ForgeFPGA Workshop v6.55

Introduction

This document provides instructions for using the ForgeFPGA™ Workshop software v6.55.

Contents

Figures

Tables

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™, AnalogPAK™, 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 design created.
- Read a programmed part and import its data into the software.
- Run simulations with external components.

1.1 Getting Started

To create and debug a new design with ForgeFPGA first download and install the Go Configure Software Hub from Go Configure Software Hub. The Hub window is organized into the following primary tabs:

- **Home** – the starting page, designed for project initialization and providing access to recent work.
  - **Open my file** and **Start new project** – provide options to open an existing project or initiate a new project on the supported part number.
  - **Example projects** – offers a list of design examples. Clicking **Show more** redirects to the **Library** tab.
  - **Recent** – displays a list of previously opened projects.
  - **Recovered**– lists the restored projects files after a crash or freeze.

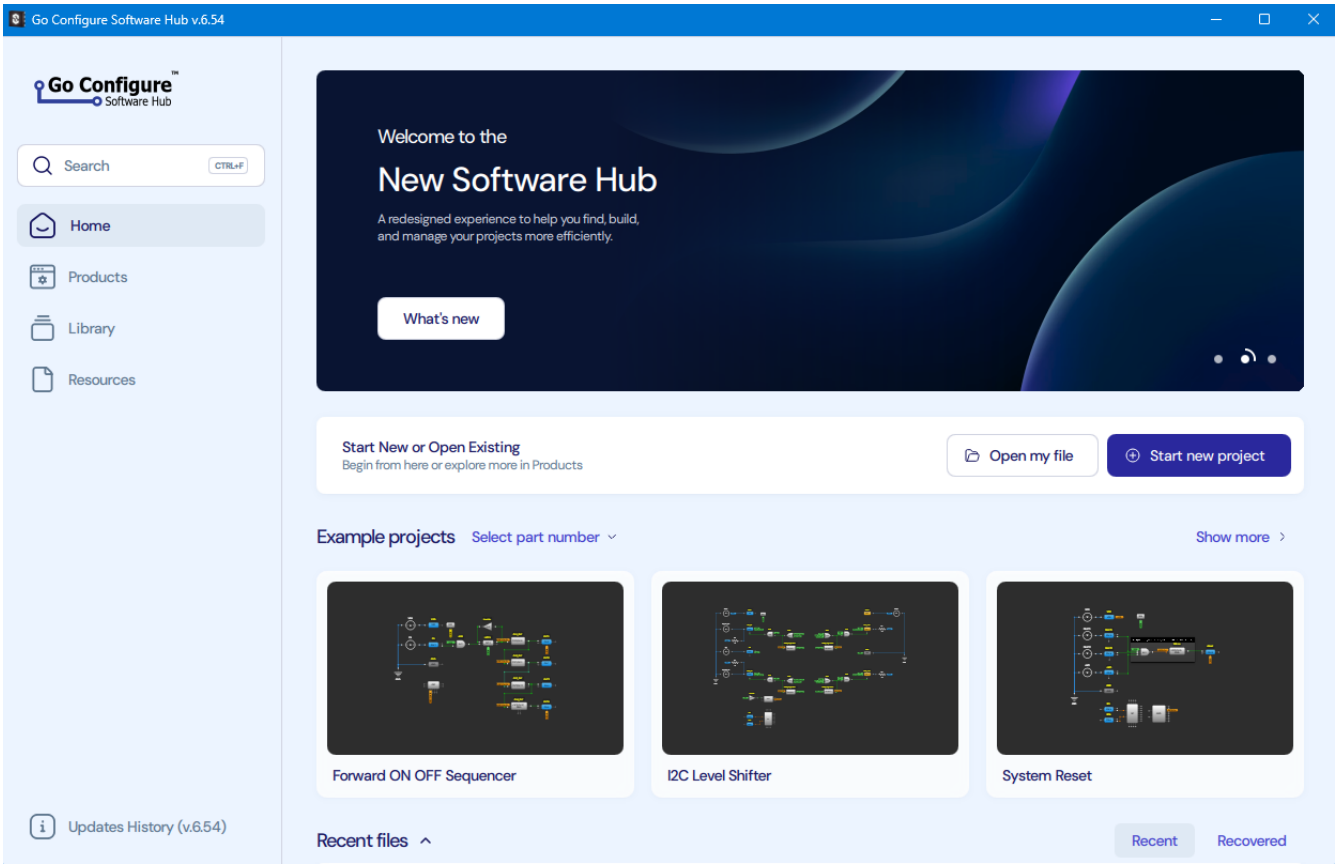

Figure 1. Home Tab of the Hub Window

- **Products** – a searchable database of all supported part numbers.
  - **Product table** – introduces available part numbers, their key characteristics, and links to datasheets.
  - **Chip information panel** – selecting a part number provides detailed characteristics, supported platforms, and relevant technical documentation.

- **Filtering and sorting** – filter and sort by product family (ForgeFPGA in this case). Click the table headers to sort the list by V_{DD}, temperature, special features.

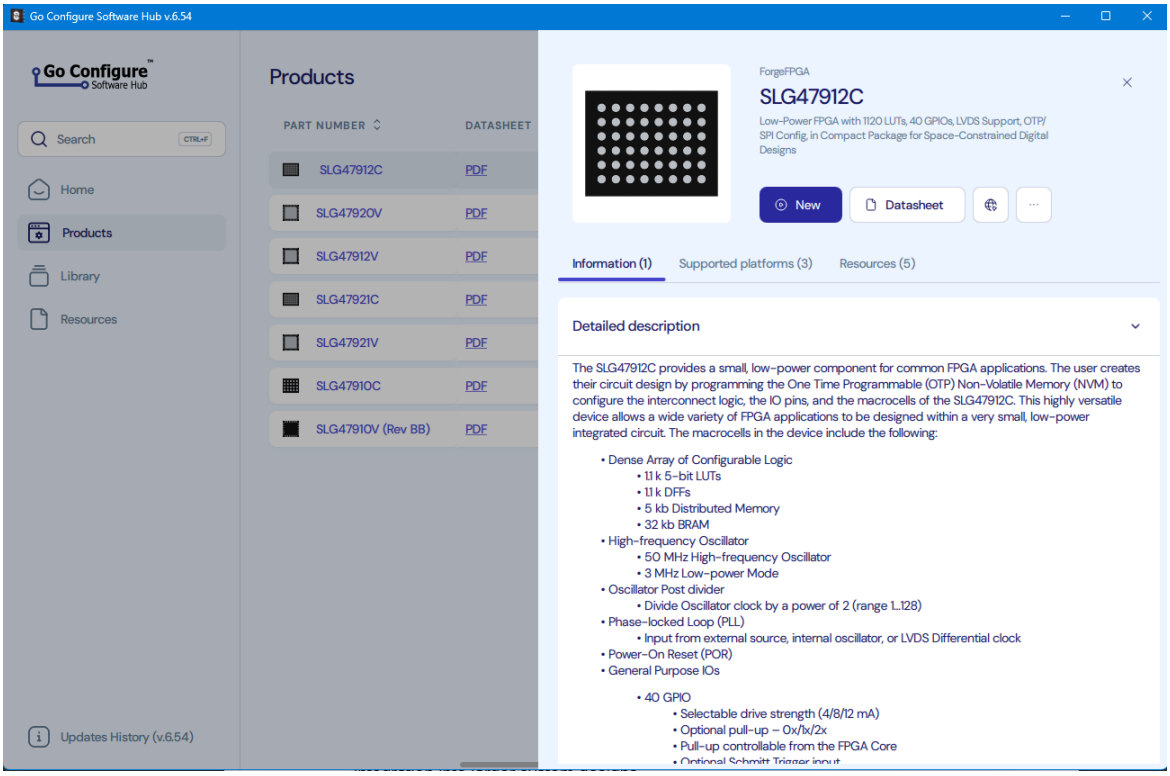

Figure 2. ForgeFPGA Tab of the Hub Window

- **Library** – a repository containing design assets, reference designs, and debugging resources. This is essentially a collection of all top-level documents used to build the designs.
  - **Example projects** – contains modular design blocks or partial project segments for integration into larger system designs.
  - **Application notes** – provides complete, ready-to-use design examples, including design descriptions and preconfigured circuit projects.
  - **Demo projects** – a collection of projects configured for use with specific demonstration boards for hardware debugging and validation.

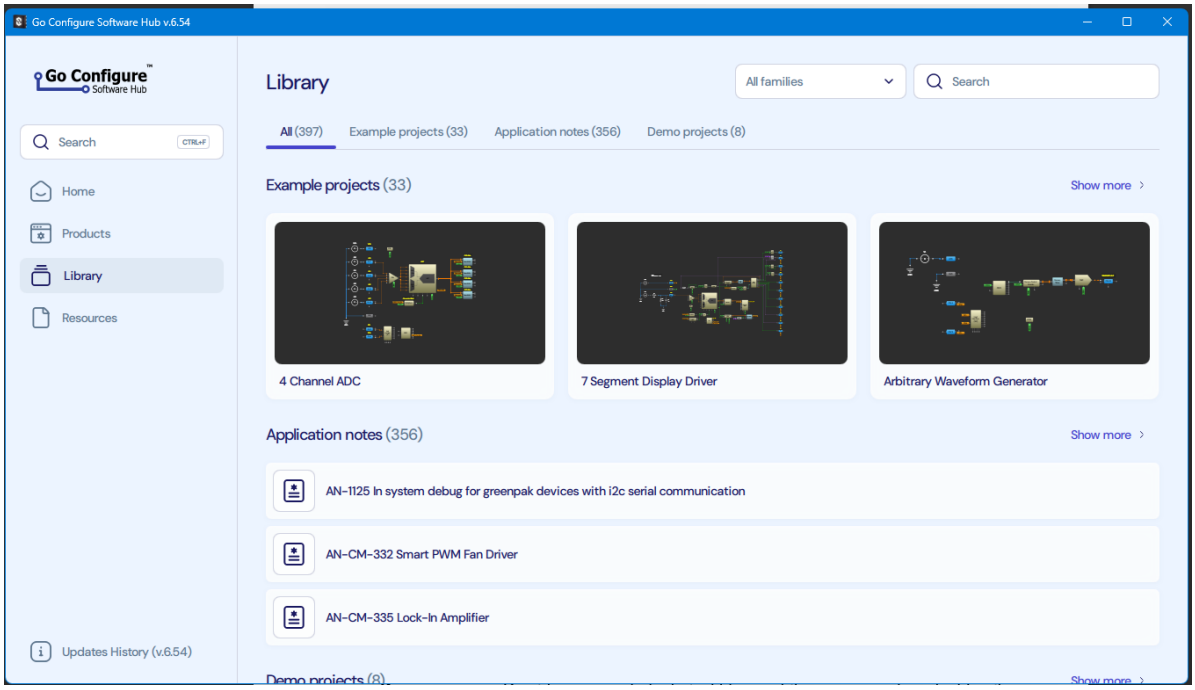

Figure 3. Library Tab of the Hub Window

- **Resources** – space for assets, useful links, documentation, and support content, helpful for finding both information and contacts to get assistance.
  - **Explore our products** – directs users to Renesas website for information on specific product lines.
  - **Key sources** – provides access to technical blogs, white papers, and product family overviews.
  - **Renesas support** – allows to reach the knowledgebase, technical support team, distributor network locations, and additional technical resources.
- **Search** – located in the upper-left corner, this field allows for applications-wide searches across part numbers, datasheets, product families, example projects, application notes, and demo projects.

1.2 Support

For more information about the Go Configure Software Hub, online support is available at www.renesas.com. Click here to gain access to the knowledgebase, technical support team, distributor network locations, and additional technical resources.

The Go Configure Software Hub can be found on Renesas social media. To visit the page, go to **Help → Social** in the main menu of the software instance. For ForgeFPGA product specific assistance, use fpga@renesas.com.

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

2. Forge FPGA Workshop

The ForgeFPGA Workshop appears in the main software window. 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).

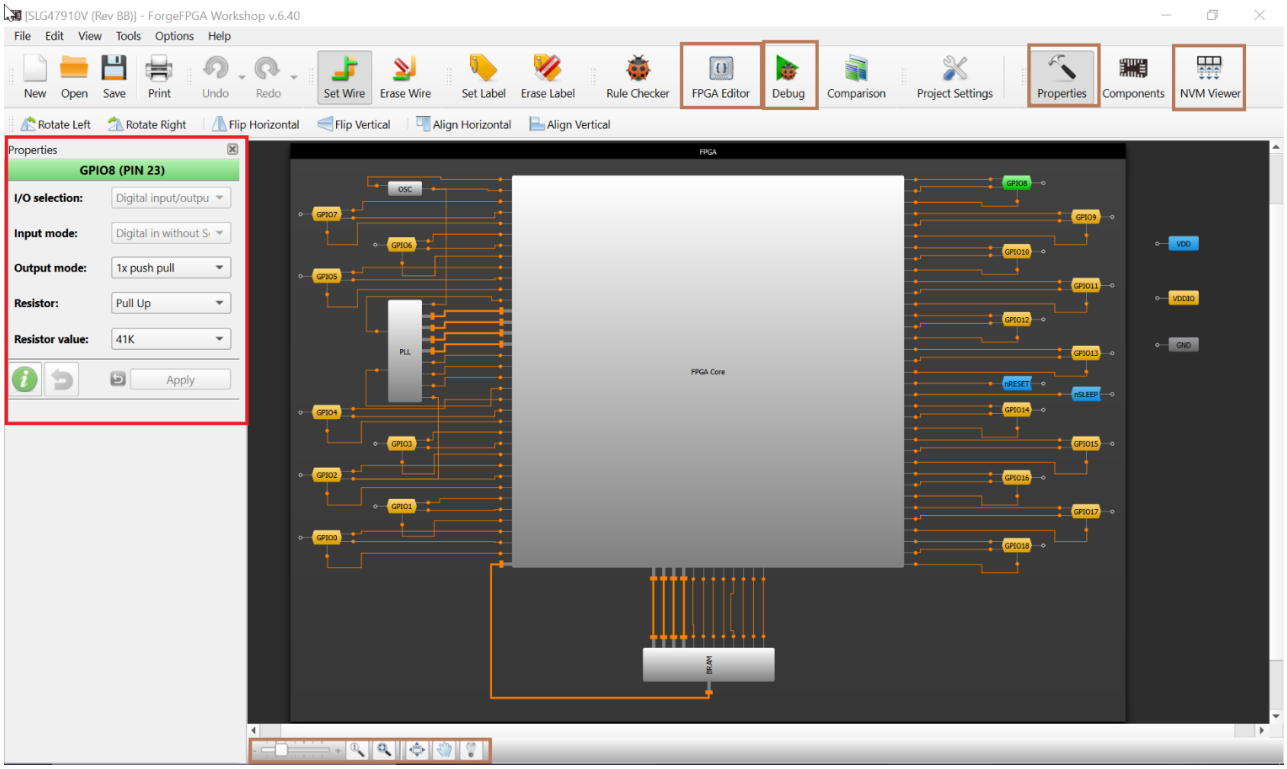

Figure 4. ForgeFPGA Workshop for SLG47910

This window's arrangement is dependent to the device selected. Figure 4 shows the layout for the SLG47910. A different setup of blocks will be shown for the SLG47912, SLG47921, and other devices depending on its pin assignment.

2.1 FPGA Editor

The **FPGA Editor** is a complex software tool designed to create and configure FPGA logic. The editor allows designs for Field-Programmable Gate Arrays (FPGAs) to be created using Verilog, a Hardware Description Language (HDL). Before a design is programmed onto the FPGA, the editor provides an option to simulate it first and verify that it is correct.

The FPGA Editor also includes a Macrocell constructor as an alternative to writing Verilog code. The required blocks can be placed manually on the work area, connected as needed, and the completed design can then be converted into its code representation.

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 additional features in this document.

2.1.1 Flowchart

The flowchart in Figure 5 provides a step-by-step guide for creating a design with the ForgeFPGA software.

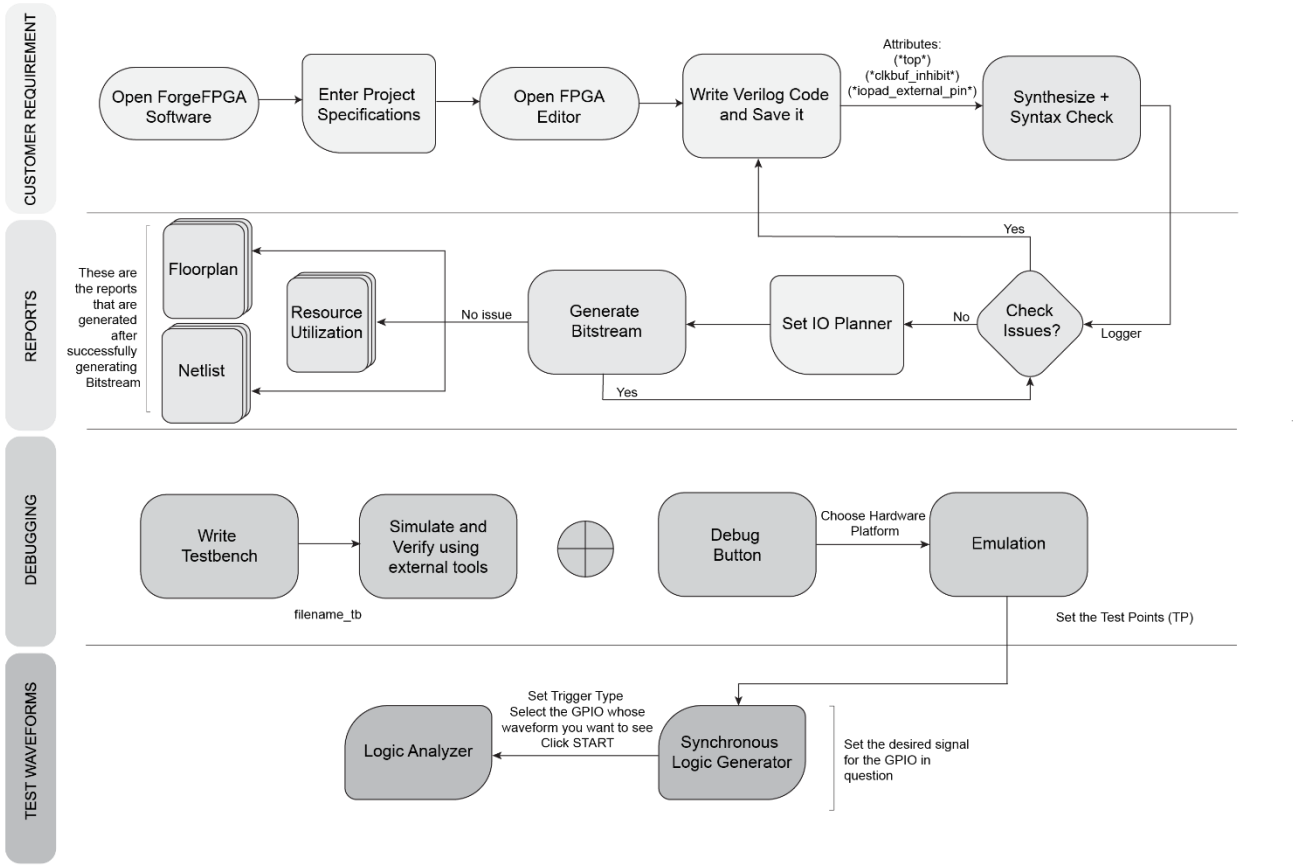

Figure 5. Toolchain Flowchart

The development software consists of the main menu, toolbar, main work area, logger panel, and control panel (see Figure 6).

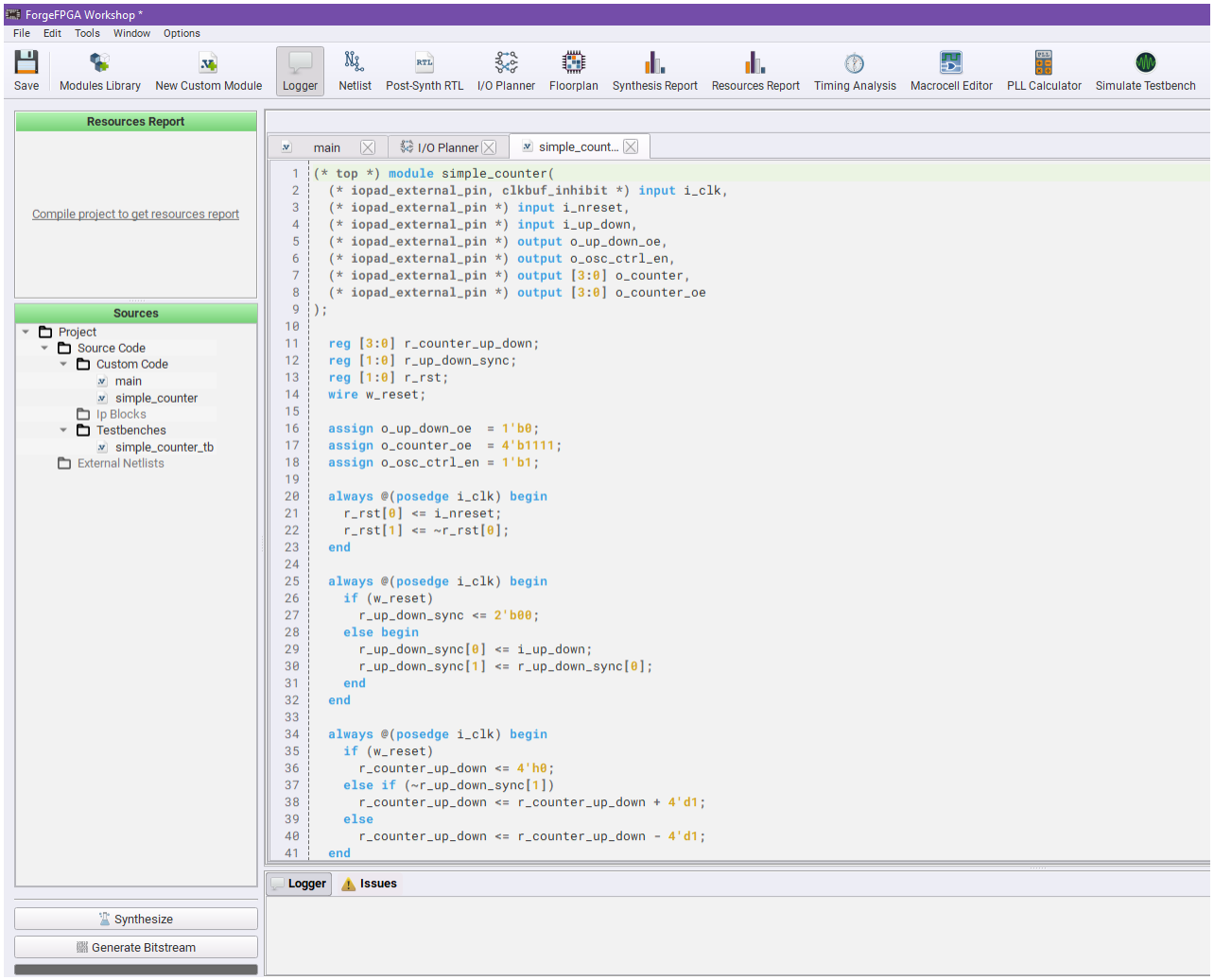

Figure 6. ForgeFPGA Workshop User Interface

2.1.2 Main Menu

The following Main Menu options are listed below with descriptions as needed:

- **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
  - New Placement Constraints – add .pdc 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

2.1.3 Toolbar

The top toolbar provides quick access to a set of tools and actions. For descriptions of the available tools, see the following sections. More information about the tools is also provided throughout the document.

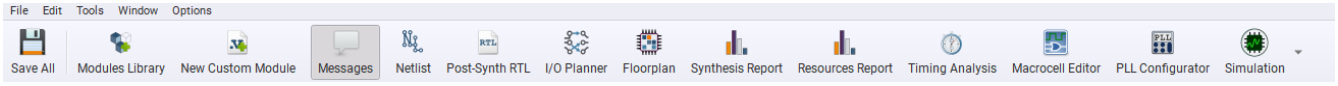

Figure 7. ForgeFPGA Workshop Toolbar

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** button to split screen (see Figure 8), 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.

