Using Vivado, write VHDL code for the implementation of a sequence detector in VHDLusing push buttons. Four push buttons should be used for entering four input buttons(btnu,btnd, btnl, btnr).Further, the push button btnc in the middleshould be used for initialisation. •Up to 10 inputs could be entered after pressing the initialisation push button. When the sequence (btnu,btnd, btnd, btnl, btnr) is entered, then the LEDs should start flashing. Please note that the right sequence of symbols need not necessarily be entered immediately after the initialisation push button is pressed. For example, the sequence "btnu,btnd, btnl, btnr,btnr,btnu,btnd, btnd, btnl, btnr" should be able to activate the flashing of the LEDs.•If inputs have been entered but the right sequence of symbols has not appeared yet, then the system should lock and the LEDs should show the following predefined pattern:on, off, on, off, on, off, on, off,on, off, on, off.•If the system is locked, the user will need to press the initialisation button in order to be allowed to start entering new symbols

I want to implement to led flash on condition 11111111 to 00000000when.....
- the sequence btnu, btnd, btnd, btnl, btnr , any other 5 buttions
- the sequence any other 5 buttions ,btnu, btnd, btnd, btnl, btnr ,
- the sequence any 1 button input, btnu, btnd, btnd, btnl, btnr , any other 4 button inputs.
-the sequence any 2 button input, btnu, btnd, btnd, btnl, btnr , any 3 button inputs,
- the sequence any 3 button input ,btnu, btnd, btnd, btnl, btnr , any 2 button input,
-the sequence any 4 buttons input, btnu, btnd, btnd, btnl, btnr , any 1 button input,

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

entity Counter is
   Port (
      clk    : in STD_LOGIC;              -- High-frequency clock signal (e.g., 100 MHz)
      btnu   : in STD_LOGIC;              -- Button-up signal
      btnd   : in STD_LOGIC;              -- Button-down signal
      btnl   : in STD_LOGIC;              -- Button-left signal
      btnr   : in STD_LOGIC;              -- Button-right signal
      btnc   : in STD_LOGIC;              -- Button-center signal
      led    : out STD_LOGIC_VECTOR(7 downto 0)  -- 8-bit LED display
   );
end Counter;

architecture Behavioral of Counter is
   signal clk_div       : STD_LOGIC := '0';  -- Slow clock signal for toggling
   signal clk_div_count : integer := 0;      -- Counter for clock division
   constant DIVISOR     : integer := 100000; -- Divide factor for clock (adjust for desired
speed)
   signal pattern       : STD_LOGIC_VECTOR(7 downto 0) := "00000000"; -- LED pattern to
display
begin

   -- Clock Divider Process
   process(clk)
   begin
```

```
        if rising_edge(clk) then
            -- Divide the clock signal by the divisor
            if clk_div_count = DIVISOR - 1 then
                clk_div <= not clk_div;  -- Toggle the clock signal
                clk_div_count <= 0;       -- Reset the counter
            else
                clk_div_count <= clk_div_count + 1;  -- Increment the clock divider counter
            end if;
        end if;
    end process;

    -- LED Control Process
    process(btnu, btnd, btnl, btnr, btnc, clk_div)
    begin
        if btnu = '1' then
            -- Button Up: Pattern 1 (Alternating LEDs)
            pattern <= "10101010";
        elsif btnd = '1' then
            -- Button Down: Pattern 2 (Inverse Alternating LEDs)
            pattern <= "01010101";
        elsif btnl = '1' then
            -- Button Left: Pattern 3 (Outer LEDs)
            pattern <= "10000001";
        elsif btnr = '1' then
            -- Button Right: Pattern 4 (Inner LEDs)
            pattern <= "01111110";
        elsif btnc = '1' then
            -- Button Center: Pattern 5 (All LEDs ON)
            pattern <= "11111111";
        else
            -- No button pressed: Turn off LEDs
            pattern <= "00000000";
        end if;
    end process;

    -- Assign the current pattern to the LEDs
    led <= pattern;

end Behavioral;
```

write me a code that display four less on four various buttons and central button to use as the reset ....include display 111111111 and 0000000 repetitively only when when the combination of button up, down left right is met

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

entity Counterc is
   Port (
      clk    : in STD_LOGIC;               -- High-frequency clock signal
      btnu   : in STD_LOGIC;               -- Button Up
      btnd   : in STD_LOGIC;               -- Button Down
      btnl   : in STD_LOGIC;               -- Button Left
      btnr   : in STD_LOGIC;               -- Button Right
      btnc   : in STD_LOGIC;               -- Central Button (Reset)
      led    : out STD_LOGIC_VECTOR(7 downto 0)  -- 8-bit LED output
   );
end Counterc;

architecture Behavioral of Counterc is
   -- State Machine Definitions
   type State_Type is (IDLE, SHOW_LESS, CHECK_COMBINATION, SUCCESS); --
FSM states
   signal state       : State_Type := IDLE;                     -- Current FSM state

   -- Signals for LED Patterns
   signal display_pattern : STD_LOGIC_VECTOR(7 downto 0) := (others => '0'); -- LED
display pattern
   signal flash_clk       : STD_LOGIC := '0';                          -- Clock for flashing LEDs
   signal flash_count     : INTEGER := 0;                              -- Counter for flashing clock
   constant FLASH_DIVISOR : INTEGER := 50000000;                         -- Flashing speed

   -- Combination Check
   signal sequence : STD_LOGIC_VECTOR(3 downto 0) := "0000";              -- Tracks
button combination
begin

   -- Clock Divider for Flashing LEDs
   process(clk)
   begin
      if rising_edge(clk) then
         if flash_count = FLASH_DIVISOR - 1 then
            flash_clk <= not flash_clk;                         -- Toggle flashing clock
            flash_count <= 0;                              -- Reset flash counter
         else
            flash_count <= flash_count + 1;                      -- Increment flash counter
```

```
    end if;
  end if;
end process;

-- Main FSM Process
process(clk)
begin
  if rising_edge(clk) then
    if btnc = '1' then
      -- Reset the system
      state <= IDLE;
      sequence <= "0000";
      display_pattern <= (others => '0');
    else
      case state is
        when IDLE =>
          -- Display four "less" on respective button presses
          if btnu = '1' then
            display_pattern <= "10000000"; -- Display less on top LED
            sequence(0) <= '1';
          elsif btnd = '1' then
            display_pattern <= "01000000"; -- Display less on bottom LED
            sequence(1) <= '1';
          elsif btnl = '1' then
            display_pattern <= "00100000"; -- Display less on left LED
            sequence(2) <= '1';
          elsif btnr = '1' then
            display_pattern <= "00010000"; -- Display less on right LED
            sequence(3) <= '1';
          end if;

          -- Move to CHECK_COMBINATION when any button is pressed
          if sequence /= "0000" then
            state <= CHECK_COMBINATION;
          end if;

        when CHECK_COMBINATION =>
          -- Check if the combination "btnu, btnd, btnl, btnr" is met
          if sequence = "1111" then
            state <= SUCCESS; -- Go to SUCCESS state if combination is met
          else
            state <= IDLE;   -- Reset to IDLE for invalid combination
          end if;

        when SUCCESS =>
          -- Flash LEDs (11111111 and 00000000 alternately)
          if flash_clk = '1' then
            display_pattern <= "11111111"; -- All LEDs ON
          else
            display_pattern <= "00000000"; -- All LEDs OFF
```

```
                end if;

              when others =>
                state <= IDLE; -- Default to IDLE for unexpected states
            end case;
          end if;
        end if;
    end process;

    -- Assign the display pattern to the LEDs
    led <= display_pattern;

end Behavioral;
```

counter to count udlr button input attempt and central button to reset

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

entity SequenceDetector is
    Port (
        clk    : in STD_LOGIC;                -- High-frequency clock signal
        btnu   : in STD_LOGIC;                -- Button Up
        btnl   : in STD_LOGIC;                -- Button Left
        btnd   : in STD_LOGIC;                -- Button Down
        btnr   : in STD_LOGIC;                -- Button Right
        btnc   : in STD_LOGIC;                -- Central Button (Initialisation)
        led    : out STD_LOGIC_VECTOR(7 downto 0)  -- 8-bit LED output
    );
end SequenceDetector;

architecture Behavioral of SequenceDetector is
    -- FSM States
    type State_Type is (IDLE, COLLECT, SUCCESS, LOCK);
    signal state       : State_Type := IDLE;

    -- Signals for sequence detection
    signal sequence_buffer : STD_LOGIC_VECTOR(39 downto 0) := (others => '0'); -- 10
symbols buffer (4 bits per symbol)
    signal current_index   : INTEGER range 0 to 9 := 0;                -- Index of entered
symbols

    -- Flashing LEDs
    signal flash_clk       : STD_LOGIC := '0';                         -- Clock for flashing LEDs
    signal flash_count     : INTEGER := 0;                             -- Counter for flashing clock
    constant FLASH_DIVISOR : INTEGER := 50000000;                      -- Flash speed
constant
```

```
    -- Predefined Pattern
    signal display_pattern : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');
begin

