Background Frame Buffer

This lab aims to implement a memory that will later be used for the background frame buffer in the Pong game, as shown in Figure 1. For this, you will reuse your design from Lab 5, with minor modifications. Before doing any practical work for this lab, please create a new independent project and import the design files from Lab 5. Use the directory structure from the project description PDF.

Hand-in instructions: Please note you should not hand in any report on this lab, instead, this lab will form a part of the final project.

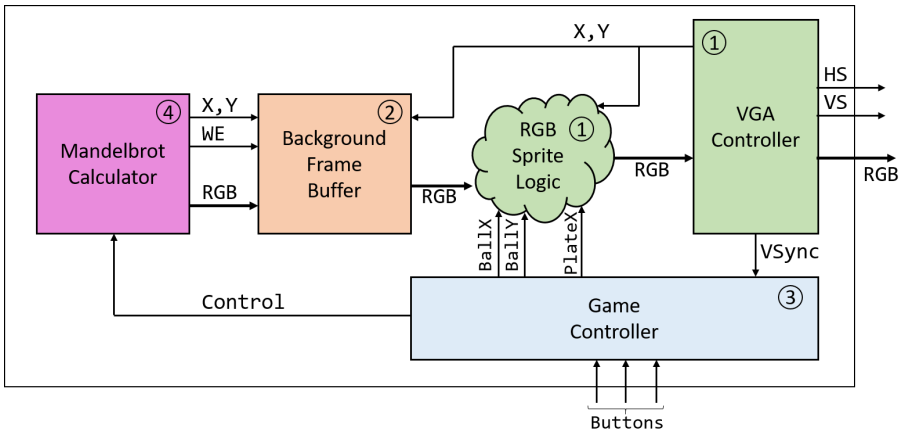

Figure 1: Full project schematic with background frame buffer receiving X- and Y-coordinates from the VGA controller to output the correct pixel values for the PONG background.

Important information ⚠

Please always check these things before you start a lab or when you have issues!

FPGA:

- Remember to use the reset button/switch on the FPGA to reset your design and to turn this off again if using a switch.
- Connect the FPGA board's Ethernet port to a computer, router, or any other device with an Ethernet port that can power the Ethernet chip on the FPGA board (no internet connection is needed). See the Lab 2 manual for an explanation.

VHDL:

- For combinational logic, use `process(all)`. Do not write your own sensitivity lists! Please remember to change files using `process(all)` to VHDL 2008. This is done by selecting the file in the `Source` tab. Look at the `Source File Properties` tab and change the type to `VHDL 2008` by clicking on the 3 dots in the `Type` field.
- For defining registers, use the clocked process style `process(CLKxCI, RSTxRI)`. Do not define combinational logic like `CNTxDP <= CNTxDP + AxDI` inside a clocked process! See Task 4 in Exercise 3 and its solution for further explanation.

- Never write to the same signal in multiple concurrent statements! This means that if you assign to a signal in a process that signal can only be assigned to in that process and nowhere else. The only place you are allowed to assign multiple times to a signal is inside the (single) process where it is assigned to.

Virtual machines:

- EDA server users must start Vivado with `vivado -source load_board_files.tcl` as described in Lab 1. If you do not see the board files in the Vivado GUI, you have likely used the command with a spelling error or something similar.

Windows users:

- Avoid spaces and special characters in your filepaths! Vivado projects can become corrupted if you have spaces and special characters in your filepaths.
- You may have to disable your antivirus tool before running simulations in Vivado.

For common questions/hints to this lab, please see the last page of this document which contains various hints and best-practices.

Task 1: Modifying Lab 5 Code

In this first task, we will modify your implementation from Lab 5. Download the provided `.zip` file from Moodle and create a new directory for Lab 6. This time, we provide you with the following files (under the `src` directory):

- `vga_controller.vhdl`: Template for the VGA controller, slightly modified from Lab 5.
- `vga_controller_top.vhdl`: Top-level containing the component instantiations and declarations for the clock circuit generator, memory generator and VGA controller. This file is not used for Task 1.
- `dsd_prj_pkg.vhdl`: Package containing various constants.

