```
--------------------------------------------------------------------------------
--                                                                            --
--  PwmAudioPlayer Logic Design                                               --
--                                                                            --
--  This is an entity declaration and architecture definition for the PWM     --
--  audio player system. The system pipelines audio samples from an EPROM     --
--  and outputs the samples via PWM at 8 kHz.                                 --
--                                                                            --
--  Inputs:                                                                   --
--           CLK              - 32MHz system clock                            --
--           RESET            - Asynchronous active high reset                --
--           BTN[3..0]        - 4 input buttons                               --
--           AudioData[7..0]  - Audio sample data input                       --
--                                                                            --
--  Outputs:                                                                  --
--           AudioAddr[18..0] - Address of EPROM to read from                 --
--           AudioPWMOut      - Output audio using PWM                        --
--                                                                            --
--  Revision History:                                                         --
--      11/11/2024  Edward Speer  Initial Revision                            --
--      11/13/2024  Edward Speer  Remap addr & btns to reconfigured modules   --
--                                                                            --
--------------------------------------------------------------------------------

-- IMPORTS
library ieee;
use ieee.std_logic_1164.all;

--
-- PwmAudioPlayer entity declaration
--

entity PwmAudioPlayer is
    port (
        CLK         : in     std_logic;                     -- 32 MHz sys clock
        RESET       : in     std_logic;                     -- Asynch reset
        BTN         : in     std_logic_vector(3 downto 0);  -- User buttons in
        AudioData   : in     std_logic_vector(7 downto 0);  -- EPROM data in
        AudioAddr   : buffer std_logic_vector(18 downto 0); -- EPROM addr out
        AudioPWMOut : out    std_logic                      -- PWM output
    );
end PwmAudioPlayer;

--
-- PwmAudioPlayer behavioral architecture
--

architecture behavioral of PwmAudioPlayer is

    --
    -- Component declaration of shared counter module
```

```
    --

    component Cntr8ClockDiv
        port (
            CLK      : in     std_logic;                     -- 32 MHz sys clk
            RESET    : in     std_logic;                     -- Async reset
            CLK_8kHz : buffer std_logic;                     -- 8 kHz clk out
            Cntr8    : out    std_logic_vector(7 downto 0)   -- 8 bit cnt out
        );
    end component;

    --
    -- Component declaration of EPROM data address unit
    --

    component AddrUnit
        port (
            CLK_8kHz  : in     std_logic;                    -- 8 kHz clock
            RESET     : in     std_logic;                    -- Async reset
            Btn       : in     std_logic_vector(3 downto 0); -- User buttons in
            AudioAddr : buffer std_logic_vector(18 downto 0);-- 19 bit addr out
            Enable    : buffer std_logic                     -- Enable PWM out
        );
    end component;

    --
    -- Component declaration of PWM driver block
    --

    component PwmDriver
        port (
            CLK_8kHz    : in  std_logic;                    -- 8 kHz clk
            RESET       : in  std_logic;                    -- Asynch reset
            enable      : in  std_logic;                    -- enable pwm out
            AudioData   : in  std_logic_vector(7 downto 0); -- 8 bit sample
            Cntr8       : in  std_logic_vector(7 downto 0); -- 8 bit counter in
            AudioPWMOut : out std_logic                     -- PWM output
        );
    end component;

    --
    -- Signals used to connect up the blocks in the system
    --

    signal CLK_8kHz  : std_logic;
    signal enable    : std_logic;
    signal Cntr8     : std_logic_vector(7 downto 0);

begin

    --
```

```
    -- Port maps for each of the components of the system
    --

    U1 : Cntr8ClockDiv
        port map (
            CLK      => CLK,
