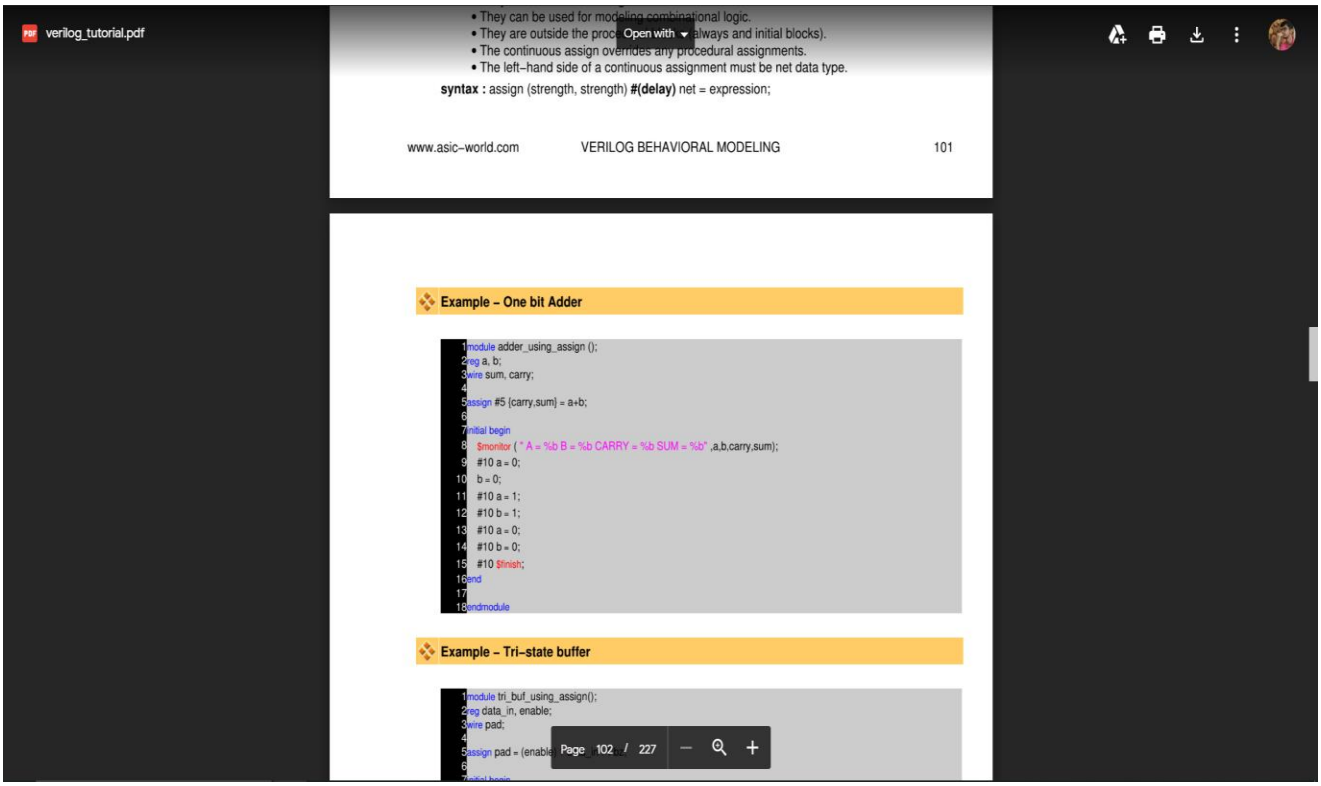
DAILY ASSESSMENT REPORT

Date:	**05 June 2020**	**Name:**	**Gagan M K**
Course:	**DIGITAL DESIGN USING HDL**	**USN:**	**4AL17EC032**
Topic:	• Verilog Tutorials and practice programs • Building/ Demo projects using FPGA	**Semester & Section:**	**6th sem & 'A' sec**
GitHub Repository:	**Alvas-education-foundation/Gagan-Git**		

FORENOON SESSION DETAILS
Image of session



Report – Report can be typed or hand written for up to two pages.

