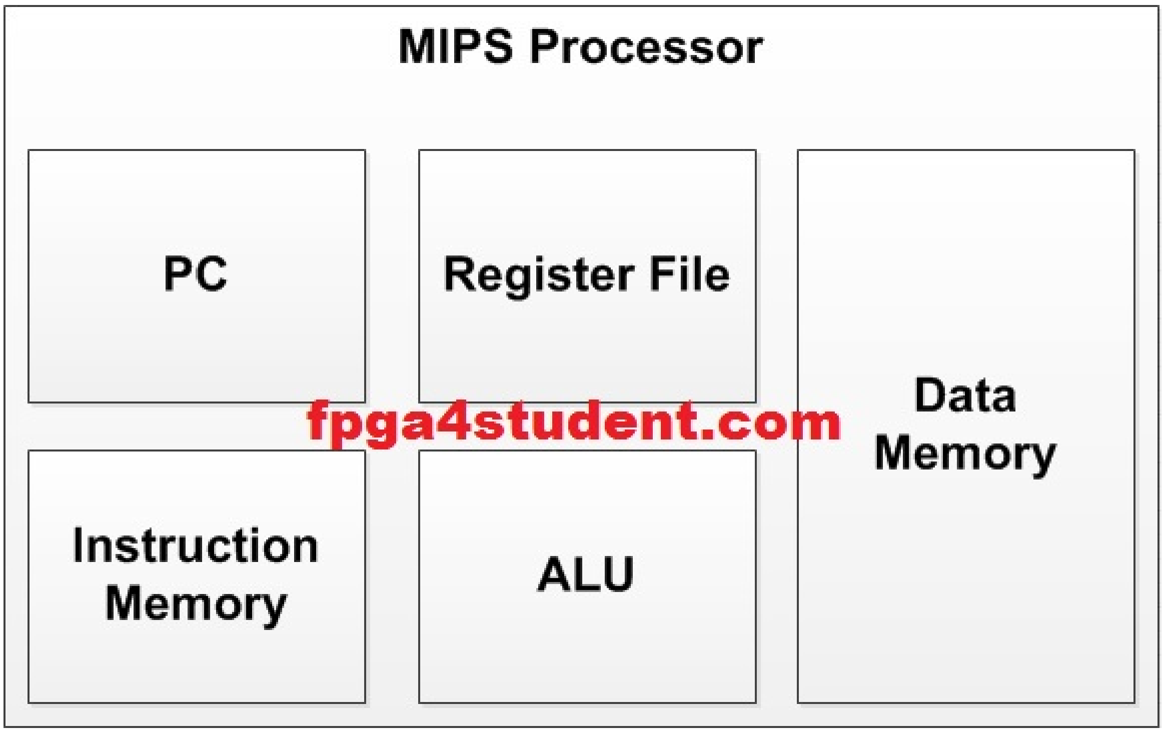
VHDL code for MIPS Processor

Last time, I presented a Verilog code for a 16-bit single-cycle MIPS processor. The instruction set and architecture design for the MIPS processor was provided here.

Today, the VHDL code for the MIPS Processor will be presented. A simple VHDL testbench for the MIPS processor will be also provided for simulation purposes.



VHDL code for Data Memory of the MIPS processor:

```
-- fpga4student.com: FPGA projects, Verilog projects, VHDL projects
-- VHDL project: VHDL code for single-cycle MIPS Processor
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE IEEE.numeric_std.all;
-- VHDL code for the data Memory of the MIPS Processor
entity Data_Memory_VHDL is
port (
 clk: in std_logic;
 mem_access_addr: in std_logic_Vector(15 downto 0);
 mem_write_data: in std_logic_Vector(15 downto 0);
 mem_write_en,mem_read:in std_logic;
 mem_read_data: out std_logic_Vector(15 downto 0)
);
```

```
end Data_Memory_VHDL;

architecture Behavioral of Data_Memory_VHDL is
signal i: integer;
signal ram_addr: std_logic_vector(7 downto 0);
type data_mem is array (0 to 255 ) of std_logic_vector (15 downto 0);
signal RAM: data_mem :=((others=> (others=>'0')));
begin

 ram_addr <= mem_access_addr(8 downto 1);
 process(clk)
 begin
  if(rising_edge(clk)) then
  if (mem_write_en='1') then
  ram(to_integer(unsigned(ram_addr))) <= mem_write_data;
  end if;
  end if;
 end process;
   mem_read_data <= ram(to_integer(unsigned(ram_addr))) when (mem_read='1') else

end Behavioral;
```