            RESET    => RESET,
            CLK_8kHz => CLK_8kHz,
            Cntr8    => Cntr8
        );

    U2 : AddrUnit
        port map (
            CLK_8kHz  => CLK_8kHz,
            RESET     => RESET,
            Btn       => BTN,
            AudioAddr => AudioAddr,
            Enable    => enable
        );

    U3 : PwmDriver
        port map (
            CLK_8kHz    => CLK_8kHz,
            RESET       => RESET,
            enable      => enable,
            AudioData   => AudioData,
            Cntr8       => Cntr8,
            AudioPWMOut => AudioPWMOut
        );

end behavioral;

-- Configure the PWMAudioPlayer implementation
configuration PWMAudioPlayer_IMPLEMENTATION of PWMAudioPlayer is
    for behavioral
        for U1 : Cntr8ClockDiv
            use entity work.Cntr8ClockDiv(implementation);
        end for;
        for U2 : AddrUnit
            use entity work.AddrUnit(implementation);
        end for;
        for U3 : PwmDriver
            use entity work.PwmDriver(behavioral);
        end for;
    end for;
end PWMAudioPlayer_IMPLEMENTATION;
```

```
--------------------------------------------------------------------------------
--                                                                            --
--  Cntr8ClockDiv Logic Design                                                --
--                                                                            --
--  This is an entity declaration and architectures for a clock divider       --
--  module that serves a dual purpose as an 8 bit counter. The unit is an 11  --
--  bit counter that divies the input clock by 4096 to produce an 8 kHz       --
--  output clock, giving the sample rate of the audio. 8 bits of the counter  --
--  are output as an 8 bit counter to be used as a PWM duty cycle generator.  --
--                                                                            --
--  Inputs:                                                                   --
--           CLK         - 32 MHz system clock                                --
--           RESET       - Asynchronous active high reset                     --
--                                                                            --
--  Outputs:                                                                  --
--           CLK_8kHz    - 8 kHz output clock                                 --
--           Cntr8[7..0] - 8 bit counter output                               --
--                                                                            --
--  Revision History:                                                         --
--      11/14/2024  Edward Speer  Initial Revision                            --
--                                                                            --
--------------------------------------------------------------------------------

--
-- Imports
--

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

--
-- Cntr8ClockDiv entity declaration
--

entity Cntr8ClockDiv is
    port (
        CLK      : in     std_logic;                    -- 32 MHz system clock
        RESET    : in     std_logic;                    -- Async reset
        CLK_8kHz : buffer std_logic;                    -- 8 kHz output clock
        Cntr8    : out    std_logic_vector(7 downto 0)  -- 8 bit counter output
    );
end Cntr8ClockDiv;

--
-- Cntr8ClockDiv behavioral architecture
--

architecture behavioral of Cntr8ClockDiv is

    -- Counter variable
```

```
    signal cnt : unsigned(12 downto 0) := "0000000000000"; -- 11 bit counter

begin

    -- Concurrent logic assignments
    Cntr8 <= std_logic_vector(cnt(7 downto 0)); -- 8 bit counter output

    process(CLK, RESET)
    begin
        if RESET = '1' then
            cnt <= "0000000000000"; -- Reset counter
            CLK_8kHz <= '0'; -- Reset 8 kHz clock
        elsif rising_edge(CLK) then
            cnt <= cnt + 1; -- Increment counter
            if cnt = 2047 then
                CLK_8kHz <= '0'; -- Toggle 8 kHz clock if cnt at max
            end if;
            if cnt = 4095 then
                CLK_8kHz <= '1'; -- Toggle 8 kHz clock if cnt at max
                cnt <= "0000000000000";
            end if;
        end if;
    end process;

end behavioral;

--
-- Cntr8ClockDiv implementation architecture
--

architecture implementation of Cntr8ClockDiv is

    -- Counter variable
    signal cnt : unsigned(10 downto 0); -- 11 bit counter

begin

    -- Concurrent logic assignments
