CPLD-based Arithmetic Memory Game using Verilog

Arnav Singh[a], **Lakshya Bhatnagar**[b], **Mahesh Pareek**[c] **and Arnav Panjla**[d]

[a]*Arnav Singh, 2023EE10968*
[b]*Lakshya Bhatnagar, 2023AM10945*
[c]*Mahesh Pareek, 2023MT10586*
[d]*Arnav Panjla, 2023EE10978*

Prof. Dhiman Malik
TA: Chithambara J

Abstract—This project presents the design and implementation of a memory-based arithmetic game on the MAX3000A CPLD using Verilog HDL. The system operates at a 1Hz clock rate, and the LFSR ensures a maximal-length pseudo-random sequence, critical for non-repetitive gameplay logic. A dedicated binary-to-BCD module ensures proper formatting for the 7-segment displays. This project reinforces concepts of sequential logic, finite state machines, and modular Verilog design in a real-world CPLD application.

Keywords—*CPLD, Verilog, Memory Game, Arithmetic Game, LFSR, FSM, BCD Conversion*

1. Introduction

This project implements a memory-based arithmetic game on the MAX3000A CPLD using Verilog HDL. The system operates on a 1Hz clock, with each game cycle divided into five distinct phases:

1. A 5-bit Linear Feedback Shift Register (LFSR) generates pseudo-random numbers, which are displayed one at a time and internally summed.
2. After displaying the numbers, the screen is cleared to allow the player to input their calculated sum using the onboard switches.
3. The system then displays the correct sum and compares it to the player's input.
4. LED indicators provide immediate feedback, showing success or failure through predefined light patterns.
5. Finally, the system resets automatically to begin a new game round.

A separate `binary_to_bcd` module handles binary to BCD conversion, enabling correct representation on 7-segment displays.

2. Motivation

The goal of this project is to practically apply concepts of Verilog programming, sequential digital design, and modular circuit development on CPLDs. By integrating a memory challenge with arithmetic operations, the project not only reinforces theoretical knowledge but also develops skills related to finite state machines, random number generation using LFSRs, and binary-BCD conversions essential for digital display systems.Incorporating FSM logic further instills principles of deterministic control flow and modular behavioral modeling.

3. Implementation Methodology

3.1. Pin Mapping

The following table summarizes the pin configuration used for interfacing the CPLD with switches, LEDs, 7-segment display outputs, and the clock:

Inputs	Pin No.
SW1	4
SW2	5
SW3	6
SW4	8
SW5	9
SW6	11
SW7	12
SW8	14

Outputs	Pin No.
LED1	24
LED 2	25
LED 3	26
LED 4	27
LED 5	28
LED 6	29
LED 7	31
LED 8	33

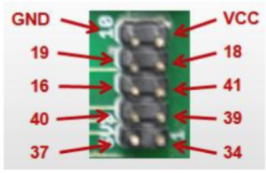


Figure 1. MAX3000A pinout for reference

Signal	Pin	Description
clk	43	Global clock input (1Hz)
o_clk	24	Debugging clock output
bcd_tens[3]	34	BCD tens place output (MSB)
bcd_tens[2]	39	BCD tens place output
bcd_tens[1]	41	BCD tens place output
bcd_tens[0]	18	BCD tens place output (LSB)
bcd_units[3]	37	BCD units place output (MSB)
bcd_units[2]	40	BCD units place output
bcd_units[1]	16	BCD units place output
bcd_units[0]	19	BCD units place output (LSB)
switch[7] (rst)	14	Reset signal input
switch[6]	12	Switch input (bit 6)
switch[5]	11	Switch input (bit 5)
switch[4]	9	Switch input (bit 4)
switch[3]	8	Switch input (bit 3)
switch[2]	6	Switch input (bit 2)
switch[1]	5	Switch input (bit 1)
switch[0]	4	Switch input (bit 0)
led[6]	33	LED indicator (MSB)
led[5]	31	LED indicator
led[4]	29	LED indicator
led[3]	28	LED indicator
led[2]	27	LED indicator
led[1]	26	LED indicator
led[0]	25	LED indicator (LSB)

Table 1. Pin Mapping of MAX3000A as done in software

3.2. Truth Table and Circuit Diagram

The truth table and Circuit Diagram below outlines the system behavior during various phases of the game based on the cycle counter value:

Note - all the data are taken when seed phrase is 10101, for other seed the value will be different.

