DIGITAL DICE
USING
RANDOM NUMBER GENERATOR

NAME : Aayush Desai

ROLL NO. : 22BEC025

COURSE : FPGA

FACULTY : Dr. Akash Mecwan

- **Abstract**

A digital dice is an electronic device designed to simulate the randomness of traditional dice used in various games. This report explores methods for generating random numbers, distinguishing between true random numbers derived from physical phenomena like atmospheric noise, and pseudo-random numbers produced through computational algorithms. Examples of pseudo-random number generators such as the Linear Congruential Generator, Middle-Square Generator, and Linear Feedback Shift Register are discussed. Understanding these methods is crucial for developing reliable digital dice systems for gaming applications.

- **Keywords**

1. Digital dice
2. Random number
3. True random number
4. Computational algorithms
5. Seed value
6. Pseudorandom number generator
7. Linear Congruential Generator (LCG)
8. Linear feedback shift register (LFSR)
9. Flip-flops
10. Exclusive-OR configuration

- **Literature Survey on Random Number Generation Techniques**

Random number generation plays a crucial role in various applications, ranging from gambling devices to cryptographic systems. In digital circuits, the generation of random numbers is essential for creating fair and unpredictable outcomes. In this literature survey, we will explore the methods employed for generating random numbers, particularly focusing on True Random Number (TRN) generation and Pseudo Random Number (PRN) generation.

True Random Number Generation

True Random Number (TRN) generation relies on physical processes to get randomness. These processes happen outside of what computers can do, making them inherently unpredictable.

Examples of sources for TRN include:

Radioactive Decay: The decay of radioactive isotopes provides a source of randomness due to its inherently probabilistic nature.

Photoelectric Effect: The emission of electrons from a material when exposed to light serves as another source of randomness.

Cosmic Background Radiation: Background radiation from outer space contributes to randomness, as it is influenced by celestial events beyond human control.

Atmospheric Noise: Variations in atmospheric conditions yield random fluctuations that can be harnessed for generating random numbers.

TRN generation involves careful measurement and adjustment to account for biases introduced during the measurement process. The resulting random numbers are deemed "true" as they are derived from inherently random physical phenomena.

Pseudo-Random Number Generation

In contrast to TRN generation, Pseudo Random Number (PRN) generation relies on computational algorithms to produce apparently random results. The term "apparently" is crucial here, as the randomness stems from the complexity of the algorithm rather than true randomness. PRN generation typically involves the following components:

Seed Value: An initial value, also known as the seed or key, serves as the starting point for the PRN generation algorithm.

Computational Algorithms: These algorithms manipulate the seed value through mathematical operations to produce sequences of numbers that appear random. However, these sequences are entirely deterministic and reproducible given the same initial seed and algorithm.

Examples of PRN generation algorithms include:

Linear Congruential Generator (LCG): This algorithm iteratively generates numbers using a linear recurrence relation.

Middle Square Generator: Numbers are generated by squaring a seed value and extracting a middle portion as the next random number.

Linear Feedback Shift Register (LFSR): LFSR employs shift registers and exclusive-OR (XOR) operations to generate sequences of binary digits with pseudo-random properties.

- **LIMITATIONS OR DRAWBACKS OF CURRENTLY AVAILABLE TECHNOLOGY**

The limitations of currently available technology for generating random numbers, particularly for applications like digital dice, are primarily related to the methods used for generating randomness.

TRUE RANDOM NUMBER GENERATION (TRNG):

Dependency on Physical Phenomena: TRNGs rely on physical phenomena like radioactive decay or atmospheric noise. While these sources are considered random, they might be challenging to harness effectively in all environments.

Hardware Constraints: Incorporating hardware for true randomness into devices might be expensive or impractical for some applications, especially small, low-cost devices like digital dice.

Bias Correction: Even physical processes deemed random might exhibit biases due to measurement or environmental factors, requiring complex correction mechanisms.

PSEUDO RANDOM NUMBER GENERATION (PRNG):

Reproducibility: PRNGs produce sequences of numbers based on deterministic algorithms and an initial seed value. If the seed value and algorithm are known, the sequence becomes predictable, making it unsuitable for cryptographic applications or scenarios requiring high unpredictability.

Periodicity: Many PRNG algorithms have a limited period after which they repeat their sequence. For certain applications requiring long sequences without repetition, this limitation can be a concern.

Quality of Randomness: Some PRNG algorithms may exhibit patterns or correlations in their output, especially if poorly designed or if the seed selection is inadequate. This can lead to non-uniform distributions or biases in the generated numbers.

