Xilinx ISE tutorial

(Szántó Péter, Csordás Péter 2013-09-06)

Required knowledge

- Digital design.
- Basic Verilog knowledge.

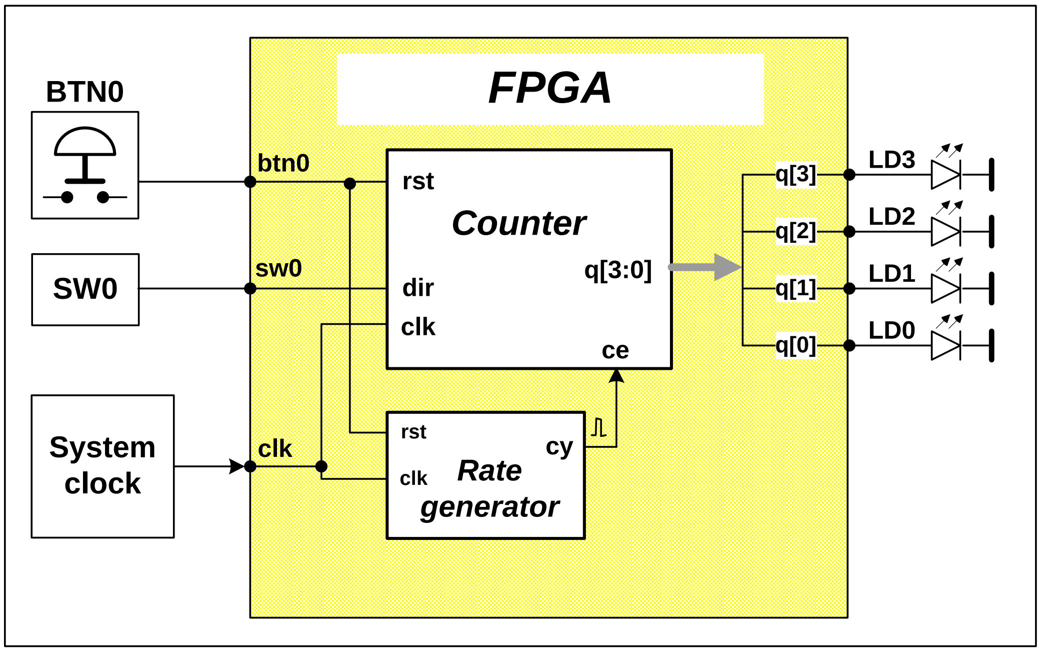
1. Tutorial project

For the Xilinx ISE tutorial the goal is to implement a one-digit BCD (binary coded decimal) second counter. That is, the counter counts from zero to nine and increments once every second. The actual value is displayed on 4 LEDs. Based on the state of the SW0 switch, the counter counts either upwards or downwards.

During Laboratory 1., it is required to design synchronous systems, that is all flip-flops in the design should operate using the 16 MHz clock input available on the development board. If part of the design should operate less frequently (just like the counter in this tutorial), an enable signal should be generated. For example, for the second counter, this enable signal has a frequency of 1 Hz – it takes value of "1" once in every second for exactly one system clock cycle long. That is, the enable signal is "0" for 15,999,999 clock cycles, then it goes to "1" for a single clock cycle.

To be able to set the system into a known state, an external reset signal should be present.

The schematic diagram of the system to be designed is shown below. The counter module is the BCD counter, while rategen generates the necessary enable signal for the counter which was described above. These two submodules are connected together in the top-level module.

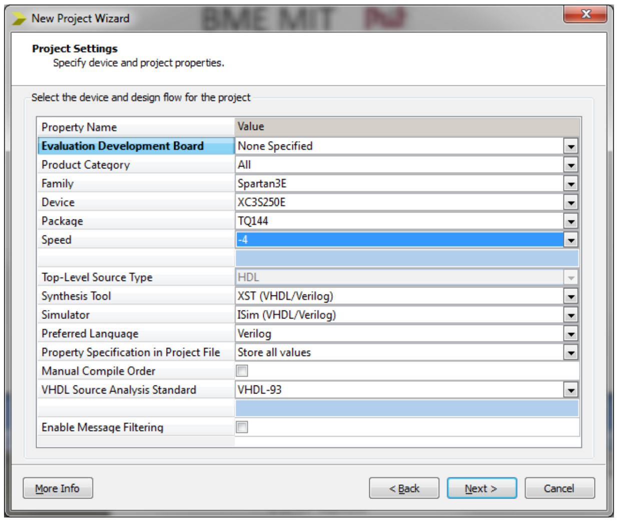


2. Creating the project

Click on the Xilinx Project Navigator icon or use the start menu item: **`Programs /Xilinx ISE Design Suite 13.2 / ISE Design Tools/ Project Navigator`**..

`File` menu, select **`New Project`** option.

- Set **`Project Name`** to *wpbev*. Create a folder on drive D for the project (do not use space or any special character in the folder name, otherwise ISE does not work correctly!).
- In the **`Top-Level Module Type`** box select **`HDL`** as the type of the top-level module.



