How to Use Distributed Memory as a ROM
SLG47910V

This Application note describes how to design a ROM using distributed memory resources in SLG47910. Simulation waveforms generated by GTKWave software and Logic Analyzer in FPGA workshop can be used to verify the functionality of the design.

This application note comes complete with design files which can be found in the Reference section.

Contents

1. Terms and Definitions

FPGA	Field Programmable Gate Array
RAM	Random Access Memory
ROM	Read-Only Memory
FPGA Editor	Main FPGA design and simulation window
ForgeFPGA workshop	Main FPGA project window for debug and IO programming
Go Configure Software Hub	Main window for device selection
CLB	Configuration Logic Block

2. References

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

Download our free Go Configure Software Hub [1] to open the. ffpga design files [2] and view the proposed circuit design.

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

[2] AN-FG-017 How to use distributed memory as a ROM.ffpga, ForgeFPGA Design File

[3] ForgeFPGA SLG47910, Datasheet, Renesas Electronics

3. Introduction

This application shows how to use the SLG47910's distributed RAM to design a ROM, the distributed ROM block is shown in Figure 1. In such cases users can define the width and depth of the ROM and define the value of ROM data in the "initial" block in Verilog.

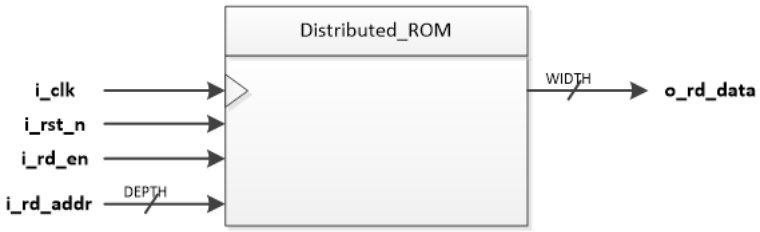

Figure 1: Distributed ROM

The following signal names are the PINs that are used in the design.

- i_clk – input clock signal
- i_rst_n – input negative reset signal
- i_rd_en – input read enable signal
- i_rd_addr – input read address signal

Using the ForgeFPGA Workshop software, the Verilog code was synthesized, and the bit stream was loaded on to the SLG47910V device. The functional waveforms below (see Figure 2) show the read-out data from address 0 to 7. The data output is 1 clock cycle delay from address input when read enable is high.

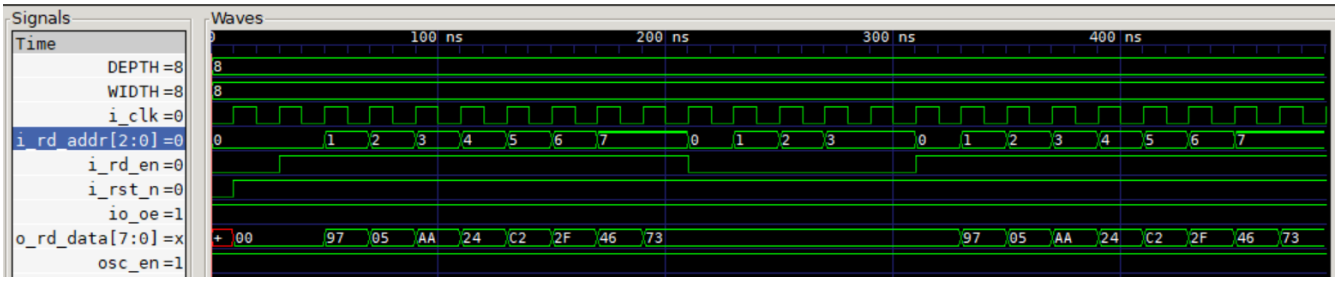

Figure 2: Functional Waveforms in GTKWave

4. ROM Description

Read-Only Memory (ROM) is a type of memory that stores fixed data, meaning its contents are pre-defined and cannot be modified during runtime. In FPGA design, ROM is commonly used for storing constant values such as lookup tables (LUTs), predefined waveforms, coefficients for digital signal processing (DSP), or microcontroller firmware. Distributed ROM is implemented using LUTs, making it an efficient choice for small ROMs. A ROM can be created in Verilog by defining values directly within the module using “initial”.

