VLSI DESIGN FLOW RTL TO GDS

PROJECT REPORT

Ishaan Singhal

Delhi Technological University

Introduction

The aim of this project is to understand and implement the **VLSI design flow**, starting from Register Transfer Level (RTL) coding and continuing up to Design for Testability (DFT). The design flow covers all the important stages that are followed in industry, but in this project, only open-source tools are used instead of licensed EDA software.

The project begins with **RTL design**, where Verilog is used to describe simple digital logic modules. The modules designed include a clock divider, barrel shifter, linear feedback shift register (LFSR), and a parity generator. These are combined in a top-level RTL module, where a control signal decides which block's output is selected.

Next, **simulation and verification** are carried out. Using **Icarus Verilog**, the RTL is compiled and executed with different testbenches, and the results are viewed in **GTKWave**. This step ensures that the written Verilog behaves as expected. To check the quality of verification, **code coverage analysis** is done using the **Covered** tool, which shows how much of the RTL code is exercised by the testbenches.

Afterwards, the flow moves towards **logic synthesis**. In this stage, the RTL code is converted into a gate-level netlist using an open-source synthesis tool such as **Yosys**. The synthesis step allows the design to be mapped to standard logic gates, making it closer to real hardware implementation.

Once synthesis is complete, **Static Timing Analysis (STA)** is performed to verify that the design meets timing requirements. This step ensures that the design can run at the intended clock frequency without timing violations.

Finally, the project studies **Design for Testability (DFT)** concepts, which make it easier to test manufactured chips for faults. Scan chains and test structures are explored to understand how DFT is included in the flow before the design can move to physical implementation.

Logic Blocks Used

Clock Divider: The clock divider reduces the frequency of an input clock signal. It works by using a counter that counts input clock cycles and toggles the output when the counter reaches a fixed value. This makes the output clock slower than the input clock. Clock dividers are commonly used to generate lower-frequency clocks from a high-speed system clock.

Barrel Shifter: The barrel shifter performs shifting operations on input data. It can shift the bits of the input left or right by a specified number of positions. Unlike normal shifters that take multiple cycles, a barrel shifter can complete the shift in a single operation using multiplexers. Barrel shifters are widely used in processors for fast multiplication, division, and bit manipulation.

Linear Feedback Shift Register: The Linear Feedback Shift Register (LFSR) is a sequential circuit that generates a sequence of pseudo-random bits. It consists of a shift register whose input is formed by XORing certain outputs (called taps). Because the sequence looks random but is generated deterministically, LFSRs are used in test pattern generation, encryption, and error-checking systems.

Parity Generator: The parity generator is used to generate a parity bit for error detection. The parity bit indicates whether the number of 1s in the input data is even or odd. This is achieved by XORing all the bits of the input word. Parity bits are widely used in digital communication and memory systems to detect errors in transmitted or stored data.

Design Specifications

For this design we read **A** and **C** as inputs into registers. Based on the value that **C** takes, we perform different operations on **A** (and internal logic) and write the results to the **out** port through a register. The **clk** and **rst** signals are also added as inputs. The reset is synchronous.

Specifications & Assumptions:

- **Inputs:** A9:09:09:0, C8:08:08:0
- **Output:** out10:010:010:0
- The design is **synchronous** and operations are performed on the positive edge of the clock.
- The reset port is synchronous with the clock, so on the next positive clock edge the input registers are reset to **0**.
- Based on the value of **C**, different operations take place:
    - When **C < 51** ⇒ **Clock Divider**
    - When **51 ≤ C < 100** ⇒ **Barrel Shifter**
    - When **100 ≤ C ≤ 128** ⇒ **Parity Generator**
    - Else (**C > 128**) ⇒ **Linear Feedback Shift Register (LFSR)**
- Based on these specifications, we chose the sizes of A, C, and the output bus.

Verilog Codes

rtl_topmodule.v

```
`timescale 1ns / 1ps
module rtl_topmodule (
    input  wire        clk,
    input  wire        rst,
    input  wire [9:0]  A,
    input  wire [8:0]  C,
    output reg  [10:0] out
);
    reg [9:0]  A_reg;
    reg [8:0]  C_reg;
    reg [10:0] out_reg;
    always @(posedge clk) begin
        if (rst) begin
            A_reg <= 10'd0;
            C_reg <= 9'd0;
        end else begin
            A_reg <= A;
            C_reg <= C;
        end
    end

    wire clk_div;
    clk_divider #(.div_value(4)) u1 (
        .clk_in (clk),
        .clk_out(clk_div)
    );

    wire [9:0] shifter_out;
    barrel_shifter u2 (
        .data_in (A_reg),
        .shift_amt(C_reg[2:0]),
        .data_out(shifter_out)
    );

    wire [9:0] lfsr_out;
    lfsr #(.WIDTH(10)) u3 (
        .clk(clk),
        .rst(rst),
        .rnd(lfsr_out)
```

```
    );

    wire parity_out;
    parity_gen u4 (
        .data_in(A_reg),
        .parity (parity_out)
    );

    always @(*) begin
        if (C_reg < 9'd51) begin
            out_reg = {10'd0, clk_div};
        end else if (C_reg < 9'd100) begin
            out_reg = {1'd0, shifter_out};
        end else if (C_reg <= 9'd128) begin
            out_reg = {10'd0, parity_out};
        end else begin
            out_reg = {1'd0, lfsr_out};
        end
    end

    always @(posedge clk) begin
        if (rst) out <= 11'd0;
        else     out <= out_reg;
    end
endmodule
```

clk_divider.v

```
// clk_divider.v
module clk_divider #(parameter div_value = 10)(
    input  wire clk_in,
    input  wire rst,
    output reg  clk_out = 1'b0);
    reg [3:0] count = 4'd0;
    always @(posedge clk_in or posedge rst) begin
        if (rst) begin
            // This logic is active on the RISING EDGE of reset.
            count   <= 4'd0;
            clk_out <= 1'b0;
        end else begin
            // This logic is active on the RISING EDGE of the clock.
            if (count == div_value - 1) begin
                count   <= 4'd0;
                clk_out <= ~clk_out;
            end else begin
                count <= count + 1;
            end
        end
    end
endmodule
```

barrel_shifter.v

```
`timescale 1ns / 1ps

module barrel_shifter (
    input  wire [9:0] data_in,     // 10-bit input
    input  wire [2:0] shift_amt,   // shift amount (0-7)
    output reg  [9:0] data_out     // shifted result
);

    always @(*) begin
        case (shift_amt)
            3'd0: data_out = data_in;
            3'd1: data_out = data_in << 1;
            3'd2: data_out = data_in << 2;
            3'd3: data_out = data_in << 3;
            3'd4: data_out = data_in << 4;
            3'd5: data_out = data_in << 5;
            3'd6: data_out = data_in << 6;
            3'd7: data_out = data_in >> 1; // just for variety, shift right
            default: data_out = data_in;
        endcase
    end

endmodule
```

parity_gen.v

```
`timescale 1ns / 1ps

module parity_gen (
    input  wire [9:0] data_in,
    output wire       parity
);

    assign parity = ^data_in;  // XOR reduction = parity bit

endmodule
```

lfsr.v

```
`timescale 1ns / 1ps

module lfsr #(parameter WIDTH = 10)(
    input  wire clk,
    input  wire rst,
    output reg [WIDTH-1:0] rnd
);

    always @(posedge clk or posedge rst) begin
        if (rst) begin
            rnd <= {WIDTH{1'b1}}; // seed: all 1s
        end else begin
            // Example taps for 10-bit LFSR: bits 9 and 6
            rnd <= {rnd[WIDTH-2:0], rnd[WIDTH-1] ^ rnd[6]};
        end
    end

endmodule
```

testbench_clk_divider.v : This only tests clk_divider.

```
`timescale 1ns/1ns

module testbench_1;

    // Declare signals
    reg clk;
    reg rst;
    reg [9:0] A;
    reg [8:0] C;
    wire [10:0] out;

    // Instantiate the DUT
    rtl_topmodule dut (
        .clk(clk),
        .rst(rst),
        .A(A),
        .C(C),
        .out(out)
    );

    // Clock generation: combined into one block to avoid race warning
    initial begin
        clk = 1'b0;
        forever #5 clk = ~clk;
    end

    // Stimulus process
    initial begin
        $display($time, " ===== Testbench Start ===== ");
        A   = 10'd15;
        C   = 9'd0;
        rst = 1;

        #24 rst = 0;
        #20 C = 9'd12; // This will trigger the clk_divider because C < 50
    end

    // VCD dump for waveform and coverage
    initial begin
        $dumpfile("testbench1.vcd");
        $dumpvars(0, testbench_1);  // Dump all hierarchy below testbench_1
    end
```