Cycle Counter	Action	Display Output	LED Status
0-3	Generate random number (LFSR)	Random value	Reflect LFSR value (padded)
4	Clear display	0	OFF
5-10	Accept user input via switches	Switch value	OFF
11-12	Display correct answer	Sum modulo 100	LEDs ON (success or pattern)
13-14	Hold result display	Sum modulo 100	LEDs remain in previous state
15	Reset game	0	LEDs ON

Table 2. Truth Table for Game Phases

Step	LFSR State (Binary)	Decimal Equivalent	Feedback (bit4 XOR bit2)
0	10101	21	$1 \oplus 1 = 0$
1	01010	10	$0 \oplus 0 = 0$
2	10100	20	$1 \oplus 1 = 0$
3	01000	8	$0 \oplus 0 = 0$
4	10000	16	$1 \oplus 0 = 1$
5	00001	1	$0 \oplus 0 = 0$
6	00010	2	$0 \oplus 0 = 0$
7	00100	4	$0 \oplus 1 = 1$
8	01001	9	$0 \oplus 0 = 0$
9	10010	18	$1 \oplus 0 = 1$
10	00101	5	$0 \oplus 1 = 1$
11	01011	11	$0 \oplus 0 = 0$
12	10110	22	$1 \oplus 1 = 0$
13	01100	12	$0 \oplus 1 = 1$
14	11001	25	$1 \oplus 0 = 1$
15	10011	19	$1 \oplus 0 = 1$
16	00111	7	$0 \oplus 1 = 1$
17	01111	15	$0 \oplus 1 = 1$
18	11111	31	$1 \oplus 1 = 0$
19	11110	30	$1 \oplus 1 = 0$
20	11100	28	$1 \oplus 1 = 0$
21	11000	24	$1 \oplus 0 = 1$
22	10001	17	$1 \oplus 0 = 1$
23	00011	3	$0 \oplus 0 = 0$
24	00110	6	$0 \oplus 1 = 1$
25	01101	13	$0 \oplus 1 = 1$
26	11011	27	$1 \oplus 0 = 1$
27	10111	23	$1 \oplus 1 = 0$
28	01110	14	$0 \oplus 1 = 1$
29	11101	29	$1 \oplus 1 = 0$
30	11011	27	$1 \oplus 0 = 1$
31	10111	23	$1 \oplus 1 = 0$

Table 3. Truth table showing the complete cycle of the 5-bit LFSR starting from 10101 as seed.