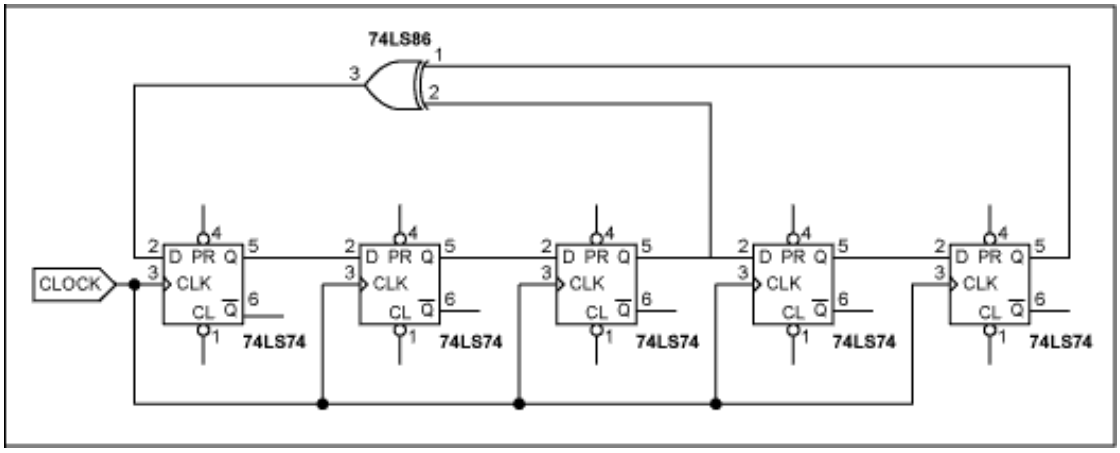
- **PROPOSED SOLUTION/METHODOLOGY**

Here's a proposed methodology:

1. **Hardware Integration for True Randomness**:
    - Incorporate sensors or modules capable of capturing physical phenomena that exhibit true randomness, such as atmospheric noise or thermal noise.
    - Use analog-to-digital converters (ADCs) to digitize the analog signals obtained from these sensors.
    - Implement conditioning and post-processing algorithms to ensure the raw data from the sensors is suitable for generating random numbers. This may involve techniques like whitening to remove biases or correlations.
2. **Seed Generation for PRNG**:
    - Utilize the true random numbers obtained from the hardware components as the seed values for a PRNG algorithm.
    - Implement a robust PRNG algorithm that is resistant to prediction attacks and exhibits desirable statistical properties.
    - Ensure that the PRNG algorithm has a sufficiently large state space to reduce the likelihood of repetition within the sequence.
3. **Hybrid Random Number Generation**:
    - Develop a hybrid random number generation module that combines outputs from the TRNG hardware and the PRNG algorithm.
    - Use statistical tests and analysis to validate the randomness and uniformity of the generated numbers.
    - Implement mechanisms to periodically reseed the PRNG using fresh true random values to enhance unpredictability and mitigate potential biases.

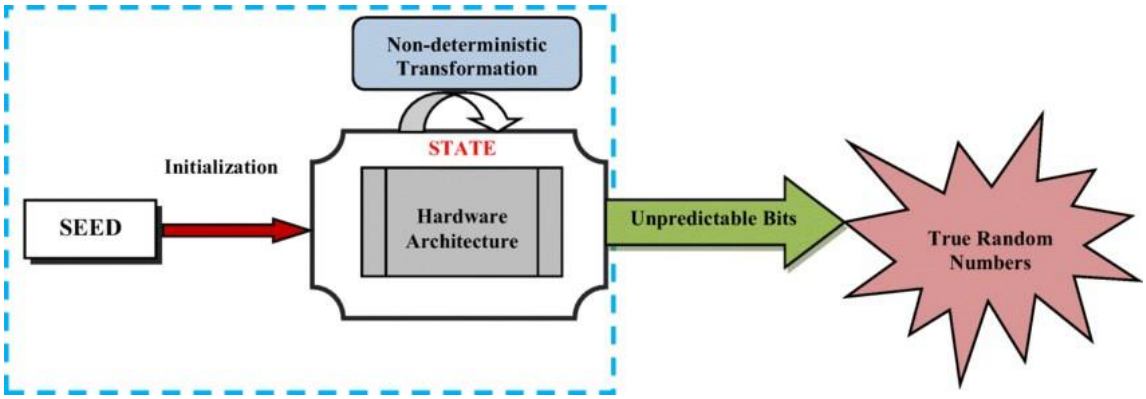
- **CIRCUIT DIAGRAM**

PRNG (PSEUDO RANDOM NUMBER GENERATOR)

