ForgeFPGA Workshop User Guide v6.45

This user guide will help you navigate the ForgeFPGA Workshop software and understand the different 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] 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] SLG47910, 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

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 HDL
- 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:

1. Download and install Go Configure Software Hub from https://www.renesas.com/us/en/software-tool/go-configure-software-hub

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

   - *Welcome* — 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 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 with various Integrated Circuits (ICs) and a preconfigured circuit project, where you can make customized changes
2. Select the SLG47910 (rev BB) Part Number & open it
3. Specify the Project Settings like V_{DD}, V_{DDIO} and Temperature
4. Write the HDL Code in the HDL Editor Window
5. Test the design with the Debug Tool, using the Simulation feature or any of the supported hardware development platforms.

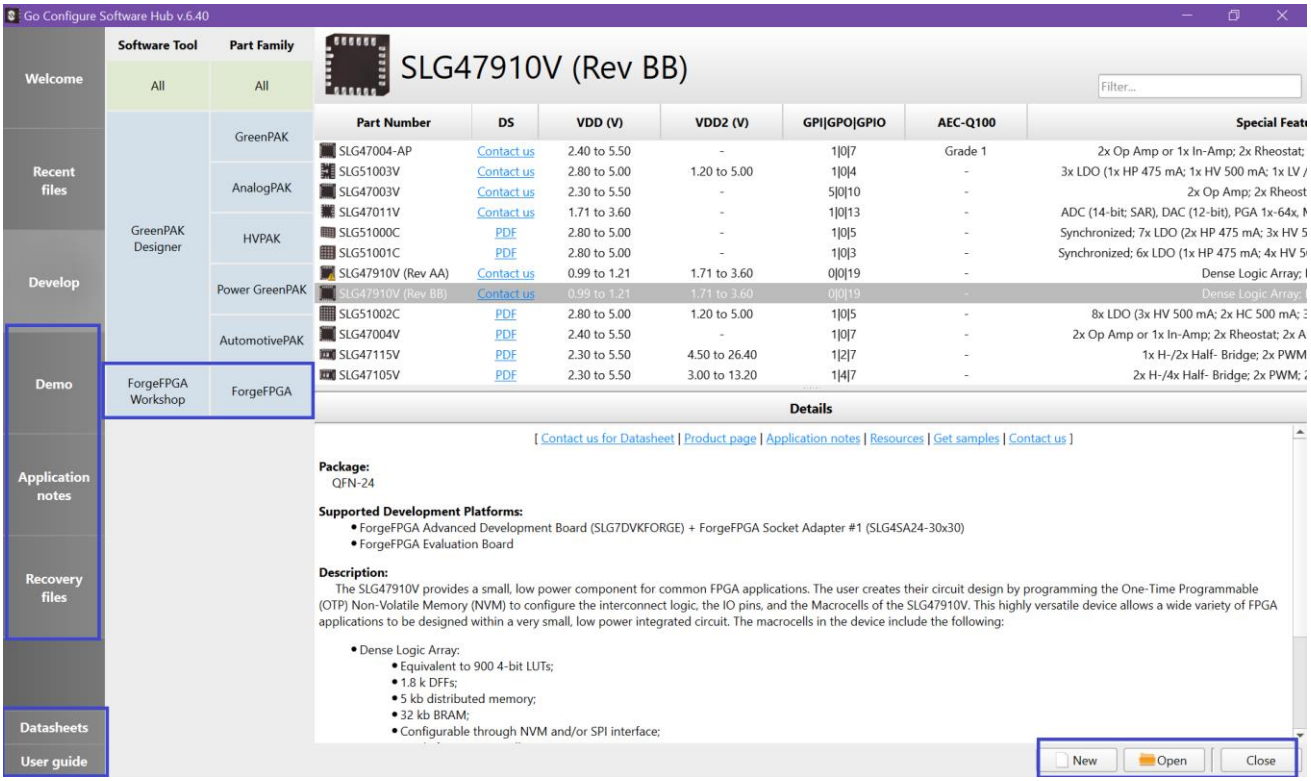


Figure 1. Go Configure Software Hub User Interface

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 19 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).

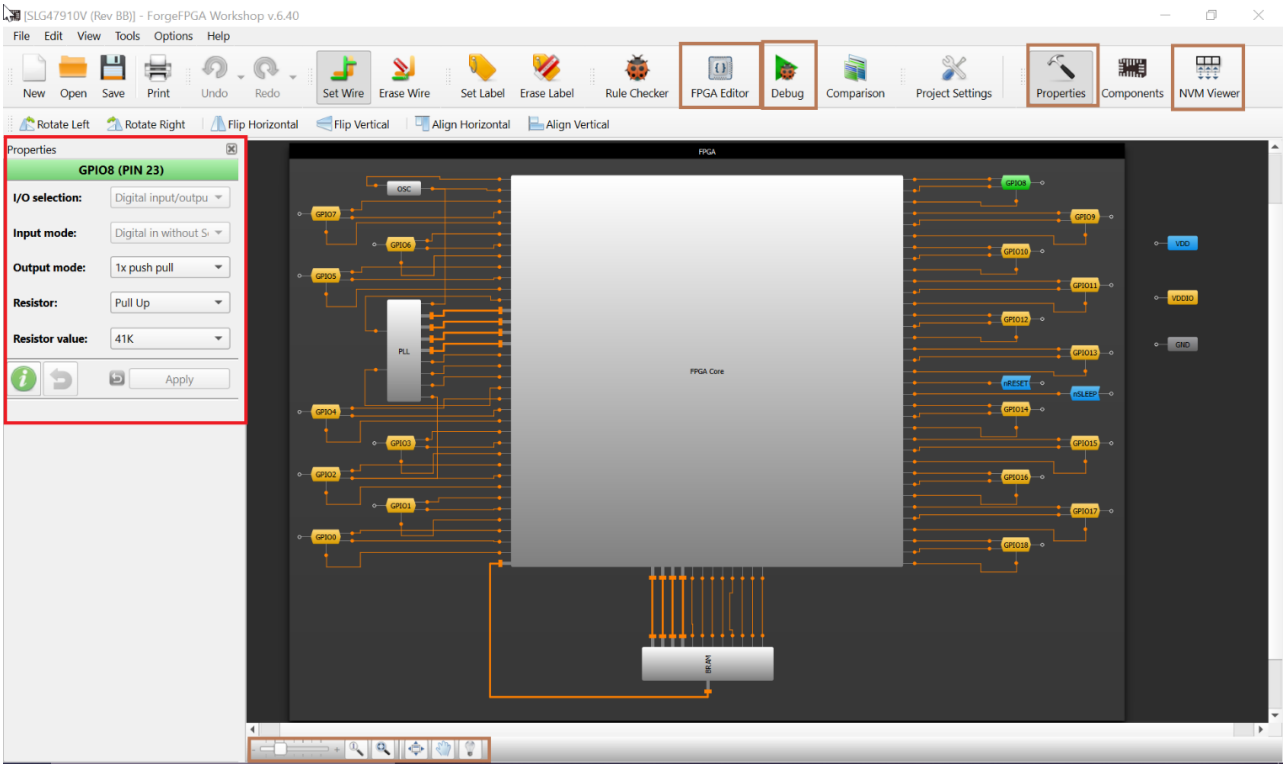


Figure 2. ForgeFPGA Workshop

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)*. 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.

2.1.1 Flowchart

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

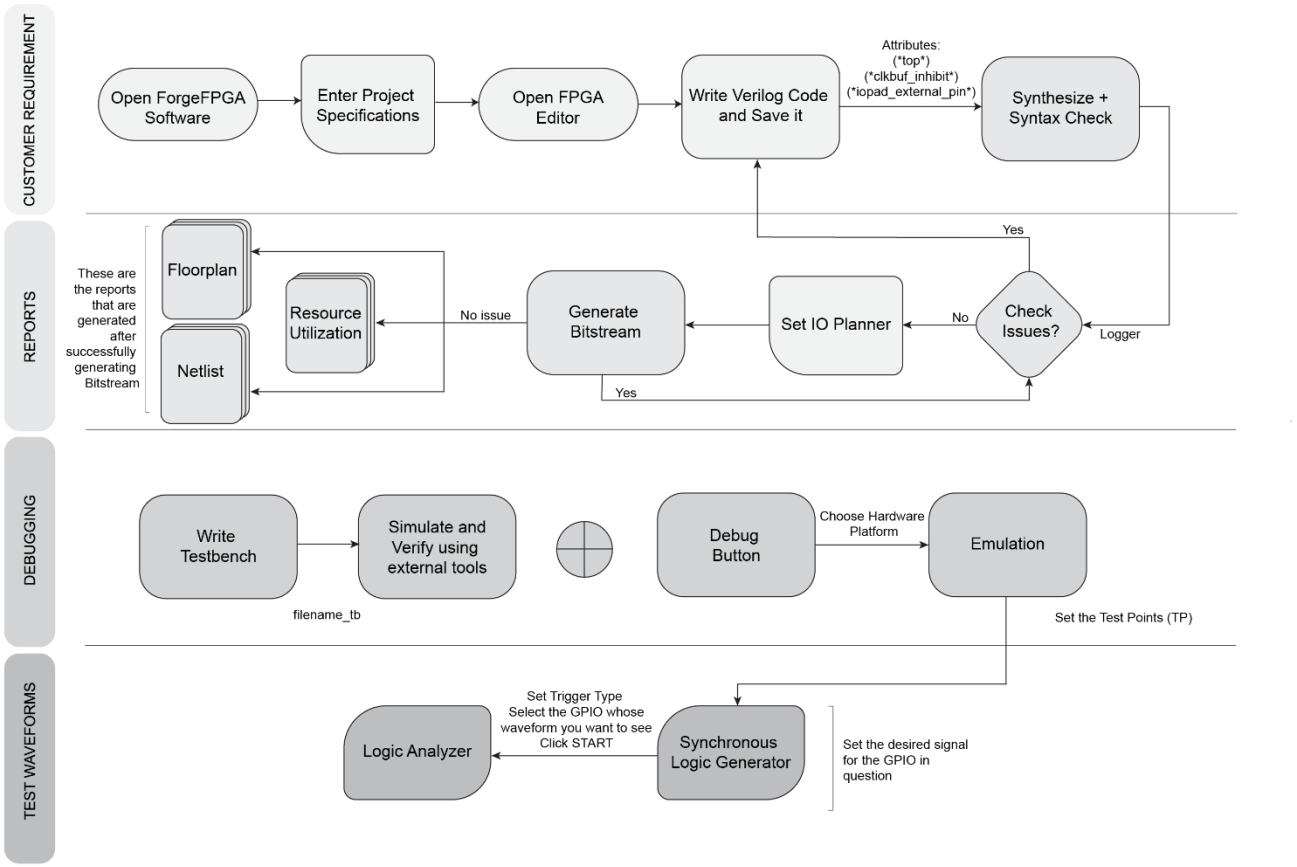


Figure 3. Toolchain Flowchart

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

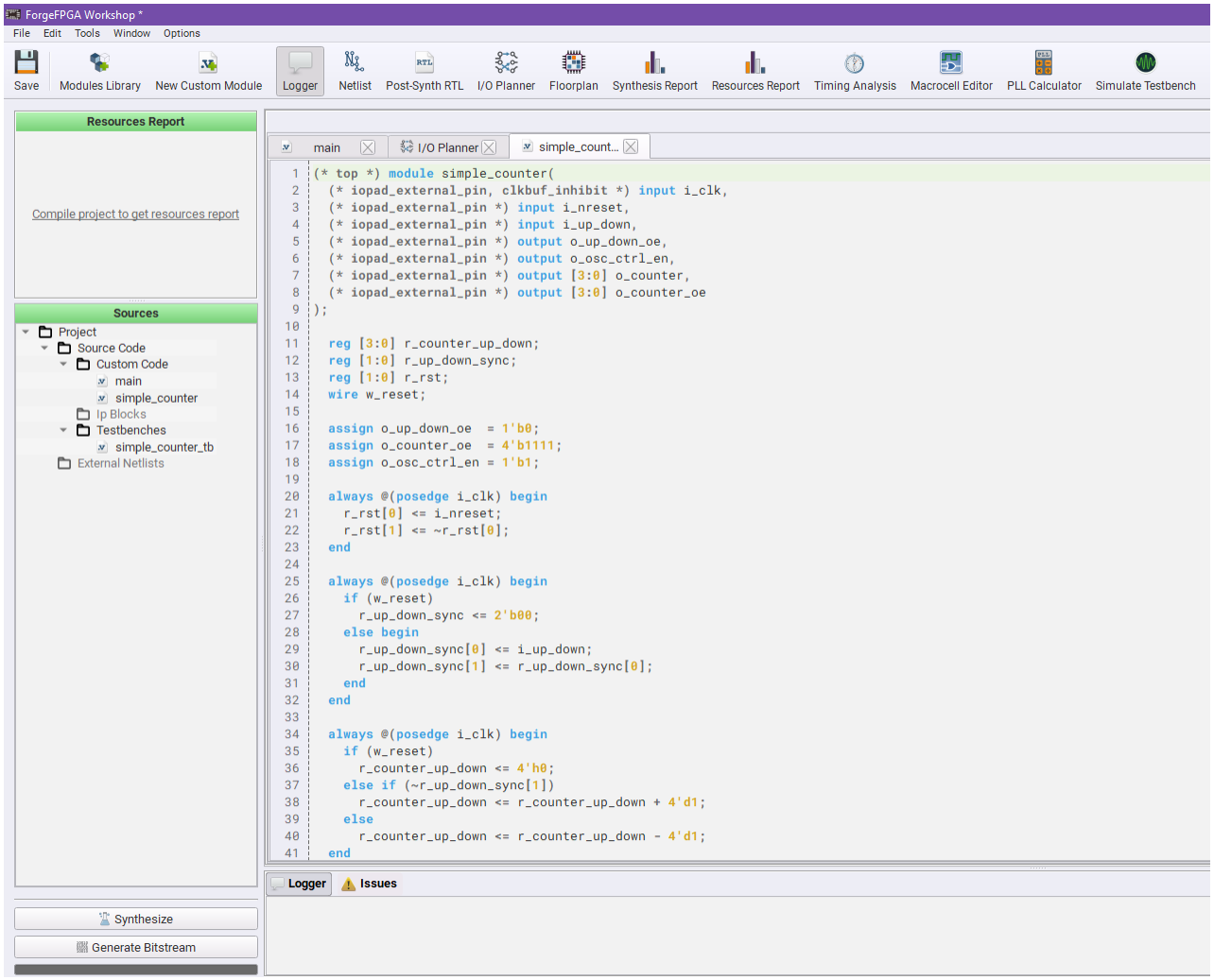


Figure 4. ForgeFPGA Workshop User Interface

2.1.2 Main Menu

Main Menu contains controls are described below:

File

- Save – save current project
- Save As - save current project under another name
- Modules Library – open IP Blocks Wizard
- New Custom Module – add new module
- New Custom Testbench – add new testbench
- Import – used to import an RTL code, testbench and Netlist from another software/synthesis tool
  - Custom Module
  - Custom Testbench
  - Netlist
- Export
  - Custom Module
  - Library Module
  - Testbench
- Load Design Template

- Simple Counter
- Close

Edit (menu works with editable tabs)

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

Tools

- Run Synthesis – synthesize HDL into low-level constructs.
- Generate Bitstream – compilation and mapping low-level constructs to specific device blocks.
- Floorplan
  - Load customer PnR
- Simulation
  - Simulate Testbench
- 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

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

2.1.3 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.

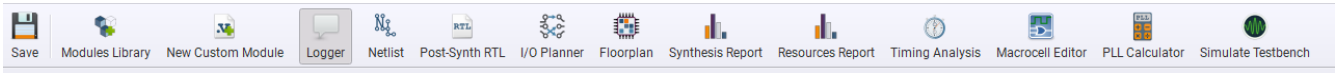


Figure 5. ForgeFPGA Workshop Toolbar

2.1.4 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.

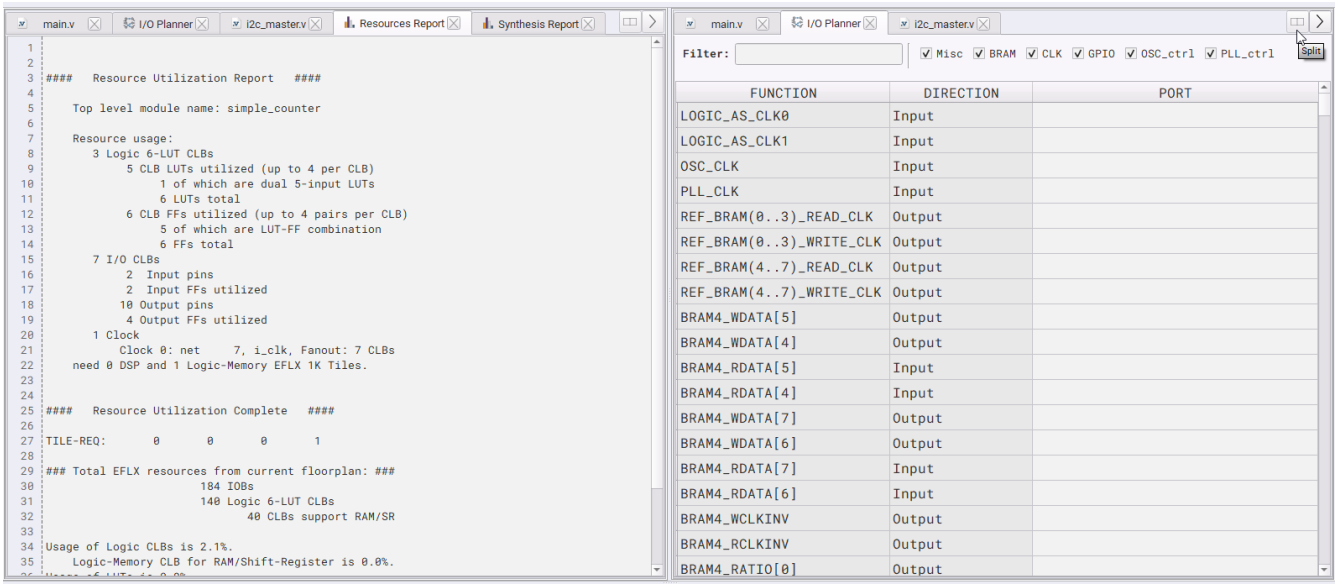


Figure 6. Work Area

2.1.5 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 7).