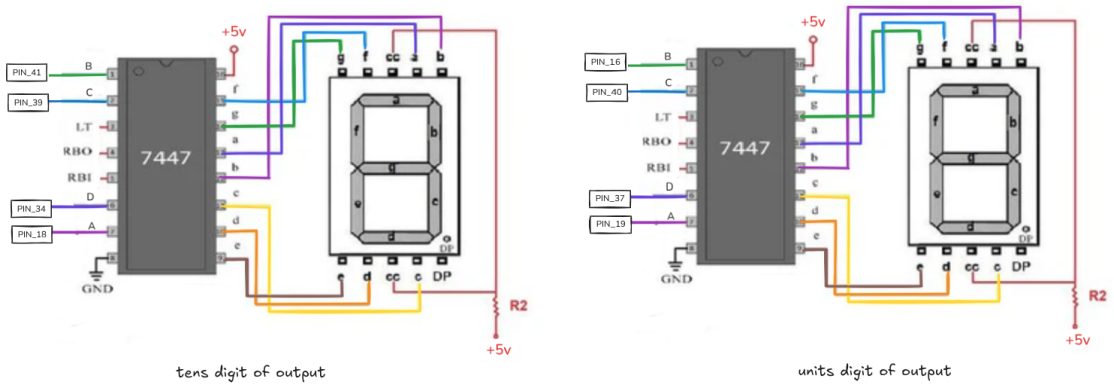
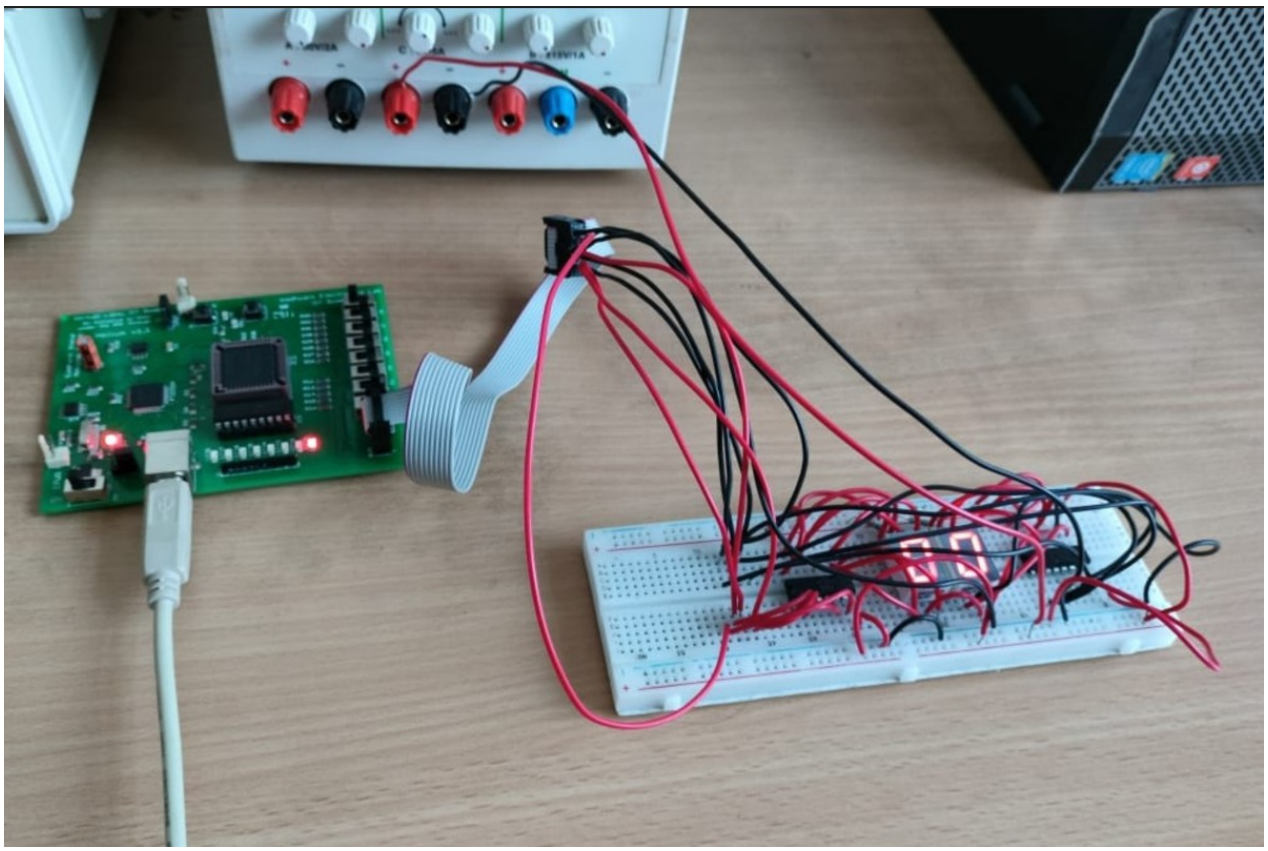