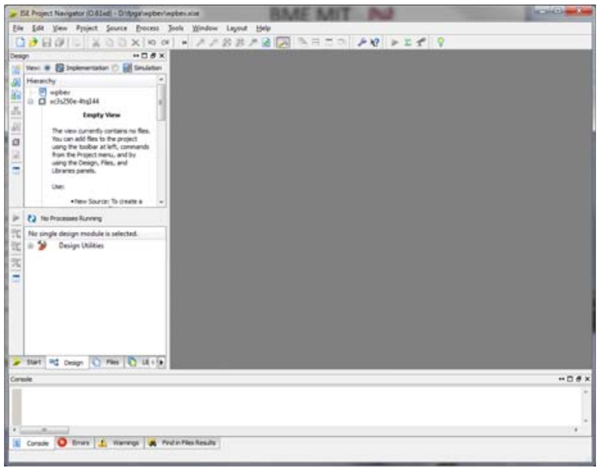
Project Settings
Specify device and project properties.

Select the device and design flow for the project

Property Name	Value
Evaluation Development Board	None Specified
Product Category	All
Family	Spartan3E
Device	XC3S250E
Package	TQ144
Speed	-4
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISim (VHDL/Verilog)
Preferred Language	Verilog
Property Specification in Project File	Store all values
Manual Compile Order	☐
VHDL Source Analysis Standard	VHDL-93
Enable Message Filtering	☐

After pressing the **`Next`** button, set the FPGA type:

> **Device Family:** Spartan3E
> **Device:** xc3s250E
> **Package:** tq144
> **Speed Grade:** -4
> **Synthesis Tool:** XST (VHDL/Verilog)
> **Simulator:** ISim (VHDL/Verilog)



After pressing **`Next`** and **`Finish`**, you will have a blank project

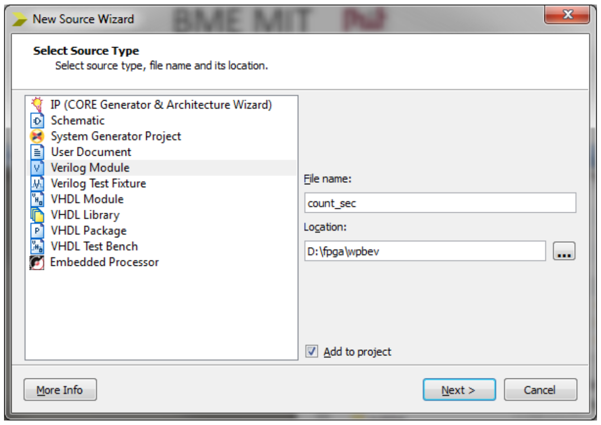
The main Window of the Project navigator consists of 4 windows:

- Top left: **`Hierarchy`** window: sources of the current project. Using the radio buttons, you can select to show source files for Implementation or for Simulation.
- **`Processes`** window shows the allowed processes to be ran using the source file selected in the Sources window.
- The **`Editor`** window is the Xilinx text editor to edit different kind of source files.
- At the bottom, the **`Console`** window shows messages from the tool. The **`Console`** view lists all messages, while **`Warnings`** and **`Errors`** lists only the selected messages..

.

3. *Verilog code for the counter module*

To create a new module select **`Project/New Source.`** In the menu select **`Verilog Module`**. The name of the module should be *count_sec*, (**`File Name`** window), and click the **`Add to Project`** option. After pressing the **`Next`** button you can define the ports of the module.

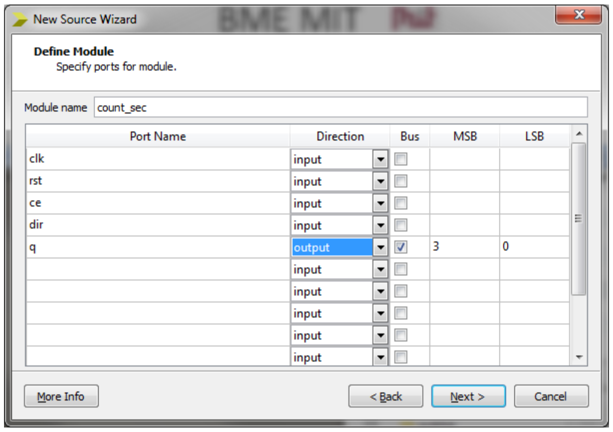


File name: count_sec

Location: D:\fpga\wpbev

[✓] Add to project

Using the schematic diagram above, create the necessary ports for the counter.



Module name: count_sec

Port Name	Direction	Bus	MSB	LSB
clk	input	☐		
rst	input	☐		
ce	input	☐		
dir	input	☐		
q	output	✓	3	0
	input	☐		
	input	☐		
	input	☐		
	input	☐		
	input	☐		

Press **`Next`**, **`Finish`** to see the module declaration in the editor window:

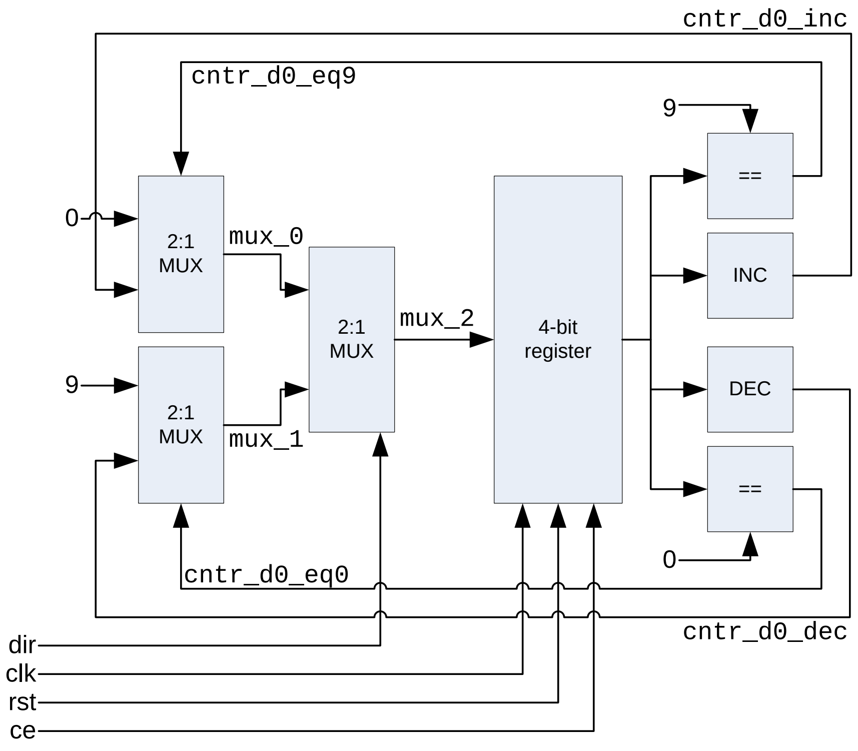
```verilog
module count_sec(
    input clk,
    input rst,
    input ce,
    input dir,
    output [3:0] q
);

endmodule
```

The counter is realized with a 4-bit register (D FF), which has reset and enable inputs. The INC and DEC blocks generate the incremented and decremented values based on the register output. The content of the register is updated only when the enable input (ce) is ‘1’. Based on the actual value, the nexz value should be:

- counting upwards (*dir = ’1’*)
    - o if the counter reached its end value (9): 0
    - o otherwise actual value + 1
- counting downwards (*dir = ’0’*)
    - o if the counter reached its end value (0): 9

- otherwise actual value -1

To detect the end-value, two combinatorial comparators are used, which compare the output of the register with 9 and 0. Based on these, the schematic diagram of the counter looks as follows:



In Verilog, it is possible to describe the exact same structure (for the register reg typed variables are declared, for the combinatorial part wire variables are used).

```
reg [3:0] cntr_d0;
wire [3:0] cntr_inc, cntr_dec;
wire [3:0] mux_0, mux_1, mux_2;
wire cntr_d0_eq0, cntr_d0_eq9;

always @(posedge clk)
if (rst)
   cntr_d0 <= 0;
else if (ce)
   cntr_d0 <= mux_2;

assign cntr_d0_inc = cntr_d0 + 1;
assign cntr_d0_dec = cntr_d0 - 1;
assign cntr_d0_eq0 = (cntr_d0 == 0);
assign cntr_d0_eq9 = (cntr_d0 == 9);

assign mux_0 = (cntr_d0_eq9) ? 0 : cntr_d0_inc;
assign mux_1 = (cntr_d0_eq0) ? 9 : cntr_d0_dec;
assign mux_2 = (dir) ? mux_0 : mux_1;
```

The only *always* block contains the rising edge of the clock signal in its event list, therefore after implementation D FFs will be generated. Updating the register content is allowed only if the enable signal (ce) is ‘1’.

Verilog is a relatively high level language, therefore it is possible to create a functionally equivalent, but more user-friendly code.

```
reg [3:0] cntr_d0;
wire cntr_d0_eq0, cntr_d0_eq9;

always @(posedge clk)
if (rst)
   cntr_d0 <= 0;
```

```
else if (ce)
   if (dir)           //DIR=1: count up
      if (cntr_d0_eq9)
         cntr_d0 <= 0; //overflow
      else
         cntr_d0 <= cntr_d0 + 1;
   else                //DIR=0: count down
      if (cntr_d0_eq0)
         cntr_d0 <= 9;
      else
         cntr_d0 <= cntr_d0 - 1;

assign cntr_d0_eq0 = (cntr_d0 == 0);
assign cntr_d0_eq9 = (cntr_d0 == 9);
```

After creating the Verilog code, select the count_sec module in the Sources window. In the Processes window double-clock on the **`Synthesize-XST / Check Syntax,`** which runs syntax checking. If there are errors, correct them before moving on.