5. Distributed ROM Verilog Code

The Distributed ROM design is available for download (AN-FG-017 How to use distributed memory as a ROM.ffpga). It contains the complete distributed ROM design using the LUT module and the FPGA Core of the SLG47910V device.

Shown below is the (*top*) module named dist_rom.

```
(* top *) module dist_rom
#(
   parameter DEPTH = 8,
   parameter WIDTH = 8
)
(
  (* iopad_external_pin, clkbuf_inhibit *) input                 i_clk,
  (* iopad_external_pin *)   input                              i_rst_n,
  (* iopad_external_pin *)   output                             osc_en,
  (* iopad_external_pin *)   input                              i_rd_en,
  (* iopad_external_pin *)   input [$clog2(DEPTH)-1:0]          i_rd_addr,
  (* iopad_external_pin *)   output reg [WIDTH-1:0]             o_rd_data,
  (* iopad_external_pin *)   output                             io_oe
);

  reg [WIDTH-1:0] mem_rom [DEPTH-1:0];
    assign osc_en = 1'b1;
    assign io_oe = 1'b1;

  initial begin
          mem_rom[0] = 8'h97;
          mem_rom[1] = 8'h05;
          mem_rom[2] = 8'hAA;
          mem_rom[3] = 8'h24;
```

```
        mem_rom[4] = 8'hC2;
        mem_rom[5] = 8'h2F;
        mem_rom[6] = 8'h46;
        mem_rom[7] = 8'h73;
 end

 always @ (posedge i_clk) begin
  if(!i_rst_n) begin
        o_rd_data <= {WIDTH{1'b0}};
  end
  else if(i_rd_en) begin
        o_rd_data <= mem_rom[i_rd_addr];
  end
 end
endmodule
```

Note :

$clog2() is a Verilog function which returns the ceiling of the logarithm to base 2. In cases where the answer is a decimal number; the function rounds it to the next integer value. The argument can be an integer or an arbitrary sized vector value. The argument shall be treated as an unsigned value, and an argument value of 0 shall produce a result of 0.

Example :

$clog2(8) = 3

$clog2(15) = 3.907 ≈ 4

$clog2(4095) = 12

Hence, with the help of $clog2() function we can determine the maximum number of bits needed for the input read address.

