Summary

The Lattice iCEcube2 development software provides a complete FPGA implementation environment for today's FPGA designers. This application note details the basic design and simulation flow using Mentor® ModelSim® simulator.

Introduction

The sample design used in this application note is a simple 4-bit binary up-counter with an associated testbench. The counter design, counter.vhd, is presented first:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity counter is port (
    clk : in  std_logic;
    reset : in  std_logic;
    count : out  std_logic_vector (3 downto 0));
end counter;

architecture behavioral of counter is
    signal q : std_logic_vector (3 downto 0);
begin
    process(clk, reset)
    begin
        if(reset = '1') then
            q <= (others=>'0');
        elsif(clk'event and clk = '1') then
            q <= q + 1;
        end if;
    end process;
    count <= q;
end behavioral;
```

The testbench design, count_tb.vhd, is presented next:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity count_tb is
end count_tb;
```
architecture testbench_arch of count_tb is

    component counter
        port (
            clk : in std_logic;
            reset : in std_logic;
            count : out std_logic_vector (3 downto 0));
    end component;

    signal clk : std_logic := '0';
    signal reset : std_logic := '0';
    signal count : std_logic_vector (3 downto 0) := "0000";

    constant period : time := 100 ns;
    constant duty_cycle : real := 0.5;
    constant offset : time := 100 ns;

    begin

        uut : counter
            port map (
                clk => clk,
                reset => reset,
                count => count);

        process -- clock generation
            begin
                wait for offset;
                clock_loop : loop
                    clk <= '0';
                    wait for (period - (period * duty_cycle));
                    clk <= '1';
                    wait for (period * duty_cycle);
                end loop clock_loop;
            end process;

        process -- reset generation
            begin
                reset <= '0';
                -- ------------ Current Time: 0ns
                wait for 100 ns;
                reset <= '1';
                -- ------------ Current Time: 100ns
                wait for 35 ns;
                reset <= '0';
                -- ------------ Current Time: 135ns
                wait for 1865 ns;
                -- ------------ Current Time: 2000ns
            end process;

    end testbench_arch;
This document explains the following scenarios:

1. Pre-Synthesis Simulation.
3. Place-and-Route Functional Simulation (Verilog).
4. Place-and-Route Timing Simulation (Verilog).
5. Place-and-Route Functional Simulation (VHDL).
6. Place-and-Route Timing Simulation (VHDL)
Pre-Simulation Simulation

This section details the steps required for pre-synthesis simulation.

1. Create a new folder and copy the design (counter.vhd) and testbench (count_tb.vhd).
2. Start ModelSim and create a new project by choosing File→New→Project. Browse to the newly created project folder and enter a project name. Click “OK”.

   Figure 1: Creating a New Project in ModelSim

3. Click “Add Existing Files” and add counter.vhd and count_tb.vhd to the project. Close the “Add items to the Project” window.

   Figure 2: Adding Files to Project

4. Choose Compile→Compile All.
5. Choose Simulate→Start Simulation.

6. Expand the work folder and highlight count_tb. Clock OK.

7. Type the command “view wave” and press the enter key. A new waveform window will show up.
8. Type the command “add wave *” and press the enter key. All I/O signals in the design will show up in the waveform window.
9. Type the command “run 2us” and press enter. The 2uS simulation waveform will show up in the waveform window.
10. Click the “unlock” button to unlock the waveform window for a better view.
11. Click “count” to expand this bus. Right-click “count” and choose Radix $\rightarrow$ unsigned to change the bus number radix. Click “Zoom Full” to have a better view.

**Generating Files Required for Post-Synthesis and Post-Route Simulations**

This section details the steps for generating the files that are required for performing post-synthesis and post-route simulations.

1. Start iCEcube2 and create a new project by clicking Project $\rightarrow$ New Project. Browse to the project folder, enter a new project name, and select an appropriate part. Click “Next”.
2. Highlight counter.vhd and click “>>” to add counter.vhd to the right panel. Click on “Finish”.

