ForgeFPGA Workshop v6.51 Software Manual

This document will help you navigate the ForgeFPGA Workshop software v6.51 and understand its built-in features of the software in detail.

Contents

Figures

References

For related documents and software, please visit our website: https://www.renesas.com/us/en

Download our free ForgeFPGA Designer software [1] and follow the steps in this user guide. User can reference [2] [8] for the datasheet. Use Configuration Document to understand the different modes of configuration [3]. Renesas Electronics provides a complete library of application notes [4] featuring design examples as well as explanations of features and blocks within the Renesas IC. Please visit the product page to download the following:

[1] Go Configure Software Hub, Software Download, Renesas Electronics

[2] ForgeFPGA SLG47910 Datasheet, Renesas Electronics

[3] ForgeFPGA Configuration Document, Renesas Electronics

[4] Application Notes, ForgeFPGA Application Notes & Design Files, Renesas Electronics

[5] ForgeFPGA Deluxe Development Board User Manual, Renesas Electronics

[6] ForgeFPGA Socket Adapter User Manual, Renesas Electronics

[7] ForgeFPGA Evaluation Board User Manual, Renesas Electronics

[8] ForgeFPGA Software Simulation User Manual, Renesas Electronics

[9] ForgeFPGA SLG47920/21 Datasheet, Renesas Electronics

[10] GoConfigure Development Board User Manual, Renesas Electronics

[11] ForgeFPGA Socket Card User Manual, Renesas Electronics

1. Software Overview

The Go Configure Software Hub is a software product used to create a design for a specific device configuration. The software provides direct access to all GreenPAK, ADCPAK, and ForgeFPGA device features and complete control over each device's routing and configuration options.

The software contains the tools that makes it possible to:

- Create an FPGA Project in Verilog
- Program a chip with the created design
- Read a programmed part and import its data into the software
- Run simulations with external components

Getting Started:

To create and debug a new design with ForgeFPGA follow the next steps: Download and install Go Configure Software Hub from GoConfigure Software Hub

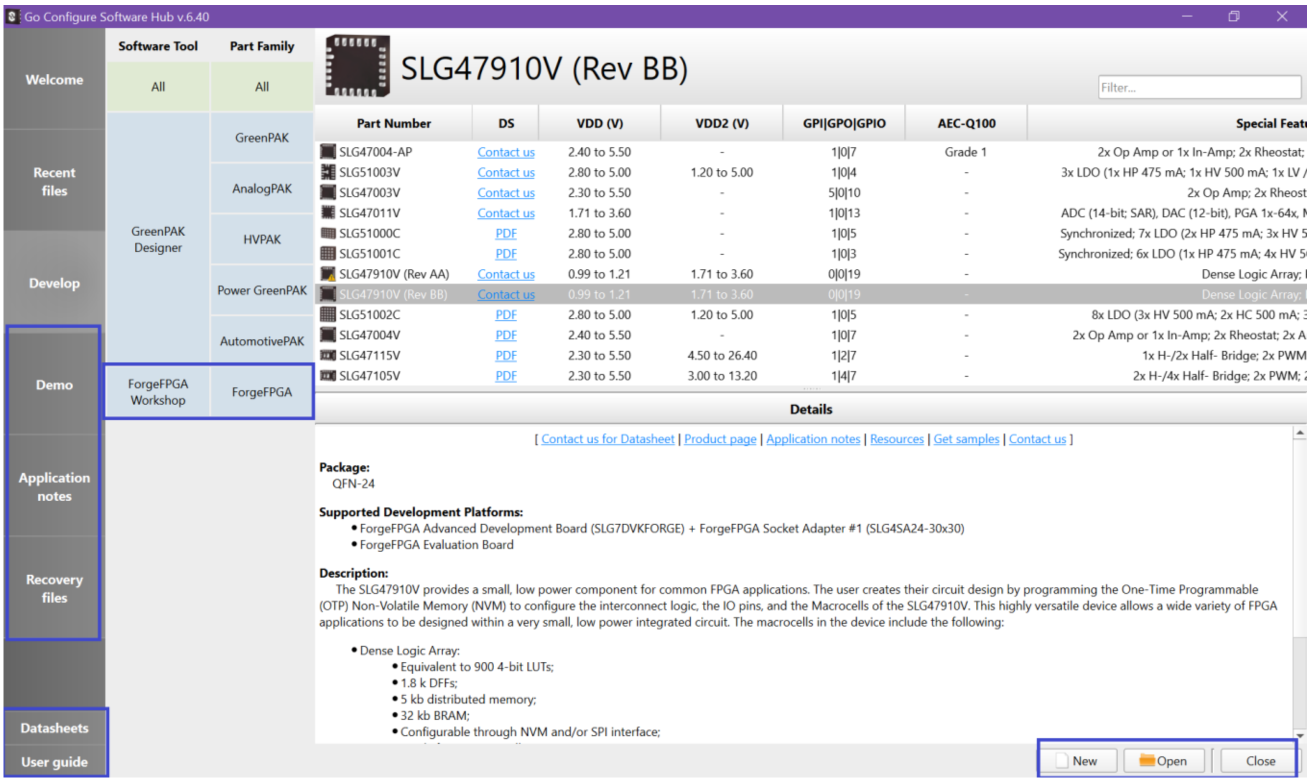
1. Start your project from the Hub window with the following sections:

- *Home* — useful info and tips for new users
- *Recent files* — the list of the recently opened project files
- *Develop* — the chip Part Number selection. See the Details section to learn more about the selected chip.

At the bottom-right of the window, you can find the New, Open, and Close buttons, which allow you to start a new project for a selected Part Number, to open an existing project, or to close the Go Configure Software Hub. The Datasheets, Product Page, Application Notes, Resources, Get Samples, Contact Us Link, and User Guides buttons redirect you to the Renesas website, where you can download the corresponding files.

- *Demo* — the list of Demo projects. You can use the specific Demo Board for project debugging
- *Application Notes* — design examples for different purposes. An application note includes a design description and a preconfigured circuit project, where you can make customized changes

2. Select the Product Family — *ForgeFPGA* — and then select the appropriate SLG479xx Part Number and open it
3. Specify the Project Settings such as V_{DDC}, V_{DDIO0}, V_{DDIO1} and Temperature (°C)
4. Write the Verilog Code in the Editor Window
5. Configure the blocks properties such as GPIO, PLL, etc. as per the design requirements
6. Test the design with the Debug Tool, using the Simulation feature or any of the supported hardware development platforms.



SLG47910V (Rev BB)

Part Number	DS	VDD (V)	VDD2 (V)	GPI|GPO|GPIO	AEC-Q100	Special Feat
SLG47004-AP	Contact us	2.40 to 5.50	-	1|0|7	Grade 1	2x Op Amp or 1x In-Amp; 2x Rheostat;
SLG51003V	Contact us	2.80 to 5.00	1.20 to 5.00	1|0|4	-	3x LDO (1x HP 475 mA; 1x HV 500 mA; 1x LV /
SLG47003V	Contact us	2.30 to 5.50	-	5|0|10	-	2x Op Amp; 2x Rheost
SLG47011V	Contact us	1.71 to 3.60	-	1|0|13	-	ADC (14-bit; SAR), DAC (12-bit), PGA 1x-64x,
SLG51000C	PDF	2.80 to 5.00	-	1|0|5	-	Synchronized; 7x LDO (2x HP 475 mA; 3x HV 5
SLG51001C	PDF	2.80 to 5.00	-	1|0|3	-	Synchronized; 6x LDO (1x HP 475 mA; 4x HV 5
SLG47910V (Rev AA)	Contact us	0.99 to 1.21	1.71 to 3.60	0|0|19	-	Dense Logic Array;
SLG47910V (Rev BB)	Contact us	0.99 to 1.21	1.71 to 3.60	0|0|19	-	Dense Logic Array;
SLG51002C	PDF	2.80 to 5.00	1.20 to 5.00	1|0|5	-	8x LDO (3x HV 500 mA; 2x HC 500 mA;
SLG47004V	PDF	2.40 to 5.50	-	1|0|7	-	2x Op Amp or 1x In-Amp; 2x Rheostat; 2x A
SLG47115V	PDF	2.30 to 5.50	4.50 to 26.40	1|2|7	-	1x H-/2x Half- Bridge; 2x PWM
SLG47105V	PDF	2.30 to 5.50	3.00 to 13.20	1|4|7	-	2x H-/4x Half- Bridge; 2x PWM;

Details

[Contact us for Datasheet | Product page | Application notes | Resources | Get samples | Contact us]

Package:
QFN-24

Supported Development Platforms:
- ForgeFPGA Advanced Development Board (SLG7DVKFORGE) + ForgeFPGA Socket Adapter #1 (SLG4SA24-30x30)
- ForgeFPGA Evaluation Board

Description:
The SLG47910V provides a small, low power component for common FPGA applications. The user creates their circuit design by programming the One-Time Programmable (OTP) Non-Volatile Memory (NVM) to configure the interconnect logic, the IO pins, and the Macrocells of the SLG47910V. This highly versatile device allows a wide variety of FPGA applications to be designed within a very small, low power integrated circuit. The macrocells in the device include the following:

- Dense Logic Array:
  - Equivalent to 900 4-bit LUTs;
  - 1.8 k DFFs;
  - 5 kb distributed memory;
  - 32 kb BRAM;
  - Configurable through NVM and/or SPI interface;

Figure 1. GoConfigure Software Hub User Interface

Support:

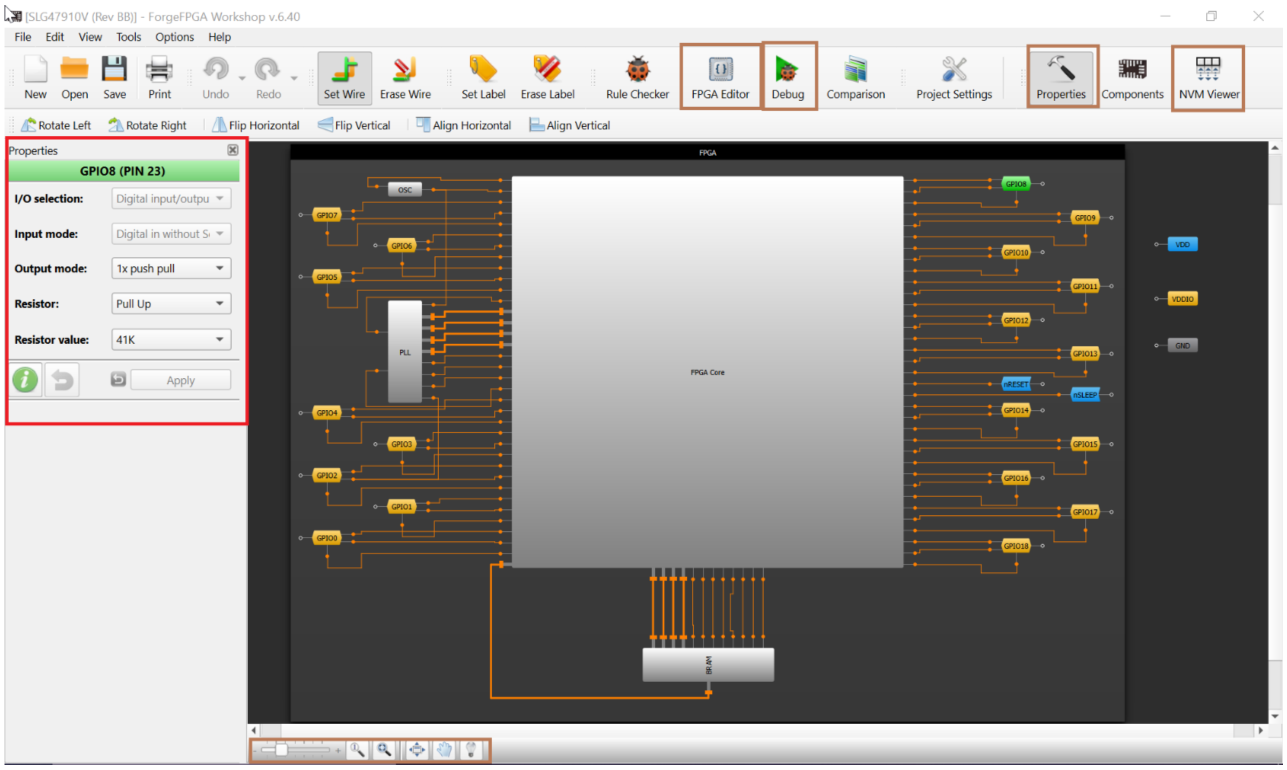
For more information about the Go Configure Software Hub, online support is available at www.renesas.com.

You can also find the Go Configure Software Hub on Renesas social media. To visit the page, go to Help → Social in the main menu.

The latest version of the software application is available on the Renesas website, on the Software page. Go Configure Software Hub will also have notifications about pending software updates.

2. Forge FPGA Workshop

The ForgeFPGA Workshop is the main window of the software. This window displays the FPGA Core and each of the special components that connect to it such as the Phase Locked Loop (PLL), Oscillator (OSC), BRAM, and the GPIOs. The FPGA Editor can be launched from the middle button on the toolbar at the top in addition to navigating to Main Menu Tools → *FPGA Editor* or by double-clicking the on the FPGA Core (grey square).



Properties

GPIO8 (PIN 23)

I/O selection:	Digital input/outpu
Input mode:	Digital in without S
Output mode:	1x push pull
Resistor:	Pull Up
Resistor value:	41K

Figure 2. ForgeFPGA Workshop

This window is designed to cater to the device selected. The above figure is for SLG47910. A different setup of blocks will be seen for SLG47921, etc. based on its pin assignment.

2.1 FPGA Editor

The FPGA Editor is an Integrated Design Environment (IDE) tool designed to create and configure the FPGA logic. The editor allows you to create designs for Field-Programmable Gate Arrays (FPGAs) using a Hardware Description Language (HDL)* - Verilog. Before the design is programmed onto the FPGA, the editor provides an option to simulate it first and ensure it operates correctly.

In addition, the FPGA Editor makes it possible to work with external designs by importing files with the created logic mapped to the components. Find the descriptions for these and other features in the following sections.

Flowchart

A flowchart is available to provide a step-by-step guide on creating a design from start to finish. The flowchart (see Figure 3) describes all the important aspects of using the ForgeFPGA software.

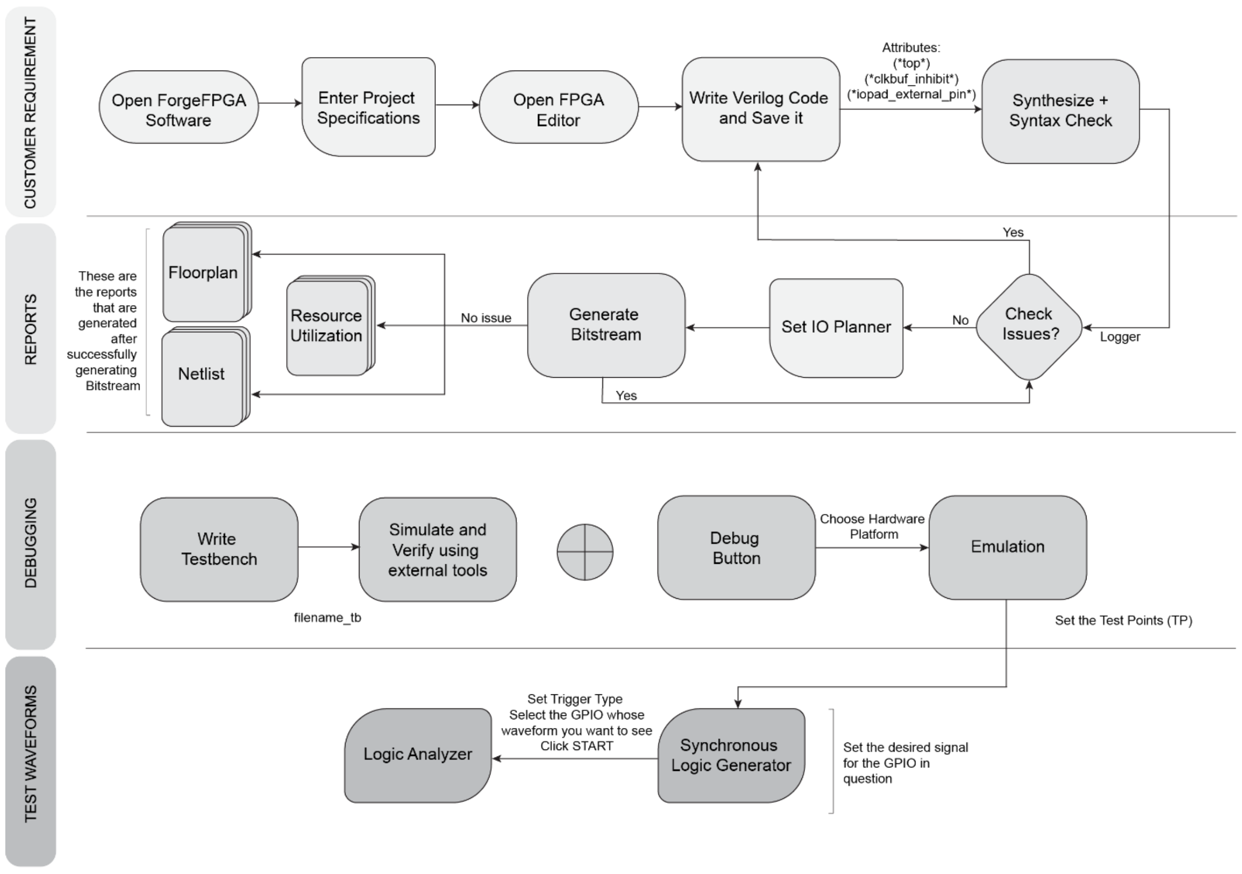
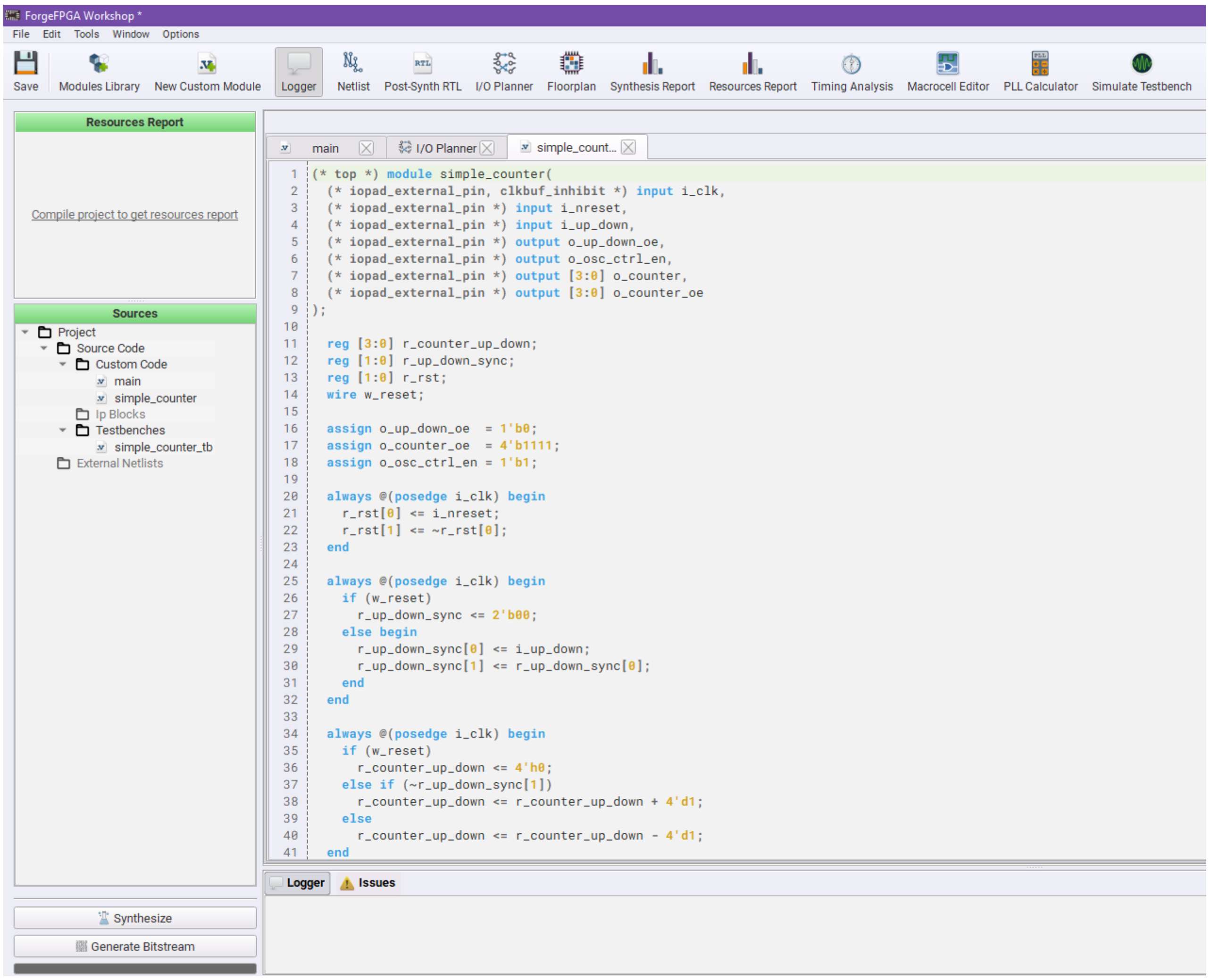

CUSTOMER REQUIREMENT
REPORTS
DEBUGGING
TEST WAVEFORMS
Open ForgeFPGA Software
Enter Project Specifications
Open FPGA Editor
Write Verilog Code and Save it
Attributes: (*top*) (*clkbuf_inhibit*) (*iopad_external_pin*)
Synthesize + Syntax Check
Logger
Check Issues?
Yes
No
Set IO Planner
Generate Bitstream
Yes
No issue
Resource Utilization
Floorplan
Netlist
These are the reports that are generated after successfully generating Bitstream
Write Testbench
filename_tb
Simulate and Verify using external tools
Debug Button
Choose Hardware Platform
Emulation
Set the Test Points (TP)
Synchronous Logic Generator
Set the desired signal for the GPIO in question
Set Trigger Type Select the GPIO whose waveform you want to see Click START
Logic Analyzer

Figure 3. Toolchain Flowchart

The development software consists of the main menu, toolbar, main work area, logger panel and control panel (see Figure 4).



```
(* top *) module simple_counter(
  (* iopad_external_pin, clkbuf_inhibit *) input i_clk,
  (* iopad_external_pin *) input i_nreset,
  (* iopad_external_pin *) input i_up_down,
  (* iopad_external_pin *) output o_up_down_oe,
  (* iopad_external_pin *) output o_osc_ctrl_en,
  (* iopad_external_pin *) output [3:0] o_counter,
  (* iopad_external_pin *) output [3:0] o_counter_oe
);

  reg [3:0] r_counter_up_down;
  reg [1:0] r_up_down_sync;
  reg [1:0] r_rst;
  wire w_reset;

  assign o_up_down_oe  = 1'b0;
  assign o_counter_oe  = 4'b1111;
  assign o_osc_ctrl_en = 1'b1;

  always @(posedge i_clk) begin
    r_rst[0] <= i_nreset;
    r_rst[1] <= ~r_rst[0];
  end

  always @(posedge i_clk) begin
    if (w_reset)
      r_up_down_sync <= 2'b00;
    else begin
      r_up_down_sync[0] <= i_up_down;
      r_up_down_sync[1] <= r_up_down_sync[0];
    end
  end

  always @(posedge i_clk) begin
    if (w_reset)
      r_counter_up_down <= 4'h0;
    else if (~r_up_down_sync[1])
      r_counter_up_down <= r_counter_up_down + 4'd1;
    else
      r_counter_up_down <= r_counter_up_down - 4'd1;
  end
```

Figure 4. ForgeFPGA Workshop User Interface

Main Menu

Main Menu contains controls are described below:

File

- Save – save current project
- Save All – save the whole project
- Modules Library – open IP Blocks Wizard
- New Custom Module – add new module
- New Custom Testbench – add new testbench
- New Timing Constraints – add .sdc file
- Import – used to import an RTL code, testbench and Netlist from another software/synthesis tool
    - Custom Module
    - Custom Testbench
    - Netlist
    - Timing Constraints
    - Full Chip Simulation Models
    - Placement Constraints
- Export
    - Custom Module

    - Library Module
    - Testbench
    - Timing Constraints
    - Full Chip Simulation Models
    - Placement Constraints
- Load Design Template
    - Simple Counter
- Close

Edit (menu works with editable tabs)

- Undo
- Redo
- Cut
- Copy
- Paste
- Delete
- Select All
- Find

Tools

- Run Synthesis – synthesize Verilog code into low-level constructs.
- Generate Bitstream – compilation and mapping low-level constructs to specific device blocks.
- Floorplan
    - Load customer PnR
- Simulation
    - Run RTL Simulation
    - Run Full chip RTL simulation
    - Generate Full Chip Models
    - Generate Testbench Template
    - Export models
- I/O Planner
    - Clear data
    - Import I/O Spec
    - Export I/O Spec
    - Import I/O Spec from CSV
    - Export I/O Spec to CSV
- PLL Configurator

Window

- Netlist – read-only tab with a description of the connectivity of an electronic circuit
- Post-Synth RTL (register transfer layer)
- I/O Planner
- Floorplan
- Synthesis Report
- Resources Report
- Timing Analysis

- Macrocell Editor
- Messages

Options

- Settings

Toolbar

The top toolbar provides quick access to a set of tools and actions. See the description of all the available tools in the next few sections.

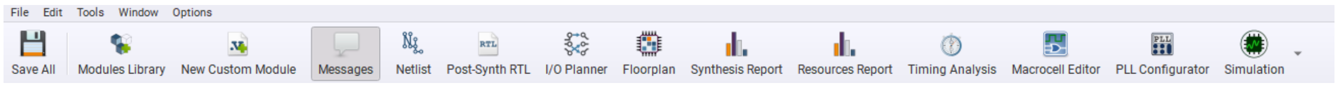
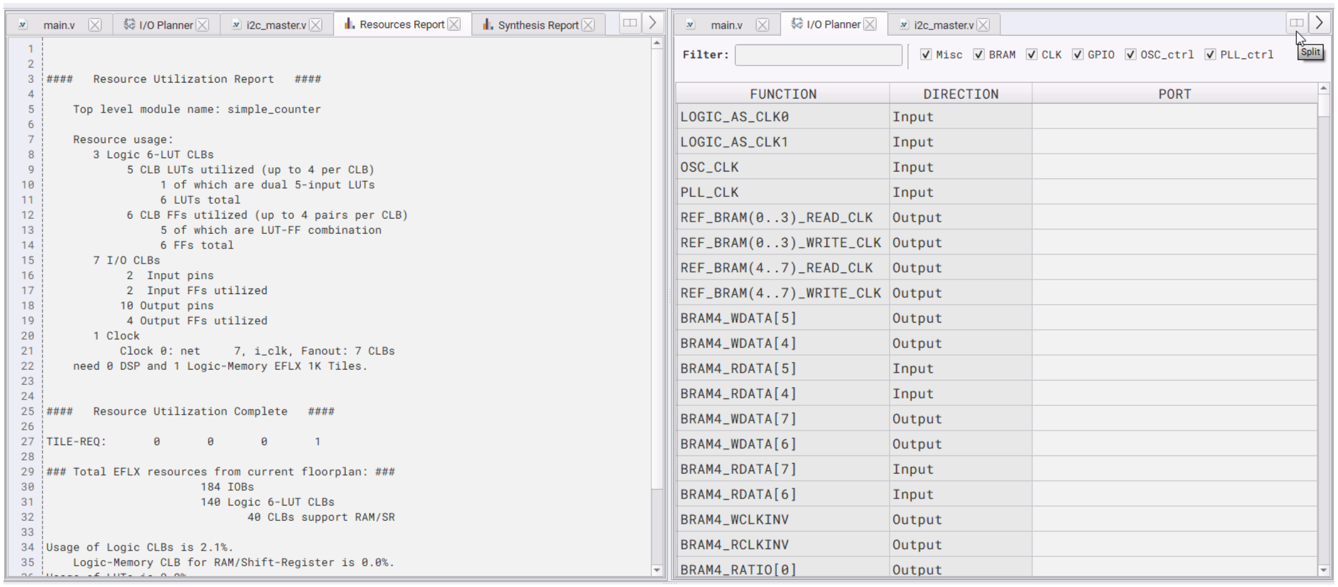


Figure 5. ForgeFPGA Workshop Toolbar

Work Area

The work area is the central white portion of the software where the user can type their desired HDL Code. The work area has a *split* screen option (see Figure 6) which allows the user to split the screen and work with more than one tool simultaneously. The user can choose which views appear in either of side of the split screen by clicking the tool. The user can also close the split screen when not needed.



```


####   Resource Utilization Report   ####

    Top level module name: simple_counter

    Resource usage:
       3 Logic 6-LUT CLBs
            5 CLB LUTs utilized (up to 4 per CLB)
                 1 of which are dual 5-input LUTs
                 6 LUTs total
            6 CLB FFs utilized (up to 4 pairs per CLB)
                 5 of which are LUT-FF combination
                 6 FFs total
       7 I/O CLBs
            2  Input pins
            2  Input FFs utilized
           10 Output pins
            4 Output FFs utilized
       1 Clock
           Clock 0: net     7, i_clk, Fanout: 7 CLBs
    need 0 DSP and 1 Logic-Memory EFLX 1K Tiles.


####   Resource Utilization Complete   ####

TILE-REQ:       0       0       0       1

### Total EFLX resources from current floorplan: ###
                       184 IOBs
                       140 Logic 6-LUT CLBs
                              40 CLBs support RAM/SR

Usage of Logic CLBs is 2.1%.
    Logic-Memory CLB for RAM/Shift-Register is 0.0%.
```

Filter: ✓ Misc ✓ BRAM ✓ CLK ✓ GPIO ✓ OSC_ctrl ✓ PLL_ctrl

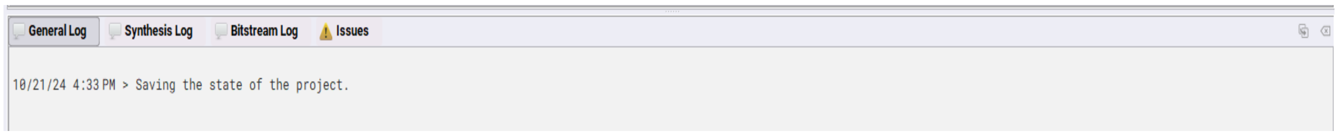
FUNCTION	DIRECTION	PORT
LOGIC_AS_CLK0	Input	
LOGIC_AS_CLK1	Input	
OSC_CLK	Input	
PLL_CLK	Input	
REF_BRAM(0..3)_READ_CLK	Output	
REF_BRAM(0..3)_WRITE_CLK	Output	
REF_BRAM(4..7)_READ_CLK	Output	
REF_BRAM(4..7)_WRITE_CLK	Output	
BRAM4_WDATA[5]	Output	
BRAM4_WDATA[4]	Output	
BRAM4_RDATA[5]	Input	
BRAM4_RDATA[4]	Input	
BRAM4_WDATA[7]	Output	
BRAM4_WDATA[6]	Output	
BRAM4_RDATA[7]	Input	
BRAM4_RDATA[6]	Input	
BRAM4_WCLKINV	Output	
BRAM4_RCLKINV	Output	
BRAM4_RATIO[0]	Output	

Figure 6. Work Area

Messages Panel

Here you can see the generated messages that appear after you use Synthesize, Generate Bitstream and Simulation procedure (see Figure 7).

