

COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Department of Electronics

Internship Project Report

on

Hardware-Based True Random Number Generator using Ring Oscillators on FPGA

Under the Guidance of

Dr. Tripti S Warriar, Ms. Simi Sukumaran, Mr. Adarsh K,

Submitted by

Sreenesh K.S

u2201199@rajagiri.edu.in

Rajagiri School of Engineering & Technology (RSET)

Cochin, Kerala

June 1, 2025

Abstract

This project implements a True Random Number Generator (TRNG) on a Xilinx Spartan-7 FPGA using multiple ring oscillators. The oscillators exploit jitter caused by physical noise sources as entropy. Von Neumann debiasing removes bias from the raw output to improve randomness. The system outputs 32-bit random numbers, displayed through LEDs, and is tested for reliable performance suitable for cryptographic applications.

Contents

1	Introduction	3
2	System Design	3
3	Key Components	3
3.1	Ring Oscillators	3
3.2	Von Neumann Debiasing	3
4	Implementation	3
5	32-bit Implementation of TRNG	4
6	Simulation Waveform Proof for 32 bit	4
7	Supporting Proof for 32 bit TRNG	5
8	8-bit Implementation of TRNG	12
9	Simulation Waveform Proof for 8 bit	12
10	Supporting Proof for 8 bit TRNG	13
11	Comparison of 8-bit and 32-bit TRNG	18
12	Presentation Slides	19
13	Conclusion and Results	23

1 Introduction

Random numbers are essential for secure communications and cryptography. This project builds a hardware-based TRNG using ring oscillators to generate entropy from physical noise and applies Von Neumann debiasing for improved randomness.

2 System Design

- 32 ring oscillators generate jitter-based signals.
- Outputs are sampled at 24 MHz using flip-flops.
- A XOR tree combines the oscillator outputs into a single entropy bit.
- Von Neumann debiasing removes bias by processing bits in pairs.
- Final output is displayed on 8 LEDs for real-time observation.

3 Key Components

3.1 Ring Oscillators

Each oscillator consists of 4 LUTs arranged in a loop. Programmable delay lines reduce correlation by slightly adjusting delay, enhancing entropy.

3.2 Von Neumann Debiasing

Pairs of bits are processed:

- ‘01’ outputs 0.
- ‘10’ outputs 1.
- ‘00’ and ‘11’ are discarded.

This removes bias caused by hardware imperfections.

4 Implementation

- FPGA: Xilinx Spartan-7 (Boolean Board)
- Clock: 10ns
- LEDs used to visualize output

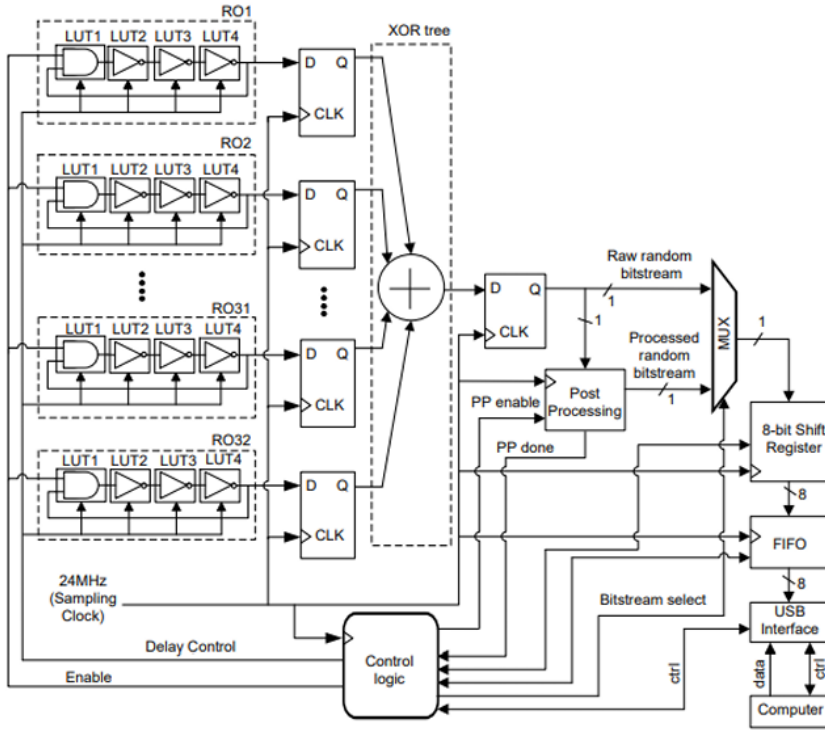


Figure 1: Overall System Block Diagram of TRNG

5 32-bit Implementation of TRNG

The evaluation results of the 32-bit TRNG implementation on the FPGA are summarized in the following table:

Parameter	Value
Clock Period (clk)	10 ns
Worst Negative Slack (WNS)	8.612 ns
Delay	1.38 ns
Number of LUTs	27
On-chip Power Consumption	0.086 W
Dynamic Power Consumption	0.003 W

Table 1: Evaluation Results of 32-bit TRNG Implementation

6 Simulation Waveform Proof for 32 bit

The following waveform illustrates the simulation results of the 32-bit TRNG output. It demonstrates the randomness and timing behavior verified through the simulation environment.

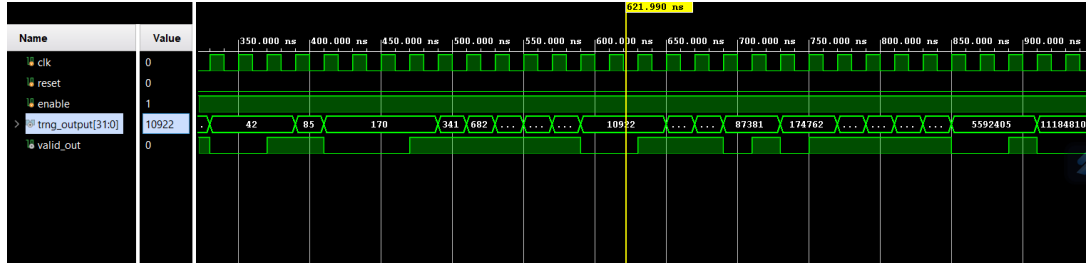


Figure 2: Simulation Waveform of 32-bit TRNG Output

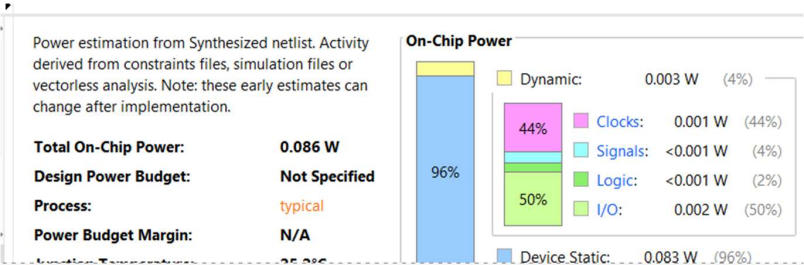
7 Supporting Proof for 32 bit TRNG

To further validate the implementation, hardware test results and real-time output observation were conducted. The following evidence supports the functionality and effectiveness of the TRNG system:

- Real-time LED output demonstrating non-repetitive random patterns.
- Photographs of the FPGA board setup during testing.
- Statistical test results confirming randomness quality.

