1D Cellular Automaton Pattern Generator
ELL201 Project Report

ELL201
Table Number 20, Monday Batch
TA: Avantika Singh

April 29, 2025

Group Members

- Prince Ramani - 2023EE30755
- Yash Mathur - 2023AM11212
- Laksh Singhal - 2023MT61222
- Pranava Modi - 2023AM10200

Abstract

This project implements a programmable 1D cellular automaton (CA) on a CPLD board, demonstrating how simple local rules can generate complex patterns. The design incorporates both combinational and sequential circuit elements to create an interactive digital system that visually displays the evolving states of cellular automata. The implementation features an 8-cell 1D cellular automaton with binary states, programmable rules via DIP switches, and both automatic and manual operation modes. Despite its simplicity, the system demonstrates the emergence of complex patterns from simple rules—a fundamental concept in complex systems and computational theory.

Contents

1 Introduction

1.1 Background

Cellular automata (CA) are discrete mathematical models studied in computer science, mathematics, physics, and theoretical biology. They consist of a grid of cells, each with a finite number of states, that evolve over time according to a set of rules based on the states of neighboring cells.

1.2 Elementary Cellular Automata

This project implements a simplified version of Stephen Wolfram's elementary cellular automata, which consist of a one-dimensional array of cells with binary states (0 or 1). Despite their simple rules, these automata can produce complex and sometimes unpredictable patterns.

1.3 Project Objectives

The main objectives of this project are:

- To implement a functional 1D cellular automaton on CPLD hardware
- To demonstrate how complex patterns can emerge from simple rules
- To create an interactive system that allows experimentation with different rules
- To apply digital design principles in a practical, educational context

2 Theoretical Background

2.1 Definition of 1D Cellular Automata

A one-dimensional cellular automaton consists of:

- A line of cells, each in one of a finite number of states (in our case, binary: 0 or 1)
- A set of rules that determine the state of a cell in the next generation based on its current state and the states of its neighbors
- Boundary conditions (in our implementation, we use wrap-around boundaries)

2.2 Rule Encoding

In elementary cellular automata, the state of each cell depends on its current state and the states of its two immediate neighbors. With three binary inputs, there are $2^3 = 8$ possible neighborhood configurations. Each configuration maps to a binary output, resulting in $2^8 = 256$ possible rules.

A rule is encoded as an 8-bit binary number, where each bit position corresponds to a specific neighborhood pattern:

Rule Bit	Left	Center	Right	Next State
0	0	0	0	rule[0]
1	0	0	1	rule[1]
2	0	1	0	rule[2]
3	0	1	1	rule[3]
4	1	0	0	rule[4]
5	1	0	1	rule[5]
6	1	1	0	rule[6]
7	1	1	1	rule[7]

