FPGA Lab Group Project

Title – Implementation of a Stochastic Number Generator using D – Flipflop in Verilog and FPGA based hardware simulation

Objective –

1. To study and implement the circuit of a simple stochastic number generator using D_FF (referring to Figure 3b)
2. Implement the circuit in Verilog on the Xilinx platform and, run behavioral simulations to confirm its characteristics.

Required Components –

1. +5/3.3 Volts regulated power supply from DC source or personal computer.
2. Xilinx Vivado software installed in pc or laptop
3. FPGA board (Xilinx PYNQ Z2 kit)
4. Jumper Wires, breadboard and LEDs

Theory

Stochastic Number Generator (SNG) using D Flip-Flops

In stochastic computing, a **Stochastic Number Generator (SNG)** is a fundamental building block that converts a deterministic binary number into a stochastic bitstream. This bitstream represents the probability value of a real number between 0 and 1 (in unipolar encoding) based on the ratio of '1's to the total bits.

The basic SNG architecture typically consists of two parts:

1. **Random Number Source (RNS):** Generates random or pseudo-random binary numbers.
2. **Comparator (CMP):** Compares the random number with a constant binary number to generate stochastic outputs.

A **Linear Feedback Shift Register (LFSR)** is used as the Random Number Source because it generates a pseudo-random sequence efficiently with minimal hardware. At every clock cycle, the LFSR generates a random number R, which is then compared with a constant binary number X.

- If R < X, the comparator outputs a '1'.
- Otherwise, it outputs a '0'.
  Thus, a stochastic bitstream is created where the probability of getting a '1' matches the original input number.

Optimization with D Flip-Flops (DFFs)

- **DFFs Inside the SNG :**
  A better design is achieved by placing DFFs inside the SNG architecture, immediately after the comparator but **before** feeding the stochastic bitstream to further blocks.

Experimental Findings:

- Power reduced from **50.2 µW** (DFFs outside) to **33.7 µW** (DFFs inside).
- Area reduced from **163.9 µm²** to **110.1 µm²**.
- Inserting DFFs inside the SNG showed better efficiency despite each DFF individually consuming some power (~4.3%) and area (~3.9%).

Thus, inserting D Flip-Flops inside the SNG architecture **optimizes** the stochastic circuit for **low-power, low-area**, and **high-reliability** operation, making it highly suitable for modern applications like neural networks, image processing, and control systems.

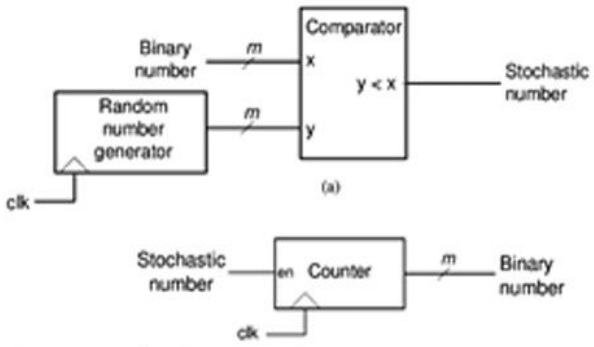


Figure 2. Number Conversion Circuits
(a) Binary to Stochastic and b) Stochastic to Binary)

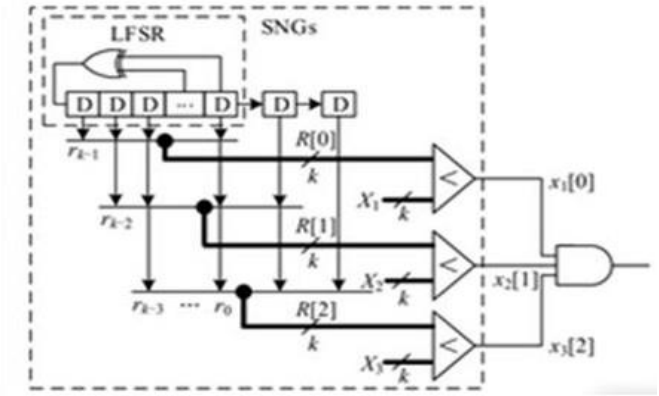


Figure 3(b). D Flip Flops Inside SNG

Gate – level architecture of the proposed circuit

Verilog Code for Implementation

(Linear Feedback Shift Register)

```
module lfsr(input rst, input clk, output reg [3:0] Q);
always @(posedge rst or posedge clk) begin
    if (rst) begin
        Q <= 4'b0001;
    end else begin
        Q <= {Q[2:0], Q[3] ^ Q[0]};
    end
end
endmodule
```