6. Floorplan: CLB Utilization

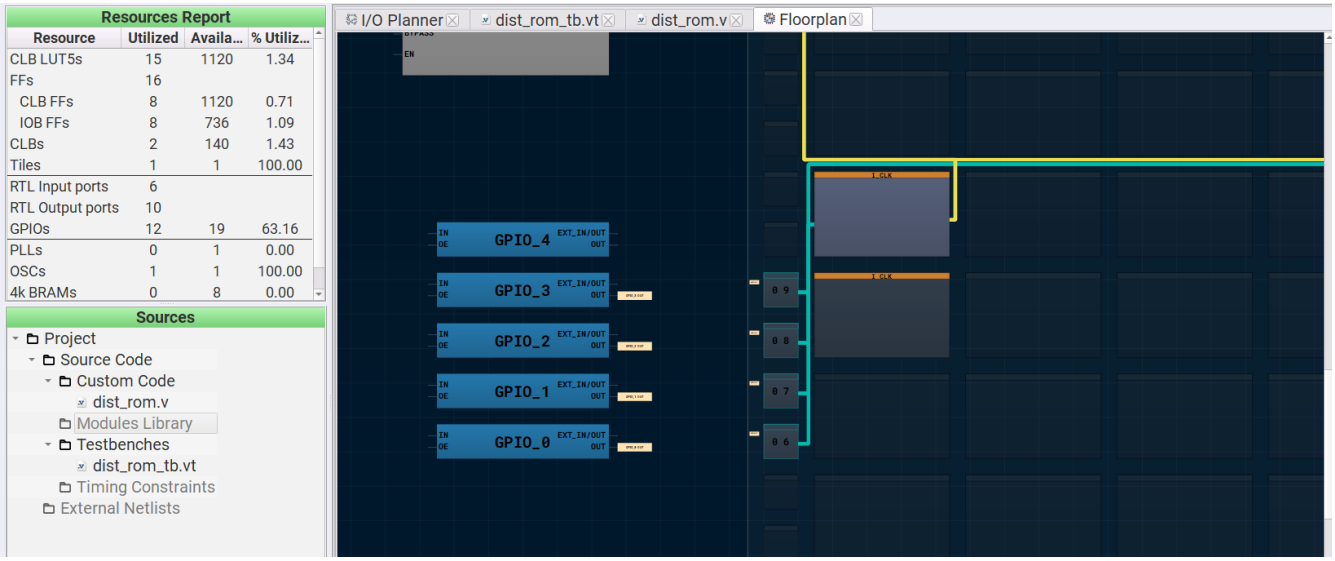

Figure 3: Distributed ROM CLB Utilization

The Floor planner tab in the FPGA Editor shows the placement of CLBs and FFs (Figure 3). The resource utilization is shown in the top left corner.

