Date :	Design,Simulation,Synthesis and Layout of Pulse Width Modulation Generator
Exp.No :	

AIM:

To design, simulate, synthesis and layout generation of Pulse Width Modulation Generator .

ABSTRACT:

This project involves the design, simulation, synthesis, and layout of an 8-bit Pulse Width Modulation (PWM) generator using Verilog HDL. The PWM signal is generated by comparing an incrementing counter with a programmable duty cycle input. A testbench is created to simulate various duty cycle levels (0% to 100%) and verify output waveform behavior. The design is synthesized for FPGA or ASIC implementation and validated for functionality and timing. This PWM generator is suitable for applications like motor control, LED dimming, and digital signal modulation.

SOFTWARE REQUIRED:

Cadence Incisive Tool.

Cadence Genus Tool.

Cadence Innovus Tool.

METHODOLOGY:

The methodology adopted in this project encompasses multiple steps, each critical in the ASIC digital design flow:

1. Design Specification and Verilog Coding

The PWM generator module is designed using Verilog at the Register Transfer Level (RTL). It generates a PWM signal by comparing an 8-bit counter with a given duty cycle. The output remains high while the counter is less than the duty cycle, enabling control over pulse width.

2. Testbench Creation and Simulation

A testbench is written to verify the functional correctness of the RTL design. The simulation is carried out using Cadence Incisive (NCLaunch) tool. Input stimuli such as reset conditions, alarm set, and time variations are provided. The waveform viewer is used to visualize signal transitions and validate output responses.

3. Synthesis with Genus

The synthesized version of the RTL is generated using Cadence Genus, which converts the behavioral Verilog code into a gate-level netlist. The synthesis step involves reading

the RTL, applying design constraints (timing, power, and area), and generating detailed reports. Reports include timing analysis, area utilization, and power consumption, ensuring the design meets hardware specifications.

4. Physical Design and Layout Generation with Innovus

The final step is to realize the physical structure of the design using Cadence Innovus. The flow includes:

- Floorplanning : Defining the core and IO area**.**
- Power Planning: Adding power and ground rings, stripes, taps, and end caps.
  - Placement: Automatically placing the standard cells and IOs.
  - Clock Tree Synthesis (CTS): Balancing clock paths to meet timing.
  - Routing: Connecting placed cells physically.
  - Reports Generation: Final timing, power, and area reports.
  - Exporting Final Files: Generating GDSII layout, netlist, and Innovus database files.
  - This step ensures that the design is fabrication-ready.

DESIGN AND IMPLEMENTATION

Verilog RTL

PROCEDURE:

Step 1: Right-click on the RedHat window, select "Open Terminal", and create a project folder using mkdir project_name && cd project_name.

Step 2: Then open the design file using gedit project_name.v, write the Verilog code, and save it.

Step 3: Then create and open the testbench file using gedit project_name _tb.v, enter the testbench code, and save it.

Step 4: Start the Cadence tool by running csh and launching NCLaunch using nclaunch - new.

Step 5: Inside NCLaunch, click Multistep → Library → Create cds.lib, save the file, and compile the Verilog design and testbench files.

Step 6: Select the testbench from worklib, elaborate it, and move to the snapshot section.

Step 7: Open the waveform viewer, right-click on the testbench, select Simulation and Run, and observe the waveform.

Step 8: Right-click on the testbench file again, choose Schematic Tracer, and generate the circuit representation.

Step 9: Create a synthesis folder using mkdir logic_synthesis && cd logic_synthesis, then move the design, testbench, constraints, and synthesis script into it.

Step 10: Open Genus Synthesis Tool by running genus, then execute synthesis by typing source rcscript.tcl to generate the netlist, timing, power, and area reports.

Step 11: Analyse the generated reports, document the results, and verify the final netlist for correctness.

S**tep 12:** Open Innovus Tool by running Innovus and type source Default.globals and init_design to start the design.

