LAB REPORT
******On Counters And Timers******

3220101111 洪晨辉

****2024.12.23****

1. Problem Description

1) Design and implement a 3-digit BCD counter. The counter should display its value on the 7-segment displays, HEX2–HEX0. Use the 50 MHz clock signal from the DE2 board to generate a one-second control signal that increments the counter. Additionally, implement functionality to reset the counter to 0 when the push-button switch KEY0 is pressed.

2) Create and implement a time-of-day clock circuit for the DE2 board. The clock should display the hours (0 to 23) on 7-segment displays HEX7–6, the minutes (0 to 59) on HEX5–4, and the seconds (0 to 59) on HEX3–2. Use switches SW15–0 to set the initial hour and minute values shown on the clock.

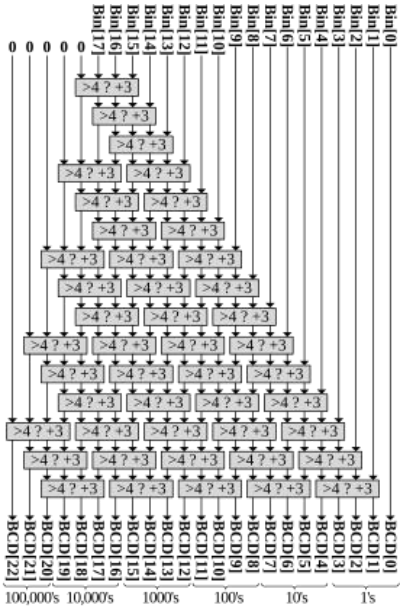
3)Design and implement a reaction-timer circuit on the DE2 board. The circuit should reset when the push button switch KEY0 is pressed. After a delay, determined by the value set in seconds on switches SW7–0, the red LED labeled LEDR0 will turn on, and a four-digit BCD counter will begin counting in intervals of milliseconds. A person being tested must press the push button KEY3 as quickly as possible to turn off the LED and freeze the counter in its current state. The resulting count, representing the reaction time, will be displayed on the 7-segment displays HEX2–0.

2. Design Formulation

Part I

The truth table method is proved too bulky for modification， hence the need of a more portable algorithm.**Double dabble method**, though often accused of being redundant, is the perfect sample in this case, which begs the question that why one haven't foreseen such urgency in the previous development.

For each group of input four bits:
If group >= 5 add 3 to group
Left shift into the output digits

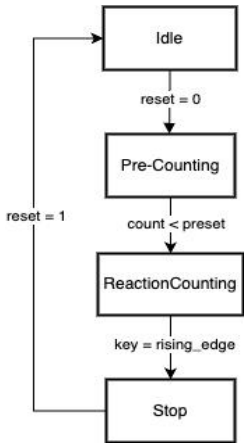


Part II

Nothing particular. The hour and minute is assigned at the very beginning, then rolling with the counter. The seven segment display reads the binary time converted into BCD.

Part III

The circuit has been implemented bearing similarity of a elementary state machine, with counting stage, reaction counting stage threaded in a linear structure.



3. Design Entry

Part I

char_7seg

```
1.  library ieee;
2.  use ieee.std_logic_1164.all;
3.
4.  entity char_7seg is
5.      port (
6.          c : in std_logic_vector(3 downto 0);
     -- 4-bit input character code
7.          display : out std_logic_vector(7 dow
    nto 0) -- 7-segment output
8.      );
9.  end char_7seg;
10.
11. architecture behavior of char_7seg is
12. begin
13.     process(c)
14.     begin
15.         case c is
16.             when "0000" => display <= "11000
    000"; -- 0
17.             when "0001" => display <= "11111
    001"; -- 1
18.             when "0010" => display <= "10100
    100"; -- 2
19.             when "0011" => display <= "10110
    000"; -- 3
20.             when "0100" => display <= "10011
    001"; -- 4
21.             when "0101" => display <= "10010
    010"; -- 5
22.             when "0110" => display <= "10000
    010"; -- 6
23.             when "0111" => display <= "11111
    000"; -- 7
24.             when "1000" => display <= "10000
    000"; -- 8
25.             when "1001" => display <= "10010
    000"; -- 9
26.             when others => display <= "11111
    111"; -- Default to blank
27.         end case;
28.     end process;
29. end behavior;
```