Name	Slice LUTs (41000)	Slice Registers (82000)	F7 Muxes (20500)	Slice (10250)	LUT as Logic (41000)	Bonded IOB (300)	BUFGCTRL (32)
trng_7seg_top	27	59	4	20	27	18	1
display_mux (seven_seg_mux)	24	23	4	16	24	0	0
vn_inst (von_neumann)	3	4	0	1	3	0	0

LUT



POWER

Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement
Path 1	8.162	2	3	display_mux/refr...unter_reg[10]/C	display_mux/digit_index_reg[0]/D	1.695	0.487	1.208	10.000
Path 2	8.162	2	3	display_mux/refr...counter_reg[4]/C	display_mux/digit_index_reg[1]/D	1.695	0.487	1.208	10.000
Path 3	8.162	2	3	display_mux/refr...counter_reg[4]/C	display_mux/digit_index_reg[2]/D	1.695	0.487	1.208	10.000
Path 4	8.213	5	3	display_mux/refr...counter_reg[1]/C	display_mux/refr...unter_reg[17]/D	1.669	1.044	0.625	10.000
Path 5	8.244	5	3	display_mux/refr...counter_reg[1]/C	display_mux/refr...unter_reg[19]/D	1.638	1.013	0.625	10.000
Path 6	8.273	4	3	display_mux/refr...counter_reg[1]/C	display_mux/refr...unter_reg[13]/D	1.609	0.984	0.625	10.000
Path 7	8.288	5	3	display_mux/refr...counter_reg[1]/C	display_mux/refr...unter_req[18]/D	1.594	0.969	0.625	10.000

DELAY

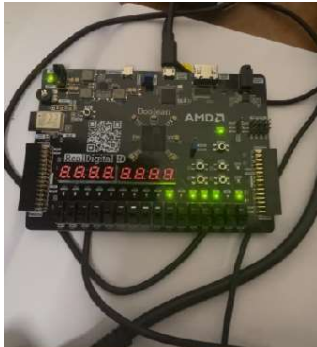
Design Timing Summary

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 8.162 ns	Worst Hold Slack (WHS): 0.063 ns	Worst Pulse Width Slack (WPWS): 4.650 ns
Total Negative Slack (TNS): 0.000 ns	Total Hold Slack (THS): 0.000 ns	Total Pulse Width Negative Slack (TPWS): 0.000 ns
Number of Failing Endpoints: 0	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0
Total Number of Endpoints: 90	Total Number of Endpoints: 90	Total Number of Endpoints: 60

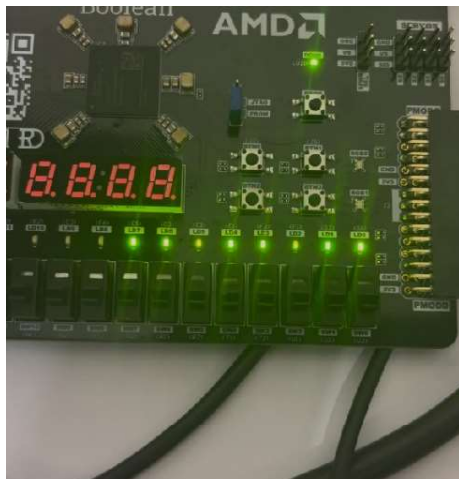
All user specified timing constraints are met.

WNS

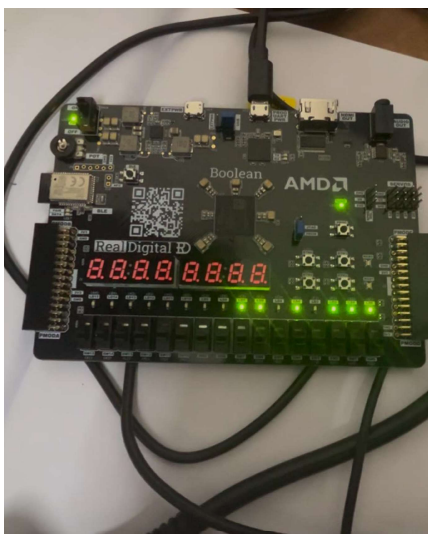
FPGA LED OUTPUT OF RANDOM NUMBERS [8BITS]



BINARY=1011101



BINARY=11111111



BINARY=11010111

Submodule

- Ring Oscillator Module

```
module ring_oscillator #(parameter ID = 0) (  
    input enable,  
    output reg ro_out  
);  
    initial ro_out = 0;  
  
    always begin  
        if (enable) begin  
            #((ID+1)*3) ro_out = ~ro_out; // Different toggle delay :  
        end else begin  
            #10; // do nothing when disabled  
        end  
    end  
end  
endmodule
```

Figure 3: Simulation: Ring Oscillator

- XOR Tree Module




```
20  //////////////////////////////////////  
21 |  
22  module xor_tree (  
23 |     input [31:0] ro_signals,  
24 |     output xor_out  
25 | );  
26 |     assign xor_out = ^ro_signals; // XOR reduction  
27  endmodule  
28 |
```

Figure 4: Simulation: XOR Tree

- Von Neumann Debiasing Module

```

module von_neumann (
    input clk,
    input in_bit,
    output reg out_bit,
    output reg valid
);
    reg [1:0] buffer;

    always @(posedge clk) begin
        buffer <= {buffer[0], in_bit};

        if (buffer == 2'b01) begin
            out_bit <= 1;
            valid <= 1;
        end else if (buffer == 2'b10) begin
            out_bit <= 0;
            valid <= 1;
        end else begin
            valid <= 0;
        end
    end
endmodule

```

Figure 5: Simulation: Von Neumann Debiasing

- Shift Register Output (Final Output Register)

