Fall 2016

ECE 414 Lab Tutorial II – Using Seven Segment display

Prepared by Randil Gajasinghe and Dr. Onur Tigli
Last edited by Alperen Toprak

Warning – ***As with any sensitive electronic device, this development board is sensitive to static electrical charges. Static charges in your body can damage the devices on the board. Please do not touch the board more than necessary. Improper handling can damage the board. Please handle the board carefully; do not drop or otherwise mishandle it.***

1. Section one – Using a single digit

First step is to understand the common anode 7-segment display configuration. There are eight 7-segment displays on the Nexys-4 board. Each display has one common anode (positive voltage) and eight cathodes, each controlling one LED, as shown in Figure 1.

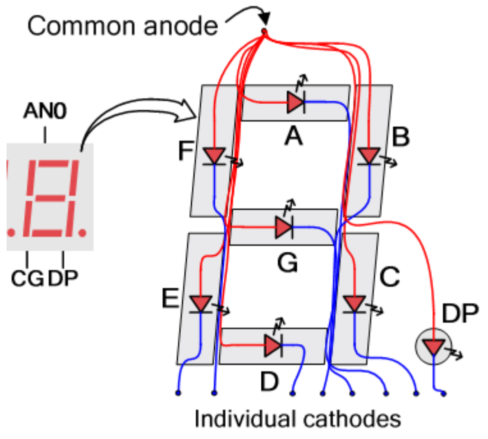


Figure 1: Common anode 7-segment display configuration on the Nexys-4 board.

Now, let's check how the eight 7-segments are configured on Nexys-4 board, which is shown in Figure 2. Cathodes of corresponding LEDs of all eight digits are connected together and connected to eight pins of the FPGA. So, cathode A inputs of all displays are shorted to each other, so is for cathode B, C, and so forth. On the other hand, common anode of each display is controlled from a single output from the FPGA. So, each pin from M1 to N6 **enable** the corresponding display. Check the circuit diagram: These are all active low inputs, if you want to enable a display, you should apply zero to the corresponding pin.

Assume that we want to use the leftmost display first. You should set the anode outputs as follows:

M1	= 0
L1, N4, N2, N5, M3, M6, N6	= 1

You already know which LEDs to turn on to show a digit on a 7-segment display. Note that it is also possible to show hexadecimal numbers by adding letters from *A* to *F*. Create a Verilog code using "case" to map a 4-bit number (including *A-F*) to a 7-bit output that will control the cathodes of a 7-segment. Note that these are active low, too: an LED is turned on if corresponding output is set to zero.

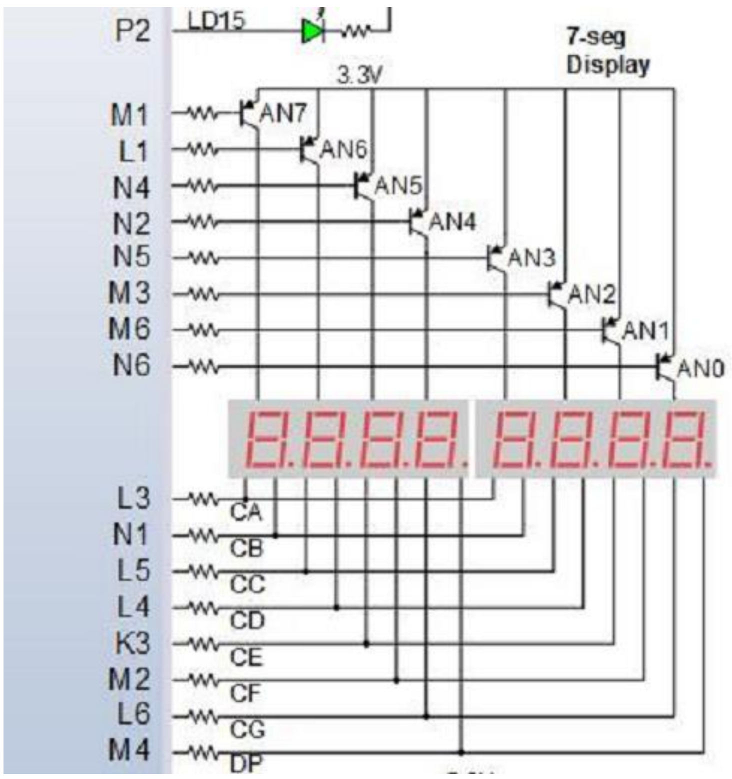


Figure 2: Configuration of the 7-segment displays on the Nexys-4 board.

