Laboratorio N° 1

Diseño Digital Avanzado

Mathias Sebastian Garcia

29 de septiembre de 2024

Índice

1. Código RTL **2**
1.1. Contador . . . 2
1.2. Shift register . . . 4
1.3. Top . . . 5

2. Testbench **7**
2.1. Código . . . 7
2.2. Capturas de simulación . . . 9

3. Implementación en FPGA **10**
3.1. Wrapper VIO . . . 10
3.2. Wrapper ILA . . . 11
3.3. Wrapper completo . . . 12
3.4. Capturas . . . 14

1. Código RTL

Link de github a los archivos de RTL.

1.1. Contador

```
module counter
#(
    // * ------------------------------------------------------------------
    // * Parameters
    // * ------------------------------------------------------------------
    parameter                           NB_COUNTER   = 32                 ,
    parameter                           NB_SW        =  3                 ,
    parameter                           ADD_PIPE_OUT =  0
)
(
    // * ------------------------------------------------------------------
    // * Outputs
    // * ------------------------------------------------------------------
    output logic                        o_count_reached                   ,

    // * ------------------------------------------------------------------
    // * Inputs
    // * ------------------------------------------------------------------
    input  logic [ NB_SW      - 1 : 0 ] i_sw                              ,

    // * ------------------------------------------------------------------
    // * Clock and reset
    // * ------------------------------------------------------------------
    input  logic                        i_reset                           ,
    input  logic                        i_clock
) ;
    // * ------------------------------------------------------------------
    // * Localparams
    // * ------------------------------------------------------------------
    localparam                          R0           = 2**(NB_COUNTER-10)-1 ;
    localparam                          R1           = 2**(NB_COUNTER-11)-1 ;
    localparam                          R2           = 2**(NB_COUNTER-12)-1 ;
    localparam                          R3           = 2**(NB_COUNTER-13)-1 ;

    // * ------------------------------------------------------------------
    // * Internal logics
    // * ------------------------------------------------------------------
    logic       [ NB_COUNTER - 1 : 0 ] count                             ;
    logic       [ NB_COUNTER - 1 : 0 ] next_count                        ;
    logic       [ NB_COUNTER - 1 : 0 ] limit_count                       ;
    logic                              limit_reached                     ;
    logic                              count_enable                      ;
    logic                              piped_limit_reached               ;

    // * ------------------------------------------------------------------
    // * Max count decode
    // * ------------------------------------------------------------------
    always_comb
    begin : proc_limit_count_decode
        unique case (i_sw[2:1]) inside
            2'b00: limit_count = R0[NB_COUNTER-1:0];
            2'b01: limit_count = R1[NB_COUNTER-1:0];
            2'b10: limit_count = R2[NB_COUNTER-1:0];
            2'b11: limit_count = R3[NB_COUNTER-1:0];
        endcase
    end

    // * ------------------------------------------------------------------
    // * Count process
    // * ------------------------------------------------------------------
    assign next_count    = count + {{NB_COUNTER-1{1'b0}}, 1'b1}   ;
    assign count_enable  = i_sw[0]                               ;
    assign limit_reached = (count >= limit_count) && count_enable ;

    always_ff @(posedge i_clock)
    begin : proc_count
        if (i_reset || limit_reached)
            count <= '0;
        else if (count_enable)
            count <= next_count;
    end

    // * ------------------------------------------------------------------
    // * Piping output
```

```
    // * -----------------------------------------------------------------
    generate
        if (ADD_PIPE_OUT)
        begin : gen_out_piped
            always_ff @(posedge i_clock)
            begin : proc_pipe_out
                piped_limit_reached <= limit_reached;
            end
        end else
        begin : gen_out_unpiped
            assign piped_limit_reached = limit_reached;
        end
    endgenerate

    // * -----------------------------------------------------------------
    // * Output assignment
    // * -----------------------------------------------------------------
    assign o_count_reached = piped_limit_reached;
endmodule
```

1.2. Shift register

```
module shiftreg
#(
    // * -----------------------------------------------------------------
    // * Parameters
    // * -----------------------------------------------------------------
    parameter                       NB_DATA   = 3
)
(
    // * -----------------------------------------------------------------
    // * Outputs
    // * -----------------------------------------------------------------
    output logic [ NB_DATA - 1 : 0 ] o_led          ,

    // * -----------------------------------------------------------------
    // * Inputs
    // * -----------------------------------------------------------------
    input  logic                    i_valid        ,

    // * -----------------------------------------------------------------
    // * Clock and reset
    // * -----------------------------------------------------------------
    input  logic                    i_reset        ,
    input  logic                    i_clock
) ;
    // * -----------------------------------------------------------------
    // * Internal logics
    // * -----------------------------------------------------------------
    logic        [ NB_DATA - 1 : 0 ] shift_register ;

    // * -----------------------------------------------------------------
    // * Shift register
    // * -----------------------------------------------------------------
    always_ff @(posedge i_clock)
    begin : proc_data_move
        if (i_reset)
            shift_register <= {{NB_DATA-1{1'b0}}, 1'b1};
        else if (i_valid)
            shift_register <= {shift_register[NB_DATA-2:0], shift_register[NB_DATA-1]};
    end

    // * -----------------------------------------------------------------
    // * Output assignment
    // * -----------------------------------------------------------------
    assign o_led = shift_register;
endmodule
```