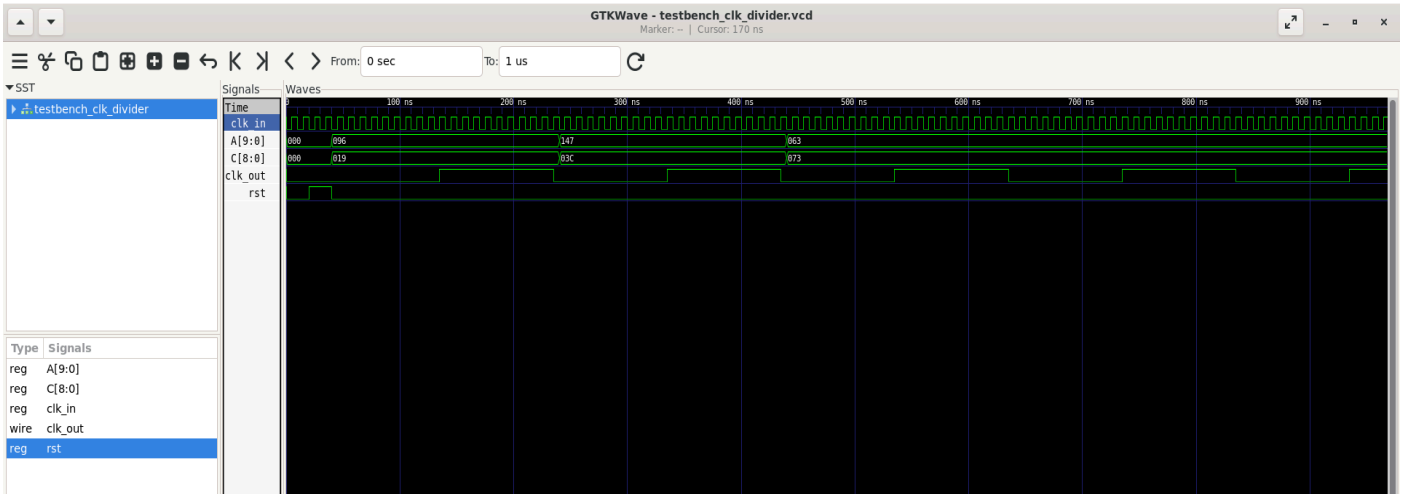
```
    // Monitor output ( Output would be shown on the terminal)
    initial begin
        $monitor("Time=%0t | clk=%b rst=%b | A=%d C=%d | out=%d",
                 $time, clk, rst, A, C, out);
    end

    // Terminate simulation after a fixed time
    initial begin
        #500 $finish;
    end

endmodule
```

GTKWave Waveform for testbench_clk_divder.v

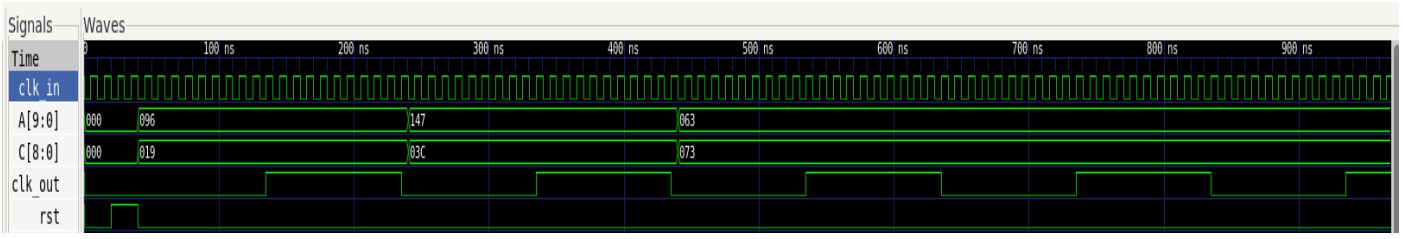
This code exclusively executes the testbench for the clock divider logic. The input 'C' is irrelevant here because rtl_topmodule.v is not passed. As depicted in the screenshot, the logic will trigger and waveforms will be generated regardless of the 'C' input.

Code to run the simulation:

- `iverilog -o sim_clk_only clk_divider.v testbench_clk_divider.v`
- `vvp sim_clk_only`
- `gtkwave testbench_clk_divider.vcd`



Enlarged View



Code Coverage for testbench_clk_divider.v

Code to run the simulation:

- covered score -v clk_divider.v -v testbench_clk_divider.v -t testbench_clk_divider -vcd testbench_clk_divider.vcd -o coverage_only_clk.cdd
- covered report -d v coverage_only_clk.cdd > coverage_report_only_clock.txt

Generated coverage_report_clk_only.txt:

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~   GENERAL INFORMATION   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* Report generated from CDD file : coverage_only_clk.cdd

* Reported by                    : Module

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~   LINE COVERAGE RESULTS   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Module/Task/Function      Filename                Hit/ Miss/Total    Percent hit
---------------------------------------------------------------------------------------------------------
  $root                   NA                        0/    0/    0      100%
  testbench_clk_divide    testbench_clk_divide     13/    0/   13      100%
  clk_divider             clk_divider.v             6/    2/    8       75%
---------------------------------------------------------------------------------------------------------
  Accumulated                                      19/    2/   21       90%
---------------------------------------------------------------------------------------------------------
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~   TOGGLE COVERAGE RESULTS   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                                                  Toggle 0 -> 1                       Toggle 1 -> 0
Module/Task/Function      Filename                Hit/ Miss/Total    Percent hit      Hit/ Miss/Total    Percent hit
-------------------------------------------------------------------------------------------------------------------
  $root                   NA                        0/    0/    0      100%             0/    0/    0      100%
  testbench_clk_divide    testbench_clk_divide     18/    4/   22       82%            10/   12/   22       45%
  clk_divider             clk_divider.v             3/    4/    7       43%             3/    4/    7       43%
-------------------------------------------------------------------------------------------------------------------
  Accumulated                                      21/    8/   29       72%            13/   16/   29       45%
-------------------------------------------------------------------------------------------------------------------
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~   COMBINATIONAL LOGIC COVERAGE RESULTS   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                                                                          Logic Combinations
Module/Task/Function                Filename                          Hit/Miss/Total    Percent hit
---------------------------------------------------------------------------------------------------------------------
  $root                             NA                                  0/    0/    0       100%
  testbench_clk_divider             testbench_clk_divider.v             2/    0/    2       100%
  clk_divider                       clk_divider.v                       6/    4/   10        60%
---------------------------------------------------------------------------------------------------------------------
  Accumulated                                                           8/    4/   12        67%
---------------------------------------------------------------------------------------------------------------------
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~   FINITE STATE MACHINE COVERAGE RESULTS   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                                                              State                              Arc
Module/Task/Function       Filename                 Hit/Miss/Total    Percent Hit    Hit/Miss/Total    Percent hit
---------------------------------------------------------------------------------------------------------------------
  $root                    NA                          0/    0/    0      100%          0/    0/    0      100%
  testbench_clk_divide     testbench_clk_divide        0/    0/    0      100%          0/    0/    0      100%
  clk_divider              clk_divider.v               0/    0/    0      100%          0/    0/    0      100%
---------------------------------------------------------------------------------------------------------------------
  Accumulated                                          0/    0/    0      100%          0/    0/    0      100%
```

testbench_clk_rtl.v : This only tests clk_divider but with the top_module.