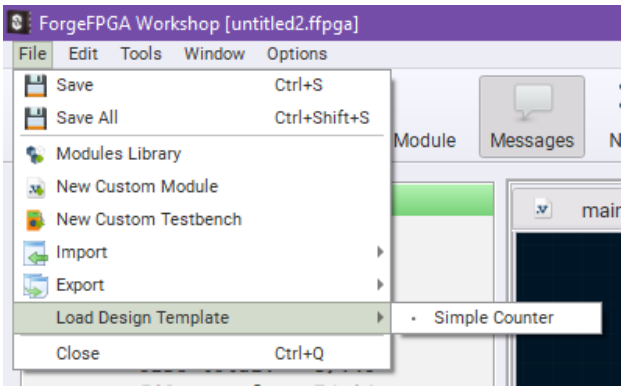


Figure 7. FPGA Design Template

Upon selecting a design template, the Design Template Wizard will appear. 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*

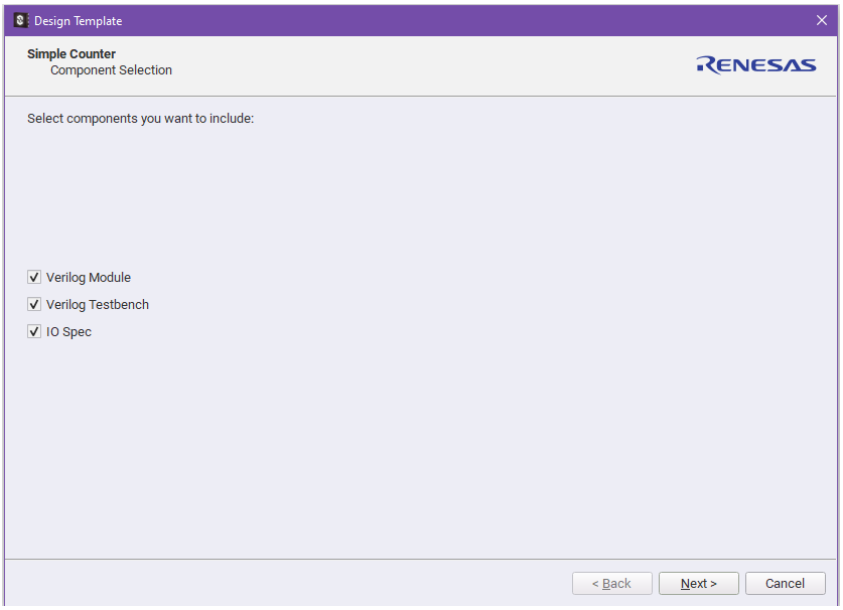


Figure 8. Design Template Component Selection

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 9). In this scenario, the tool will prompt you straight to the Module name.

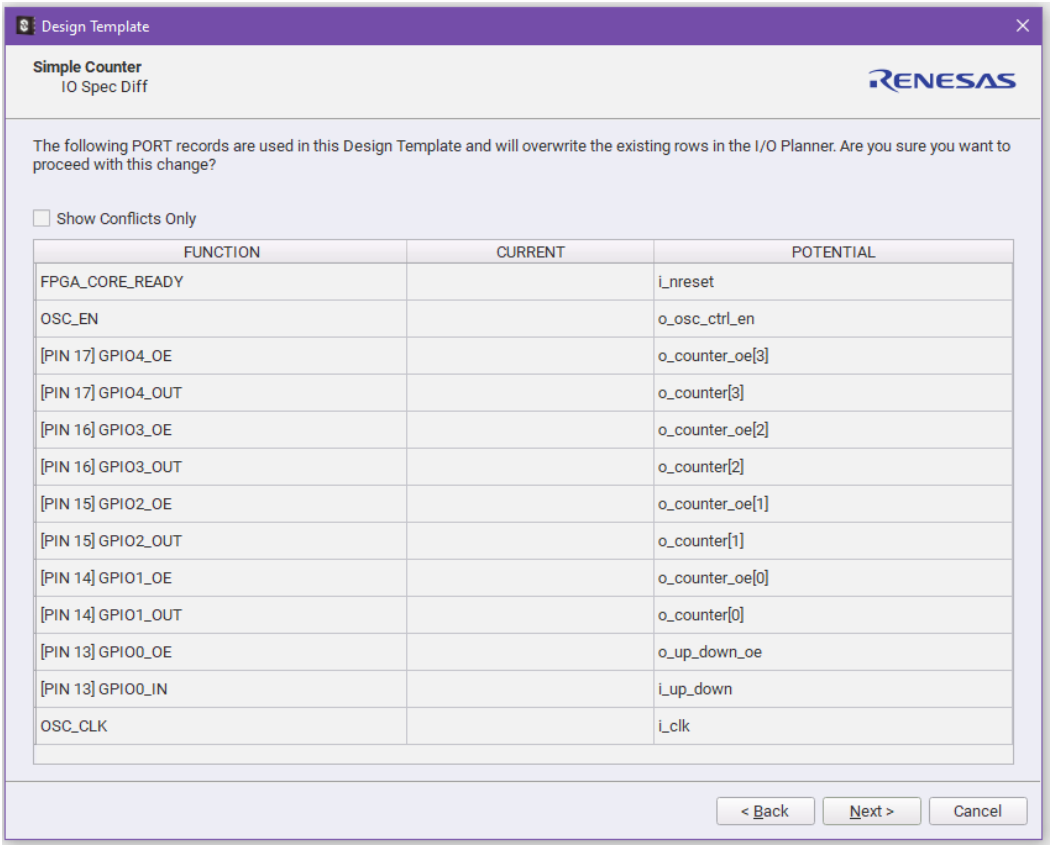


Figure 9. Design Template IO Spec Diff

- *Module Name* assigns names to 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.

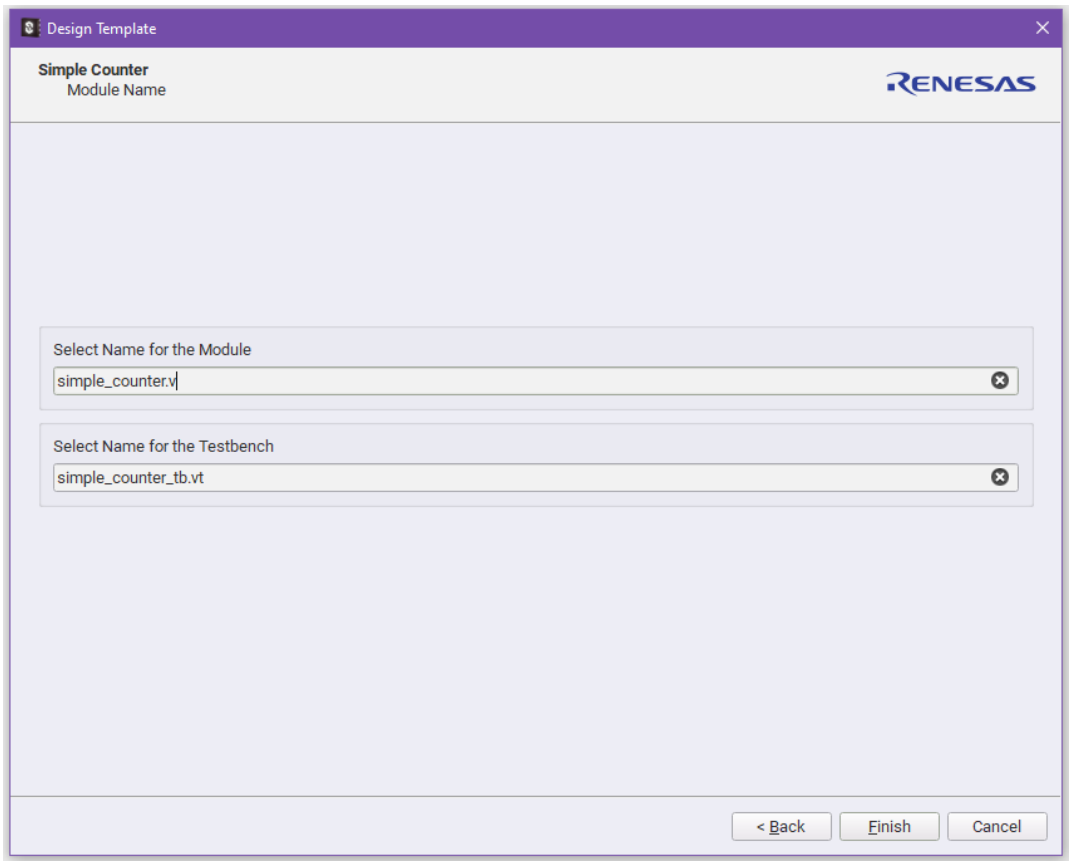


Figure 10. Design Template Module Name

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.

2.1.6 Messages Panel

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

- *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.

Figure 11. Messages Panel

2.1.7 Control Panel

From the left control panel, you can access (see Figure 12):

- *Resources Report* – Extracted from the resource usage report. It shows the part of available resources utilized by the design sources.
- *Sources* – a list of all HDL sources and external netlists that are in the current project file. Here you can find the following subcategories:
  - Custom Code
  - Modules Library
  - Testbenches
  - External Netlists

  The Sources list can be customized via the context menu with the 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.

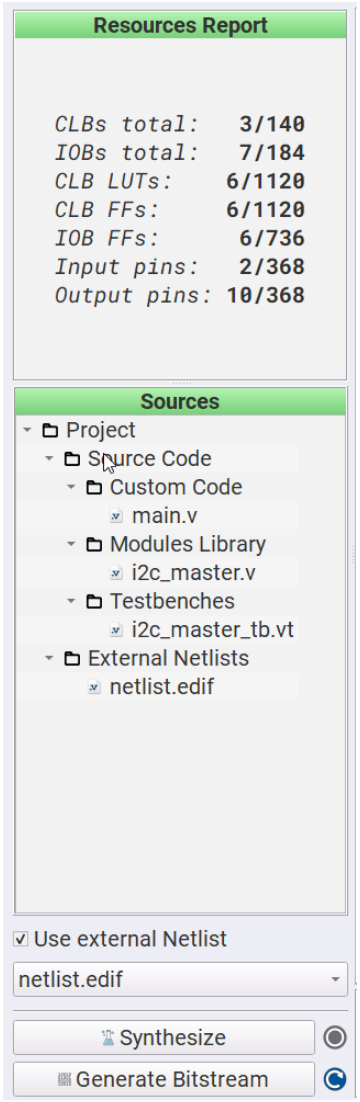


Figure 12. Control Panel

2.1.8 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.

A. 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.

i. *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.

ii. *No DSP* – Do not use DSPs to implement multipliers and associated logic during the synthesis procedure.

iii. *Keep* – Create extra KEEP nets by allowing a cell to drive multiple nets while generating the EDIF netlist.

iv. *Use ABC9* – Use ABV9 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.

v. *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.

i. *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.

ii. *High Density Packing Logic* -Allows you to achieve higher resource utilization per CLB but can worsen congestion and timing.

iii. *Clock-Concurrent Optimization (CCopt)*- Enables Clock-Concurrent Optimization to further optimize timing (must disable Ideal Clock) at the expense of potentially higher clocking power.

iv. *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.

v. *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.

vi. *Maximum Routing Iterations*- Maximum number of routing iterations before exiting with the last clean solution with the best achievable timing.

vii. *Additional arguments*- Add additional arguments to the Generate Bitstream procedure.

c. *Simulation* – User can choose to save their dumpfile 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 dumpfiles. 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.

B. User Settings

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

b. *Tools*

i. *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 if it can.

ii. *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 if it can.

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

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

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

Note: Changes made in the Project Settings category only apply to the project currently being working 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 description of the options to avoid errors and improve your design.

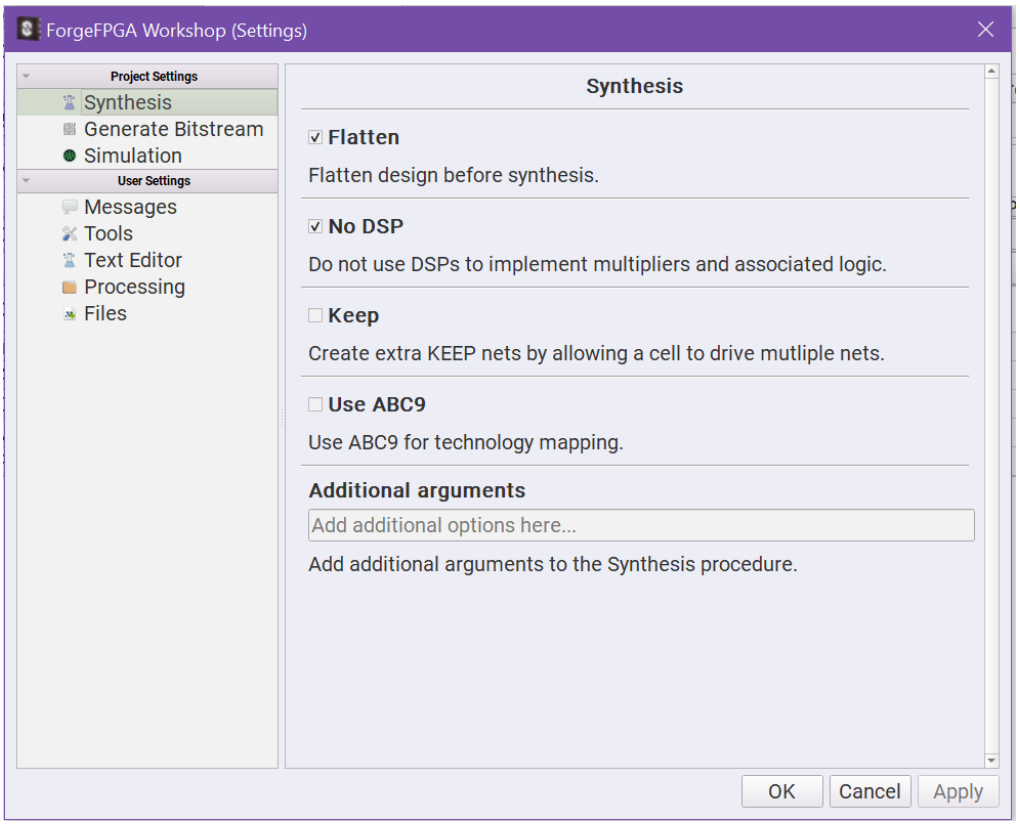


Figure 13. ForgeFPGA Workshop Settings

2.2 Writing HDL 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:

i. `(* top *)` – the main 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.

ii. `(* 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.

iii. `(*iopad_external_pin*)` – all external pins that are used in any module need to be marked with this attribute.

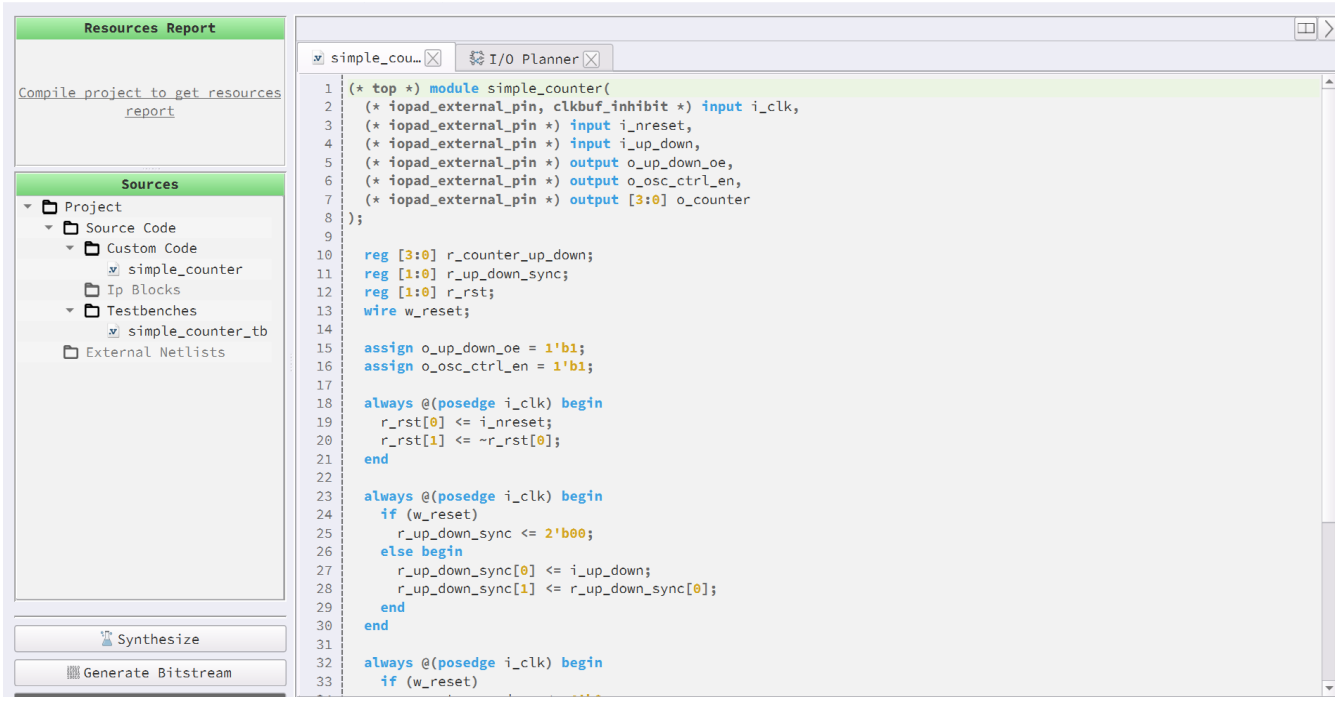


Figure 14. Working with Verilog example

2.2.1 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)

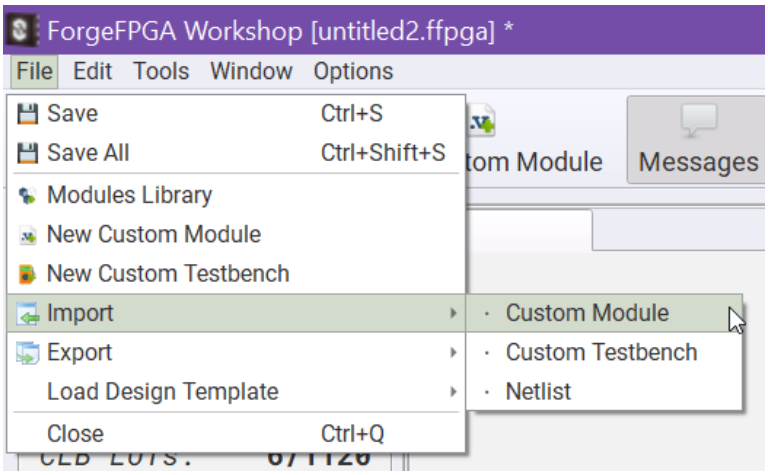


Figure 15. Importing RTL Files

Similarly, you can export files too. You can export a custom module, library module or a testbench that you have designed using ForgeFPGA Workshop and save in your system. You can export via *Main Menu → Export → Custom Module*

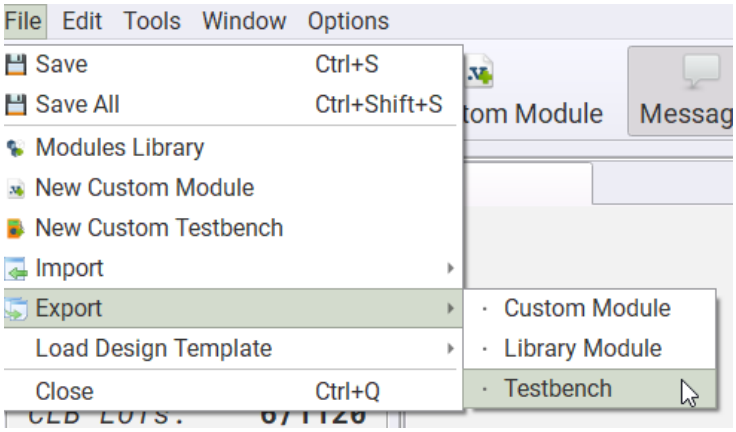


Figure 16. Exporting RTL Files

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 block selected. 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.