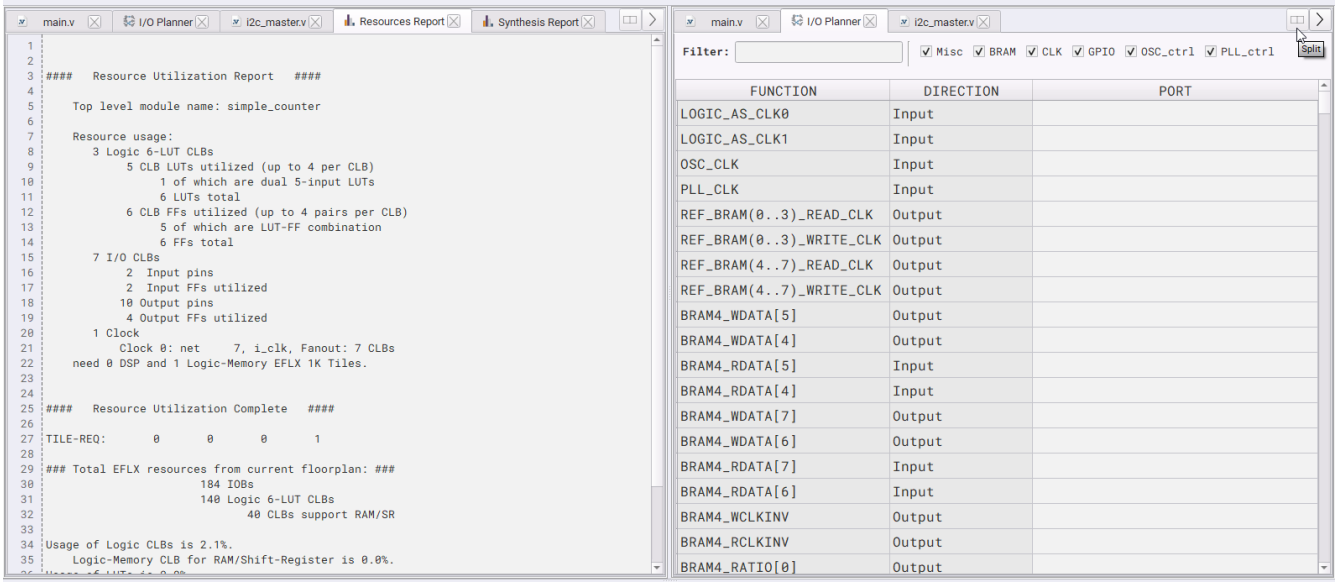

Figure 8. Work Area

2.1.5 Messages

Here the generated Messages appear after the Synthesize, Generate Bitstream, and Simulation procedure (see Figure 9).

- **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 a syntax error is received, right-click on it and select **Show in Editor** to highlight the line in the HDL code where the mistake was detected.

Figure 9. Messages Panel

2.1.6 Control Panel

From the left control panel, the following information can be accessed (see Figure 10):

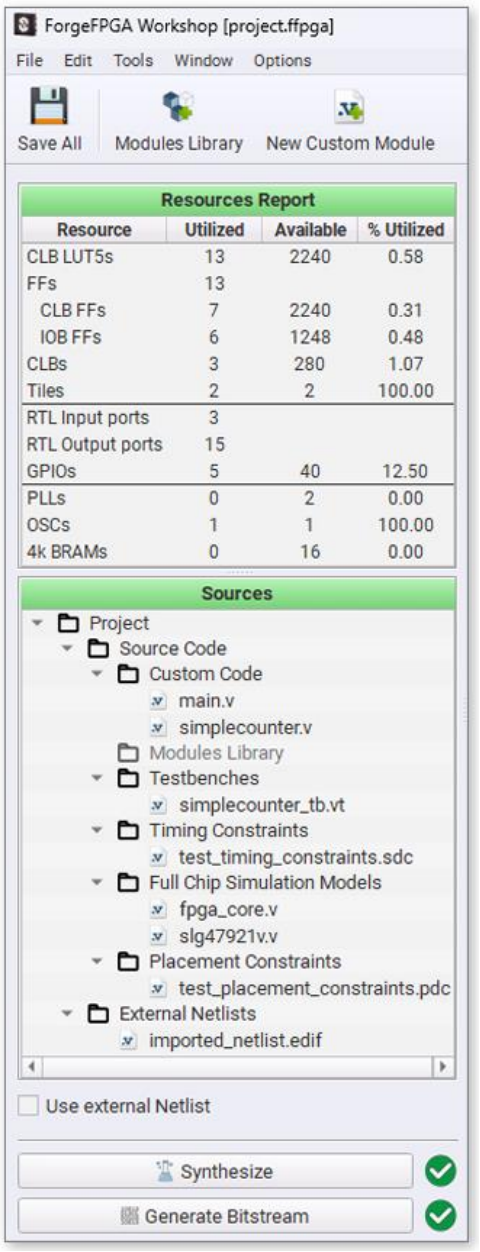

Figure 10. Control Panel

- **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 other supported files that project may contain. A module can be added using the toolbar by clicking **New Custom Modules** or by selecting one from the **Modules Library** (see section 2.3 Modules Library). All supported sources can also be added via **Main Menu → File**. Additionally, external sources can be imported by going to **File → Import** (see section 2.2.1 Importing/Export RTL Files). Find the following subcategories here:
  - 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 the 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.

Note: Make sure all modified modules are saved before running procedures so that the most up-to-date data is processed. Configure the saving options in Settings, Processing tab (see 2.1.7 Settings).

2.1.7 Settings

The user can access the settings from **Main Menu → Options → Settings**. The user can change various settings for the project and change some user settings in this GUI (see Figure 11).

- **Project Settings**
  - **Synthesize** – a process where the HDL code is compiled and converts a high-level description down to lower-level description, for example logic gates. Select the tool version desired.
    - **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.
    - **Autoname** – assign meaningful names to the signals instead of using $auto$identifiers.
    - **No FSM** – disable FSM optimizations. Useful when preserving the original control logic or signals is required, as FSM transformations may merge or remove intermediate signals.
    - **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.
  - **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 utilization I/O but can worsen congestion and timing.
    - **High Density Packing Logic** – can 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. Setting this option to ‘1’ ensures identical Bitstreams when using the same input data.
    - **Maximum number of timing analysis paths** – maximum number of timing analysis paths to report. If – ‘1’, all paths are reported.
    - **Timing Analysis Corner** – select the corner lot that will be used during Static Timing Analysis from typical, SS and FF corners.
    - **Additional arguments** – add additional arguments to the Generate Bitstream procedure.
  - **Simulation** – the user can choose to save their dump 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 is 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** – the user can choose to opt for erasing the **Synthesis Log** and **Bitstream Log** each time they run a new procedure by checking the box.
  - **Tools**:
    - **Icarus Verilog** – the path to the Icarus Verilog tool. The path to the simulation engine binary can be set so it can be found during the simulation procedure if the system environment does not 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. The path to the simulation results viewer binary is set so it can be found after the simulation procedure if the system environment does not 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 (see section 2.4.1 I/O Planner) table such as position, description, and coordinates of the tile.
  - **Processing** – checking the box assists in saving 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 a switch is made 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 the design.

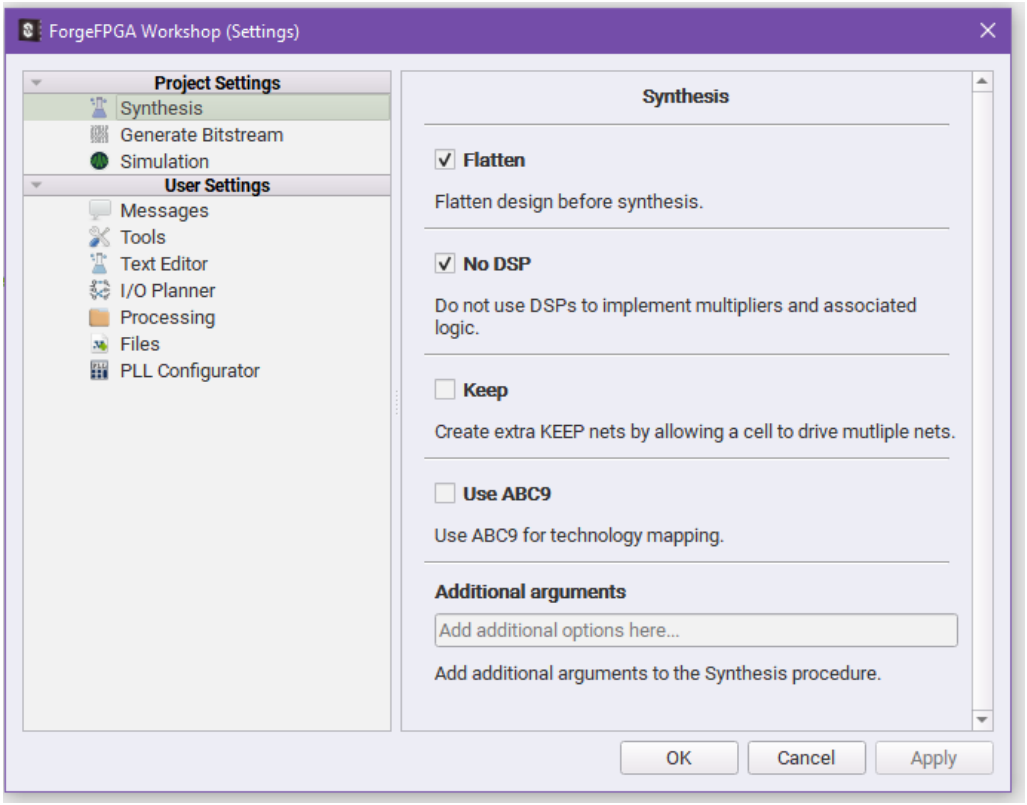

Figure 11. ForgeFPGA Workshop Settings

2.1.8 Design Template

The Design Template offers pre-built designs that can be seamlessly integrated into the project. It can be found under **FPGA Editor → File → Load Design Template** (see Figure 12).

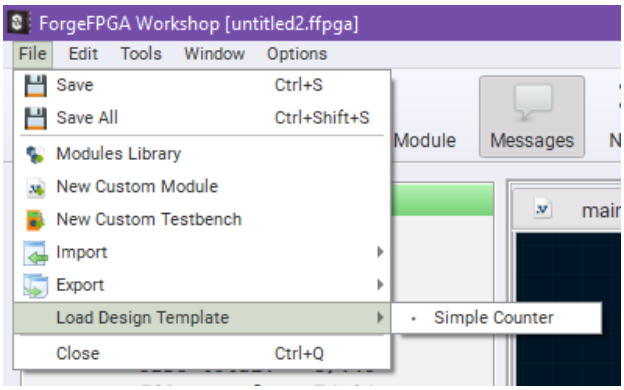

Figure 12. FPGA Design Template

Upon selecting a design template, the Design Template Wizard will appear (see Figure 13). It will guide the user 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**

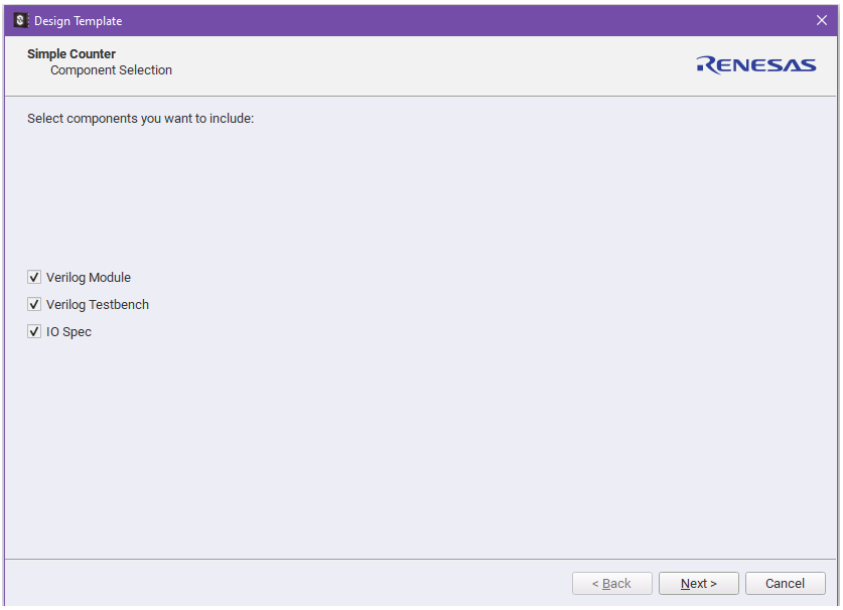

Figure 13. Design Template Component Selection

All components can be selected simultaneously or individually, depending on the desired outcome for the project. The choices made at this stage determine the next steps in the process.

- **IO Spec Diff** – 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 the current port configurations could potentially be overwritten by those of the chosen template. The software highlights conflicting entries in yellow. It Is possible to focus solely on conflicts by selecting the **Show Conflicts Only** button option. However, if the IO Specs are not reviewed, the software streamlines the process by skipping the IO Spec Diff (see Figure 14). In this scenario, the tool proceeds directly to Module name.

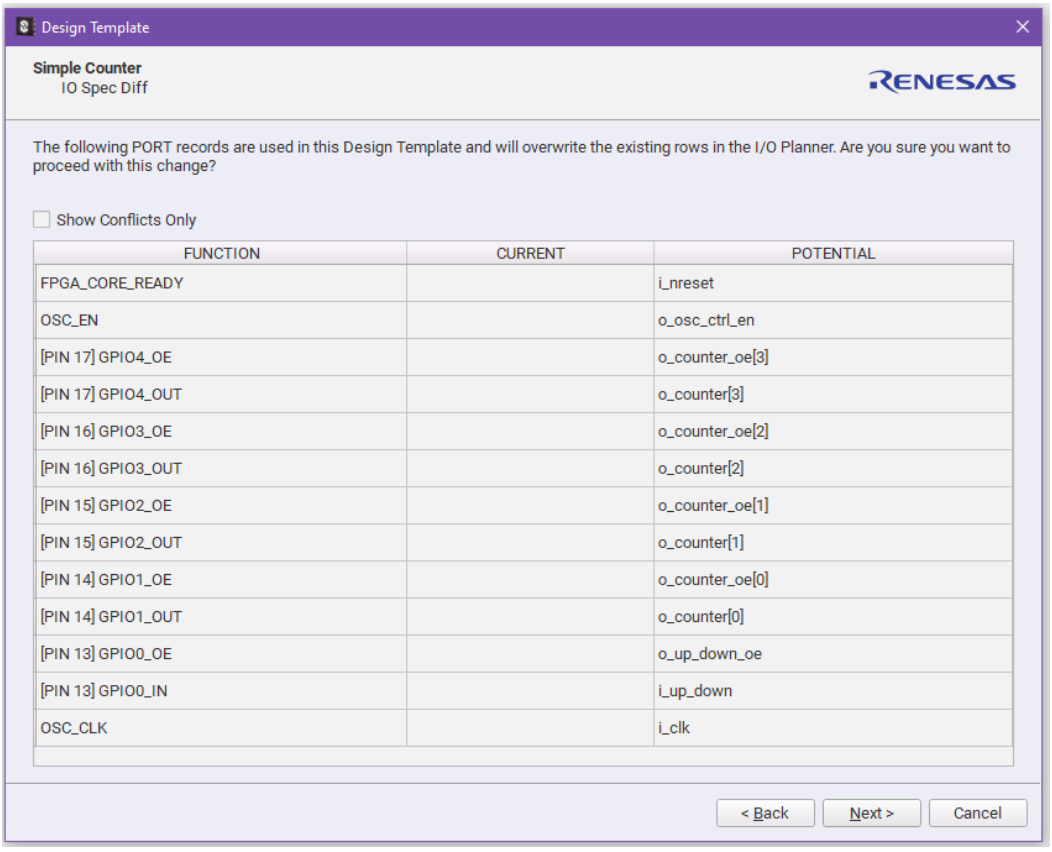

Figure 14. Design Template IO Spec Diff

- **Module Name** – a component that assists in assigning names to Verilog Module and Testbench. If the Verilog Module and Verilog Testbench options are skipped in the Component Selection window, default names are assigned.

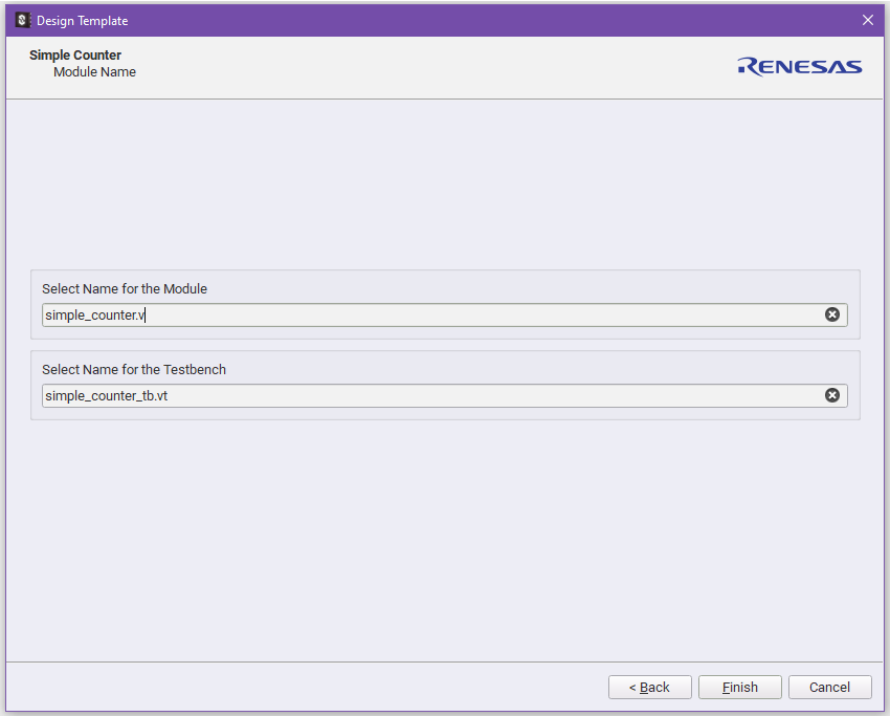

Figure 15. Design Template Module Name

When 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.

