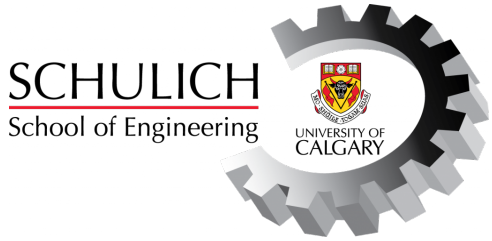


Department of Electrical and Computer Engineering
ENEL453: Digital System Design

Lab 2: Digital Clock

Instruction Manual

1 Overview

In this lab, we will design a circuit displaying minutes and seconds on the FPGAs four 7-segment displays. The input clock frequency coming from the FPGA is 25MHz. The design includes:

1. A clock divider circuit capable of producing clock-like signals of different frequencies. To display seconds, we need to downscale to 1 Hz (1/sec). To display ten seconds, we have to downscale to 0.1Hz etc. Therefore, we have to divide the input frequency 25MHz by using divide-by-25 counter as the divider. It shall be followed by divide-by-10 counter, and another one, and so on.
2. A decoder to handle translation of regular numbers to seven-segment display codes.
3. A state machine to switch between four of the seven-segment displays (to display seconds, tens of seconds, minutes and tens of minutes.
4. A top-level design, connecting all three blocks together in order to display a digital clock on the Basys2 board.

This lab is worth 8% of your term grade and broken up as:

- Pre-lab: 2%
- In-lab Demo: 3%
- Lab Report: 3%

2 Before You Begin

- Make sure you've completed the pre-lab exercises. These will help you complete the lab faster! You also need to have the pre-lab signed off by a TA when you come into the lab.
- Read the entire lab instructions manual
- **A maximum of two students may work together at any station.** Each group member should be able to perform the whole project individually. **Winter 2015:** Due to the size of the class, we will probably have some groups of 3 students.
- Back-up your project frequently. Save all intermediate materials that you generate (figures, screenshots, code snippets).
- **The lab report is due by 11:59am on Tuesday of the week following your lab session.**
- Do not share your code. Similar sources will be awarded a mark of **zero**. We will use Moss (Measure of Software Similarity developed at Stanford University), which is an automatic system for determining the similarity of programs.
- If you need more time to complete the lab, you can sign out the board for use outside of the lab hours. The boards are available from Mr John Shelley, room ICT 204.
- We recommend using the lab station for performing the lab rather than personal computers. If you want to work on the projects at home or on your laptop, you can download the CAD tools (free) from Xilinx: Xilinx ISE Design Suite Webpack: `http://www.xilinx.com/products/design-tools/ise-design-suite/ise-webpack.html`.

3 The Basys2 Board

For more details you can consult the pre-lab document, or the Basys2 manual, available at `https://www.digilentinc.com/data/products/basys2/basys2_rm.pdf`.

4 Setup the Project

Follow the same process as in Lab 1 to set up a new project in Xilinx ISE. Be sure to give it a descriptive name. We will proceed through this lab by incrementally adding

modules that you prepared during the pre-lab, and testing them. Make sure you have all of your developed code available.

Begin your lab report:
Create a new Word document and name it `ENEL453_LAB2_LASTNAME1_LASTNAME2`. Create a title page, which includes:

- The title "ENEL 453 Laboratory 2"
- Lab section number (B01, B02, etc)
- Names of all group members in full
- Date that the lab was held on
- The statement: "We (I) declare that this laboratory report is entirely our (my) own work and includes no material which has been copied from any other source *excepting that material which is clearly identified as the work of others.*"

5 Clock divider

First, we will test that the clock divider works as expected. In your Project Navigator, right click the project name and select "Add Source..." (see Figure 1). Navigate to your `clock_divider.vhd` file and add it.

