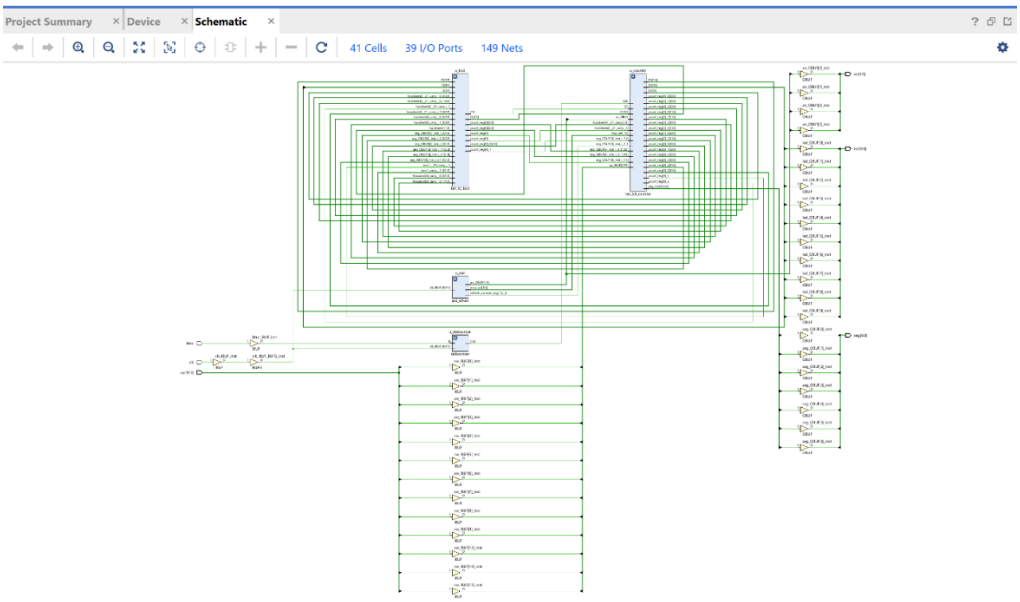
10-Bit Up/Down Counter on Basys3 FPGA

Ethan Suttor
ECE 510/511

Introduction

This ECE 510/511 project is to design and implement a 10-bit synchronous Up/Down counter in VHDL along with the Basys3 FPGA development board. The goal is to reinforce the understanding of digital design principles by combining VHDL elements with actual hardware implementation. The counter can increment or decrement its value within the range of 0 to 1000 based on user input given through push buttons and slide switches. To realize this, the design interacts with several elements of the Basys3 board, including the slide switches (SW0–SW15), the center push button (BTNC), and the 7-segment display. Certain switches are reserved for inputting predetermined values, controlling count direction, and initiating asynchronous or synchronous resets.

The VHDL design has been split into modular components to increase readability and ease scalability, including a counter module, binary-to-BCD converter, and 7-segment display driver. Following is the hierarchical schematic of the design as synthesized in Vivado, highlighting the main components and the signal flow between the modules.



The schematic shows the hierarchical nature of the design, with the distinct segregation in the counter logic, control inputs, BCD conversion, and display interface. This modularity makes it easy to verify, in addition to debugging while testing.

Design Description

The center of this project is a 10-bit synchronous Up/Down counter that is designed and implemented entirely in VHDL and run on the Basys3 FPGA board. The counter dynamically responds to user input, changing its numerical value between 0 and 1000 based on input from the onboard switches and buttons. Every clock pulse, which is triggered by the pressing of the central push button (BTNC), increments or decrements the value of the counter. To ensure that the input signal is free from glitches or bouncing effects that are usually associated with mechanical switches, the BTNC input is passed through a debouncing circuit realized using a D flip-flop.

The operational mode of the counter depends upon the setting of switch SW13. When SW13 is set to logic '0', the counter counts by a value of 1 with every positive edge of the clock. However, when SW13 is set to logic '1', the counter decreases its value. In addition to this, the system includes both a preset and reset mode. The synchronous preset mode is controlled by switch SW14; when this switch is activated and a clock pulse is sensed, the counter will store a pre-set value based upon the 10 least-significant slide switches (SW0–SW9). In contrast, the asynchronous reset function is triggered by switch SW15. This reset command overrides all others at any time and sets the counter to 0 regardless of the clock status or the input levels. To make the counter more understandable to the user, the counter output, initially presented in

binary format, is converted to a four-digit Binary-Coded Decimal (BCD) format. This conversion is done by a standalone VHDL component to provide better ease of visualizing the count on the Basys3 7-segment displays. Time-division multiplexing is used by the system to control displays, and this minimizes flicker and creates a consistent, continuous visual output. Time-division multiplexing involves lighting one digit at a time in rapid succession, taking advantage of the persistence of vision effect to create the illusion of a simultaneous display.