Table 1: Rule encoding for elementary cellular automata

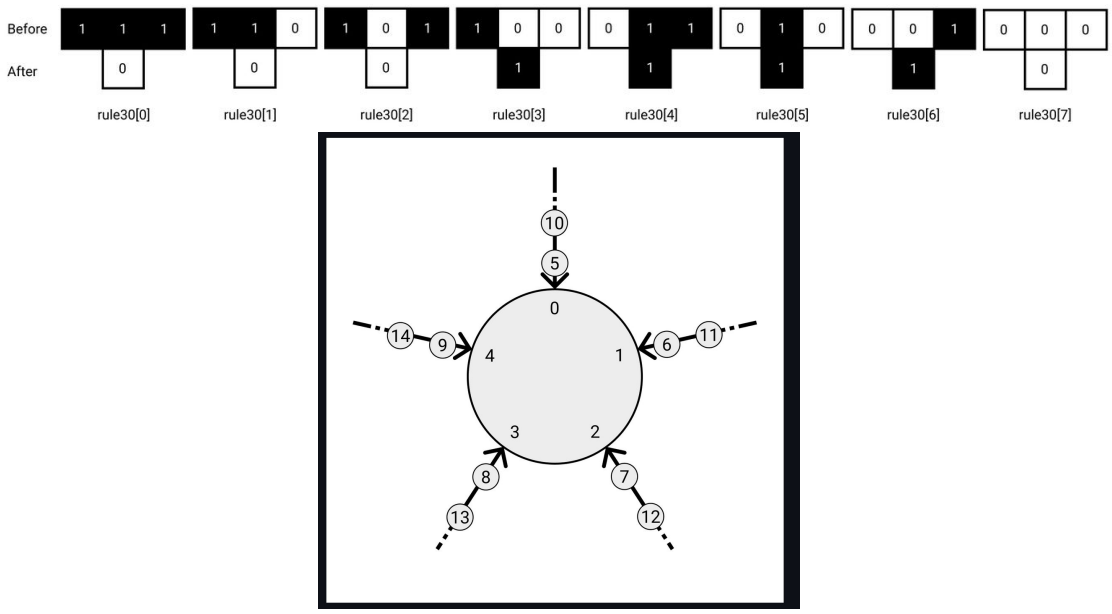

Figure 1: Rule encoding for Rule 30 and its wrap around property

2.3 Notable Rules

Several rules produce particularly interesting patterns:

- **Rule 30 (00011110)**: Creates chaotic, random-looking patterns
- **Rule 90 (01011010)**: Creates a Sierpinski triangle fractal pattern
- **Rule 110 (01101110)**: Proven to be computationally universal
- **Rule 94 (01011110)**: Creates repetitive symmetric patterns

3 Design Methodology

3.1 System Overview

The system consists of:

- An 8-cell 1D cellular automaton with binary states

- Programmable rules via 8 DIP switches
- LED output display for visualizing cell states
- User controls loading initial states, and stepping through evolution

3.2 Block Diagram

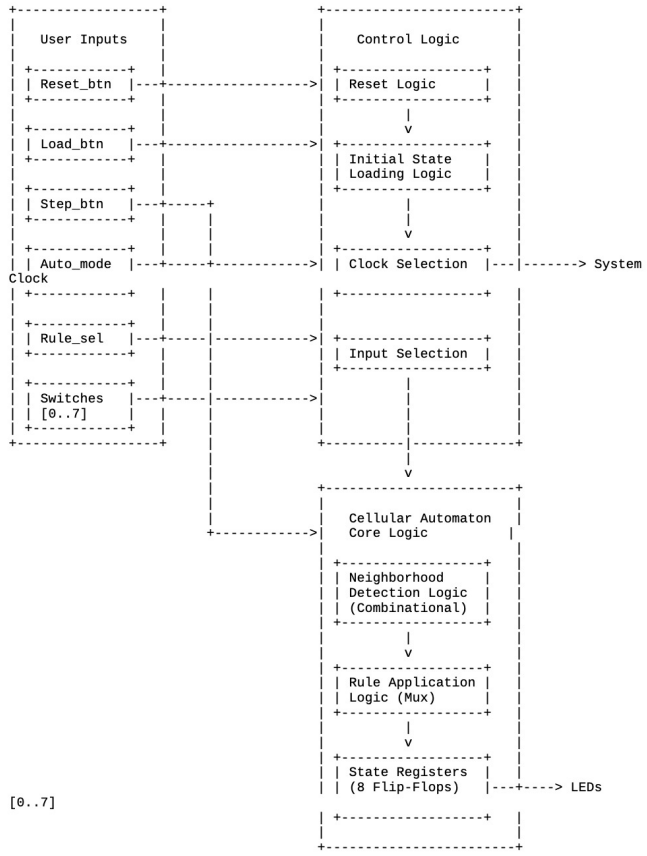

Figure 2: System block diagram

3.3 Circuit Design

The circuit comprises several key components:

3.3.1 User Input Interface

- Switches for rule/initial state input
- Local rules pre-defined in our code

3.3.2 Control Logic

- Input debouncing and synchronization
- Clock divider for automatic updates
- Mode selection logic

3.3.3 Cellular Automaton Engine

- Neighborhood detection logic (combinational)
- Rule application logic (combinational)
- State registers (sequential)

