# Lab 2: Implementing an adder circuit with VHDL

Bulat Khusainov and Eric Kerrigan

## Learning outcomes

By the end of this lab the student should be able to synthesize an integer addition circuit and implement it on the Zedboard.

## Files provided

| | |
|---|---|
| **lab2.vhd** | Adder circuit description |
| **lab2_constraints.xdc** | Circuit constraints |
| **spi_control.vhd*** | SPI control circuit |
| **delay.vhd*** | Time delay circuit |
| **encoder.vhd*** | Characters encoder |
| **oled_init.vhd*** | OLED display initialization circuit |
| **oled_print.vhd*** | OLED display printing circuit |
| **top_level_circuit.vhd*** | Top level circuit for the entire project |

Files denoted with (*) are used for visualizing the results on an organic LED (OLED) display, but not for the addition operation itself. For the purposes of this lab they can be considered as black box circuits.

## Target platform

In this lab we will use the following interfaces (Figure 1):

- JTAG/Debug for programming the platform.
- Slide switches to define the numbers for addition.
- The OLED display to check the output values.
- The middle push switch (denoted with P16 on the board) for resetting the circuit.

## Circuit

The circuit of interest (top_level_circuit.vhd) consists of three subcircuits (Figure 2):

- oled_init.vhd is responsible for initializing the OLED display. In this context *initializing* means turning the display on, setting brightness, etc. Note that this circuit is connected directly to one of the push buttons to allow the user to initialize the display manually.
- oled_print.vhd prints the result of the addition to the display.
- lab2.vhd is the core unit of the circuit that performs addition. It will be considered in the sections below. As can be seen from Figure 2, input numbers are set by slide switches.