When the integration of the design template is done, its functionality can be verified 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 the 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, see Figure 16).
- `(* 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, see Figure 16).

```
// Sample 4-bit counter

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

  reg [3:0] r_counter_up_down = 4'b0000;
  reg [1:0] r_up_down_sync = 2'b00;
  reg [2:0] r_rst = 3'b000;
  wire      w_reset;

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

Figure 16. 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.

2.2.1 Importing/Export RTL Files

External RTL files and Netlist files can be imported in the ForgeFPGA Editor that has been created by other software. A Custom Module file, Custom Testbench file can also be imported. The RTL File can be imported by navigating to **Main Menu** → **Import** → **Custom Module** (see Figure 17).

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)

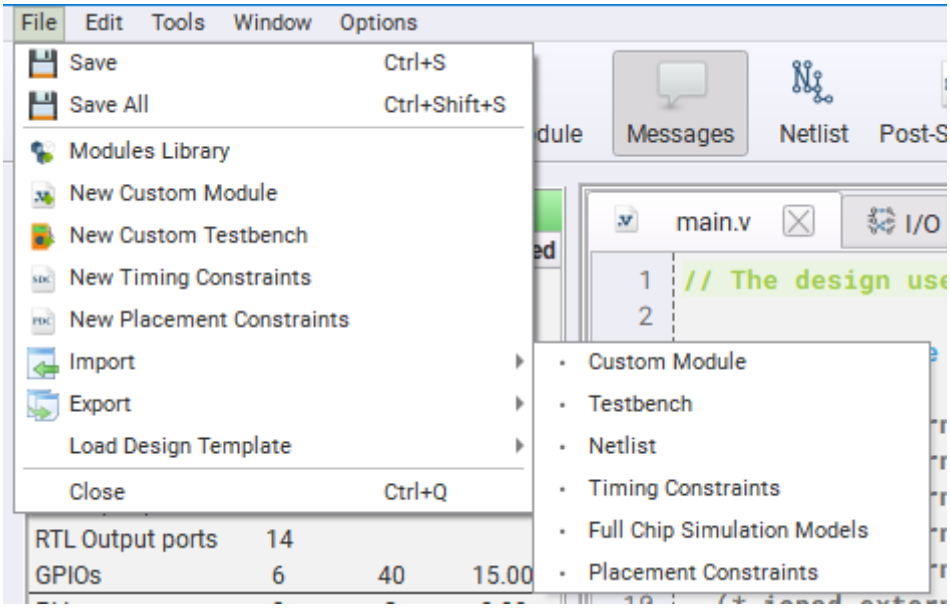

Figure 17. Importing RTL Files

Similarly, files can also be exported (see Figure 18). The Custom Module, Library Module, Testbench, or a Timing Constraint file that is designed using ForgeFPGA Workshop can also be exported to be saved. Navigate to **Main Menu → Export → Custom Module** path to export files.

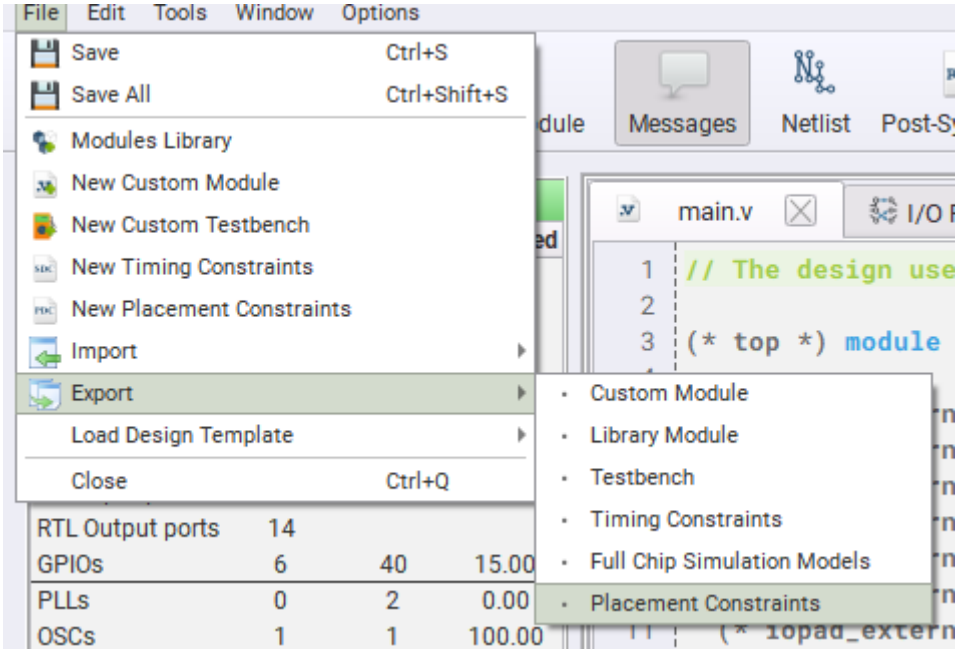

Figure 18. Exporting RTL Files

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 (see section 2.6 Simulation).

To open the Modules Library, click the corresponding **Modules Library** 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 19).

Inside the Modules Library GUI, the schematic and port information is found along with the description of the selected block. The GUI gives a detailed explanation of all the input and output pins of the block which can further be changed as per the requirement. This provides the flexibility to create the HDL code of the module needed and its associated testbench with just a few clicks.

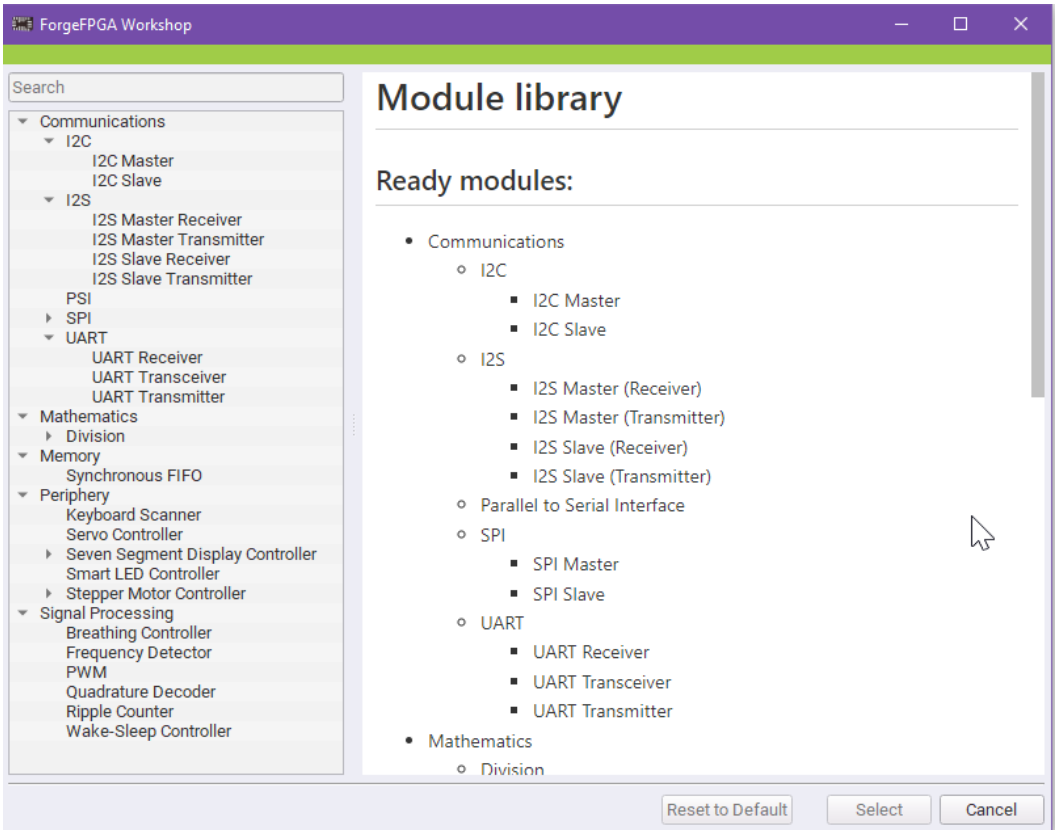

Figure 19. Modules Library GUI

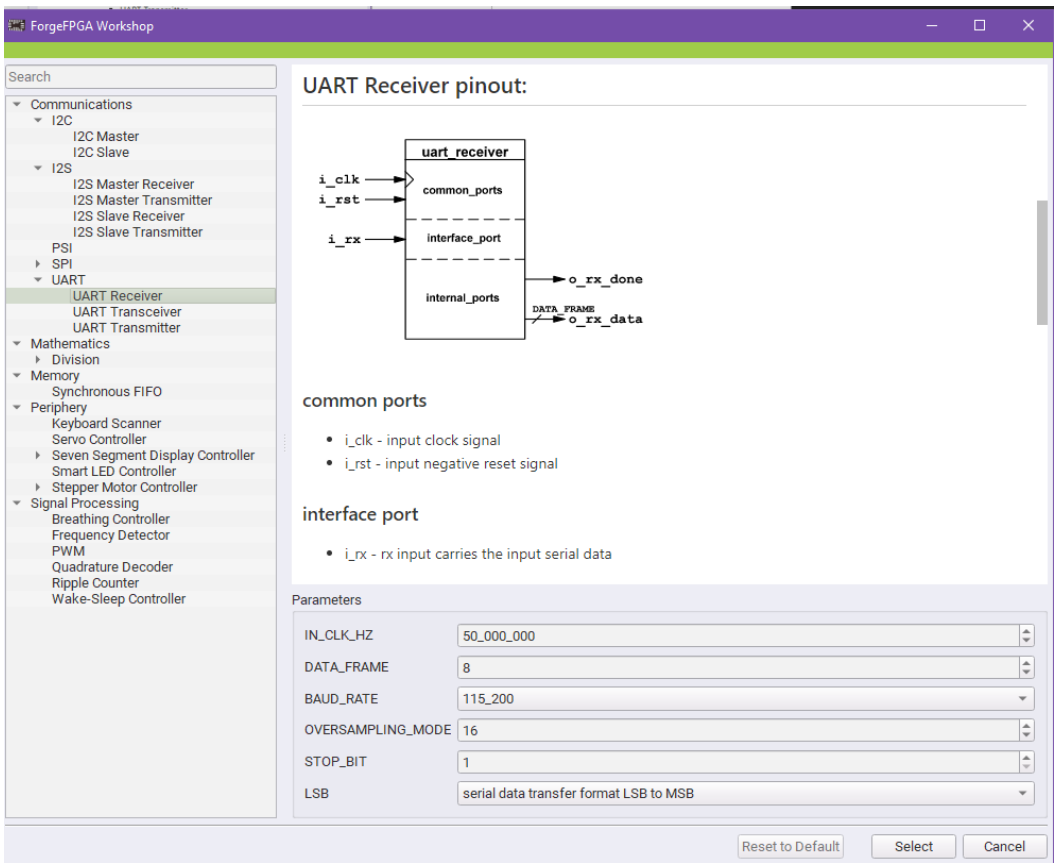

Figure 20. Module Library GUI with Parameters

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 section 2.1.6 Control Panel. Multiple modules can also be added to the design and used simultaneously.

2.4 RTL Synthesis

The ForgeFPGA Editor comes with a built-in Synthesis tool (Yosys, v0.37, 0.59) 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 the current design, press the **Synthesis** button on the bottom of the Control Panel (see section 2.1.6 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 (see section 2.1.5 Messages). The Synthesis process will not be completed until the code is free of any syntax errors.

The 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**.

2.4.1 I/O Planner

Each available I/O port on the FPGA has a dedicated function that can be mapped to the current 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 it 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** –dedicated function assigned to the port.
- **Port** – editable column, where ports from the Verilog design can be entered to connect them to the desired functionality. Double-click a cell to see the list of all ports defined in the Verilog code.
- **Core Direction** – the mentioned direction is with respect to FPGA Core.
- **Description** – data describing the port type and the function it fulfills.
- **Tile** – represents the signal is part of tile [0,0] or tile [1,0].
- **Position** – gives the coordinates of the signal.

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 21).

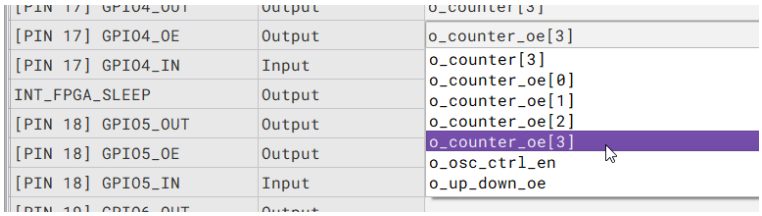

Figure 21. I/O Port Signal Assignment

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	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 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 (see Figure 23). The user can sort any column by clicking on the header of the respective column.

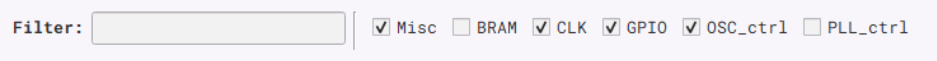

Figure 23. I/O Planner Filter Selection

The user can also clear all the data entered into the I/O Planner by going to main menu and 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.**

2.4.2 Synthesis Report

After performing Synthesis procedure on the Verilog code, a Synthesis Report is generated. This report shows the number of primitive objects used to represent the design in the generated Netlist (see section 2.4.4 Netlist).

To open the **Synthesis Report** window, click the corresponding button on the toolbar or navigate via the main menu, **Window → Synthesis Report** (see Figure 24).

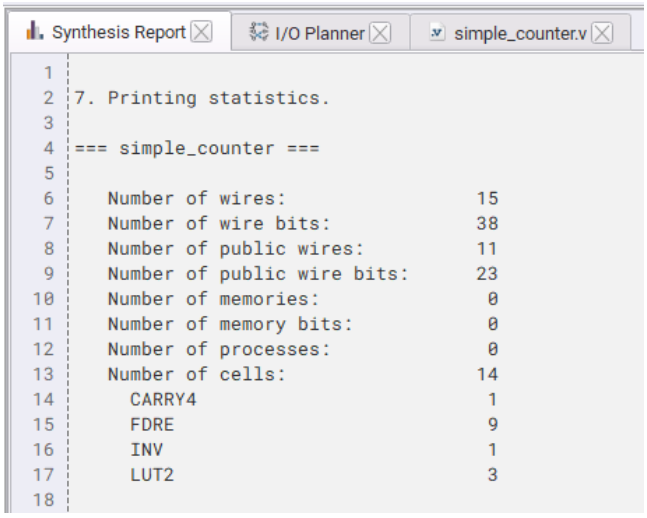

Figure 24. Synthesis Report

2.4.3 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 (see section 2.4.4 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.

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.

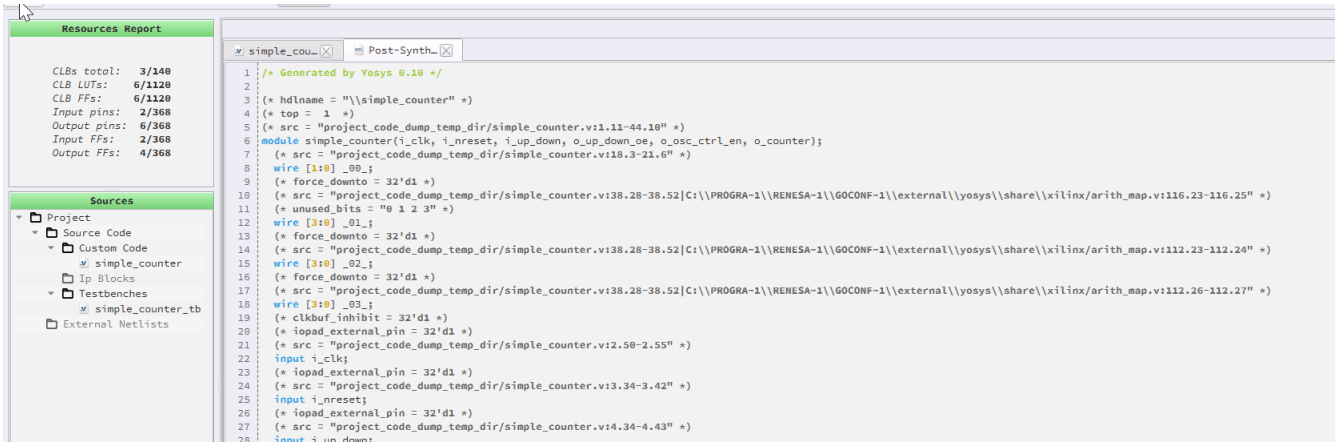

Figure 25. Post-Synth RTL

2.4.4 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. The Netlist can be accessed by clicking the **Netlist** button on the toolbar or from the main menu, **Window → Netlist**.

External Netlist can be imported by selecting **File → Import** (see section 2.2.1 Importing/Export RTL Files) → **Netlist** from the Main Menu.

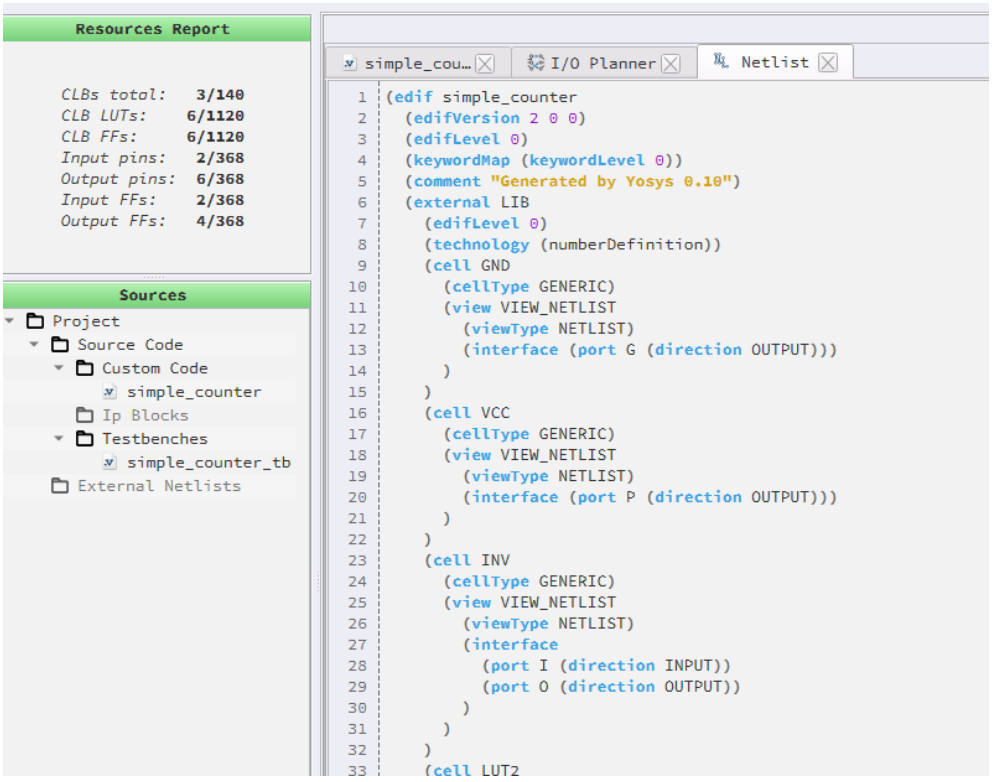

Figure 26. Netlist

2.5 Generating the Bitstream

To prepare the 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 (see section 2.4.4 Netlist) is successfully generated. To generate the Bitstream data,

click the **Generate Bitstream** button in the bottom left corner of the FPGA Editor (see section 2.1 FPGA Editor) or, from the main menu, **Tools** → **Generate Bitstream**.

Check the **Bitstream Log** tab on the **Message** panel to inspect the background steps after clicking the **Generate Bitstream** button. 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 outcome is available on the **Messages** panel (see section 2.1.5 Messages).

The generated Bitstream is saved as a txt and bin file which, if generated correctly, can be used for debugging the design. For more information about chip and flash procedures and their locations, see section 3.2 Debugging Controls.

2.5.1 AES Encryption

The Go Configure Software Hub can encrypt the Bitstream with the use of the AES128 Key. The user can do this by accessing the properties of the **AES128 Config** block in the ForgeFPGA Window.

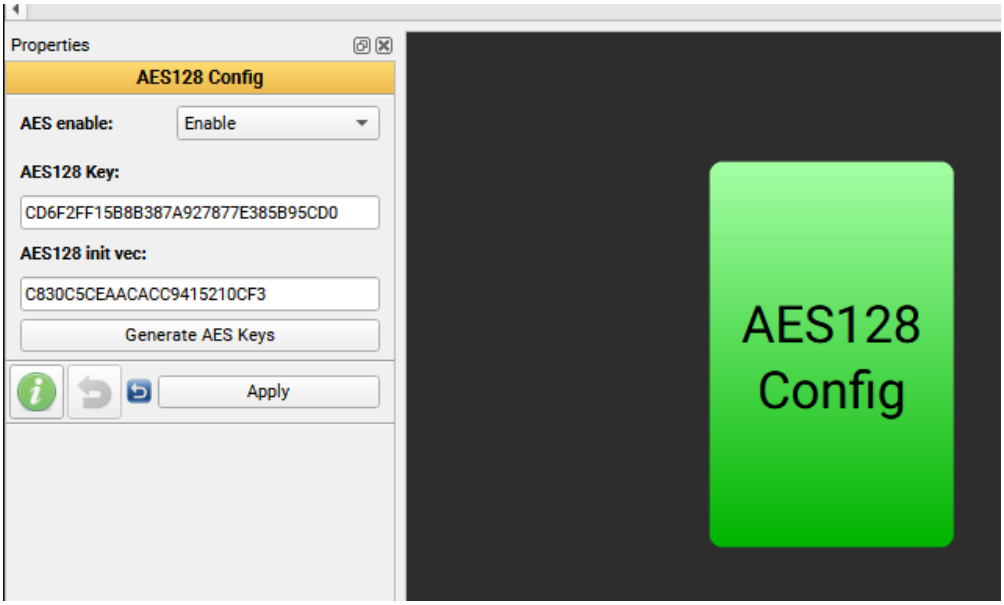

Figure 27. AES128 Config Block

When enabled, the compiler encrypts the resulting FPGA_bitstream_MCU and FPGA_bitstream_FLASH_MEM bitstreams to secure the design data. The AES128 Config block cannot generate an encrypted OTP Bitstream (see Figure 27).

The user can generate a random AES128 key and AES128 initialization vector pair by clicking on the **Generate AES Key** in the properties panel for the AES128 Config block or the user can also specify these values by typing in the hexadecimal values for **AES128 Key** and **AES128 init vec** (initialization vector) for encryption.

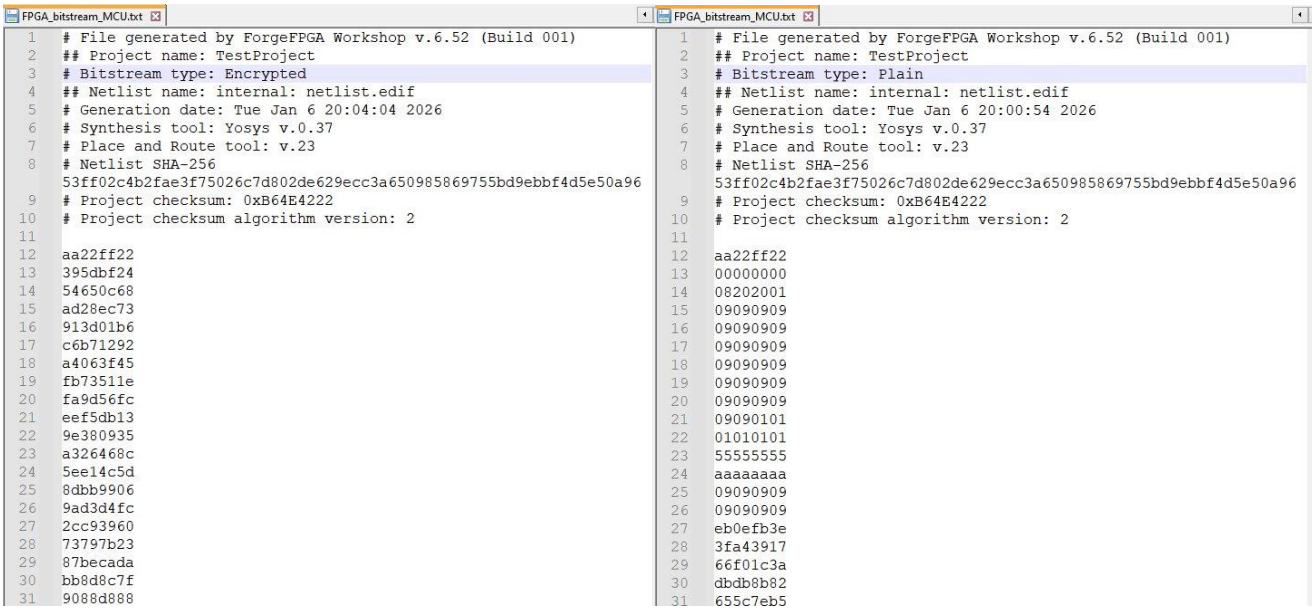

Figure 28. Example of Encrypted Bitstream

Figure 28 shows an example of an encrypted MCU/Flash Bitstream.

Note: The user can find the AES128 Config block in SLG47912V/C, SLG47920V, and SLG47921V. Recompilation is not required after modifying encryption settings. The generated bitstream files are updated automatically.

2.5.2 Bitstream Content Management

Configure the contents of the generated FPGA_bitstream_OTP bitstream file using OTP configuration file includes setting:

- **Full FPGA configuration** – the generated OTP file will contain the complete FPGA design along with the boot configuration and user data.
- **Boot config + User OTP data only** – the generated OTP file will contain only the boot settings and specific User OTP data, excluding the main FPGA design.

The generated OTP bitstream file will indicate Bitstream type in the file header.

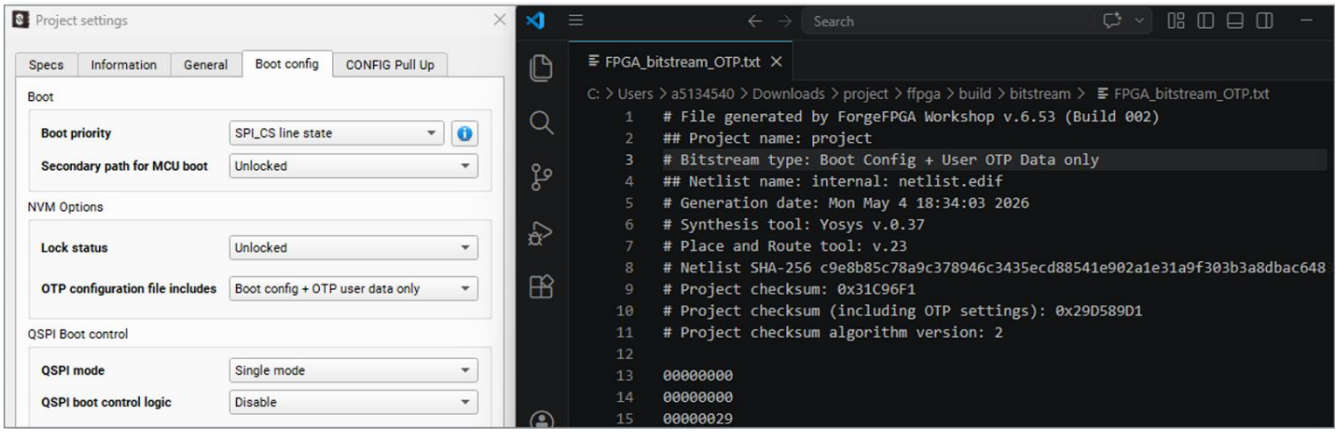

Figure 29. OTP Configuration File Includes setting in the NVM Options

Note: Recompilation is not required after modifying OTP configuration file includes setting under the NVM Options. The generated bitstream file is updated automatically.

2.5.3 Floorplan

The results of the place-and-route procedure are visualized in the **Floorplan** tool. 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.

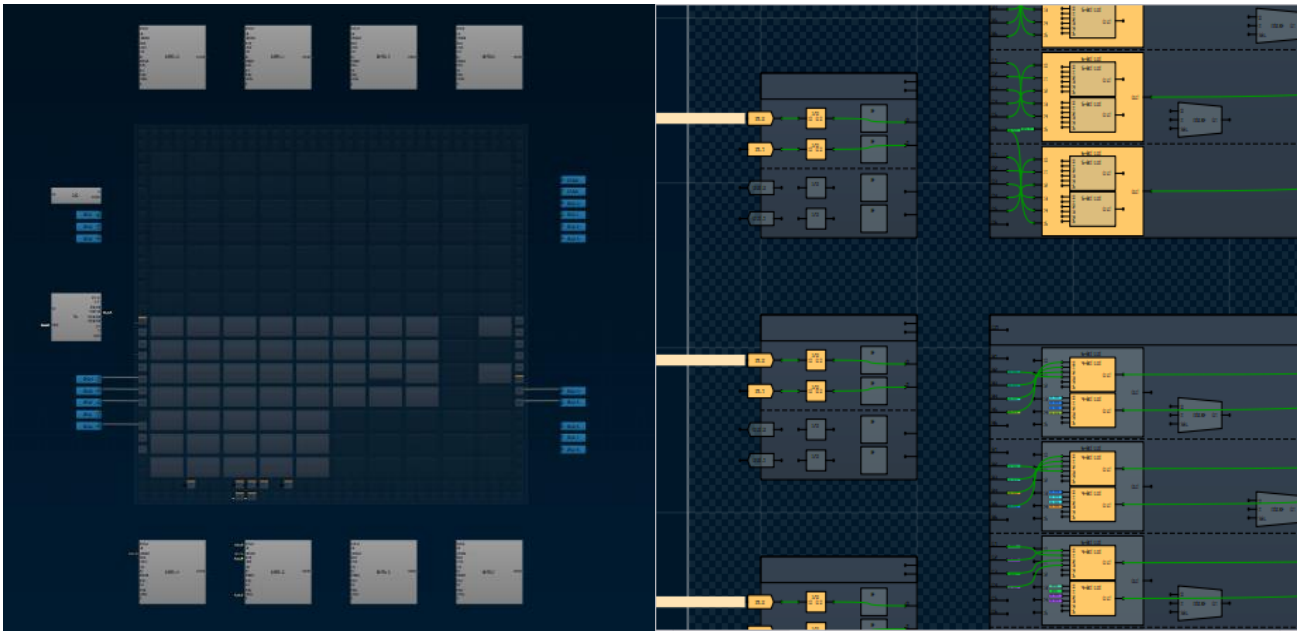

Figure 30. Floorplan Window (SLG47910)

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 on the internal components (**CLB**, **IOB**, **LUT**, **CARRY4**, **FF**, and **Port-Net**) to see more information on the **Block Tree** and **Properties** info panels (see Figure 32). Filter out the required data in the respective field as needed. The user can understand the position of the specific IOB being used from gathering information from the **Properties** panel or obtain a list of the input/output Ports used.

The user can also select any **Net**, which highlights its path through the busses routed between CLBs and IOB Blocks.

When the project state no longer matches the compiled data, a warning banner stating to regenerate the bitstream appears in Floorplan, Resource Report, and Timing Analysis tabs. It is triggered by design modifications, including RTL code edit, I/O Planner pin reassignment, or constraint update.

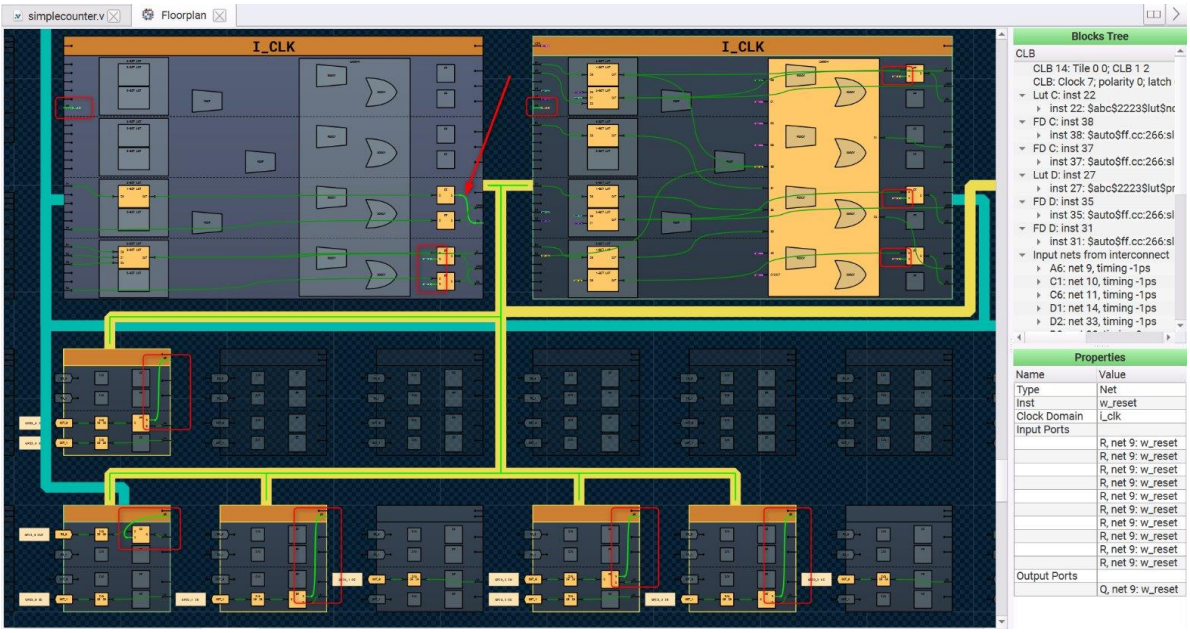

Figure 31. Net Routing

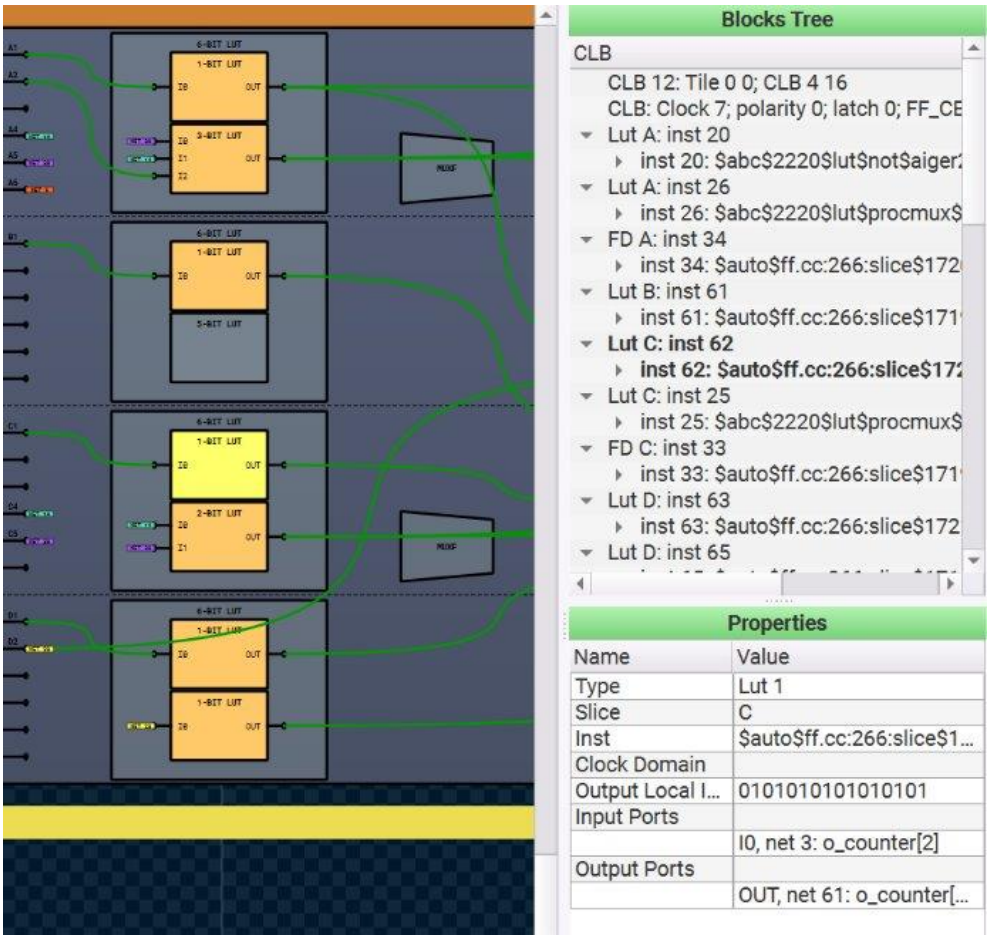

Figure 32. Floorplan Properties

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 33):

- **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 in need of zoom.
- **Align Vertical/Horizontal**: align Macrocells relative to each other in the work area.
- **Snap to grid**: use for precise graphic alignment of the Macrocells along a grid.

Figure 33. Footer Controls

2.5.4 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, LUTs, Shift-register, and Distributed memories are being used for the synthesized design.

A summary of these utilized resources is also mentioned in the **Control Panel** (see section 2.1.6 Control Panel).

Clicking the **Resources** button on the toolbar or selecting **Windows** → **Resources Report** from the main menu launches the **Resources Report** window.

I/O Planner | simplecounter.v | Resources Report | Floorplan

