ForgeFPGA Running String Example
SLG47910V

Abstract

This application note shows how to create a Running String effect on an LED Dot Matrix by using the SLG47910 FPGA. This application note comes complete with design files which can be found in the References section.

Contents

1. Terms and Definitions

FPGA	Field Programmable Gate Array
FPGA Editor	Main FPGA design and simulation window
Go Configure Software Hub	Main window for device selection
ForgeFPGA Window	Main FPGA project window for debug and IO programming

2. 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] Go Configure Software Hub, Software Download and User Guide, Renesas Electronics

[2] AN-FG-015 ForgeFPGA Running String Example.ffpga, AN-FG-015 ForgeFPGA Running String U2 Design (SLG46582V).gp5, ForgeFPGA Design Files

[3] SLG47910, ForgeFPGA Datasheet, Renesas Electronics

Authors: Andrii Sviatobatko, Ivan Pavlyk

3. Introduction

This application note shows how to utilize an LED Dot Matrix for scrolling LEDs to create a Running String effect by using the ForgeFPGA SLG47910. This design uses the SPI module (Module Library) for reading the symbol arrays from external flash memory to BRAM and sending the output text to the LED Dot Matrix based on a MAX7219. Every period the data is shifted by one column. Also, a GreenPAK is used for level shifting between the FPGA and LED Dot Matrix.

The connections between each component are shown in **Figure 1** below.

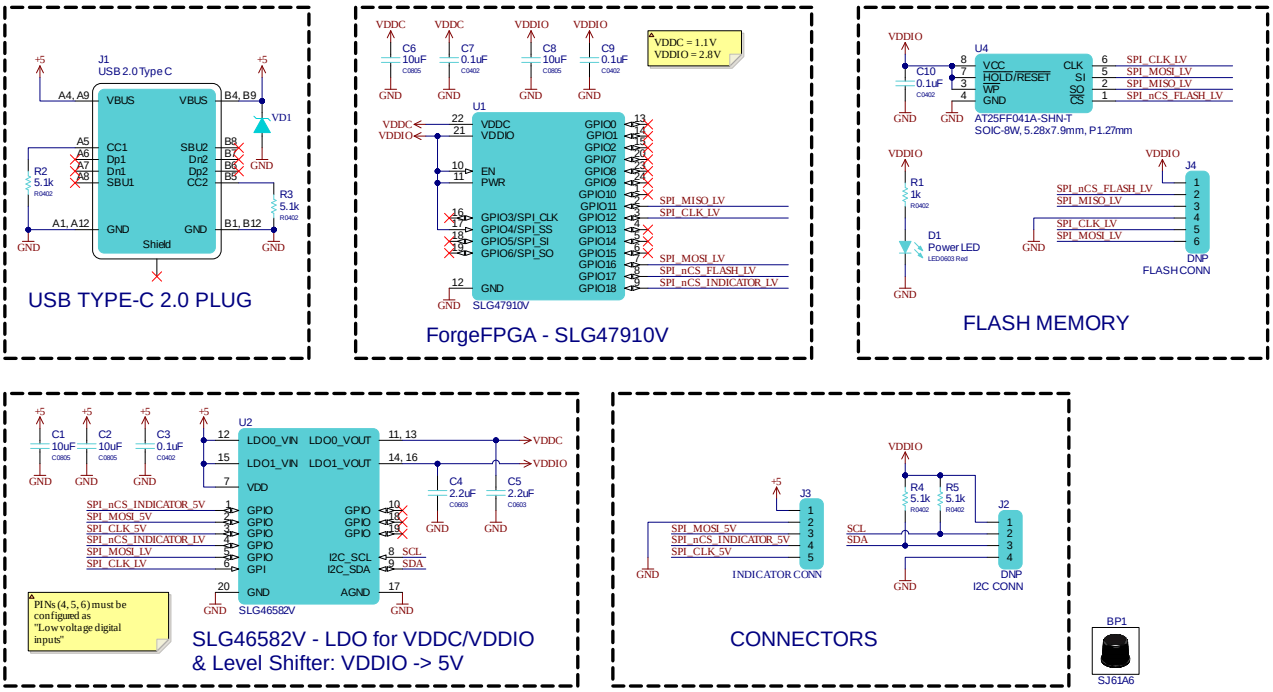

Figure 1: ForgeFPGA Running String Demo Board connections.

In the Verilog code we have ten sub-modules:

- **input_reset_buf** - Module for the input reset buffer
- **init_data_mux** – Module to determine the values sent to the dot matrix
- **init_bram** - Module to control and initialize BRAM (bram0, bram1)
- **fsm_buffer** - FSM module prepares the data buffer for indicator
- **segment_bram** - Module for control and connection for segment BRAM (bram4, bram5, bram6, bram7)
- **fsm_prepare_bram** - FSM module control to process read data from flash and write it to BRAM
- **indicator** - Module for data and control for indicator
- **pause_between_shift** - Module to pause between shifting data on the dot matrix.

- **spi_master** - SPI master, with one slave (Serial Peripheral Interface) is a 4-wire synchronous serial communication interface
- **spi_io_buf** - Module for SPI GPIO buffer
- **indicator_top** - All sub-modules are connected to each other to form the top module.

In **Figure 2**, the connections between the Verilog block are shown. The user can cross check the defined signal and wire names with Verilog Code.