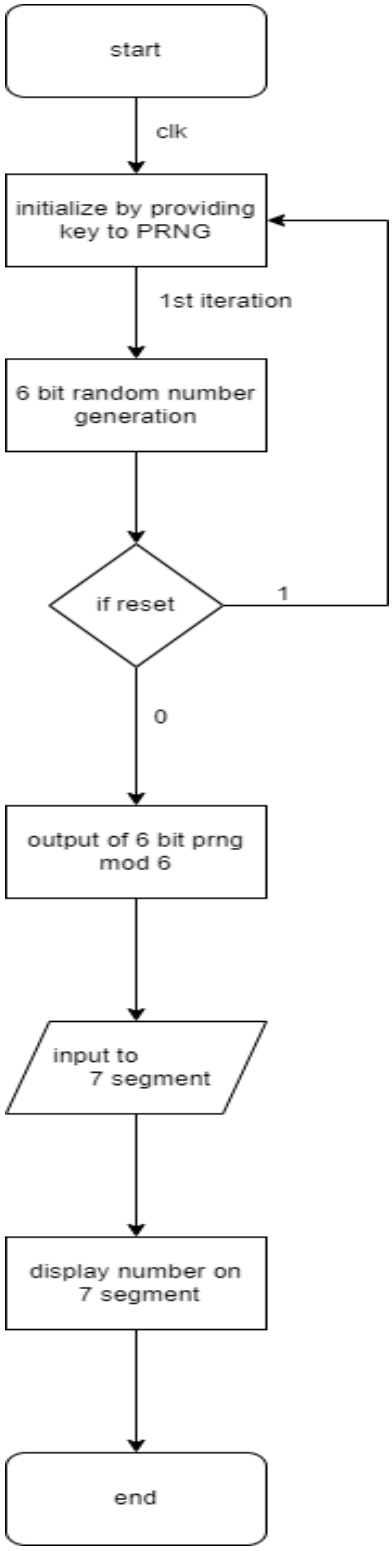


BLOCK DIAGRAM

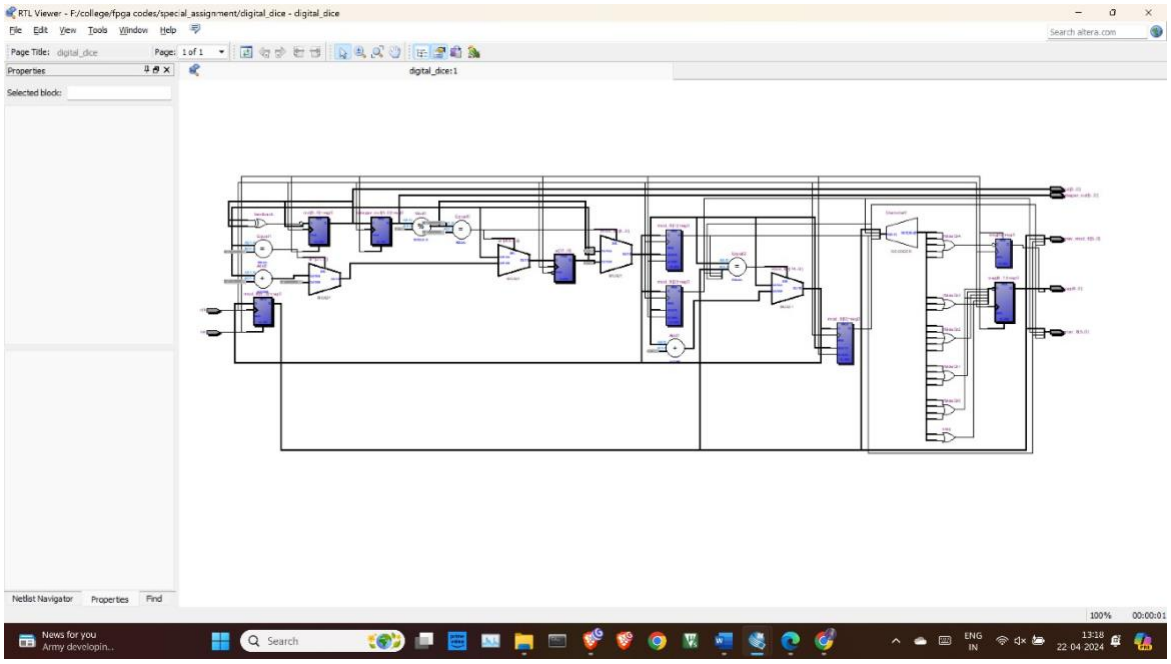
TRNG (TRUE RANDOM NUMBER GENERATOR)