VHDL code for ALU of the MIPS processor:

```
-- fpga4student.com: FPGA projects, Verilog projects, VHDL projects
-- VHDL project: VHDL code for single-cycle MIPS Processor
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_signed.all;
-- VHDL code for ALU of the MIPS Processor
entity ALU_VHDL is
port(
 a,b : in std_logic_vector(15 downto 0); -- src1, src2
 alu_control : in std_logic_vector(2 downto 0); -- function select
 alu_result: out std_logic_vector(15 downto 0); -- ALU Output Result
 zero: out std_logic -- Zero Flag
 );
end ALU_VHDL;

architecture Behavioral of ALU_VHDL is
signal result: std_logic_vector(15 downto 0);
begin
process(alu_control,a,b)
begin
 case alu_control is
 when "000" =>
  result <= a + b; -- add
 when "001" =>
  result <= a - b; -- sub
 when "010" =>
  result <= a and b; -- and
 when "011" =>
  result <= a or b; -- or
 when "100" =>
  if (a<b) then
```

```
    result <= x"0001";
   else
    result <= x"0000";
   end if;
  when others => result <= a + b; -- add
  end case;
 end process;
   zero <= '1' when result=x"0000" else '0';
   alu_result <= result;
 end Behavioral;
```

VHDL code for ALU Control Unit of the MIPS processor:

```
-- fpga4student.com: FPGA projects, Verilog projects, VHDL projects
-- VHDL project: VHDL code for single-cycle MIPS Processor
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
-- VHDL code for ALU Control Unit of the MIPS Processor
entity ALU_Control_VHDL is
port(
  ALU_Control: out std_logic_vector(2 downto 0);
  ALUOp : in std_logic_vector(1 downto 0);
  ALU_Funct : in std_logic_vector(2 downto 0)
);
end ALU_Control_VHDL;

architecture Behavioral of ALU_Control_VHDL is
begin
process(ALUOp,ALU_Funct)
begin
case ALUOp is
when "00" =>
 ALU_Control <= ALU_Funct(2 downto 0);
when "01" =>
 ALU_Control <= "001";
when "10" =>
 ALU_Control <= "100";
when "11" =>
 ALU_Control <= "000";
when others => ALU_Control <= "000";
end case;
end process;
end Behavioral;
```

VHDL code for Register File of the MIPS processor:

```
-- fpga4student.com: FPGA projects, Verilog projects, VHDL projects
-- VHDL project: VHDL code for single-cycle MIPS Processor
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE IEEE.numeric_std.all;
-- VHDL code for the register file of the MIPS Processor
entity register_file_VHDL is
port (
 clk,rst: in std_logic;
 reg_write_en: in std_logic;
 reg_write_dest: in std_logic_vector(2 downto 0);
 reg_write_data: in std_logic_vector(15 downto 0);
 reg_read_addr_1: in std_logic_vector(2 downto 0);
 reg_read_data_1: out std_logic_vector(15 downto 0);
 reg_read_addr_2: in std_logic_vector(2 downto 0);
 reg_read_data_2: out std_logic_vector(15 downto 0)
);
end register_file_VHDL;

architecture Behavioral of register_file_VHDL is
type reg_type is array (0 to 7 ) of std_logic_vector (15 downto 0);
signal reg_array: reg_type;
begin
 process(clk,rst)
 begin
 if(rst='1') then
   reg_array(0) <= x"0001";
   reg_array(1) <= x"0002";
   reg_array(2) <= x"0003";
   reg_array(3) <= x"0004";
   reg_array(4) <= x"0005";
   reg_array(5) <= x"0006";
   reg_array(6) <= x"0007";
   reg_array(7) <= x"0008";
 elsif(rising_edge(clk)) then
   if(reg_write_en='1') then
    reg_array(to_integer(unsigned(reg_write_dest))) <= reg_write_data;
   end if;
 end if;
 end process;

 reg_read_data_1 <= x"0000" when reg_read_addr_1 = "000" else reg_array(to_integ
 reg_read_data_2 <= x"0000" when reg_read_addr_2 = "000" else reg_array(to_integ

end Behavioral;
```

