Introduction to Programmable Logic

In this lab exercise you will

- Learn to develop digital logic using the VHDL hardware definition language.
- Design and test a digital coincidence counter circuit

Preparations before the lab:

We will use the Xilinx ISE design suite for this lab exercise. A small number of workstations are available for use by those unable to install their own copy of ISE, but if you have a Windows or Linux laptop with you, you may download and install the freeware ISE WebPACK version of the design tools at:

http://www.xilinx.com/support/download/index.htm

Xilinx asks for a login, which also allows you access to their support forums. If you do not have a login, create one and download the tools. This is a large installation, so plan for enough time to download, unpack and install the tools.

Once the installation is complete, the Xilinx License Configuration Manager will come up. Select "Get Free ISE WebPack License", log in, and generate a node-locked license for ISE. Download the generated license file. Finally in the License Configuration Manager click the “Copy Licenses ..." button and navigate to the file. Now you have a working version of ISE.

To configure the Digilent BASYS 2 FPGA developer board from your machine, Linux users will also need to install the Digilent Plugin for Xilinx Users, which can be downloaded at:

http://www.digilentinc.com/Products/Detail.cfm?Prod=DIGILENT-PLUGIN

Task 1: Create a new project, and specify an AND gate in VHDL:

In your home directory, create a folder for your projects (e.g. `vhdl`). In that folder, create a subdirectory for the first part of this lab exercise (e.g. `~/vhdl/my_and`.

Set up and open the Xilinx ISE design suite.

Once ISE opens, create a new project:

- From the main ISE window, click 'New Project...' to bring up the new project wizard.
- On the first page of the wizard, give the project a new name (e.g. my_and), and point the project location and working directory to the project subdirectory that you just created. Do not use a dash (-) in your names, because the VHDL synthesis tools will interpret this as a minus sign.
  You will create the design in VHDL, so for 'Top-level source type' choose option HDL (for Hardware Definition Language).

- On the second page of the wizard, select the target FPGA and other preferences. The Digilent developer board has a Spartan 3E FPGA with approximately 100,000 equivalent logic gates and a 132-pins package:

  Family: Spartan3E
  Device: XC3S100E
  Package: CP132
  Speed: -4

  Use the standard Xilinx-provided tools and options:

  Synthesis Tool: XST (VHDL/Verilog)
  Simulator: ISim (VHDL/Verilog)
  Preferred Language: VHDL
  VHDL Source Analysis Standard: VHDL-93

- When you are done with this click 'Next'. On the third page, review your selections and click 'Finish' to open the new project.

In your design window, choose the 'Implementation' view. Under the project name (my_and) you will see the target device (xc3s100e-4cp132) with no associated files.

Create a new top-level design file with the New Source Wizard:

- Select source type 'VHDL Module'.
- For file name, give a name such as `my_and`.
- Make sure the 'Add to project' box is ticked and then click 'Next'.

On the second page, define the design module properties:

- Use the default names for the entity and architecture names.
- Define the ports of the entity: two input ports (a,b) and one output port (q).
- Review your choices, and click 'Next'.

On the third page, review your choices and click 'Finish'. Don't worry too much about making a mistake here. You can always change things later in the VHDL editor.

If you have done everything correctly, you will see a top-level VHDL design (my_and) in your design hierarchy. You can open and view the VHDL source, which will contain an entity declaration (defining the input and output ports) and an empty architecture (where the behavior is specified).

After the “begin” line in the architecture, define the behavior of the circuit. Since this circuit is a simple logic gate, you can do this with a single VHDL statement that uses the native **and** operator:

```
q <= a and b;
```

Save the design file and check that no syntax errors are reported.

At this point you would simulate the design to verify that it works correctly, but we will skip over this step for this exercise.

Task 3: Assign FPGA input and output pins

Now that you have specified the behavior, the next step is to create a "user constraint file" (UCF) to guide the design tools in fitting your design within the target FPGA. For this design you simply need to assign which physical I/O pins on the FPGA should be connected to the input and output ports of your entity.

In the process window, expand 'User Constraints' and double-click on 'I/O Pin Planning Constraints (Pre-Synthesis)'. The Plan-ahead program will open a new window. In the center of the PlanAhead window is a section titled 'I/O Ports' where you can easily make these assignment easily.

Expand the list of ports to view the details. The 'Site' column is where the placement of the I/Os is specified. The Digilent developer board is clearly labeled with the FPGA pins names connected to each peripheral.

For inputs `a` and `b`, choose two of the eight switches on the bottom. For instance, SW0 and SW1 are connected to FPGA pins P11 and L3, respectively. Enter these in the 'Site' column for ports a and b in PlanAhead. For output `q`, choose one of the small LEDs located above the row of switches. For instance, LED LD0 is connected to pin M5.

If you select the 'Package' display tab, you will see the corresponding pins marked by small rectangles as they are assigned. This can be a good check to make sure your assignments are sensible.

Save your constraints and exit PlanAhead.

Task 4: Implement your design:

In the process window, scroll down and double click on 'Generate Programming File'. ISE will follow the design flow up to that point, including synthesis and place-and-route. If all proceeds without error, ISE will finally generate a configuration file (`my_and.bit`) that you may download directly to the FPGA.

Connect the FPGA developer board to your computer with the USB cable, and turn on power. Make sure the jumper JP3 is correctly set to configure from the 'PC' (not ROM).

Double-click on 'Configure Target Device' and let ISE open the iMPACT application. You will see several choices for configuring FPGAs. Here we will use 'Boundary Scan' to program the FPGA directly using the JTAG interface.

