UART to 7-Segment Display

SLG47910

This application shows how to design a UART to Seven-Segment Display. We are using a PmodSSD for the 2-digit seven segment display connected to the Evaluation Board. The working of the UART Terminal is also discussed in this AN. Simulation waveforms generated by GTKWave software 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

Terms and Definitions

FPGA	Field Programmable Gate Array
UART	Universal Asynchronous Receiver Transmitter
FPGA Core	Circuit Block that contains the digital array macro cells
ForgeFPGA Workshop	Top level FPGA display and control window
FPGA Editor	Main FPGA design and simulation window
CLB	Configuration Logic Block
EVB	Evaluation Board
Pmod	Peripheral Module Interface

References

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

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

[1] ForgeFPGA Designer Software, Software Download and User Guide

[2] AN-FG-012 UART to 7seg.ffpga, ForgeFPGA Design File

[3] SLG47910, Preliminary Datasheet, Renesas Electronics

1. UART

UART or Universal Asynchronous Receiver-Transmitter is a commonly utilized communication protocol between devices. It provides a direct way to communicate with an FPGA, allowing for the exchange of commands between a device which have a UART interface and the FPGA. With proper configuration, UART is compatible with various serial protocols that require the transmission and reception of serial data. In serial communication, data is transmitted bit by bit using a single line or wire, and in bi-directional communication, two wires are employed to enable successful serial data transfer.

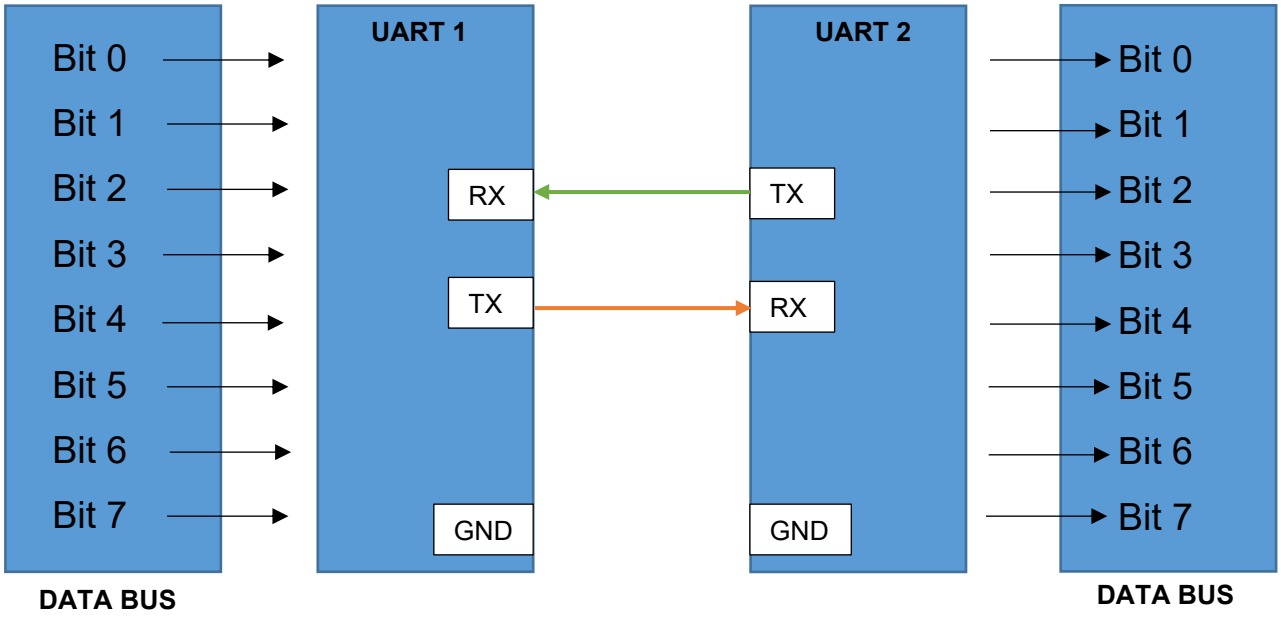


Figure 1: UART Data Transfer

The two signals of each UART device are named:

- Transmitter (Tx)
- Receiver (Rx)

The main purpose of a transmitter and receiver line for each device is to transmit and receive serial data intended for serial communication (see Figure 1). For a UART, the baud rate needs to be set the same on both the transmitting and the receiving device. UART devices have a few important parameters used for setting different configurations (see Figure 2):

1. **Baud Rate**: It is the rate at which information is transferred between communication channels. It can also be defined as the maximum number of bits per seconds to be transferred. Some common Baud Rates for an UART device is 9600,19200 and 115200.

2. **Number of Data Bits**: Data bits are contained inside a Data Frame. It can be 5 bits up to 8 bits long if parity bit is used. If no parity bit is used, then the Data Frame can be 9 bits long. The parity bit can be appended after the data is sent.
3. **Stop Bit**: To signal the end of the data packet, the sending UART drives the data transmission line from a low voltage to a high voltage. The Stop Bit is most preferably set to 1. Stop Bits are 1, 1.5 and 2 bits.

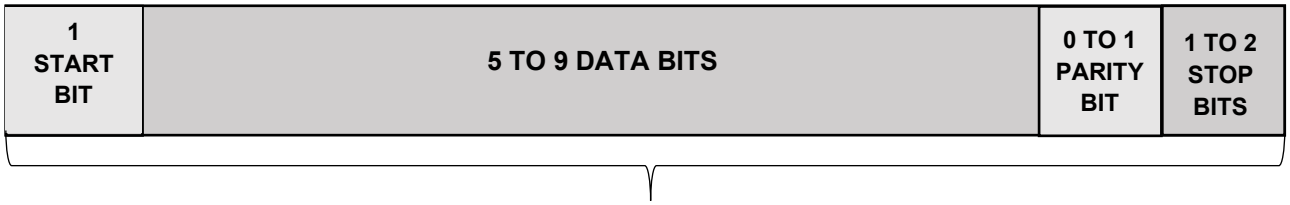


Figure 2: Data Frame for Receiver

4. **Parity Bit**: Parity is used by the receiving UART to detect any changes that may have occurred during transmission. Once the data frame is received, the receiving UART counts the number of bits with a value of 1 and checks if the total is odd or even based on the value of the parity bit. In case of even parity (parity bit set to 0), the total number of 1 bit in the data frame should be even, whereas in case of odd parity (parity bit set to 1), the total number of 1 bit in the data frame should be odd. The UART considers the transmission error-free when the parity bit matches the data. However, if the total is odd when the parity bit is set to 0 or even when the parity bit is set to 1, the UART detects that the data frame has been modified.

