EC311 - Lab 2: Sequential Logic and 7-Segment Display

Student Name: Giacomo Cappelletto
Student ID: U91023753
Date: November 21, 2025

---

1. Objective

This lab consists of two main parts:

Part 1: Design and implement a 8-bit counter system with proper button input handling, which included:

- A 2-flip-flop synchronizer to prevent metastability issues
- A debouncer to eliminate mechanical bounce from button presses
- A rising edge detector to generate clean single-cycle pulses
- An 8-bit counter that increments on each pulse (button press)
- LEDs to display the counter value in binary

Part 2: Extend the counter system to 16 bits and add 7-segment 4-bit hexadecimal display capability. The system must include:

- A 16-bit counter with dual operation modes (manual button and automatic 1Hz)
- Clock dividers to generate 1kHz and 1Hz clocks from the 100MHz system clock
- A display controller using time-division duty cycle multiplexing to drive four 7-segment displays
- A seven-segment decoder to convert hexadecimal digits to display patterns
- Proper signal conditioning (synchronization, debouncing, edge detection) for user inputs as in Part 1

Both parts must be implemented in Verilog, synthesized, and tested on the Nexys A7 FPGA board. Testbenches must be developed to verify functionality in simulation before hardware deployment.

2. Methodology

2.1. Design Approach

The lab is divided into two parts, each building upon sequential logic design principles with increasing complexity. The overall design approach is a modular one, with each part being implemented as a separate module, and then integrated into a top level module.

2.1.1. Part 1: Button-Controlled 8-bit Counter

The design in Part 1 follows this signal conditioning pipeline:

1. **Synchronization:** The raw button input, which is asynchronous to the system clock, passes through a 2-flip-flop synchronizer. This prevents metastability issues that can occur when sampling asynchronous signals like mechanical buttons.
2. **Debouncing:** Mechanical buttons exhibit contact bounce, producing multiple transitions during a single press. A debouncer module filters these glitches by requiring the input to remain stable for approximately 20ms before propagating the change and signal onwards in the pipeline.
3. **Edge Detection:** To generate a single increment pulse per button press, a rising edge detector compares the current debounced signal with its previous value, outputting a one-cycle pulse on 0 to 1 transitions. This is done by storing the previous value in a register and comparing it with the current value.

4. **Counter:** An 8-bit counter with asynchronous active-low reset increments on each pulse. The counter value directly drives eight LEDs for visual feedback from each of the 8 bits of the register.

2.1.2. Part 2: 16-bit Counter with 7-Segment Display

The second part extends the counter to 16 bits and implements a sophisticated display system with dual operating modes:

1. **Clock Generation:** Two parameterized clock dividers generate a 1kHz clock (for display multiplexing) and a 1Hz clock (for automatic counting mode) from the 100MHz system clock.
2. **Mode Selection:** A switch selects between manual mode (button increments from the user) and automatic mode (1Hz increments). Both increment sources undergo the same signal conditioning as described in Part 1 (sync, debounce, edge detect) before reaching the counter.
3. **Display Multiplexing:** Since the Nexys A7 has two sets of four 7-segment displays sharing common cathode lines, time-division multiplexing displays one digit at a time. A display controller cycles through digits at 1kHz (250Hz per digit), fast enough that persistence of vision creates the illusion of simultaneous display in the first set of displays. The second set of displays is kept off by driving the digit select lines high.
4. **Seven-Segment Decoding:** A combinational decoder converts each 4-bit hexadecimal digit (0-F) to the appropriate 7-segment pattern using active-low outputs as described in the Data Sheet for the Nexys A7.

2.2. Verilog Implementation

2.2.1. Part 1 Modules

2.2.1.1. 2-FF Synchronizer (`sync_2ff.v`)

Implements a two-stage flip-flop chain to safely synchronize asynchronous inputs to the system clock domain, by only passing the data signa (button press) on to the next stage on posedge clock. Also handles the reset signal by resetting the first flip-flop on negedge reset.

(See Section 5.1.1 for code listing.)

2.2.1.2. Debouncer (`debouncer.v`)

Outputs a signal only if `COUNT_MAX` clock cycles (now set to 2,000,000 which is 20ms at 100MHz). Uses a counter that resets whenever the input disagrees with the current stable state (eg. the button was not pressed for 20ms).

(See Section 5.1.2 for code listing.)

2.2.1.3. Edge Detector (`edge_detect.v`)

Detects rising edges by storing the previous input value and outputting a one-cycle pulse when `level_in & ~level_dly` is true (which is only on the rising edge of the input signal).

(See Section 5.1.3 for code listing.)

2.2.1.4. 8-bit Counter (`counter_8bit.v`)

A simple synchronous counter with asynchronous active-low reset that increments when the enable signal is asserted and wraps around from 255 to 0 when it reaches 256.

(See Section 5.2.1 for code listing.)

2.2.1.5. Top Module Part 1 (`counter_8bit.v`)

Integrates all Part 1 modules, connecting the signal conditioning pipeline from raw button input to counter increment, with the counter output driving the LEDs. The active low reset button is converted to active high internally by the synchronizer with `reset_n = ~btn_rst_raw;`.

