Exercise 9:

a) Graphic Output

The entity *graphic_output* groups the previous three modules together and provides the connections between them. It is also a multiplexer for the color component outputs *o_red*, *o_green* and *o_blue*. These outputs are fed from the currently selected register or set to 0 when the *in_active_region* signal is not set by the *sync_pulse_generator*.
The component declaration of the modules utilized should not be placed inside the architecture of *graphic_output*, but instead in a separate package file named *graphic_output_package.vhd*. This file will then be included in the *graphic_output* module.

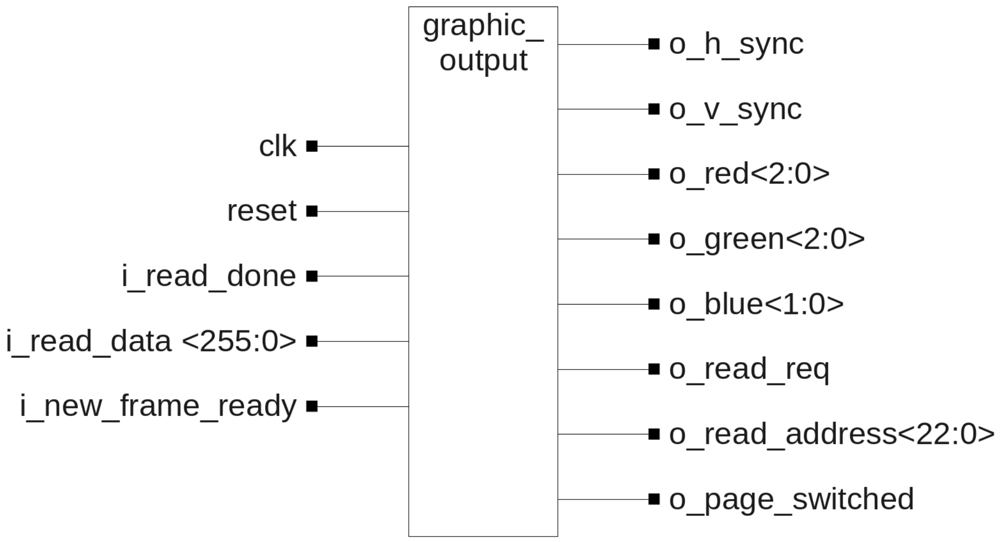


Figure 1: graphic_output

Port declaration:

● *clk*	: in std_logic
● *reset*	: in std_logic
● *i_read_done*	: in std_logic
● *i_read_data*	: in std_logic_vector (255 downto 0)
● *i_new_frame_ready*	: in std_logic
● *o_h_sync*	: out std_logic
● *o_v_sync*	: out std_logic
● *o_red*	: out std_logic_vector(2 downto 0)
● *o_green*	: out std_logic_vector(2 downto 0)
● *o_blue*	: out std_logic_vector(1 downto 0)
● *o_read_req*	: out std_logic
● *o_read_address*	: out std_logic_vector (22 downto 0)
● *o_page_switched*	: out std_logic

Generic declaration:
The same generics as in the module *sync_pulse_generator* are used.

Your first task is to design the entity and architecture of the module *graphic_output* and to create the package file. Have a look at the schematic of the graphic_output module in the overview and think about all the interconnections before writing any VHDL code.

Since you should have previously tested the functionality of the sub-modules of graphic_output, you must now only test the interconnection of these modules and their interaction. Most importantly, the multiplexing between the two buffers must be tested. Therefore, you should have a detailed look at the signals of the two buffers and the three data outputs.
This time, no vector file with stimuli for the testbench is provided, so you must create your own vector file called **input_data.vec**. This file should feed all the inputs of *graphic_output* with stimuli. Try to create a reasonable test case by carefully examining the function of the module before writing the vector file!

After a successful simulation, the entire module and all its sub-modules will be integrated into a test project, implemented on the FPGA and tested on a real monitor.