```
####   Resource Utilization Report   ####

    Top level module name: simplecounter

    Resource usage:

        CLB LUT5s:    7/2240 (0.31%)
            Lut5:                          7 (0.31%)
            Lut5 for Shift-register:       0 (0.00%)
            Lut5 for Distributed memory:   0 (0.00%)

        FFs:    13
        CLB FFs: 7/2240 (0.31%)
        IOB FFs: 6/1248 (0.48%)

        CLBs:      3/280 (1.07%)
        Tiles:     2/2 (100.00%)

        RTL Input ports:     3
        RTL Output ports:    15
        GPIOs:               5/32 (15.63%)

        PLLs:        0/2 (0.00%)
        OSCs:        1/1 (100.00%)
        4k BRAMs:    0/16 (0.00%)

        1 Clock
           Clock 0: net     2, i_clk, Fanout: 9 CLBs
           Total 1 unique clocks

####   Resource Utilization Complete   ####

TILE-REQ:       0       0       0       1

### Total FPGA resources from current floorplan: ###
                      312 IOBs
                      280 Logic 6-LUT CLBs
                             80 CLBs support RAM/SR

Usage of Logic CLBs is 1.1%.
    Logic-Memory CLB for RAM/Shift-Register is 0.0%.
Usage of LUTs is 0.5%.
    dual-LUTs is 0.1%.
Usage of FF is 0.4%.
    dual-FF is 0.2%.
Usage of IO CLB is 3.8%.
```

Figure 34. Resources Utilization Report

2.5.5 Timing Analysis

The Timing Analysis tool extracts the timing of the design 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.

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

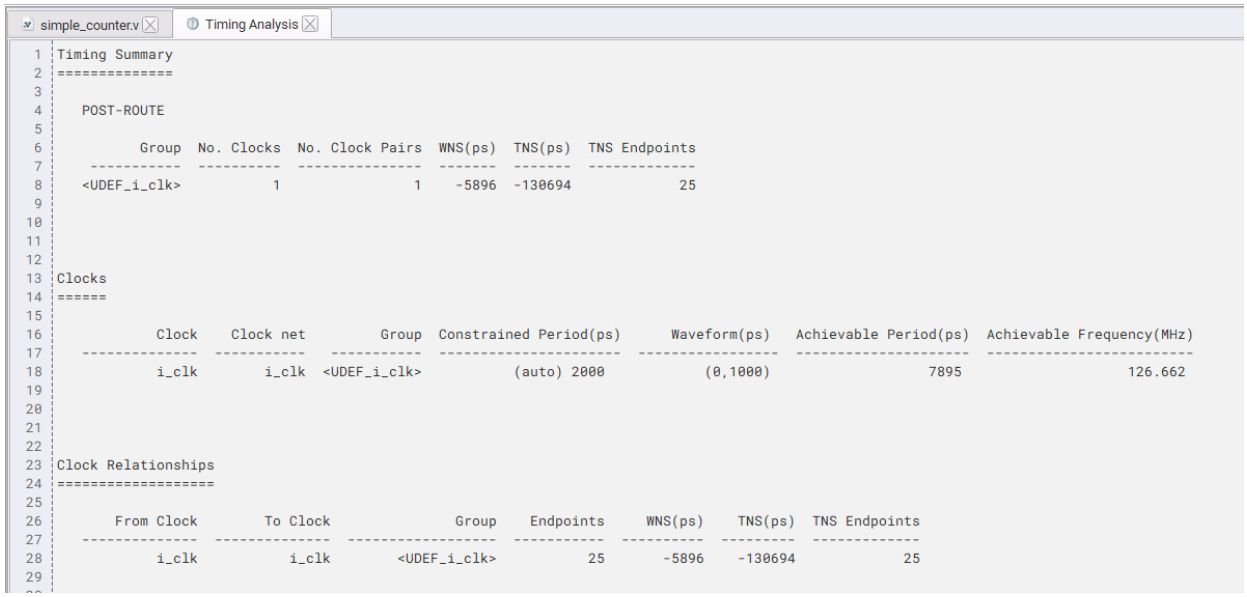

Figure 35. Timing Analysis

Add Timing Constraint files (.sdc file) to define clock settings and path-specific requirements (add or import the file (see section 2.2.1 Importing/Export RTL Files) 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.

See the list of supported SDC commands and arguments 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 excludes 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 Analysis 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 35 for examples of negative WNS and negative TNS reports).

The Timing Analysis report is used identify the violations and modify the .sdc file. Re-run the **Bitstream** to generate a new Timing Analysis report after any modification is made to the .sdc file.

2.5.6 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 block's 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 (see section 2.1 FPGA Editor) works in correspondence with third-party software for simulating the testbench, called Icarus Verilog, and for verifying the functionality of the design by viewing simulation results, GTKWave software is used. Please refer to the *ForgeFPGA Software Simulation User Guide*, [7], for the installation process of the additional software and quick start guide.

Run the process using the **Simulation** button on the toolbar or from **Tools** → **Simulation**.

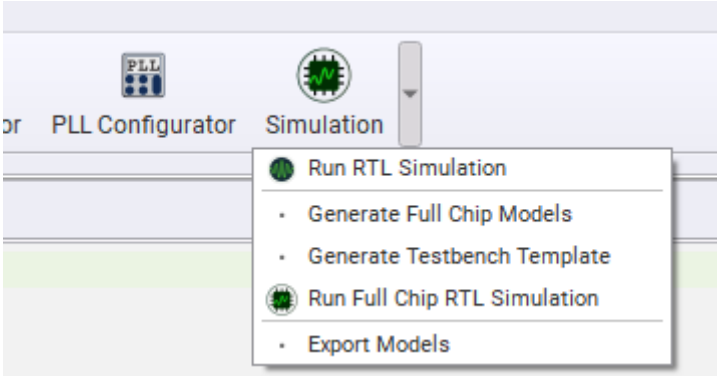

Figure 36. FPGA Simulation on Toolbar

RTL Simulation executes the RTL only. Full Chip RTL Simulation allows extended behavioral simulation of the 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, and others). This simulation type also includes basic delays of peripheral circuits, producing results that more accurately reflect the device's real behavior.

2.6.1 RTL Simulation

- **Writing a testbench** – to work with RTL Simulation, a testbench module for the FPGA design is required. A testbench can be added from the Modules Library, or a module can be imported from the main menu, **File → New Customer Testbench**.

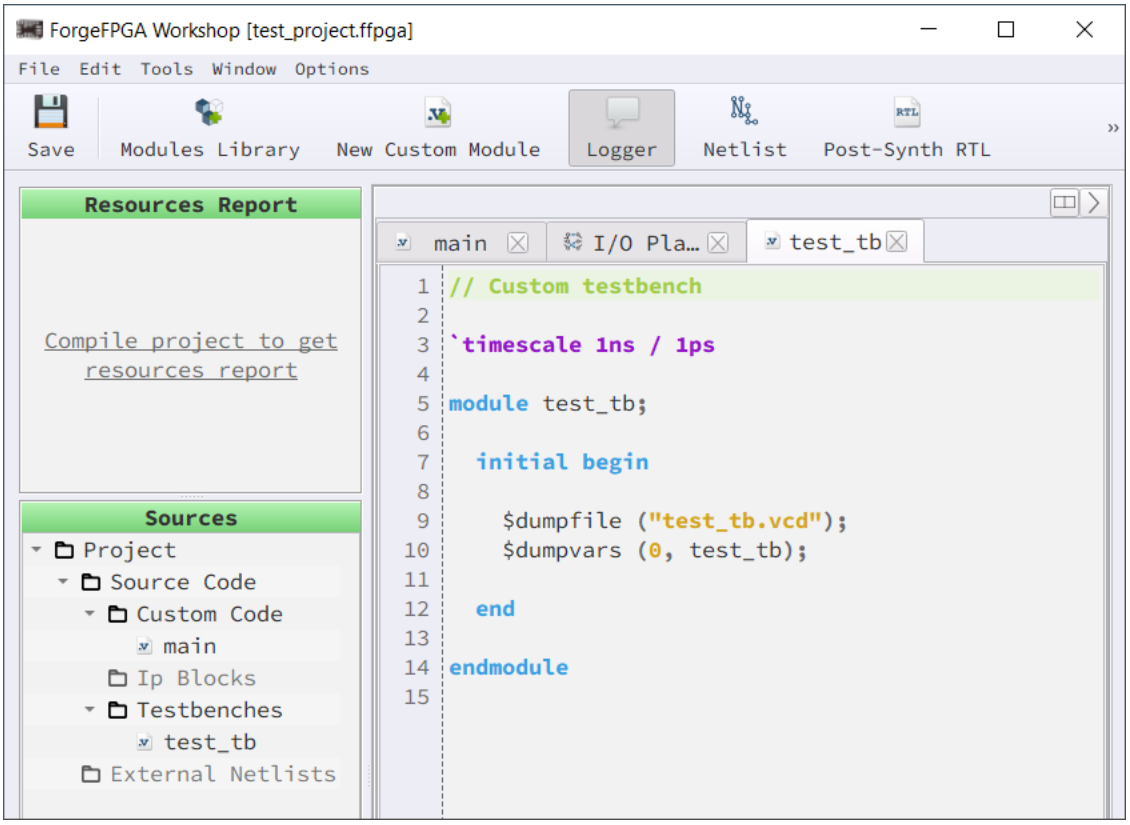

Figure 37. Custom Testbench Example

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 the *ForgeFPGA Software Simulation User Guide*, [7].

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 38. Simple Counter Testbench Example from Template Design

- **Simulating a Testbench** – after the testbench is added, click **Simulation → Run RTL Simulation** on the toolbar to launch Icarus Verilog and GTKWave, or use the main menu **Tools → Simulation → Run RTL Simulation**. The simulation stage handled by Icarus Verilog is performed in the background, which GTKWave provides a visual representation. If the written testbench is correct and does not have any syntax errors, the GTKWave software will launch automatically, otherwise check the **Messages** panel (see 2.1.5 Messages) for any syntax related issues in the written testbench and make any necessary changes.

2.6.2 Full Chip RTL Simulation

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

1. Ensure the project contains error-free RTL code.
2. Perform successful Synthesis stage (netlist is used for top module ports). The generated Netlist serves as the source for extracting the top-level module name and its port definitions.
3. Verify that Synthesis results are up to date with the design (see section 2.4 RTL Synthesis).
4. Perform successful Bitstream generation stage (see section 2.5 Generating the Bitstream). Post PnR results are utilized to assign correct delays in the generated models.

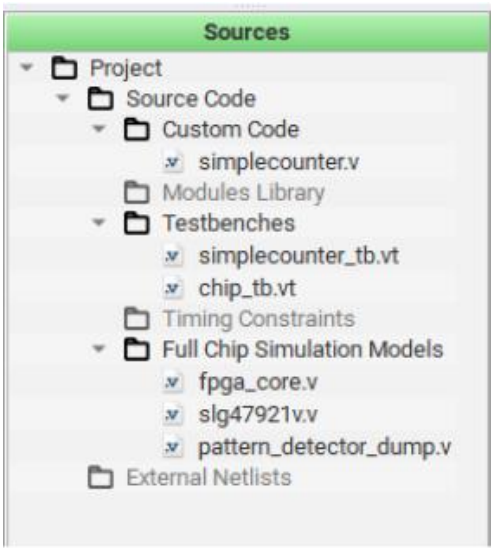

Figure 39. Full Chip Simulation in Source Tree

5. Optional: perform template testbench generation (**Simulation → Generate Testbench Template**).
6. Full chip models are automatically generated after clicking Run Full Chip RTL Simulation, or generate them manually (**Simulation → Generate Full Chip Models**). Find them added in the Sources tree. To manage model generation behavior, reach **Options → Settings**.
7. Run simulation for selected testbench (**Simulation → Run Full Chip RTL Simulation**).

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

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-chip simulation in another project (**Simulation → Export Models**).

In case of simulation failure, check the **Messages** panel (General Log or Issues Tab, see section 2.1.5 Messages) and make necessary changes.

The FPGA Editor ensures that work can continue from the last saved state of the simulation results. Once the necessary configuration changes (for example, add signals, adjust their graphical representation, and others) are made, save the progress in the GTKWave tool. Next time the same testbench is simulated, the previous state is restored

For information on how to run Full Chip Simulation, see the application note *AN-FG-025 How to use Full-Chip RTL Simulation*, [3].

2.7 Part Number Migration

The part number migration feature allows the design to be transferred between compatible part numbers without starting a new project. The user can migrate design between the SLG47921V/C ⇔ SLG47912V/C. The software automatically migrates the configuration data, resources, and settings after part number switch.