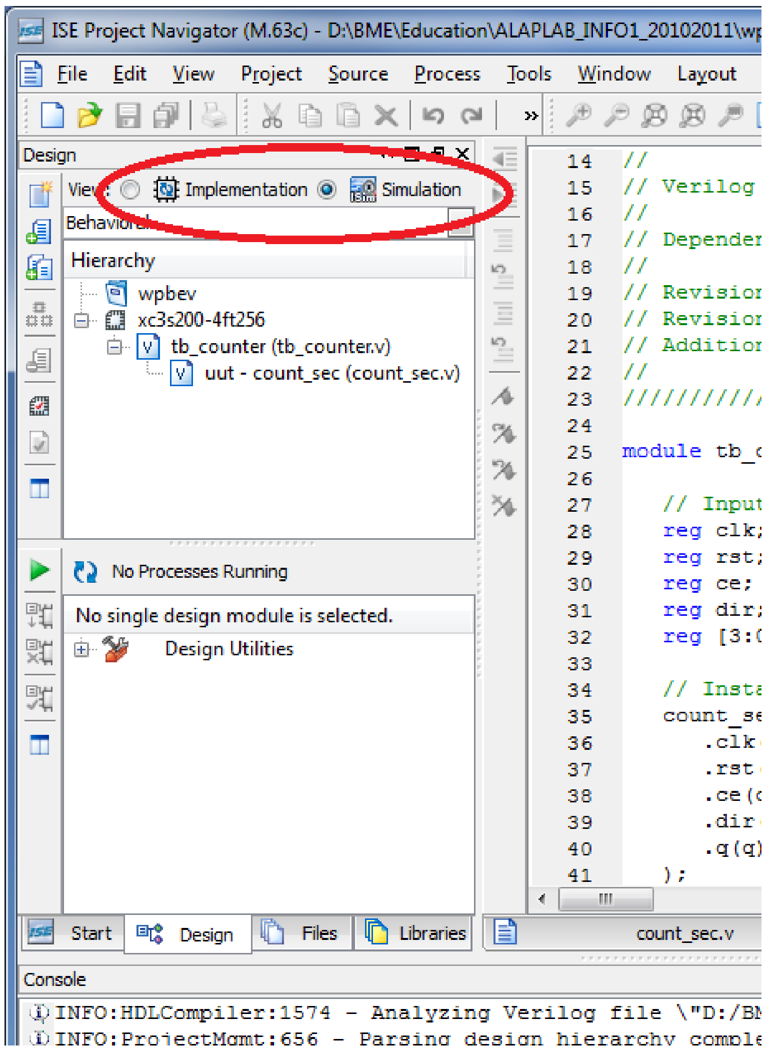
4. Simulation

4.1 Creating the testbench

To be able to functionally verify the module you have just created, you have to generate input signals for the module. This is done in the so-called testbench. Firstly, create a Verilog Test Fixture using **`Project / New Source`** and selecting **`Verilog Test Fixture`** as the type of the source. The file name should be **`tb_counter.`**

In the next window select the module you want to create a testbench for. In our case we have only one module (count_sec), so select it.

Press **`Next, Finish`** to have the file generated. In the top-left window select **`Simulation`** to see the source which are only valid for simulation. As you can see, the module to be tested (count_sec) is a submodule for the testbench (tb_counter).


ISE Project Navigator (M.63c) - D:\BME\Education\ALAPLAB_INFO1_20102011\wp
File Edit View Project Source Process Tools Window Layout
Design
View: Implementation Simulation
Behavioral
Hierarchy
wpbev
xc3s200-4ft256
tb_counter (tb_counter.v)
uut - count_sec (count_sec.v)
No Processes Running
No single design module is selected.
Design Utilities
Start Design Files Libraries
count_sec.v
Console

```
//
// Verilog
//
// Depender
//
// Revisior
// Revisior
// Additior
//
//////////

module tb_c

    // Input
    reg clk;
    reg rst;
    reg ce;
    reg dir;
    reg [3:(

    // Insta
    count_se
        .clk
        .rst
        .ce(c
        .dir
        .q(q)
    );
```

```
INFO:HDLCompiler:1574 - Analyzing Verilog file \"D:/BM
INFO:ProjectMgmt:656 - Parsing design hierarchy comple
```

4.2 Creating input signals

The automatically generated testbench file contains the following:

- instantiation of the module to be tested with the name UUT (unit under test)
- reg type variables for the input signals
- wire type variables for the output signals
- all inputs are set to 0 at the beginning of the simulation (initial block)

```
`timescale 1ns / 1ps
module tb_conter;

// Inputs
reg clk;
reg rst;
reg ce;
reg dir;


// Outputs
wire [3:0] q;

// Instantiate the Unit Under Test (UUT)
count_sec uut (
      .clk(clk),
      .rst(rst),
      .ce(ce),
      .dir(dir),
      .q(q)
);

initial begin
      // Initialize Inputs
      clk = 0;
      rst = 0;
      ce = 0;
      dir = 0;

      // Wait 100 ns for global reset to finish
      #100;

      // Add stimulus here

end

endmodule
```

As the counter contains sequential logic, a clock signal should be generated:

```
always #5
   clk <= ~clk;
```

For the other inputs generate the following waveforms.