```
`timescale 1ns/1ns

module testbench_clk_rtl;

    // Declare signals
    reg clk;
    reg rst;
    reg [9:0] A;
    reg [8:0] C;
    wire [10:0] out;

    // Instantiate the DUT (Device Under Test)
    rtl_topmodule dut (
        .clk(clk),
        .rst(rst),
        .A(A),
        .C(C),
        .out(out)
    );

    // Clock generation
    initial begin
        clk = 1'b0;
        forever #5 clk = ~clk;
    end

    // ============================================================
    // --- Stimulus process with THE CORRECTED RESET SEQUENCE ---
    // ============================================================
    initial begin
        $display($time, " ===== Testbench Start ===== ");

        // 1. Start with reset LOW to allow for a rising edge.
        rst = 0;
```

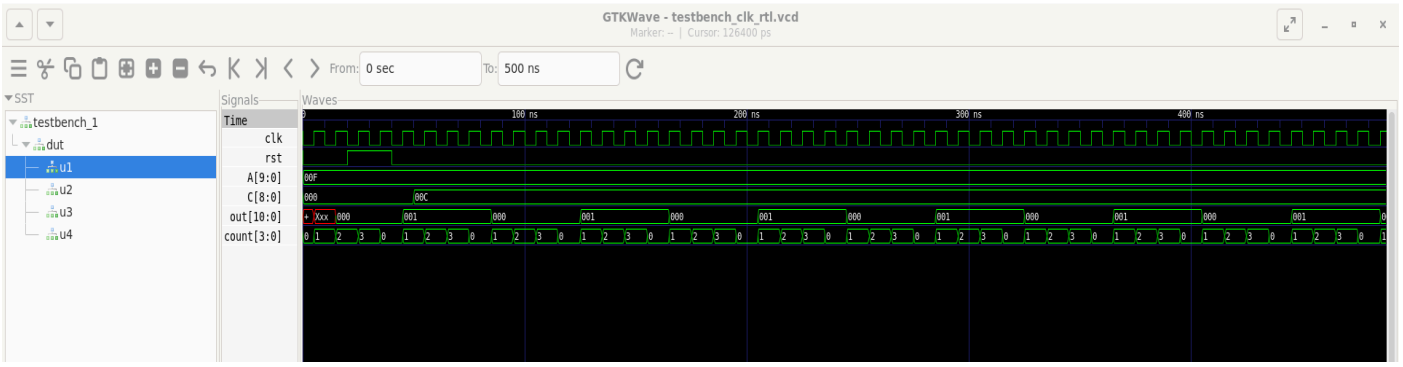
```
        A   = 10'd15;
        C   = 9'd0;
        #20; // Wait for 2 clock cycles to ensure simulation is stable.

        // 2. Apply the reset pulse (0 -> 1 -> 0). This will correctly initialize
the counter.
        rst = 1;
        #20; // Hold reset high for 20ns.
        rst = 0; // De-assert reset to begin normal operation.

        // 3. Now, apply the stimulus to test the clock divider.
        #10; // Wait a moment after reset.
        C = 9'd12; // This will trigger the clk_divider because C < 51.
    end

    // VCD dump for waveform and coverage
    initial begin
        $dumpfile("testbench_clk_rtl.vcd");
        $dumpvars(0, testbench_clk_rtl);  // Dump all hierarchy below testbench_1
    end

    // Monitor output (Output would be shown on the terminal)
    initial begin
        $monitor("Time=%0t | clk=%b rst=%b | A=%d C=%d | out=%d",
                 $time, clk, rst, A, C, out);
    end

    // Terminate simulation after a fixed time
    initial begin
        #500 $finish;
    end

endmodule
```

GTKWave Waveform for testbench_clk_rtl.v

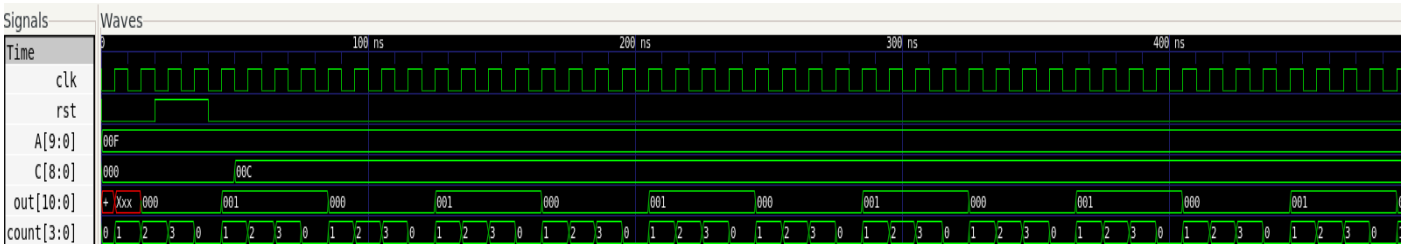
This code executes the complete testbench, encompassing the clk_divider, parity generator, LFSR, and barrel shifter. However, only the clk_divider will be activated due to the input condition C < 50.

Code to run the simulation:

- `iverilog -o sim_clk_rtl rtl_topmodule.v clk_divider.v barrel_shifter.v lfsr.v parity_gen.v testbench_clk_rtl.v`
- `vvp sim_clk_rtl`
- `gtkwave testbench_clk_rtl.vcd`



Enlarged View



Code Coverage for testbench_clk_rtl.v

Code to run the simulation:

- covered score -v rtl_topmodule.v -v clk_divider.v -v barrel_shifter.v -v lfsr.v -v parity_gen.v -v testbench_clk_rtl.v -t testbench_clk_rtl -vcd testbench_clk_rtl.vcd -o coverage_clk_rtl.cdd
- covered report -d v coverage_clk_rtl.cdd > coverage_report_clk_rtl.txt

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~   GENERAL INFORMATION   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* Report generated from CDD file : coverage_clk_rtl.cdd

* Reported by                    : Module

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~   LINE COVERAGE RESULTS   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Module/Task/Function      Filename                Hit/ Miss/Total    Percent hit
---------------------------------------------------------------------------------------------------------------------
  $root                   NA                        0/    0/    0      100%
  testbench_clk_rtl       testbench_clk_rtl.v       8/    0/    8      100%
  rtl_topmodule           rtl_topmodule.v          16/    2/   18       89%
  clk_divider             clk_divider.v             4/    4/    8       50%
  barrel_shifter          barrel_shifter.v          4/    6/   10       40%
  lfsr                    lfsr.v                    4/    0/    4      100%
  parity_gen              parity_gen.v              1/    0/    1      100%
---------------------------------------------------------------------------------------------------------------------
  Accumulated                                      37/   12/   49       76%
---------------------------------------------------------------------------------------------------------------------
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~   TOGGLE COVERAGE RESULTS   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                                                         Toggle 0 -> 1                      Toggle 1 -> 0
Module/Task/Function      Filename                Hit/ Miss/Total    Percent hit    Hit/ Miss/Total    Percent hit
---------------------------------------------------------------------------------------------------------------------
  $root                   NA                        0/    0/    0      100%           0/    0/    0      100%
  testbench_clk_rtl       testbench_clk_rtl.v       5/   27/   32       16%           3/   29/   32        9%
  rtl_topmodule           rtl_topmodule.v          31/   53/   84       37%          23/   61/   84       27%
  clk_divider             clk_divider.v             2/    5/    7       29%           2/    5/    7       29%
  barrel_shifter          barrel_shifter.v         13/   10/   23       57%           8/   15/   23       35%
  lfsr                    lfsr.v                   12/    0/   12      100%          12/    0/   12      100%
  parity_gen              parity_gen.v              4/    7/   11       36%           4/    7/   11       36%
---------------------------------------------------------------------------------------------------------------------
  Accumulated                                      67/  102/  169       40%          52/  117/  169       31%
---------------------------------------------------------------------------------------------------------------------
```

COMBINATIONAL LOGIC COVERAGE RESULTS

		Logic Combinations	
Module/Task/Function	Filename	Hit/Miss/Total	Percent hit
$root	NA	0/ 0/ 0	100%
testbench_clk_rtl	testbench_clk_rtl.v	2/ 0/ 2	100%
rtl_topmodule	rtl_topmodule.v	15/ 12/ 27	56%
clk_divider	clk_divider.v	3/ 7/ 10	30%
barrel_shifter	barrel_shifter.v	6/ 15/ 21	29%
lfsr	lfsr.v	9/ 1/ 10	90%
parity_gen	parity_gen.v	1/ 1/ 2	50%
Accumulated		36/ 36/ 72	50%

