Project Title: FPGA Counter and LED Display System

Author: B Hydara

Department: Dept. of Computer Eng. Tech.

Date: 8/07/2025

Table of contents Page

Abstract

This project documents the design and implementation of a frequency divider with a 4-bit bidirectional counter and decoder on the Intel DE0-CV FPGA board. The project integrates three key components:

- A 25-bit frequency divider (LPM counter) to slow the 50MHz clock to ~1.5Hz.
- A 4-bit up/down counter with enable/clear controls.
- A hex-to-7-segment decoder and 4-bit GPIO LED pattern generator.

The system demonstrates counter, decoders, and FPGA I/O interfacing (onboard LEDs, 7-segment display, and breadboard LEDs). All functionalities were verified through simulation and hardware testing.

Introduction to FPGA in Digital System Design

FPGAs enable rapid prototyping of digital systems by combining programmable logic and custom I/O configurations. This lab highlights:

- Hierarchical design: Integrating LPM modules with custom Verilog code.
- Clock management: Using a frequency divider for human-observable outputs.
- Input/Output interfacing: Switches control the counter, while outputs drive LEDs and a 7-segment display.

Problem Definition and System Schematic

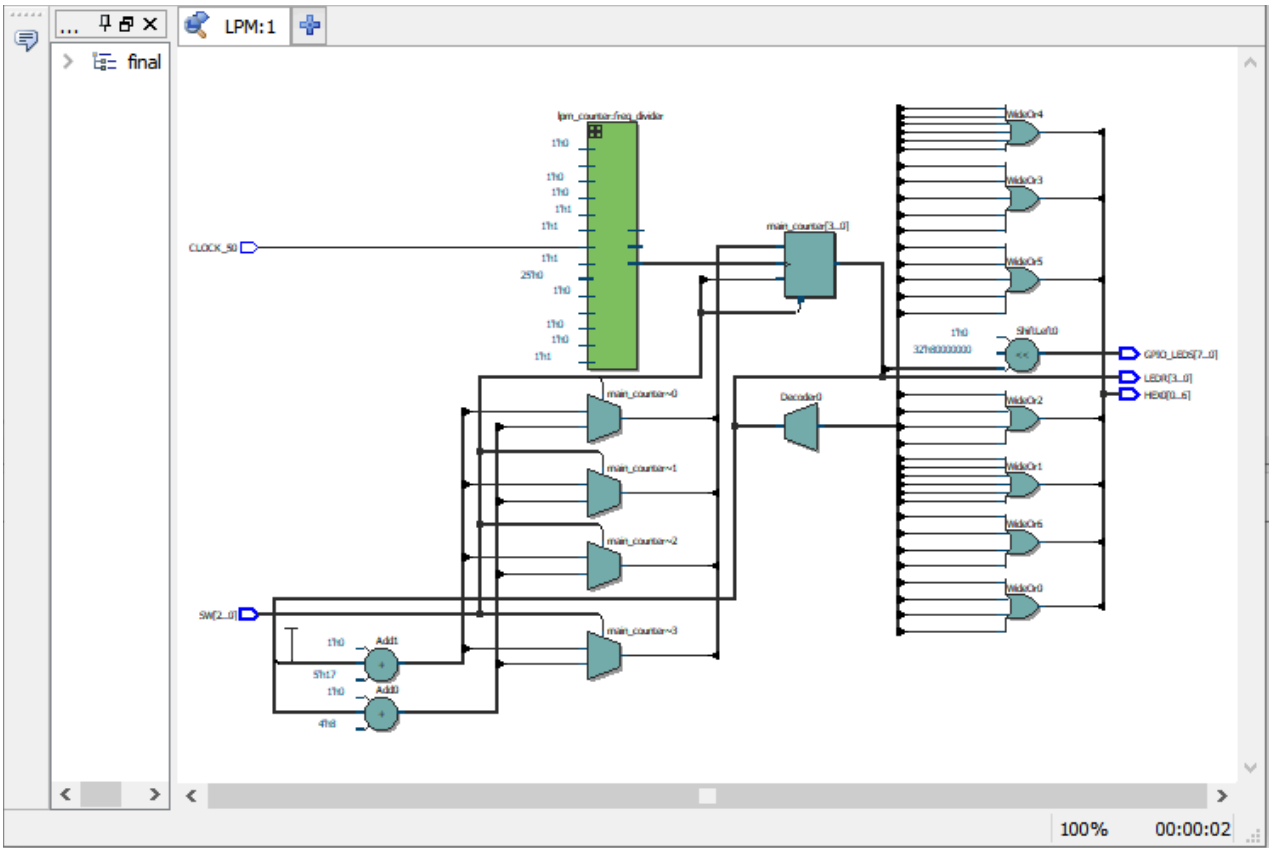
Design an FPGA-based system that:

- Uses the 50 MHz clock in a real-time application.
- Integrates at least three modules from the course.
- Displays a binary count on a 7-segment display and both onboard and breadboard LEDs.