Counter

```
1.  library IEEE;
2.  use IEEE.STD_LOGIC_1164.ALL;
3.  use IEEE.NUMERIC_STD.ALL; -- For unsigned ar
    ithmetic
4.
5.  entity Counter is
6.      Port (
7.          clk : in STD_LOGIC; -- Clock signal
8.          rst : in STD_LOGIC; -- Reset signal
9.          hex2, hex1, hex0 : out STD_LOGIC_VEC
    TOR(7 downto 0) -- 7-segment displays
10.     );
11. end Counter;
12.
13. architecture Behavioral of Counter is
14.     -- Component declaration for 7-segment d
    ecoder
```

```
    component char_7seg
        port (
            c : in std_logic_vector(3 downto 0); -- 4-bit BCD input
            display : out std_logic_vector(7 downto 0) -- 7-segment output
        );
    end component;

    -- Signals for the counter and BCD representation
    signal counter : unsigned(9 downto 0) := "0000000000"; -- 10-bit counter for 0 to 999
    signal bcd_hundreds : STD_LOGIC_VECTOR(3 downto 0);
    signal bcd_tens : STD_LOGIC_VECTOR(3 downto 0);
    signal bcd_ones : STD_LOGIC_VECTOR(3 downto 0);
    signal clock_divider : integer := 0;
    signal slow_clock : STD_LOGIC;

begin
 process (clk,rst)
    begin
  if rst = '1' then
   clock_divider <= 0;
        elsif rising_edge(clk) then
            if clock_divider = 10 - 1 then -- Board frequency 50MHz, this is a faster version for waveform simulation convenience
                slow_clock <= not slow_clock;
                clock_divider <= 0;
            else
                clock_divider <= clock_divider + 1;
            end if;
        end if;
    end process;

    -- Process for the counter with clock and reset
    process(slow_clock, rst)
    begin
        if rst = '1' then
            counter <= (others => '0'); -- Reset counter to 0
        elsif rising_edge(slow_clock) then
            if counter = 999 then
                counter <= (others => '0'); -- Wrap around after 999
            else
                counter <= counter + 1; -- Increment counter
            end if;
        end if;
    end process;

    -- Binary-to-BCD conversion using Double-Dabble
    process(counter)
        variable bin : unsigned(9 downto 0); -- Binary input as unsigned
        variable bcd : unsigned(11 downto 0); -- BCD representation (3 digits)
    begin
        -- Initialize variables
        bin := counter;
        bcd := (others => '0');

        -- Shift and adjust algorithm
        for i in 9 downto 0 loop
            -- Adjust BCD digits if >= 5
            if bcd(11 downto 8) >= "0101" then
                bcd(11 downto 8) := bcd(11 downto 8) + 3;
            end if;
            if bcd(7 downto 4) >= "0101" then
                bcd(7 downto 4) := bcd(7 downto 4) + 3;
            end if;
            if bcd(3 downto 0) >= "0101" then
```

```
                bcd(3 downto 0) := bcd(3 downto 0) + 3;
            end if;

            -- Shift BCD and binary left by 1 bit
            bcd := bcd(10 downto 0) & bin(9);
            bin := bin(8 downto 0) & '0';
        end loop;

        -- Assign BCD digits to output signals
        bcd_hundreds <= std_logic_vector(bcd(11 downto 8));
        bcd_tens <= std_logic_vector(bcd(7 downto 4));
        bcd_ones <= std_logic_vector(bcd(3 downto 0));
    end process;

    -- Instantiate 7-segment decoders
    Hex_2: char_7seg
        port map (c => bcd_hundreds, display => hex2);

    Hex_1: char_7seg
        port map (c => bcd_tens, display => hex1);

    Hex_0: char_7seg
        port map (c => bcd_ones, display => hex0);

end Behavioral;
```