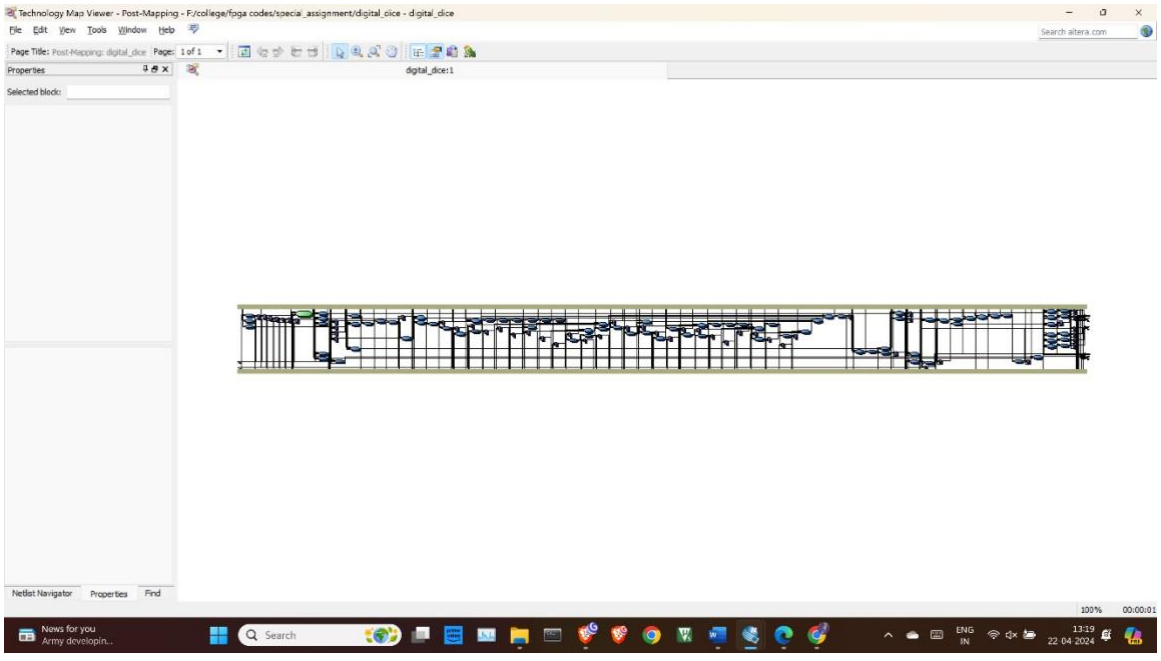
- **FLOWCHART**


start
clk
initialize by providing key to PRNG
1st iteration
6 bit random number generation
if reset
1
0
output of 6 bit prng mod 6
input to 7 segment
display number on 7 segment
end

