Name: Dhirajzen Bagawath Geetha Kumaravel
NetID: db5309

# Report on Verilog Model Generation using ChipChat in Colab

## Introduction

This report documents my experiments with ChipChat, inspired by the paper 'Chip-Chat: Challenges and Opportunities in Conversational Hardware Design'. The goal was to generate Verilog models for 4 different example specifications using conversational prompting in ChipChat Colab script. For each example, I recorded the prompts, the outputs generated, the errors encountered, and the final working designs.

## Tools and Settings

- **Tools:** Icarus Verilog (iverilog), Google Colab, ChipChat (GenAI), OpenRouter API
- **LLM model:** openrouter/sonoma-sky-alpha (Python SDK via `openai` client with `base_url="https://openrouter.ai/api/v1"`)
- **Date used:** 09/17/2025
- **Interface:** Google Colab notebook
- **Parameters controlled:** max_tokens = 1700000; other parameters (temperature, top_p, system prompt) left at API defaults

## Prompting Strategy:

- Start with a **natural language spec** of the required module (inputs, outputs, behavior).
- Provide **constraints** (no SystemVerilog constructs, synthesizable).
- If an error occurred (compilation or testbench mismatch), append the **error log + "resolve this error"** to the next prompt.

## Iteration Workflow:

(i) Generate initial design.
(ii) Compile and note errors/failures.
(iii) Feedback errors directly into the prompt.
(iv) Repeat until all test cases passed.

Name: Dhirajzen Bagawath Geetha Kumaravel
NetID: db5309

## Table of Contents

Name: Dhirajzen Bagawath Geetha Kumaravel
NetID: db5309

## Example 1

**binary_to_bcd_converter:**

- Inputs:
  - Binary input (5-bits)
- Outputs:
  - BCD (8-bits: 4-bits for the 10's place and 4-bits for the 1's place)

### Prompts and Outputs

**Prompt 1:**

```
Create a Verilog model binary_to_bcd_converter for a binary to binary-coded-decimal
converter. It must meet the following specifications:
    - Inputs:
        - Binary input (5-bits)
    - Outputs:
        - BCD (8-bits: 4-bits for the 10's place and 4-bits for the 1's place)

How would I write a design that meets these specifications?
```

LLM Design Output 1:

```
module binary_to_bcd_converter (
    input  [4:0] binary_input,  // 5-bit binary input (0-31)
    output [7:0] bcd_output     // 8-bit BCD output: [7:4]=tens, [3:0]=units
);

    // Combinational logic: Compute tens and units using arithmetic
    assign bcd_output = {binary_input / 5'd10, binary_input % 5'd10};

endmodule
```

Error/Observation:

```
Testing Binary-to-BCD Converter...
VCD info: dumpfile my_design.vcd opened for output.
Error: Test case 10 failed. Expected BCD: 8'b10000, Got: 8'b100000
```

**Prompt 2:**

Appended the generated code along with the error and added the line – "`resolve this error by modifying the design`." to the end.

LLM Design Output 2:

```
module binary_to_bcd_converter (
    input  [4:0] binary_input,  // 5-bit binary input (0-31)
    output [7:0] bcd_output     // 8-bit BCD output: [7:4]=tens, [3:0]=units
);
```

Name: Dhirajzen Bagawath Geetha Kumaravel
NetID: db5309

```
    // Combinational logic: Compute tens and units using arithmetic with proper width slicing
    assign bcd_output = { (binary_input / 5'd10)[3:0], (binary_input % 5'd10)[3:0] };

endmodule
```

Error/Observation:

```
binary_to_bcd.v:8: syntax error
binary_to_bcd.v:8: error: syntax error in continuous assignment
```

**Prompt 3:**

Appended the generated code along with the error and added the line – "`resolve this error by modifying the design`." to the end.