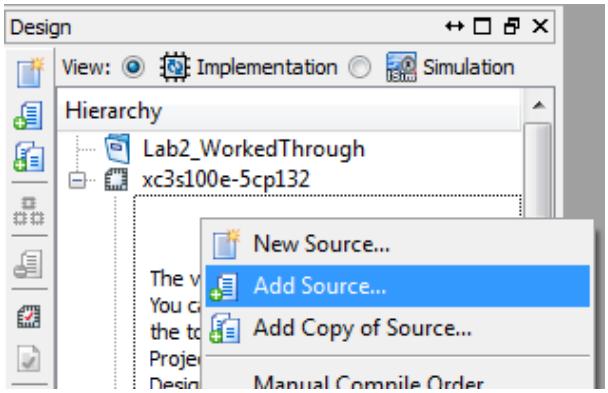


Figure 1: Adding an existing file to your project.

If you now expand the component (click the small + sign next to the name), you will see that all inner components are shown with orange question marks:

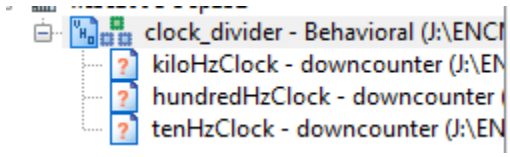


Figure 2: The ISE tool cannot locate the inner components.

To resolve this problem, right click the project again, and select "Add Source..." once again. Navigate to `downcounter.vhd` and that as well. The orange question marks should disappear.

Now add a testbench for the `clock_divider` (right click the project name and select "New Source..." to get started; instructions are the same as for Lab 1). Make sure your `clock_divider` synthesizes without errors first, though. In order to test your design in a reasonable amount of time, change the `period` of the very first divider block (`kHz` one) to a smaller number - something like 10 should do the trick.

In your testbench, make sure all inputs have values assigned to them; clock should be handled for you already in the generated code. Run the testbench.

In order to be able to analyze this without constantly decoding binary, you need to change the radix of the displayed numbers. Right-click the signal name representing a particular clock digit (such as `sec_dig1`) and select "Radix" → "Unsigned Decimal":

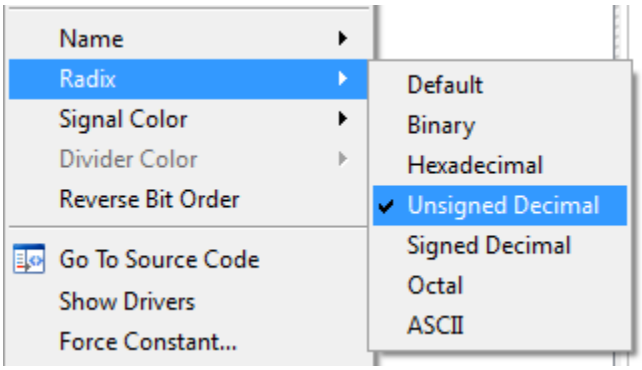


Figure 3: Menu to change the radix of the values displayed in ISim.

When you run it initially, you most likely will not see many signals changing. This means we need to run it for a longer amount of time. At the top of the menu, there is a small window where you can enter the desired simulation duration:

Figure 4: Menu to change the simulation runtime in ISim.

Change this value to 30ms and hit Enter. This might take a few minutes to complete. Zoom out of the waveform, and you should see something that looks like Figure 5; a zoomed-in version is shown in Figure 6.

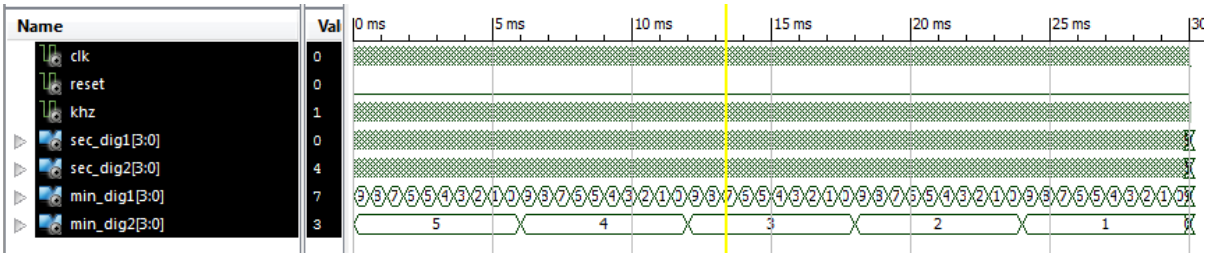


Figure 5: Clock divider testbench output showing minutes counting.