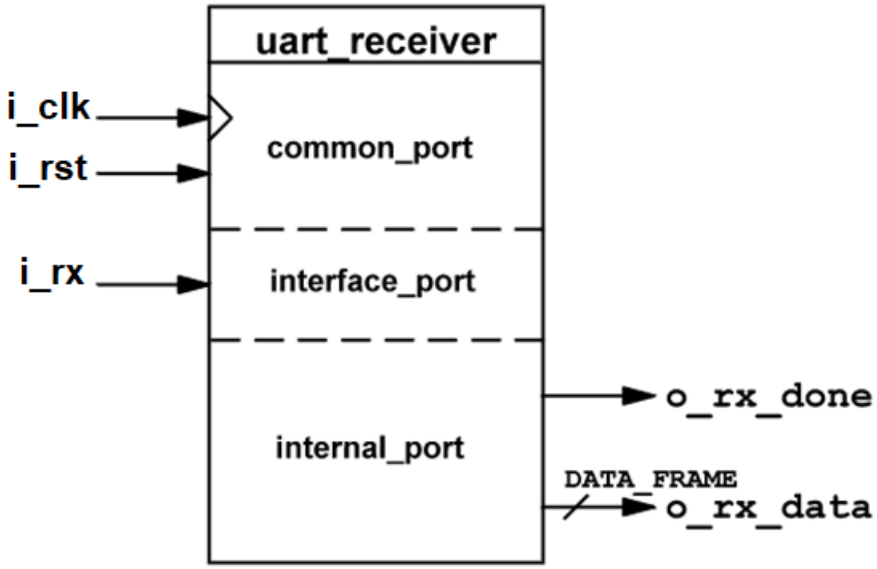


Figure 3: UART Receiver Block

2. Seven Segment Display

The seven-segment display is made up of seven LEDs that are arranged in a rectangular shape. Each LED is referred to as “segment” because it forms part of a numerical digit when illuminated. In some cases, an extra 8th LED may be included in the same package to indicate a decimal point (DP). When multiple seven-segment displays are connected to display numbers the DP LED can be used to differentiate between whole numbers and decimals. The displays common pin is generally used to identify which type of seven-segment display it is. As

each LED has two connecting pins, one called the “Anode” and the other called the “Cathode”, there are therefore two types of LED seven-segment display called: **Common Cathode** (CC) and **Common Anode** (CA).

In the common cathode display, all the cathode connections of the LED segments are joined together to logic “0” or ground. The individual segments are illuminated by application of a “HIGH”, or logic “1” signal via a current limiting resistor to forward bias the individual Anode terminals (a-g). In the common anode display, all the anode connections of the LED segments are joined together to logic “1”. The individual segments are illuminated by applying a ground, logic “0” or “LOW” signal via a suitable current limiting resistor to the Cathode of the segment (a-g). See Figure 4

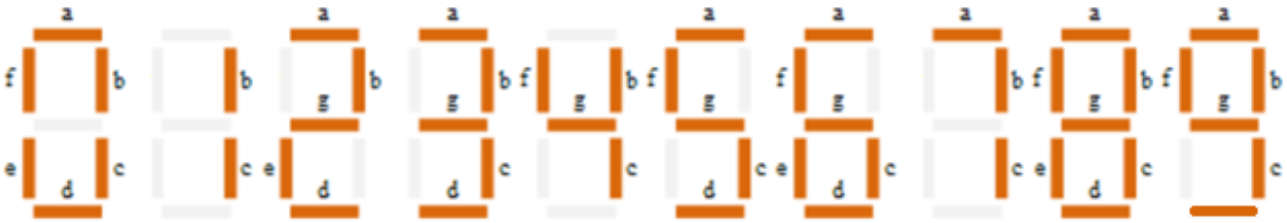


Figure 4: Digital Segments for all numbers

2.1 Seven-Segment Display Truth Table

Decimal Digit	Individual Segments Illuminated						
	a	b	c	d	e	f	g
0	H	H	H	H	H	H	L
1	L	H	H	L	L	L	L
2	H	H	L	H	H	L	H
3	H	H	H	H	L	L	H
4	L	H	H	L	L	H	H
5	L	H	H	H	L	H	H
6	H	L	H	H	H	H	H
7	H	H	H	L	L	L	L
8	H	H	H	H	H	H	H
9	H	H	H	L	L	H	H

3. Ingredients

- SLG47910 Device
- Latest Revision of the ForgeFPGA Workshop software
- ForgeFPGA Evaluation Board and cables
- PmodSSD Device (2-Digits)
- Jump Wire

4. Verilog Code

The UART to seven-Segment design is available for download (AN-FG-012 UART to 7seg.ffpga). It contains one top module which has instances of two IP Modules: UART Receiver and the seven-Segment display. These two modules are connected, and the necessary parameters are specified. The UART Receiver IP module has two more sub-modules inside which is used to define the baud rate and to define the working of the receiver.