FINITE STATE MACHINE COVERAGE RESULTS

		State				Arc			
Module/Task/Function	Filename	Hit/Miss/Total			Percent Hit	Hit/Miss/Total			Percent hit
$root	NA	0/	0/	0	100%	0/	0/	0	100%
testbench_clk_rtl	testbench_clk_rtl.v	0/	0/	0	100%	0/	0/	0	100%
rtl_topmodule	rtl_topmodule.v	0/	0/	0	100%	0/	0/	0	100%
clk_divider	clk_divider.v	0/	0/	0	100%	0/	0/	0	100%
barrel_shifter	barrel_shifter.v	0/	0/	0	100%	0/	0/	0	100%
lfsr	lfsr.v	0/	0/	0	100%	0/	0/	0	100%
parity_gen	parity_gen.v	0/	0/	0	100%	0/	0/	0	100%
Accumulated		0/	0/	0	100%	0/	0/	0	100%

testbench_barrel_shifter_rtl.v : This only tests barrel_shifter but with the top_module.

```
// testbench_top_shifter.v
`timescale 1ns/1ns

module testbench_top_shifter;

    reg clk, rst;
    reg [9:0] A;
    reg [8:0] C;
    wire [10:0] out;

    rtl_topmodule dut (
        .clk(clk), .rst(rst), .A(A), .C(C), .out(out)
    );

    // Clock and VCD setup
    initial begin clk = 0; forever #5 clk = ~clk; end
    initial begin $dumpfile("testbench_barrel_shifter_rtl.vcd"); $dumpvars(0, testbench_top_shifter); end

    // Stimulus
    initial begin
        // Standard reset pulse
        rst = 0; A = 0; C = 0;
        #20; rst = 1; #20; rst = 0;

        // --- Test the Barrel Shifter Mode ---
        $display("Testing Barrel Shifter...");

        // Set C to a value between 51 and 99
        // C=70 means C[2:0] is 3'b110, or 6. We expect a shift left by 6.
        C = 9'd70;
        A = 10'b0000001101; // Input value
        // Expected result: 10'b0011010000
        #100;
```

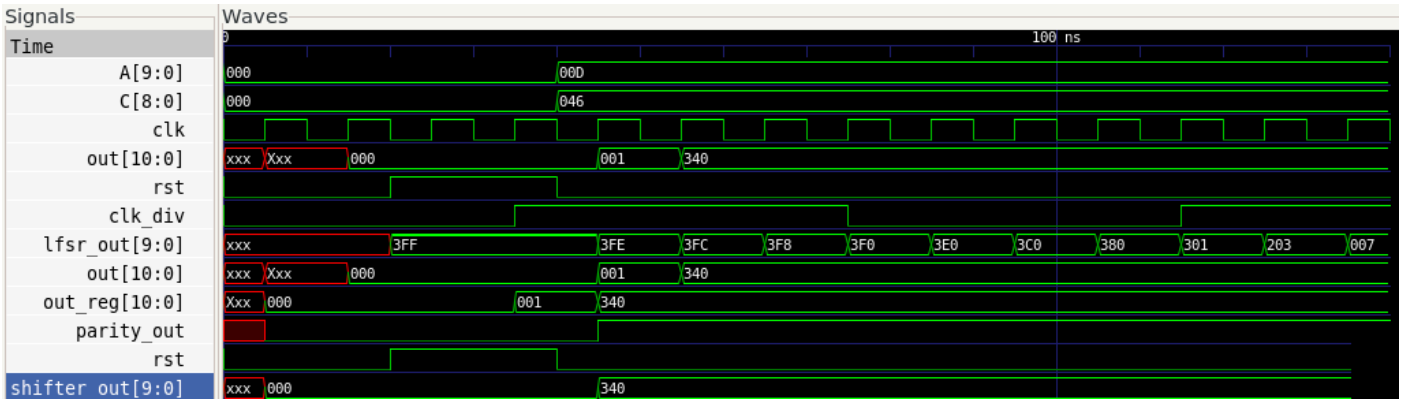
```
        $finish;
    end

    initial begin
        $monitor("Time=%0t | C=%d, A=%b | out[9:0]=%b", $time, C, A, out[9:0]);
    end

endmodule
```

GTKWave Waveform for testbench_barrel_shifter_rtl.v

Only the barrel_shifter will be activated due to the input condition C = 70 (b/w 51 and 100)

Code to run the simulation:

- `iverilog -o sim_barrel_shifter_rtl rtl_topmodule.v clk_divider.v barrel_shifter.v lfsr.v parity_gen.v testbench_clk_rtl.v`
- `vvp sim_barrel_shifter_rtl`
- `gtkwave testbench_barrel_shifter_rtl.vcd`



Analysis:

1. **`rst`:** You can see the perfect `0 -> 1 -> 0` reset pulse at the beginning of the simulation. This is excellent.
2. **`C[8:0]`:** After the reset, the `C` input changes from `000` to `046` (hexadecimal). `0x046` is **70** in decimal. This is exactly what you programmed the testbench to do.
3. **The `if-else` Logic:** Because `C` is `70` (which is `>= 51` and `< 100`), the `rtl_topmodule` correctly executes this line of code: `out_reg = {1'd0, shifter_out};`
4. **`A[9:0]`:** Your testbench sets `A` to `00D` (hex), which is `0000001101` in binary. This is the `data_in` for the shifter.
5. **`shifter_out[9:0]`:** This is the internal output of the barrel shifter.
   - The shifter's `shift_amt` input is connected to `C[2:0]`.
   - Since `C` is `70` (`...1000110`), `C[2:0]` is `110`, which is **6** in decimal.
   - The shifter takes `A` (`0000001101`) and shifts it left by 6.
   - The result is `0011010000`, which is **`340`** in hexadecimal.
   - The waveform correctly shows `shifter_out` becoming `340`.
6. **`out[10:0]`:** This is the final output of the entire system. It shows the value of `shifter_out` (`340`) being registered on the next clock edge. This proves that the `rtl_topmodule` correctly selected the shifter and connected its output to the final stage.

Code Coverage for testbench_barrel_shifter_rtl.v

Code to run the simulation:

- covered score -v rtl_topmodule.v -v clk_divider.v -v barrel_shifter.v -v lfsr.v -v parity_gen.v -v testbench_barrel_shifter_rtl.v -t testbench_barrel_shifter_rtl -vcd testbench_barrel_shifter_rtl.vcd -o coverage_barrel_shifter_rtl.cdd
- covered report -d v coverage_barrel_shifter_rtl.cdd > coverage_report_barrel_shifter_rtl.txt

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~   GENERAL INFORMATION   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* Report generated from CDD file : ./coverage_barrel_shifter_rtl.cdd