System Schematic:

- **Clock Source:** 50 MHz onboard oscillator (PIN_M9).
- **LPM Counter:** Divides clock to ~6 Hz using bit 23 of the counter.

- **4-bit Counter:** Increments or decrements based on `ud` switch input; enabled via `en`; cleared with `clrn`.
- **Decoder:** Maps binary value to HEX0 display segments.
- **Outputs:** Four onboard LEDs (LEDR0–3) and four breadboard LEDs via GPIO_1_D0–D3.



Project Task Breakdown and Procedure:

1. **Create a New Quartus Project** targeting the DE0-CV board.
2. **Write Verilog Modules:**
    - `counter4.v` – 4-bit up/down counter with enable and clear.
    - `Decoder4bit.v` – 7-segment decoder for HEX0.
    - Top module integrating:
        - LPM counter for frequency division.
        - Counter and decoder.

        - Dual LED outputs (onboard & GPIO).

3. **Instantiate LPM Counter** via MegaWizard, configured as:
    - Direction: UP
    - Width: 28 bits
    - Only `clk` and `q[]` used.
4. **Assign Pins** in Pin Planner:

Signal	FPGA Pin	Board Component
CLOCK_50	PIN_M9	50MHz Clock
SW[0] (Enable)	PIN_U13	Switch 0
LEDR[0]	PIN_AA2	Onboard LED 0
GPIO_LEDS[0]	PIN_H16	Breadboard LED 0

5. **Compile & Program** the FPGA.
6. **Test** on both onboard LEDs and breadboard with resistors and LEDs connected to GPIO pins.
7. **Verify** output matches expected binary counting sequence.

Data Analysis | Waveform Simulation

- Part I: `counter3.v` waveform shows clean counting from 000 to 111 with enable high and clrn de-asserted
- Part III: `counter4.v` waveform confirms counting up/down based on `dir` value, and simulation shows transitions on `Q[3:0]` as expected