(See Section 5.2.2 for code listing.)

2.2.1.6. Constraints File (`nexys_a7.xdc`)

Sets the clock frequency to 100MHz, 50% duty cycle for the clock signal, and the buttons and LEDs to the appropriate pins on the Nexys A7 board following led[0] => LSB and led[7] => MSB.

(See Section 5.2.3 for code listing.)

2.2.2. Part 2 Modules

2.2.2.1. Clock Divider (`clock_divider.v`)

Generates a 50% duty cycle output clock whose frequency is `clk_in / DIVIDE_BY`. Based on a parameter `DIVIDE_BY` calculates the following:

- COUNTER_WIDTH: width of counter needed to count up to `DIVIDE_BY/2` (half the output period, measured in input clock cycles), calculated using `$clog2(DIVIDE_BY/2)` (log base 2) to determine the minimum number of bits required.
- COUNTER_MAX: maximum value of the counter, which is `DIVIDE_BY/2 - 1` since the counter starts from 0 and needs to count `DIVIDE_BY/2` cycles total.

The module then uses a counter that increments on each input clock cycle. When the counter reaches `COUNTER_MAX`, it toggles the output clock and resets the counter to 0. This creates a complete output period of `DIVIDE_BY` input cycles, with the output high for `DIVIDE_BY/2` cycles and low for `DIVIDE_BY/2` cycles, resulting in a 50% duty cycle. (See Section 5.3.1 for code listing.)

2.2.2.2. 16-bit Counter (`counter16.v`)

Extended version of the 8-bit counter, now 16 bits wide to display four hex digits. In hidsignt, could have simply parametrized the counter width enabling counter width selection at instantiation without having to duplicate the file. (See Section 5.3.2 for code listing.)

2.2.2.3. Display Controller (`display_control.v`)

Implements time-division multiplexing by cycling a 2-bit counter at 1kHz. This counter generates active-low digit select signals (one of four dispalys) and routes the corresponding 4-bit part of the current 16-bit count signal (LSB to MSB) to the decoder. (See Section 5.3.3 for code listing.)

2.2.2.4. Seven-Segment Decoder (`seven_segment_decoder.v`)

A combinational lookup table that maps 4-bit hex values (0-F) to active-low 7-segment patterns. Each pattern illuminates the appropriate segments to display the corresponding character as described in the Data Sheet for the Nexys A7. (See Section 5.3.4 for code listing.)

2.2.2.5. Top Module Part 2 (`top.v`)

Integrates all Part 2 modules. Button and 1Hz increment sources are conditioned separately, then muxed based on mode select. The 16-bit counter output feeds the display controller, which multiplexes digits to the decoder and 7-segment displays. The second display (AN[7:4]) is always off driving high in the constraints file (`assign digit_select_off = 4'b1111;`) (See Section 5.3.5 for code listing.)

2.2.2.6. Constraints File (`nexys_a7.xdc`)

Sets the follwing on the Nexys A7 board:

- Clock signal to E3 at 100MHz
- Mode select switch to J15 (SW0) where 0 is manual mode and 1 is automatic mode
- Increment button to P18 (BTND)
- Reset button to C12 (CPU_RESETN)
- Second display (AN[7:4]) to digit_select_off[3:0] (always off driving high in the top module)
- First display (AN[3:0]) to digit_select[3:0] (display controller output)
- Seven-segment decoder segments seg[6:0] to T10, R10, K16, K13, P15, T11, L18 as described in the Data Sheet for the Nexys A7

(See Section 5.3.6 for code listing.)

2.3. Simulation and Testing

Testbenches were developed for both parts to verify functionality before hardware deployment. The testbenches reduce debouncer and clock divider parameters to accelerate simulation time while maintaining functional equivalence.

2.3.1. Part 1 Testbench

The Part 1 testbench (`counter_8bit_tb.v`) exercises the complete signal path by asserting reset, then simulating multiple button presses with realistic hold times. The debouncer parameter is reduced to `COUNT_MAX=4` for faster simulation. The testbench verifies that each button press increments the counter once, and that reset properly clears the count. (See Section 5.4.1 for code listing.)

2.3.2. Part 2 Testbench

The Part 2 testbench (`top_tb.v`) tests both operating modes. Clock divider parameters are scaled down (`DIVIDE_BY=10` for 1kHz, `DIVIDE_BY=1000` for 1Hz) while maintaining their ratio. The testbench:

- Tests manual mode with multiple button presses
- Switches to automatic mode and observes autonomous counting
- Verifies display multiplexing by monitoring digit select and segment outputs
- Confirms that segments correctly display the counter value in hexadecimal

(See Section 5.4.2 for code listing.)

3. Observation

3.1. Simulation Results

Both parts of the lab were simulated and functionality was verified through testbenches before hardware deployment.

3.1.1. Part 1: Button-Controlled Counter

The Part 1 simulation demonstrated proper signal conditioning and counter operation:

- **Reset behavior:** The counter correctly initialized to 0 when the reset button was asserted (active-high, button BTNC on the Nexys A7 pressed), converting to active-low internally.
- **Synchronization:** The 2-FF synchronizer introduced a 2-clock-cycle delay, successfully eliminating potential metastability.
- **Debouncing:** With `COUNT_MAX=4` in simulation, the debouncer filtered button bounces and only propagated stable transitions after 4 clock cycles of consistency.
- **Edge detection:** Each button press, regardless of hold duration, generated exactly one single-cycle increment pulse.