3.3.4 Output Display

- 8 LEDs driven directly by cell state registers

3.4 Detailed Cell Circuit

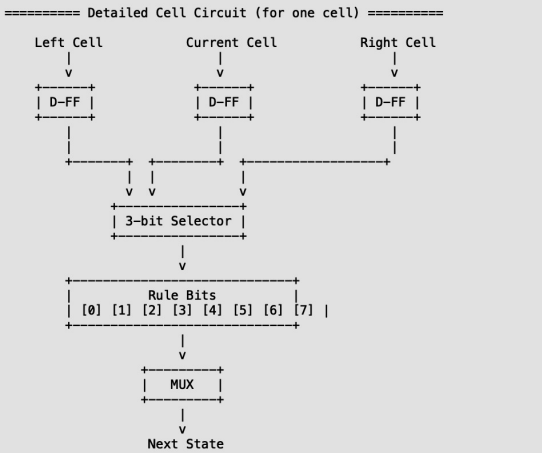

Figure 3: Detailed circuit for a single cell

4 Implementation

4.1 Hardware Components

- CPLD Board (e.g., Xilinx XC9572XL)
- 8 LEDs for cell state display
- DIP switches for initial state input
- Toggle switches for mode selection

4.2 Verilog Implementation

The implementation is divided into three main Verilog modules:

4.2.1 Cellular Automaton Module with the implementation on Rule30

```
module cellular_automaton(
    input wire clk,           // Onboard 1Hz clock (PIN 43)
    input wire [7:0] sw,      // 8-bit switch input (PIN
   4,5,6,8,9,11,12,14)
    output reg [7:0] led      // 8-bit LED output (PIN
   24,25,26,27,28,29,31,33)
);
    // Registers for storing states
    reg [7:0] current_state;
    reg [7:0] rule;
    reg initialized;          // Flag to track if initial pattern is
   loaded

    // Internal signals
    wire [7:0] next_state;
    wire any_switch_on;

    // Check if any switch is turned on
    assign any_switch_on = |sw;

    // Calculate next state for each cell based on rule
    genvar i;
    generate
        for (i = 0; i < 8; i = i + 1) begin: next_state_logic
            wire [2:0] neighborhood;
            assign neighborhood[2] = current_state[(i == 0) ? 7 : i-1];
        // Left neighbor (wrap around)
            assign neighborhood[1] = current_state[i];
         // Current cell
            assign neighborhood[0] = current_state[(i == 7) ? 0 : i+1];
         // Right neighbor (wrap around)

            // Apply rule based on neighborhood pattern
            assign next_state[i] = rule[neighborhood];
        end
    endgenerate

    // State control logic
    always @(posedge clk) begin
        if (!initialized && any_switch_on) begin
            // First time a switch is on, load the initial state
            current_state <= sw;
            rule <= 8'b00011110;  // Rule 30
            initialized <= 1;     // Mark as initialized
        end
        else if (initialized) begin
            // Auto-advance to next generation on each clock cycle
            current_state <= next_state;
        end
    end

    // Output current state to LEDs
    always @(*) begin
        led = current_state;
    end

    // Initialize state
    initial begin
```

```
        current_state = 8'b00000000;
        rule = 8'b00011110;       // Rule 30
        initialized = 0;
    end
endmodule
```

Listing 1: cellular_automaton.v

4.2.2 Testbench for rule 94

```
`timescale 1s

module tb_cellular_automaton_rule94();
    reg clk;
    reg reset;
    reg [7:0] init_state;
    reg [7:0] rule;
    reg load_init;
    wire [7:0] cells;

    // Instantiate cellular automaton
    cellular_automaton uut (
        .clk(clk),
        .reset(reset),
        .init_state(init_state),
        .rule(rule),
        .load_init(load_init),
        .cells(cells)
    );

    // Clock generation
    initial begin
        clk = 0;
        forever #10 clk = ~clk; // 50MHz clock (20ns period)
    end

    // Stimulus
    initial begin
        // Initial conditions
        reset = 1;
        init_state = 8'b00010000; // Start with single center cell
        rule = 8'b01011110;       // Rule 94
        load_init = 0;

        #100;
        reset = 0;

        #20;
        load_init = 1;
        #20;
        load_init = 0;

        // Run for 32 clock cycles
        repeat(32) begin
            @(posedge clk);
            #1;
            $display("%b", cells);
```

```
        end

        $finish;
    end
endmodule
```