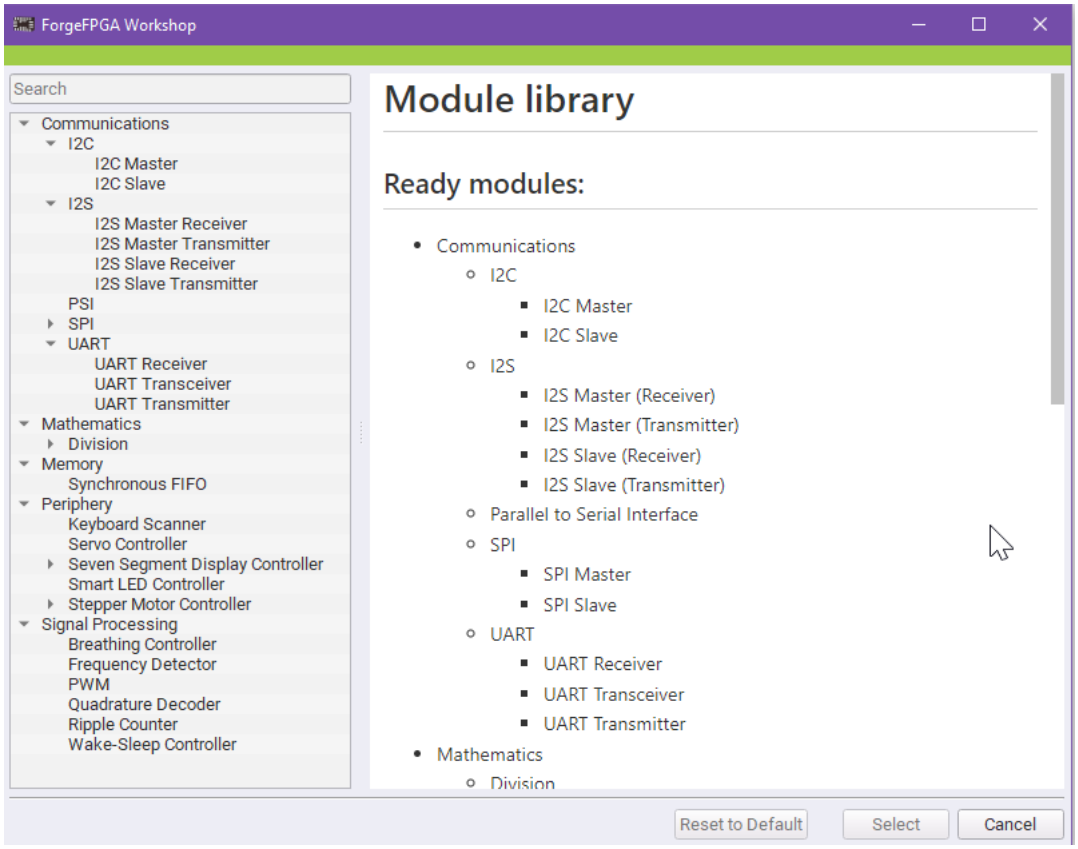


Figure 17. Modules Library GUI

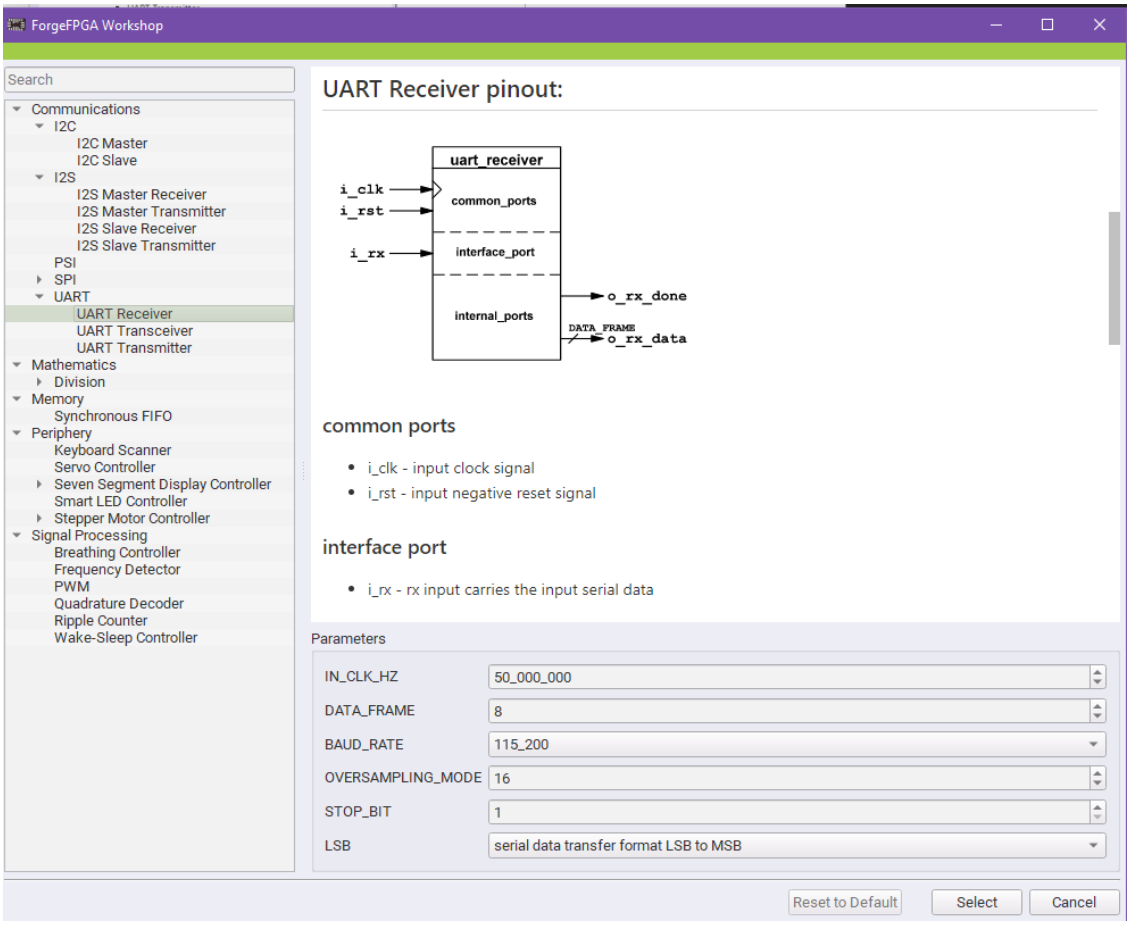


Figure 18. Module Library GUI with Parameters

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 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 synthesising the design or from *Windows → Synthesis Report.*

2.4.1 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, FDREs and Carry-Chain logic.

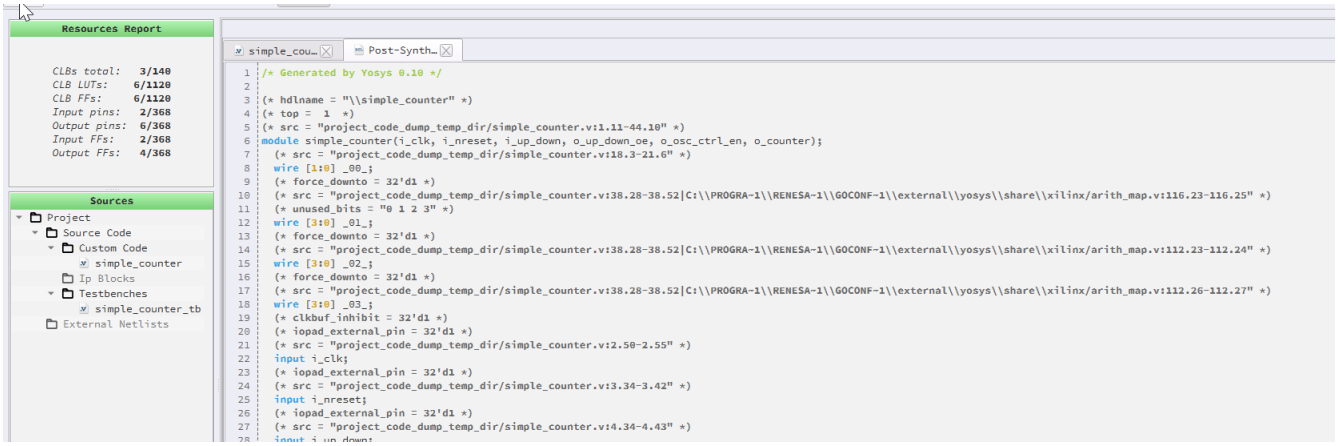


Figure 19. Post-Synth RTL

2.4.2 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.

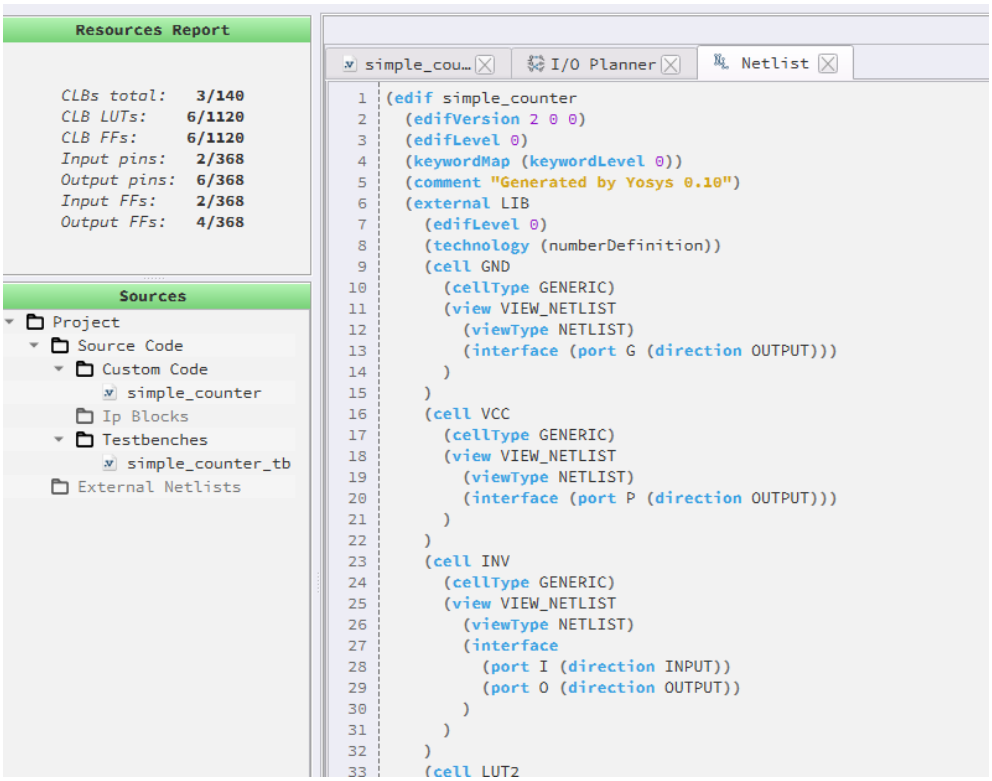


Figure 20. Netlist

2.4.3 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:

- *Position* – the coordinates of the specific device I/O port on the FPGA.
- *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 of all the ports defined in their HDL Code.

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.

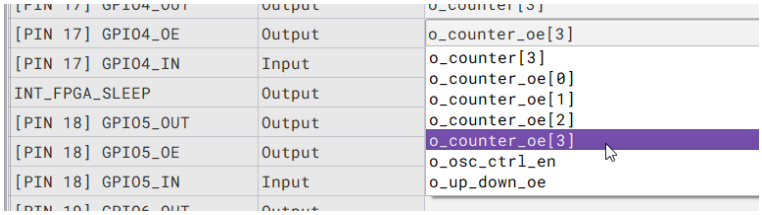
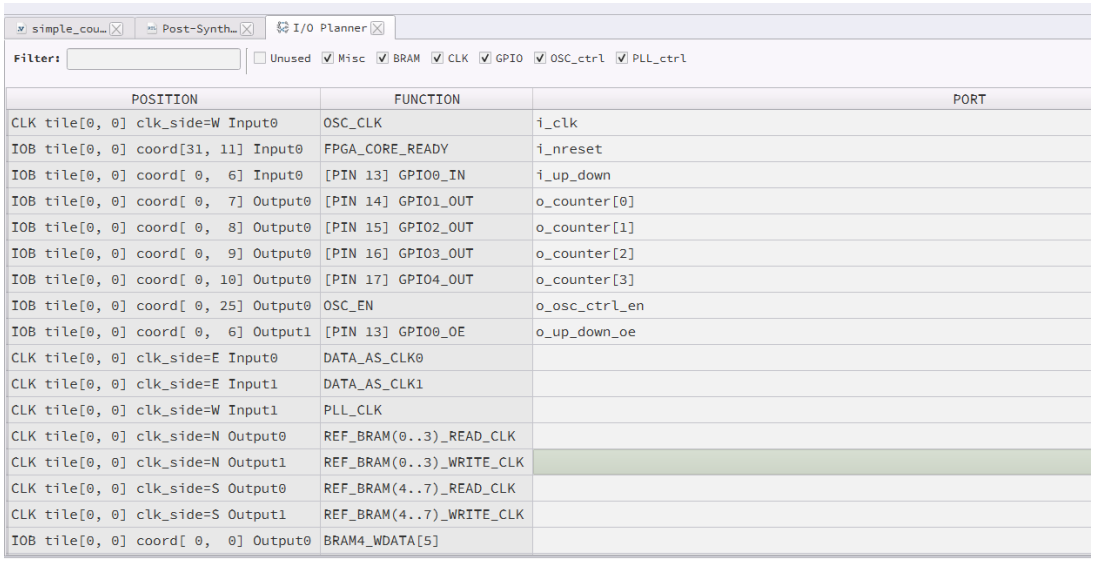


Figure 21. I/O Port Signal Assignment

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	PORT
CLK tile[0, 0] clk_side=W Input0	OSC_CLK	i_clk
IOB tile[0, 0] coord[31, 11] Input0	FPGA_CORE_READY	i_nreset
IOB tile[0, 0] coord[0, 6] Input0	[PIN 13] GPIO0_IN	i_up_down
IOB tile[0, 0] coord[0, 7] Output0	[PIN 14] GPIO1_OUT	o_counter[0]
IOB tile[0, 0] coord[0, 8] Output0	[PIN 15] GPIO2_OUT	o_counter[1]
IOB tile[0, 0] coord[0, 9] Output0	[PIN 16] GPIO3_OUT	o_counter[2]
IOB tile[0, 0] coord[0, 10] Output0	[PIN 17] GPIO4_OUT	o_counter[3]
IOB tile[0, 0] coord[0, 25] Output0	OSC_EN	o_osc_ctrl_en
IOB tile[0, 0] coord[0, 6] Output1	[PIN 13] GPIO0_OE	o_up_down_oe
CLK tile[0, 0] clk_side=E Input0	DATA_AS_CLK0	
CLK tile[0, 0] clk_side=E Input1	DATA_AS_CLK1	
CLK tile[0, 0] clk_side=W Input1	PLL_CLK	
CLK tile[0, 0] clk_side=N Output0	REF_BRAM(0..3)_READ_CLK	
CLK tile[0, 0] clk_side=N Output1	REF_BRAM(0..3)_WRITE_CLK	
CLK tile[0, 0] clk_side=S Output0	REF_BRAM(4..7)_READ_CLK	
CLK tile[0, 0] clk_side=S Output1	REF_BRAM(4..7)_WRITE_CLK	
IOB tile[0, 0] coord[0, 0] Output0	BRAM4_WDATA[5]	

Figure 22. 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.

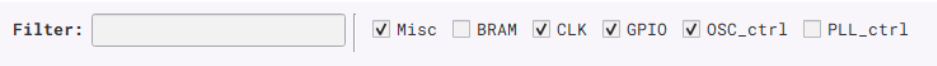


Figure 23. I/O Planner Filter selection

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*.

A detailed explanation of a few important signals in the I/O Planner is provided in *IO Planner Signals*.