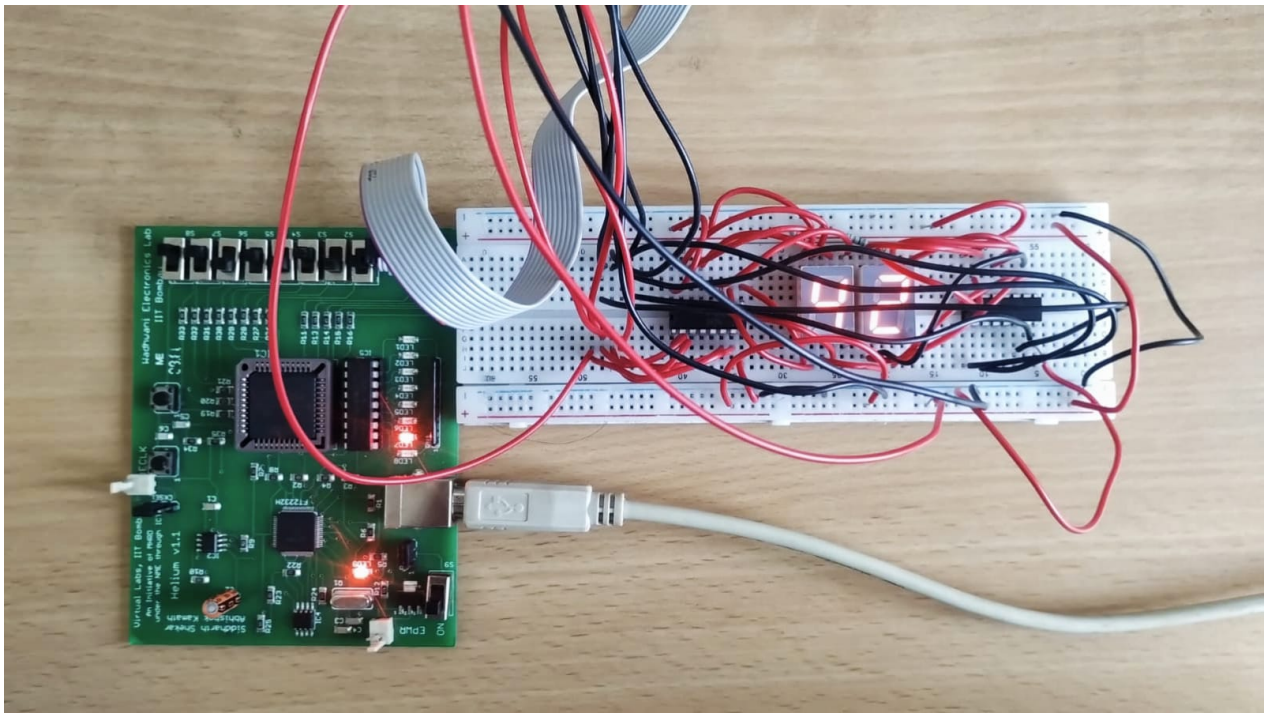