The user can access the option of migrating their design from **ForgeFPGA Workshop Window → Project Settings → Specs → Part Number Migration.**

The user can choose which part number to migrate their present design to from the available options in the drop-down menu (see Figure 40).

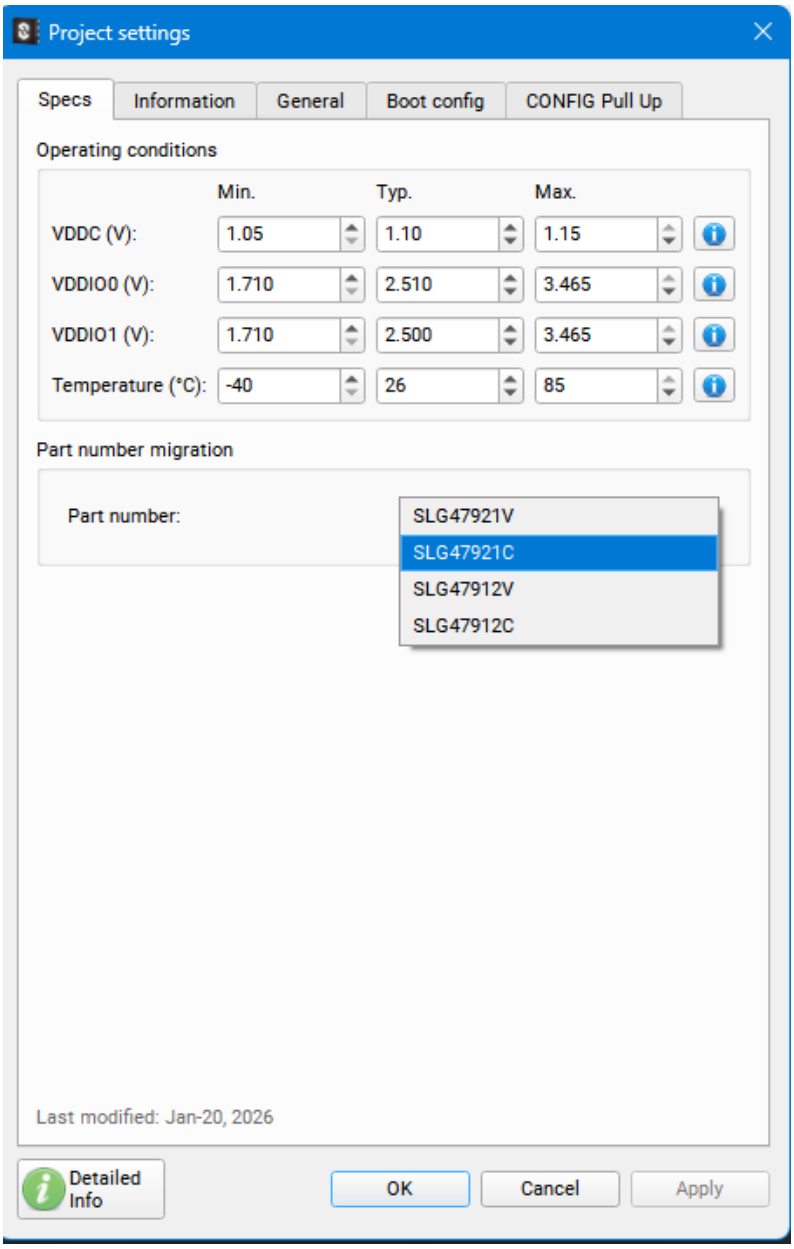

Figure 40. Part Number Migration

Note 1: This property is accessible only if the FPGA Editor window is closed. An error will be flagged in the Project Settings dialog box with a message saying “Migration is paused because other tools are running. Close all active tools to continue.”.

Note 2: If the target device does not provide sufficient resources or lacks features required by the existing design, the migration will proceed normally, and an appropriate warning is displayed.

After successful migration, the Bitstream will reset. All the previously generated files will be removed, as well as the Resource Utilization and Timing Analysis results. The complete Bitstream pipeline will be regenerated.

In cases where the core resources exceed the chip capacity migrated to, a message will be displayed indicating that the migration has been cancelled.

The SLG47912 does not support PLL1, BRAM2, and BRAM3. When migrating a design from the SLG47921 to the SLG47912, components and configuration related to PLL1, BRAM2, and BRAM3 will be removed automatically for successful migration. This will be followed by a message indicating this as well (see Figure 41).

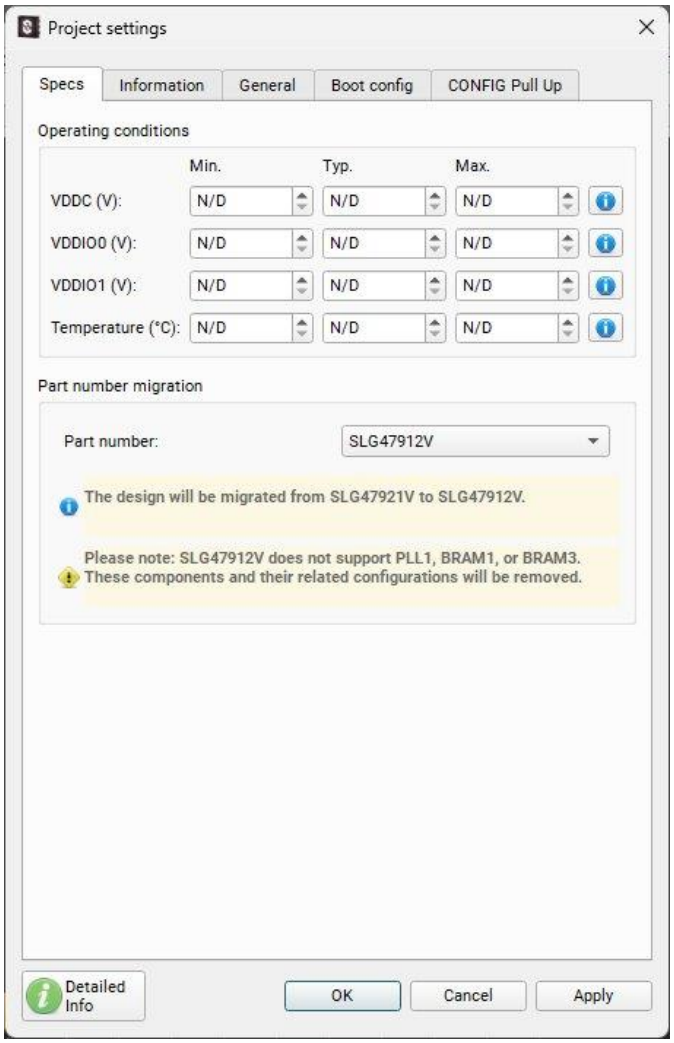

Figure 41. SLG47912V/C Migration Warning

2.8 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** from the toolbar**.** The PLL Configurator has three tabs as described below:

PLL Calculator tab

The main tab allows configuration of the parameters and assignment to the Verilog template, I/O Planner (see section 2.4.1 I/O Planner) and registers (see section 8 NVM Viewer and Figure 42). For part numbers with multiple PLLs, tick the checkboxes located on the right to enable the component and proceed with configuration.

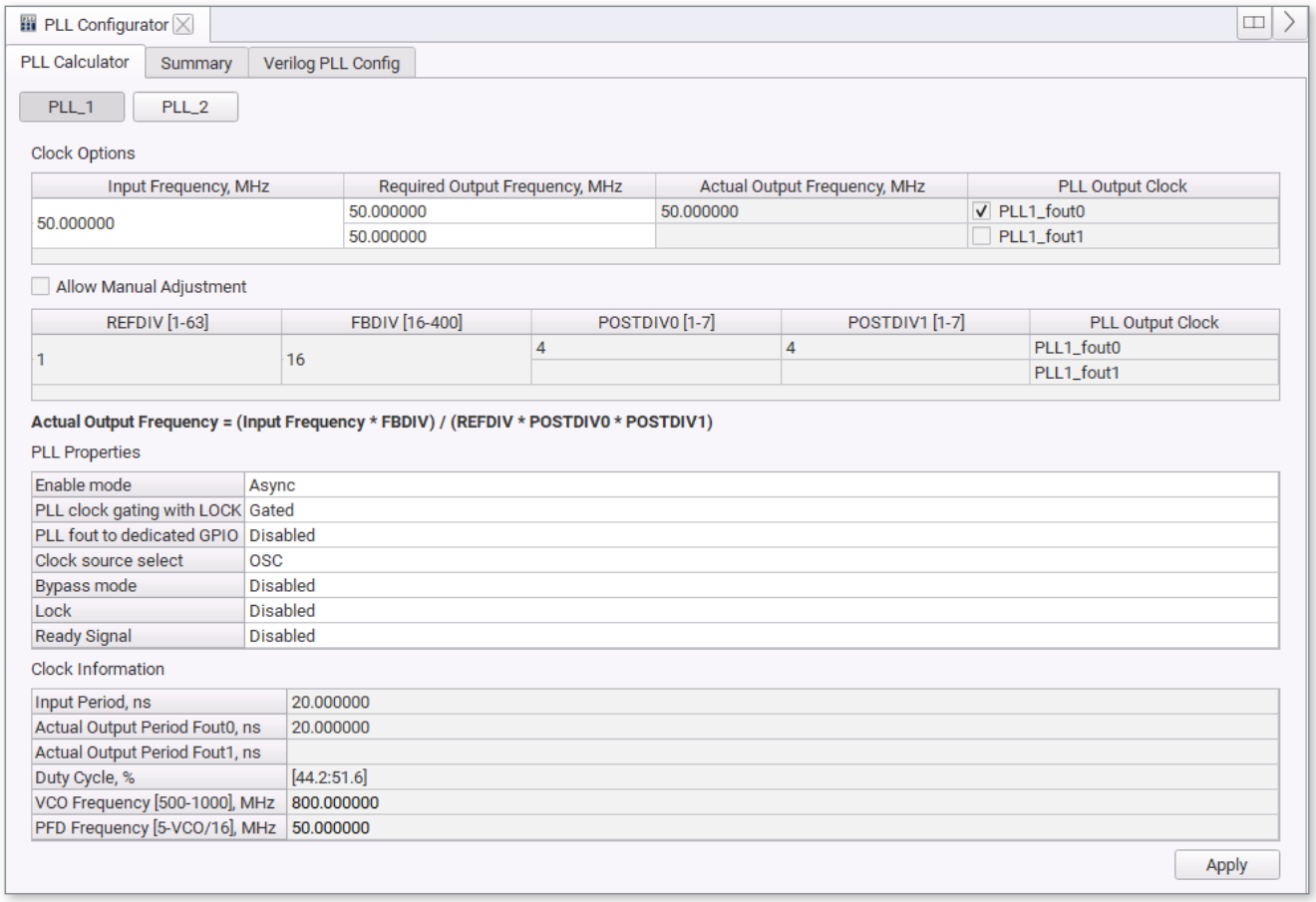

Figure 42. PLL Calculator

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** button. The **Apply** button saves changes for each PLL separately. Manage the behavior of the confirmation window before applying the changes in Settings (see section 2.1.7 Settings). The **Apply** button populates the I/O Planner's (see section 2.4.1 I/O Planner) 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 section 2.1 FPGA Editor and Figure 43). Modifying settings may remap specific ports in I/O Planner. Conflict triggers a resolution window.

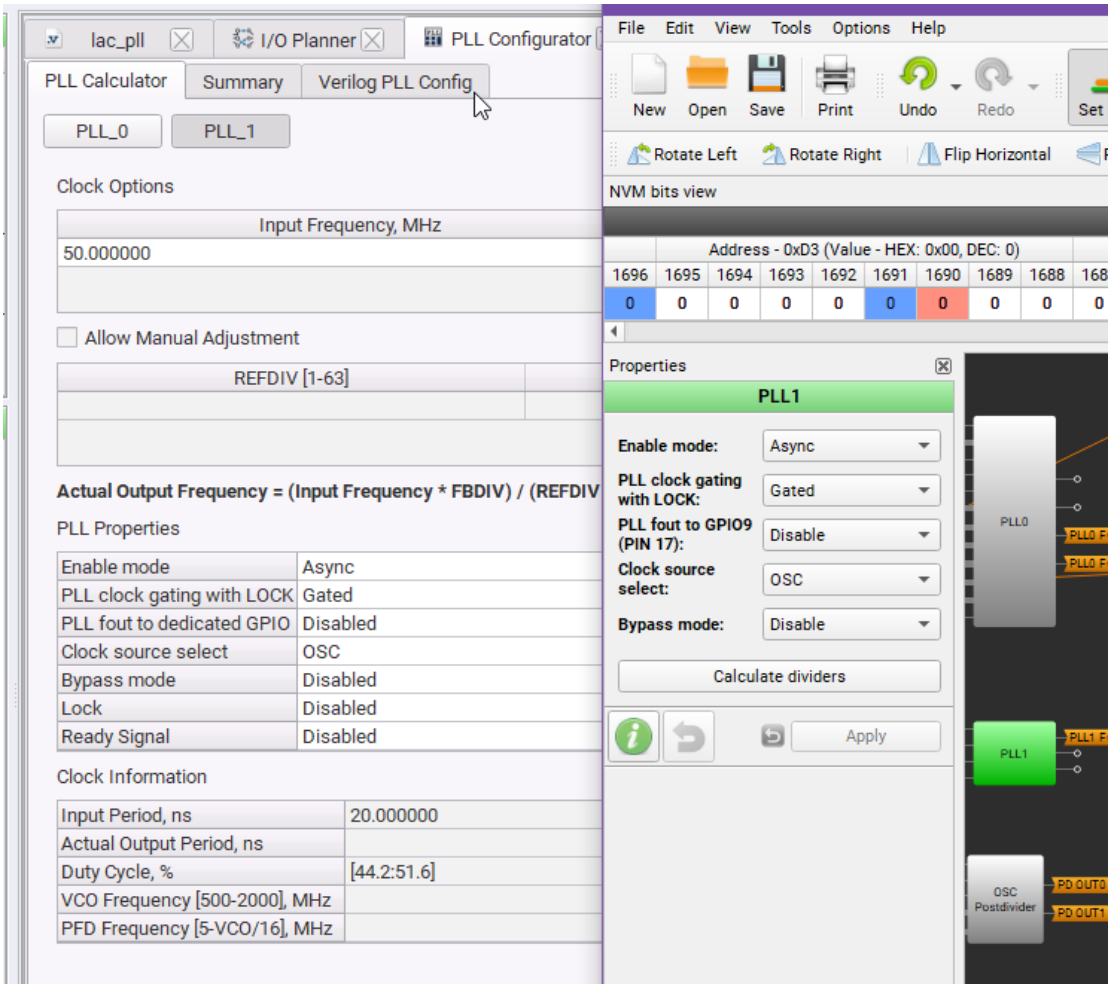

Figure 43. PLL Properties in PLL Configurator and PLL Properties

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 displays a list for both of the PLLs.

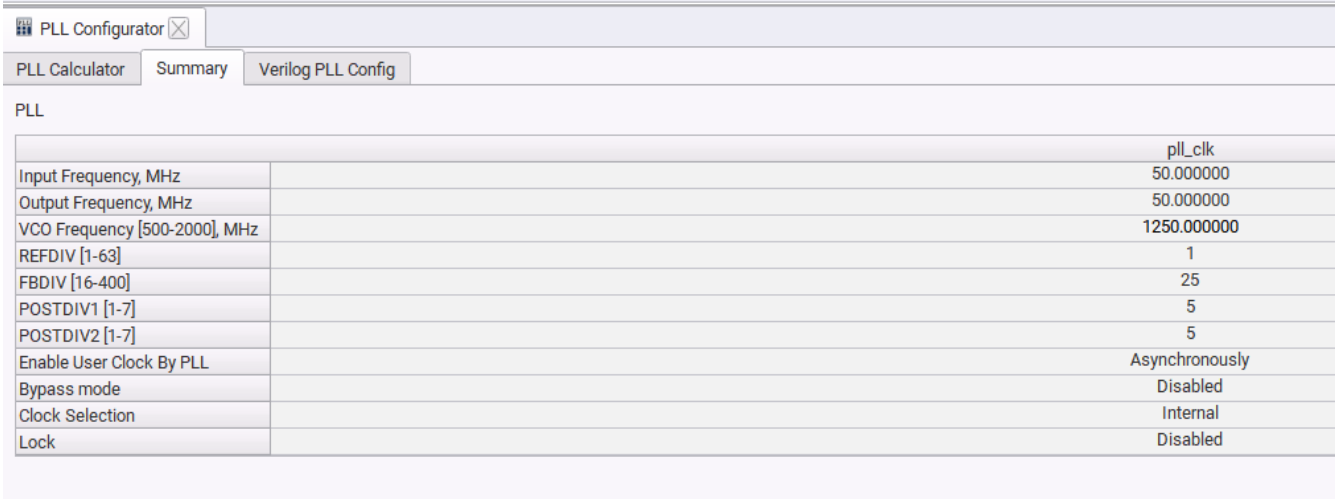

Figure 44. Summary Tab

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. User can copy and paste the auto generated Verilog PLL Configurator code into the top module of the Verilog code.

Note: The port names generated in the Verilog code has default names and it matches the automatically generated signals in the I/O Planner.

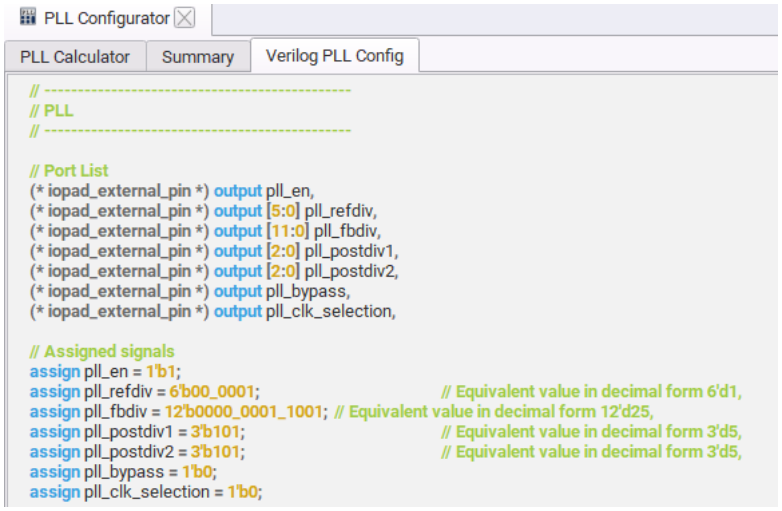

Figure 45. Verilog PLL Config Tab

2.9 Macrocell Editor

Macrocell Editor is a tool that allows the user to create the desired circuit in a schematic view. Launch the 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.**

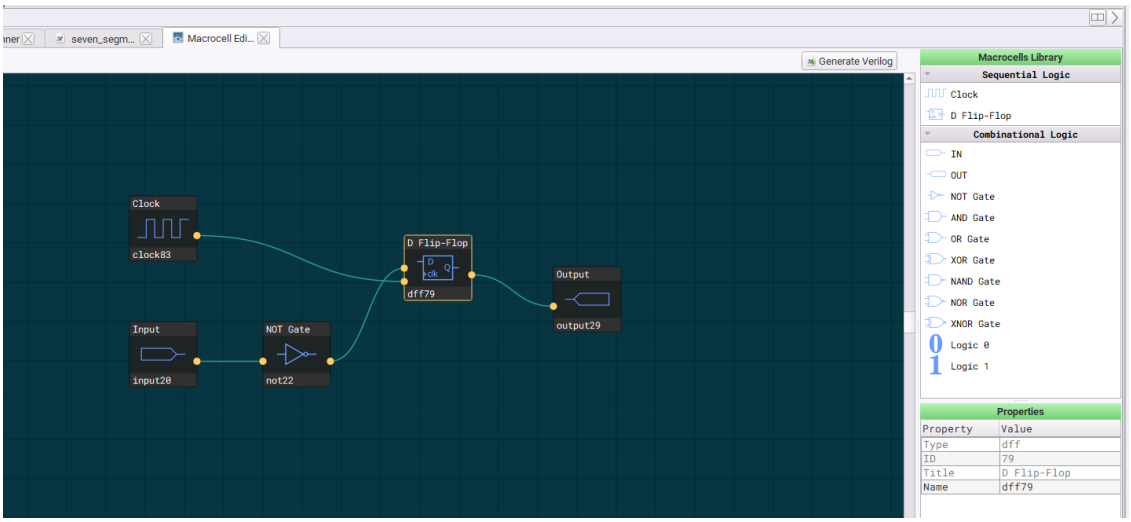

Figure 46. Macrocell Editor

The Macrocell Editor consists of two parts; the green screen panel 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 **Clock Sources**, **Sequential Logic**, **Combinational Logic**, **Blocks**, and **IP Modules** components. Click the desired component from the library and then the work area to add it to 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 47).

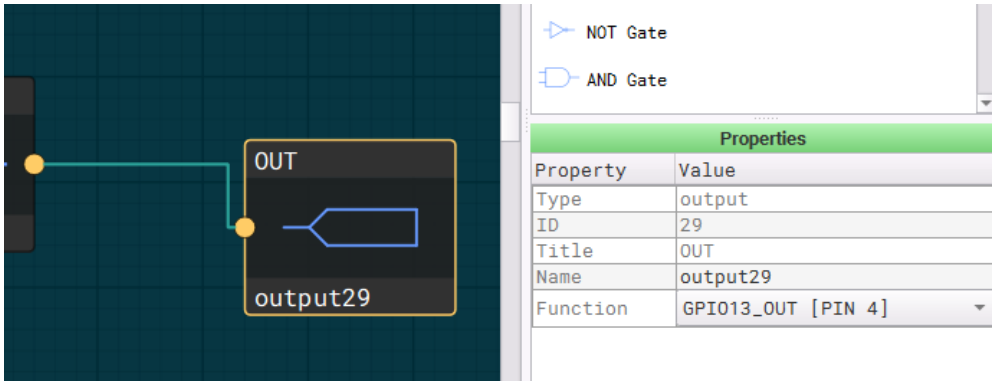

Figure 47. Changing The Names of Input/Output Pins

For IN and OUT **Combinational Logic** blocks, map the ports directly in the **Properties** panel via the **Function** drop-down. Tick **Auto set ports to I/O Planner** before clicking **Generate Verilog** to display the desired mapping in the I/O Planner tool (see section 2.4.1 I/O Planner).

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 (see section 2.1.6 Control Panel). This code can be used for Synthesis along with other modules added to the design.

2.10 Project Directory Folder

The FPGA Workshop 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 of the modules and netlists which can be added or imported to the Sources tree in the FPGA Editor. Any changes made to the sources will synchronize between the FPGA Editor and the file on disk.

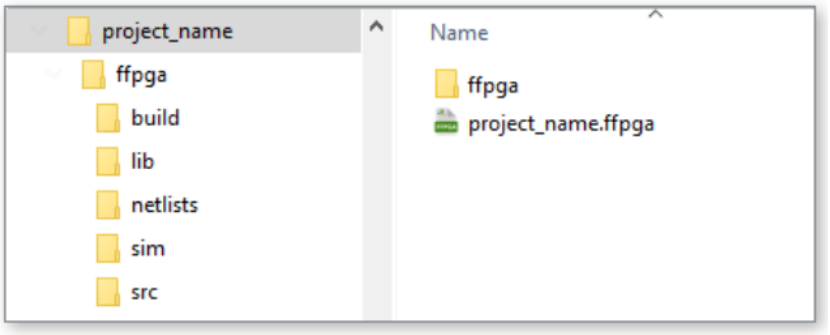

Figure 48. FPGA Project Folder Structure

The named folders below store the following types of files:

- **build** – result files generated after Synthesis and Bitstream generation
- **lib** – modules from the Modules Library (see section 2.3 Modules Library)
- **sim** – testbench and Simulation results files
- **src** – main (top module), mcm_generated (created by clicking Generate Verilog in Macrocell Editor (see section 2.9 Macrocell Editor)), and imported modules
- **netlists** – imported Netlists
- **timing-constraints** – files from Timing Constraints
- **placement-constraints** – files from Placement Constraint
- **full-chip-simulation-models** – generated modules required for performing Full Chip Simulation (see section 2.6.2 Full Chip RTL Simulation)

Use the **Open Containing Folder** option in the Sources Panel to quickly locate the file on disk (see Figure 49).

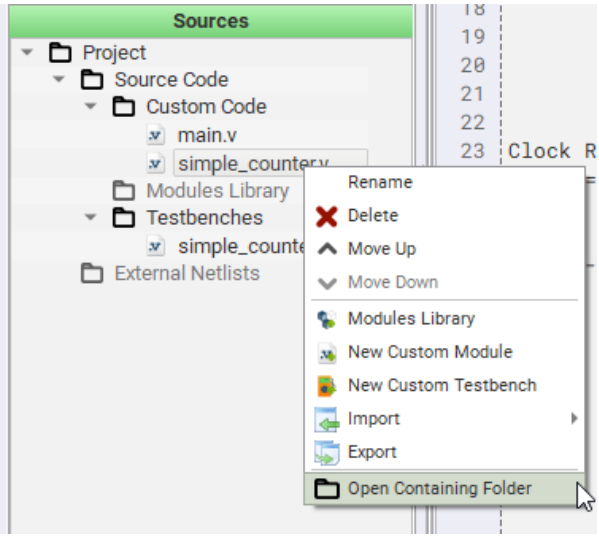

Figure 49. Check Source File Location

As soon as the user performs the procedures (Synthesis or Bitstream Generation), the certain files are generated to the build folder. The list of generated files depends on the performed procedures. Below is a description of the most important files in the build folder:

- **bitstream** – contains the Bitstream files in binary (.bin) and hexadecimal (.txt) format.
  - **FPGA_bitstream_FLASH_MEM** –the Bitstream sequence that can be used for on-board FLASH programming.
  - **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_message** – stores log files containing potential warnings encountered during the parsing of SDC commands in the Timing Constraints file.