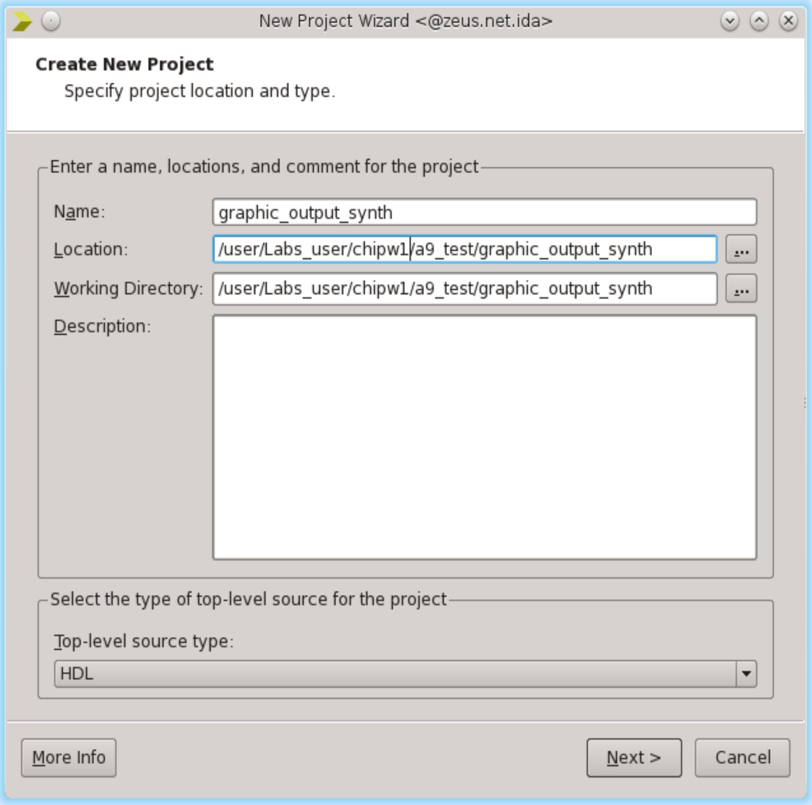
b) Synthesis of graphic_output

Working with Xilinx ISE

To synthesize your code, we use the FPGA design software Xilinx ISE.
To setup Xilinx ISE (in this lab, we use Xilinx ISE14.6) to run on the IDA application servers, run:

```
$ setup-lm xilinx ise14.6
$ ise &
```

ISE will start with the last project opened. If no project has been created yet, click on **File** => ***New Project***. Enter a project name and change the path to your exercise directory.


New Project Wizard <@zeus.net.ida>
Create New Project
Specify project location and type.
Enter a name, locations, and comment for the project

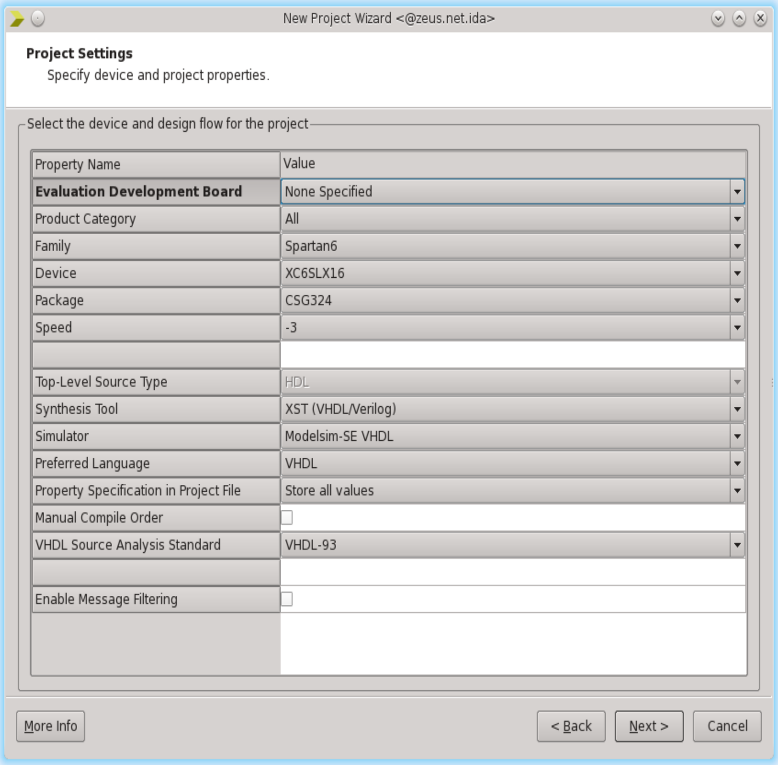
Name: graphic_output_synth
Location: /user/Labs_user/chipw1/a9_test/graphic_output_synth
Working Directory: /user/Labs_user/chipw1/a9_test/graphic_output_synth
Description:

Select the type of top-level source for the project

Top-level source type: HDL

Figure 2: Project Wizard – Create new project

Now enter all of the required data for your FPGA and click ***Next***. Note that the device properties vary depending on the board you use. You can find the correct properties on your device. For the Nexys3 boards it should be: Device=XC6SLX16, Package=CSG324 and Speed=-3


New Project Wizard <@zeus.net.ida>

Project Settings

Specify device and project properties.

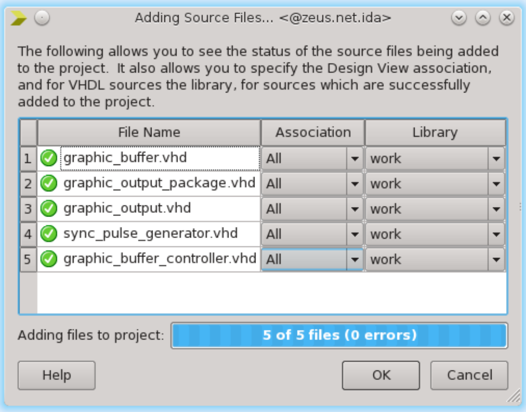
Select the device and design flow for the project

Property Name	Value
Evaluation Development Board	None Specified
Product Category	All
Family	Spartan6
Device	XC6SLX16
Package	CSG324
Speed	-3
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	Modelsim-SE VHDL
Preferred Language	VHDL
Property Specification in Project File	Store all values
Manual Compile Order	☐
VHDL Source Analysis Standard	VHDL-93
Enable Message Filtering	☐

Figure 3: Project Wizard - Device properties

After finishing the project setup, you should see a plain project without any sources. First you have to add some sources of your previous exercises. In the main menu, select ***Project → Add Source*** and add the following files:

- graphic_buffer.vhd
- graphic_buffer_controller.vhd
- sync_pulse_generator.vhd
- graphic_output.vhd
- graphic_output_package.vhd


Adding Source Files... <@zeus.net.ida>

The following allows you to see the status of the source files being added to the project. It also allows you to specify the Design View association, and for VHDL sources the library, for sources which are successfully added to the project.

	File Name	Association	Library
1	✓ graphic_buffer.vhd	All	work
2	✓ graphic_output_package.vhd	All	work
3	✓ graphic_output.vhd	All	work
4	✓ sync_pulse_generator.vhd	All	work
5	✓ graphic_buffer_controller.vhd	All	work

Adding files to project: 5 of 5 files (0 errors)

Figure 4: Project Wizard - Add sources

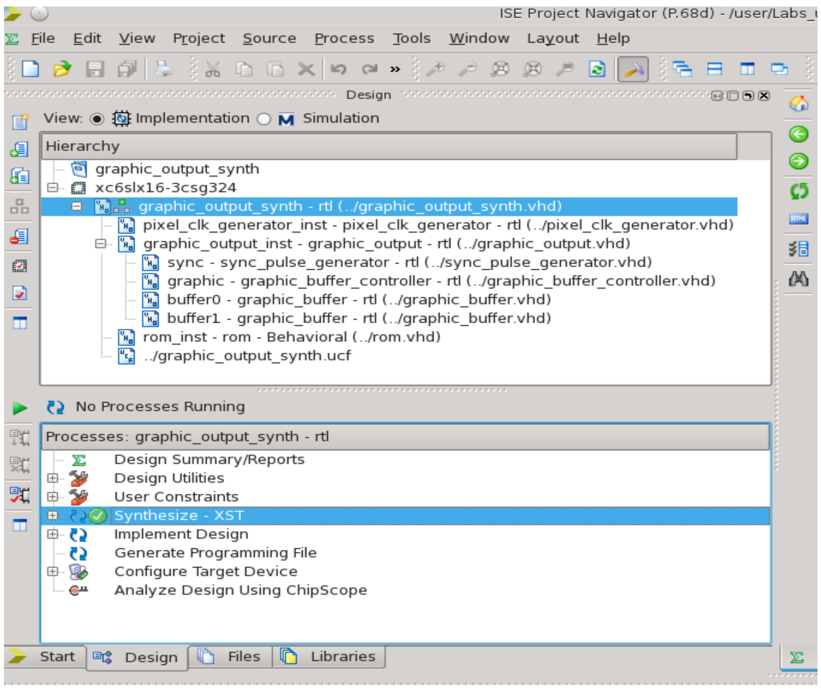
You should not use the “Add Copy of Source” option, so that you do not have to replace the file in the project directory once it is changed in its destination directory. Xilinx ISE will do a quick syntax check and if everything is fine, it will present you some green check marks.

