# VLSI LAB PROJECT REPORT


**SUBJECT:** VLSI DESIGN LAB

**CLASS:** ECE-A

**COURSE CODE:** 19ECE383

**PROJECT TITLE:** DIGITAL CLOCK

**GROUP NUMBER :** 03


| S.NO. | NAME | ROLL NUMBER |
|---|---|---|
| 1. | DEEPIKA | CB.EN.U4ECE21009 |
| 2. | SHAAHID | CB.EN.U4ECE21011 |
| 3. | SANTHOSH | CB.EN.U4ECE21012 |

TOOLS REQUIRED: FPGA nexys-3 board

SOFTWARE USED: ModelSim

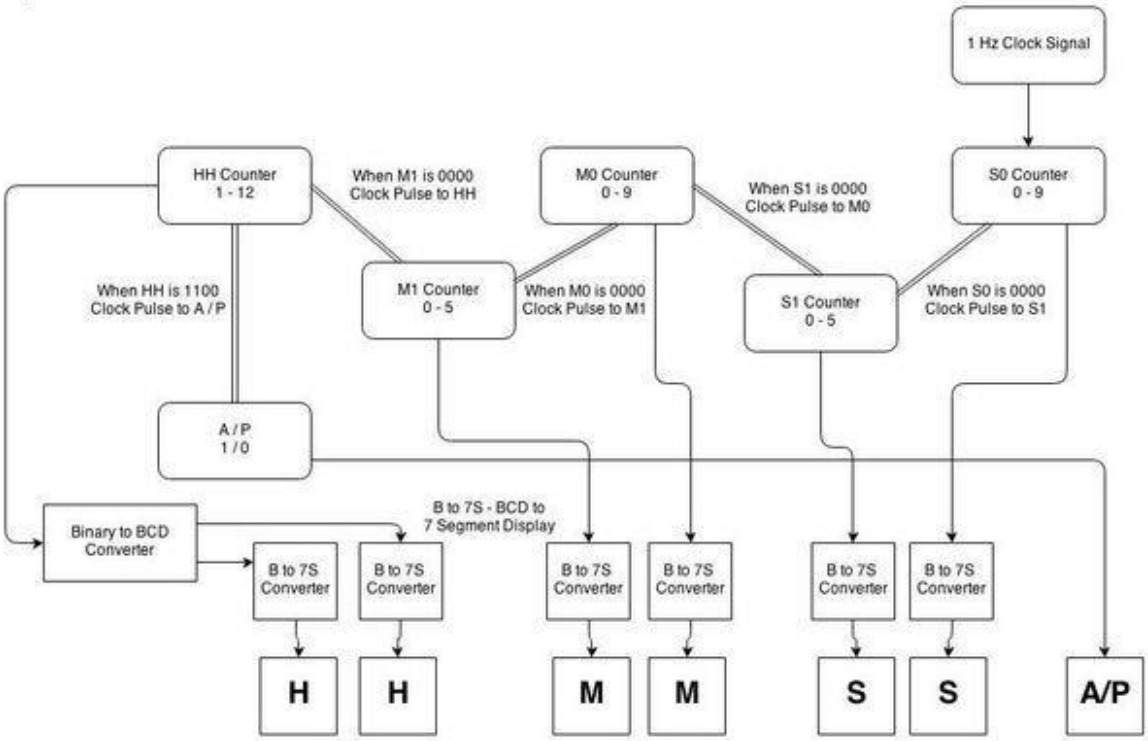
# Abstract:

This project focuses on making a digital clock using VHDL (a special computer language for making hardware). The main aim is to create a clock that shows the time accurately, including hours, minutes, and seconds. We used VHDL to describe how the clock works at a basic level. The circuit of the digital clock is divided into three modules, namely the frequency division module, counting module, and decoding display module. The timing process is through the 7 segment LED display, displaying "Hours", "Minutes", and "Seconds" which are displayed in two digits. The frequency division module divides a 50 MHz input signal to obtain a 1 Hz clock signal, and the counting time module can count and adjust the time of the clock, minutes, and seconds, and then display it on the FPGA development board through the decoding display module. After clock simulation, the system realizes the function of the digital clock and meets the design requirements.