- set *rst* to '1' during 7 – 27 ns
- set *ce* to '1' after 107 ns
- set *dir* to '1' after 1007 ns

These can be generated with the following code:

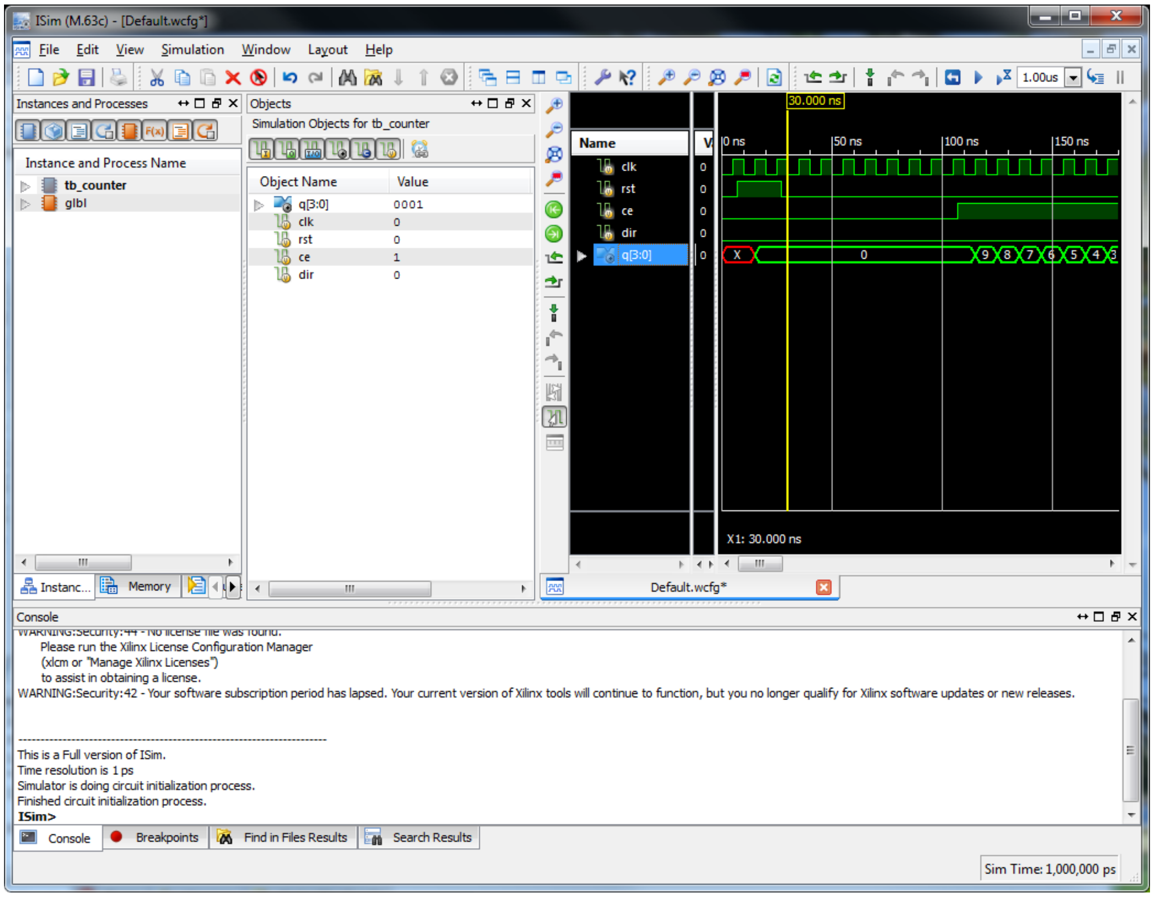
```
initial
begin
   #7 rst <= 1;
   #20 rst <= 0;
end

initial
   #107 ce <= 1;

initial
   #1007 dir <= 1;
```

The delays in one initial blocks will be added, but all the initial blocks run simultaneously.

In the Sources window select the testbench, than in the processes window double click on the **`Simulate Behavioral Model`** option to start the simulator.