Figure 2. CPLD Maths Game Circuit Diagram and IO Mapping



(a) Circuit connection CPLD and Breadboard used in the project



(b) Setup during Demonstration of project.

Figure 3. CPLD demonstration

4. Finite State Machine (FSM) Design

The core control logic of the system is implemented using a synchronous Finite State Machine (FSM), which sequences through distinct operational phases of the arithmetic game. The FSM operates based on a classical Mealy-style structure that includes both present-state and next-state logic, as com-

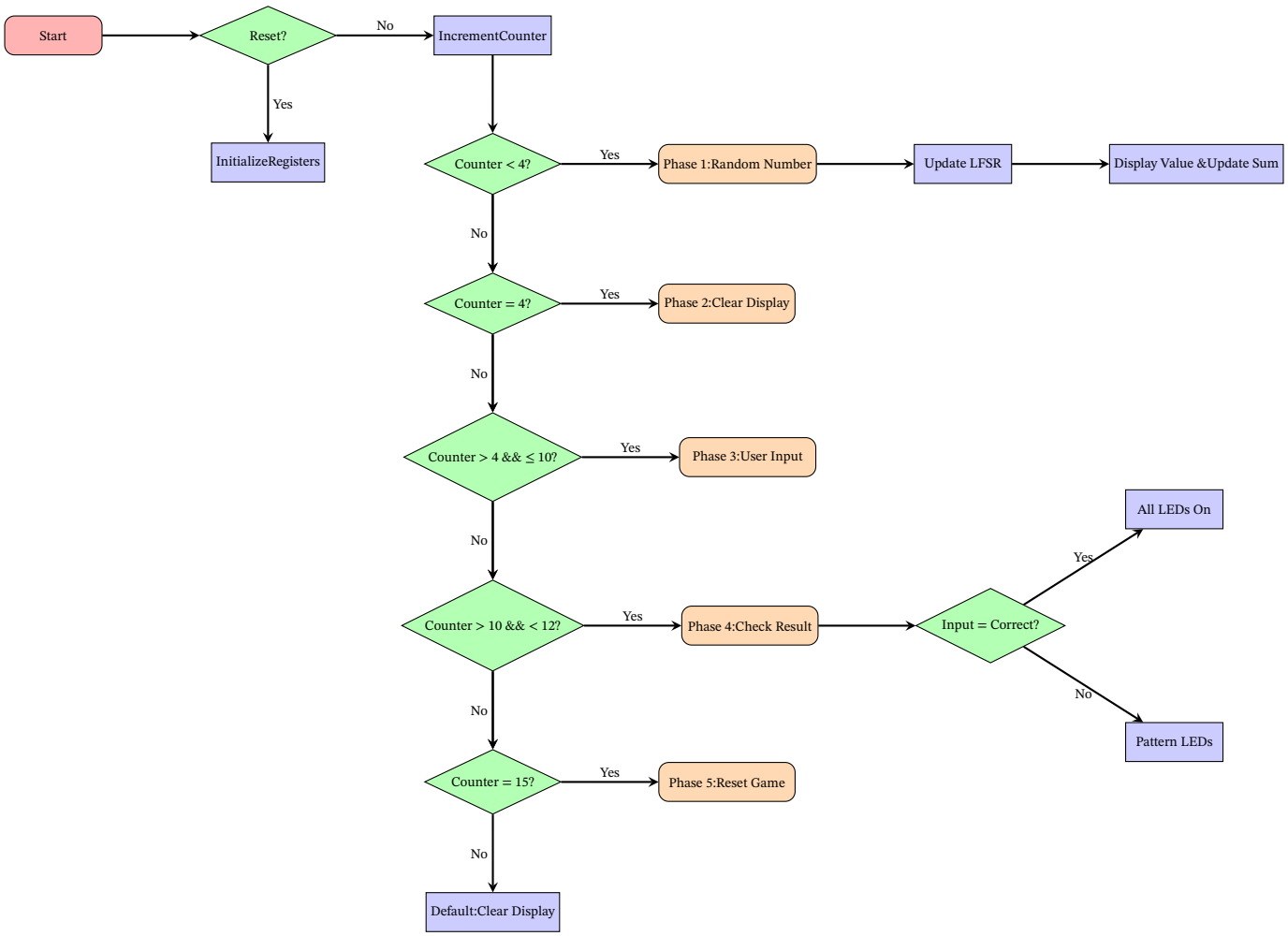


Figure 4. Control flow diagram for the CPLD Maths Game implementation

monly recommended in structured Verilog designs.