# Objective:

1. **Design and Implement a Digital Clock Using VHDL:** The primary objective is to design a digital clock using VHDL that can display hours, minutes, and seconds in two digits on a 7-segment LED display.
2. **Develop Three Key Modules:** Develop the three key modules that make up the digital clock: the frequency division module, the counting module, and the decoding display module.
3. **Frequency Division Module:** Design a frequency division module that can divide a 50 MHz input signal to obtain a 1 Hz clock signal.
4. **Counting Time Module:** Develop a counting time module that can count and adjust the time of the clock, minutes, and seconds.
5. **Decoding Display Module:** Implement a decoding display module that can take the output from the counting time module and display it on the FPGA development board.
6. **Simulation and Testing:** Simulate the digital clock using appropriate tools and methodologies to ensure it functions as expected.
7. **Implementation on FPGA:** Implement the digital clock design on an FPGA development board and validate its operation.

## Block Diagram:




## PROGRESS:

| S.no | Module name | Module description | Status |
|---|---|---|---|
| 1 | Frequency module | To generate clock pulse to the counter | completed |
| 2 | Counting module | To count hours, minutes and seconds | completed |
| 3 | Binary to BCD | Binary to BCD conversion | Ongoing |

# PROGRAM:

```
--Library declaration
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

--Binary to BCD Converter
entity bin2bcd is
port(       binary_in : in unsigned(5 downto 0);
            bcd_out : out unsigned(7 downto 0)     --8 bits(4 bits each for 2 digits)
            );
end bin2bcd;

architecture Behavioral of bin2bcd is

--actual value=0.0001100110011001101, so divide by 2^19.
constant one_by_ten : unsigned(15 downto 0) := "1100110011001101";
signal result_div_by_ten : unsigned(21 downto 0);
signal msb_digit : unsigned(3 downto 0);

begin

--divide the input by 10. In VHDL, division by a constant is
--easily done by multiplication by its inverse.
result_div_by_ten <=  binary_in*one_by_ten;
--get the decimal part of the result. Bits 18:0 are fractional part
msb_digit <= '0' & result_div_by_ten(21 downto 19);
bcd_out(7 downto 4) <= msb_digit;      --assign it to MSB part of output
--subtract the product, 10*msb_digit, from the input binary number to get LSB digit.
bcd_out(3 downto 0) <= to_unsigned(to_integer(binary_in) - to_integer(msb_digit)*10, 4);

end Behavioral;
```

## Frequency Module: digital_clock_freq_module.vhd

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

--Top module which instantiates and connect together the 3 components:
--Digital clock, Binary to BCD converter, BCD to 7 segment code converter
entity digital_clock_topmodule is
 --frequency of the clock passed as a generic parameter.
generic(
                    CLOCK_FREQ : integer := 50000000
                    );
port(
                    Clock : in std_logic;  --system clock
                    reset : in std_logic;  --resets the time
```

```
                inc_secs : in std_logic;  --set a pulse here to increment the seconds by 1.
                inc_mins : in std_logic;  --set a pulse here to increment the minutes by 1.
                inc_hrs : in std_logic;  --set a pulse here to increment the hours by 1.
                secs_7seg1 :out unsigned(6 downto 0);  --seconds LSB digit
                secs_7seg10 :out unsigned(6 downto 0);  --seconds MSB digit
                mins_7seg1 :out unsigned(6 downto 0);  --minutes LSB digit
                mins_7seg10 :out unsigned(6 downto 0);  --minutes MSB digit
                hrs_7seg1 :out unsigned(6 downto 0);  --hours LSB digit
                hrs_7seg10 :out unsigned(6 downto 0)  --hours MSB digit
                );
end digital_clock_topmodule;

architecture Behavioral of digital_clock_topmodule is

        --Digital clock component
        COMPONENT digital_clock
        GENERIC( CLOCK_FREQ : integer := 50000000);
        PORT(
                Clock : IN  std_logic;
                reset : IN  std_logic;
                inc_secs : IN  std_logic;
                inc_mins : IN  std_logic;
                inc_hrs : IN  std_logic;
                seconds : OUT  unsigned(5 downto 0);
                minutes : OUT  unsigned(5 downto 0);
                hours : OUT  unsigned(4 downto 0)
                );
        END COMPONENT;

