```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

-------------------------------------------------------------------------------------
--Title: MemoryGame
--Name: Nathaniel Roberts, Mitch Walker
--Date: 4/23/25
--Prof: Scott Tippens
--Desc: Memory Game design file
--      This module contains the main operating instructions of the Memory game.
--
--      To start the game a player must press the center push button.
--      Then 5 separate leds will light on the 16 wide led display once each second.
--
--      They player must recall the order of the leds by flipping the corresponding
--      slide switch. Once all numbers were guessed correctly, a short victory patter
--      lights, the score increments and the next round starts automatically.
--
--      Each successive round is faster than the last. The game ends at any point the
--      player guesses the wrong number during their turn.
-------------------------------------------------------------------------------------

entity MemoryGame is
    generic(
        MAX_COUNT_SCALER : integer;
        SCALE_AMOUNT : integer;
        MAX_TOGGLE_COUNT : integer;
        BLINK_COUNT : integer
    );
    port(
        switches : in std_logic_vector(15 downto 0);
        start : in std_logic;

        clock : in std_logic;
        reset : in std_logic;

        leds : out std_logic_vector(15 downto 0);

        outputScore : out std_logic_vector(7 downto 0);

        blanks : out std_logic_vector(3 downto 0)
    );
end entity MemoryGame;


architecture MemoryGame_ARCH of MemoryGame is

    -------------------------------------------------------------------------------------
    --constants and status signals
    -------------------------------------------------------------------------------------

    --this is subtracted from the toggling counter, making the leds blink faster
    signal countScaler : integer range 0 to MAX_COUNT_SCALER;
```

```
----------------------------------------------------------------------------------
--component definitions
----------------------------------------------------------------------------------
component RandomNumbers is
    port(
        generateEN : in  std_logic;
        clock      : in  std_logic;
        reset      : in  std_logic;

        readyEN    : out std_logic;

        number0    : out std_logic_vector(3 downto 0);
        number1    : out std_logic_vector(3 downto 0);
        number2    : out std_logic_vector(3 downto 0);
        number3    : out std_logic_vector(3 downto 0);
        number4    : out std_logic_vector(3 downto 0)
    );
end component RandomNumbers;

component NumberChecker is
    port(
        switches    : in  std_logic_vector(15 downto 0);

        number0     : in  std_logic_vector(3 downto 0);
        number1     : in  std_logic_vector(3 downto 0);
        number2     : in  std_logic_vector(3 downto 0);
        number3     : in  std_logic_vector(3 downto 0);
        number4     : in  std_logic_vector(3 downto 0);

        readMode    : in  std_logic;

        clock       : in  std_logic;
        reset       : in  std_logic;

        gameOverEN  : out std_logic;
        gameWinEN   : out std_logic
    );
end component NumberChecker;

component WinPattern
    generic(BLINK_COUNT : natural);
    port(
        winPatternMode   : in  std_logic;
        reset            : in  std_logic;
        clock            : in  std_logic;
        leds             : out std_logic_vector(15 downto 0)
    );
end component WinPattern;

component LosePattern
    generic(BLINK_COUNT : natural);
    port(
        losePatternEN     : in  std_logic;
        reset             : in  std_logic;
        clock             : in  std_logic;
        leds              : out std_logic_vector(15 downto 0);
        losePatternIsBusy : out std_logic
    );
end component LosePattern;

component LedSegments
    port(
        binary4Bit : in  std_logic_vector(3 downto 0);
        outputMode : in  std_logic;
        leds       : out std_logic_vector(15 downto 0)
    );
end component LedSegments;
```

```
    component BCD
        port(
            binary4Bit  : in  std_logic_vector(3 downto 0);
            decimalOnes : out std_logic_vector(3 downto 0);
            decimalTens : out std_logic_vector(3 downto 0)
        );
    end component BCD;

    ----------------------------------------------------------------------------------
    -- connecting signals
    ----------------------------------------------------------------------------------
    --Signals for RNG_GENERATOR
    signal readyEN : std_logic;
    signal number0 : std_logic_vector(3 downto 0);
    signal number1 : std_logic_vector(3 downto 0);
    signal number2 : std_logic_vector(3 downto 0);
    signal number3 : std_logic_vector(3 downto 0);
    signal number4 : std_logic_vector(3 downto 0);
    signal outputNumber : std_logic_vector(3 downto 0);

    --Control signal for the RNG_GENERATOR
    signal startControl : std_logic;

    --Signals for CHECK_NUMBERS
    signal readMode : std_logic;
    signal nextRoundEN : std_logic;
    signal gameOverEN : std_logic;
    signal gameWinEN : std_logic;
    signal ledMode : std_logic;

    --Score represented as a integer and later converted to vector
    signal score : integer range 0 to 15;
    signal scoreVector : std_logic_vector(3 downto 0);

    --Level signal to make sure user cannot press the start button
    --while the lose pattern is playing.
    signal losePatternIsBusy : std_logic;

    --Timer controlled state signal
    signal SMTimer : integer;

    --Level controls for the win and lose patters
    signal loseMode : std_logic;
    signal winMode : std_logic;

begin

    ----------------------------------------------------------------------------------
    -- Random number generator component controlled by a combinational control signal
    ----------------------------------------------------------------------------------
    startControl <= (start and readMode and (not losePatternIsBusy)) or nextRoundEN;
    RNG_GENERATOR : component RandomNumbers port map(
        generateEN => startControl,

        clock      => clock,
        reset      => reset,

        -------------outputs
        readyEN    => readyEN,

        number0    => number0,
        number1    => number1,
        number2    => number2,
        number3    => number3,
        number4    => number4
    );
```

```
-----------------------------------------------------------------------------------
-- Number checker component that ouputs a pulse when all numbers intered are correct
-----------------------------------------------------------------------------------
CHECK_NUMBERS : component NumberChecker port map(
    switches    => switches,

    number0     => number0,
    number1     => number1,
    number2     => number2,
    number3     => number3,
    number4     => number4,

    readMode    => readMode,

    clock       => clock,
    reset       => reset,

    --------------outputs
    gameOverEN  => gameOverEN,
    gameWinEN   => gameWinEN
);

-----------------------------------------------------------------------------------
-- Win pattern generator that is mode controlled, outpus a pulse to start next round
-----------------------------------------------------------------------------------
WIN_PATTERN_DRIVER : component WinPattern
    generic map(
        BLINK_COUNT => BLINK_COUNT
    )
    port map(
        winPatternMode   => winMode,
        reset            => reset,
        clock            => clock,

        --------------outputs
        leds             => leds
);

-----------------------------------------------------------------------------------
-- Lose pattern generator that is pulse controlled,
-- ouputs a level control high if it is activley controlling the leds.
-----------------------------------------------------------------------------------
LOSE_PATTERN_DRRIVER : LosePattern
    generic map(
        BLINK_COUNT => BLINK_COUNT
    )
    port map(
        losePatternEN     => gameOverEN,
        reset             => reset,
        clock             => clock,

        --------------outputs
        leds              => leds,
        losePatternIsBusy => losePatternIsBusy
);
```

```
-----------------------------------------------------------------------------------
-- Takes a 4 bit binary word and converts to a positional 16 bit wide vector for leds.
-- This component drives all leds to Z if the outputMode port is low.
-----------------------------------------------------------------------------------
GAME_LED_DRIVER : LedSegments port map(
    binary4Bit => outputNumber,
    outputMode => ledMode, --if outputMode is low, then this output is (others => 'Z')

    -------------outputs
    leds       => leds
);

-----------------------------------------------------------------------------------
-- Code conversion component to change a binary number to BCD
-----------------------------------------------------------------------------------
scoreVector <= std_logic_vector(to_unsigned(score, 4));
SCORE_NUMBER_BCD : BCD port map(
    binary4Bit  => scoreVector,

    -------------outputs
    decimalOnes => outputScore(3 downto 0),
    decimalTens => outputScore(7 downto 4)
);


-----------------------------------------------------------------------------------
-- Process that increments score and speed with each win.
-- Whenever the gameOverEN signal pulses, the loseMode latch is set, which activates
-- the LOSE_MODE_TIMER process.
-----------------------------------------------------------------------------------
GAME_DRIVER : process (clock, reset)
begin
    if reset = '1' then
        score <= 0;
        countScaler <= 0;
        loseMode <= '0';
    elsif rising_edge(clock) then
        if gameWinEN  = '1' then
            score <= score + 1;
            if countScaler <= MAX_COUNT_SCALER - 1 then
                countScaler <= countScaler + SCALE_AMOUNT;
            end if;
        end if;
        if gameOverEN = '1' then
            countScaler <= 0;
            loseMode <= '1';
        end if;
    end if;
end process;

-----------------------------------------------------------------------------------
-- Timer that handles blinking the final score until reset happens.
-- This process can only run the LosePattern module if loseMode was set by GAME_DRIVER.
-----------------------------------------------------------------------------------
LOSE_MODE_TIMER : process(clock, reset)
    variable counter : integer range 0 to MAX_TOGGLE_COUNT - 1;
    variable toggle : std_logic;
begin
    if reset = '1' then
        counter := 0;
        toggle := '0';
    elsif rising_edge(clock) then
        if loseMode = '1' then
            counter := counter + 1;
            if (counter >= (MAX_TOGGLE_COUNT - 1)) then
                toggle := not toggle;
                counter := 0;
```

```vhdl
            end if;
            if toggle = '1' then
                blanks <= "0011";
            elsif toggle = '0' then
                blanks <= "1111";
            end if;

        elsif loseMode = '0' then
            counter := 0;
            blanks <= "0011";
        end if;
    end if;
end process;

-------------------------------------------------------------------------------------
--Timer that handles automatic start and level controlled win pattern.
-------------------------------------------------------------------------------------
WIN_MODE_TIMER : process(clock, reset)
    variable counter : integer range 0 to (2 * MAX_TOGGLE_COUNT) - 1;
    variable latch : std_logic;
    variable godMode : std_logic;
begin
    if reset = '1' then
        counter := 0;
        latch := '0';
        nextRoundEN <= '0';
        godMode := '0';
    elsif rising_edge(clock) then

        --Once the number checker component sends the gameWinEN pulse
        --then the latch variable goes high. Same for the godMode latch.
        nextRoundEN <= '0';
        if gameWinEN = '1' then
            latch := '1';
        elsif score = 15 then
            godMode := '1';
        end if;

        --Depending if one latch or another was set, then then
        --mode controlled WinPattern component takes over the leds.
        if godMode = '1' then
            winMode <= '1';
        elsif latch = '1' then
            winMode <= '1';
            counter := counter + 1;
            if (counter >= ((2 * MAX_TOGGLE_COUNT) - 1)) then
                latch := '0';
                counter := 0;
                nextRoundEN <= '1';
            end if;
        elsif latch = '0' then
            counter := 0;
            winMode <= '0';
        else
            counter := 0;
            winMode <= '0';
        end if;
    end if;
end process;
```

```
--------------------------------------------------------------------------------
--Timer that directs the display the values of RNG_GENERATOR.
--------------------------------------------------------------------------------
DISPLAY_SM_TIMER : process(clock,reset)
    variable counter : integer range 0 to MAX_TOGGLE_COUNT - 1;
    variable countMode : std_logic;
begin
    if reset = '1' then
        SMTimer <= 0;
        counter := 0;
        countMode := '0';
    elsif rising_edge(clock) then

        --The readyEN pulse from the RNG_GENERATOR component will latch this
        --internal variable high, enabling the next if statement.
        if readyEN = '1' then
            countMode := '1'; --latch in and start counting
        end if;

        --The counter variable reaches it's terminal value once each second.
        --Each time the terminal value is reached, the SMTimer signal increments.
        --After reaching that terminal value 9 times, it goes back to a steady state.
        if countMode = '1' then
            counter := counter + 1;
            if (counter >= (MAX_TOGGLE_COUNT - countScaler - 1)) then
                SMTimer <= SMTimer + 1;
                counter := 0;
            end if;
        elsif countMode = '0' then
            SMTimer <= 0;
            counter := 0;
        end if;

        if SMTimer = 10 then
            SMTimer <= 0;
            countMode := '0';
            counter := 0;
        end if;
    end if;
end process;

--------------------------------------------------------------------------------
-- State machine responsible for multiplexing the random number to leds
-- in addition to writing control signals.
-- This state machine was designed this way to cut down dignificantly on the
-- unintentional latches of previous display state machine designs.
--------------------------------------------------------------------------------
DISPLAY_SM : process(SMTimer, number0, number1, number2, number3, number4, loseMode) is
begin
    case (SMTimer) is
        -----------------------------------------------------------------BLANK
        when 0 =>
            readMode <= '1';                --allow the start button to be pressed
            ledMode <= '0';                 --deactivate LedSegments component

        -----------------------------------------------------------------NUM1
        when 1 =>
            readMode <= '0';
            ledMode <= '1';                 --activate LedSegments component
            outputNumber <= number0;

        -----------------------------------------------------------------BLANK
        when 2 =>
            readMode <= '0';
            ledMode <= '0';                 --deactivate LedSegments component

        -----------------------------------------------------------------NUM2
```

```
            when 3 =>
                readMode <= '0';
                ledMode <= '1';                 --activate LedSegments component
                outputNumber <= number1;

            ------------------------------------------------------------------BLANK
            when 4 =>
                readMode <= '0';
                ledMode <= '0';                 --deactivate LedSegments component

            ------------------------------------------------------------------NUM3
            when 5 =>
                readMode <= '0';
                ledMode <= '1';                 --activate LedSegments component
                outputNumber <= number2;

            ------------------------------------------------------------------BLANK
            when 6 =>
                readMode <= '0';
                ledMode <= '0';                 --deactivate LedSegments component

            ------------------------------------------------------------------NUM4
            when 7 =>
                readMode <= '0';
                ledMode <= '1';                 --activate LedSegments component
                outputNumber <= number3;

            ------------------------------------------------------------------BLANK
            when 8 =>
                readMode <= '0';
                ledMode <= '0';                 --deactivate LedSegments component

            ------------------------------------------------------------------NUM5
            when 9 =>
                readMode <= '0';
                ledMode <= '1';                 --activate LedSegments component
                outputNumber <= number4;

            ----------------------------------------------------------DEFAULT case
            when others =>
                readMode <= '0';
                ledMode <= '0';                 --deactivate LedSegments component

        end case;

    end process;

end architecture MemoryGame_ARCH;
```

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

----------------------------------------------------------------------------------
--Title: RandomNumbers.vhd
--Name: Nathaniel Roberts, Mitch Walker
--Date: 3/26/25
--Prof: Scott Tippens

-- Description: Random Number Generator
--     This component utilizes 5 counters incrementing at different rates to
--     generate 5 psuedo-random numbers
--
--     Each RandCounter has a sub counter. Once the sub counter reaches its
--     max value (set externally through maxSubCount) the randNum counter is
--     incremented
--
--     Once generateEN recieves a pulse, SET_RAND_NUM sets the output signals
--     to their corresponding randNum counter value. SET_RAND_NUM them sends
--     a pulsed signal through readyEN to signify that new random numbers
--     have been set
--     Dependencies: RandCounter.vhd
----------------------------------------------------------------------------------

entity RandomNumbers is
    port (
        generateEN  : in std_logic;
        clock       : in std_logic;
        reset       : in std_logic;

        readyEN     : out std_logic;
        number0     : out std_logic_vector(3 downto 0);
        number1     : out std_logic_vector(3 downto 0);
        number2     : out std_logic_vector(3 downto 0);
        number3     : out std_logic_vector(3 downto 0);
        number4     : out std_logic_vector(3 downto 0)
    );
end RandomNumbers;

architecture RandomNumbers_ARCH of RandomNumbers is
    ------------------------------------------------------------------------------
    -- Constants
    ------------------------------------------------------------------------------
    constant ACTIVE                   : std_logic := '1'; -- Active value
    constant COUNTER_MAX_VALUE        : integer   := 15;  -- Max value for 'random' counters
    constant maxSubCount_Length       : integer   := 3;   -- Length of the subCount signal within RandCounter
component

    constant MAX_SUB_COUNT_0 : integer   := 3;   -- Max value for sub counter 1
    constant MAX_SUB_COUNT_1 : integer   := 1;   -- Max value for sub counter 2
    constant MAX_SUB_COUNT_2 : integer   := 2;   -- Max value for sub counter 3
    constant MAX_SUB_COUNT_3 : integer   := 5;   -- Max value for sub counter 4
    constant MAX_SUB_COUNT_4 : integer   := 4;   -- Max value for sub counter 5

    -------------------------------------------------------------------------------
    -- Components
    -------------------------------------------------------------------------------
    component RandCounter is
        port (
            maxSubCount : in std_logic_vector(2 downto 0);
            clock       : in std_logic;
            reset       : in std_logic;

            randNum     : out std_logic_vector(3 downto 0)
        );
    end component;
    ---------------------------------------------------------------------------------
    -- Signals
    ---------------------------------------------------------------------------------
    signal randNum0 : std_logic_vector(3 downto 0);
    signal randNum1 : std_logic_vector(3 downto 0);
    signal randNum2 : std_logic_vector(3 downto 0);
    signal randNum3 : std_logic_vector(3 downto 0);
    signal randNum4 : std_logic_vector(3 downto 0);


begin

    --------------------------------------------------------------------------------
    -- Component Instatiations
    --------------------------------------------------------------------------------
    -- Used to generate pseudo-random value for randNum0
    RAND_COUNTER_0 : RandCounter port map (
      maxSubCount   => std_logic_vector(to_unsigned(MAX_SUB_COUNT_0, maxSubCount_Length)),
      clock         => clock,
      reset         => reset,
      randNum       => randNum0
    );

    RAND_COUNTER_1 : RandCounter port map (
      maxSubCount   => std_logic_vector(to_unsigned(MAX_SUB_COUNT_1, maxSubCount_Length)),
      clock         => clock,
      reset         => reset,
      randNum       => randNum1
    );

    RAND_COUNTER_2 : RandCounter port map (
      maxSubCount   => std_logic_vector(to_unsigned(MAX_SUB_COUNT_2, maxSubCount_Length)),
      clock         => clock,
      reset         => reset,
      randNum       => randNum2
    );

    RAND_COUNTER_3 : RandCounter port map (
      maxSubCount   => std_logic_vector(to_unsigned(MAX_SUB_COUNT_3, maxSubCount_Length)),
      clock         => clock,
      reset         => reset,
      randNum       => randNum3
    );

    RAND_COUNTER_4 : RandCounter port map (
      maxSubCount   => std_logic_vector(to_unsigned(MAX_SUB_COUNT_4, maxSubCount_Length)),
      clock         => clock,
      reset         => reset,
      randNum => randNum4
    );
    --------------------------------------------------------------------------------
    -- Checks for generateEN signal
    -- Sets the output numbers to the current values of their respective counters
    -- Outputs a pule through readyEN
    --------------------------------------------------------------------------------
    SET_RAND_NUM : process(clock, reset)
        variable newNumSet : std_logic := not ACTIVE;
    begin
        if reset = ACTIVE then
            readyEN     <= not ACTIVE;
            newNumSet   := not ACTIVE;

            number0     <= "0000";
            number1     <= "0000";
            number2     <= "0000";
            number3     <= "0000";
            number4     <= "0000";
        elsif rising_edge(clock) then
            if generateEN = ACTIVE then
                number0     <= randNum0;
                number1     <= randNum1;
                number2     <= randNum2;
                number3     <= randNum3;
                number4     <= randNum4;
                newNumSet   := ACTIVE;
            end if;

            if newNumSet = ACTIVE then
                readyEN     <= ACTIVE;
                newNumSet   := not ACTIVE;
            else
                readyEN     <= not ACTIVE;
            end if;
        end if;
    end process;

end RandomNumbers_ARCH;
```

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

----------------------------------------------------------------------------------
--Title: RandCounter.vhd
--Name: Nathaniel Roberts, Mitch Walker
--Date: 3/26/25
--Prof: Scott Tippens
--Desc: Each clock cycle, cubCount increments by 1
--      Once subCount surpasses maxSubCount, subCount is reset to 0,
--      and randNum is incremented by 1.
--
--      Once randNum surpasses 15, randNum is reset to 0
----------------------------------------------------------------------------------

entity RandCounter is
    Port (
        ----------- Inputs
        maxSubCount : in std_logic_vector(2 downto 0);
        clock       : in std_logic;
        reset       : in std_logic;

        ----------- Outputs
        randNum     : out std_logic_vector(3 downto 0)
    );
end RandCounter;

architecture RandCounter_ARCH of RandCounter is
    constant ACTIVE     : std_logic     := '1';
    constant MAX_COUNT  : integer       := 15;
begin

    Counter : process(clock, reset)
        -- Counters used to generate pseudo-random number
        variable subCount   : integer := 0;
        variable count      : integer := 0;
    begin
        if reset = ACTIVE then
            subCount    := 0;
            count       := 0;
            randNum     <= "0000";
        elsif rising_edge(clock) then
            -- Checks if subcount has surpassed maxSubCount
            if subCount < to_integer(unsigned(maxSubCount)) then
                subCount    := subCount + 1;
            else
                subCount    := 0;
                count       := count + 1;
            end if;

            -- Checks if count has surpassed MAX_COUNT
            if count > MAX_COUNT then
                count := 0;
            end if;

            -- Assigns the count to randNum as a std_logic_vector
            randNum <= std_logic_vector(to_unsigned(count, randNum'length));
        end if;
    end process;
end RandCounter_ARCH;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
----------------------------------------------------------------------------------
--Title: NumberChecker
--Name: Nathaniel Roberts, Mitch Walker
--Date: 4/23/25
--Prof: Scott Tippens
--Desc: Number Checker component
--      The component will output a pulse on one of three pins to indicate if
--      the user made all the correct entries or made an error.

--      When there are 5 numbers to check, the gameWinEN gets a pulse
--      if all are entered correctly.

--      At any point a wrong number is made, gameOverEN gets a pulse.
----------------------------------------------------------------------------------
entity NumberChecker is
    port(
        switches : in std_logic_vector(15 downto 0); --the pulsed switch inputs go here

        number0 : in std_logic_vector(3 downto 0);
        number1 : in std_logic_vector(3 downto 0);
        number2 : in std_logic_vector(3 downto 0);
        number3 : in std_logic_vector(3 downto 0);
        number4 : in std_logic_vector(3 downto 0);

        readMode : in std_logic;     -- pull this port low when the display is used

        clock : in std_logic;
        reset : in std_logic;

        gameOverEN : out std_logic;
        gameWinEN : out std_logic
    );
end entity NumberChecker;

architecture NumberChecker_ARCH of NumberChecker is
    signal compare : integer range 0 to 16;

    signal num0 : integer range 0 to 16;
    signal num1 : integer range 0 to 16;
    signal num2 : integer range 0 to 16;
    signal num3 : integer range 0 to 16;
    signal num4 : integer range 0 to 16;

    signal progressCounter : natural range 1 to 5;

begin

    with switches select
        compare <= 1 when "0000000000000001",
                   2 when "0000000000000010",
                   3 when "0000000000000100",
                   4 when "0000000000001000",
                   5 when "0000000000010000",
                   6 when "0000000000100000",
                   7 when "0000000001000000",
                   8 when "0000000010000000",
                   9 when "0000000100000000",
                  10 when "0000001000000000",
                  11 when "0000010000000000",
                  12 when "0000100000000000",
                  13 when "0001000000000000",
                  14 when "0010000000000000",
                  15 when "0100000000000000",
                  16 when "1000000000000000",
```

```vhdl
                  0 when others;

    num0 <= to_integer(unsigned(number0)) + 1;
    num1 <= to_integer(unsigned(number1)) + 1;
    num2 <= to_integer(unsigned(number2)) + 1;
    num3 <= to_integer(unsigned(number3)) + 1;
    num4 <= to_integer(unsigned(number4)) + 1;

    -----------------------------------------------------------------------------------
    -- Main number checker process:
    -- procressCounter keeps track of which number needs to be checked
    -- Each if statement will exit by pulling gameOverEN low if the numx mismatches compare
    -- Inputs are ignored if no switches are active during any rising_edge()
    -----------------------------------------------------------------------------------
    CHECK_NUMBERS : process(clock, reset)
    begin
        if reset = '1' then
            gameOverEN <= '0';
            gameWinEN <= '0';
            progressCounter <= 1;
        elsif rising_edge(clock) then
            gameOverEN <= '0';
            gameWinEN <= '0';

            if (readMode = '1') and (to_integer(unsigned(switches)) > 0) then

                if progressCounter = 1 then
                    if compare = num0 then
                        progressCounter <= progressCounter + 1;
                    else
                        gameOverEN <= '1';
                        progressCounter <= 1;
                    end if;
                elsif progressCounter = 2 then
                    if compare = num1 then
                        progressCounter <= progressCounter + 1;
                    else
                        gameOverEN <= '1';
                        progressCounter <= 1;
                    end if;
                elsif progressCounter = 3 then
                    if compare = num2 then
                        progressCounter <=  progressCounter + 1;
                    else
                        gameOverEN <= '1';
                        progressCounter <= 1;
                    end if;
                elsif progressCounter = 4 then
                    if compare = num3 then
                        progressCounter <=  progressCounter + 1;
                    else
                        gameOverEN <= '1';
                        progressCounter <= 1;
                    end if;
                elsif progressCounter = 5 then
                    if compare = num4 then
                        gameWinEN <= '1';
                        progressCounter <= 1;
                    else
                        gameOverEN <= '1';
                        progressCounter <= 1;
                    end if;
                end if;
            end if;
        end if;
   end process;
end architecture NumberChecker_ARCH;
```

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

----------------------------------------------------------------------------------
--Title: WinPattern.vhd
--Name: Nathaniel Roberts, Mitch Walker
--Date:
--Prof: Scott Tippens
--Desc: Win pattern generator
--      This module handles drawing a specific win patternt to the 16 leds.
--
--      When inactive, the leds are put into high impedace to avoid multiple driven nets
--
----------------------------------------------------------------------------------

entity WinPattern is
    generic(BLINK_COUNT : natural); --(100000000/4)-1;
    port(
        winPatternMode : in std_logic;

        reset: in std_logic;
        clock: in std_logic;

        leds: out std_logic_vector(15 downto 0)
    );
end WinPattern;

architecture WinPattern_ARCH of WinPattern is

    constant ACTIVE: std_logic := '1';

    constant BLANK_LEDS : std_logic_vector(15 downto 0) := "ZZZZZZZZZZZZZZZZ";
    constant PATTERN0_LEDS : std_logic_vector(15 downto 0) := "1010101010101010";
    constant PATTERN1_LEDS : std_logic_vector(15 downto 0) := "0101010101010101";

    signal displayMode : std_logic;
    signal toggle : std_logic;

begin

    ------------------------------------------------------------------------------------
    -- The if this module's WinPatternMode port is high then it will begin displaying
    -- a shifing patter of leds. The toggle signal flips back and fourth between
    -- selecting one pattern or another. If this componenent is not selected, then
    -- the output is in high impedance.
    ------------------------------------------------------------------------------------
    WIN_SHIFT : process(clock, reset)
    begin
        if reset = '1' then
            leds <= BLANK_LEDS;
        elsif rising_edge(clock) then
            if winPatternMode = '1' then
                if toggle = '1' then
                    leds <= PATTERN0_LEDS;
                elsif toggle = '0' then
                    leds <= PATTERN1_LEDS;
                end if;
            else
                leds <= BLANK_LEDS;
            end if;
        end if;
    end process;
    ----------------------------------------------------------------------------------
    -- This process handles how fast the toggle signal fill flip back and fourth.
    ----------------------------------------------------------------------------------
    DISPLAY_RATE: process(reset, clock)
        variable count: integer range 0 to BLINK_COUNT;
    begin
        if (reset = ACTIVE) then
            count := 0;
            toggle <=  not ACTIVE;
        elsif (rising_edge(clock)) then
            if winPatternMode = '1' then
                if (count >= BLINK_COUNT) then
                    count := 0;
                    toggle <= not toggle;
                else
                    count := count + 1;
                end if;
            else
                count := 0;
                toggle <= not ACTIVE;
            end if;
        end if;
    end process DISPLAY_RATE;

end WinPattern_ARCH;
```

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

-----------------------------------------------------------------------------------
--Title: LosePattern.vhd
--Name: Nathaniel Roberts, Mitch Walker
--Date:
--Prof: Scott Tippens
--Desc: Lose Pattern generator
--      And overly complex desin in code to essentially display a draining health bar.
--
--      Each pattern that is ouput to the leds is represented by a custom state.
--
--      The state machine of this file switches through all 16 patterns at the rate of
--      once every 1/4 second
-----------------------------------------------------------------------------------

entity LosePattern is
    generic(BLINK_COUNT : natural);
    port(

        losePatternEN : in std_logic;

        reset: in std_logic;
        clock: in std_logic;

        leds: out std_logic_vector(15 downto 0);

        losePatternIsBusy : out std_logic
    );
end LosePattern;

architecture LosePattern_ARCH of LosePattern is

    type States_t is (BLANK,
                      PATTERN0,
                      PATTERN1,
                      PATTERN2,
                      PATTERN3,
                      PATTERN4,
                      PATTERN5,
                      PATTERN6,
                      PATTERN7,
                      PATTERN8,
                      PATTERN9,
                      PATTERN10,
                      PATTERN11,
                      PATTERN12,
                      PATTERN13,
                      PATTERN14,
                      PATTERN15);
    signal currentState: States_t;
    signal nextState: States_t;

    constant ACTIVE: std_logic := '1';
    constant HIGH_Z_LEDS : std_logic_vector(15 downto 0) := "ZZZZZZZZZZZZZZZZ";
    constant PATTERN0_LEDS : std_logic_vector(15 downto 0)  := "1111111111111111";
    constant PATTERN1_LEDS : std_logic_vector(15 downto 0)  := "1111111111111110";
    constant PATTERN2_LEDS : std_logic_vector(15 downto 0)  := "1111111111111100";
    constant PATTERN3_LEDS : std_logic_vector(15 downto 0)  := "1111111111111000";
    constant PATTERN4_LEDS : std_logic_vector(15 downto 0)  := "1111111111110000";
    constant PATTERN5_LEDS : std_logic_vector(15 downto 0)  := "1111111111100000";
    constant PATTERN6_LEDS : std_logic_vector(15 downto 0)  := "1111111111000000";
    constant PATTERN7_LEDS : std_logic_vector(15 downto 0)  := "1111111110000000";
    constant PATTERN8_LEDS : std_logic_vector(15 downto 0)  := "1111111100000000";
    constant PATTERN9_LEDS : std_logic_vector(15 downto 0)  := "1111111000000000";
    constant PATTERN10_LEDS : std_logic_vector(15 downto 0) := "1111110000000000";
    constant PATTERN11_LEDS : std_logic_vector(15 downto 0) := "1111100000000000";
    constant PATTERN12_LEDS : std_logic_vector(15 downto 0) := "1111000000000000";
    constant PATTERN13_LEDS : std_logic_vector(15 downto 0) := "1110000000000000";
    constant PATTERN14_LEDS : std_logic_vector(15 downto 0) := "1100000000000000";
    constant PATTERN15_LEDS : std_logic_vector(15 downto 0) := "1000000000000000";

    signal stateMachineControl: std_logic;

begin

    ------------------------------------------------------------------------------------
    -- The state machine register that keeps track of when to switch to the next state.
    ------------------------------------------------------------------------------------
    STATE_REGISTER: process(reset, clock)
        begin
            if reset = ACTIVE then
                currentState <= BLANK;
            elsif rising_edge(clock) then
                currentState <= nextState;
            end if;
    end process;

    ------------------------------------------------------------------------------------
    -- The overly complex and long winded state machine that selects different signals
    -- to reach the output. If this module is done, then the ouput is high-Z. While
    -- this module runs, output a busy signal to tell the rest of the design it busy.
    ------------------------------------------------------------------------------------
    WIN_PATTERN_SM: process(currentState, stateMachineControl, losePatternEN)
    begin
        case CurrentState is
    -------------------------------------------------------------------------------BLANK
            when BLANK =>
                leds <= HIGH_Z_LEDS;
                losePatternIsBusy <= not ACTIVE;
                if (losePatternEN = ACTIVE) then
                    nextState <= PATTERN0;
                else
                    nextState <= BLANK;
                end if;
    ----------------------------------------------------------------------------PATTERN0

            when PATTERN0 =>
                leds <= PATTERN0_LEDS;
                losePatternIsBusy <= ACTIVE;
                if (stateMachineControl = not ACTIVE) then
                    nextState <= PATTERN1;
                else
                    nextState <= PATTERN0;
                end if;
    -----------------------------------------------------------------------------PATTERN1

            when PATTERN1 =>
                leds <= PATTERN1_LEDS;
                losePatternIsBusy <= ACTIVE;
                if (stateMachineControl = ACTIVE) then
                nextState <= PATTERN2;
            else
                nextState <= PATTERN1;
            end if;
----------------------------------------------------------------------------PATTERN2

        when PATTERN2 =>
            leds <= PATTERN2_LEDS;
            losePatternIsBusy <= ACTIVE;
            if (stateMachineControl = not ACTIVE) then
                nextState <= PATTERN3;
            else
                nextState <= PATTERN2;
            end if;
----------------------------------------------------------------------------PATTERN3

        when PATTERN3 =>
            leds <= PATTERN3_LEDS;
            losePatternIsBusy <= ACTIVE;
            if (stateMachineControl = ACTIVE) then
                nextState <= PATTERN4;
            else
                nextState <= PATTERN3;
            end if;
----------------------------------------------------------------------------PATTERN4

        when PATTERN4 =>
            leds <= PATTERN4_LEDS;
            losePatternIsBusy <= ACTIVE;
            if (stateMachineControl = not ACTIVE) then
                nextState <= PATTERN5;
            else
                nextState <= PATTERN4;
            end if;
----------------------------------------------------------------------------PATTERN5

        when PATTERN5 =>
            leds <= PATTERN5_LEDS;
            losePatternIsBusy <= ACTIVE;
            if (stateMachineControl = ACTIVE) then
                nextState <= PATTERN6;
            else
                nextState <= PATTERN5;
            end if;
----------------------------------------------------------------------------PATTERN6

        when PATTERN6 =>
            leds <= PATTERN6_LEDS;
            losePatternIsBusy <= ACTIVE;
            if (stateMachineControl = not ACTIVE) then
                nextState <= PATTERN7;
            else
                nextState <= PATTERN6;
            end if;
----------------------------------------------------------------------------PATTERN7

        when PATTERN7 =>
            leds <= PATTERN7_LEDS;
            losePatternIsBusy <= ACTIVE;
            if (stateMachineControl = ACTIVE) then
                nextState <= PATTERN8;
            else
                nextState <= PATTERN7;
            end if;
----------------------------------------------------------------------------PATTERN8

        when PATTERN8 =>
```

```
            leds <= PATTERN8_LEDS;
            losePatternIsBusy <= ACTIVE;
            if (stateMachineControl = not ACTIVE) then
                nextState <= PATTERN9;
            else
                nextState <= PATTERN8;
            end if;
-----------------------------------------------------------------------------PATTERN9

        when PATTERN9 =>
            leds <= PATTERN9_LEDS;
            losePatternIsBusy <= ACTIVE;
            if (stateMachineControl = ACTIVE) then
                nextState <= PATTERN10;
            else
                nextState <= PATTERN9;
            end if;
-----------------------------------------------------------------------------PATTERN10

        when PATTERN10 =>
            leds <= PATTERN10_LEDS;
            losePatternIsBusy <= ACTIVE;
            if (stateMachineControl = not ACTIVE) then
                nextState <= PATTERN11;
            else
                nextState <= PATTERN10;
            end if;
-----------------------------------------------------------------------------PATTERN11

        when PATTERN11 =>
            leds <= PATTERN11_LEDS;
            losePatternIsBusy <= ACTIVE;
            if (stateMachineControl = ACTIVE) then
                nextState <= PATTERN12;
            else
                nextState <= PATTERN11;
            end if;
-----------------------------------------------------------------------------PATTERN12

        when PATTERN12 =>
            leds <= PATTERN12_LEDS;
            losePatternIsBusy <= ACTIVE;
            if (stateMachineControl = not ACTIVE) then
                nextState <= PATTERN13;
            else
                nextState <= PATTERN12;
            end if;
-----------------------------------------------------------------------------PATTERN13

        when PATTERN13 =>
            leds <= PATTERN13_LEDS;
            losePatternIsBusy <= ACTIVE;
            if (stateMachineControl = ACTIVE) then
                nextState <= PATTERN14;
            else
                nextState <= PATTERN13;
            end if;
-----------------------------------------------------------------------------PATTERN14

        when PATTERN14 =>
            leds <= PATTERN14_LEDS;
            losePatternIsBusy <= ACTIVE;
            if (stateMachineControl = not ACTIVE) then
                nextState <= PATTERN15;
            else
                nextState <= PATTERN14;
            end if;
```

```
    ----------------------------------------------------------------------------PATTERN15

            when PATTERN15 =>
                leds <= PATTERN15_LEDS;
                losePatternIsBusy <= ACTIVE;
                if (stateMachineControl = ACTIVE) then
                    nextState <= BLANK;
                else
                    nextState <= PATTERN15;
                end if;
        end case;
    end process;

    DISPLAY_RATE: process(reset, clock)
        variable count: integer range 0 to BLINK_COUNT;
    begin
        if (reset = ACTIVE) then
            count := 0;
            stateMachineControl <=  not ACTIVE;
        elsif (rising_edge(clock)) then
            if (count >= BLINK_COUNT) then
                count := 0;
                stateMachineControl <= not stateMachineControl;
            else
                count := count + 1;
            end if;
        end if;
    end process DISPLAY_RATE;

end LosePattern_ARCH;
```

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

----------------------------------------------------------------------------------
--Title: LedSegments
--Name: Nathaniel Roberts, Mitch Walker
--Date: 3/26/25
--Prof: Scott Tippens
--Desc: Bar Led Driver file
--      This file simply chooses which led on the Basys3 board will recieve the active
--      level. The inupt is a 4 bit binary number and output is 16 bit wide vector.
--      matching the position of each of the 16 leds on the Basys3 board.
----------------------------------------------------------------------------------


entity LedSegments is
    port(
        binary4Bit : in std_logic_vector(3 downto 0);
        outputMode : in std_logic;

        leds : out std_logic_vector(15 downto 0)
    );
end entity LedSegments;

architecture LedSegments_ARCH of LedSegments is

begin

    DRIVE_LEDS : process(binary4Bit, outputMode)
    begin
        if outputMode = '0' then
            leds <= (others => 'Z');
        else
            for i in 0 to 15 loop
                if to_integer(unsigned(binary4BIt)) = i then
                    leds(i) <= '1';
                else
                    leds(i) <= '0';
                end if;
            end loop;
        end if;
    end process;

end architecture LedSegments_ARCH;
```

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

--------------------------------------------------------------------------------------
--Title: BCD
--Name: Nathaniel Roberts, Mitch Walker
--Date: 3/26/25
--Prof: Scott Tippens
--Desc: Binary to BCD converter file.
--      This component simply converts a 4 bit binary number into two seperate
--      4 bit numbers to represent the Tens and ones place combinationally.
--------------------------------------------------------------------------------------


entity BCD is
    port(
        binary4Bit : in std_logic_vector(3 downto 0);

        decimalOnes : out std_logic_vector(3 downto 0);
        decimalTens : out std_logic_vector(3 downto 0)
    );
end entity BCD;


architecture BCD_ARCH of BCD is
begin
    with binary4Bit select
        decimalOnes <= "0000" when "0000",
                       "0001" when "0001",
                       "0010" when "0010",
                       "0011" when "0011",
                       "0100" when "0100",
                       "0101" when "0101",
                       "0110" when "0110",
                       "0111" when "0111",
                       "1000" when "1000",
                       "1001" when "1001",
                       -------------------
                       "0000" when "1010",
                       "0001" when "1011",
                       "0010" when "1100",
                       "0011" when "1101",
                       "0100" when "1110",
                       "0101" when "1111",
                       (others => '0') when others;

    with binary4Bit select
        decimalTens <= "0000" when "0000",
                       "0000" when "0001",
                       "0000" when "0010",
                       "0000" when "0011",
                       "0000" when "0100",
                       "0000" when "0101",
                       "0000" when "0110",
                       "0000" when "0111",
                       "0000" when "1000",
                       "0000" when "1001",
                       -------------------
                       "0001" when "1010",
                       "0001" when "1011",
                       "0001" when "1100",
                       "0001" when "1101",
                       "0001" when "1110",
                       "0001" when "1111",
                       (others => '0') when others;
end architecture BCD_ARCH;
```

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

----------------------------------------------------------------------------------
--Title: Memory Game
--Name: Nathaniel Roberts, Mitch Walker
--Date: 4/23/25
--Prof: Scott Tippens
--Desc: Memory Game testbench file
--      This file uses seperate processes to find what the numerical display numbers
--      are and output the cooresponding one pulse wide switch signal
--
--      The testbench can be configured for any timing range based on the generic
--      parameters.
----------------------------------------------------------------------------------

entity MemoryGameTB is
end entity;

architecture Behavioral of MemoryGameTB is

    ----------------------------------------------------------------------------------
    --Unit under test definition
    ----------------------------------------------------------------------------------
    component MemoryGame
        generic(
            MAX_COUNT_SCALER : integer;
            SCALE_AMOUNT     : integer;
            MAX_TOGGLE_COUNT : integer;
            BLINK_COUNT      : integer
        );
        port(
            switches    : in  std_logic_vector(15 downto 0);
            start       : in  std_logic;
            clock       : in  std_logic;
            reset       : in  std_logic;
            leds        : out std_logic_vector(15 downto 0);
            outputScore : out std_logic_vector(7 downto 0);
            blanks      : out std_logic_vector(3 downto 0)
        );
    end component MemoryGame;

    ----------------------------------------------------------------------------------
    --testbench signals
    ----------------------------------------------------------------------------------
    signal switches : std_logic_vector(15 downto 0) := (others => '0');
    signal start : std_logic;
    signal clock : std_logic;
    signal reset : std_logic;
    signal leds : std_logic_vector(15 downto 0);
    signal blanks : std_logic_vector(3 downto 0);
    signal outputScore : std_logic_vector(7 downto 0);

    constant CLOCK_PERIOD     : time := 10 ns;
    constant MAX_COUNT_SCALER : integer := 0;
    constant SCALE_AMOUNT     : integer := 50;
    constant MAX_TOGGLE_COUNT : integer := 100;
    constant BLINK_COUNT      : integer := 25;

    type testVector_t is array(0 to 4) of integer;
    signal number_reg : testVector_t;

    signal load_number_regEN : std_logic := '0';
    signal load_number_reg_completeEN : std_logic := '0';

    --MAX_TOGGLE_COUNT * CLOCK_PERIOD = 1000
    constant getNumberDelay : time := 1000 ns;
    --BLINK_COUNT * COCK_PERIOD = 250
    --25 * 10 = 250
    constant patternDelay : time := 250 ns;


begin

    ------------------------------------------------------------------------------------
    --Design instantiation
    ------------------------------------------------------------------------------------
    UUT : MemoryGame
    generic map(
        MAX_COUNT_SCALER => MAX_COUNT_SCALER,
        SCALE_AMOUNT     => SCALE_AMOUNT,
        MAX_TOGGLE_COUNT => MAX_TOGGLE_COUNT,
        BLINK_COUNT      => BLINK_COUNT
    )
    port map(
        switches          => switches,
        start             => start,
        clock             => clock,
        reset             => reset,

        ------------------outputs
        leds              => leds,
        outputScore       => outputScore,
        blanks            => blanks
    );


    ------------------------------------------------------------------------------------
    -- Clock generator
    ------------------------------------------------------------------------------------
    CLOCK_GEN : process
    begin
        clock <= '0';
        wait for (CLOCK_PERIOD / 2);
        clock <= '1';
        wait for (CLOCK_PERIOD / 2);
    end process;

    ------------------------------------------------------------------------------------
    -- Reset button generator
    ------------------------------------------------------------------------------------
    RESET_GEN : process
    begin
        wait until rising_edge(clock);
        wait until rising_edge(clock);
        reset <= '1';
        wait until rising_edge(clock);
        reset <= '0';
        wait;
    end process;

    ------------------------------------------------------------------------------------
    -- This process can scan the output of the switches and assign those values into
    -- an array of integers.
    ------------------------------------------------------------------------------------
    TEST_REGISTER : process
    begin
        wait until rising_edge(load_number_regEN);
        wait for getNumberDelay;
        for k in 0 to 4 loop
            for i in 0 to 15 loop
                if leds(i) = '1' then
                    number_reg(k) <= i;
                end if;
            end loop;
            wait for getNumberDelay; --dont start i loop until next number pops up
            wait for getNumberDelay;
        end loop;

        load_number_reg_completeEN <= '1';
        wait until rising_edge(clock);
        load_number_reg_completeEN <= '0';
        wait until rising_edge(clock);


    end process;

    -------------------------------------------------------------------------------------
    -- Main stimulus process with necessary timing procedures
    -------------------------------------------------------------------------------------
    STIMULUS : process
    begin
        wait for 500 ns;

        start <= '1';                   --start button pressed
        wait until rising_edge(clock);
        start <= '0';
        wait until rising_edge(clock);

        --first round-----------------------------------------------------------------
        load_number_regEN <= '1';        --trigger the TEST_REGISTER process
        wait until rising_edge(clock);   --to load numbers as the numbers are brought up.
        load_number_regEN <= '0';
        wait until rising_edge(clock);

        wait until rising_edge(load_number_reg_completeEN); --wait for the loop to be done

        switches(number_reg(0)) <= '1'; --plug in numbers
        wait until rising_edge(clock);
        switches(number_reg(0)) <= '0';
        wait until rising_edge(clock);

        switches(number_reg(1)) <= '1';
        wait until rising_edge(clock);
        switches(number_reg(1)) <= '0';
        wait until rising_edge(clock);

        switches(number_reg(2)) <= '1';
        wait until rising_edge(clock);
        switches(number_reg(2)) <= '0';
        wait until rising_edge(clock);

        switches(number_reg(3)) <= '1';
        wait until rising_edge(clock);
        switches(number_reg(3)) <= '0';
        wait until rising_edge(clock);

        switches(number_reg(4)) <= '1';
        wait until rising_edge(clock);
        switches(number_reg(4)) <= '0';
        wait until rising_edge(clock);

        --second round----------------------------------------------------------------
        wait for 2000 ns;

        load_number_regEN <= '1';        --trigger the TEST_REGISTER process
        wait until rising_edge(clock);   --to load numbers as the numbers are brought up.
        load_number_regEN <= '0';
        wait until rising_edge(clock);
```

```
        wait until rising_edge(load_number_reg_completeEN); --wait for the loop to be done

        switches(number_reg(0)) <= '1'; --plug in numbers
        wait until rising_edge(clock);
        switches(number_reg(0)) <= '0';
        wait until rising_edge(clock);

        switches(number_reg(1)) <= '1';
        wait until rising_edge(clock);
        switches(number_reg(1)) <= '0';
        wait until rising_edge(clock);

        switches(number_reg(2)) <= '1';
        wait until rising_edge(clock);
        switches(number_reg(2)) <= '0';
        wait until rising_edge(clock);

        switches(number_reg(3)) <= '1';
        wait until rising_edge(clock);
        switches(number_reg(3)) <= '0';
        wait until rising_edge(clock);

        switches(number_reg(4)) <= '1';
        wait until rising_edge(clock);
        switches(number_reg(4)) <= '0';
        wait until rising_edge(clock);

        --third round---------------------------------------------------------------------
        wait for 1100 ns;

        load_number_regEN <= '1';        --trigger the TEST_REGISTER process
        wait until rising_edge(clock);   --to load numbers as the numbers are brought up.
        load_number_regEN <= '0';
        wait until rising_edge(clock);

        wait until rising_edge(load_number_reg_completeEN); --wait for the loop to be done

        switches(number_reg(0)) <= '1'; --plug in numbers
        wait until rising_edge(clock);
        switches(number_reg(0)) <= '0';
        wait until rising_edge(clock);

        switches(number_reg(1)) <= '1';
        wait until rising_edge(clock);
        switches(number_reg(1)) <= '0';
        wait until rising_edge(clock);

        switches(number_reg(2)) <= '1';
        wait until rising_edge(clock);
        switches(number_reg(2)) <= '0';
        wait until rising_edge(clock);

        switches(number_reg(3)) <= '1';
        wait until rising_edge(clock);
        switches(number_reg(3)) <= '0';
        wait until rising_edge(clock);

        switches(number_reg(4)) <= '1';
        wait until rising_edge(clock);
        switches(number_reg(4)) <= '0';
        wait until rising_edge(clock);
        wait;
    end process;

end architecture Behavioral ;
```

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

-------------------------------------------------------------------------------------
--Title: MemoryGame
--Name: Nathaniel Roberts, Mitch Walker
--Date: 4/23/25
--Prof: Scott Tippens
--Desc: Wrapper file
--
--      This Top wrapper uses 3 generate statements for the array of 16 switches
--      The first in the signal chain is a SynchronizerChain component to handle
--      meta stability. The second component debounces the switches and the final
--      component pulsees each switch.
--
--      The same hardware signal chain exists for the start button.
--
-------------------------------------------------------------------------------------

entity MemoryGame_Basys3 is
    generic(
        NUM_OF_SWITCHES : integer := 16;
        CHAIN_SIZE : integer := 2;
        DELAY_COUNT : integer := 10_000_000 --10ms on a 100MHz clock = 1M
        );
    port(
        sw : in std_logic_vector(NUM_OF_SWITCHES-1 downto 0);
        btnC : in std_logic; --start/restart button
        btnD : in std_logic; --reset/blanks the screens
        clk : in std_logic;

        led : out std_logic_vector(15 downto 0);
        an  : out std_logic_vector(3 downto 0);
        seg : out std_logic_vector(6 downto 0)
    );
end entity;


architecture MemoryGame_Basys3_ARCH of MemoryGame_Basys3 is

    -------------------------------------------------------------------------------------
    --component definitions
    -------------------------------------------------------------------------------------
    component MemoryGame
       generic(
              MAX_COUNT_SCALER : integer;
              SCALE_AMOUNT     : integer;
              MAX_TOGGLE_COUNT : integer;
              BLINK_COUNT      : integer
        );
       port(
              switches    : in  std_logic_vector(15 downto 0);
              start       : in  std_logic;
              clock       : in  std_logic;
              reset       : in  std_logic;
              leds        : out std_logic_vector(15 downto 0);
              outputScore : out std_logic_vector(7 downto 0);
              blanks      : out std_logic_vector(3 downto 0)
       );
    end component MemoryGame;

    component SevenSegmentDriver is
        port(
            reset: in std_logic;
            clock: in std_logic;
            digit3: in std_logic_vector(3 downto 0);    --leftmost digit
            digit2: in std_logic_vector(3 downto 0);    --2nd from left digit
            digit1: in std_logic_vector(3 downto 0);    --3rd from left digit
            digit0: in std_logic_vector(3 downto 0);    --rightmost digit

            blank3: in std_logic;    --leftmost digit
            blank2: in std_logic;    --2nd from left digit
            blank1: in std_logic;    --3rd from left digit
            blank0: in std_logic;    --rightmost digit

            sevenSegs: out std_logic_vector(6 downto 0);    --MSB=g, LSB=a
            anodes:    out std_logic_vector(3 downto 0)    --MSB=leftmost digit
        );
    end component;

    component SynchronizerChain is
        generic (CHAIN_SIZE: positive);
        port (
            reset:    in  std_logic;
            clock:    in  std_logic;
            asyncIn:  in  std_logic;
            syncOut:  out std_logic
        );
    end component;

    component Debouncer is
        generic( DELAY_COUNT : positive);
        port(
            bitIn : in std_logic;
            clock : in std_logic;
            reset : in std_logic;

            debouncedOut : out std_logic
        );
    end component;

    component LevelDetector is
        port (
            reset:     in  std_logic;
            clock:     in  std_logic;
            trigger:   in  std_logic;
            pulseOut:  out std_logic
        );
    end component;

    -------------------------------------------------------------------------------
    -- synchronized 16 bit wide vector of switches
    -- vector is fed to a debouncer then finally a pulse controller
    -------------------------------------------------------------------------------
    signal synchedSwitches : std_logic_vector(NUM_OF_SWITCHES-1 downto 0);
    signal debouncedSwitches : std_logic_vector(NUM_OF_SWITCHES-1 downto 0);
    signal pulsedSwitches : std_logic_vector(NUM_OF_SWITCHES-1 downto 0);

    -------------------------------------------------------------------------------
    -- synchronised start button signal path
    -------------------------------------------------------------------------------
    signal startButtonSync : std_logic;
    signal startButtonDebounced : std_logic;
    signal startButtonPulsed : std_logic;

    -------------------------------------------------------------------------------
    --signals for SevenSegmentDriver component
    -------------------------------------------------------------------------------
    signal blanks : std_logic_vector(3 downto 0);
    signal outputScore : std_logic_vector(7 downto 0);
```

```
begin------------------------------------------------------------------------------begin

-----------------------------------------------------------------------------------
--component insantiations
-----------------------------------------------------------------------------------

MEMORY_GAME : component MemoryGame
    generic map(
          MAX_COUNT_SCALER => 90_000_000,
          SCALE_AMOUNT     => 15_000_000,
          MAX_TOGGLE_COUNT => 100_000_000,
          BLINK_COUNT      => 25_000_000
    )
port map(
    switches         => pulsedSwitches,

    start            => startButtonPulsed,

    clock            => clk,
    reset            => btnD,

    leds             => led,

    outputScore      => outputScore,

    blanks           => blanks
);

SEGMENT_DRIVER : component SevenSegmentDriver port map(
    reset     => btnD,
    clock     => clk,

    digit3    => outputScore(7 downto 4),
    digit2    => outputScore(3 downto 0),
    digit1    => "0000",
    digit0    => "0000",

    blank3    => blanks(3),
    blank2    => blanks(2),
    blank1    => blanks(1),
    blank0    => blanks(0),


    sevenSegs => seg,
    anodes    => an
);

-----------------------------------------------------------------------------------
-- generate chain of 2 flip-flops for each vector switch element
-----------------------------------------------------------------------------------
SYNC : for i in 0 to NUM_OF_SWITCHES-1 generate
    SYNC_X : component SynchronizerChain
        generic map(
            CHAIN_SIZE => CHAIN_SIZE
        )
        port map(
            reset   => btnD,
            clock   => clk,
            asyncIn => sw(i),
            syncOut => synchedSwitches(i) --all switches now clock synced
        );
end generate;

-----------------------------------------------------------------------------------
-- generate debounce control for each vector switch element
-----------------------------------------------------------------------------------
DEBOUNCE : for i in 0 to NUM_OF_SWITCHES-1 generate
```

```vhdl
        DEBOUNCE_X : component Debouncer
            generic map(
                DELAY_COUNT => DELAY_COUNT
            )
            port map(
                bitIn        => synchedSwitches(i),
                clock        => clk,
                reset        => btnD,
                debouncedOut => debouncedSwitches(i) --all switches now debounced
            );
    end generate;

    -------------------------------------------------------------------------------
    -- make each switch pulse, once per activation, reset once the switch is down
    -------------------------------------------------------------------------------
    PULSE : for i in 0 to NUM_OF_SWITCHES-1 generate
        PULSE_X : component LevelDetector
            port map(
                reset    => btnD,
                clock    => clk,
                trigger  => debouncedSwitches(i),
                pulseOut => pulsedSwitches(i) --all switches now pulse controlled
            );
    end generate;


    -------------------------------------------------------------------------------
    -- The following 3 components synchronize, debounce and pulse the start button.
    -------------------------------------------------------------------------------
    START_BUTTON_SYNC : SynchronizerChain
        generic map(
            CHAIN_SIZE => CHAIN_SIZE
        )
        port map(
            reset   => btnD,
            clock   => clk,
            asyncIn => btnC,
            syncOut => startButtonSync --start button is now synced
    );

    START_BUTTON_DEBOUNCE : component Debouncer
        generic map(
            DELAY_COUNT => DELAY_COUNT
        )
        port map(
            bitIn        => startButtonSync,
            clock        => clk,
            reset        => btnD,
            debouncedOut => startButtonDebounced --start button is now debounced
    );

    START_BUTTON_PULSE : LevelDetector
        port map(
            reset    => btnD,
            clock    => clk,
            trigger  => startButtonDebounced,
            pulseOut => startButtonPulsed --start button is now pulsed
    );

end MemoryGame_Basys3_ARCH;
```

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

-------------------------------------------------------------------------------------
--Title: Debouncer.vhd
--Name: Nathaniel Roberts, Mitch Walker
--Date: 4/23/25
--Prof: Scott Tippens
--Desc: Counter based denouncing component.
--      This component checks to see if the user input has remained in its position
--      for the full duration of the count. If that signal input is different than the
--      output and the counter has also not yet reached its terminal value, then the
--      input is not yet stable and the component keeps the signal from reflecting to
--      the output.
-------------------------------------------------------------------------------------

entity Debouncer is
    generic(
        DELAY_COUNT : positive
    );
    port(
        bitIn : in std_logic;
        clock : in std_logic;
        reset : in std_logic;

        debouncedOut : out std_logic
    );
end entity Debouncer;


architecture Debouncer_ARCH of Debouncer is
    signal counter : integer range 0 to DELAY_COUNT;
    signal bitReg : std_logic;
begin
    SCAN_INPUT : process(clock, reset)
    begin
        if reset = '1' then
            counter <= 0;
        elsif rising_edge(clock) then
            if bitIn /= bitReg and counter < DELAY_COUNT then
                counter <= counter + 1;
            elsif counter >= DELAY_COUNT then
                bitReg <= bitIn;
                counter <= 0;
            else
                counter <= 0;
            end if;
        end if;
    end process;
    debouncedOut <= bitReg;
end architecture Debouncer_ARCH;
```