3. Click the “Launch Synthesis Tool” button to launch Synplify Pro® for synthesis. To generate a post-synthesis simulation model in Verilog/VHDL, set the “Write Mapped Verilog Netlist/Write Mapped VHDL Netlist” option in the synthesis tool by going through Implementation Options → Implementation Results, before doing synthesis. Click RUN in Synplify Pro to synthesize the design.
4. Close Synplify Pro after synthesis. This will bring you back to the iCEcube2 tool window. The synthesis outputs “PRJNAME.edf” and “PRJNAME.scf” are automatically added to “Select Implementation” in “P&R Flow” of iCEcube2. Click “Run All” to run placement, routing and Bitmap Generation.

After running the complete flow, the following outputs will be generated.
- PRJNAME.vm, post-synthesis simulation Verilog model
- PRJNAME.vhm, post-synthesis simulation VHDL model
- counter_sbt.v, post-route timing simulation Verilog model
- counter_sbt.vhd, post-route timing simulation VHDL model
- counter_sbt.sdf, for post-route timing Verilog model simulation
- counter_sbt_vital.sdf, for post-route timing VHDL model simulation

Post-synthesis simulation models can be found typically in PRJNAME/PRJNAME_Implmnt/. These are generated only if you keep the settings mentioned in Step 3 during synthesis.
The post-synthesis simulation VHDL model PRJNAME.vhm needs to be edited to remove the following two lines for ModelSim Lattice Edition.

```vhdl
library synplify;
use synplify.components.all;
```

Post-route timing models can be found typically in `PRJNAME/PRJNAME_Implmnt/sbt/outputs/simulation_netlist`

SDF is a standardized representation of timing data commonly used when exchanging timing information between design and simulation tools.

### Post-Synthesis Functional Simulation (Verilog/VHDL)

1. Generate the files that are required for a post-synthesis functional simulation model using the steps described in the section “Generating Files Required for Post-Synthesis and Post-Route Simulations”.
2. Open ModelSim. Create new project using File → New → Project. In the New Project window, give a project name and browse to a folder where the project needs to be saved. Click OK.

![Create New Project](image)

3. Click “Add Existing Files” and add the following files:
   a. PRJNAME.vm, sb_ice_syn.v, counter_tb.vhd for Verilog post-synthesis simulation
   b. PRJNAME.vhm, vcomponent_vital.vhd, sb_ice_lc_vital.vhd, sb_ice_syn_vital.vhd, counter_tb.vhd for VHDL post-synthesis simulation

   `sb_ice_syn.v` can be found in `$INST_DIR/verilog` and `vcomponent_vital.vhd`, `sb_ice_lc_vital.vhd`, and `sb_ice_syn_vital.vhd` can be found in `$INST_DIR/VHDL`.

   If your design contains PLL, add `ABIPTBS8.v` and `ABIWTCZ4.v` in `$INST_DIR/verilog`. To perform post-synthesis simulation on a VHDL design having PLL, you need a mixed-language simulator, since the PLL model (`ABIPTBS8.v`) is available only in Verilog format. If the design contains **Hardened IP** primitives, add the encrypted Verilog simulation library `sb_ice_ipenc_modelsim.v`, available in `$INST_DIR/Verilog`.

   Close the “Add items to project” window once all the files are added.
4. Choose Compile → Compile All.
   For VHDL post-synthesis simulation, vcomponent_vital.vhd should be compiled first, then
   sb_ice_syn_vital.vhd, sb_ice_lc_vital.vhd, and other files.

5. Start simulation from Simulate → Start Simulation.

6. In the Simulation window, select the Libraries tab and add the ice_vlg resource library for Verilog
   simulation or ice resource library for VHDL simulation in the search libraries path.
7. Expand the work folder in the “Start Simulation” window and highlight “counter_tb”. Click OK.

Figure 16: Run Simulation

8. This will open the simulation window. Add the signals wave window and perform simulation as explained in the “Pre-Synthesis Simulation” section from point 7 onwards.

Place-and-Route Functional Simulation (Verilog)

1. Generate the files that are required for post-route functional simulation model using the steps described in the section “Generating Files Required for Post-Synthesis and Post-Route Simulations.”

2. Open ModelSim. Create new project using File → New → Project. In the New Project window, give a project name and browse to a folder where the project needs to be saved. Click OK.
3. Click “Add Existing Files” and add the following files:
   a. counter_sbt.v
   b. counter_tb.vhd

Add the following Verilog simulation libraries available in $INST_DIR/verilog
   a. sb_ice_syn.v
   b. sb_ice_lc.v

If your design contains PLL, add ABIPTBS8.v & ABIWTCZ4.v in $INST_DIR/verilog.
If the design contains Hardened IP primitives, add the encrypted simulation library sb_ice_ipenc_modelsim.v available in $INST_DIR/Verilog.

