How to Use Full-Chip RTL Simulation

SLG47921

Abstract

This application note describes how to use Full-Chip RTL Simulation in the Renesas ForgeFPGA™ series of devices (SLG47910, SLG47912, SLG47920/21). The SLG47921 is used in this example. Simulation waveforms generated by the GTKWave software are used to verify the functionality of the design.

This application note comes complete with design files, which can be found in section 6 References.

Contents

Figures

1. Introduction

This document describes how to use the Full-Chip RTL Simulation functionality of the Renesas ForgeFPGA series of devices to verify the operation of an entire digital system at the register-transfer logic (RTL) level. This approach makes it possible to trace the interaction between all modules in the silicon such as BRAMs, PLLs, LVDSs, the FPGA core among others, and detect potential errors early in the development process before prototyping with actual hardware.

2. Simple Counter (Single-Chip Simulation) Example

2.1 Overview

A Simple Counter is used as a code example to demonstrate Full-Chip RTL Simulation. It is already available in the ForgeFPGA Workshop as a ready-made project.

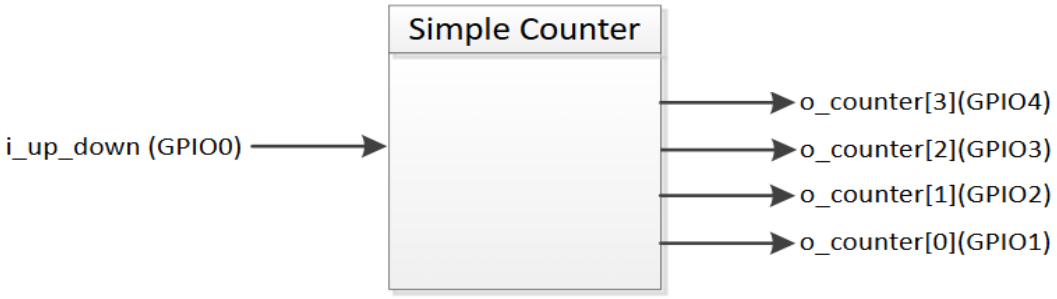

Figure 1. Block Diagram

In the **File** menu of the ForgeFPGA workshop select **Load Design Template** → **Simple Counter**. After loading the Simple Counter template, use it to run a full simulation to see how the basic digital circuit works in the Full-Chip Simulation environment.

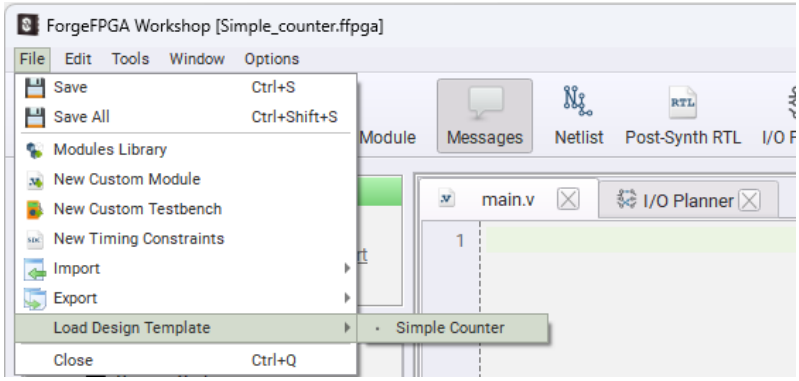

Figure 2. Simple Counter Template

The Simple Counter template includes Verilog code and also a ready-made port mapping and testbench for simulation, which is already configured in advance. Initial values can be assigned to registers in the Verilog code, as shown in Figure 3.

Note: When declaring registers in Verilog, it is recommended to assign them an initial value. Without initialization, registers are left in an undefined (X) state during the Full-Chip RTL Simulation, which can lead to incorrect results or unexpected behavior.

```
reg [3:0] r_counter_up_down;
reg [1:0] r_up_down_sync;
reg [2:0] r_rst;
```