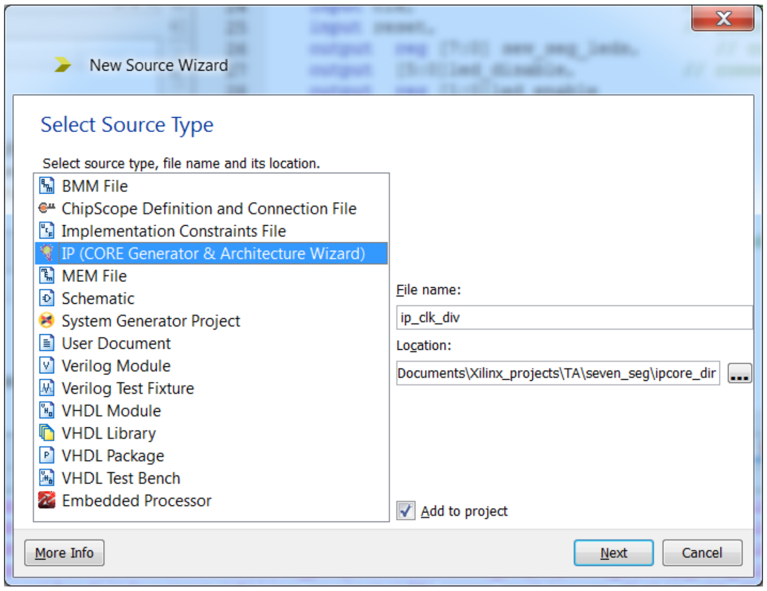
2. Section two – Using multiple digits

Multiple digit operation will be demonstrated using two digits in this tutorial. We will provide two 4-bit numbers as inputs using slide switches, and two 7-segment displays will show the corresponding hexadecimal numbers. You will be given the necessary files to perform this task; all you need to do is follow the steps in the tutorial to add a clock generator, and then you can generate the bit file. Note that the given codes are not explained here; each code has comment lines that provide detailed descriptions of its blocks. **Please study the provided codes as you follow the procedure given below, and make sure that you understand their operation**.

Remember that the cathodes are shorted on Nexys4 boards; therefore, we cannot show two different digits at the exact same time instant. However, thanks to the finite time constant of the LEDs, it is possible to observe two different digits simultaneously if we switch the displays fast enough. We are going to need a clock in order to switch the inputs between the displays. The onboard oscillator clock is 100 MHz, which is too fast for this application. Slower (or faster) clocks can be generated using an IP core provided by Xilinx. The minimum allowed clock generated using an IP core is 5 MHz, which is still too high. Therefore, we will create this 5 MHz clock using the Core Generator of Xilinx ISE, and then we will further divide it with a simple clock divider to obtain the clock that we need to switch the displays.

1. Create a new project with the proper settings (You can refer to Tutorial 1 for the settings).

2. Add copies of the provided 3 v files and 1 ucf file (*sev_seg_with_clk_top.v*, *sev_seg_with_clk.v, sev_seg_decoder.v,* and *sev_seg_with_clk_top.ucf*) to your project by using the "Add Copy of Source" option (Do NOT use "Add Source", use "Add Copy of Source"). Make sure that *sev_seg_with_clk_top.v* is the top module.
3. Click *new source* button and select *IP (CORE Generator & Architecture Wizard)*. Give it the name *ip_clk_div* and click next. (You can use different names in your future projects).



Select Source Type

Select source type, file name and its location.