The total design is done by applying a top-down VHDL design methodology. Every component, starting with the counter and the debouncer, followed by the BCD converter and the display driver, is separately designed and debugged in the earlier labs before their integration within a top file. The top file oversees the interaction between the various modules and controls the inputs and outputs appropriately. This not only increases the efficiency of the development cycle but also assists in readability, makes debugging much easier, and gives room in the future to handle more complex projects. The used Xilinx's Vivado Design Suite, with the capability to perform simulation and implementation flows and synthesis. A schematic that was easily expressed by the Vivado tool flow depicted the modular system structure, e.g., counter logic, debounce filter, binary to BCD conversion, and time-multiplexed 7-segment display driver.

The Vivado software played an important role in the project. The Integrated Logic Analyzer (ILA) and waveform viewers were used to identify timing issues and logic correctness. Behavioral simulation in the Vivado simulator provided early indications of module compatibility, and timing analysis and synthesis reports provided insight into resource usage and where potential bottlenecks in terms of performance would be. The design discipline was standard digital engineering practice with revision control, incremental test, and modular test.

The project was done by using design concepts that were expressed in the class. Every action, whether in port mapping, synchronization in the clock domain, or interface to the hardware, was made based upon an understanding of digital systems. The result proved that modularity, documentation, and iterative debugging are all essential to good hardware design.

VHDL Code Overview and Lab Integration

The VHDL design was based upon the general concepts of modular design. Reusability was followed by reusing previously designed code in Labs 6 to 8. Debounce logic in Lab 6 was reused to suppress the mechanical bounce of the push button that was employed to provide the manual clock input. Time-multiplexed 7-segment control of Lab 7 was employed, providing steady visual output. Binary to BCD conversion logic of Lab 8 was employed to convert binary counter output to a format that could be displayed numerically.

Even if the modules performed as desired separately, combining them at the top level had some problems. Every module had different timing assumptions, signal widths, and reset behaviors. Getting all those aspects to match was a matter of signal interface adjustment and uniforming control logic. Port mismatches and clock domain differences proved very problematic at the initial integration. On-board test and iterative simulation were employed to test for functionality. Parameterization was added via VHDL generics to make the design more reusable. The counter bit width and multiplexing rate of the displays were made variable without code restructuring. This accommodates more general use in subsequent designs.

There was a rigorous test process, with every module having been verified against specialized testbenches. Signal assertions, timing verifications, and waveform inspections guaranteed correctness in behavior. The testbenches not only acted as vehicles of verification but also documentation of the module behavior. The architecture was hierarchical. A principal function was placed in each VHDL entity and was instantiated within a top module. Declarative top-level signals enabled ordered directional routing and fault minimization at the integration level. Debugging and expansion of the system were easy with this architecture.

Results and Time Spent

After full integration and synthesis, the system performed as expected on the Basys3 board. The counter reacted appropriately to clock inputs and control inputs, i.e., direction toggling, preset values, and reset. The 7-segment display appropriately displayed the contents of the counter, and time-division multiplexing supplied smooth transitions. Another issue with testing was the screen freezing at "0000". Debugging indicated conflicts between BCD converter output and expectations of the display logic. The issue was fixed by correcting segment mappings and timing logic. A second bug was encountered when the counter to get above its 1000 limit, which was fixed by the inclusion of comparison logic.

The entire project was roughly 10 work hours over a span of two days. This included integration, simulation, synthesizing, programming the board, and documenting. There was a lot of time spent debugging integration issues and confirming behaviors.

FPGA Utilization and Scaling Potential

Synthesis reports indicated that the design only made use of part of resources on the Basys3. Owing to shared clock and centralized control, in theory 104 copies of the counter system could fit on a single chip based on logic constraints rather than I/O constraints. The project used about 117 logic slices and the Basys 3 board has 20800 logic slices:

$$\frac{20800}{117} \approx 117$$

Such resource efficiency is an example to hierarchical VHDL design's power. The modular nature allowed for easy synthesis without sacrificing functionality. The logic resources were mostly occupied by display and counter logic, with debounce and control modules occupying very little space. If implemented on a larger I/O, DSP-enabled FPGA, this design can become a larger, more involved system, a multi-channel signal processor or real-time monitor interface, showing how core design scales to advanced applications.