The FSM transitions through the following states:

- **IDLE** – Initial state; waits for system reset or start condition.
- **GENERATE** – Activates a 5-bit Linear Feedback Shift Register (LFSR) to generate a sequence of pseudo-random numbers. The LFSR, being a maximal-length feedback register, cycles through $2^5-1 = 31$ unique non-zero states **nandland_lfsr**.
- **DISPLAY** – Sequentially outputs generated numbers to the 7-segment display. Internally, a running sum is updated synchronously.
- **INPUT** – Accepts user input through DIP switches; values are latched using synchronous sampling techniques.
- **CHECK** – Compares the stored sum with the player's input. Non-blocking assignments ensure correct state progression.
- **FEEDBACK** – Drives LED outputs to indicate correctness and resets the FSM after a brief delay.

All transitions are driven by a 1Hz system clock. The FSM design adheres to synchronous logic principles and uses non-blocking assignments ('<=') to avoid race conditions. Debouncing is handled externally or assumed negligible due to clock throttling.

5. Linear Feedback Shift Register (LFSR) Logic

In this project, a 5-bit Linear Feedback Shift Register (LFSR) is employed as a hardware-based pseudo-random number generator (PRNG) to introduce non-determinism into the arithmetic memory game. The LFSR operates synchronously with the system clock and updates its state on every positive clock edge.

The implemented LFSR uses a primitive characteristic polynomial of the form:

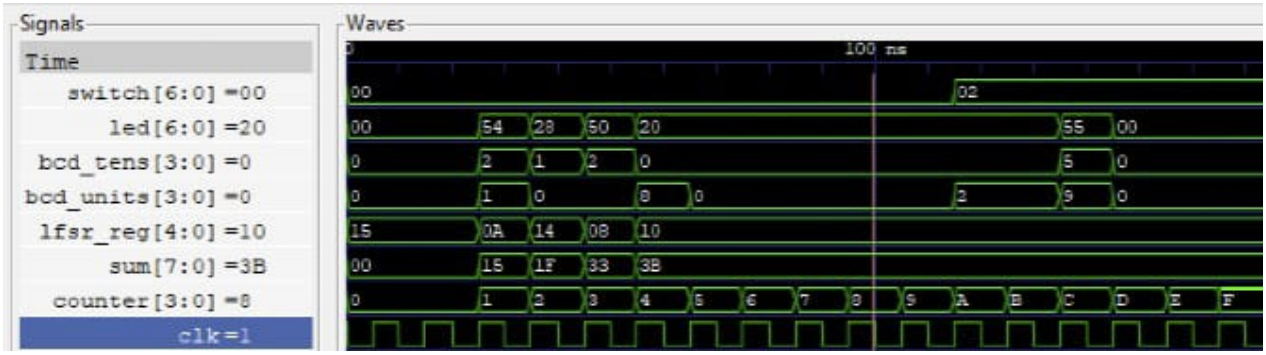
$$P(x) = x^5 + x^3 + 1 \tag{1}$$

This corresponds to feedback taps at bit positions 4 and 2 (0-indexed), generating a maximal-length sequence of $2^5-1 = 31$ non-zero states before repetition. The feedback logic for the next input bit is given by:

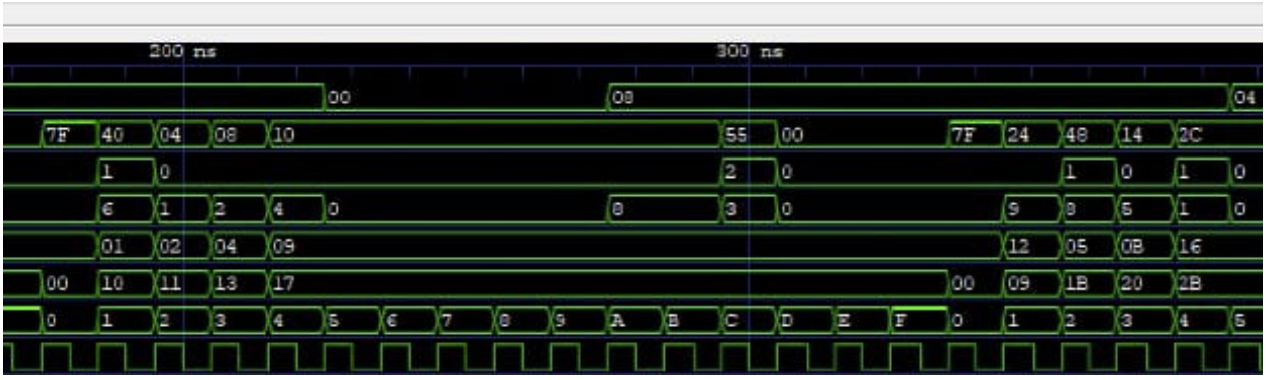
$$\texttt{next\_bit} = \texttt{lfsr}[4] \oplus \texttt{lfsr}[2] \tag{2}$$