The remaining files inside the `src` directory are for Task 2. Furthermore, we have the `.xdc` file and the board-files (when using the servers) as usual.

You should then proceed as follows:

1. Familiarize yourself with the content of the files.
2. Copy your code from Lab 5 into the Lab 6 directory and modify your code from Lab 5 with the new files we have provided. For this, you only have to make small changes, like adding the new constants from `dsd_prj_pkg.vhdl`.

Task 2: Adding Memory

In this task, we now consider adding the memory and initializing it with an image. The additional files under the `src` directory are:

- `epfl.bmp`: Image of EPFL in `.bmp` file format.
- `epfl.coe`: Memory initialization file used to initialize the memory on the FPGA with the image of EPFL.
- `ImageToCOE.m`: MATLAB function to generate a `.coe` file from a `.bmp` file. Note that this function down-scales the image to 256×192 as the image will otherwise not fit on the FPGA. We use this file in Task 3.

You should then proceed as follows:

1. Familiarize yourself with the content of the above files.
2. Add the Lab 6 top-level file `vga_controller_top.vhdl` to your Vivado project.
3. This top-level instantiates a memory generator, which you have to add to the project similarly to the clock generator from Lab 5. There is a video on Moodle for today (This link.) which shows how to add the memory generator. Note that you also have to add the clock generator to this project. When initializing the memory, you should use the `epfl.coe` file already provided. Please note that depending on the settings you pick when creating the memory generator, you may have differences compared to the template we provide. Figure 2 shows you how to see the instantiation template from Vivado.
4. Modify `vga_controller_top.vhdl` such that the `RdAddrBxD` signal, which represents the read address to the memory, uses the X- and Y-coordinates outputted from the VGA controller to index into the memory. Please remember that the VGA controller uses a 1024×768 screen size, whereas the image is downsampled to 256×192. Therefore, you have to skip some of the outputted coordinates. The specifics of how to do this mapping and which coordinates to skip is left for the reader.
5. Verify that your design is working by programming the FPGA with the bit-file. You should see an image as shown in Figure 3.

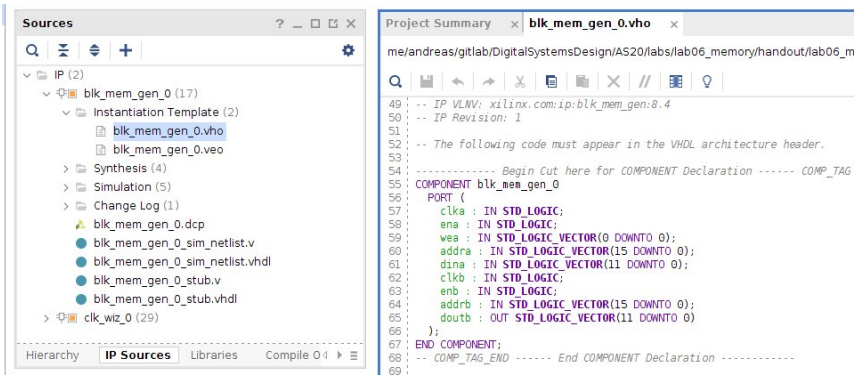

Figure 2: You can see the instantiation template under the **IP sources tab** in the Sources window. Note that this memory has been created with the enable on port B, which you do not need!

Figure 3: A successful implementation of a VGA controller with the described modification for the top-level should show an image of EPFL. In some cases there may be black borders depending on your screen size. The use of an HDMI adapter may also cause some flicker on the screen which is acceptable.

Task 3: Custom Images

Finally, you should try to use the `ImageToCOE` function with an image of your choosing and initialize the memory with this image.