Development Challenges and Key Insights

The biggest challenge was integrating modules from multiple labs into a single system. Even though they operated as stand-alone modules, integration exposed incompatibilities. Debugging involved tracking signal paths, checking on synchronizations of clocks, as well as making sure there was coherent reset behavior. The biggest big faced was 7-segment non-updating despite updates in counters. A faulty route from BCD output to digit multiplexer was discovered. The simulation and observation of waveforms lead to logic timing and ordering

control fixes. The remainder included logic synthesis and overflow/counter bugs like misplaced signals. The bugs were fixed thorough examination, simulation, and modular verification. Thorough development in iterative phases with documentation made it easier to manage despite the complexity.

This activity reaffirmed my understanding of digital design flows. It demonstrated how HDL-based design brings theoretical concepts into practice. The flow of the entire process, starting with modular design, simulation, integration, and hardware verification, was very much like real engineering flows. The Basys3 board was ideal for experimentation. It enabled rapid prototyping because it has an open interface and is supported by Vivado. In contrast with other environments, Basys3 had an excellent balance when it comes to power, flexibility, and modularity. This assignment made me develop a greater appreciation for VHDL and an improved understanding for how to design with FPGAs. I now appreciate modularity, verification, and documentation in constructing complicated systems. I feel more comfortable with complicated digital design and integrating multiple functional building blocks into large systems.

Appendix

Top_counter_sysyem:

```
-- Top-Level Module for 10-bit Up/Down Counter System

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity top_counter_system is
    Port (
        clk         : in  STD_LOGIC;
        btnc        : in  STD_LOGIC;
        sw          : in  STD_LOGIC_VECTOR(15 downto 0);
        seg         : out STD_LOGIC_VECTOR(6 downto 0);
        an          : out STD_LOGIC_VECTOR(3 downto 0);
        led         : out STD_LOGIC_VECTOR(9 downto 0)

    );
end top_counter_system;

architecture Behavioral of top_counter_system is

    -- Internal signals
    signal debounced_clk : STD_LOGIC;
    signal counter_out    : STD_LOGIC_VECTOR(9 downto 0);
    signal bcd_out        : STD_LOGIC_VECTOR(15 downto 0);
```

```
component debouncer
   Port (
      clk     : in  STD_LOGIC;
      btn_in  : in  STD_LOGIC;
      btn_out : out STD_LOGIC
   );
end component;

component ten_bit_counter
   Port (
      clk       : in  STD_LOGIC;
      rst_async  : in  STD_LOGIC;
      preset     : in  STD_LOGIC;
      up_down    : in  STD_LOGIC;
      preset_val : in  STD_LOGIC_VECTOR(9 downto 0);
      count_out  : out STD_LOGIC_VECTOR(9 downto 0)
   );
end component;

component bin_to_bcd
   Port (
      bin_in  : in  STD_LOGIC_VECTOR(9 downto 0);
      bcd_out : out STD_LOGIC_VECTOR(15 downto 0)
   );
end component;

component ssd_driver
   Port (
      clk     : in  STD_LOGIC;
```

```
        bcd_in  : in  STD_LOGIC_VECTOR(15 downto 0);
        seg     : out STD_LOGIC_VECTOR(6 downto 0);
        an      : out STD_LOGIC_VECTOR(3 downto 0)
    );
end component;

begin


u_debouncer: debouncer
    Port map (
        clk     => clk,
        btn_in  => btnc,
        btn_out => debounced_clk
    );

u_counter: ten_bit_counter
    Port map (
        clk        => debounced_clk,
        rst_async  => sw(15),
        preset     => sw(14),
        up_down    => sw(13),
        preset_val => sw(9 downto 0),
        count_out  => counter_out
    );

u_bcd: bin_to_bcd
    Port map (
        bin_in  => counter_out,
```

```
            bcd_out => bcd_out
        );

  u_ssd: ssd_driver
      Port map (
      clk    => clk,
      bcd_in => bcd_out,
       seg    => seg,
       an     => an
      );

    led <= counter_out;

end Behavioral;

u_bouncer
-- Debouncer Module for Button Clock Signal

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity debouncer is
    Port (
        clk     : in  STD_LOGIC;
        btn_in  : in  STD_LOGIC;
        btn_out : out STD_LOGIC
    );
end debouncer;
```

```
architecture Behavioral of debouncer is
    signal shift_reg : STD_LOGIC_VECTOR(15 downto 0) := (others => '0');
    signal debounced : STD_LOGIC := '0';
begin

    process(clk)
    begin
        if rising_edge(clk) then
            shift_reg <= shift_reg(14 downto 0) & btn_in;
            if shift_reg = X"FFFF" then
                debounced <= '1';
            elsif shift_reg = X"0000" then
                debounced <= '0';
            end if;
        end if;
    end process;

    btn_out <= debounced;

end Behavioral;
```

