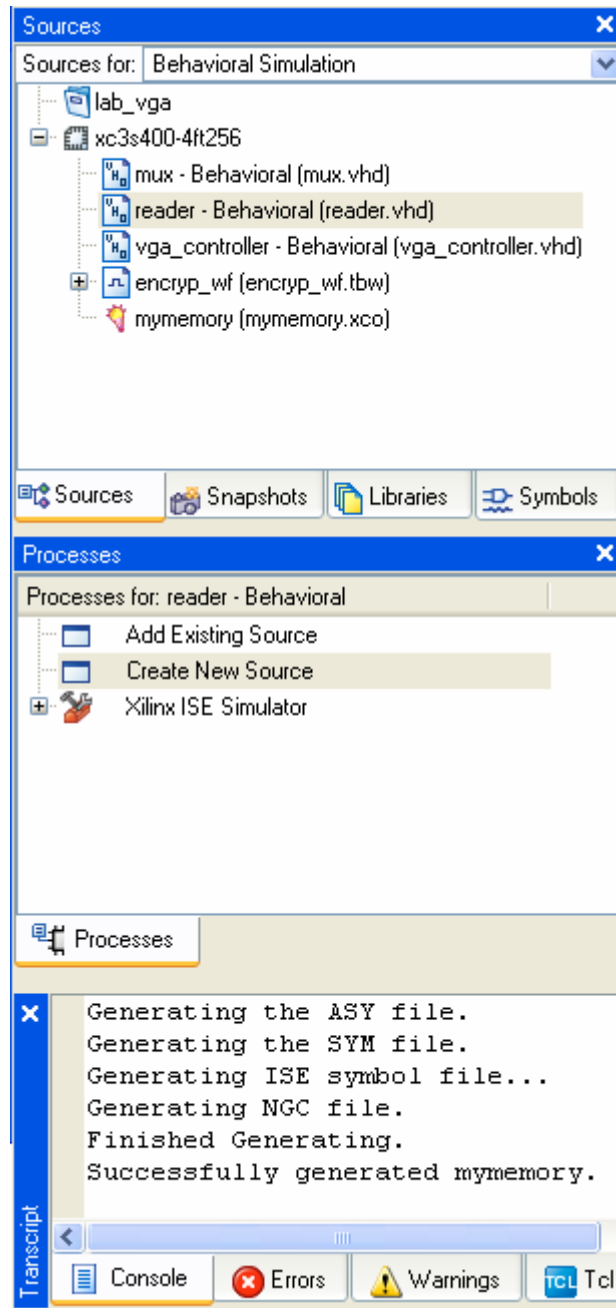


Figure 6. Generation of the memory.

g. If you want to see the VHDL of your memory, highlight the source **mymemory** in **Sources** and double click on **View HDL Functional Model** (Figure 7). **Don't change anything in this code!**

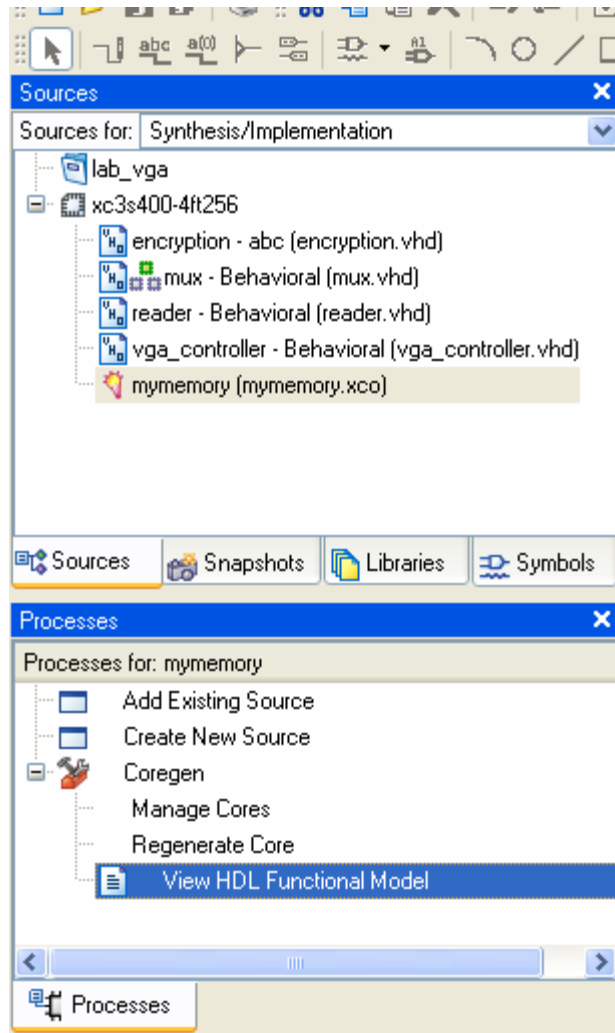


Figure 7. View HDL source of the memory.