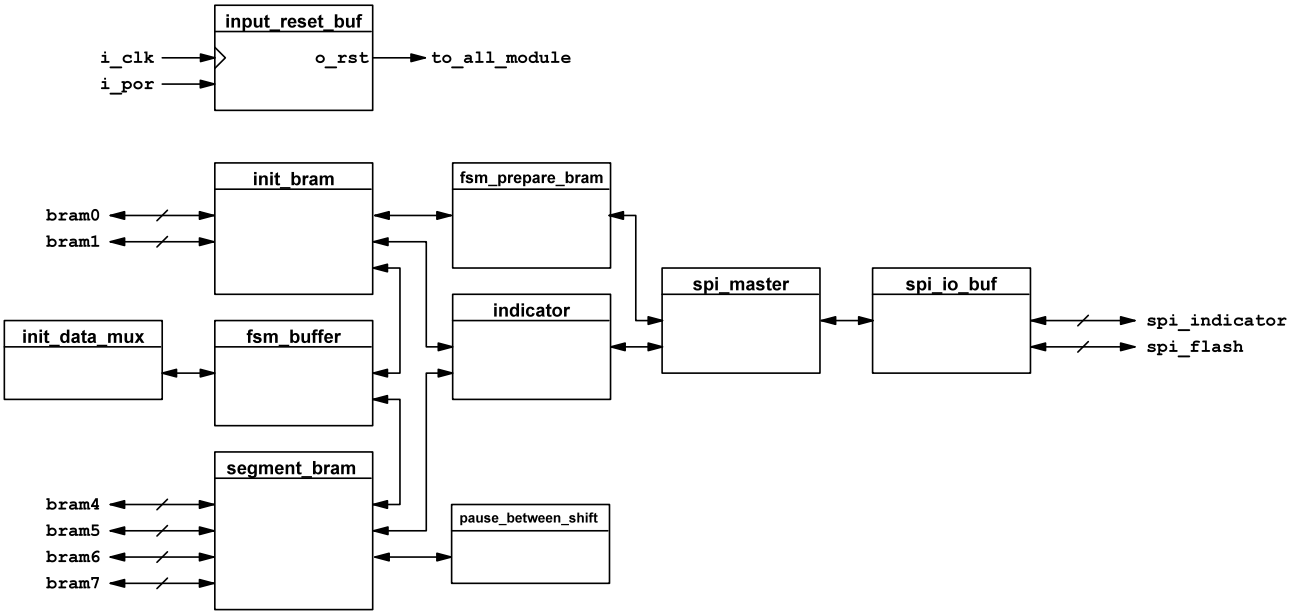

Figure 2: ForgeFPGA Running String Software structure

3.1 FSM Buffer

FSM module prepares the data buffer for indicator. FSM Buffer diagram is shown in **Figure 3**.

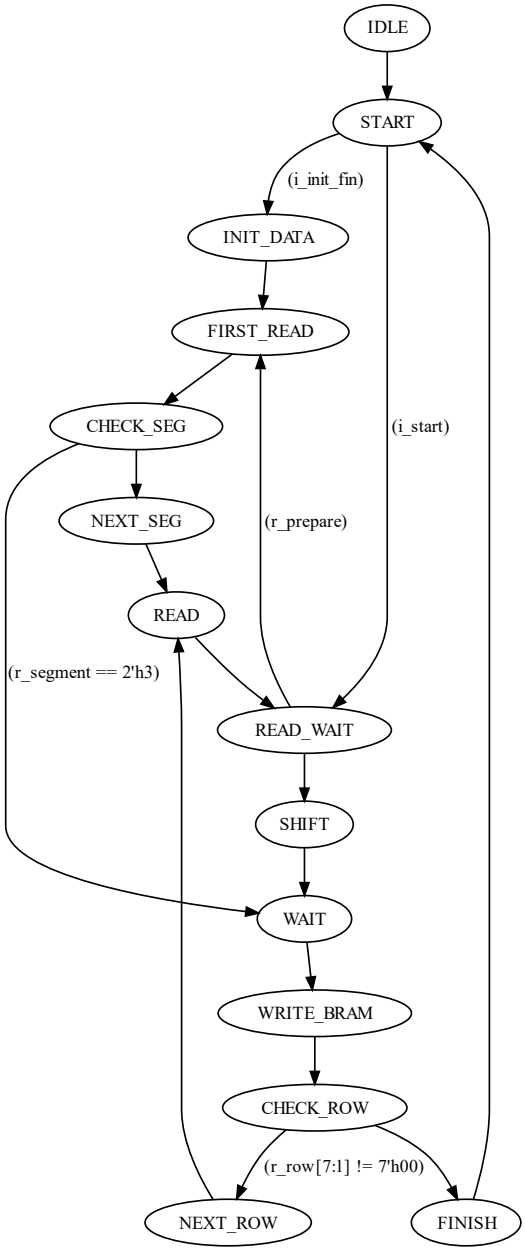

Figure 3: FSM Buffer

3.2 FSM Indicator

FSM module initializes and sends the data and controls the byte to be sent. FSM Indicator diagram is shown in **Figure 4**.

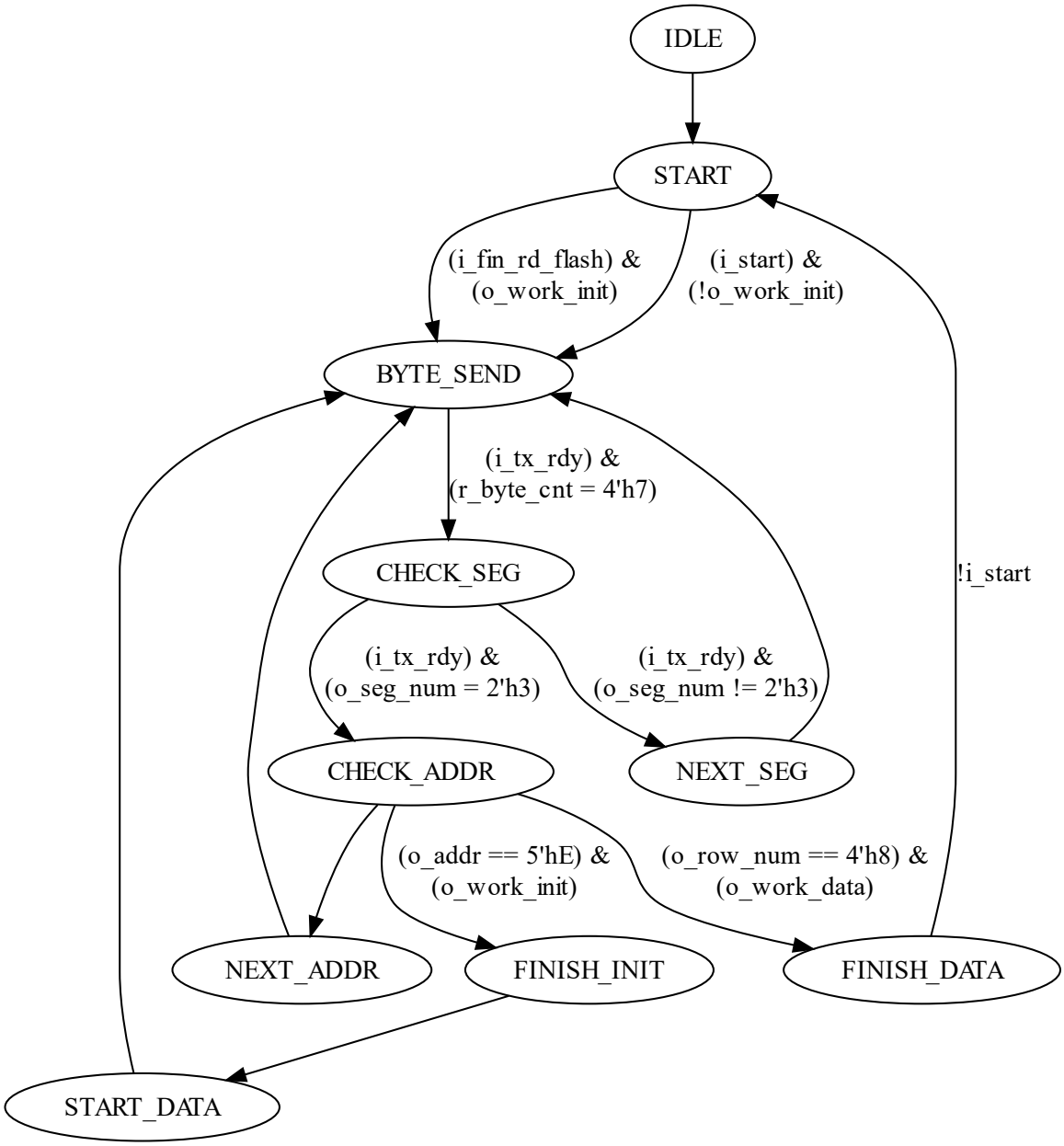

Figure 4: FSM Indicator

3.3 FSM Prepare BRAM

FSM module control process reads data from flash and writes it to BRAM. FSM Prepare BRAM diagram is shown in **Figure 5**.