- **Counter operation:** The 8-bit counter incremented correctly on each pulse, wrapping from 255 to 0 naturally thanks to 8-bit overflow.

3.1.2. Part 2: 7-Segment Display System

The Part 2 simulation verified both operating modes and display functionality:

- **Manual mode:** Button presses incremented the 16-bit counter correctly after signal conditioning, identical to Part 1 but with wider counter width.
- **Automatic mode:** When the mode select switch was set high, the counter incremented autonomously at the divided 1Hz clock rate (accelerated in simulation with scaled parameters).
- **Clock division:** Both clock dividers generated 50% duty cycle outputs at the expected frequencies (scaled for simulation).
- **Display multiplexing:** The display controller cycled through digits 0 => 1 => 2 => 3 => 0 repeatedly at the 1kHz rate. The digit select signals were correctly active-low with only one digit enabled at a time.
- **Seven-segment decoding:** Each 4-bit nibble of the counter was correctly decoded to its hexadecimal representation. For example, count value `0x1234` produced appropriate segment patterns for '4', '3', '2', '1' as the display cycled.

3.2. Hardware Validation

After successful simulation, both designs were synthesized, generated bitstreams, and programmed onto the Nexys A7 FPGA board.

3.2.1. Part 1 Hardware Results

The button-controlled counter worked as expected on hardware:

- The reset button (BTNC) cleared the counter
- The increment button (BTND) reliably incremented the counter once per press
- LEDs displayed the binary count value with LED[0] as LSB and LED[7] as MSB
- No double-counting or bounce issues were observed, confirming effective debouncing

3.2.2. Part 2 Hardware Results

The 7-segment display system operated correctly:

- Reset button () cleared the counter
- In manual mode, button presses incremented the displayed hexadecimal value
- In automatic mode, the display counted up at approximately 1Hz
- All four digits displayed clearly with no visible flicker, confirming adequate multiplexing rate at 1kHz.
- The second display (AN[7:4]) was off, as expected.
- The counter rolled over from `F` to `0` correctly

4. Conclusion

4.1. Summary of Results

This lab successfully demonstrated the design and implementation of sequential logic systems with proper synchronization, timing control, and display interfacing. Both parts functioned correctly in simulation and on hardware, meeting all specified requirements.

Part 1 achieved reliable button-controlled counting by implementing a complete signal conditioning pipeline that addressed metastability, contact bounce, and edge detection challenges which come with in interfacing an asynchronous human input to synchronous digital systems like the Nexys A7 FPGA board.

Part 2 extended the system with clock division and display multiplexing, demonstrating time-division multiplexing principles and hexadecimal decoding. The dual-mode operation (manual/automatic) showcased input muxing and provided flexibility for different use cases.

The modular design approach, with separate modules for synchronization, debouncing, edge detection, clock division, and display control, promoted reusability between the two parts and simplified testing. Parameters in modules like the debouncer and clock divider allowed easy adaptation for both simulation (fast parameters) and hardware deployment (real-time parameters).

4.2. Challenges and Solutions

Several challenges were encountered and resolved during implementation:

Debouncer timing: Initial attempts at debouncing used insufficient delay, causing occasional double-counts. Increasing `COUNT_MAX` to 2,000,000 (20ms at 100MHz) provided robust debouncing for the mechanical buttons.

Simulation performance: Running testbenches with full hardware parameters (100M clock cycles for 1Hz) was impractically slow. Using `defparam` to override parameters in the testbench reduced simulation time while maintaining functional equivalence by preserving frequency ratios.

Active-low signals: The Nexys A7 uses active-low digit select and segment signals. Initially using active-high logic resulted in inverted or blank displays. Changed polarity in both the display controller (digit select) and decoder (segment patterns) resolved this issue.

Second display: Initially, the second display (AN[7:4]) was not off and was flickering. Added a select line for the second display and `assign digit_select_off = 4'b1111;` to the top module to keep the second display off by always driving it high.

5. Appendix

5.1. Common Module Listings

5.1.1. 2-FF Synchronizer (`sync_2ff.v`)