Part II

char_7seg remains the same with two new VHDL file added,

BCD

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

entity BCD is
    Port (
        binary_in : in STD_LOGIC_VECTOR(5 downto 0); -- 6-bit binary input
     hex1,hex0 : out STD_LOGIC_VECTOR(7 downto 0)
    );
end BCD;

architecture TruthTable of BCD is

 -- Component declarations
    component char_7seg
        port (
            c : in std_logic_vector(3 downto 0);   -- 3-bit input for the character
            display : out std_logic_vector(7 downto 0) -- 7-segment output
        );
    end component;

 signal bcd_tens : STD_LOGIC_VECTOR(3 downto 0);
 signal bcd_ones  : STD_LOGIC_VECTOR(3 downto 0);

begin
    -- Map binary input to BCD tens digit (MSB)
    with binary_in select
        bcd_tens <=
            "0000" when "000000" | "000001" | "000010" | "000011" |
                        "000100" | "000101" | "000110" | "000111" |
                        "001000" | "001001",
            "0001" when "001010" | "001011" | "001100" | "001101" |
                        "001110" | "001111" | "010000" | "010001" |
                        "010010" | "010011",
            "0010" when "010100" | "010101" | "010110" | "010111" |
                        "011000" | "011001" | "011010" | "011011" |
```

```
                        "011100" | "011101" ,
            "0011" when "011110" | "011111"
| "100000" | "100001" |
      "100010" | "100011" | "100100" | "1001
01" |
      "100110" | "100111" ,
            "0100" when "101000" | "101001"
| "101010" | "101011" |
      "101100" | "101101" | "101110" | "1011
11" |
      "110000" | "110001" ,
            "0101" when "110010" | "110011"
| "110100" | "110101" |
      "110110" | "110111" | "111000" | "1110
01" |
      "111010" | "111011" ,
            "0110" when "111100" | "111101"
| "111110" | "111111",
            "0000" when others;

    -- Map binary input to BCD ones digit (L
SB)
    with binary_in select
        bcd_ones <=
            "0000" when "000000" | "001010"
| "010100" | "011110" |
                        "101000" | "110010" |
 "111100",
            "0001" when "000001" | "001011"
| "010101" | "011111" |
                        "101001" | "110011" |
 "111101",
            "0010" when "000010" | "001100"
| "010110" | "100000" |
                        "101010" | "110100" |
 "111110",
            "0011" when "000011" | "001101"
| "010111" | "100001" |
                        "101011" | "110101" |
 "111111",
            "0100" when "000100" | "001110"
| "011000" | "100010" |
                        "101100" | "110110",
            "0101" when "000101" | "001111"
| "011001" | "100011" |
                        "101101" | "110111",
            "0110" when "000110" | "010000"
| "011010" | "100100" |
                        "101110" | "111000",
            "0111" when "000111" | "010001"
| "011011" | "100101" |
                        "101111" | "111001",
            "1000" when "001000" | "010010"
| "011100" | "100110" |
                        "110000" | "111010",
            "1001" when "001001" | "010011"
| "011101" | "100111" |
                        "110001" | "111011",
            "0000" when others;

    Hex_0: char_7seg
        port map (c => bcd_ones, display =>
hex0);

    Hex_1: char_7seg
        port map (c => bcd_tens, display =>
hex1);