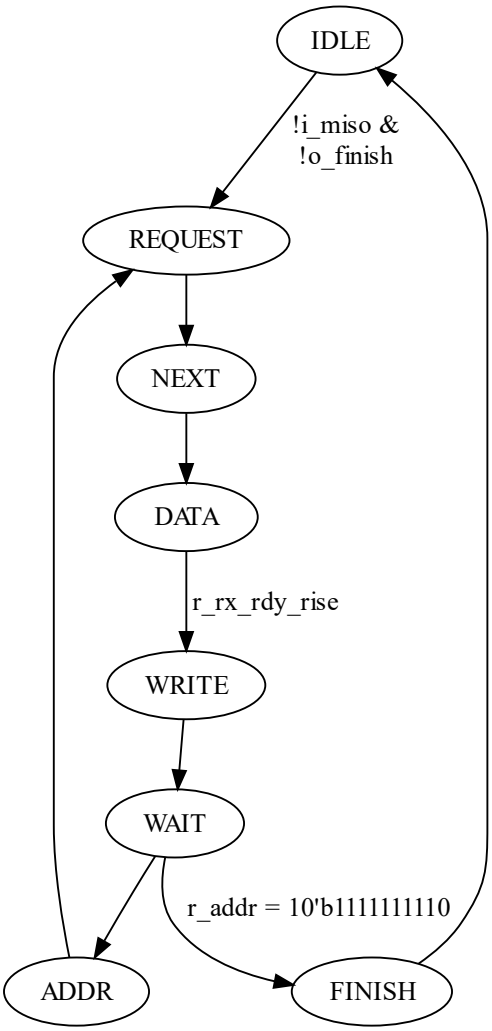

Figure 5: FSM Prepare BRAM

3.4 Waveform SPI for use with the Indicator

SPI packages and package sequence for controlling the dot matrix display. Waveforms are shown in **Figure 6**.

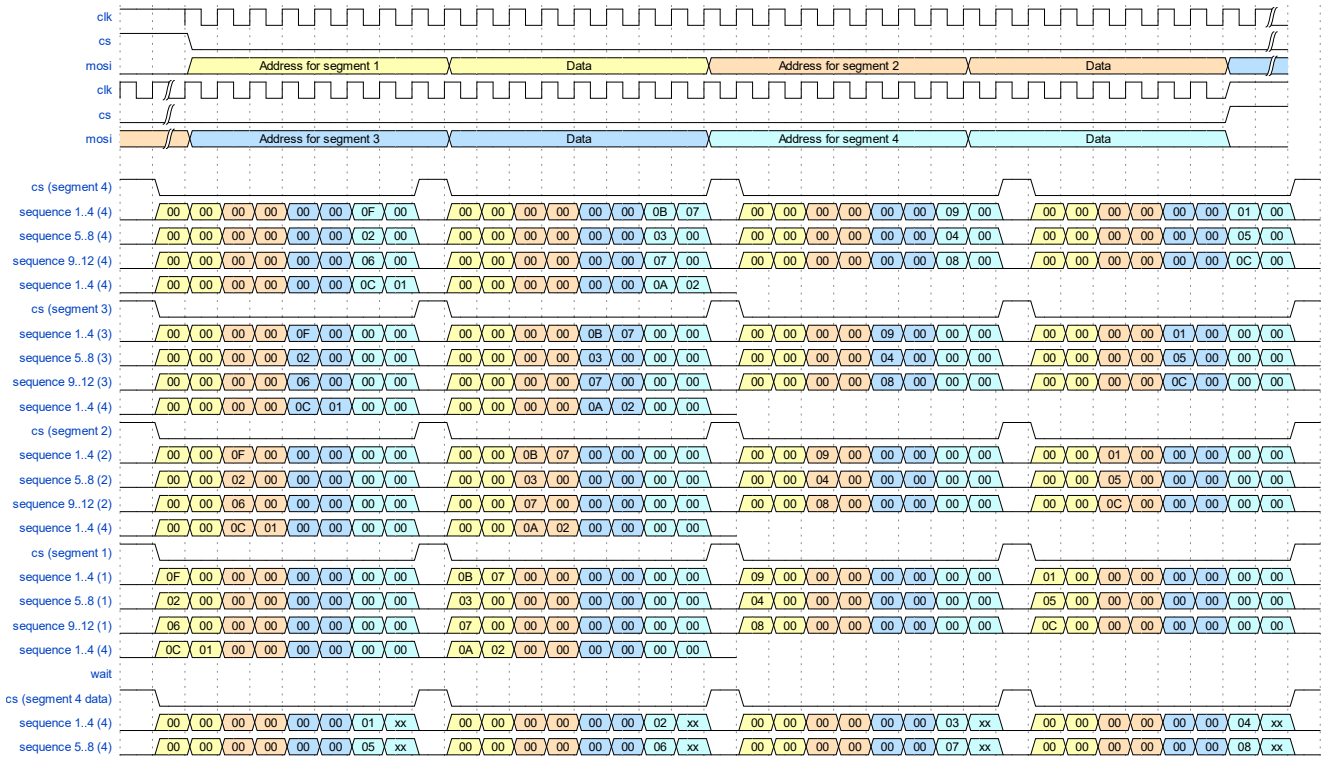

Figure 6: Waveform SPI packages for use with the Indicator

PARAMETERS THAT CAN BE CHANGED.

Using the ForgeFPGA Workshop software, the code is synthesized, and the bit stream is loaded into the SLG47910 device. The design uses the internal 50 MHz oscillator as reference for the PLL that provides a 10 MHz clocking source for the application.

The information in the init_data_mux module can change what is shown using symbols code, with two symbols for each one byte for a total of 64 symbols available.

Also, the running line speed can be changed in the pause_between_shift module.

To change symbols, data must be edited in the flash and in the init_data_mux module.

4. Components

The following components are used in this example:

- ForgeFPGA IC's SLG47910V
- ForgeFPGA Development Board with USB cable and power supply
- GreenPAK Universal Dev.Board with USB cable
- Flash memory AT25FFA-SHN-T
- Latest Revision of ForgeFPGA Workshop software
- ForgeFPGA Running String Demo Board R1.0
- LED Dot Matrix 8x32

5. ForgeFPGA Running String Demo Board

The ForgeFPGA Running String Demo board is small and is designed to demonstrate how to use the ForgeFPGA SLG47910V in this application.

The board layout of the PCB is shown in **Figure 7**.