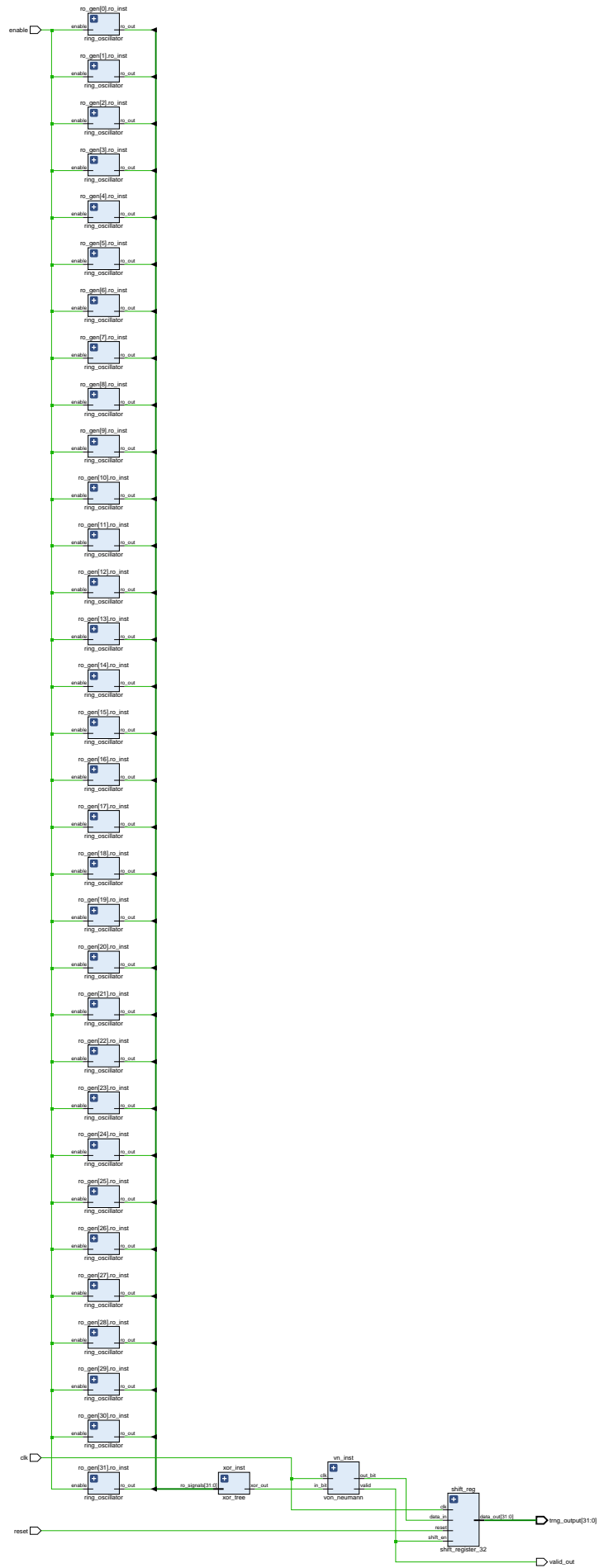
```

module shift_register_32 (
    input clk,
    input reset,
    input shift_en,
    input data_in,
    output reg [31:0] data_out
);
    always @(posedge clk or posedge reset) begin
        if (reset)
            data_out <= 32'b0;
        else if (shift_en)
            data_out <= {data_out[30:0], data_in};
    end
endmodule

```

Figure 6: Simulation: Final Output Shift

- Elaborated schematic



8 8-bit Implementation of TRNG

The evaluation results of the 8-bit TRNG implementation on the FPGA are summarized in the following table:

Parameter	Value
Clock Period (clk)	10 ns
Worst Negative Slack (WNS)	8.602 ns
Delay	1.247ns
Number of LUTs	10
On-chip Power Consumption	0.072 W
Dynamic Power Consumption	0.001 W

Table 2: Evaluation Results of 8-bit TRNG Implementation

9 Simulation Waveform Proof for 8 bit

The following waveform illustrates the simulation results of the 8-bit TRNG output. It demonstrates the randomness and timing behavior verified through the simulation environment.

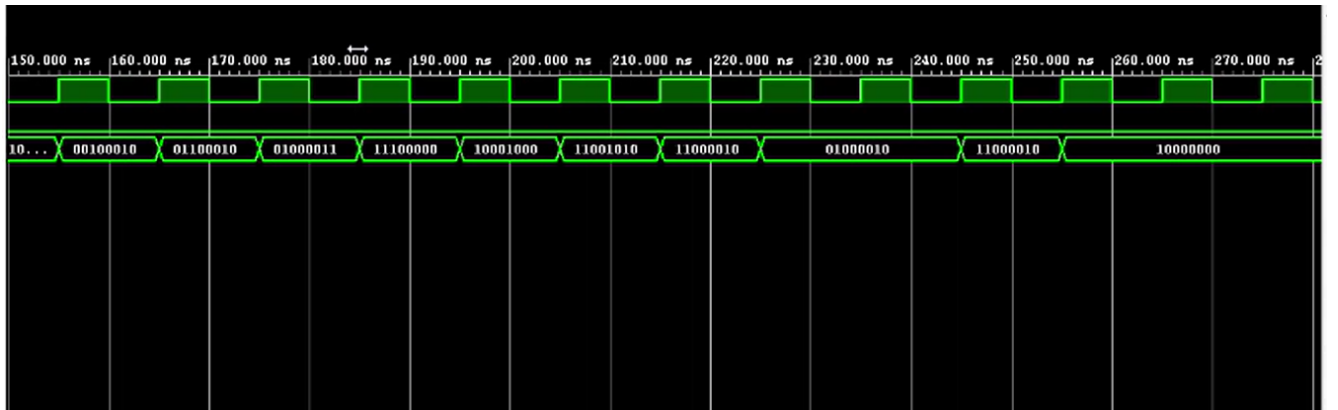


Figure 7: Simulation Waveform of 8-bit TRNG Output

10 Supporting Proof for 8 bit TRNG

LUT Usage

Tcl Console	Messages	Log	Reports	Design Runs	Power	Utilization	Timing
Hierarchy							
Hierarchy		Name		Slice LUTs (32600)	Slice Registers (65200)	Bonded IOB (210)	BUFGCTRL (32)
▼ trng_top_8bit				10	21	11	2
▼ Slice Logic							
▼ Slice LUTs (<1%)							
LUT as Logic (<1%)							
▼ Slice Registers (<1%)							
Register as Flip Flop (<1%)							
Register as Latch (<1%)							
Memory							
DSP							
▼ IO and GT Specific							
Bonded IOB (5%)							
▼ Clocking							
ro_gen[0].ro_inst (ring_oscillator)				1	1	0	0
ro_gen[1].ro_inst (ring_oscillator_parameterized0)				1	1	0	0
ro_gen[2].ro_inst (ring_oscillator_parameterized1)				1	1	0	0
ro_gen[3].ro_inst (ring_oscillator_parameterized2)				1	1	0	0
ro_gen[4].ro_inst (ring_oscillator_parameterized3)				1	1	0	0
ro_gen[5].ro_inst (ring_oscillator_parameterized4)				1	1	0	0
ro_gen[6].ro_inst (ring_oscillator_parameterized5)				1	1	0	0
ro_gen[7].ro_inst (ring_oscillator_parameterized6)				1	1	0	0
sr_inst (shift_register_8)				0	8	0	0
vn_inst (von_neumann)				2	5	0	0

Figure 8: LUT utilization report for the 8-bit TRNG

Power Report

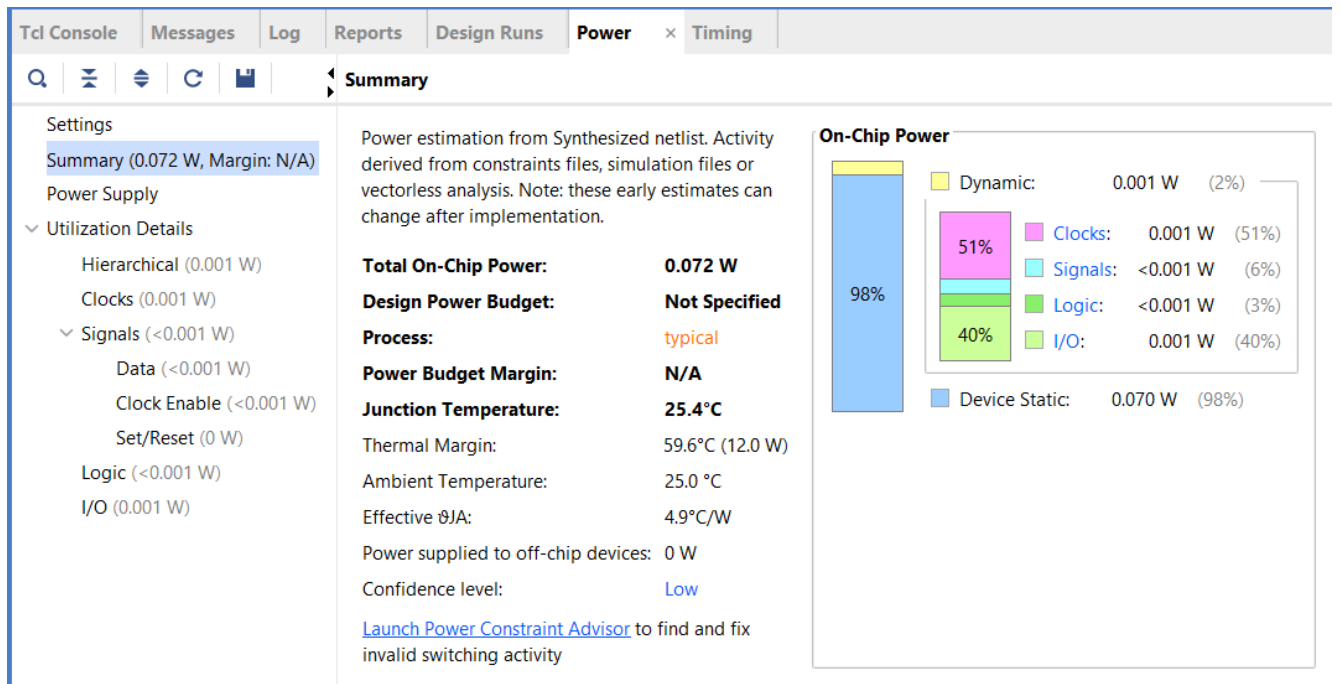


Figure 9: Power estimation for the 8-bit TRNG

Delay Analysis

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 1	8.602	1	2	3	vn_inst/toggle_reg/C	vn_inst/valid_reg/D	1.247	0.630	0.617	10.0	clk
Path 2	8.621	1	2	3	vn_inst/buffer_reg[0]/C	vn_inst/out_bit_reg/D	1.228	0.611	0.617	10.0	clk
Path 3	8.852	0	1	8	vn_inst/valid_reg/C	sr_inst/data_out_reg[0]/CE	0.676	0.379	0.297	10.0	clk
Path 4	8.852	0	1	8	vn_inst/valid_reg/C	sr_inst/data_out_reg[1]/CE	0.676	0.379	0.297	10.0	clk
Path 5	8.852	0	1	8	vn_inst/valid_reg/C	sr_inst/data_out_reg[2]/CE	0.676	0.379	0.297	10.0	clk
Path 6	8.852	0	1	8	vn_inst/valid_reg/C	sr_inst/data_out_reg[3]/CE	0.676	0.379	0.297	10.0	clk
Path 7	8.852	0	1	8	vn_inst/valid_reg/C	sr_inst/data_out_reg[4]/CE	0.676	0.379	0.297	10.0	clk
Path 8	8.852	0	1	8	vn_inst/valid_reg/C	sr_inst/data_out_reg[5]/CE	0.676	0.379	0.297	10.0	clk
Path 9	8.852	0	1	8	vn_inst/valid_reg/C	sr_inst/data_out_reg[6]/CE	0.676	0.379	0.297	10.0	clk
Path 10	8.852	0	1	8	vn_inst/valid_reg/C	sr_inst/data_out_reg[7]/CE	0.676	0.379	0.297	10.0	clk

Figure 10: Path delay analysis for the 8-bit TRNG

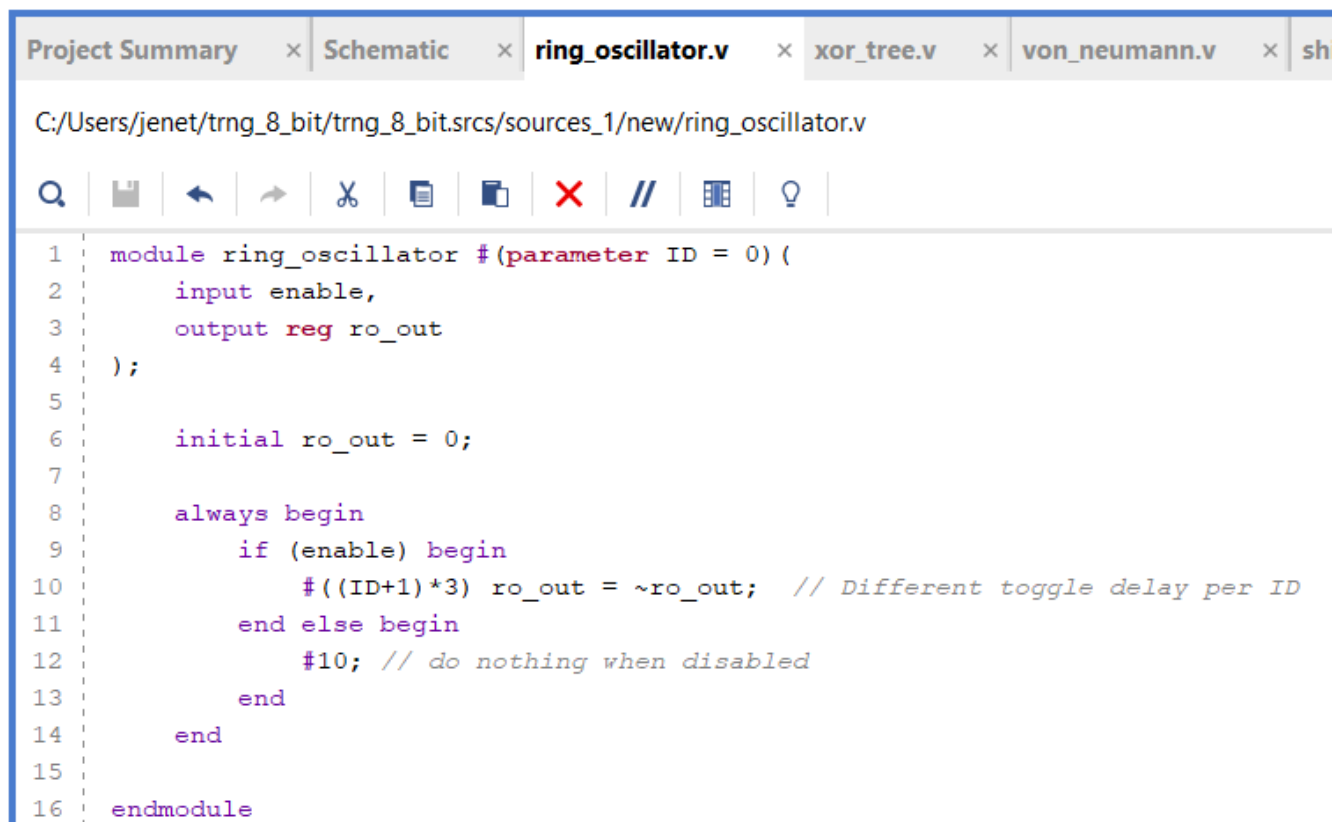
Worst Negative Slack (WNS)

Tcl Console Messages Log Reports Design Runs Power Timing			
Design Timing Summary			
General Information			
Timer Settings			
Design Timing Summary			
Clock Summary (1)			
Methodology Summary			
> Check Timing (33)			
> Intra-Clock Paths			
Inter-Clock Paths			
Other Path Groups			
User Ignored Paths			
> Unconstrained Paths			
Setup			
Worst Negative Slack (WNS): 8.602 ns			
Total Negative Slack (TNS): 0.000 ns			
Number of Failing Endpoints: 0			
Total Number of Endpoints: 20			
Hold			
Worst Hold Slack (WHS): 0.131 ns			
Total Hold Slack (THS): 0.000 ns			
Number of Failing Endpoints: 0			
Total Number of Endpoints: 20			
Pulse Width			
Worst Pulse Width Slack (WPWS): 4.500 ns			
Total Pulse Width Negative Slack (TPWS): 0.000 ns			
Number of Failing Endpoints: 0			
Total Number of Endpoints: 14			
All user specified timing constraints are met.			

Figure 11: Timing slack (WNS) report for the 8-bit TRNG

Submodule

- Ring Oscillator Module

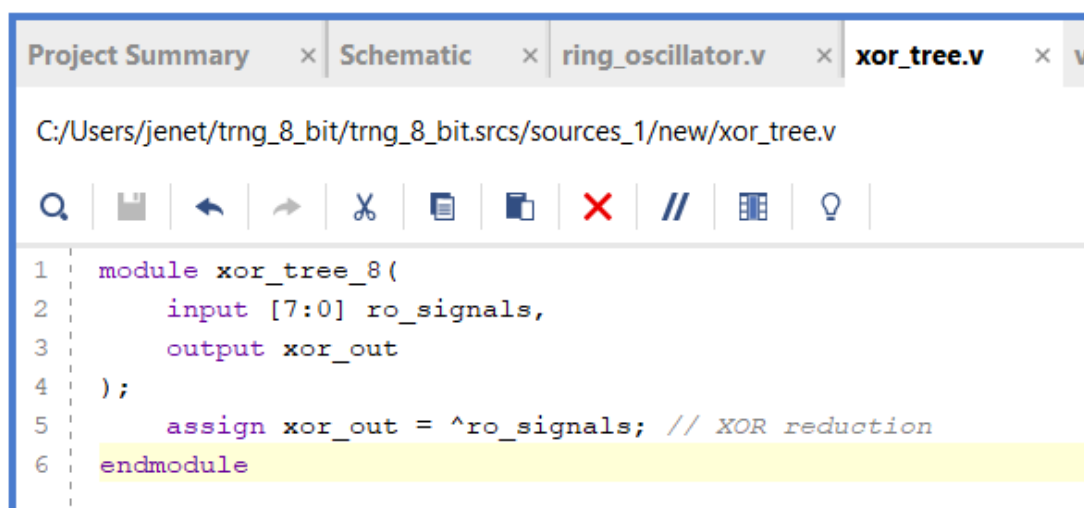


The screenshot shows a Verilog code editor with the following tabs: Project Summary, Schematic, ring_oscillator.v, xor_tree.v, von_neumann.v, and sh. The file path is C:/Users/jenet/trng_8_bit/trng_8_bit.srscs/sources_1/new/ring_oscillator.v. The code defines a module ring_oscillator with a parameter ID, an input enable, and an output reg ro_out. It includes an initial value of 0 for ro_out and an always block that toggles ro_out based on enable and ID.

```
1 module ring_oscillator #(parameter ID = 0) (  
2     input enable,  
3     output reg ro_out  
4 );  
5  
6     initial ro_out = 0;  
7  
8     always begin  
9         if (enable) begin  
10             #((ID+1)*3) ro_out = ~ro_out; // Different toggle delay per ID  
11         end else begin  
12             #10; // do nothing when disabled  
13         end  
14     end  
15  
16 endmodule
```

Figure 12: Simulation: Ring Oscillator

- XOR Tree Module

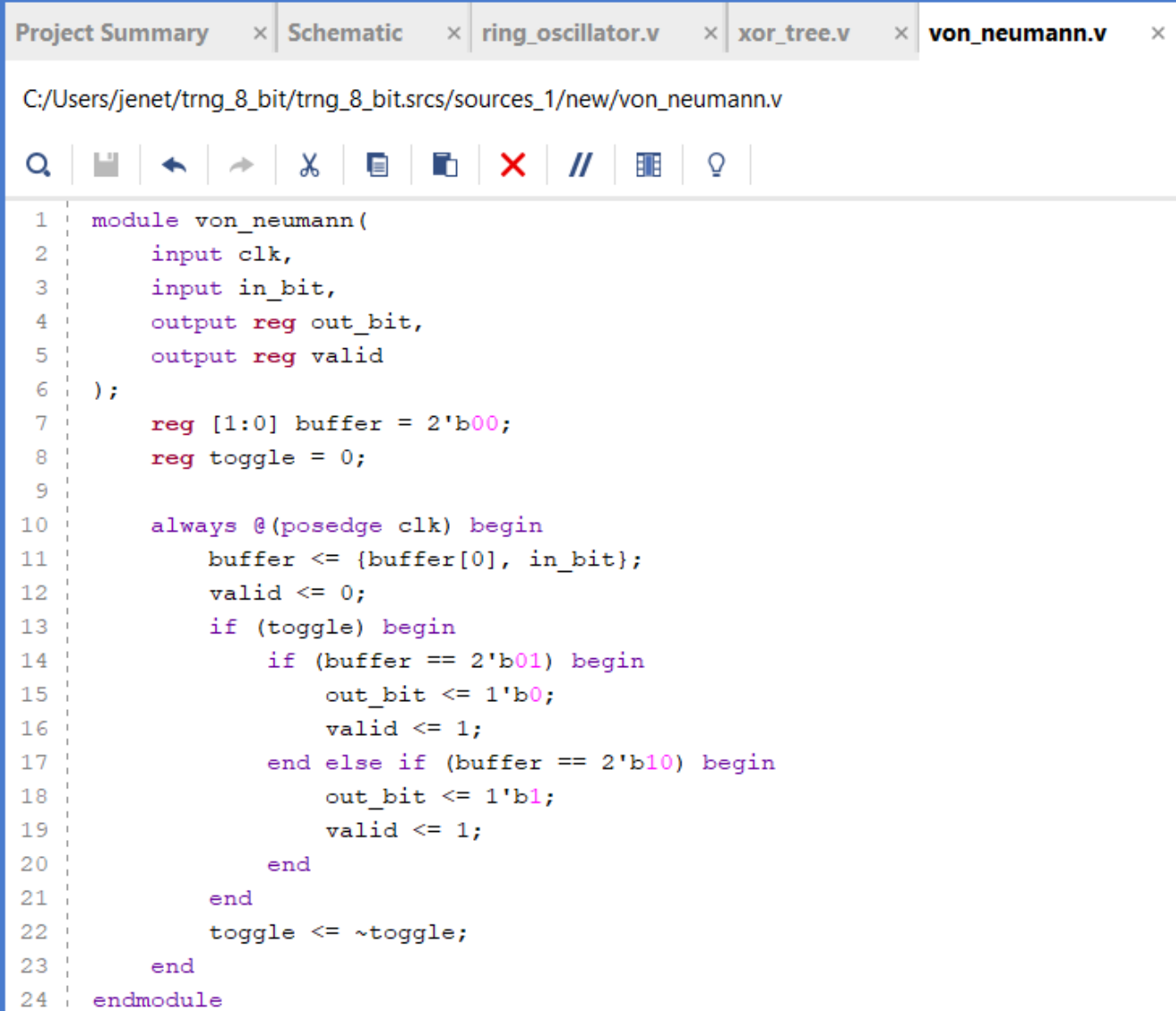


The screenshot shows a Verilog code editor with the following tabs: Project Summary, Schematic, ring_oscillator.v, xor_tree.v, and v. The file path is C:/Users/jenet/trng_8_bit/trng_8_bit.srscs/sources_1/new/xor_tree.v. The code defines a module xor_tree_8 with an input [7:0] ro_signals and an output xor_out. It includes an assign statement for xor_out and an endmodule statement.

```
1 module xor_tree_8(  
2     input [7:0] ro_signals,  
3     output xor_out  
4 );  
5     assign xor_out = ^ro_signals; // XOR reduction  
6 endmodule
```

Figure 13: Simulation: XOR Tree

- Von Neumann Debiasing Module

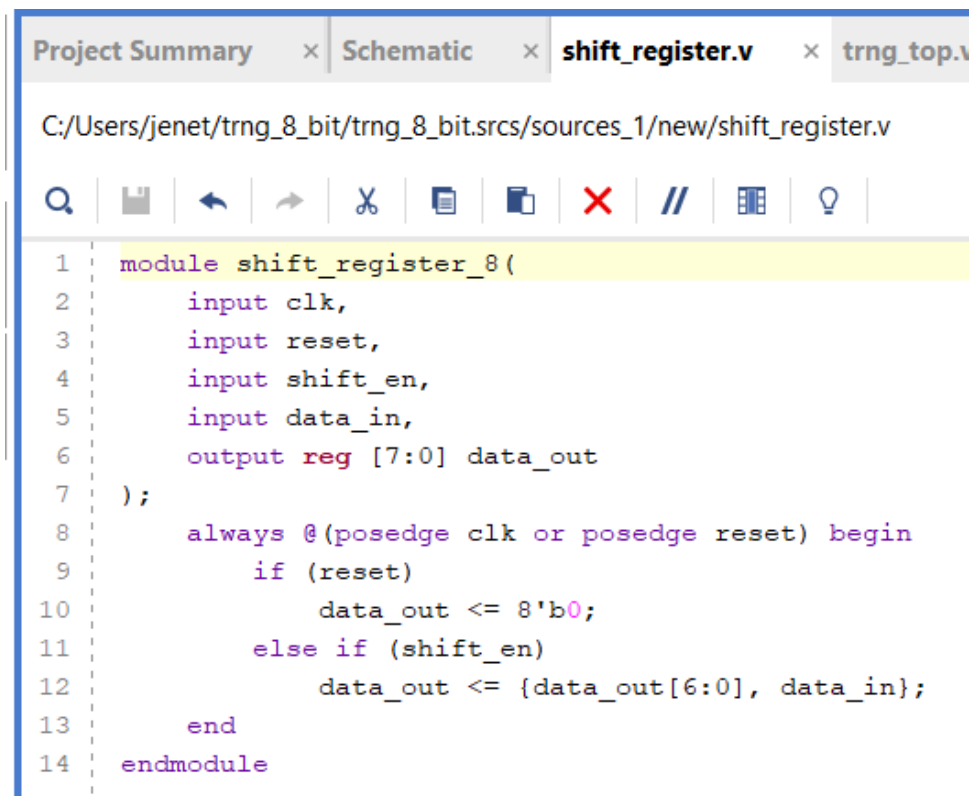


The screenshot shows a Verilog code editor with the following tabs: Project Summary, Schematic, ring_oscillator.v, xor_tree.v, and von_neumann.v. The file path is C:/Users/jenet/trng_8_bit/trng_8_bit.srscs/sources_1/new/von_neumann.v. The code is as follows:

```
1  module von_neumann(  
2      input clk,  
3      input in_bit,  
4      output reg out_bit,  
5      output reg valid  
6  );  
7      reg [1:0] buffer = 2'b00;  
8      reg toggle = 0;  
9  
10     always @(posedge clk) begin  
11         buffer <= {buffer[0], in_bit};  
12         valid <= 0;  
13         if (toggle) begin  
14             if (buffer == 2'b01) begin  
15                 out_bit <= 1'b0;  
16                 valid <= 1;  
17             end else if (buffer == 2'b10) begin  
18                 out_bit <= 1'b1;  
19                 valid <= 1;  
20             end  
21         end  
22         toggle <= ~toggle;  
23     end  
24 endmodule
```

Figure 14: Simulation: Von Neumann Debiasing

- Shift Register Output (Final Output Register)



The image shows a screenshot of a Verilog code editor. The top of the editor has tabs for 'Project Summary', 'Schematic', 'shift_register.v', and 'trng_top.v'. The file path is 'C:/Users/jenet/trng_8_bit/trng_8_bit.srscs/sources_1/new/shift_register.v'. The code is as follows:

```
1  module shift_register_8(  
2      input clk,  
3      input reset,  
4      input shift_en,  
5      input data_in,  
6      output reg [7:0] data_out  
7  );  
8      always @(posedge clk or posedge reset) begin  
9          if (reset)  
10             data_out <= 8'b0;  
11         else if (shift_en)  
12             data_out <= {data_out[6:0], data_in};  
13     end  
14 endmodule
```