    Cntr8 <= std_logic_vector(cnt(7 downto 0)); -- 8 bit counter output

    process(CLK, RESET)
    begin
        if RESET = '1' then
            cnt <= (others => '0'); -- Reset counter
            CLK_8kHz <= '0'; -- Reset 8 kHz clock
        elsif rising_edge(CLK) then
            cnt <= cnt + 1; -- Increment counter
            if cnt = 2047 then
                CLK_8kHz <= not CLK_8kHz; -- Toggle 8 kHz clock if cnt at max
            end if;
        end if;
```

```
    end process;

end implementation;
```

```
--------------------------------------------------------------------------------
--                                                                            --
--  Cntr8ClockDiv Test Bench                                                  --
--                                                                            --
--  This is a test bench for the Cntr8ClockDiv block of the PWM Audio Player  --
--  system. The test bench thoroughly tests the entity by exercising it and   --
--  checking the timing of the output 8 kHz clock and the 8 bit counter       --
--  output. The test bench entity is called Cntr8ClockDiv_tb and it is        --
--  currently defined to test the behavioral architecture of the              --
--  Cntr8ClockDiv unit.                                                       --
--                                                                            --
--  Revision History:                                                         --
--      11/14/2024  Edward Speer  Initial revision                            --
--                                                                            --
--------------------------------------------------------------------------------

--
-- Imports
--

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

entity Cntr8ClockDiv_tb is
end Cntr8ClockDiv_tb;

--
-- Cntr8ClockDiv_tb TB_ARCHITECTURE
--

architecture TB_ARCHITECTURE of Cntr8ClockDiv_tb is

    --
    -- Component declaration of the device under test
    --

    component Cntr8ClockDiv
        port (
            CLK      : in     std_logic;                    -- 32 MHz sys clk
            RESET    : in     std_logic;                    -- Async reset
            CLK_8kHz : buffer std_logic;                    -- 8 kHz clk out
            Cntr8    : out    std_logic_vector(7 downto 0)  -- 8 bit cnt out
        );
    end component;

    --
    -- Stimulus signals
    --

    signal CLK   : std_logic := '0'; -- 32 MHz system clock
```

```
    signal RESET : std_logic := '0'; -- Asynchronous reset

    --
    -- Observed signals
    --

    signal CLK_8kHz : std_logic := '0';
    signal Cntr8    : std_logic_vector(7 downto 0) := "00000000";

    --
    -- Flag used to end simulation
    --

    signal END_SIM : BOOLEAN := FALSE;

begin

    --
    -- Device under test port map
    --

    DUT : Cntr8ClockDiv
        port map (
            CLK      => CLK,
            RESET    => RESET,
            CLK_8kHz => CLK_8kHz,
            Cntr8    => Cntr8
        );

    process

    variable c80 : std_logic := '0'; -- Store the 8 bit counter output

    begin        -- TEST_CLK8KHZ
        RESET <= '1';
        wait for 64000 ns;
        RESET <= '0';
        -- Test that the 8 kHz clock signal is generated correctly
        for i in 100 downto 0 loop
            c80 := CLK_8kHz;

            -- Wait until 2048 32 MHz clocks have elapsed
            wait for 64000 ns;
            -- After allowing one 8 kHz clock cycle to elapse, check that the
            -- 8 kHz clock signal has toggled
            if i < 99 then
                assert (std_match(c80, not CLK_8kHz))
                    report "8 kHz clock signal did not toggle"
                    severity ERROR;
            end if;
        end loop;
```

```
        -- End the simulation
        END_SIM <= TRUE;
        wait;

    end process; -- TEST_CLK8KHZ

    process      -- TEST_CNT8

        variable cnt : unsigned(7 downto 0) := (others => '0'); -- 8 bit counter

    begin
        wait for 1 sec;
        if END_SIM = FALSE then
            cnt := unsigned(Cntr8);
            wait for 31250 ps;
            assert (std_match(Cntr8, std_logic_vector(cnt + 1)))
                report "8 bit counter output does not match expected value"
                severity ERROR;
        else
            wait;
        end if;

    end process; -- TEST_CNT8

    CLOCK_CLK : process
    -- This process generates a 32 MHz x 50% duty cycle clock, and stops the
    -- clock when the end of simulation is reached.