Final Correct Design:

```
module binary_to_bcd_converter (
    input  [4:0] binary_input,  // 5-bit binary input (0-31)
    output [7:0] bcd_output     // 8-bit BCD output: [7:4]=tens, [3:0]=units
);

    // Intermediate wires to hold division and modulo results
    wire [4:0] tens_wire = binary_input / 5'd10;
    wire [4:0] units_wire = binary_input % 5'd10;

    // Combinational logic: Concatenate lower 4 bits of each for proper BCD
    assign bcd_output = {tens_wire[3:0], units_wire[3:0]};

endmodule
```

Final Correct Output:

```
Testing Binary-to-BCD Converter...
VCD info: dumpfile my_design.vcd opened for output.
All test cases passed!
```

## Observations

- The model required **three iterations** to converge on a correct design. Initially, it produced a logically correct structure but failed due to improper width handling in the concatenation, which caused incorrect BCD output for certain inputs (e.g., decimal 10).
- The second attempt introduced slicing but resulted in a **syntax error** because bit-slicing was applied directly on arithmetic expressions inside a continuous assignment.
- The issue was resolved in the final iteration by introducing **intermediate wires** (tens_wire and units_wire) and then concatenating their lower 4 bits to form the final BCD output.

Name: Dhirajzen Bagawath Geetha Kumaravel
NetID: db5309

## Example 2

**Sequence Detector:**

-input
- Clock
- Active-low reset
- Data (3 bits)

-Outputs:
- Sequence found Inputs:

While enabled, it should detect the following sequence of binary input values:
- 0b001
- 0b101
- 0b110
- 0b000
- 0b110
- 0b110
- 0b011
- 0b101

## Prompts and Outputs

**Prompt 1:**

```
I am trying to create a Verilog model for a sequence detector. It must meet the following
specifications:
    -   - Clock
        - Active-low reset
        - Data (3 bits)
    - Outputs:
        - Sequence found Inputs:


While enabled, it should detect the following sequence of binary input values:
    - 0b001
    - 0b101
    - 0b110
    - 0b000
    - 0b110
    - 0b110
    - 0b011
    - 0b101

How would I write a design that meets these specifications?
```

LLM Design Output 1:

```
module sequence_detector (
    input wire clk,
    input wire reset_n,      // Active-low synchronous reset
    input wire [2:0] data,   // 3-bit input data
```

Name: Dhirajzen Bagawath Geetha Kumaravel
NetID: db5309

```
    output reg seq_found     // Output: high for one clock when sequence is detected
);

    // State register: 3 bits sufficient for states 0-7
    reg [2:0] state;

    // Define the sequence symbols for reference (not synthesizable, for clarity)
    // seq[1] = 3'b001
    // seq[2] = 3'b101
    // seq[3] = 3'b110
    // seq[4] = 3'b000
    // seq[5] = 3'b110
    // seq[6] = 3'b110
    // seq[7] = 3'b011
    // seq[8] = 3'b101

    always @(posedge clk) begin
        if (!reset_n) begin
            state <= 3'b000;  // State 0
            seq_found <= 1'b0;
        end else begin
            seq_found <= 1'b0;  // Default: deassert output
            case (state)
                3'b000: begin  // State 0: No match
                    if (data == 3'b001) begin
                        state <= 3'b001;  // Go to state 1
                    end else begin
                        state <= 3'b000;  // Stay in state 0
                    end
                end
                3'b001: begin  // State 1: Matched seq[1]
                    if (data == 3'b101) begin
                        state <= 3'b010;  // Go to state 2
                    end else if (data == 3'b001) begin
                        state <= 3'b001;  // Restart with new match to seq[1]
                    end else begin
                        state <= 3'b000;  // Reset to state 0
                    end
                end
                3'b010: begin  // State 2: Matched seq[1..2]
                    if (data == 3'b110) begin
                        state <= 3'b011;  // Go to state 3
                    end else if (data == 3'b001) begin
                        state <= 3'b001;  // Restart with new match to seq[1]
                    end else begin
                        state <= 3'b000;  // Reset to state 0
                    end
```