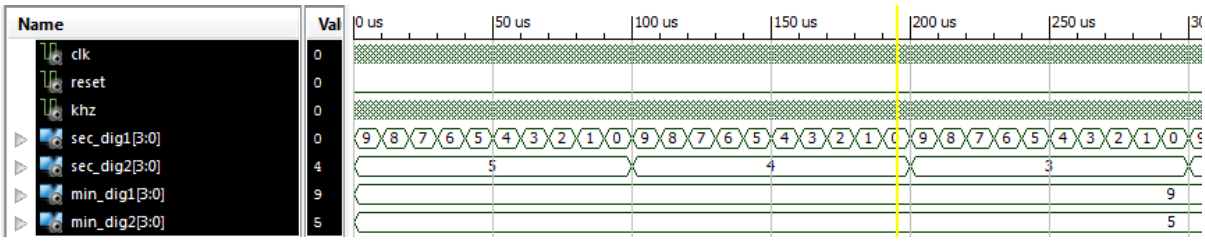


Figure 6: Clock divider testbench output (zoomed-in to show seconds counting).

Add Screenshots:

Take a screenshot (or two!) showing that your clock divider outputs overflow at proper values (tens of minutes shouldn't exceed 5, for example), and that they work as expected. Explain what you are seeing and any discrepancies you notice.

6 Seven-Segment Decoder

Return to your project and once again "Add Source...", but this time select the `sevensegment.vhd` and add it to the project. It will appear separate from the `clock_divider`. Save the file and make sure no immediate errors appear (fix them if they do).

Switch to Simulation view (radio button next to Implementation). Now we need to add the testbench you created in pre-lab for this component. Select `tb_sevensegment.vhd` through the "Add Source..." menu. Select the testbench in the file list and first perform a syntax check:

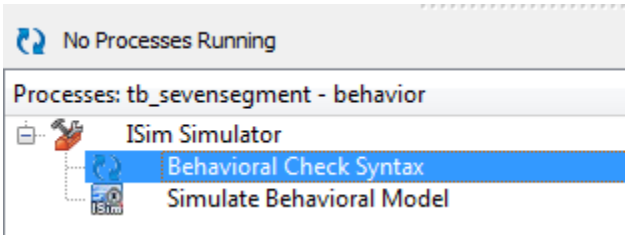


Figure 7: Option to perform a syntax check on both the testbench code and the component before simulation.

If any errors come up, make sure you fix them. Once you are ready to go, run the simulation for a significant enough amount of time to see the output change in response to the input.

Add Screenshot of Simulation:
Take a screenshot showing that the decoder works as expected giving all possible inputs (make sure `data` cycles through all values between 0 and 9). Also ensure you show that `dp` changes in response to `dp_in`.

7 Seven-segment Digit Selector

Return to Implementation view and use "Add Source..." to add in the last component along with its testbench - `sevensegment_selector.vhd` and `tb_selector.vhd`.

Perform a syntax check and run the simulation to capture the intended behaviour. The pre-lab contains a screenshot of ISim that your simulation output should look like.

Add Screenshot of Simulation:
Take a screenshot showing that the selector works as expected. Be sure to capture at least a full cycle of `output` values.

8 Digital Clock

Now that you have tested all of the individual components, it is time to connect them all and create our digital clock!

Return to the Implementation view and create a new course (VHDL Module), name it something descriptive (something like `digital_clock.vhd` works). For input ports fill it in as:

Port Name	Direction	Bu
clk	in	
reset	in	
CA	out	
CB	out	
CC	out	
CD	out	
CE	out	
CF	out	
CG	out	
DP	out	
AN1	out	
AN2	out	
AN3	out	
AN4	out	
	in	

Figure 8: Pins for the top-level design. These should match those in your top-level diagram (if you added something, make sure you add it here as well).