- *General Log* – shows information messages along with warnings and errors that were recorded while processing the design.
- *Synthesis Log and Bitstream Log* – provides the output generated while the procedure is performed
- *Issues* – displays the warning and error messages that are automatically generated by the software when required. Once you receive a syntax error, you can right-click on it and select ‘Show in Editor’ to highlight the line in your HDL code where the mistake was detected.

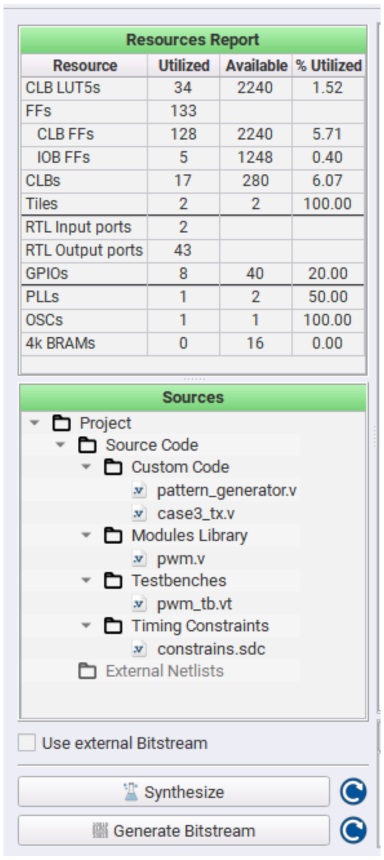


```
10/21/24 4:33 PM > Saving the state of the project.
```

Figure 7. Messages Panel

Control Panel

From the left control panel, you can access the following information (see Figure 8):



Resources Report

Resource	Utilized	Available	% Utilized
CLB LUT5s	34	2240	1.52
FFs	133		
CLB FFs	128	2240	5.71
IOB FFs	5	1248	0.40
CLBs	17	280	6.07
Tiles	2	2	100.00
RTL Input ports	2		
RTL Output ports	43		
GPIOs	8	40	20.00
PLLs	1	2	50.00
OSCs	1	1	100.00
4k BRAMs	0	16	0.00

Sources

- Project
  - Source Code
    - Custom Code
      - pattern_generator.v
      - case3_tx.v
    - Modules Library
      - pwm.v
    - Testbenches
      - pwm_tb.vt
    - Timing Constraints
      - constrains.sdc
  - External Netlists

☐ Use external Bitstream

Figure 8. Control Panel

- *Resources Report* – Extracted from the resource usage report. It shows the part of available resources utilized by the design sources. Visually, it is divided into three sections. The upper part reflects the internal logic used in the project against the total available. The middle section shows connection-related information (RTL ports, GPIOs). The last part summarizes the utilization of internal components, such as OSCs, PLLs, and BRAMs.
- *Sources* – a list of all Verilog design files and external netlists (if used) that are in the current project file. Add *Sources* from the toolbar by clicking *New Custom Modules* or select one from the Modules Library. All supported sources can also be added via M*ain Menu → File.* Additionally, external sources can be imported by going to *File → Import*. Here you can find the following subcategories:
  - Custom Code
  - Modules Library
  - Testbenches
  - Full Chip Simulation models
  - Timing Constraints
  - Placement Constraints
  - External Netlists

  The *Sources* list can be customized via the context menu with options to Add, Delete, or Rename the sources.
- *Synthesize and Generate Bitstream* – main controls to run toolchain on your design to produce chip configuration. Hover over the icon next to the procedure button to see the status details, such as whether the procedure was successful, failed or if the Verilog file has been changed.

Settings

The user can access the settings from the *Main Menu → Options → Settings*. The user can change various settings for the project and change some user settings in this GUI (see Figure 9).

Project Settings

a. *Synthesize* – a process where the HDL code is compiled and converts a high-level description down to lower-level description, for example logic gates.

   - *Flatten* – Flatten design before synthesis. This option instructs the tool to fully flatten the hierarchy leaving only the top level. Cells and/or modules with the 'keep_hierarchy' attributes set will not be flattened by this command.
   - *No DSP* – Do not use DSPs to implement multipliers and associated logic during the synthesis procedure.
   - *Keep* – Create extra KEEP nets by allowing a cell to drive multiple nets while generating the EDIF netlist.
   - *Use ABC9* – Use ABC9 for technology mapping. The ABC9 pass uses much newer optimizers and mapping, for a modest improvement in optimization, and a possible big improvement in LUT mapping.
   - *Enable hard multiplexer resources* – enables the hard multiplexer interface.
   - *Minimum number of multiplexer inputs* – specifies the minimum number of inputs needed to activate hard resources
   - *Additional arguments* – Add additional arguments to the synthesis procedure.

b. *Generate Bitstream* – this is the final step before programming the FPGA. Under this process, Yosys performs processes such as Translation & Mapping, Place & Routing and Resource Utilization calculations. A binary file is generated that contains configuration information.

   - *High Density IO Packing* – Allows packing of input and output IOs into a single IO cell. Improves I/O utilization but can worsen congestion and timing.
   - *High Density Packing Logic* – Allows you to achieve higher resource utilization per CLB but can worsen congestion and timing.
   - *Clock-Concurrent Optimization (CCopt)* – Enables Clock-Concurrent Optimization to further optimize timing (must disable Ideal Clock) at the expense of potentially higher clocking power.
   - *Place-and-Trial-Route* – Iteratively run place-and-trail-route refinement to attempt timing improvements. Iteration count is defined by the "Place-and-trail-route iteration count' option.
   - *Place-and-Trial-route Iteration count* – Used with "Place-and-Trial-Route", sets the number of iterations for place-and-trial-route to attempt timing improvements.
   - *Maximum Routing Iterations* – Maximum number of routing iterations before exiting with the last clean solution with the best achievable timing.
   - *Maximum number of CPUs for routing* – Defines the maximum number of CPUs used for routing. Settings this option to 1 ensures identical bitstreams when using the same input data
   - *Timing Analysis Corner* – select the corner lot that will be used during Static Timing Analysis
   - *Additional arguments*- Add additional arguments to the Generate Bitstream procedure.

c. *Simulation* – User can choose to save their dump filedump file produced during simulation in two formats - .vcd and. fst format. The difference between both formats is how much memory the file can handle. Users can opt for .fst format when simulating a memory heavy file as it produces a compact binary format file that offers much better performance for very large dump files. It's fast to write and fast to read. VCD is a test format that is easy to read, and which is supported by a lot of different software packages.

User Settings

Messages – User can choose to opt for erasing the Synthesis and Bitstream log each time they run a new procedure by checking the box.

Tools:

- *Icarus Verilog* –The path to the Icarus Verilog tool. You need to set the path to the simulation engine binary so it can be found during the simulation procedure if the system environment doesn't have the proper path set. An "Autodetect" option is available, and it will try to automatically find the path to the tool.

- *GTKWave* – The path to the GTKWave tool. You need to set the path to the simulation results viewer binary so it can be found after the simulation procedure if the system environment doesn't have the proper path set. An "Autodetect" option is available, and it will try to automatically find the path to the tool.

Text Editor – Number of spaces in tab characters that are shown for all text editors.

I/O Planner – serves to manage the display of information in the I/O Planner table

Processing – Checking the box under this setting allows you to save all files before proceeding with the design

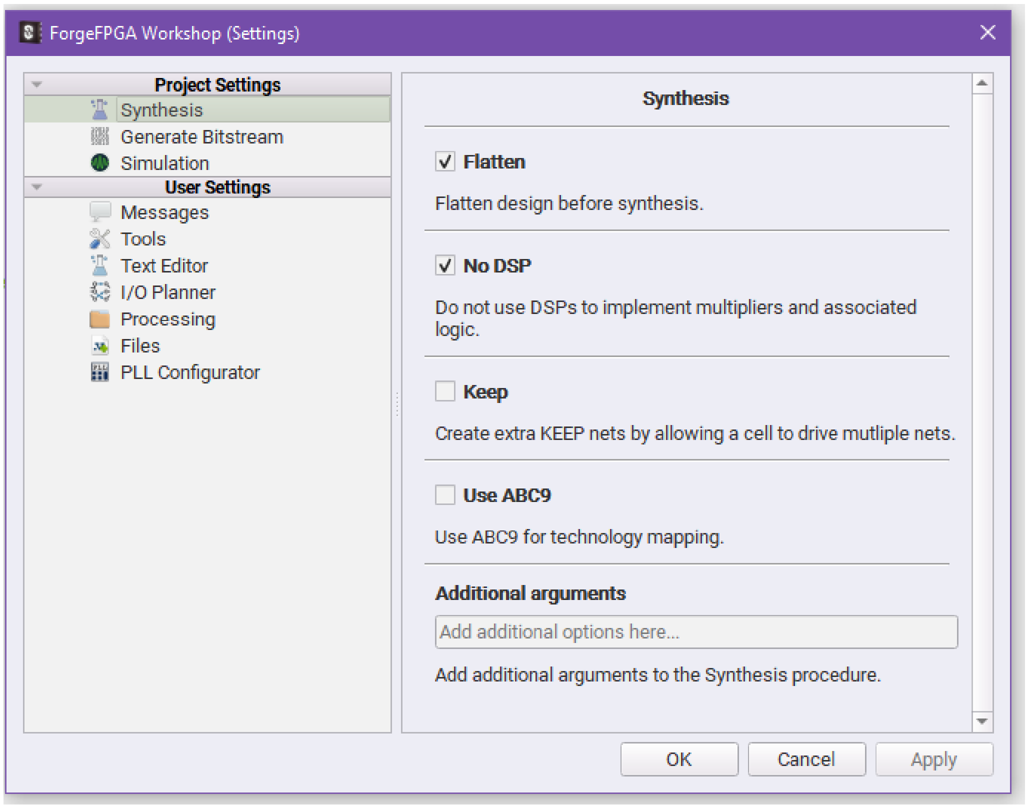
Files – Enables to software to add _tb suffix to the names of newly created testbench if unspecified

PLL Configurator – provides the ability to manage the behavior of the confirmation window before applying changes to the PLL Configurator'

Simulation – enables/disables the confirmation window when generating modules

Note: Changes made in the Project Settings category only applies to the project currently being worked on. Adjustments made in the User Settings section have a global impact, ensuring consistency across all FPGA-related projects on a particular device. If you switch to another computer, these settings will either be reset to default values or adapted to the configurations of that specific computer.

Follow the instructions provided in the options' descriptions to avoid errors and improve your design.



Synthesis

[✓] **Flatten**

Flatten design before synthesis.

[✓] **No DSP**

Do not use DSPs to implement multipliers and associated logic.

[ ] **Keep**

Create extra KEEP nets by allowing a cell to drive mutliple nets.

[ ] **Use ABC9**

Use ABC9 for technology mapping.

Additional arguments

Add additional options here...

Add additional arguments to the Synthesis procedure.

Figure 9. ForgeFPGA Workshop Settings

Design Template

The Design Template offers pre-built designs that can be seamlessly integrated into your project. It can be found under *FPGA Editor→ File → Load Design Template* (see Figure 10).

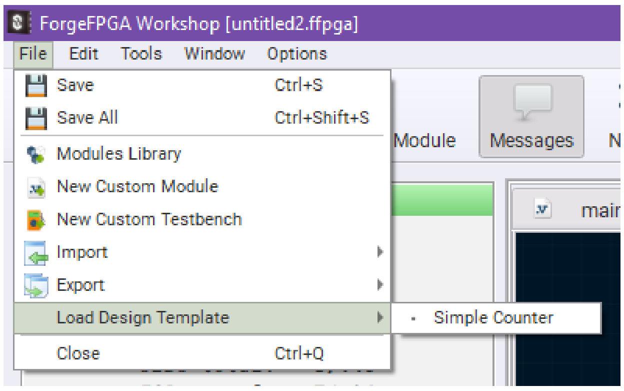
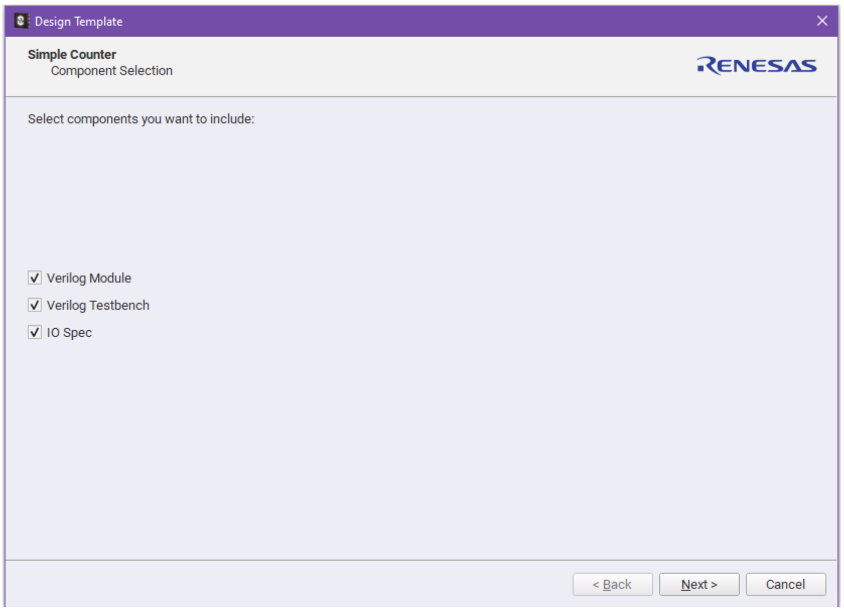


Figure 10. FPGA Design Template

Upon selecting a design template, the Design Template Wizard will appear (see Figure 11). It will guide you through component selection, IO Spec diff review, and module name suggestions.

Component Selection makes it possible to choose the components to proceed with:

- *Verilog Module*
- *Verilog Testbench*
- *IO Spec*



Design Template

Simple Counter
Component Selection

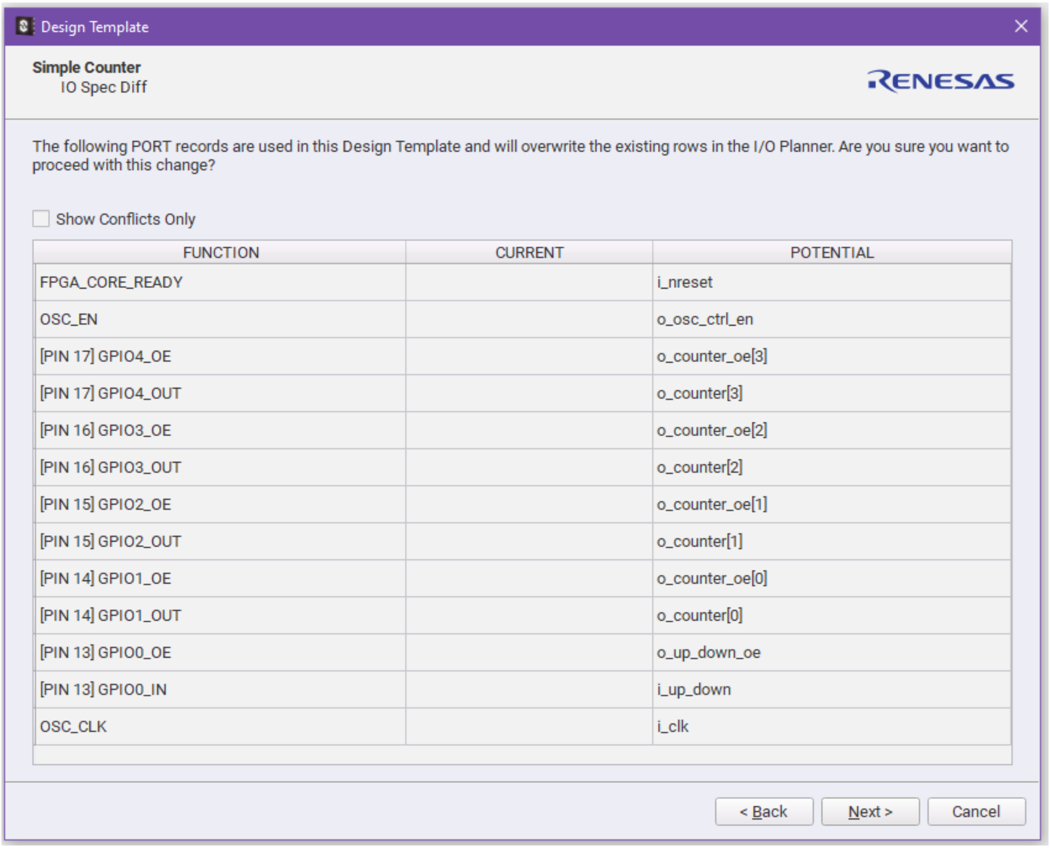
Select components you want to include:

- [x] Verilog Module
- [x] Verilog Testbench
- [x] IO Spec

Figure 11. Design Template Component Selection

Select all components or individually select them as needed depending on the desired outcome for your project. The choices made at this stage determine the next steps in the process.

- *IO Spec* -generates an *IO Spec Diff*, presenting a comparison between the existing design port records and the ones associated with the selected template. This serves as a cautionary step as your current port configurations could potentially be overwritten by those of the chosen template. The software highlights conflicting entries in yellow. You can choose to focus solely on conflicts by selecting the *Show Conflicts Only* option. However, if you prefer not to review the IO Specs, the software streamlines the process by skipping the IO Spec Diff (see Figure 12). In this scenario, the tool will prompt you straight to the Module name.



Design Template

Simple Counter
IO Spec Diff

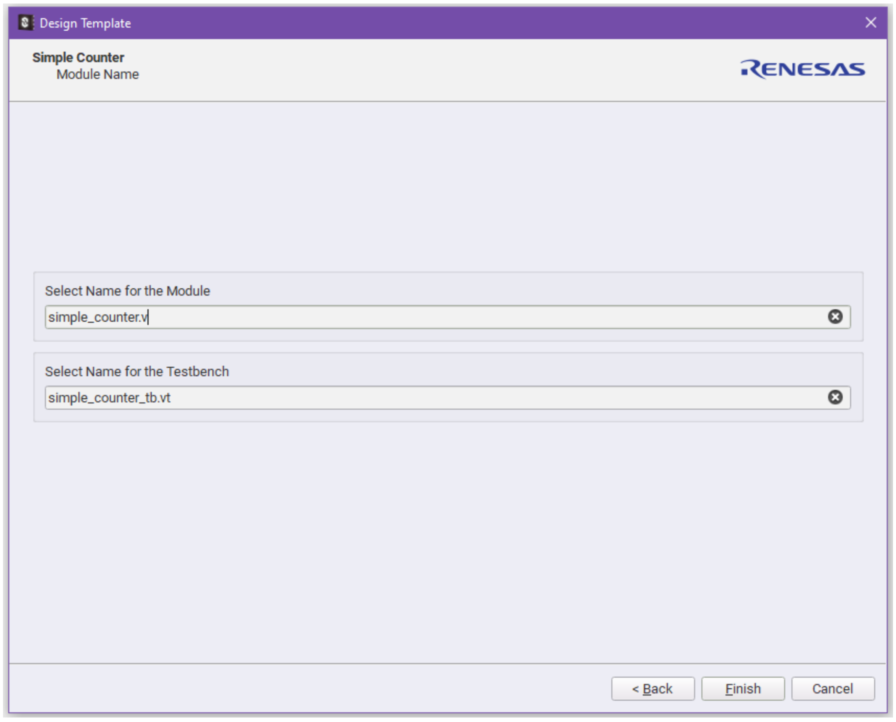
The following PORT records are used in this Design Template and will overwrite the existing rows in the I/O Planner. Are you sure you want to proceed with this change?

- [ ] Show Conflicts Only

FUNCTION	CURRENT	POTENTIAL
FPGA_CORE_READY		i_nreset
OSC_EN		o_osc_ctrl_en
[PIN 17] GPIO4_OE		o_counter_oe[3]
[PIN 17] GPIO4_OUT		o_counter[3]
[PIN 16] GPIO3_OE		o_counter_oe[2]
[PIN 16] GPIO3_OUT		o_counter[2]
[PIN 15] GPIO2_OE		o_counter_oe[1]
[PIN 15] GPIO2_OUT		o_counter[1]
[PIN 14] GPIO1_OE		o_counter_oe[0]
[PIN 14] GPIO1_OUT		o_counter[0]
[PIN 13] GPIO0_OE		o_up_down_oe
[PIN 13] GPIO0_IN		i_up_down
OSC_CLK		i_clk

Figure 12. Design Template IO Spec Diff

Module Name - name your Verilog Module and/or Testbench. If you choose to skip Verilog Module and Verilog Testbench in the Component Selection window, the default names are assigned.



Select Name for the Module
simple_counter.v

Select Name for the Testbench
simple_counter_tb.vt

Figure 13. Design Template Module Name

Once the template setup is complete, new elements will be made available in the workspace.

The Design Template Verilog code and Testbench names will appear in the Sources list, with corresponding tabs displaying the actual code and Testbench details.

Additionally, the I/O Planner tab will reflect these changes as well, showcasing the inclusion of port records from the selected design.

Once the integration of the design template is done, you can assess its functionality by running a simulation.

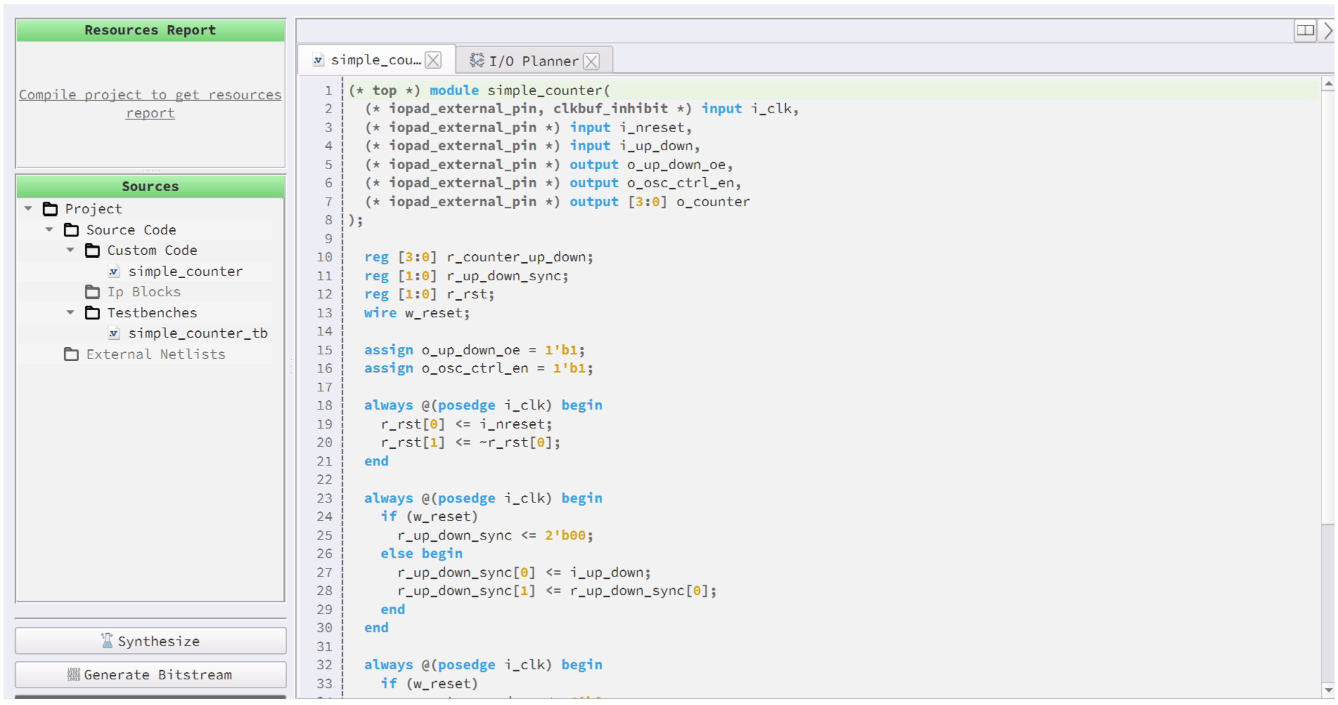
2.2 Writing Verilog Code

The toolchain of the ForgeFPGA Workshop supports Verilog 2005 (IEEE Standard 1364-2005) and System Verilog Syntaxes.
There are a few special code attributes that are important to keep in mind while working with the synthesis tools:

- **(* top *)** – the top module of your design should be marked with this attribute so the toolchain can successfully recognize which of the modules in the design is the top one. [Line 1 – Figure 14]

- **(* clkbuf_inhibit *)** – clock signals in the input list of the main module should be marked with this attribute to prevent clock buffer insertion by the synthesis tool, which may lead to the distortion of the clock signal name in the resulting netlist. This attribute belongs only to Synthesis; the compile process requires the customer to assign IOBs manually on the I/O Planner that are predefined clock trees. [Line 2 – Figure 14]

- **(*iopad_external_pin*)** – all external pins that are used in any module need to be marked with this attribute. [Line 2-7 – Figure 14]



```
(* top *) module simple_counter(
  (* iopad_external_pin, clkbuf_inhibit *) input i_clk,
  (* iopad_external_pin *) input i_nreset,
  (* iopad_external_pin *) input i_up_down,
  (* iopad_external_pin *) output o_up_down_oe,
  (* iopad_external_pin *) output o_osc_ctrl_en,
  (* iopad_external_pin *) output [3:0] o_counter
);

  reg [3:0] r_counter_up_down;
  reg [1:0] r_up_down_sync;
  reg [1:0] r_rst;
  wire w_reset;

  assign o_up_down_oe = 1'b1;
  assign o_osc_ctrl_en = 1'b1;

  always @(posedge i_clk) begin
    r_rst[0] <= i_nreset;
    r_rst[1] <= ~r_rst[0];
  end

  always @(posedge i_clk) begin
    if (w_reset)
      r_up_down_sync <= 2'b00;
    else begin
      r_up_down_sync[0] <= i_up_down;
      r_up_down_sync[1] <= r_up_down_sync[0];
    end
  end

  always @(posedge i_clk) begin
    if (w_reset)
```

Figure 14. Working with Verilog Example

The *ForgeFPGA* Workshop includes integrated linting support to detect syntax errors, coding issues, and design rule violations. The tool is compatible with Verilator v5.034

Importing/Export RTL Files

The ForgeFPGA Editor allows you to import external RTL files and netlist that has been created by other software. You can import a Custom Module file, Custom Testbench file or a Netlist. You can import an RTL File by navigating to *Main Menu →Import →Custom Module* (see Figure 15).

