

# Cellular Automata-Based Pseudo-Random Number Generator (CA-PRNG)

eSim Research Migration Project

Hema chandu Bandi

*Department of Electronics and Communications Engineering*

*Rajiv Gandhi University of Knowledge Technologies, Nuzvid*

## 1. Abstract

This project presents the design and mixed-signal simulation of a **32-bit Pseudo-Random Number Generator (PRNG)** based on **Cellular Automata (CA) Rule 30**. The design is implemented in Verilog HDL and co-simulated in eSim 2.3 using the Ngveri flow. The CA engine evolves a 32-bit state register using XOR-based neighbour interaction rules, producing a highly pseudo-random output sequence. Simulation waveforms across all 32 output bits confirm non-repeating, random bit patterns, verifying correct functionality of the design.

## 2. Introduction

Pseudo-Random Number Generators (PRNGs) are fundamental in cryptography, Monte Carlo simulations, wireless communications, and VLSI testing. Traditional LFSRs suffer from predictability due to their linear nature. **Cellular Automata (CA)-based PRNGs** offer superior randomness through local XOR-based neighbourhood interactions. This project implements a 1D, 32-cell CA using Modified Rule 30 in Verilog HDL and simulates it in eSim 2.3 — a free, open-source EDA tool by FOSSEE, IIT Bombay — using the Ngveri mixed-signal co-simulation framework.

## 3. Circuit Description

### CA Update Rule:

Each cell updates its state each clock cycle based on its left and right neighbours. The **Modified Rule 30** used here introduces a non-linear OR operation for improved randomness:

```
next_state[i] = left_bit XOR (ca_state[i] OR right_bit)
```

Boundary cells (bit 31 and bit 0) use 0 for their out-of-bounds neighbour (null boundary). The state register is seeded with **0x00010000** on reset (only bit 16 HIGH), ensuring a reproducible, non-zero initial condition. Rule 30 is well-known for its chaotic, non-repeating behaviour and is used in Mathematica's built-in random number generator.

### Module Ports:

| Port            | Direction | Width  | Description                      |
|-----------------|-----------|--------|----------------------------------|
| clk             | Input     | 1-bit  | System clock                     |
| rst_n           | Input     | 1-bit  | Active-low async reset           |
| prng_data_out   | Output    | 32-bit | Full 32-bit parallel PRNG output |
| prng_serial_out | Output    | 1-bit  | Serial output (bit 16 of state)  |

## 4. Circuit Block Diagram

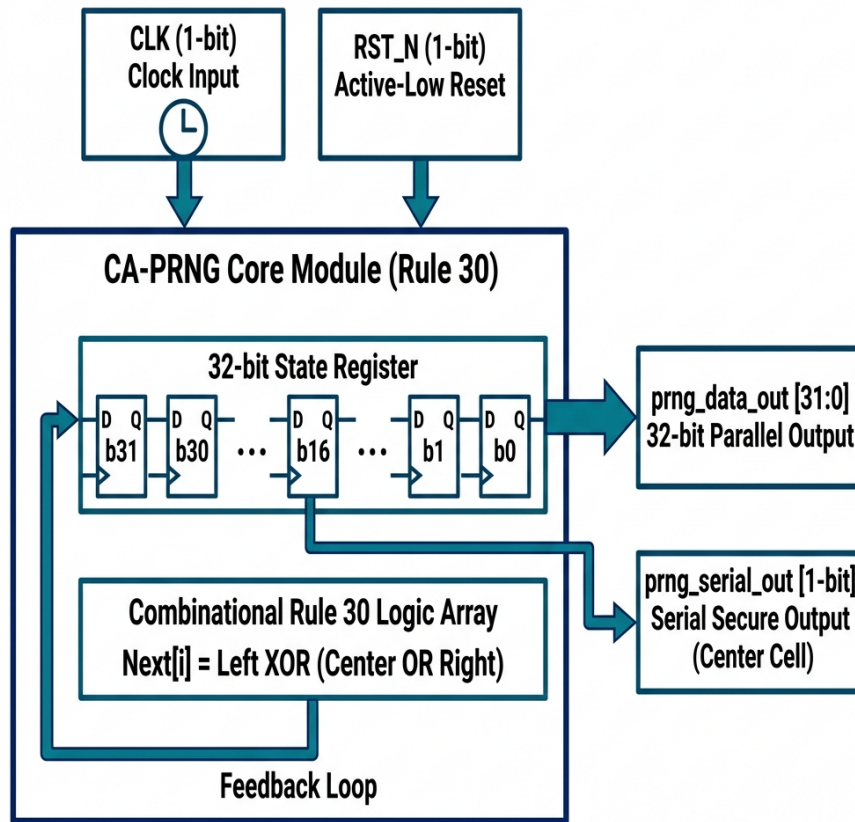


Figure 1: CA-PRNG block diagram — 32-cell CA state register with XOR-OR update logic