* Reported by                    : Module

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~   LINE COVERAGE RESULTS   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Module/Task/Function      Filename                Hit/ Miss/Total    Percent hit
---------------------------------------------------------------------------------------------------
  $root                   NA                        0/    0/    0      100%
  testbench_barrel_shi    testbench_barrel_shi      9/    0/    9      100%
  rtl_topmodule           rtl_topmodule.v          17/    1/   18       94%
  clk_divider             clk_divider.v             4/    4/    8       50%
  barrel_shifter          barrel_shifter.v          4/    6/   10       40%
  lfsr                    lfsr.v                    4/    0/    4      100%
  parity_gen              parity_gen.v              1/    0/    1      100%
---------------------------------------------------------------------------------------------------
  Accumulated                                      39/   11/   50       78%
---------------------------------------------------------------------------------------------------
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~   TOGGLE COVERAGE RESULTS   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                                                  Toggle 0 -> 1                       Toggle 1 -> 0
Module/Task/Function      Filename                Hit/ Miss/Total    Percent hit      Hit/ Miss/Total    Percent hit
--------------------------------------------------------------------------------------------------------------------
  $root                   NA                        0/    0/    0      100%             0/    0/    0      100%
  testbench_barrel_shi    testbench_barrel_shi     12/   20/   32       38%             3/   29/   32        9%
  rtl_topmodule           rtl_topmodule.v          30/   54/   84       36%            15/   69/   84       18%
  clk_divider             clk_divider.v             2/    5/    7       29%             2/    5/    7       29%
  barrel_shifter          barrel_shifter.v          8/   15/   23       35%             0/   23/   23        0%
  lfsr                    lfsr.v                    5/    7/   12       42%            12/    0/   12      100%
  parity_gen              parity_gen.v              4/    7/   11       36%             0/   11/   11        0%
--------------------------------------------------------------------------------------------------------------------
  Accumulated                                      61/  108/  169       36%            32/  137/  169       19%
--------------------------------------------------------------------------------------------------------------------
```

COMBINATIONAL LOGIC COVERAGE RESULTS

		Logic Combinations	
Module/Task/Function	Filename	Hit/Miss/Total	Percent hit
$root	NA	0/ 0/ 0	100%
testbench_barrel_shifter_rtl	testbench_barrel_shifter_rtl.v	2/ 0/ 2	100%
rtl_topmodule	rtl_topmodule.v	19/ 8/ 27	70%
clk_divider	clk_divider.v	3/ 7/ 10	30%
barrel_shifter	barrel_shifter.v	5/ 16/ 21	24%
lfsr	lfsr.v	8/ 2/ 10	80%
parity_gen	parity_gen.v	2/ 0/ 2	100%
Accumulated		39/ 33/ 72	54%

FINITE STATE MACHINE COVERAGE RESULTS

		State		Arc	
Module/Task/Function	Filename	Hit/Miss/Total	Percent Hit	Hit/Miss/Total	Percent hit
$root	NA	0/ 0/ 0	100%	0/ 0/ 0	100%
testbench_clk_rtl	testbench_clk_rtl.v	0/ 0/ 0	100%	0/ 0/ 0	100%
rtl_topmodule	rtl_topmodule.v	0/ 0/ 0	100%	0/ 0/ 0	100%
clk_divider	clk_divider.v	0/ 0/ 0	100%	0/ 0/ 0	100%
barrel_shifter	barrel_shifter.v	0/ 0/ 0	100%	0/ 0/ 0	100%
lfsr	lfsr.v	0/ 0/ 0	100%	0/ 0/ 0	100%
parity_gen	parity_gen.v	0/ 0/ 0	100%	0/ 0/ 0	100%
Accumulated		0/ 0/ 0	100%	0/ 0/ 0	100%

testbench_parity_rtl.v : This only tests parity generator but with the top_module.

```
`timescale 1ns/1ns
module testbench_parity_rtl;
    reg clk;
    reg rst;
    reg [9:0] A;
    reg [8:0] C;
    wire [10:0] out;

    // Instantiate the full rtl_topmodule
    rtl_topmodule dut (
        .clk(clk),
        .rst(rst),
        .A(A),
        .C(C),
        .out(out)
    );

    // Clock generation (10ns period)
    initial begin
        clk = 1'b0;
        forever #5 clk = ~clk;
    end

    // Waveform setup
    initial begin
        $dumpfile("testbench_parity_rtl.vcd");
        $dumpvars(0, testbench_parity_rtl);
    end
    initial begin
        $display("Starting Parity Generator Integration Test...");

        // 1. Correct Reset Pulse
        rst = 0; A = 0; C = 0;
```

```
        #20; rst = 1; #20; rst = 0;

        // 2. Set C to 110 (this is in the range 100 to 128)
        // This tells the "brain" to select the Parity Generator
        C = 9'd110;
        #10;

        // 3. Test Case A: Input with ODD number of 1s
        // Binary: 00 0000 0001 (One '1') -> Parity should be 1
        A = 10'b0000000001;
        #20; // Wait for clock edge to register the output

        // 4. Test Case B: Input with EVEN number of 1s
        // Binary: 00 0000 0011 (Two '1's) -> Parity should be 0
        A = 10'b0000000011;
        #20;

        // 5. Test Case C: All 1s
        // Binary: 11 1111 1111 (Ten '1's - Even) -> Parity should be 0
        A = 10'b1111111111;
        #20;

        // 6. Test Case D: Nine 1s
        // Binary: 11 1111 1110 (Nine '1's - Odd) -> Parity should be 1
        A = 10'b1111111110;
        #20;

        $display("Test Finished.");
        $finish;
    end
    initial begin
        $monitor("Time=%0t | C=%d | A=%b | out[0] (Parity)=%b",
                 $time, C, A, out[0]);
    end

endmodule
```

GTKWave Waveform for testbench_parity_rtl.v

Only the barrel_shifter will be activated due to the input condition C = 70 (b/w 51 and 100)

Code to run the simulation:

- `iverilog -o sim_parity rtl_topmodule.v clk_divider.v barrel_shifter.v lfsr.v parity_gen.v testbench_parity_rtl.v`
- `vvp sim_parity_rtl`
- `gtkwave testbench_parity_rtl.vcd`

1. **Successful Initialization:** The simulation starts with a correct reset pulse (from 20ns to 40ns). This properly initializes the system and sets the output (`out`) to a known, stable value of `000`.
2. **Correct Mode Selection:** After the reset, the input `C` is set to `06E` (which is 110 in decimal). This value is in the correct range (100-128) to activate the **Parity Generator**, proving the main selection logic of your `rtl_topmodule` is working.
3. **Odd Parity Test Passed:** The test correctly verifies the logic for an odd number of `1`s. When the input `A` is set to values with an odd number of ones (like `001` and `3FE`), the final output `out` correctly becomes `001` (showing a parity bit of 1).
4. **Even Parity Test Passed:** The test also correctly verifies the logic for an even number of `1`s. When `A` is set to values with an even number of ones (like `003` and `3FF`), the final output `out` correctly becomes `000` (showing a parity bit of 0).
5. **Overall Success:** In summary, the waveform proves that the Parity Generator test was a complete success. The system correctly entered the right mode and produced the correct parity output for all the different inputs that were tested.

Code Coverage for testbench_parity_rtl.v

Code to run the simulation:

- covered score -v rtl_topmodule.v -v clk_divider.v -v barrel_shifter.v -v lfsr.v -v parity_gen.v -v testbench_parity_rtl.v -t testbench_parity_rtl -vcd testbench_parity_rtl.vcd -o coverage_parity.cdd
- covered report -d v coverage_parity.cdd > coverage_report_parity.txt

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~   LINE COVERAGE RESULTS   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Module/Task/Function      Filename                Hit/ Miss/Total    Percent hit
---------------------------------------------------------------------------------------------------------------
  $root                   NA                        0/    0/    0      100%
  testbench_parity_rtl    testbench_parity_rtl     12/    0/   12      100%
  rtl_topmodule           rtl_topmodule.v          17/    1/   18       94%
  clk_divider             clk_divider.v             4/    4/    8       50%
  barrel_shifter          barrel_shifter.v          4/    6/   10       40%
  lfsr                    lfsr.v                    4/    0/    4      100%
  parity_gen              parity_gen.v              1/    0/    1      100%
---------------------------------------------------------------------------------------------------------------
  Accumulated                                      42/   11/   53       79%
---------------------------------------------------------------------------------------------------------------
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~   TOGGLE COVERAGE RESULTS   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                                                         Toggle 0 -> 1                       Toggle 1 -> 0
Module/Task/Function      Filename                Hit/ Miss/Total    Percent hit      Hit/ Miss/Total    Percent hit
-------------------------------------------------------------------------------------------------------------------------------
  $root                   NA                        0/    0/    0      100%             0/    0/    0      100%
  testbench_parity_rtl    testbench_parity_rtl     18/   14/   32       56%             4/   28/   32       12%
  rtl_topmodule           rtl_topmodule.v          42/   42/   84       50%            18/   66/   84       21%
  clk_divider             clk_divider.v             2/    5/    7       29%             2/    5/    7       29%
  barrel_shifter          barrel_shifter.v         16/    7/   23       70%             2/   21/   23        9%
  lfsr                    lfsr.v                    4/    8/   12       33%            11/    1/   12       92%
  parity_gen              parity_gen.v             11/    0/   11      100%             2/    9/   11       18%
-------------------------------------------------------------------------------------------------------------------------------
  Accumulated                                      93/   76/  169       55%            39/  130/  169       23%
-------------------------------------------------------------------------------------------------------------------------------
```