The software supports the following formats for importing:

- Custom Module (supported file formats *.v, *.vh, *.verilog, *.vlg, *.vt, *.sv)
- Custom Testbench (supported file formats *.v, *.vh, *.verilog, *.vlg, *.vt, *.sv)
- Netlist (supported file formats *.edif, *.edn)
- Timing Constraints (supported file format *.sdc)
- Full Chip Simulation Models (supported file format *.v)
- Placement Constraints (supported file format *.pdc)

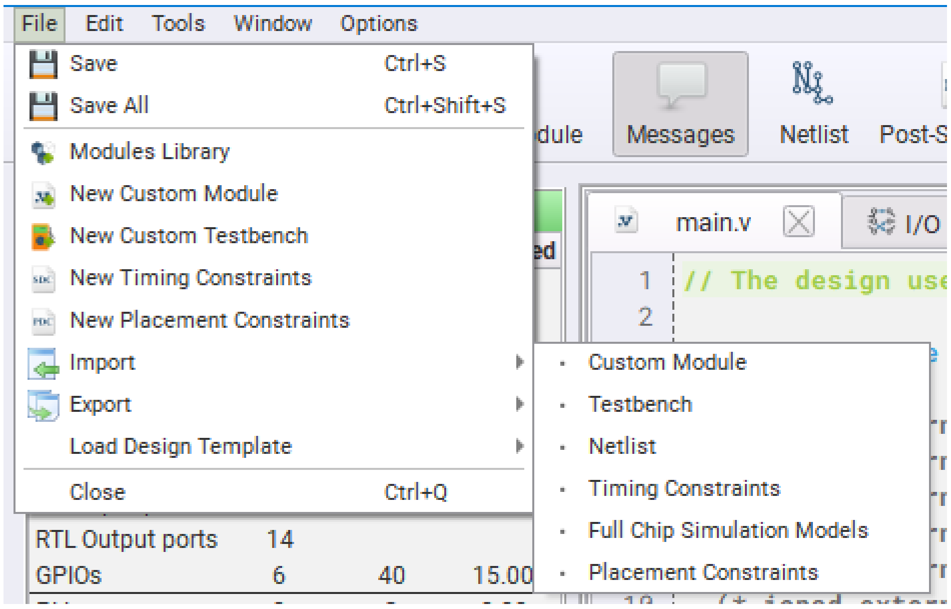


Figure 15. Importing RTL Files

Similarly, files can also be exported (see Figure 16). You can export a custom module, library module, testbench or a timing constraint file that you have designed using ForgeFPGA Workshop and save in your system. You can export via *Main Menu → Export → Custom Module*

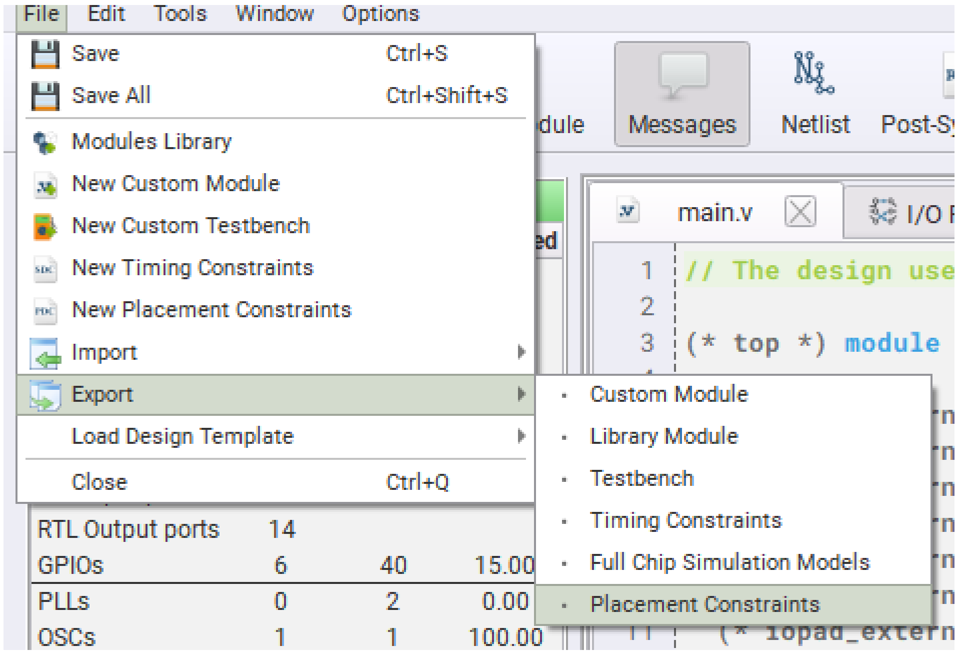


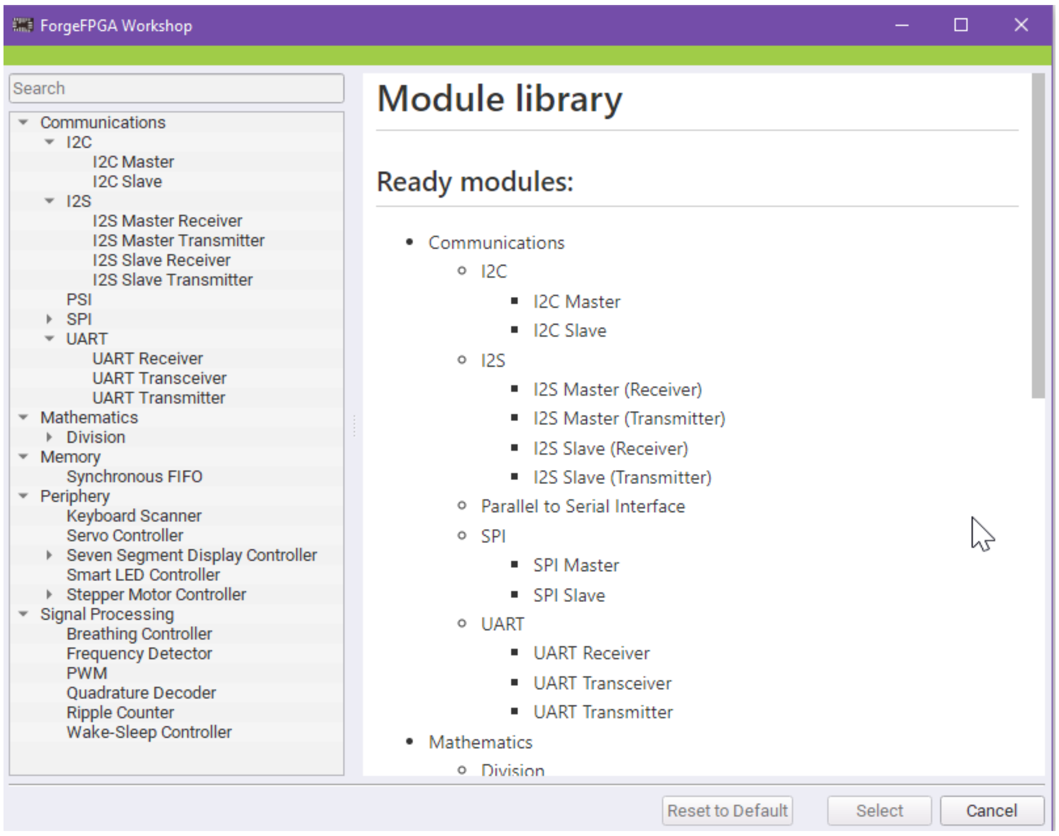
Figure 16. Exporting RTL Files

2.3 Modules Library

The *Modules Library* is a comprehensive repository of pre-designed and pre-verified easy-to-integrate modules. This tool provides HDL code for various hardware modules, accompanied by the testbenches to check their functionality using Simulation.

To open the *Modules Library*, click the corresponding button on the toolbar or navigate the main menu, *File → Modules Library*. Choose the module, set the required configurations, and add the name to complete the module creation (see Figure 17).

Inside the *Modules Library* GUI, you can find the schematic and port information along with the description of the selected block. The GUI gives a detailed explanation of all the input and output pins of the block and allows you to change the parameters as desired. This gives you the flexibility to create the HDL code of the module needed and its associated testbench with just a few clicks.

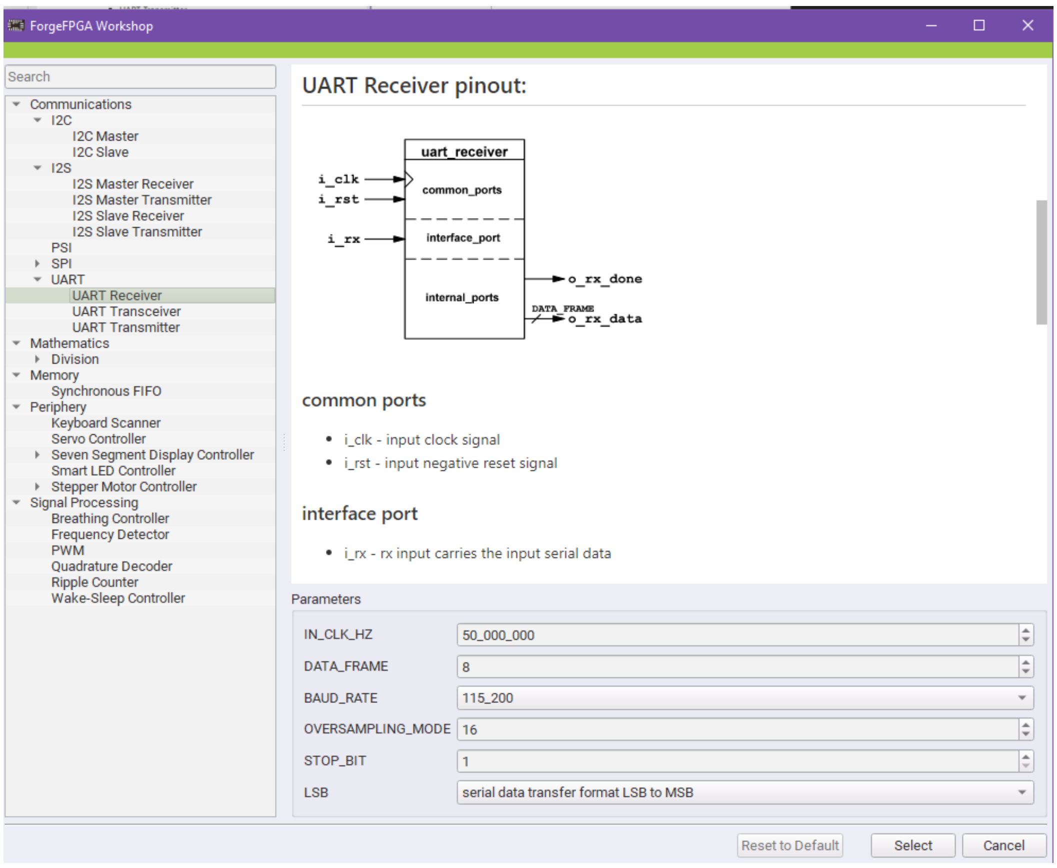


Module library

Ready modules:

- Communications
  - I2C
    - I2C Master
    - I2C Slave
  - I2S
    - I2S Master (Receiver)
    - I2S Master (Transmitter)
    - I2S Slave (Receiver)
    - I2S Slave (Transmitter)
  - Parallel to Serial Interface
  - SPI
    - SPI Master
    - SPI Slave
  - UART
    - UART Receiver
    - UART Transceiver
    - UART Transmitter
- Mathematics
  - Division

Figure 17. Modules Library GUI



- Communications
  - I2C
    - I2C Master
    - I2C Slave
  - I2S
    - I2S Master Receiver
    - I2S Master Transmitter
    - I2S Slave Receiver
    - I2S Slave Transmitter
  - PSI
  - SPI
  - UART
    - UART Receiver
    - UART Transceiver
    - UART Transmitter
- Mathematics
  - Division
- Memory
  - Synchronous FIFO
- Periphery
  - Keyboard Scanner
  - Servo Controller
  - Seven Segment Display Controller
  - Smart LED Controller
  - Stepper Motor Controller
- Signal Processing
  - Breathing Controller
  - Frequency Detector
  - PWM
  - Quadrature Decoder
  - Ripple Counter
  - Wake-Sleep Controller

UART Receiver pinout:

common ports

- i_clk - input clock signal
- i_rst - input negative reset signal

interface port

- i_rx - rx input carries the input serial data

Parameters

IN_CLK_HZ	50_000_000
DATA_FRAME	8
BAUD_RATE	115_200
OVERSAMPLING_MODE	16
STOP_BIT	1
LSB	serial data transfer format LSB to MSB

Figure 18. Module Library GUI with Parameters

The user can create multiple Module Blocks simultaneously as well. All created Module Blocks and their respective testbenches can be seen listed under the sources tab in the *Control Panel*.

2.4 RTL Synthesis

The ForgeFPGA Editor comes with a built-in synthesis tool (Yosys, v0.37) that takes a design as input and produces a Netlist out of it.

While performing synthesis, the input design is analyzed and converted into gate-level representation. To run synthesis on your design, you can press the **Synthesis** button on the bottom of the *Control Panel* or from the main menu *Tools → Run Synthesis*

During the process of synthesis, the software also checks for any **Syntax errors** in the written HDL code and indicates which line has the error in the *Messages Panel*. The Synthesis process will not be completed until the code is free of any syntax errors.

User can access the Synthesis Report from the Synthesis Report button in the toolbar on top after successfully synthesizing the design or from *Windows → Synthesis Report.*

I/O Planner

Each available I/O port on the FPGA has a dedicated function that can be mapped to your design using the *I/O Planner* tool to generate a special configuration file which is then used during the place-and-route procedure.

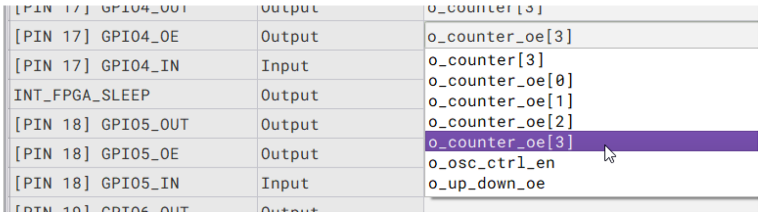
The user can access the *I/O Planner* by clicking the I/O Planner button on the toolbar or selecting it from the main menu *Windows → I/O Planner*.

The *I/O Planner tool* is represented as a table with the following columns:

- *Function* – the dedicated function assigned to the port.

- *Port* – editable column, where you can input ports from your HDL design, to connect them to the desired functionality. By double-clicking the desired port row, the user can choose from the list (available after Synthesis) of all the ports defined in their HDL Code.
- *Direction* – the mentioned direction is with respect to FPGA Core.
- *Description* – data describing the port type and the function it fulfills.

After the design has been synthesized, the signal from port list can be dedicated to the desired function by double-clicking on the white cell under corresponding port and selecting the signal from the list (see Figure 19).

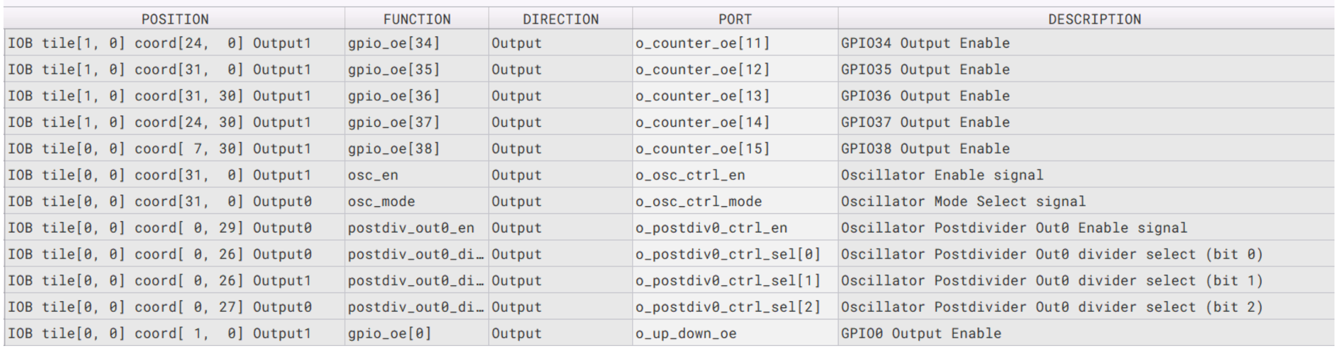


[PIN 17] GPIO4_OUT	Output	o_counter[3]
[PIN 17] GPIO4_OE	Output	o_counter_oe[3]
[PIN 17] GPIO4_IN	Input	
INT_FPGA_SLEEP	Output	
[PIN 18] GPIO5_OUT	Output	
[PIN 18] GPIO5_OE	Output	
[PIN 18] GPIO5_IN	Input	
[PIN 19] GPIO6_OUT	Output	

```
o_counter[3]
o_counter_oe[0]
o_counter_oe[1]
o_counter_oe[2]
o_counter_oe[3]
o_osc_ctrl_en
o_up_down_oe
```

Figure 19. I/O Port Signal Assignment

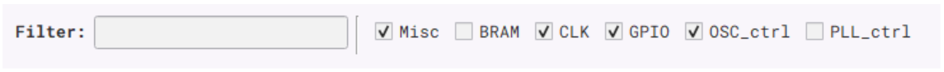
Note: The cell will show a warning icon when the port name is invalid. Hover over the cell to trigger the info pop-up with more details. Correct the name to perform the procedures successfully.



POSITION	FUNCTION	DIRECTION	PORT	DESCRIPTION
IOB tile[1, 0] coord[24, 0] Output1	gpio_oe[34]	Output	o_counter_oe[11]	GPIO34 Output Enable
IOB tile[1, 0] coord[31, 0] Output1	gpio_oe[35]	Output	o_counter_oe[12]	GPIO35 Output Enable
IOB tile[1, 0] coord[31, 30] Output1	gpio_oe[36]	Output	o_counter_oe[13]	GPIO36 Output Enable
IOB tile[1, 0] coord[24, 30] Output1	gpio_oe[37]	Output	o_counter_oe[14]	GPIO37 Output Enable
IOB tile[0, 0] coord[7, 30] Output1	gpio_oe[38]	Output	o_counter_oe[15]	GPIO38 Output Enable
IOB tile[0, 0] coord[31, 0] Output1	osc_en	Output	o_osc_ctrl_en	Oscillator Enable signal
IOB tile[0, 0] coord[31, 0] Output0	osc_mode	Output	o_osc_ctrl_mode	Oscillator Mode Select signal
IOB tile[0, 0] coord[0, 29] Output0	postdiv_out0_en	Output	o_postdiv0_ctrl_en	Oscillator Postdivider Out0 Enable signal
IOB tile[0, 0] coord[0, 26] Output0	postdiv_out0_di…	Output	o_postdiv0_ctrl_sel[0]	Oscillator Postdivider Out0 divider select (bit 0)
IOB tile[0, 0] coord[0, 26] Output1	postdiv_out0_di…	Output	o_postdiv0_ctrl_sel[1]	Oscillator Postdivider Out0 divider select (bit 1)
IOB tile[0, 0] coord[0, 27] Output0	postdiv_out0_di…	Output	o_postdiv0_ctrl_sel[2]	Oscillator Postdivider Out0 divider select (bit 2)
IOB tile[0, 0] coord[1, 0] Output1	gpio_oe[0]	Output	o_up_down_oe	GPIO0 Output Enable

Figure 20. Mapping I/O Ports

To make navigation easier, the I/O Planner provides several filters with checkboxes, that can show/hide groups of ports according to their functionality (see Figure 21). User can also right click on any of the column headers to sort the signals in that column in an ascending or descending order.



Filter: [✓ Misc] BRAM [✓ CLK] [✓ GPIO] [✓ OSC_ctrl] PLL_ctrl

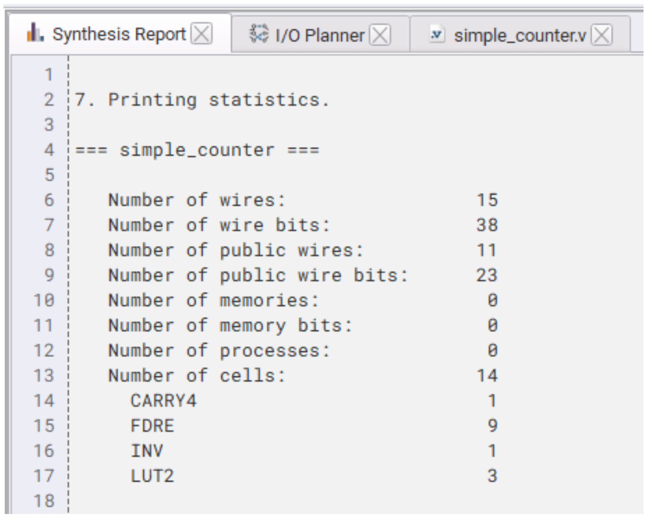
Figure 21. I/O Planner Filter Selection

The user can also clear all the data fed inside the I/O Planner by going to main menu by selecting *Tools → I/O Planner → Clear data* as well as import/export the port data (.txt or .csv format) via main menu by selecting *Tools → I/O Planner → Import I/O Spec*.

Synthesis Report

After performing synthesis of your current project, a new report is generated. This report shows the number of primitive objects used to represent the design in the generated netlist.

To open the synthesis report window, click the corresponding button on the toolbar or via main menu, *Window → Synthesis Report* (see Figure 22).



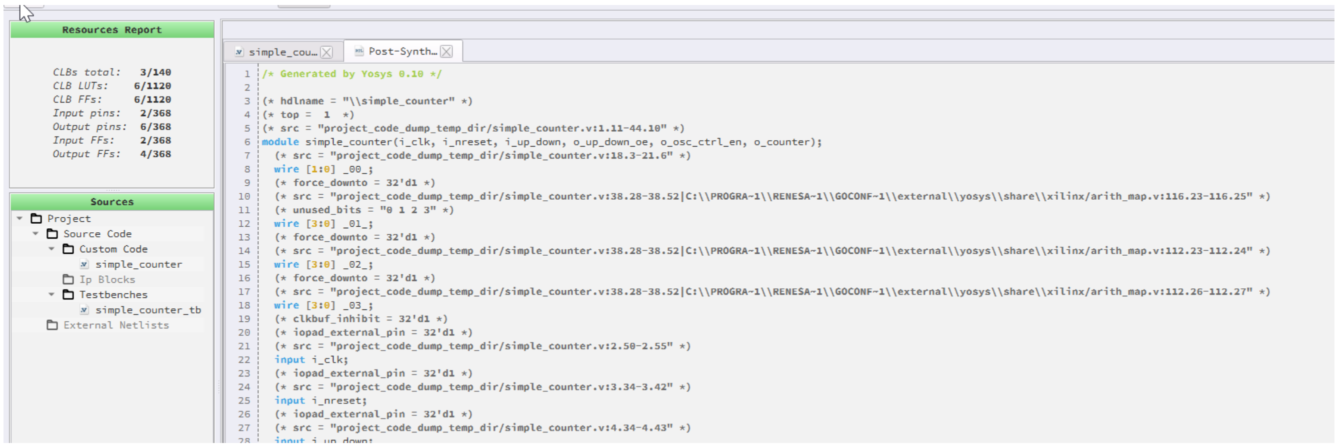
```

7. Printing statistics.

=== simple_counter ===

   Number of wires:                 15
   Number of wire bits:             38
   Number of public wires:          11
   Number of public wire bits:      23
   Number of memories:               0
   Number of memory bits:            0
   Number of processes:              0
   Number of cells:                 14
     CARRY4                          1
     FDRE                            9
     INV                             1
     LUT2                            3

```

Figure 22. Synthesis Report

Post-Synth RTL

At the Register-Transfer Level, the design is represented by combinational data paths and registers. RTL synthesis is easy as each circuit node element in the *Netlist* is replaced with an equivalent gate-level circuit. In Post-Synthesis RTL, the synthesized inputs are taken as a netlist. It helps in providing information about the clock and other clock-related logic in the design, which enables additional I/O planning.

You can find the Post-Synth RTL report by clicking the respective toolbar button or from the main menu, *Window → Post-Synth RTL*. The report contains all the connections made within the module between the LUTs and Carry-Chain logic.



```
CLBs total:    3/140
CLB LUTs:     6/1120
CLB FFs:      6/1120
Input pins:    2/368
Output pins:   6/368
Input FFs:     2/368
Output FFs:    4/368
```

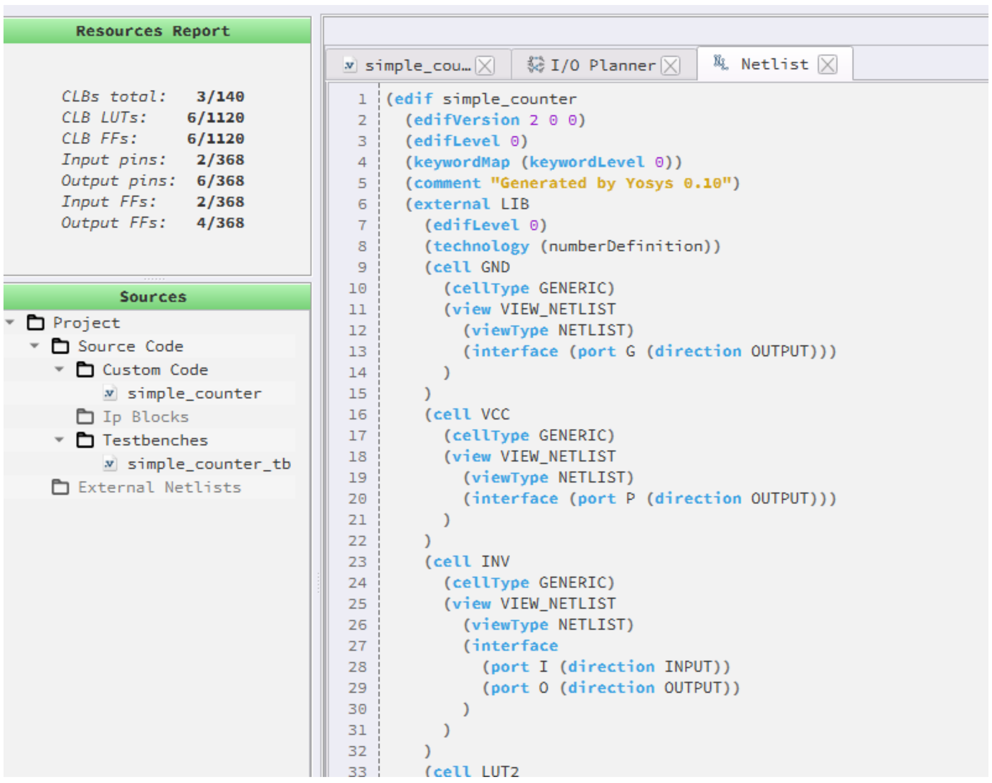
```
/* Generated by Yosys 0.10 */

(* hdlname = "\\simple_counter" *)
(* top =  1  *)
(* src = "project_code_dump_temp_dir/simple_counter.v:1.11-44.10" *)
module simple_counter(i_clk, i_nreset, i_up_down, o_up_down_oe, o_osc_ctrl_en, o_counter);
  (* src = "project_code_dump_temp_dir/simple_counter.v:18.3-21.6" *)
  wire [1:0] _00_;
  (* force_downto = 32'd1 *)
  (* src = "project_code_dump_temp_dir/simple_counter.v:38.28-38.52|C:\\PROGRA~1\\RENESA~1\\GOCONF~1\\external\\yosys\\share\\xilinx/arith_map.v:116.23-116.25" *)
  (* unused_bits = "0 1 2 3" *)
  wire [3:0] _01_;
  (* force_downto = 32'd1 *)
  (* src = "project_code_dump_temp_dir/simple_counter.v:38.28-38.52|C:\\PROGRA~1\\RENESA~1\\GOCONF~1\\external\\yosys\\share\\xilinx/arith_map.v:112.23-112.24" *)
  wire [3:0] _02_;
  (* force_downto = 32'd1 *)
  (* src = "project_code_dump_temp_dir/simple_counter.v:38.28-38.52|C:\\PROGRA~1\\RENESA~1\\GOCONF~1\\external\\yosys\\share\\xilinx/arith_map.v:112.26-112.27" *)
  wire [3:0] _03_;
  (* clkbuf_inhibit = 32'd1 *)
  (* iopad_external_pin = 32'd1 *)
  (* src = "project_code_dump_temp_dir/simple_counter.v:2.50-2.55" *)
  input i_clk;
  (* iopad_external_pin = 32'd1 *)
  (* src = "project_code_dump_temp_dir/simple_counter.v:3.34-3.42" *)
  input i_nreset;
  (* iopad_external_pin = 32'd1 *)
  (* src = "project_code_dump_temp_dir/simple_counter.v:4.34-4.43" *)
  input i_up_down;
```

Figure 23. Post-Synth RTL

Netlist

The system generates a netlist file after successful synthesis of the project. It describes the components and connectivity of the source design and is required to perform the subsequent place-and-route procedure. You can access the netlist by clicking the Netlist button on the toolbar or from the main menu, *Window → Netlist*.

You can also import an external netlist by selecting *File → Import → Netlist* from the Main Menu.



```
CLBs total:    3/140
CLB LUTs:     6/1120
CLB FFs:      6/1120
Input pins:    2/368
Output pins:   6/368
Input FFs:     2/368
Output FFs:    4/368
```

```
Project
  Source Code
    Custom Code
      simple_counter
    Ip Blocks
    Testbenches
      simple_counter_tb
  External Netlists
```

```
(edif simple_counter
  (edifVersion 2 0 0)
  (edifLevel 0)
  (keywordMap (keywordLevel 0))
  (comment "Generated by Yosys 0.10")
  (external LIB
    (edifLevel 0)
    (technology (numberDefinition))
    (cell GND
      (cellType GENERIC)
      (view VIEW_NETLIST
        (viewType NETLIST)
        (interface (port G (direction OUTPUT)))
      )
    )
    (cell VCC
      (cellType GENERIC)
      (view VIEW_NETLIST
        (viewType NETLIST)
        (interface (port P (direction OUTPUT)))
      )
    )
    (cell INV
      (cellType GENERIC)
      (view VIEW_NETLIST
        (viewType NETLIST)
        (interface
          (port I (direction INPUT))
          (port O (direction OUTPUT))
        )
      )
    )
    (cell LUT2
```

Figure 24. Netlist

2.5 Generating the Bitstream

To prepare your design to be programmed onto the device, the place-and-route procedure must be performed, that takes the elements of the synthesized netlist and maps its primitives to FPGA physical resources. This can be done once the *Netlist* files are successfully generated. To generate the bitstream data, click the *Generate Bitstream* button in the bottom left corner of *FPGA Editor* or, from the main menu, *Tools* → *Generate Bitstream*.

You can check Bitstream Log tab on the Message Panel to inspect the background steps after you click Generate Bitstream. In the background, the software automatically performs technology mapping, clustering and floor planning, placement and optimization, routing, and resource calculation. If an issue occurs in any of these steps, the bitstream generation will be incomplete, and you will see outcome on the Messages Panel.

The generated bitstream is saved as a txt and bin file which, if generated correctly, can be used for debugging your design.

After bitstream generation is completed, you can see the generated info for the tools described later in this section.

Floorplan

To help with the visualization of your design, the results of the place-and-route procedure are visualized in the Floorplan window. Here you can check how the primitives from the netlist are placed and interconnected, as well as how I/O ports are mapped to the internal blocks and GPIOs.