- **FPGA_bitstream_AXI.log** – contains the configuration part of the Bitstream that represents the logic of the FPGA core. This file is used while interacting with the development hardware when performing chip and 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** button.
- **PNR_IO.log** – I/O ports mapping file generated after successful Bitstream generation procedure.
- **netlist.edif** –resulta of Yosys data processing, describing the components and connectivity within the source design. An external netlist from another software/synthesis tool can also be imported.
- **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 the design.
- **simulation** –contains 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 50). Each Development Platform comes with its own set of Debugging Control settings discussed in section 3.2 Debugging Controls.

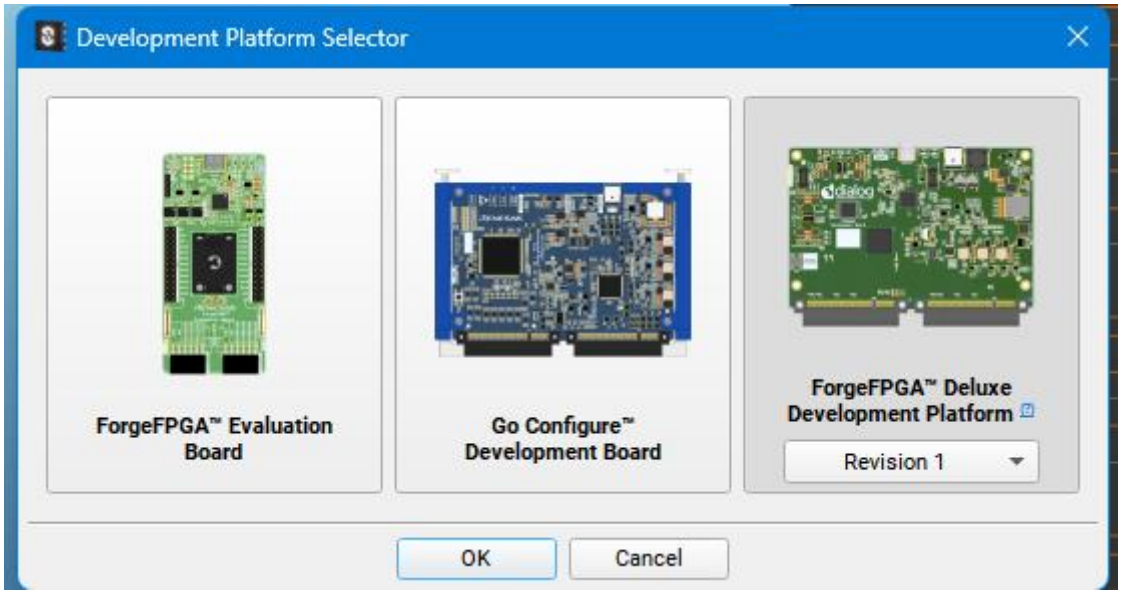

Figure 50. Development Platform Selector

3.1.1 ForgeFPGA Deluxe Development Platform (SLG7DVKFORGE)

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 51). 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.

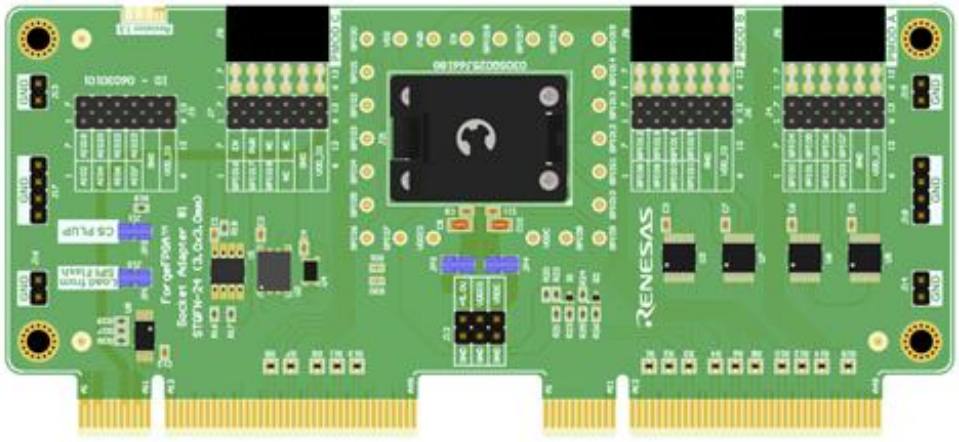

Figure 51. Socket Board Adapter

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.

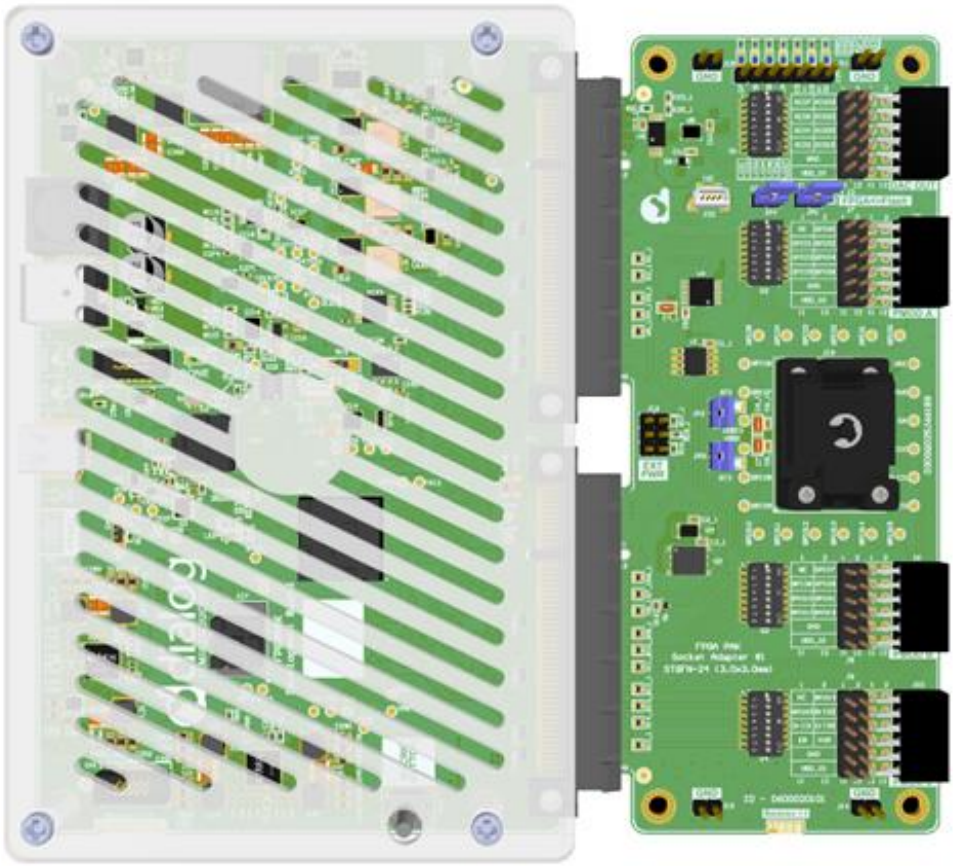

Figure 52. Assembled Equipment for Working with a Chip in the Socket

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, refer to the *ForgeFPGA Advanced Development Board R1.2 User Manual* (*ForgeFPGA Deluxe Development Board User Manual)*, [4], and *ForgeFPGA Socket Adapter #1 User Manual*, [5].

3.1.2 ForgeFPGA Evaluation Board (SLG7EVBFORGE)

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.

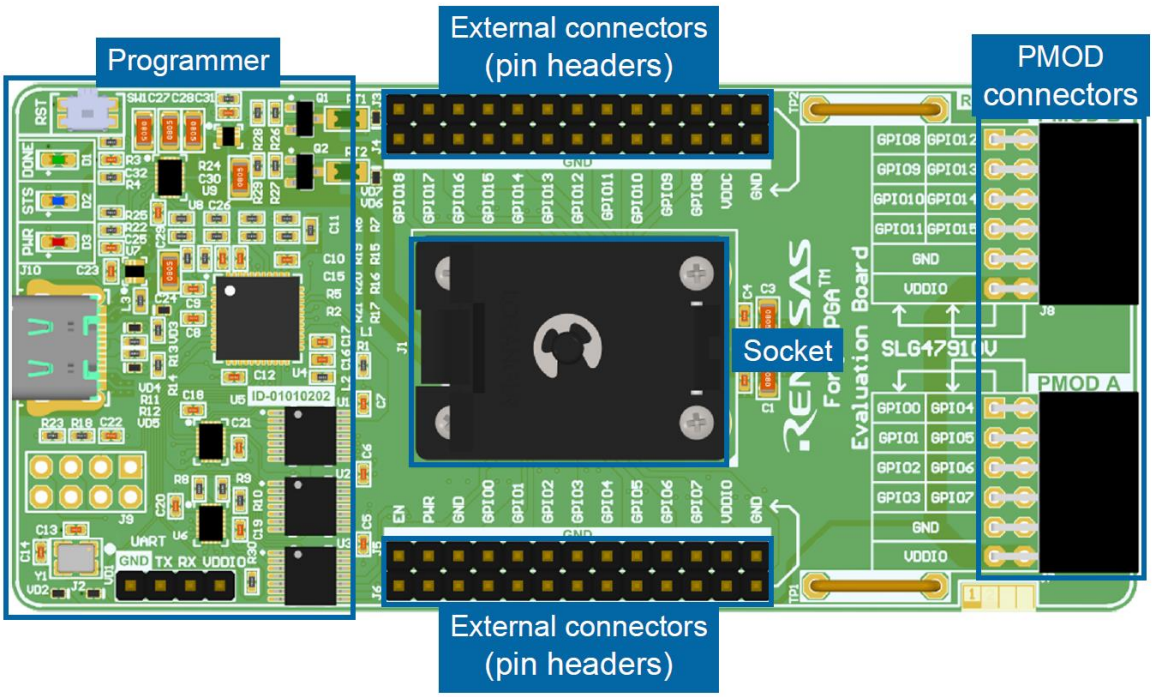

Figure 53. ForgeFPGA Evaluation Board v2.0

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 about the Evaluation Board, see the *ForgeFPGA Evaluation Board R2.0 User Manual*, [6].

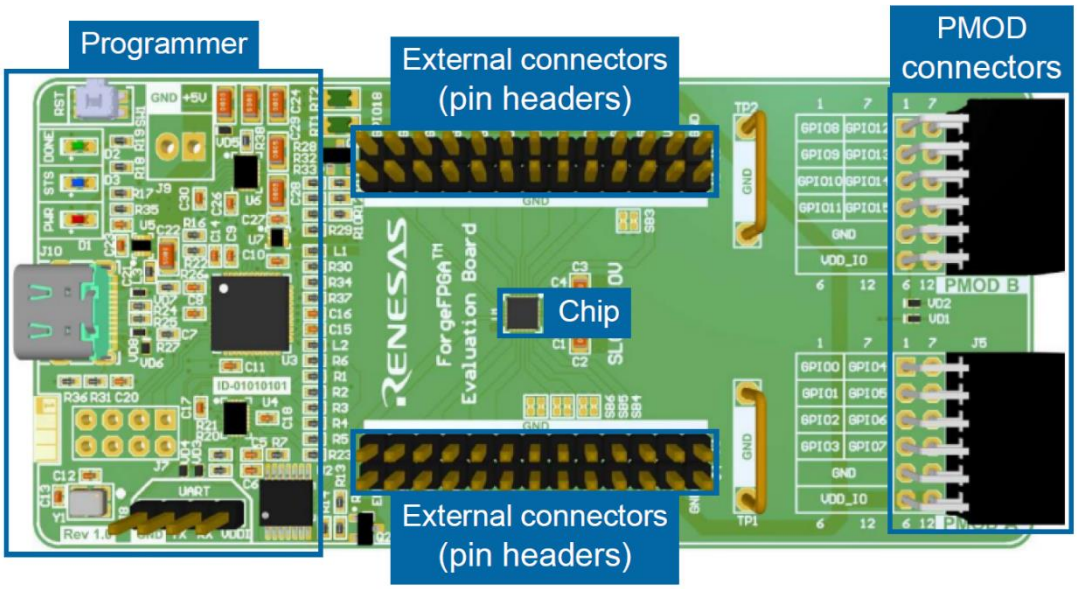

Figure 54. ForgeFPGA Evaluation Board v1.0

Note: There are two variations of ForgeFPGA Evaluation Board: version 2.0 and 1.0. ForgeFPGA Evaluation Board v.1.0 is a simplified version of the platform and does not support chip programming. Since the socket is absent, the SLG47910 IC is soldered directly on the board.

3.1.3 Go Configure Development Board (SLG4DVKGOCONF)

The Go Configure Development Board (see Figure 55) 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 Go Configure Development Board and for specific socket cards, see the ForgeFPGA *SLG47920/SLG47921 Datasheet* and *Go Configure Development Board R1.0 Manual*, [2]**Error! Reference source not found.**.

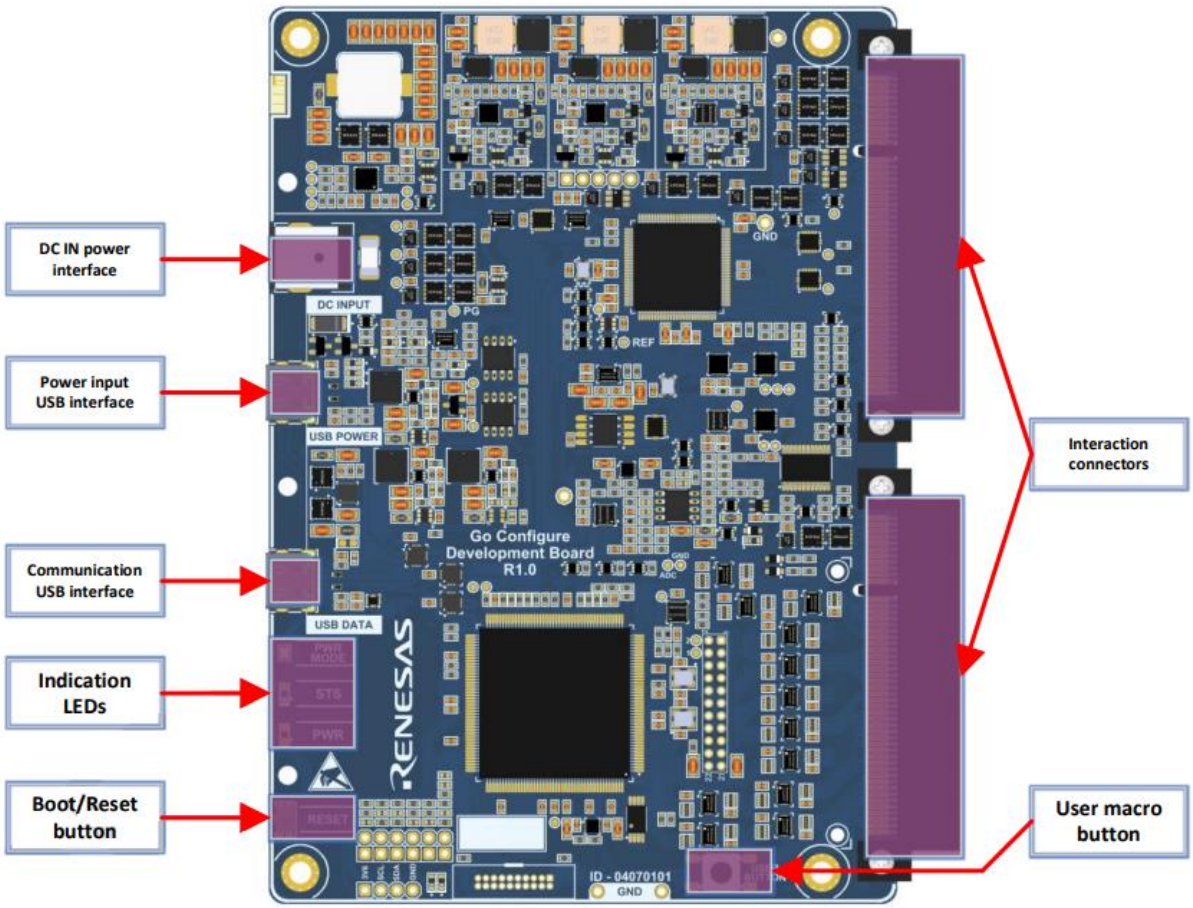

Figure 55. Go Configure Development Board

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 56). Each ForgeFPGA device and board have different options in the Debugging Control panel.

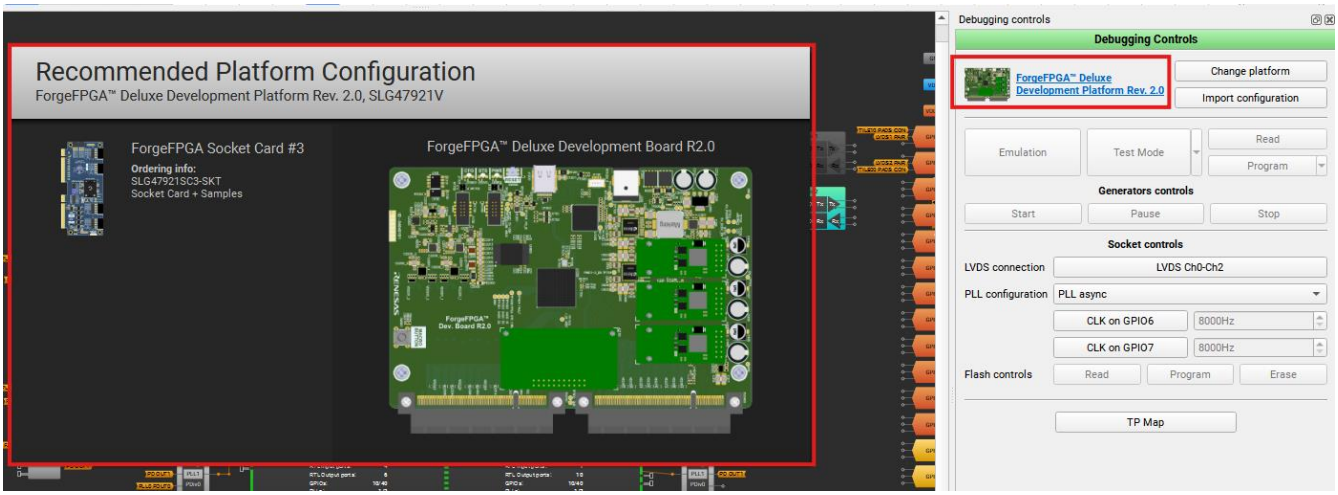

Figure 56. Platform Configuration Guide

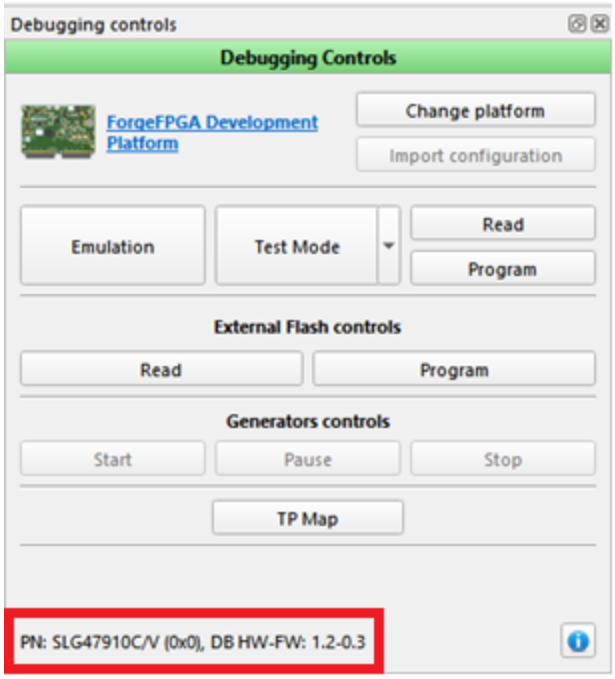

Figure 57. Debugging Controls (SLG47910)

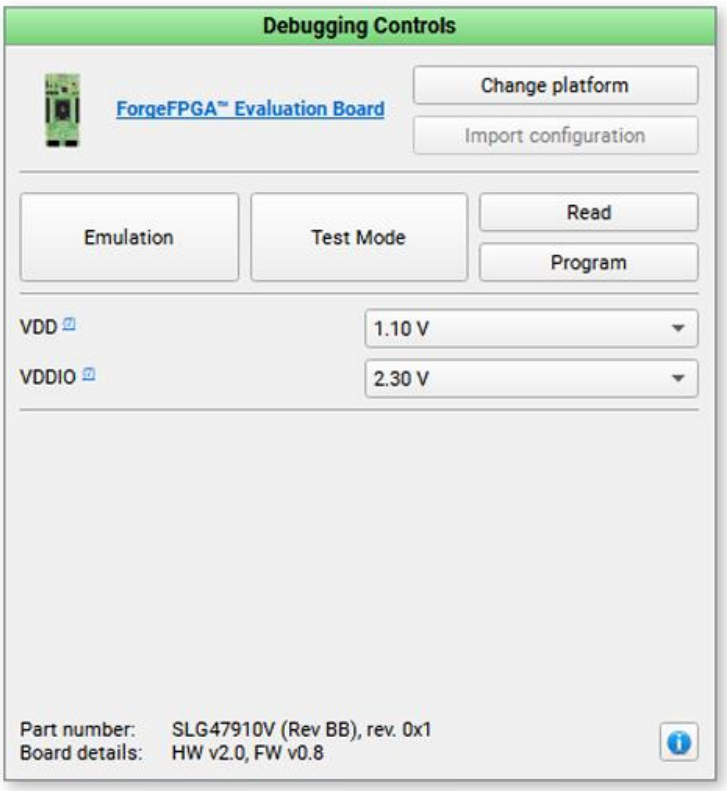

Figure 58. ForgeFPGA Evaluation Board Debugging Controls

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** – imports the configuration of test points from another platforms.

Chip Procedures

- **Emulation** – the current project loads to the chip (but not programmed) when the control is active and will be ready for test on the hardware board.
- **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, the 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 can control the power manually. Another feature of the Test Mode is that it can test the conditions from Flash Memory by selecting Test Mode (*)
- **Read** – reads the device using the hardware board or from the external flash memory.
- **Program** – programs the device or the external device with the current project.

Flash Procedures

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

- **Test Mode (*)** – loads data from flash to the device.
- **Read** – reads the chip project from the flash memory.
- **Program** – programs a flash memory with the current project.
- **Erase** – cleans the flash memory (erased flash memory address values will be read as 0xFF).

Generator Controls

- **Start/Pause/Stop** – used during Emulation to start, pause or stop the controls from running.
- **TP Map** – shows 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

- **LVDS Connection** –LVDS setting enables LVDS switches, configuring all channels simultaneously.
- **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 the SLG47920/21.

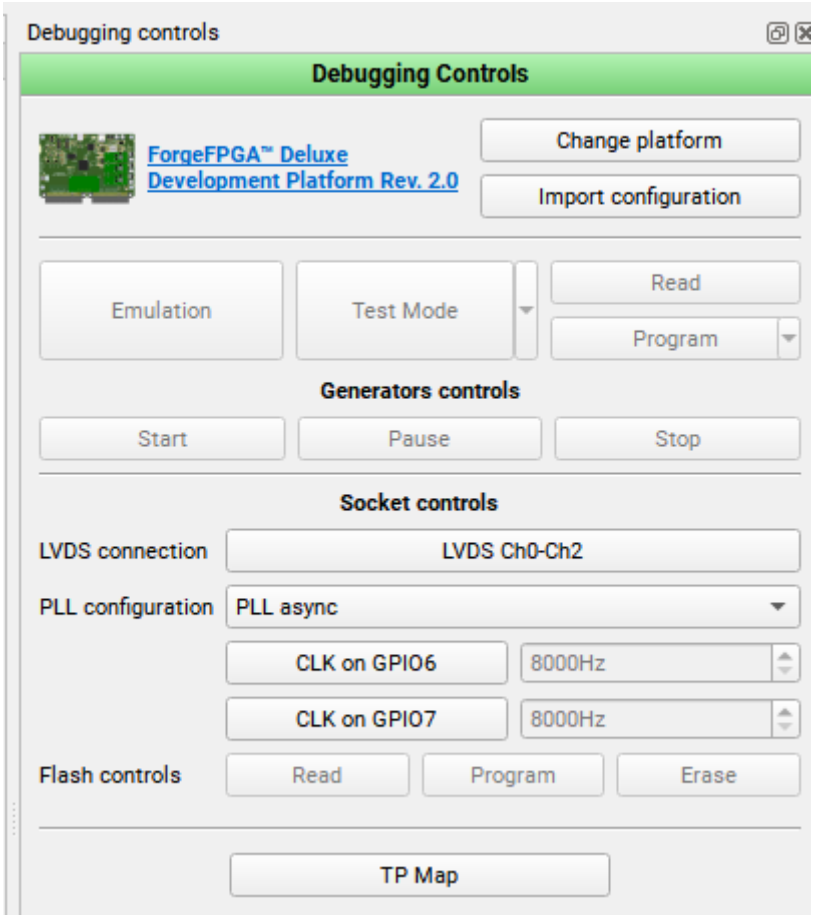

Figure 59. Debugging Controls for SLG47920/21

External Bitstream

External Bitstream for **MCU(Emulation)**, **OTP (Programming)**, and **FLASH (configuration from flash)** can be used after enabling the checkbox **External Bitstream**.

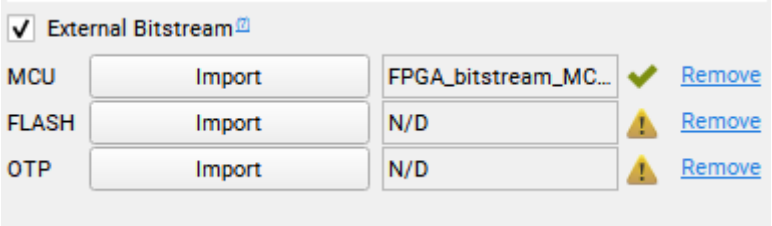

Figure 60. External Bitstream Option

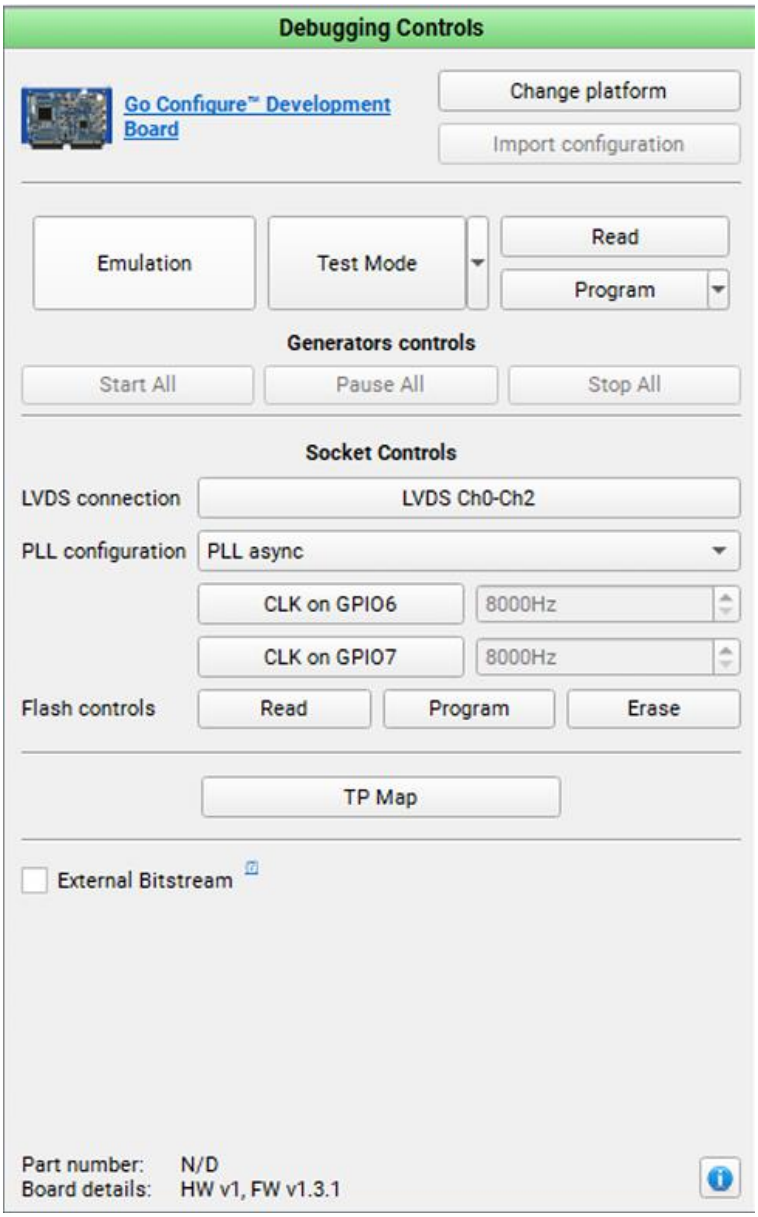

Figure 61. Go Configure Development Board with ForgeFPGA Device

Voltage level controls

V_{DD}/V_{DDIO} – 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, for example, the 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.

3.2.1 ForgeFPGA Board Controls

A comparison of controls available on ForgeFPGA boards can be found in Table 1.

Table 1. ForgeFPGA Board Controls

Feature	ForgeFPGA Evaluation Board	ForgeFPGA Deluxe Development Board	Go Configure Development Board
Debug Configuration	✓	✓	✓
Chip Procedures	✓	✓	✓
Flash Procedures		✓	✓
Generator controls		✓	✓
TP map		✓	✓
Socket controls		✓	✓
External bitstream		✓	✓
Voltage level controls	✓		
Info details	✓	✓	✓

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 the device input signals can be managed.

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

Use the context menu on GPIOs with a right click to see the connectivity options (see Figure 62).

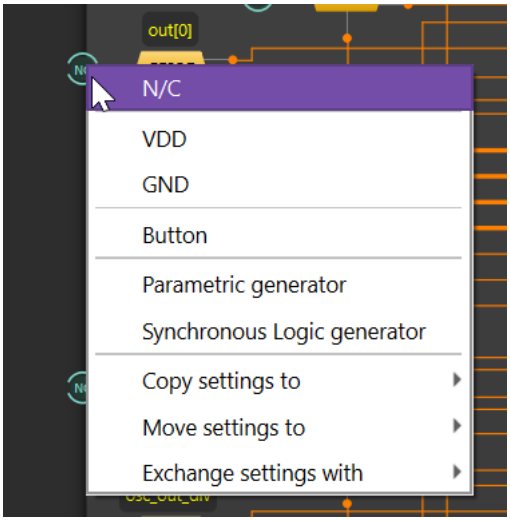

Figure 62. Debug Tool

- **NC (not connected)**

Figure 63. NC (not connected)

- **VDD**

Figure 64. Set to VDD

- **GND**

Figure 65. 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 V_{DD}, GND, or Hi-Z. If the user wants the GPIO to be a fixed connection to V_{DD}, then the **LATCH** option should be selected. The **LATCH** option will make sure that the GPIO is connected to a particular signal (V_{DD}, GND, Hi-Z). This will enable the button to be latched to V_{DD} unless it is changed (see Figure 66, Figure 67, and Figure 68).

The default connection option is V_{DD} or 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 67, the Upper Connection "U" corresponds to Hi-Z, and the Bottom Connection "B" corresponds to GND is noted.