    -- Clock Divider for Flashing LEDs
    process(clk)
    begin
        if rising_edge(clk) then
            if flash_count = FLASH_DIVISOR - 1 then
                flash_clk <= not flash_clk;                    -- Toggle flashing clock
                flash_count <= 0;                          -- Reset flash counter
            else
                flash_count <= flash_count + 1;                 -- Increment flash counter
            end if;
        end if;
    end process;

    -- Main FSM Logic
    process(clk)
        variable input_symbol : STD_LOGIC_VECTOR(3 downto 0);
    begin
        if rising_edge(clk) then
            if btnc = '1' then
                -- Reset the system
                state <= IDLE;
                sequence_buffer <= (others => '0');
                current_index <= 0;
                display_pattern <= (others => '0');
            else
                case state is
                    when IDLE =>
                        if btnu = '1' then
                            input_symbol := "0001"; -- Up button = '1'
                        elsif btnl = '1' then
                            input_symbol := "0010"; -- Left button = '2'
                        elsif btnd = '1' then
                            input_symbol := "0011"; -- Down button = '3'
                        elsif btnr = '1' then
                            input_symbol := "0100"; -- Right button = '4'
                        else
                            input_symbol := (others => '0');
                        end if;

                        if input_symbol /= (others => '0') then
                            state <= COLLECT;
                        end if;

                    when COLLECT =>
                        -- Collect up to 10 symbols
                        sequence_buffer <= sequence_buffer(35 downto 0) & input_symbol;
                        if current_index < 9 then
                            current_index <= current_index + 1;
                        else
                            -- Check if sequence matches "41123" (Right -> Up -> Down -> Left ->
Up)
                            if sequence_buffer = "01000001000100110010" then
                                state <= SUCCESS;
                            else
                                state <= LOCK;
                            end if;
                        end if;

                    when SUCCESS =>
                        -- Flash LEDs (11111111 and 00000000 alternately)
                        if flash_clk = '1' then
                            display_pattern <= "11111111"; -- All LEDs ON
                        else
                            display_pattern <= "00000000"; -- All LEDs OFF
                        end if;

                    when LOCK =>
                        -- Locked state: Show predefined pattern
                        if flash_clk = '1' then
                            display_pattern <= "10101010";
                        else
                            display_pattern <= "01010101";
                        end if;

                        -- Return to IDLE if central button is pressed
                        if btnc = '1' then
                            state <= IDLE;
                        end if;

                    when others =>
                        state <= IDLE; -- Default state
                end case;
            end if;
        end if;
    end process;

    -- Assign LED display pattern
    led <= display_pattern;

end Behavioral;
```

sequence detector that detect up down left right !!!

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

entity Counterc is
   Port (
      clk    : in STD_LOGIC;              -- High-frequency clock signal
      btnu   : in STD_LOGIC;              -- Button Up
      btnd   : in STD_LOGIC;              -- Button Down
      btnl   : in STD_LOGIC;              -- Button Left
      btnr   : in STD_LOGIC;              -- Button Right
      btnc   : in STD_LOGIC;              -- Central Button (Reset)
      led    : out STD_LOGIC_VECTOR(7 downto 0)  -- 8-bit LED output
   );
end Counterc;

architecture Behavioral of Counterc is
   -- State Machine Definitions
   type State_Type is (IDLE, CHECK_SEQUENCE, SUCCESS); -- FSM states
   signal state       : State_Type := IDLE;          -- Current FSM state

   -- Signals for LED Patterns
   signal display_pattern : STD_LOGIC_VECTOR(7 downto 0) := (others => '0'); -- LED
display pattern
   signal flash_clk       : STD_LOGIC := '0';                         -- Clock for flashing LEDs
   signal flash_count     : INTEGER := 0;                             -- Counter for flashing clock
   constant FLASH_DIVISOR : INTEGER := 1000000;                             -- Flash speed
constant (smaller for testing)

   -- Sequence Detection
   signal sequence_index  : INTEGER range 0 to 3 := 0;                    -- Tracks current
sequence position
   constant target_sequence : STD_LOGIC_VECTOR(3 downto 0) := "1101";       --
Encoded target sequence: btnu, btnd, btnl, btnr

   -- Debouncing Signals
   signal btnu_stable, btnd_stable, btnl_stable, btnr_stable, btnc_stable : STD_LOGIC := '0';
   signal btnu_count, btnd_count, btnl_count, btnr_count, btnc_count : INTEGER := 0;
   constant DEBOUNCE_COUNT : INTEGER := 100000;                          -- Debounce
delay
begin

   -- Button Debouncing Process
   process(clk)
   begin
      if rising_edge(clk) then
         -- Debounce btnu
         if btnu = '1' then
            if btnu_count < DEBOUNCE_COUNT then
               btnu_count <= btnu_count + 1;
            else
               btnu_stable <= '1';
```

```
    end if;
else
    btnu_count <= 0;
    btnu_stable <= '0';
end if;

-- Debounce btnd
if btnd = '1' then
    if btnd_count < DEBOUNCE_COUNT then
        btnd_count <= btnd_count + 1;
    else
        btnd_stable <= '1';
    end if;
else
    btnd_count <= 0;
    btnd_stable <= '0';
end if;

-- Debounce btnl
if btnl = '1' then
    if btnl_count < DEBOUNCE_COUNT then
        btnl_count <= btnl_count + 1;
    else
        btnl_stable <= '1';
    end if;
else
    btnl_count <= 0;
    btnl_stable <= '0';
end if;

-- Debounce btnr
if btnr = '1' then
    if btnr_count < DEBOUNCE_COUNT then
        btnr_count <= btnr_count + 1;
    else
        btnr_stable <= '1';
    end if;
else
    btnr_count <= 0;
    btnr_stable <= '0';
end if;

-- Debounce btnc
if btnc = '1' then
    if btnc_count < DEBOUNCE_COUNT then
        btnc_count <= btnc_count + 1;
    else
        btnc_stable <= '1';
    end if;
else
```

```
            btnc_count <= 0;
            btnc_stable <= '0';
        end if;
    end if;
end process;

-- Clock Divider for Flashing LEDs
process(clk)
begin
    if rising_edge(clk) then
        if flash_count = FLASH_DIVISOR - 1 then
            flash_clk <= not flash_clk;                 -- Toggle flashing clock
            flash_count <= 0;                          -- Reset flash counter
        else
            flash_count <= flash_count + 1;                -- Increment flash counter
        end if;
    end if;
end process;

-- Main FSM Process
process(clk)
begin
    if rising_edge(clk) then
        if btnc_stable = '1' then
            -- Reset the system
            state <= IDLE;
            sequence_index <= 0;
            display_pattern <= (others => '0');
        else
            case state is
                when IDLE =>
                    -- Detect button presses in sequence
                    if btnu_stable = '1' and sequence_index = 0 then
                        sequence_index <= 1;
                    elsif btnd_stable = '1' and sequence_index = 1 then
                        sequence_index <= 2;
                    elsif btnl_stable = '1' and sequence_index = 2 then
                        sequence_index <= 3;
                    elsif btnr_stable = '1' and sequence_index = 3 then
                        state <= SUCCESS; -- Sequence matched
                    end if;

                    -- Show "less" on LEDs based on button presses
                    if btnu_stable = '1' then
                        display_pattern <= "10000000";
                    elsif btnd_stable = '1' then
                        display_pattern <= "01000000";
                    elsif btnl_stable = '1' then
                        display_pattern <= "00100000";
                    elsif btnr_stable = '1' then
```

```
                     display_pattern <= "00010000";
                  end if;

               when SUCCESS =>
                  -- Flash LEDs (11111111 and 00000000 alternately)
                  if flash_clk = '1' then
                     display_pattern <= "11111111"; -- All LEDs ON
                  else
                     display_pattern <= "00000000"; -- All LEDs OFF
                  end if;

               when others =>
                  state <= IDLE; -- Default state
            end case;
         end if;
      end if;
   end process;

   -- Assign LED display pattern
   led <= display_pattern;

end Behavioral;
```

detect up down down left right and flash

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

entity Counterc is
   Port (
      clk    : in STD_LOGIC;               -- High-frequency clock signal
      btnu   : in STD_LOGIC;               -- Button Up
      btnd   : in STD_LOGIC;               -- Button Down
      btnl   : in STD_LOGIC;               -- Button Left
      btnr   : in STD_LOGIC;               -- Button Right
      btnc   : in STD_LOGIC;               -- Central Button (Reset)
      led    : out STD_LOGIC_VECTOR(7 downto 0)  -- 8-bit LED output
   );
end Counterc;

architecture Behavioral of Counterc is
   -- State Machine Definitions
```

```
  type State_Type is (IDLE, WAIT_DOWN1, WAIT_DOWN2, WAIT_LEFT,
WAIT_RIGHT, SUCCESS, LOCK);
  signal state      : State_Type := IDLE;                    -- Current FSM state

  -- Signals for LED Patterns
  signal display_pattern : STD_LOGIC_VECTOR(7 downto 0) := (others => '0'); -- LED
display pattern
  signal flash_clk      : STD_LOGIC := '0';                         -- Clock for flashing LEDs
  signal flash_count    : INTEGER := 0;                             -- Counter for flashing clock
  constant FLASH_DIVISOR : INTEGER := 5000000;                          -- Flashing speed