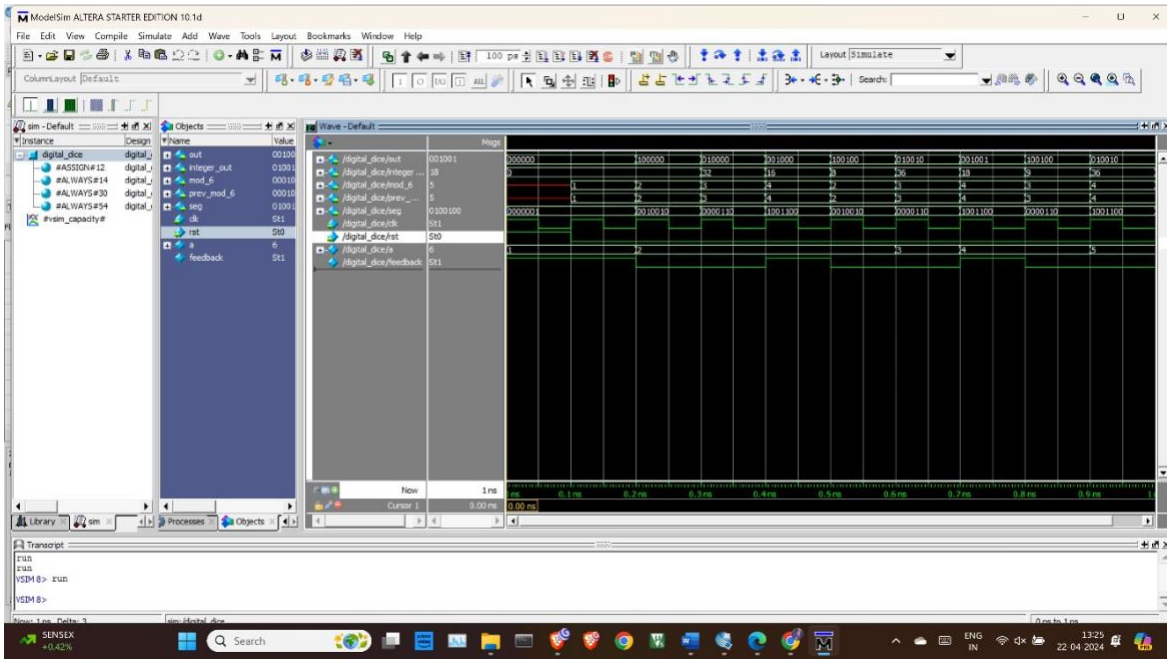
- **RTL**


RTL Viewer - F:/college/fpga codes/special_assignment/digital_dice - digital_dice
File Edit View Tools Window Help
Page Title: digital_dice
Page: 1 of 1
Properties
Selected block:
digital_dice:1
Netlist Navigator
Properties
Find
100%
00:00:01
News for you
Army developin...
Search
ENG
IN
13:18
22-04-2024

- **TTL**


Technology Map Viewer - Post-Mapping - F:/college/fpga codes/special_assignment/digital_dice - digital_dice
File Edit View Tools Window Help
Page Title: Post-Mapping: digital_dice
Page: 1 of 1
Properties
Selected block:
digital_dice:1
Netlist Navigator
Properties
Find
100%
00:00:01
News for you
Army developin...
Search
ENG
IN
13:19
22-04-2024

- **WAVEFORM (RTL SIMULATION)**

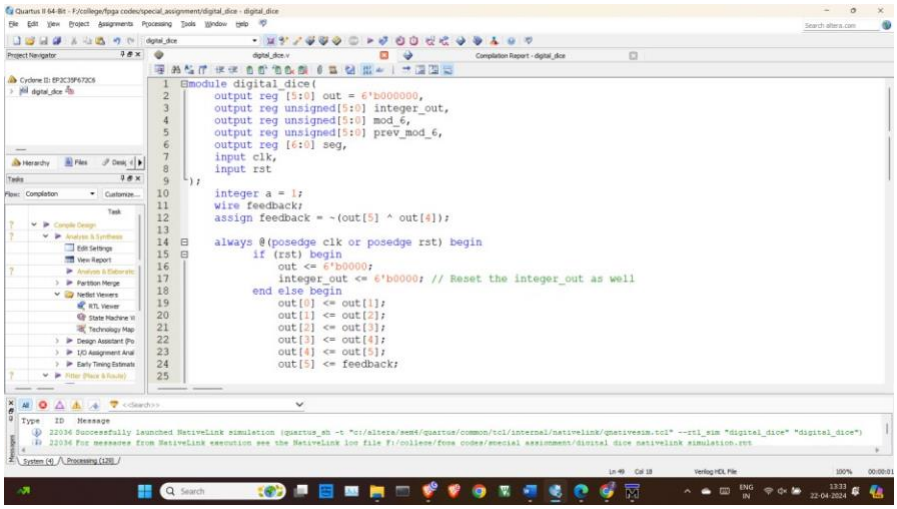


- **CONCLUSION**

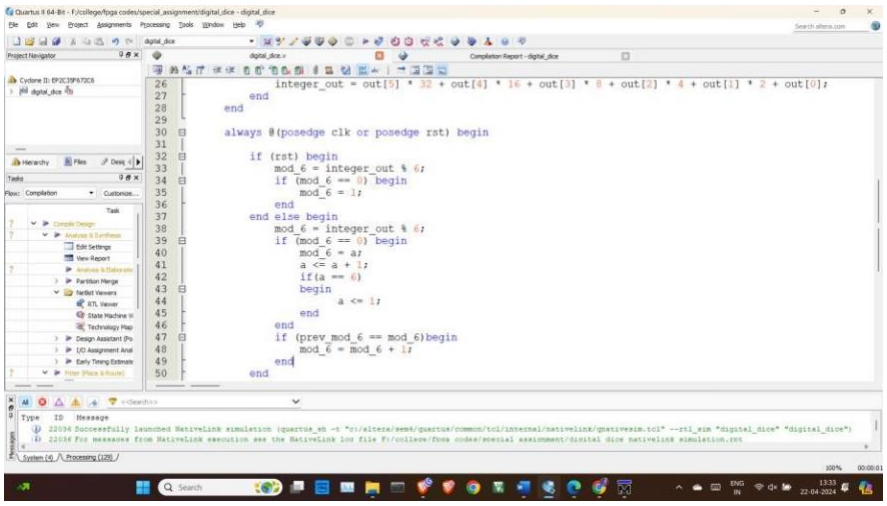
To summarize, digital dice have transformed how random numbers are generated, particularly in gaming and probability. Understanding the two main methods of generating random numbers - True Random Number and Pseudo Random Number helps to explain how these devices work.