4.3 Using ISE simulator

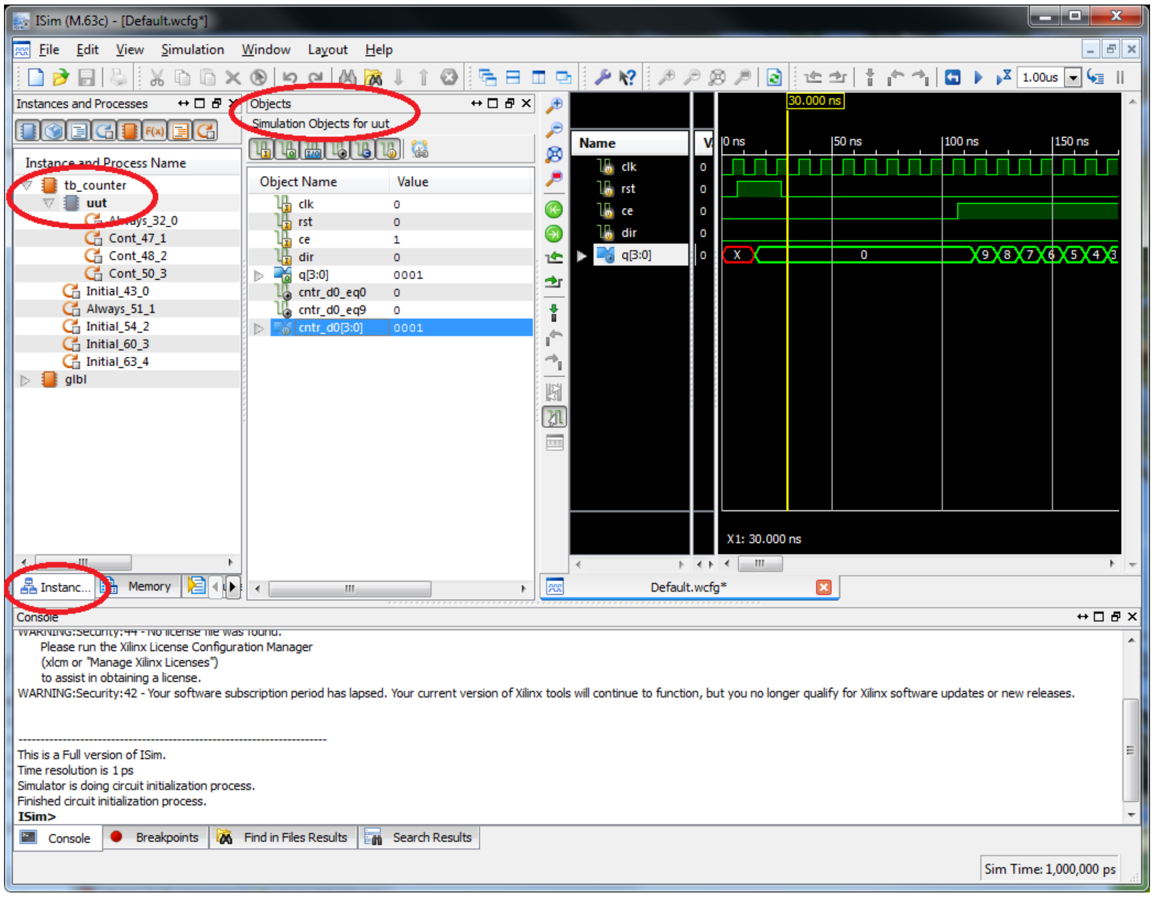
Simulation results are presented in a new window. Note, that before the reset, registers contain undefined values (red X).



```
WARNING:Security:44 - No license file was found.
    Please run the Xilinx License Configuration Manager
    (xlcm or "Manage Xilinx Licenses")
    to assist in obtaining a license.
WARNING:Security:42 - Your software subscription period has lapsed. Your current version of Xilinx tools will continue to function, but you no longer qualify for Xilinx software updates or new releases.


----------------------------------------------------------------------
This is a Full version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
ISim>
```

By right clicking on the signals you can change the display format – set q[3:0] to hexadecimal.

The simulator only shows the inputs and output automatically. However, often it is necessary to inspect internal signals of the system. Internal signals can be added to the simulator waveform without modifying the Verilog code.Select **`Instances and Processes`** tab in the left window to see the testbench hierarchy. In the hierarchy select the counter instance (UUT), then select the Objects tab. Here you can see all internal signals of the counter, which can be drag-and-dropped into the waveform window. After restarting the simulation (restart/play icons at the top) you will see the values of the newly added signals.



ISim (M.63c) - [Default.wcfg*]

Object Name	Value
clk	0
rst	0
ce	1
dir	0
q[3:0]	0001
cntr_d0_eq0	0
cntr_d0_eq9	0
cntr_d0[3:0]	0001

```
WARNING:Security:44 - No license file was found.
      Please run the Xilinx License Configuration Manager
      (xlcm or "Manage Xilinx Licenses")
      to assist in obtaining a license.
WARNING:Security:42 - Your software subscription period has lapsed. Your current version of Xilinx tools will continue to function, but you no longer qualify for Xilinx software updates or new releases.


------------------------------------------------------------------------
This is a Full version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
ISim>
```

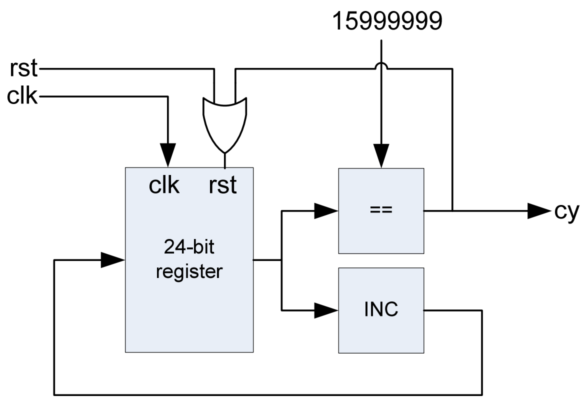
Sim Time: 1,000,000 ps

4.4. Rategen module

Create a new module with a name rategen (**`Project / New Source)`**.

The rate genertaror which generates the enable signal for the counter is basically a counter which counts from 0 to 59,999,999 to be able to have 16,000,000 different states. To be able to do that, a 24 bit counter is required. The end-value of this counter will be the enable signal for the BCD counter.