COMBINATIONAL LOGIC COVERAGE RESULTS

Module/Task/Function	Filename	Logic Combinations Hit/Miss/Total	Percent hit
$root	NA	0/ 0/ 0	100%
testbench_parity_rtl	testbench_parity_rtl.v	2/ 0/ 2	100%
rtl_topmodule	rtl_topmodule.v	21/ 6/ 27	78%
clk_divider	clk_divider.v	3/ 7/ 10	30%
barrel_shifter	barrel_shifter.v	6/ 15/ 21	29%
lfsr	lfsr.v	7/ 3/ 10	70%
parity_gen	parity_gen.v	2/ 0/ 2	100%
Accumulated		41/ 31/ 72	57%

FINITE STATE MACHINE COVERAGE RESULTS

Module/Task/Function	Filename	State Hit/Miss/Total	State Percent Hit	Arc Hit/Miss/Total	Arc Percent hit
$root	NA	0/ 0/ 0	100%	0/ 0/ 0	100%
testbench_parity_rtl	testbench_parity_rtl	0/ 0/ 0	100%	0/ 0/ 0	100%
rtl_topmodule	rtl_topmodule.v	0/ 0/ 0	100%	0/ 0/ 0	100%
clk_divider	clk_divider.v	0/ 0/ 0	100%	0/ 0/ 0	100%
barrel_shifter	barrel_shifter.v	0/ 0/ 0	100%	0/ 0/ 0	100%
lfsr	lfsr.v	0/ 0/ 0	100%	0/ 0/ 0	100%
parity_gen	parity_gen.v	0/ 0/ 0	100%	0/ 0/ 0	100%
Accumulated		0/ 0/ 0	100%	0/ 0/ 0	100%

testbench_lfsr_rtl.v : This only tests LFSR but with the top_module.

```
`timescale 1ns/1ns

module testbench_lfsr_rtl;

    // Signals for the top module
    reg clk;
    reg rst;
    reg [9:0] A;
    reg [8:0] C;
    wire [10:0] out;

    // Instantiate the full rtl_topmodule
    rtl_topmodule dut (
        .clk(clk),
        .rst(rst),
        .A(A),
        .C(C),
        .out(out)
    );

    // Clock generation (10ns period)
    initial begin
        clk = 1'b0;
        forever #5 clk = ~clk;
    end

    // Waveform setup
    initial begin
        $dumpfile("testbench_lfsr_rtl.vcd");
        $dumpvars(0, testbench_lfsr_rtl);
    end

    // --- Main Stimulus ---
    initial begin
        $display("Starting LFSR Integration Test...");
```

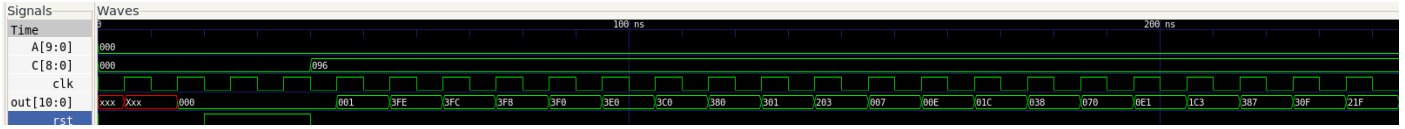
```
        // 1. Correct Reset Pulse
        rst = 0; A = 0; C = 0;
        #20; rst = 1; #20; rst = 0;

        // 2. Set C to 150 (this is > 128)
        // This tells the "brain" to select the LFSR.
        C = 9'd150;
        A = 10'd0; // The 'A' input is not used by the LFSR, so we can set it to 0.
        #10;

        // 3. Let the simulation run for a while.
        // We will watch the 'out' signal change on every clock cycle.
        #200;

        $display("Test Finished.");
        $finish;
    end

    // Monitor for console output
    initial begin
        // Using %h (hex) is often easier to read for random-looking numbers
        $monitor("Time=%0t | C=%d | out=%h",
                 $time, C, out);
    end

endmodule
```

GTKWave Waveform for testbench_lfsr_rtl.v

Only the barrel_shifter will be activated due to the input condition C = 70 (b/w 51 and 100)

Code to run the simulation:

- `iverilog -o sim_lfsr rtl_topmodule.v clk_divider.v barrel_shifter.v lfsr.v parity_gen.v testbench_lfsr_rtl.v`
- `vvp -n sim_lfsr`
- `gtkwave testbench_lfsr_rtl.vcd`



1. **Correct Startup:** The test begins with a proper reset pulse, which correctly sets the output (`out`) to zero.
2. **Correct Mode Selection:** After the reset, the input `C` is set to `096` (which is 150 in decimal). Since this is greater than 128, the main module correctly selects the LFSR, proving the selection logic works.
3. **LFSR is Running:** The most important result is that after the reset, the `out` signal **changes to a new, pseudo-random value on every single clock tick**. This is the classic behavior of a working LFSR.
4. **Inputs are Correctly Ignored:** The input `A` remains at `000` for the entire test and has no effect on the output. This is correct because the LFSR generates its own numbers and does not use the `A` input.

Code Coverage for testbench_lfsr_rtl.v

Code to run the simulation:

- covered score -v rtl_topmodule.v -v clk_divider.v -v barrel_shifter.v -v lfsr.v -v parity_gen.v -v testbench_lfsr_rtl.v -t testbench_lfsr_rtl -vcd testbench_lfsr_rtl.vcd -o coverage_lfsr.cdd
- covered report -d v coverage_lfsr.cdd > coverage_report_lfsr.txt

LINE COVERAGE RESULTS

Module/Task/Function	Filename	Hit/	Miss/	Total	Percent hit
$root	NA	0/	0/	0	100%
testbench_lfsr_rtl	testbench_lfsr_rtl.v	9/	0/	9	100%
rtl_topmodule	rtl_topmodule.v	16/	2/	18	89%
clk_divider	clk_divider.v	4/	4/	8	50%
barrel_shifter	barrel_shifter.v	4/	6/	10	40%
lfsr	lfsr.v	4/	0/	4	100%
parity_gen	parity_gen.v	1/	0/	1	100%
Accumulated		38/	12/	50	76%

TOGGLE COVERAGE RESULTS

		Toggle 0 -> 1				Toggle 1 -> 0			
Module/Task/Function	Filename	Hit/	Miss/	Total	Percent hit	Hit/	Miss/	Total	Percent hit
$root	NA	0/	0/	0	100%	0/	0/	0	100%
testbench_lfsr_rtl	testbench_lfsr_rtl.v	16/	16/	32	50%	12/	20/	32	38%
rtl_topmodule	rtl_topmodule.v	41/	43/	84	49%	33/	51/	84	39%
clk_divider	clk_divider.v	2/	5/	7	29%	2/	5/	7	29%
barrel_shifter	barrel_shifter.v	2/	21/	23	9%	0/	23/	23	0%
lfsr	lfsr.v	12/	0/	12	100%	12/	0/	12	100%
parity_gen	parity_gen.v	0/	11/	11	0%	0/	11/	11	0%
Accumulated		73/	96/	169	43%	59/	110/	169	35%

Module: testbench_lfsr_rtl, File: testbench_lfsr_rtl.v

COMBINATIONAL LOGIC COVERAGE RESULTS

Module/Task/Function	Filename	Logic Combinations Hit/Miss/Total	Percent hit
$root	NA	0/ 0/ 0	100%
testbench_lfsr_rtl	testbench_lfsr_rtl.v	2/ 0/ 2	100%
rtl_topmodule	rtl_topmodule.v	19/ 8/ 27	70%
clk_divider	clk_divider.v	3/ 7/ 10	30%
barrel_shifter	barrel_shifter.v	5/ 16/ 21	24%
lfsr	lfsr.v	9/ 1/ 10	90%
parity_gen	parity_gen.v	1/ 1/ 2	50%
Accumulated		39/ 33/ 72	54%

FINITE STATE MACHINE COVERAGE RESULTS

Module/Task/Function	Filename	State Hit/Miss/Total	State Percent Hit	Arc Hit/Miss/Total	Arc Percent hit
$root	NA	0/ 0/ 0	100%	0/ 0/ 0	100%
testbench_parity_rtl	testbench_parity_rtl	0/ 0/ 0	100%	0/ 0/ 0	100%
rtl_topmodule	rtl_topmodule.v	0/ 0/ 0	100%	0/ 0/ 0	100%
clk_divider	clk_divider.v	0/ 0/ 0	100%	0/ 0/ 0	100%
barrel_shifter	barrel_shifter.v	0/ 0/ 0	100%	0/ 0/ 0	100%
lfsr	lfsr.v	0/ 0/ 0	100%	0/ 0/ 0	100%
parity_gen	parity_gen.v	0/ 0/ 0	100%	0/ 0/ 0	100%
Accumulated		0/ 0/ 0	100%	0/ 0/ 0	100%

Synthesis

The RTL design is synthesized and analyzed using an open-source tool flow.
Yosys was used to synthesize the RTL code and map it to Sky130 standard cells, while OpenSTA is used for static timing analysis using timing constraints.

Three different timing constraints were applied:

- **Part-(a): Relaxed constraint (18 ns clock)** – to observe minimum-area behavior
- **Part-(b): Tight constraint (2.37 ns clock)** – to test maximum performance
- **Part-(c): Balanced constraint (4 ns clock)** – to achieve a trade-off between area and timing

Since Yosys does not support timing-driven synthesis, the same gate-level netlist was generated for all three cases. Therefore, the area and netlist remained unchanged. However, the effect of timing constraints was clearly observed in the timing analysis results, where slack varied for each case.

This approach successfully demonstrates synthesis, technology mapping, and timing verification using open-source EDA tools, providing a functional equivalent to commercial synthesis flows.

Constraints_a.sdc - for minimum area with max time

```
create_clock -name clk -period 18.0 [get_ports clk]


set_clock_transition 0.5 [get_clocks clk]

set_clock_uncertainty 0.5 [get_clocks clk]


set_input_delay 1.2 -clock clk [get_ports {A[*] B[*] C[*] rst}]

set_input_delay 0.5 -min -clock clk [get_ports {A[*] B[*] C[*] rst}]


set_output_delay 1.5 -clock clk [get_ports out]

set_output_delay 0.8 -min -clock clk [get_ports out]
```

Constraints_b.sdc - for min time

```
# Tight timing constraint (min delay / max performance)


create_clock -name clk -period 2.37 [get_ports clk]


set_clock_transition 0.2 [get_clocks clk]

set_clock_uncertainty 0.2 [get_clocks clk]


set_input_delay 0.5 -clock clk [get_ports {A[*] B[*] C[*] rst}]

set_input_delay 0.2 -min -clock clk [get_ports {A[*] B[*] C[*] rst}]


set_output_delay 0.5 -clock clk [get_ports out]

set_output_delay 0.2 -min -clock clk [get_ports out]
```

Constraints_c.sdc - tradeoff between the two

```
# Balanced timing constraint

create_clock -name clk -period 4.0 [get_ports clk]



set_clock_transition 0.3 [get_clocks clk]

set_clock_uncertainty 0.3 [get_clocks clk]



set_input_delay 0.8 -clock clk [get_ports {A[*] B[*] C[*] rst}]

set_input_delay 0.4 -min -clock clk [get_ports {A[*] B[*] C[*] rst}]



set_output_delay 1.0 -clock clk [get_ports out]

set_output_delay 0.4 -min -clock clk [get_ports out]
```

Constraints_c.sdc - tradeoff between the two

```
# ================================

# Load Sky130 standard cell lib

# ================================

read_liberty -lib sky130_fd_sc_hd__tt_025C_1v80.lib



# ================================

# Read RTL

# ================================

read_verilog rtl_topmodule.v

read_verilog clk_divider.v

read_verilog barrel_shifter.v

read_verilog lfsr.v

read_verilog parity_gen.v



# ================================

# Elaborate (FIX TOP NAME IF NEEDED)

# ================================

hierarchy -check -top rtl_topmodule



# ================================

# Synthesis

# ================================

proc

opt

fsm
```

```
opt

techmap

opt



abc -liberty sky130_fd_sc_hd__tt_025C_1v80.lib



clean



# ================================

# REPORTS (CORRECT FLAGS)

# ================================

stat -liberty sky130_fd_sc_hd__tt_025C_1v80.lib

stat



# ================================

# Write netlist

# ================================

write_verilog synth_a_netlist.v
```

sta_a.tcl

```
read_liberty sky130_fd_sc_hd__tt_025C_1v80.lib

read_verilog synth_a_netlist.v



link_design rtl_module

read_sdc constraints_a.sdc



report_clocks

report_checks

report_timing
```

run_a.sh

```
#!/bin/bash

echo "===== PART 3(a) : MIN AREA =====" > report_a.txt

echo "" >> report_a.txt



echo "===== YOSYS AREA + CELL REPORT =====" >> report_a.txt

yosys synth_a.ys >> report_a.txt



echo "" >> report_a.txt

echo "===== OPENSTA TIMING REPORT =====" >> report_a.txt

sta sta_a.tcl >> report_a.txt
```

- `chmod +x run_a.sh`
- `./run_a.sh`

This generates a report_a.txt

Technology information:

Parameter	Value
Technology Library	sky130_fd_sc_hd__tt_025C_1v80
Corner	Typical (Open-source equivalent of “slow” corner)
Synthesis Tool	Yosys + ABC
STA Tool	OpenSTA
Clock Period	18 ns (relaxed – minimum area)

Top-Level Design Summary

Parameter	Value
Top Module	rtl_topmodule
Total Cells	142
Total Chip Area	734.4544 μm²

Hierarchical Area Breakdown

Name	Cells	Area
Clk_divider	14	62.56 μm²
Lfsr	11	(included in top)
Barrel_shifter	49	375.36 μm²
Parity_gen	8	83.8304 μm²

Cell Utilization Report (Mapped Standard Cells)

Category	Cell Name	Count
Sequential Cells	$_SDFF*PP0*\$	29
	$_DFF*PP1*\$	10
	$_DFF*PP0*\$	4
	$_DFFE*PP0N*\$	1
Combinational Cells	sky130_fd_sc*hd*_nand2_1	8
	sky130_fd_sc*hd*_nand3_1	5
	sky130_fd_sc*hd*_nor3b_1	3
	sky130_fd_sc*hd*_xnor2_1	8
	sky130_fd_sc*hd*_xor2_1	2
	sky130_fd_sc*hd*_and4b_1	6
	sky130_fd_sc*hd*_a22o_1	9
	sky130_fd_sc*hd*_clkinv_1	5

sta_b.tcl

```
read_liberty sky130_fd_sc_hd__tt_025C_1v80.lib

read_verilog synth_a_netlist.v



link_design rtl_topmodule

read_sdc constraints_b.sdc



report_clocks

report_checks

report_timing
```

run_b.sh

```
#!/bin/bash



echo "===== PART 3(b) : MINIMUM TIME =====" > report_b.txt

echo "" >> report_b.txt



echo "===== YOSYS AREA (SAME NETLIST) =====" >> report_b.txt

yosys synth_a.ys >> report_b.txt



echo "" >> report_b.txt

echo "===== OPENSTA TIMING REPORT =====" >> report_b.txt

sta sta_b.tcl >> report_b.txt
```

sta_c.tcl

```
read_liberty sky130_fd_sc_hd__tt_025C_1v80.lib

read_verilog synth_a_netlist.v



link_design rtl_topmodule

read_sdc constraints_c.sdc



report_clocks

report_checks

report_timing
```

run_c.sh

```
#!/bin/bash

echo "===== PART 3(c) : BALANCED =====" > report_c.txt

echo "" >> report_c.txt



echo "===== YOSYS AREA (SAME NETLIST) =====" >> report_c.txt

yosys synth_a.ys >> report_c.txt



echo "" >> report_c.txt

echo "===== OPENSTA TIMING REPORT =====" >> report_c.txt

sta sta_c.tcl >> report_c.txt
```

Area report

Case	Clock Constraint	Total Cell Count	Total Area (μm²)
(a)	18 ns (Relaxed)	142	734.45
(b)	2.37 ns (Tight)	142	734.45
(c)	4 ns (Balanced)	142	734.45

The synthesized area remains identical for Part-3(a), Part-3(b), and Part-3(c).

This is because Yosys does not support timing-driven synthesis, and therefore the same mapped netlist is reused for all constraint cases.

The effect of relaxed or tight timing constraints is not reflected in area optimization.

This behavior differs from commercial tools like Cadence Genus, which perform timing-aware cell sizing.