Shown below is the (*top*) module named `uart_to_7seg`. The Verilog code for UART to seven-segment can be found in the complete design example. It is available for download (AN-FG-012 UART to 7seg.ffpga)

Multiple `always` block in the Verilog code allows the user to configure the counter and the UART FSM structure to ensure communication between blocks and data flow. The signals in the UART Receiver IP Module can be seen in Figure 3.

```
(* top *) module uart_to_7seg #(
  parameter IN_CLK_HZ         = 50_000_000,
  parameter DATA_FRAME        = 8,          // number of data bits
  parameter BAUD_RATE         = 115200,     // transmitting speed
  parameter OVERSAMPLING_MODE = 16,         // bit offset or overlap
  parameter STOP_BIT          = 1,          // length of stop bit
  parameter LSB               = 1,          // determine serial data transfer 1=
MSB to LSB
  parameter SEL_CA            = 1           // common cathode
) (
// Main inputs
  (* iopad_external_pin, clkbuf_inhibit *) input i_clk,
  (* iopad_external_pin *) input i_por,
// OSC config outputs
  (* iopad_external_pin *) output osc_ctrl_en,

// Custom IO
  (* iopad_external_pin *) input  i_rx,
  (* iopad_external_pin *) output rx_oe,
  (* iopad_external_pin *) output o_cat, // determines which active digit
  (* iopad_external_pin *) output o_out_a,
  (* iopad_external_pin *) output o_out_b,
  (* iopad_external_pin *) output o_out_c,
  (* iopad_external_pin *) output o_out_d,
  (* iopad_external_pin *) output o_out_e,
  (* iopad_external_pin *) output o_out_f,
  (* iopad_external_pin *) output o_out_g,
  (* iopad_external_pin *) output o_cat_oe,
  (* iopad_external_pin *) output o_out_a_oe,
  (* iopad_external_pin *) output o_out_b_oe,
  (* iopad_external_pin *) output o_out_c_oe,
  (* iopad_external_pin *) output o_out_d_oe,
```

```
  (* iopad_external_pin *) output o_out_e_oe,
  (* iopad_external_pin *) output o_out_f_oe,
  (* iopad_external_pin *) output o_out_g_oe
  );

  wire        w_rx_done;
  wire [7:0]  w_rx_data, w_segment;
  wire [1:0]  w_active_digit;
  reg  [11:0] r_counter;
  reg  [1:0]  r_rx;
  reg         r_tick;

  assign osc_ctrl_en   = 1'b1;

 //oe
      assign rx_oe = 0;
      assign o_cat_oe   = 1;
      assign o_out_a_oe = 1;
      assign o_out_b_oe = 1;
      assign o_out_c_oe = 1;
      assign o_out_d_oe = 1;
      assign o_out_e_oe = 1;
      assign o_out_f_oe = 1;
      assign o_out_g_oe = 1;
  // reset buffer
  wire w_rst;

  input_reset_buf impl_input_reset_buf (
    .i_clk        (i_clk),
    .i_por        (i_por),
    .o_rst        (w_rst)
  );

  //Synchronized rx to avoid metastability
  always @(posedge i_clk) begin
    if(w_rst) begin
      r_rx <= 2'b11;
    end else begin
      r_rx <= {r_rx[0], i_rx};
    end
  end

  assign o_cat = w_active_digit[1];

  always @(posedge i_clk) begin
    if (w_rst) begin
      r_counter <= 'h00;
      r_tick    <= 1'b0;
    end else begin
      if(r_counter <= 2499) begin
        r_counter <= r_counter + 1;
        r_tick    <= 1'b0;
      end else begin
        r_counter <= 'h00;
        r_tick    <= 1'b1;
      end
    end
  end

// instantiate the UART Receiver
  uart_receiver #(
```

```
    .IN_CLK_HZ          (IN_CLK_HZ         ),
    .DATA_FRAME         (DATA_FRAME        ),
    .BAUD_RATE          (BAUD_RATE         ),
    .OVERSAMPLING_MODE  (OVERSAMPLING_MODE),
    .STOP_BIT           (STOP_BIT          ),
    .LSB                (LSB               )
  ) impl_uart_rx (
    .i_clk              (i_clk    ),
    .i_rst              (w_rst    ),
    .i_rx               (r_rx[1]  ),
    .o_rx_data          (w_rx_data),
    .o_rx_done          (w_rx_done)
  );

// instantiate the 7-segment Module
  seven_seg_disp_ctrl_2d #(
    .SEL_CA (SEL_CA)
  ) impl_seven_seg_disp_ctrl_2d (
    .i_clk            (i_clk             ),
    .i_load           (w_rx_done         ),
    .i_en             (1'b1              ),
    .i_rst            (w_rst             ),
    .i_refresh_clock  (r_tick            ),
    .i_data           ({2'b00,w_rx_data}),
    .o_active_digit   (w_active_digit    ),
    .o_segment        (w_segment         )
  );

  assign o_out_a = w_segment[7];
  assign o_out_b = w_segment[6];
  assign o_out_c = w_segment[5];
  assign o_out_d = w_segment[4];
  assign o_out_e = w_segment[3];
  assign o_out_f = w_segment[2];
  assign o_out_g = w_segment[1];

endmodule
```