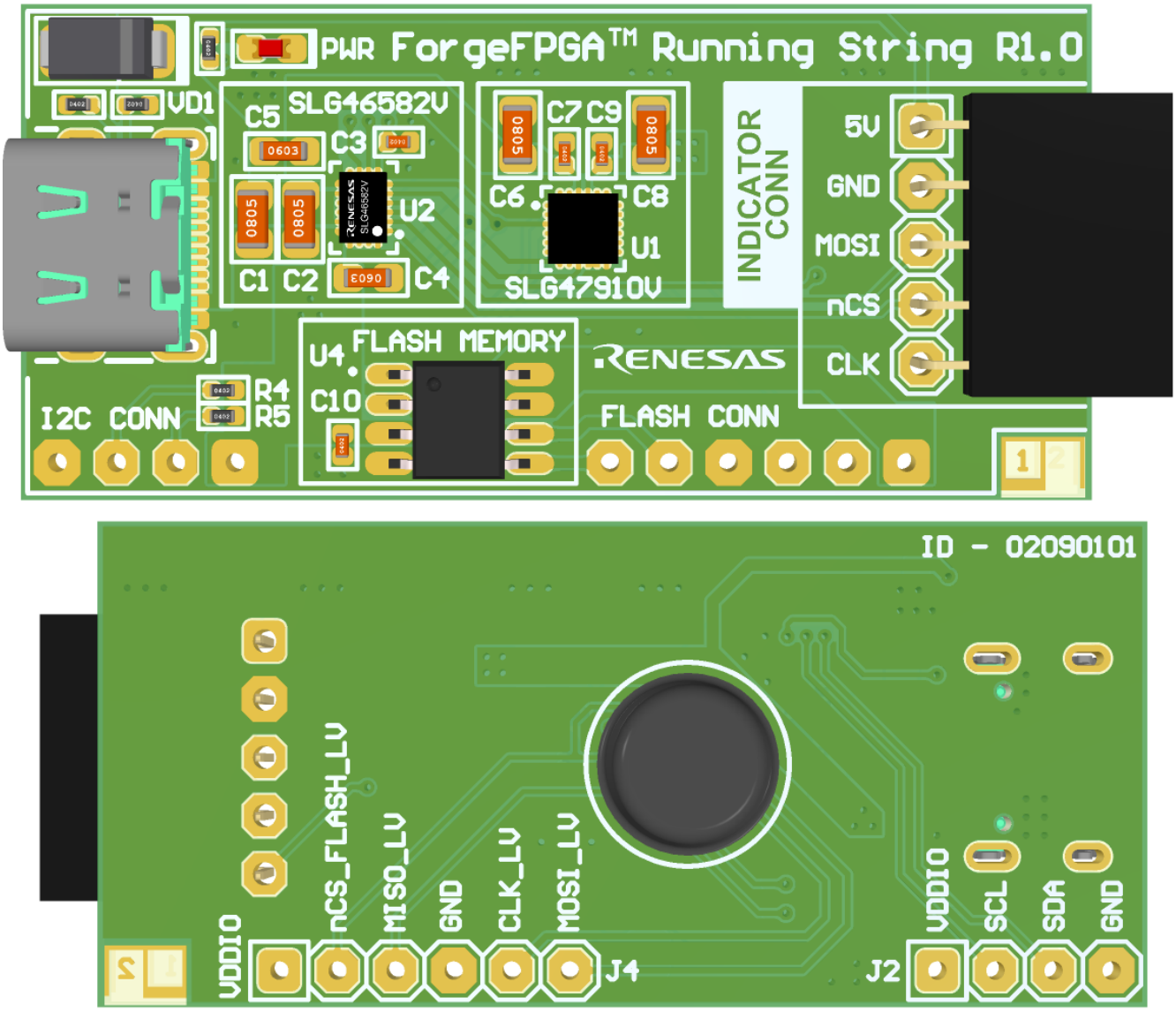

Figure 7: ForgeFPGA Running String Demo Board

6. Verilog Code for ForgeFPGA Running String

The indicator_top module code is shown below. The named (*top*) module has interconnections between all submodules shown in **Figure 2**. The Verilog code for the Running String can be found in the complete design example which is available for download AN-FG-015 ForgeFPGA Running String Example.ffpga.

```
(* top *) module indicator_top (
// Main inputs
  (* iopad_external_pin, clkbuf_inhibit *) input i_clk,
  (* iopad_external_pin *) input i_por,
// OSC config outputs
  (* iopad_external_pin *) output o_osc_en,
// PLL config outputs
  (* iopad_external_pin *) output o_pll_en,
  (* iopad_external_pin *) output o_pll_sel,
  (* iopad_external_pin *) output o_pll_byp,
  (* iopad_external_pin *) output [11:0] o_pll_fbdiv,
  (* iopad_external_pin *) output [2:0]  o_pll_postdiv1,
  (* iopad_external_pin *) output [2:0]  o_pll_postdiv2,
  (* iopad_external_pin *) output [5:0]  o_pll_refdiv,
// IO
  (* iopad_external_pin *) input  i_miso,
  (* iopad_external_pin *) output oe_miso,
  (* iopad_external_pin *) output o_flash_ncs,
  (* iopad_external_pin *) output o_indicator_ncs,
  (* iopad_external_pin *) output o_clk,
  (* iopad_external_pin *) output o_mosi,
  (* iopad_external_pin *) output oe_flash_ncs,
  (* iopad_external_pin *) output oe_indicator_ncs,
  (* iopad_external_pin *) output oe_clk,
  (* iopad_external_pin *) output oe_mosi,
  (* iopad_external_pin *) output reg o_dff,
  (* iopad_external_pin *) output o_dff_oe,

//BRAM
  (* iopad_external_pin *) input      [7:0] i_bram0_data_out,
  (* iopad_external_pin *) output     [1:0] o_bram0_ratio,
  (* iopad_external_pin *) output reg [7:0] o_bram0_data_in,
  (* iopad_external_pin *) output reg       o_bram0_web,
  (* iopad_external_pin *) output reg       o_bram0_wclken,
  (* iopad_external_pin *) output reg [8:0] o_bram0_write_addr,
  (* iopad_external_pin *) output reg       o_bram0_reb,
  (* iopad_external_pin *) output reg       o_bram0_rclken,
  (* iopad_external_pin *) output reg [8:0] o_bram0_read_addr,
  (* iopad_external_pin *) input      [7:0] i_bram1_data_out,
  (* iopad_external_pin *) output     [1:0] o_bram1_ratio,
  (* iopad_external_pin *) output reg [7:0] o_bram1_data_in,
  (* iopad_external_pin *) output reg       o_bram1_web,
  (* iopad_external_pin *) output reg       o_bram1_wclken,
```

```
  (* iopad_external_pin *) output reg [8:0] o_bram1_write_addr,
  (* iopad_external_pin *) output reg       o_bram1_reb,
  (* iopad_external_pin *) output reg       o_bram1_rclken,
  (* iopad_external_pin *) output reg [8:0] o_bram1_read_addr,
  (* iopad_external_pin *) input      [7:0] i_bram4_data_out,
  (* iopad_external_pin *) output     [1:0] o_bram4_ratio,
  (* iopad_external_pin *) output reg [7:0] o_bram4_data_in,
  (* iopad_external_pin *) output reg       o_bram4_web,
  (* iopad_external_pin *) output reg       o_bram4_wclken,
  (* iopad_external_pin *) output reg [8:0] o_bram4_write_addr,
  (* iopad_external_pin *) output reg       o_bram4_reb,
  (* iopad_external_pin *) output reg       o_bram4_rclken,
  (* iopad_external_pin *) output reg [8:0] o_bram4_read_addr,
  (* iopad_external_pin *) input      [7:0] i_bram5_data_out,
  (* iopad_external_pin *) output     [1:0] o_bram5_ratio,
  (* iopad_external_pin *) output reg [7:0] o_bram5_data_in,
  (* iopad_external_pin *) output reg       o_bram5_web,
  (* iopad_external_pin *) output reg       o_bram5_wclken,
  (* iopad_external_pin *) output reg [8:0] o_bram5_write_addr,
  (* iopad_external_pin *) output reg       o_bram5_reb,
  (* iopad_external_pin *) output reg       o_bram5_rclken,
  (* iopad_external_pin *) output reg [8:0] o_bram5_read_addr,
  (* iopad_external_pin *) input      [7:0] i_bram6_data_out,
  (* iopad_external_pin *) output     [1:0] o_bram6_ratio,
  (* iopad_external_pin *) output reg [7:0] o_bram6_data_in,
  (* iopad_external_pin *) output reg       o_bram6_web,
  (* iopad_external_pin *) output reg       o_bram6_wclken,
  (* iopad_external_pin *) output reg [8:0] o_bram6_write_addr,
  (* iopad_external_pin *) output reg       o_bram6_reb,
  (* iopad_external_pin *) output reg       o_bram6_rclken,
  (* iopad_external_pin *) output reg [8:0] o_bram6_read_addr,
  (* iopad_external_pin *) input      [7:0] i_bram7_data_out,
  (* iopad_external_pin *) output     [1:0] o_bram7_ratio,
  (* iopad_external_pin *) output reg [7:0] o_bram7_data_in,
  (* iopad_external_pin *) output reg       o_bram7_web,
  (* iopad_external_pin *) output reg       o_bram7_wclken,