2.4.4 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*

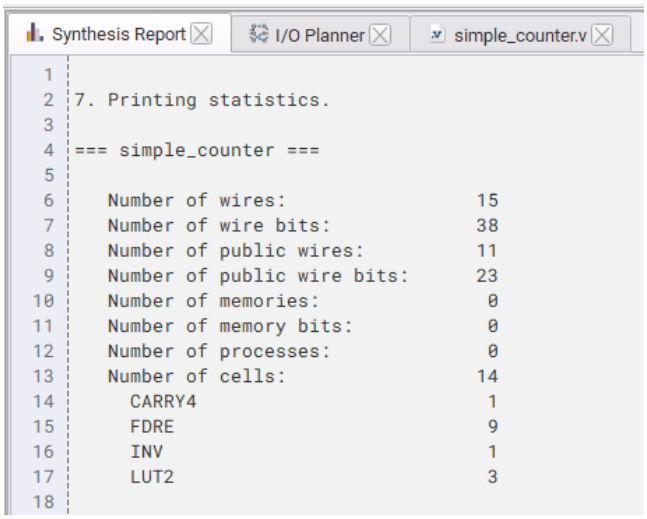


Figure 24. Synthesis Report

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 hex 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.

2.5.1 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.

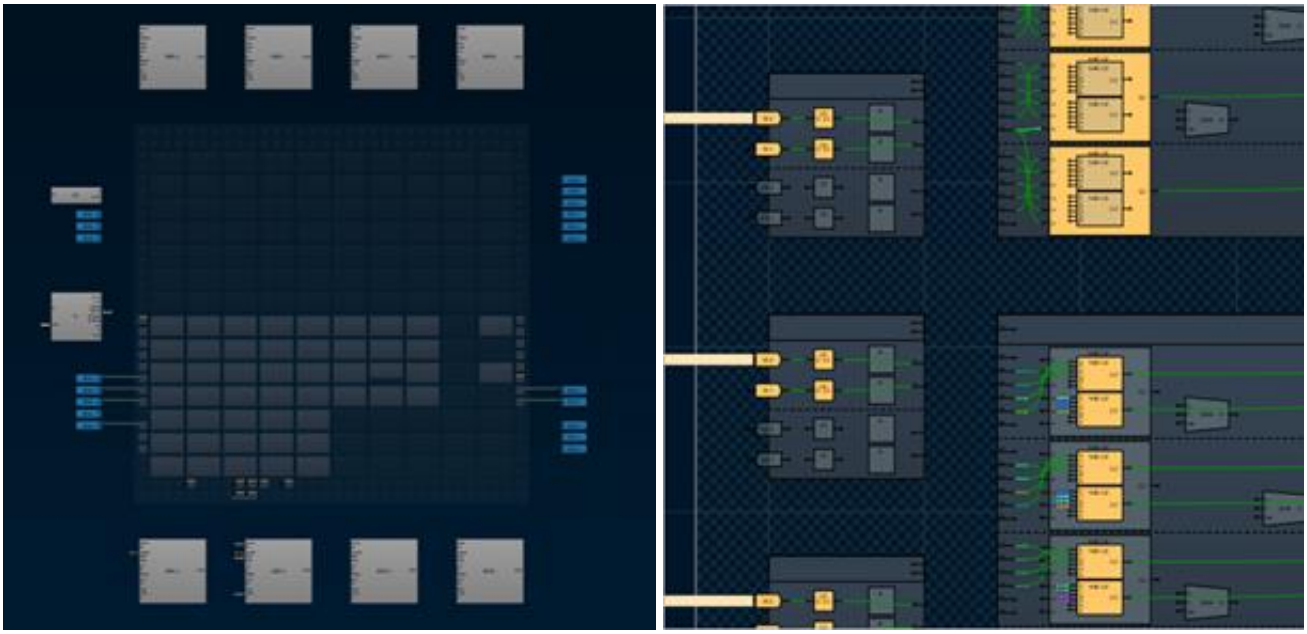


Figure 25. Floorplan Window

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.

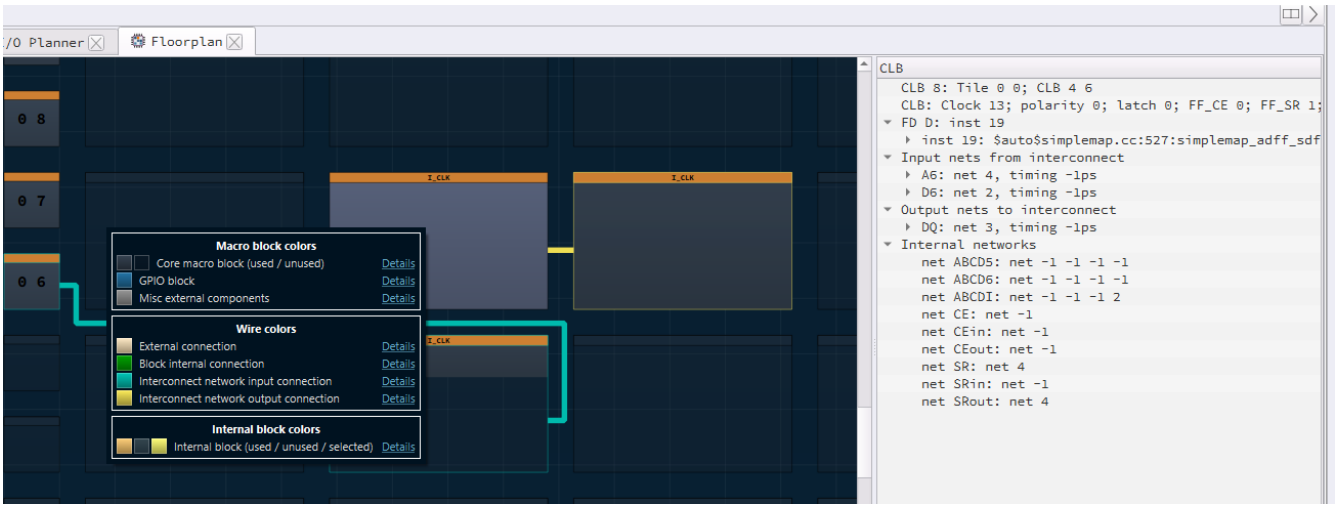


Figure 26. Place and Route Results

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:

- *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

2.5.2 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*.

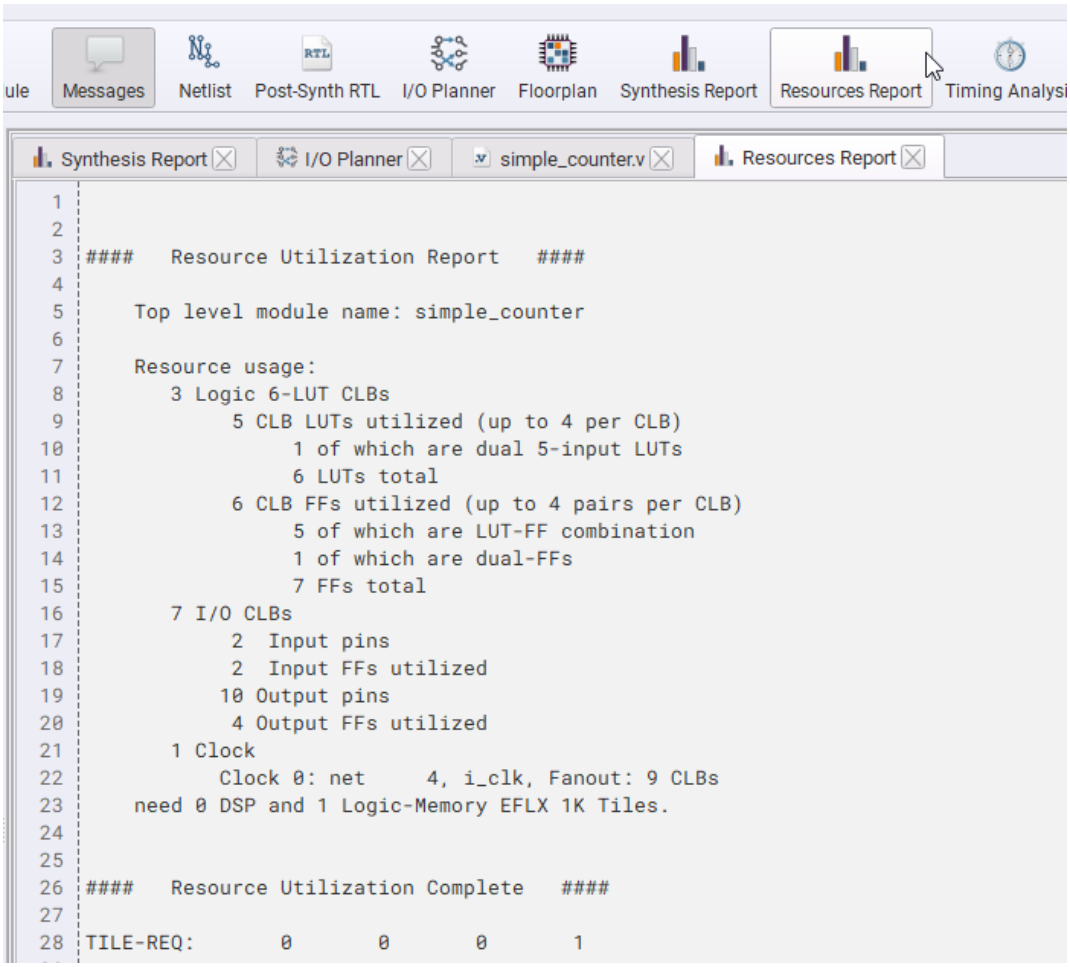


Figure 28. Resources Utilization Report

2.5.3 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.

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

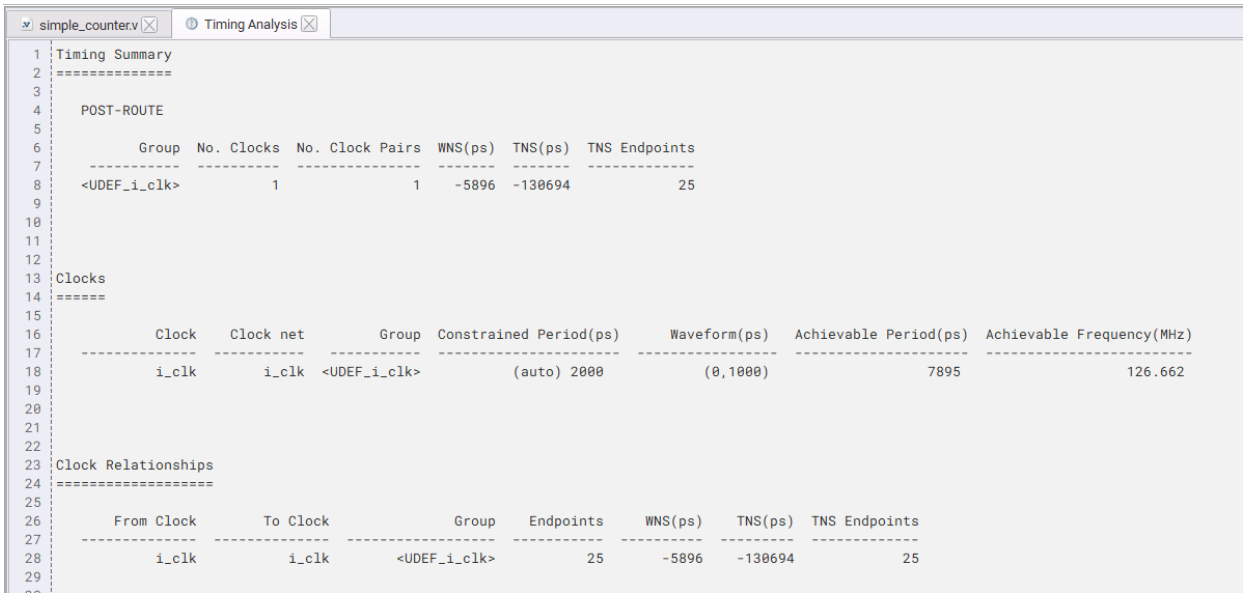


Figure 29. Timing Analysis

Note: The Timing Analysis feature is currently under development.

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. Verifying a complex system after implementing the hardware is not a wise choice. It is ineffective in terms of money, time, and resources. Hence, in the case of FPGA, 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 [8] for the installation process of the additional software and quick start guide.

2.6.1 Writing a Testbench

To work with the Simulation, you should have a testbench module for the FPGA design. You can add a testbench from the Modules Library or open a module from the main menu by going to *File → New Custom Testbench* and saving the name of the *testbench.*

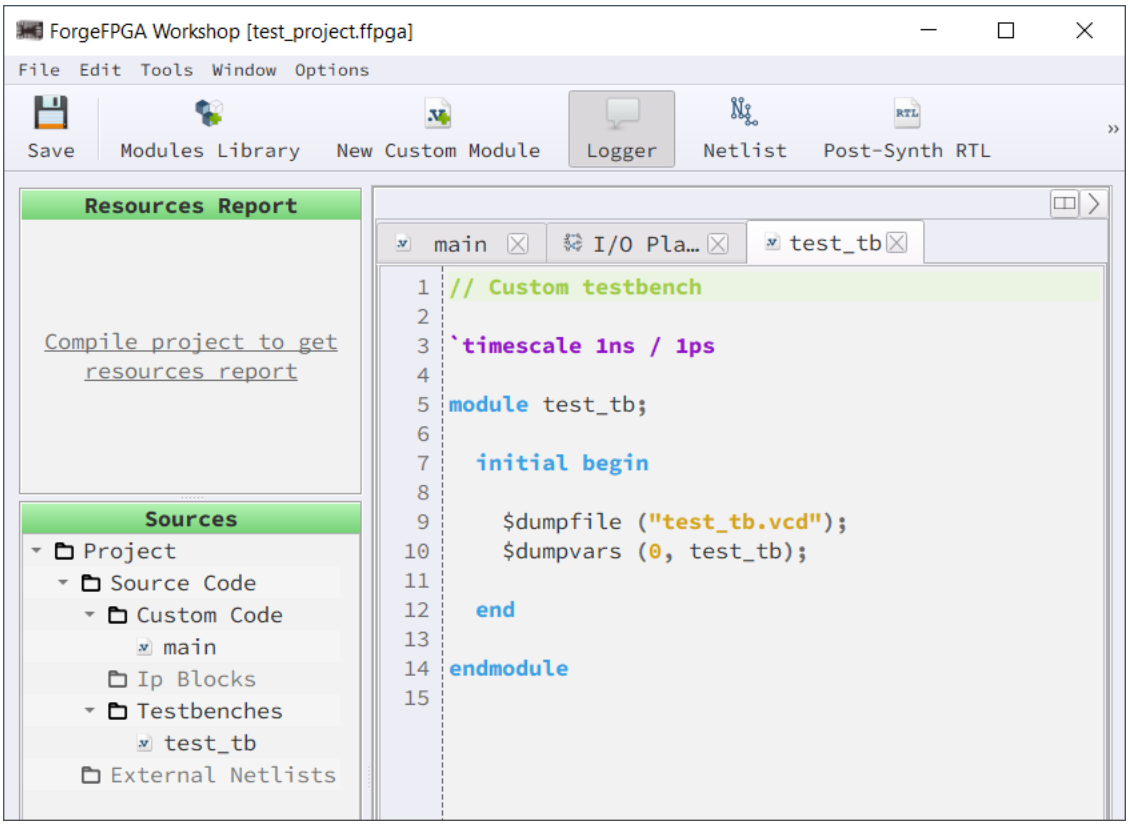


Figure 30. Custom Testbench example

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. Please refer to [8] for details on how to write a testbench.

simple_counter.v | Timing Analysis | simple_counter_tb.vt

```
1  `timescale 1ns / 1ps
2
3  module simple_counter_tb();
4
5    // Declare the testbench variables
6    reg        r_clk;
7    reg        r_nreset;
8    reg        r_up_down;
9    wire       w_up_down_oe;
10   wire       w_osc_ctrl_en;
11   wire [3:0] w_counter;
12   wire [3:0] w_counter_oe;
13
14   // Instantiate the simple_counter design and connect it with the testbench variables
15   simple_counter dut(
16     .i_clk         (r_clk        ),
17     .i_nreset      (r_nreset     ),
18     .i_up_down     (r_up_down    ),
19     .o_up_down_oe  (w_up_down_oe ),
20     .o_osc_ctrl_en (w_osc_ctrl_en),
21     .o_counter     (w_counter    ),
22     .o_counter_oe  (w_counter_oe )
23   );
24
25   // Generate a clk that drives the design that flips value every 10ns
26   always #10 r_clk = ~r_clk;      // Frequency: 50MHz
27
28   // The initial block forms the stimulus for the testbench
29   initial begin
30
31     $dumpfile ("simple_counter_tb.vcd");      // GTKWave simulation dump file
32     $dumpvars (0, simple_counter_tb);         // Dump variable file
33
```

Figure 31. Simple Counter Testbench example from Template Design

2.6.2 Simulating a Testbench

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

The FPGA Editor ensures you can keep working from the last saved state of 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 the previous time.