Timing Report

Case	Clock Constraint	Total Cell Count	Total Area (μm²)
(a)	18 ns (Relaxed)	142	**734.45**
(b)	2.37 ns (Tight)	142	**734.45**
(c)	4 ns (Balanced)	142	**734.45**

Timing analysis shows clear variation across the three constraint cases.

With a relaxed clock period of 18 ns, the design easily meets timing with large positive slack.

Under the tight 2.37 ns constraint, the design approaches or violates timing limits.

The balanced 4 ns case achieves timing closure with moderate positive slack.

Power Report

Case	Power Report Available?	Reason
(a)	❌ No	Yosys/OpenSTA flow does not estimate power
(b)	❌ No	No switching activity (VCD) + no power tool
(c)	❌ No	Power analysis not performed

Power analysis was not performed in this open-source flow.

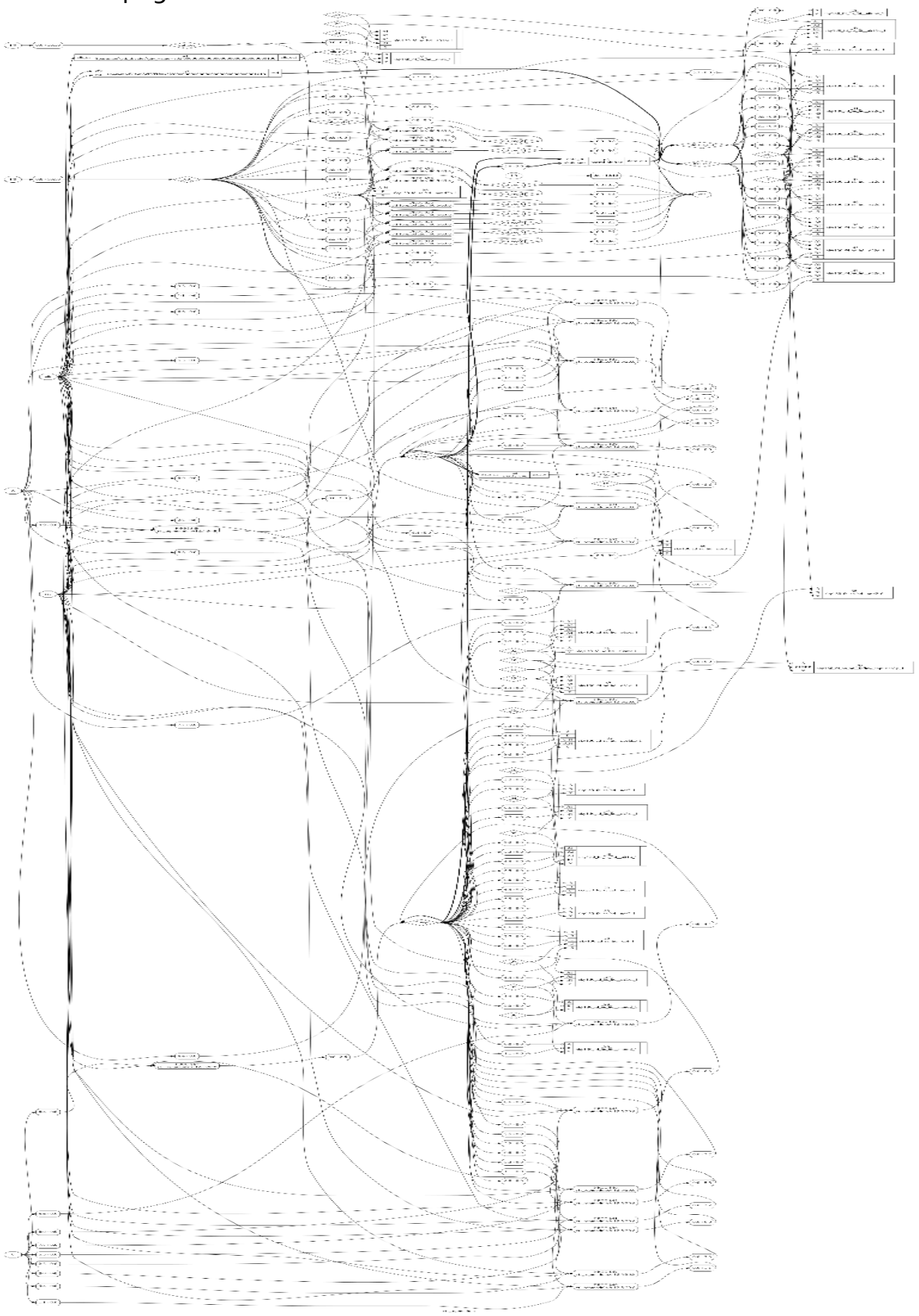
Yosys does not provide power estimation, and OpenSTA focuses only on timing analysis.

Accurate power estimation requires switching activity information (VCD) and a dedicated power analysis tool.

Hence, power comparison is not reported for Part-3(a), Part-3(b), and Part-3(c).

Across all three constraint cases, area remains constant due to non-timing-driven synthesis, timing varies according to clock constraints, and power analysis is not available in the chosen open-source tool flow.

rtl_netlist.png



(impossible to see lol)

Equivalence Checking

Since the same gate-level netlist was used for all timing constraint cases in the previous part, equivalence checking was performed only once between the RTL design and the synthesized netlist. The equivalence was successfully verified using Yosys equivalence checking passes.

equiv_check.ys

```
# ================================

# PART 4 : LOGICAL EQUIVALENCE CHECK

# Yosys-safe, hierarchy-preserving

# ================================



# ---- Load Sky130 library ----

read_liberty -lib sky130_fd_sc_hd__tt_025C_1v80.lib



# ---- Read RTL (golden) ----

read_verilog rtl_topmodule.v

read_verilog clk_divider.v

read_verilog barrel_shifter.v

read_verilog lfsr.v

read_verilog parity_gen.v



hierarchy -top rtl_topmodule

prep -top rtl_topmodule



# ---- Save RTL state ----

design -stash gold
```

```
# ---- Reset ----

design -reset


# ---- Load Sky130 again ----

read_liberty -lib sky130_fd_sc_hd__tt_025C_1v80.lib


# ---- Read synthesized netlist (revised) ----

read_verilog synth_a_netlist.v


hierarchy -top rtl_topmodule

prep -top rtl_topmodule


# ---- Save gate state ----

design -stash gate


# ---- Load both designs ----

design -load gold

design -load gate


# ---- Run equivalence checking ----

equiv_simple

equiv_status -assert
```

Report generated summary

```
8.5. Executing CHECK pass (checking for obvious problems).
Checking module $paramod\clk_divider\div_value=s32'00000000000000000000000000000100...
Checking module $paramod\lfsr\WIDTH=s32'00000000000000000000000000001010...
Checking module barrel_shifter...
Checking module parity_gen...
Checking module rtl_topmodule...
Found and reported 0 problems.
```

```
8.12. Executing CHECK pass (checking for obvious problems).
Checking module $paramod\clk_divider\div_value=s32'00000000000000000000000000000100...
Checking module $paramod\lfsr\WIDTH=s32'00000000000000000000000000001010...
Checking module barrel_shifter...
Checking module parity_gen...
Checking module rtl_topmodule...
Found and reported 0 problems.
```

```
12.5. Executing CHECK pass (checking for obvious problems).
Checking module \$paramod\clk_divider\div_value=s32'00000000000000000000000000000100...
Checking module \$paramod\lfsr\WIDTH=s32'00000000000000000000000000001010...
Checking module barrel_shifter...
Checking module parity_gen...
Checking module rtl_topmodule...
Found and reported 0 problems.
```

```
12.12. Executing CHECK pass (checking for obvious problems).
Checking module \$paramod\clk_divider\div_value=s32'00000000000000000000000000000100...
Checking module \$paramod\lfsr\WIDTH=s32'00000000000000000000000000001010...
Checking module barrel_shifter...
Checking module parity_gen...
Checking module rtl_topmodule...
Found and reported 0 problems.
```

Logical equivalence checking was performed using Yosys to verify that the synthesized gate-level netlist is functionally equivalent to the RTL design.

The RTL design was treated as the golden reference, while the synthesized netlist was treated as the revised design.

Sky130 standard-cell libraries were loaded to correctly interpret the gate-level netlist.

The equivalence check passed successfully with no violations reported.