Name: Dhirajzen Bagawath Geetha Kumaravel
NetID: db5309

```
                end
                3'b011: begin  // State 3: Matched seq[1..3]
                    if (data == 3'b000) begin
                        state <= 3'b100;  // Go to state 4
                    end else if (data == 3'b001) begin
                        state <= 3'b001;  // Restart with new match to seq[1]
                    end else begin
                        state <= 3'b000;  // Reset to state 0
                    end
                end
                3'b100: begin  // State 4: Matched seq[1..4]
                    if (data == 3'b110) begin
                        state <= 3'b101;  // Go to state 5
                    end else if (data == 3'b001) begin
                        state <= 3'b001;  // Restart with new match to seq[1]
                    end else begin
                        state <= 3'b000;  // Reset to state 0
                    end
                end
                3'b101: begin  // State 5: Matched seq[1..5]
                    if (data == 3'b110) begin
                        state <= 3'b110;  // Go to state 6
                    end else if (data == 3'b001) begin
                        state <= 3'b001;  // Restart with new match to seq[1]
                    end else begin
                        state <= 3'b000;  // Reset to state 0
                    end
                end
                3'b110: begin  // State 6: Matched seq[1..6]
                    if (data == 3'b011) begin
                        state <= 3'b111;  // Go to state 7
                    end else if (data == 3'b001) begin
                        state <= 3'b001;  // Restart with new match to seq[1]
                    end else begin
                        state <= 3'b000;  // Reset to state 0
                    end
                end
                3'b111: begin  // State 7: Matched seq[1..7], waiting for seq[8]
                    if (data == 3'b101) begin
                        seq_found <= 1'b1;  // Sequence detected!
                        state <= 3'b000;    // seq[8] != seq[1], so reset to state 0
                    end else if (data == 3'b001) begin
                        state <= 3'b001;  // Restart with new match to seq[1]
                    end else begin
                        state <= 3'b000;  // Reset to state 0
                    end
                end
```

Name: Dhirajzen Bagawath Geetha Kumaravel
NetID: db5309

```
                default: begin
                    state <= 3'b000;  // Fallback
                end
            endcase
        end
    end

endmodule
```

Error/Observation:

```
sequence_detector_tb.v:10: error: port ``sequence_found'' is not a port of dut.
1 error(s) during elaboration.
```

**Prompt 2:**

Appended the generated code along with the error and added the line – "`resolve this error by modifying the design`." to the end.

LLM Design Output 2:

```
module sequence_detector (
    input wire clk,
    input wire reset_n,      // Active-low synchronous reset
    input wire [2:0] data,   // 3-bit input data
    output reg sequence_found     // Output: high for one clock when sequence is detected
);

    // State register: 3 bits sufficient for states 0-7
    reg [2:0] state;

    // Define the sequence symbols for reference (not synthesizable, for clarity)
    // seq[1] = 3'b001
    // seq[2] = 3'b101
    // seq[3] = 3'b110
    // seq[4] = 3'b000
    // seq[5] = 3'b110
    // seq[6] = 3'b110
    // seq[7] = 3'b011
    // seq[8] = 3'b101

    always @(posedge clk) begin
        if (!reset_n) begin
            state <= 3'b000;  // State 0
            sequence_found <= 1'b0;
```

Name: Dhirajzen Bagawath Geetha Kumaravel
NetID: db5309

```
        end else begin
            sequence_found <= 1'b0;  // Default: deassert output
            case (state)
                3'b000: begin  // State 0: No match
                    if (data == 3'b001) begin
                        state <= 3'b001;  // Go to state 1
                    end else begin
                        state <= 3'b000;  // Stay in state 0
                    end
                end
                3'b001: begin  // State 1: Matched seq[1]
                    if (data == 3'b101) begin
                        state <= 3'b010;  // Go to state 2
                    end else if (data == 3'b001) begin
                        state <= 3'b001;  // Restart with new match to seq[1]
                    end else begin
                        state <= 3'b000;  // Reset to state 0
                    end
                end
                3'b010: begin  // State 2: Matched seq[1..2]
                    if (data == 3'b110) begin
                        state <= 3'b011;  // Go to state 3
                    end else if (data == 3'b001) begin
                        state <= 3'b001;  // Restart with new match to seq[1]
                    end else begin
                        state <= 3'b000;  // Reset to state 0
                    end
                end
                3'b011: begin  // State 3: Matched seq[1..3]
                    if (data == 3'b000) begin
                        state <= 3'b100;  // Go to state 4
                    end else if (data == 3'b001) begin
                        state <= 3'b001;  // Restart with new match to seq[1]
                    end else begin
                        state <= 3'b000;  // Reset to state 0
                    end
                end
                3'b100: begin  // State 4: Matched seq[1..4]
                    if (data == 3'b110) begin
                        state <= 3'b101;  // Go to state 5
                    end else if (data == 3'b001) begin
                        state <= 3'b001;  // Restart with new match to seq[1]
                    end else begin
                        state <= 3'b000;  // Reset to state 0
                    end
                end
                3'b101: begin  // State 5: Matched seq[1..5]
```