(Comparator)

```
module cmp(input [3:0] R, input [3:0] X, output m);
assign m = R < X;
endmodule
```

(D_FF)

```
module dff(input clk, input rst, input d, output reg q);
always @(posedge rst or posedge clk) begin
    if (rst) begin
        q <= 1'b0;
    end else begin
        q <= d;
    end
end
endmodule
```

(Stochastic Number Generator)

```
module sngd(input rst, input clk, output B);
wire [3:0] Q;
wire [2:0] N;
wire q4;
wire q5;

lfsr l0(rst, clk, Q);
dff d1(clk, rst, Q[0], q4);
dff d2(clk, rst, q4, q5);

cmp c1(Q, 4'd15, N[0]);
cmp c2({Q[2:0], q4}, 4'd15, N[1]);
cmp c3({Q[1:0], q4, q5}, 4'd15, N[2]);

assign B = N[0] & N[1] & N[2];
endmodule
```

(Top Level Module)

```
module fout(input clk, input rst, output B, output reg [3:0] count, output reg [14:0] W);
sngd s1(rst, clk, B);
wire B;
reg enable = 1'b1;

always @(posedge clk or posedge rst) begin
    if (rst) begin
        W <= 15'd0;
        count <= 4'b0000;
        enable <= 1'b1;
    end else if (enable) begin
        if (count == 4'b1110) begin
            count <= 4'b0000;
            enable <= 1'b0;
        end else begin
            W[count] = B;
            count = count + 1;
        end
    end else begin
        W <= W;
    end
end
endmodule
```

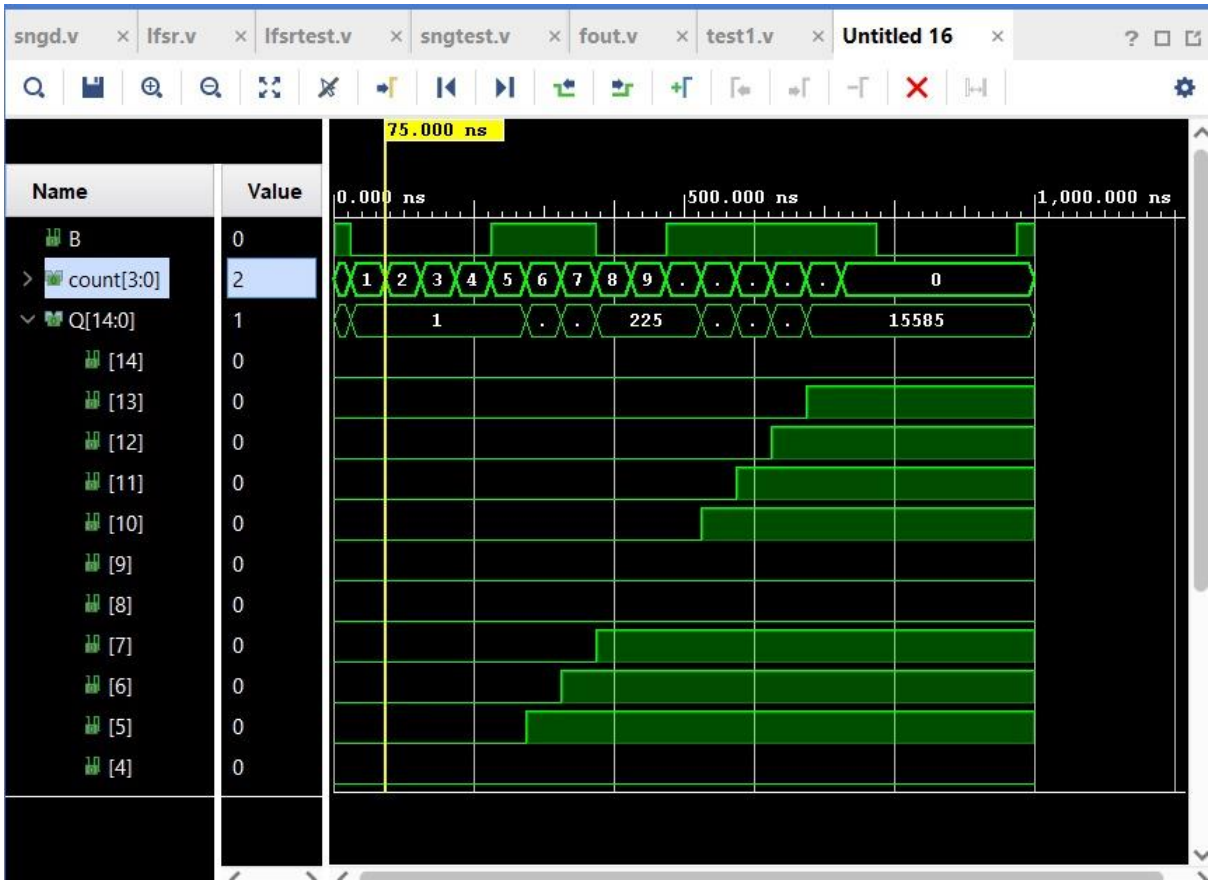
(Clock Divider Module)

```
module clk_divide(
input clk_in,
output reg clk_out
);

parameter divisor = 28'd2;
reg [27:0]count = 28'd0;

always@(posedge clk_in) begin
    if(count<divisor - 1)
        count<= count+28'd1;
    else
        count<=28'd0;

        if(count<divisor>>1)
            clk_out = 1;
        else
            clk_out = 0;
end
endmodule
```