Common Questions

Common questions/remarks for this lab are:

- **How do I create a VHDL file?** After creating a project in Vivado you can click `File` → `Add Sources` → `Add or create design sources`. Alternatively, just use your normal code editor and create new files with the `.vhdl` (recommended) or `.vhd` extensions.
- **How do I resolve the warning 'The PS7 cell must be used in this Zynq design ...?** This warning can be safely ignored as it's unrelated to what we do on the FPGA.
- **How do I resolve the error 'Unconstrained Logical Port'?** While VHDL itself is case-insensitive, **`.xdc` constraints are case sensitive** and your port names should match those in the `.xdc` file in case as well.
- **Outlook does not allow `.vhd` files:** You cannot send `.vhd` files in Outlook, try renaming to `.vhdl` as the `.vhd` extension is also used for **V**irtual **H**ard **D**isk on Windows.
- **Remember that order matters in processes!** Since the order of assignments are done sequentially in a process, meaning that in the example below `DxSO` is only assigned `AxSI and BxSI` and never `AxSI or BxSI`.

Listing 1: This implementation ignores the line `AxSI or BxSI` as it is always overwritten by the final assignment to `DxSO`.

```
process(all)
begin
  if (CxSO = '0') then
    DxSO <= AxSI or BxSI;
  end if;

  CxSO <= not AxSI;
  DxSO <= AxSI and BxSI;
end process;
```

- **Remember to separate the description of the flip-flops from the combinational logic!** Use a single process for updating the flip-flops and a separate process or concurrent assignments for updating the adder as shown below. This is really important! We also discuss this in Exercise 3.

Listing 2: VHDL code to show how to define flip-flops for a counter.

```
CNTxDN <= CNTxDP + 1; -- Increment outside clock-process

process(CLKxCI, RSTxRI)
begin
  if (RSTxRI = '1') then
    CNTxDP <= (others => '0');
  elsif CLKxCI'event and CLKxCI = '1' then
    CNTxDP <= CNTxDN;
  end if;
end process;
```

- **Remember to never write to the same signal in multiple concurrent statements!** When assigning to a signal in a process, you can only assign to that signal in that (single) process. The code below in Listing 3 shows the signal `CNTxDN` being assigned to in two different concurrent statements, which is not allowed. With this code, you will see an 'X' for `CNTxDN` in the waveform viewer. The solution is to put the default assignment in a process like shown in Listing 4.

Listing 3: VHDL code which shows how not to assign to a signal!

```
CNTxDN <= CNTxDP;

process(all)
begin
  if (In0xSI = '1') then
    CNTxDN <= CNTxDP + 1;
  end if;
end process;
```

- **Use default values in your process!** To avoid introducing errors in your code from missing assignments to signals, you should always use a default value for all signals assigned to in a `process(all)` when describing combinational logic as shown in Listing 4. This is done as the first thing in a process.

Listing 4: VHDL code which shows the assignment of a default value.

```
process(all)
begin
  -- Default values
  CNTxDN <= CNTxDP;
  AxD    <= (others => '0');
  BxD    <= (others => '0');

  -- Actual logic after default values
  if (In0xSI = '1') then
    CNTxDN <= CNTxDP + 1;
    AxD    <= In1xSI;
  elsif (In1xSI = '1') then
    CNTxDN <= CNTxDP - 1;
    BxD    <= In1xSI;
  end if;
end process;
```

- **I get a critical warning or error stating I have an unconnected pin?** The error may be because you generated the memory with the enable on the b port, but in the template, it is not there (hence the critical warning telling you it is not there). Depending on which settings you use in the block memory generation you can get different ports, but you can always check by looking at the `.vho` file as shown below. So, either you can remove it during the generation or change the top-level to have it, using the code shown below as a reference.