2.7 PLL Calculator

User can use the PLL Calculator to calculate which parameters are compatible with the SLG47910 Specifications. You can access it through the FPGA Editor → PLL Calculator in the toolbar. The PLL Calculator uses the below equation to calculate the value of PLL_CLK

$$PLL\_CLK = \frac{f_{reference\_clock} \times PLL\_FBDIV}{PLL\_REFDIV \times PLL\_POSTDIV1 \times PLL\_POSTDIV2}$$

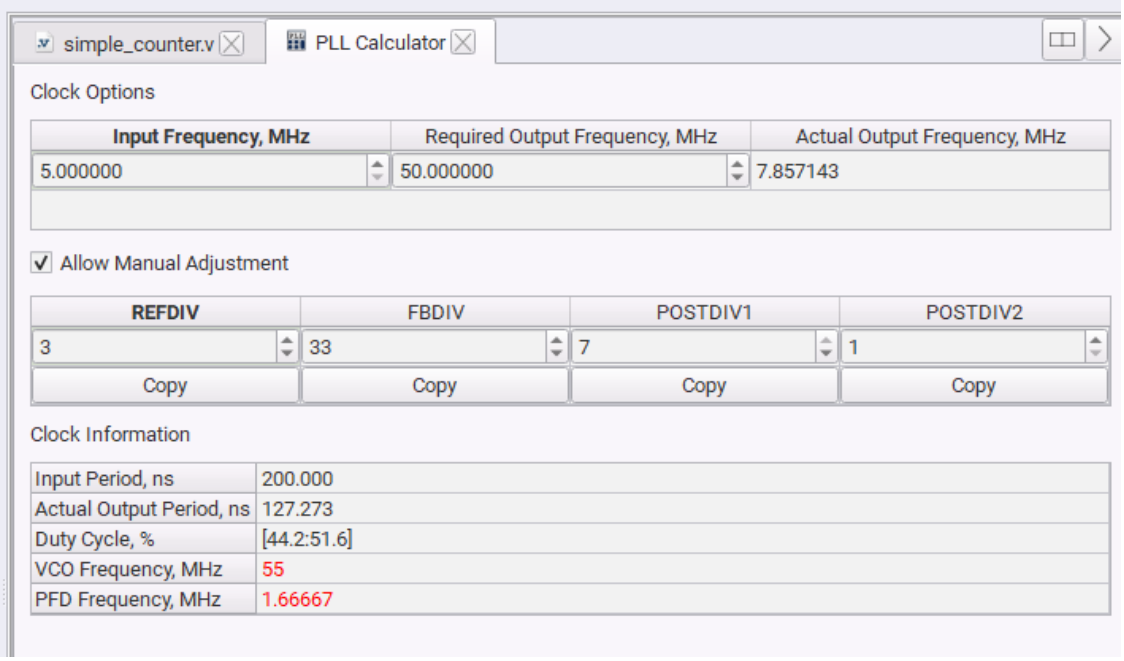


Figure 32. PLL Calculator

The tool has two calculation modes:

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

Note: Check *Allow Manual Adjustment* to enable Manual Adjustment mode

The values in the Clock Information table are based on the set data. If the user inputs a value which is outside the accepted range, then the resulting value changes to red color indicating an error (see Figure 32).

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*.

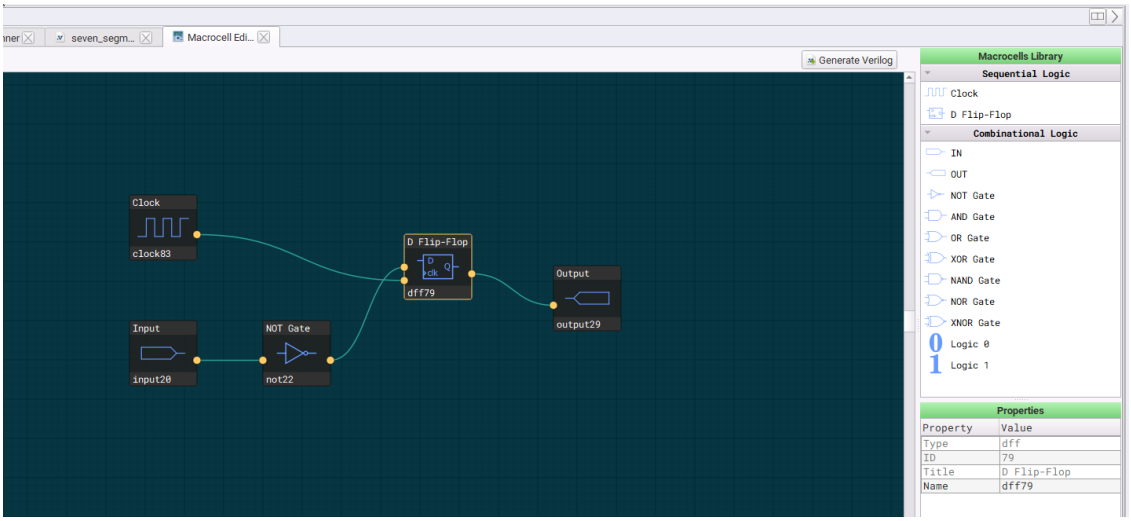


Figure 33. Macrocell Editor

The Macrocell Editor has two parts; the green screen is where the schematic of the circuit is designed by using the components on the right side of the screen which is a library of the all the *Sequential Logic & Combinational Logic* 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 (double-click the field or macrocell)

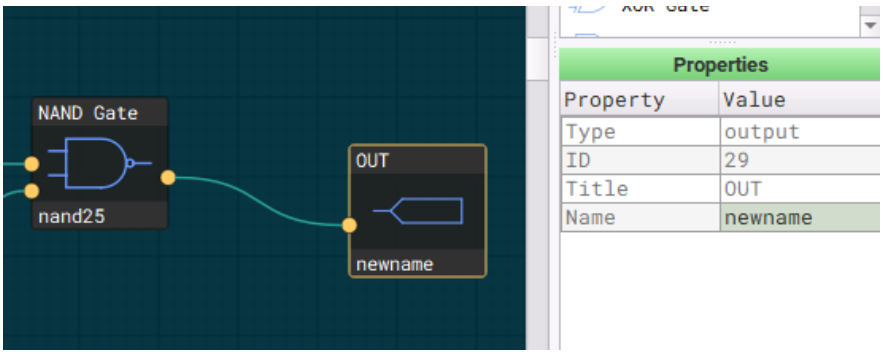


Figure 34. Changing the names of Input/Output Pins

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 build 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.

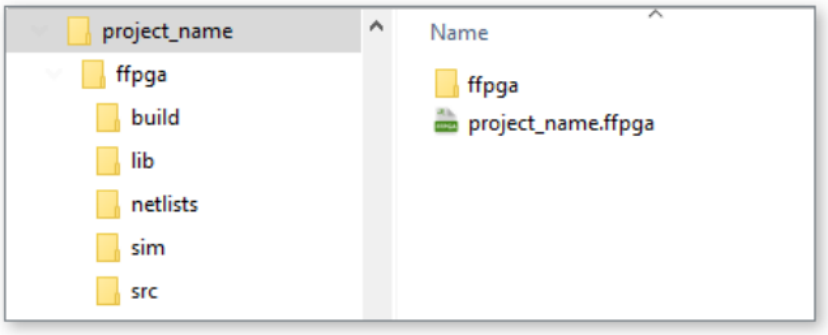


Figure 35. FPGA Project Folder Structure

You can also use the Open Containing Folder feature in the Sources Panel to quickly locate the file on disk (see Figure 36)

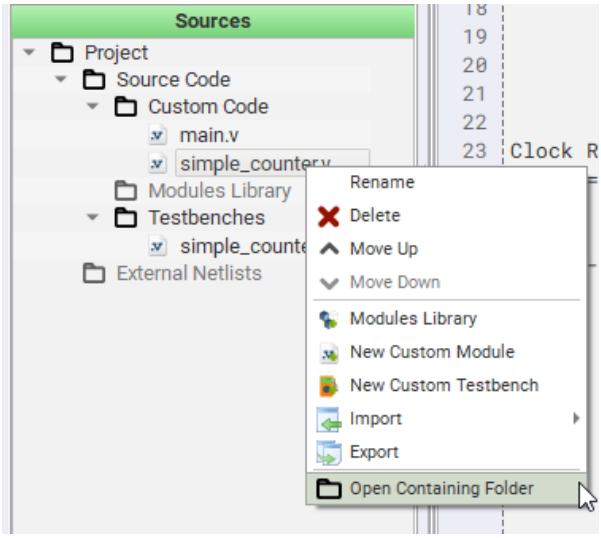


Figure 36. Check Source File location

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_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 needs to now 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 in order to configure the emulation process. The Test Point controls allow the user to configure the GPIOs in different options.

3.1 Hardware Platforms

After the Debug button is pressed, the Development Platform Selector and the two hardware options will be displayed (See Figure 37).

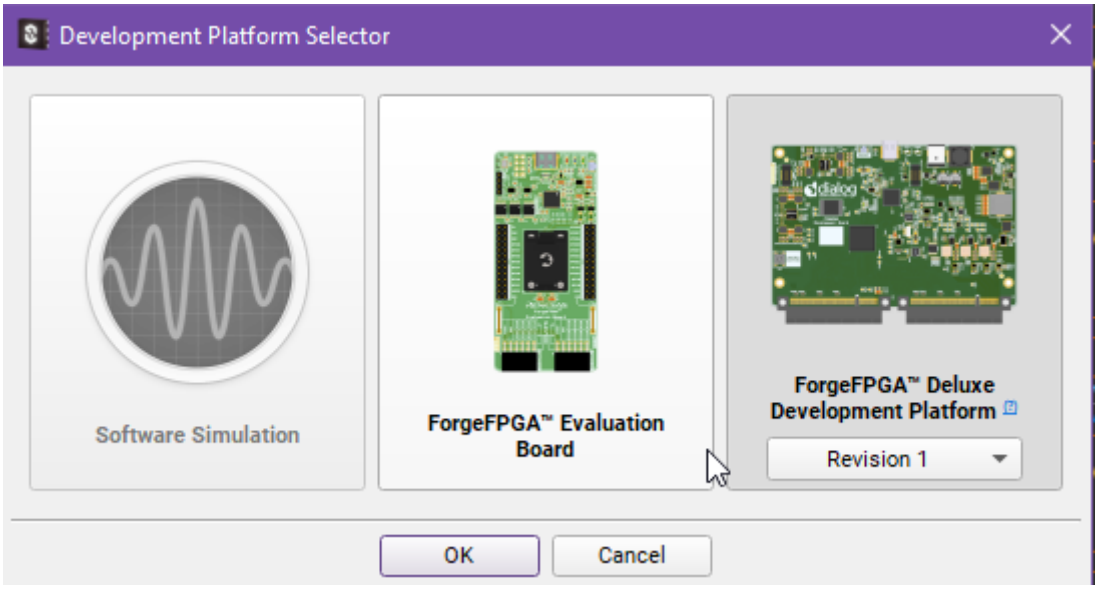


Figure 37. Development Platform Selector

3.1.1 ForgeFPGA Deluxe Development Board

The ForgeFPGA Deluxe Development Board 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. 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 Board. To implement this, the platform uses a Dual PCIe connector.

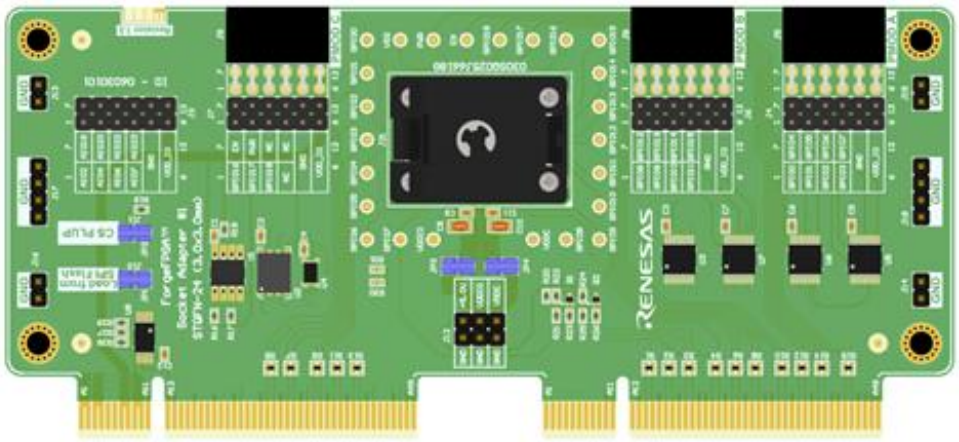


Figure 38. Socket Board Adapter

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.

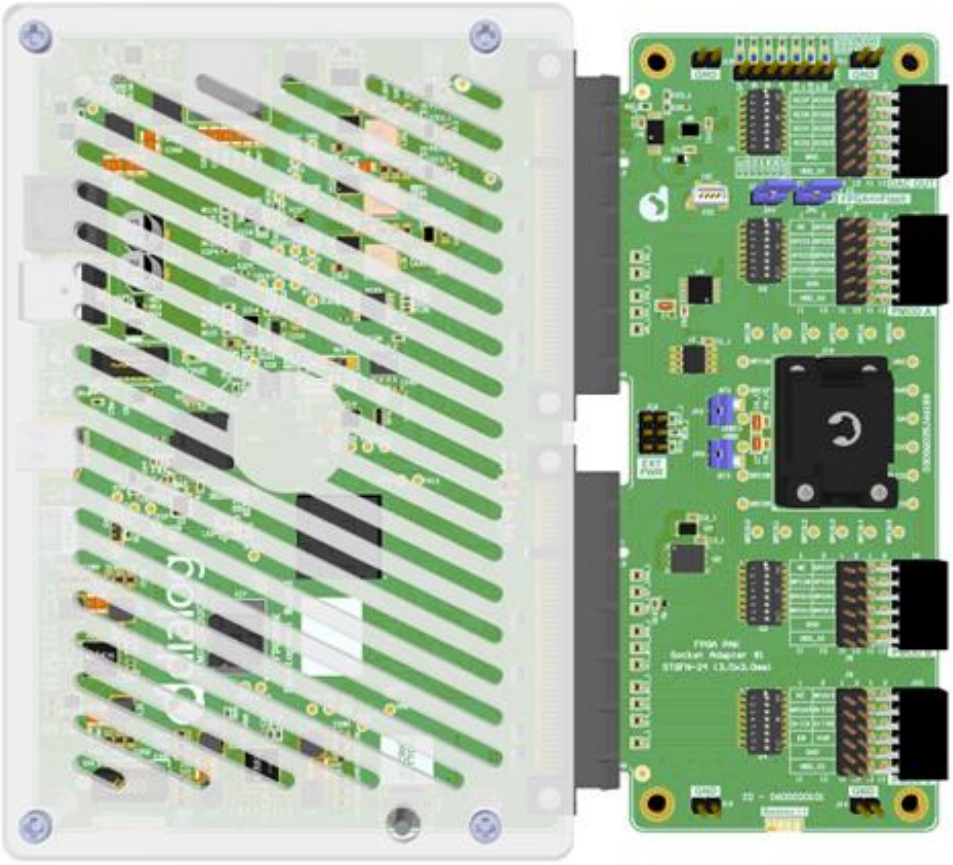


Figure 39. Assembled equipment for working with a chip in the socket.

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].

3.1.2 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 SLG47910 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.

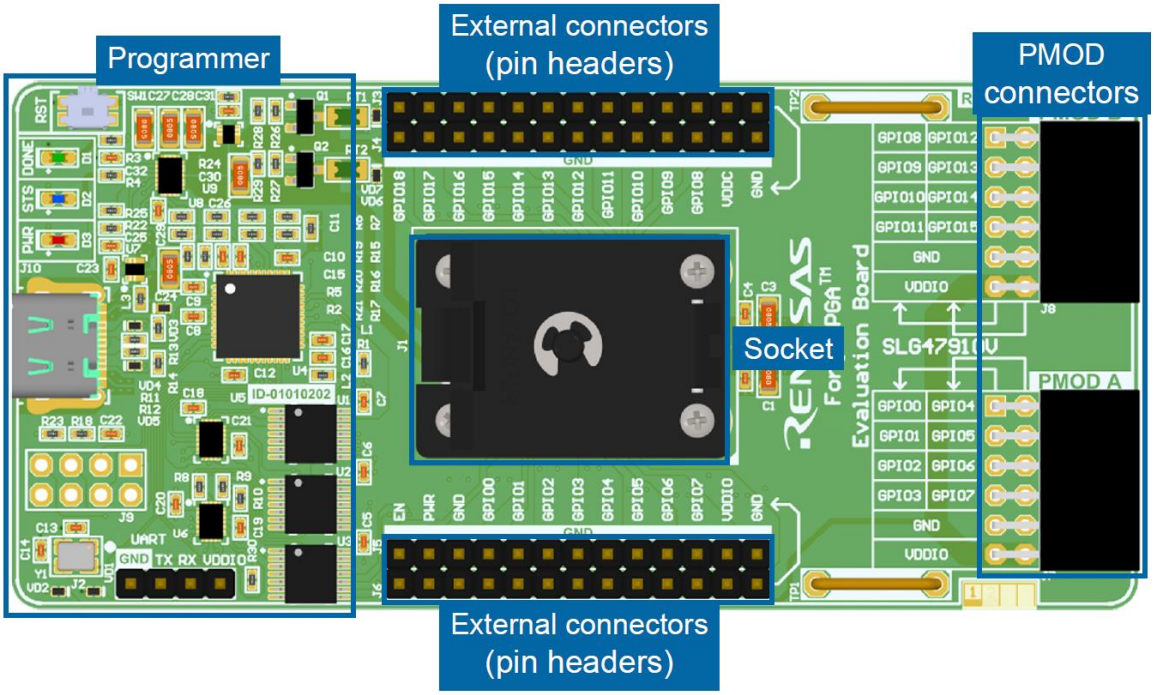


Figure 40. ForgeFPGA Evaluation Board v2.0

The ForgeFPGA Evaluation Board v1.0 is a simplified version of the platform and therefore, has limited functionality. Since the socket is absent, the SLG47910 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 refer to [7].

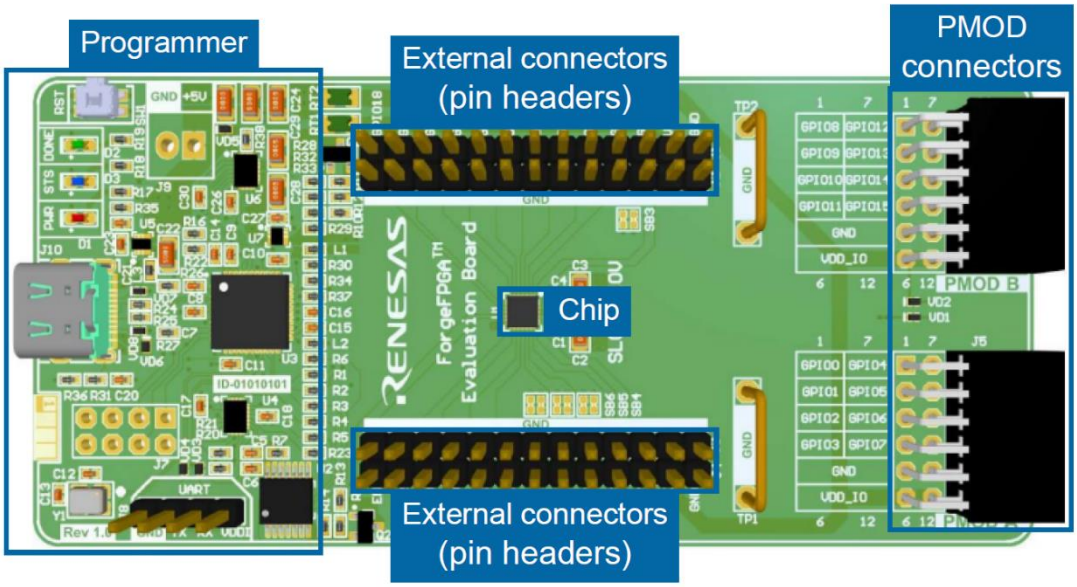


Figure 41. ForgeFPGA Evaluation Board v1.0

3.2 Platform Configuration Guide

Under the Debug tab on the main menu, the Recommended Platform Configuration guide contains information about suitable sockets, adapters, 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 42).