Step 13: Select **Floorplan → Specify Floorplan** to modify/add concerned values to the above Factors.

Step 14: Under **Connect Global Net Connects**, we create two pins, one for VDD and one for VSS connecting them to corresponding Global Nets as mentioned in Globals file. Select **Power → Connect Global Nets..** to create "Pin" and "Connect to Global Net" as shown and use "Add to list".

Step 15: Select **Power → Power Planning → Add Rings** to add Power rings 'around Core Boundary'.

Similarly, Power Stripes are added using similar content to that of Power Rings.

Step 16: To add End Caps, Select **Place → Physical Cell → Add End Caps** and "Select" the 'FILL' s from the available list. Higher Fills have Higher Widths. To add Well Taps, Select **Place → Physical Cell → Add Well Tap →Select → FillX.**

Step 17: Select **Place → Place Standard Cell → Run Full Placement → Mode → Enable 'Place I/O Pins' → OK → OK .**

Step 18: To generate Timing Report**, Timing → Report Timing → Design Stage – PreCTS → Analysis Type – Setup → OK**

Step 19: To generate Area Report, switch to the Terminal and type the command, **report_area** to see the Cell Count and Area Occupied.

Step 20: To generate Power Report, In the Terminal type the command **report_power** to see the Power Consumption numbers.

Step 21: To optimize the Design, Select **ECO → Optimize Design → Design Stage [PreCTS] →**

Optimization Type – Setup → OK.

Step 22: Saving Design => File →Save Design →Data Type : Innovus → <DesignName>.enc →OK

Step 23: Saving Netlist => File →Save →Netlist →<NetlistName>.v →OK

Step 24: Saving GDS => File →Save →GDS/OASIS →<FileName>.gds →OK

PROGRAM:

Pulse Width Modulation Generator:

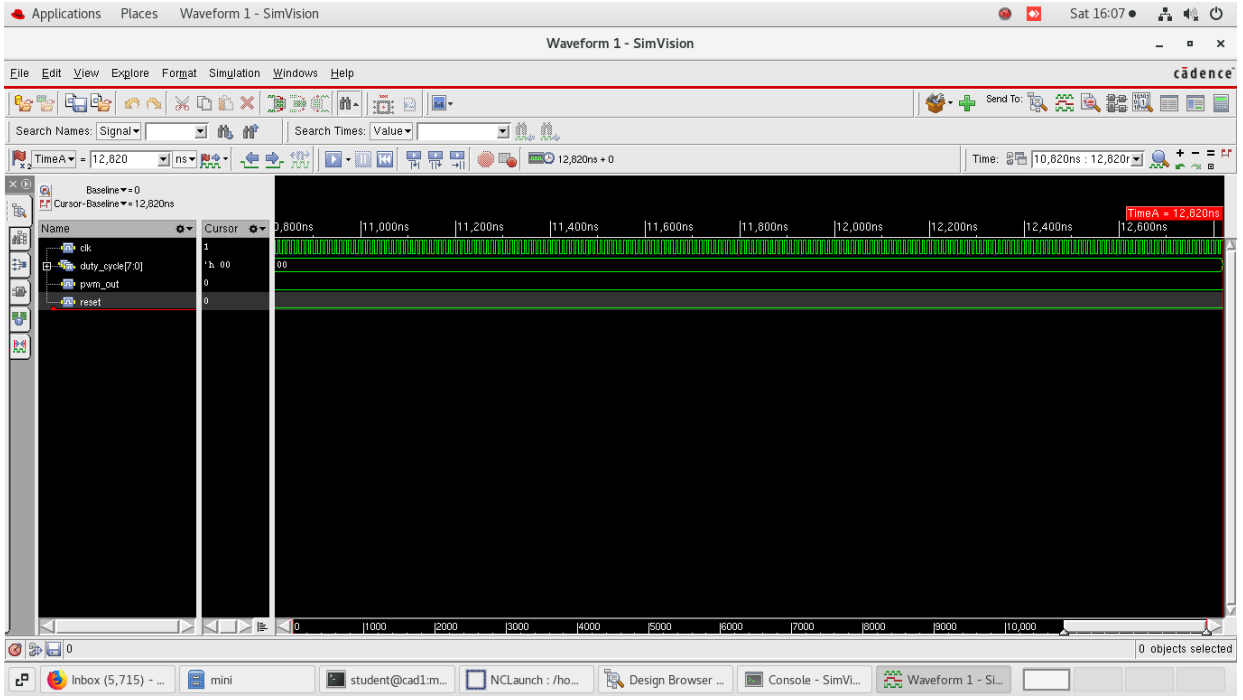
```
      module pwm_generator(
   input clk,
   input reset,
   input [7:0] duty_cycle,
   output reg pwm_out
);
   reg [7:0] counter;
 always @(posedge clk or posedge reset) begin
       if (reset) begin
          counter <= 8'd0;
          pwm_out <= 0;
       end else begin
          counter <= counter + 1;

          if (counter < duty_cycle)
             pwm_out <= 1;
          else
             pwm_out <= 0;
       end
   end
endmodule
```