  -- Debounce Counters
  signal btnu_count, btnd_count, btnl_count, btnr_count : INTEGER := 0;
  constant DEBOUNCE_COUNT : INTEGER := 500000;                         -- Debounce
threshold

  -- Stable Signals
  signal btnu_stable, btnd_stable, btnl_stable, btnr_stable : STD_LOGIC := '0';
begin

  -- Clock Divider for Flashing LEDs
  process(clk)
  begin
    if rising_edge(clk) then
      if flash_count = FLASH_DIVISOR - 1 then
        flash_clk <= not flash_clk;                     -- Toggle flashing clock
        flash_count <= 0;                             -- Reset flash counter
      else
        flash_count <= flash_count + 1;                     -- Increment flash counter
      end if;
    end if;
  end process;

  -- Debouncing Logic for Each Button
  process(clk)
  begin
    if rising_edge(clk) then
      -- Debounce btnu
      if btnu = '1' then
        if btnu_count < DEBOUNCE_COUNT then
          btnu_count <= btnu_count + 1;
        else
          btnu_stable <= '1';
        end if;
      else
        btnu_count <= 0;
        btnu_stable <= '0';
      end if;

      -- Debounce btnd
```

```
        if btnd = '1' then
            if btnd_count < DEBOUNCE_COUNT then
                btnd_count <= btnd_count + 1;
            else
                btnd_stable <= '1';
            end if;
        else
            btnd_count <= 0;
            btnd_stable <= '0';
        end if;

        -- Debounce btnl
        if btnl = '1' then
            if btnl_count < DEBOUNCE_COUNT then
                btnl_count <= btnl_count + 1;
            else
                btnl_stable <= '1';
            end if;
        else
            btnl_count <= 0;
            btnl_stable <= '0';
        end if;

        -- Debounce btnr
        if btnr = '1' then
            if btnr_count < DEBOUNCE_COUNT then
                btnr_count <= btnr_count + 1;
            else
                btnr_stable <= '1';
            end if;
        else
            btnr_count <= 0;
            btnr_stable <= '0';
        end if;
    end if;
end process;

-- Main FSM Process
process(clk)
begin
    if rising_edge(clk) then
        if btnc = '1' then
            -- Reset the system
            state <= IDLE;
            display_pattern <= (others => '0');
        else
            case state is
                when IDLE =>
                    -- Detect first button press (btnu)
                    if btnu_stable = '1' then
```

```
                        state <= WAIT_DOWN1;
                    end if;

                when WAIT_DOWN1 =>
                    -- Detect first btnd
                    if btnd_stable = '1' then
                        state <= WAIT_DOWN2;
                    end if;

                when WAIT_DOWN2 =>
                    -- Detect second btnd
                    if btnd_stable = '1' then
                        state <= WAIT_LEFT;
                    end if;

                when WAIT_LEFT =>
                    -- Detect btnl
                    if btnl_stable = '1' then
                        state <= WAIT_RIGHT;
                    end if;

                when WAIT_RIGHT =>
                    -- Detect btnr and move to SUCCESS
                    if btnr_stable = '1' then
                        state <= SUCCESS;
                    end if;

                when SUCCESS =>
                    -- Flash LEDs (11111111 and 00000000 alternately)
                    if flash_clk = '1' then
                        display_pattern <= "11111111"; -- All LEDs ON
                    else
                        display_pattern <= "00000000"; -- All LEDs OFF
                    end if;

                when LOCK =>
                    -- Locked state: Reset with central button
                    display_pattern <= "10000001"; -- Indicate locked state
                    if btnc = '1' then
                        state <= IDLE;
                    end if;

                when others =>
                    state <= IDLE; -- Default to IDLE for unexpected states
            end case;
        end if;
    end if;
end process;

-- Assign LED display pattern
```

```
   led <= display_pattern;

end Behavioral;
```

Counter that count how many total number of btn up down left right count

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

entity Counterc is
   Port (
      clk    : in STD_LOGIC;               -- Clock signal
      btnu   : in STD_LOGIC;               -- Button Up
      btnd   : in STD_LOGIC;               -- Button Down
      btnl   : in STD_LOGIC;               -- Button Left
      btnr   : in STD_LOGIC;               -- Button Right
      btnc   : in STD_LOGIC;               -- Reset Button
      led    : out STD_LOGIC_VECTOR(7 downto 0)  -- 8-bit LED output
   );
end Counterc;

architecture Behavioral of Counterc is
   -- Button Press Counter
   signal total_count : INTEGER range 0 to 255 := 0; -- Total count of button presses

   -- Debounce Signals
   signal btnu_stable, btnd_stable, btnl_stable, btnr_stable : STD_LOGIC := '0';
   signal btnu_debounce, btnd_debounce, btnl_debounce, btnr_debounce :
INTEGER range 0 to 500000 := 0;
   constant DEBOUNCE_THRESHOLD : INTEGER := 500000; -- Adjust for
debounce timing
```

```
    -- Clock Divider for LED Updates
    signal slow_clk : STD_LOGIC := '0';
    signal clk_count : INTEGER := 0;
    constant CLK_DIVIDER : INTEGER := 500000; -- Adjust for slower clock
begin

    -- Clock Divider Process
    process(clk)
    begin
        if rising_edge(clk) then
            if clk_count = CLK_DIVIDER then
                slow_clk <= not slow_clk; -- Toggle slow clock
                clk_count <= 0;
            else
                clk_count <= clk_count + 1;
            end if;
        end if;
    end process;

    -- Debounce Process
    process(clk)
    begin
        if rising_edge(clk) then
            -- Debounce btnu
            if btnu = '1' then
                if btnu_debounce < DEBOUNCE_THRESHOLD then
                    btnu_debounce <= btnu_debounce + 1;
                else
                    btnu_stable <= '1';
                end if;
            else
                btnu_debounce <= 0;
                btnu_stable <= '0';
            end if;

            -- Debounce btnd
            if btnd = '1' then
                if btnd_debounce < DEBOUNCE_THRESHOLD then
                    btnd_debounce <= btnd_debounce + 1;
                else
                    btnd_stable <= '1';
                end if;
            else
                btnd_debounce <= 0;
                btnd_stable <= '0';
            end if;

            -- Debounce btnl
            if btnl = '1' then
```

```
                if btnl_debounce < DEBOUNCE_THRESHOLD then
                    btnl_debounce <= btnl_debounce + 1;
                else
                    btnl_stable <= '1';
                end if;
            else
                btnl_debounce <= 0;
                btnl_stable <= '0';
            end if;

            -- Debounce btnr
            if btnr = '1' then
                if btnr_debounce < DEBOUNCE_THRESHOLD then
                    btnr_debounce <= btnr_debounce + 1;
                else
                    btnr_stable <= '1';
                end if;
            else
                btnr_debounce <= 0;
                btnr_stable <= '0';
            end if;
        end if;
    end process;

    -- Counting Button Presses
    process(slow_clk)
    begin
        if rising_edge(slow_clk) then
            if btnc = '1' then
                -- Reset the counter
                total_count <= 0;
            else
                -- Increment the counter for each stable button press
                if btnu_stable = '1' or btnd_stable = '1' or btnl_stable = '1' or btnr_stable =
'1' then
                    total_count <= total_count + 1;
                end if;
            end if;
        end if;
    end process;

    -- LED Output
    process(slow_clk)
    begin
        if rising_edge(slow_clk) then
            -- Display total_count on LEDs
            led <= std_logic_vector(to_unsigned(total_count, 8));
        end if;
    end process;
end Behavioral;
```

Count the input and display 1010 pattern on 10th attempt

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

entity SequenceDetector is
    Port (
        clk      : in  STD_LOGIC;                -- Clock signal
        btnc     : in  STD_LOGIC;                -- Initialization button (middle)
        btnu     : in  STD_LOGIC;                -- Button Up
        btnd     : in  STD_LOGIC;                -- Button Down
        btnl     : in  STD_LOGIC;                -- Button Left
        btnr     : in  STD_LOGIC;                -- Button Right
        led      : out STD_LOGIC_VECTOR(6 downto 0) -- LED output for feedback
    );
end SequenceDetector;

architecture Behavioral of SequenceDetector is
    -- Constants
    constant MAX_ATTEMPTS : integer := 10;
    constant FLASH_PATTERN : STD_LOGIC_VECTOR(6 downto 0) := "1010100";

    -- Signal declarations
    type SymbolArray is array(0 to MAX_ATTEMPTS - 1) of std_logic_vector(1
downto 0);
```

```
    signal entered_symbols : SymbolArray := (others => "00"); -- Stores entered symbols
    signal current_index : integer range 0 to MAX_ATTEMPTS := 0; -- Tracks the current input index
    signal flash_led : STD_LOGIC := '0'; -- Indicates flashing state

    -- State management
    type DisplayState is (SHOW_COUNT, FLASH_PATTERN_STATE);
    signal state : DisplayState := SHOW_COUNT;

    -- Debounce signals
    signal btnc_stable, btnu_stable, btnd_stable, btnl_stable, btnr_stable : STD_LOGIC := '0';
    signal btnc_debounce, btnu_debounce, btnd_debounce, btnl_debounce, btnr_debounce : integer range 0 to 1000000 := 0;
    constant DEBOUNCE_THRESHOLD : integer := 500000; -- Debounce threshold for stable signals