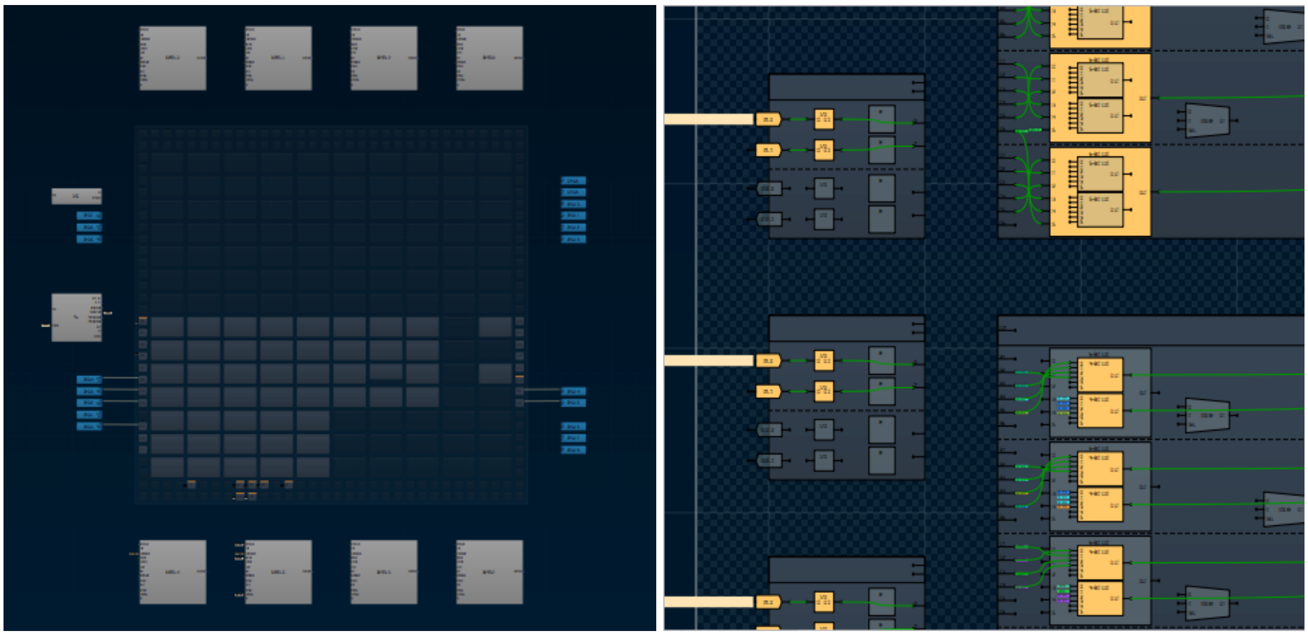
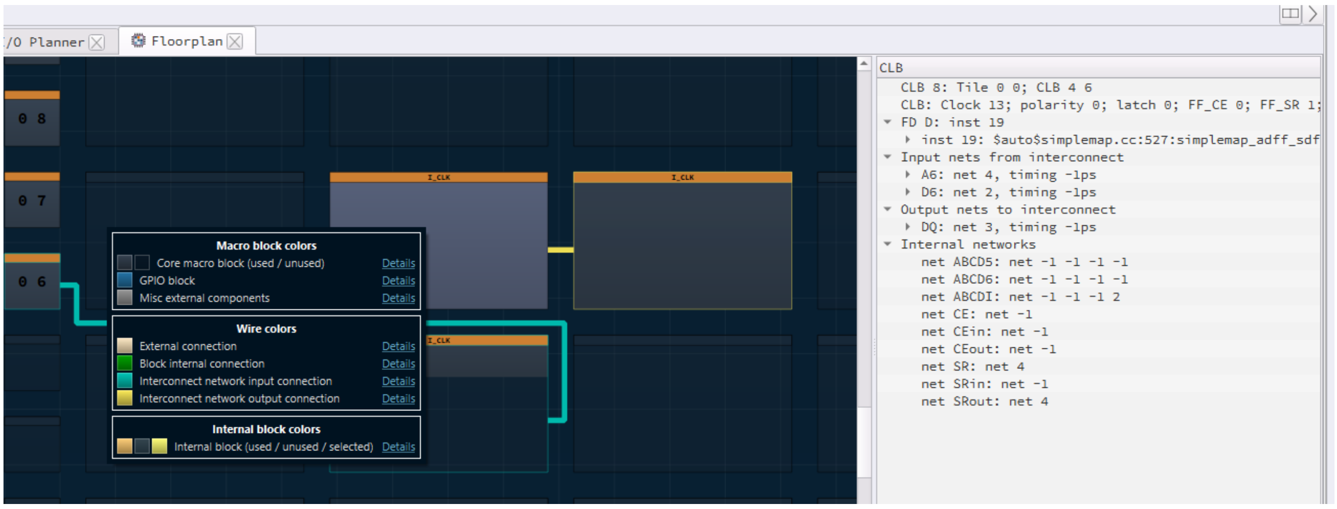


Figure 25. Floorplan Window (SLG47910)

Note: The above floorplan is for the SLG47910. Each device will have a different orientation in the floorplan as per the device design and blocks in it. However, the below mentioned functionality will remain consistent

Launch the Floorplan window by clicking the Floorplan button on the toolbar or from the main menu *Windows → Floorplan*. Use the bottom toolbar controls to take a closer look and navigate within the tool.

Click the internal components to see its details in the block configuration info panel. Also see the color scheme of the components and connections by clicking on the Legend Box icon at the bottom right (see Figure 26).



Macro block colors
- Core macro block (used / unused) Details
- GPIO block Details
- Misc external components Details

Wire colors
- External connection Details
- Block internal connection Details
- Interconnect network input connection Details
- Interconnect network output connection Details

Internal block colors
- Internal block (used / unused / selected) Details

```
CLB
   CLB 8: Tile 0 0; CLB 4 6
   CLB: Clock 13; polarity 0; latch 0; FF_CE 0; FF_SR 1;
 ▾ FD D: inst 19
    ▸ inst 19: $auto$simplemap.cc:527:simplemap_adff_sdf
 ▾ Input nets from interconnect
    ▸ A6: net 4, timing -1ps
    ▸ D6: net 2, timing -1ps
 ▾ Output nets to interconnect
    ▸ DQ: net 3, timing -1ps
 ▾ Internal networks
      net ABCD5: net -1 -1 -1 -1
      net ABCD6: net -1 -1 -1 -1
      net ABCDI: net -1 -1 -1 2
      net CE: net -1
      net CEin: net -1
      net CEout: net -1
      net SR: net 4
      net SRin: net -1
      net SRout: net 4
```

Figure 26. Place and Route Results

Users can also load external custom Place and Route settings (.log file) to display their desired connections in the floorplan. To do so, the user can select *Tools → Floorplan → Load custom PnR*.

Footer Controls (see Figure 27):

- *Zoom*: Zoom in/out the work area
- *Pan Mode*: click, hold and drag the cursor to move the work area (use the middle mouse button as an alternative)
- *Zoom to selection*: click, hold and drag a cross cursor over the element you want to zoom
- *Align Vertical/Horizontal*: align the Macrocells relative to each other on the work area

- *Snap to grid*: use for precise graphic alignment of the Macrocells along a grid.

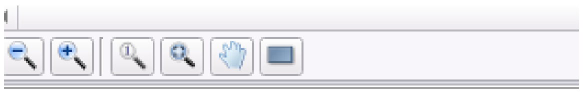


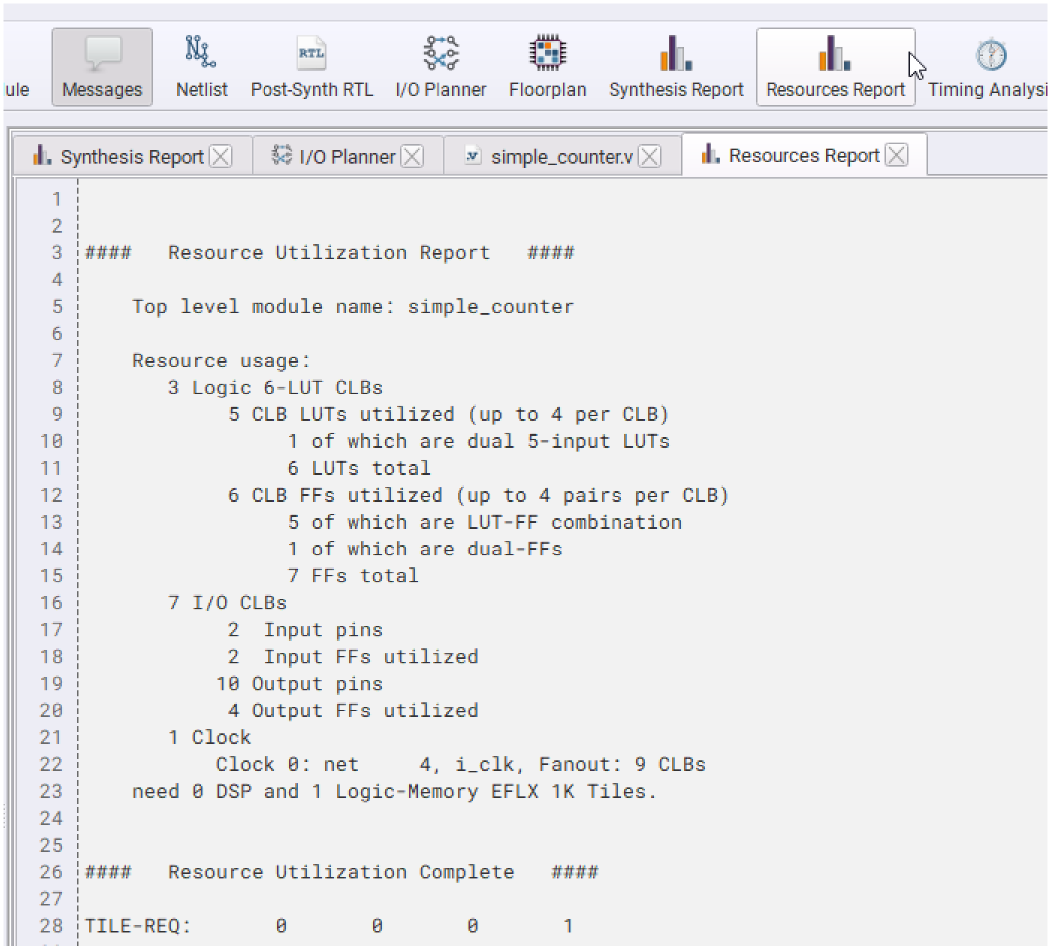
Figure 27. Footer Controls

Resources Report

After successfully performing the bitstream generation procedure, a full report of available resource usage is generated. This report shows the amount of CLBs, FFs, Pins, LUTs are being used for the synthesized design.

A summary of these resources utilized are also mentioned in the *Control Panel*.

The Resources Report window is launched by clicking the **Resources** button on the toolbar or selecting from the main menu *Windows → Resources Report*.



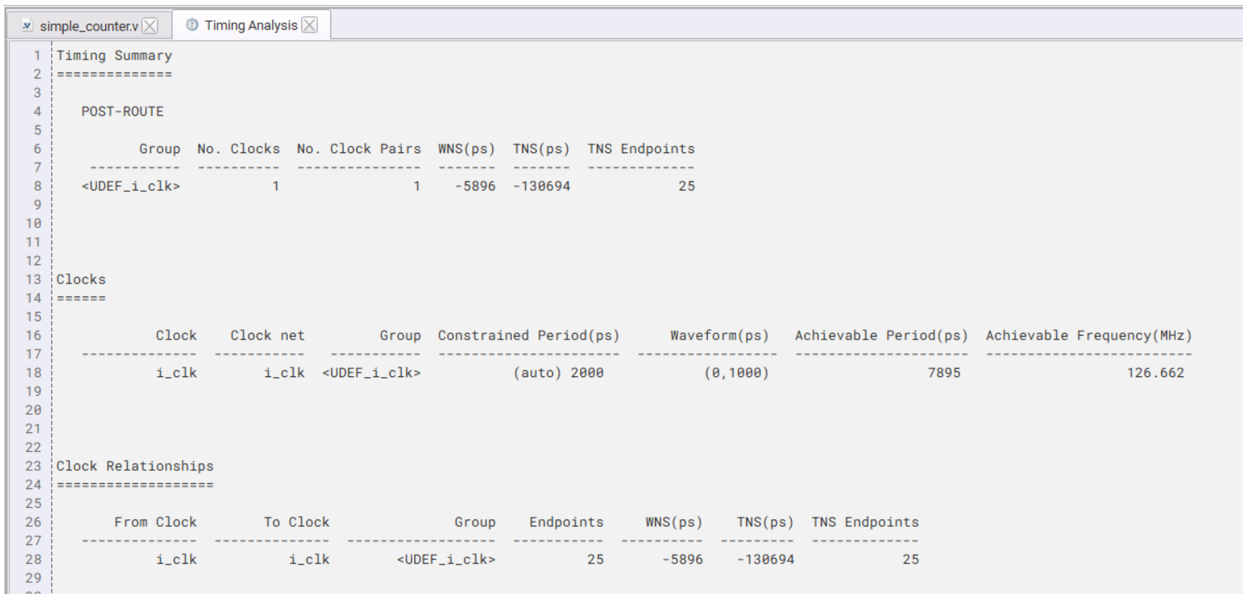
```


####   Resource Utilization Report   ####

    Top level module name: simple_counter

    Resource usage:
       3 Logic 6-LUT CLBs
            5 CLB LUTs utilized (up to 4 per CLB)
                 1 of which are dual 5-input LUTs
                 6 LUTs total
            6 CLB FFs utilized (up to 4 pairs per CLB)
                 5 of which are LUT-FF combination
                 1 of which are dual-FFs
                 7 FFs total
       7 I/O CLBs
            2  Input pins
            2  Input FFs utilized
           10 Output pins
            4 Output FFs utilized
       1 Clock
           Clock 0: net     4, i_clk, Fanout: 9 CLBs
    need 0 DSP and 1 Logic-Memory EFLX 1K Tiles.


####   Resource Utilization Complete   ####

TILE-REQ:       0       0       0       1
```

Figure 28. Resources Utilization Report

Timing Analysis

This tool extracts the timing and checks for any timing violations associated with any internal registers. The results show whether each of the set-up, hold, and pulse-width times are being met or not and lets you correct critical timing issues.

You can find the Timing Analysis tool on the toolbar or by selecting *Window → Timing Analysis* in the main menu.



```
Timing Summary
==============

  POST-ROUTE

         Group  No. Clocks  No. Clock Pairs  WNS(ps)  TNS(ps)  TNS Endpoints
   -----------  ----------  ---------------  -------  -------  -------------
  <UDEF_i_clk>           1                1    -5896  -130694             25




Clocks
======

           Clock    Clock net        Group  Constrained Period(ps)      Waveform(ps)  Achievable Period(ps)  Achievable Frequency(MHz)
  --------------  -----------  -----------  ----------------------  ----------------  ---------------------  -------------------------
           i_clk        i_clk  <UDEF_i_clk>            (auto) 2000          (0,1000)                   7895                    126.662




Clock Relationships
===================

      From Clock       To Clock              Group    Endpoints      WNS(ps)    TNS(ps)  TNS Endpoints
  --------------  -------------  -----------------  -----------  ----------  ---------  -------------
           i_clk          i_clk       <UDEF_i_clk>           25       -5896    -130694             25
```

Figure 29. Timing Analysis

Add timing constraint files (.sdc file) to define clock settings and path-specific requirements (*import* the file via the main menu or right click on Timing Constraints in the Source tree to create a new constraint). The constraints help to validate timing and optimize performance to meet design timing goals.

You can create your own .sdc file with the supported SDC commands and then if needed modify the. sdc constraints to meet your design timing goals by reanalyzing the timing results.

Supported SDC commands are as follows:

- create_clock — creates clock object and defines its characteristics

  ```
  create_clock -period<arg> -name <arg> [get_ports _<arg>] -waveform
  <arg>
  ```

  - `-name clockName [-add] {objectList} |-name clockName [-add] [{objectList}] | [-name clockName [-add]] {objectList}`
  - `-period value`
  - `[-waveform {riseValue fallValue}]`
  - `[-disable]`
  - `[-comment commentString]`

- set_clock_groups — defines relationship between groups of clocks

  ```
  set_clock_groups -group<arg> -group<arg>
  ```

  - `-asynchronous |-physically_exclusive |-logically_exclusive`
  - `[-name clockGroupname]`
  - `-group {clockList} [-group {clockList} <85> ]`
  - `-derive`
  - `[-disable]`
  - `[-comment commentString]`

- set_false_path — sets specific timing paths as being false and excluded from the timing analysis

  ```
  set_false_path -from [ get_pins <arg>] – to [ get_pins <arg>]
  ```

  - `[-setup |-hold]`
  - `[-from {objectList}]`
  - `[-through {objectList} [-through {objectList} ...] ]`
  - `[-to {objectList}]`
  - `[-forward_propagate]`
  - `[-disable]`
  - `[-comment commentString]`

- set_multicycle_path — determines how many clocks cycle a path can take for setup and hold checks

```
set_multicycle_path -from [get_clocks <arg>] -to [get_clocks <arg>] -
from/to [get_pins <arg>] -setup <arg> – hold <arg>
```

  - `[-start |-end]`
  - `[-setup |-hold]`
  - `[-from {objectList}]`
  - `[-through {objectList} [-through {objectList} ...] ]`
  - `[-to {objectList}]`
  - `pathMultiplier`
  - `[-disable]`
  - `[-comment commentString]`

- set_hierarchy_separator — defines the character used to separate different levels of hierarchy in the design
  - {sepchar} | sepchar

The commands below have basic support:

- get_cells
- get_ports
- get_nets
- get_pins
- get_clocks

In case warnings appear during SDC parsing, they can be found in build → ta directory.

The Timing Analyzer examines the timing path in the design, calculates propagation delays, checks for timing violations and reports back Negative slack (WNS) for each path. Total Negative Slack (TNS) indicates the cumulative timing violations in the design. High TNS means more violations. 0 TNS is an ideal case of no violations (see Figure 29 for examples of negative WNS and negative TNS reports).

The Timing Analysis report allows you to identify the violations and modify the .sdc file. If you add, remove or modify any constraints in the .sdc file, re-run the bitstream to generate a new timing report.

Placement Constraints

Add a placement constraint file to control the location of the logic elements on the FPGA IC. Let's break down the supported commands:

- `create_placement_group name_of_the_group` — defines a named placement region with specific coordinates. The first four numbers give the tile X/Y and X/Y within the tile, for the bottom-left corner. The next four numbers give the top-right corner. Note that the corner coordinates are included in the region. See the following syntax example:
  ```
  chip_tile_x=tilex, chip_tile_y=tiley, chip_x=x, chip_y=y,
  chip_tile_x=tilex, chip_tile_y=tiley, chip_x=x, chip_y=y
  ```
  Note: Find the blocks' coordinates on the Floorplan. Clicking the block within the FPGA core opens the info panel with the coordinates data available on the first line.
- `create_cluster_only_placement_group name_of_the_group` — creates a specialized placement group that is restricted to clustered placement only. Upon compilation, the cells (logic instances) in this group are packed near each other to minimize interconnect delay
- `add_to_placement_group name_of_the_group [get_cells cells]` — adds one or more cells (logic instances) to an existing placement group. During placement, the tool ensures these instances stay within the designated region (or next to each other)
- `add_to_placement_group name_of_the_group [get_ports ports]` — adds I/O ports (input or output pins) to an existing placement group, which may help to maintain routing efficiency between I/O and nearby logic

  Note 1: The ordering of the constraints matters. When an instance is assigned to more than one group by different constraints, the last constraint will used.

  Note 2: To include all cells or ports from the design, use the wildcard character (*) instead of listing specific names. See the following example:
- `add_to_placement_group name_of_the_group [get_cells *]`
- `add_to_placement_group name_of_the_group [get_ports *]`

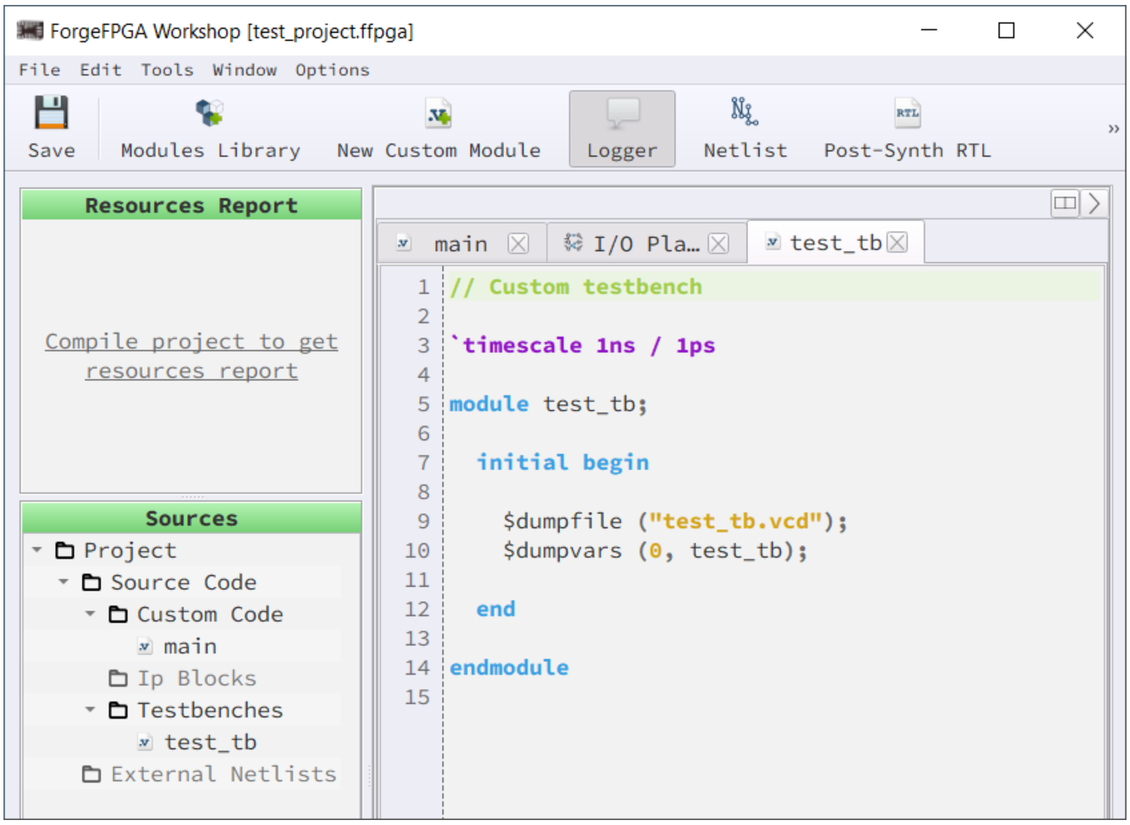
2.6 Simulation

The most crucial step in successfully implementing any system is to verify the design and its functionality in response to different inputs without the need for physical hardware. A testbench is used to test the HDL source code.

The *FPGA Editor* works in correspondence with 3rd party software for simulating the testbench, called Icarus Verilog, and for verifying the functionality of the design by viewing simulation results we use GTKWave. Please refer to the ForgeFPGA Simulation User Guide [7] for the installation process of the additional software and quick start guide.

Writing a Testbench

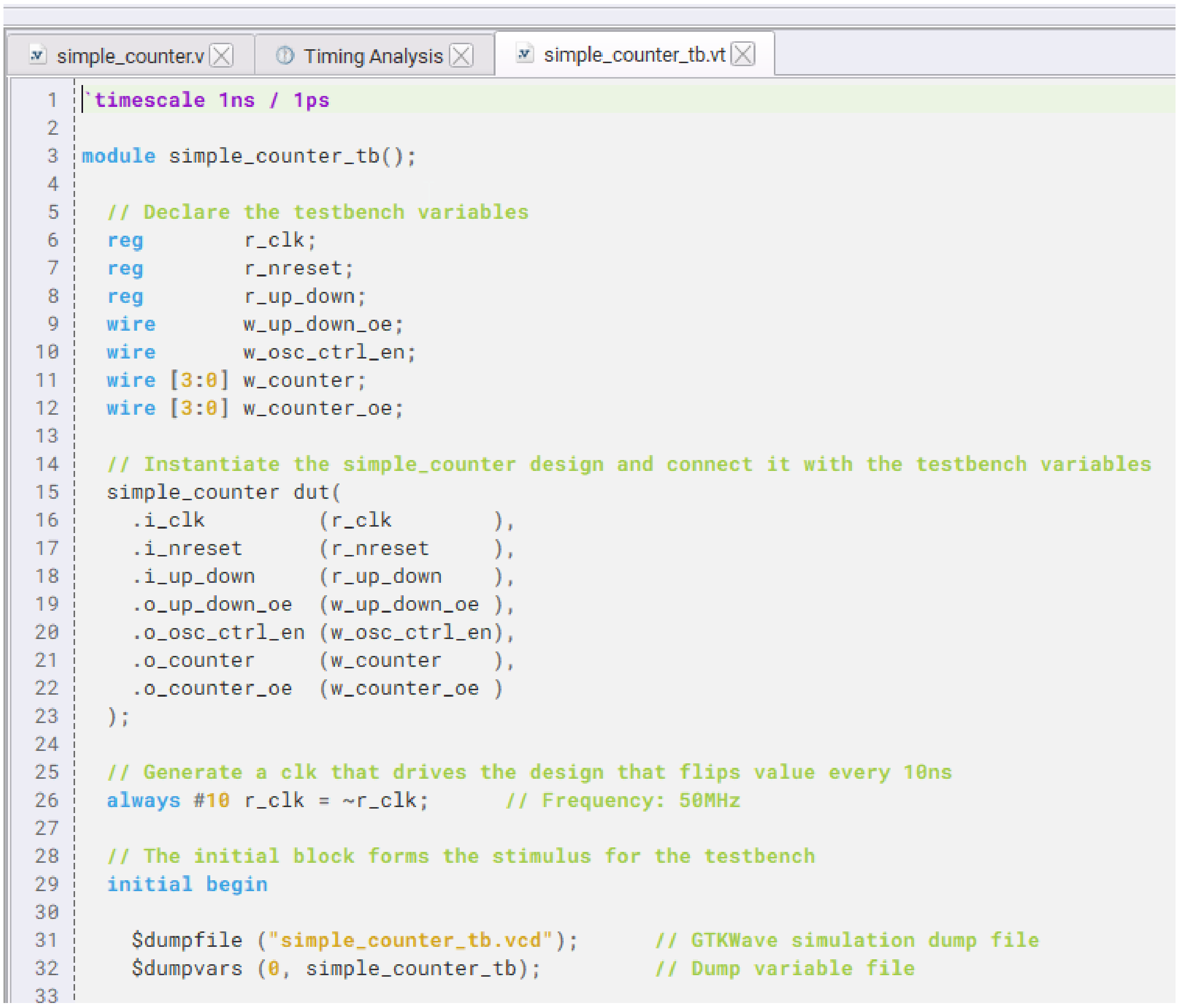
To work with the Simulation, there needs to be a testbench module for the FPGA design. A testbench can be added from the Modules Library or opened from the main menu by going to *File → New Custom Testbench* and saving the name of the testbench. User can also import a testbench module.



```
// Custom testbench

`timescale 1ns / 1ps

module test_tb;

  initial begin

    $dumpfile ("test_tb.vcd");
    $dumpvars (0, test_tb);

  end

endmodule
```

Figure 30. Custom Testbench Example

To make things easy, the software opens the testbench module with a few lines of code to act as a guideline for writing the testbench. For information on how to write a testbench, see [8].



```
`timescale 1ns / 1ps

module simple_counter_tb();

  // Declare the testbench variables
  reg         r_clk;
  reg         r_nreset;
  reg         r_up_down;
  wire        w_up_down_oe;
  wire        w_osc_ctrl_en;
  wire [3:0]  w_counter;
  wire [3:0]  w_counter_oe;

  // Instantiate the simple_counter design and connect it with the testbench variables
  simple_counter dut(
    .i_clk         (r_clk        ),
    .i_nreset      (r_nreset     ),
    .i_up_down     (r_up_down    ),
    .o_up_down_oe  (w_up_down_oe ),
    .o_osc_ctrl_en (w_osc_ctrl_en),
    .o_counter     (w_counter    ),
    .o_counter_oe  (w_counter_oe )
  );

  // Generate a clk that drives the design that flips value every 10ns
  always #10 r_clk = ~r_clk;      // Frequency: 50MHz

  // The initial block forms the stimulus for the testbench
  initial begin

    $dumpfile ("simple_counter_tb.vcd");       // GTKWave simulation dump file
    $dumpvars (0, simple_counter_tb);          // Dump variable file

```

Figure 31. Simple Counter Testbench Example From Template Design

Simulation Options

RTL Simulation executes the RTL only. *Full Chip RTL Simulation* allows you to perform extended behavioral simulation of your design, which simulates not only the logic behavior of the code but also the behavior of the FPGA Core's periphery components (including GPIOs, BRAMs, LVDS, OSC, PLLs, etc.). This simulation type also includes basic delays of peripheral circuits, producing results that more accurately reflect the device's real behavior.

RTL Simulation

After the user is satisfied with the written testbench, click the Simulation button on the toolbar to launch Icarus Verilog and the GTKWave software, or from the main menu go to *Tools → Simulation → Run RTL Simulation.*

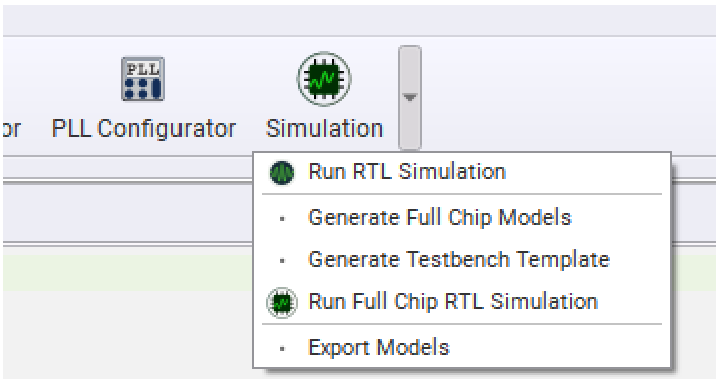

PLL Configurator
Simulation
Run RTL Simulation
Generate Full Chip Models
Generate Testbench Template
Run Full Chip RTL Simulation
Export Models

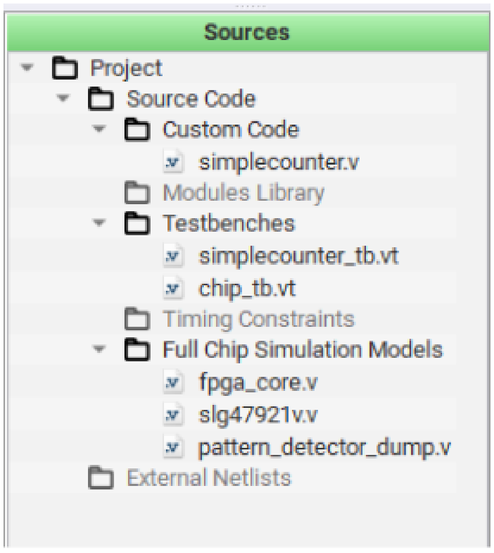
Figure 32. FPGA Simulation On Toolbar

The simulation stage is handled by Icarus Verilog in the background, while the GTKWave shows a visual representation. If the written testbench is correct and doesn't have any syntax errors, the GTKWave software will launch automatically, otherwise check the Messages Panel for any syntax related issues in the written testbench and make any necessary changes.

