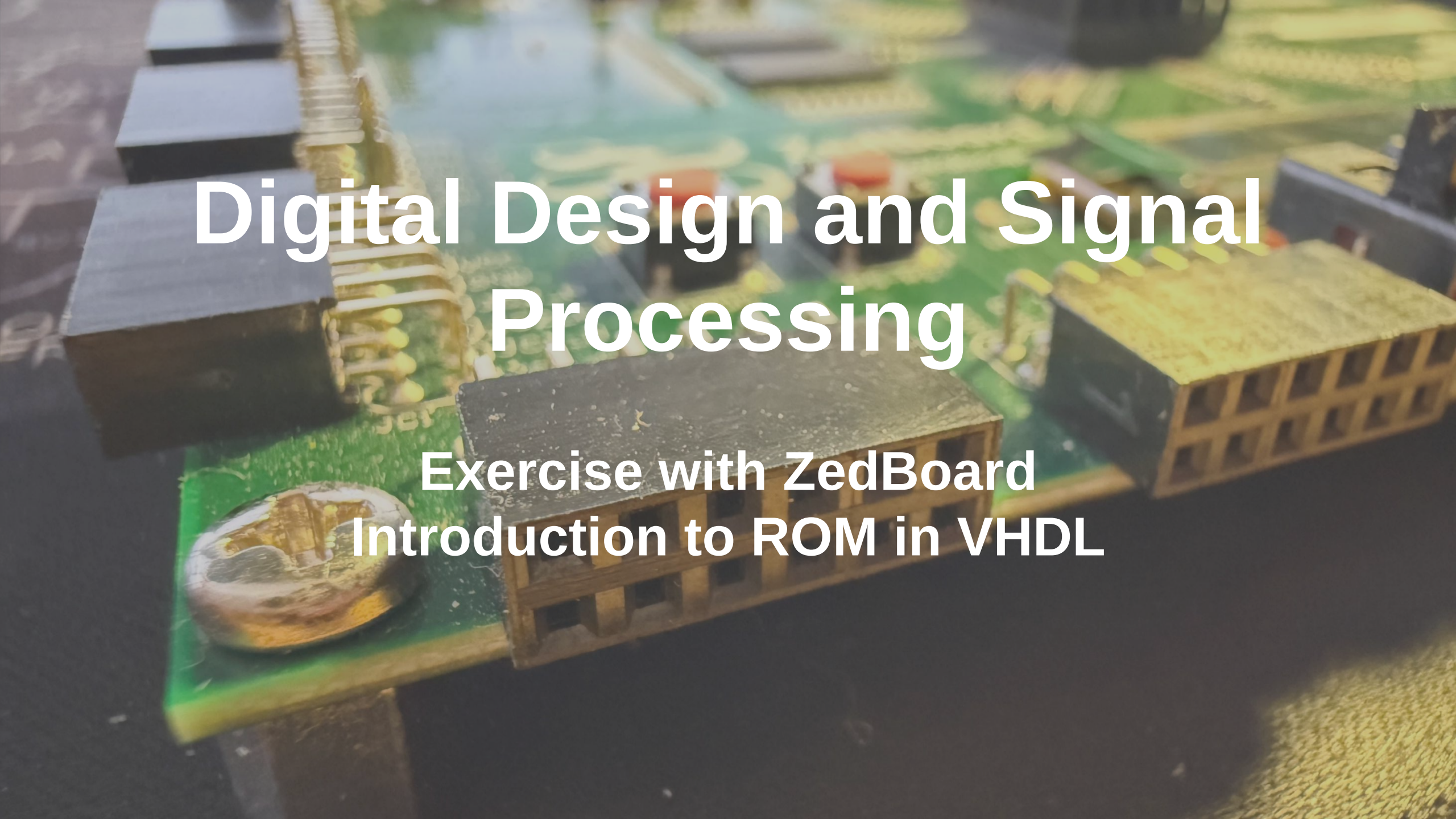

Digital Design and Signal Processing

Exercise with ZedBoard
Introduction to ROM in VHDL

Task for Zedboard

Implement different logic gates on the Zed board (one input and different Leds for output)

Task for Zedboard

Implement 1bit full adder on the Zed board

Task for Zedboard

Implement 1bit subtractor on the Zed board

Assignment 3

Goal 1: Create a ROM with 32 samples (constant data) – 8 bit resolution – representing one period of a sine wave. Document via relevant simulations that it functions as expected.

Tips: 12 bit clk counter, use last 4 bit as increment.

Goal 2: Create a PWM generator, with 8 bit resolution, - that is: using an 8-bit counter – and document the functionality with relevant simulations.