Name: Dhirajzen Bagawath Geetha Kumaravel
NetID: db5309

```
                    if (data == 3'b110) begin
                        state <= 3'b110;  // Go to state 6
                    end else if (data == 3'b001) begin
                        state <= 3'b001;  // Restart with new match to seq[1]
                    end else begin
                        state <= 3'b000;  // Reset to state 0
                    end
                end
                3'b110: begin  // State 6: Matched seq[1..6]
                    if (data == 3'b011) begin
                        state <= 3'b111;  // Go to state 7
                    end else if (data == 3'b001) begin
                        state <= 3'b001;  // Restart with new match to seq[1]
                    end else begin
                        state <= 3'b000;  // Reset to state 0
                    end
                end
                3'b111: begin  // State 7: Matched seq[1..7], waiting for seq[8]
                    if (data == 3'b101) begin
                        sequence_found <= 1'b1;  // Sequence detected!
                        state <= 3'b000;    // seq[8] != seq[1], so reset to state 0
                    end else if (data == 3'b001) begin
                        state <= 3'b001;  // Restart with new match to seq[1]
                    end else begin
                        state <= 3'b000;  // Reset to state 0
                    end
                end
                default: begin
                    state <= 3'b000;  // Fallback
                end
            endcase
        end
    end

endmodule
```

Error/Observation:

```
Error: Cycle 8, Expected: 1, Got: 0
```

**Prompt 3:**

Appended the generated code along with the error and added the line – "`resolve this error by modifying the design`." to the end.

Final Correct Design:

```
module binary_to_bcd_converter (
```

Name: Dhirajzen Bagawath Geetha Kumaravel
NetID: db5309

```
module sequence_detector (
    input wire clk,
    input wire reset_n,      // Active-low synchronous reset
    input wire [2:0] data,   // 3-bit input data
    output wire sequence_found     // Output: high when sequence is detected (combinatorial)
);

    // State register: 3 bits sufficient for states 0-7
    reg [2:0] state;

    // Combinatorial output: high if in final state and receiving the last symbol
    assign sequence_found = (state == 3'b111 && data == 3'b101);

    // Define the sequence symbols for reference (not synthesizable, for clarity)
    // seq[1] = 3'b001
    // seq[2] = 3'b101
    // seq[3] = 3'b110
    // seq[4] = 3'b000
    // seq[5] = 3'b110
    // seq[6] = 3'b110
    // seq[7] = 3'b011
    // seq[8] = 3'b101

    always @(posedge clk) begin
        if (!reset_n) begin
            state <= 3'b000;  // State 0
        end else begin
            case (state)
                3'b000: begin  // State 0: No match
                    if (data == 3'b001) begin
                        state <= 3'b001;  // Go to state 1
                    end else begin
                        state <= 3'b000;  // Stay in state 0
                    end
                end
                3'b001: begin  // State 1: Matched seq[1]
                    if (data == 3'b101) begin
                        state <= 3'b010;  // Go to state 2
                    end else if (data == 3'b001) begin
                        state <= 3'b001;  // Restart with new match to seq[1]
                    end else begin
                        state <= 3'b000;  // Reset to state 0
                    end
                end
                3'b010: begin  // State 2: Matched seq[1..2]
                    if (data == 3'b110) begin
                        state <= 3'b011;  // Go to state 3
```

Name: Dhirajzen Bagawath Geetha Kumaravel
NetID: db5309

```
                    end else if (data == 3'b001) begin
                        state <= 3'b001;  // Restart with new match to seq[1]
                    end else begin
                        state <= 3'b000;  // Reset to state 0
                    end
                end
                3'b011: begin  // State 3: Matched seq[1..3]
                    if (data == 3'b000) begin
                        state <= 3'b100;  // Go to state 4
                    end else if (data == 3'b001) begin
                        state <= 3'b001;  // Restart with new match to seq[1]
                    end else begin
                        state <= 3'b000;  // Reset to state 0
                    end
                end
                3'b100: begin  // State 4: Matched seq[1..4]
                    if (data == 3'b110) begin
                        state <= 3'b101;  // Go to state 5
                    end else if (data == 3'b001) begin
                        state <= 3'b001;  // Restart with new match to seq[1]
                    end else begin
                        state <= 3'b000;  // Reset to state 0
                    end
                end
                3'b101: begin  // State 5: Matched seq[1..5]
                    if (data == 3'b110) begin
                        state <= 3'b110;  // Go to state 6
                    end else if (data == 3'b001) begin
                        state <= 3'b001;  // Restart with new match to seq[1]
                    end else begin
                        state <= 3'b000;  // Reset to state 0
                    end
                end
                3'b110: begin  // State 6: Matched seq[1..6]
                    if (data == 3'b011) begin
                        state <= 3'b111;  // Go to state 7
                    end else if (data == 3'b001) begin
                        state <= 3'b001;  // Restart with new match to seq[1]
                    end else begin
                        state <= 3'b000;  // Reset to state 0
                    end
                end
                3'b111: begin  // State 7: Matched seq[1..7], waiting for seq[8]
                    if (data == 3'b101) begin
                        state <= 3'b000;    // seq[8] != seq[1], so reset to state 0
                    end else if (data == 3'b001) begin
                        state <= 3'b001;  // Restart with new match to seq[1]
```