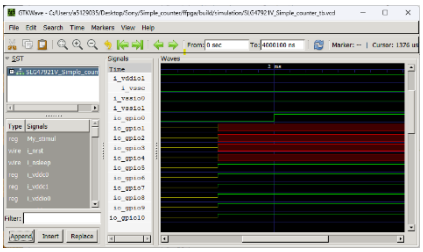

Figure 3. Incorrect Full-Chip Simulation Results due to Undefined Registers

```
// Sample 4-bit counter

// The (*top*) attribute is needed on the topmost module for synthesis
(* top *) module simplecounter(
  (* iopad_external_pin, clkbuf_inhibit *) input i_clk,      // declare input port
for clk to allow the counter to count up/down
  (* iopad_external_pin *) input        i_nreset,           // declare input port
for the reset to allow the counter to be reset
  (* iopad_external_pin *) input        i_up_down,          // declare input port
for the counter to count up or down
  (* iopad_external_pin *) output       o_up_down_oe,       // declare output port
for setting i_up_down GPIO as input
  (* iopad_external_pin *) output       o_osc_ctrl_en,      // declare output port
for enabling/disabling the on-chip oscillator
  (* iopad_external_pin *) output       o_osc_ctrl_mode,    // declare output port
for selecting 3.41MHz/50MHz on-chip oscillator mode
  (* iopad_external_pin *) output [3:0] o_counter,          // declare a 4-bit
output port to get the counter values
  (* iopad_external_pin *) output [3:0] o_counter_oe,       // declare output port
for setting o_counter GPIOs as output
  (* iopad_external_pin *) output       o_postdiv0_ctrl_en, // declare output port
for enabling/disabling on-chip oscillator posdivier0
  (* iopad_external_pin *) output [2:0] o_postdiv0_ctrl_sel  // declare a 3-bit
posdivider selector output port for the on-chip oscillator
);

  reg [3:0] r_counter_up_down = 0; // <- Registers should have an initial value
  reg [1:0] r_up_down_sync    = 0; // <- Registers should have an initial value
  reg [2:0] r_rst             = 0; // <- Registers should have an initial value
  wire      w_reset;

  // The OE (output enable) is used to define whether the GPIO operates as an
output or an input
  // assign OE = 1 to function as an output
  // assign OE = 0 to function as an input. If not mentioned, then the default
value is 0

  assign o_up_down_oe  = 1'b0;
  assign o_counter_oe  = 4'b1111;

  // Enable the on-chip oscillator by assigning 1
  assign o_osc_ctrl_en = 1'b1;

  // Select on-chip oscillator mode
  // assign o_osc_ctrl_mode = 0 correspond to 3.41MHz
  // assign o_osc_ctrl_mode = 1 correspond to 50MHz

  assign o_osc_ctrl_mode = 1'b1;

  // Enable on-chip oscillator posdivier0 by assigning 1
```

```
  // If one of the on-chip oscillator post-dividers is enabled, it automatically
forces the oscillator to operate

  assign o_postdiv0_ctrl_en = 1'b1;

  /* Select divider value for on-chip oscillator posdivier0
    3'b000 - OSC
    3'b001 - OSC/2
    --------------
    3'b110 - OSC/164
    3'b111 - OSC/128
  */
  assign o_postdiv0_ctrl_sel = 3'b000;

  // This always block synchronizes and inverts input i_reset signal
  always @(posedge i_clk) begin
    r_rst[0] <=  i_nreset;
    r_rst[1] <=  r_rst[0];
    r_rst[2] <= ~r_rst[1];
  end

  // This always block synchronizes input i_up_down signal
  always @(posedge i_clk) begin
    if (w_reset) begin
      r_up_down_sync <= 2'b00;
    end else begin
      r_up_down_sync[0] <= i_up_down;
      r_up_down_sync[1] <= r_up_down_sync[0];
    end
  end

  // Once inside this block, it checks if the w_reset is 0,
  // If yes, the counter value is 0
  // Otherwise, it checks the r_up_down_sync[1] value and counts up or down
  always @(posedge i_clk) begin
    if (w_reset) begin
      r_counter_up_down <= 4'h0;
    end else if (~r_up_down_sync[1]) begin
      r_counter_up_down <= r_counter_up_down + 4'd1; //count up
    end else begin
      r_counter_up_down <= r_counter_up_down - 4'd1; //count down
    end
  end

  assign o_counter = r_counter_up_down;
  assign w_reset   = r_rst[2];

endmodule
```

The next step is to choose the correct clock source on the CLK MUX in the main window. To do this, select MUX0 on the main window; the Simple Counter expects a clock from CLK MUX0, which is clearly visible in the I/O Planner. The Verilog code also contains the settings of POSTDIVIDER0, so it should be chosen on CLK MUX0.