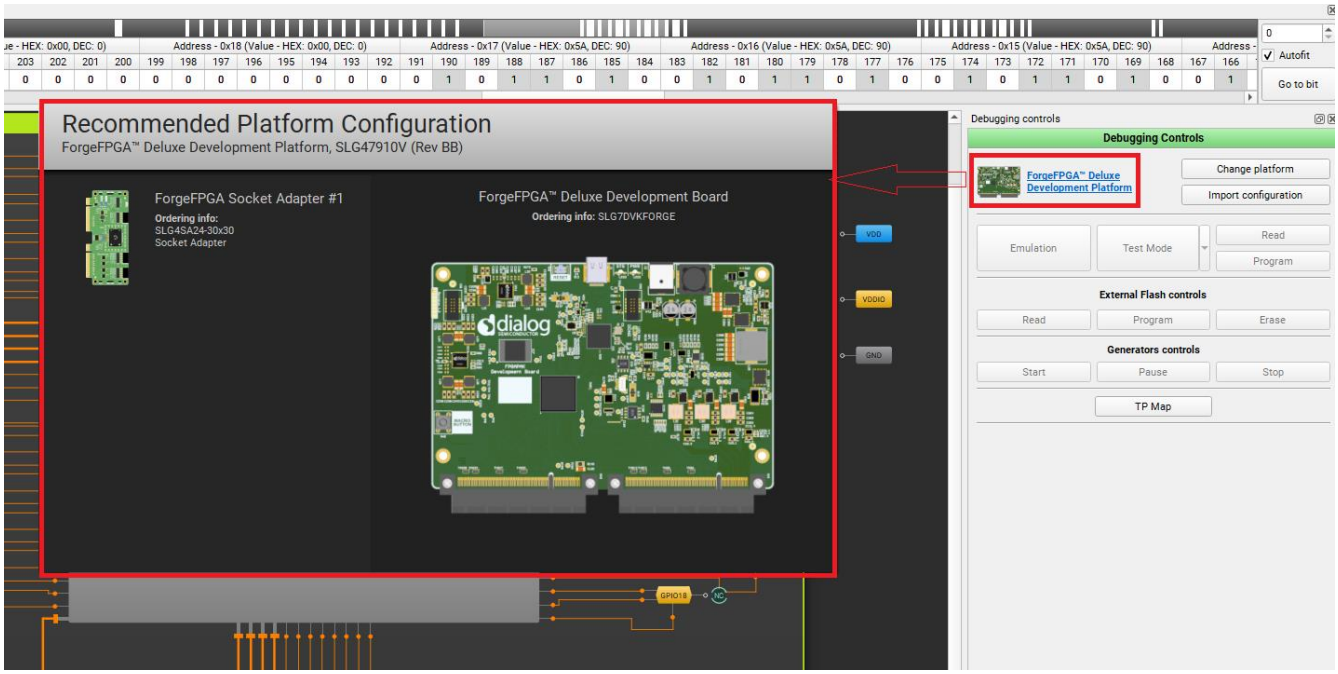


Figure 42. Platform Configuration Guide

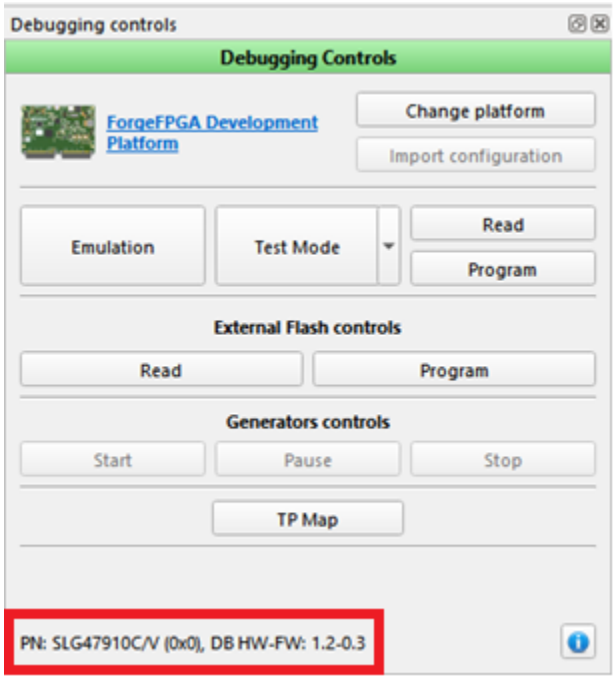


Figure 43. Debugging Controls

The Debugging Controls panel contains the UI controls for chip communication and programming. Now let's review the different parts of the Debugging Controls (Figure 43).

a. *Change platform* – Opens the list of development platforms that are supported
b. *Import configuration* – Import's the configuration of test points from another platforms.
c. *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.
d. *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. To do this, turn on Test Mode and the internal VDD button. 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.
e. *Test Mode (*)* – Load data from flash to the device.
f. *Read* – Read the device using the hardware board or from the external flash memory.
g. *Program* – Program the device or the external with the current project. As the SLG47910 is OTP, it can be programmed only once.
h. *External Flash Controls* – Data can also be read from External Flash. The read data can be accessed from the Project Data window after the data has been read. The current bitstream can also be loaded onto the external flash by pressing the Program button under this category.
i. *Generator Controls* – During emulation, you can start all the test points together or pause them or even Stop them from running altogether.
j. *TP Map* – The Test Points of each pin will be shown next to their respective pin (Figure 44).
k. *PN (Part number and Versions)* – SLG47910 C/V (0x0), DB HW-FW: 1.2-0.3
After the Development board with the chip in the Socket Adapter Board has been connected, we can see the PN (Part Number) SLG47910 being displayed at the bottom of the Debugging Controls Dialog Box (Figure 43). The Debugging Control Window also indicates the Development Boards (DB) Hardware (HW) and Firmware (FW) version.

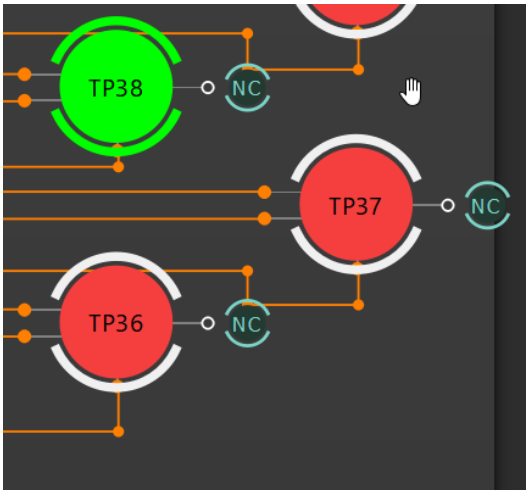


Figure 44. TP Map

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.

Use the context menu on GPIOs with a right click to see the connectivity options (Figure 45).

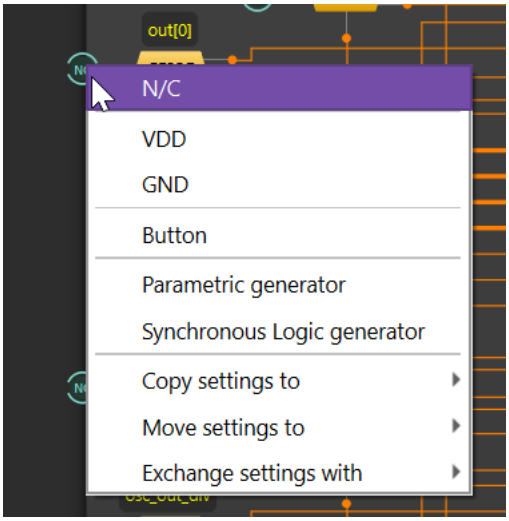


Figure 45. Debug Tool

- **NC (not connected)**

Figure 46. NC (not connected)

- **VDD**

Figure 47. Set to VDD

- **GND**

Figure 48. Set to GND

3.3.1 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 49, Figure 50, Figure 51).

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 50, 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 50).

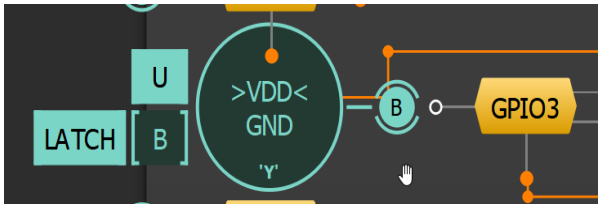


Figure 49. Latched Button with Upper Connection as VDD

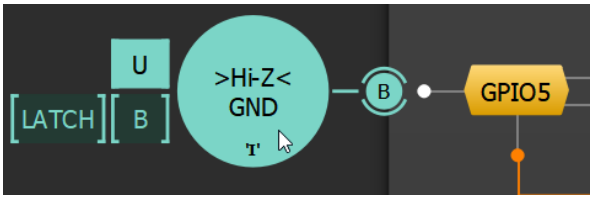


Figure 50. Unlatched Button with Upper Connection as Hi-Z

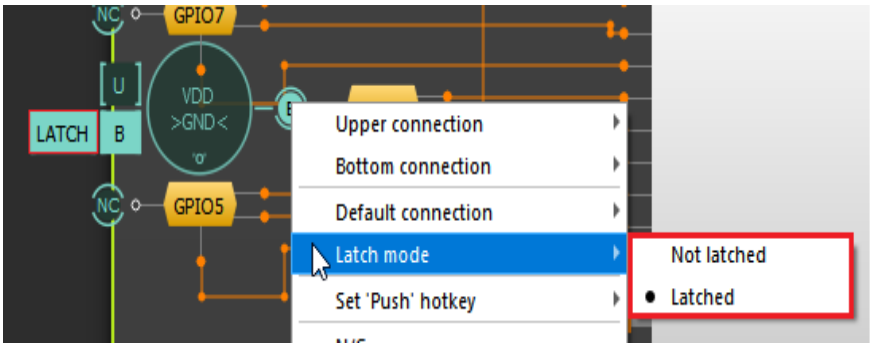


Figure 51. Context menu options for Configurable Button

3.3.2 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.

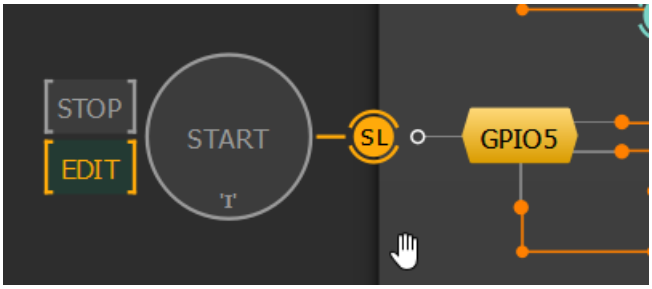


Figure 52. Synchronous Logic Generator

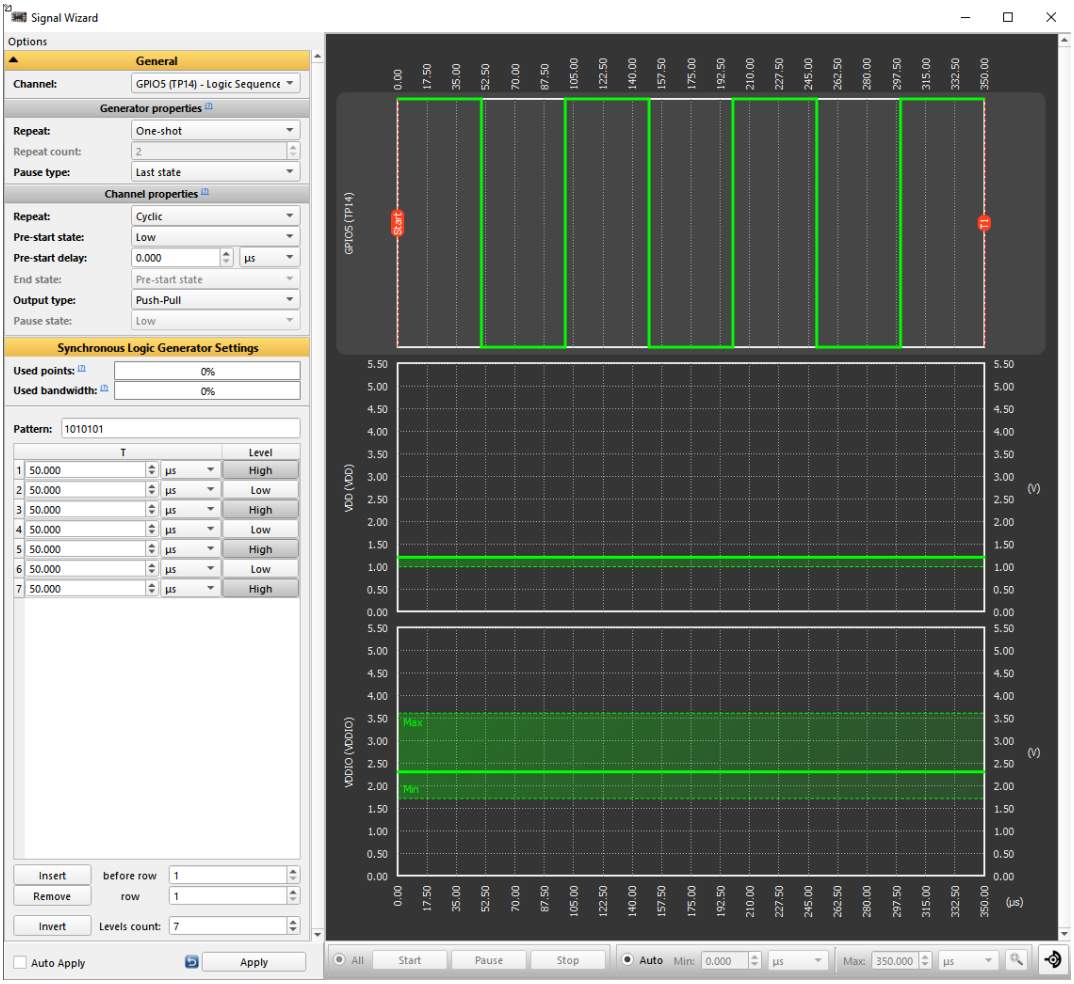


Figure 53. Signal Wizard for Synchronous Logic Generator

Below are the different features in the Signal Wizard for Synchronous Logic Generator (Figure 53).

i. *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.

ii. *Used Bandwidth*: Percentage of used resources needed to successfully execute generator's pattern.

iii. *Pattern*: 0 = low, 1 = high level. We can set the pattern of pulse levels.

iv. *Repeat*: One Shot / Cyclic / Custom.

v. *T / levels*: it sets the duration of each pulse.

vi. *Insert*: To insert pulse before selected position.

vii. *Remove*: remove pulse from the selected position.

viii. *Invert*: to invert the pattern from (For ex. '1010' to '0101').

ix. *Level Count*: to insert the total number of pulses to be generated.

3.3.3 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

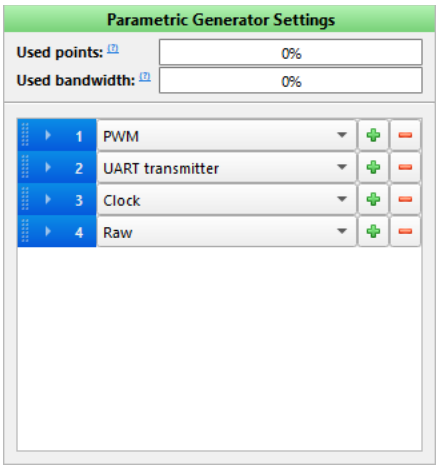


Figure 54. Parametric Generator Command Editor

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

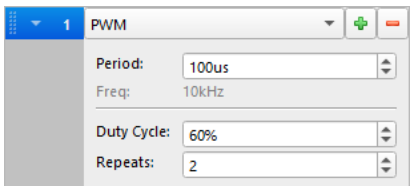


Figure 55. PWM Command Editor

- *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

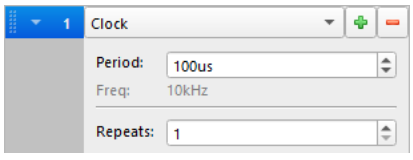


Figure 56. Clock Command Editor

- *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

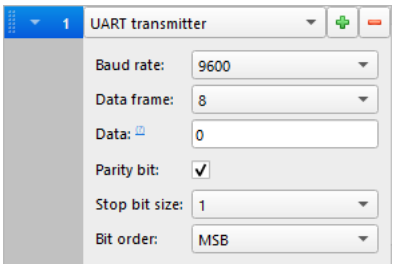


Figure 57. Clock Command Editor

- *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. 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.