    -- Clock divider
    signal slow_clk : STD_LOGIC := '0';
    signal clk_count : integer := 0;
    constant CLK_DIVIDER : integer := 10000000; -- Slower clock for processing
begin
    -- Clock divider process
    process(clk)
    begin
        if rising_edge(clk) then
            if clk_count = CLK_DIVIDER then
                slow_clk <= not slow_clk;
                clk_count <= 0;
            else
                clk_count <= clk_count + 1;
            end if;
        end if;
    end process;

    -- Debounce logic for buttons
    process(clk)
    begin
        if rising_edge(clk) then
            -- Debounce btnc
            if btnc = '1' then
                if btnc_debounce < DEBOUNCE_THRESHOLD then
                    btnc_debounce <= btnc_debounce + 1;
                else
                    btnc_stable <= '1';
                end if;
            else
                btnc_debounce <= 0;
                btnc_stable <= '0';
            end if;

            -- Debounce btnu
            if btnu = '1' then
                if btnu_debounce < DEBOUNCE_THRESHOLD then
                    btnu_debounce <= btnu_debounce + 1;
                else
                    btnu_stable <= '1';
                end if;
            else
                btnu_debounce <= 0;
                btnu_stable <= '0';
            end if;

            -- Debounce btnd
            if btnd = '1' then
                if btnd_debounce < DEBOUNCE_THRESHOLD then
                    btnd_debounce <= btnd_debounce + 1;
                else
                    btnd_stable <= '1';
                end if;
            else
                btnd_debounce <= 0;
                btnd_stable <= '0';
            end if;

            -- Debounce btnl
            if btnl = '1' then
                if btnl_debounce < DEBOUNCE_THRESHOLD then
                    btnl_debounce <= btnl_debounce + 1;
                else
                    btnl_stable <= '1';
                end if;
            else
                btnl_debounce <= 0;
                btnl_stable <= '0';
            end if;

            -- Debounce btnr
            if btnr = '1' then
                if btnr_debounce < DEBOUNCE_THRESHOLD then
                    btnr_debounce <= btnr_debounce + 1;
                else
                    btnr_stable <= '1';
                end if;
            else
                btnr_debounce <= 0;
                btnr_stable <= '0';
            end if;
        end if;
```

```
    end process;

    -- Sequence input logic
    process(slow_clk)
    begin
        if rising_edge(slow_clk) then
            case state is
                when SHOW_COUNT =>
                    if btnc_stable = '1' then
                        -- Reset the sequence and index
                        current_index <= 0;
                        entered_symbols <= (others => "00");
                        flash_led <= '0';
                        state <= SHOW_COUNT;
                    elsif current_index < MAX_ATTEMPTS then
                        -- Record the symbol based on the button pressed
                        if btnu_stable = '1' then
                            entered_symbols(current_index) <= "00"; -- Up
                            current_index <= current_index + 1;
                        elsif btnd_stable = '1' then
                            entered_symbols(current_index) <= "01"; -- Down
                            current_index <= current_index + 1;
                        elsif btnl_stable = '1' then
                            entered_symbols(current_index) <= "10"; -- Left
                            current_index <= current_index + 1;
                        elsif btnr_stable = '1' then
                            entered_symbols(current_index) <= "11"; -- Right
                            current_index <= current_index + 1;
                        end if;

                        -- Transition to FLASH_PATTERN_STATE if max attempts are
reached
                        if current_index = MAX_ATTEMPTS then
                            state <= FLASH_PATTERN_STATE;
                        end if;
                    end if;

                when FLASH_PATTERN_STATE =>
                    flash_led <= '1'; -- Indicate flashing state
            end case;
        end if;
    end process;

    -- LED output process
    process(slow_clk)
    begin
        if rising_edge(slow_clk) then
            if state = FLASH_PATTERN_STATE then
                -- Flash the LED pattern after 10 inputs
                led <= FLASH_PATTERN;
            else
                -- Display the number of inputs entered
                led <= std_logic_vector(to_unsigned(current_index, 7));
            end if;
        end if;
    end process;

end Behavioral;
```

```
############################Best Ever 90%work############################
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity SequenceDetector is
    Port (
        clk      : in  STD_LOGIC;               -- Clock signal
        btnc     : in  STD_LOGIC;               -- Initialization button (middle)
        btnu     : in  STD_LOGIC;               -- Button Up
        btnd     : in  STD_LOGIC;               -- Button Down
        btnl     : in  STD_LOGIC;               -- Button Left
        btnr     : in  STD_LOGIC;               -- Button Right
        led      : out STD_LOGIC_VECTOR(7 downto 0) -- LED output
    );
end SequenceDetector;

architecture Behavioral of SequenceDetector is
    -- Constants
    constant MAX_ATTEMPTS : integer := 10; -- Max number of button presses stored
    constant SEQUENCE_LENGTH : integer := 5;
    constant LOCK_PATTERN : STD_LOGIC_VECTOR(7 downto 0) := "10101010"; --
Predefined locked pattern

    -- Target sequence definition (btnu, btnd, btnd, btnl, btnr)
    constant TARGET_SEQUENCE : std_logic_vector(1 downto 0) := "00";
    constant TARGET_SEQUENCE_ARRAY : std_logic_vector(9 downto 0) :=
"0010011011"; -- Encoded

    -- Signal declarations
    type SymbolArray is array(0 to MAX_ATTEMPTS - 1) of std_logic_vector(1 downto 0);
    signal entered_symbols : SymbolArray := (others => "00");
    signal current_index : integer range 0 to MAX_ATTEMPTS := 0; -- Tracks current input
position
    signal sequence_detected : STD_LOGIC := '0'; -- Indicates sequence is found
    signal locked : STD_LOGIC := '0'; -- Indicates system is locked
    signal flash_clk : STD_LOGIC := '0'; -- Flashing clock for LEDs
    signal flash_count : integer := 0;

    -- Button debounce signals
    signal btnu_stable, btnd_stable, btnl_stable, btnr_stable, btnc_stable : STD_LOGIC := '0';
    signal btnu_debounce, btnd_debounce, btnl_debounce, btnr_debounce, btnc_debounce :
integer range 0 to 1000000 := 0;
    constant DEBOUNCE_THRESHOLD : integer := 500000; -- Debounce threshold

    -- Clock divider
    signal slow_clk : STD_LOGIC := '0';
    signal clk_count : integer := 0;
    constant CLK_DIVIDER : integer := 10000000; -- Slower clock for processing
begin
    -- Clock divider process
    process(clk)
    begin
        if rising_edge(clk) then
            if clk_count = CLK_DIVIDER then
                slow_clk <= not slow_clk;
                clk_count <= 0;
            else
                clk_count <= clk_count + 1;
            end if;

            -- Flash clock for LEDs
            if flash_count = CLK_DIVIDER / 2 then
                flash_clk <= not flash_clk;
                flash_count <= 0;
            else
                flash_count <= flash_count + 1;
            end if;
        end if;
    end process;

    -- Debounce process for stable button signals
    process(clk)
    begin
        if rising_edge(clk) then
            -- Debounce btnu
            if btnu = '1' then
                if btnu_debounce < DEBOUNCE_THRESHOLD then
                    btnu_debounce <= btnu_debounce + 1;
                else
                    btnu_stable <= '1';
                end if;
            else
                btnu_debounce <= 0;
                btnu_stable <= '0';
            end if;
```

```vhdl
            -- Debounce btnd
            if btnd = '1' then
                if btnd_debounce < DEBOUNCE_THRESHOLD then
                    btnd_debounce <= btnd_debounce + 1;
                else
                    btnd_stable <= '1';
                end if;
            else
                btnd_debounce <= 0;
                btnd_stable <= '0';
            end if;

            -- Debounce btnl
            if btnl = '1' then
                if btnl_debounce < DEBOUNCE_THRESHOLD then
                    btnl_debounce <= btnl_debounce + 1;
                else
                    btnl_stable <= '1';
                end if;
            else
                btnl_debounce <= 0;
                btnl_stable <= '0';
            end if;

            -- Debounce btnr
            if btnr = '1' then
                if btnr_debounce < DEBOUNCE_THRESHOLD then
                    btnr_debounce <= btnr_debounce + 1;
                else
                    btnr_stable <= '1';
                end if;
            else
                btnr_debounce <= 0;
                btnr_stable <= '0';
            end if;

            -- Debounce btnc
            if btnc = '1' then
                if btnc_debounce < DEBOUNCE_THRESHOLD then
                    btnc_debounce <= btnc_debounce + 1;
                else
                    btnc_stable <= '1';
                end if;
            else
                btnc_debounce <= 0;
                btnc_stable <= '0';
            end if;
        end if;
    end process;
```