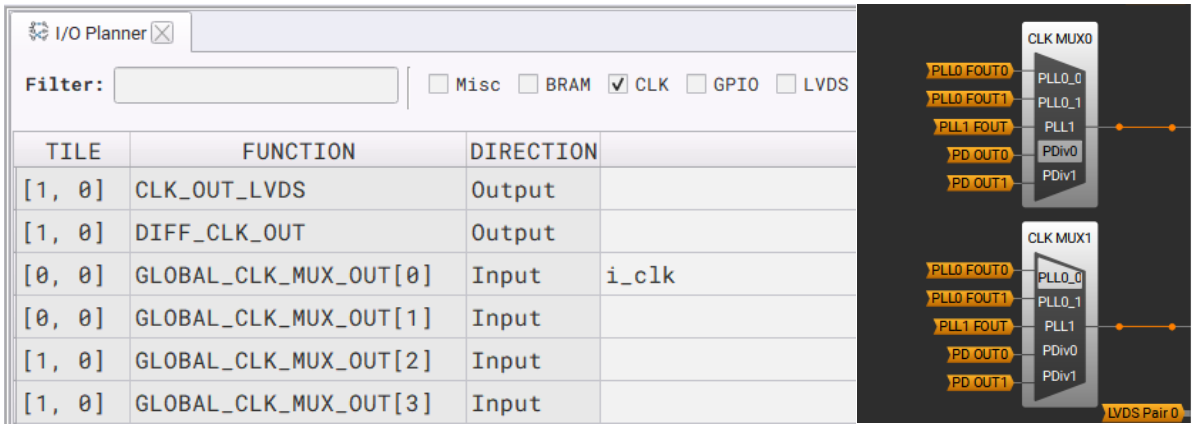

Figure 4. Clock Source Settings

After the registers are initialized and the clock source is correctly configured, the user can proceed to test the project using Full-Chip Simulation.

2.2 Design Steps

1. Synthesize the project and generate a bitstream – this step is required to build models of the entire circuit within the environment (Figure 5).

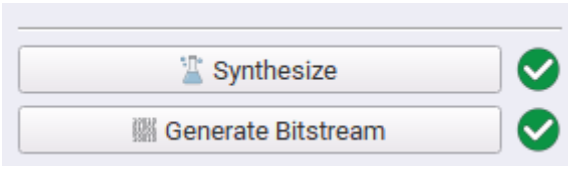

Figure 5. Synthesis and Generate Bitstream Process

2. Generate simulation models – after synthesis and bitstream generation are finished, open the **Simulation** tab and select **Generate Full-Chip Models** (Figure 6). This will automatically create the simulation models needed for verifying the design using Full-Chip RTL Simulation.

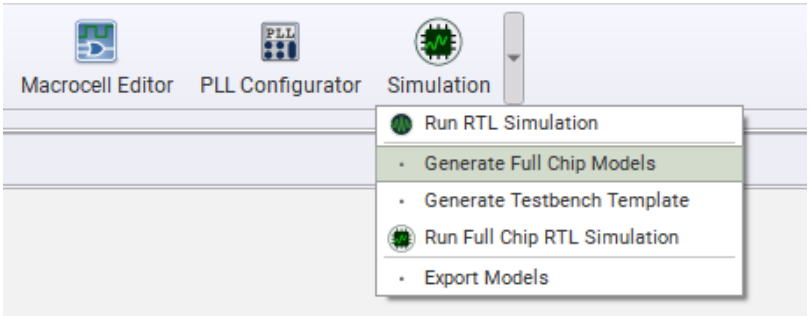

Figure 6. Generating Full-Chip Models

3. Create a testbench – before running the simulation, generate a testbench template by selecting **Generate Testbench Template** shown in Figure 6. The ForgeFPGA Workshop facilitates this process by providing the ability to generate a testbench template (*chip_tb.vt*), which can then be seen in the Source Tree (Figure 7).