Figure 15: Simulation: Final Output Shift

- Elaborated schematic

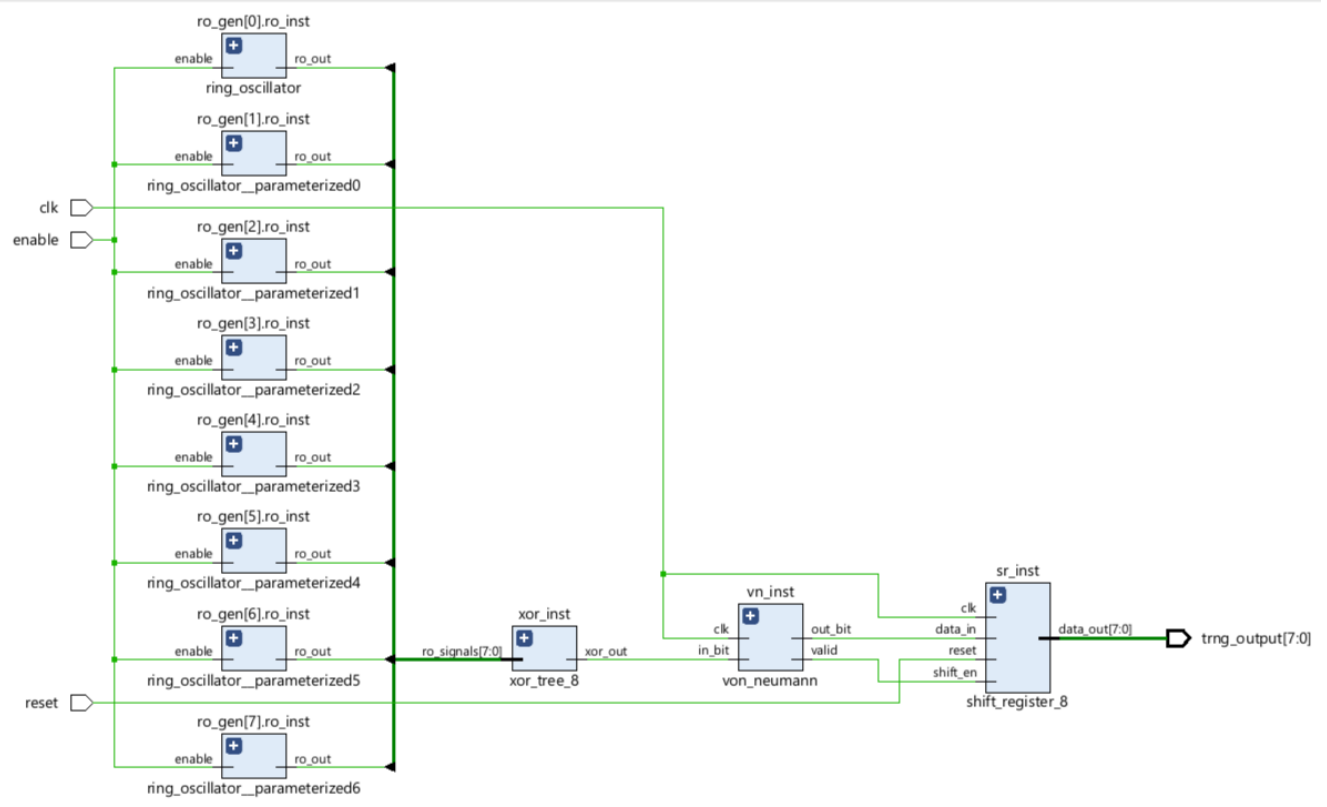


Figure 16: Elaborated Schematic Diagram

11 Comparison of 8-bit and 32-bit TRNG

Parameter	8-bit	32-bit
Number of LUTs	10	27
On-chip Power Consumption	0.072 W	0.086 W
Dynamic Power Consumption	0.001 W	0.003 W
Delay	1.247 ns	1.388 ns
Worst Negative Slack (WNS)	8.602 ns	8.612 ns

Table 3: Comparison of 8-bit and 32-bit TRNG Implementations

12 Presentation Slides

True Random Number Generator (TRNG)

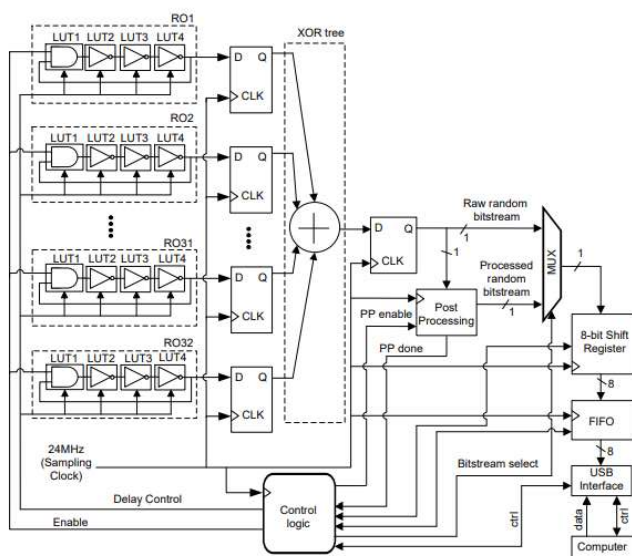
Using Ring Oscillators on FPGA

by applying Von Neumann Extractor

GROUP MEMBERS: SREENESH K.S , JENET JOSEPH, DHARSHAN.S

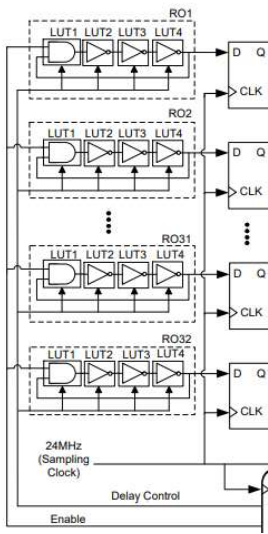
RAJAGIRI SCHOOL OF ENGINEERING & TECHNOLOGY

Overall System Block Diagram & Flow

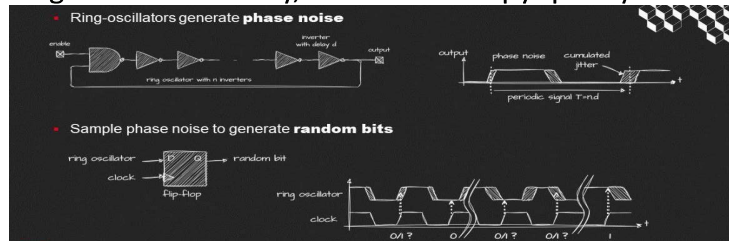
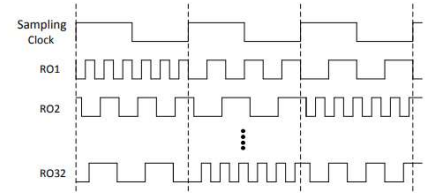


- 32 Ring Oscillators generate jitter-based signals
- Outputs sampled by D Flip-Flops at 24 MHz
- XOR Tree combines 32 bits into 1 bit of entropy
- Raw bitstream sampled again by DFF
- Von Neumann debiaser removes bias from raw bits
- MUX selects raw or debiased stream
- Shift register groups bits into bytes
- FIFO buffers data for smooth USB transfer
- USB interface sends random data to PC
- Control logic manages entire operation

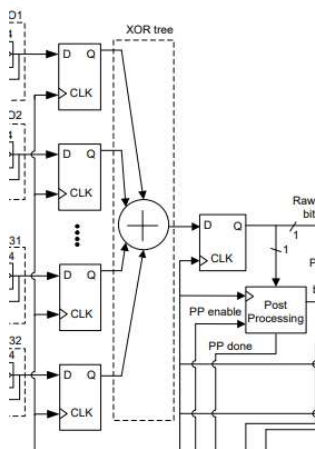
Ring Oscillators & Delay Lines



- Each RO made of 4 LUTs forming oscillating loop
- 3 LUTs form inverter loop; 4th LUT adds control/delay
- Oscillations have jitter due to physical noise sources
- Programmable delay lines control (PDLs) adjust LUT inputs
- PDLs reduce correlation among ROs providing slight different delay, increase entropy quality



Signal Sampling & XOR Tree



- DFFs sample each RO output synchronized to fixed clock (24Mhz)
- XOR Tree mixes all 32 sampled bits into a single bit producing high entropy
- As high entropy= less predictability
- XOR reduces bias and combines entropy sources effectively
- Final DFF samples XOR output to produce raw bitstream ie original bitstream

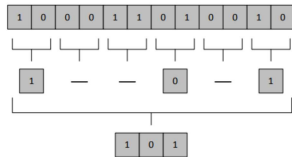
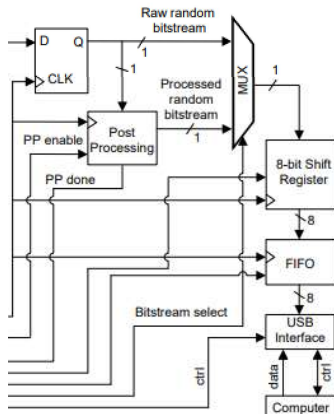


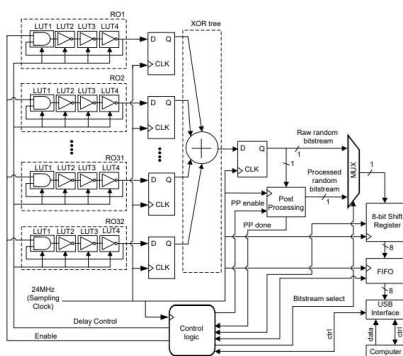
Fig. 7: Principle of operation of the Von Neumann.

Von Neumann Debiasing & Output Processing



- Von Neumann block processes raw bits in pairs:
- Discards pairs 00 or 11
- Outputs 0 for 01, 1 for 10
- Removes bias, improves randomness quality
- Controlled by enable signals (PP enable/done)
- MUX selects raw or processed bitstream output
- 8-bit shift register groups bits into bytes
- FIFO buffers bytes for USB transmission

Control Logic & USB Interface



- Control logic enables/disables ROs and sets delay control
- Selects bitstream output (raw or debiased)
- Manages USB and data transfer

ADV

- Generates true random numbers from physical noise (not algorithmic)
- High entropy output (close to 1.0) after debiasing

DIS

Uses

- Generate secure random numbers for cryptography
- Provide true randomness for games, simulations, and AI applications
- Use physical noise (ring oscillator jitter) as a hardware entropy source
- Prevent prediction or attacks, unlike software-based RNGs
- Integrate into FPGA systems for secure hardware designs

- Von Neumann extractor reduces bit rate (some bits are discarded)
- Requires careful tuning of RO delays to avoid correlation

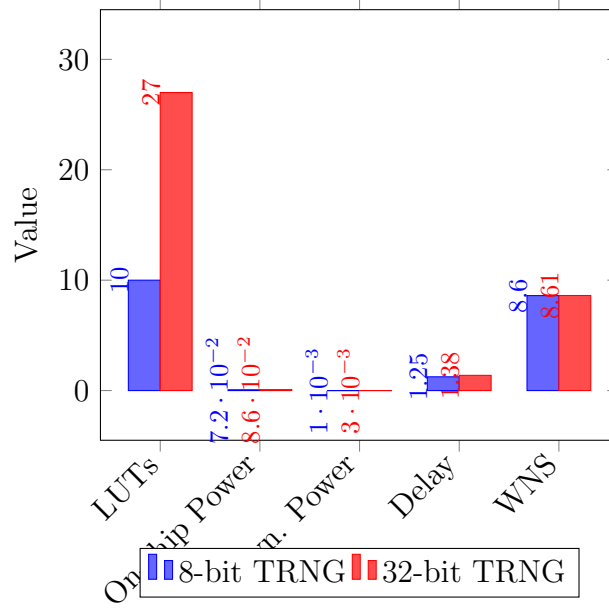


Figure 17: Comparison of 8-bit and 32-bit TRNG Implementations

Project Files: [Google Drive Link of working Video on FPGA](#)

13 Conclusion and Results

This project successfully demonstrates the design and implementation of a hardware-based True Random Number Generator (TRNG) using ring oscillators on an FPGA. By leveraging jitter from ring oscillators and applying Von Neumann debiasing, a statistically unbiased random bitstream was achieved. Both 8-bit and 32-bit versions were implemented and verified through simulation and hardware testing.

The 32-bit TRNG exhibited excellent performance with low resource utilization (27 LUTs), minimal delay (1.38 ns), and efficient power consumption (0.086 W on-chip). Similarly, the 8-bit version achieved even greater efficiency, using only 10 LUTs and consuming 0.072 W on-chip, while maintaining a negligible delay of 1.247 ns. Both designs met timing constraints with robust Worst Negative Slack (WNS) values (8.6 ns), ensuring reliable operation at the target clock period of 10 ns.

Comparison of 8-bit vs 32-bit TRNG: Scaling from 8-bit to 32-bit implementation shows a near-linear increase in LUT utilization (10 to 27) with proportional power consumption growth (0.072W to 0.086W), while maintaining excellent timing performance (1.247ns to 1.38ns delay). The 32-bit version provides higher throughput without compromising entropy quality, making it preferable for applications requiring greater random bit generation rates, whereas the 8-bit version remains ideal for ultra-low-power constrained implementations. Both designs demonstrate robust randomness characteristics suitable for cryptographic applications.

The project's results indicate that the proposed TRNG architecture is efficient, synthesizable, and reliable for embedded systems and cryptographic use cases. The modu-

larity of the design allows for scalability and adaptability in future work, such as integrating with encryption engines or expanding to higher bit-width outputs. Overall, the hardware-based TRNG proves to be a practical and secure solution for generating true random numbers in FPGA-based systems.