Full Chip RTL Simulation

Performing a valid Full Chip Simulation requires the following steps and conditions:

- Your project should contain error-free RTL code
- Perform successful *synthesis* stage (netlist is used for top module ports)
- *Synthesis* results should be up to date with your design
- Perform successful *bitstream generation* stage (post PnR results are utilized to assign correct delays in the generated models)
- Perform model generation (*Simulation → Generate Full Chip Models*). The will appear in the Sources tree



Sources
- Project
  - Source Code
    - Custom Code
      - simplecounter.v
    - Modules Library
    - Testbenches
      - simplecounter_tb.vt
      - chip_tb.vt
    - Timing Constraints
    - Full Chip Simulation Models
      - fpga_core.v
      - slg47921v.v
      - pattern_detector_dump.v
  - External Netlists

Figure 33. Full Chip Simulation in Source Tree

***Note**:* Generated models are not part of the design and are not synthesized. They are used for full chip simulation only.

- Option: perform template testbench generation (Simulation → Generate Testbench Template)
- Run simulation for the selected testbench (Simulation → Full Chip RTL Simulation)

Models generated for the currently opened project can be exported to a dump file. It can be imported and used, for example, to perform two chips simulation in another project (Simulation Export Models).

In case of simulation failure, check the *Messages* panel (General Log or Issues tab) and make necessary changes.

The *FPGA Editor* ensures you can keep working from the last saved state of the simulation results. Once you make the necessary configuration changes (e.g. add signals, adjust their graphical representation, etc.), save the progress in the GTKWave tool. Next time you simulate the same testbench, you will start from where you left off from the previous attempt.

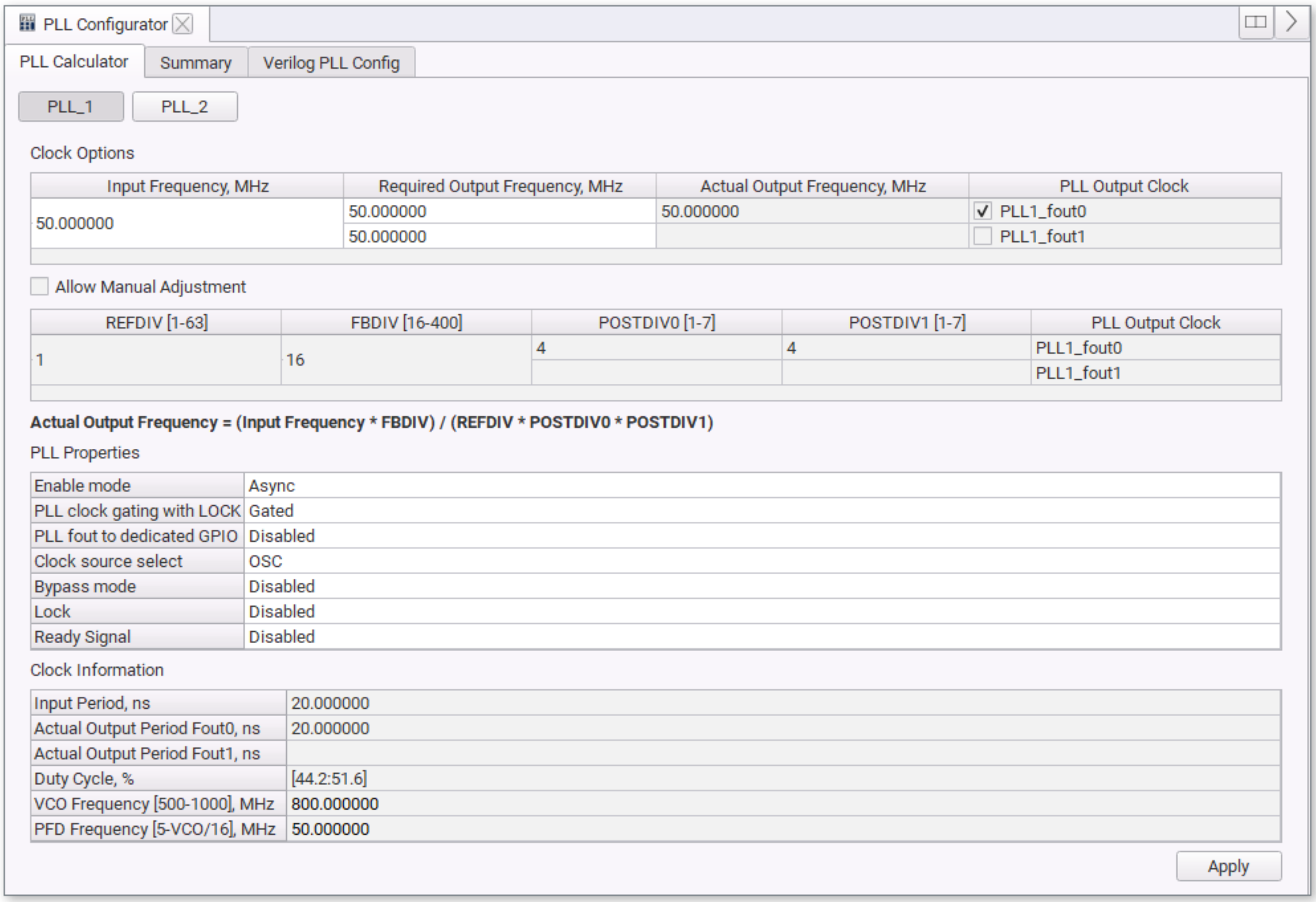
Check Application Notes [4] to know more about how to run Full Chip Simulation.

2.7 PLL Configurator

The PLL Configurator helps to determine the parameters for the desired output signal (or dividers). It can be accessed via *FPGA Editor → PLL Configurator*.

PLL Calculator tab

The main tab allows configuration of the parameters and assignment to the Verilog template, I/O Planner and registers (see Figure 34). For Part Numbers with multiple PLLs, tick the checkboxes located on the right to enable the component and proceed with configuration.



Clock Options

Input Frequency, MHz	Required Output Frequency, MHz	Actual Output Frequency, MHz	PLL Output Clock
50.000000	50.000000	50.000000	✓ PLL1_fout0
	50.000000		☐ PLL1_fout1

☐ Allow Manual Adjustment

REFDIV [1-63]	FBDIV [16-400]	POSTDIV0 [1-7]	POSTDIV1 [1-7]	PLL Output Clock
1	16	4	4	PLL1_fout0
				PLL1_fout1

Actual Output Frequency = (Input Frequency * FBDIV) / (REFDIV * POSTDIV0 * POSTDIV1)

PLL Properties

Enable mode	Async
PLL clock gating with LOCK	Gated
PLL fout to dedicated GPIO	Disabled
Clock source select	OSC
Bypass mode	Disabled
Lock	Disabled
Ready Signal	Disabled

Clock Information

Input Period, ns	20.000000
Actual Output Period Fout0, ns	20.000000
Actual Output Period Fout1, ns	
Duty Cycle, %	[44.2:51.6]
VCO Frequency [500-1000], MHz	800.000000
PFD Frequency [5-VCO/16], MHz	50.000000

Figure 34. PLL Calculator

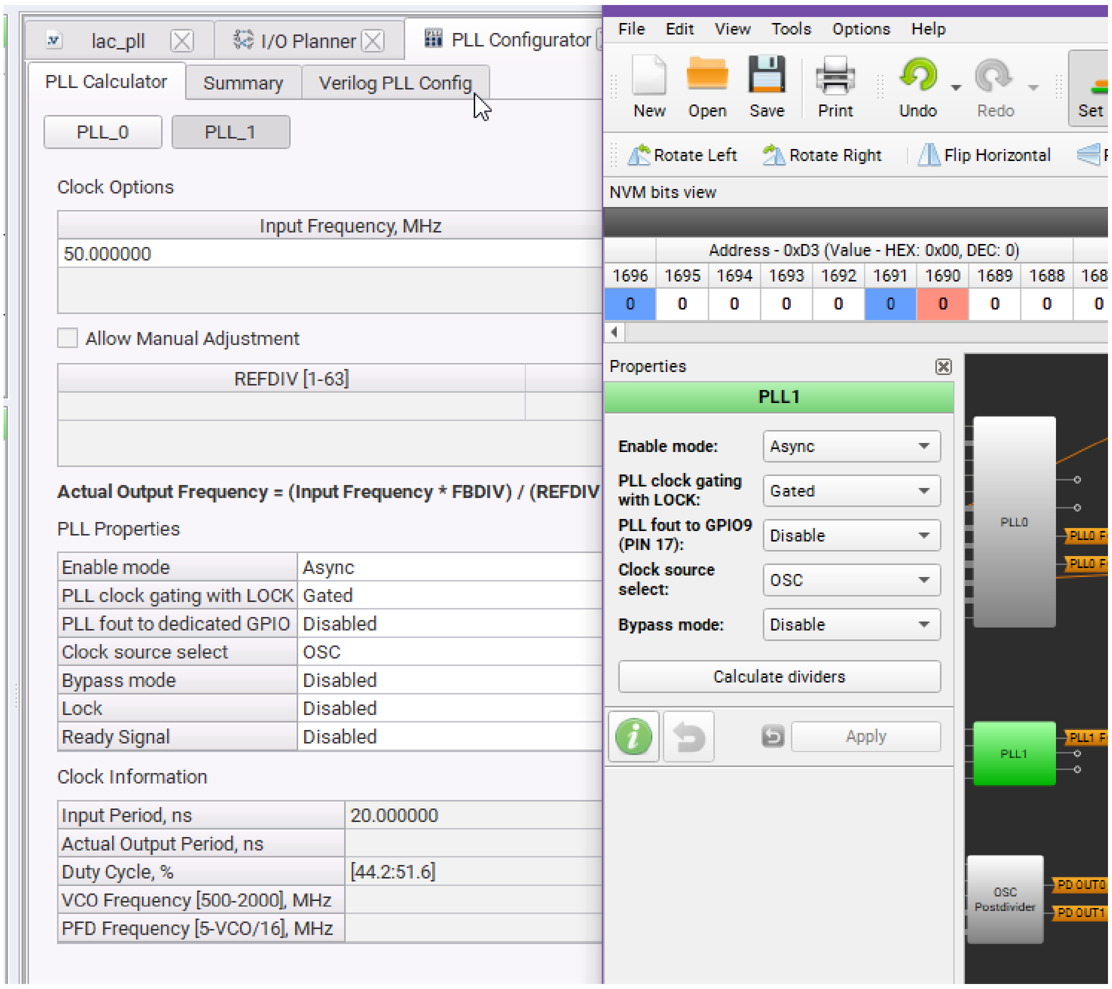
The PLL Configurator tool has the following calculation modes:

- Dividers auto calculation — set the Input Frequency and Required Output Frequency parameters to achieve the desired divider values (REFDIV, FBDIV, POSTDIV1, and POSTDIV2)
- Dividers manual adjustment — enter Input Frequency along with all four dividers to receive the Actual Output Frequency value

Note: Check *Allow Manual Adjustment* to activate the mode.

Configure the parameters via the *PLL Properties* table and pass the data to registers by clicking *Apply*. The *Apply* button saves changes for each PLL separately. Manage the behavior of the confirmation window before applying the changes in Settings. The *Apply* button populates the IO Planner's PLL signals with the default names.

PLL data can also be configured via the component's Properties panel. The configurations are synchronized between the panel and *FPGA Editor* (see Figure 35).



Clock Options

Input Frequency, MHz
50.000000

Allow Manual Adjustment

REFDIV [1-63]

Actual Output Frequency = (Input Frequency * FBDIV) / (REFDIV

PLL Properties

Enable mode	Async
PLL clock gating with LOCK	Gated
PLL fout to dedicated GPIO	Disabled
Clock source select	OSC
Bypass mode	Disabled
Lock	Disabled
Ready Signal	Disabled

Clock Information

Input Period, ns	20.000000
Actual Output Period, ns	
Duty Cycle, %	[44.2:51.6]
VCO Frequency [500-2000], MHz	
PFD Frequency [5-VCO/16], MHz	

Address - 0xD3 (Value - HEX: 0x00, DEC: 0)

1696	1695	1694	1693	1692	1691	1690	1689	1688	168
0	0	0	0	0	0	0	0	0	0

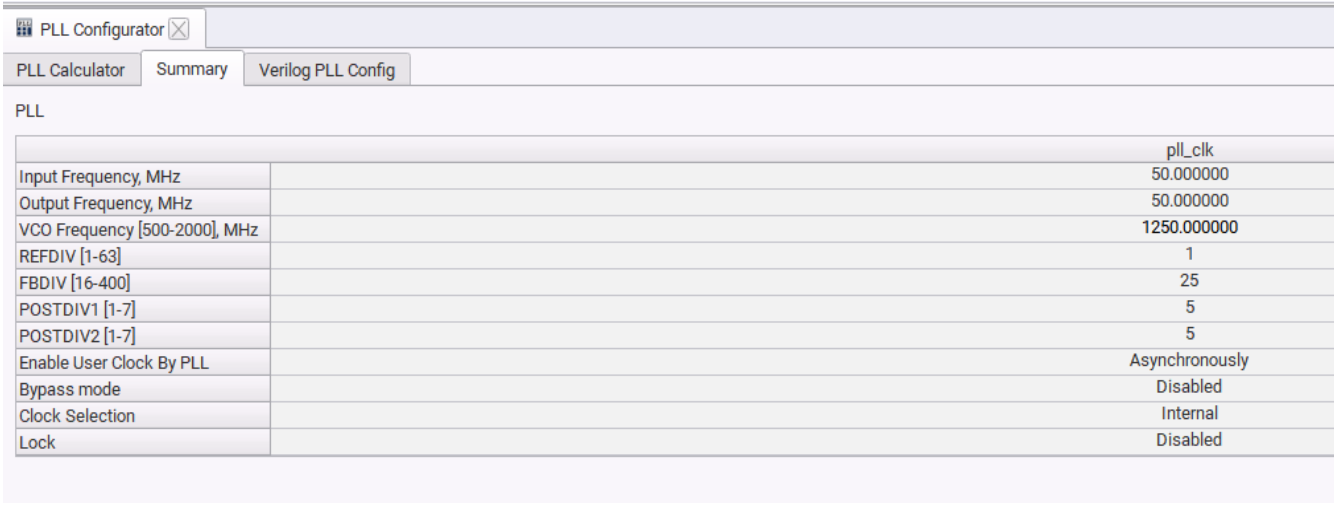
Enable mode:	Async
PLL clock gating with LOCK:	Gated
PLL fout to GPIO9 (PIN 17):	Disable
Clock source select:	OSC
Bypass mode:	Disable

Figure 35. PLL Properties in PLL Configurator and PLL Properties

The Clock Information table values are based on the set data. The results exceeding the range are highlighted in red.

Summary tab

The current page displays the list of all configured parameters on the *PLL Calculator* tab in one place. When more than one PLL is used, the summary displayed the list of both the PLLs.



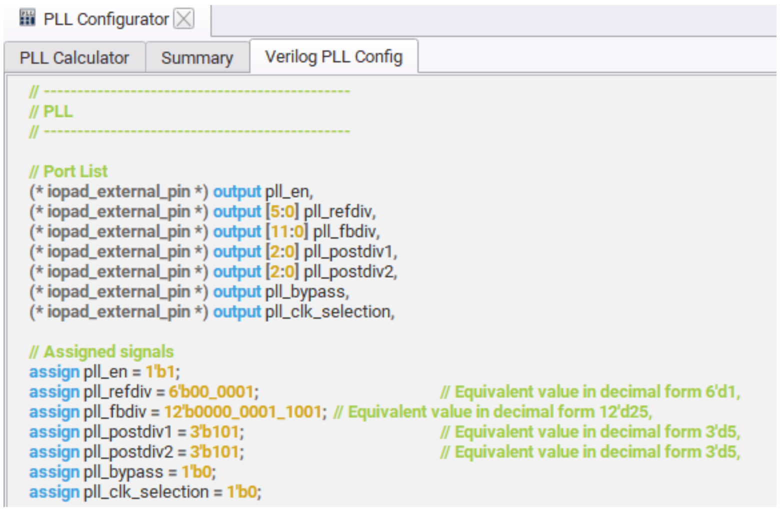
PLL

	pll_clk
Input Frequency, MHz	50.000000
Output Frequency, MHz	50.000000
VCO Frequency [500-2000], MHz	1250.000000
REFDIV [1-63]	1
FBDIV [16-400]	25
POSTDIV1 [1-7]	5
POSTDIV2 [1-7]	5
Enable User Clock By PLL	Asynchronously
Bypass mode	Disabled
Clock Selection	Internal
Lock	Disabled

Figure 36. Summary Tab

Verilog PLL Config tab

This tab shows the dynamically generated Verilog code based on the predefined settings in the *PLL Configurator* tool, which can be used to configure the PLL component. You can copy and paste the auto generated Verilog PLL Configurator code into your top module.
Note: The port names generated in the Verilog code has default names and it matches the automatically generated signals in the IO Planner.

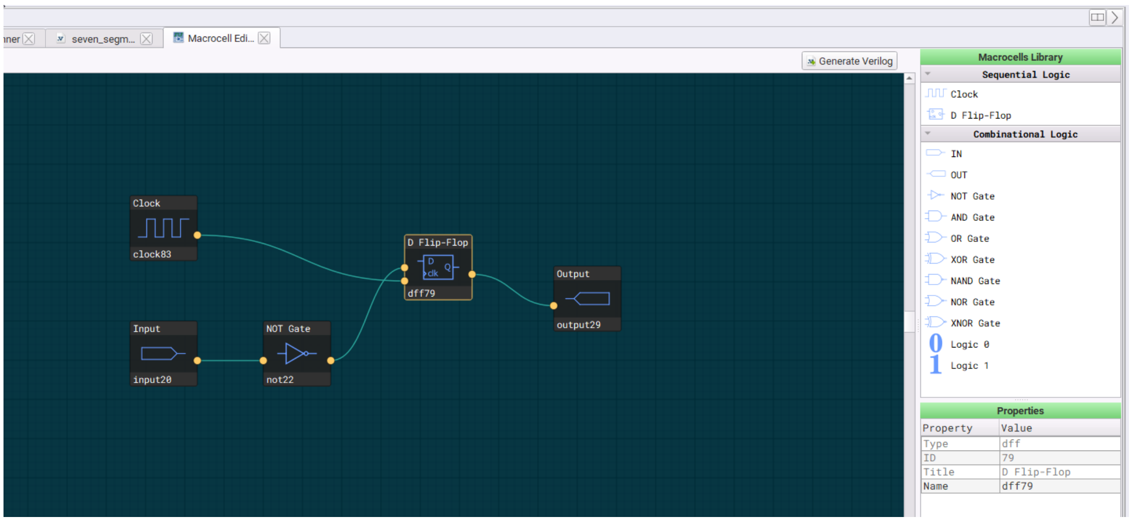


```
// ---------------------------------------------
// PLL
// ---------------------------------------------

// Port List
(* iopad_external_pin *) output pll_en,
(* iopad_external_pin *) output [5:0] pll_refdiv,
(* iopad_external_pin *) output [11:0] pll_fbdiv,
(* iopad_external_pin *) output [2:0] pll_postdiv1,
(* iopad_external_pin *) output [2:0] pll_postdiv2,
(* iopad_external_pin *) output pll_bypass,
(* iopad_external_pin *) output pll_clk_selection,

// Assigned signals
assign pll_en = 1'b1;
assign pll_refdiv = 6'b00_0001;                    // Equivalent value in decimal form 6'd1,
assign pll_fbdiv = 12'b0000_0001_1001; // Equivalent value in decimal form 12'd25,
assign pll_postdiv1 = 3'b101;                      // Equivalent value in decimal form 3'd5,
assign pll_postdiv2 = 3'b101;                      // Equivalent value in decimal form 3'd5,
assign pll_bypass = 1'b0;
assign pll_clk_selection = 1'b0;
```

Figure 37. Verilog PLL Config Tab

2.8 Macrocell Editor

Macrocell mode is a tool that allows the user to create the desired circuit in a schematic view. To launch Macrocell mode, the user can click the *Macrocell Editor button* on the toolbar, or the user can go to the main menu *Window → Macrocell Editor*.

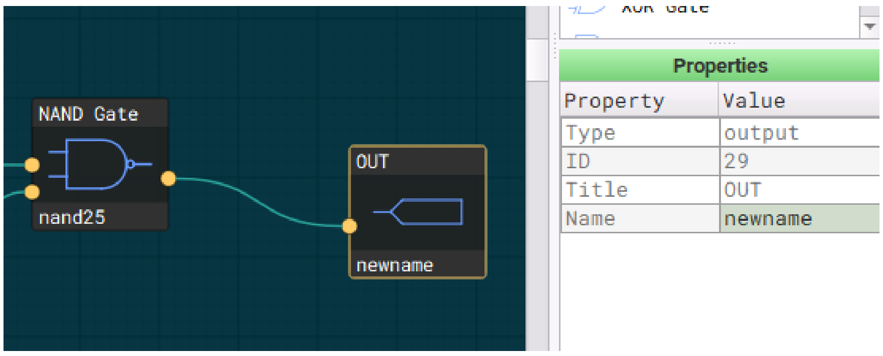


Property	Value
Type	dff
ID	79
Title	D Flip-Flop
Name	dff79

Figure 38. Macrocell Editor

The *Macrocell Editor* consists of two parts; the green screen pane on the left is where the schematic of the circuit is designed by using the components on the right side of the screen which shows a library of the all the *Sequential Logic*, *Combinational Logic, Blocks, and IP Modules* components. The user can simply drag and drop the desired component from the library on to the Green Screen to complete the circuit. Two components can be joined by clicking the two associated ports.

The *Properties* Panel show the details for the selected Macrocell or connection. The Name field is editable by double-clicking the field or Macrocell (see Figure 39).



Property	Value
Type	output
ID	29
Title	OUT
Name	newname

Figure 39. Changing The Names Of Input/Output Pins

The Macrocells may have clock and logic port types, and which can be distinguished by color. Clicking two ports of the same type creates the connection between the Macrocells.

To generate the Verilog code based on the created design, click the *Generate Verilog* button on the right side of the green window. The created Verilog code is listed as *mcm_generated* under the *Custom Code* section of the control panel. You can use this code for the synthesis along with other modules added in the design

2.9 Project Directory Folder

The software creates a folder containing the generated files as soon as any procedures are performed (synthesis, or both synthesis and bitstream generation). The folder contains the project file and a .ffpga folder (described later in this section) and will also store any the modules and netlists, which can be added or imported to the Sources tree in the FPGA Editor. The changes made to the sources will synchronize between the FPGA Editor and the file on disk.

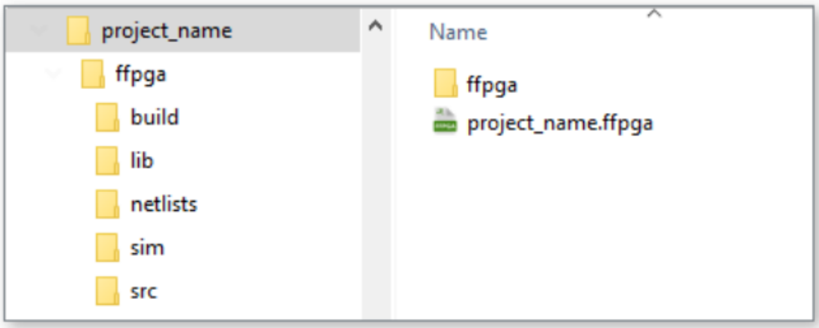


Figure 40. FPGA Project Folder Structure

The folders below store the following files:

- *Build* – result files generated after synthesis and bitstream generation
- *lib* – modules from the Modules Library
- *sim* – testbench and simulation results files
- *src – main (top module), mcm_generated (*created by clicking *Generate Verilog* in *Macrocell Editor*), and imported modules
- *netlists* – imported netlists
- *timing-constraints* – timing constraints files

You can also use the Open Containing Folder feature in the Sources Panel to quickly locate the file on disk (see Figure 41).

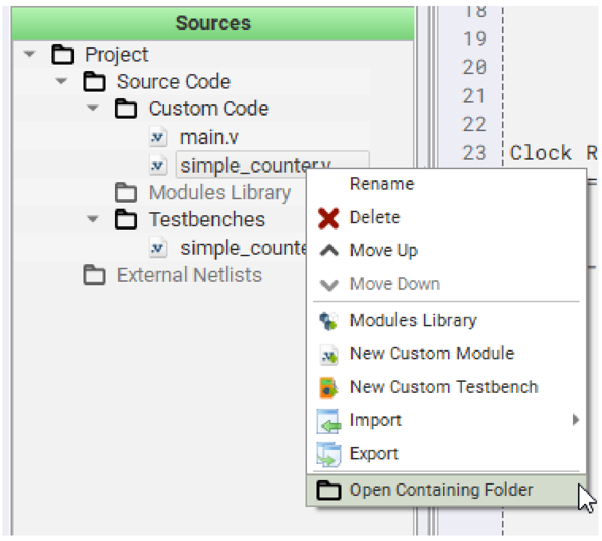


Figure 41. Check Source File location

The list of generated files depends on the procedures that have been performed. The descriptions of the most important files present in the build folder are the following:

- *Bitstream* – The folder containing the bitstream files in binary (.bin) and hexadecimal (.txt) format
  - *FPGA_bitstream*_FLASH_MEM – the bitstream sequence that can be used to program the flash
  - FPGA_bitstream_MCU – the bitstream sequence that can be used for MCU programming
  - FPGA_bitstream_OTP – the bitstream sequence that can be used for OTP programming
- *ta* – stores log files containing potential warnings encountered during the parsing of SDC commands in the Timing Constraints file
- *FPGA_bitstream_AXI.log* – Contains the generated bitstream that represents your project's logic. This file is used while interacting with the development hardware upon performing the chip/flash procedures.
- *io_spec_in.txt* – Lists the I/O ports specification added in the I/O Planner, which is created upon clicking Generate Bitstream.
- *PNR_IO.log* – I/O ports mapping file generated after successful bitstream generation procedure.
- *netlist.edif* – The result of Yosys data processing, describing the components and connectivity within the source design. You can also import an external netlist from another software/synthesis tool.
- *PNR_PACK_PLACE.log* - Source data for building a floorplan.
- *Post_synth_results.v* – Yosys output data in the Verilog code format
- *Resource-utilization-report.log* – Information about the resources amount required for your design.
- *Simulation* – Folder containing simulation results and temporary files.

3. Debug

The design is now ready to be tested out on the development board. The generated bitstream is automatically sent to the device which is ready to be programmed further. The user now needs to switch to the main screen of the ForgeFPGA Software.

The Debug button in the toolbar starts the *Debug Tool* in the ForgeFPGA Workshop window. The Debug tool enables electronic circuit emulation and chip programming, which uses a specific hardware platform to replicate the behavior of the chip components in the design. Before starting the emulation process, test point (TP) controls need to be added to configure the emulation process. The Test Point controls allow the user to configure the GPIOs in different options.

3.1 Development Platforms

After the Debug button is pressed, the Development Platform Selector and the three hardware options will be displayed (see Figure 42). Each Development Platform comes with its own set of Debugging Control settings discussed in Section 3.2.

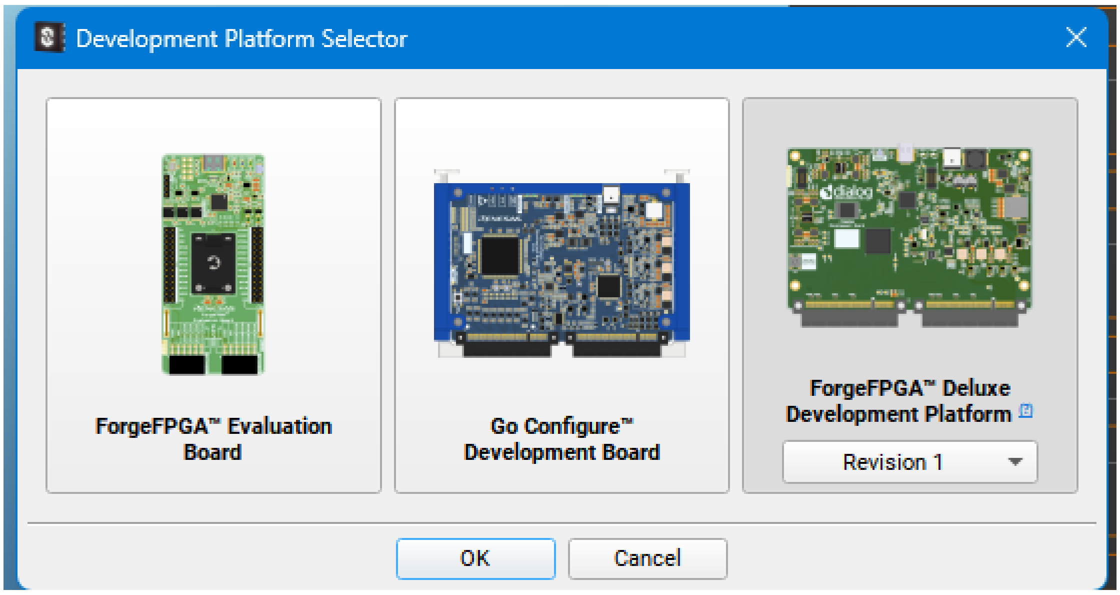

Development Platform Selector
ForgeFPGA™ Evaluation Board
Go Configure™ Development Board
ForgeFPGA™ Deluxe Development Platform
Revision 1
OK
Cancel

Figure 42. Development Platform Selector

ForgeFPGA Deluxe Development Platform

The ForgeFPGA Deluxe Development Platform is a multi-functional tool that makes it possible to develop, program, and emulate FPGA designs by providing an onboard power source, digital signal generation, and logic analysis capabilities. The platform can connect additional external boards called socket adapters (see Figure 43). The function of the socket adapter board is to implement an electrical connection between the pins of the chip under test and the ForgeFPGA Deluxe Development Platform. To implement this, the platform uses a Dual PCIe connector.