VHDL code for Control Unit of the MIPS processor:

```
-- fpga4student.com: FPGA projects, Verilog projects, VHDL projects
-- VHDL project: VHDL code for single-cycle MIPS Processor
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
-- VHDL code for Control Unit of the MIPS Processor
entity control_unit_VHDL is
port (
  opcode: in std_logic_vector(2 downto 0);
  reset: in std_logic;
  reg_dst,mem_to_reg,alu_op: out std_logic_vector(1 downto 0);
  jump,branch,mem_read,mem_write,alu_src,reg_write,sign_or_zero: out std_logic
 );
end control_unit_VHDL;

architecture Behavioral of control_unit_VHDL is

begin
process(reset,opcode)
```

```
begin
 if(reset = '1') then
   reg_dst <= "00";
   mem_to_reg <= "00";
   alu_op <= "00";
   jump <= '0';
   branch <= '0';
   mem_read <= '0';
   mem_write <= '0';
   alu_src <= '0';
   reg_write <= '0';
   sign_or_zero <= '1';
 else
 case opcode is
  when "000" => -- add
    reg_dst <= "01";
    mem_to_reg <= "00";
    alu_op <= "00";
    jump <= '0';
    branch <= '0';
    mem_read <= '0';
    mem_write <= '0';
    alu_src <= '0';
    reg_write <= '1';
    sign_or_zero <= '1';
  when "001" => -- sliu
   reg_dst <= "00";
   mem_to_reg <= "00";
   alu_op <= "10";
   jump <= '0';
   branch <= '0';
   mem_read <= '0';
   mem_write <= '0';
   alu_src <= '1';
   reg_write <= '1';
   sign_or_zero <= '0';
  when "010" => -- j
   reg_dst <= "00";
   mem_to_reg <= "00";
   alu_op <= "00";
   jump <= '1';
   branch <= '0';
   mem_read <= '0';
   mem_write <= '0';
   alu_src <= '0';
   reg_write <= '0';
   sign_or_zero <= '1';
 when "011" =>-- jal
   reg_dst <= "10";
   mem_to_reg <= "10";
   alu_op <= "00";
   jump <= '1';
   branch <=  '0';
   mem_read <=  '0';
   mem_write <=  '0';
```

```
      alu_src <= '0';
      reg_write <=  '1';
      sign_or_zero <= '1';
    when "100" =>-- lw
      reg_dst <= "00";
      mem_to_reg <= "01";
      alu_op <= "11";
      jump <= '0';
      branch <= '0';
      mem_read <= '1';
      mem_write <= '0';
      alu_src <= '1';
      reg_write <= '1';
      sign_or_zero <= '1';
    when "101" => -- sw
      reg_dst <= "00";
      mem_to_reg <= "00";
      alu_op <= "11";
      jump <= '0';
      branch <= '0';
      mem_read <= '0';
      mem_write <= '1';
      alu_src <= '1';
      reg_write <= '0';
      sign_or_zero <= '1';
    when "110" => -- beq
      reg_dst <= "00";
      mem_to_reg <= "00";
      alu_op <= "01";
      jump <= '0';
      branch <= '1';
      mem_read <= '0';
      mem_write <= '0';
      alu_src <= '0';
      reg_write <= '0';
      sign_or_zero <= '1';
    when "111" =>-- addi
      reg_dst <= "00";
      mem_to_reg <= "00";
      alu_op <= "11";
      jump <= '0';
      branch <= '0';
      mem_read <= '0';
      mem_write <= '0';
      alu_src <= '1';
      reg_write <= '1';
      sign_or_zero <= '1';
    when others =>
       reg_dst <= "01";
       mem_to_reg <= "00";
       alu_op <= "00";
       jump <= '0';
       branch <= '0';
       mem_read <= '0';
       mem_write <= '0';
       alu_src <= '0';
       reg_write <= '1';
       sign_or_zero <= '1';
    end case;
    end if;
   end process;

   end Behavioral;
```