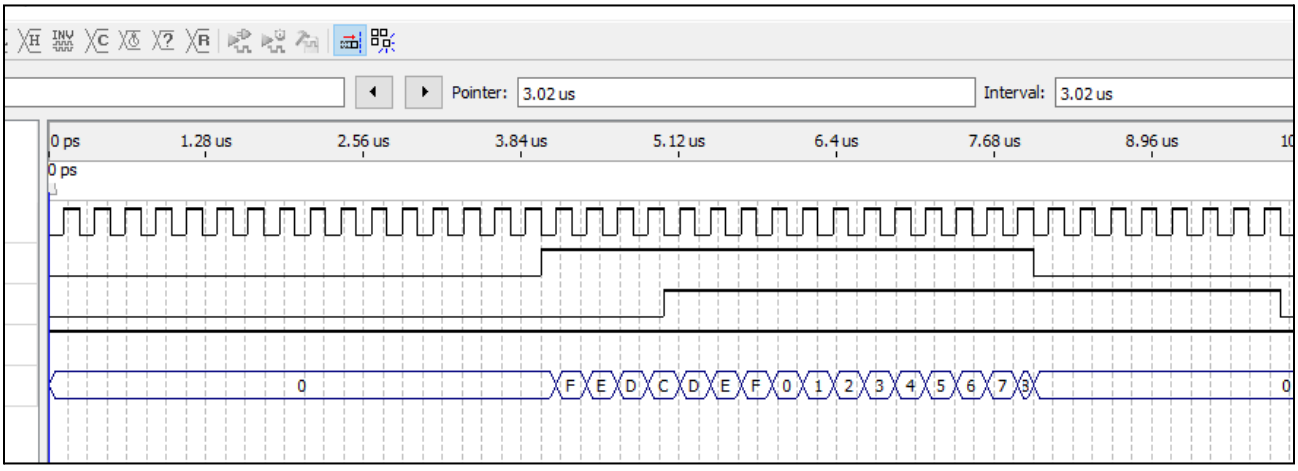
Demonstration

- ## Photos with description

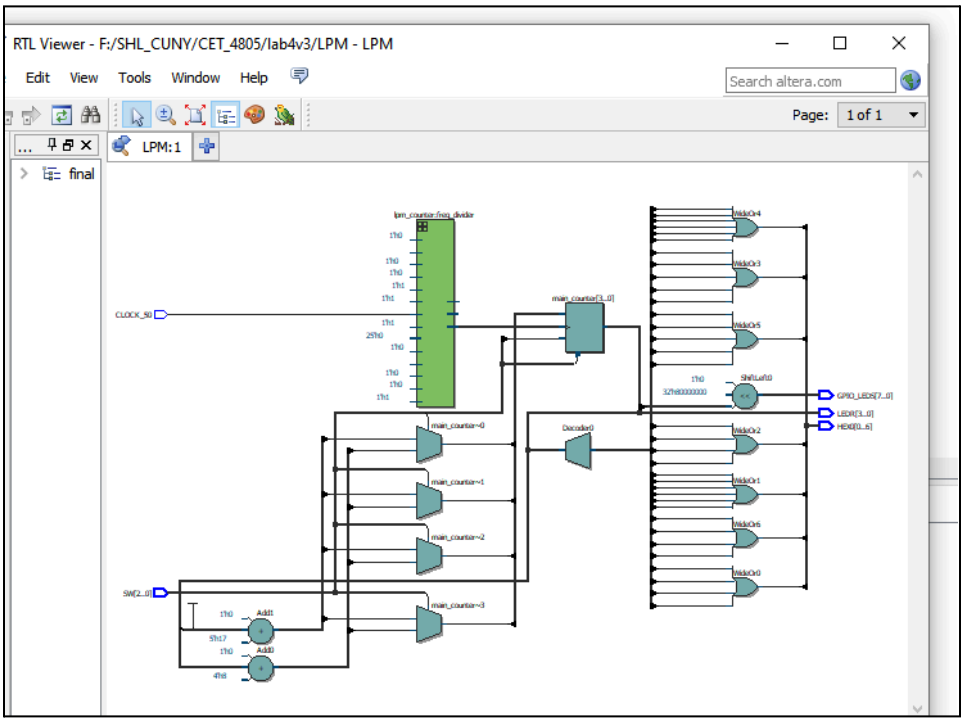
This is the pin planner of the project

Node Name	Direction	Location	I/O Bank	
CLOCK_50	Input	PIN_M9	3B	B
GPIO_LEDS[7]	Output			
GPIO_LEDS[6]	Output			
GPIO_LEDS[5]	Output			
GPIO_LEDS[4]	Output			
GPIO_LEDS[3]	Output	PIN_H16	7A	B
GPIO_LEDS[2]	Output	PIN_A12	7A	B
GPIO_LEDS[1]	Output	PIN_H15	7A	B
GPIO_LEDS[0]	Output	PIN_B12	7A	B
HEX0[0]	Output	PIN_U21	4A	B
HEX0[1]	Output	PIN_V21	4A	B
HEX0[2]	Output	PIN_W22	4A	B
HEX0[3]	Output	PIN_W21	4A	B
HEX0[4]	Output	PIN_Y22	4A	B
HEX0[5]	Output	PIN_Y21	4A	B
HEX0[6]	Output	PIN_AA22	4A	B
LEDR[3]	Output	PIN_Y3	2A	B
LEDR[2]	Output	PIN_W2	2A	B
LEDR[1]	Output	PIN_AA1	2A	B
LEDR[0]	Output	PIN_AA2	2A	B
SW[2]	Input	PIN_T13	4A	B
SW[1]	Input	PIN_V13	4A	B
SW[0]	Input	PIN_U13	4A	B