Verilog Tutorials and practice programs:

- **Tasks are used in all programming languages, generally known as Procedures or sub routines.**
- **Many lines of code are enclosed in task.... end task brackets. Data is passed to the task, the processing done, and the result returned to a specified value. They have to be specifically called, with data ins and outs, rather than just wired in to the general netlist. Included in the main body of code they can be called many times, reducing code repetition.**
- **Task is defined in the module in which they are used. it is possible to define task in separate file and use compile directive 'include to include the task in the file which instantiates the task.**
- **Task can include timing delays, like posedge, negedge, # delay and wait.**
- **Task can have any number of inputs and outputs.**
- **The variables declared within the task are local to that task. The order of declaration within the task defines how the variables passed to the task by the caller are used.**
- **Task can take, drive and source global variables, when no local variables are used. When local variables are used, it basically assigned output only at the end of task execution.**
- **Task can call another task or function.**
- **Task can be used for modelling both combinational and sequential logic.**
- **A task must be specifically called with a statement, it cannot be used within an expression as a function can**

- **Simple Function:**

```
module simple_function();

function myfunction;
input a, b, c, d;
begin
   myfunction = ((a+b) + (c–d));
end
endfunction

endmodule
```

Building/ Demo projects using FPGA:

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.NUMERIC_STD.all;

---------- ALU 8-bit VHDL ----------

```
entity ALU is
  generic (
    constant N: natural := 1  -- number of shited or rotated bits
   );

   Port (
   A, B     : in  STD_LOGIC_VECTOR(7 downto 0);  -- 2 inputs 8-bit
   ALU_Sel  : in  STD_LOGIC_VECTOR(3 downto 0);  -- 1 input 4-bit for selecting function
   ALU_Out   : out  STD_LOGIC_VECTOR(7 downto 0); -- 1 output 8-bit
   Carryout : out std_logic        -- Carryout flag
   );
end ALU;
architecture Behavioral of ALU is

signal ALU_Result : std_logic_vector (7 downto 0);
signal tmp: std_logic_vector (8 downto 0);

begin
   process(A,B,ALU_Sel)
 begin
  case(ALU_Sel) is
  when "0000" => -- Addition
   ALU_Result <= A + B ;
  when "0001" => -- Subtraction
   ALU_Result <= A - B ;
  when "0010" => -- Multiplication
   ALU_Result <= std_logic_vector(to_unsigned((to_integer(unsigned(A)) *
to_integer(unsigned(B))),8)) ;
  when "0011" => -- Division
   ALU_Result <= std_logic_vector(to_unsigned(to_integer(unsigned(A)) /
to_integer(unsigned(B)),8)) ;
  when "0100" => -- Logical shift left
   ALU_Result <= std_logic_vector(unsigned(A) sll N);
  when "0101" => -- Logical shift right
   ALU_Result <= std_logic_vector(unsigned(A) srl N);
  when "0110" => --  Rotate left
   ALU_Result <= std_logic_vector(unsigned(A) rol N);
  when "0111" => -- Rotate right
   ALU_Result <= std_logic_vector(unsigned(A) ror N);
  when "1000" => -- Logical and
   ALU_Result <= A and B;
  when "1001" => -- Logical or
   ALU_Result <= A or B;
  when "1010" => -- Logical xor
   ALU_Result <= A xor B;
  when "1011" => -- Logical nor
```

```
  ALU_Result <= A nor B;
 when "1100" => -- Logical nand
  ALU_Result <= A nand B;
 when "1101" => -- Logical xnor
  ALU_Result <= A xnor B;
 when "1110" => -- Greater comparison
  if(A>B) then
   ALU_Result <= x"01" ;
  else
   ALU_Result <= x"00" ;
  end if;
 when "1111" => -- Equal comparison
  if(A=B) then
   ALU_Result <= x"01" ;
  else
   ALU_Result <= x"00" ;
  end if;
 when others => ALU_Result <= A + B ;
 end case;
end process;
ALU_Out <= ALU_Result; -- ALU out
tmp <= ('0' & A) + ('0' & B);
Carryout <= tmp(8); -- Carryout flag
end Behavioral;
```