5. Floorplan: CLB Utilization

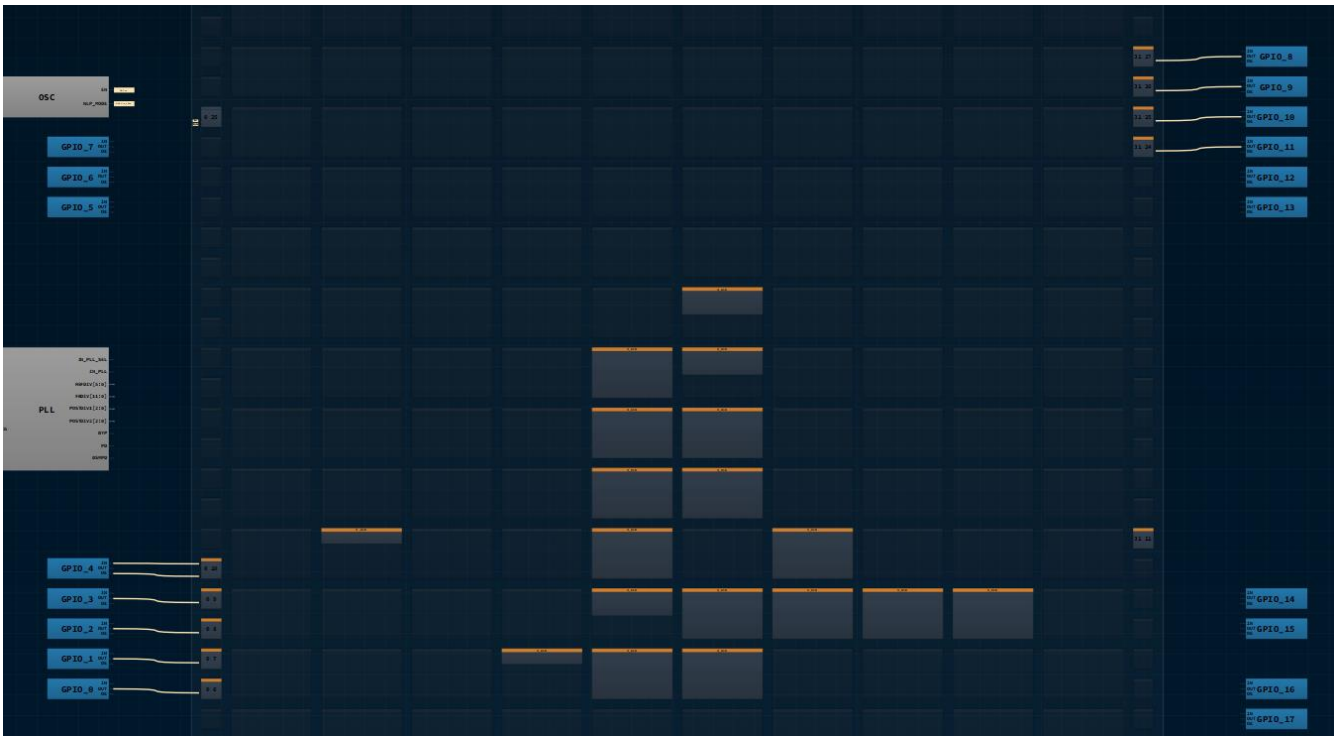


Figure 5: Floorplan

From Figure 5, we can see a part of the floorplan for this application note.

From the Verilog code, the user can observe which GPIOs are being used for this Application Note. This can also be observed under the floorplan tab in the software. The connection between each CLB in the floorplan can be observed by clicking on the CLB of interest and observing the yellow(input) and blue(output) wires connecting the CLBs internally.

6. Design Steps

1. Launch the latest version of the Go Configure Software Hub. Select the SLG47910V device and the ForgeFPGA Workshop software will load.

2. From the ForgeFPGA tool bar, select the FPGA Editor tab.

3. Enter the Verilog code into the HDL editor and save the code using the save button on the top left corner of the FPGA Editor. Make sure that there are five tabs containing different parts of the code. (See Figure 6)

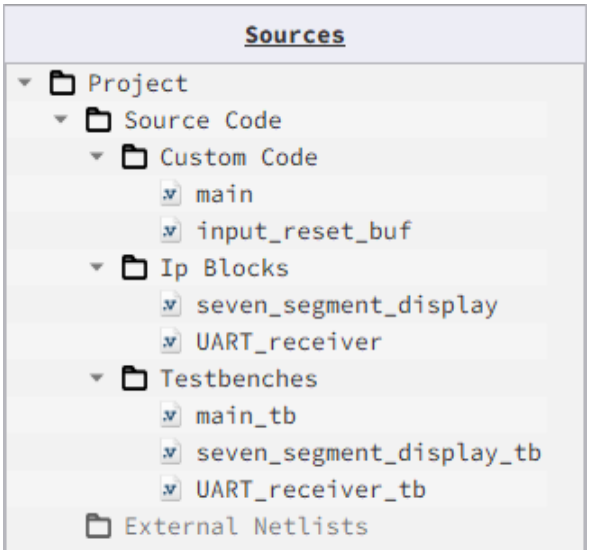
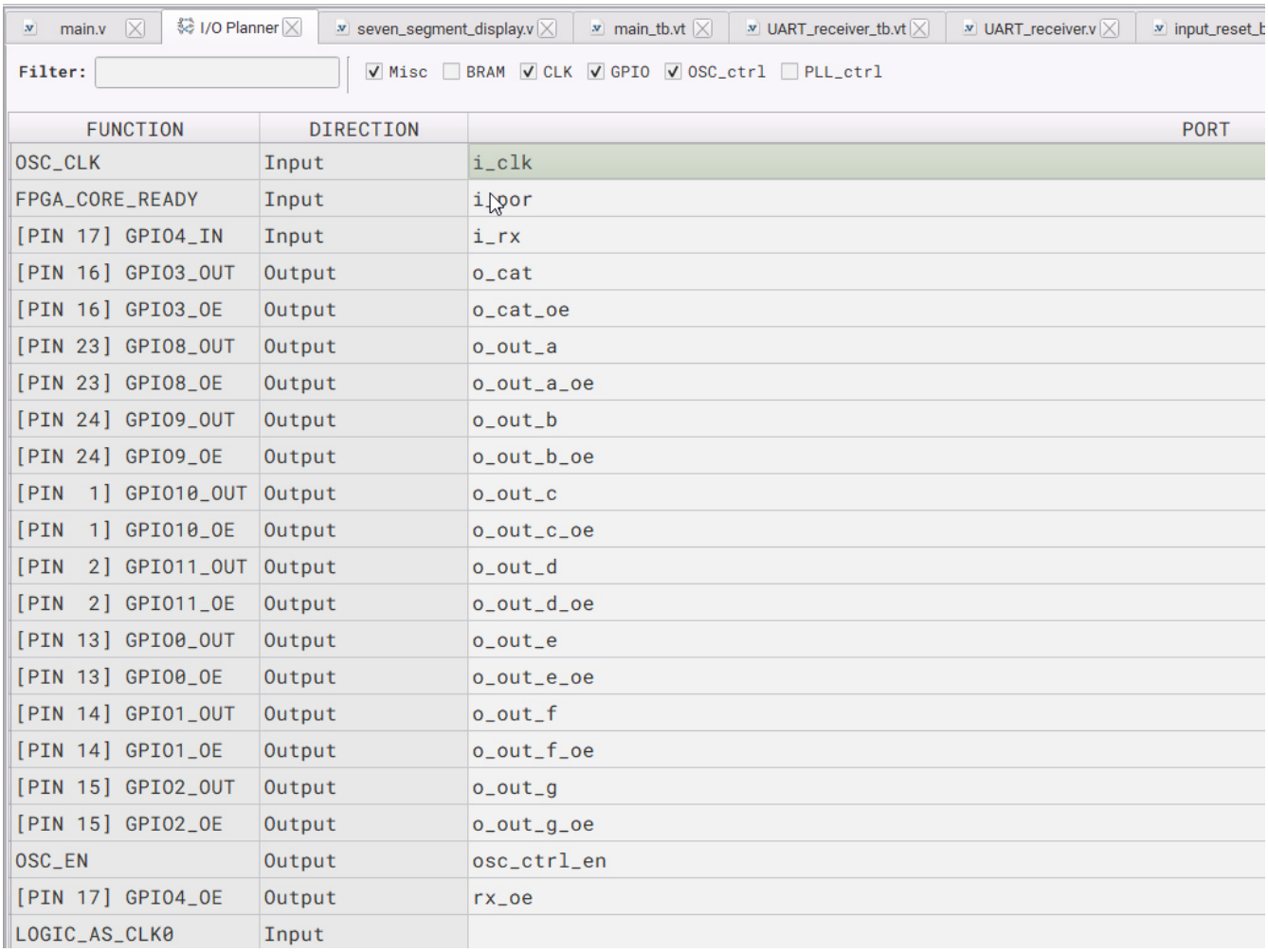