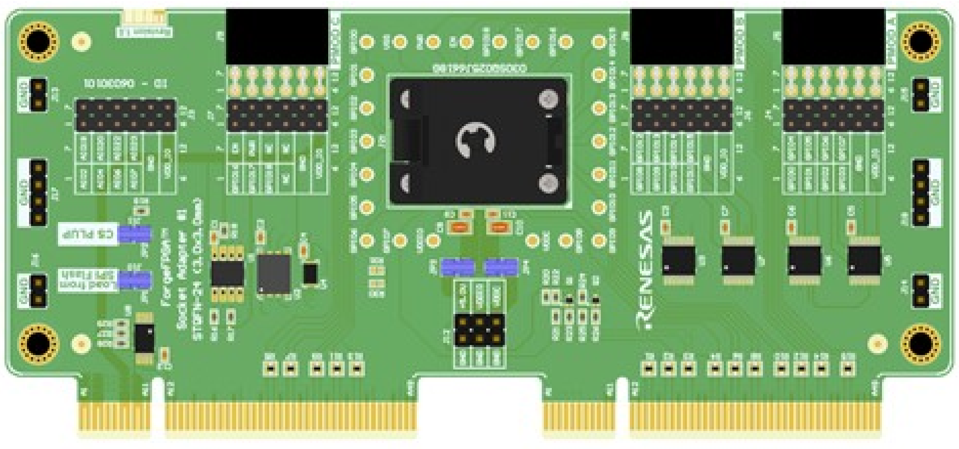


Figure 43. Socket Board Adapter

The board can also be used as an independent unit. The chip can be powered through the EXT PWR connector and signals can be read through the through-hole 12-pin PMOD connectors.

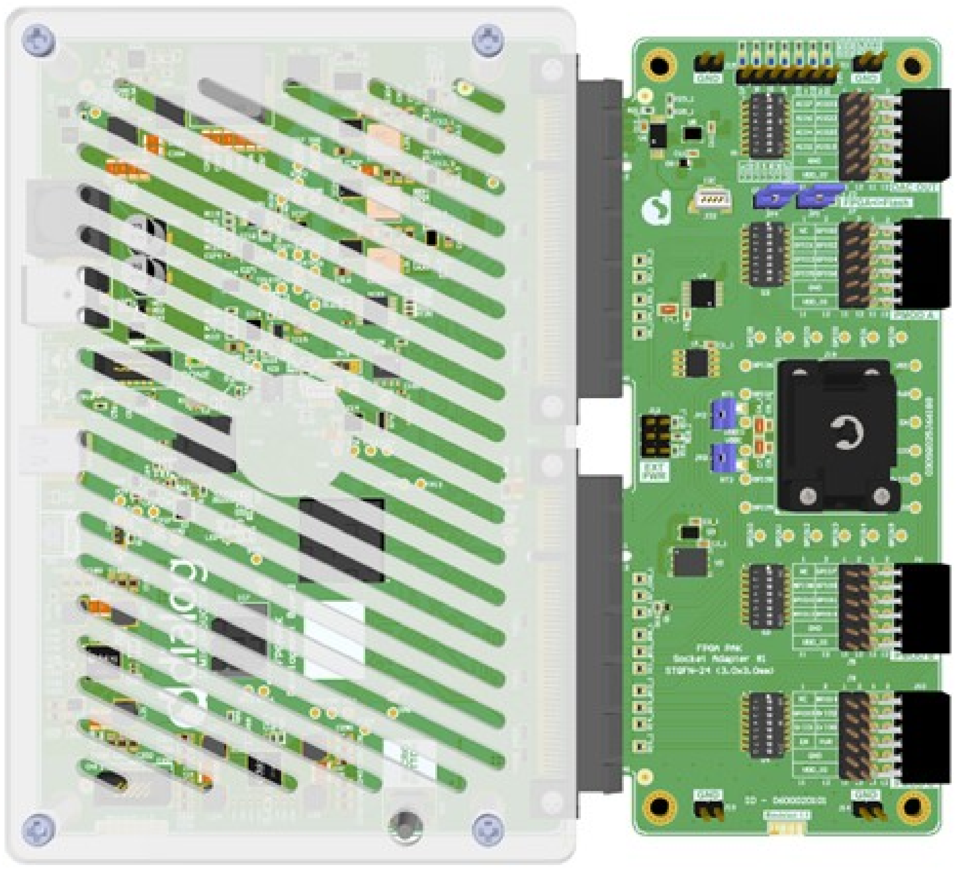


Figure 44. Assembled equipment for working with a chip in the socket.

To start working with the FPGA device, connect the development board to the computer via a USB Type-C cable and connect the power supply. If all the connections are correct, then the red LED (PWR) and blue LED (STS) will light up. For more information on the Deluxe Development Board and the Socket Adaptor please refer to [5][6].

ForgeFPGA Evaluation Board

ForgeFPGA Evaluation Board is a compact, easy-to-use, USB powered hardware tool. There are two versions of this platform (version 2.0 and 1.0).

ForgeFPGA Evaluation Board v2.0 provides the IC with hardware support for design emulation, programming, internal UART terminal options, and real-time testing. The platform mainly consists of the following blocks: programmer, socket, GPIO external connectors, and PMOD connectors. This board uses a USB Type-C connector for communications and power supply.

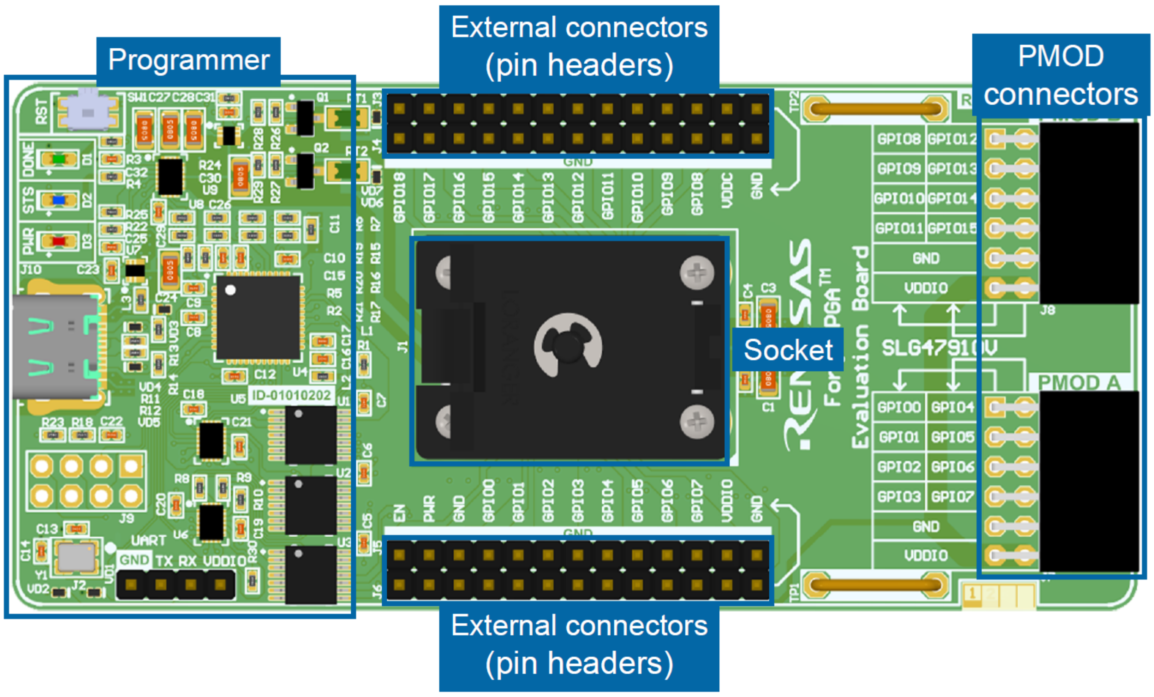

Programmer
External connectors (pin headers)
PMOD connectors
Socket
External connectors (pin headers)

Figure 45. ForgeFPGA Evaluation Board v2.0

The ForgeFPGA Evaluation Board v1.0 is a simplified version of the platform and therefore, has limited functionality. Since the socket is absent, the device is soldered directly on the board. The platform provides emulation possibilities along with access to the UART terminal.

For more information on the Evaluation Board, please see [7].

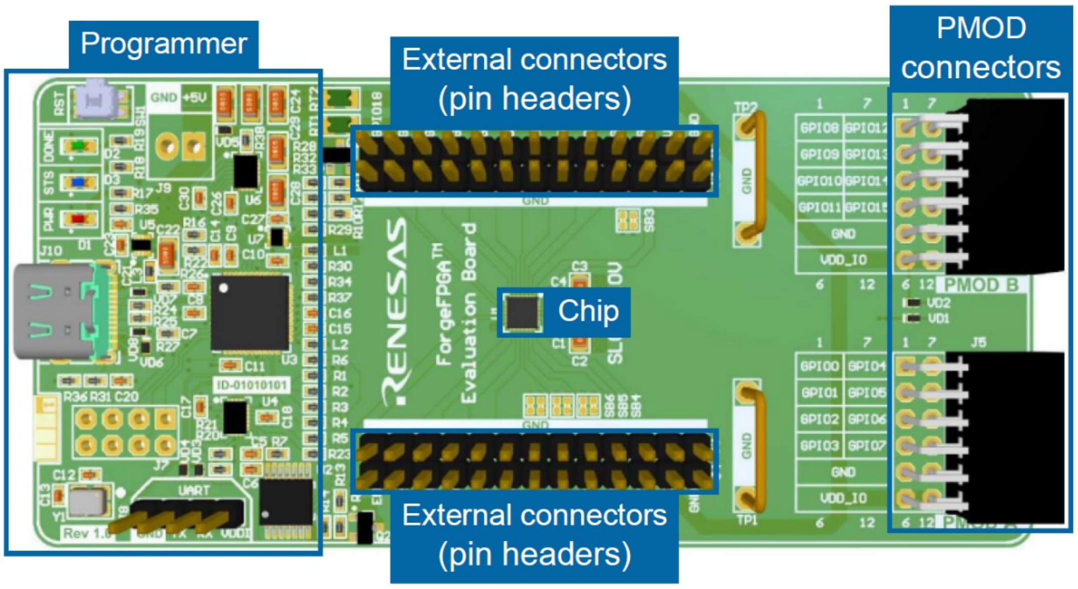

Programmer
External connectors (pin headers)
PMOD connectors
Chip
External connectors (pin headers)

Figure 46. ForgeFPGA Evaluation Board v1.0

Note: The ForgeFPGA Evaluation Board is compatible only with SLG47910.

GoConfigure Development Board

The Go Configure Development Board (see Figure 47) is an updated platform from Renesas, designed for assisting in the development process for Renesas products such as GreenPAK family mixed signal ICs and ForgeFPGA programmable arrays. The Development Board works in combination with the Go Configure Software Hub and provides rich functionality, such as:

- Flexible power management with software-configurable protection (OVP, OCP)
- Three programmable power sources and three fixed power sources
- Dual PCIe Connector
- 32-channel Digital Pattern Generators
- 16-channel Analog Waveform Generators (AWG)
- Onboard sequencer for Reading/Programming/Emulating Renesas products
- Up to 80 multi-functional configurable I/O lines (availability depending on the connected target board)

Working with target devices is done through interaction connectors and corresponding mating boards such as ForgeFPGA Socket Cards. All onboard signals, communication interfaces and power lines are exposed to connectors.

Development of ForgeFPGA products is possible through specialized Socket Cards. These cards are built for specific parts and packages and provide a minimum required set of external components for proper IC configuration and operation, as well as extra features such as power selector, clocks, I/Os exposed to PMOD connectors, and others.

For more information on the GoConfigure Development Board and for specific Socket Cards, see [9][10].

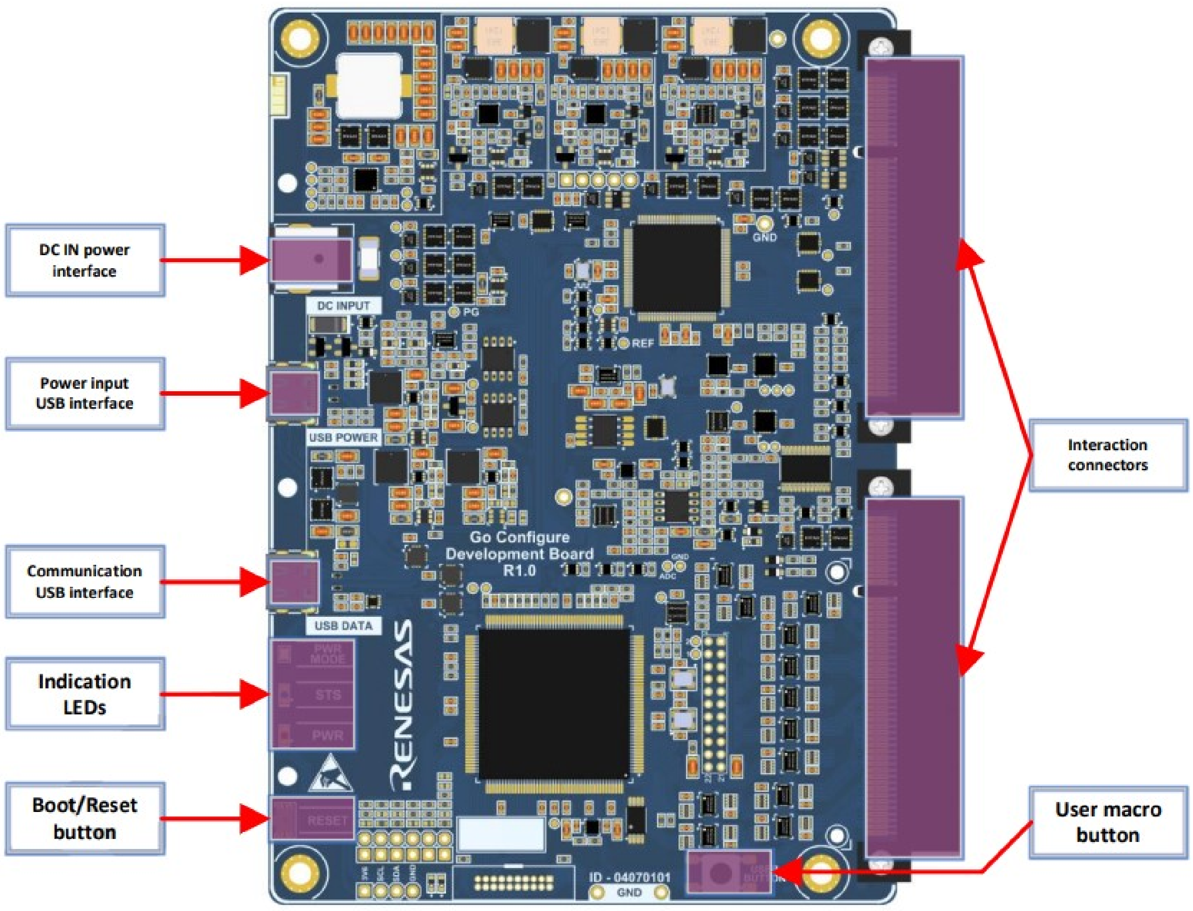
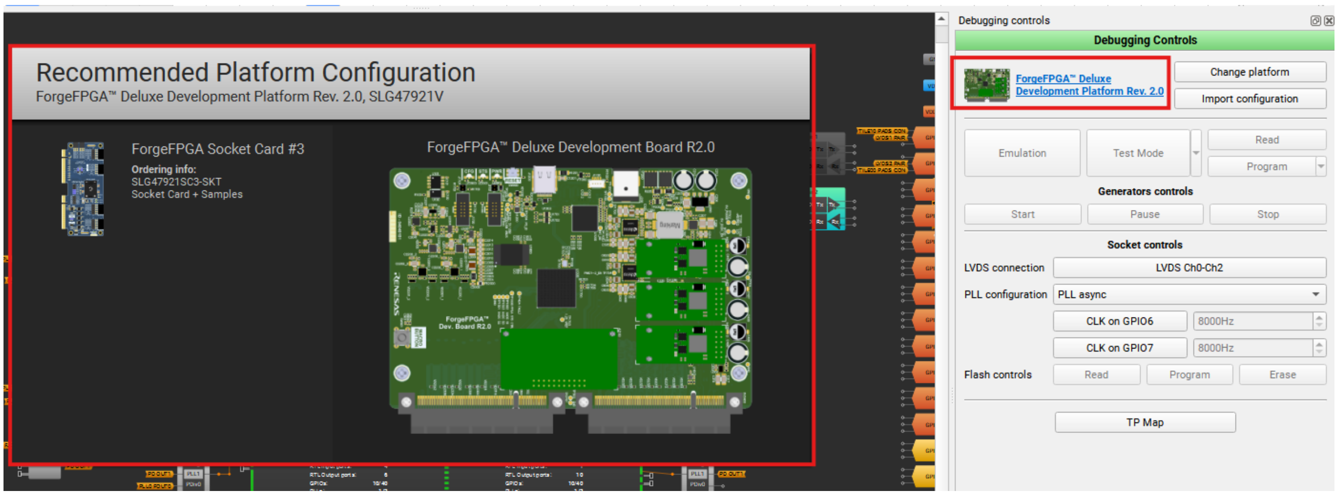


Figure 47. GoConfigure Development Board

3.2 Debugging Controls

Under the Debug tab on the main menu, the Recommended Platform Configuration Guide contains information about suitable sockets, adapters, and development boards, along with the device ordering information for each of the different devices. The guide is accessed by clicking on the platform's name in the Debugging controls panel (see Figure 48). Each part and board has different options in the Debugging Control panel.


Recommended Platform Configuration
ForgeFPGA™ Deluxe Development Platform Rev. 2.0, SLG47921V

ForgeFPGA Socket Card #3
Ordering info:
SLG47921SC3-SKT
Socket Card + Samples

ForgeFPGA™ Deluxe Development Board R2.0

LVDS connection: LVDS Ch0-Ch2
PLL configuration: PLL async
CLK on GPIO6: 8000Hz
CLK on GPIO7: 8000Hz

Figure 48. Platform Configuration Guide

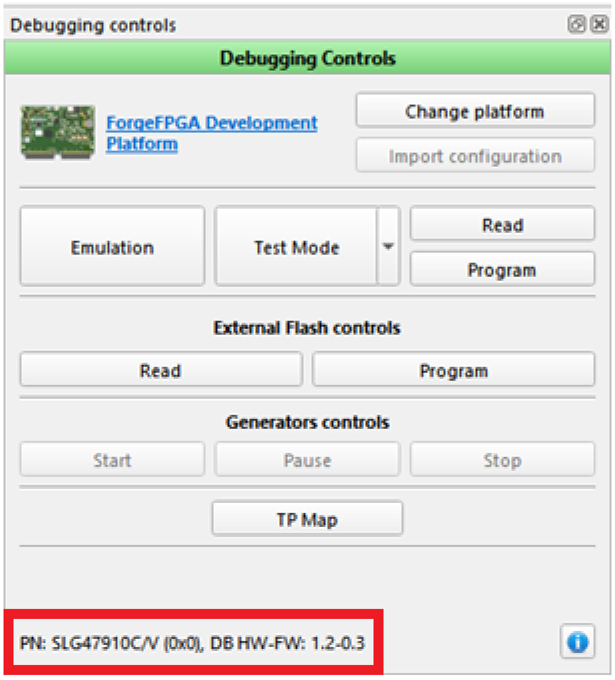
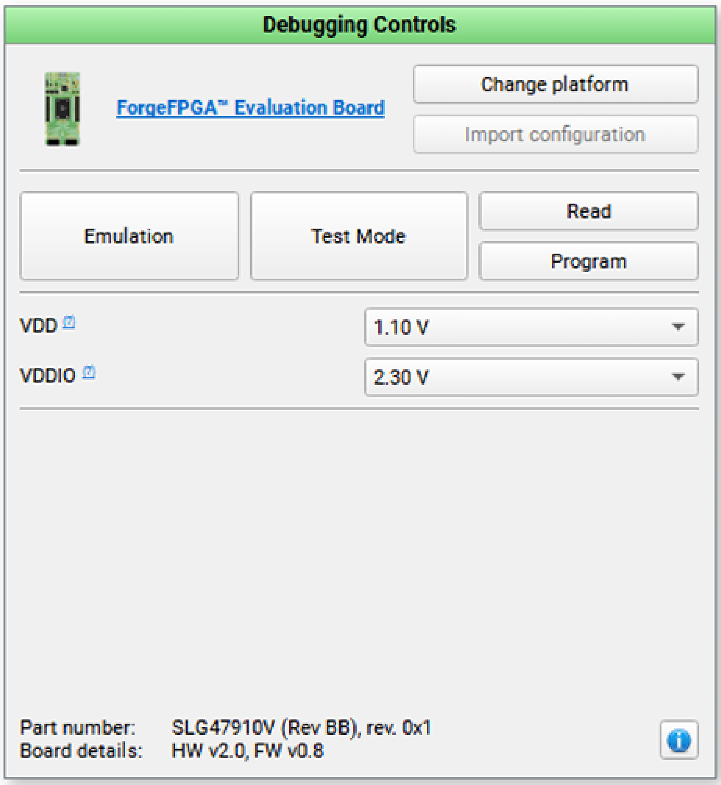

PN: SLG47910C/V (0x0), DB HW-FW: 1.2-0.3

Figure 49. Debugging Controls (SLG47910)



VDD: 1.10 V
VDDIO: 2.30 V

Part number: SLG47910V (Rev BB), rev. 0x1
Board details: HW v2.0, FW v0.8

Figure 50. ForgeFPGA Evaluation Board Debugging Controls

The Debugging Controls panel contains the UI controls for chip communication and programming, brief information about the connected device, and other features.

Device Configuration

- *Change platform* – Opens the list of development platforms that are supported
- *Import configuration* – Import's the configuration of test points from another platforms.

Chip Procedures

1. *Emulation* – The current project will be loaded to the chip (but not programmed) when the control is active and will be ready for test on the hardware board.
2. *Test Mode* – Test mode is used for connecting or disconnecting the chip's I/O pads to Test Point controls, configured by the user. Also, a user can check the programmed device using the test mode without emulation (OTP Mode).. Test mode can work without power on the chip. The user will control the power manually. Another feature of the Test Mode is that it can test with the conditions from Flash Memory by selecting Test Mode(*)
3. *Read* – Read the device using the hardware board or from the external flash memory.
4. *Program* – Program the device or the external device with the current project.

Flash Procedures

Flash memory is located on the FPGA Sockets. The controls are below:

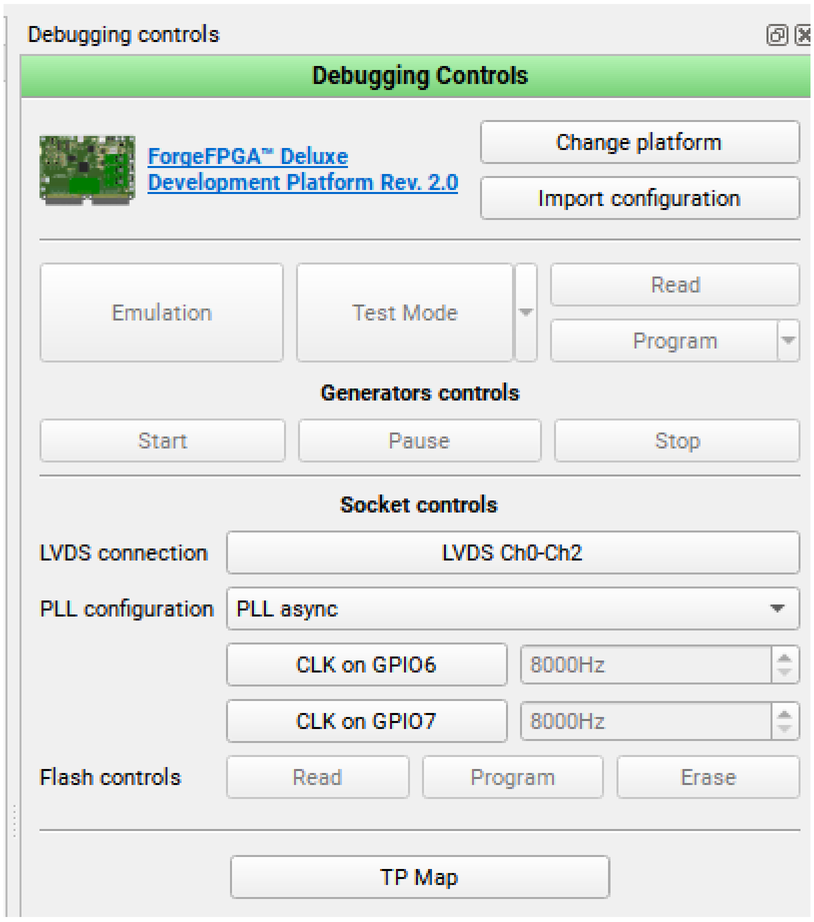
1. *Test Mode (*)* – Load data from flash to the device.
2. *Read* – read the chip project from the flash memory
3. *Program* – program a flash memory with the current project
4. *Erase* – clean flash memory (erased flash memory address values will be read as 0xFF)

Generator Controls

1. *Start/Pause/Stop* – During emulation, you can start all the test points together, pause them, or Stop them from running altogether.
2. *TP Map* – show the test point map on the work area to reflect the physical test points on the development platform. The Test Points of each pin will be shown next to their respective pin.

Socket Controls

1. *LVDS Connection* – the setting LVDS enables LVDS switches, configuring all channels simultaneously
2. *PLL Configuration* – the PLL generator operates in two modes: sync and async. In sync mode, both generators start simultaneously with a 90-degree phase shift at a set frequency. In async mode, each generator can be individually enabled or disabled and operates independently. Output is available on GPIO8 and GPIO9 (located under the socket as J21 and J22) in case of SLG47920/21.

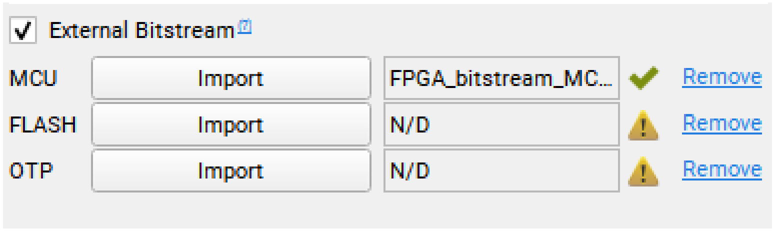


LVDS connection: LVDS Ch0-Ch2
PLL configuration: PLL async
CLK on GPIO6: 8000Hz
CLK on GPIO7: 8000Hz

Figure 51. Debugging Controls for SLG47920/21

External Bitstream

This option enables you to use the bitstream from another project for Emulation (MCU), Programming (OTP), and configuration from flash (FLASH).



MCU	Import	FPGA_bitstream_MC...	✓	Remove
FLASH	Import	N/D	⚠	Remove
OTP	Import	N/D	⚠	Remove

Figure 52. External Bitstream Option

Voltage level controls

VDD/ VDDIO – sets the voltage level on the corresponding TP. If a selected value exceeds any of the board limitations, the dependent controls are blocks and a warning is shown.

Information details

Brief information about the last detected chip (where e.g. SLG47910 C/V is part number and package name and (0x0) is a metal revision. DB HW-FW: 1.2-0.3 explains that the Development Board is detected with rev 1.2 of the Hardware and rev 1.11.4 for Firmware.

3.3 Debug Tools

The *Debug Tool* controls are used to configure input signals on the inputs of external devices. There are many ways in which we can manage the device input signals.

Note: Some debug functions are not available when using ForgeFPGA Evaluation Board for testing.

Use the context menu on GPIOs with a right click to see the connectivity options (Figure 53).

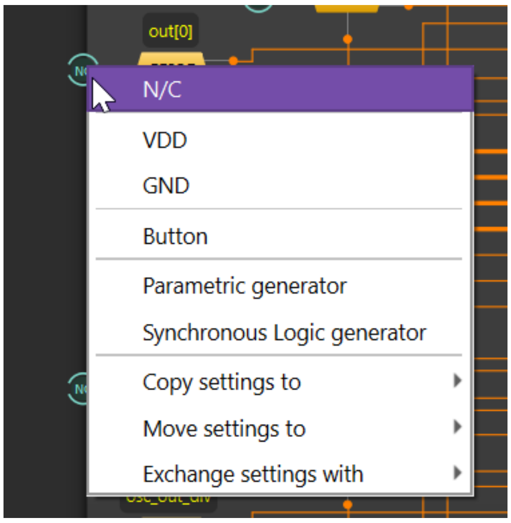


Figure 53. Debug Tool

- **NC (not connected)**

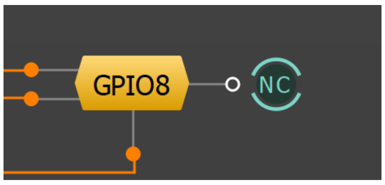


Figure 54. NC (not connected)

- **VDD**

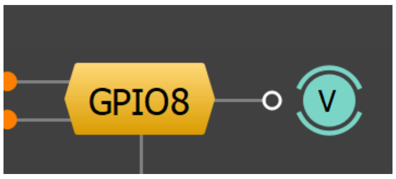


Figure 55. Set to VDD

- **GND**

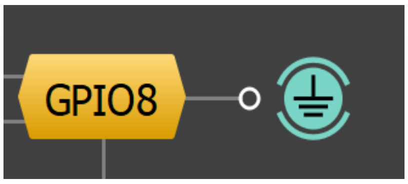


Figure 56. Set to GND

Configuration Button

The Configuration Button has three functions. The user can configure the button to establish the connection as VDD, GND or Hi-Z. If the user wants the GPIO to be a fixed connection to VDD, then the LATCH option should be selected. The LATCH option will make sure that the GPIO is connected to a particular signal (VDD/GND/Hi-Z). This will enable the button to be LATCHED to VDD unless it is changed (see Figure 57, Figure 58, and Figure 59).

The default connection option is VDD/GND, but it can be changed to Hi-Z by selecting Hi-Z option from the Upper Connection or the Lower Connection option as needed from the context menu. In Figure 58, you can see that the Upper Connection "U" corresponds to Hi-Z, and the Bottom Connection "B" corresponds to GND.

If the configuration is set as UNLATCHED and the default connection is set as Upper Connection "U" which equals to Hi-Z and the Bottom Connection "B" to GND, whenever the button is pressed, there will be toggle in the waveform between Hi-Z and GND at that very moment with the default waveform being at Hi-Z (see Figure 58).