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 67).

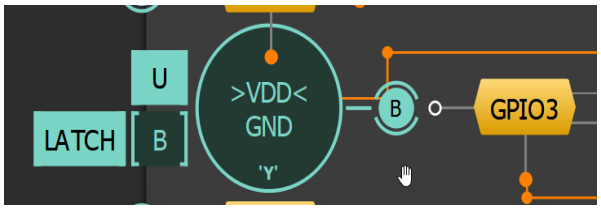

Figure 66. Latched Button with Upper Connection as V_{DD}

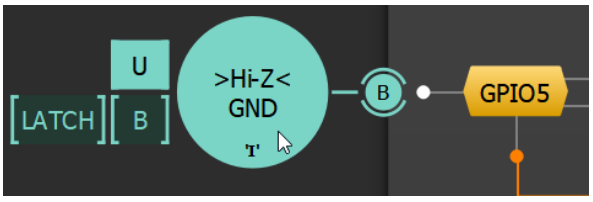

Figure 67. Unlatched Button with Upper Connection as Hi-Z

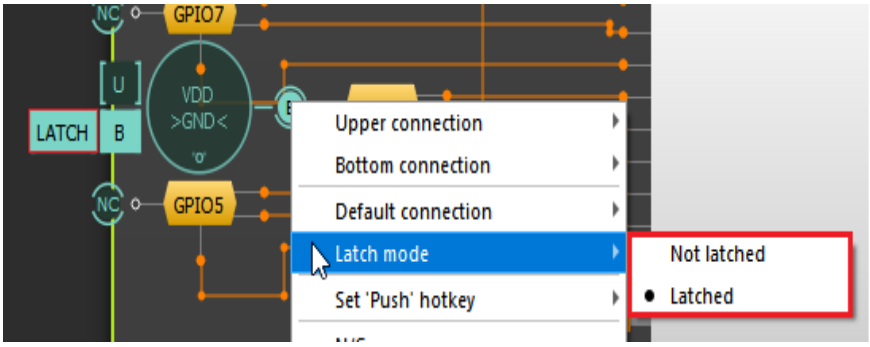

Figure 68. Context Menu Options for Configurable Button

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.

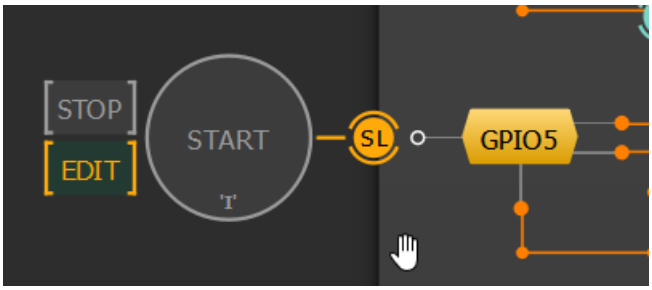

Figure 69. Synchronous Logic Generator

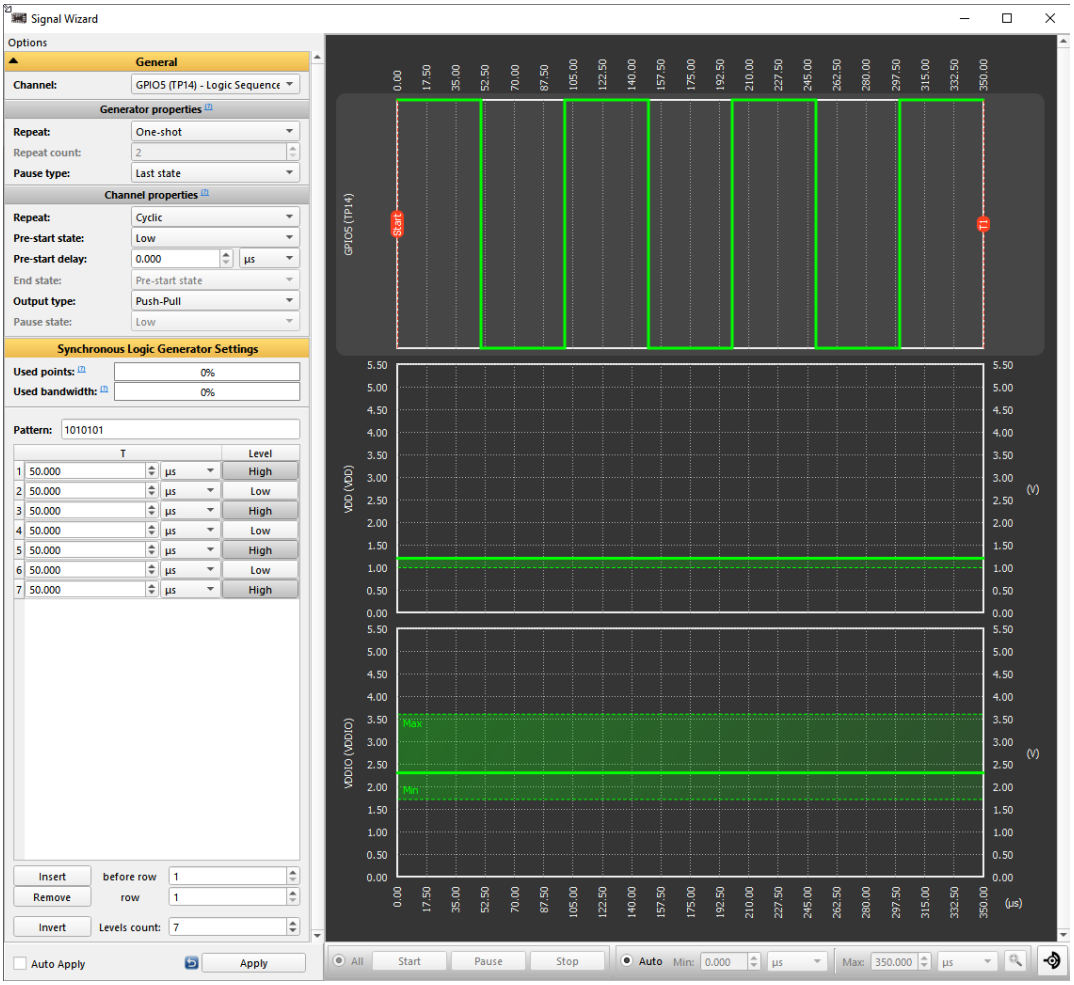

Figure 70. Signal Wizard for Synchronous Logic Generator

Below are the different features in the Signal Wizard for Synchronous Logic Generator (see Figure 70).

- **Used Points** – percentage of the total number of points already used by all patterns on all channels. A point is an instance where the Logic Generator initiates a state change on at least one channel.
- **Used Bandwidth** – percentage of used resources needed to successfully execute the generator's pattern.
- **Pattern** – 0 = low, 1 = high level. The pattern of pulse levels can be set.
- **Repeat** – One shot, Cyclic, Custom.
- **T/Level** – set the duration of each pulse.
- **Insert** – inserts pulse before selected position.
- **Remove** – removes pulse from the selected position.
- **Invert** – inverts the pattern from (for example, 1010’ to ‘0101’).
- **Level Count** – inserts 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.

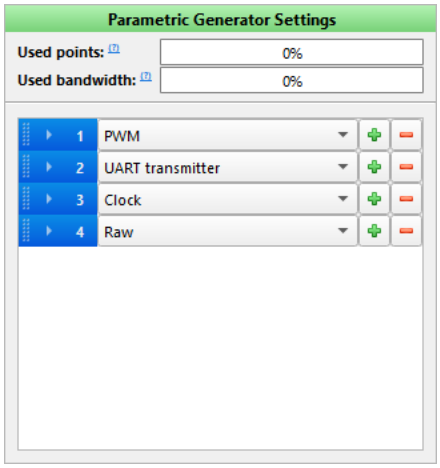

Figure 71. Parametric Generator Command Editor

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** –percentage of the total duration in the high state.
  - **Repeats** – pattern repeat count.

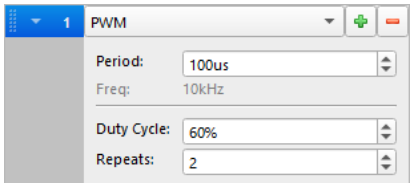

Figure 72. PWM Command Editor

- **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.

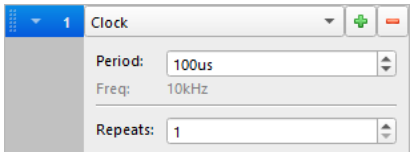

Figure 73. Clock Command Editor

- **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.

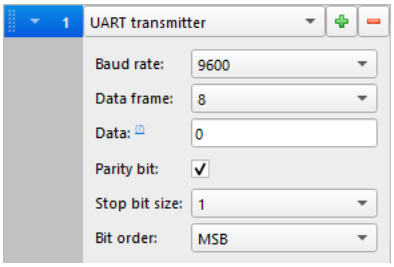

Figure 74. Clock Command Editor

- **Raw** – this pattern 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 capture of 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.

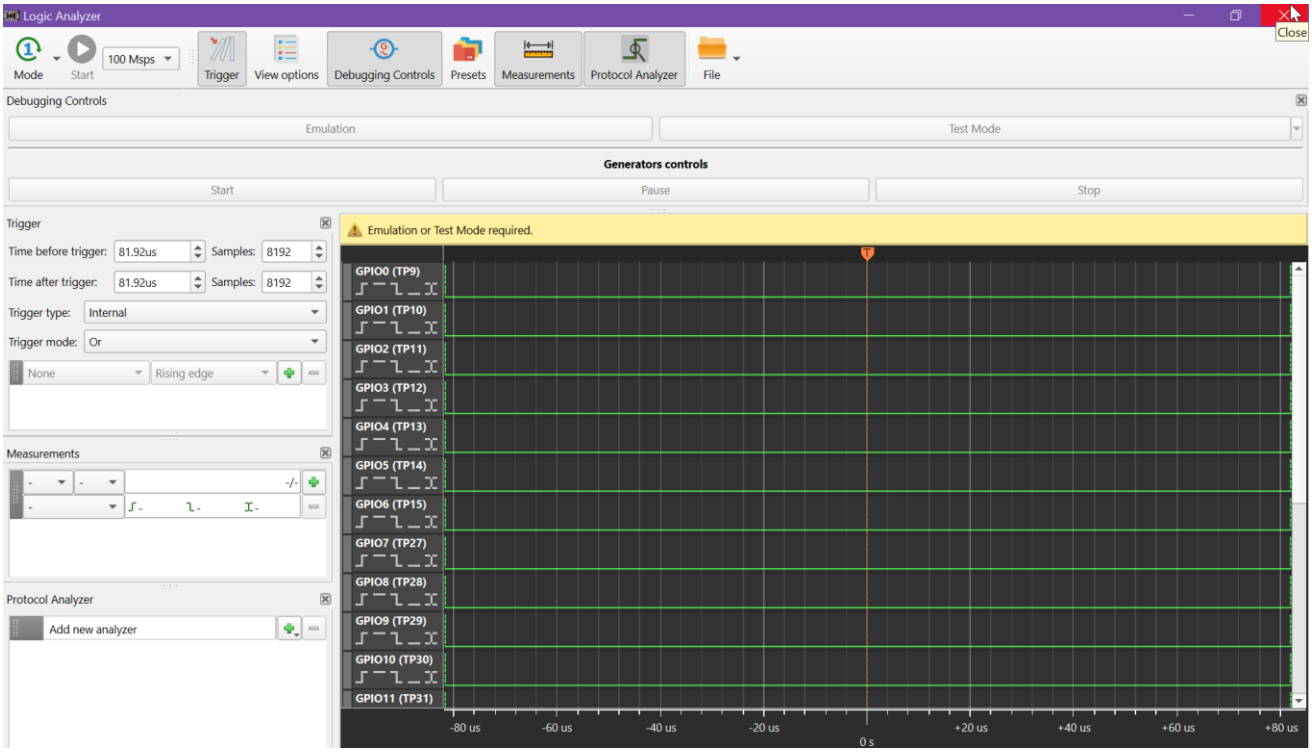

Figure 75. Logic Analyzer

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 Main Operational Controls

- **Mode** – three operating modes are available:
  - **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.
- **Start** – launches 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 Triggers

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

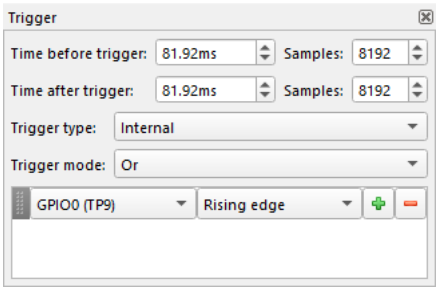

Figure 76. Trigger Parameters

The trigger type can also be configured (see Figure 76):

- **Hardware button** – the trigger is activated by pressing a physical button on the board
- **Internal** – the trigger is activated when the trigger conditions, which are set in the trigger list, are met. Additional settings are available for the Internal trigger type:
  - **Trigger mode** – depending on the selected option (And or Or), any or all the trigger conditions added to the trigger list will be met.
  - **Trigger list** – adds or removes triggers, assigns the trigger to the channel, and selects the trigger conditions amongst the following: **Rising edge**, **Falling edge**, **Both edges**, **High state**, and **Low state**.

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

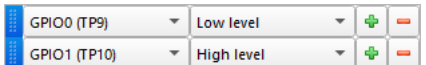

Figure 77. Trigger Conditions

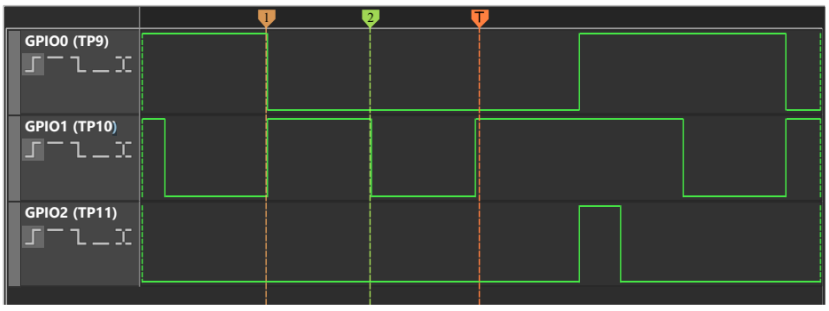

Figure 78. Trigger Configuration

3.4.3 Debugging Controls

The **Debugging Control** panel is responsible for **Emulation**, **Test Mode**, and **Generators controls** functionality.

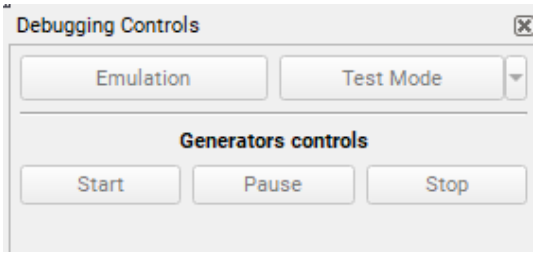

Figure 79. Debugging Controls

The channels in the **View options** panel can be shown or hidden using the **Show all** or **Hide all** buttons.

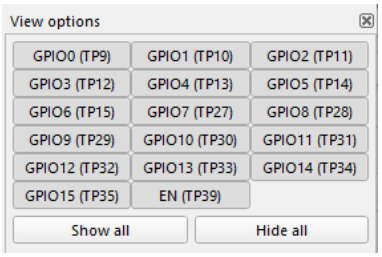

Figure 80. View Options

3.4.4 Presets

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

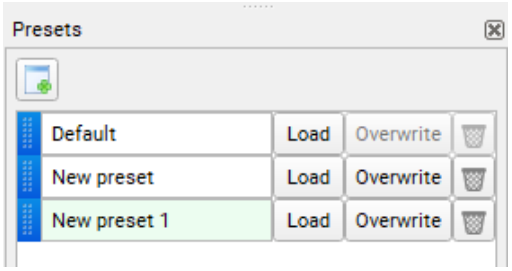

Figure 81. Logic Analyzer Configuration Presets

- **New Preset** – creates a new preset for the current Logic Analyzer configuration.
- **Load** – loads a selected preset configuration.
- **Overwrite** – overwrites preset with the current Logic Analyzer configuration.
- **Delete Preset** – removes the selected preset.
- **Default** – loads the default preset of the Logic Analyzer window.
- **Autosave** [time] – saves the modified preset each time the Logic Analyzer window, the Debug tool, or ForgeFPGA workshop is closed.

3.4.5 Protocol Analyzer

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

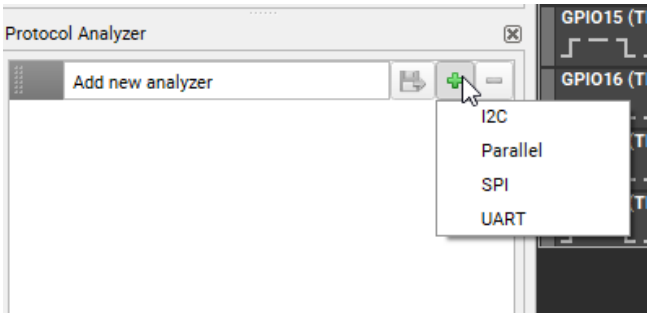

Figure 82. Protocol Analyzer Decode Options

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

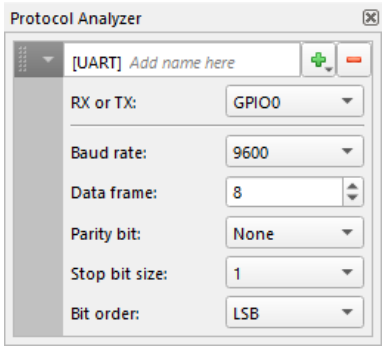

Figure 83. Protocol Analyzer Options

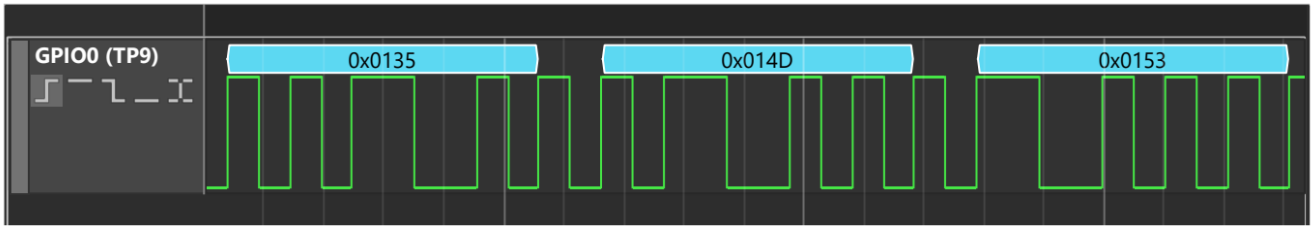

Figure 84. Decoded Data

Next, 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.

3.4.6 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.

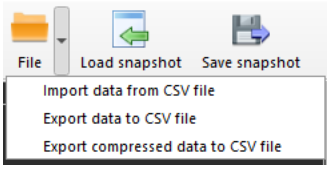

Figure 85. Import/Export from/to CSV Format

3.4.7 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.

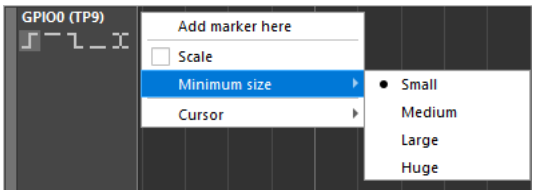

Figure 86. Plot Widget

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.

3.4.8 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.

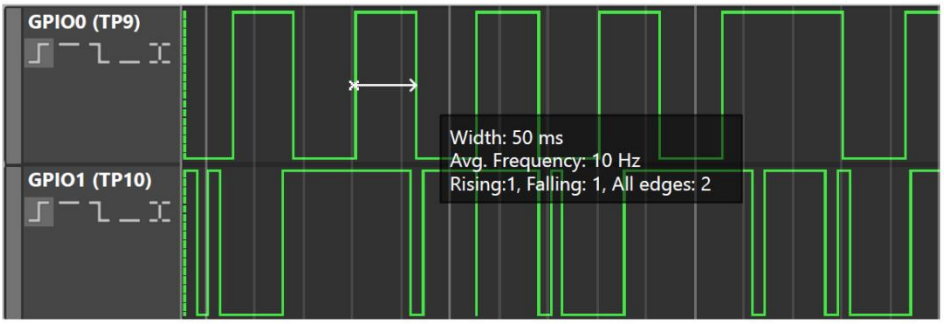

Figure 87. Half Period Cursor

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.

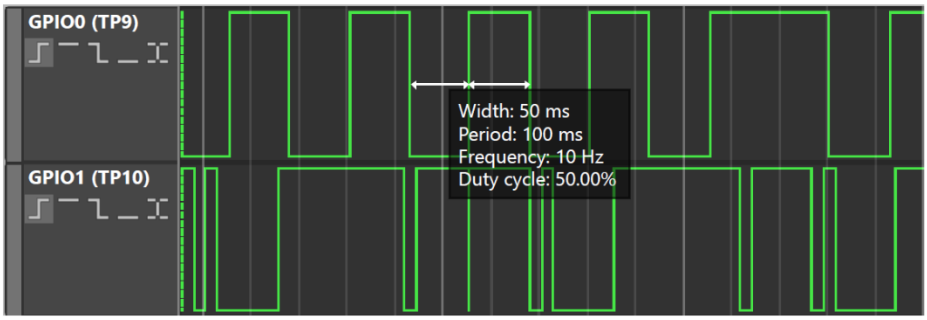

Figure 88. Period Cursor

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 from which the measurement should start and hover over the edge to which the measurement should extend. This provides the total duration of the measured section, the calculated average frequency, and the quantity of rising and falling edges as well as their sum.

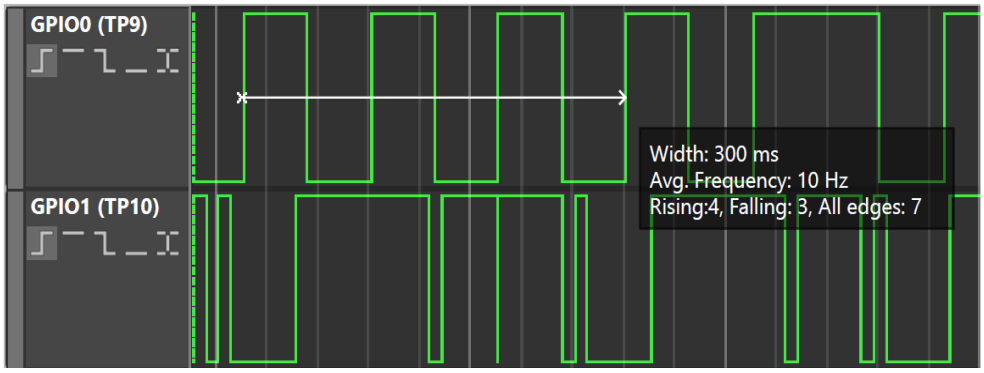

Figure 89. Adjustable Period Cursor for Measurement

The width between the edges on the distinct waveforms can also be measured.

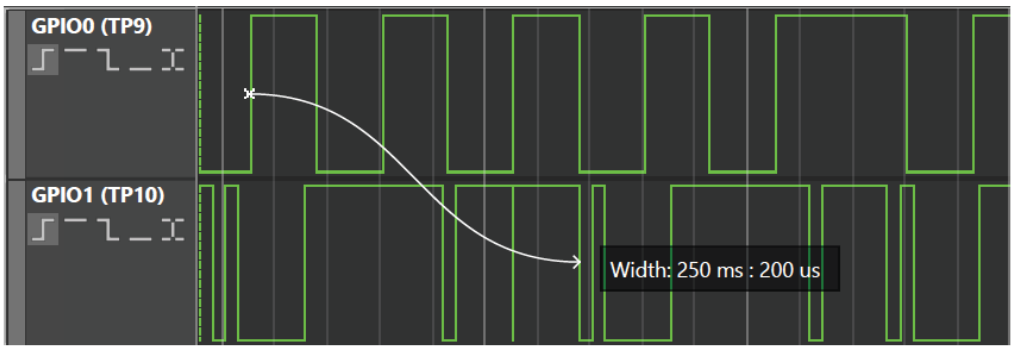

Figure 90. Adjustable Period Cursor Measurement between Waveforms

3.4.9 Markers

The following actions with markers can be performed:

- 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.

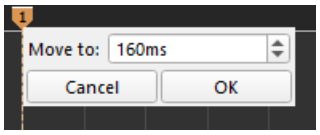

Figure 91. Moving Markers with Context Menu

- 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.

3.4.9.1 Marker Measurements