Testbench code:

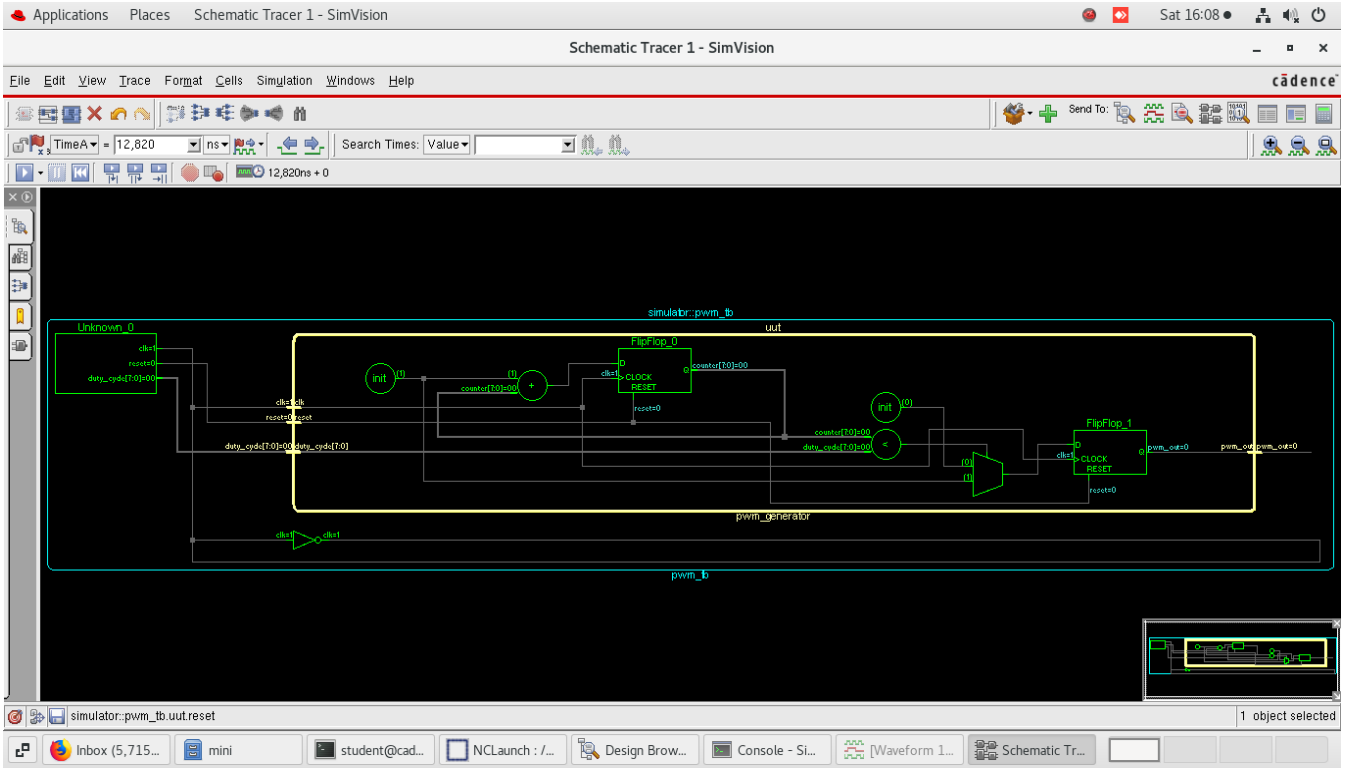
```
module pwm_tb;
   reg clk;
   reg reset;
   reg [7:0] duty_cycle;
   wire pwm_out;
 pwm_generator uut (
       .clk(clk),
       .reset(reset),
       .duty_cycle(duty_cycle),
       .pwm_out(pwm_out)
   );
always #5 clk = ~clk;
 initial begin
       clk = 0;
       reset = 1;
       duty_cycle = 8'd0;
       #20 reset = 0;
       duty_cycle = 8'd64;
       #2560;
       duty_cycle = 8'd128;
       #2560;
       duty_cycle = 8'd192;
       #2560;


       duty_cycle = 8'd255;
       #2560;
       duty_cycle = 8'd0;
       #2560;
       $stop;
   End
Endmodule
```