1.3. Top

```
module shiftleds
#(
    // * -------------------------------------------------
    // * Parameters
    // * -------------------------------------------------
    parameter                       N_LEDS     =  4       ,
    parameter                       NB_COUNTER = 32       ,
    parameter                       NB_SW      =  4)
(
    // * -------------------------------------------------
    // * Outputs
    // * -------------------------------------------------
    output logic [ N_LEDS  - 1 : 0 ] o_led                ,
    output logic [ N_LEDS  - 1 : 0 ] o_led_b              ,
    output logic [ N_LEDS  - 1 : 0 ] o_led_g              ,

    // * -------------------------------------------------
    // * Inputs
    // * -------------------------------------------------
    input  logic [ NB_SW   - 1 : 0 ] i_sw                 ,

    // * -------------------------------------------------
    // * Clock and reset
    // * -------------------------------------------------
    input  logic                     i_reset              ,
    input  logic                     i_clock
) ;
    // * -------------------------------------------------
    // * Internal logics
    // * -------------------------------------------------
    logic                            counter_count_reached ;
    logic        [ N_LEDS  - 1 : 0 ] shiftreg_led          ;

    // * -------------------------------------------------
    // * Counter instantiation
    // * -------------------------------------------------
    counter
    #(
        .NB_COUNTER      ( NB_COUNTER            ) ,
        .NB_SW           ( NB_SW - 1             )
    )
    u_counter
    (
        .o_count_reached ( counter_count_reached ) ,
        .i_sw            ( i_sw      [NB_SW-2:0] ) ,
        .i_reset         ( ~i_reset              ) ,
        .i_clock         ( i_clock               )
    );

    // * -------------------------------------------------
    // * Shift register instantiation
    // * -------------------------------------------------
    shiftreg
    #(
        .NB_DATA         ( N_LEDS                )
    )
    u_shiftreg
    (
        .o_led           ( shiftreg_led          ) ,
        .i_valid         ( counter_count_reached ) ,
        .i_reset         ( ~i_reset              ) ,
        .i_clock         ( i_clock               )
    ) ;

    // * -------------------------------------------------
    // * Output assignment
    // * -------------------------------------------------
    assign o_led   = shiftreg_led                          ;
    assign o_led_b = (i_sw[3]) ? shiftreg_led : '0         ;
    assign o_led_g = (i_sw[3]) ? '0           : shiftreg_led ;

    // * -------------------------------------------------
    // * Assertions
    // * -------------------------------------------------
    `ifdef ENABLE_SVA
        logic enable_sva = 1'b1;

        assert property (
            @(posedge i_clock)
```

```
            disable iff (!enable_sva)
            counter_count_reached |=> !counter_count_reached
        ) else $error("Valid signal for shift register lasted more than one clock");

        assert property (
            @(posedge i_clock)
            disable iff (!enable_sva)
            $rose(counter_count_reached) |=> (u_counter.count == 0)
        ) else $error("Counter limit reached but count register didnt go back to zero. Count value: %0d", u_counter.count);

        assert property (
            @(posedge i_clock)
            disable iff (!enable_sva)
            (u_counter.count == u_counter.limit_count) |-> $rose(counter_count_reached)
        ) else $error("Limit count reached but count reached never rose");

        assert property (
            @(posedge i_clock)
            disable iff (!enable_sva)
            $fell(i_reset) |=> (u_counter.count == 0)
        ) else $error("Reset negedge happened but counter didnt set to default value. Count value: %0d", u_counter.count);

        assert property (
            @(posedge i_clock)
            disable iff (!enable_sva)
            $fell(i_reset) |=> (u_shiftreg.shift_register == {{N_LEDS-1{1'b0}}, 1'b1})
        ) else $error("Reset negedge happened but shift register didnt set to default value. Value found: %0b", u_shiftreg.shift_register);

        assert property (
            @(posedge i_clock)
            disable iff (!enable_sva)
            !i_sw[0] |=> $stable(u_counter.count)
        ) else $error("Count is not stable during disable");
    `endif
endmodule
```

2. Testbench

Link de github al archivo de testbench.