Double-click 'Boundary Scan', and in the main window, right-click and select 'Initialize Chain'. Ignore and cancel the pop-up windows for now. iMPACT will scan the serial programming chain and display the connected devices. On your board are two devices. The first (xc3s100e) is the FPGA. The second (xcf02s) is a configuration PROM that can be ignored for now.

Right-click on the FPGA (xc3s100e) and select 'Assign New Configuration File'. In the pop-up window select and open `my_and.bit`. Do not attach a PROM to this device when asked.

Now you are ready to program the FPGA. Right click on the FPGA icon and select 'Program'. Click 'OK' on the following dialog (if one appears), and the download will proceed.

Congratulations! You have completed your first FPGA design. Test by trying different on/off combinations for the two switches, and see which combinations cause the LED to light.

If time permits, you might want to change the behavior of your design to another kind of gate (OR, XOR). Edit and modify the VHDL code, save, and implement the design. Finally, download the new `.bit` file to the developer board and test the new functionality.

Task 5: Design a digital coincidence counter circuit

Now that you have gone through the design process, the next task is to design and implement a circuit to count coincidences between two input signals. These inputs can either be two pushbuttons on the developer board, or cables from two scintillator detectors to count cosmic muon events.

In ISE, create a new project `coincidence` in a new directory, following the design flow you used for the previous tasks.

The inputs are two buttons and two scintillator signals, both of them vectors (buses) with index 1 down to 0. There is also a 50 MHz clock provided on the board. The outputs are to the seven segment LED display. This includes seven cathodes (CA, CB, … CF, CG) and a vector (bus) of four anodes 'an' with index 3 down to 0.

Create a top-level VHDL module with these input and output signals. The final entity declaration will look something like this:

```
entity coincidence is
    Port ( button : in  STD_LOGIC_VECTOR (1 downto 0);
           scint : in  STD_LOGIC_VECTOR (1 downto 0);
           clk : in  STD_LOGIC;
           CA : out  STD_LOGIC;
           CB : out  STD_LOGIC;
           CC : out  STD_LOGIC;
           CD : out  STD_LOGIC;
           CE : out  STD_LOGIC;
           CF : out  STD_LOGIC;
           CG : out  STD_LOGIC;
           an : out  STD_LOGIC_VECTOR (3 downto 0));
end coincidence;
```

In your architecture, you will need to declare some internal signals, plus a component for handling the seven-segment LED display.

Before the "begin" statement in the architecture, start by declaring two signals `a` and `b`, which are the internal names of the two signals for which you want to detect coincidences. Let them be initialized to 0 as a default:

```
signal a, b : std_logic := '0'
```

You need to have a 'state' signal that keeps track of whether the circuit is waiting for a coincidence, or whether a coincidence has been found and the circuit is waiting for both signals to return to zero. You also need a 16-bit 'counter'signal to keep track of how many coincidences have been seen:

```
signal state : std_logic := '0';
signal counter: std_logic_vector(15 downto 0);
```

Finally you need to declare the component ‘disp4’ that handles the 7-segment LED display. You can download it at URL:

http://www.sysf.physto.se/kurser/digsyst/disp4.vhd

Import the file into your design and declare it in the architecture:

```
component disp4
    Port (
           clk:       in std_logic;
           disp_in :  in  std_logic_vector(15 downto 0);
           an : out std_logic_vector (3 downto 0);
           CA, CB, CC, CD, CE, CF, CG : out std_logic);
end component;
```

Now it is time to write the circuit behavior after the ‘begin’ statement in the architecture.

Start by allowing the circuit to use either a coincidence between the two buttons or between the two scintillator inputs:

```
a <= button(0) or scint(0);
b <= button(1) or scint(1);
```

The next step is to write a simple state machine that checks states `a` and `b` on every rising edge of the 50 MHz clock. If both signals are true (1) then a coincidence is detected, and the counter is incremented by one. Both signals must return to false (0) before another coincidence is allowed. Do this with the following VHDL *process*:

```
c_counter : process(a, b, clk)
begin
     if rising_edge(clk) then
          case state is
               when '0' => -- Waiting for a coincidence
                    if a='1' and b='1' then
                         counter <= counter + 1;
                         state <= '1';
                    end if;
               when others =>  -- Coincidence!
                    if a='0' and b='0' then
                         state <= '0';
                    end if;
          end case;
     end if;
end process c_counter;
```

The final step is to *instantiate* the driver component for the seven-segment LED display. The input ports are the 50 MHz clock and the counter value, while the output ports are the anode and cathode signals to the display:

```
display : disp4
      port map (clk => clk, disp_in => counter, an => an,
                CA => CA, CB => CB, CC => CC, CD => CD,
                CE => CE, CF => CF, CG => CG);
```

Save everything and check for syntax error messages.

Task 6: Implement and test the coincidence counter

Open the PlanAhead program as before, and create a new set of user constraints.

The two buttons are on the bottom right of the Digilent board, and the first two have locations G12 and C11, respectively.

For the scintillator inputs choose the first two pins on any of the six-pin PMOD ports at the top of the board. For PMOD port B, for example, you would use pins C6 and B6.

The 50 MHz clock is at location B8.

The six LED display cathodes CA to CF have locations L14, H12, N14, N11, P12, L13, and M12, respectively.

Finally, the LED display anodes an0 to an4 are located at K14, M13, J12 and F12.

Save and exit PlanAhead, and implement the design, generating a configuration file as before.

Download the configuration to your board and test the coincidence counter with the two buttons. If all goes well, ask your instructor to help connect your counter to the scintillator detectors, and try to detect cosmic rays!