  (* iopad_external_pin *) output reg [8:0] o_bram7_write_addr,
  (* iopad_external_pin *) output reg       o_bram7_reb,
  (* iopad_external_pin *) output reg       o_bram7_rclken,
  (* iopad_external_pin *) output reg [8:0] o_bram7_read_addr,
);
// assign configuration pins
  assign o_osc_en       = 1'b1;
  assign o_pll_en       = 1'b1;
  assign o_pll_sel      = 1'b0;
  assign o_pll_byp      = 1'b0;
  assign o_pll_fbdiv    = 20;
  assign o_pll_postdiv1 = 7;
```

```
  assign o_pll_postdiv2 = 7;
  assign o_pll_refdiv   = 2;

  assign oe_flash_ncs     = 1'b1;
  assign oe_indicator_ncs = 1'b1;
  assign oe_clk   = 1'b1;
  assign oe_mosi  = 1'b1;
  assign oe_miso  = 1'b0;
  assign o_dff_oe = 1'b1;

  wire  [7:0] w_rx_byte, w_spi_data, w_tx_indic_byte, w_tx_flash_byte;
  wire  [4:0] w_address_init;
  wire        w_next_show, w_next_byte, w_rx_rdy, w_finish_read_flash;

  wire  [8:0] w_segment_row_rd_addr, w_segment_row_wr_addr;
  wire  [2:0] w_row_number_read;
  wire  [1:0] w_segment_no_init;
  wire        w_work_data, w_work_init;

  wire  [7:0] w_segment_init_value;
  wire [11:0] w_value;
  wire  [8:0] w_addr_prepare_bram;
  wire        w_wen_prepare, w_prepare_data, w_init_finish, w_data_finish;
  wire [31:0] w_data_to_bram, w_data_from_bram;
  wire        w_new_wen;

  wire        w_spi_clk, w_spi_miso, w_spi_mosi, w_indicator_en, w_ncs_indicator_en;
  wire        w_flash_en, w_ncs_flash_en, w_ncs_spi_en;

  always @(posedge i_clk) begin
    if (w_rst) begin
      o_dff <= 1'b0;
    end else begin
      o_dff <= ~o_dff;
    end
  end

// reset buffer
  wire w_rst;
  input_reset_buf impl_input_reset_buf (
    .i_clk        (i_clk),
    .i_por        (i_por),
    .o_rst        (w_rst)
  );

// SPI external connection for two slave
  spi_io_buf impl_spi_io_buf (
    .i_clk                (i_clk),
```

```
    .i_rst                (w_rst),
    .i_flash_miso         (i_miso),
    .o_flash_miso         (w_spi_miso),
    .i_spi_mosi           (w_spi_mosi),
    .o_flash_mosi         (o_mosi),
    .o_indicator_mosi     (o_mosi),
    .i_spi_clk            (w_spi_clk),
    .o_flash_clk          (o_clk),
    .o_indicator_clk      (o_clk),
    .i_flash_ncs          (w_flash_en),
    .o_flash_ncs          (o_flash_ncs),
    .i_indicator_ncs      (w_indicator_en),
    .o_indicator_ncs      (o_indicator_ncs),

    .i_flash_ncs_en       (w_ncs_flash_en),
    .i_indicator_ncs_en   (w_ncs_indicator_en),
    .o_spi_ncs_en         (w_ncs_spi_en),

    .i_tx_flash_byte      (w_tx_flash_byte),
    .i_tx_indic_byte      (w_tx_indic_byte),
    .o_spi_data           (w_spi_data)
  );

// SPI
  spi_master impl_spi_master (
    .clk                  (i_clk),
    .rst                  (w_rst),
    .sck                  (w_spi_clk),
    .mosi                 (w_spi_mosi),
    .nss_en               (w_ncs_spi_en),
    .nss                  (),
    .tx_data_valid        (1'b1),
    .miso                 (w_spi_miso),
    .tx_data              (w_spi_data),
    .rx_data              (w_rx_byte),
    .tx_int               (w_next_byte),
    .rx_int               (w_rx_rdy)
  );

// data and control for indicator
  indicator impl_indicator (
    .i_clk                (i_clk),
    .i_rst                (w_rst),
    .i_tx_rdy             (w_next_byte),
    .i_start              (w_next_show),
    .i_fin_rd_flash       (w_finish_read_flash),
    .i_init_data          (w_segment_init_value),
    .i_data_from_bram     (w_data_from_bram),
```

```
    .o_ctrl_cen             (w_ncs_indicator_en),
    .o_ctrl_indic           (w_indicator_en),
    .o_row                  (w_row_number_read),
    .o_tx_data              (w_tx_indic_byte),
    .o_addr                 (w_address_init),
    .o_work_init            (w_work_init),
    .o_work_data            (w_work_data),
    .o_init_finish          (w_init_finish),
    .o_data_finish          (w_data_finish)
  );

// Message on indicator
  init_data_mux impl_init_data_mux (
    .i_clk                  (i_clk),
    .i_rst                  (w_rst),
    .i_seg_row_addr         (w_segment_row_wr_addr[5:3]),
    .i_seg_num_init         (w_segment_no_init),
    .o_value                (w_value)
  );

// bram contains symbol code and initial sequence of dot matrix
  init_bram impl_init_bram(
    .i_clk                  (i_clk),
    .i_rst                  (w_rst),

    .i_addr_prepare_bram    (w_addr_prepare_bram),
    .i_wen_prepare          (w_wen_prepare),
    .i_prepare_data         (w_prepare_data),
    .i_work_init            (w_work_init),
    .i_rx_byte              (w_rx_byte),
    .i_address_init         (w_address_init),