Using your top-level diagram from the pre-lab, connect `clock_divider`, `sevensegment` and `sevensegment_selector` together. The code below gives you some detail to get started:

```
architecture Behavioral of digital_clock is
-- Internal signal names have to go here
signal i_dp: STD_LOGIC;
signal i_an: STD_LOGIC_VECTOR(3 downto 0);
signal i_kHz: STD_LOGIC;
signal digit_to_display: STD_LOGIC_VECTOR(3 downto 0);

-- Declare components here:
-- These act as C/C++ function prototypes - each one should only occur one
component sevensegment_selector is
    Port ( clk : in STD_LOGIC;
           switch : in STD_LOGIC;
           output : out STD_LOGIC_VECTOR (3 downto 0);
           reset : in STD_LOGIC);
end component;

begin
-- Instantiate components here:
-- These are kind of like C/C++ function calls
SELECTOR: sevensegment_selector
port map( clk => clk,
          switch => i_kHz,   -- This is an output of the clock divider circuit
          output => i_an,   -- Wire bus to connect to anodes
          reset => reset);

-- Mux to choose a digit to display
-- Make sure you add all time digit logic vectors to the sensitivity list
digit_mux: process(i_an, tens_minutes, -- add more here )
begin
   case (i_an) is
      when "0001" => digit_to_display <= tens_minutes;
      -- Continue mux connections here to choose a digit:
      when others => digit_to_display <= "0000";
   end case;
end process;

-- Connect internal signals to outputs here:
DP <= i_dp;
AN1 <= not i_an(0);
AN2 <= not i_an(1);
AN3 <= not i_an(2);
AN4 <= not i_an(3);

-- Write logic for the decimal point (i_dp) here:
```

```
end Behavioral;
```

The easiest way to create component code is to simply copy the `entity` declaration from the relevant component's file, and change `entity` to `component`, as well as `end COMPONENTNAME;` to `end component;`.

When you create an instance of the component, it needs to have a unique name. In the code above, for example, the `sevensegment_segment` is instantiated as `SELECTOR`.

Once you are satisfied with the component and your code, you can start testing it. Create a testbench for this component, set all inputs to values and run it.

You will not see many useful values the moment because it is only showing top-chip's inputs and outputs. In order to view internal signals, you will need to navigate into the `uut`. In ISim on the left-hand side, expand the menus as such:

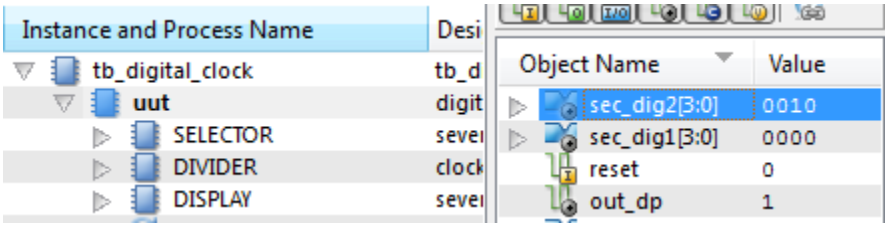


Figure 9: You can keep expanding the components - for example, you can open up `SELECTOR` as well, and view its internal signals as well.

Once a signal you wish to view appears in the Objects list (such as `sec_dig2` above), you can simply drag it into the waveform to view its values. You may need to rerun the Simulation, however. For this, click the blue button with an arrow on it, followed by the arrow with an X:

Figure 10: These buttons allow you to rerun the simulation from the start. Press the left button first, to delete current waveforms, followed by "Run to specified time" (the arrow with an X).

Run the simulation and change the VHDL design as needed until it performs as expected.

Add Screenshot of Simulation:
Take a screenshot showing that the digital clock works as expected. Describe how you know that it works properly.

9 Download the Digital Clock

Now that we have designed and tested the entire digital clock, it is time to add a pin constraints file to the project, synthesize and implement it!

First of all, do not yet change the first divider's value back to 24999, we will do this a bit later.