2.1. Código

```
`timescale 1ns/100ps

`define N_LEDS           4
`define NB_SEL           2
`define NB_COUNTER      32
`define NB_SW            4
`define CLOCK_PERIOD_NS 10

module tb_shiftleds();
    // * -----------------------------------------------
    // * Localparams
    // * -----------------------------------------------
    localparam                  N_LEDS     = `N_LEDS     ;
    localparam                  NB_SEL     = `NB_SEL     ;
    localparam                  NB_COUNTER = `NB_COUNTER ;
    localparam                  NB_SW      = `NB_SW      ;

    // * -----------------------------------------------
    // * Declaration
    // * -----------------------------------------------
    wire [ N_LEDS     - 1 : 0 ] o_led                   ;
    wire [ N_LEDS     - 1 : 0 ] o_led_b                 ;
    wire [ N_LEDS     - 1 : 0 ] o_led_g                 ;
    reg  [ NB_SW      - 1 : 0 ] i_sw                    ;
    reg                         i_reset                 ;
    reg                         i_clock                 ;

    wire [ NB_COUNTER - 1 : 0 ] active_count_limit      ;
    wire [ NB_COUNTER - 1 : 0 ] counter_limit [4]       ;

    reg  [ NB_COUNTER - 1 : 0 ] counter_record          ;

    // * -----------------------------------------------
    // * Observers assigns
    // * -----------------------------------------------
    assign counter_limit[0] = u_shiftleds.u_counter.R0;
    assign counter_limit[1] = u_shiftleds.u_counter.R1;
    assign counter_limit[2] = u_shiftleds.u_counter.R2;
    assign counter_limit[3] = u_shiftleds.u_counter.R3;

    assign active_count_limit = (i_sw[2:1] == 2'b00 ) ? counter_limit[0] :
                                (i_sw[2:1] == 2'b01 ) ? counter_limit[1] :
                                (i_sw[2:1] == 2'b10 ) ? counter_limit[2] :
                                                        counter_limit[3] ;

    // * -----------------------------------------------
    // * Clock generation
    // * -----------------------------------------------
    initial
    begin : proc_clock_initializacion
        i_clock = '0;
    end
    always #(`CLOCK_PERIOD_NS/2) i_clock = ~i_clock;

    // * -----------------------------------------------
    // * Tasks
    // * -----------------------------------------------
    task automatic wait_for_n_clocks(int n_clocks = 1, bit safe_check = 1'b1);
        repeat(n_clocks) @(posedge i_clock);
        if (safe_check) #1ns;
    endtask : wait_for_n_clocks

    task automatic run_reset(int clock_duration = 1);
        wait_for_n_clocks(.n_clocks(1), .safe_check(0));
        force i_reset = 1'b0;
        wait_for_n_clocks(.n_clocks(clock_duration), .safe_check(0));
        force i_reset = 1'b1;
    endtask : run_reset

    task automatic check_led_change(int n_changes = N_LEDS);
        logic [N_LEDS-1:0] led_record;
        repeat (n_changes) begin : proc_check_repeatedly
            led_record = u_shiftleds.o_led;
            wait_for_n_clocks(active_count_limit);
```

```
        assert(u_shiftleds.counter_count_reached == 1'b1);
        wait_for_n_clocks(1);
        assert(o_led == {led_record[N_LEDS-2:0], led_record[N_LEDS-1]});
        if (i_sw[3]) begin
            assert (o_led_b == o_led);
            assert (o_led_g == '0   );
        end else begin
            assert (o_led_b == '0   );
            assert (o_led_g == o_led);
        end
        $display("Led value: %4b - Led recorded: %4b - Expected value: %4b", o_led, led_record, {led_record[N_LEDS-2:0], led_record[N_LEDS-1]}
    end
endtask : check_led_change

// * ------------------------------------------------
// * Assertion handling
// * ------------------------------------------------
initial
begin : proc_bypass_sva_first_clock
    $assertoff(0,u_shiftleds);
    wait_for_n_clocks(.n_clocks(2), .safe_check(0));
    $asserton(0,u_shiftleds);
end

// * ------------------------------------------------
// * Stimulus generation
// * ------------------------------------------------
initial
begin : proc_stimulus
    i_sw    = '0;
    i_reset = 1'b0;
    run_reset(5);
    wait_for_n_clocks(20);

    i_sw[0] = 1'b1;
    check_led_change(8);
    wait_for_n_clocks(4);

    i_sw[2:1] = 2'b01;
    run_reset();
    check_led_change(8);

    i_sw[3]   = 1'b1 ;
    i_sw[2:1] = 2'b10;
    run_reset();
    check_led_change(8);

    i_sw[2:1] = 2'b11;
    run_reset();
    check_led_change(8);

    wait_for_n_clocks(.n_clocks(100), .safe_check(0));
    i_sw[0] = 1'b0;
    wait_for_n_clocks(.n_clocks(2000), .safe_check(0));

    counter_record = u_shiftleds.u_counter.count;
    i_sw[0] = 1'b1;
    wait_for_n_clocks(1);
    assert(u_shiftleds.u_counter.count == counter_record+1);

    $finish;
end

// * ------------------------------------------------
// * DUT instantiation
// * ------------------------------------------------
shiftleds
#(
    .N_LEDS     ( N_LEDS     ) ,
    .NB_COUNTER ( NB_COUNTER ) ,
    .NB_SW      ( NB_SW      )
)
u_shiftleds
(
    .o_led      ( o_led      ) ,
    .o_led_b    ( o_led_b    ) ,
    .o_led_g    ( o_led_g    ) ,
    .i_sw       ( i_sw       ) ,
    .i_reset    ( i_reset    ) ,
    .i_clock    ( i_clock    )
) ;
```

```
endmodule
```

