Using memory

Toki Nishikawa

November 8 2022
6-8pm Tuesday lab with Kim Nguyen/Nick Boudreau

The objective of the lab is to create a blinking animation on a seven-segment display.

1 Testing

Initially, none of the LEDs on the seven-segment display were turning on. I checked the code associated with the *sevenseg* output of the top-level module and found that the *rom_out* output of the ROM module was being port-mapped to a signal that I had previously decided not to use. However, there were still no lights after this change. I decided to check the voltages of each of the seven segments to ensure that they were all being set high as opposed to floating. They were all set high. I suspected that the 'when others' case in the ROM module was constantly being triggered since this sets *rom_out* to '1111111'. Thus, I changed the 'when others' case to set *rom_out* to '0000000' instead. No lights. Now, I was very puzzled. How could all seven segments be high when that case doesn't exist?

I decided to check the voltages of the anode pins - they were also both high. Again, there is no case were the anode pins are both high, and so this was very confusing. My plan then was to see if I could output any specified value. I deleted the logic associated with the anode pins and set the *rom_anode* output in the ROM module, which is port-mapped to the *anode* output in the top-level module, to '01'. Both were still high. Then, I deleted *rom_anode* from the ROM module and set *anode* to '01'. Still, both were high. I was very, very confused. I tried outputting logic values to other random pins on the FPGA, and they were always high. Time to call for help. It took Professor Bell around five minutes to realize that I was uploading the wrong file to my FPGA.

Now that I was uploading the correct file, I was getting lights. However, only segment C on digit 1 was turned on, and nothing was changing. A diagram of the letters associated with each segment is shown in **Figure 1**. I hit reset, and it started flashing between '-', corresponding to '1111110', and '8', corresponding to '0000000', on digit 1. It seemed that the display was alternating between the first and last case, '00000' and 'others'. Note that, at this point in time, the 'others' case was still set to '0000000'. I checked the sensitivity list for the first process in the ROM module and saw that it was 'rom_clock'. Since we want *rom_out* to update whenever the address changes, it should be 'adr' or 'all'. However, the output was the same after this fix.

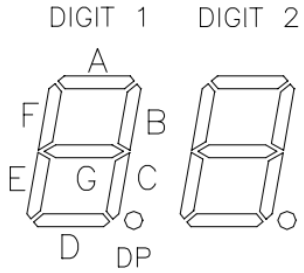

Figure 1: Letters associated with each segment

Next, I checked my counter module to ensure that its output *count* was incrementing correctly. The initial logic is displayed below:

```
process (counter_clk) begin
    if rising_edge(counter_clk) then
        if (counter_reset = '0') then
            count <= "00000";
        else
            a <= a + 1;
            if (a(25) = '1') then
                count <= count + 1;
            else
                count <= count;
            end if;
        end if;
    end if;
end process;
```

This code increments *count* when a(25) is 1, and resets *count* when a push-button is pressed. Note that, in this code, *a* is a signal with 26 bits. It's evident what the issue is: *count* will not increment for a while until a(25) becomes 1, at which point it will increment super fast on every rising edge of the clock until *a* rolls over. The correct way to implement the counter is shown below:

```
process (counter_clk) begin
    if rising_edge(counter_clk) then
        if (counter_reset = '0') then
            a <= "00000000000000000000000000";
        else
            a <= a + 1;
            count(4) <= a(27);
            count(3) <= a(26);
            count(2) <= a(25);
            count(1) <= a(24);
            count(0) <= a(23);
        end if;
    end if;
end process;
```

In contrast, this code assigns the five most significant bits of *a* to the five bits in *count*, incrementing *a* on every rising edge of the clock. When the reset button is pressed, *a* is reset rather than *count*. In this code, *a* is a signal with 28 bits so that the segments switch at a nice speed. Working through the logic in your head, you should be able to visualize how the code slowly and steadily increments *count*. Since the cases only go up to a *count* of '10000', there is some time before the animation will reset itself. This is why I chose to include a reset button.