Figure 6: Source files for this project

4. Open the IO planner tab on the FPGA editor. Assign the IOs that are in the Verilog code to GPIO pins on the device and save. (See Figure 7)

5. Next select the Synthesize button on the lower left side of the FPGA editor.

6. Select the Generate Bitstream 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. Now click on the Floorplan tab and see the CLB utilization. Press the Ctrl and the mouse wheel to zoom-in. (Figure 5). Confirm that the IOs selected in the IO Planner (Figure 7) are shown in the floorplan.



FUNCTION	DIRECTION	PORT
OSC_CLK	Input	i_clk
FPGA_CORE_READY	Input	i_por
[PIN 17] GPIO4_IN	Input	i_rx
[PIN 16] GPIO3_OUT	Output	o_cat
[PIN 16] GPIO3_OE	Output	o_cat_oe
[PIN 23] GPIO8_OUT	Output	o_out_a
[PIN 23] GPIO8_OE	Output	o_out_a_oe
[PIN 24] GPIO9_OUT	Output	o_out_b
[PIN 24] GPIO9_OE	Output	o_out_b_oe
[PIN 1] GPIO10_OUT	Output	o_out_c
[PIN 1] GPIO10_OE	Output	o_out_c_oe
[PIN 2] GPIO11_OUT	Output	o_out_d
[PIN 2] GPIO11_OE	Output	o_out_d_oe
[PIN 13] GPIO0_OUT	Output	o_out_e
[PIN 13] GPIO0_OE	Output	o_out_e_oe
[PIN 14] GPIO1_OUT	Output	o_out_f
[PIN 14] GPIO1_OE	Output	o_out_f_oe
[PIN 15] GPIO2_OUT	Output	o_out_g
[PIN 15] GPIO2_OE	Output	o_out_g_oe
OSC_EN	Output	osc_ctrl_en
[PIN 17] GPIO4_OE	Output	rx_oe
LOGIC_AS_CLK0	Input	

Figure 7: IO Planner

7. This design file also comes with a testbench to verify the working of UART and the Seven-Segment display. The simulation waveforms can be verified by using the inbuilt GTKWave software. Uart_receiver_tb is the testbench for UART side of the design file and the seven_segment_tb is the other testbench used to verify the working of the seven_segment display. The user can launch the GTKWave software by clicking on the Simulate Testbench icon on the toolbar. This will automatically launch the GTKWave software provided there are no errors in your testbench. The errors/issues can be observed from the console at the bottom of window.

8. In the GTKWave software, the user can insert the desired signal in the window from the list of all the signals in the code and observe the waveform. Through the generated waveform the user can verify the functionality of UART and the seven-segment display without going through the hassle of connecting the development board. (See Section 7)

9. Once the user is satisfied with the waveform, the user can Debug the design file. Close the FPGA Editor and go to the ForgeFPGA widow. Selecting the Debug tab will enable the debug controls. Double click on the V_{DD} pin and set V_{DD} = 1.1V. Then double click on V_{DDIO} pin and set V_{DDIO} = 2.3V.

10. In the ForgeFPGA Workshop window, select Change platform on the Debugging Controls tab. Choose the ForgeFPGA Development Platform then select Emulation. The Emulation button will toggle the design on and off.

11.To observe the outputs of this application note, connect the Evaluation Board along with the PmodSSD connected to its GPIO. User the upper slots of the GPIO in the EVB to connect to the PmodSSD.