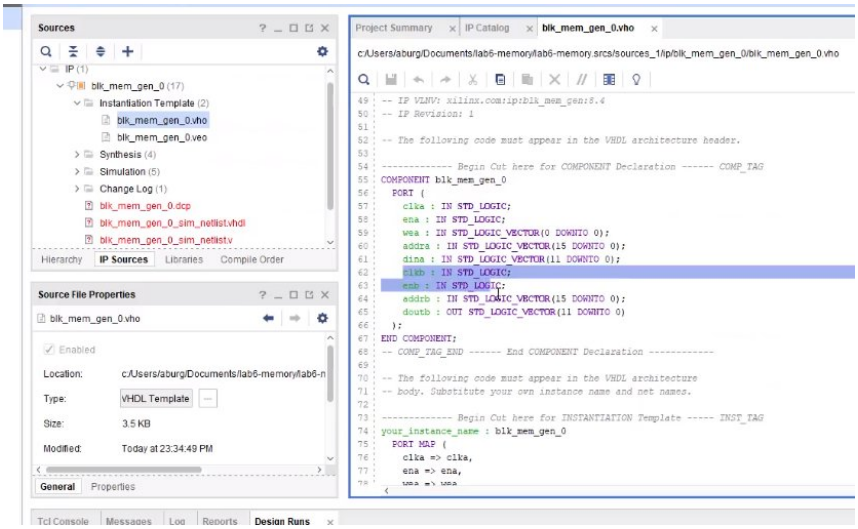

Figure 4: Example of memory with enable on port B.

- **I get a .veo (Verilog output) file for my memory instead of a .vho (VHDL output) file** This can happen if your project is setup to use Verilog as the target language instead of VHDL. You can change this in the settings as shown below. Then try to rerun the memory generation.

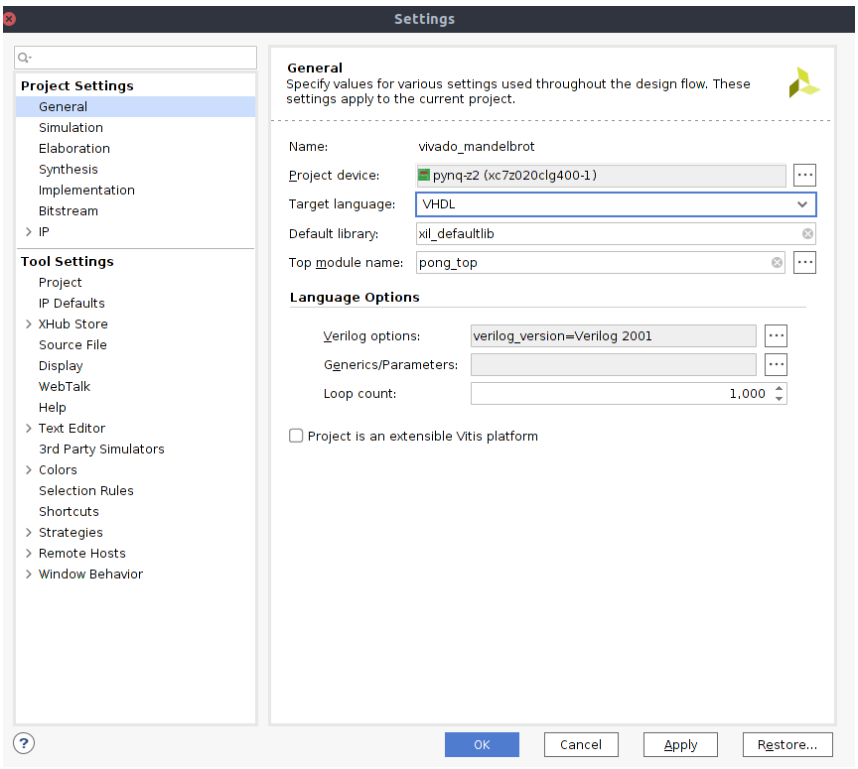

Figure 5: Vivado project settings for the target language.