ELL201 Sem-2 2024-25 Project Report

CPLD Maths Game using Verilog

Arnav Singh[a], **Lakshya Bhatnagar**[b], **Mahesh Pareek**[c] y 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 1 Hz clock rate and guides players through phases of random number display, sum calculation, input validation, and feedback through LEDs. 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, 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.

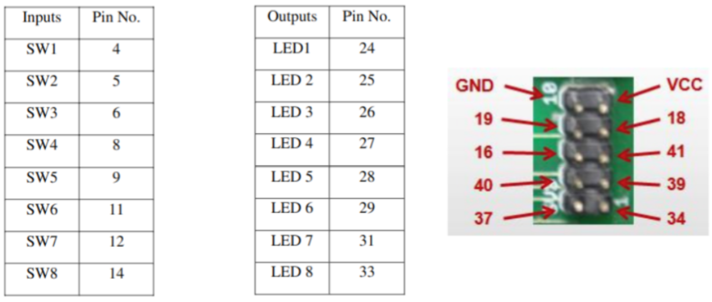
2. Motivation

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.

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

GND
VCC
19
18
16
41
40
39
37
34

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.

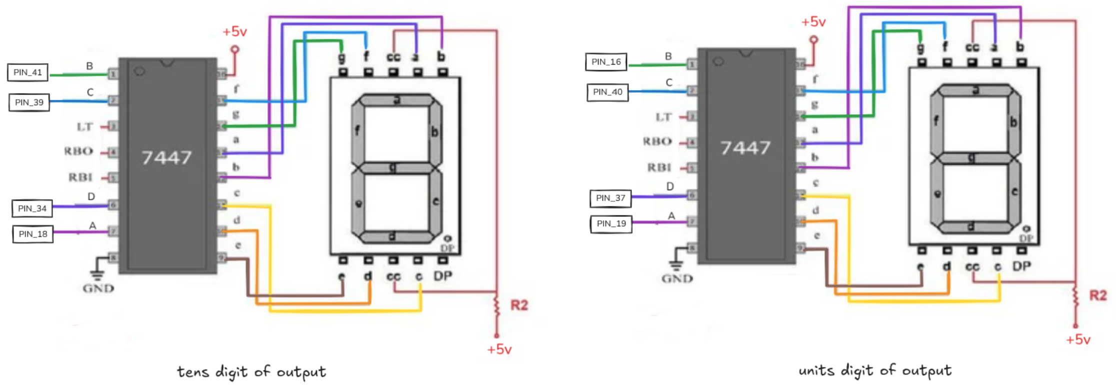


Figure 2. CPLD Maths Game Circuit Diagram and IO Mapping

4. Test Bench Waveforms

4.1. Tables

Table **??** shows an example table. The `\tabletext{}` is used to add notes to tables easily.

Table 4. Astronomical Object Data

Object	Distance (Light Years)
Alpha Centauri	4.37
Betelgeuse	642.5
Andromeda Galaxy	2.537 million

Note: The table contains data of some famous celestial objects.

5. Tau packages

5.1. Tauenvs

This template has its own environment package designed to enhance the presentation of the document. Among these custom environments

There are two environments which have a predefined title. These can be included by the commands

5.2. Taubabel

In previous versions, we included a package called, which have all the commands that automatically translate from English to Spanish when this language package is defined.

By default, tau displays its content in English. However, at the beginning of the document you will find a recommendation when writing in Spanish.

You may modify this package if you want to use other language than English or Spanish. This will make easier to translate the document without having to modify the class document.

6. Linear Feedback Shift Register, LFSR Logic

In this project, a 5-bit Linear Feedback Shift Register (LFSR) is used to generate pseudo-random numbers for the game. The LFSR advances based on a feedback polynomial that defines which bits are XORed together to create the next input.

The feedback logic can be described by Equation **??**:

$$\text{next\_bit} = \text{lfsr}[4] \oplus \text{lfsr}[2] \tag{1}$$

At every clock cycle, the LFSR shifts its bits to the right by one position, and the newly generated `next_bit` is inserted into the most significant bit (MSB) position. This mechanism ensures a repeating but seemingly random sequence of numbers, suitable for a simple memory-based game.
The feedback taps (bits 4 and 2) correspond to a primitive polynomial, ensuring maximal sequence length before repetition.

7. Verilog codes

```verilog
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 differnt 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
```

CódigoCode 1. Example of Matlab code.