5. It's time to connect all our blocks in our main system. **Create a new VHDL file** and call it **main.vhd**. The **inputs** of your system will be:

- clk – Clock signal, port A8 of the FPGA.
- reset – Push button 0 (BTN0)
- U – 3-bit user input password.
 - Bit 0 – BTN1.
 - Bit 1 – BTN2.
 - Bit 2 – BTN3.

Outputs:

- hsync – Horizontal synchronism of the VGA. Port R6 of the FPGA.
- vsync – Vertical synchronism of the VGA. Port R7.
- VGA_out_red – Red signal to the FPGA.
 - Bit 0: P16.

- Bit 1: P15.
 - Bit 2: T7.
 - Bit 3: R5.
- VGA_out_green – Green signal.
 - Bit 0: N15.
 - Bit 1: J16.
 - Bit 2: K16.
 - Bit 3: K15.
- VGA_out_blue – Blue signal.
 - Bit 0: L15.
 - Bit 1: M16.
 - Bit 2: M15.
 - Bit 3: N16.

6. Connecting the blocks to the system.

First, declare all the **components** you will use (as you did in the lab of the 4-bit counter with the 7-segment decoder): **reader**, **mux**, **vga_controller**, **encryption**, and **mymemory**. Remember that the declaration of the components is done **between the initialization of the architecture and its begin**. To know the ports of **mymemory**, you need to open its VHDL code as you did it before.

All the internal connections among components are done using **signals** (again, remember the previous lab). The connections will be:

i. Component: *reader*

Inputs

- **clk** → **clk** of the system.
- **reset** → **reset** of the system.
- **row** → signal **rows** (same type and size as **row**).
- **col** → signal **cols** (same type and size as **col**).
- **datain** → signal **datamem** (same type and size as **datain**).

Outputs

- **ennormal** → signal **enormal** (same type as **ennormal**).
- **enencryp** → signal **eencryp** (same type as **enencryp**).
- **addr** → signal **address** (same type and size as **addr**).
- **dataout** → signal **dataread** (same type and size as **dataout**).

ii. Component: *mymemory*

Inputs

- **clk** → **clk** of the system.
- **addr** → signal **address** (*the same one that you connected to reader*).

Outputs

- **dout** → signal **datamem** (*the same one that you connected to reader*).

iii. Component: *encryption*

Inputs

- **U** → **U** of the system.
- **P** → signal **dataread**.

Outputs

- **O** → signal **dataencryp**.

iv. Component: *mux*

Inputs

- **clk** → **clk** of the system.
- **ennormal** → signal **enormal**.
- **enencryp** → signal **eencryp**.
- **data_normal** → signal **dataread**.
- **data_encryp** → signal **dataencryp**.

Outputs

- **dataout** → signal **colors**.

v. Component: *vga_controller* (*this mapping is written after the next inputs/outputs declaration*)

Inputs

- **clk** → **clk** of the system.
- **reset** → **reset** of the system.
- **VGA_in_red(0)** → signal **colors(2)**. The bits 1 and 2 of **VGA_in_red** have to be connected to a constant '0'.

- **VGA_in_green(0)** → signal **colors(1)**. The bits 1 and 2 of **VGA_in_green** have to be connected to a constant '0'.
- **VGA_in_blue(0)** → signal **colors(0)**. The bits 1 and 2 of **VGA_in_blue** have to be connected to a constant '0'.

Outputs

- **col** → signal **cols**.
- **row** → signal **rows**.
- **VGA_out_red** → **VGA_out_red** of the system.
- **VGA_out_green** → **VGA_out_green** of the system.
- **VGA_out_blue** → **VGA_out_blue** of the system.
- **hsync** → **hsync** of the system.
- **vsync** → **vsync** of the system.

Mapping declaration of the **vga_controller**:

```
myvga: vga_controller
  port map (
    clk => clk,
    reset => reset,
    VGA_in_red(0) => colors(2),
    VGA_in_red(3 downto 1) => "000",
    VGA_in_green(0) => colors(1),
    VGA_in_green(3 downto 1) => "000",
    VGA_in_blue(0) => colors(0),
    VGA_in_blue(3 downto 1) => "000",
    col => cols,
    row => rows,
    VGA_out_red => VGA_out_red,
    VGA_out_green => VGA_out_green,
    VGA_out_blue => VGA_out_blue,
    hsync => hsync,
    vsync => vsync
  );
```

7. **Synthesize** your design.

8. **Adding the UCF file.** Instead of creating the UCF file as you did in previous labs, you will **add the UCF file** that you downloaded from the web. Go to **Adding Existing Source** and select the file **main_ucf.ucf**. **You must have the same names for the inputs and outputs as those described here.**

9. Implement your design into the FPGA.