The schematic diagram is:



The corresponding Verilog code:

```
module rategen(
    input clk,rst,
    output cy
    );
    //Generate 1 clock wide pulse on output CY
reg [23:0] Q;

always @(posedge clk)
begin
   if (rst | cy)
      Q <= 0;
   else
      Q <= Q + 1;
end

assign cy = (Q == 15999999);
//assign cy = (Q == 4);

endmodule
```

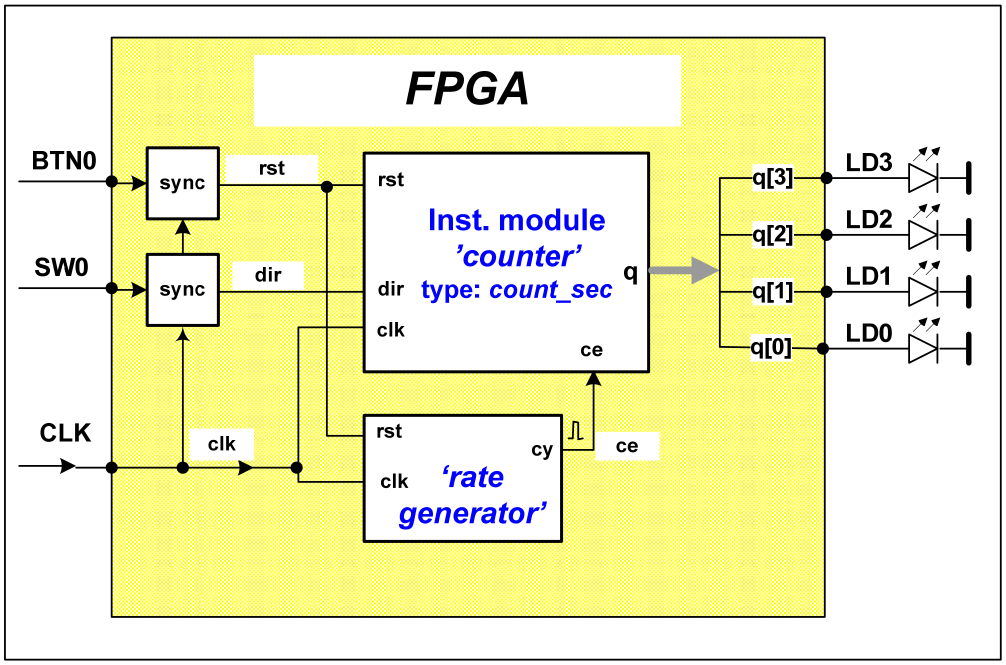
To make simulation faster you can use a more frequent enable signal which is shown commented in the code above.

4.5. Top-level module

The top-level module describes the connection between its sub-modules and connection to the outer world (that is, connection to the FPGA pins).

Create a new Verilog source named wpbevtop1 (**`Project / New Source / Verilog module`**).

To define its ports, take a look at the schematic below.



The top level declaration therefore should look like the code below:

```
module wpbevtop1(
    input clk,btn0, sw0,
    output [3:0] q
    );
```

The external asynchronous signals (button, switch) are synchronized to the system clock by sampling them into registers:

```
reg rst, dir;
always @(posedge clk)
//Synchronize inputs
begin
   rst <= btn0;
   dir <= sw0;
end
```

The next task is to instantiate the submodules, namely create one instance from rategen and one instance from count_sec.

Instantiation of rategen:

```
wire ce;
rategen rategenerator(
   .clk(clk),
   .rst(rst),
   .cy(ce)
);
```

Instantiation of the counter:

```
count_sec counter(
   .clk(clk),
   .rst(rst),
   .ce(ce),
   .dir(dir),
   .q(q)
);
```

So, with the input synchronization and all submodule instantiation the top-level module code is:

```
module wpbevtop1(
    input clk,btn0, sw0,
    output [3:0] q
    );

reg rst, dir;
always @(posedge clk)
//Synchronize inputs
begin
   rst <= btn0;
   dir <= sw0;
end

wire ce;
rategen rategenerator(
   .clk(clk),
   .rst(rst),
   .cy(ce)
);

count_sec counter(
   .clk(clk),
   .rst(rst),
   .ce(ce),
   .dir(dir),
   .q(q)
);

endmodule
```

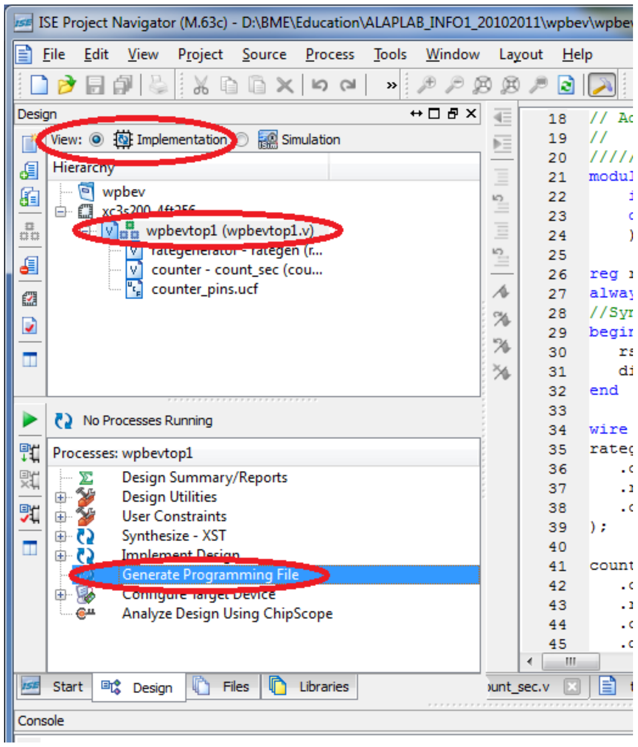
The ports of the top-level module must be assigned to pins of the FPGA. Firstly, create an Implementation Constraint File (.ucf) with **`Project / New Source`** menu, and select **`Implementation Constraint File`**. Name it to *counter_pins*.