```
-- Sequence input and detection process
process(slow_clk)
begin
   if rising_edge(slow_clk) then
      if btnc_stable = '1' then
         -- Reset system
         current_index <= 0;
         entered_symbols <= (others => "00");
         sequence_detected <= '0';
         locked <= '0';
      elsif current_index < MAX_ATTEMPTS then
         -- Record button press
         if btnu_stable = '1' then
            entered_symbols(current_index) <= "00";
            current_index <= current_index + 1;
         elsif btnd_stable = '1' then
            entered_symbols(current_index) <= "01";
            current_index <= current_index + 1;
         elsif btnl_stable = '1' then
            entered_symbols(current_index) <= "10";
            current_index <= current_index + 1;
         elsif btnr_stable = '1' then
            entered_symbols(current_index) <= "11";
            current_index <= current_index + 1;
         end if;

         -- Check for sequence
         for i in 0 to MAX_ATTEMPTS - SEQUENCE_LENGTH loop
            if entered_symbols(i) = "00" and
              entered_symbols(i + 1) = "01" and
              entered_symbols(i + 2) = "01" and
              entered_symbols(i + 3) = "10" and
              entered_symbols(i + 4) = "11" then
               sequence_detected <= '1';
               locked <= '0';
               exit;
            end if;
         end loop;

         -- Lock system if full and no sequence found
         if current_index = MAX_ATTEMPTS and sequence_detected = '0' then
            locked <= '1';
         end if;
      end if;
   end if;
end process;

-- LED output control
process(flash_clk)
```

```
  begin
    if sequence_detected = '1' then
      -- Flash LEDs
      led <= (others => flash_clk);
    elsif locked = '1' then
      -- Show locked pattern
      led <= LOCK_PATTERN;
    else
      -- Default: show number of inputs
      led <= std_logic_vector(to_unsigned(current_index, 8));
    end if;
  end process;

end Behavioral;
```

GENERATE 10101010 ON EVERY 11 TH BOTTOM PRESS

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

entity Counterc is
  Port (
    clk    : in STD_LOGIC;              -- Clock signal
    btnu   : in STD_LOGIC;              -- Button Up
    btnd   : in STD_LOGIC;              -- Button Down
    btnl   : in STD_LOGIC;              -- Button Left
    btnr   : in STD_LOGIC;              -- Button Right
    btnc   : in STD_LOGIC;              -- Reset Button
    led    : out STD_LOGIC_VECTOR(7 downto 0)  -- 8-bit LED output
  );
end Counterc;

architecture Behavioral of Counterc is
  -- Button Press Counter
  signal total_count : INTEGER range 0 to 255 := 0; -- Total count of button presses

  -- Debounce Signals
  signal btnu_stable, btnd_stable, btnl_stable, btnr_stable : STD_LOGIC := '0';
  signal btnu_debounce, btnd_debounce, btnl_debounce, btnr_debounce : INTEGER range 0
to 500000 := 0;
  constant DEBOUNCE_THRESHOLD : INTEGER := 500000; -- Adjust for debounce
timing

  -- Clock Divider for LED Updates
  signal slow_clk : STD_LOGIC := '0';
  signal clk_count : INTEGER := 0;
  constant CLK_DIVIDER : INTEGER := 500000; -- Adjust for slower clock
begin
```

```
-- Clock Divider Process
process(clk)
begin
    if rising_edge(clk) then
        if clk_count = CLK_DIVIDER then
            slow_clk <= not slow_clk; -- Toggle slow clock
            clk_count <= 0;
        else
            clk_count <= clk_count + 1;
        end if;
    end if;
end process;

-- Debounce Process
process(clk)
begin
    if rising_edge(clk) then
        -- Debounce btnu
        if btnu = '1' then
            if btnu_debounce < DEBOUNCE_THRESHOLD then
                btnu_debounce <= btnu_debounce + 1;
            else
                btnu_stable <= '1';
            end if;
        else
            btnu_debounce <= 0;
            btnu_stable <= '0';
        end if;

        -- Debounce btnd
        if btnd = '1' then
            if btnd_debounce < DEBOUNCE_THRESHOLD then
                btnd_debounce <= btnd_debounce + 1;
            else
                btnd_stable <= '1';
            end if;
        else
            btnd_debounce <= 0;
            btnd_stable <= '0';
        end if;

        -- Debounce btnl
        if btnl = '1' then
            if btnl_debounce < DEBOUNCE_THRESHOLD then
                btnl_debounce <= btnl_debounce + 1;
            else
                btnl_stable <= '1';
            end if;
        else
```

```
         btnl_debounce <= 0;
         btnl_stable <= '0';
      end if;

      -- Debounce btnr
      if btnr = '1' then
         if btnr_debounce < DEBOUNCE_THRESHOLD then
            btnr_debounce <= btnr_debounce + 1;
         else
            btnr_stable <= '1';
         end if;
      else
         btnr_debounce <= 0;
         btnr_stable <= '0';
      end if;
   end if;
end process;

-- Counting Button Presses
process(slow_clk)
begin
   if rising_edge(slow_clk) then
      if btnc = '1' then
         -- Reset the counter
         total_count <= 0;
      else
         -- Increment the counter for each stable button press
         if btnu_stable = '1' or btnd_stable = '1' or btnl_stable = '1' or btnr_stable = '1' then
            total_count <= total_count + 1;
         end if;
      end if;
   end if;
end process;

-- LED Output
process(slow_clk)
begin
   if rising_edge(slow_clk) then
      if btnc = '1' then
         -- Clear LEDs on reset
         led <= (others => '0');
      elsif total_count mod 11 = 0 and total_count /= 0 then
         -- Display 10101010 on every 11th button press
         led <= "10101010";
      else
         -- Display total_count on LEDs
         led <= std_logic_vector(to_unsigned(total_count, 8));
      end if;
   end if;
end process;
end Behavioral;
```

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!lock the system when 11 is inputted!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

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

entity Counterc is
   Port (
      clk    : in STD_LOGIC;                -- Clock signal
      btnu   : in STD_LOGIC;                -- Button Up
      btnd   : in STD_LOGIC;                -- Button Down
      btnl   : in STD_LOGIC;                -- Button Left
      btnr   : in STD_LOGIC;                -- Button Right
      btnc   : in STD_LOGIC;                -- Reset Button
      led    : out STD_LOGIC_VECTOR(7 downto 0)  -- 8-bit LED output
   );
end Counterc;

architecture Behavioral of Counterc is
   -- Button Press Counter
   signal total_count : INTEGER range 0 to 255 := 0; -- Total count of button presses

   -- Debounce Signals
   signal btnu_stable, btnd_stable, btnl_stable, btnr_stable : STD_LOGIC := '0';
   signal btnu_debounce, btnd_debounce, btnl_debounce, btnr_debounce : INTEGER range 0
to 500000 := 0;
   constant DEBOUNCE_THRESHOLD : INTEGER := 500000; -- Adjust for debounce
timing

   -- Clock Divider for LED Updates
   signal slow_clk : STD_LOGIC := '0';
   signal clk_count : INTEGER := 0;
   constant CLK_DIVIDER : INTEGER := 21_000_000; -- Slightly increased for slower
operation
   signal stop_flag : STD_LOGIC := '0'; -- Flag to stop the system
begin

   -- Clock Divider Process
   process(clk)
   begin
      if rising_edge(clk) then
         if clk_count = CLK_DIVIDER then
            slow_clk <= not slow_clk; -- Toggle slow clock
            clk_count <= 0;
         else
            clk_count <= clk_count + 1;
         end if;
```

```
    end if;
end process;

-- Debounce Process
process(clk)
begin
    if rising_edge(clk) then
        -- Debounce btnu
        if btnu = '1' then
            if btnu_debounce < DEBOUNCE_THRESHOLD then
                btnu_debounce <= btnu_debounce + 1;
            else
                btnu_stable <= '1';
            end if;
        else
            btnu_debounce <= 0;
            btnu_stable <= '0';
        end if;

        -- Debounce btnd
        if btnd = '1' then
            if btnd_debounce < DEBOUNCE_THRESHOLD then
                btnd_debounce <= btnd_debounce + 1;
            else
                btnd_stable <= '1';
            end if;
        else
            btnd_debounce <= 0;
            btnd_stable <= '0';
        end if;

        -- Debounce btnl
        if btnl = '1' then
            if btnl_debounce < DEBOUNCE_THRESHOLD then
                btnl_debounce <= btnl_debounce + 1;
            else
                btnl_stable <= '1';
            end if;
        else
            btnl_debounce <= 0;
            btnl_stable <= '0';
        end if;

        -- Debounce btnr
        if btnr = '1' then
            if btnr_debounce < DEBOUNCE_THRESHOLD then
                btnr_debounce <= btnr_debounce + 1;
            else
                btnr_stable <= '1';
            end if;
            else
                btnr_debounce <= 0;
                btnr_stable <= '0';
            end if;
        end if;
    end process;

    -- Counting Button Presses
    process(slow_clk)
    begin
        if rising_edge(slow_clk) then
            if btnc = '1' then
                -- Reset the counter and clear the stop flag
                total_count <= 0;
                stop_flag <= '0';
            elsif stop_flag = '0' then
                -- Increment the counter for each stable button press if not stopped
                if btnu_stable = '1' or btnd_stable = '1' or btnl_stable = '1' or btnr_stable = '1' then
                    total_count <= total_count + 1;
                end if;