        --Binary to BCD converter component
        COMPONENT bin2bcd is
        PORT(
                binary_in : in unsigned(5 downto 0);
                bcd_out : out unsigned(7 downto 0)
                );
        END COMPONENT;

        --Function to convert a BCD digit into a 7 segment code
        --Source: https://vhdlguru.blogspot.com/2010/03/vhdl-code-for-bcd-to-7-segment-
display.html
        --The function is created by converting the code from the above link.
        function bcd2seg7(bcd_in : unsigned(3 downto 0)) return unsigned is
                variable segment7 : unsigned(6 downto 0);
        begin
                case bcd_in is
                        when "0000"=> segment7 :="0000001";  -- '0'
                        when "0001"=> segment7 :="1001111";  -- '1'
                        when "0010"=> segment7 :="0010010";  -- '2'
                        when "0011"=> segment7 :="0000110";  -- '3'
                        when "0100"=> segment7 :="1001100";  -- '4'
                        when "0101"=> segment7 :="0100100";  -- '5'
                        when "0110"=> segment7 :="0100000";  -- '6'
```

```
                        when "0111"=> segment7 :="0001111";  -- '7'
                        when "1000"=> segment7 :="0000000";  -- '8'
                        when "1001"=> segment7 :="0000100";  -- '9'
                         --nothing is displayed when a number more than 9 is given as
input.
                        when others=> segment7 :="1111111";
                end case;
                return segment7;
        end bcd2seg7;


        --Declare internal signals
        signal seconds : unsigned(5 downto 0);
        signal minutes : unsigned(5 downto 0);
        signal hours : unsigned(4 downto 0);
        signal bcd_secs,bcd_mins,bcd_hrs : unsigned(7 downto 0);
        signal hours_extended : unsigned(5 downto 0);


begin

-- Instantiate the Digital Clock component
        uut: digital_clock
                GENERIC MAP(CLOCK_FREQ => CLOCK_FREQ)
                PORT MAP (
                        Clock => Clock,
                        reset => reset,
                        inc_secs => inc_secs,
                        inc_mins => inc_mins,
                        inc_hrs => inc_hrs,
                        seconds => seconds,
                        minutes => minutes,
                        hours => hours
                        );

        --convert binary to BCD for seconds
        bin2bcd_secs : bin2bcd
                port map(
                        binary_in => seconds,
                        bcd_out => bcd_secs
                        );

        --convert binary to BCD for minutes
        bin2bcd_mins : bin2bcd
                port map(
                        binary_in => minutes,
                        bcd_out => bcd_mins
                        );

        hours_extended <= '0' & hours;          --just make it the same size as seconds and
minutes.
        --convert binary to BCD for hours
```

```
        bin2bcd_hrs : bin2bcd
                port map(
                        binary_in => hours_extended,
                        bcd_out => bcd_hrs
                        );

        --Call the bcd2seg7 function to convert each BCD digit into a
 --format which can be used on the 7 segment display
        secs_7seg1 <= bcd2seg7(bcd_secs(3 downto 0));
        secs_7seg10 <= bcd2seg7(bcd_secs(7 downto 4));
        mins_7seg1 <= bcd2seg7(bcd_mins(3 downto 0));
        mins_7seg10 <= bcd2seg7(bcd_mins(7 downto 4));
        hrs_7seg1 <= bcd2seg7(bcd_hrs(3 downto 0));
        hrs_7seg10 <= bcd2seg7(bcd_hrs(7 downto 4));