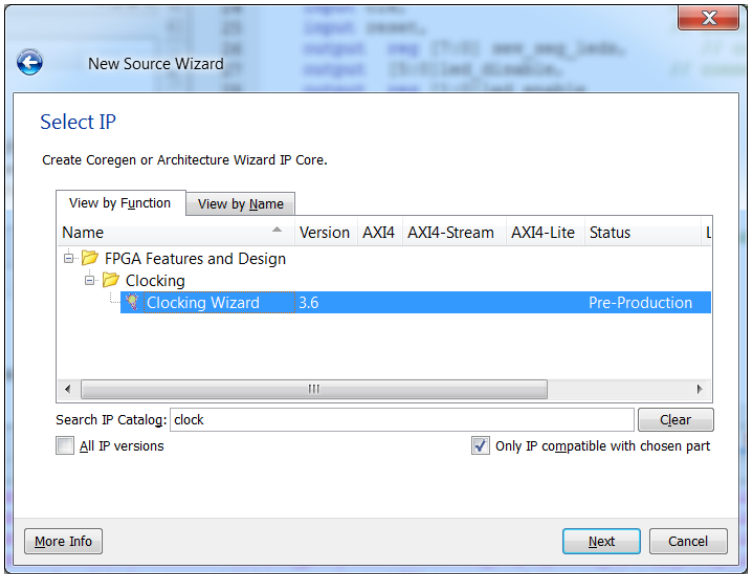
- BMM File
- ChipScope Definition and Connection File
- Implementation Constraints File
- IP (CORE Generator & Architecture Wizard)
- MEM File
- Schematic
- System Generator Project
- User Document
- Verilog Module
- Verilog Test Fixture
- VHDL Module
- VHDL Library
- VHDL Package
- VHDL Test Bench
- Embedded Processor

File name: ip_clk_div

Location: Documents\Xilinx_projects\TA\seven_seg\ipcore_dir

✓ Add to project

4. Select *Clocking Wizard* and click next and finish.



Select IP

Create Coregen or Architecture Wizard IP Core.

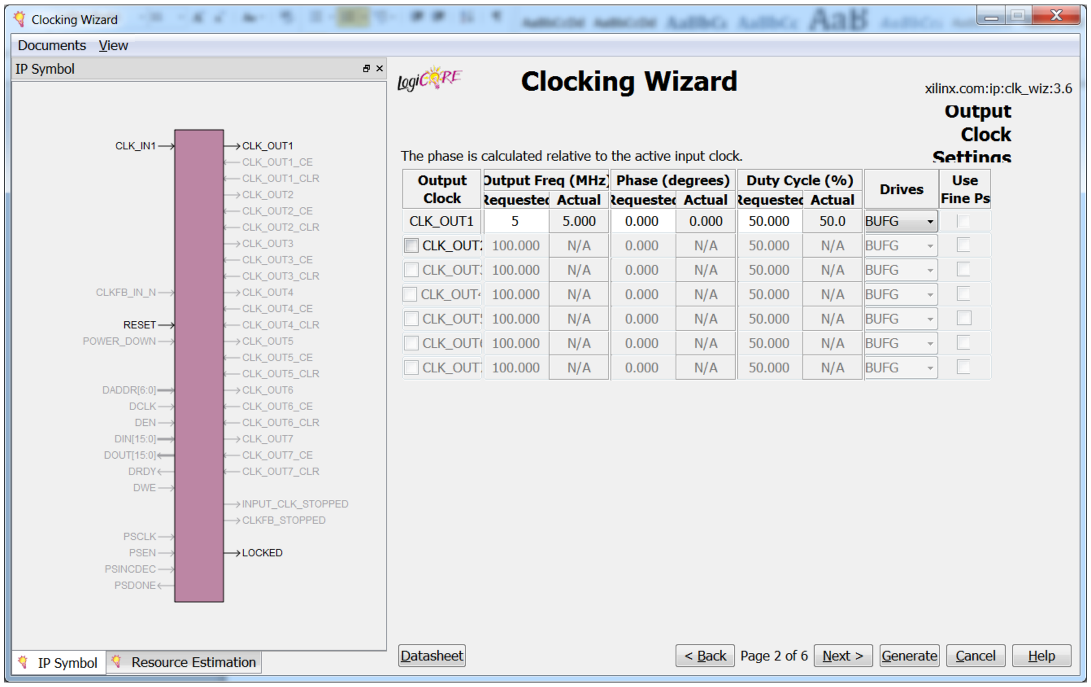
Name	Version	AXI4	AXI4-Stream	AXI4-Lite	Status
FPGA Features and Design					
Clocking					
Clocking Wizard	3.6				Pre-Production