    .i_segment_row_rd_addr  (w_segment_row_rd_addr[2:0]),
    .i_row_number_read      (w_row_number_read),
    .i_rd_value_bram0       (w_value[5:0]),
    .i_rd_value_bram1       (w_value[11:6]),
    .o_segment_init_value   (w_segment_init_value),

    .o_bram_pd              (),

    .i_bram0_data_out       (i_bram0_data_out),
    .o_bram0_ratio          (o_bram0_ratio),
    .o_bram0_data_in        (o_bram0_data_in),
    .o_bram0_web            (o_bram0_web),
    .o_bram0_wclken         (o_bram0_wclken),
    .o_bram0_write_addr     (o_bram0_write_addr),
    .o_bram0_reb            (o_bram0_reb),
    .o_bram0_rclken         (o_bram0_rclken),
    .o_bram0_read_addr      (o_bram0_read_addr),
```

```
    .i_bram1_data_out      (i_bram1_data_out),
    .o_bram1_ratio         (o_bram1_ratio),
    .o_bram1_data_in       (o_bram1_data_in),
    .o_bram1_web           (o_bram1_web),
    .o_bram1_wclken        (o_bram1_wclken),
    .o_bram1_write_addr    (o_bram1_write_addr),
    .o_bram1_reb           (o_bram1_reb),
    .o_bram1_rclken        (o_bram1_rclken),
    .o_bram1_read_addr     (o_bram1_read_addr)
  );

//  FSM control process read data from flash and write it to BRAM
  fsm_prepare_bram impl_fsm_prepare_bram (
    .i_clk                 (i_clk),
    .i_rst                 (w_rst),
    .i_rx_rdy              (w_rx_rdy),
    .i_miso                (w_spi_miso),
    .o_spi_ce              (w_ncs_flash_en),
    .o_flash_en            (w_flash_en),
    .o_tx_byte             (w_tx_flash_byte),
    .o_addr                (w_addr_prepare_bram),
    .o_wen                 (w_wen_prepare),
    .o_finish              (w_finish_read_flash)
  );

// indicator bram
  segment_bram impl_segment_bram(
    .i_clk                 (i_clk),
    .i_rst                 (w_rst),

    .i_data_to_bram        (w_data_to_bram),
    .i_segment_row_rd_addr (w_segment_row_rd_addr),
    .i_segment_row_wr_addr (w_segment_row_wr_addr),
    .i_wen                 (w_new_wen),
    .i_row_number_read     (w_row_number_read),
    .i_work_data           (w_work_data),

    .o_data_from_bram      (w_data_from_bram),
    .o_bram_pd             (),

    .i_bram4_data_out      (i_bram4_data_out),
    .o_bram4_ratio         (o_bram4_ratio),
    .o_bram4_data_in       (o_bram4_data_in),
    .o_bram4_web           (o_bram4_web),
    .o_bram4_wclken        (o_bram4_wclken),
    .o_bram4_write_addr    (o_bram4_write_addr),
    .o_bram4_reb           (o_bram4_reb),
```

```
    .o_bram4_rclken        (o_bram4_rclken),
    .o_bram4_read_addr     (o_bram4_read_addr),

    .i_bram5_data_out      (i_bram5_data_out),
    .o_bram5_ratio         (o_bram5_ratio),
    .o_bram5_data_in       (o_bram5_data_in),
    .o_bram5_web           (o_bram5_web),
    .o_bram5_wclken        (o_bram5_wclken),
    .o_bram5_write_addr    (o_bram5_write_addr),
    .o_bram5_reb           (o_bram5_reb),
    .o_bram5_rclken        (o_bram5_rclken),
    .o_bram5_read_addr     (o_bram5_read_addr),

    .i_bram6_data_out      (i_bram6_data_out),
    .o_bram6_ratio         (o_bram6_ratio),
    .o_bram6_data_in       (o_bram6_data_in),
    .o_bram6_web           (o_bram6_web),
    .o_bram6_wclken        (o_bram6_wclken),
    .o_bram6_write_addr    (o_bram6_write_addr),
    .o_bram6_reb           (o_bram6_reb),
    .o_bram6_rclken        (o_bram6_rclken),
    .o_bram6_read_addr     (o_bram6_read_addr),

    .i_bram7_data_out      (i_bram7_data_out),
    .o_bram7_ratio         (o_bram7_ratio),
    .o_bram7_data_in       (o_bram7_data_in),
    .o_bram7_web           (o_bram7_web),
    .o_bram7_wclken        (o_bram7_wclken),
    .o_bram7_write_addr    (o_bram7_write_addr),
    .o_bram7_reb           (o_bram7_reb),
    .o_bram7_rclken        (o_bram7_rclken),
    .o_bram7_read_addr     (o_bram7_read_addr)
  );

//  FSM prepare data buffer for indicator
  fsm_buffer impl_fsm_buffer (
    .i_clk                 (i_clk),
    .i_rst                 (w_rst),
    .i_init_fin            (w_init_finish),
    .i_start               (w_data_finish),
    .i_init_data           (w_segment_init_value),
    .i_prev_data           (w_data_from_bram),
    .o_new_data            (w_data_to_bram),
    .o_wen                 (w_new_wen),
    .o_segment             (w_segment_no_init),
    .o_addr_rd             (w_segment_row_rd_addr),
    .o_addr_wr             (w_segment_row_wr_addr),
    .o_prepare             (w_prepare_data)
```

```
  );

// shift timer
  pause_between_shift impl_pausa_between_shift (
    .i_clk                  (i_clk),
    .i_rst                  (w_rst),
    .o_next_show            (w_next_show)
  );

endmodule
```