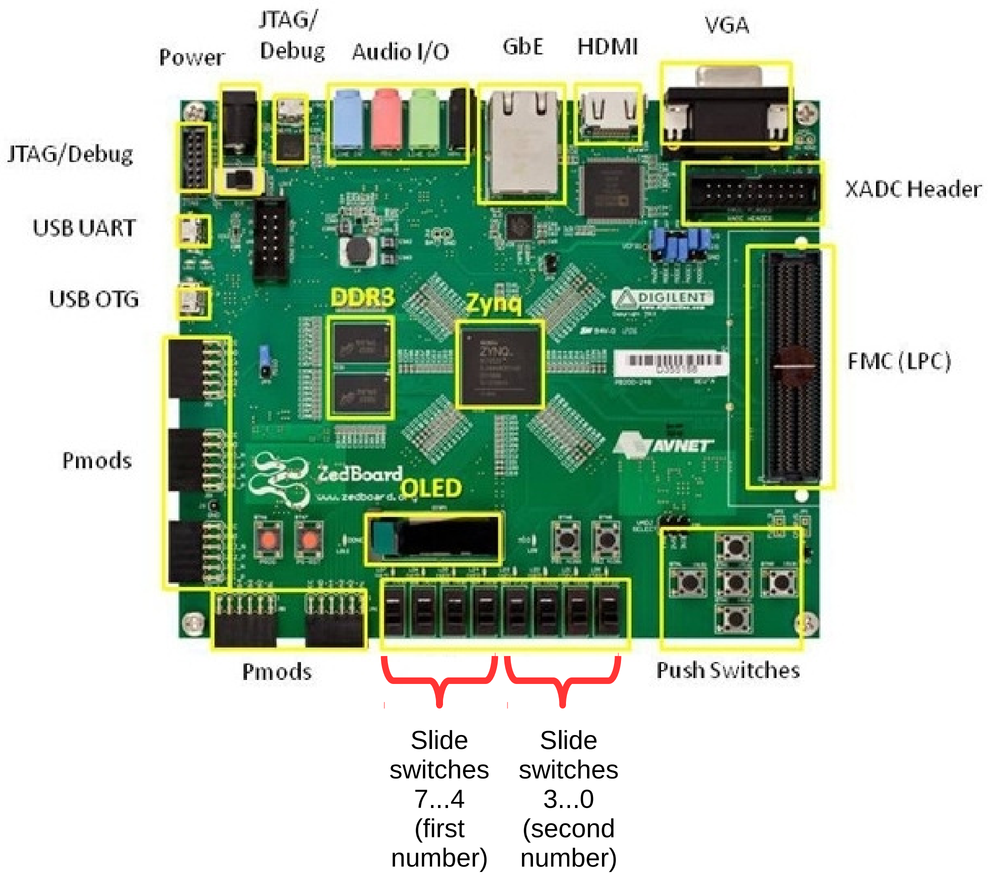

Figure 1: Zedboard

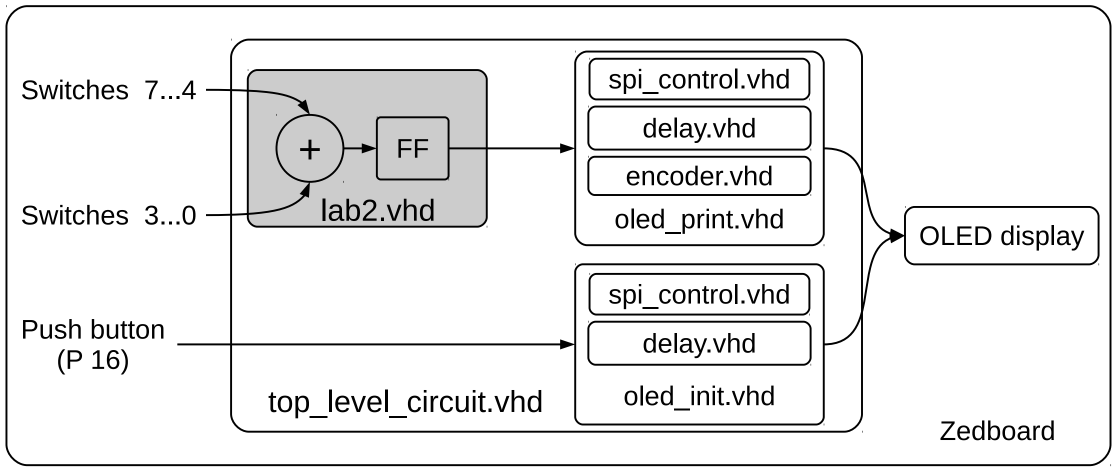

Figure 2: Circuit of interest

Note that oled_init.vhd and oled_print.vhd use other subcircuits on the underlying levels. We do not consider their internal operation in this lab.

## VHDL code

In this lab, in addition to the *std_logic_1164* package, we will use the *numeric_std* package (line 4), which introduces SIGNED and UNSIGNED data types. Note that the STD_LOGIC_VECTOR data type represents a sequence of bits, which is not equivalent to a number, since the same bit sequence can be interpreted as a number in different ways.

As can be seen from lines 7–10, the circuit has three input and one output interfaces. Note that the y interface, which returns the addition result, is one bit longer compared to operands $x_1$ and $x_2$. The additional *carry* bit ensures avoiding overflow errors. Clock interface (clk) is used for time synchronization purposes.

The key component of the circuit architecture description is the PROCESS statement (line 17), which denotes a sequential piece of code. The statement is followed by a *sensitivity list*, which in this case consists of one signal, namely clk. This means that process body commands (lines 19–21) are executed only when changes in the clk signal happen. Provided that the clock signal is a square wave, one should expect two types of signal changes: falling edge and rising edge. The IF statement inside the PROCESS (line 19) ensures that the output signal update happens only on the rising edge.

The addition operation itself is described in the line 20. Since the '+' operator implies having the same number of bits for operands and the result, operands are concatenated with '0'. Note that casting is performed explicitly: firstly operands of the STD_LOGIC_VECTOR type are casted to the UNSIGNED data type and, following this, the addition result of an UNSIGNED type is casted to the STD_LOGIC_VECTOR type.

```
1  ----------------------------------------------------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.ALL;
4  USE ieee.numeric_std.ALL;
5  ----------------------------------------------------------------------------------
6  ENTITY lab2 IS
7      PORT ( x_1 : IN  STD_LOGIC_VECTOR (3 DOWNTO 0);
8             x_2 : IN  STD_LOGIC_VECTOR (3 DOWNTO 0);
9             y   : OUT STD_LOGIC_VECTOR (4 DOWNTO 0);
10            clk : IN  STD_LOGIC);
11 END lab2;
12 ----------------------------------------------------------------------------------
13 ARCHITECTURE Behavioral OF lab2 IS
14
15 BEGIN
16
17     PROCESS (clk)
18     BEGIN
19         IF(clk'EVENT AND clk='1') THEN
20             y <= STD_LOGIC_VECTOR(UNSIGNED('0' & x_1) + UNSIGNED('0' & x_2));
21          END IF;
22     END PROCESS;
23 end Behavioral;
```

24

## Synthesising VHDL code with Vivado

We will follow the same steps with the lab 1 to synthesize the circuit using the Vivado software tool from Xilinx. Creating a project:

- Open Vivado → *File* → *New Project* → *Next.*
- Enter the project name ('lab2') and choose the directory. *Next.*
- *RTL project* (selected), *Do not specify sources* (checked). *Next.*
- *Boards* → *Zedboard* → *Next* → *Finish.*

Let us import *.vhd files to the project:

- *Sources* pane → Right mouse click on *Design Sources* → *Add sources* → *Add or create design sources* → *Next* → *Add files* → Select 'lab2.vhd', 'spi_control.vhd', 'delay.vhd', 'encoder.vhd', 'oled_init.vhd', 'oled_print.vhd' and 'top_level_circuit.vhd', → 'Copy sources into project' checked → *Finish.*
- Open the file from the *Sources* pane to make sure it was imported correctly.