12. To send the data from the UART to the Pmod, we need to use the inbuilt UART Terminal. The EVB has a connection for the UART. The user needs to connect the **TX within the UART** on the board to **GPIO4** for the UART Terminal to work. (See Figure 8)

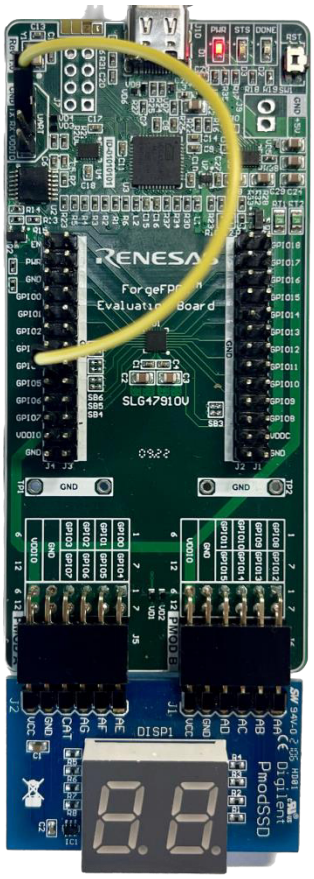


Figure 8: UART Connection on EVB

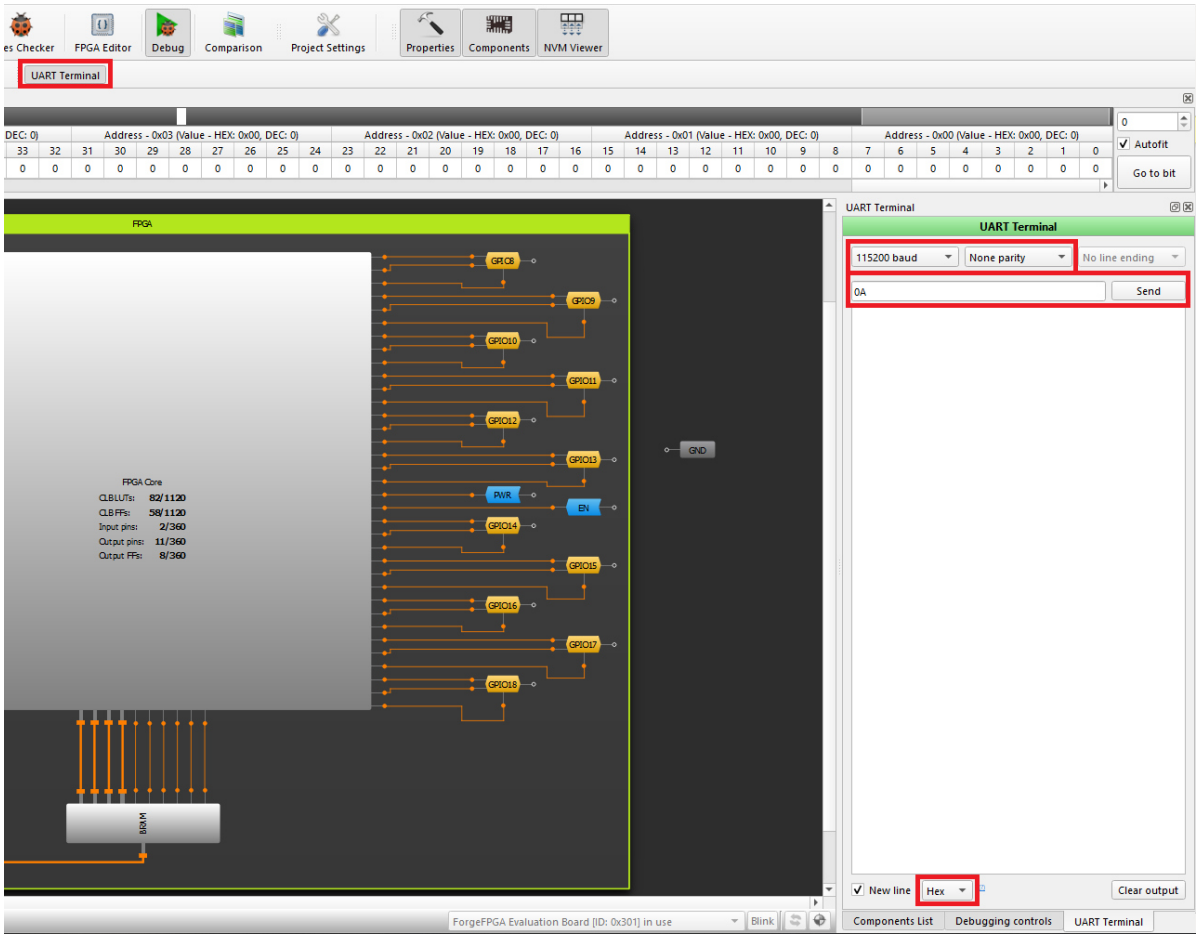


Figure 9: UART Terminal

13. The user can then send the desired data from the UART Terminal to the PmodSSD in Hexadecimal form. The user needs to set the Baud Rate as 115200 and set the Parity Bit to “None Parity”. The user can type the desired data (HEX) they want to send to the Pmod in the UART Terminal (See Figure 9) and then click Send to see it appear on the PmodSSD. (See Figure 10)