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Boston University
// Engineer: Giacomo Cappelletto
//
// Create Date: 10/31/2025 11:55:26 AM
// Design Name: Flip-Flop Sync
// Module Name: sync_2ff
// Project Name: Counter Lab 2.1
// Target Devices: NEXYS A7
// Tool Versions:
// Description: 2-bit Flip-Flop Synchronizer for Counter Lab 2.1
//
// Dependencies: none
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module sync_2ff (
    input  wire clk,
    input  wire reset_n,
    input  wire d_async,
    output reg  q_sync
);
    reg q1;
    always @(posedge clk or negedge reset_n) begin
        if (!reset_n) begin
            q1     <= 1'b0;
            q_sync <= 1'b0;
        end else begin
            q1     <= d_async;
            q_sync <= q1;
        end
    end
endmodule
```

5.1.2. Debouncer (`debouncer.v`)

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Boston University
// Engineer: Giacomo Cappelletto
//
// Create Date: 10/31/2025 11:55:26 AM
// Design Name: Debouncer
// Module Name: debouncer
// Project Name: Counter Lab 2.1
// Target Devices: NEXYS A7
```

```
// Tool Versions:
// Description: Debouncer for Counter Lab 2.1
//
// Dependencies: none
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module debouncer #(
    parameter integer COUNT_MAX = 2_000_000  // ~20 ms @ 100 MHz
)(
    input  wire clk,
    input  wire reset_n,
    input  wire noisy_in,
    output wire clean_out
);
    localparam integer COUNTER_WIDTH = (COUNT_MAX > 1) ? $clog2(COUNT_MAX) : 1;

    reg [COUNTER_WIDTH-1:0] counter;
    reg stable_state;

    always @(posedge clk or negedge reset_n) begin
        if (!reset_n) begin
            stable_state <= 1'b0;
            counter <= 0;
        end else begin
            if (noisy_in == stable_state) begin
                counter <= 0;
            end else begin
                if (counter == COUNT_MAX - 1) begin
                    stable_state <= noisy_in;
                    counter <= 0;
                end else begin
                    counter <= counter + 1'b1;
                end
            end
        end
    end

    assign clean_out = stable_state;

endmodule
```

5.1.3. Edge Detector (`edge_detect.v`)

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Boston University
// Engineer: Giacomo Cappelletto
//
// Create Date: 10/31/2025 11:55:26 AM
// Design Name: Egde Detecter
// Module Name: edge_detect
// Project Name: Counter Lab 2.1
```

```
// Target Devices: NEXYS A7
// Tool Versions:
// Description: Edge Detecter for Counter Lab 2.1
//
// Dependencies: none
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module edge_detect (
    input  wire clk,
    input  wire reset_n,
    input  wire level_in,
    output reg  pulse_out
);

    reg level_dly;

    always @(posedge clk or negedge reset_n) begin
        if (!reset_n) begin
            level_dly  <= 0;
            pulse_out  <= 0;
        end else begin
            level_dly  <= level_in;
            pulse_out  <= level_in & ~level_dly;  // 1 clock pulse on rising edge
        end
    end

endmodule
```

5.2. Part 1 Specific Modules

5.2.1. 8-bit Counter (`counter_8bit.v`)

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Boston University
// Engineer: Giacomo Cappelletto
//
// Create Date: 10/31/2025 11:55:26 AM
// Design Name: 8-bit Counter
// Module Name: counter
// Project Name: Counter Lab 2.1
// Target Devices: NEXYS A7
// Tool Versions:
// Description: 8-bit Counter for Counter Lab 2.1
//
// Dependencies: none
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
```

```
module counter_8bit (
    input  wire       clock,
    input  wire       reset,
    input  wire       increment,
    output reg [7:0]  count
);
    always @(posedge clock or negedge reset) begin
        if (!reset)
            count <= 8'd0;
        else if (increment)
            count <= count + 8'd1;
    end
endmodule
```

5.2.2. Top Module Part 1 (`counter_8bit.v`)

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Boston University
// Engineer: Giacomo Cappelletto
//
// Create Date: 10/31/2025 11:55:26 AM
// Design Name: Main Assembly
// Module Name: top_part1
// Project Name: Counter Lab 2.1
// Target Devices: NEXYS A7
// Tool Versions:
// Description: Main Assembly for Counter Lab 2.1
//
// Dependencies: sync_2ff.v, debouncer.v, edge_detect.v, counter.v
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module top_part1 (
    input  wire       clk_100mhz,
    input  wire       btn_rst_raw, // active-high reset button (BTNC)
    input  wire       btn_inc_raw, // increment button (BTND)
    output wire [7:0] led          // LEDs on the board
);
    // Convert active-high reset button to active-low internal reset
    wire reset_n = ~btn_rst_raw;
    // Sync button to the clock as a metastability filter
    wire btn_sync;
    sync_2ff u_sync (
        .clk     (clk_100mhz),
        .reset_n (reset_n),
        .d_async (btn_inc_raw),
        .q_sync  (btn_sync)
    );

    // Debounce the synchronized button
    wire btn_debounced;
```

```
    debouncer #(.COUNT_MAX(2_000_000)) u_db (
        .clk       (clk_100mhz),
        .reset_n   (reset_n),
        .noisy_in  (btn_sync),
        .clean_out (btn_debounced)
    );

    // Detect rising edge to produce a one-cycle increment pulse
    wire inc_pulse;
    edge_detect u_edge (
        .clk       (clk_100mhz),
        .reset_n   (reset_n),
        .level_in  (btn_debounced),
        .pulse_out (inc_pulse)
    );

    // Counter
    wire [7:0] count;
    counter_8bit u_counter (
        .clock     (clk_100mhz),
        .reset     (reset_n),
        .increment (inc_pulse),
        .count     (count)
    );
    assign led = count; // LSB -> led[0], MSB -> led[7]
endmodule
```

5.2.3. Constraints File Part 1 (`nexys_a7.xdc`)