2.2. Capturas de simulación

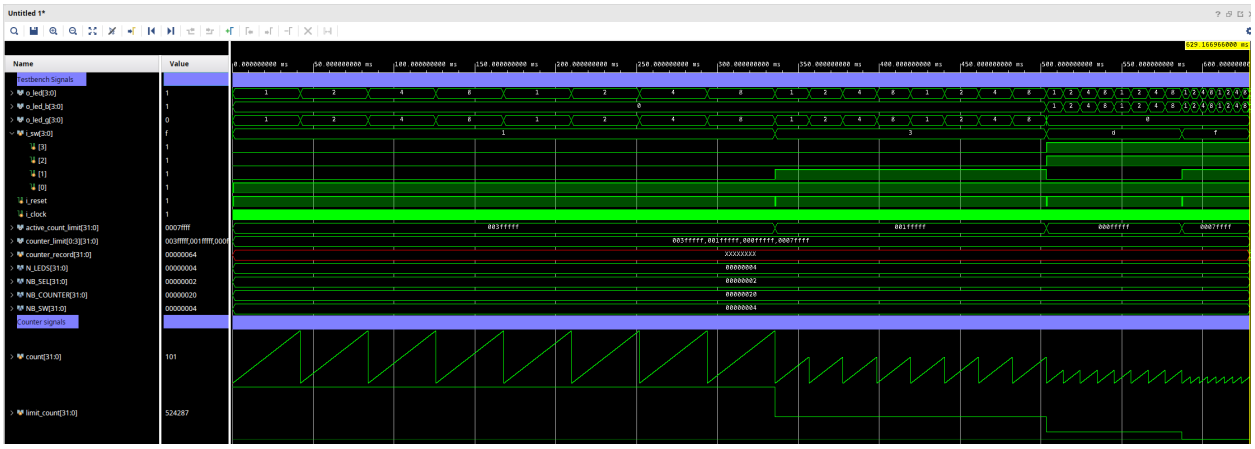


Figura 1: Operación normal

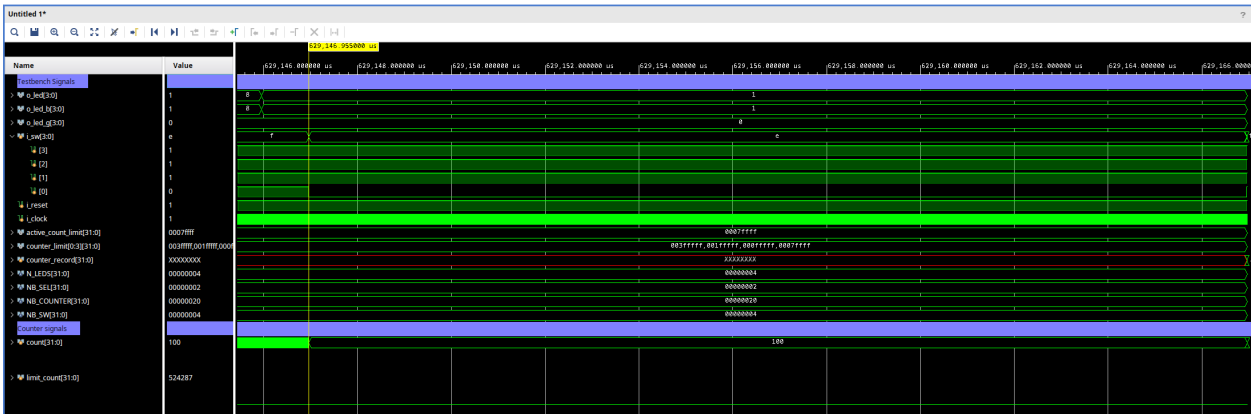


Figura 2: $sw[0] = 0$ mantiene la cuenta y los leds sin alterar

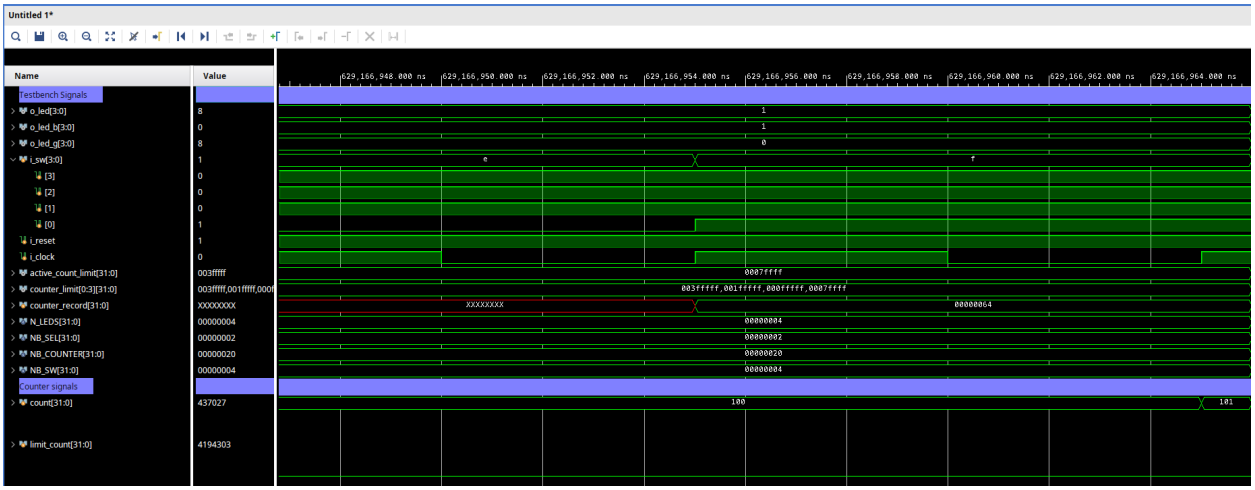


Figura 3: Volver a setear $sw[0] = 1$ retoma la cuenta desde el valor previo

3. Implementación en FPGA

Link de github a los archivos de los wrappers.

3.1. Wrapper VIO

```
//Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.
//Copyright 2022-2023 Advanced Micro Devices, Inc. All Rights Reserved.
//--------------------------------------------------------------------------------
//Tool Version: Vivado v.2023.1 (lin64) Build 3865809 Sun May  7 15:04:56 MDT 2023
//Date        : Sat Sep 28 23:19:27 2024
//Host        : dsktp running 64-bit Fedora Linux 40 (KDE Plasma)
//Command     : generate_target vio_wrapper.bd
//Design      : vio_wrapper
//Purpose     : IP block netlist
//--------------------------------------------------------------------------------
`timescale 1 ps / 1 ps

module vio_wrapper
   (clk_0,
    probe_in0_0,
    probe_in1_0,
    probe_in2_0,
    probe_out0_0,
    probe_out1_0,
    probe_out2_0);
  input clk_0;
  input [3:0]probe_in0_0;
  input [3:0]probe_in1_0;
  input [3:0]probe_in2_0;
  output [0:0]probe_out0_0;
  output [0:0]probe_out1_0;
  output [3:0]probe_out2_0;

  wire clk_0;
  wire [3:0]probe_in0_0;
  wire [3:0]probe_in1_0;