After rewriting the counter logic, I was finally seeing an animation. The first problem I noticed was that the digits were flipped. That was easy - I just had to change the order of the bits that I was assigning to *rom_anode*. The second problem was that the animation was skipping the first segment displayed on the second digit. This was clearly due to an error in the logic in the ROM module that assigns the appropriate digit to each case. Originally, the first if statement, which tells *rom_anode* to light up digit 1, was executed when 'adr < 9'. However, that includes the case '01000', which corresponds to the first segment displayed on the second digit. Thus, I changed it to 'adr < 8', and everything worked.

2 VHDL code

2.1 Top Module

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity lab7 is
    port(
        reset : in std_logic;
        sevenseg : out std_logic_vector(6 downto 0); -- configuration of sevenseg
        anode : out std_logic_vector(1 downto 0) := "01" -- configuration of anode pins
    );
end;

architecture synth of lab7 is

    signal adr_temp : unsigned(4 downto 0);
    signal clk : std_logic;

    component HSOSC is
        generic(
            CLKHF_DIV : String := "0b00"
        );
        port(
            CLKHFPU : in std_logic := 'X';
            CLKHFEN : in std_logic := 'X';
            CLKHF : out std_logic := 'X'
        );
    end component;


    component counter is
```

```
        port(
            counter_reset : in std_logic;
            counter_clk : in std_logic;
            count : out unsigned(4 downto 0)
        );
    end component;

    component rom is
        port(
            adr : in unsigned(4 downto 0);
            rom_clk : in std_logic;
            rom_out : out std_logic_vector(6 downto 0);
            rom_anode : out std_logic_vector(1 downto 0)
        );
    end component;



begin

    HSOSC_inst : HSOSC
        port map (
            CLKHF => clk,
            CLKHFPU => '1',
            CLKHFEN => '1'
        );

    counter_inst : counter
        port map(
            counter_reset => reset,
            counter_clk => clk,
            count => adr_temp
        );
    rom_inst : rom
        port map(
            adr => adr_temp,
            rom_clk => clk,
            rom_out => sevenseg,
            rom_anode => anode
        );


end;
```

2.2 Counter Module

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity counter is
    port(
        counter_reset : in std_logic := '1';
        counter_clk : in std_logic;
        count : out unsigned(4 downto 0) := "00000"
    );
end;

architecture synth of counter is

    signal a : unsigned (27 downto 0) := "0000000000000000000000000000";

begin

    process (counter_clk) begin
        if rising_edge(counter_clk) then
            if (counter_reset = '0') then
                a <= "0000000000000000000000000000";
            else
                a <= a + 1;
                count(4) <= a(27);
                count(3) <= a(26);
                count(2) <= a(25);
                count(1) <= a(24);
                count(0) <= a(23);
            end if;
        end if;
    end process;
end;
```

2.3 ROM Module

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity rom is
    port(
        adr : in unsigned(4 downto 0);
        rom_clk : in std_logic;
```

```
        rom_out : out std_logic_vector(6 downto 0);
        rom_anode : out std_logic_vector(1 downto 0)
    );
end rom;

architecture synth of rom is


begin

    process(all) begin
        case adr is
            when "00000" => rom_out <= "1111110";
            when "00001" => rom_out <= "1111101";
            when "00010" => rom_out <= "0111111";
            when "00011" => rom_out <= "1011111";
            when "00100" => rom_out <= "1101111";
            when "00101" => rom_out <= "1110111";
            when "00110" => rom_out <= "1111011";
            when "00111" => rom_out <= "1111110";
            when "01000" => rom_out <= "1111110";
            when "01001" => rom_out <= "1011111";
            when "01010" => rom_out <= "0111111";
            when "01011" => rom_out <= "0111111";
            when "01100" => rom_out <= "1111101";
            when "01101" => rom_out <= "1111011";
            when "01110" => rom_out <= "1110111";
            when "01111" => rom_out <= "1101111";
            when "10000" => rom_out <= "1111110";
            when others => rom_out <= "1111111";
        end case;
    end process;


    process (rom_clk) begin
        if rising_edge(rom_clk) then
            if (adr < 8) then -- when count is less than 01000 (9), we want the first digit
                rom_anode <= "01";
            else
                rom_anode <= "10";
            end if;
        end if;
    end process;
end;
```