```
## Clock signal
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { clk_100mhz }];
#IO_L12P_T1_MRCC_35 Sch=clk100mhz
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {clk_100mhz}];

## LEDs
set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports { led[0] }];
#IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 } [get_ports { led[1] }];
#IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 } [get_ports { led[2] }];
#IO_L17N_T2_A25_15 Sch=led[2]
set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 } [get_ports { led[3] }];
#IO_L8P_T1_D11_14 Sch=led[3]
set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { led[4] }];
#IO_L7P_T1_D09_14 Sch=led[4]
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports { led[5] }];
#IO_L18N_T2_A11_D27_14 Sch=led[5]
set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports { led[6] }];
#IO_L17P_T2_A14_D30_14 Sch=led[6]
set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports { led[7] }];
#IO_L18P_T2_A12_D28_14 Sch=led[7]


##Buttons
set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 } [get_ports { btn_rst_raw }];
#IO_L9P_T1_DQS_14 Sch=btnc
```

```
set_property -dict { PACKAGE_PIN P18   IOSTANDARD LVCMOS33 } [get_ports { btn_inc_raw }];
#IO_L9N_T1_DQS_D13_14 Sch=btnd
```

5.3. Part 2 Specific Modules

5.3.1. Clock Divider (`clock_divider.v`)

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Boston University
// Engineer: Giacomo Cappelletto
//
// Create Date: 11/14/2025 11:23:56 AM
// Design Name: Clock Divider
// Module Name: clock_divider
// Project Name: Counter Lab 2.2
// Target Devices: NEXYS A7
// Tool Versions:
// Description: Parameterized clock divider module
//              Divides input clock frequency by DIVIDE_BY parameter
//              Output duty cycle is 50%
//
// Dependencies: none
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module clock_divider #(
    parameter integer DIVIDE_BY = 100_000  // Default: 100MHz -> 1kHz
)(
    input  wire clk_in,     // 100 MHz input clock
    input  wire reset_n,    // Active-low reset
    output reg  clk_out     // Divided output clock
);
    // Calculate counter width based on DIVIDE_BY parameter
    // Need to count up to DIVIDE_BY/2, so width is log2(DIVIDE_BY/2)
    localparam integer COUNTER_WIDTH = (DIVIDE_BY > 2) ? $clog2(DIVIDE_BY/2) : 1;
    localparam integer COUNTER_MAX = (DIVIDE_BY / 2) - 1;  // Count to half period

    // Internal counter
    reg [COUNTER_WIDTH-1:0] counter;

    // Clock divider logic toggles output every half period of the input clock
    always @(posedge clk_in or negedge reset_n) begin
        if (!reset_n) begin
            counter <= {COUNTER_WIDTH{1'b0}};  // Reset counter to 0
            clk_out <= 1'b0;                   // Reset output to 0
        end else begin
            if (counter == COUNTER_MAX) begin
                // Reached half period, toggle output and reset counter
                clk_out <= ~clk_out;
                counter <= {COUNTER_WIDTH{1'b0}};
            end else begin
                // Increment counter every input clock cycle
```

```
                counter <= counter + 1'b1;
            end
        end
    end
endmodule
```

5.3.2. 16-bit Counter (`counter16.v`)

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Boston University
// Engineer: Giacomo Cappelletto
//
// Create Date: 11/14/2025 11:23:56 AM
// Design Name: 16-bit Counter
// Module Name: counter
// Project Name: Counter Lab 2.2
// Target Devices: NEXYS A7
// Tool Versions:
// Description: 16-bit Counter for Counter Lab 2.2
//
// Dependencies: none
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module counter16 (
    input  wire       clk,
    input  wire       reset_n,
    input  wire       inc_pulse, // 1-cycle pulse
    output reg [15:0]  count
);
    always @(posedge clk or negedge reset_n) begin
        if (!reset_n)
            count <= 16'd0;
        else if (inc_pulse)
            count <= count + 16'd1; // should wrap naturally to 0 when it reaches 65535
    end
endmodule
```

5.3.3. Display Controller (`display_control.v`)

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Boston University
// Engineer: Giacomo Cappelletto
//
// Create Date: 11/14/2025 11:23:56 AM
// Design Name: Display Control
// Module Name: display_control
// Project Name: Counter Lab 2.2
// Target Devices: NEXYS A7
// Tool Versions:
```

```
// Description: Multiplexes 16-bit counter value across 4 seven-segment displays
//              Uses round-robin time-division multiplexing at 1kHz refresh rate
//
// Dependencies: none
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module display_control (
    input  wire        clk_1khz,      // 1kHz clock from clk divider
    input  wire        reset_n,
    input  wire [15:0] count,          // 16-bit counter value from counter
    output reg  [3:0]  digit_select,  // Active-low digit enable (AN signals)
    output reg  [3:0]  segment_data   // 4-bit hex digit to display
);
    // 2-bit counter to cycle through (0, 1, 2, 3)
    reg [1:0] digit_counter;

    // 2-bit counter increments on 1kHz clock and cycles 0->1->2->3->0
    always @(posedge clk_1khz or negedge reset_n) begin
        if (!reset_n)
            digit_counter <= 2'd0;
        else
            digit_counter <= digit_counter + 2'd1; // Wraps from 3 to 0
    end

    // Only one digit  active low at a time
    always @(*) begin
        case (digit_counter)
            2'd0: digit_select = 4'b1110; // Digit 0 (rightmost): AN0=0, others=1
            2'd1: digit_select = 4'b1101; // Digit 1: AN1=0, others=1
            2'd2: digit_select = 4'b1011; // Digit 2: AN2=0, others=1
            2'd3: digit_select = 4'b0111; // Digit 3 (leftmost): AN3=0, others=1
            default: digit_select = 4'b1111; // All off as default
        endcase
    end

    // Segment data mux selects which 4-bit segment of count to display, synced with
digit_select so correct digit value is shown
    always @(*) begin
        case (digit_counter)
            2'd0: segment_data = count[3:0];   // LSB: rightmost digit
            2'd1: segment_data = count[7:4];    // Second digit
            2'd2: segment_data = count[11:8];   // Third digit
            2'd3: segment_data = count[15:12];  // MSB: leftmost digit
            default: segment_data = 4'h0;       // Default to 0
        endcase
    end
endmodule
```