                -- Stop when total_count reaches 11
                if total_count = 11 then
                    stop_flag <= '1'; -- Activate the stop flag
                end if;
            end if;
        end if;
    end process;

    -- LED Output
    process(slow_clk)
    begin
        if rising_edge(slow_clk) then
            if btnc = '1' then
                -- Clear LEDs on reset
                led <= (others => '0');
            elsif stop_flag = '1' then
                -- Freeze LEDs to 10101010 when stopped
                led <= "10101010";
            else
                -- Display total_count on LEDs
                led <= std_logic_vector(to_unsigned(total_count, 8));
            end if;
        end if;
    end process;

end Behavioral;
```

!!!!!!!!!!!!!!!!!!!!!!Sequence detector!detect pattern up,down,down,left,right!!!!!!!!!!!!!!!!!!!!

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

entity SequenceDetector is
   Port (
      clk     : in  STD_LOGIC;              -- Clock signal
      btnc    : in  STD_LOGIC;              -- Initialization button (center)
      btnu    : in  STD_LOGIC;              -- Button Up
      btnd    : in  STD_LOGIC;              -- Button Down
      btnl    : in  STD_LOGIC;              -- Button Left
      btnr    : in  STD_LOGIC;              -- Button Right
      led     : out STD_LOGIC_VECTOR(7 downto 0) -- LED output
   );
end SequenceDetector;

architecture Behavioral of SequenceDetector is
   -- Constants
   constant MAX_ATTEMPTS : integer := 15; -- Maximum button presses stored
   constant SEQUENCE_LENGTH : integer := 5;

   -- Signal declarations
   type SymbolArray is array(0 to MAX_ATTEMPTS - 1) of std_logic_vector(1 downto 0);
   signal entered_symbols : SymbolArray := (others => "00");
   signal current_index : integer range 0 to MAX_ATTEMPTS := 0; -- Tracks current input
position
   signal sequence_detected : STD_LOGIC := '0'; -- Indicates sequence is found
   signal led_output : STD_LOGIC_VECTOR(7 downto 0) := (others => '0'); -- LED output

   -- Button debounce signals
   signal btnu_stable, btnd_stable, btnl_stable, btnr_stable, btnc_stable : STD_LOGIC := '0';
   signal btnu_debounce, btnd_debounce, btnl_debounce, btnr_debounce, btnc_debounce :
integer range 0 to 1000000 := 0;
   constant DEBOUNCE_THRESHOLD : integer := 500000; -- Debounce threshold

   -- Clock divider
   signal slow_clk : STD_LOGIC := '0';
   signal clk_count : integer := 0;
   constant CLK_DIVIDER : integer := 20000000; -- Slower clock for processing
begin
   -- Clock divider process
   process(clk)
   begin
      if rising_edge(clk) then
         if clk_count = CLK_DIVIDER then
            slow_clk <= not slow_clk;
            clk_count <= 0;
            else
                clk_count <= clk_count + 1;
            end if;
        end if;
    end process;

    -- Debounce process for stable button signals
    process(clk)
    begin
        if rising_edge(clk) then
            -- Debounce logic for each button
            if btnu = '1' then
                if btnu_debounce < DEBOUNCE_THRESHOLD then
                    btnu_debounce <= btnu_debounce + 1;
                else
                    btnu_stable <= '1';
                end if;
            else
                btnu_debounce <= 0;
                btnu_stable <= '0';
            end if;

            if btnd = '1' then
                if btnd_debounce < DEBOUNCE_THRESHOLD then
                    btnd_debounce <= btnd_debounce + 1;
                else
                    btnd_stable <= '1';
                end if;
            else
                btnd_debounce <= 0;
                btnd_stable <= '0';
            end if;

            if btnl = '1' then
                if btnl_debounce < DEBOUNCE_THRESHOLD then
                    btnl_debounce <= btnl_debounce + 1;
                else
                    btnl_stable <= '1';
                end if;
            else
                btnl_debounce <= 0;
                btnl_stable <= '0';
            end if;

            if btnr = '1' then
                if btnr_debounce < DEBOUNCE_THRESHOLD then
                    btnr_debounce <= btnr_debounce + 1;
                else
                    btnr_stable <= '1';
                end if;
```

```
      else
         btnr_debounce <= 0;
         btnr_stable <= '0';
      end if;

      if btnc = '1' then
         if btnc_debounce < DEBOUNCE_THRESHOLD then
            btnc_debounce <= btnc_debounce + 1;
         else
            btnc_stable <= '1';
         end if;
      else
         btnc_debounce <= 0;
         btnc_stable <= '0';
      end if;
   end if;
end process;

-- Sequence detection and LED output process
process(slow_clk)
   variable temp_sequence : SymbolArray; -- Temporary sequence holder
begin
   if rising_edge(slow_clk) then
      if btnc_stable = '1' then
         -- Reset system
         current_index <= 0;
         entered_symbols <= (others => "00");
         sequence_detected <= '0';
         led_output <= (others => '0');
      elsif current_index < MAX_ATTEMPTS then
         -- Record button press
         if btnu_stable = '1' then
            entered_symbols(current_index) <= "00";
            current_index <= current_index + 1;
         elsif btnd_stable = '1' then
            entered_symbols(current_index) <= "01";
            current_index <= current_index + 1;
         elsif btnl_stable = '1' then
            entered_symbols(current_index) <= "10";
            current_index <= current_index + 1;
         elsif btnr_stable = '1' then
            entered_symbols(current_index) <= "11";
            current_index <= current_index + 1;
         end if;

         -- Check for the specific sequences
         temp_sequence := entered_symbols;
         if (temp_sequence(0) = "00" and temp_sequence(1) = "01" and
            temp_sequence(2) = "01" and temp_sequence(3) = "10" and
            temp_sequence(4) = "11") then
```

```
                sequence_detected <= '1';
            elsif (temp_sequence(1) = "00" and temp_sequence(2) = "01" and
                   temp_sequence(3) = "01" and temp_sequence(4) = "10" and
                   temp_sequence(5) = "11") then
                sequence_detected <= '1';
            end if;
        end if;

        -- Set LED output based on sequence detection
        if sequence_detected = '1' then
            led_output <= (others => '1'); -- Display `11111111`
        else
            led_output <= std_logic_vector(to_unsigned(current_index, 8)); -- Default to
current index
        end if;
    end if;
end process;

-- Assign LED output
led <= led_output;

end Behavioral;
```

!!!Combined System!!

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

entity CombinedSystem is
    Port (
        clk    : in  STD_LOGIC;                -- Clock signal
        btnc   : in  STD_LOGIC;                -- Reset button
        btnu   : in  STD_LOGIC;                -- Button Up
        btnd   : in  STD_LOGIC;                -- Button Down
        btnl   : in  STD_LOGIC;                -- Button Left
        btnr   : in  STD_LOGIC;                -- Button Right
        led    : out STD_LOGIC_VECTOR(7 downto 0) -- LED output
    );
end CombinedSystem;

architecture Behavioral of CombinedSystem is
    -- Constants
    constant MAX_ATTEMPTS : integer := 15; -- Maximum button presses stored
    constant SEQUENCE_LENGTH : integer := 5;
    constant DEBOUNCE_THRESHOLD : integer := 500000; -- Debounce threshold
    constant CLK_DIVIDER : integer := 15000000; -- Reduced Clock divider for faster
processing
```

```
    -- Button Press Counter
    signal total_count : INTEGER range 0 to 255 := 0; -- Total count of button presses
    signal stop_flag : STD_LOGIC := '0'; -- Stops counting at 11

    -- Sequence Detection
    type SymbolArray is array(0 to MAX_ATTEMPTS - 1) of std_logic_vector(1 downto 0);
    signal entered_symbols : SymbolArray := (others => "00");
    signal current_index : integer range 0 to MAX_ATTEMPTS := 0; -- Tracks current input
position
    signal sequence_detected : STD_LOGIC := '0'; -- Indicates sequence is found

    -- Clock Divider
    signal slow_clk : STD_LOGIC := '0';
    signal clk_count : integer := 0;

    -- Debounce Signals
    signal btnu_stable, btnd_stable, btnl_stable, btnr_stable, btnc_stable : STD_LOGIC := '0';
    signal btnu_debounce, btnd_debounce, btnl_debounce, btnr_debounce, btnc_debounce :
integer range 0 to 1000000 := 0;

    -- LED Output
    signal led_output : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');
begin
    -- Clock Divider Process
    process(clk)
    begin
        if rising_edge(clk) then
            if clk_count = CLK_DIVIDER then
                slow_clk <= not slow_clk;
                clk_count <= 0;
            else
                clk_count <= clk_count + 1;
            end if;
        end if;
    end process;

    -- Debounce Process
    process(clk)
    begin
        if rising_edge(clk) then
            -- Debounce btnu
            if btnu = '1' then
                if btnu_debounce < DEBOUNCE_THRESHOLD then
                    btnu_debounce <= btnu_debounce + 1;
                else
                    btnu_stable <= '1';
                end if;
            else
                btnu_debounce <= 0;
                btnu_stable <= '0';
            end if;