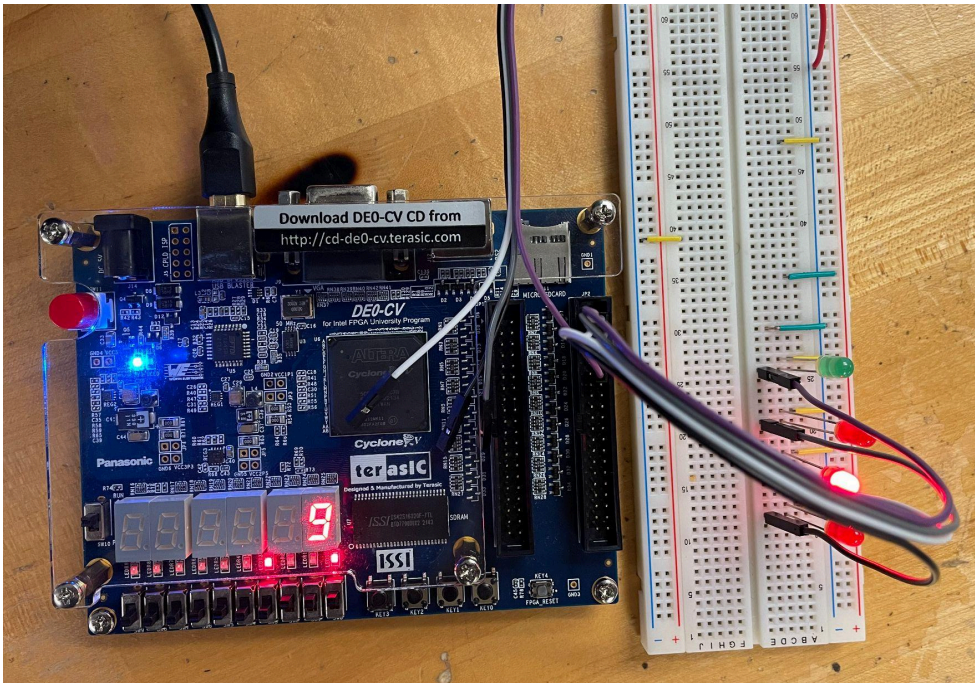
Waveform simulation



System Schematic



Testing on the board



Future Work and Conclusion

Future Work

- Implement clock divider selection for adjustable LED blink rates.
- Add hardware debouncing for switch inputs.
- Extend design to drive multiple 7-segment displays

Conclusion

The design successfully met all requirements:

- Frequency divider provided stable slow clock.
- Counter responded correctly to control switches.
- All output devices (LEDs, 7-segment, GPIO) displayed expected patterns.

References

- DE0-CV User Manual
- Quartus Prime Documentation

Appendix A (Source code)

```
module final (

    // Clock and control inputs

    input CLOCK_50,           // 50MHz clock input (PIN_M9)

    input [2:0] SW,           // Control switches:


    // Onboard outputs

    output [3:0] LEDR,        // Onboard red LEDs (4-bit binary count)


    output [0:6] HEX0,        // 7-segment display (PIN_U21 to PIN_AA22)


    // Breadboard outputs

    output [7:0] GPIO_LEDS    // GPIO-connected LEDs (8-bit walking pattern)


);


    // ================================================

    // 1. Frequency Divider (25-bit counter)

    // ================================================

    wire [24:0] div_counter;


    lpm_counter freq_divider (

        .clock (CLOCK_50),

        .q (div_counter),

        .aclr (1'b0),
```

```
    .aload (1'b0),
    .aset (1'b0),
    .cin (1'b1),
    .clk_en (1'b1),
    .cnt_en (1'b1),
    .cout (),
    .data ({25{1'b0}}),
    .eq (),
    .sclr (1'b0),
    .sload (1'b0),
    .sset (1'b0),
    .updown (1'b1)
);

defparam
    freq_divider.lpm_direction = "UP",
    freq_divider.lpm_port_updown = "PORT_UNUSED",
    freq_divider.lpm_type = "LPM_COUNTER",
    freq_divider.lpm_width = 25;

// 2. Main 4-bit Counter
reg [3:0] main_counter;

always @(posedge slow_clock, negedge SW[1]) begin
    if (!SW[1]) begin           // Active-low clear
        main_counter <= 4'b0000;
```

```
    end
    else if (SW[0]) begin       // Enable
        if (SW[2])              // Direction (1=up, 0=down)
            main_counter <= main_counter + 1;
        else
            main_counter <= main_counter - 1;
    end
end

    // 3. Decoders

// Binary to 8-bit walking pattern decoder
wire [7:0] walking_pattern;
assign walking_pattern = (1 << main_counter[2:0]);

// Hex to 7-segment decoder
reg [0:6] hex_display;
always @(*) begin
    case (main_counter)
        4'h0: hex_display = 7'b0000001;  // 0
        4'h1: hex_display = 7'b1001111;  // 1
        4'h2: hex_display = 7'b0010010;  // 2
        4'h3: hex_display = 7'b0000110;  // 3
        4'h4: hex_display = 7'b1001100;  // 4
        4'h5: hex_display = 7'b0100100;  // 5
        4'h6: hex_display = 7'b0100000;  // 6
```

```
            4'h7: hex_display = 7'b0001111;  // 7
            4'h8: hex_display = 7'b0000000;  // 8
            4'h9: hex_display = 7'b0000100;  // 9
            4'hA: hex_display = 7'b0001000;  // A
            4'hB: hex_display = 7'b1100000;  // b
            4'hC: hex_display = 7'b0110001;  // C
            4'hD: hex_display = 7'b1000010;  // d
            4'hE: hex_display = 7'b0110000;  // E
            4'hF: hex_display = 7'b0111000;  // F
            default: hex_display = 7'bxxxxxxx;
        endcase
    end

    // ================================================
    // 4. Output Assignments
    // ================================================
    assign LEDR = main_counter;      // Binary count on onboard LEDs
    assign HEX0 = hex_display;       // Hex value on 7-segment
    assign GPIO_LEDS = walking_pattern; // Walking pattern on breadboard

endmodule
```

Acknowledgments

Special thanks to Dr. Yu Wang for guidance on Quartus and ModelSim setup.