Close the “Add items to project” window once all the files are added.

4. Click on Compile → Compile All.
5. Start simulation from Simulate → Start Simulation.

6. In the Simulation window, select the Libraries tab and add the ice_vlg resource library in the search libraries path.

7. Expand the work folder in the “Start Simulation” window and highlight “counter_tb”. Click OK.
8. This will open the simulation window. Add the signals wave window and perform simulation as explained in the “Pre-Synthesis Simulation” section from point 7 onwards.

Place-and-Route Timing Simulation (Verilog)

1. Generate the files that are required for the post-route timing simulation model using the steps described in the section “Generating Files Required for Post-Synthesis and Post-Route Simulations”.

2. Open ModelSim. Create a new project using File → New → Project. In the New Project window, give a project name and browse to a folder where the project needs to be saved. Click OK.

   Figure 9: Create New Project

3. Click “Add Existing Files” and add the following files:
   counter_sbt.v, sb_ice_syn.v, sb_ice_lc.v, counter_tb.vhd for Verilog post-route timing simulation
   The sb_ice_syn.v, sb_ice_lc.v verilog files can be found in $INST_DIR/verilog.
   If your design contains PLL, add ABIPTBS8.v in $INST_DIR/verilog.
   If the design contains Hardened IP primitives, add the encrypted simulation library sb_ice_ipenc_modelsim.v available in $INST_DIR/Verilog.
Close the “Add items to Project” window once all the files are added.

**Figure 10: Add Files to Project**

4. In the Project Compiler Settings window, select the “Verilog & System Verilog” tab and click “Macro”. In the Define Macro window, set the macro name to “TIMINGCHECK” and set the value to 1. Click OK.

**Figure 11: Set TIMINGCHECK Macro**

5. Click on Compile → Compile All.

**Figure 12: Compile All**
6. Start simulation using Simulate → Start Simulation

![Figure 13: Start Simulation](image)

7. In the Simulation window, select the “Libraries” tab and add the “ice_timing_vlg” resource library in the search libraries path.

![Figure 27: Add “ice_timing_vlg” resource library to the project](image)

8. In the Simulation window, select the SDF tab. Add the SDF file counter_sbt.sdf and type “/count_tb/uut” in “Apply to Region”. Click OK.
9. In the Design tab, select “count_tb” under work. Make sure that the “Time Resolution” was set to PS. Click OK.

10. This will open the simulation window. Add the signals wave window and perform simulation as explained in the “Pre-Synthesis Simulation” section from point 7 onwards.

Place-and-Route Functional Simulation (VHDL)

1. Generate the files that are required for post-route functional simulation model using the steps described in the section “Generating Files Required for Post-Synthesis and Post-Route Simulations”.

2. Create a new project using File → New → Project. In the New Project window, give a project name and browse to a folder where the project needs to be saved. Click OK.
3. Click “Add Existing File” and add the following files:
   a. counter_sbt.vhd
   b. counter_tb.vhd
   Close the “Add items to the Project” window once all the files are added.

4. Click Compile → Compile All.

5. Start simulation from Simulate → Start Simulation.
6. In the simulation window, select the tab “Libraries” and add the “ice” resource library in the search libraries path.

![Image of Mentor ModelSim Simulator with Lattice iCEcube2](image1)

**Figure 33: Start Simulation**

7. Expand the work folder in the “Start Simulation” window and highlight “counter_tb”. Click OK.

![Image of Mentor ModelSim Simulator with Lattice iCEcube2](image2)

**Figure 34: Add “ice” resource library to the project**

**Figure 35: Run Simulation**
Using Mentor ModelSim Simulator with Lattice iCEcube2

This will open the simulation window. Add the signals wave window and perform simulation as explained in the “Pre-Synthesis Simulation” section from point 7 onwards.

Place-and-Route Timing Simulation (VHDL)

1. Generate the files that are required for the post-route timing simulation model using the steps described in the section “Generating Files Required for Post-Synthesis and Post-Route Simulations”.
2. Create new project using File → New → Project. In the New Project window, give a project name and browse to a folder where the project needs to be saved. Click OK.