            -- Debounce btnd
            if btnd = '1' then
                if btnd_debounce < DEBOUNCE_THRESHOLD then
                    btnd_debounce <= btnd_debounce + 1;
                else
                    btnd_stable <= '1';
                end if;
            else
                btnd_debounce <= 0;
                btnd_stable <= '0';
            end if;

            -- Debounce btnl
            if btnl = '1' then
                if btnl_debounce < DEBOUNCE_THRESHOLD then
                    btnl_debounce <= btnl_debounce + 1;
                else
                    btnl_stable <= '1';
                end if;
            else
                btnl_debounce <= 0;
                btnl_stable <= '0';
            end if;

            -- Debounce btnr
            if btnr = '1' then
                if btnr_debounce < DEBOUNCE_THRESHOLD then
                    btnr_debounce <= btnr_debounce + 1;
                else
                    btnr_stable <= '1';
                end if;
            else
                btnr_debounce <= 0;
                btnr_stable <= '0';
            end if;

            -- Debounce btnc
            if btnc = '1' then
                if btnc_debounce < DEBOUNCE_THRESHOLD then
                    btnc_debounce <= btnc_debounce + 1;
                else
                    btnc_stable <= '1';
                end if;
            else
                btnc_debounce <= 0;
                btnc_stable <= '0';
            end if;
        end if;
       end process;

       -- Counter and Sequence Detection Process
       process(slow_clk)
        variable temp_sequence : SymbolArray; -- Temporary sequence holder
        variable match_count : integer := 0;  -- Tracks matching symbols
       begin
        if rising_edge(slow_clk) then
            if btnc_stable = '1' then
                -- Reset system
                total_count <= 0;
                stop_flag <= '0';
                current_index <= 0;
                entered_symbols <= (others => "00");
                sequence_detected <= '0';
                led_output <= (others => '0');
            elsif sequence_detected = '0' then
                -- Count button presses
                if stop_flag = '0' then
                    if btnu_stable = '1' or btnd_stable = '1' or btnl_stable = '1' or btnr_stable = '1' then
                       total_count <= total_count + 1;
                    end if;

                    -- Stop counting at 11
                    if total_count = 11 then
                       stop_flag <= '1';
                    end if;
                end if;

                -- Record button press for sequence detection
                if btnu_stable = '1' then
                    entered_symbols(current_index) <= "00";
                    current_index <= current_index + 1;
                elsif btnd_stable = '1' then
                    entered_symbols(current_index) <= "01";
                    current_index <= current_index + 1;
                elsif btnl_stable = '1' then
                    entered_symbols(current_index) <= "10";
                    current_index <= current_index + 1;
                elsif btnr_stable = '1' then
                    entered_symbols(current_index) <= "11";
                    current_index <= current_index + 1;
                end if;

                -- Check for target sequence in any position
                for i in 0 to MAX_ATTEMPTS - SEQUENCE_LENGTH loop
                    match_count := 0;
                    if entered_symbols(i) = "00" then
                       match_count := match_count + 1;
                    end if;
```

```
          if entered_symbols(i + 1) = "01" then
            match_count := match_count + 1;
          end if;
          if entered_symbols(i + 2) = "01" then
            match_count := match_count + 1;
          end if;
          if entered_symbols(i + 3) = "10" then
            match_count := match_count + 1;
          end if;
          if entered_symbols(i + 4) = "11" then
            match_count := match_count + 1;
          end if;

          if match_count = 5 then
            sequence_detected <= '1';
            exit;
          end if;
        end loop;
      end if;

      -- Set LED output
      if sequence_detected = '1' then
        led_output <= (others => '1'); -- Display `11111111`
      elsif stop_flag = '1' then
        led_output <= "10101010"; -- Display `10101010` when stopped
      else
        led_output <= std_logic_vector(to_unsigned(total_count, 8)); -- Default to count
      end if;
    end if;
  end process;

  -- Assign LED output
  led <= led_output;

end Behavioral;
```

!!!Flash Implemenation!!

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

entity CombinedSystem is
   Port (
      clk    : in  STD_LOGIC;              -- Clock signal
      btnc   : in  STD_LOGIC;              -- Reset button
      btnu   : in  STD_LOGIC;              -- Button Up
      btnd   : in  STD_LOGIC;              -- Button Down
      btnl   : in  STD_LOGIC;              -- Button Left
      btnr   : in  STD_LOGIC;              -- Button Right
      led    : out STD_LOGIC_VECTOR(7 downto 0) -- LED output
   );
end CombinedSystem;

architecture Behavioral of CombinedSystem is
   -- Constants
   constant MAX_ATTEMPTS : integer := 15; -- Maximum button presses stored
   constant SEQUENCE_LENGTH : integer := 5;
   constant DEBOUNCE_THRESHOLD : integer := 500000; -- Debounce threshold
   constant CLK_DIVIDER : integer := 15000000; -- Reduced Clock divider for faster
processing
   constant FLASH_DIVIDER : integer := 10000000; -- Divider for blinking LEDs

   -- Button Press Counter
   signal total_count : INTEGER range 0 to 255 := 0; -- Total count of button presses
   signal stop_flag : STD_LOGIC := '0'; -- Stops counting at 11

   -- Sequence Detection
   type SymbolArray is array(0 to MAX_ATTEMPTS - 1) of std_logic_vector(1 downto 0);
   signal entered_symbols : SymbolArray := (others => "00");
   signal current_index : integer range 0 to MAX_ATTEMPTS := 0; -- Tracks current input
position
   signal sequence_detected : STD_LOGIC := '0'; -- Indicates sequence is found

   -- Clock Divider
   signal slow_clk : STD_LOGIC := '0';
   signal flash_clk : STD_LOGIC := '0'; -- Blinking clock
   signal clk_count : integer := 0;
   signal flash_count : integer := 0;

   -- Debounce Signals
   signal btnu_stable, btnd_stable, btnl_stable, btnr_stable, btnc_stable : STD_LOGIC := '0';
   signal btnu_debounce, btnd_debounce, btnl_debounce, btnr_debounce, btnc_debounce :
integer range 0 to 1000000 := 0;

   -- LED Output
```

```
    signal led_output : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');
    signal flash_pattern : STD_LOGIC_VECTOR(7 downto 0) := (others => '0'); -- Temporary
pattern for blinking
begin
    -- Clock Divider Process
    process(clk)
    begin
        if rising_edge(clk) then
            -- Generate slow clock
            if clk_count = CLK_DIVIDER then
                slow_clk <= not slow_clk;
                clk_count <= 0;
            else
                clk_count <= clk_count + 1;
            end if;

            -- Generate flash clock for blinking
            if flash_count = FLASH_DIVIDER then
                flash_clk <= not flash_clk;
                flash_count <= 0;
            else
                flash_count <= flash_count + 1;
            end if;
        end if;
    end process;

    -- Debounce Process
    process(clk)
    begin
        if rising_edge(clk) then
            -- Debounce btnu
            if btnu = '1' then
                if btnu_debounce < DEBOUNCE_THRESHOLD then
                    btnu_debounce <= btnu_debounce + 1;
                else
                    btnu_stable <= '1';
                end if;
            else
                btnu_debounce <= 0;
                btnu_stable <= '0';
            end if;

            -- Debounce btnd
            if btnd = '1' then
                if btnd_debounce < DEBOUNCE_THRESHOLD then
                    btnd_debounce <= btnd_debounce + 1;
                else
                    btnd_stable <= '1';
                end if;
            else
```

```
         btnd_debounce <= 0;
         btnd_stable <= '0';
      end if;

      -- Debounce btnl
      if btnl = '1' then
         if btnl_debounce < DEBOUNCE_THRESHOLD then
            btnl_debounce <= btnl_debounce + 1;
         else
            btnl_stable <= '1';
         end if;
      else
         btnl_debounce <= 0;
         btnl_stable <= '0';
      end if;

      -- Debounce btnr
      if btnr = '1' then
         if btnr_debounce < DEBOUNCE_THRESHOLD then
            btnr_debounce <= btnr_debounce + 1;
         else
            btnr_stable <= '1';
         end if;
      else
         btnr_debounce <= 0;
         btnr_stable <= '0';
      end if;

      -- Debounce btnc
      if btnc = '1' then
         if btnc_debounce < DEBOUNCE_THRESHOLD then
            btnc_debounce <= btnc_debounce + 1;
         else
            btnc_stable <= '1';
         end if;
      else
         btnc_debounce <= 0;
         btnc_stable <= '0';
      end if;
   end if;
end process;

-- Counter and Sequence Detection Process
process(slow_clk)
   variable temp_sequence : SymbolArray; -- Temporary sequence holder
   variable match_count : integer := 0;  -- Tracks matching symbols
begin
   if rising_edge(slow_clk) then
      if btnc_stable = '1' then
         -- Reset system
```

```
        total_count <= 0;
        stop_flag <= '0';
        current_index <= 0;
        entered_symbols <= (others => "00");
        sequence_detected <= '0';
        led_output <= (others => '0');
    elsif sequence_detected = '0' then
        -- Count button presses
        if stop_flag = '0' then
            if btnu_stable = '1' or btnd_stable = '1' or btnl_stable = '1' or btnr_stable = '1' then
                total_count <= total_count + 1;
            end if;

            -- Stop counting at 11
            if total_count = 11 then
                stop_flag <= '1';
            end if;
        end if;

        -- Record button press for sequence detection
        if btnu_stable = '1' then
            entered_symbols(current_index) <= "00";
            current_index <= current_index + 1;
        elsif btnd_stable = '1' then
            entered_symbols(current_index) <= "01";
            current_index <= current_index + 1;
        elsif btnl_stable = '1' then
            entered_symbols(current_index) <= "10";
            current_index <= current_index + 1;
        elsif btnr_stable = '1' then
            entered_symbols(current_index) <= "11";
            current_index <= current_index + 1;
        end if;

        -- Check for target sequence in any position
        for i in 0 to MAX_ATTEMPTS - SEQUENCE_LENGTH loop
            match_count := 0;
            if entered_symbols(i) = "00" then
                match_count := match_count + 1;
            end if;
            if entered_symbols(i + 1) = "01" then
                match_count := match_count + 1;
            end if;
            if entered_symbols(i + 2) = "01" then
                match_count := match_count + 1;
            end if;
            if entered_symbols(i + 3) = "10" then
                match_count := match_count + 1;
            end if;
            if entered_symbols(i + 4) = "11" then
```

```
                        match_count := match_count + 1;
                    end if;

                    if match_count = 5 then
                        sequence_detected <= '1';
                        exit;
                    end if;
                end loop;
            end if;

            -- Set LED blinking patterns
            if sequence_detected = '1' then
                flash_pattern <= (others => flash_clk); -- Blink `11111111` and `00000000`
            elsif stop_flag = '1' then
                if flash_clk = '1' then
                    flash_pattern <= "10101010";
                else
                    flash_pattern <= "01010101";
                end if;
            else
                flash_pattern <= std_logic_vector(to_unsigned(total_count, 8)); -- Default to count
            end if;

            led_output <= flash_pattern;
        end if;
    end process;

    -- Assign LED output
    led <= led_output;

end Behavioral;
```