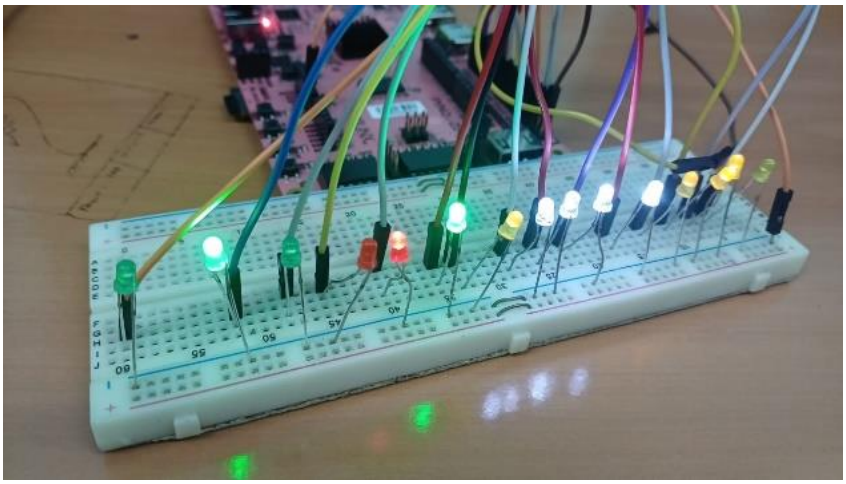
(Testbench for software simulation)

```
module test1(output B, output [3:0] count, output [14:0] Q);
reg clk;
reg rst;

fout l0(clk, rst, B, count, Q);

initial begin
    clk = 0;
    rst = 1;
    #3 rst = 0;
end

always #25 clk <= ~clk;

endmodule
```

Behavioral Simulation Results -



Hardware Simulation



Output:

We observe that 11 bulbs are glowing, which can be explained as follows:

The values chosen for X1, X2, X3 are each 15. In every cycle where all three values R[0], R[1], and R[2] are less than 15, the AND gate produces a HIGH output, causing the corresponding bulb to glow. Conversely, if any one of R[0], R[1], or R[2] equals 15, the AND gate outputs LOW, and the bulb does not glow.

Since the LFSR used is 4 bits wide, its maximum output value is 15 (2^4-1). Therefore, no value greater than 15 can occur. Importantly, the bulbs fail to glow when any input is exactly 15.

Because R[1] and R[2] are delayed versions of R[0], the value 15 appears separately at R[0], R[1], and R[2] in three different cycles. As a result, three bulbs corresponding to these cycles do not glow. Additionally, the LFSR cannot produce the state 0000, as it would cause a lock-out condition where the LFSR gets stuck permanently at zero. Thus, one more bulb does not glow for this missing state.

In total, out of 15 possible states, four bulbs remain off (three due to the value 15 appearing separately at R[0], R[1], and R[2], and one due to the forbidden 0000 state), leading to 11 bulbs glowing, which matches the observed output.