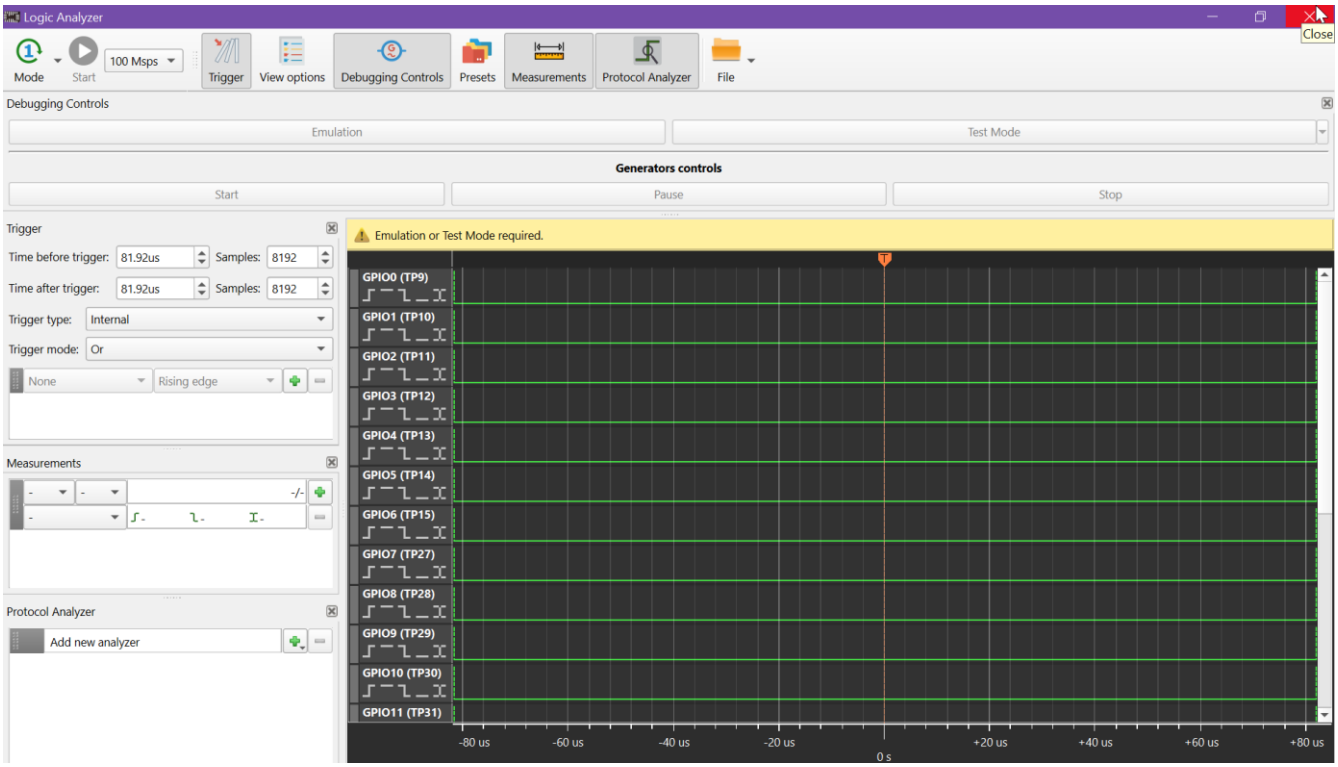


Figure 58. Logic Analyzer

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

3.4.1 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.

3.4.2 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.

3.4.3 Triggers

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

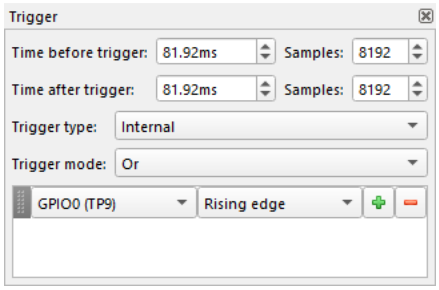


Figure 59. Trigger Parameters

The trigger type can also be configured (see Figure 59):

- *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).

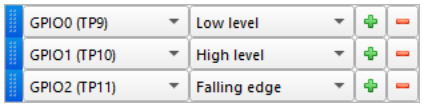


Figure 60. Trigger Conditions

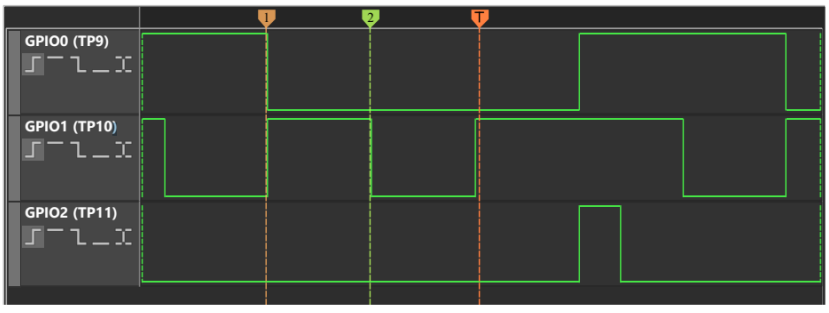


Figure 61. Trigger Configuration

3.4.4 Debugging Controls

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

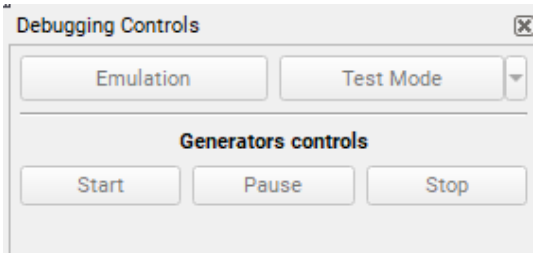


Figure 62. Debugging Controls

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

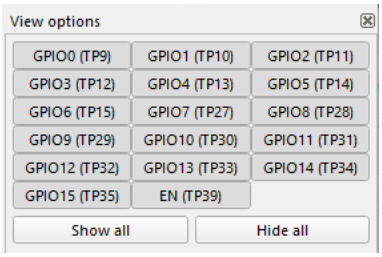


Figure 63. View Options

3.4.5 Presets

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

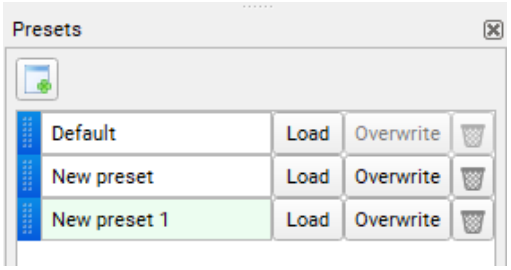


Figure 64. Logic Analyzer Configuration Presets

- *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.

3.4.6 Protocol Analyzer

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

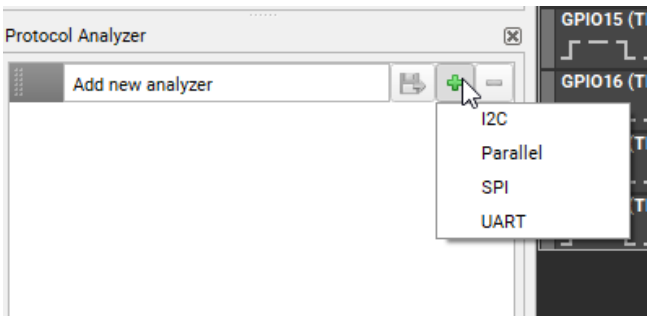


Figure 65. Protocol Analyzer Decode Options

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

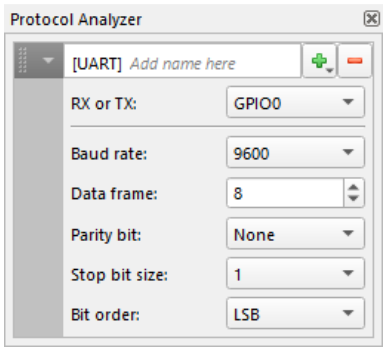


Figure 66. Protocol Analyzer Options

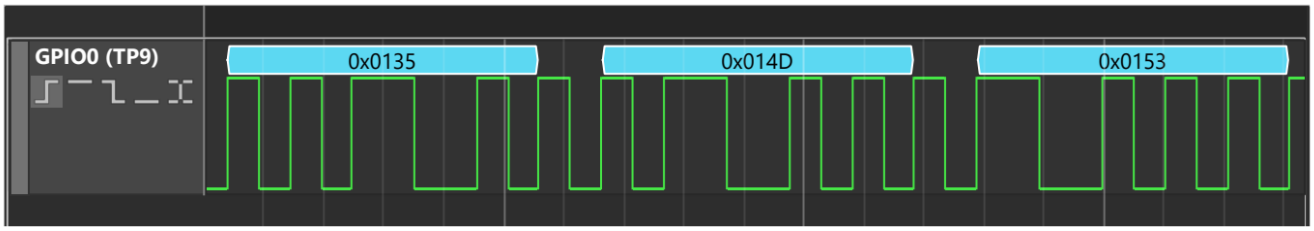


Figure 67. Decoded Data

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.

3.4.7 Import/Export

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

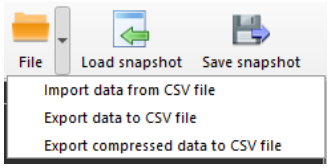


Figure 68. Import/Export from/to CSV format

3.4.8 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.

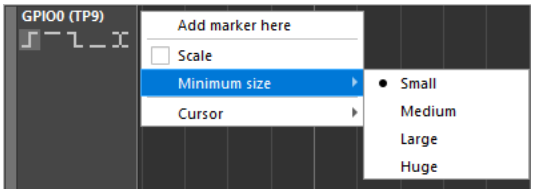


Figure 69. Plot Widget

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

3.4.9 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.

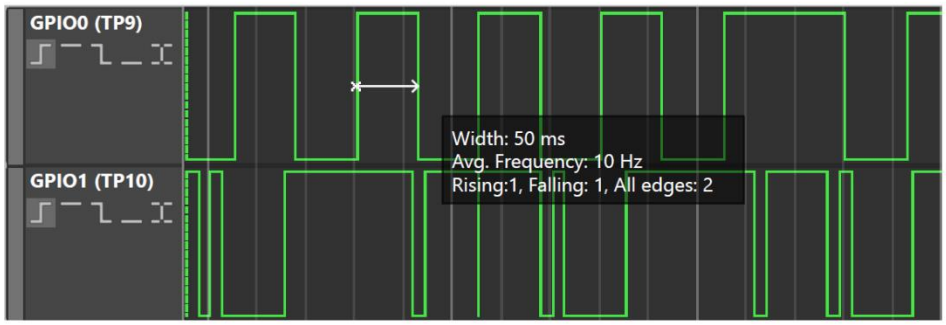


Figure 70. Half Period Cursor

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.

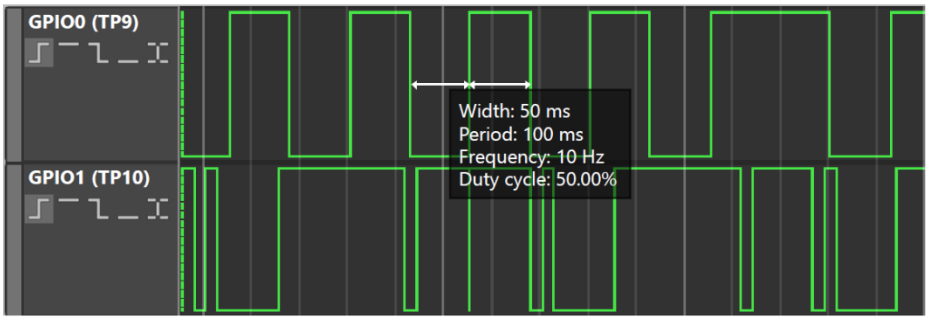


Figure 71. Period Cursor

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.

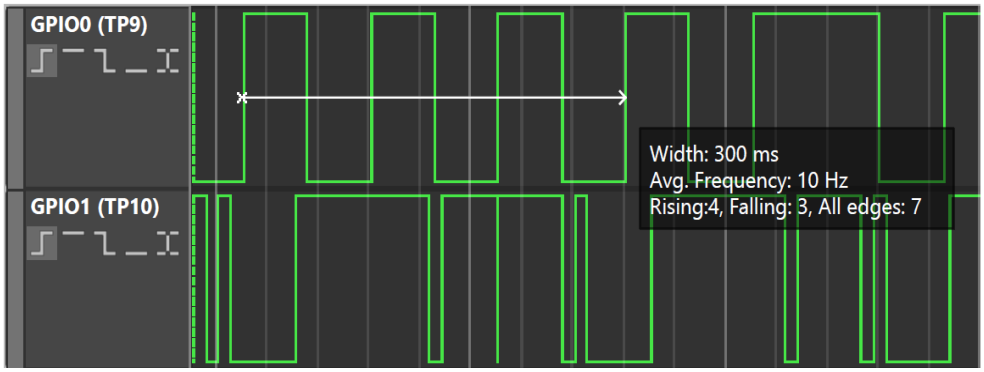


Figure 72. Adjustable Period Cursor for Measurement

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

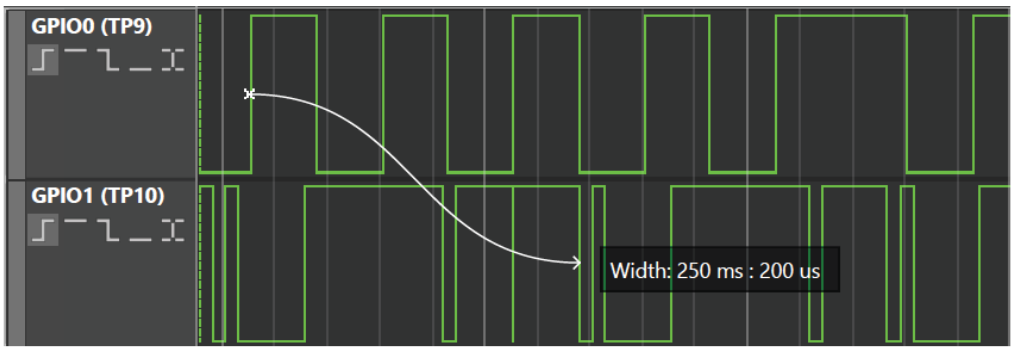


Figure 73. Adjustable Period Cursor Measurement between waveforms

3.4.10 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

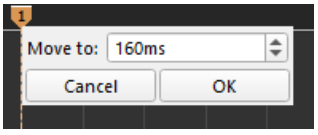


Figure 74. Moving markers with context menu

- 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

3.4.10.1 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.

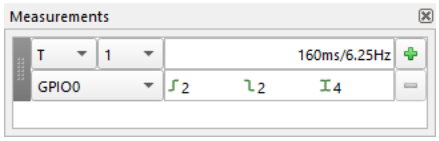


Figure 75. Marker Measurements

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.

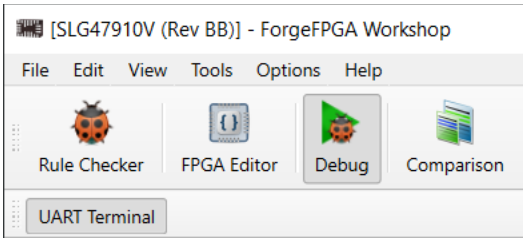


Figure 76. UART Terminal Tool

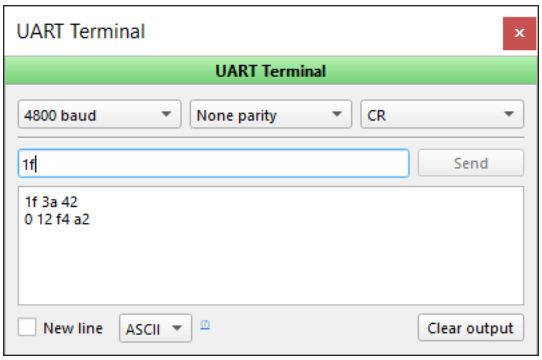


Figure 77. UART Terminal Window

- *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.

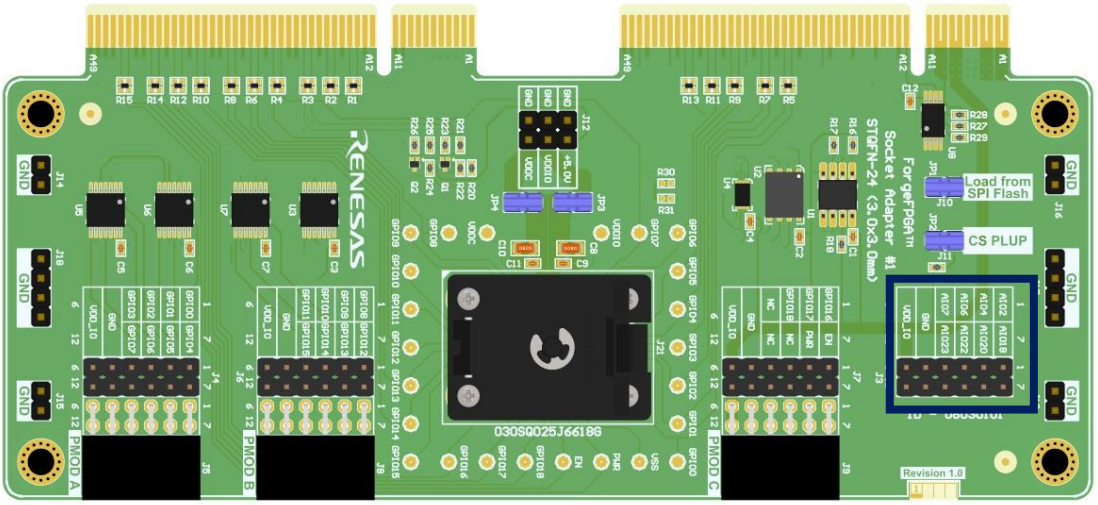


Figure 78. Analog I/O Pins on ForgeFPGA Socket Adapter

Start by selecting the appropriate development platform and opening the DAC Tool from the toolbar.

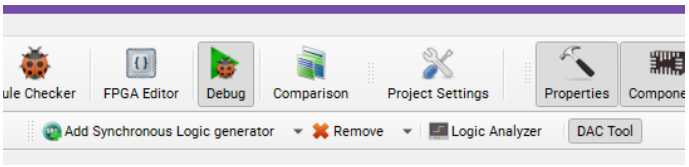


Figure 79. DAC Tool on the toolbar

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 80).