Search IP Catalog: clock

☐ All IP versions

✓ Only IP compatible with chosen part

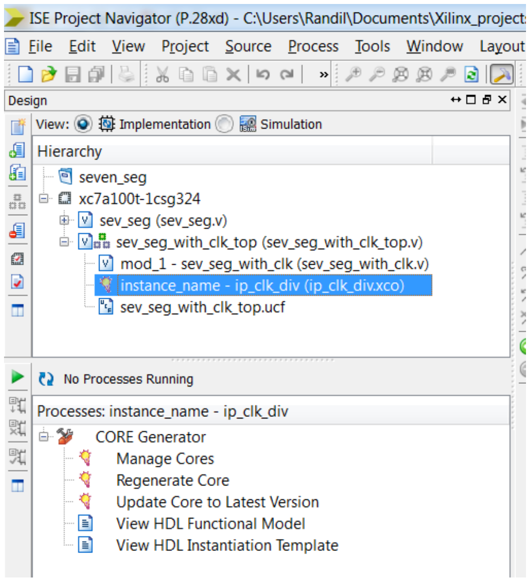
5. Clocking wizard will open. We will create a simple clock generator. Leave default selections in the first page and click next. Enter 5MHz (minimum allowed value) for *CLK_OUT1*. Click next.



The phase is calculated relative to the active input clock.

Output Clock	Dutput Freq (MHz)		Phase (degrees)		Duty Cycle (%)		Drives	Use Fine Ps
	Requestec	Actual	Requestec	Actual	Requestec	Actual		
CLK_OUT1	5	5.000	0.000	0.000	50.000	50.0	BUFG	☐
☐ CLK_OUT:	100.000	N/A	0.000	N/A	50.000	N/A	BUFG	☐
☐ CLK_OUT:	100.000	N/A	0.000	N/A	50.000	N/A	BUFG	☐
☐ CLK_OUT·	100.000	N/A	0.000	N/A	50.000	N/A	BUFG	☐
☐ CLK_OUT!	100.000	N/A	0.000	N/A	50.000	N/A	BUFG	☐
☐ CLK_OUT(	100.000	N/A	0.000	N/A	50.000	N/A	BUFG	☐
☐ CLK_OUT:	100.000	N/A	0.000	N/A	50.000	N/A	BUFG	☐

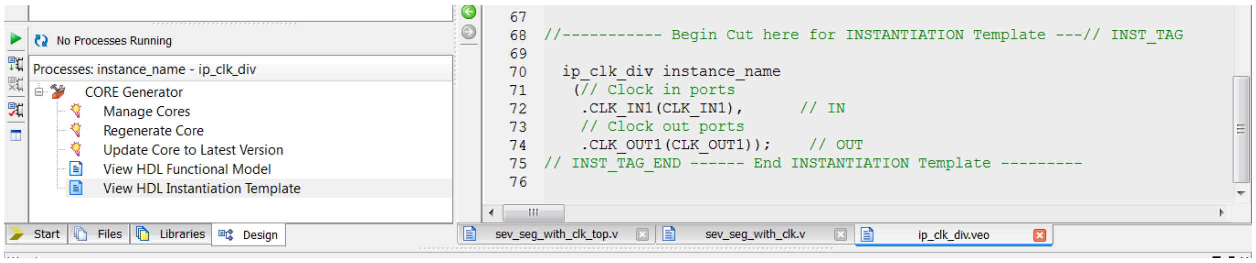
6. Uncheck *RESET* and *LOCKED* in *optional Inputs / Outputs*. Click next three times and click generate. The generated IP Core will be shown in the Project navigator similar to any source file; but the icon will be different. The created IP Core can be instantiated in a design just like a module. You can find the instantiation template by double clicking on *view HDL instantiation template* in "Processes" window under *Core Generator*.