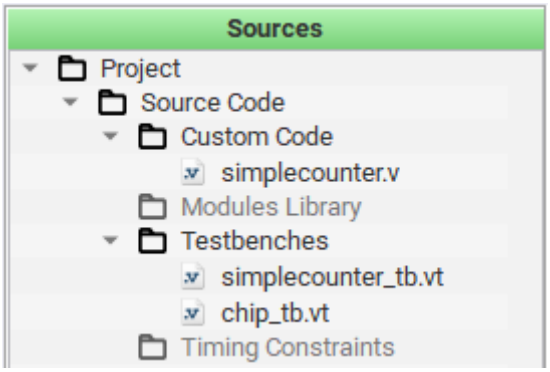

Figure 7. Source Tree

4. Add stimuli – insert your own stimuli into the generated testbench and connect them to the required GPIOs.

```
55  wire io_gpio38;
56  wire io_gpio39;
57
58  reg My_stimulus;
```

```
112 assign  i_nsleep  = r_nsleep;
113 assign  i_nrst    = r_nrst;
114 assign  io_gpio0  = My_stimulus;
115
```

Figure 8. Declaring Stimuli and Connecting Stimuli to Specific GPIOs

5. Initialize and define the stimulus behavior – in the initial 'begin ... end' section of the testbench, set initial values for all stimuli and describe how each stimuli will change over time (see Figure 9).

```
117 initial begin
118     $dumpfile ("SLG47921V_Simple_counter_tb.vcd");
119     $dumpvars (0, SLG47921V_Simple_counter_tb);
120
121     My_stimulus = 0;
122     i_vssc      = 0;
123     i_vddc0     = 0;
```

```
142     // User stimulus
143     #2000000
144     My_stimulus    = 1;
145     #2000000
146     My_stimulus    = 0;
147     // End simulation
148     $finish;
```

Figure 9. Initialization and Variation of Stimuli Over Time

6. Select **Run Full Chip RTL Simulation** (see Figure 6) – execute the Full-Chip RTL Simulation by choosing the *chip_tb.vt* testbench to test the operation of the project (see Figure 10).