7. Floorplan: CLB Utilization

The Floorplan tab in the FPGA Editor shows the placement of each of the CLBs and FFs (**Figure 8**). The resource utilization is shown in the top left corner.

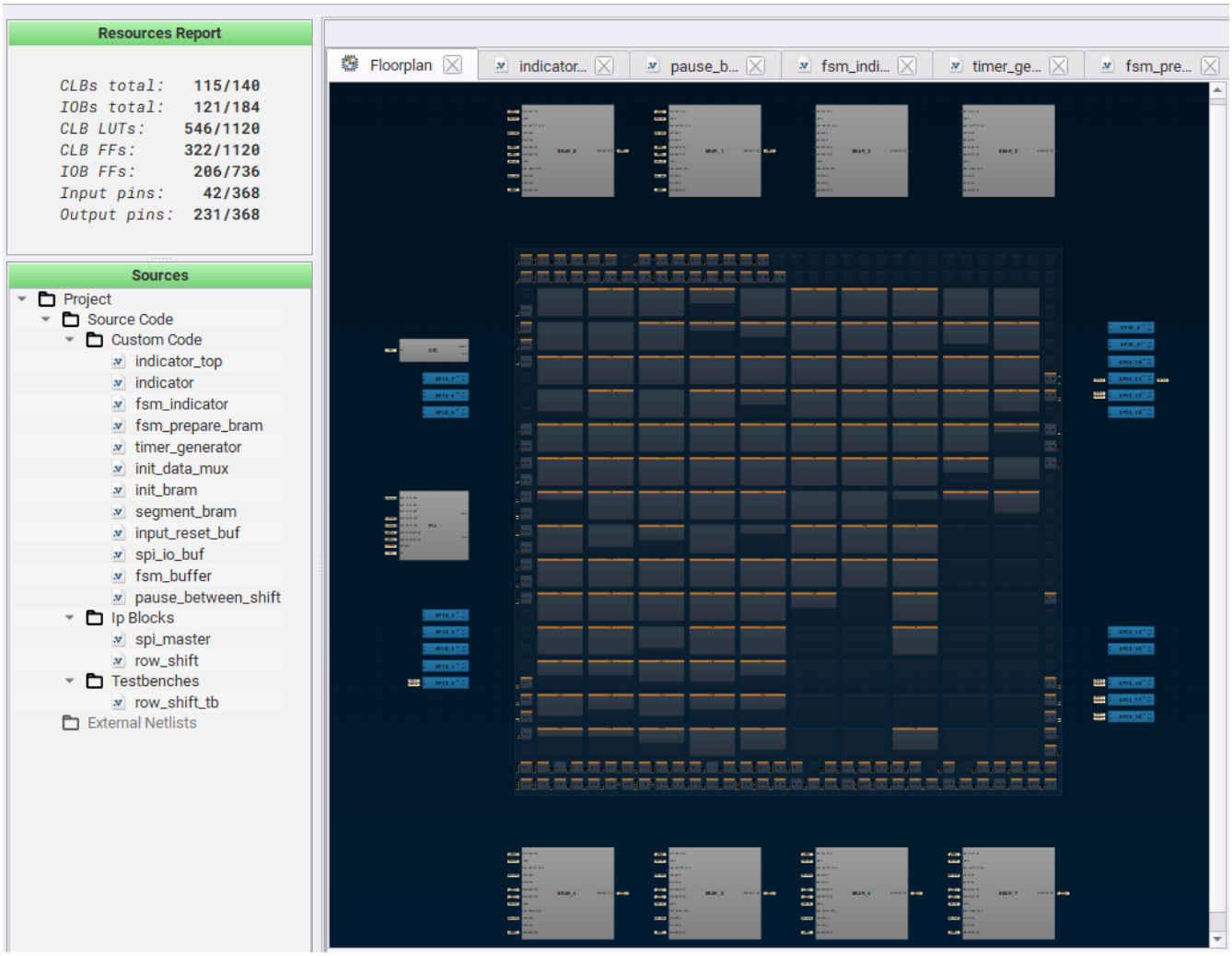

Figure 8: ForgeFPGA Running String CLB Utilization

8. Design Steps

1. If the ForgeFPGA Running String Demo board (**Figure 7**) is available, it can be used to demo this project. Plug it into your USB port.
2. Otherwise, the rest of the components listed in **Section 4** are needed.
3. Make all of the connections as shown in **Figure 9**.