In gaming and beyond, the choice between true and pseudo-randomness often depends on the level of unpredictability required and the computational resources available.

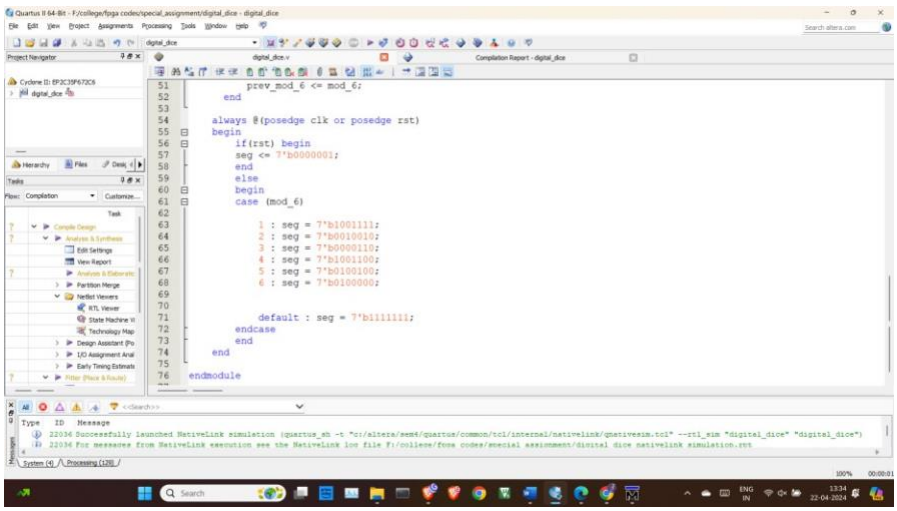
CODE



```
module digital_dice(
    output reg [5:0] out = 6'b000000,
    output reg unsigned[5:0] integer_out,
    output reg unsigned[5:0] mod_6,
    output reg unsigned[5:0] prev_mod_6,
    output reg [6:0] seg,
    input clk,
    input rst
);
    integer a = 1;
    wire feedback;
    assign feedback = ~(out[5] ^ out[4]);

    always @(posedge clk or posedge rst) begin
        if (rst) begin
            out <= 6'b0000;
            integer_out <= 6'b0000; // Reset the integer_out as well
        end else begin
            out[0] <= out[1];
            out[1] <= out[2];
            out[2] <= out[3];
            out[3] <= out[4];
            out[4] <= out[5];
            out[5] <= feedback;
```



```
            integer_out = out[5] * 32 + out[4] * 16 + out[3] * 8 + out[2] * 4 + out[1] * 2 + out[0];
        end
    end

    always @(posedge clk or posedge rst) begin

        if (rst) begin
            mod_6 = integer_out % 6;
            if (mod_6 == 0) begin
                mod_6 = 1;
            end
        end else begin
            mod_6 = integer_out % 6;
            if (mod_6 == 0) begin
                mod_6 = a;
                a <= a + 1;
                if(a == 6)
                begin
                    a <= 1;
                end
            end
            if (prev_mod_6 == mod_6)begin
                mod_6 = mod_6 + 1;
            end
        end
```



```
            prev_mod_6 <= mod_6;
        end

    always @(posedge clk or posedge rst)
    begin
        if(rst) begin
        seg <= 7'b0000001;
        end
        else
        begin
        case (mod_6)

            1 : seg = 7'b1001111;
            2 : seg = 7'b0010010;
            3 : seg = 7'b0000110;
            4 : seg = 7'b1001100;
            5 : seg = 7'b0100100;
            6 : seg = 7'b0100000;


            default : seg = 7'b1111111;
        endcase
        end
    end

endmodule
```