Name: Dhirajzen Bagawath Geetha Kumaravel
NetID: db5309

```
                    end else begin
                        state <= 3'b000;  // Reset to state 0
                    end
                end
                default: begin
                    state <= 3'b000;  // Fallback
                end
            endcase
        end
    end

endmodule
```

Final Correct Output:

```
All test cases passed.
```

## Observations

- In Example 2, the sequence detector required three iterations to converge. The first design failed because the testbench expected the output port name sequence_found while the design used seq_found.
- After renaming, the simulation still failed at cycle 8 due to an off-by-one timing issue, as the registered output pulse was de-asserted too early.
- The final fix changed the output to a combinational condition (state == 3'b111 && data == 3'b101), which allowed detection in the same cycle as the last symbol and matched the testbench's expectation. This highlighted the importance of aligning port names with the testbench and ensuring the output logic matches the timing model required for sequence detection.

Name: Dhirajzen Bagawath Geetha Kumaravel
NetID: db5309

## Example 3

**Shift Register:**

- Inputs:
  - Clock
  - Active-low reset
  - Data (1 bit)
  - Shift enable
- Outputs:
  - Data (8 bits)

## Prompts and Outputs

**Prompt 1:**

```
I am trying to create a Verilog model for a shift register. It must meet the following
specifications:
    - Inputs:
        - Clock
        - Active-low reset
        - Data (1 bit)
        - Shift enable
    - Outputs:
        - Data (8 bits)

How would I write a design that meets these specifications?
```

LLM Output Design:

```
module shift_register (
    input  wire clk,      // Clock input
    input  wire rst_n,    // Active-low reset input
    input  wire din,      // 1-bit serial data input
    input  wire en,       // Shift enable input (active-high)
    output wire [7:0] dout // 8-bit parallel data output
);

    // Internal 8-bit shift register
    reg [7:0] shift_reg;

    // Continuous assignment for output
    assign dout = shift_reg;
```

Name: Dhirajzen Bagawath Geetha Kumaravel
NetID: db5309

```
    // Sequential logic: Shift on posedge clk, async reset
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            // Asynchronous reset: Clear to 0
            shift_reg <= 8'b0;
        end else if (en) begin
            // Shift right: New din enters LSB, contents shift toward MSB
            shift_reg <= {shift_reg[6:0], din};
        end
        // If !en, hold current value (no else needed)
    end

endmodule
```

Observation:

```
All test cases passed.
```

## Observations

In Example 3, the shift register design worked correctly on the first attempt. The model implemented an 8-bit register with asynchronous active-low reset, a serial 1-bit data input, and a shift-enable signal. On each positive clock edge, when enabled, the register shifted its contents right while loading the new input bit into the least significant position. The output was continuously assigned to the internal register, ensuring that the parallel 8-bit value was always visible. The design passed all test cases without requiring further iterations, showing that the LLM produced a fully correct solution on the first try.

Name: Dhirajzen Bagawath Geetha Kumaravel
NetID: db5309

## Example 4

**Dice Roller:**

- Inputs:
  - Clock
  - Active-low reset
  - Die select (2-bits)
  - Roll
- Outputs:
  - Rolled number (up to 8-bits)

## Prompts and Outputs

**Prompt 1:**

```
I am trying to create a Verilog model for a simulated dice roller. It must meet the following
specifications:
    - Inputs:
        - Clock
        - Active-low reset
        - Die select (2-bits)
        - Roll
    - Outputs:
        - Rolled number (up to 8-bits)

The design should simulate rolling either a 4-sided, 6-sided, 8-sided, or 20-sided die, based
on the input die select. It should roll when the roll input goes high and output the random
number based on the number of sides of the selected die.

How would I write a design that meets these specifications?
```