5.3.4. Seven-Segment Decoder (`seven_segment_decoder.v`)

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Boston University
// Engineer: Giacomo Cappelletto
//
// Create Date: 11/14/2025 11:23:56 AM
// Design Name: Seven Segment Decoder
// Module Name: seven_segment_decoder
// Project Name: Counter Lab 2.2
// Target Devices: NEXYS A7
// Tool Versions:
// Description: Binary to 7-segment display decoder for hex digits
//
// Dependencies: none
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module seven_segment_decoder (
    input  wire [3:0] hex_digit,  // 4-bit hex in
    output reg  [6:0] seven       // 7-bit out: seven[6:0] = {CG, CF, CE, CD, CC, CB, CA}
);
    // Segment mapping: seven[6:0] = {CG, CF, CE, CD, CC, CB, CA}
    // CA = segment A (top horizontal)
    // CB = segment B (top-right vertical)
    // CC = segment C (bottom-right vertical)
    // CD = segment D (bottom horizontal)
    // CE = segment E (bottom-left vertical)
    // CF = segment F (top-left vertical)
    // CG = segment G (middle horizontal)
    // Active LOW: 0 = segment ON, 1 = segment OFF

    always @(*) begin
        case (hex_digit)
            // Dec Part
            4'h0: seven = 7'b1000000; // 0: all except CG
            4'h1: seven = 7'b1111001; // 1: CB,CC ON
            4'h2: seven = 7'b0100100; // 2: CA,CB,CD,CE,CG ON
            4'h3: seven = 7'b0110000; // 3: CA,CB,CC,CD,CG ON
            4'h4: seven = 7'b0011001; // 4: CB,CC,CF,CG ON
            4'h5: seven = 7'b0010010; // 5: CA,CC,CD,CF,CG ON
            4'h6: seven = 7'b0000010; // 6: CA,CC,CD,CE,CF,CG ON
            4'h7: seven = 7'b1111000; // 7: CA,CB,CC ON
            4'h8: seven = 7'b0000000; // 8: All ON
            4'h9: seven = 7'b0010000; // 9: CA,CB,CC,CD,CF,CG ON
            // Hex Part
            4'hA: seven = 7'b0001000; // A: CA,CB,CC,CE,CF,CG ON
            4'hB: seven = 7'b0000011; // B: CC,CD,CE,CF,CG ON
            4'hC: seven = 7'b1000110; // C: CA,CD,CE,CF ON
            4'hD: seven = 7'b0100001; // D: CB,CC,CD,CE,CG ON
            4'hE: seven = 7'b0000110; // E: CA,CD,CE,CF,CG ON
```