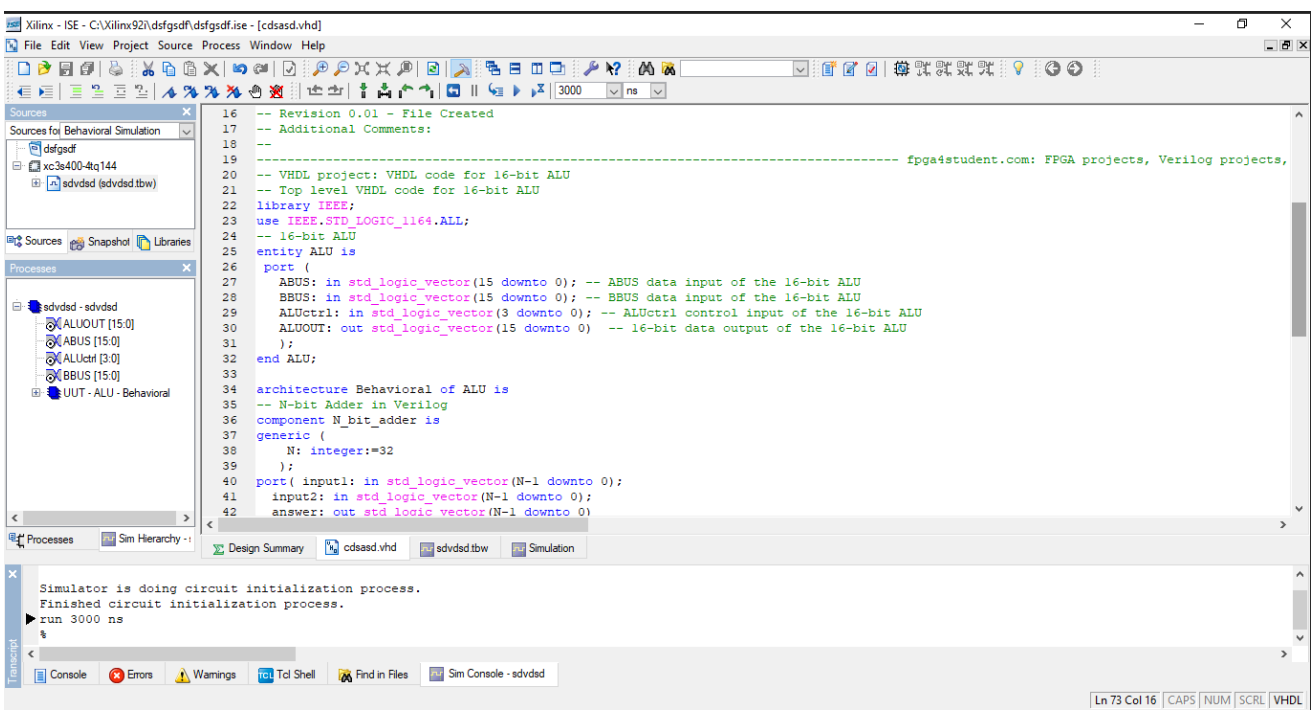
Test Bench:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
use IEEE.std_logic_unsigned.all;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;

ENTITY tb_ALU IS
END tb_ALU;

ARCHITECTURE behavior OF tb_ALU IS

   -- Component Declaration for the Unit Under Test (UUT)