LLM Design Output 1:

```
module dice_roller (
    input  wire        clk,          // Clock input
    input  wire        rst_n,        // Active-low reset
    input  wire [1:0]  die_sel,      // Die select: 00=4, 01=6, 10=8, 11=20 sides
    input  wire        roll,         // Roll trigger (rising edge detected)
    output reg  [7:0]  rolled_num    // Rolled number (1 to N sides)
);

    // LFSR register for pseudo-random generation (8-bit)
    reg [7:0] lfsr;

    // For rising edge detection on roll
    reg roll_prev;

    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
```

Name: Dhirajzen Bagawath Geetha Kumaravel
NetID: db5309

```
            // Reset: seed LFSR (non-zero), clear output, clear roll history
            lfsr       <= 8'h01;
            rolled_num <= 8'd0;
            roll_prev  <= 1'b0;
        end else begin
            // Update roll history
            roll_prev <= roll;

            // Detect rising edge of roll and generate new value
            if (roll && !roll_prev) begin
                // Compute number of sides based on die_sel
                integer sides;
                case (die_sel)
                    2'b00: sides = 4;
                    2'b01: sides = 6;
                    2'b10: sides = 8;
                    2'b11: sides = 20;
                    default: sides = 6;  // Default to 6-sided if invalid
                endcase

                // Generate pseudo-random number: use current LFSR value, map to 1-sides
                rolled_num <= ((lfsr % sides) + 1);

                // Advance LFSR (shift left, primitive polynomial taps: 7,5,3,2)
                // Feedback: lfsr[7] ^ lfsr[5] ^ lfsr[3] ^ lfsr[2]
                lfsr <= {lfsr[6:0], (lfsr[7] ^ lfsr[5] ^ lfsr[3] ^ lfsr[2])};
            end
            // Output holds value until next roll
        end
    end

endmodule
```

Error/Observation:

```
dice_roller.v:30: error: Variable declaration in unnamed block requires SystemVerilog.
```

**Prompt 2:**

Appended the generated code along with the error and added the line – "`resolve this error by modifying the design`." to the end.

LLM Design Output 2:

```
module dice_roller (
    input  wire        clk,          // Clock input
    input  wire        rst_n,        // Active-low reset
    input  wire [1:0]  die_sel,      // Die select: 00=4, 01=6, 10=8, 11=20 sides
```

Name: Dhirajzen Bagawath Geetha Kumaravel
NetID: db5309

```
    input  wire        roll,         // Roll trigger (rising edge detected)
    output reg  [7:0]  rolled_num    // Rolled number (1 to N sides)
);

    // LFSR register for pseudo-random generation (8-bit)
    reg [7:0] lfsr;

    // For rising edge detection on roll
    reg roll_prev;

    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            // Reset: seed LFSR (non-zero), clear output, clear roll history
            lfsr       <= 8'h01;
            rolled_num <= 8'd0;
            roll_prev  <= 1'b0;
        end else begin
            // Update roll history
            roll_prev <= roll;

            // Detect rising edge of roll and generate new value
            if (roll && !roll_prev) begin
                // Compute rolled number based on die_sel (map LFSR to 1-sides)
                case (die_sel)
                    2'b00: rolled_num <= ((lfsr % 4) + 1);   // 4-sided
                    2'b01: rolled_num <= ((lfsr % 6) + 1);   // 6-sided
                    2'b10: rolled_num <= ((lfsr % 8) + 1);   // 8-sided
                    2'b11: rolled_num <= ((lfsr % 20) + 1);  // 20-sided
                    default: rolled_num <= ((lfsr % 6) + 1); // Default to 6-sided if invalid
                endcase

                // Advance LFSR (shift left, primitive polynomial taps: 7,5,3,2)
                // Feedback: lfsr[7] ^ lfsr[5] ^ lfsr[3] ^ lfsr[2]
                lfsr <= {lfsr[6:0], (lfsr[7] ^ lfsr[5] ^ lfsr[3] ^ lfsr[2])};
            end
            // Output holds value until next roll
        end
    end

endmodule
```

Error/Observation:

```
dice_roller_tb.v:10: error: port ``die_select'' is not a port of dut.
dice_roller_tb.v:10: error: port ``rolled_number'' is not a port of dut.
2 error(s) during elaboration.
```

Name: Dhirajzen Bagawath Geetha Kumaravel
NetID: db5309

**Prompt 3:**

Appended the generated code along with the error and added the line – "`resolve this error by modifying the design`." to the end.