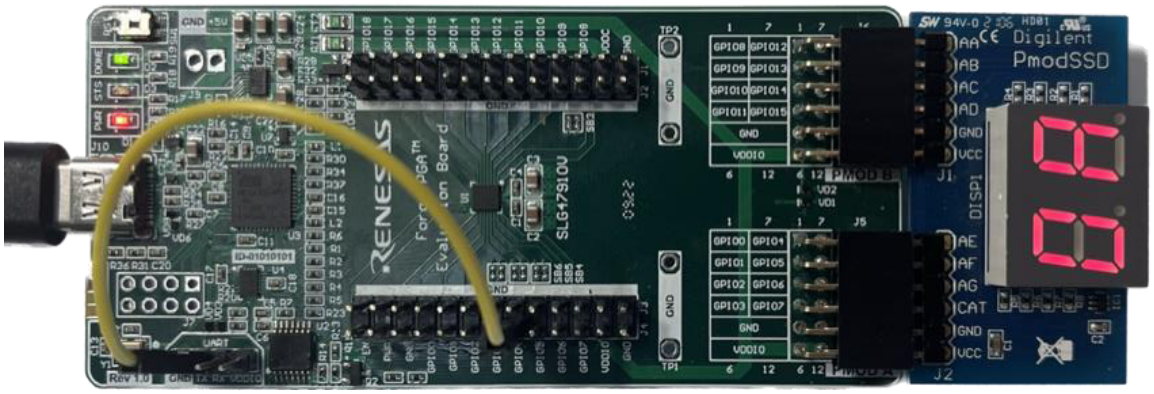
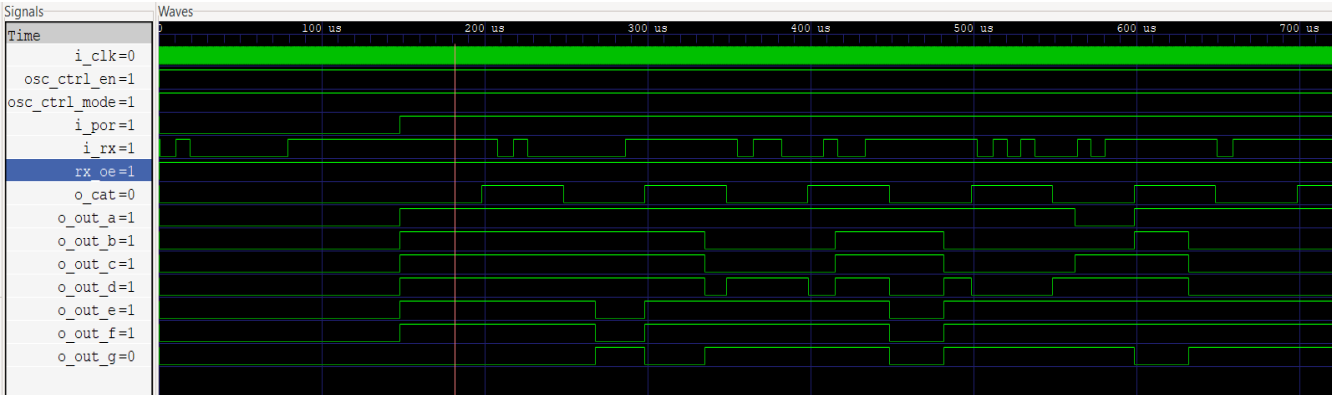


Figure 10: PmodSSD Results

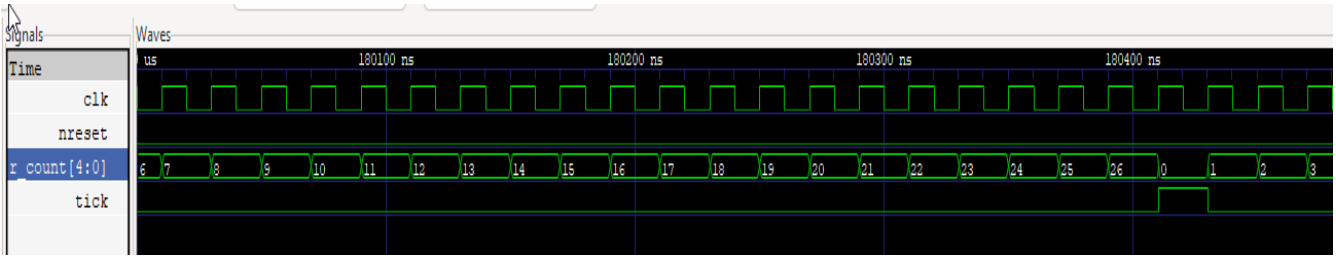
7. GTKWave Simulation Waveform

The design file has three sets of testbenches which can be used to verify the functionality of the design. There are three testbench files, uart_receiver_tb, seven_segment_display_tb and main_tb. Inside the uart_receiver_tb, we have two sub modules, baud_rate_generator and uart_receiver_rx. We can observe the waveform for each submodule in GTKWave to verify. The main_tb is used to verify the total functionality of the seven-segment receiving data from the UART.

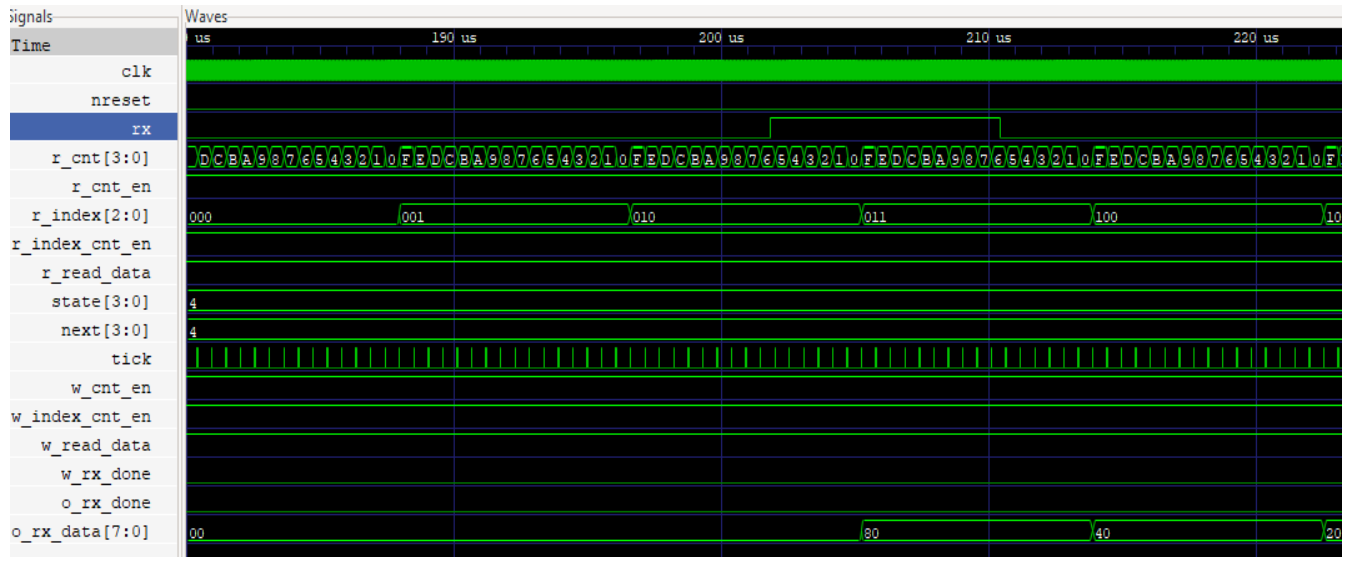
7.1 UART to 7-Segment (main_tb)