   COMPONENT ALU
   PORT(
       A : IN  std_logic_vector(7 downto 0);
       B : IN  std_logic_vector(7 downto 0);
```

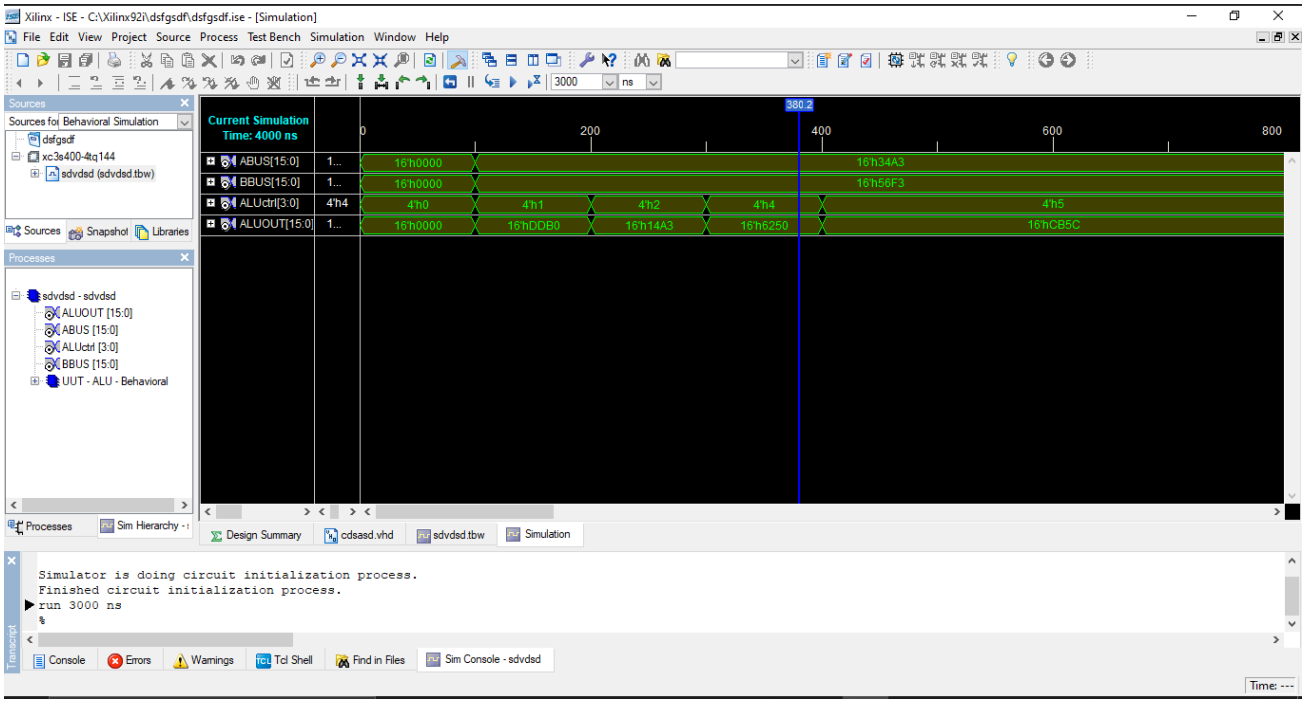
```
        ALU_Sel : IN  std_logic_vector(3 downto 0);
        ALU_Out : OUT  std_logic_vector(7 downto 0);
        Carryout : OUT  std_logic
       );
   END COMPONENT;
      --Inputs
  signal A : std_logic_vector(7 downto 0) := (others => '0');
  signal B : std_logic_vector(7 downto 0) := (others => '0');
  signal ALU_Sel : std_logic_vector(3 downto 0) := (others => '0');

 --Outputs
  signal ALU_Out : std_logic_vector(7 downto 0);
  signal Carryout : std_logic;

 signal i:integer;
BEGIN

 -- Instantiate the Unit Under Test (UUT)
  uut: ALU PORT MAP (
        A => A,
        B => B,
        ALU_Sel => ALU_Sel,
        ALU_Out => ALU_Out,
        Carryout => Carryout
       );


  -- Stimulus process
  stim_proc: process
  begin
     -- hold reset state for 100 ns.
     A <= x"0A";
 B <= x"02";
 ALU_Sel <= x"0";

 for i in 0 to 15 loop
  ALU_Sel <= ALU_Sel + x"1";
  wait for 100 ns;
 end loop;
     A <= x"F6";
 B <= x"0A";
     wait;
  end process;