u_counter:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ten_bit_counter is
    Port (
        clk        : in  STD_LOGIC;
        rst_async  : in  STD_LOGIC;
        preset     : in  STD_LOGIC;
        up_down    : in  STD_LOGIC;
        preset_val : in  STD_LOGIC_VECTOR(9 downto 0);
        count_out  : out STD_LOGIC_VECTOR(9 downto 0)
    );
end ten_bit_counter;

architecture Behavioral of ten_bit_counter is
    signal count : STD_LOGIC_VECTOR(9 downto 0) := (others => '0');
begin

    process(clk, rst_async)
    begin
        if rst_async = '1' then
            count <= (others => '0');

        elsif rising_edge(clk) then
            if preset = '1' then
                if preset_val > "1111101000" then  -- 1000 in binary
```

```
    count <= "1111101000";          -- force to 1000
else
    count <= preset_val;
end if;


            else
                if up_down = '0' then
    if count = 1000 then
        count <= (others => '0');
    else
        count <= count + 1;
    end if;
else  -- Counting down
    if count = 0 then
        count <= "1111101000";
    else
        count <= count - 1;
    end if;
end if;


            end if;
        end if;
    end process;


    count_out <= count;


end Behavioral;
bin_to_bcd:
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity bin_to_bcd is
    Port (
        bin_in  : in  STD_LOGIC_VECTOR(9 downto 0);
        bcd_out : out STD_LOGIC_VECTOR(15 downto 0)
    );
end bin_to_bcd;

architecture Behavioral of bin_to_bcd is
    signal temp_bin : INTEGER range 0 to 1023;
    signal thousands, hundreds, tens, ones : INTEGER range 0 to 9;
begin

    process(bin_in)
    begin
        temp_bin <= CONV_INTEGER(bin_in);

        thousands <= temp_bin / 1000;
        hundreds <= (temp_bin mod 1000) / 100;
        tens <= (temp_bin mod 100) / 10;
        ones <= temp_bin mod 10;

        bcd_out <= CONV_STD_LOGIC_VECTOR(thousands, 4) &
                   CONV_STD_LOGIC_VECTOR(hundreds, 4) &
                   CONV_STD_LOGIC_VECTOR(tens, 4) &
                   CONV_STD_LOGIC_VECTOR(ones, 4);
    end process;
```

```
end Behavioral;
```

ssd_driver:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ssd_driver is
    Port (
        clk     : in  STD_LOGIC;
        bcd_in  : in  STD_LOGIC_VECTOR(15 downto 0);
        seg     : out STD_LOGIC_VECTOR(6 downto 0);
        dp      : out STD_LOGIC;
        an      : out STD_LOGIC_VECTOR(3 downto 0)
    );
end ssd_driver;

architecture Behavioral of ssd_driver is

    signal refresh_counter : STD_LOGIC_VECTOR(19 downto 0) := (others => '0');
    signal mux_sel : STD_LOGIC_VECTOR(1 downto 0);
    signal digit : STD_LOGIC_VECTOR(3 downto 0);

begin

    process(clk)
    begin
        if rising_edge(clk) then
```

```
        refresh_counter <= refresh_counter + 1;
    end if;
end process;

mux_sel <= refresh_counter(19 downto 18);

process(mux_sel, bcd_in)
begin
    case mux_sel is
        when "00" => digit <= bcd_in(3 downto 0);
        when "01" => digit <= bcd_in(7 downto 4);
        when "10" => digit <= bcd_in(11 downto 8);
        when others => digit <= bcd_in(15 downto 12);
    end case;
end process;

process(mux_sel)
begin
    case mux_sel is
        when "00" => an <= "1110";
        when "01" => an <= "1101";
        when "10" => an <= "1011";
        when others => an <= "0111";
    end case;
end process;

process(digit)
begin
    case digit is
        when "0000" => seg <= "1000000";
```

```
            when "0001" => seg <= "1111001";
            when "0010" => seg <= "0100100";
            when "0011" => seg <= "0110000";
            when "0100" => seg <= "0011001";
            when "0101" => seg <= "0010010";
            when "0110" => seg <= "0000010";
            when "0111" => seg <= "1111000";
            when "1000" => seg <= "0000000";
            when "1001" => seg <= "0010000";
            when others => seg <= "1111111";
        end case;
    end process;

    dp <= '1';

end Behavioral;
```