7. Logic Analyzer: ROM Read Out

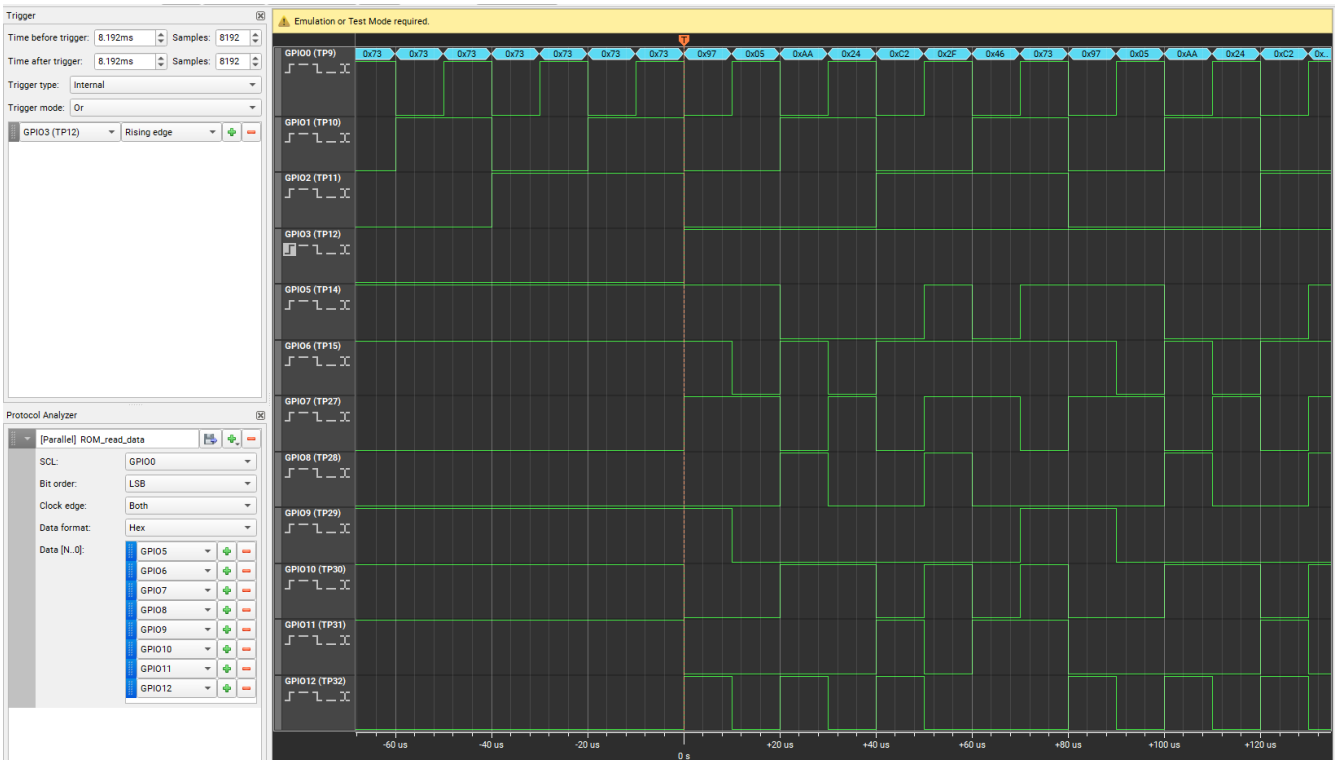

Figure 4: Distributed ROM Read Out

The Logic Analyzer in the FPGA Editor shows the ROM read out (Figure 3). We can set the GPIO0, GPIO1, GPIO2 as ROM read address, and use Synchronous Logic generator, to make sure the address input comes from 0~7, and GPIO3 as read enable, also use Synchronous Logic generator, to make sure when we set address from 0~7, we can see the read enable asserted, in the Logic Analyzer we can set the GPIO5, GPIO6, GPIO7, GPIO8, GPIO9, GPIO10, GPIO11, GPIO12 as the ROM data output to be a parallel bus, so that we can see the read out data in Logic Analyzer.

8. Design Steps

1. Launch the latest version of the Go Configure Software Hub. Select SLG47910V device and the ForgeFPGA Workshop software will load.
2. Download the design example AN-FG-017 How to use distributed memory as a ROM.ffpga. If you are not familiar with the ForgeFPGA Workshop software, review the How to use Distributed memory as a ROM application notes that cover the basic design steps.
3. Open the How to use distributed memory as a ROM.ffpga file after downloading.
4. Open the FPGA editor and review the Verilog code and the testbench code. Change the Width and Depth of ROM you would like to use and define the value in initial block.
5. Open the IO planner tab on the FPGA editor and review the pin assignment.
6. Next select the Synthesize button on the lower left side of the FPGA editor. Select the Generate Bitstream
7. button on the lower left side of the FPGA editor. Check the Logger and Issues tabs to make sure that the bitstream was generated correctly.
8. Now click on the Floorplan tab and see the CLB utilization (Figure 3). Press the Ctrl and the mouse wheel to zoom-in. Confirm that the IOs selected in the IO Planner are shown in the floorplan.
9. Now click on the Simulate Testbench button at the upper top. The GTKWave will automatically open if there are no Syntax errors in the testbench. Check logger for errors.
10. In the GTKWave software, select the signals you want to view and Click Insert on the left corner to insert the signals in the wave window. Once the desired signals are selected, click on Reload (Figure 2).
11. You can observe the ROM data read out in the waveform displayed in the GTKWave software. The same results can also be observed in the Logic Analyzer of the ForgeFPGA Workshop software.

9. Conclusions

This application note shows how the SLG47910 can be used to implement a ROM, and the ROM width and depth could be modified by users according to the project need, the input read address can be determined using $clog2() function accurately. This testcase is available for download (AN-FG-017 How to use distributed memory as a ROM.ffpga). If interested, please contact the ForgeFPGA Business Support Team.

10. Revision History

Revision	Date	Description
1.00	Mar 24, 2025	Initial release.

IMPORTANT NOTICE AND DISCLAIMER

RENESAS ELECTRONICS CORPORATION AND ITS SUBSIDIARIES (“RENESAS”) PROVIDES TECHNICAL

SPECIFICATIONS AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD-PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for developers who are designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only to develop an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third-party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising from your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

(Disclaimer Rev.1.01 Mar 2025)

Corporate Headquarters

TOYOSU FORESIA,3-2-24 Toyosu,

Koto-ku, Tokyo 135-0061, Japan

www.renesas.com

Contact Information

For further information on a product, technology, the most up-to-data version of a document, or your nearest sales office, please visit www.renesas.com/contact-us/.

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks are registered Trademarks are the property of their respective owners.