ISE Project Navigator (P.28xd) - C:\Users\Randil\Documents\Xilinx_project
File Edit View Project Source Process Tools Window Layout
Design
View: Implementation Simulation
Hierarchy
No Processes Running

```
seven_seg
xc7a100t-1csg324
    sev_seg (sev_seg.v)
    sev_seg_with_clk_top (sev_seg_with_clk_top.v)
        mod_1 - sev_seg_with_clk (sev_seg_with_clk.v)
        instance_name - ip_clk_div (ip_clk_div.xco)
        sev_seg_with_clk_top.ucf
```

Processes: instance_name - ip_clk_div

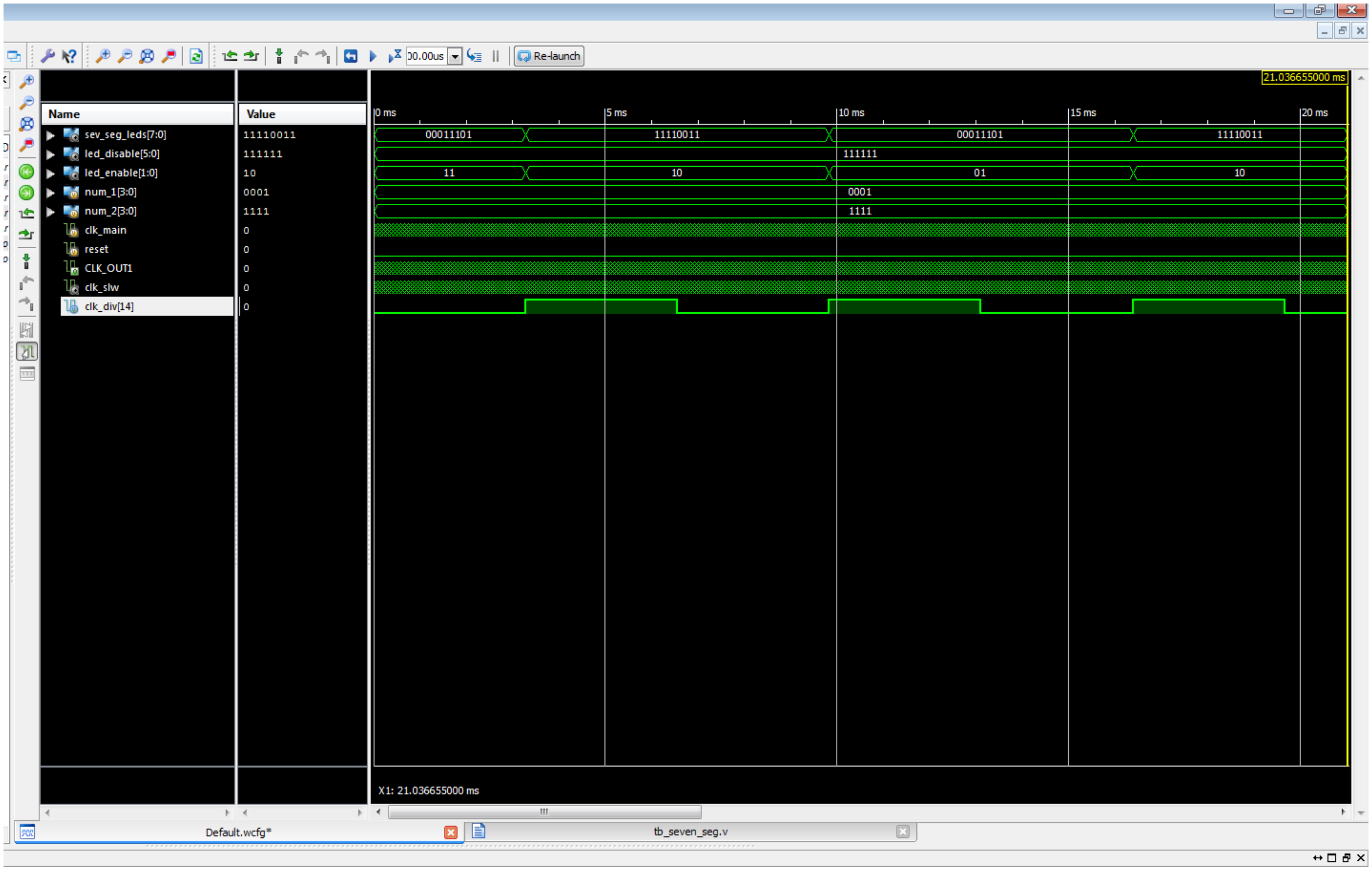
```
CORE Generator
    Manage Cores
    Regenerate Core
    Update Core to Latest Version
    View HDL Functional Model
    View HDL Instantiation Template
```



```

//----------- Begin Cut here for INSTANTIATION Template ---// INST_TAG

  ip_clk_div instance_name
   (// Clock in ports
    .CLK_IN1(CLK_IN1),      // IN
    // Clock out ports
    .CLK_OUT1(CLK_OUT1));    // OUT
// INST_TAG_END ------ End INSTANTIATION Template ---------