end Behavioral;
```

## Testbench for the top module: tb_digitalClock_topModule.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
--the below library is used for finishing the simulation after we are done.
--Otherwise it will run continously.
library std;
use std.env.finish;

ENTITY tb_digitalClock_topModule IS
END tb_digitalClock_topModule;

ARCHITECTURE behavior OF tb_digitalClock_topModule IS

   -- Component Declaration for the Unit Under Test (UUT)
   COMPONENT digital_clock_topmodule is
        --frequency of the clock passed as a generic parameter.
        generic(
                CLOCK_FREQ : integer := 50000000
                        );
        port(
                Clock : in std_logic;  --system clock
                reset : in std_logic;  --resets the time
                inc_secs : in std_logic;  --set a pulse here to increment the seconds by 1.
                inc_mins : in std_logic;  --set a pulse here to increment the minutes by 1.
                inc_hrs : in std_logic;  --set a pulse here to increment the hours by 1.
                secs_7seg1 :out unsigned(6 downto 0);  --seconds LSB digi
                secs_7seg10 :out unsigned(6 downto 0);  --seconds MSB digit
                mins_7seg1 :out unsigned(6 downto 0);  --minutes LSB digit
                mins_7seg10 :out unsigned(6 downto 0);  --minutes MSB digit
                hrs_7seg1 :out unsigned(6 downto 0);  --hours LSB digit
                hrs_7seg10 :out unsigned(6 downto 0)  --hours MSB digit
                        );
end COMPONENT;
```

```
--Inputs
signal Clock : std_logic := '0';
signal reset : std_logic := '0';
signal inc_secs : std_logic := '0';
signal inc_mins : std_logic := '0';
signal inc_hrs : std_logic := '0';

        --Outputs
signal secs_7seg1,secs_7seg10 : unsigned(6 downto 0);
signal mins_7seg1,mins_7seg10 : unsigned(6 downto 0);
signal hrs_7seg1,hrs_7seg10 : unsigned(6 downto 0);
-- Clock period definitions
constant Clock_period : time := 10 ns;

        --Clock frequency in Hz. Use a smaller value for testbench.
--When testing on board it need to be set as 50 million, 100 million etc.
        constant CLOCK_FREQ : integer := 10;

BEGIN

        -- Instantiate the Unit Under Test (UUT)
uut: digital_clock_topmodule
                GENERIC MAP(CLOCK_FREQ => CLOCK_FREQ)
                PORT MAP (
      Clock => Clock,
      reset => reset,
      inc_secs => inc_secs,
      inc_mins => inc_mins,
      inc_hrs => inc_hrs,
      secs_7seg1 => secs_7seg1,
                        secs_7seg10 => secs_7seg10,
                        mins_7seg1 => mins_7seg1,
                        mins_7seg10 => mins_7seg10,
                        hrs_7seg1 => hrs_7seg1,
                        hrs_7seg10 => hrs_7seg10
   );

-- Clock process definitions
Clock_process :process
begin
                Clock <= '0';
                wait for Clock_period/2;
                Clock <= '1';
                wait for Clock_period/2;
end process;


-- Stimulus process
stim_proc: process
begin
                reset <= '1';
```

```
    -- hold reset state for 100 ns.
    wait for 100 ns;
                        reset <= '0';
    wait for Clock_period*CLOCK_FREQ*60*60*25;  --run the clock for 25 hours

                        --increment seconds
                        inc_secs <= '1';       wait for Clock_period;
                        inc_secs <= '0';
    wait for Clock_period*CLOCK_FREQ*5;--wait for 5 secs after incrementing seconds once.

                        --increment seconds 60 times
                        inc_secs <= '1';       wait for Clock_period*60;
                        inc_secs <= '0';
    wait for Clock_period*CLOCK_FREQ*5;      --wait for 5 secs after incrementing seconds.

                        --increment minutes
                        inc_mins <= '1';       wait for Clock_period;
                        inc_mins <= '0';
    wait for Clock_period*CLOCK_FREQ*5;--wait for 5 mins after incrementing minutes once.

                        --increment minutes 60 times
                        inc_mins <= '1';       wait for Clock_period*60;
                        inc_mins <= '0';
    wait for Clock_period*CLOCK_FREQ*5;      --wait for 5 mins after incrementing minutes.

                        --increment hours
                        inc_hrs <= '1';        wait for Clock_period;
                        inc_hrs <= '0';
    wait for Clock_period*CLOCK_FREQ*5;--wait for 5 hours after incrementing hours once.

                        --increment hours 25 times
                        inc_hrs <= '1';        wait for Clock_period*25;
                        inc_hrs <= '0';
    wait for Clock_period*CLOCK_FREQ*5;      --wait for 5 hours after incrementing hours.

                        --apply reset
                        reset <= '1';

                        --wait for 100 Clock cycles and then finish the simulation.
                        --with the current settings it will run around 9 ms of simualtion time.
                        wait for Clock_period*100;
                        finish;
  end process;
END;
```