The peripheral – pin name associations can be read directly from the development board (e.g. clock is connected to pin p56). The association should be typed in the ucf file in the following form:

```
NET "clk"  LOC="p56";
NET "btn0" LOC="p38";
NET "sw0"  LOC="p101";
NET "q[3]" LOC="p53";
NET "q[2]" LOC="p54";
NET "q[1]" LOC="p58";
NET "q[0]" LOC="p59";
```

5. Implementing the design

Select the top-level Verilog module in the Sources window. Then, in the Processes window click on the Generate Programming File option to implement the design and generate the configuration file.



In the Lab, we use a unique download cable, which emulates a standard Xilinx cable. Besides implementing the JTAG interface, our cable serves as power source for the board and presents serial communication interface. The cable driver program must be started separately: After starting **`XilinxUSBCable.exe`**, you get a taskbar icon. After right-clicking select **`Open selected device`**, **`Enable 5V power`** and **`Attach Xilinx USB cable`**. After these steps, the iMPACT downloader can be started from the ISE.

In order to start iMPACT, select **`Configure Target Device`** in the **`Process`** window. Configure the download process with the following steps:

- Select **`Boundary Scan`** in the top-left window, right click and **`Initialize Chain`** (or press Ctrl-I) in the right window!
- Click **`Yes`**, and browse for the *.bit* file to download. (*wpbevtop1.bit*)!
- We do not use external ROM, so you can answer **`No`** for the next question, and click **`Ok`** after that.
- Select the FPGA symbol in the JTAG chain with right click and click the **`Set Target Device`** option!
- Selecting **`Program`** in the bottom-left window, you can download the FPGA configuration.
- While closing iMPACT, you should save the download iMPACT project file. Return to ISE and associate this file with your project: right click on **`Configure Target Device`** select **`Process`**

`Properties`, set **`iMPACT project file`**. After this, you can download your configuration simply double-clicking **`Configure Target Device`**. (iMPACT runs unattended based on your download configuration.)

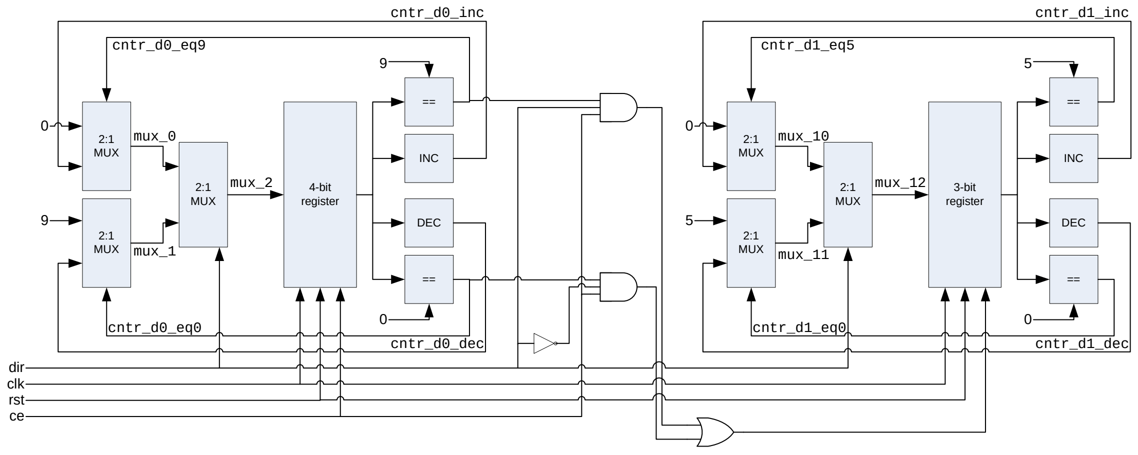
6. Modifying the counter

At this point you should have a correctly operating one-digit BCD counter. The next step is to modify the source code of the counter to have a real second counter which is able to count from 0 to 59, still in BCD format. For this, two BCD digits are required: the least significant one counts from 0 to 9, while the most significant one counts from 0 to 5. Till now, you have created the LS digit.

Let us think about the operation of the MS digit. Unlike the LS digit, which counts whenever its enable signal is '1', the MS digit is only enabled when the LS digit reached its final value: 9 in the case of up-counting and 0 in the case of down-counting. Thus, the MS digit should be only changed if:

- counting upwards (*dir='1'*), LS digit is 9 (*cntr_d0_eq9='1'*) and the external enable signal is active (*ce='1'*) OR
- counting downwards (*dir='0'*), LS digit is 0 (*cntr_d0_eq0='1'*) and the external enable signal is active (*ce='1'*)

Based on these, the resulting schematic diagram looks as follows:



Modify the Verilog sources to implement the 2-digit counter:

- In the count_sec module declare the necessary variables and describe the functionality of the MS digit IN A SEPARATE always block. (It can be derived from the existing block, as described before.)
- Increase the output port width in the counter and in the top-module.
- Connect the new output bits to LEDs 6 – 4

Simulate the modified count_sec module!

Implement the design and configure the FPGA!