VHDL code for Instruction Memory of the MIPS processor:

```
-- fpga4student.com: FPGA projects, Verilog projects, VHDL projects
-- VHDL project: VHDL code for single-cycle MIPS Processor
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE IEEE.numeric_std.all;
-- VHDL code for the Instruction Memory of the MIPS Processor
entity Instruction_Memory_VHDL is
port (
 pc: in std_logic_vector(15 downto 0);
 instruction: out  std_logic_vector(15 downto 0)
);
end Instruction_Memory_VHDL;

architecture Behavioral of Instruction_Memory_VHDL is
signal rom_addr: std_logic_vector(3 downto 0);
 -- lw $3, 0($0) -- pc=0
 -- Loop: sltiu  $1, $3, 11= pc = 2
 -- beq $1, $0, Skip = 4 --PCnext=PC_current+2+BranchAddr
 -- add $4, $4, $3 = 6
 -- addi $3, $3, 1 = 8
 -- beq $0, $0, Loop--a
 -- Skip c = 12 = 4 + 2 + br
 type ROM_type is array (0 to 15 ) of std_logic_vector(15 downto 0);
 constant rom_data: ROM_type:=(
   "1000000110000000",
   "0010110010001011",
   "1100010000000011",
   "0001000111000000",
   "1110110110000001",
   "1100000001111011",
   "0000000000000000",
   "0000000000000000",
   "0000000000000000",
   "0000000000000000",
   "0000000000000000",
   "0000000000000000",
   "0000000000000000",
   "0000000000000000",
   "0000000000000000",
   "0000000000000000"
  );
begin

 rom_addr <= pc(4 downto 1);
  instruction <= rom_data(to_integer(unsigned(rom_addr))) when pc < x"0020" else
```

```
end Behavioral;
```

VHDL code for the MIPS Processor:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_signed.all;
-- fpga4student.com: FPGA projects, Verilog projects, VHDL projects
-- VHDL project: VHDL code for single-cycle MIPS Processor
entity MIPS_VHDL is
port (
 clk,reset: in std_logic;
 pc_out, alu_result: out std_logic_vector(15 downto 0)
);
end MIPS_VHDL;

architecture Behavioral of MIPS_VHDL is
 signal pc_current: std_logic_vector(15 downto 0);
 signal pc_next,pc2: std_logic_vector(15 downto 0);
 signal instr: std_logic_vector(15 downto 0);
 signal reg_dst,mem_to_reg,alu_op: std_logic_vector(1 downto 0);
 signal jump,branch,mem_read,mem_write,alu_src,reg_write: std_logic;
 signal reg_write_dest: std_logic_vector(2 downto 0);
 signal reg_write_data: std_logic_vector(15 downto 0);
 signal reg_read_addr_1: std_logic_vector(2 downto 0);
 signal reg_read_data_1: std_logic_vector(15 downto 0);
 signal reg_read_addr_2: std_logic_vector(2 downto 0);
 signal reg_read_data_2: std_logic_vector(15 downto 0);
 signal sign_ext_im,read_data2,zero_ext_im,imm_ext: std_logic_vector(15 downto 0
 signal JRControl: std_logic;
 signal ALU_Control: std_logic_vector(2 downto 0);
 signal ALU_out: std_logic_vector(15 downto 0);
 signal zero_flag: std_logic;
 signal im_shift_1, PC_j, PC_beq, PC_4beq,PC_4beqj,PC_jr: std_logic_vector(15 do
 signal beq_control: std_logic;
 signal jump_shift_1: std_logic_vector(14 downto 0);
 signal mem_read_data: std_logic_vector(15 downto 0);
 signal no_sign_ext: std_logic_vector(15 downto 0);
 signal sign_or_zero: std_logic;
 signal tmp1: std_logic_vector(8 downto 0);
begin
-- PC of the MIPS Processor in VHDL
process(clk,reset)
begin
 if(reset='1') then
  pc_current <= x"0000";
 elsif(rising_edge(clk)) then
  pc_current <= pc_next;
 end if;
end process;
-- PC + 2
  pc2 <= pc_current + x"0002";
-- instruction memory of the MIPS Processor in VHDL
Instruction_Memory: entity work.Instruction_Memory_VHDL
```