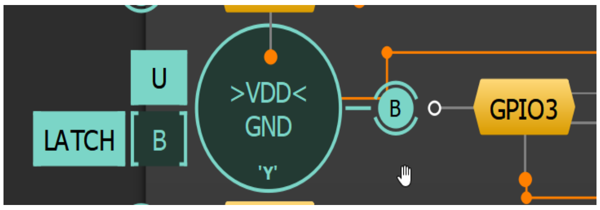


Figure 57. Latched Button with Upper Connection as VDD

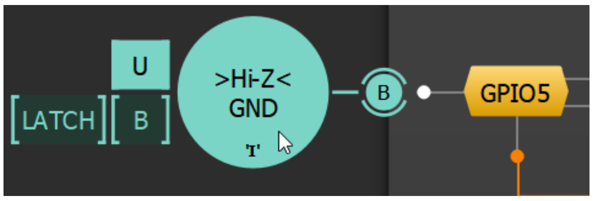


Figure 58. Unlatched Button with Upper Connection as Hi-Z

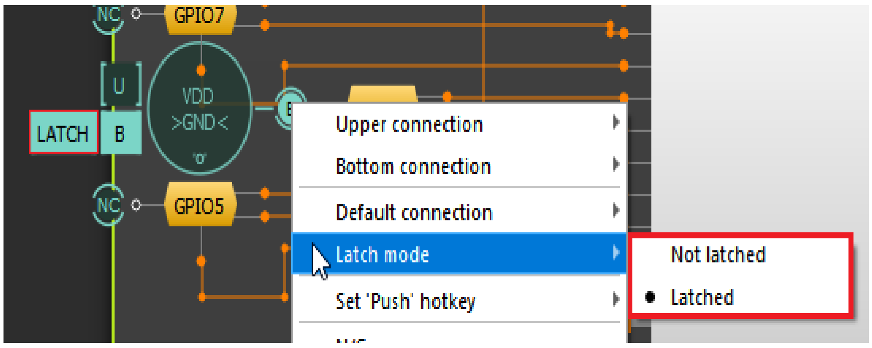


Figure 59. Context menu options for Configurable Button

Synchronous Logic Generator

Right click on the NC of the desired GPIO and from the context menu select the Synchronous Logic Generator option. The synchronous Logic Generator is used for generating the logic pulses and waveforms for each of the GPIOs. The 'Edit' Button allows the signal to be configured.

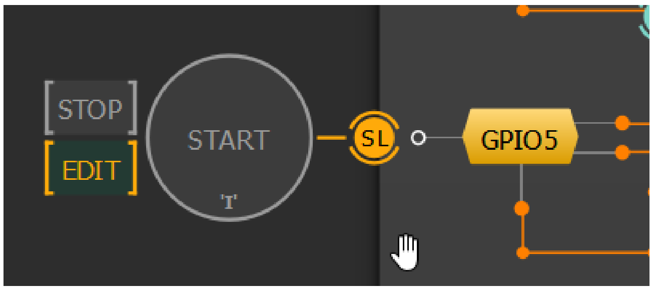
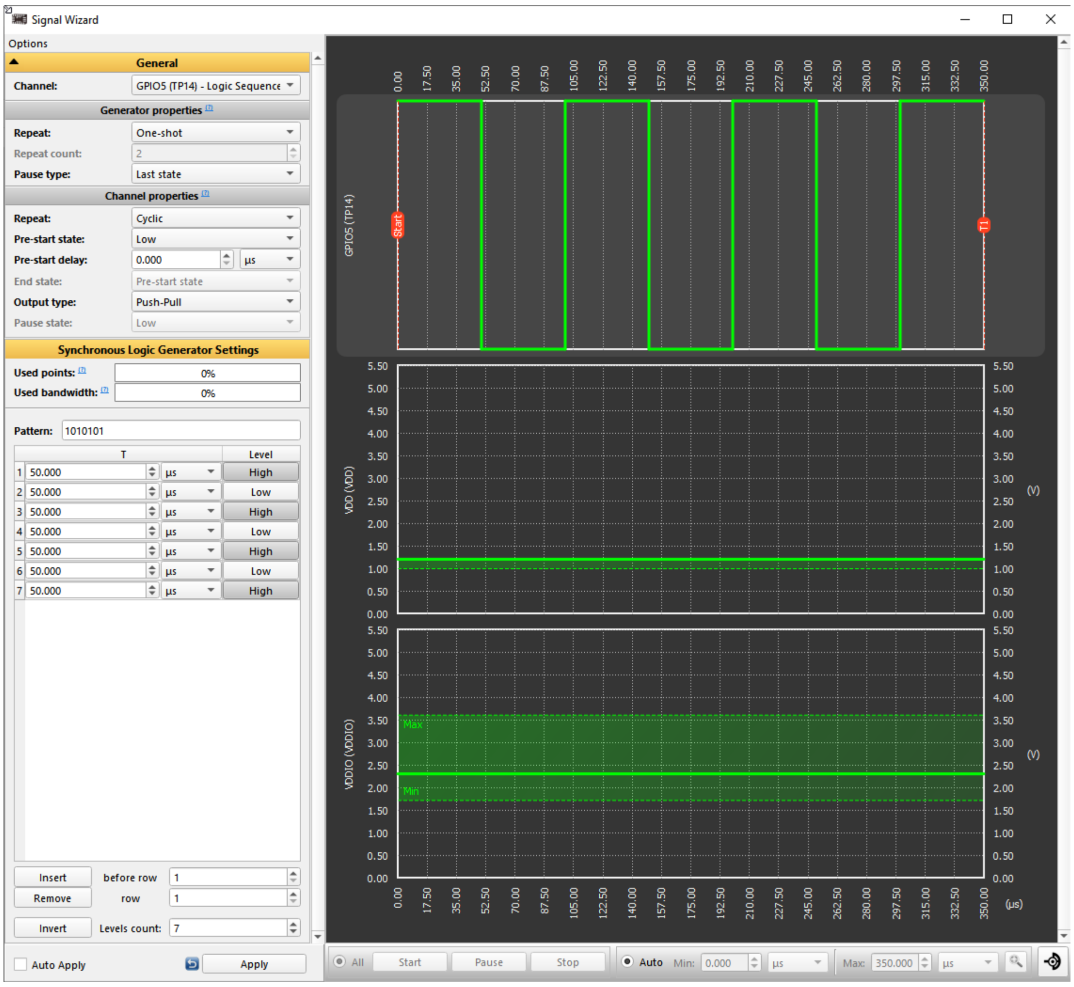


Figure 60. Synchronous Logic Generator


Signal Wizard

Options

General

Channel: GPIO5 (TP14) - Logic Sequence

Generator properties

Repeat: One-shot
Repeat count: 2
Pause type: Last state

Channel properties

Repeat: Cyclic
Pre-start state: Low
Pre-start delay: 0.000 µs
End state: Pre-start state
Output type: Push-Pull
Pause state: Low

Synchronous Logic Generator Settings

Used points: 0%
Used bandwidth: 0%

Pattern: 1010101

	T		Level
1	50.000	µs	High
2	50.000	µs	Low
3	50.000	µs	High
4	50.000	µs	Low
5	50.000	µs	High
6	50.000	µs	Low
7	50.000	µs	High

Insert before row 1
Remove row 1
Invert Levels count: 7

Auto Apply
Apply

Figure 61. Signal Wizard for Synchronous Logic Generator

Below are the different features in the Signal Wizard for Synchronous Logic Generator (Figure 61).

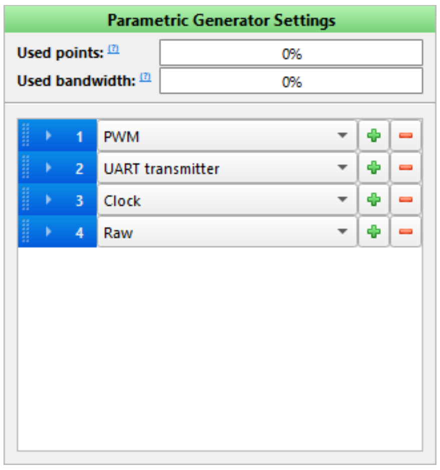
- *Used Points*: Amounts of points which are already used by patterns on all channels. Point indicates a moment when generator changes a state on at least on channel.
- *Used Bandwidth*: Percentage of used resources needed to successfully execute generator's pattern.
- *Pattern*: 0 = low, 1 = high level. We can set the pattern of pulse levels.
- *Repeat*: One Shot / Cyclic / Custom.
- *T / levels*: it sets the duration of each pulse.
- *Insert*: To insert pulse before selected position.
- *Remove*: remove pulse from the selected position.
- *Invert*: to invert the pattern from (For ex. ‘1010’ to ‘0101’).
- *Level Count*: to insert the total number of pulses to be generated.

Parametric Generator

The parametric Generator generates logic pulses that follow different protocol sequences. It can be selected from the context menu of all GPIOs.

In the signal Wizard, a special editor shows the sequence of commands. Actions available in the command editor:

- Add or remove commands by clicking + or –
- Change command parameters
- Change the order of commands by dragging the commands to another position



Parametric Generator Settings

Used points: 0%

Used bandwidth: 0%

1 PWM

2 UART transmitter

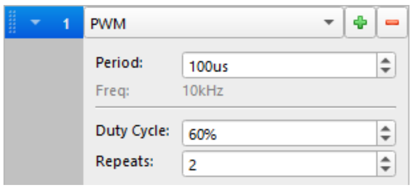
3 Clock

4 Raw

Figure 62. Parametric Generator Command Editor

The list of available commands:

- *PWM (Pulse Width Modulation)* – the PWM command editor has three input fields:
  - *Period* – a united duration of high and low states per repeat
  - *Duty cycle* – the percentage of the total duration in the high state
  - *Repeats* – pattern repeat count



1 PWM

Period: 100us

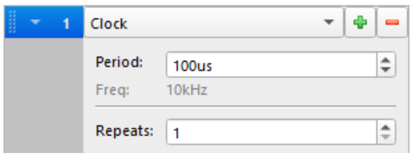
Freq: 10kHz

Duty Cycle: 60%

Repeats: 2

Figure 63. PWM Command Editor

- *Clock* – the command generates a signal oscillating between a high and a low state. The editor has two input fields:
  - *Period* – the united duration of high and low states per repeat
  - *Repeats* – pattern repeat count



1 Clock

Period: 100us

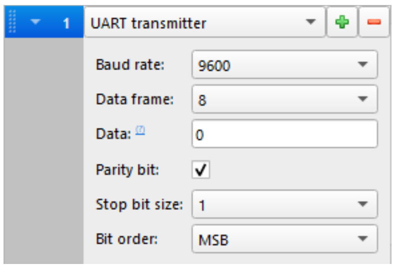
Freq: 10kHz

Repeats: 1

Figure 64. Clock Command Editor

- *UART Transmitter* – generates signal according to the UART standard:
  - *Baud rate* – signal's frequency
  - *Data frame* – number of bits allocated for user input

- *Data* – user input in hex format. In case the data frame is lower than the bits required to represent data, more significant bits are ignored
- *Parity bit* – create parity bit for error detection
- *Stop bit size* – duration of the stop bit
- *Bit order* – serial data transfer format



1 UART transmitter

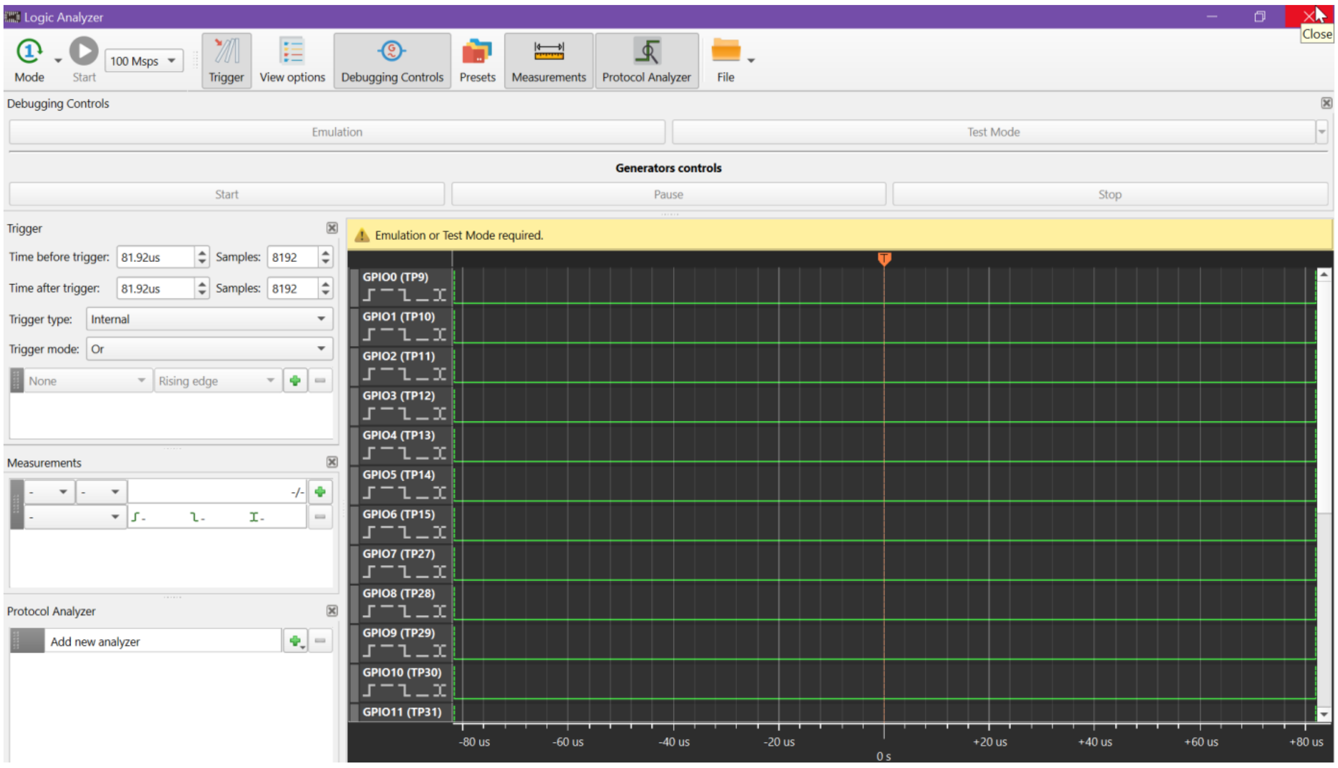
Baud rate: 9600
Data frame: 8
Data: 0
Parity bit: ✓
Stop bit size: 1
Bit order: MSB

Figure 65. Clock Command Editor

- *Raw* – this parent works as a typical Logic Generator

3.4 Logic Analyzer

The ForgeFPGA Workshop has a built-in Logic Analyzer which can be used to observe the waveforms during testing via the ForgeFPGA Deluxe Development Board. The *Logic Analyzer* tool allows you to capture multiple signals from a digital circuit. It has advanced triggering capabilities and a protocol decoder that helps to see the timing relationship between multiple signals and decode them.



100 Msps

Debugging Controls

Trigger
Time before trigger: 81.92us Samples: 8192
Time after trigger: 81.92us Samples: 8192
Trigger type: Internal
Trigger mode: Or
None Rising edge

Measurements

Protocol Analyzer
Add new analyzer

Emulation or Test Mode required.

Figure 66. Logic Analyzer

The characteristics of the Logic Analyzer are the following:

- Frequency range – 500 Hz to 200 MHz
- Buffer Size – 16384 samples
- SPI, I2C, Parallel and UART protocol support

Operational Controls

- *Start* – Launch the Logic Analyzer. The Start button becomes active after Emulation or Test Mode is started.
- *Sampling rate* – Drop down selector for the signal sampling frequency.

Mode

There are three operating modes:

- *Auto mode* – Trigger events are ignored. The signal on the pins is shown as a continuous waveform.
- *Single shot* – Refreshes the waveform pattern once a trigger event is detected.
- *Normal mode* – Refreshes the waveform pattern each time when a trigger condition is met.

Triggers

Set time parameters to choose the trigger time position or a specific sample number.

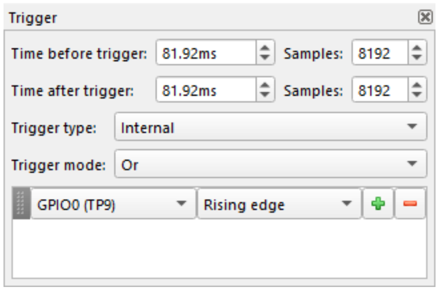

Trigger
Time before trigger: 81.92ms Samples: 8192
Time after trigger: 81.92ms Samples: 8192
Trigger type: Internal
Trigger mode: Or
GPIO0 (TP9) Rising edge

Figure 67. Trigger Parameters

The trigger type can also be configured (see Figure 67):

- *Hardware button* – the trigger is activated by pressing a physical button on the board
- *Internal* – the trigger is activated when the trigger conditions, which is set in the trigger list is met.

The internal trigger can also be configured with the following options:

- *Or* – any of the trigger conditions added to the trigger list are met
- *And* – all trigger conditions added to the trigger list are met

An internal trigger must have the following settings specified:

- *Channel* – assign the trigger to the specified channel
- *Condition* – rising edge, falling edge, both edges, and high and low state

Note: set proper trigger conditions for successful sampling (e.g. Rising edge and Falling edge together with the trigger mode may result in improper trigger work).

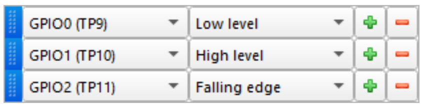

GPIO0 (TP9) Low level
GPIO1 (TP10) High level
GPIO2 (TP11) Falling edge

Figure 68. Trigger Conditions

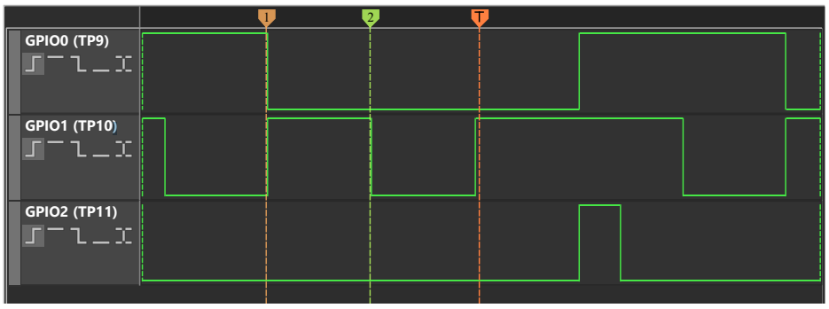


Figure 69. Trigger Configuration

Debugging Controls

The Debugging Control panel is responsible for Emulation, Test Mode, and Generator control functionality.

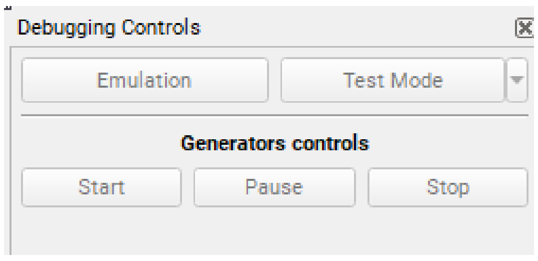


Figure 70. Debugging Controls

The channels in the View options panel can be shown or hidden using the buttons at the bottom.

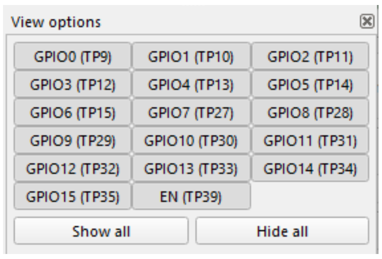


Figure 71. View Options

Presets

The Logic Analyzer configurations can be stored in presets and restored from a previous configuration when needed.

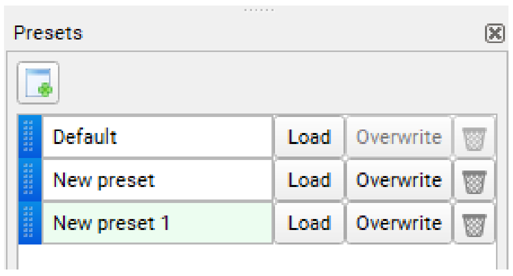


Figure 72. Logic Analyzer Configuration Presets

- *New Preset* – Create a new preset for the current Logic Analyzer configuration
- *Load* – load a selected preset configuration
- *Overwrite* – overwrite preset with the current Logic Analyzer configuration
- *Delete Preset* – remove the selected preset
- *Default* – load the default preset of the Logic Analyzer window
- *Autosave [time]* – the modified preset is saved each time the Logic Analyzer window, the Debug tool, or ForgeFPGA workshop is closed.

Protocol Analyzer

The *Protocol Analyzer* decodes data according to the selected protocol.

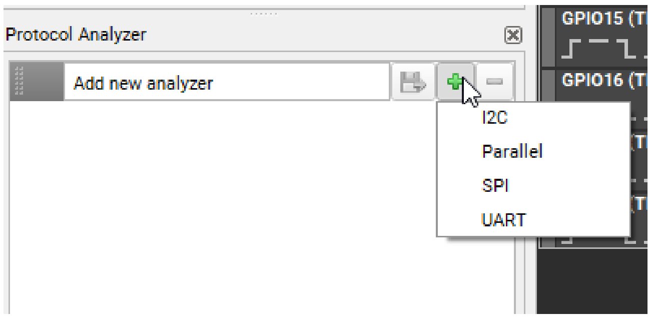
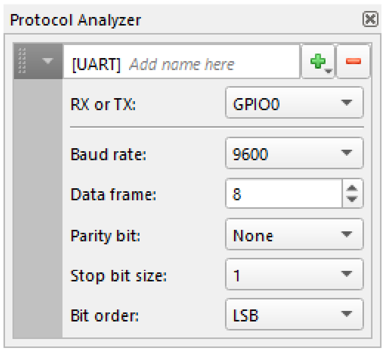


Figure 73. Protocol Analyzer Decode Options

To analyze captured data, click the + button and select one of the protocols.


Protocol Analyzer

[UART] *Add name here*

RX or TX:	GPIO0
Baud rate:	9600
Data frame:	8
Parity bit:	None
Stop bit size:	1
Bit order:	LSB

Figure 74. Protocol Analyzer Options

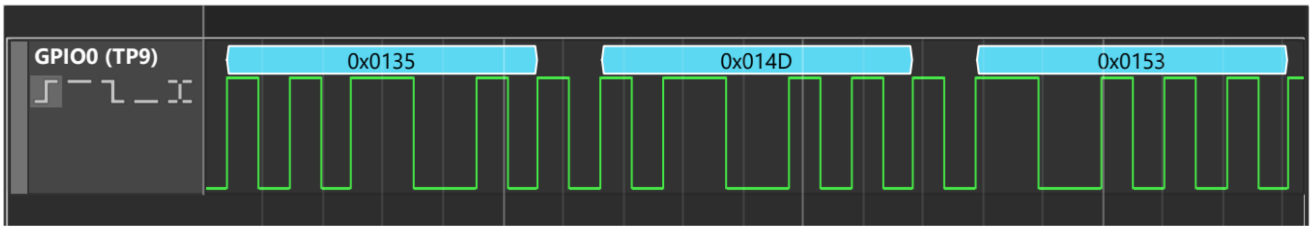


Figure 75. Decoded Data

Then, choose a channel for analysis and modify the protocol settings if necessary. The decoded data will be displayed above the corresponding plot.

Data can be easily exported from the Protocol Analyzer by clicking the Save button next to the Protocol Analyzer name field. If the parameters are selected correctly, a CSV file of the data will be saved.

Import/Export

The waveform data can be imported or exported in CSV format. These options are grouped under the File Menu on the top toolbar.

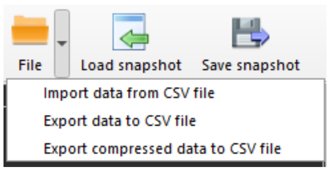


Figure 76. Import/Export from/to CSV Format

Plot Widget

The plot widget displays the waveforms in the Logic Analyzer window. The way a plot is shown can be changed via the plot context menu. Right-click on a plot area to add a marker, show or hide the time scale, change the plot height, and select the cursor width.

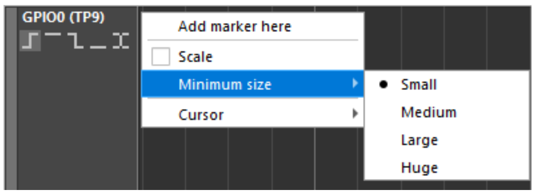


Figure 77. Plot Widget

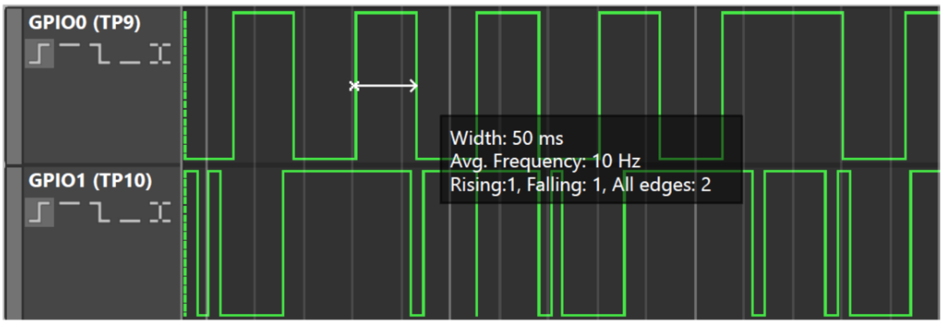
The plot area can be navigated using the following controls:

- Move left/right by left click + drag left/right
- Scroll up/down by mouse wheel up/down
- Zoom in/out by CTRL + mouse wheel up/down
- Quick zoom in/out with the middle mouse button click + drag up/down
- Reorder waveforms by Drag and Drop

Cursors

A cursor is a measurement tool for calculating waveform data between the edge of one or more plots. To see the cursor, hover the mouse over a measured section of a waveform.

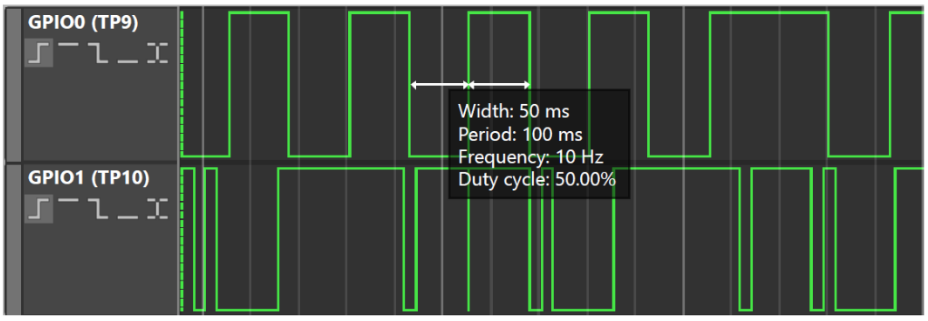
A *Half Period* cursor measures the distance between the two nearest edges at a hovered section of a waveform.



Width: 50 ms
Avg. Frequency: 10 Hz
Rising:1, Falling: 1, All edges: 2

Figure 78. Half Period Cursor

A Period cursor measures the width of the hovered half period, the duration of a full period, and calculates the frequency and what fraction of the full period the hovered area of the period is, which is the Duty Cycle.



Width: 50 ms
Period: 100 ms
Frequency: 10 Hz
Duty cycle: 50.00%

Figure 79. Period Cursor

To change the cursor width, open the context menu and select either the Period or Half Period option.

To measure data at a distance that exceeds one period, left click the edge you want to measure from and hover over the edge you want to measure to. This will give you the total duration of the measured section, the calculated average frequency, and the quantity of rising and falling edges well as their sum.

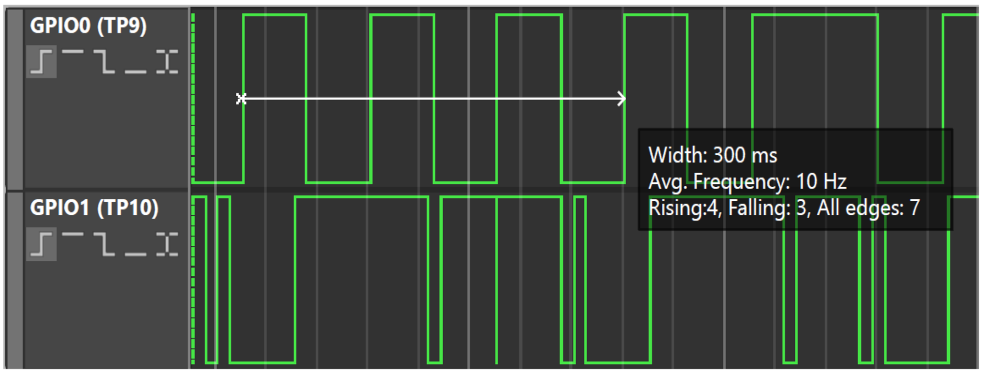


Figure 80. Adjustable Period Cursor for Measurement

You can also measure the width between the edges on the distinct waveforms.

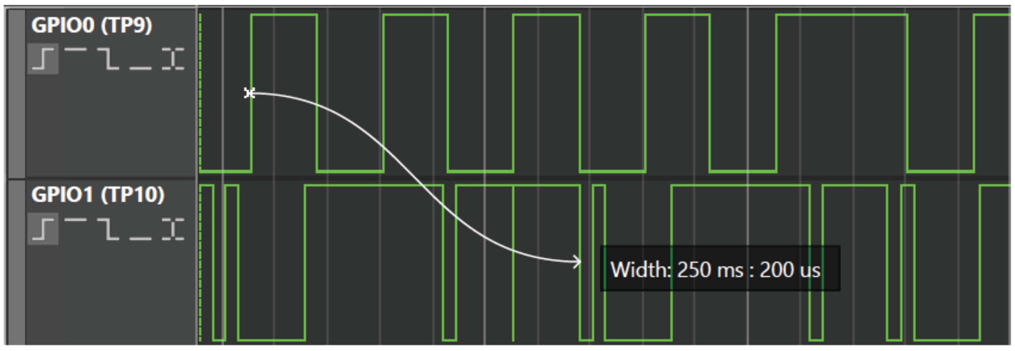


Figure 81. Adjustable Period Cursor Measurement between waveforms

Markers

You can do the following actions with markers:

- Add a new marker with a CTRL + left click on the markers panel
- Set a new marker from the plot's context menu
- Remove the marker with a CTRL + right-click on the marker head
- Move the marker by a left-click + drag the mouse cursor

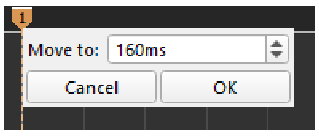


Figure 82. Moving markers with context menu

- Move from the context menu by a left-click on the marker's head
- Select the marker by clicking on the marker head, the selected marker has a white line on top
- Move the selected marker to the closest visible left/right line by CTRL + LEFT/RIGHT arrow key
- Move the selected marker left/right by one sample with CTRL + SHIFT + LEFT/RIGHT arrow key

3.4.1.1 Marker Measurements

- *Measurements* – Period and frequency. The frequency value is rounded to four decimals.
- *Additional measurements* – counts the rising edges, falling edges, and all edges in the period between the markers

To calculate all measurements between two markers, select the markers and a channel in combination boxes.