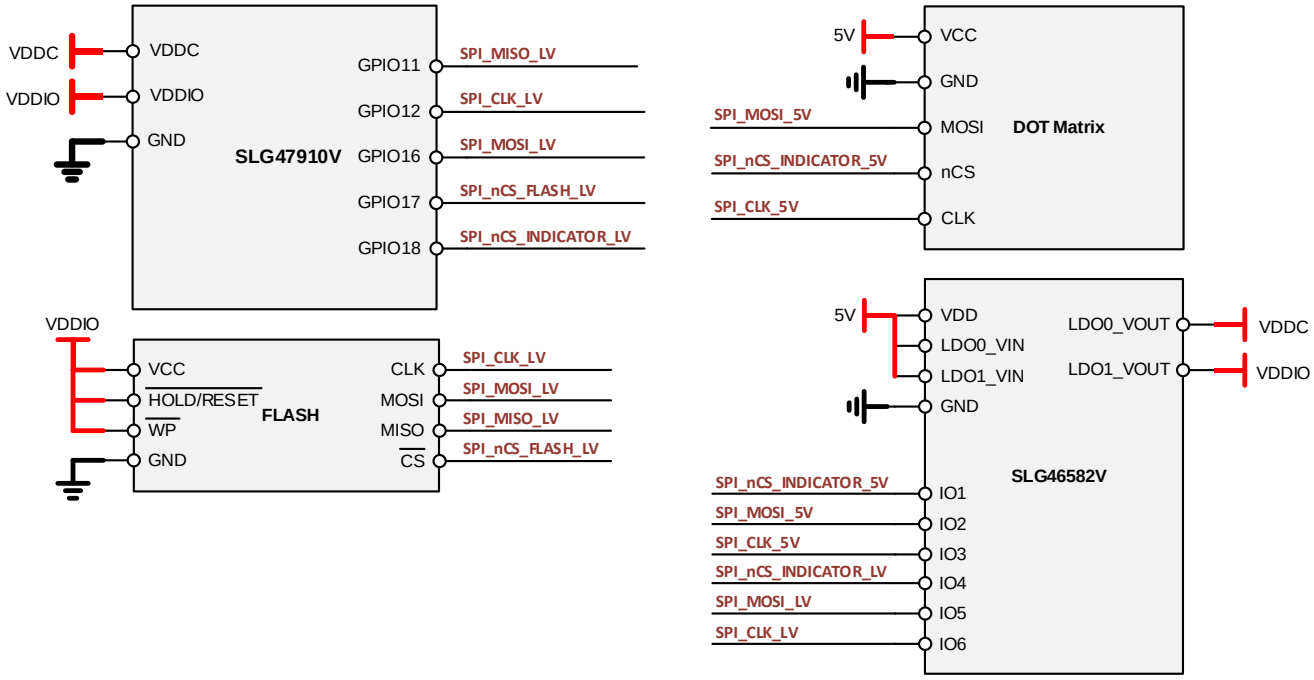

Figure 9: ForgeFPGA Running String Connections

4. Launch the latest version of the Go Configure Software Hub. We will need to open two windows, one for ForgeFPGA (SLG47910V) and the second for GreenPAK (SLG46582V).
5. Select the SLG47910V device and the ForgeFPGA Workshop software will load.
6. Download the design example AN-FG-015 ForgeFPGA Running String Example.ffpga as well as AN-FG-015 ForgeFPGA Running String U2 Design (SLG46582V).gp5 for GreenPAK.
7. In the first window open the AN-FG-015 ForgeFPGA Running String Example.ffpga file after downloading. In the second window open the AN-FG-015 ForgeFPGA Running String U2 Design (SLG46582V).gp5.
8. Open the FPGA editor and review the Verilog code.
9. Open the IO Planner tab in the FPGA editor and review the pin assignment.
10. Next press the Synthesize button on the lower left side of the FPGA editor.
11. Press 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.
12. Now click on the Floorplan tab and check the CLB utilization (**Figure 8**). Press Ctrl + mouse wheel up to zoom-in. Confirm that the IOs selected in the IO Planner are shown in the floorplan.
13. Close the FPGA Editor and go to the ForgeFPGA Workshop window (**Figure 10**). Selecting the Debug tab will enable the debug controls. Double-click on the VDD pin and set VDD = 1.1 V. Then double-click on the VDDIO pin and set VDDIO = 2.75 V.

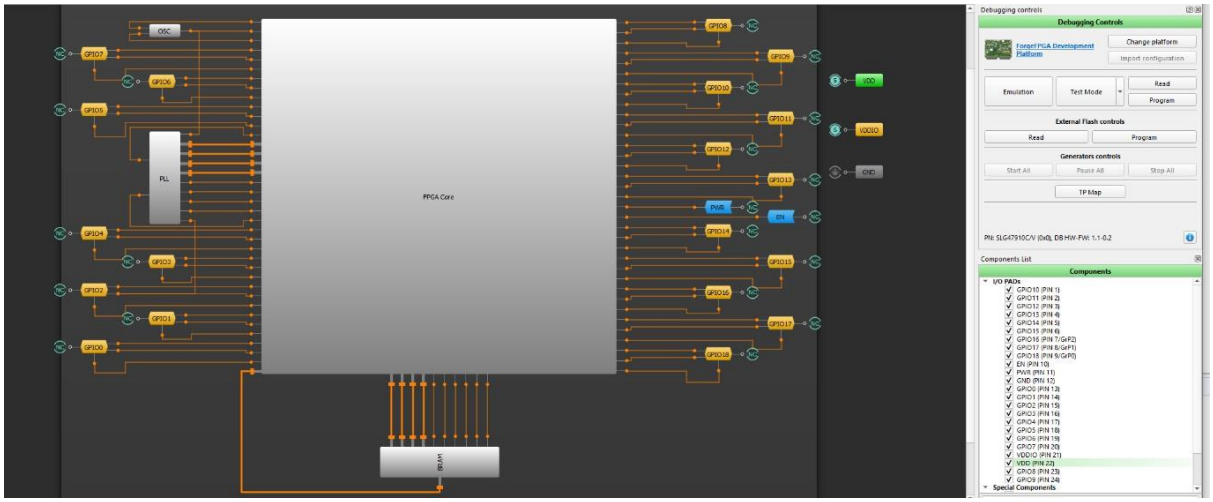

Figure 10: Forge FPGA Window

14. Double-click on EN and PWR and select Force On. This ensures that all blocks are powered up.
15. In the GreenPAK Designer window, open Debug and run Emulation for level shifting (**Figure**).

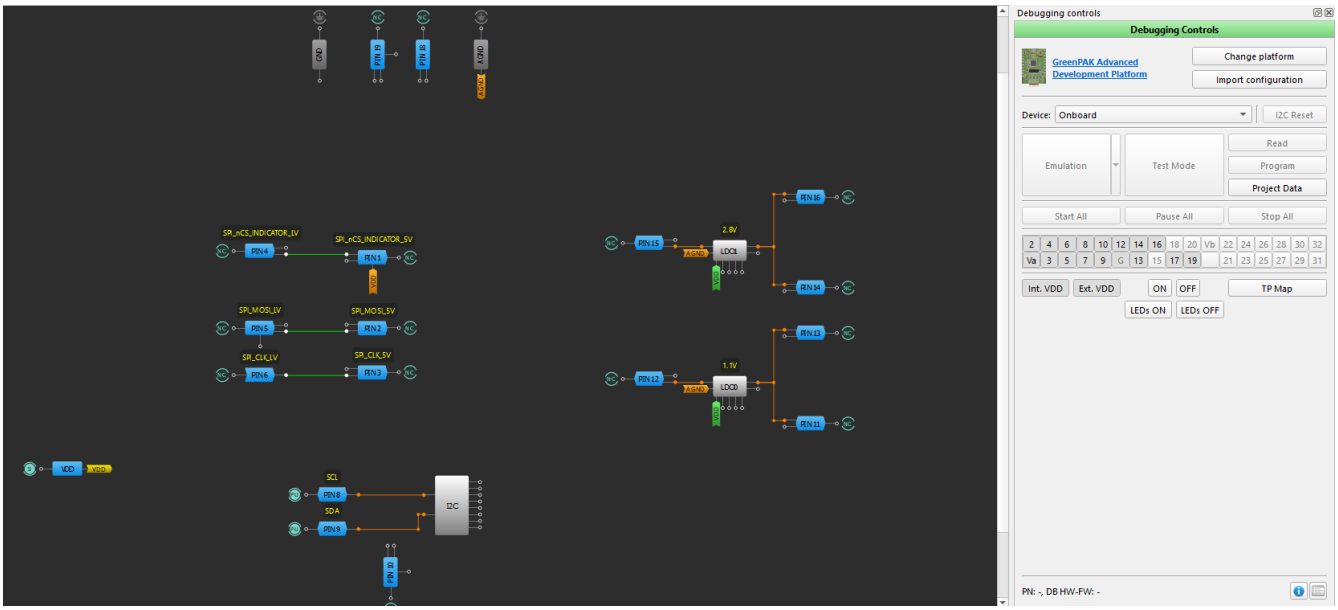

Figure 11: GreenPAK Designer Window

16. The ForgeFPGA Running String design is now ready to load onto the FPGA using the development board. Connect the development board to the adapter board. Connect the DOT Matrix and Flash board to the adapter board. Connect the USB and the power supply to the development board.
17. In the ForgeFPGA Workshop window, select Change platform on the Debugging Controls tab.Choose the ForgeFPGA Development Platform then select Emulation.

Now the design is loaded onto the FPGA device.

9. Conclusion

This application note shows how the SLG47910 can be used to control a complicated sequencing of a Dot matrix as a running string. This testcase is available for download from our website (AN-FG-015 ForgeFPGA Running String Example.ffpga). If interested, please contact the ForgeFPGA Business Support Team.

10. Revision History

Revision	Date	Description
1.00	Jun 5, 2024	Initial release.