```
        port map
       (
        pc=> pc_current,
        instruction => instr
        );
-- jump shift left 1
 jump_shift_1 <= instr(13 downto 0) & '0';
-- control unit of the MIPS Processor in VHDL
control: entity work.control_unit_VHDL
   port map
   (reset => reset,
    opcode => instr(15 downto 13),
    reg_dst => reg_dst,
    mem_to_reg => mem_to_reg,
    alu_op => alu_op,
    jump => jump,
    branch => branch,
    mem_read => mem_read,
    mem_write => mem_write,
    alu_src => alu_src,
    reg_write => reg_write,
    sign_or_zero => sign_or_zero
    );
-- multiplexer regdest
  reg_write_dest <= "111" when  reg_dst= "10" else
        instr(6 downto 4) when  reg_dst= "01" else
        instr(9 downto 7);
-- register file instantiation of the MIPS Processor in VHDL
 reg_read_addr_1 <= instr(12 downto 10);
 reg_read_addr_2 <= instr(9 downto 7);
register_file: entity work.register_file_VHDL
 port map
 (
 clk => clk,
 rst => reset,
 reg_write_en => reg_write,
 reg_write_dest => reg_write_dest,
 reg_write_data => reg_write_data,
 reg_read_addr_1 => reg_read_addr_1,
 reg_read_data_1 => reg_read_data_1,
 reg_read_addr_2 => reg_read_addr_2,
 reg_read_data_2 => reg_read_data_2
 );
-- sign extend
 tmp1 <= (others => instr(6));
 sign_ext_im <=  tmp1 & instr(6 downto 0);
 zero_ext_im <= "000000000"& instr(6 downto 0);
 imm_ext <= sign_ext_im when sign_or_zero='1' else zero_ext_im;
-- JR control unit of the MIPS Processor in VHDL
 JRControl <= '1' when ((alu_op="00") and (instr(3 downto 0)="1000")) else '0';
-- ALU control unit of the MIPS Processor in VHDL
ALUControl: entity work.ALU_Control_VHDL port map
  (
   ALUOp => alu_op,
   ALU_Funct => instr(2 downto 0),
   ALU_Control => ALU_Control
   );
-- multiplexer alu_src
 read_data2 <= imm_ext when alu_src='1' else reg_read_data_2;
-- ALU unit of the MIPS Processor in VHDL
alu: entity work.ALU_VHDL port map
  (
   a => reg_read_data_1,
   b => read_data2,
   alu_control => ALU_Control,
   alu_result => ALU_out,
   zero => zero_flag
   );
-- immediate shift 1
 im_shift_1 <= imm_ext(14 downto 0) & '0';
 no_sign_ext <= (not im_shift_1) + x"0001";
-- PC beq add
 PC_beq <= (pc2 - no_sign_ext) when im_shift_1(15) = '1' else (pc2 +im_shift_1);
-- beq control
   beq_control <= branch and zero_flag;
-- PC_beq
   PC_4beq <= PC_beq when beq_control='1' else pc2;
-- PC_j
 PC_j <= pc2(15) & jump_shift_1;
-- PC_4beqj
 PC_4beqj <= PC_j when jump = '1' else  PC_4beq;
-- PC_jr
 PC_jr <= reg_read_data_1;
-- PC_next
 pc_next <= PC_jr when (JRControl='1') else PC_4beqj;
-- data memory of the MIPS Processor in VHDL
data_memory: entity work.Data_Memory_VHDL port map
  (
  clk => clk,
  mem_access_addr => ALU_out,
  mem_write_data => reg_read_data_2,
  mem_write_en => mem_write,
  mem_read => mem_read,
  mem_read_data => mem_read_data
  );
-- write back of the MIPS Processor in VHDL
 reg_write_data <= pc2 when (mem_to_reg = "10") else
       mem_read_data when (mem_to_reg = "01") else ALU_out;
-- output
 pc_out <= pc_current;
 alu_result <= ALU_out;

end Behavioral;
```

VHDL testbench for the MIPS processor:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
-- fpga4student.com: FPGA projects, Verilog projects, VHDL projects
-- VHDL project: VHDL code for single-cycle MIPS Processor
```