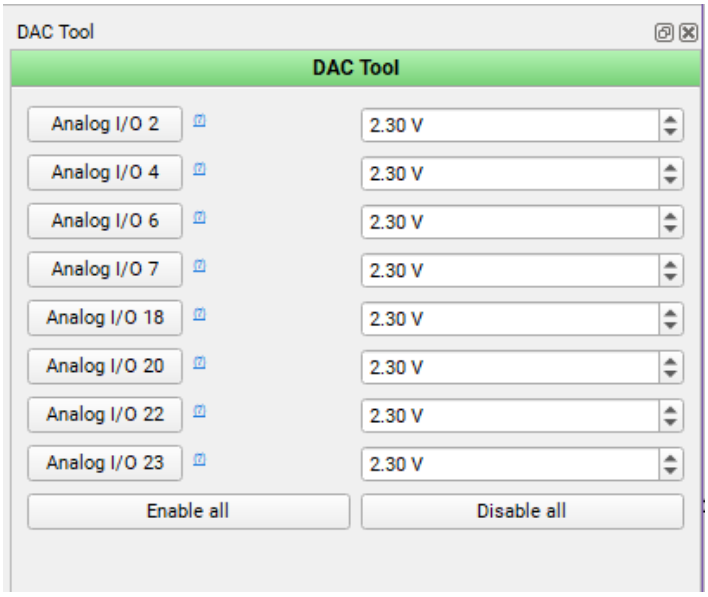


Figure 80. DAC Tool Settings

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

4. Block Properties

The GoConfigure software's ForgeFPGA Workshop displays the FPGA core and its connections to the different blocks on the board such as Oscillator, PLL, BRAM, and GPIO. Each of these blocks have a set of properties that can be set by selecting the appropriate options for each parameter. To see the Properties of each block, click on the Properties tab from the top toolbar. This will open the properties section on the left side of the window. You can read more about each property for each block by clicking the info button .

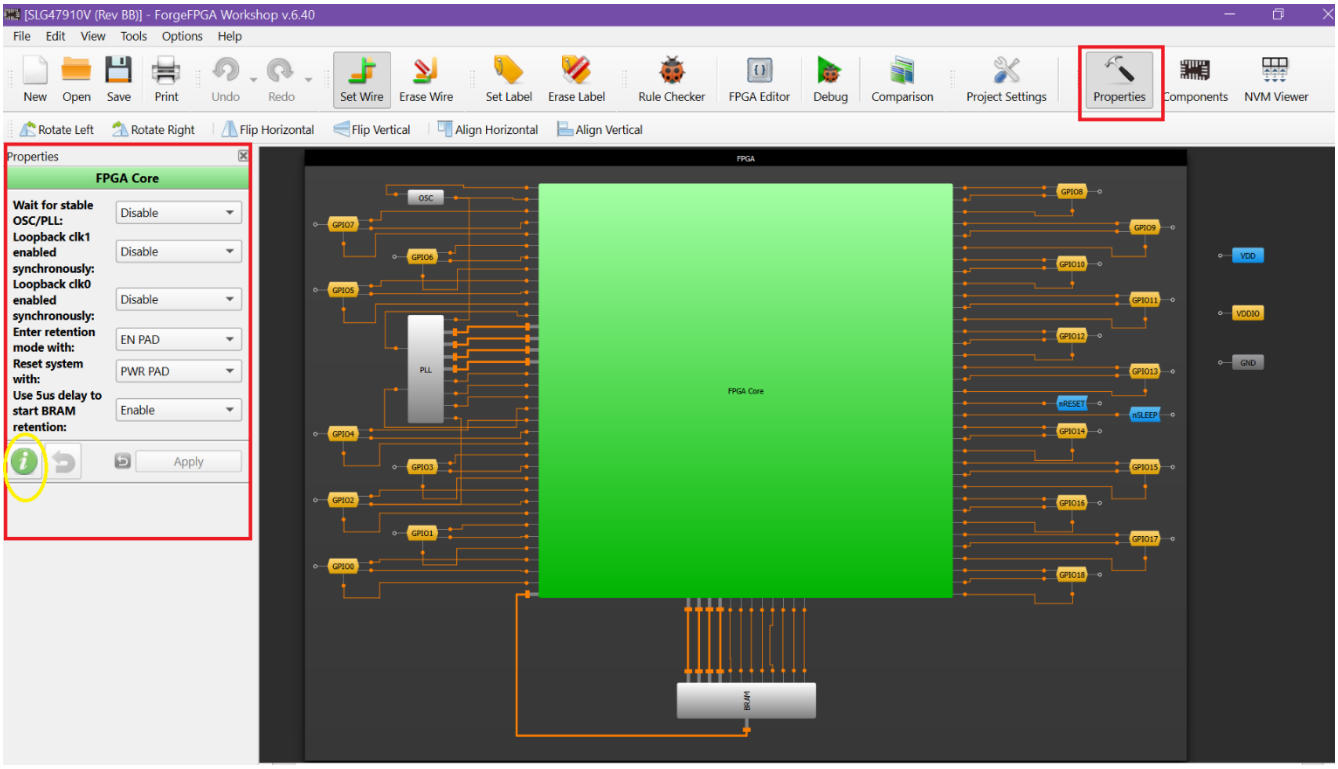


Figure 81. Block Property Editing

4.1 FPGA Core

The properties of the FPGA core can be viewed by simply clicking on the block in the main view. The FPGA Core is the main block that is connected to all the GPIOs, OSC, and the PLL. All connections go though this block, so the properties of FPGA Core affect the other blocks as well.

Depending on the design requirement, the user can select the different properties of different parameters from its respective dropdown option. The parameters are (see Figure 82):

a. Wait for stable OSC/PLL:
   - *Disable*: system will not wait for a stable OSC/PLL clock during the transition from Configuration/Retention mode to Functional mode
   - *Enable*: system will wait for a stable OSC/PLL clock during the transition from Configuration/Retention mode to Functional mode

b. Logic-As-Clock1 enabled synchronously:
   - *Disable*: Logic-As-Clock1 is enabled/disabled asynchronously

c. Enable: Logic-As-Clock1 is enabled/disabled synchronously Loopback clk0 enabled synchronously:
   - *Disable*: Logic-As-Clock0 is enabled/disabled asynchronously
   - *Enable*: Logic-As-Clock0 is enabled/disabled synchronously

d. Enter Sleep mode with:

- *nSLEEP*: Use EN (nSLEEP) pin to enter Sleep mode
- *Core IOB*: Use INT_FPGA_SLEEP to enter Sleep mode

e. Reset system with:

- *nRESET*: Use PWR (nRST) to reset the system
- *Core IOB*: Use INT_FPGA_RESET to reset the system

f. Use 5 µs delay to start BRAM retention:

- *Enable*: 5 µs delay will used to start the RAM Retention sequence and block global clocks
- *Disable*: 5 µs delay will not be used to start the BRAM Retention sequence and block global clocks

g. GPIO Keep

- *Enable*: GPIO pins keep their state when the chip is in reset
- *Disable*: GPIO pins keep their state when the chip is in reset

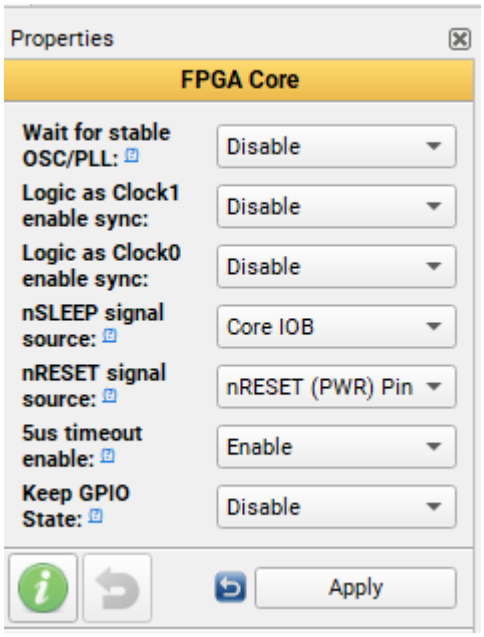


Figure 82. FPGA Core's Properties

4.2 Phase Locked Loop (PLL)

The SLG47910 has one PLL on board which sources its reference clock either from GPIO2 externally or by the 50 MHz OSC. Under the properties tab of the PLL, there is a parameter which can enable the clock to operate asynchronously or synchronously.

4.3 Oscillator (OSC)

A high frequency 50 MHz oscillator on-board is used as a clock to the SLG47910. Under the properties tab for the OSC, there is a parameter which either enables the OSC to operate asynchronously or synchronously.

4.4 Block Read Access Memory (BRAM)

We have eight 4K BRAMs available in the SLG47910 with four BRAMs each placed on the top (North) and the bottom (South) of the FPGA Core. The BRAM has three parameters under its property tab:

a. Memory Retention
   - *Enable* – Enable the memory retention
   - *Disable* – Disable the feature for the BRAM to retain the memory

b. North BRAM Enable
   - *Enable* – User needs to Enable this bit when using the memory from BRAM [0:3]
   - *Disable* – User needs to enable this bit when using the memory from BRAM [4:7]

c. South BRAM Enable
   - *Enable* – User needs to Enable this bit when using the memory from BRAM [4:7]
   - *Disable* – User needs to Enable this bit when using the memory from BRAM [0:3]

4.5 EN (nSLEEP) & PWR (nRST)

The EN pin is used to enter the FPGA Core into retention state and is active high.

The PWR pin is used to reset the FPGA Core and is active low. All GPIOs are in a Hi-Z state when PWR = 0.

4.6 General Purpose Input Output Pin

The SLG47910 contains 24 pins. Of these, 19 are digital GPIOs. The remaining pins are V_{DD}, V_{DDIO}, GND, as well as input only pins for EN (nSLEEP) and PWR (nRST).

The Following Configuration options are available for each GPIO pin:

- Input
- Output, 1x Drive Push-pull
- Output, 2x Drive Push-pull
- Output, 1x Drive Open-drain
- Output, 2x Drive Open-drain

All Input & Output options may additionally include the x1 or x2 Resistor Pull-up option.

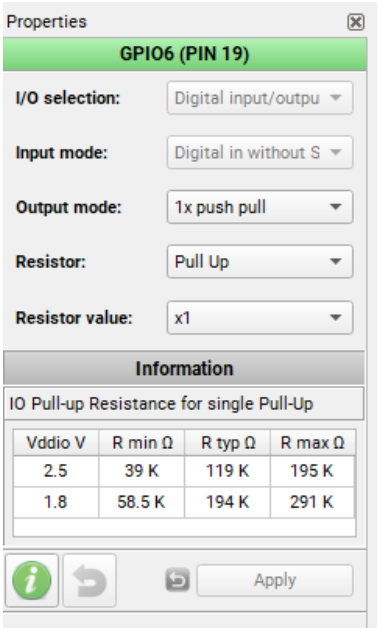


Figure 83. GPIO Properties

The desired configuration can be set from the properties window for each GPIO separately. Similarly, the values for V_{DD} and V_{DDIO} can also be set from under the Property tab.

5. 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 display if a particular function is enabled or disabled.

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

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

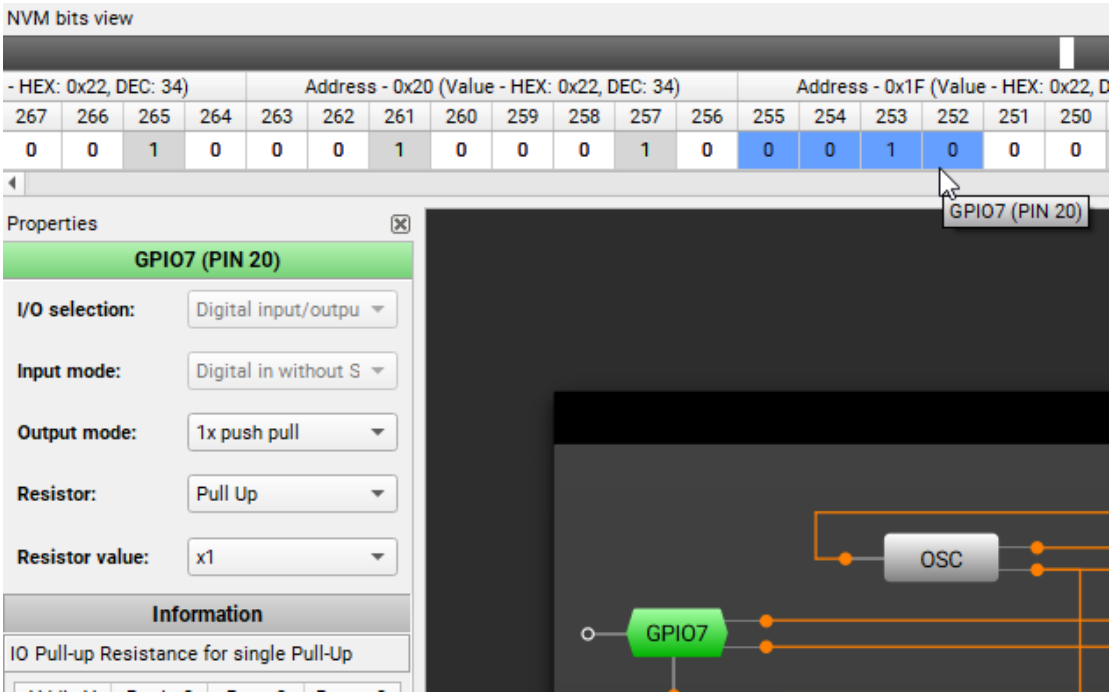


Figure 84. NVM Bits for GPIO7

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 85)

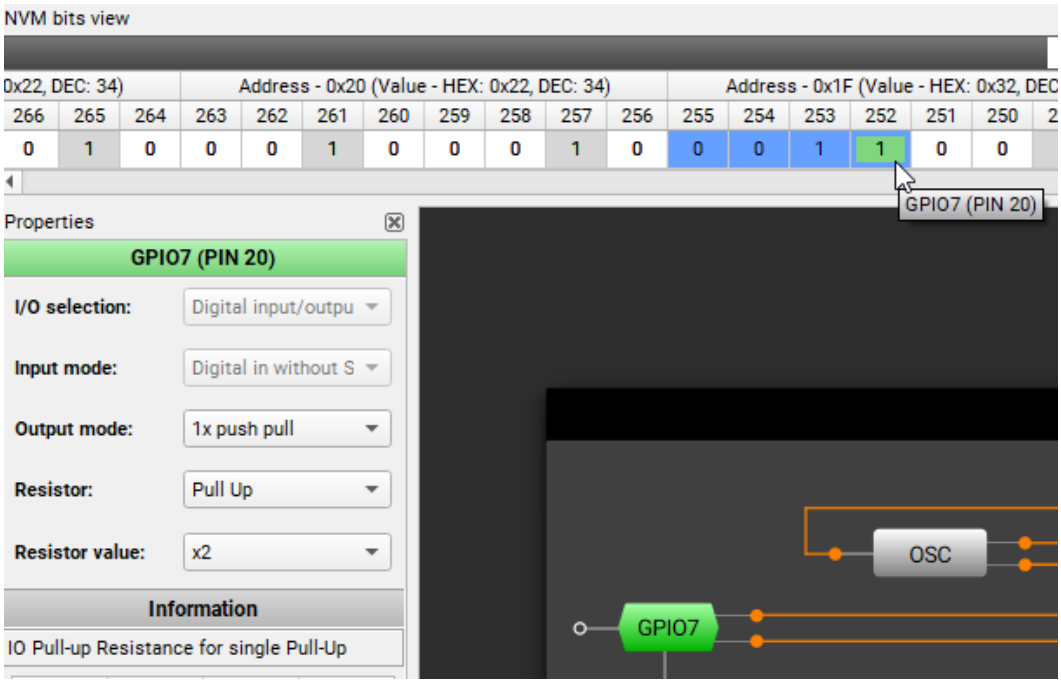


Figure 85. NVM Bits 0011 after the property has been modified

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 Appendix of the SLG47910's Datasheet.

6. IO Planner Signals

Below are few signals that are mentioned in the IO Planner section of the software. The table below explains the functions of a few signals.

Signal Name	Direction	Function
OSC_READY	Input (to the FPGA Core)	Indicates that Oscillator output frequency is stable; gates OSC_CLK while OSC_READY=0
FPGA_CORE_READY	Input	Signal that indicates when FPGA Core has been configured. Goes HIGH before entering Functional Mode; doesn't go LOW in Retention (Sleep) Mode; goes LOW during transition into Reset Mode. Can be used as reset or enable signal for user logic
FUNC_MODE	Input	Signal that indicates that the device is in Functional Mode. When it goes high - user clocks are ungated available for FPGA Core; when it goes LOW - clocks gating is initiated. Signal goes LOW when exiting from Functional Mode and transits to Retention (Sleep) or Reset Mode. This signal can be used as reset or enable signal if logic requires to be reset when not in Functional Mode or to be enabled only when in Functional mode
INT_FPGA_SLEEP	Output (from the FPGA Core)	Internal Sleep start signal - initiates transition to Retention (Sleep) mode from IOB (requires Reg [215] to be set)
INT_FPGA_RESET	Output	Internal Reset start signal - initiates transition to Reset mode from IOB (requires Reg [216] to be set)
REF_LOGIC_AS_CLK0	Output	Logic as Clock 0 Reference signal - output signal that is looped through LaC circuit and returned to core as clock
REF_LOGIC_AS_CLK1	Output	Logic as Clock 1 Reference signal - output signal that is looped through LaC circuit and returned to core as clock
LOGIC_AS_CLK0_EN	Output	Logic as Clock 0 Enable (active HIGH)
LOGIC_AS_CLK1_EN	Output	Logic as Clock 1 Enable (active HIGH)
LOGIC_AS_CLK0	Input	Logic as Clock 0
LOGIC_AS_CLK1	Input	Logic as Clock 1
REF_BRAM (0..3)_READ_CLK	Output	BRAM Slices 0..3 Read Clock
REF_BRAM (0..3)_WRITE_CLK	Output	BRAM Slices 0..3 Write Clock
REF_BRAM (4..7)_READ_CLK	Output	BRAM Slices 4..7 Read Clock
REF_BRAM (4..7)_WRITE_CLK	Output	BRAM Slices 4..7 Write Clock
GPIOX_OUT	Output	GPIOX Output value
GPIOX_OE	Output	GPIOX Output Enable (active HIGH)
GPIOX_IN	Input	GPIOX Input value

7. 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 • Updated PLL Calculator • Updated Project Directory Folder • Changed Advanced Dev Board name to Deluxe Dev Board • DAC Tool • Updated Block Properties

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

1. 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.

2. 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.

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

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

4. 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.

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

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

6. Multiple conflicting drivers for <N>: <K>

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

7. Wire <N> is used but has no driver.

   >>No driver was assigned to a wire.

8. 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.

9. 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.

10. 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.