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,

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!Five buttiosn with five displays!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

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;  
  
            -- 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 Implementation!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

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;

!!!!!!!!!!!!!!!!!!Stable debounce implementation !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!  
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 CLK\_DIVIDER       : integer := 12000000; -- Slow clock frequency divider  
    constant FLASH\_DIVIDER     : integer := 8000000; -- Flash clock frequency divider  
  
    -- 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;  
  
    -- Debouncer Outputs  
    signal btnu\_debounced, btnd\_debounced, btnl\_debounced, btnr\_debounced, btnc\_debounced : std\_logic;  
  
    -- 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 - 1 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 - 1 then  
                flash\_clk <= not flash\_clk;  
                flash\_count <= 0;  
            else  
                flash\_count <= flash\_count + 1;  
            end if;  
        end if;  
    end process;  
  
    -- Instantiate Debouncer for Each Button  
    debounce\_btnu: entity work.Debouncer  
        Port map (  
            clk    => slow\_clk, -- Use slow clock  
            input  => btnu,  
            output => btnu\_debounced  
        );  
  
    debounce\_btnd: entity work.Debouncer  
        Port map (  
            clk    => slow\_clk, -- Use slow clock  
            input  => btnd,  
            output => btnd\_debounced  
        );  
  
    debounce\_btnl: entity work.Debouncer  
        Port map (  
            clk    => slow\_clk, -- Use slow clock  
            input  => btnl,  
            output => btnl\_debounced  
        );  
  
    debounce\_btnr: entity work.Debouncer  
        Port map (  
            clk    => slow\_clk, -- Use slow clock  
            input  => btnr,  
            output => btnr\_debounced  
        );  
  
    debounce\_btnc: entity work.Debouncer  
        Port map (  
            clk    => slow\_clk, -- Use slow clock  
            input  => btnc,  
            output => btnc\_debounced  
        );  
  
    -- Counter and Sequence Detection Process  
    process(slow\_clk)  
        variable match\_count : integer := 0;  -- Tracks matching symbols  
    begin  
        if rising\_edge(slow\_clk) then  
            if btnc\_debounced = '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\_debounced = '1' or btnd\_debounced = '1' or btnl\_debounced = '1' or btnr\_debounced = '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\_debounced = '1' then  
                    entered\_symbols(current\_index) <= "00";  
                    current\_index <= current\_index + 1;  
                elsif btnd\_debounced = '1' then  
                    entered\_symbols(current\_index) <= "01";  
                    current\_index <= current\_index + 1;  
                elsif btnl\_debounced = '1' then  
                    entered\_symbols(current\_index) <= "10";  
                    current\_index <= current\_index + 1;  
                elsif btnr\_debounced = '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;  
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  
library IEEE;  
use IEEE.STD\_LOGIC\_1164.ALL;  
  
entity Debouncer is  
    Port (  
        clk    : in std\_logic;  
        input  : in std\_logic;  
        output : out std\_logic  
    );  
end Debouncer;  
  
architecture Behavioral of Debouncer is  
    signal delay1, delay2, delay3 : std\_logic := '0';  
begin  
    process(clk)  
    begin  
        if rising\_edge(clk) then  
            delay1 <= input;  
            delay2 <= delay1;  
            delay3 <= delay2;  
        end if;  
    end process;  
  
    output <= delay2 and not delay3; -- Detect stable rising edge  
end Behavioral;  
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

XXXXXXXXXXXXXXXXXXXClock adjustmentXXXXXXXXXXXXXXXXXXXXXXX

OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO

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 CLK\_DIVIDER       : integer := 3000000; -- Slow clock frequency divider  
    constant FLASH\_DIVIDER     : integer := 2000000; -- Flash clock frequency divider  
  
    -- 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;  
  
    -- Debouncer Outputs  
    signal btnu\_debounced, btnd\_debounced, btnl\_debounced, btnr\_debounced, btnc\_debounced : std\_logic;  
  
    -- 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 - 1 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 - 1 then  
                flash\_clk <= not flash\_clk;  
                flash\_count <= 0;  
            else  
                flash\_count <= flash\_count + 1;  
            end if;  
        end if;  
    end process;

    -- Instantiate Debouncer for Each Button  
    debounce\_btnu: entity work.Debouncer  
        Port map (  
            clk    => slow\_clk, -- Use slow clock  
            input  => btnu,  
            output => btnu\_debounced  
        );  
  
    debounce\_btnd: entity work.Debouncer  
        Port map (  
            clk    => slow\_clk, -- Use slow clock  
            input  => btnd,  
            output => btnd\_debounced  
        );  
  
    debounce\_btnl: entity work.Debouncer  
        Port map (  
            clk    => slow\_clk, -- Use slow clock  
            input  => btnl,  
            output => btnl\_debounced  
        );  
  
    debounce\_btnr: entity work.Debouncer  
        Port map (  
            clk    => slow\_clk, -- Use slow clock  
            input  => btnr,  
            output => btnr\_debounced  
        );  
  
    debounce\_btnc: entity work.Debouncer  
        Port map (  
            clk    => slow\_clk, -- Use slow clock  
            input  => btnc,  
            output => btnc\_debounced  
        );

    -- Counter and Sequence Detection Process  
    process(slow\_clk)  
        variable match\_count : integer := 0;  -- Tracks matching symbols  
    begin  
        if rising\_edge(slow\_clk) then  
            if btnc\_debounced = '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\_debounced = '1' or btnd\_debounced = '1' or btnl\_debounced = '1' or btnr\_debounced = '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\_debounced = '1' then  
                    entered\_symbols(current\_index) <= "00";  
                    current\_index <= current\_index + 1;  
                elsif btnd\_debounced = '1' then  
                    entered\_symbols(current\_index) <= "01";  
                    current\_index <= current\_index + 1;  
                elsif btnl\_debounced = '1' then  
                    entered\_symbols(current\_index) <= "10";  
                    current\_index <= current\_index + 1;  
                elsif btnr\_debounced = '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;

OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO

library IEEE;  
use IEEE.STD\_LOGIC\_1164.ALL;  
  
entity Debouncer is  
    Port (  
        clk    : in std\_logic;  
        input  : in std\_logic;  
        output : out std\_logic  
    );  
end Debouncer;  
  
architecture Behavioral of Debouncer is  
    signal delay1, delay2, delay3 : std\_logic := '0';  
begin  
    process(clk)  
    begin  
        if rising\_edge(clk) then  
            delay1 <= input;  
            delay2 <= delay1;  
            delay3 <= delay2;  
        end if;  
    end process;  
  
    output <= delay2 and not delay3; -- Detect stable rising edge  
end Behavioral;

OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO