Mälardalen University

DVA494

Lab 2: Multiplexed 7-Segment Display & Continuous Shifter

Spring 2026

1 Introduction and Objectives

This lab focuses on driving a common multiplexed output device on FPGA boards: the 4-digit 7-segment display. Unlike individual LEDs, the four digits share the same segment (cathode) lines; each digit is selected via its anode. To show different symbols on each digit the controller must rapidly activate digits one at a time **(time-division multiplexing)**.

You will apply modular design, entity/component instantiation, and synchronous logic to first build a static scanning display controller, then extend it with a dynamic shifter animation.

Prerequisites

You should be comfortable with: synchronous logic (flip-flops, registers), binary/hexadecimal encoding, basic VHDL entity/architecture structure, and writing simple testbenches.

Hardware

We use the **Basys3 Development board** (100 MHz clock, active-low anodes, active-low segments).

Key Learning Objectives

- Implement a scanning display controller for a multiplexed 4-digit 7-segment display.
- Build a clock-divider that derives an appropriate refresh tick from the 100 MHz system clock.
- Design small building blocks (2-to-4 decoder, 4-to-1 multiplexer) and integrate them.
- Implement a shift-register based shifter to animate digits across the display.

Setup

- Create a new Vivado project targeting Basys3 (Artix-7 XC7A35T-1CPG236C).
- Add the provided constraints file `basys3.xdc` and ensure pins for `{an[3:0], seg[6:0], dp}` and `sw[15:0]` are correctly mapped.
- Use 100 MHz on-board clock as `clk`. Basys3 uses **active-low** for anodes and segments: logic `0` turns on the corresponding LED segment.

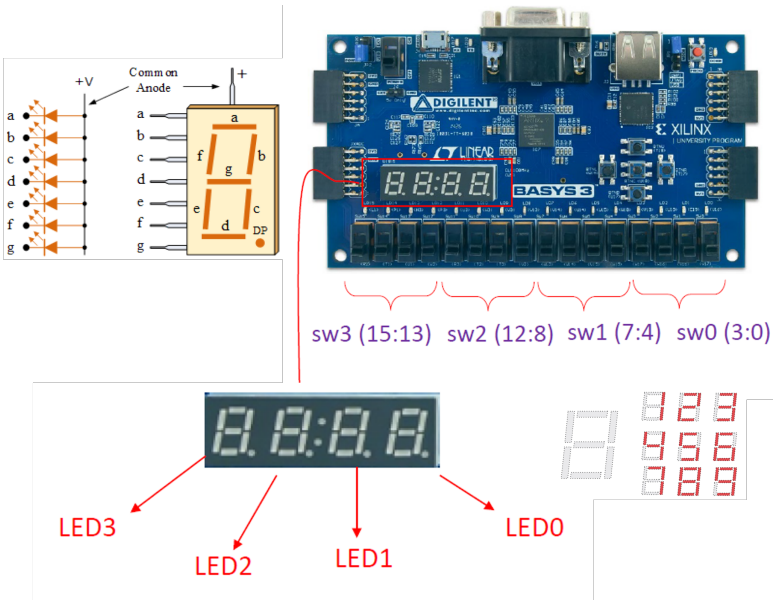
2 Part 1: Static Display Driver

2.1 Theory of Operation

Basys3 has four 7-segment digits. The wiring pattern is:

- **Segments (A–G):** shared by all digits; these lines determine which segment bars light up.
- **Anodes (AN0–AN3):** one per digit; used to select which digit is active (**active-low**).

Hex	Common Anode	a	b	c	d	e	f	g	Cathode(6:0)
0	high	low	low	low	low	low	low	high	0000001
1	high	high	low	low	high	high	high	high	1001111
2	high	low	low	high	low	low	high	low	0010010
3	high	low	low	low	low	high	high	low	0000110
4	high	low	low	low	high	high	low	low	1001100
5	high	low	high	low	low	high	low	low	0100100
6	high	low	high	low	low	low	low	low	0100000
7	high	low	low	low	high	high	high	high	0001111
8	high	low	low	low	low	low	low	low	0000000
9	high	low	low	low	low	high	low	low	0000100
A	high	low	low	low	low	low	high	low	0000010
B	high	high	high	low	low	low	low	low	1100000
C	high	low	high	high	low	low	low	high	0110001
D	high	high	low	low	low	low	high	low	1000010
E	high	low	high	high	low	low	low	low	0110000
F	high	low	high	high	high	low	low	low	0111000



To display `"1234"` the controller cycles through digit selects fast enough that persistence of vision makes all digits appear lit simultaneously. Human fusion threshold is roughly 50–60 Hz for flicker; with 4 digits we target an overall frame rate $f_{\text{frame}} \geq 200\,\text{Hz}$ (each digit about 50 Hz). Generate a per-digit dwell time and a refresh tick so that:

$$f_{\text{digit}} = \frac{1}{T_{\text{dwell}}}, \quad f_{\text{frame}} = 4\, f_{\text{digit}}, \quad N_{\text{divider}} = \frac{f_{\text{clk}}}{f_{\text{digit}}}$$

For a 100 MHz clock and $f_{digit} = 250\,\text{Hz}$ ($T_{dwell} = 4\,\text{ms}$), $f_{frame} \approx 1000\,\text{Hz}$ and $N_{divider} = 400{,}000$.

On each refresh tick:

1. Activate the next anode (active-low): `AN = 1110, 1101, 1011, 0111`.
2. Drive the segment lines with the pattern for the selected digit's nibble.

2.2 Module Specification

Create a module `display_driver` with these ports as specified in the snippet below:

```
entity display_driver is
    port (
        clk      : in std_logic;  -- System clock (100 MHz)
        reset_n  : in std_logic;  -- Active-low reset
        digit1, digit2, digit3, digit4   : in std_logic_vector(3 downto 0); -- 4 BCD digits (16 bits)
        an       : out std_logic_vector(3 downto 0); -- Anode control for 7-segment display
        sseg     : out std_logic_vector(6 downto 0)  -- 7-segment output
    );
end display_driver;
-----------------------
entity display_digits is
    port (
        clk     : in  std_logic;                      -- 100 MHz on Basys3
        sw      : in  std_logic_vector(15 downto 0);  -- 16 board switches
        an      : out std_logic_vector(3 downto 0);   -- Anodes (active low)
        seg     : out std_logic_vector(6 downto 0);   -- Segments a..g (active low)
        dp      : out std_logic                       -- Decimal point (active low)
    );
end display_digits;
```

2.3 Implementation Tasks

Work through the steps below. Aim for clean, modular VHDL with separate processes for sequential and combinational logic. Instantiate components where appropriate.

1. **Top-Level `display_digits`:** Implement the required top-level module `display_digits` with ports strictly as specified above. Internally instantiate `display_driver`.

   Source the four BCD nibbles from the Basys3 switches: `digit1 <= SW(3 downto 0)`, `digit2 <= SW(7 downto 4)`, `digit3 <= SW(11 downto 8)`, `digit4 <= SW(15 downto 12)`.
2. **Clock Divider:** Divide 100 MHz to your chosen per-digit dwell tick (e.g., 250 Hz). Parameterize via a generic if desired.
3. **Scanner Counter:** 2-bit counter cycling $00 \to 01 \to 10 \to 11$ on each tick to select the active digit.
4. **Anode Decoder:** Map counter value to active-low anode patterns: $00 \to$ `1110`, $01 \to$ `1101`, $10 \to$ `1011`, $11 \to$ `0111`.
5. **Digit Multiplexer:** Select the corresponding nibble from `digit1`..`digit4` based on the scanner counter.
6. **Segment Decoder:** Convert the selected nibble (0..F) into 7-bit active-low segment pattern A..G. Reuse a prior hex-to-7seg decoder or implement a lookup table/case statement.
7. **Testbench (required):** Provide `tb_display_digits` that applies a few nibble patterns (driven as if from switches) and verifies segment outputs and anode sequencing. Submodule testbenches are optional but encouraged.
8. **Synthesize & Program:** Generate a bitstream and verify on hardware by setting different values on `SW[15:0]`.

3 Part 2: Continuous Shifter (Dynamic)

Once the static driver works, implement a shifter that scrolls a 4-digit pattern across the display.

3.1 Top-Level Specification

Implement the required top-level module `running_digits`:

```vhdl
entity running_digits is
    generic (
        clk_freq : integer := 100_000_000;  -- 100 MHz
        step_hz  : integer := 1             -- (rotation rate)
    );
    port (
        clk : in  std_logic;
        sw  : in  std_logic_vector(15 downto 0); -- Connect to switches on the Basys3
        an  : out std_logic_vector(3 downto 0);
        seg : out std_logic_vector(6 downto 0);
        dp  : out std_logic
    );
end running_digits;
```

- **Instantiation:** Instantiate your `display_driver` and connect its `digit1`..`digit4` to the rotating register contents.

3.2 Implementation Tasks

1. **Pattern Source (Switches): Always** source the four BCD nibbles from `SW(15:0)` on Basys3.
2. **Speed Timer (≤ 2 Hz):** Implement a counter that generates an enable pulse at a rate selected by `SW(3:0)`. Keep the maximum rotation speed at or below 2 Hz (e.g., a range like 0.25 Hz to 2 Hz). Use a divider based on $N = f_{clk}/f_{step}$.

3. **Shift Register (Circular):** Implement four 4-bit registers. On each enable pulse, perform a **circular left rotation**:
   - `Digit 3` ← `Digit 2`
   - `Digit 2` ← `Digit 1`
   - `Digit 1` ← `Digit 0`
   - `Digit 0` ← `Digit 3` (no new value injected)
4. **Integration:** Connect the four rotated nibbles to `display_driver`'s `digit1` .. `digit4`. Keep `dp` off (`'1'`) unless needed.
5. **Testbench (required):** Provide `tb_running_digits` that accelerates timing (shortened dividers) to observe multiple circular rotations. Verify rotation order, and that display scanning still meets the refresh requirement.
6. **Synthesize & Demo:** Generate bitstream and demonstrate on Basys3.

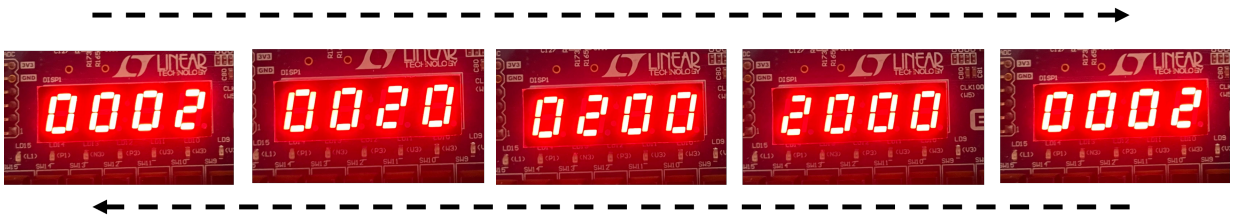


Figure 1: Expected functionality: digits enter from the right and shift left and wrap around (or left to right).

4 Testing & Debugging

- **Active-Low Signals:** Anodes and segments are active-low. A lit segment corresponds to logic `'0'`.
- **Refresh Rate:** If you see flicker, increase per-digit refresh rate (reduce dwell time) and confirm your divider math.
- **Constraints:** Verify `basys3.xdc` pin mappings for `an, seg, dp, sw`. Ensure no unassigned pins.
- **Simulation:** Use faster simulated clocks by reducing divider constants to observe scanning and shifting in reasonable sim time.
- **Modularity:** Keep the display driver independent of the shifter logic so it can be reused.
- **Required Testbenches:** Submit `tb_display_digits` and `tb_running_digits`. Submodule testbenches are optional.

5 Deliverables

- **Synthesizable VHDL Code:** `display_driver.vhd`, top-levels `display_digits.vhd` and `running_digi`
  You may include additional submodules as needed.
- **Required Testbenches:** `tb_display_digits.vhd` and `tb_running_digits.vhd`. (Submodule testbenches optional.)
- **Bitstream:** Program Basys3 and demonstrate both the static display and the running shifter.
- **Short Report:** A few pages summarizing your implementation, simulation, and any challenges you faced. You may use block diagrams and images to enhance you report.