Final Correct Design:

```
module dice_roller (
    input  wire        clk,              // Clock input
    input  wire        rst_n,            // Active-low reset
    input  wire [1:0]  die_select,       // Die select: 00=4, 01=6, 10=8, 11=20 sides
    input  wire        roll,             // Roll trigger (rising edge detected)
    output reg  [7:0]  rolled_number     // Rolled number (1 to N sides)
);

    // LFSR register for pseudo-random generation (8-bit)
    reg [7:0] lfsr;

    // For rising edge detection on roll
    reg roll_prev;

    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            // Reset: seed LFSR (non-zero), clear output, clear roll history
            lfsr         <= 8'h01;
            rolled_number <= 8'd0;
            roll_prev    <= 1'b0;
        end else begin
            // Update roll history
            roll_prev <= roll;

            // Detect rising edge of roll and generate new value
            if (roll && !roll_prev) begin
                // Compute rolled number based on die_select (map LFSR to 1-sides)
                case (die_select)
                    2'b00: rolled_number <= ((lfsr % 4) + 1);    // 4-sided
                    2'b01: rolled_number <= ((lfsr % 6) + 1);    // 6-sided
                    2'b10: rolled_number <= ((lfsr % 8) + 1);    // 8-sided
                    2'b11: rolled_number <= ((lfsr % 20) + 1);   // 20-sided
                    default: rolled_number <= ((lfsr % 6) + 1);  // Default to 6-sided if
invalid
                endcase

                // Advance LFSR (shift left, primitive polynomial taps: 7,5,3,2)
                // Feedback: lfsr[7] ^ lfsr[5] ^ lfsr[3] ^ lfsr[2]
                lfsr <= {lfsr[6:0], (lfsr[7] ^ lfsr[5] ^ lfsr[3] ^ lfsr[2])};
            end
```

Name: Dhirajzen Bagawath Geetha Kumaravel
NetID: db5309

```
            // Output holds value until next roll
        end
    end

endmodule
```

Final Correct Output:

```
VCD info: dumpfile my_design.vcd opened for output.
Results for die_select 00:
  Rolled           1:        314 times
  Rolled           2:        271 times
  Rolled           3:        271 times
  Rolled           4:        144 times
Results for die_select 01:
  Rolled           1:        142 times
  Rolled           2:        128 times
  Rolled           3:        222 times
  Rolled           4:        159 times
  Rolled           5:        224 times
  Rolled           6:        125 times
Results for die_select 10:
  Rolled           1:        160 times
  Rolled           2:        159 times
  Rolled           3:        159 times
  Rolled           4:        111 times
  Rolled           5:        159 times
  Rolled           6:        111 times
  Rolled           7:        110 times
  Rolled           8:         31 times
Results for die_select 11:
  Rolled           1:         63 times
  Rolled           2:         79 times
...
  Rolled          18:         63 times
  Rolled          19:         63 times
  Rolled          20:         48 times
Testbench completed successfully.
```

## Observations

- In Example 5, the dice-roller required two fixes before passing. The first design failed to compile because it declared integer sides; inside an always block- legal in SystemVerilog but not plain Verilog- so we removed the runtime variable and mapped die_sel to modulus constants directly in a case.
- The second failure was an **interface mismatch**: the testbench expected ports die_select and rolled_number while the design used die_sel and rolled_num; renaming the ports resolved elaboration errors.
- The final design keeps a rising-edge detector on roll (roll_prev) and updates an 8-bit LFSR (taps 7,5,3,2) each roll, then computes (lfsr % N) + 1 based on the selected die. Simulation completed successfully with per-face counts that were broadly reasonable for the short run (expected variance with modulo mapping and finite samples), confirming correct functionality and interface compliance.

Name: Dhirajzen Bagawath Geetha Kumaravel
NetID: db5309

## Example 5

**PWM Generator:**

Specifications:

- Module name: pwm
- Parameters:
  - WIDTH = 16 (default)
- Inputs:
  - clk : clock
  - rst_n : active-low synchronous reset
  - enable : when high, PWM runs; when low, output forced low and counter holds
  - period : total cycle length (WIDTH bits, >0)
  - duty : duty cycle (WIDTH bits, 0..period, clamp if > period)
- Outputs:
  - pwm_out : high for 'duty' cycles out of each 'period'
  - tick : single-cycle pulse when counter resets (start of period)

- Behavior:
  - Counter increments each clk when enabled, resets to 0 after reaching (period-1).
  - pwm_out is high while counter < duty.
  - tick asserted for one cycle at start of each PWM period.
  - On reset, counter clears to 0, outputs cleared.