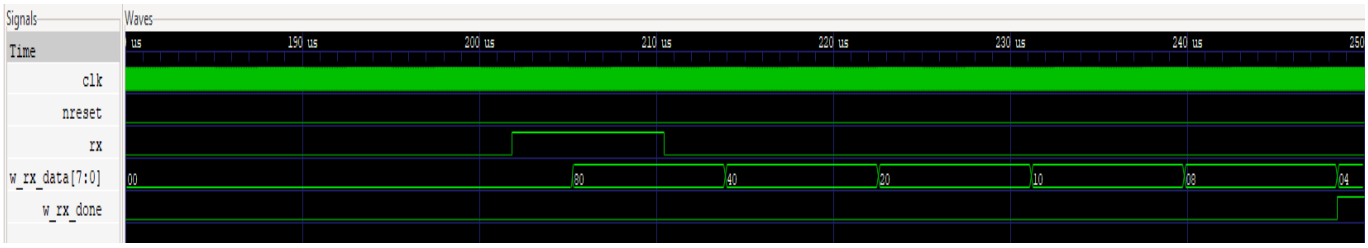
7.2 Baud Rate Generator



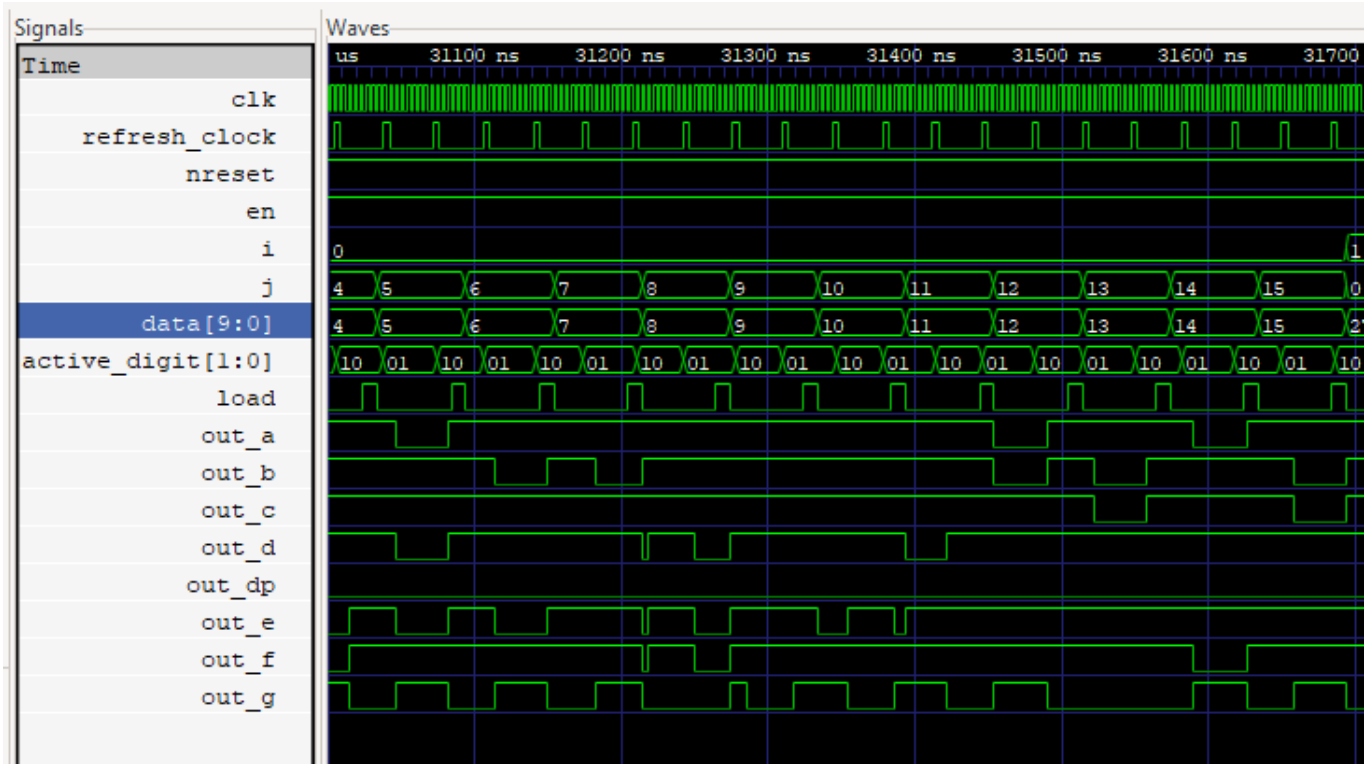
7.3 UART_Receiver_rx



7.4 UART Receiver



7.5 Seven_Segment Display



8. Conclusion

In this application, we integrated two IP Modules together and created a design which allows the data to flow from a UART to a seven-segment display. The two digits can be observed on a PmodSSD which is connected to the ForgeFPGA's Evaluation Board. Through this Application Note we also understood the working of the UART Terminal.

If interested, please contact the ForgeFPGA Business Support Team.

9. Revision History

Revision	Date	Description
1.0	March 18, 2023	Initial release.
2.0	Feb 23, 2024	Updated as per BB Revision
3.0	July 17, 2024	Updated as per ForgeFPGA Workshop v6.43

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)

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-date 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 and registered trademarks are the property of their respective owners.