END;
```

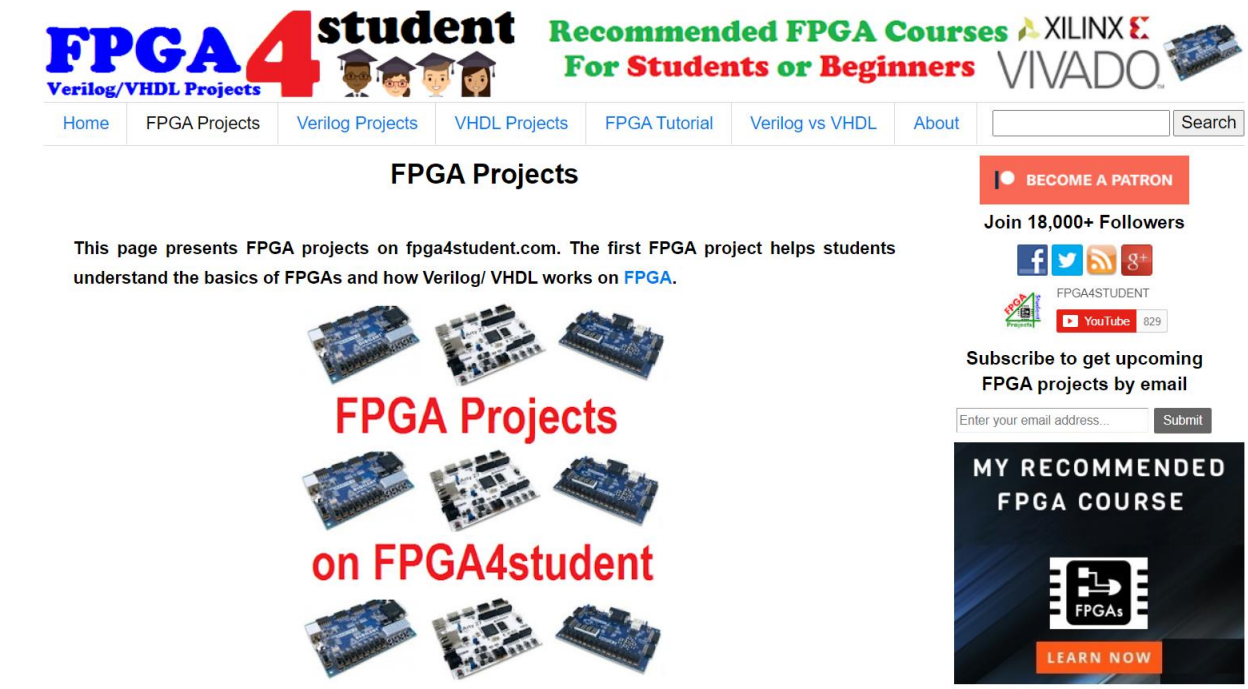
Implement a verilog module to count number of 0's in a 16 bit number in compiler:

```
module num_zero_ones(
     input [15:0] In,
     output reg [4:0] ones,
     output reg [4:0] zeros);
integer i, o, z;
always@(In)
begin
   o = 0;                    //initialize count variable.
   z = 0;                    //initialize count variable.
   for(i=0;i<16;i=i+1)   //check for all the bits.
     If(In[i] == 1'b1)     //check if the bit is '1'
          o = o + 1;       //if its one, increment the count.
      z = 16-o; //number of zeros.
      ones = o;
      zeros = z;
end
endmodule
```



Compiler Output:

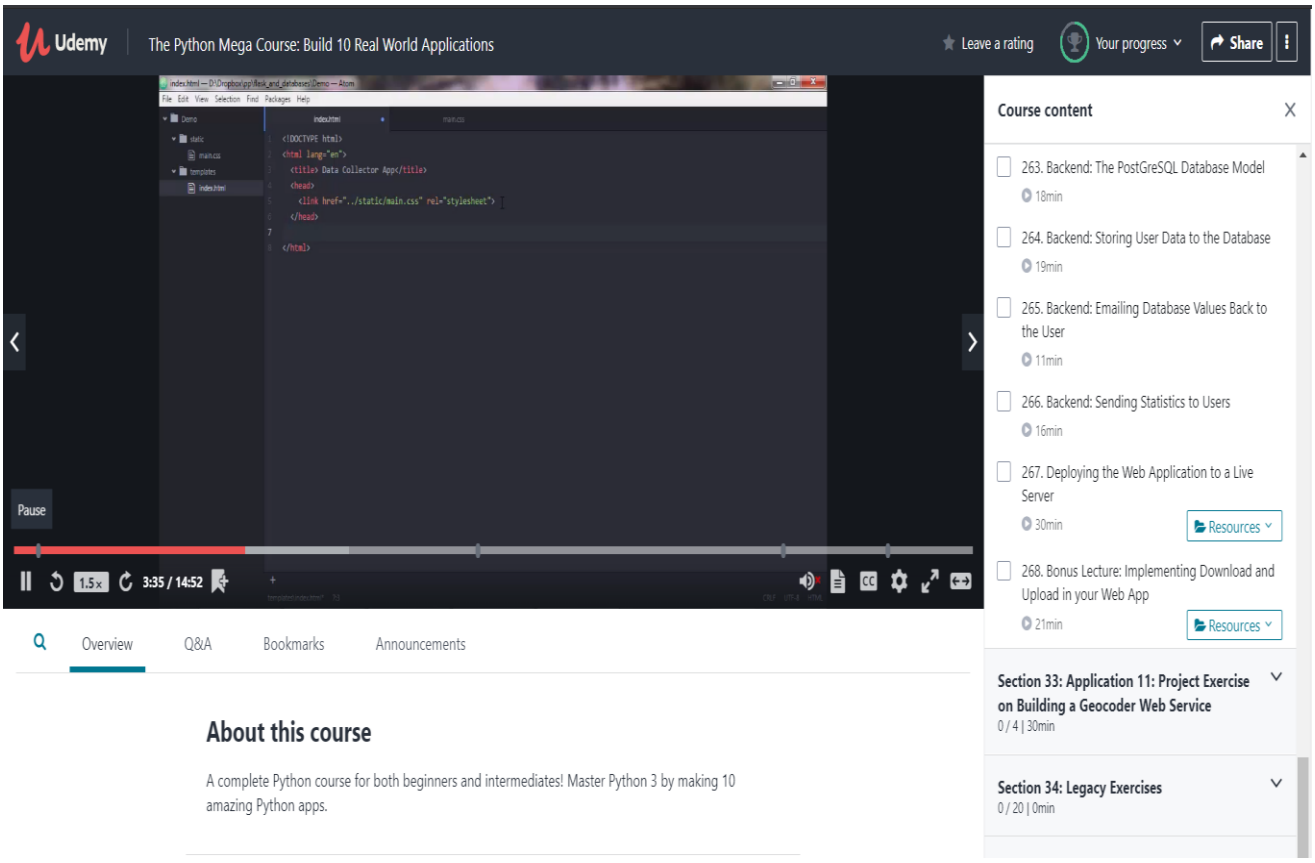




Date:	05 June 2020	Name:	Gagan M K
Course:	The Python Mega Course	USN:	4AL17EC032
Topic:	Application 10: Build a Data Collector Web App with PostGreSQL and Flask	Semester & Section:	6th sem & 'A' sec

AFTERNOON SESSION DETAILS

Image of session:



Report – Report can be typed or hand written for up to two pages.

- **Data Collector Web App saw how The Output.**
- **Used PostGreSQL Database Web App with Flask:**
- **Making Frontend using HTML.**
- **Creating Frontend using CSS.**
- **Backend which is Getting User Input**
- **Backend: The PostGreSQL Database Model**
- **Learnt how to Store User Data to the Database**
- **Backend: Emailing Database Values Back to the User**
- **Learnt how to Send Statistics to Users**
- **Deploying the Web Application to a Live Server was shown.**
- **Bonus Lecture: Implementing Download and Upload in your Web App**

- **The data collected is shown in the below picture.**

	A	B	C	D	E	F	G	H	I
1		Address	Area	Beds	Full Baths	Half Baths	Locality	Lot Size	Price
2	0	0 Gateway					Rock Springs, WY 82		$725,000
3	1	1003 Winchester Blv		4	4		Rock Sprir	0.21 Acres	$452,900
4	2	3239 Spea	3,076	4	3	1	Rock Sprir	Under 1/2	$379,900
5	3	600 Tallad	3,154	5	3		Rock Springs, WY 82		$379,000
6	4	3457 Briso	3,236	5	3		Rock Sprir	0.34 Acres	$349,900
7	5	234 Via Sp	2,688	4	3		Rock Sprir	Under 1/2	$330,000
8	6	2425 Cripp	8,263	4	35		Rock Springs, WY 82		$279,900
9	7	522 Emera	1,172	3	3		Rock Sprir	Under 1/2	$254,000
10	8	1302 Vete	1,932	4	2		Rock Sprir	0.27 Acres	$252,900
11	9	343 Via Rucce		3	2	1	Rock Sprir	0.16 Acres	$219,900
12	10	913 Madis	1,344	3	2		Rock Sprir	Under 1/2	$209,000