end TruthTable;
```

Clock

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL; -- For unsigned ar
ithmetic

entity Clock is
    Port (
        clk50MHz : in STD_LOGIC;           -
- 50 MHz clock input
        set    : in STD_LOGIC;           --
Asynchronous reset
        SW       : in STD_LOGIC_VECTOR(15 do
wnto 0); -- Switch inputs for setting time
        HEX7     : out STD_LOGIC_VECTOR(7 do
wnto 0); -- Hour tens digit
        HEX6     : out STD_LOGIC_VECTOR(7 do
wnto 0); -- Hour units digit
```

```
        HEX5     : out STD_LOGIC_VECTOR(7 downto 0); -- Minute tens digit
        HEX4     : out STD_LOGIC_VECTOR(7 downto 0); -- Minute units digit
        HEX3     : out STD_LOGIC_VECTOR(7 downto 0); -- Second tens digit
        HEX2     : out STD_LOGIC_VECTOR(7 downto 0)  -- Second units digit
    );
end Clock;

architecture Behavioral of Clock is

 component char_7seg
  port (
   c : in std_logic_vector(3 downto 0); -- 4-bit BCD input
   display : out std_logic_vector(7 downto 0) -- 7-segment output
  );
    end component;

    component BCD
  Port (
   binary_in : in STD_LOGIC_VECTOR(5 downto 0); -- 6-bit binary input
   hex1,hex0 : out STD_LOGIC_VECTOR(7 downto 0)
  );
 end component;

    signal clk_1Hz    : STD_LOGIC := '0'; -- 1 Hz clock signal
    signal counter    : INTEGER range 0 to 49_999_999 := 0; -- Clock divider counter
    signal seconds    : INTEGER range 0 to 59 := 0; -- Seconds counter
    signal minutes    : INTEGER range 0 to 59 := 0; -- Minutes counter
    signal hours      : INTEGER range 0 to 23 := 0; -- Hours counter

 signal hours_bi   : std_logic_vector(5 downto 0);
 signal minutes_bi : std_logic_vector(5 downto 0);
 signal seconds_bi : std_logic_vector(5 downto 0);

begin

    -- Clock Divider: 50 MHz to 1 Hz
    process (clk50MHz, set)
    begin
        if set = '1' then
            counter <= 0;
            clk_1Hz <= '0';
        elsif rising_edge(clk50MHz) then
            if counter = 9 then --if counter = 49_999_999 then
                counter <= 0;
                clk_1Hz <= not clk_1Hz;
            else
                counter <= counter + 1;
            end if;
        end if;
    end process;

    -- Seconds Counter
    process (clk_1Hz, SW, set)
    begin
        if set = '1' then
            seconds <= 0;
            hours <= to_integer(unsigned(SW(15 downto 12))) * 10 + to_integer(unsigned(SW(11 downto 8)));
   minutes <= to_integer(unsigned(SW(7 downto 4))) * 10 + to_integer(unsigned(SW(3 downto 0)));
        elsif rising_edge(clk_1Hz) then
            if seconds = 59 then
                seconds <= 0;
                if minutes = 59 then
                    minutes <= 0;
                    if hours = 23 then
                        hours <= 0;
                    else
                        hours <= hours + 1;
```

```
                    end if;
                else
                    minutes <= minutes + 1;
                end if;
            else
                seconds <= seconds + 1;
            end if;
        end if;

    end process;

 hours_bi <= std_logic_vector(to_unsigned(hours, 6));
 minutes_bi <= std_logic_vector(to_unsigned(minutes, 6));
 seconds_bi <= std_logic_vector(to_unsigned(seconds, 6));

 BCD_hours : BCD
  port map (binary_in => hours_bi, hex1 =>hex7, hex0 => hex6);

 BCD_minutes : BCD
  port map (binary_in => minutes_bi, hex1 => hex5, hex0 => hex4);

BCD_seconds : BCD
 port map (binary_in => seconds_bi, hex1 => hex3, hex0 => hex2);

end Behavioral;
```

Part III

With char_7seg and BCD retains, only the top layer structure was modified

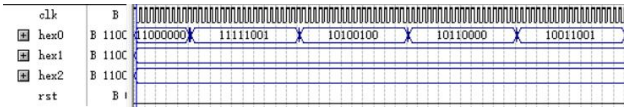
Reactivity(It's a bad name, I know)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity Reactivity is
    Port (
        clk50MHz : in STD_LOGIC;          -- 50 MHz clock input
        reset    : in STD_LOGIC;          -- KEY0: Reset
        stop     : in STD_LOGIC;          -- KEY3: Stop the timer
        SW       : in STD_LOGIC_VECTOR(7 downto 0); -- Set delay time in seconds
        LEDR0    : out STD_LOGIC;         -- Red LED output
        HEX3     : out STD_LOGIC_VECTOR(7 downto 0); -- Display digit 3
        HEX2     : out STD_LOGIC_VECTOR(7 downto 0); -- Display digit 2
        HEX1     : out STD_LOGIC_VECTOR(7 downto 0); -- Display digit 1
        HEX0     : out STD_LOGIC_VECTOR(7 downto 0)  -- Display digit 0
    );
end Reactivity;