At each clock cycle, the current LFSR state undergoes a right-shift operation, with `next_bit` injected into the most significant bit (MSB). The resulting sequence ensures that the game remains challenging and memory-driven.

The design is fully synchronous and avoids the use of blocking assignments, preserving timing determinism across state updates. The LFSR is reset asynchronously to a non-zero seed value. This implementation ensures compliance with standard digital PRNG design principles.



(a) Testbench waveform output (Part 1).



(b) Testbench waveform output (Part 2).

Figure 5. Combined simulation waveforms showing the DUT behavior over two capture windows.

6. Test Bench Waveforms

```
`timescale 1ns / 1ps

module tb_gmp;
    reg clk, rst;
    reg [6:0] switch;
    wire o_clk;
    wire [6:0] led;
    wire [3:0] bcd_tens, bcd_units;

    // DUT
    gmp uut (
        .clk(clk),
        .rst(rst),
        .o_clk(o_clk),
        .led(led),
        .switch(switch),
        .bcd_tens(bcd_tens),
        .bcd_units(bcd_units)
    );

    // Clock generation: 10ns period => 100MHz
    always #5 clk = ~clk;

    integer cycle;
    reg [7:0] correct_sum;
    integer correct;

    initial begin
        $dumpfile("gmp_tb.vcd");
        $dumpvars(0, tb_gmp);

        clk = 0;
        rst = 1;
        switch = 0;
        cycle = 0;
        correct_sum = 0;
        correct = 0;

        #20 rst = 0; // Deassert reset after some time

        // Run for 4 full rounds (16 cycles per round)
        repeat (64) begin
            @(posedge clk);
            cycle = cycle + 1;

            // Detect phase for user input
            if (cycle % 16 == 5) begin
                // Phase 3 begins: we already accumulated LFSR output
                correct_sum = 0;
            end

            // Track sum of LFSR-generated values (cycles 03 of each round)
            if (cycle % 16 < 4) begin
                correct_sum = correct_sum + uut.lfsr_reg;
            end

            // Inject switch value in input phase
            if ((cycle % 16) >= 5 && (cycle % 16) <= 10) begin
```

```
                // 75% of the time, input correct answer
                if ($urandom_range(0, 3) < 3) begin
                    switch = correct_sum % 100;
                    correct = 1;
                end else begin
                    switch = (correct_sum + $urandom_range(1, 10)) % 100;
                    correct = 0;
                end
            end

            // Optional debug output
            if ((cycle % 16) == 12) begin
                $display("Cycle: %0d | Expected: %0d | Input: %0d | BCD: %0d%0d | %s",
                         cycle,
                         correct_sum % 100,
                         switch,
                         bcd_tens,
                         bcd_units,
                         (switch == correct_sum % 100) ? "Correct" : "Wrong");
            end
        end

        $finish;
    end
endmodule
```