```
            4'hF: seven = 7'b0001110; // F: CA,CE,CF,CG ON

            default: seven = 7'b1111111; // all off as default
        endcase
    end
endmodule
```

5.3.5. Top Module Part 2 (`top.v`)

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Boston University
// Engineer: Giacomo Cappelletto
//
// Create Date: 11/14/2025 11:23:56 AM
// Design Name: Top Level Part 2
// Module Name: top
// Project Name: Counter Lab 2.2
// Target Devices: NEXYS A7
// Tool Versions:
// Description: Top level module for Lab 2 Part 2
//              Implements 16-bit counter with 7-segment display multiplexing
//
// Dependencies: clock_divider.v, counter16.v, display_control.v,
//               seven_segment_decoder.v, debouncer.v, sync_2ff.v, edge_detect.v
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module top (
    input  wire       clock,      // 100 MHz system clock
    input  wire       reset,         // Active-low global reset
    input  wire       mode_select,      // Mode select switch (0=manual/button, 1=auto/1Hz)
    input  wire       increment,       // Increment button input
    output wire [3:0] digit_select,    // 4-bit digit select (AN[3:0] signals, active-low)
    output wire [6:0] seg,            // 7-bit seven segment output (CA-CG, active-low)
    output wire [3:0] digit_select_off // AN[7:4] - keep second display off (hardware
requirement)
);
    wire clk_1khz;
    wire clk_1hz;

    clock_divider #(.DIVIDE_BY(100_000)) u_clk_div_1khz (
        .clk_in  (clock),
        .reset_n (reset),
        .clk_out (clk_1khz)
    );

    clock_divider #(.DIVIDE_BY(100_000_000)) u_clk_div_1hz (
        .clk_in  (clock),
        .reset_n (reset),
        .clk_out (clk_1hz)
    );
```

```
wire btn_sync;
sync_2ff u_sync (
    .clk     (clock),
    .reset_n (reset),
    .d_async (increment),
    .q_sync  (btn_sync)
);

wire btn_debounced;
debouncer #(.COUNT_MAX(2_000_000)) u_debouncer (
    .clk       (clock),
    .reset_n   (reset),
    .noisy_in  (btn_sync),
    .clean_out (btn_debounced)
);

wire btn_pulse;
edge_detect u_edge_btn (
    .clk       (clock),
    .reset_n   (reset),
    .level_in  (btn_debounced),
    .pulse_out (btn_pulse)
);

wire clk_1hz_pulse;
edge_detect u_edge_1hz (
    .clk       (clock),
    .reset_n   (reset),
    .level_in  (clk_1hz),
    .pulse_out (clk_1hz_pulse)
);

wire inc_pulse;
assign inc_pulse = mode_select ? clk_1hz_pulse : btn_pulse;

wire [15:0] count;
counter16 u_counter (
    .clk       (clock),
    .reset_n   (reset),
    .inc_pulse (inc_pulse),
    .count     (count)
);

wire [3:0] segment_data;
display_control u_display_control (
    .clk_1khz     (clk_1khz),
    .reset_n      (reset),
    .count        (count),
    .digit_select (digit_select),
    .segment_data (segment_data)
);

// Seven segment decoder converts 4-bit hex to 7-segment pattern
seven_segment_decoder u_decoder (
    .hex_digit (segment_data),
```

```
        .seven      (seg)
    );

    // Keep second display (AN[7:4]) always off by driving them high (inactive)
    assign digit_select_off = 4'b1111;

endmodule
```

5.3.6. Constraints File Part 2 (`nexys_a7.xdc`)

```
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { clock }];
#IO_L12P_T1_MRCC_35 Sch=clk100mhz
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {clock}];

##Switches
set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { mode_select }];
#IO_L24N_T3_RS0_15 Sch=sw[0]

##7 segment display
## seg[6:0] = {CG, CF, CE, CD, CC, CB, CA}
## Mapping: seg[0]=CA, seg[1]=CB, seg[2]=CC, seg[3]=CD, seg[4]=CE, seg[5]=CF, seg[6]=CG
set_property -dict { PACKAGE_PIN T10   IOSTANDARD LVCMOS33 } [get_ports { seg[0] }];
#IO_L24N_T3_A00_D16_14 Sch=ca
set_property -dict { PACKAGE_PIN R10   IOSTANDARD LVCMOS33 } [get_ports { seg[1] }]; #IO_25_14
Sch=cb
set_property -dict { PACKAGE_PIN K16   IOSTANDARD LVCMOS33 } [get_ports { seg[2] }]; #IO_25_15
Sch=cc
set_property -dict { PACKAGE_PIN K13   IOSTANDARD LVCMOS33 } [get_ports { seg[3] }];
#IO_L17P_T2_A26_15 Sch=cd
set_property -dict { PACKAGE_PIN P15   IOSTANDARD LVCMOS33 } [get_ports { seg[4] }];
#IO_L13P_T2_MRCC_14 Sch=ce
set_property -dict { PACKAGE_PIN T11   IOSTANDARD LVCMOS33 } [get_ports { seg[5] }];
#IO_L19P_T3_A10_D26_14 Sch=cf
set_property -dict { PACKAGE_PIN L18   IOSTANDARD LVCMOS33 } [get_ports { seg[6] }];
#IO_L4P_T0_D04_14 Sch=cg

## digit_select[3:0] maps to AN[3:0] (anode enables, active-low)
set_property -dict { PACKAGE_PIN J17   IOSTANDARD LVCMOS33 } [get_ports { digit_select[0] }];
#IO_L23P_T3_FOE_B_15 Sch=an[0]
set_property -dict { PACKAGE_PIN J18   IOSTANDARD LVCMOS33 } [get_ports { digit_select[1] }];
#IO_L23N_T3_FWE_B_15 Sch=an[1]
set_property -dict { PACKAGE_PIN T9    IOSTANDARD LVCMOS33 } [get_ports { digit_select[2] }];
#IO_L24P_T3_A01_D17_14 Sch=an[2]
set_property -dict { PACKAGE_PIN J14   IOSTANDARD LVCMOS33 } [get_ports { digit_select[3] }];
#IO_L19P_T3_A22_15 Sch=an[3]
## digit_select_off[3:0] maps to AN[7:4] - keep second display always off
set_property -dict { PACKAGE_PIN P14   IOSTANDARD LVCMOS33 } [get_ports
{ digit_select_off[0] }]; #IO_L8N_T1_D12_14 Sch=an[4]
set_property -dict { PACKAGE_PIN T14   IOSTANDARD LVCMOS33 } [get_ports
{ digit_select_off[1] }]; #IO_L14P_T2_SRCC_14 Sch=an[5]
set_property -dict { PACKAGE_PIN K2    IOSTANDARD LVCMOS33 } [get_ports
{ digit_select_off[2] }]; #IO_L23P_T3_35 Sch=an[6]
set_property -dict { PACKAGE_PIN U13   IOSTANDARD LVCMOS33 } [get_ports
{ digit_select_off[3] }]; #IO_L23N_T3_A02_D18_14 Sch=an[7]
```

```
##Buttons
## Reset button (CPU_RESETN is active-low, maps directly to reset)
set_property -dict { PACKAGE_PIN C12   IOSTANDARD LVCMOS33 } [get_ports { reset }];
#IO_L3P_T0_DQS_AD1P_15 Sch=cpu_resetn
## Increment button (BTND - down button)
set_property -dict { PACKAGE_PIN P18   IOSTANDARD LVCMOS33 } [get_ports { increment }];
#IO_L9N_T1_DQS_D13_14 Sch=btnd
```