To implement the complete test project, you must also add the following files:

- pixel_clk_generator.vhd
- rom.vhd
- graphic_output_synth.vhd
- graphic_output_synth.ucf

You can find these files in the folder of this exercise. Have a look at the *.ucf file, which contains the pin assignment of your FPGA. This file is known as a “constraints” file and is necessary to map the signals in your design to the physical FPGA inputs and outputs.

Now you see the Project Navigator window with all files (in fact you see the entities or instances of entities). Missing entities are marked with a question mark. The top entity is marked with three small squares. In most cases, the top entity is correctly chosen by ISE. If you have to change it, click on the entity you want to move to the top and right-click on it. Then select ***Set as Top Module***. To add files, right click in the Project Navigator and select either ***Add Source...*** for existing files or ***New Source...*** to create a new source file.



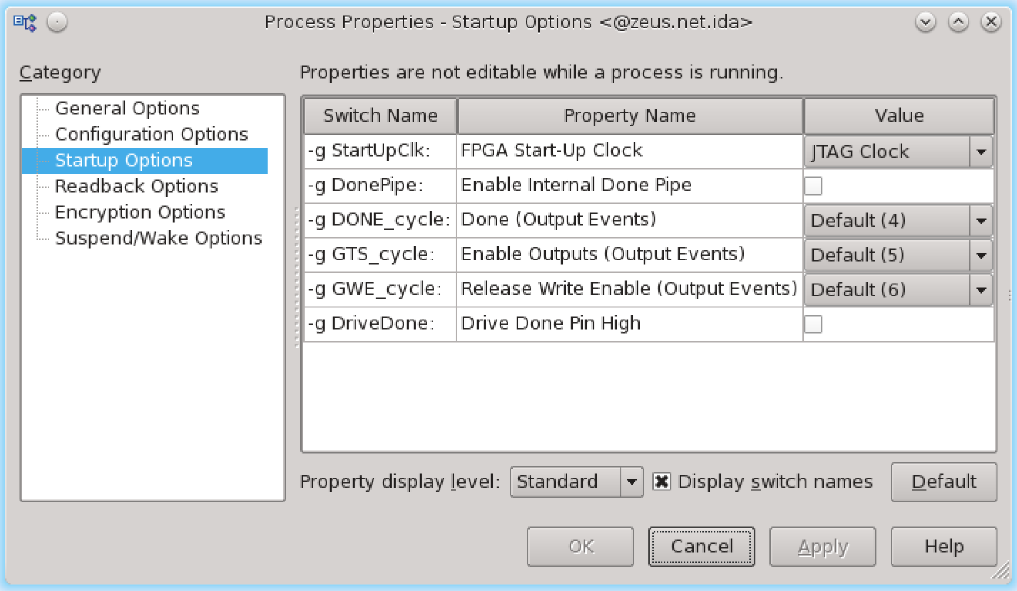
Hierarchy
graphic_output_synth
xc6slx16-3csg324
graphic_output_synth - rtl (../graphic_output_synth.vhd)
pixel_clk_generator_inst - pixel_clk_generator - rtl (../pixel_clk_generator.vhd)
graphic_output_inst - graphic_output - rtl (../graphic_output.vhd)
sync - sync_pulse_generator - rtl (../sync_pulse_generator.vhd)
graphic - graphic_buffer_controller - rtl (../graphic_buffer_controller.vhd)
buffer0 - graphic_buffer - rtl (../graphic_buffer.vhd)
buffer1 - graphic_buffer - rtl (../graphic_buffer.vhd)
rom_inst - rom - Behavioral (../rom.vhd)
../graphic_output_synth.ucf

No Processes Running

Processes: graphic_output_synth - rtl
Design Summary/Reports
Design Utilities
User Constraints
Synthesize - XST
Implement Design
Generate Programming File
Configure Target Device
Analyze Design Using ChipScope

Figure 5: Project Navigator window

Once you have added all files, you must right click on ***Generate Programming File*** in the Processes window and select ***Properties***. Select ***Startup Options***, and change FPGA Start-Up Clock to JTAG Clock.