    begin
        -- Generates 32 MHz clock
        if END_SIM = FALSE then
            CLK <= '0';
            wait for 15625 ps;
        else
            wait;
        end if;

        if END_SIM = FALSE then
            CLK <= '1';
            wait for 15625 ps;
        else
            wait;
        end if;

    end process; -- CLOCK_CLK

end TB_ARCHITECTURE;

-- Configure use of Cntr8ClockDiv behavioral architecture in test
configuration TESTBENCH_FOR_CNTR8CLOCKDIV_BEHAVIORAL of Cntr8ClockDiv_tb is
    for TB_ARCHITECTURE
```

```
        for DUT : Cntr8ClockDiv
            use entity work.Cntr8ClockDiv(behavioral);
        end for;
    end for;
end TESTBENCH_FOR_CNTR8CLOCKDIV_BEHAVIORAL;

-- Configure use of Cntr8ClockDiv implementation architecture in test
configuration TESTBENCH_FOR_CNTR8CLOCKDIV_IMPLEMENTATION of Cntr8ClockDiv_tb is
    for TB_ARCHITECTURE
        for DUT : Cntr8ClockDiv
            use entity work.Cntr8ClockDiv(implementation);
        end for;
    end for;
end TESTBENCH_FOR_CNTR8CLOCKDIV_IMPLEMENTATION;
```

```
--------------------------------------------------------------------------------
--                                                                            --
--  AddrUnit Logic Design                                                     --
--                                                                            --
--  This is an entity declaration and architectures for an address unit that  --
--  serves as the first stage of the PWM Audio Player system. The unit takes  --
--  in the 8 kHz sample clock, and user button inputs. The unit outputs the   --
--  address of the EPROM to be read from to obtain the requested audio        --
--  sample, as well as an audio output enable signal.                         --
--                                                                            --
--  Inputs:                                                                   --
--           CLK_8kHz          - 8 kHz sample clock                           --
--           Btn[3..0]         - User button inputs                           --
--           RESET             - Asynchronous, active high reset              --
--                                                                            --
--  Outputs:                                                                  --
--           AudioAddr[18..0]  - Address output                               --
--           Enable            - Audio output enable signal                   --
--                                                                            --
--------------------------------------------------------------------------------

--
-- Imports
--

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

--
-- AddrUnit entity declaration
--

entity AddrUnit is
    port (
        CLK_8kHz     : in     std_logic;                     -- 8 kHz sample clk
        RESET        : in     std_logic;                     -- Asynch reset
        Btn          : in     std_logic_vector(3 downto 0); -- User btn inputs
        AudioAddr    : buffer std_logic_vector(18 downto 0);-- Address output
        Enable       : buffer std_logic                     -- Audio out enable
    );
end AddrUnit;

--
-- AddrUnit behavioral architecture
--

architecture behavioral of AddrUnit is

    -- Top end of address range
    signal AddrStop : std_logic_vector(18 downto 0);
```

```
    -- Filtered user button presses (Mutually exclusive)
    signal FilBtn   : std_logic_vector(3 downto 0);

    -- Address range indicated by user button presses
    signal BtnAddr0 : std_logic_vector(18 downto 0);
    signal BtnAddr1 : std_logic_vector(18 downto 0);

    -- Button => addresses mapping
    constant B3_START : std_logic_vector(18 downto 0) := "1000000000000000000";
    constant B2_START : std_logic_vector(18 downto 0) := "1001000000000000000";
    constant B1_START : std_logic_vector(18 downto 0) := "1011000000000000000";
    constant B0_START : std_logic_vector(18 downto 0) := "1111000000000000000";
    constant B0_END   : std_logic_vector(18 downto 0) := "1111111111111111111";

begin

    -- Concurrent logic assignments