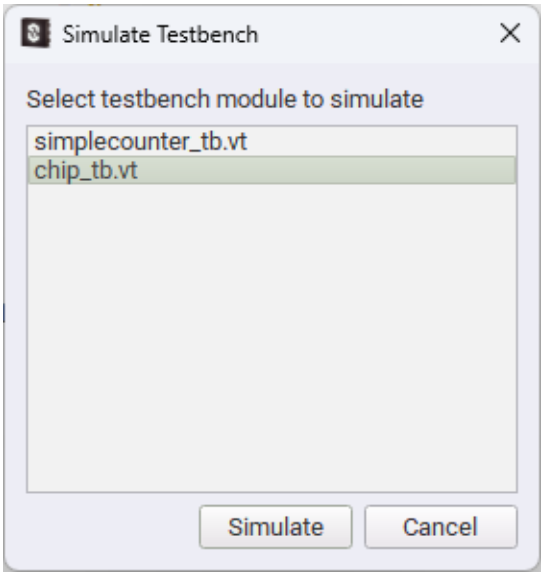

Figure 10. Simulate Testbench

7. Verify the simulation results – review the output and confirm that the project behaves as expected (see Figure 11).

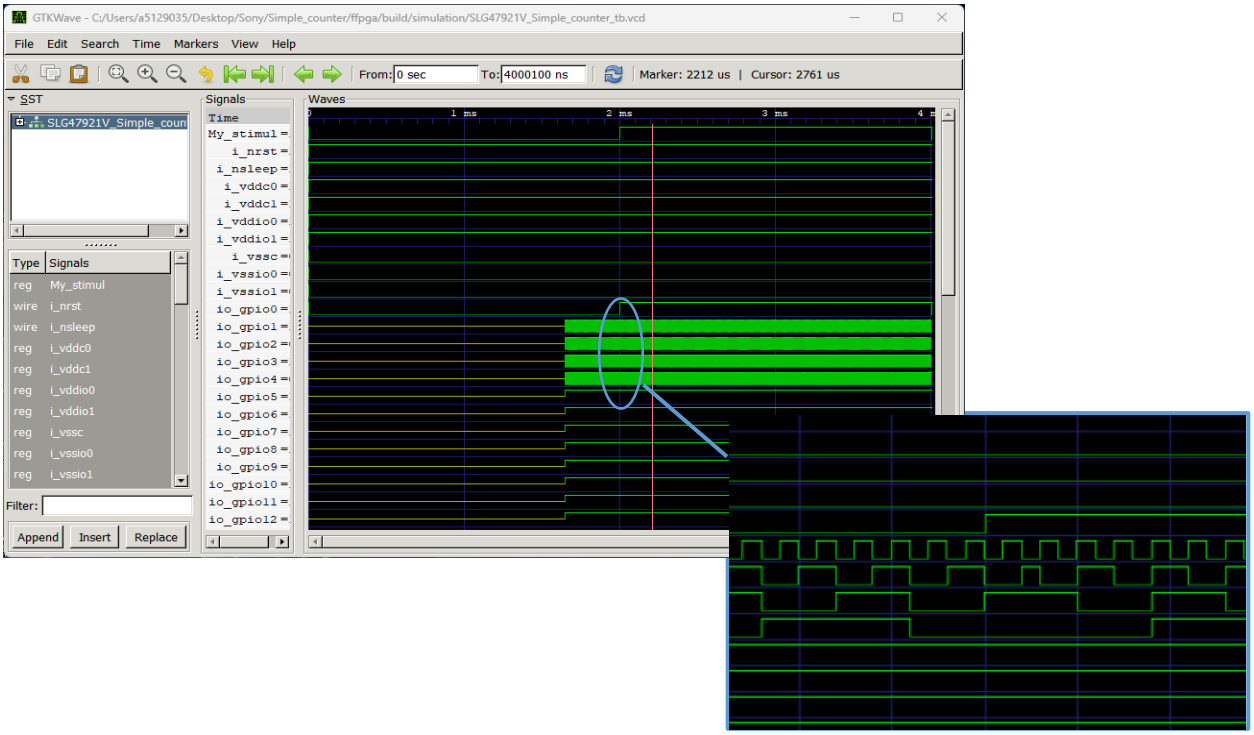

Figure 11. Full-Chip RTL Simulation Results

3. Pattern Detector (Multi-Chip Simulation) Example

3.1 Overview

As a code example to demonstrate Full-Chip RTL Simulation in a multi-chip solution, the previous example of a Simple Counter can be used in addition to a simple Pattern Detector project. This project receives a 4-bit input and provides two separate outputs: one that goes high when all input bits are set to a logic '1', and another that goes high when the input value equals 0100 (decimal 4).

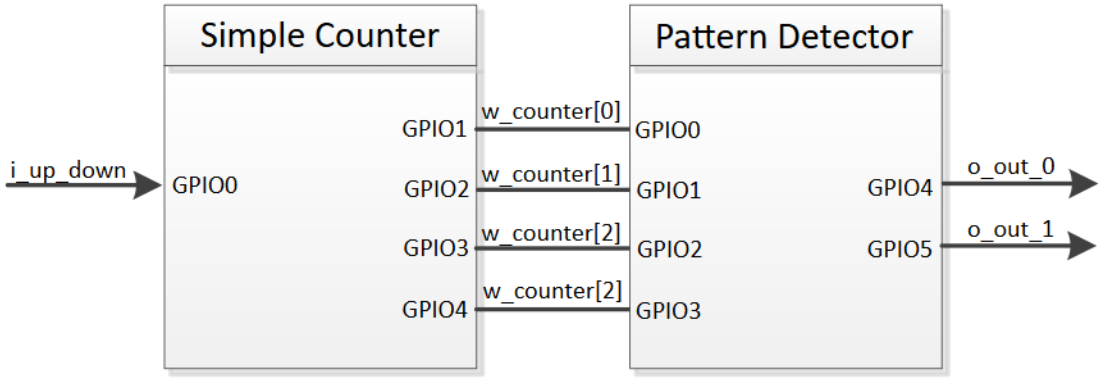

Figure 12. Multi-Chip Solution Example

To implement this Pattern Detector project, create a new project by adding Verilog code map signals to particular IOBs (as shown in Figure 13).

```
// Pattern Detector

// The (*top*) attribute is needed on the topmost module for synthesis
(* top *) module pattern_detector(
  input [3:0] i_data,
```

```
  output      o_out_0,
  output      o_out_1,
  output      o_out_0_oe,
  output      o_out_1_oe
);

 // The OE (output enable) is used to define whether the GPIO operates as an output
or an input
 // assign OE = 1 to function as an output
 // assign OE = 0 to function as an input. If not mentioned, then the default value
is 0

 assign o_out_0_oe = 1'b1;
 assign o_out_1_oe =1'b1;

 assign o_out_0 = (i_data == 4'b1111);
 assign o_out_1 = (i_data == 4'b0010);

endmodule
```