  wire [3:0]probe_in2_0;
  wire [0:0]probe_out0_0;
  wire [0:0]probe_out1_0;
  wire [3:0]probe_out2_0;

  vio vio_i
       (.clk_0(clk_0),
        .probe_in0_0(probe_in0_0),
        .probe_in1_0(probe_in1_0),
        .probe_in2_0(probe_in2_0),
        .probe_out0_0(probe_out0_0),
        .probe_out1_0(probe_out1_0),
        .probe_out2_0(probe_out2_0));
endmodule
```

3.2. Wrapper ILA

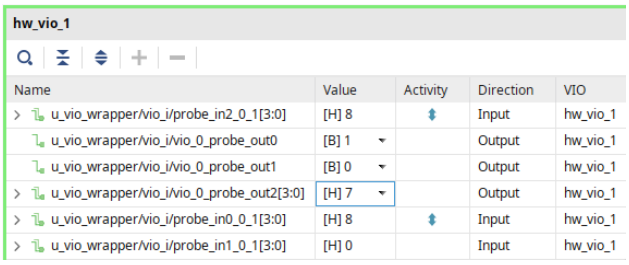
```
//Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.
//Copyright 2022-2023 Advanced Micro Devices, Inc. All Rights Reserved.
//--------------------------------------------------------------------------------
//Tool Version: Vivado v.2023.1 (lin64) Build 3865809 Sun May  7 15:04:56 MDT 2023
//Date        : Sat Sep 28 23:21:04 2024
//Host        : dsktp running 64-bit Fedora Linux 40 (KDE Plasma)
//Command     : generate_target ila_wrapper.bd
//Design      : ila_wrapper
//Purpose     : IP block netlist
//--------------------------------------------------------------------------------
`timescale 1 ps / 1 ps

module ila_wrapper
   (clk_0,
    probe0_0,
    probe1_0,
    probe2_0);
  input clk_0;
  input [3:0]probe0_0;
  input [3:0]probe1_0;
  input [3:0]probe2_0;

  wire clk_0;
  wire [3:0]probe0_0;
  wire [3:0]probe1_0;
  wire [3:0]probe2_0;

  ila ila_i
       (.clk_0(clk_0),
        .probe0_0(probe0_0),
        .probe1_0(probe1_0),
        .probe2_0(probe2_0));
endmodule
```