    -- Filter button presses to be mutually exclusive
    FilBtn(3) <= Btn(3);
    FilBtn(2) <= Btn(2) and not Btn(3);
    FilBtn(1) <= Btn(1) and not Btn(2) and not Btn(3);
    FilBtn(0) <= Btn(0) and not Btn(1) and not Btn(2) and not Btn(3);

    -- Set start & stop addresses for each button press
    BtnAddr0 <= B3_START when FilBtn = "1000" else
                B2_START when FilBtn = "0100" else
                B1_START when FilBtn = "0010" else
                B0_START;

    BtnAddr1 <= B2_START when FilBtn = "1000" else
                B1_START when FilBtn = "0100" else
                B0_START when FilBtn = "0010" else
                B0_END;

    -- Output is enabled when we haven't reached a stop address
    enable <= '1' when std_match(AudioAddr, AddrStop) = FALSE else '0';

    process(CLK_8kHz, RESET)
    begin
        if RESET = '1' then
            AudioAddr <= "00000000000000000000";
            AddrStop  <= "00000000000000000000";
        elsif rising_edge(CLK_8kHz) then
            if std_match(enable, '0') = TRUE then
                -- If we are at the end of the track, wait until a new button
                -- press occurs to load a new address range
                if std_match(FilBtn, "0000") = FALSE then
                    AudioAddr <= BtnAddr0;
                    AddrStop  <= BtnAddr1;
```

```
                end if;
            else
                -- Otherwise simply increment the address
                AudioAddr <= std_logic_vector(unsigned(AudioAddr) + 1);
            end if;
        end if;
    end process;

end behavioral;

--
-- AddrUnit implementation architecture
--

architecture implementation of AddrUnit is

    -- Top end of address range
    signal AddrStop : std_logic_vector(18 downto 0);

    -- Filtered user button inputs (mutually exclusive)
    signal FilBtn : std_logic_vector(2 downto 0);

    -- Address range indicated by user button presses
    signal BtnAddr0       : std_logic_vector(18 downto 0);
    signal BtnAddr1       : std_logic_vector(18 downto 0);
    signal BtnAddr1_other : std_logic_vector(13 downto 0);

begin

    -- Concurrent logic assignment

    -- Filter button presses to be mutually exclusive
    FilBtn(2) <= Btn(2) and not Btn(3);
    FilBtn(1) <= Btn(1) and not Btn(2) and not Btn(3);
    FilBtn(0) <= Btn(0) and not Btn(1) and not Btn(2) and not Btn(3);

    -- Compute invididual terms of Button addresses based on simple equations
    BtnAddr0     <= '1' & FilBtn(0) & (FilBtn(0) or FilBtn(1)) & (FilBtn(0) or
                    FilBtn(1) or FilBtn(2)) & FilBtn(0) & "00000000000000";

    BtnAddr1_other <= (others => FilBtn(0));
    BtnAddr1     <= '1' & (FilBtn(1) or FilBtn(0)) & (FilBtn(2) or FilBtn(1) or
                    FilBtn(0)) & '1' & (FilBtn(1) or FilBtn(0))
                    & BtnAddr1_other;

    -- Output is enabled when we haven't reached a stop address
    enable <= '1' when std_match(AudioAddr, AddrStop) = FALSE else '0';

    process(CLK_8kHz, RESET)
    begin
        if RESET = '1' then
```

```
            AudioAddr <= "00000000000000000000";
            AddrStop  <= "00000000000000000000";
        elsif rising_edge(CLK_8kHz) then
            if std_match(enable, '0') = TRUE then
                -- If we are at the end of the track, wait until a new button
                -- press occurs to load a new address range
                if std_match(Btn(3) & FilBtn, "0000") = FALSE then
                    AudioAddr <= BtnAddr0;
                    AddrStop  <= BtnAddr1;
                end if;
            else
                -- Otherwise simply increment the address
                AudioAddr <= std_logic_vector(unsigned(AudioAddr) + 1);
            end if;
        end if;
    end process;


end implementation;
```

```
--------------------------------------------------------------------------------
--                                                                            --
--  AddrUnit Test Bench                                                       --
--                                                                            --
--  This is a test bench for the AddrUnit block of the PWM Audio Player       --