Goal 3: Calculate the maximum frequency of the sinewave signal, based on the clock frequency of the ZEDBOARD.

Introduction to ROM in VHDL

What is ROM?

- Read-Only Memory (ROM) is a non-volatile memory used for storing fixed data.
- In VHDL, ROM can be implemented using arrays.

Why Use ROM in VHDL?

- Stores lookup tables (LUTs) and coefficients.
- Used in digital circuits like microprocessors and signal processing.
- Saves FPGA resources compared to flip-flop-based storage.

Defining ROM in VHDL

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

entity ROM_Example is
Port ( address : in  STD_LOGIC_VECTOR (3 downto 0);
           data_out : out STD_LOGIC_VECTOR (7 downto 0));
end ROM_Example;

architecture Behavioral of ROM_Example is

type ROM_TYPE is array (0 to 15) of STD_LOGIC_VECTOR(7 downto 0);

constant ROM : ROM_TYPE := (
x"00", x"11", x"22", x"33",
x"44", x"55", x"66", x"77",
x"88", x"99", x"AA", x"BB",
x"CC", x"DD", x"EE", x"FF"
);
```

```
begin

process (address)

begin
data_out <= ROM(to_integer(unsigned(address)));

end process;

end Behavioral;
```

ROM_TYPE: Defines a 16-location ROM, each storing an 8-bit value.

ROM Constant: Stores predefined hexadecimal values.

Testbench for ROM

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

entity ROM_Testbench is
end ROM_Testbench;

architecture Behavioral of ROM_Testbench is
component ROM_Example
Port ( address : in  STD_LOGIC_VECTOR (3 downto 0);
data_out : out STD_LOGIC_VECTOR (7 downto 0));
end component;
```

```
signal address : STD_LOGIC_VECTOR (3 downto 0);
signal data_out : STD_LOGIC_VECTOR (7 downto 0);
begin    uut: ROM_Example port map (address => address,
data_out => data_out);
process
begin
for i in 0 to 15 loop
address <= std_logic_vector(to_unsigned(i, 4));
wait for 10 ns;
end loop;
wait;
end process;
```

Task

Make a ROM array which is triggered using the clock

One period of a sine wave

```
library IEEE;use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;

entity Sine_Wave_32 is
port (CLK : in std_logic;                                    --- CLK: Clock input, used for timing.
      Reset : in std_logic;                                  --- Reset: Active-high reset signal.
      Div_value : in std_logic_vector(7 downto 0);           --- Div_value: Determines the frequency of the sine wave.
      DATA : out std_logic_vector(7 downto 0));              --- DATA: 8-bit output representing sine wave samples.
end Sine_Wave_32;


architecture Behavioral of Sine_Wave_32 is
type rom_type is array (0 to 31) of std_logic_vector (7 downto 0);    --- rom_type: Defines an array (0 to 31) where each element is an 8bit
    signal ROM : rom_type:= (ROM Constant: Stores predefined hexadecimal values);
```

One period of a sine wave

```
begin
   process (CLK, Reset)
   variable clk_count : std_logic_vector(11 downto 0);
   variable ADDR : std_logic_vector(4 downto 0);
   Variable Div_Value_var : std_logic_vector(11 downto 0);
   Begin
   Div_Value_var(11 downto 4) := Div_Value;

   Div_Value_var(3 downto 0) := "0000";
   if (Reset = '1') then
        Clk_count := (others => '0');
        ADDR := (others => '0');
   else
        if (Falling_edge(Clk)) then
           if (Clk_count = Div_value_var) then
                DATA <= ROM(conv_integer(ADDR));
                ADDR := ADDR +1;
                Clk_count := (others => '0');
                else clk_count := clk_count +1;
            end if;
        end if;
    end if;
    end process;
end Behavioral;
```

--- 12-bit counter to track clock cycles.

--- 5-bit variable (since 2^5 = 32) to cycle through the ROM table.

--- Stores a 12-bit extended version of Div_value (which is 8-bit).

--- Extends Div_value from **8 bits to 12 bits** by appending four **zeroes** at the least significant bits.

--- **If Reset is high (1)**, reset clk_count and ADDR to 0

--- if (Falling_edge(Clk)) then

--- If clk_count **reaches** Div_value_var, the next sine wave value is output.

--- **DATA <= ROM(conv_integer(ADDR))** retrieves the current sample from ROM.

--- ADDR := ADDR + 1 moves to the **next sine wave sample**.

--- clk_count is reset and starts counting again.

This mechanism **controls the output frequency** by adjusting Div_value. A **higher** Div_value means **slower transitions**, resulting in a **lower frequency sine wave**.