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!Final and clock adjustment!!

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

entity CombinedSystem is
  Port (
    clk   : in  STD_LOGIC;              -- Clock signal
    btnc  : in  STD_LOGIC;              -- Reset button
    btnu  : in  STD_LOGIC;              -- Button Up
    btnd  : in  STD_LOGIC;              -- Button Down
    btnl  : in  STD_LOGIC;              -- Button Left
    btnr  : in  STD_LOGIC;              -- Button Right
    led   : out STD_LOGIC_VECTOR(7 downto 0) -- LED output
  );
end CombinedSystem;

architecture Behavioral of CombinedSystem is
  -- Constants
  constant MAX_ATTEMPTS : integer := 15; -- Maximum button presses stored
  constant SEQUENCE_LENGTH : integer := 5;
  constant DEBOUNCE_THRESHOLD : integer := 500000; -- Debounce threshold
  constant CLK_DIVIDER : integer := 12000000; -- Slightly faster processing
  constant FLASH_DIVIDER : integer := 8000000; -- Slightly faster blinking

  -- Button Press Counter
  signal total_count : INTEGER range 0 to 255 := 0; -- Total count of button presses
  signal stop_flag : STD_LOGIC := '0'; -- Stops counting at 11

  -- Sequence Detection
  type SymbolArray is array(0 to MAX_ATTEMPTS - 1) of std_logic_vector(1 downto 0);
  signal entered_symbols : SymbolArray := (others => "00");
  signal current_index : integer range 0 to MAX_ATTEMPTS := 0; -- Tracks current input
position
  signal sequence_detected : STD_LOGIC := '0'; -- Indicates sequence is found

  -- Clock Divider
  signal slow_clk : STD_LOGIC := '0';
  signal flash_clk : STD_LOGIC := '0'; -- Blinking clock
  signal clk_count : integer := 0;
  signal flash_count : integer := 0;

  -- Debounce Signals
  signal btnu_stable, btnd_stable, btnl_stable, btnr_stable, btnc_stable : STD_LOGIC := '0';
  signal btnu_debounce, btnd_debounce, btnl_debounce, btnr_debounce, btnc_debounce :
integer range 0 to 1000000 := 0;

  -- LED Output
  signal led_output : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');
```

```
    signal flash_pattern : STD_LOGIC_VECTOR(7 downto 0) := (others => '0'); -- Temporary
pattern for blinking
begin
    -- Clock Divider Process
    process(clk)
    begin
        if rising_edge(clk) then
            -- Generate slow clock
            if clk_count = CLK_DIVIDER then
                slow_clk <= not slow_clk;
                clk_count <= 0;
            else
                clk_count <= clk_count + 1;
            end if;

            -- Generate flash clock for blinking
            if flash_count = FLASH_DIVIDER then
                flash_clk <= not flash_clk;
                flash_count <= 0;
            else
                flash_count <= flash_count + 1;
            end if;
        end if;
    end process;

    -- Debounce Process
    process(clk)
    begin
        if rising_edge(clk) then
            -- Debounce btnu
            if btnu = '1' then
                if btnu_debounce < DEBOUNCE_THRESHOLD then
                    btnu_debounce <= btnu_debounce + 1;
                else
                    btnu_stable <= '1';
                end if;
            else
                btnu_debounce <= 0;
                btnu_stable <= '0';
            end if;

            -- Debounce btnd
            if btnd = '1' then
                if btnd_debounce < DEBOUNCE_THRESHOLD then
                    btnd_debounce <= btnd_debounce + 1;
                else
                    btnd_stable <= '1';
                end if;
            else
                btnd_debounce <= 0;
```

```
         btnd_stable <= '0';
      end if;

      -- Debounce btnl
      if btnl = '1' then
         if btnl_debounce < DEBOUNCE_THRESHOLD then
            btnl_debounce <= btnl_debounce + 1;
         else
            btnl_stable <= '1';
         end if;
      else
         btnl_debounce <= 0;
         btnl_stable <= '0';
      end if;

      -- Debounce btnr
      if btnr = '1' then
         if btnr_debounce < DEBOUNCE_THRESHOLD then
            btnr_debounce <= btnr_debounce + 1;
         else
            btnr_stable <= '1';
         end if;
      else
         btnr_debounce <= 0;
         btnr_stable <= '0';
      end if;

      -- Debounce btnc
      if btnc = '1' then
         if btnc_debounce < DEBOUNCE_THRESHOLD then
            btnc_debounce <= btnc_debounce + 1;
         else
            btnc_stable <= '1';
         end if;
      else
         btnc_debounce <= 0;
         btnc_stable <= '0';
      end if;
   end if;
end process;

-- Counter and Sequence Detection Process
process(slow_clk)
   variable temp_sequence : SymbolArray; -- Temporary sequence holder
   variable match_count : integer := 0;  -- Tracks matching symbols
begin
   if rising_edge(slow_clk) then
      if btnc_stable = '1' then
         -- Reset system
         total_count <= 0;
```

```
    stop_flag <= '0';
    current_index <= 0;
    entered_symbols <= (others => "00");
    sequence_detected <= '0';
    led_output <= (others => '0');
elsif sequence_detected = '0' then
    -- Count button presses
    if stop_flag = '0' then
        if btnu_stable = '1' or btnd_stable = '1' or btnl_stable = '1' or btnr_stable = '1' then
            total_count <= total_count + 1;
        end if;

        -- Stop counting at 11
        if total_count = 11 then
            stop_flag <= '1';
        end if;
    end if;

    -- Record button press for sequence detection
    if btnu_stable = '1' then
        entered_symbols(current_index) <= "00";
        current_index <= current_index + 1;
    elsif btnd_stable = '1' then
        entered_symbols(current_index) <= "01";
        current_index <= current_index + 1;
    elsif btnl_stable = '1' then
        entered_symbols(current_index) <= "10";
        current_index <= current_index + 1;
    elsif btnr_stable = '1' then
        entered_symbols(current_index) <= "11";
        current_index <= current_index + 1;
    end if;

    -- Check for target sequence in any position
    for i in 0 to MAX_ATTEMPTS - SEQUENCE_LENGTH loop
        match_count := 0;
        if entered_symbols(i) = "00" then
            match_count := match_count + 1;
        end if;
        if entered_symbols(i + 1) = "01" then
            match_count := match_count + 1;
        end if;
        if entered_symbols(i + 2) = "01" then
            match_count := match_count + 1;
        end if;
        if entered_symbols(i + 3) = "10" then
            match_count := match_count + 1;
        end if;
        if entered_symbols(i + 4) = "11" then
            match_count := match_count + 1;
```

```
               end if;

               if match_count = 5 then
                  sequence_detected <= '1';
                  exit;
               end if;
            end loop;
         end if;

         -- Set LED blinking patterns
         if sequence_detected = '1' then
            flash_pattern <= (others => flash_clk); -- Blink `11111111` and `00000000`
         elsif stop_flag = '1' then
            if flash_clk = '1' then
               flash_pattern <= "10101010";
            else
               flash_pattern <= "01010101";
            end if;
         else
            flash_pattern <= std_logic_vector(to_unsigned(total_count, 8)); -- Default to count
         end if;

         led_output <= flash_pattern;
      end if;
   end process;

   -- Assign LED output
   led <= led_output;

end Behavioral;
```