- **Measurements** – period and frequency. The frequency value is rounded to four decimals.
- **Additional measurements** – count 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.

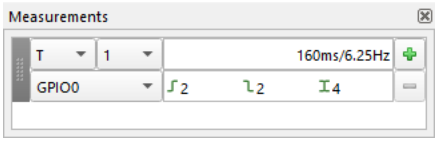

Figure 92. Marker Measurements

4. 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 is used to read and write via the UART protocol. This UART terminal is available only on the ForgeFPGA Evaluation Board (see section 3.1.2 ForgeFPGA Evaluation Board (SLG7EVBFORGE)).

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

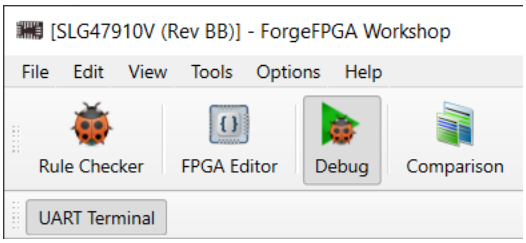

Figure 93. UART Terminal Tool

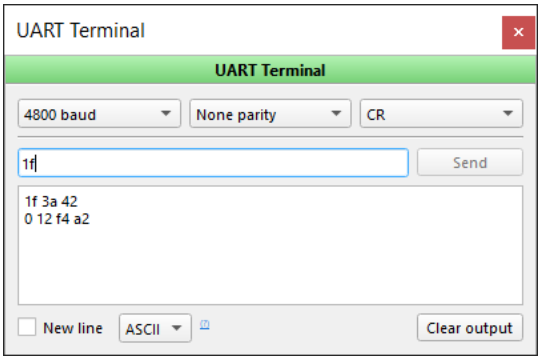

Figure 94. UART Terminal Window

The following fields are available under UART Terminal for modification:

- **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** and **Carriage Return**. This option is available in ASCII mode only.
- **New line** – adds 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 and paste operations. The output field only supports copying of the received data.

5. DAC Tool

The DAC Tool allows configuration of the voltage level on special analog output socket pins.

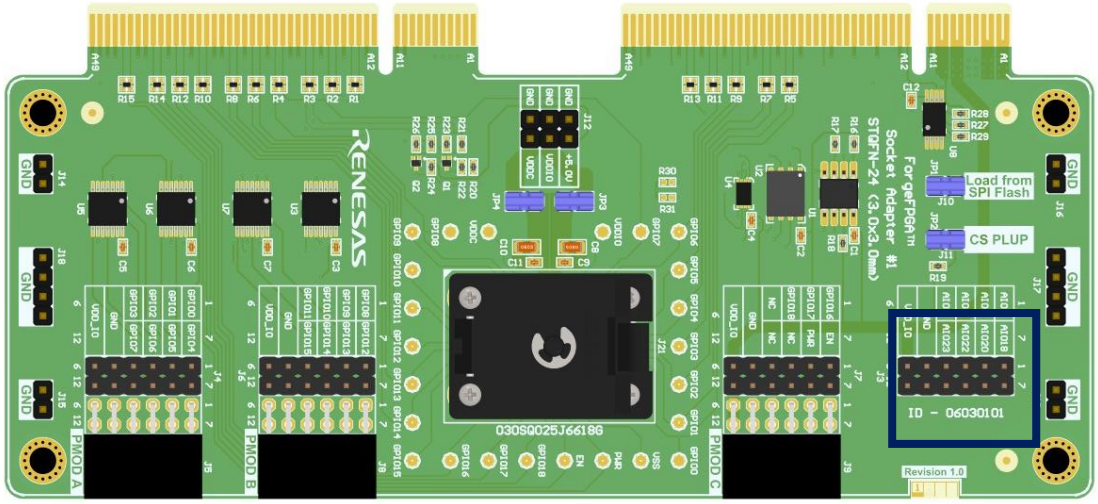

Figure 95. Analog I/O Pins on ForgeFPGA Socket Adapter

Start by selecting the appropriate development platform and clicking on the **DAC Tool** button from the toolbar (see Figure 96).

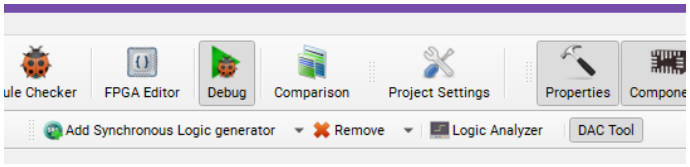

Figure 96. DAC Tool on the Toolbar

All Analog I/O pins are displayed by default. Use an **Analog I/O n** button to enable a respective output pin and use the numeric stepper on the right to set a desired voltage level on it (see Figure 97).

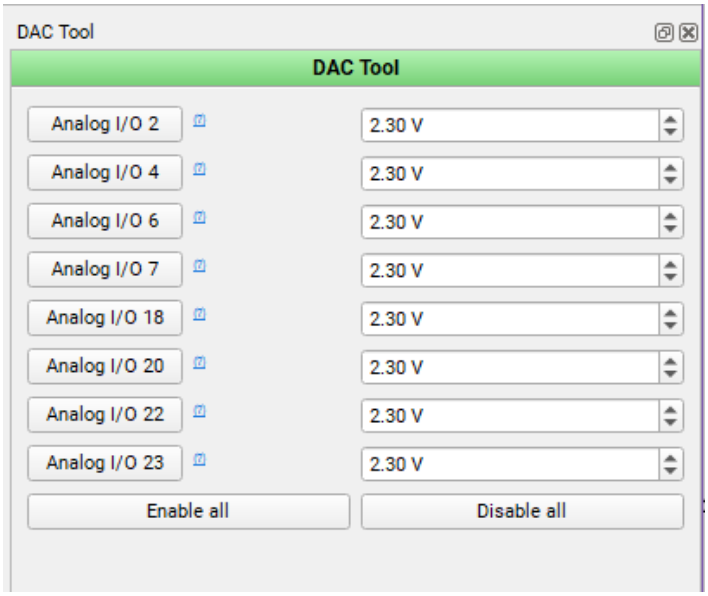

Figure 97. DAC Tool Settings

Note: The ability to set a voltage level on Analog I/O pins is enabled only if Emulation or Test Mode is activated.

6. HBRAM OTP Data Editor

The **HBRAM OTP Data Editor** allows modification of the bit values of BRAM and User OTP Macrocell registers. Open the tool from the toolbar or through the BRAM's **Properties** panel.

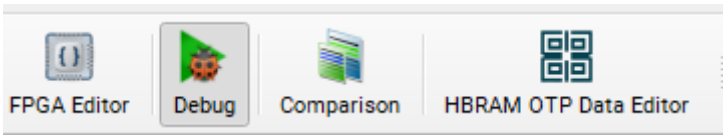

Figure 98. HBRAM OTP Data Editor on the Toolbar

This tool provides separate tables for configuring BRAM and User OTP Macrocell bits. The data can be displayed in either decimal or hexadecimal format.

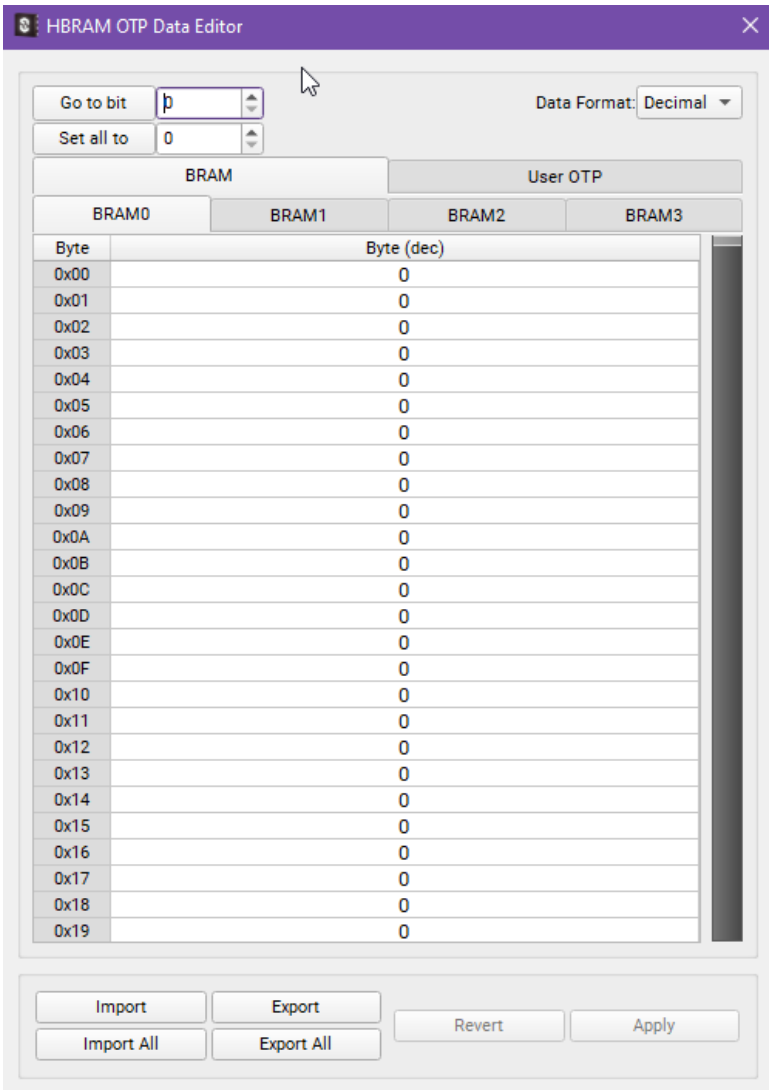

Figure 99. HBRAM OTP Data Editor (BRAM Tab)

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

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

7. 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 what data use via the **Load Bitstream** or **Load Custom Data** buttons (see Figure 100).

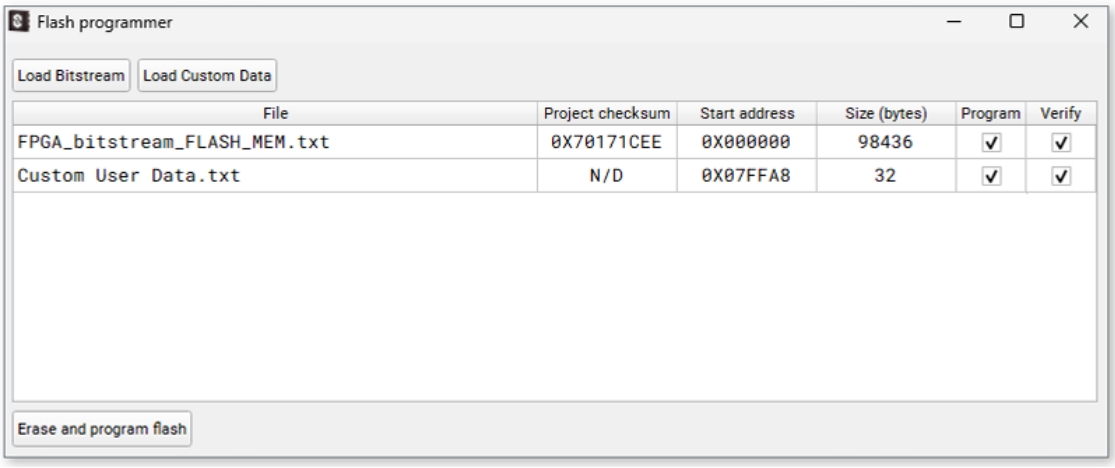

Figure 100. Flash Programmer Tool

- **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 the 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 instructions for flash parameter configuration. Specify the external flash size and set the SPI frequency to match the device specifications (see Figure 101).

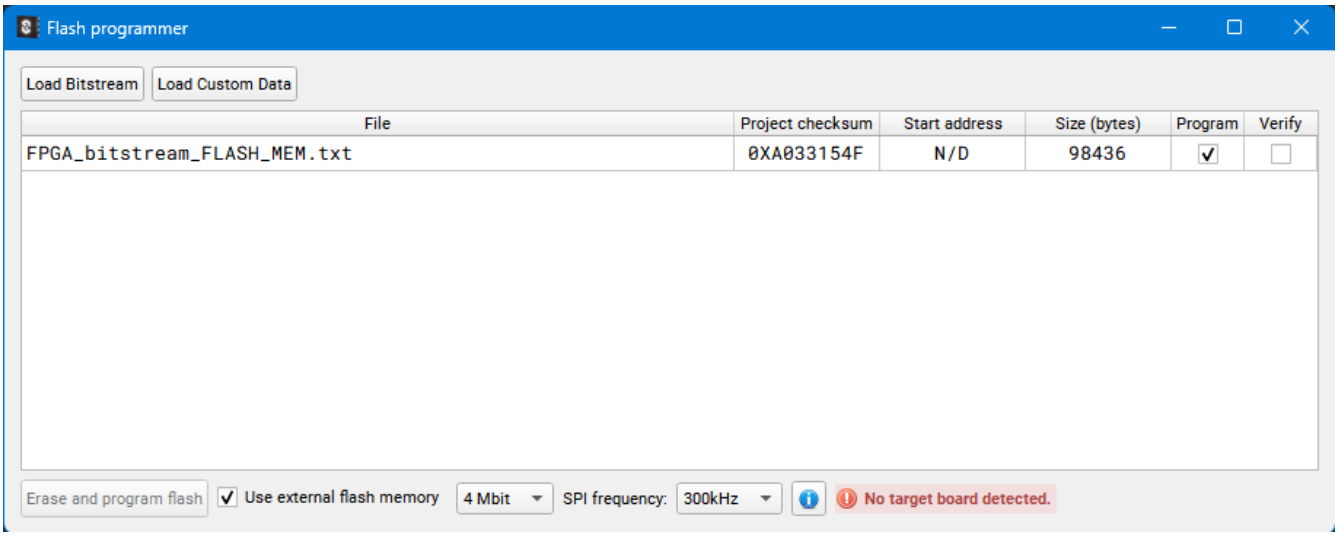

Figure 101. External Flash Programming Option

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.

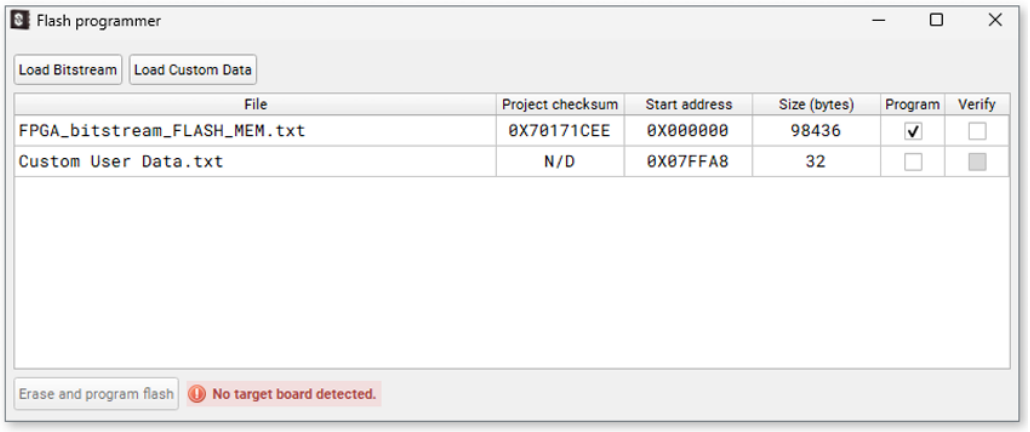

Figure 102. Flash Programmer Tool Info Hint Example

Optionally, enable the **Verify** checkbox. 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 for its location.

Note: The entire flash memory is erased before programming.

8. NVM Viewer

In the ForgeFPGA Workshop, the user can view all bits that correspond to each function in the NVM Viewer. ‘1’ and ‘0’ 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

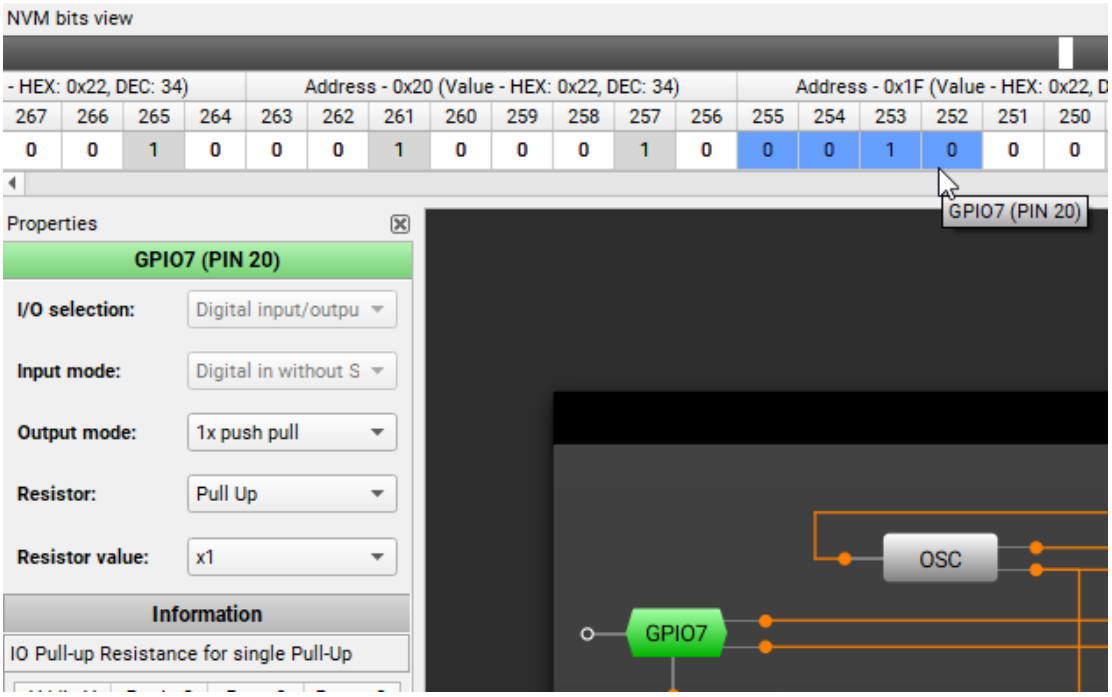

Figure 103. NVM Bits for GPIO7

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 103).

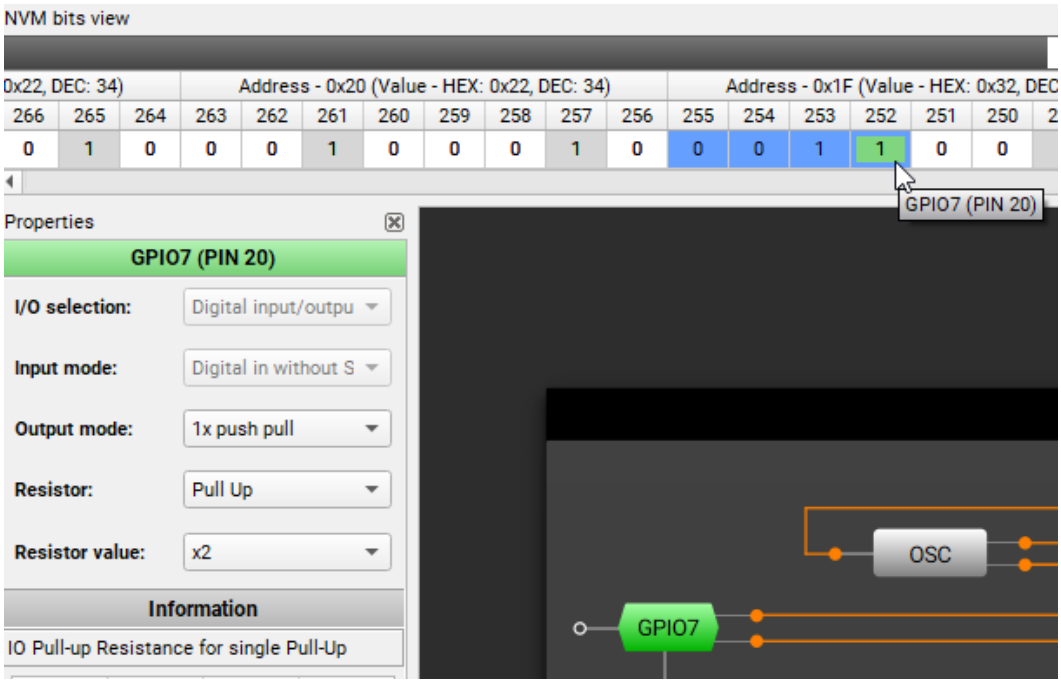

Figure 104. NVM Bits 0011 After Property Has Been Modified

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 104).

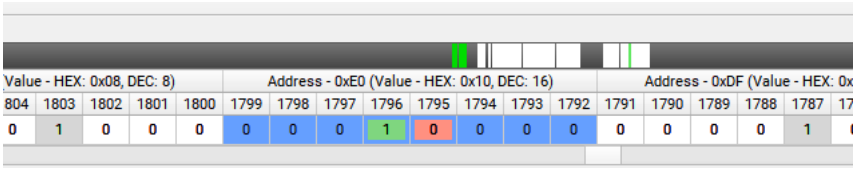

Figure 105. NVM Viewer Top Bar

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.

9. References

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

Download our free ForgeFPGA Designer software [1] and follow the steps in this user guide. The user can reference [2] for the datasheets, user manuals and guides. Renesas Electronics provides a complete library of application notes [2] featuring design examples, as well as explanations of features and blocks within the Renesas IC. Visit the Renesas ForgeFPGA product page to download the following:

[1] Go Configure Software Hub, Software Download

[2] Product Selector: ForgeFPGA Low-Density FPGAs (select the product; for datasheets, user manuals and guides, and application notes, go to the **Documentation** tab; for individual socket card user manuals and the *Go Configure Development Board R1.0 Manual*, go to the **Boards & Kits** tab).

[3] AN-FG-026 How to use Full-Chip Simulation in ForgeFPGA Design

[4] ForgeFPGA Advanced Development Board R1.2 User Manual (ForgeFPGA Deluxe Development Board User Manual)

[5] ForgeFPGA Socket Adapter #1 User Manual

[6] ForgeFPGA Evaluation Board R2.0 User Manual

[7] ForgeFPGA Software Simulation User Guide

10. Revision History

Revision	Date	Description
2.03	Aug 20, 2026	Added new section ▪ Part Number Migration ▪ ForgeFPGA Board control comparison table ▪ Updated software features till software v6.55 ▪ AES Encryption & Bitstream content management ▪ New information on new software Hub window ▪ Go Configure Development Board Debugging controls Updated information in sections ▪ Floorplan specifications ▪ Macrocell Editor section Corrected typos Minor Screenshot updates
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 ▪ Go Configure Development Board ▪ Flash Programmer ▪ Placement Constraints ▪ Full Chip Simulation ▪ Software Support
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
2.00	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
1.00	May 20, 2024	Initial release.

A. Appendix: Warnings

Below are a list of warnings and their respective meaning that the Yosys displays in the Logger section of the software after Synthesis (see section 2.4 RTL Synthesis) and Generate Bitstream (see section 2.5 Generating the 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>.
  Unmapped cells are left after synthesis, make sure to 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 the code.
- Multiple conflicting drivers for <N>: <K>.
  Several wires were defined to drive a port. Please correct the 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 the 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 the ports are mapped correctly.