In addition to VHDL code we import a constraints file, which describes the mapping between the circuit's ports and FPGA pins. In this lab we map input numbers to Zedboard switch buttons, the initialization circuit to the middle push switch and output ports to the OLED display.

- *Sources* pane → Right mouse click on *Constraints* → *Add sources* → *Add or create constraints* → *Next* → *Add files* → Select 'lab2_constraints.xdc' → 'Copy constraints files into project' checked → *Finish.*
- Open the file from the *Sources* pane to make sure it was imported correctly.

Now that we have the source files ready, we will go through the steps of the Flow Navigator pane:

- *RTL ANALYSIS* → *Open Elebaorated Design.* At this stage Vivado will show the schematic representation of the circuit.
- *SYNTHESIS* → *Run synthesis.* Synthesis is converting VHDL code into a *netlist.* Netlist is a graph that shows connections between logic blocks.
- *IMPLEMENTATION* → *Run Implementation.* In this context implementation means placing and routing, i.e. mapping the netlist onto a particular FPGA.
- *PROGRAM AND DEBUG* → *Generate Bitstream.* Converting the final circuit into a sequence of bits.

Once the bitstream is ready it can be uploaded to the FPGA. Make sure that your Zedboard has a power supply and USB-JTAG is connected to the Desktop machine. To upload the bitstream: *PROGRAM AND DEBUG* → *Open hardware manager* → *Open target* → *Auto connect.* Then select *Program Device* → *Program.* Once the device is programmed press the middle push switch (Figure 1) to initiate the display. The result of addition should appear on the display (Figure 3). Try changing the slide switches to see the impact on the calculation result.

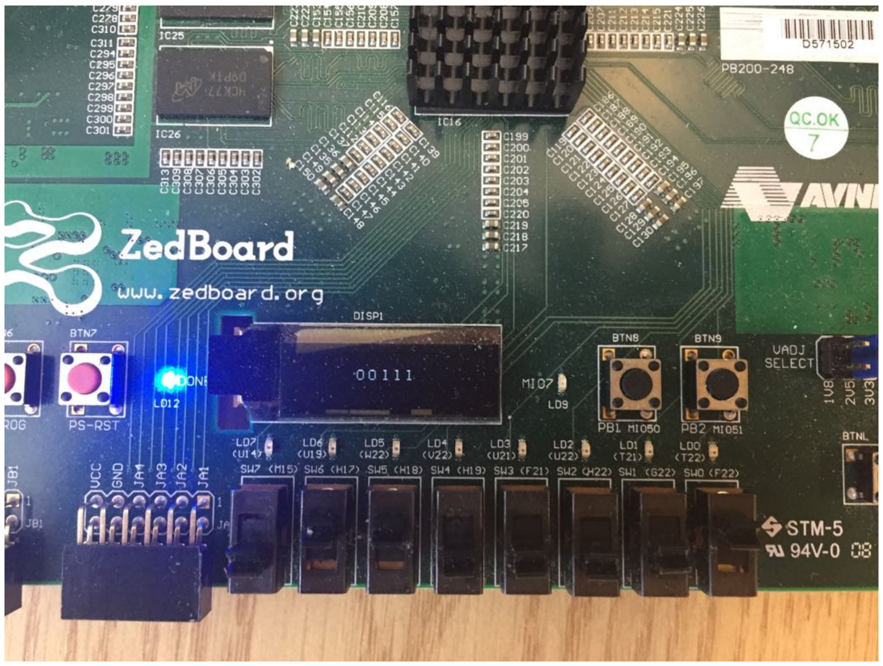

Figure 3: Addition circuit in action.

## Further reading and references

- V. Pedroni. Circuit Design with VHDL. MIT Press. 2010. Chapters 6-7.
- L. Crockett et.al. The Zynq Book. 2015.
- The VHDL code for interfacing the OLED is based on: `https://github.com/faab64/OLED_on_ZedBoard`