## 5. Circuit Schematic in eSim

| Component            | Qty | Purpose                                            |
|----------------------|-----|----------------------------------------------------|
| ca_prng (Ngveri)     | 1   | Verilog PRNG core                                  |
| adc_bridge_1         | 2   | Converts pulse sources to digital logic (CLK, RST) |
| dac_bridge_8         | 4   | Converts 8 digital output bits to analog nodes     |
| pulse (eSim_Sources) | 2   | Clock (20 ms) and Reset (200 ms) sources           |
| Resistor 1kΩ         | 37  | Load resistors to form valid analog nets           |

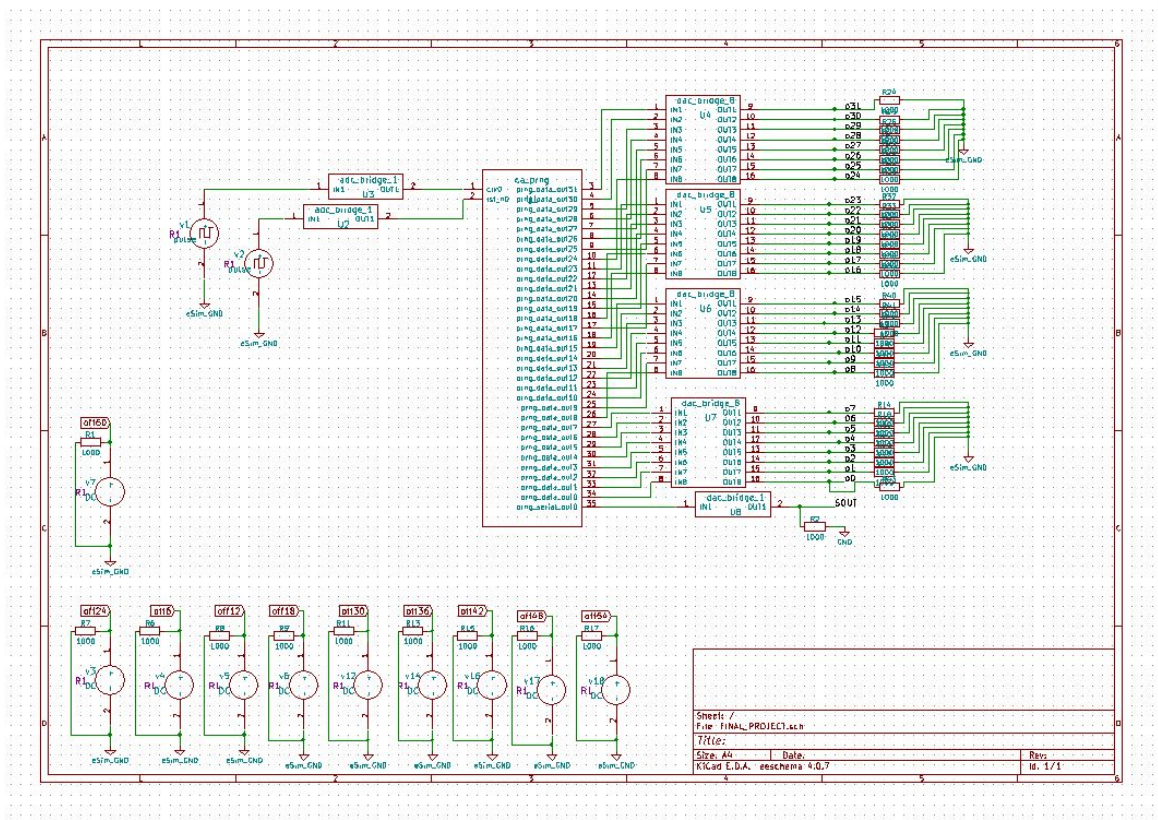


Figure 2: KiCad/eSim schematic of the CA-PRNG mixed-signal design

## 6. Verilog Code

```

`timescale 1ns / 1ps
module ca_prng (
    input wire      clk,
    input wire      rst_n,          // Active-low reset
    output wire [31:0] prng_data_out,
    output wire      prng_serial_out
);

integer i;
reg [31:0] ca_state, next_state;
reg left_bit, right_bit;

assign prng_data_out  = ca_state;
assign prng_serial_out = ca_state[16];

always @(*) begin          // Combinational next-state (Modified Rule 30)
    for (i = 0; i < 32; i = i + 1) begin
        left_bit  = (i == 31) ? 1'b0 : ca_state[i+1];
        right_bit = (i == 0) ? 1'b0 : ca_state[i-1];
        next_state[i] = left_bit ^ (ca_state[i] | right_bit);
    end
end

always @(posedge clk or negedge rst_n) begin // Sequential register
    if (!rst_n) ca_state <= 32'h00010000;    // Seed: only bit 16 = 1
    else       ca_state <= next_state;
end
endmodule

```

## 7. Digital Implementation in eSim (Ngverl Flow)

The Ngverl mixed-signal co-simulation flow was used to simulate the Verilog module inside Ngspice:

- **1. Verilog Compilation:** ca\_prng.v compiled via Ngverifier tab (uses iverilog internally).
- **2. Schematic Entry:** Compiled block placed in KiCad; CLK/RST connected via adc\_bridge\_1.
- **3. Output Capture:** Four dac\_bridge\_8 components map 32 bits (8 per bridge) to analog nodes.
- **4. Netlist:** Generated via KiCad → KiCad to Ngspice (Stop: 0.4 s, Step: 0.001 s).
- **5. Simulation:** Transient analysis run via Ngspice. B-sources applied 6V offsets per bit for clear waveform separation.