OUTPUTS :

Waveform :



SCHEMATIC :



RESULTS AND DISCUSSION

Functional Verification:

- The PWM generator design was simulated successfully using the Cadence Incisive Tool.
- Waveform results confirmed correct PWM output, where the high pulse width matched the set duty cycle.
- The testbench validated different duty cycle values (25%, 50%, 75%, 100%) to ensure dynamic PWM control.

Synthesis Results (Cadence Genus):

- Timing Report: All paths met setup and hold requirements with no violations.
- Area Report: Compact logic with minimal area due to simple counter and comparator design.
- Power Report: Low dynamic power consumption suitable for energy-efficient applications.
- Netlist: Gate-level netlist was generated for physical implementation**.**

Physical Design Results (Cadence Innovus):

- Floorplanning: Defined core area and reserved power domains.
- Placement & Routing: Optimized cell placement and complete routing with no DRC violations.
- Clock Tree Synthesis: Balanced clock tree ensured minimal skew across flip-flops**.**
- **Reports:**
  - Timing: Passed both pre and post-CTS timing analysis.
  - Area: Efficient and compact layout achieved.
  - Power: Optimized for low switching activity.
- **Files Generated:**
  - GDSII layout file.
  - Synthesized netlist.
  - Innovus database for future enhancements**.**

CONCLUSION:

This project successfully demonstrates the RTL design, functional simulation, synthesis, and layout generation of a PWM Generator using Cadence tools. The design reliably generates a pulse-width modulated signal based on an 8-bit duty cycle input. All design stages were verified to meet functional and timing requirements, with optimized area and power usage. The project provided valuable practical exposure to the complete digital design flow, paving the way for more advanced VLSI and SoC development projects.