3.3. Wrapper completo

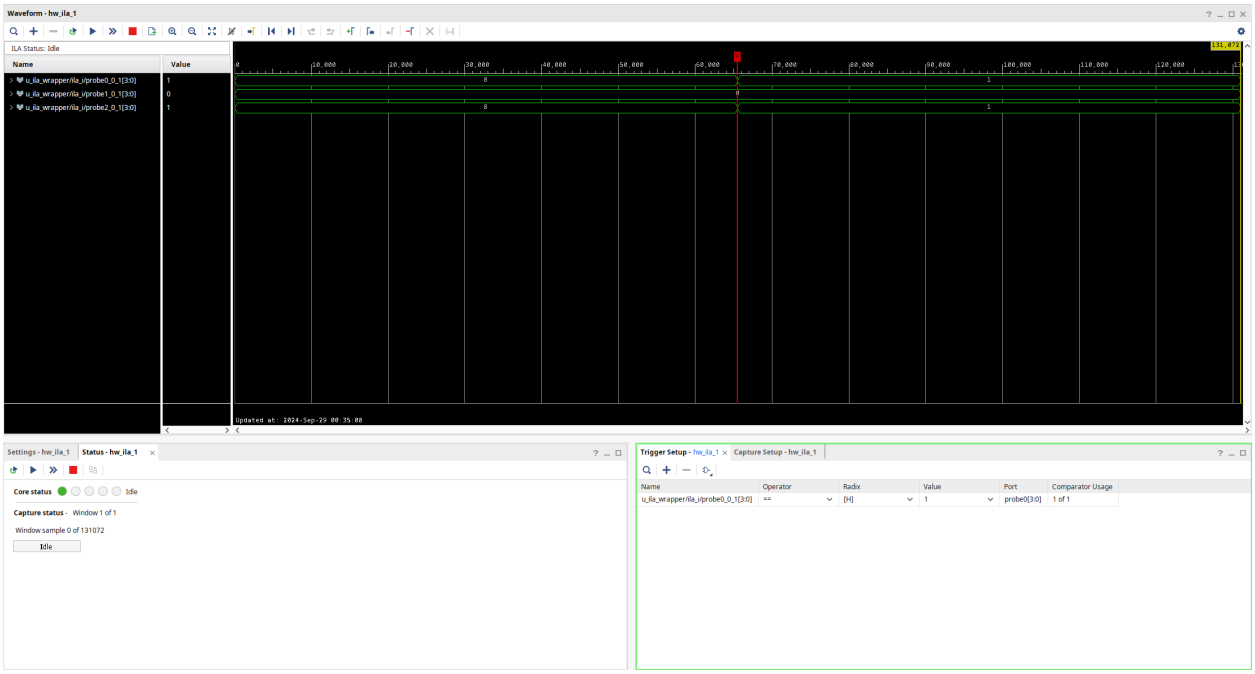
```
module shiftleds_wrapper (
    // * -------------------------------------------------
    // * Outputs
    // * -------------------------------------------------
    output logic [4-1:0] o_led          ,
    output logic [4-1:0] o_led_b        ,
    output logic [4-1:0] o_led_g        ,

    // * -------------------------------------------------
    // * Inputs
    // * -------------------------------------------------
    input logic  [4-1:0] i_sw           ,

    // * -------------------------------------------------
    // * Clock and reset
    // * -------------------------------------------------
    input logic          i_reset_n      ,
    input logic          i_clock
) ;
    // * -------------------------------------------------
    // * Internal logics
    // * -------------------------------------------------
    logic        [4-1:0] shiftleds_led   ;
    logic        [4-1:0] shiftleds_led_b ;
    logic        [4-1:0] shiftleds_led_g ;
    logic        [4-1:0] vio_sw          ;
    logic                vio_reset       ;
    logic                vio_enable      ;
    logic        [4-1:0] sw_to_device    ;
    logic                reset_to_device ;

    // * -------------------------------------------------
    // * VIO instantion
    // * -------------------------------------------------
    vio_wrapper
    u_vio_wrapper
    (
        .clk_0        ( i_clock         ) ,
        .probe_in0_0  ( shiftleds_led   ) ,
        .probe_in1_0  ( shiftleds_led_b ) ,
        .probe_in2_0  ( shiftleds_led_g ) ,
        .probe_out0_0 ( vio_enable      ) ,
        .probe_out1_0 ( vio_reset       ) ,
        .probe_out2_0 ( vio_sw          )
    );

    // * -------------------------------------------------
    // * ILA instantion
    // * -------------------------------------------------
    ila_wrapper
    u_ila_wrapper
    (
        .clk_0        ( i_clock         ) ,
        .probe0_0     ( shiftleds_led   ) ,
        .probe1_0     ( shiftleds_led_b ) ,
        .probe2_0     ( shiftleds_led_g )
    );

    // * -------------------------------------------------
    // * Muxing shiftleds input
    // * -------------------------------------------------
    assign sw_to_device    = (vio_enable) ? vio_sw     : i_sw      ;
    assign reset_to_device = (vio_enable) ? ~vio_reset : i_reset_n ;

    // * -------------------------------------------------
    // * Main device
    // * -------------------------------------------------
    shiftleds
    u_shiftleds
    (
        .o_led        ( shiftleds_led   ) ,
        .o_led_b      ( shiftleds_led_b ) ,
        .o_led_g      ( shiftleds_led_g ) ,
        .i_sw         ( sw_to_device    ) ,
        .i_reset      ( reset_to_device ) ,
        .i_clock      ( i_clock         )
    ) ;

    // * -------------------------------------------------
```

```
    // * Output assign
    // * ------------------------------------------------
    assign o_led   = shiftleds_led   ;
    assign o_led_b = shiftleds_led_b ;
    assign o_led_g = shiftleds_led_g ;
endmodule
```