--  system. The test bench thoroughly tests the entity by exercising it and   --
--  checking the output address and enable signals. The test bench entity is  --
--  called AddrUnit_tb and it is currently defined to test the behavioral     --
--  architecture of the AddrUnit unit.                                        --
--                                                                            --
--  Revision History:                                                         --
--      11/14/2024  Edward Speer  Initial revision                            --
--                                                                            --
--------------------------------------------------------------------------------

--
-- Imports
--

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

entity AddrUnit_tb is
end AddrUnit_tb;

--
-- AddrUnit_tb TB_ARCHITECTURE
--

architecture TB_ARCHITECTURE of AddrUnit_tb is

    --
    -- Component declaration of the device under test
    --

    component AddrUnit
        port (
            CLK_8kHz  : in     std_logic;                    -- Sample rate clk
            RESET     : in     std_logic;                    -- Asynch reset
            enable    : buffer std_logic;                    -- enable audio out
            Btn       : in     std_logic_vector(3 downto 0); -- Button input
            AudioAddr : buffer std_logic_vector(18 downto 0) -- address output
        );
    end component;

    --
    -- Stimulus signals
    --

    signal CLK_8kHz  : std_logic := '0';
```

```
    signal RESET     : std_logic := '0';
    signal Btn       : std_logic_vector(3 downto 0) := "0000";

    --
    -- Observed signals
    --

    signal AudioAddr : std_logic_vector(18 downto 0) := "0000000000000000000";
    signal enable    : std_logic;

    --
    -- Flag used to end simulation
    --

    signal END_SIM : BOOLEAN := FALSE;

begin

    --
    -- Device under test port map
    --

    DUT : AddrUnit
    port map (
        CLK_8kHz  => CLK_8kHz,
        RESET     => RESET,
        enable    => enable,
        Btn       => Btn,
        AudioAddr => AudioAddr
    );

    --
    -- Stimulus process
    --

    process
    begin

        -- Initialize the system with a reset
        RESET <= '1';
        wait for 126000 ns;
        RESET <= '0';

        -- Test a button press
        Btn <= "1000";
        wait for 126000 ns;
        Btn <= "0000";
        wait for 10 sec;
        assert (std_match(AudioAddr, "1001000000000000000") = TRUE)
            report "INCORRECT ADDRESS"
            severity ERROR;
```

```
    -- Test the system with a button press and then a resettling period.
    Btn <= "0100";
    wait for 126000 ns;
    assert (enable = '1')
        report "MISSED BUTTON PRESS"
        severity ERROR;
    Btn <= "0000";
    wait for 10 sec;
    assert (enable = '0')
        report "BUTTON PRESS NOT CLEARED"
        severity ERROR;
    assert (std_match(AudioAddr, "10110000000000000000") = TRUE)
        report "INCORRECT ADDRESS"
        severity ERROR;

    -- Try a new button press with multiple buttons
    Btn <= "0011";
    wait for 126000 ns;
    assert (enable = '1')
        report "MISSED BUTTON PRESS"
        severity ERROR;
    Btn <= "0000";
    wait for 20 sec;
    assert (enable = '0')
        report "BUTTON PRESS NOT CLEARED"
        severity ERROR;
    assert (std_match(AudioAddr, "11111000000000000000") = TRUE)
        report "INCORRECT ADDRESS"
        severity ERROR;

    -- Check address on last button
    Btn <= "0001";
    wait for 126000 ns;
    assert (enable = '1')
        report "MISSED BUTTON PRESS"
        severity ERROR;
    Btn <= "0000";
    wait for 10 sec;
    assert (enable = '0')
        report "BUTTON PRESS NOT CLEARED"
        severity ERROR;
    assert (std_match(AudioAddr, "11111111111111111111") = TRUE)
        report "INCORRECT ADDRESS"
        severity ERROR;

    -- End the simulation
    END_SIM <= TRUE;
    wait;

end process;
```

```
    CLOCK_CLK : process
        -- This process generates an 8kHz x 50% duty cycle clock, and stops the
        -- clock when the end of simulation is reached.