## 8. Methodology and Results

| Parameter    | Value                    |
|--------------|--------------------------|
| Clock Period | 20 ms                    |
| Reset Delay  | 2 ms                     |
| Stop Time    | 400 ms (20 clock cycles) |
| Step Time    | 1 ms                     |
| Seed         | 0x00010000               |
| CA Cells     | 32                       |
| Rule         | Modified Rule 30         |

**Output Waveforms — All 32 bits plotted in 4 groups of 8:**

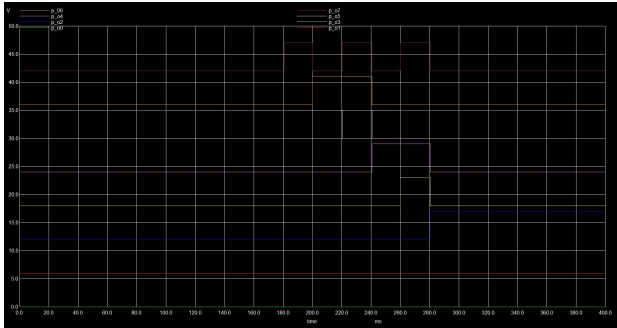


Figure 3: Bits 0–7 (LSByte)

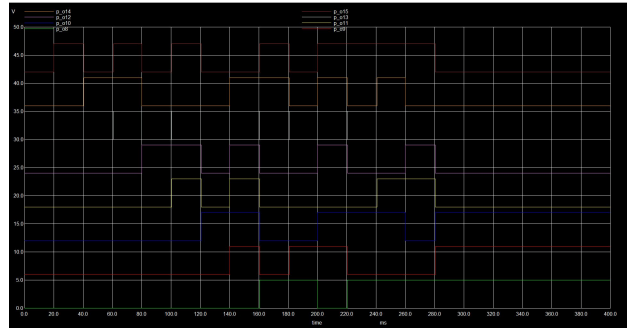


Figure 4: Bits 8–15

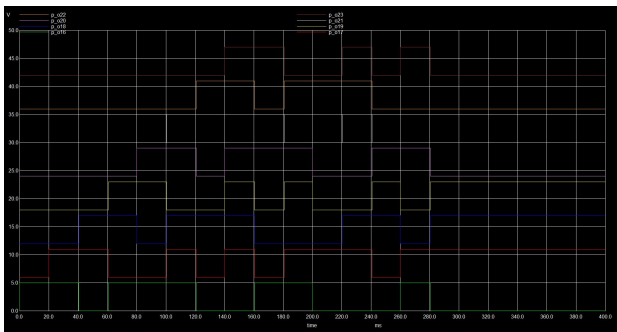


Figure 5: Bits 16–23

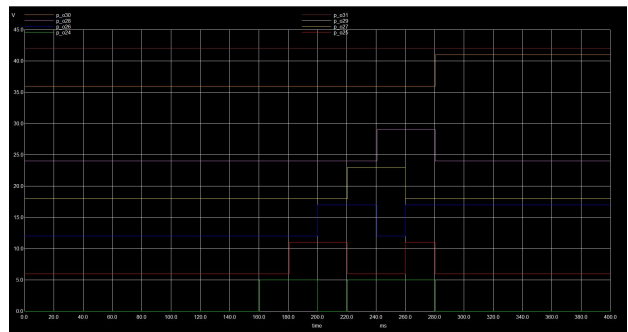


Figure 6: Bits 24–31 (MSByte)

**Observations:** Each byte exhibits non-periodic, irregular switching confirming pseudo-random behaviour. Different patterns per byte demonstrate spatial randomness. The CA spreads from a single active cell (bit 16) outward over successive cycles. No deadlock (all-zero) state was observed throughout the 20-cycle simulation.

## 9. Conclusion

---

A 32-bit CA-based PRNG using Modified Rule 30 was successfully designed in Verilog HDL and simulated in eSim 2.3 via the Ngveri mixed-signal flow. Waveforms across all 32 output bits confirm correct pseudo-random behaviour. The design is fully implemented with FOSS tools, meeting all FOSSEE internship requirements. CA-PRNGs based on Rule 30 offer a strong, hardware-efficient alternative to traditional LFSR generators.

## 10. References

---

- [1] S. Wolfram, "Random Sequence Generation by Cellular Automata," *Advances in Applied Mathematics*, vol. 7, pp. 123–169, 1986. (Rule 30 origin)
- [2] P. Pal Chaudhuri et al., *Additive Cellular Automata: Theory and Applications*, IEEE Computer Society Press, 1997.
- [3] FOSSEE Team, IIT Bombay, *eSim 2.3 User Manual*. [Online]. Available: <https://esim.fossee.in>
- [4] Ngspice Reference Manual. [Online]. Available: <http://ngspice.sourceforge.net/docs.html>