   Figure 36: Create New Project

3. Click “Add Existing Files” and add the following files:
   a. counter_sbt.vhd
   b. counter_tb.vhd
   Close the “Add items to the Project” window once all the files are added.

   Figure 37: Add Files to Project

4. Choose Compile → Compile All.
5. Start simulation using Simulate → Start Simulation.

6. In the Simulation window, select the “Libraries” tab and add the “ice” resource library in the search libraries path.

7. In the Simulation window, select the SDF tab. Add the SDF file counter_sbt_vital.sdf and type “/count_tb/uut” in “Apply to Region”. Click OK.
8. In the Design tab, select “count_tb” under work. Make sure that the “Time Resolution” was set to PS. Click OK.

9. This will open the simulation window. Add the signals wave window and perform simulation as explained in the “Pre-Synthesis Simulation” section from point 7 onwards.

Conclusion

The Lattice iCEcube2 development software provides a complete FPGA implementation environment for today’s FPGA designers. For simulation needs, the Mentor ModelSim simulator complements the iCEcube2 development software and is a cost effective, easy to use simulation tool.

References

For more information on products, solutions, and applications enabled by Lattice Semiconductor Corporation, take the next step and visit www.latticesemi.com.
Revision History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.7</td>
<td>24-Nov-2020</td>
<td>Updated for ModelSim usage changes.</td>
</tr>
<tr>
<td>1.6</td>
<td>18-Dec-2013</td>
<td>Added resource library details for iCE40LM primitives.</td>
</tr>
<tr>
<td>1.5</td>
<td>26-APR-2013</td>
<td>Added resource library generation details for post route VHDL simulations.</td>
</tr>
<tr>
<td>1.4</td>
<td>24-DEC-2010</td>
<td>Added sections for pre/post synthesis, functional/timing, Verilog/VHDL simulations</td>
</tr>
<tr>
<td>1.3</td>
<td>23-DEC-2008</td>
<td>Updated corporate contact information.</td>
</tr>
<tr>
<td>1.2</td>
<td>20-OCT-2008</td>
<td>Initial release, updated iCEcube screen shots and timing simulation information.</td>
</tr>
<tr>
<td>1.1</td>
<td>06-AUG-2008</td>
<td>First draft, functional simulation only.</td>
</tr>
</tbody>
</table>
Using Mentor ModelSim Simulator with Lattice iCEcube2

Copyright

Copyright © 2020 Lattice Semiconductor Corporation. All rights reserved. This document may not, in whole or part, be reproduced, modified, distributed, or publicly displayed without prior written consent from Lattice Semiconductor Corporation (“Lattice”).

Trademarks

All Lattice trademarks are as listed at www.latticesemi.com/legal. Synopsys and Synplify Pro are trademarks of Synopsys, Inc. Aldec and Active-HDL are trademarks of Aldec, Inc. ModelSim and Questa are trademarks or registered trademarks of Siemens Industry Software Inc. or its subsidiaries in the United States or other countries. All other trademarks are the property of their respective owners.

Disclaimers

NO WARRANTIES: THE INFORMATION PROVIDED IN THIS DOCUMENT IS “AS IS” WITHOUT ANY EXPRESS OR IMPLIED WARRANTY OF ANY KIND INCLUDING WARRANTIES OF ACCURACY, COMPLETENESS, MERCHANTABILITY, NONINFRINGEMENT OF INTELLECTUAL PROPERTY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL LATTICE OR ITS SUPPLIERS BE LIABLE FOR ANY DAMAGES WHATSOEVER (WHETHER DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL, INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OF OR INABILITY TO USE THE INFORMATION PROVIDED IN THIS DOCUMENT, EVEN IF LATTICE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. BECAUSE SOME JURISDICTIONS PROHIBIT THE EXCLUSION OR LIMITATION OF CERTAIN LIABILITY, SOME OF THE ABOVE LIMITATIONS MAY NOT APPLY TO YOU.

Lattice may make changes to these materials, specifications, or information, or to the products described herein, at any time without notice. Lattice makes no commitment to update this documentation. Lattice reserves the right to discontinue any product or service without notice and assumes no obligation to correct any errors contained herein or to advise any user of this document of any correction if such be made. Lattice recommends its customers obtain the latest version of the relevant information to establish that the information being relied upon is current and before ordering any products.