        begin
            -- Generates 32 MHz clock
            if END_SIM = FALSE then
                CLK_8kHz <= '0';
                wait for 62500 ns;
            else
                wait;
            end if;

            if END_SIM = FALSE then
                CLK_8kHz <= '1';
                wait for 62500 ns;
            else
                wait;
            end if;

    end process; -- CLOCK_CLK

end TB_ARCHITECTURE;

-- Configure use of AddrUnit behavioral architecture in test
configuration TESTBENCH_FOR_ADDRUNIT_BEHAVIORAL of AddrUnit_tb is
    for TB_ARCHITECTURE
        for DUT : AddrUnit
            use entity work.AddrUnit(behavioral);
        end for;
    end for;
end TESTBENCH_FOR_ADDRUNIT_BEHAVIORAL;

-- Configure use of AddrUnit implementation architecture in test
configuration TESTBENCH_FOR_ADDRUNIT_IMPLEMENTATION of AddrUnit_tb is
    for TB_ARCHITECTURE
        for DUT : AddrUnit
            use entity work.AddrUnit(implementation);
        end for;
    end for;
end TESTBENCH_FOR_ADDRUNIT_IMPLEMENTATION;
```

```
--------------------------------------------------------------------------------
--                                                                            --
--  PwmDriver Logic Design                                                    --
--                                                                            --
--  This is an entity declaration and architectures for a PWM driver module   --
--  that serves as the final stage of the PWM Audio Player system. The unit   --
--  takes in an 8 bit audio sample, an 8 kHz sample clock, and the output of  --
--  the system 8 bit counter. The unit outputs a PWM signal of the correct    --
--  duty cycle to be used as the audio output. The module includes an enable  --
--  signal that may be used to enable or disable the audio output.            --
--                                                                            --
--  Inputs:                                                                   --
--           CLK_8kHz        - 8 kHz sample clock                             --
--           RESET           - Asynch, active hi reset signal                 --
--           enable          - enable audio output                            --
--           AudioData[7..0] - 8 bit audio sample from EPROM                  --
--           Cntr8[7..0]     - 8 bit counter output                           --
--                                                                            --
--  Outputs:                                                                  --
--           AudioPWMOut - PWM output                                         --
--                                                                            --
--  Revision History:                                                         --
--      11/14/2024  Edward Speer  Initial Revision                            --
--                                                                            --
--------------------------------------------------------------------------------

--
-- Imports
--

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

--
-- PwmDriver entity declaration
--

entity PwmDriver is
    port (
        CLK_8kHz    : in  std_logic;                    -- 8 kHz sample clock
        RESET       : in  std_logic;                    -- Asynch reset
        enable      : in  std_logic;                    -- enable audio output
        AudioData   : in  std_logic_vector(7 downto 0); -- 8 bit audio sample
        Cntr8       : in  std_logic_vector(7 downto 0); -- 8 bit counter output
        AudioPWMOut : out std_logic                     -- PWM output
    );
end PwmDriver;

--
-- PwmDriver behavioral architecture
```

```
--

architecture behavioral of PWMdriver is

    -- Audio data latch
    signal AudioCache : std_logic_vector(7 downto 0);

begin

    -- Concurrent logic assignments

    -- PWM output is computed 16 times per sample period. In each of the 16
    -- periods, the PWM output is high if the counter value is less than the
    -- audio data value. The PWM output is low otherwise.
    AudioPWMOut <= '1' when std_match(enable, '1') = TRUE and
                            unsigned(Cntr8) < unsigned(AudioCache) else '0';

    -- Latch in audio data on the sampling clock
    process(CLK_8kHz, RESET)
    begin
        if std_match(RESET, '1') = TRUE then
            AudioCache <= "00000000";
        elsif rising_edge(CLK_8kHz) then
            AudioCache <= AudioData;
        end if;
    end process;

end behavioral;
```

```
--------------------------------------------------------------------------------
--                                                                            --
--  PwmDriver Test Bench                                                      --
--                                                                            --
--  This is a test bench for the PwmDriver block of the PWM Audio Player      --