```

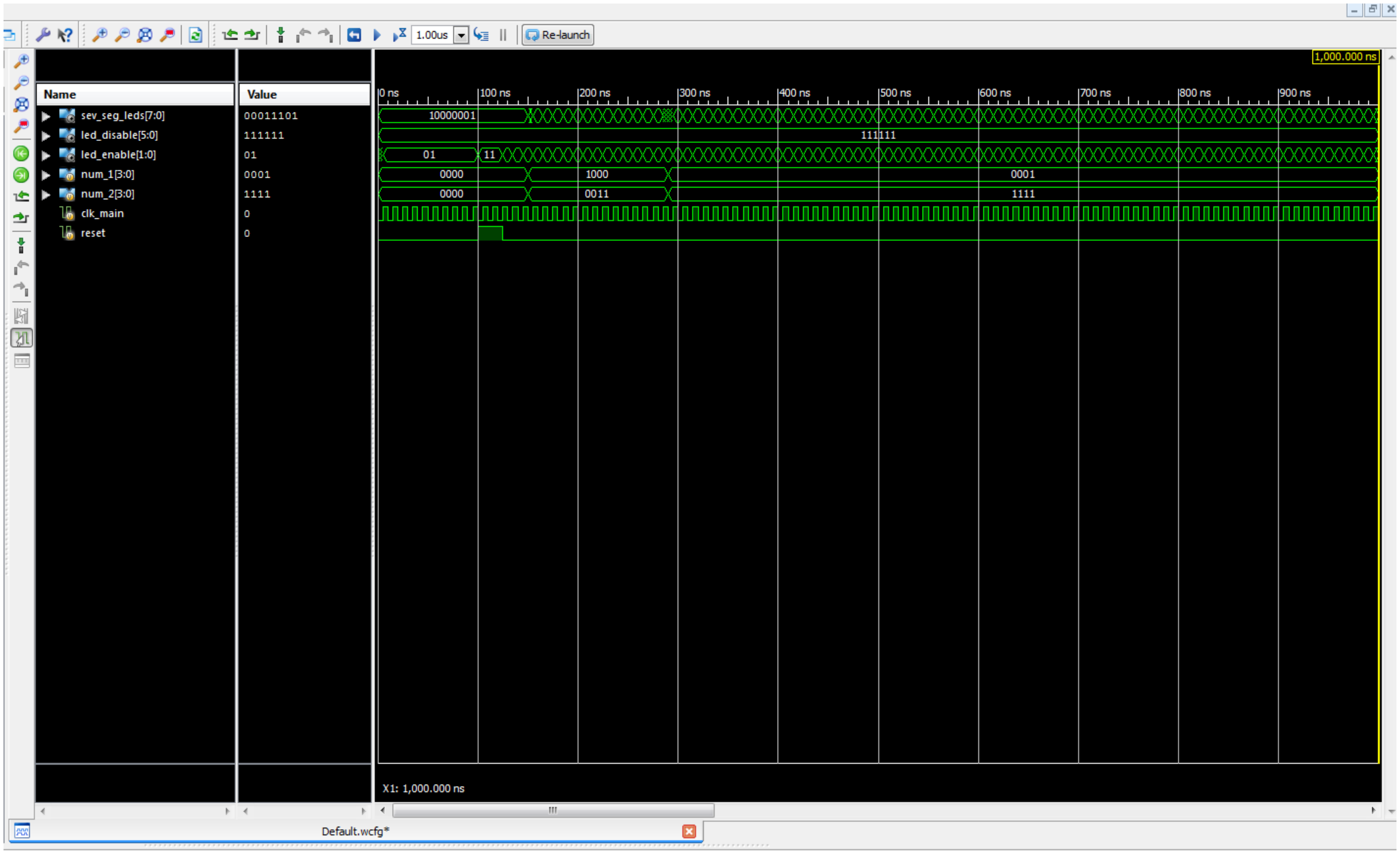
Now, compare the instantiation code in the *sev_seg_with_clk_top.v* file with the instantiation template. You can see that we instantiate a clock generator module, name it *clk_5M* and assign its input and output clock pins to wires *clk_main* and *clk_slw*, respectively. You will follow the same procedure to instantiate core generated modules in your future designs. Now, synthesize and implement your design, generate the bit file and download it on your board. You will see that the leftmost 4 switches control the leftmost display, rightmost 4 switches control the one next to it, and the middle pushbutton labeled BTNC turns off all.