Listing 2: tb_94.v

4.2.3 Test bench for rule 110

```
`timescale 1s

module tb_cellular_automaton_rule110();
    reg clk;
    reg reset;
    reg [7:0] init_state;
    reg [7:0] rule;
    reg load_init;
    wire [7:0] cells;

    cellular_automaton uut (
        .clk(clk),
        .reset(reset),
        .init_state(init_state),
        .rule(rule),
        .load_init(load_init),
        .cells(cells)
    );

    initial begin
        clk = 0;
        forever #10 clk = ~clk;
    end

    initial begin
        reset = 1;
        init_state = 8'b00010000; // Single cell active in the middle
        rule = 8'b01101110;       // Rule 110
        load_init = 0;

        #100;
        reset = 0;

        #20;
        load_init = 1;
        #20;
        load_init = 0;

        repeat(32) begin
            @(posedge clk);
            #1;
            $display("%b", cells);
        end
    end
endmodule
```

Listing 3: tb_110.v

5 Simulation Results

5.1 Testbench Setup

The testbench simulates the cellular automaton with two different rules:

- Rule 30 (00011110) with a single active cell in the middle
- Rule 94 (01011110) with an symmetric pattern

5.2 Waveform Analysis

Figure 4: Simulation waveform showing evolution of cell states in rule 30

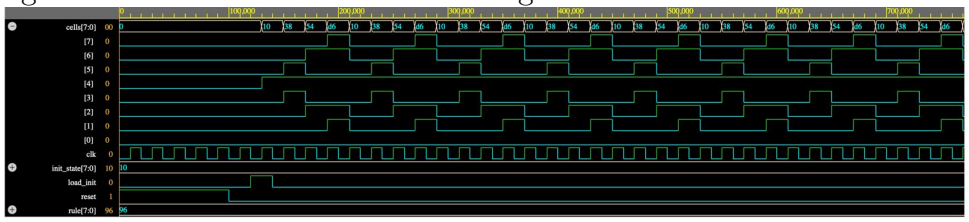

Figure 5: Simulation waveform showing evolution of cell states in rule 94

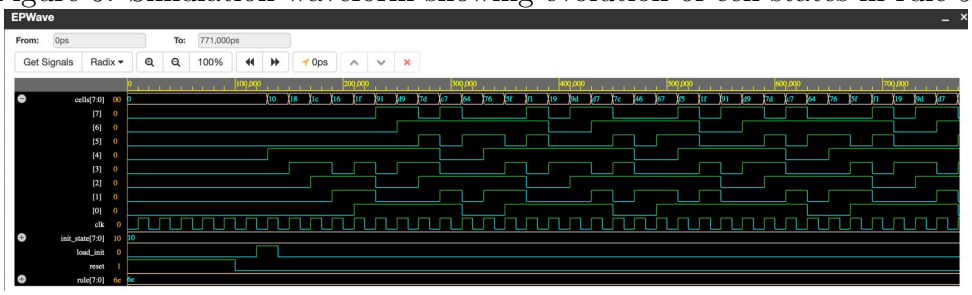

Figure 6: Simulation waveform showing evolution of cell states in rule 110

5.3 Verification

The simulation results were verified by:

- Comparing the evolution patterns with known patterns for Rules 30 and 94 and 110
- Checking boundary conditions (wrap-around behavior)
- Verifying proper state transitions based on the rule definitions

6 Testing

6.1 Testing Methodology

The hardware implementation was tested by:

- Verifying basic functionality (load, step)
- Testing different rules and initial states
- Verifying pattern mode operation with preset rules

6.2 Sample Results

6.2.1 Rule 30 (00011110) - Chaotic Pattern

Initial state: 00010000

```
Step 1:       00111000
Step 2:       01001100
Step 3:       11110110
Step 4:       00010010
Step 5:       00111111
Step 6:       11000001
Step 7:       01100010
Step 8:       10110111
```

6.2.2 Rule 110 (01101110) - Computationally Universal Pattern

Initial state: 00010000

```
Step 1:       00111000
Step 2:       01001100
Step 3:       11101110
Step 4:       10111001
Step 5:       11001111
Step 6:       01110001
Step 7:       11001011
Step 8:       01111010
```

6.2.3 Rule 94 (01011110) - Symmetric Pattern

Initial state: 00010000

```
Step 1:       00111000
Step 2:       01000100
Step 3:       11101110
Step 4:       10111010
Step 5:       11000110
Step 6:       01101001
Step 7:       11111111
Step 8:       10000000
```