architecture Behavioral of Reactivity is

 component BCD is
  Port (
   bin : in std_logic_vector(9 downto 0); -- Binary input (0 to 1023)
   hex3, hex2, hex1, hex0 : out STD_LOGIC_VECTOR(7 downto 0) -- 7-segment displays
  );
 end component;

 component char_7seg
     port (
   c : in std_logic_vector(3 downto 0); -- 4-bit input character code
   display : out std_logic_vector(7 downto 0) -- 7-segment output
  );
 end component;

    signal clk_1kHz : STD_LOGIC := '0';    -- 1 kHz clock
    signal delay_counter : INTEGER := 0; -- Delay counter (seconds)
```

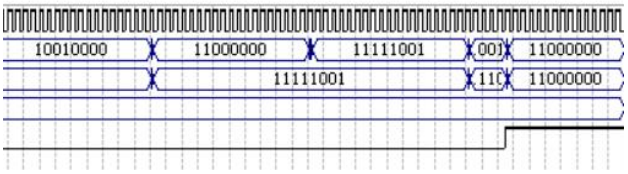
```
    signal reaction_counter : INTEGER := 0; -- Reaction time counter (milliseconds)
    signal led_on : STD_LOGIC := '0';     -- LEDR0 state
    signal counting : STD_LOGIC := '0';  -- Counter state
    signal reaction_bin : std_logic_vector(9 downto 0) := (others=>'0');

    -- Clock Divider: 50 MHz to 1 kHz
    signal clk_divider : INTEGER range 0 to 49_999_999 := 0;

begin

    -- Clock Divider Process
    process (clk50MHz, reset)
    begin
        if reset = '1' then
            clk_divider <= 0;
            clk_1kHz <= '0';
        elsif rising_edge(clk50MHz) then
            if clk_divider = 9 then --if clk_divider = 49_999_999 then
                clk_divider <= 0;
                clk_1kHz <= not clk_1kHz;
            else
                clk_divider <= clk_divider + 1;
            end if;
        end if;
    end process;


    process (clk_1kHz, reset, stop)
    begin
   -- Delay Logic Process
        if reset = '1' then
            delay_counter <= 0;
            led_on <= '0';
            counting <= '0';
        elsif rising_edge(clk_1kHz) then
            if delay_counter < to_integer(unsigned(SW)) then --if delay_counter < to_integer(unsigned(SW))*1000
                delay_counter <= delay_counter + 1;
            else
                led_on <= '1'; -- Turn on LEDR0
                counting <= '1'; -- Start reaction timer
            end if;
        end if;

        -- Reaction Timer Process
        if reset = '1' then
            reaction_counter <= 0;
        elsif rising_edge(clk_1kHz) then
            if counting = '1' and stop = '0' then
                reaction_counter <= reaction_counter + 1;
            elsif stop = '1' then
                counting <= '0'; -- Stop counting when KEY3 is pressed
            end if;
        end if;

    end process;

 reaction_bin <= std_logic_vector(to_unsigned(reaction_counter,10));

    -- Assign LEDR0
    LEDR0 <= led_on;

    -- BCD Conversion and Display
 BCD_0 : BCD
  port map(bin => reaction_bin,hex3 => HEX3, hex2 => HEX2 , hex1=> HEX1, hex0 => HEX0);