Process Properties - Startup Options <@zeus.net.ida>

Category

- General Options
- Configuration Options
- Startup Options
- Readback Options
- Encryption Options
- Suspend/Wake Options

Properties are not editable while a process is running.

Switch Name	Property Name	Value
-g StartUpClk:	FPGA Start-Up Clock	JTAG Clock
-g DonePipe:	Enable Internal Done Pipe	☐
-g DONE_cycle:	Done (Output Events)	Default (4)
-g GTS_cycle:	Enable Outputs (Output Events)	Default (5)
-g GWE_cycle:	Release Write Enable (Output Events)	Default (6)
-g DriveDone:	Drive Done Pin High	☐

Property display level: Standard ☒ Display switch names

Figure 6: Process properties

To generate a programming file, simply double-click on ***Generate Programming File***, or you can run all three necessary steps one after another by clicking on them separately. At the end you should have a file named [*project name*].bit in your project directory. This FPGA bitstream file is then used to program the FPGA.

Working with Digilent Adept tools

Digilent Adept is used to program the FPGA. First thing to do is connect the FPGA board to the PC with the USB cable plugged into the micro-usb socket called "USB PROG" and connect the monitor via VGA. To use the programming tool, open a terminal window and navigate to your project directory where your bit-file is located. Check the connection by issuing:

```
        $ djtgcfg enum
```

The output should look like:

```
Found 1 device(s)

Device: Nexys3
        Product Name:   Nexys3
        User Name:      Nexys3
        Serial Number:  210182475328
```

Then initialize the JTAG chain with:

```
        $ djtgcfg init -d Nexys3
```

You will see something like:

```
Initializing scan chain...
Found Device ID: 34002093

Found 1 device(s):
        Device 0: XC6SLX16
```

To program the FPGA run (insert the correct name of your bit-file):

```
        $ djtgcfg prog -d Nexys3 -i 0 -f [project name].bit
```

The result should be:

```
Programming device. Do not touch your board. This may take a few minutes...
Programming succeeded.
```

If the programming of the FPGA was successful, your monitor should display a test screen which alternates between Figure 7 and Figure 8. Otherwise, one of your modules is not functioning correctly.

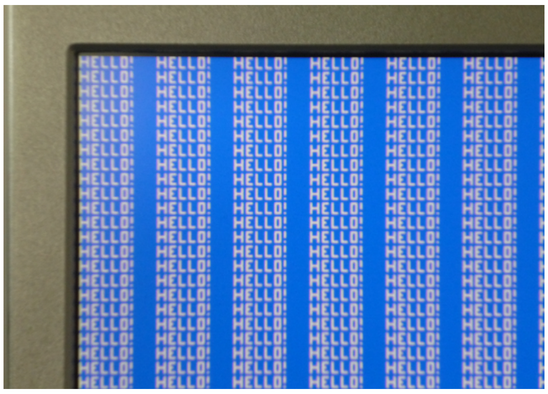


Figure 7: First test screen

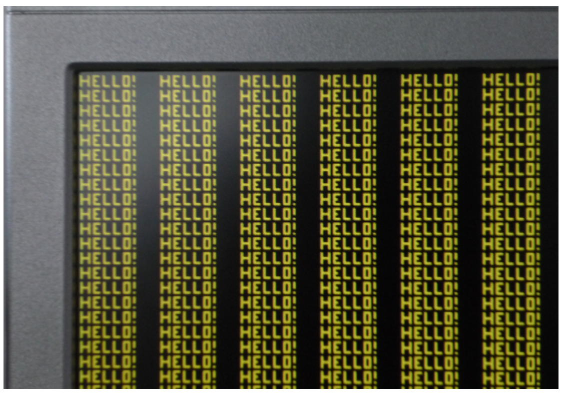


Figure 8: Second test screen