7 Analysis and Discussion

7.1 Pattern Analysis

- **Rule 30**: Produces chaotic, seemingly random patterns despite deterministic rules
- **Rule 110**: Shows complex behavior that has been proven to be computationally universal
- **Rule 94**: Exhibits symmetric patterns with predictable repetition

7.2 Hardware Performance

The CPLD implementation successfully demonstrated:

- Efficient use of hardware resources
- Reliable operation at the designed clock frequencies
- Proper handling of user inputs and mode switching
- Clear visual representation of cellular automaton states

7.3 Challenges and Solutions

- **Challenge**: Implementing wrap-around boundary conditions
- **Solution**: Used conditional assignments in the neighborhood detection logic
- **Challenge**: Creating a user-friendly interface with limited I/O
- **Solution**: Implemented dual-purpose switches and mode selection
- **Challenge**: Achieving appropriate update speeds for visualization
- **Solution**: Implemented adjustable clock divider for different animation speeds

8 Educational Value

This project demonstrates several important concepts in digital design:

- **Combinational Logic**: The rule application logic that determines each cell's next state
- **Sequential Logic**: The state registers that store cell states and update synchronously
- **Clock Management**: Dividing the system clock for different operational modes
- **User Interface Design**: Creating intuitive controls for system interaction
- **Complex System Behavior**: Showing how simple rules generate complex patterns

9 Conclusion

9.1 Project Achievements

The 1D Cellular Automaton Pattern Generator successfully:

- Implemented a functional 8-cell 1D cellular automaton on CPLD hardware
- Demonstrated the generation of complex patterns from simple local rules
- Provided an interactive platform for experimenting with different rules and initial states
- Showcased the principles of digital design in a practical application

9.2 Future Enhancements

Potential extensions to this project include:

- Expanding to a 2D cellular automaton (like Conway's Game of Life)
- Adding memory to store and recall favorite rules
- Implementing a display to show the evolution history (multiple generations at once)
- Adding a serial interface to connect with a computer for more extensive visualization
- Implementing a learning mechanism to discover rules that generate specific patterns

10 References

1. Weisstein, E. W. "Elementary Cellular Automaton." From MathWorld–A Wolfram Web Resource.
2. Visit wiki for rule 110 `https://en.wikipedia.org/wiki/Rule_110`
3. Cook, M. (2004). "Universality in Elementary Cellular Automata." Complex Systems, 15(1), 1-40.
4. Weisstein, E. W. "Elementary Cellular Automaton." From MathWorld–A Wolfram Web Resource.
5. Ilachinski, A. (2001). Cellular Automata: A Discrete Universe. World Scientific.

A Pin Assignments

B Complete Truth Table for Rule Application

Signal Name	CPLD Pin	Description
clk_1Hz	Pin 43	System clock input
sw[0]	Pin 4	Switch input bit 0
sw[1]	Pin 5	Switch input bit 1
sw[2]	Pin 6	Switch input bit 2
sw[3]	Pin 8	Switch input bit 3
sw[4]	Pin 9	Switch input bit 4
sw[5]	Pin 10	Switch input bit 5
sw[6]	Pin 11	Switch input bit 6
sw[7]	Pin 13	Switch input bit 7
leds[0]	Pin 24	LED output bit 0
leds[1]	Pin 25	LED output bit 1
leds[2]	Pin 26	LED output bit 2
leds[3]	Pin 27	LED output bit 3
leds[4]	Pin 28	LED output bit 4
leds[5]	Pin 29	LED output bit 5
leds[6]	Pin 31	LED output bit 6
leds[7]	Pin 33	LED output bit 7

Table 2: CPLD pin assignments

Left	Center	Right	Rule 30	Rule 94	Rule 110	Rule 60	Binary	Decimal
0	0	0	0	0	0	0	000	0
0	0	1	1	1	1	0	001	1
0	1	0	1	1	1	1	010	2
0	1	1	1	1	1	1	011	3
1	0	0	1	1	0	1	100	4
1	0	1	0	0	1	1	101	5
1	1	0	0	1	1	0	110	6
1	1	1	0	0	0	0	111	7

Table 3: Complete truth table for selected rules