3.4. Capturas

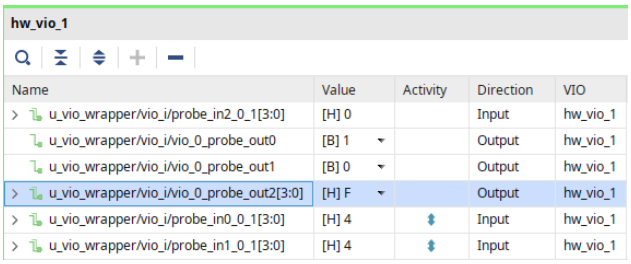


(a) VIO

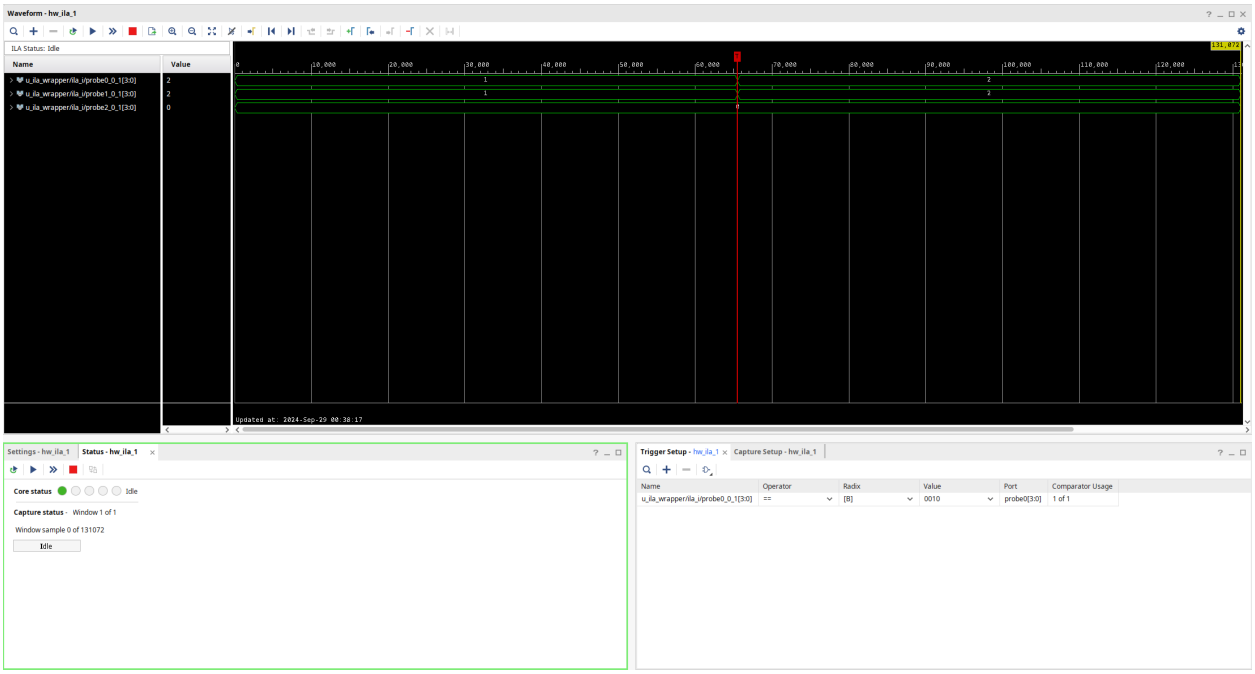


(b) ILA

Figura 4: Led verde seleccionado

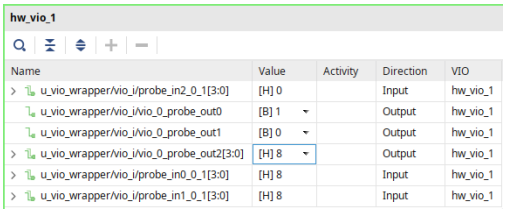


(a) VIO

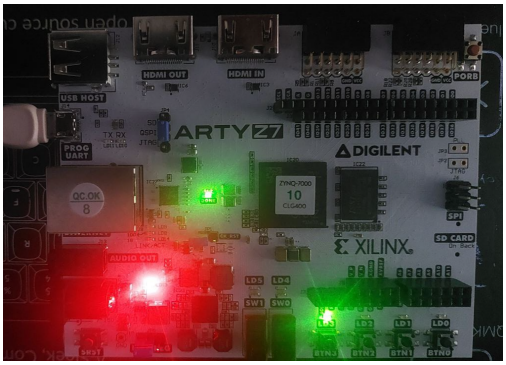


(b) ILA