- Constraints:
  - Use synchronous reset logic.

## Prompts and Outputs

**Prompt 1:**

```
Generate a synthesizable Verilog module for a parameterizable PWM generator.

Specifications:
- Module name: pwm
- Parameters:
    WIDTH = 16 (default)
- Inputs:
    clk      : clock
    rst_n    : active-low synchronous reset
    enable   : when high, PWM runs; when low, output forced low and counter
holds
    period   : total cycle length (WIDTH bits, >0)
    duty     : duty cycle (WIDTH bits, 0..period, clamp if > period)
- Outputs:
    pwm_out  : high for 'duty' cycles out of each 'period'
    tick     : single-cycle pulse when counter resets (start of period)

Behavior:
```

Name: Dhirajzen Bagawath Geetha Kumaravel
NetID: db5309

```
- Counter increments each clk when enabled, resets to 0 after reaching (period-
1).
- pwm_out is high while counter < duty.
- tick asserted for one cycle at start of each PWM period.
- On reset, counter clears to 0, outputs cleared.

Constraints:
- Use synchronous reset logic.
```

LLM Design Output 1:

```
module pwm #(
    parameter WIDTH = 16
) (
    input  wire                clk,
    input  wire                rst_n,
    input  wire                enable,
    input  wire [WIDTH-1:0]    period,
    input  wire [WIDTH-1:0]    duty,
    output reg                 pwm_out,
    output reg                 tick
);

    reg [WIDTH-1:0] cnt;

    always @(posedge clk) begin
        if (!rst_n) begin
            cnt    <= {WIDTH{1'b0}};
            pwm_out <= 1'b0;
            tick    <= 1'b0;
        end else begin
            if (!enable) begin
                cnt    <= cnt;
                pwm_out <= 1'b0;
                tick    <= 1'b0;
            end else begin
                if (cnt == period - 1'b1) begin
                    cnt  <= {WIDTH{1'b0}};
                    tick <= 1'b1;
                end else begin
                    cnt  <= cnt + 1'b1;
                    tick <= 1'b0;
                end
                pwm_out <= (cnt < duty);
```

Name: Dhirajzen Bagawath Geetha Kumaravel
NetID: db5309

```
            end
        end
    end

endmodule
```

Output:

```
VCD info: dumpfile pwm.vcd opened for output.
[PASS] duty=0%: expected highs=0, measured=0 (tol=0)
[PASS] duty=25%: expected highs=25, measured=25 (tol=0)
[PASS] duty=50%: expected highs=50, measured=50 (tol=0)
[PASS] duty=75%: expected highs=75, measured=75 (tol=0)
[PASS] y>=period (100%): expected highs=100, measured=100 (tol=0)
[PASS] duty=33 (~33.3%): expected highs=33, measured=33 (tol=0)
All PWM tests completed successfully!
```

## Observations

The PWM generator worked correctly on the first attempt. The design matched the specification and passed all test cases (0%, 25%, 50%, 75%, 100%, and 33% duty), with measured outputs exactly as expected.

Name: Dhirajzen Bagawath Geetha Kumaravel
NetID: db5309

## Role of ChatGPT in Report Writing

- While the Verilog code and simulation results for each example were obtained directly through the ChipChat (LLM) workflow and verified with Icarus Verilog, the **observation sections** in this report were prepared with the assistance of **ChatGPT**. ChatGPT was used to refine my raw notes and error logs into concise, structured descriptions that summarize the iteration process, highlight the issues encountered, and explain how they were resolved.

- Its role was limited to **documentation support**: organizing outcomes into clear paragraphs, ensuring consistency across examples, and improving readability. All technical content — including the Verilog designs, testbenches, simulation runs, and error traces, came from my own experiments with the ChipChat Colab setup.

## New Learnings

- Understood how to refine LLM-generated Verilog by combining prompts with simulation error feedback.
- Learned to identify and fix common issues: port mismatches, width handling, and timing misalignment.
- Observed that LLMs are effective for generating baseline designs but still require manual debugging and verification.
- Improved skills in structuring results and writing concise observations for technical reports.
- Gained confidence in using AI tools to accelerate hardware design workflows while maintaining engineering rigor.