TILE	FUNCTION	DIRECTION	PORT
[0, 0]	GPIO0_IN [PIN 2]	Input	i_data[0]
[0, 0]	GPIO1_IN [PIN 3]	Input	i_data[1]
[0, 0]	GPIO2_IN [PIN 4]	Input	i_data[2]
[0, 0]	GPIO3_IN [PIN 5]	Input	i_data[3]
[0, 0]	GPIO4_OUT [PIN 6]	Output	o_out_0
[0, 0]	GPIO4_OE [PIN 6]	Output	o_out_0_oe
[0, 0]	GPIO5_OUT [PIN 7]	Output	o_out_1
[0, 0]	GPIO5_OE [PIN 7]	Output	o_out_1_oe

Figure 13. Pattern Detector I/O Planner Mapping

After these steps have been taken, the user can run this project in Full-Chip Simulation to simulate two chips simultaneously (the original Simple Counter example along with the pattern generator project in a multi-chip solution). Because the Pattern Detector project consists only of combinatorial logic, there is no need to initialize any registers or set any clock sources as it was done with the Simple Counter.

3.2 Design Steps

1. Synthesize the project and generate a bitstream – this step is required to build models of the entire circuit within the environment (Figure 5).
2. Generate simulation models – after synthesis and bitstream generation are finished, open the **Simulation** tab and select **Generate Full-Chip Models** (Figure 6).
3. Export Models – open the **Simulation** tab and select **Export Models** (Figure 14).

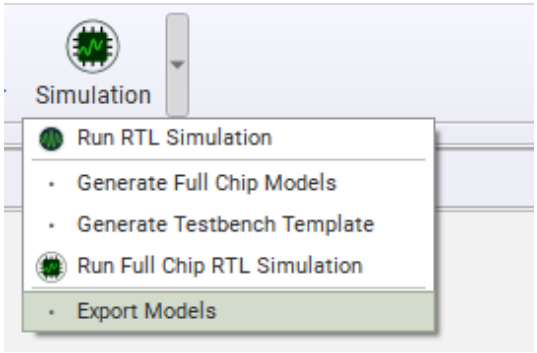

Figure 14. Export Models

4. Import models – to simulate two chips (multi-chip solution), previously generated model should be added. Select **File** → **Import** → **Full-Chip Simulation** and chose the model that was previously generated (Simple Counter). This model should now be available in our Sources.