Secondly, we need to set the `digital_clock` component to be the top-chip module. In the list of components, right click `digital_clock` and select "Set as Top Module":

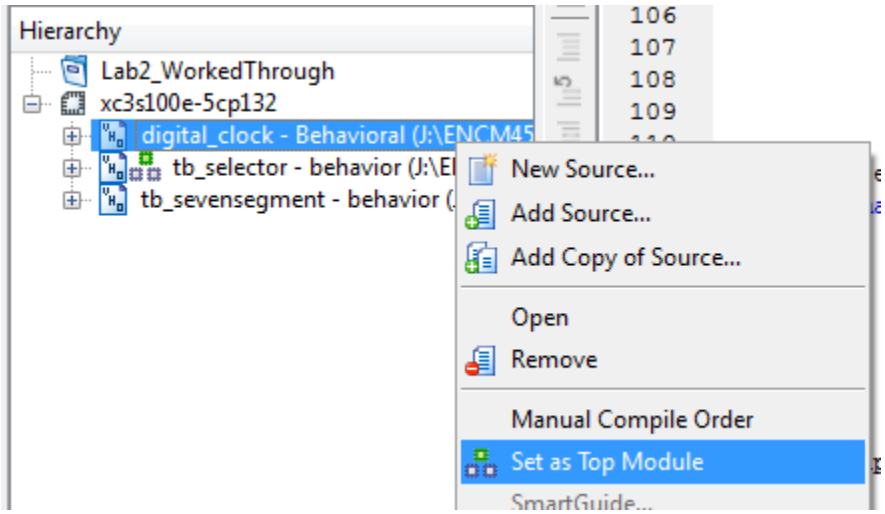


Figure 11: Using this option you can change the component that gets synthesized as your top-chip.

Using the "New Source..." option, add a new user implementation constraints file. Use your pre-lab work to assign pin numbers to all of the `digital_clock`'s entity ports. Here is the syntax to get your .ucf file started:

```
NET "clk" LOC = "B8";  # 25MHz clock signal
```

Once you are satisfied with your pin assignments, switch to `digital_clk` and run Synthesize, followed by Implement. We are going to perform a new simulation at this point - a post-translate one. This simulation is typically used to make sure that the design functions as expected after the synthesis and routing tools had a go at interpreting what it is you were trying to do.

Expand the menu and run "Generate Post-Translate Simulation Model":

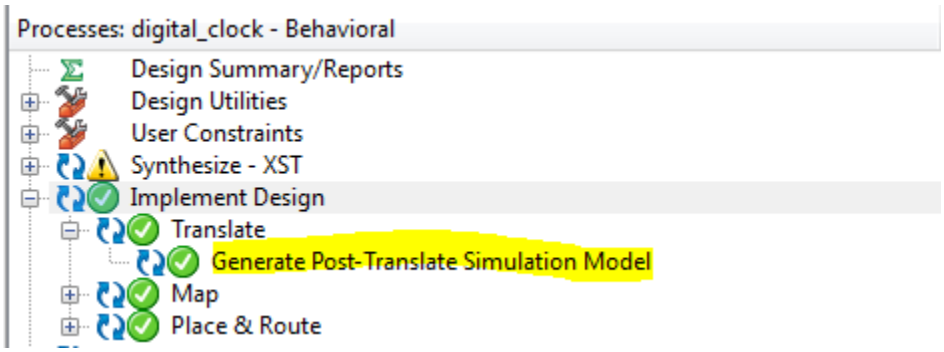


Figure 12: Use this option to generate a post-translate representation of your VHDL design.

This will produce a .vhd file (in your project directory) where all of the components are replaced by components that will be used on the FPGA, such as look-up-tables (LUTs) and buffers.

Click the Simulation radio button and in the drop down list under it, select "Post-Translate":

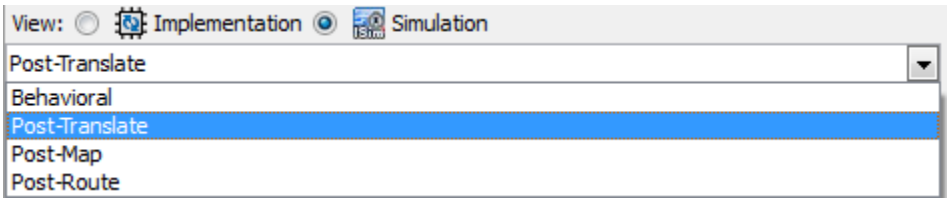


Figure 13: Select the "Post-Translate" option to select the new representation of your design to be used in the testbench.