--  system. The test bench thoroughly tests the entity by exercising it and   --
--  checking the output PWM produced. The test bench entity is called         --
--  PwmDriver_tb and it is currently defined to test the behavioral           --
--  architecture of the PwmDriver unit.                                       --
--                                                                            --
--  Revision History:                                                         --
--      11/14/2024  Edward Speer  Initial revision                            --
--                                                                            --
--------------------------------------------------------------------------------

--
-- Imports
--

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

entity PwmDriver_tb is
end PwmDriver_tb;

--
-- PwmDriver_tb TB_ARCHITECTURE
--

architecture TB_ARCHITECTURE of PwmDriver_tb is

    --
    -- Component declaration of the device under test
    --

    component PwmDriver
        port (
            CLK_8kHz    : in  std_logic;                    -- 8 kHz sample clk
            RESET       : in  std_logic;                    -- Reset PwmDriver
            enable      : in  std_logic;                    -- enable pwm output
            AudioData   : in  std_logic_vector(7 downto 0); -- 8 bit sample
            Cntr8       : in  std_logic_vector(7 downto 0); -- 8 bit cntr input
            AudioPWMOut : out std_logic                     -- PWM output
        );
    end component;

    --
    -- Stimulus signals
    --
```

```
    signal CLK_8kHz  : std_logic := '0'; -- 8 kHz sample clock
    signal RESET     : std_logic := '0'; -- Reset signal
    signal enable    : std_logic := '1'; -- enable audio output
    signal AudioData : std_logic_vector(7 downto 0) := "00000000";
    signal Cntr8     : std_logic_vector(7 downto 0) := "00000000";

    --
    -- Observed signals
    --

    signal AudioPWMOut : std_logic := '0';

    --
    -- Flag used to end simulation
    --

    signal END_SIM : BOOLEAN := FALSE;

begin

    -- Device under test port map
    DUT : PwmDriver port map (
        CLK_8kHz    => CLK_8kHz,
        RESET       => RESET,
        enable      => enable,
        AudioData   => AudioData,
        Cntr8       => Cntr8,
        AudioPWMOut => AudioPWMOut
    );

    -- Stimulus process
    process      -- STIMULUS_PROCESS
    begin

        -- First, reset the unit
        RESET <= '1';

        -- Wait for a clock cycle to allow the reset to take effect
        wait for 126000 ns;
        RESET <= '0';

        -- Simply verify the PWM output is correct based on the AudioData and
        -- Cntr8 inputs
        AudioData <= "00000000"; -- 0
        Cntr8     <= "00000001"; -- 0
        wait for 126000 ns;
        assert AudioPWMOut = '0'
            report "PWM HIGH WHEN CNTR > DATA"
            severity ERROR;

        AudioData <= "00000001"; -- 1
```

```
    Cntr8     <= "00000000"; -- 0
    wait for 126000 ns;
    assert AudioPWMOut = '1'
        report "PWM LOW WHEN CNTR < DATA"
        severity ERROR;

    -- Ensure AudioData latched only on the 8 kHz clock
    AudioData <= "10000000"; -- 0
    Cntr8     <= "11000000"; -- 0
    assert AudioPWMOut = '1'
        report "AudioData latched on wrong edge"
        severity ERROR;
    wait for 126000 ns;
    assert AudioPWMOut = '0'
        report "PWM HIGH WHEN CNTR > DATA"
        severity ERROR;

    -- Ensure that setting the enable line low disables the PWM output
    enable <= '0';
    AudioData <= "10000000"; -- 0
    Cntr8     <= "00000001"; -- 0
    wait for 126000 ns;
    assert AudioPWMOut = '0'
        report "PWM HIGH WHEN ENABLE LOW"
        severity ERROR;
    wait for 126000 ns;
    assert AudioPWMOut = '0'
        report "PWM HIGH WHEN ENABLE LOW"
        severity ERROR;

    enable <= '1';
    wait for 126000 ns;
    assert AudioPWMOut = '1'
        report "PWM LOW WHEN CNTR < DATA"
        severity ERROR;

    -- End the simulation
    END_SIM <= TRUE;
    wait;

end process; -- STIMULUS_PROCESS

CLOCK_CLK : process
    -- This process generates an 8 kHz x 50% duty cycle clock, and stops the
    -- clock when the end of simulation is reached.
    begin
        -- Generates 8 kHz clock
        if END_SIM = FALSE then
            CLK_8kHz <= '0';
            wait for 62500 ns;
        else
```

```
                wait;
            end if;

            if END_SIM = FALSE then
                CLK_8kHz <= '1';
                wait for 62500 ns;
            else
                wait;
            end if;

        end process; -- CLOCK_CLK

end TB_ARCHITECTURE;

-- Configure use of PwmDriver behavioral architecture in test
configuration TESTBENCH_FOR_PWMDRIVER_BEHAVIORAL of PwmDriver_tb is
    for TB_ARCHITECTURE
        for DUT : PwmDriver
            use entity work.PwmDriver(behavioral);
        end for;
    end for;
end TESTBENCH_FOR_PWMDRIVER_BEHAVIORAL;
```