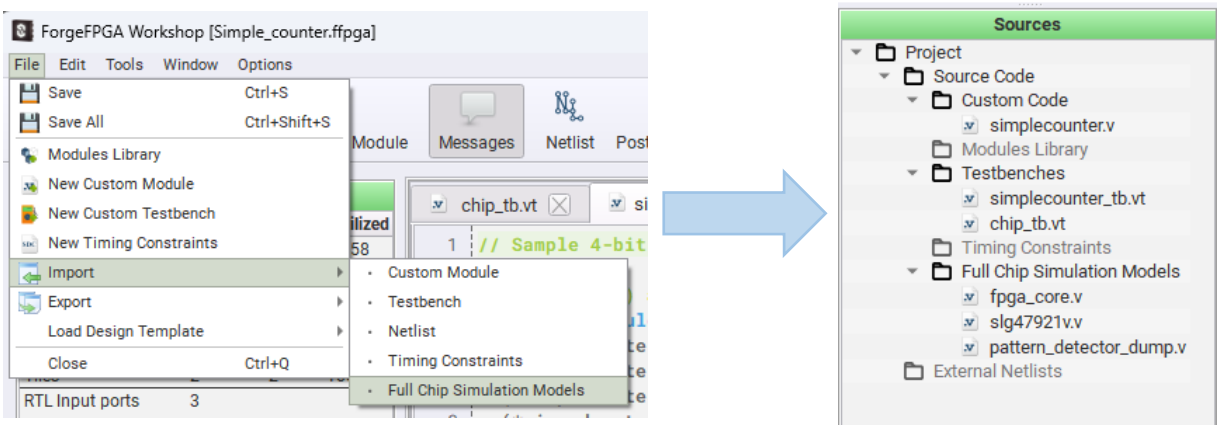

Figure 15. Import Models

5. Modify the testbench – create an additional set of wires, registers, stimuli, and others, for the second chip in the testbench.

```
reg i_vssc  ;
reg i_vddc0 ;
reg i_vddc1 ;
reg i_vddio1;
reg i_vddio0;
reg i_vssio0;
reg i_vssio1;
reg r_nsleep;
reg r_nrst  ;
wire i_nsleep;
wire i_nrst  ;
wire io_gpio0;
wire io_gpio1;
wire io_gpio2;
wire io_gpio3;
wire io_gpio4;
wire io_gpio5;
wire io_gpio6;
wire io_gpio7;
```

```
reg i_vssc,     i_ext_model_vssc  ;
reg i_vddc0,    i_ext_model_vddc0 ;
reg i_vddc1,    i_ext_model_vddc1 ;
reg i_vddio1,   i_ext_model_vddio1;
reg i_vddio0,   i_ext_model_vddio0;
reg i_vssio0,   i_ext_model_vssio0;
reg i_vssio1,   i_ext_model_vssio1;
reg r_nsleep,   r_ext_model_nsleep;
reg r_nrst,     r_ext_model_nrst  ;
wire i_nsleep,  i_ext_model_nsleep;
wire i_nrst,    i_ext_model_nrst  ;
wire io_gpio0,  io_ext_model_gpio0;
wire io_gpio1,  io_ext_model_gpio1;
wire io_gpio2,  io_ext_model_gpio2;
wire io_gpio3,  io_ext_model_gpio3;
wire io_gpio4,  io_ext_model_gpio4;
wire io_gpio5,  io_ext_model_gpio5;
wire io_gpio6,  io_ext_model_gpio6;
wire io_gpio7,  io_ext_model_gpio7;
```

Figure 16. Create Additional Set of Wires and Registers

6. Add an additional instance for export model – it is possible to copy and paste the existing instance. The name of instance should be same as the name of export model (marked purple in Figure 17).

```
SLG47921V_pattern_detector dut_export_model(
    .i_vssc     ( i_ext_model_vssc    ),
    .i_vddc1    ( i_ext_model_vddc1   ),
    .i_vddio0   ( i_ext_model_vddio0  ),
    .i_vddc0    ( i_ext_model_vddc0   ),
    .i_vddio1   ( i_ext_model_vddio1  ),
    .i_vssio1   ( i_ext_model_vssio1  ),
    .i_vssio0   ( i_ext_model_vssio0  ),
    .i_nsleep   ( i_ext_model_nsleep  ),
    .i_nrst     ( i_ext_model_nrst    ),
    .io_gpio0   ( i_ext_model_gpio1   ),
    .io_gpio1   ( io_ext_model_gpio2  ),
    .io_gpio2   ( io_ext_model_gpio3  ),
    .io_gpio3   ( io_ext_model_gpio4  ),
    .io_gpio4   ( io_ext_model_gpio4  ),
    .io_gpio5   ( io_ext_model_gpio5  ),
    .io_gpio6   ( io_ext_model_gpio6  ),
    .io_gpio7   ( io_ext_model_gpio7  ),
```

Figure 17. Instantiation of the Exported Model

7. Add corresponding connections between the two chips according to the requirements of the simulation. In this case, the Simple Counter outputs need to be connected to the input bus of the Pattern Detector.
8. Add additional stimuli if needed – insert your own stimuli into the testbench and connect them to the required GPIOs.
9. Select **Run Full Chip RTL Simulation** (see Figure 6) – execute the Full-Chip RTL Simulation by choosing the *chip_tb.vt* testbench to test the operation of the project (Figure 10).
10. Verify the simulation results – review the output and confirm that the project functions as expected (Figure 18).