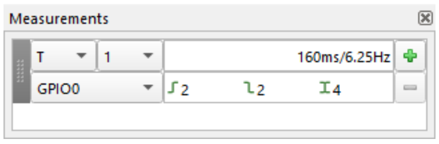


Figure 83. Marker Measurements

3.5 UART Terminal

The UART Terminal is a console tool used for serial communication between the board and an external device. While developing a project, the UART terminal helps to *read* and *write* via the UART protocol. This UART terminal is available only on the ForgeFPGA Evaluation Board.

To start working with the terminal, ensure that the Evaluation Board platform is selected. Open the UART Terminal tool on the top toolbar.

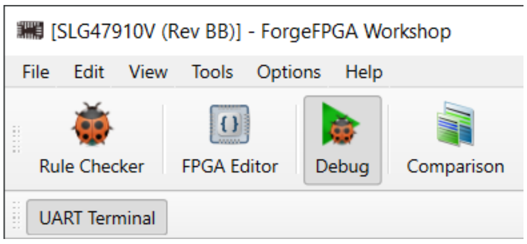
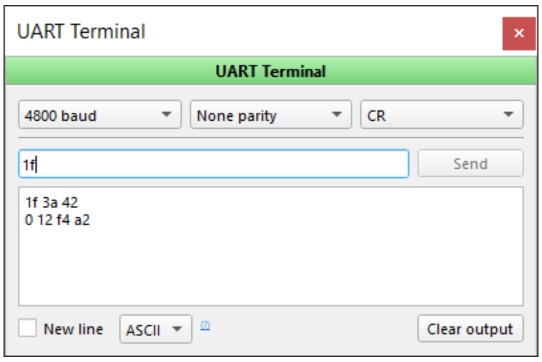


Figure 84. UART Terminal Tool



```
1f
```

```
1f 3a 42
0 12 f4 a2
```

Figure 85. UART Terminal Window

The following fields are available under UART Terminal for edit -

- *Baud rate* – Selects the baud rate from the list for read and write operations.
- *Parity control* – Selects whether and how the parity control bit is used.
- *Line ending* – Selects which character is used as a new line character. No line ending, new line (NL), Carriage Return (CR), or both New Line & Carriage Return. This option is available in ASCII mode only.
- *New line* – Add a new line after each byte sent if set; otherwise, send the entire message at once.
- *Data format* – Selects the data format: ASCII or Hex. For Hex, the input case is independent, and numbers are separated by a space.

The input field supports copy/paste operations. The output field only supports copying of the received data.

3.6 DAC Tool

The DAC Tool allows you to configure the voltage level on special analog output socket pins.

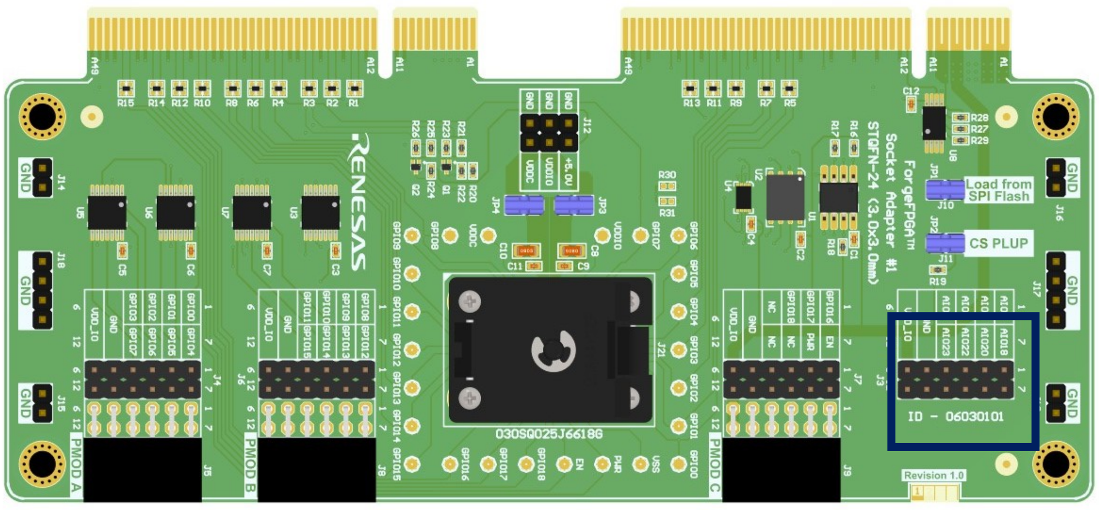


Figure 86. Analog I/O Pins on ForgeFPGA Socket Adapter

Start by selecting the appropriate development platform and opening the DAC Tool from the toolbar (see Figure 87).

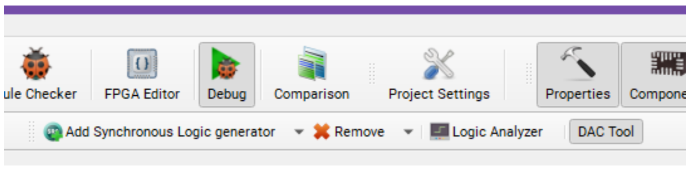
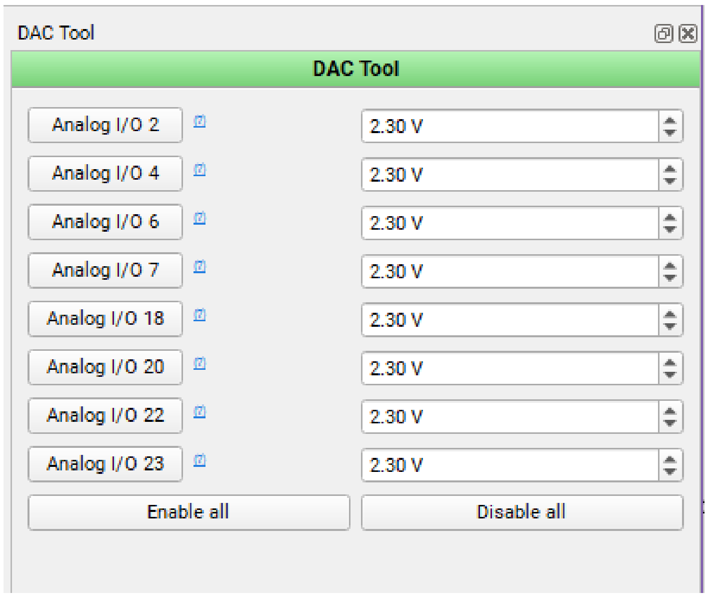


Figure 87. DAC Tool on the Toolbar

All the analog I/O pins are displayed by default. Use an “Analog I/O n” button to enable a respective output pin and use a spin box at the right to set a desired voltage level on it (see Figure 88).



Analog I/O 2	2.30 V
Analog I/O 4	2.30 V
Analog I/O 6	2.30 V
Analog I/O 7	2.30 V
Analog I/O 18	2.30 V
Analog I/O 20	2.30 V
Analog I/O 22	2.30 V
Analog I/O 23	2.30 V

Figure 88. DAC Tool Settings

Note: *Setting a voltage level on analog I/O pins is enabled only if Emulation or Test Mode is activated.*

3.7 HBRAM OTP Data Editor

HBRAM OTP Data Editor allows you to modify the bit values of BRAM and User OTP Macrocell registers. Open the tool from the toolbar, or BRAM/User OTP Macrocell Properties panel.

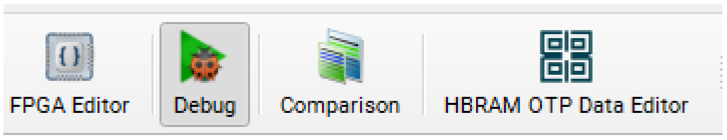
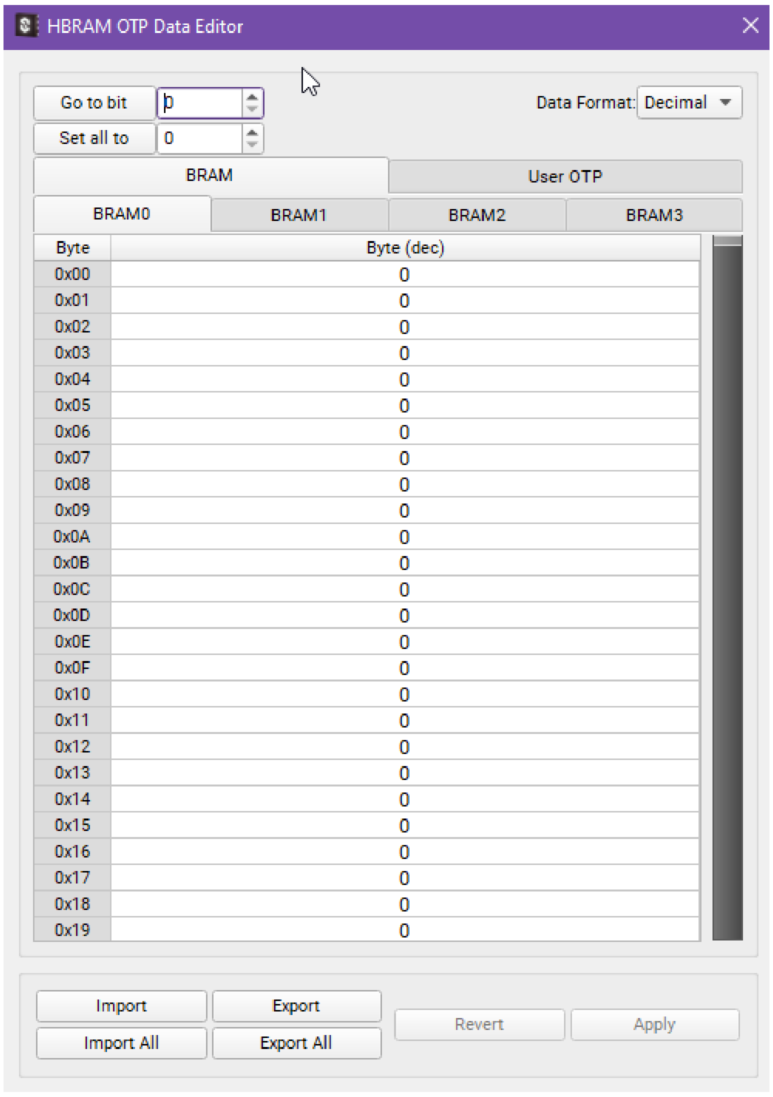


Figure 89. HBRAM OTP Data Editor on the Toolbar

The tool provides separate tables for configuring BRAM and User OTP Macrocell bits. Choose whether to represent the data in decimal or hexadecimal format.



Byte	Byte (dec)
0x00	0
0x01	0
0x02	0
0x03	0
0x04	0
0x05	0
0x06	0
0x07	0
0x08	0
0x09	0
0x0A	0
0x0B	0
0x0C	0
0x0D	0
0x0E	0
0x0F	0
0x10	0
0x11	0
0x12	0
0x13	0
0x14	0
0x15	0
0x16	0
0x17	0
0x18	0
0x19	0

Figure 90. HBRAM OTP Data Editor (BRAM tab)

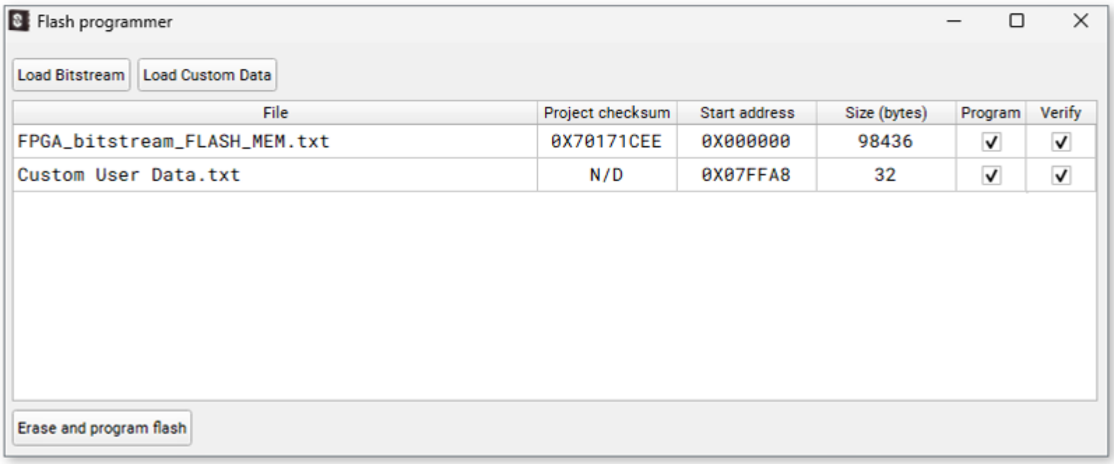
Use the upper controls to navigate through the table with the *Go to bit* button. You can also set a specified value for all bits in the active tab using the *Set all to* button.

The bottom controls provide an option to import or export the data either from the active tab using the Import/Export buttons or from all tabs with Import all/Export all buttons (the latter options are applicable for BRAM only).

3.8 Flash Programmer

The *Flash programmer tool* allows flexible programming of the flash memory using its full addressing range, enabling the programming of both the bitstream and custom user data.

To start working with the tool, choose data via the *Load Bitstream* or *Load Custom Data* buttons in the upper section:



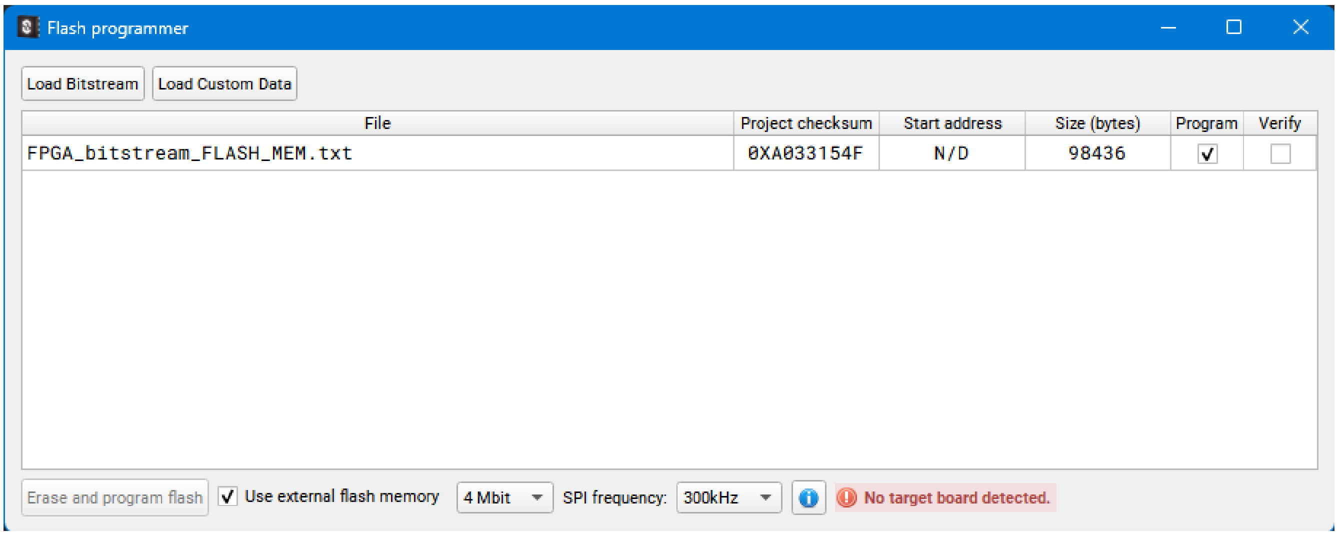
File	Project checksum	Start address	Size (bytes)	Program	Verify
FPGA_bitstream_FLASH_MEM.txt	0X70171CEE	0X000000	98436	✓	✓
Custom User Data.txt	N/D	0X07FFA8	32	✓	✓

Figure 91. Flash Programmer Tool

- *Load Bitstream* — upon adding a file, the tool validates file size against the expected bitstream size and calculates the checksum
- *Load Custom Data* — loads user-provided data, which should be in the same format as the data in the bitstream file

To program the flash, select one or multiple files using checkboxes in the Program column.

The Flash programmer tool supports programming of external flash memory. To enable this feature, tick the corresponding checkbox in the footer. Click the info icon to check the instruction for the flash parameter configuration. Specify the external flash size and set the SPI frequency to match your device specifications (see Figure 92).



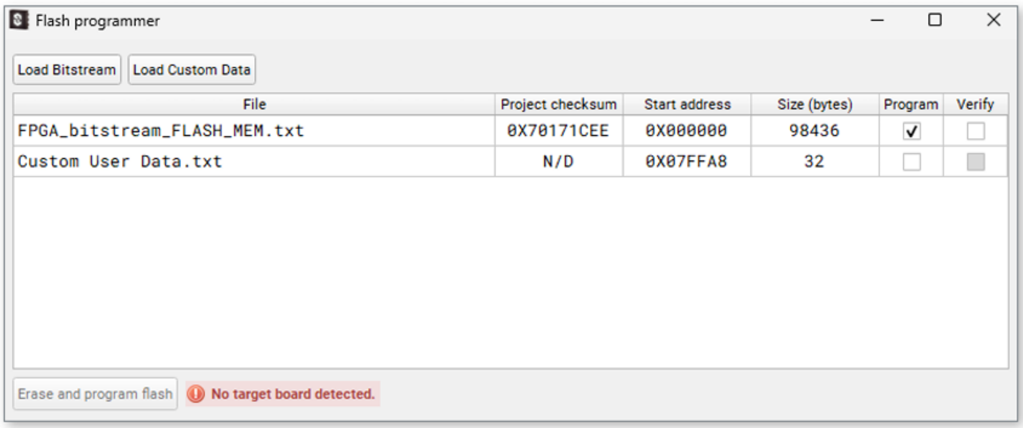
File	Project checksum	Start address	Size (bytes)	Program	Verify
FPGA_bitstream_FLASH_MEM.txt	0XA033154F	N/D	98436	✓	☐

✓ Use external flash memory 4 Mbit SPI frequency: 300kHz

No target board detected.

Figure 92. External Flash Programming Option

Observe the warnings triggered (at the bottom right) in case any inconsistencies or other issues occur during interaction with the tool. Only selected files pass the validation.



File	Project checksum	Start address	Size (bytes)	Program	Verify
FPGA_bitstream_FLASH_MEM.txt	0X70171CEE	0X000000	98436	✓	☐
Custom User Data.txt	N/D	0X07FFA8	32	☐	☐

Figure 93. Flash Programmer Tool Info Hint Example

Optionally, enable the *Verify* column. When activated, the *Flash programmer* reads the programmed range defined by the Start address and Size columns and compares it with the loaded file data.

Right-click the file name to access options to remove or replace the file via the context menu. Hover over the file to see its location on your computer.

Note: The entire flash memory is erased before programming

4. NVM Viewer

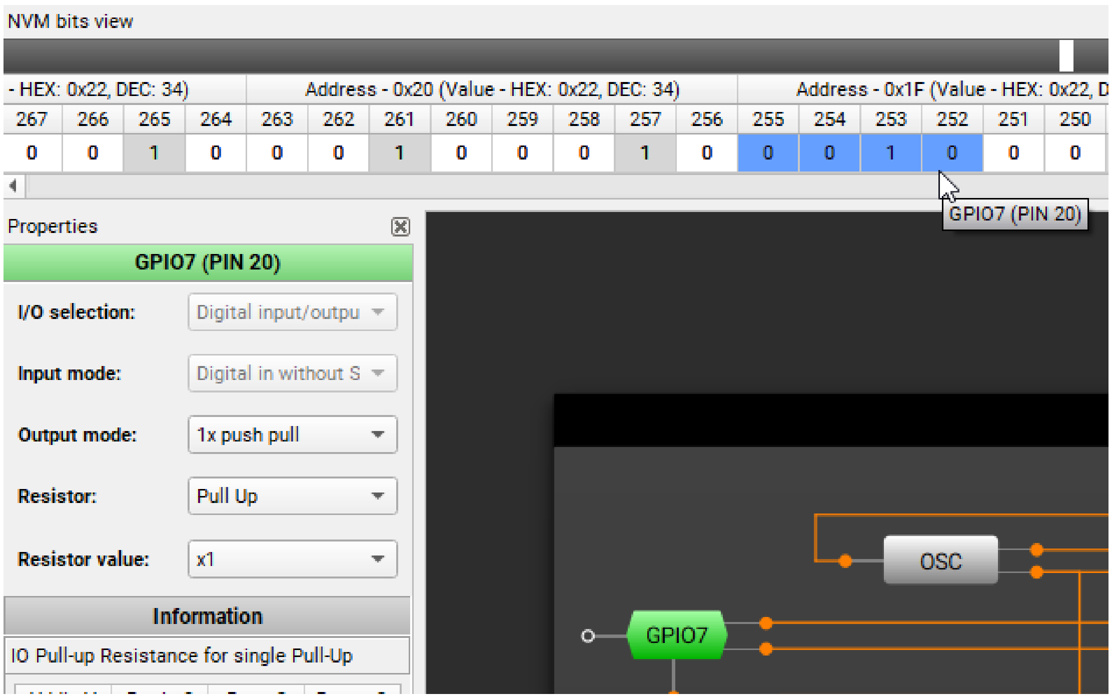
In the ForgeFPGA Workshop the user can view all bits that correspond to each function in the NVM Viewer. 1s and 0s in the NVM Viewer indicate whether a particular function is enabled or disabled.

The NVM Viewer can be viewed by clicking the appropriate button on the toolbar. The NVM Viewer is divided into byte size addresses. In total the number of bits range from 0 to 479.

When a block property is changed, the corresponding bit of that parameter changes and it is represented by a change of color.

NVM viewer cells can exist in the following states:

🟦 0	Bit range of a selected block
⬜ 0	Bits with 0 value
▨ 1	Bits with 1 value
🟥 0	Latest bit changed to 0 value
🟩 1	Latest bit change to 1 value



NVM bits view

- HEX: 0x22, DEC: 34)				Address - 0x20 (Value - HEX: 0x22, DEC: 34)								Address - 0x1F (Value - HEX: 0x22, D					
267	266	265	264	263	262	261	260	259	258	257	256	255	254	253	252	251	250
0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0

GPIO7 (PIN 20)

Properties

GPIO7 (PIN 20)

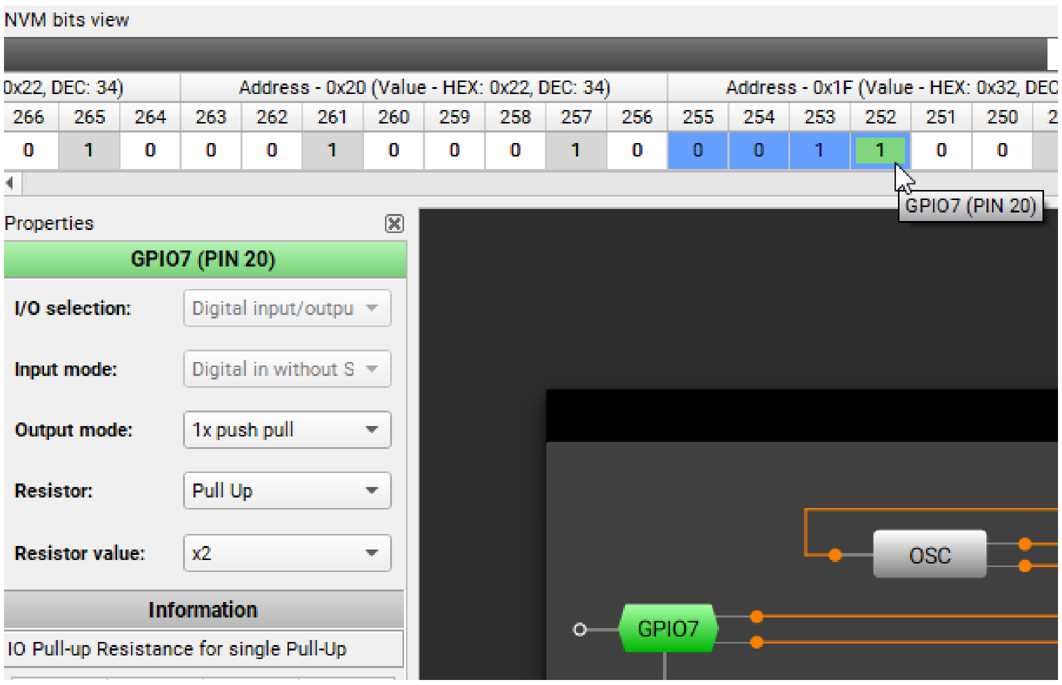
I/O selection:	Digital input/outpu
Input mode:	Digital in without S
Output mode:	1x push pull
Resistor:	Pull Up
Resistor value:	x1

Information

IO Pull-up Resistance for single Pull-Up

Figure 94. NVM Bits for GPIO7

If the resistor value is changed from x1 to x2, this results in the change of bits for GPIO7 from 0010 to 0011 (see Figure 94).



NVM bits view

0x22, DEC: 34)			Address - 0x20 (Value - HEX: 0x22, DEC: 34)								Address - 0x1F (Value - HEX: 0x32, DEC						
266	265	264	263	262	261	260	259	258	257	256	255	254	253	252	251	250	2
0	1	0	0	0	1	0	0	0	1	0	0	0	1	1	0	0	

GPIO7 (PIN 20)

Properties

GPIO7 (PIN 20)

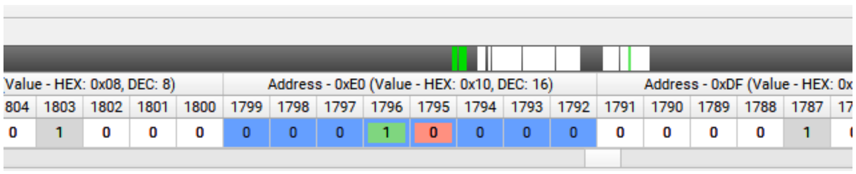
I/O selection:	Digital input/outpu
Input mode:	Digital in without S
Output mode:	1x push pull
Resistor:	Pull Up
Resistor value:	x2

Information

IO Pull-up Resistance for single Pull-Up

Figure 95. NVM Bits 0011 After Property Has Been Modified

The NVM viewer navigation bar can also identify the location of either selected or changed bits in the table (the color on the bar corresponds the color of the cell described above). In addition, a gray color represents 0 values while white represents 1 value (see Figure 95).



Value - HEX: 0x08, DEC: 8)					Address - 0xE0 (Value - HEX: 0x10, DEC: 16)								Address - 0xDF (Value - HEX: 0x					
804	1803	1802	1801	1800	1799	1798	1797	1796	1795	1794	1793	1792	1791	1790	1789	1788	1787	17
0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	(

Figure 96. NVM Viewer Top Bar

The user can hover over any bit in the NVM to check which functionality is represented by those bits. A detailed representation of each bit in NVM can be found in the SLG47910's Datasheet in the Appendix.

5. Revision History

Revision	Date	Description
1.00	May 20, 2024	Initial release.
2.00	Oct 22, 2024	Added new sections on • Import/Export RTL Files • Design Template • Synthesis Report • DAC Tool Updated PLL Calculator Updated Project Directory Folder Changed Advanced Dev Board name to Deluxe Dev Board Updated Block Properties
2.01	May 03, 2025	Updated information in sections • Settings • I/O Planner • Timing Analysis • Project Directory Folder • NVM Viewer • PLL Configurator • Rearranged index • Debugging Control Added new section – HBRAM OTP Data Editor Removed Block Properties Section
2.02	Dec 01. 2025	Deleted section on I/O Planner signals Updated information in sections • Commands for Timing Analysis • Import/export file • Macrocell Editor • Debugging Controls Added new section • GoConfigure Development Board • Flash Programmer • Placement Constraints • Full Chip Simulation • Software Support

A. Appendix: Warnings

Below is a list of warnings & its respective meaning that the Yosys displays in the Logger section of the software after Synthesis and Generate Bitstream Process

The network is combinational (run "fraig" or "fraig_sweep").

>> This is an error generated when handling a purely combinational input. It is produced by the optimization tool and can be simply ignored.

Bit <N> of cell port <port> driven by <D> will be left unconnected in EDIF output.

>> Port is driven by an undefined value. Connect the port to a proper driver.

Exporting x-bit on <N> as zero bit.

>>A port is in an undefined state. Make sure it is driven by a proper signal.

Cell <M> is an unmapped internal cell of type <T>.

>>After synthesis you are left with unmapped cells, make sure you use valid synthesizable Verilog.

Drivers conflicting with a constant <N> driver: <K>

>> A port was driven by both a constant value and a signal. Please correct your code.

Multiple conflicting drivers for <N>: <K>

>>Several wires were defined to drive a port. Please correct your code.

Wire <N> is used but has no driver.

>>No driver was assigned to a wire.

Wire <N> has 'init' attribute and is not driven by an FF cell.

>>A wire is set to be initialized to a certain value but is not driven by an FF.

I/O pin name not found! <N>

>> A port name was specified in the I/O Planner, but the match for it wasn't found in the synthesized netlist. Please make sure the names of the ports in the I/O Planner correspond to your design.

Input IOB port specification differs from the resulting one. Please check if all IOB ports were mapped properly.

>> Input and output I/O Planner specifications differ. Check if you've mapped the ports of your design correctly.