Code 1. Verilog code

7. Verilog codes

```
module gmp (
    input wire clk,    // PIN_43
    input wire rst,    // PIN_14
    output wire o_clk, // PIN_24
    output reg [6:0] led,
    input wire [6:0] switch,
    output wire [3:0] bcd_tens,
    output wire [3:0] bcd_units
);
    assign o_clk = clk; // for debugging purposes

    reg [7:0] current_output;
    reg [4:0] lfsr_reg;
    reg [3:0] counter;
    reg [7:0] sum;

    // Instantiate BCD converter for 7-segment display
    binary_to_bcd bcd_inst (
        .binary_in(current_output),
        .tens(bcd_tens),
        .units(bcd_units)
    );

    // Each cycle is 1sec, due to 1Hz frequency
    // Phase 1: Random number generation (cycles 0-3)
    // Phase 2: Display clearing (cycle 4)
    // Phase 3: User input (cycles 5-10)
    // Phase 4: Result checking (cycles 11-14)
    // Phase 5: Game reset (cycle 15)
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            current_output <= 0;
            lfsr_reg <= 5'b10101;
            led <= 7'b0000000;
            counter <= 0;
            sum <= 0;
        end else begin
            counter <= counter + 1; // Increment cycle counter

            if (counter < 4'd4) begin
                lfsr_reg <= {lfsr_reg[3:0], lfsr_reg[4] ^ lfsr_reg[2]};
                current_output <= {3'b000, lfsr_reg};
                led <= {lfsr_reg, 2'b00};
                sum <= sum + {3'b000, lfsr_reg};
            end
            else if (counter == 4'd4) begin
                current_output <= 8'b00000000;
            end
            else if (counter > 4'd4 && counter <= 4'd10) begin
                current_output <= {1'b0, switch};
            end
            else if (counter > 4'd10 && counter < 4'd12) begin
                current_output <= (sum % 100);
                // sum checking
                if ({1'b0, switch} == (sum % 100)) begin
                    led <= 7'b1111111; // All LEDs
                end
                else begin
                    led <= 7'b1010101; // some diffent pattern
                end
            end
            else if (counter == 4'd15) begin
                led <= 7'b1111111;
                counter <= 0;
                current_output <= 0;
                sum <= 0;
            end
            else begin
                current_output <= 0;
                led <= 7'b0000000;
            end
        end
    end
endmodule

module binary_to_bcd (
    input  wire [7:0] binary_in,
    output wire [3:0] tens,
    output wire [3:0] units
);
    wire [7:0] clamped_input = (binary_in > 8'd99) ? 8'd99 : binary_in;
    assign tens = clamped_input / 10;
    assign units = clamped_input % 10;
endmodule
```

Code 2. Verilog code

8. Future Improvements

Several enhancements can be incorporated to expand the functionality and interactivity of the current system:

1. **Random Seed:** Currently, the LFSR uses a hardcoded seed (`10101`) for pseudo-random number generation. A future improvement could allow the user to input a 5-bit seed value through switches, making each game session more unpredictable and engaging. This feature was not implemented due to hardware resource constraints (63 out of 64 macrocells were already utilized).
2. **Difficulty Levels:** Adding levels of increasing difficulty, such as displaying more numbers or introducing subtraction/multiplication could make the game more challenging and educational.
3. **Score Display:** Enhancing the system to retain and display the player's score over multiple rounds would improve game continuity and user engagement.

9. Conclusion

This project demonstrates the application of Verilog HDL and CPLD-based digital design principles to build an interactive and educational arithmetic memory game. By combining modular FSM-based control logic, pseudo-random number generation through LFSR, and real-time user interaction using switches and displays, we were able to deliver a robust embedded system. The game not only tests a player's arithmetic skills but also offers immediate feedback, reinforcing user engagement.

Through this implementation, we deepened our understanding of digital logic, timing constraints, and hardware debugging using the MAX3000A platform. The project serves as a foundation for more sophisticated embedded applications involving gamified learning or educational electronics. Future iterations could explore porting the design to FPGAs or integrating soft-core processors for extended functionality.

10. Acknowledgements

We would like to express our sincere gratitude to our Teaching Assistant, Chithambara J, for their continuous support and insightful feedback throughout the semester. Their guidance was immensely helpful in enhancing our technical understanding and in fostering a deeper interest in digital system design and embedded applications.

——— Thank You ———