A simple testbench file is also provided along with the other files if you want to simulate the module. **However, there is an important point regarding the simulation of the modules that contain clock dividers**: Please check the simulation screenshot on the next page. It is taken from the functional simulation of the *sev_seg_with_clk_top.v* module with a 100 MHz clock generated in the provided testbench file. The last row shows *clk_div[14]*, which is the clock signal that we supply to the *sev_seg_with_clk.v* module in order to switch between the displays. Note that the simulation shows only 3 periods of that signal, which correspond to only 3 display switching events (see the *led_enable* signal), although it was run for several minutes. This happens since the simulator makes calculations at every edge of the *clk_main* signal, which is a 100 MHz clock. Our module takes this clock and generates a 5 MHz clock (*clk_slw*), which is then further divided by 2^{15}. Note that the first part (5 MHz clock) is done by IP Core generator, and the second part (clock divider) is a very simple block. Therefore, you may want to skip simulating these blocks and instead, simulate more critical parts of your circuit at a faster rate. In this case, you can simply feed the main clock into your critical blocks to simulate its operation. The second screenshot shows the simulation results of the same circuit with *clk_main* sent to *sev_seg_with_clk* module instead of *clk_div[14]*. The only modification to the code was in the instantiation of the module (we changed *.clk(clk_div[14])* to *.clk(clk_main)*). It takes less than a few seconds to run, and yet there are tens of display switching events this time. In summary, if the clock dividers cause excessively long simulation times, you can feed the main clock directly to your other blocks for simulation purposes. Try to do this with minimum changes in your code, and do not forget to change your code back once your simulations are done.



```
Name              Value
sev_seg_leds[7:0] 11110011   00011101 | 11110011 | 00011101 | 11110011
led_disable[5:0]  111111     111111
led_enable[1:0]   10         11 | 10 | 01 | 10
num_1[3:0]        0001       0001
num_2[3:0]        1111       1111
clk_main          0
reset             0
CLK_OUT1          0
clk_slw           0
clk_div[14]       0

X1: 21.036655000 ms
```

Simulation result of the full module with IP clock generator and clock divider



Name	Value
sev_seg_leds[7:0]	00011101
led_disable[5:0]	111111
led_enable[1:0]	01
num_1[3:0]	0001
num_2[3:0]	1111
clk_main	0
reset	0

X1: 1,000.000 ns

Simulation result of the module with only the main clock