Now, proceed to select the `digital_clock` testbench and run it as you normally would, this time selecting "Simulating Post-Translate Model" in the ISim options.

In the ISim window, expand `tb_digital_clock` option under Instance and Process name on the left (see Figure below), followed by `uut`, and add the 4 divider outputs to your waveform.

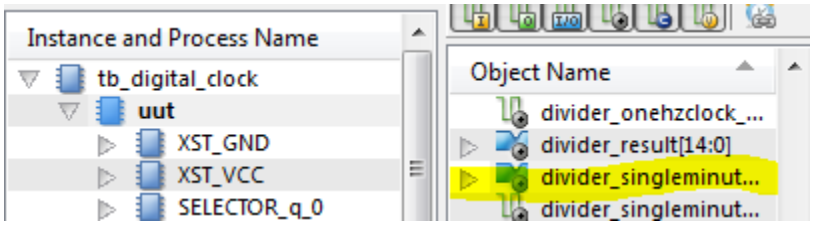
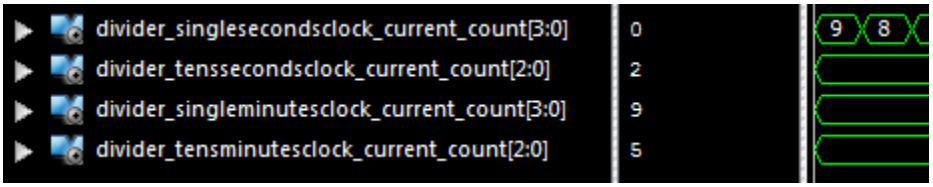


Figure 14: The signals you are looking for will mostly likely be called something like `divider_singleminutes`, with the `divider_` prefix added by the synthesis tool. Look for signal vectors that contain 4 bits :).



You need to run the simulation for about 10 ms once again to get a good view of signals changing. This might take a while. If it is taking too long (or your computer is slow), go back and changing the first divider's value to a smaller number, re-synthesize, re-implement design and try simulating again. Your final waveform should look very similar to what you saw in the original testbench.

Add Screenshot of Simulation:
Take a screenshot showing that the digital clock still works as expected. Describe how you know that it works properly. Discuss any differences you note between the pre-synthesis (original) and post-translate models.

Now we need to change the first clock divider's value back to 24999 (it was 10 for simulation purposes). After that, right-click the "Generate Programming File" and select "ReRun All" to create a bit file.

Use Adept to download the design to your Basys2 board and verify that it works as intended.

Demo to a TA:
Show a TA that your code works as intended, and that reset works.

10 Additional Questions

Add a section for additional questions:
Add a section to your report with answers for the following questions. These questions can be completed as a group.

1. Why did we choose to use a single clock signal to drive all parts of the system? What is the impact of this design choice on power consumption?
2. What is the difference between an `entity`, a `component` and an `instance`?
3. Assume you decide to choose a `period` = 25000 and `WIDTH` = 6 for a downcounter? What effect would this have on a digital clock if you used this one as a clock divider?
4. For implementing the downcounter, we have used three libraries: IEEE.STD_LOGIC_1164.ALL, IEEE.NUMERIC_STD.ALL and IEEE.STD_LOGIC_UNSIGNED.ALL. Why not just IEEE.STD_LOGIC_1164.ALL, as we did in Lab 1?
5. What is the reason for using "generics"?

You are finished!
Make sure you have demonstrated the digital clock to a TA. Your lab reports are due one week from today in the online boxes on Desire2Learn. Remember to back up your code! You are required to submit:

- Report in .doc or .pdf format
- The .bit file for your design. **Do not try to open this file. You will not be able to read it, it is just composed of a large number of bits to describe the circuit. Do not paste it into Word.**
- The synthesis report for your design (a file in your project folder with extension .syr
- All of your .vhd code - for this lab there should be 4 testbenches and 4 component files (do not include `downcounter.vhd`).
- The implementation constraints file - the .ucf file

You can simply zip up all of the above files together, or upload them separately. **Do not submit the entire project directory.**