5.4. Testbench Listings

5.4.1. Part 1 Testbench (`counter_8bit_tb.v`)

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Boston University
// Engineer: Giacomo Cappelletto
//
// Create Date: 10/31/2025 01:27:32 PM
// Design Name: Counter Testbench
// Module Name: top_part1_tb
// Project Name: Counter Lab 2.1
// Target Devices:
// Tool Versions:
// Description: Counter Testbench for Counter Lab 2.1
//
// Dependencies: top_part1.v
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module top_part1_tb;
    localparam CLK_PERIOD_NS = 10; // 100 MHz

    reg clk_100mhz = 0;
    reg btn_rst_raw = 0;
    reg btn_inc_raw = 0;
    wire [7:0] led;

    top_part1 dut (
        .clk_100mhz (clk_100mhz),
        .btn_rst_raw(btn_rst_raw),
        .btn_inc_raw(btn_inc_raw),
        .led        (led)
    );

    defparam dut.u_db.COUNT_MAX = 4;

    always #(CLK_PERIOD_NS/2) clk_100mhz = ~clk_100mhz;

    initial begin
        btn_rst_raw = 1'b1;
        repeat (3) @(posedge clk_100mhz);
```

```
        btn_rst_raw = 1'b0;

        repeat (20) @(posedge clk_100mhz);

        // first press
        btn_inc_raw = 1'b1;
        repeat (5) @(posedge clk_100mhz);
        btn_inc_raw = 1'b0;

        repeat (50) @(posedge clk_100mhz);

        // second press
        btn_inc_raw = 1'b1;
        repeat (5) @(posedge clk_100mhz);
        btn_inc_raw = 1'b0;

        repeat (50) @(posedge clk_100mhz);

        btn_rst_raw = 1'b1;
        repeat (3) @(posedge clk_100mhz);
        btn_rst_raw = 1'b0;

        repeat (20) @(posedge clk_100mhz);
        $finish;
    end
endmodule
```

5.4.2. Part 2 Testbench (`top_tb.v`)

```
`timescale 1ns / 1ps

module top_tb;
    localparam CLK_PERIOD_NS = 10;

    reg clock = 0;
    reg reset = 1;
    reg increment = 0;
    reg mode_select = 0;

    wire [3:0] digit_select;
    wire [3:0] digit_select_off;
    wire [6:0] seg;

    top dut (
        .clock            (clock),
        .reset            (reset),
        .increment        (increment),
        .mode_select      (mode_select),
        .digit_select     (digit_select),
        .digit_select_off (digit_select_off),
        .seg              (seg)
    );

    defparam dut.u_clk_div_1khz.DIVIDE_BY = 10;
    defparam dut.u_clk_div_1hz.DIVIDE_BY = 1000;
    defparam dut.u_debouncer.COUNT_MAX = 4;
```

```
    always #(CLK_PERIOD_NS/2) clock = ~clock;

    task press_button;
        begin
            increment = 1'b1;
            repeat (10) @(posedge clock);
            increment = 1'b0;
            repeat (50) @(posedge clock);
        end
    endtask

    initial begin
        reset = 1'b1;
        increment = 1'b0;
        mode_select = 1'b0;
        repeat (10) @(posedge clock);

        reset = 1'b0;
        repeat (20) @(posedge clock);
        reset = 1'b1;
        repeat (20) @(posedge clock);

        mode_select = 1'b0;
        repeat (50) @(posedge clock);

        press_button();
        repeat (100) @(posedge clock);

        press_button();
        repeat (100) @(posedge clock);

        press_button();
        repeat (100) @(posedge clock);

        mode_select = 1'b1;
        repeat (200) @(posedge clock);

        repeat (1200) @(posedge clock);

        repeat (1200) @(posedge clock);

        repeat (1200) @(posedge clock);

        repeat (200) @(posedge clock);
        $finish;
    end

    initial begin
        $monitor("Time: %0t | reset=%b | mode_select=%b | increment=%b | digit_select=%b |
seg=%b",
                 $time, reset, mode_select, increment, digit_select, seg);
    end

endmodule
```