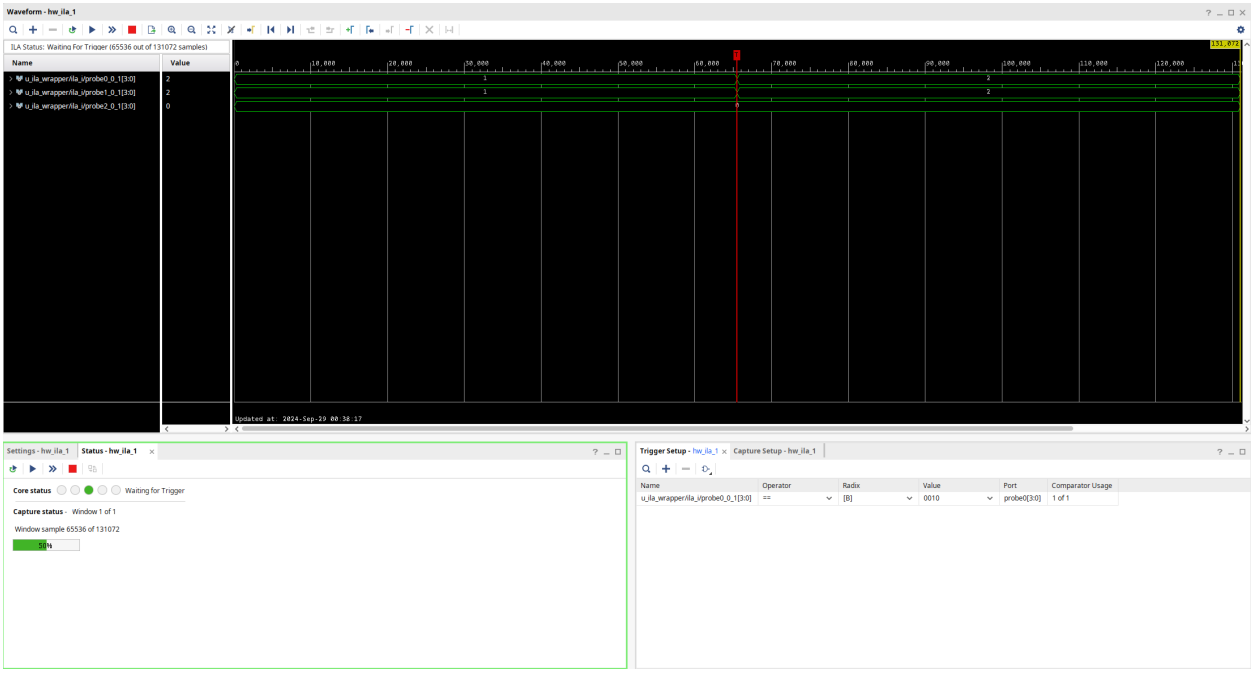
Figura 5: Led azul seleccionado



(a) VIO



(b) FPGA - Led 4 prendido



(c) ILA - Sampling constante en 50 %

Figura 6: Led quieto

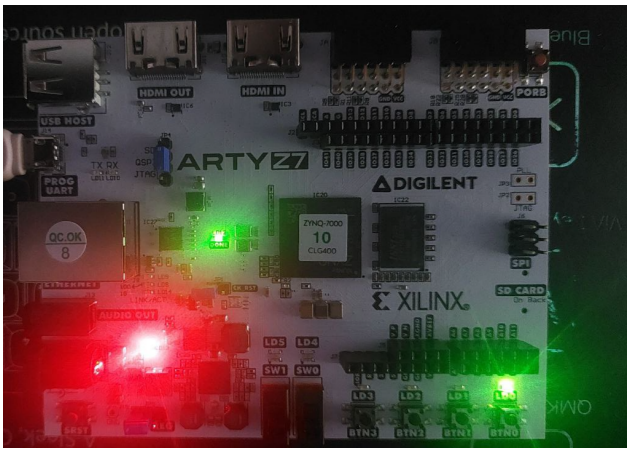


Figura 7: FPGA en reset - Led 0 prendido