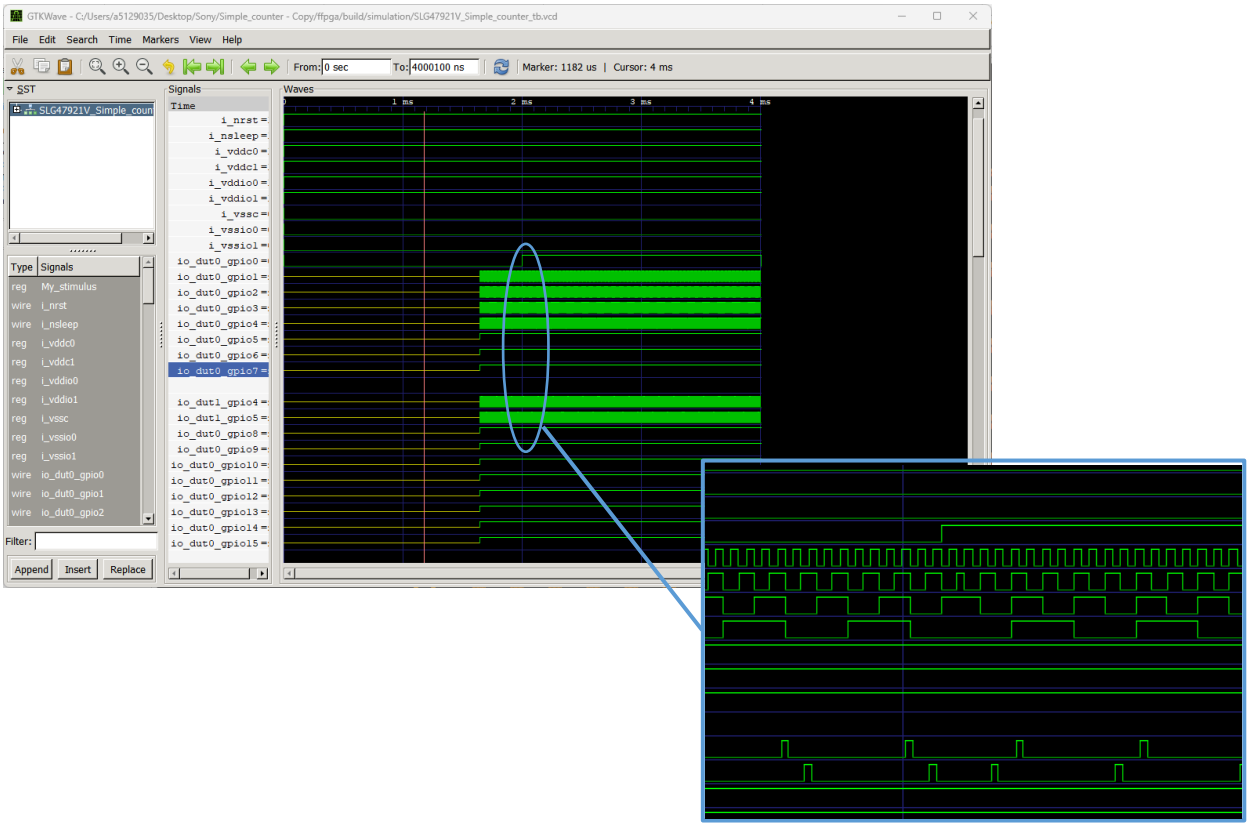

Figure 18. Full-Chip RTL Simulation Results (Multi-Chip Solution)

4. Conclusion

This application note shows how to use Full-Chip RTL Simulation for both a single-chip and a multi-chip solution. The files for this example are available in section 6 References. For more information, please contact the ForgeFPGA™ Business Support Team.

5. Terms and Definitions

HDL	Editor workspace where Verilog code is entered
FPGA	Field Programmable Gate Array
FPGA Editor	Main FPGA design and simulation window
Go Configure Software Hub	Main software window for device selection
ForgeFPGA	Main FPGA project window for debug and IO programming

6. References

For related documents and software, visit: ForgeFPGA Low-density FPGAs | Renesas.

Download our free ForgeFPGA™ Designer software [1] to open the .ffpga design files [2] and view the proposed circuit design.

[1] Go Configure Software Hub, Software Download and User Guide, Renesas Electronics

[2] AN-FG-026 How to Use Full-Chip RTL Simulation in ForgeFPGA design.ffpga, Design file, Renesas Electronics

[3] Product Selector: ForgeFPGA Low-density FPGAs, Renesas Electronics

[4] ForgeFPGA Software User Guide, Renesas Electronics

7. Revision History

Revision	Date	Description
1.00	March 5, 2026	Initial release