end Behavioral;
```

4. Simulation and Synthesis Results

Part I

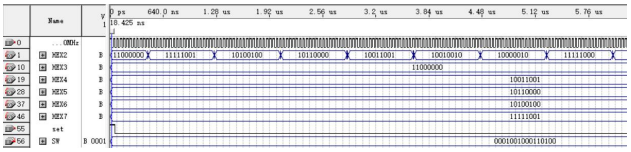


Without reset



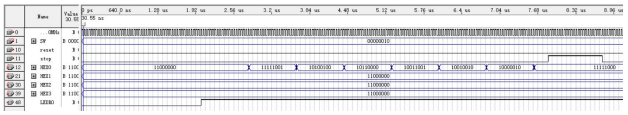
With reset

Part II



Upon setting, the clock is initialized and running when set latch is off. The waveform simulation shows the 12:34:XX scenario.

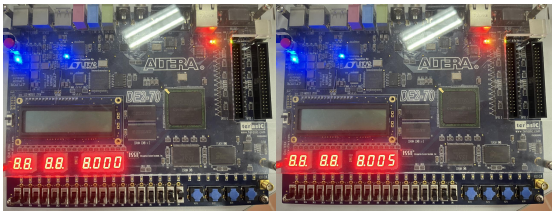
Part III



After certain time set(In this case 3 microsecond), the red light is on ,and upon pushing the stop button, the display freezes.

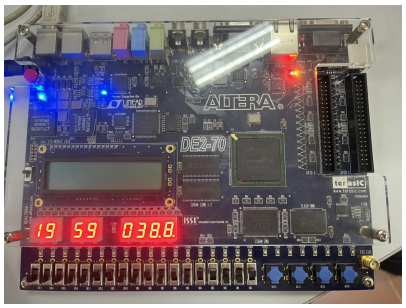
5. Experimental Results

Part I



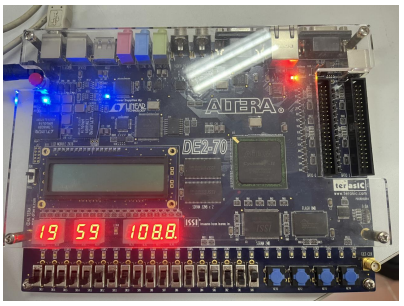
Counts normally.

Part II

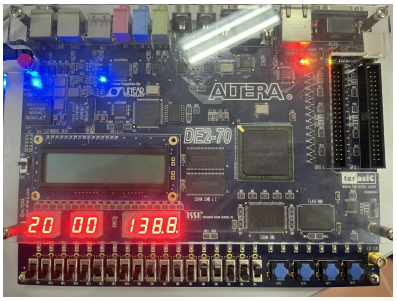


The hours and minutes are initiated once the set latch is on. Its value was dependent on the SW given separately.

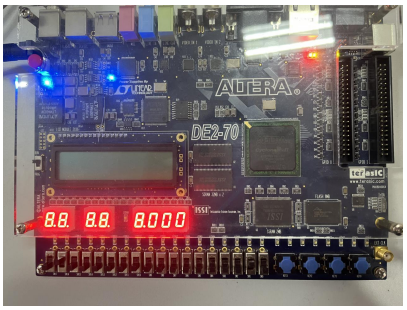
And it flows like it normally should,



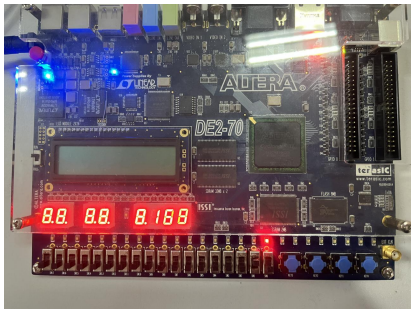
Once reached its maximum, the value simply resets, just as a real clock would.



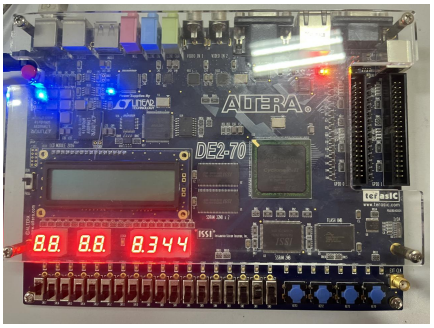
Part III



The clock is meant to be already flowing once start or when the reset latch was pulled down. And after a setting time, in this case two seconds, the red light glows,



...which freezes once the player push down the button, or in our case, when the latch was pulled up, since the key functions in a improper manner.



6. Discussion and Conclusion

The projects highlight the potential of FPGA-based designs for practical applications while opening avenues for further development. Transitioning from a basic implementation to a more VHDL-centric design can significantly enhance performance, scalability, and modularity. By leveraging VHDL's strengths in resource utilization, concurrency, and hierarchical design, these circuits could evolve into robust, real-world systems. As FPGA technology advances, such extensions not only offer academic enrichment but also prepare for industrial-grade applications.