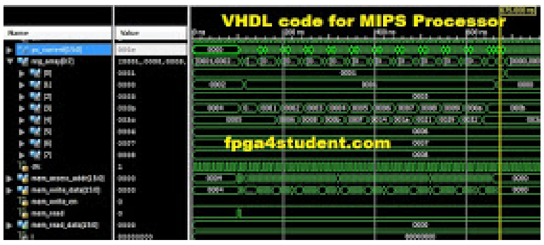
```vhdl
-- VHDL testbench code for single-cycle MIPS Processor
ENTITY tb_MIPS_VHDL IS
END tb_MIPS_VHDL;

ARCHITECTURE behavior OF tb_MIPS_VHDL IS
    -- Component Declaration for the single-cycle MIPS Processor in VHDL
    COMPONENT MIPS_VHDL
    PORT(
         clk : IN  std_logic;
         reset : IN  std_logic;
         pc_out : OUT  std_logic_vector(15 downto 0);
         alu_result : OUT  std_logic_vector(15 downto 0)
        );
    END COMPONENT;
   --Inputs
   signal clk : std_logic := '0';
   signal reset : std_logic := '0';
   --Outputs
   signal pc_out : std_logic_vector(15 downto 0);
   signal alu_result : std_logic_vector(15 downto 0);
   -- Clock period definitions
   constant clk_period : time := 10 ns;
BEGIN
 -- Instantiate the for the single-cycle MIPS Processor in VHDL
   uut: MIPS_VHDL PORT MAP (
          clk => clk,
          reset => reset,
          pc_out => pc_out,
          alu_result => alu_result
        );

   -- Clock process definitions
   clk_process :process
   begin
  clk <= '0';
  wait for clk_period/2;
  clk <= '1';
  wait for clk_period/2;
   end process;
   -- Stimulus process
   stim_proc: process
   begin
      reset <= '1';
      wait for clk_period*10;
  reset <= '0';
      -- insert stimulus here
      wait;
   end process;

END;
```

Simulation Waveform for the MIPS processor in VHDL:



You can change the instructions in the instruction memory and run the simulation for many different combinations of the instructions.

Verilog code for Pipelined MIPS Processor

Verilog code for RISC Processor

Verilog code for single-cycle MIPS Processor

Recommended VHDL projects:

1. **What is an FPGA? How VHDL works on FPGA**
2. **VHDL code for FIFO memory**
3. **VHDL code for FIR Filter**
4. **VHDL code for 8-bit Microcontroller**
5. **VHDL code for Matrix Multiplication**
6. **VHDL code for Switch Tail Ring Counter**
7. **VHDL code for digital alarm clock on FPGA**
8. **VHDL code for 8-bit Comparator**
9. **How to load a text file into FPGA using VHDL**
10. **VHDL code for D Flip Flop**
11. **VHDL code for Full Adder**
12. **PWM Generator in VHDL with Variable Duty Cycle**
13. **VHDL code for ALU**
14. **VHDL code for counters with testbench**
15. **VHDL code for 16-bit ALU**
16. **Shifter Design in VHDL**
17. **Non-linear Lookup Table Implementation in VHDL**
18. **Cryptographic Coprocessor Design in VHDL**
19. **Verilog vs VHDL: Explain by Examples**
20. **VHDL Code for Clock Divider on FPGA**
21. **How to generate a clock enable signal instead of creating another clock domain**
22. **VHDL code for debouncing buttons on FPGA**
23. **VHDL code for Traffic light controller**
24. **VHDL code for a simple 2-bit comparator**
25. **VHDL code for a single-port RAM**
26. **VHDL code for Car Parking System using FSM**
27. **VHDL coding vs Software Programming**
28. **VHDL code for MIPS Processor**
29. **VHDL code for Moore FSM Sequence Detector**
30. **VHDL code for Seven-Segment Display on Basys 3 FPGA**

Facebook Twitter Pinterest

21
SHARES

2 comments:

Unknown May 15, 2020 at 1:04 PM

do you implement 32 bit MIPS with SIMD

Reply

Unknown May 15, 2020 at 1:50 PM

dear i run the whole project, but can't see the same output which you attach at the end. can you share with me further details?? Thanks

Reply

Enter your comment...

Comment as: prof.hosam010 **Sign out**

Publish **Preview** **Notify me**

‹ Home ›

View web version

BECOME A PATRON

Join 18,000+ Followers

Subscribe to More Upcoming FPGA/Verilog/VHDL Projects

Email your email address... Submit