Netlist:

```
// Generated by Cadence Genus(TM) Synthesis Solution 20.11-s111_1
// Generated on: May  3 2025 16:11:30 IST (May  3 2025 10:41:30 UTC)
// Verification Directory fv/pwm_generator
module pwm_generator(clk, reset, duty_cycle, pwm_out);
input clk, reset;
input [7:0] duty_cycle;
output pwm_out;
wire clk, reset;
wire [7:0] duty_cycle;
wire pwm_out;
wire [7:0] counter;
wire n_0, n_1, n_2, n_3, n_4, n_5, n_6, n_7;
wire n_8, n_9, n_10, n_11, n_12, n_13, n_14, n_15;
wire n_16, n_17, n_18, n_19, n_20, n_21, n_22, n_23;
wire n_24, n_25, n_26, n_27, n_28, n_29, n_30, n_31;
wire n_32, n_33, n_34;
SDFFRHQX1 \counter_reg[7] (.RN (n_34), .CK (clk), .D (n_21), .SI
    (counter[7]), .SE (n_32), .Q (counter[7]));
DFFRHQX1 \counter_reg[6] (.RN (n_34), .CK (clk), .D (n_33), .Q
    (counter[6]));
OA21XL g733__2398(.A0 (counter[6]), .A1 (n_31), .B0 (n_32), .Y
    (n_33));
DFFRHQX2 pwm_out_reg(.RN (n_34), .CK (clk), .D (n_29), .Q (pwm_out));
NAND2XL g735__5107(.A (counter[6]), .B (n_31), .Y (n_32));
AOI2BB1X1 g736__6260(.A0N (counter[5]), .A1N (n_27), .B0 (n_31), .Y
    (n_30));
AOI221X1 g746__4319(.A0 (n_6), .A1 (n_18), .B0 (n_20), .B1 (n_26),
    .C0 (n_22), .Y (n_29));
AOI21XL g739__8428(.A0 (n_25), .A1 (n_24), .B0 (n_27), .Y (n_28));
AND2X1 g738__5526(.A (counter[5]), .B (n_27), .Y (n_31));
AOI221X1 g750__6783(.A0 (duty_cycle[4]), .A1 (n_25), .B0
    (duty_cycle[3]), .B1 (n_19), .C0 (n_17), .Y (n_26));
NOR2XL g741__3680(.A (n_25), .B (n_24), .Y (n_27));
OA21XL g742__1617(.A0 (counter[3]), .A1 (n_13), .B0 (n_24), .Y
    (n_23));
OAI21X1 g754__2802(.A0 (duty_cycle[7]), .A1 (n_21), .B0 (n_16), .Y
  (n_22));
OAI222XL g751__1705(.A0 (duty_cycle[2]), .A1 (n_14), .B0 (n_7), .B1
    (n_10), .C0 (n_19), .C1 (duty_cycle[3]), .Y (n_20));
INVXL g752(.A (n_17), .Y (n_18));
NAND3XL g756__5122(.A (n_12), .B (n_11), .C (counter[6]), .Y (n_16));
AOI21X1 g747__8246(.A0 (n_14), .A1 (n_9), .B0 (n_13), .Y (n_15));
NAND2XL g745__7098(.A (counter[3]), .B (n_13), .Y (n_24));
OAI221X1 g753__6131(.A0 (n_12), .A1 (counter[6]), .B0 (n_0), .B1
```

```
    (counter[5]), .C0 (n_11), .Y (n_17));
  AOI211X1 g759__1881(.A0 (n_4), .A1 (counter[1]), .B0 (n_5), .C0
    (counter[0]), .Y (n_10));
  NOR2XL g748__5115(.A (n_14), .B (n_9), .Y (n_13));
  OA21XL g755__7482(.A0 (counter[0]), .A1 (counter[1]), .B0 (n_9), .Y (n_8));
  AO22XL g757__4733(.A0 (duty_cycle[2]), .A1 (n_14), .B0     (duty_cycle[1]), .B1 (n_3),
.Y (n_7));
  OAI22X1 g758__6161(.A0 (duty_cycle[4]), .A1 (n_25), .B0
    (duty_cycle[5]), .B1 (n_2), .Y (n_6));

  NAND2XL g761__9315(.A (duty_cycle[7]), .B (n_21), .Y (n_11));
  NAND2XL g762__9945(.A (counter[0]), .B (counter[1]), .Y (n_9));
  INVXL g769(.A (duty_cycle[0]), .Y (n_5));
  INVXL g770(.A (duty_cycle[1]), .Y (n_4));
  INVXL g771(.A (duty_cycle[6]), .Y (n_12));
  INVXL g768(.A (reset), .Y (n_34));
  INVXL g763(.A (duty_cycle[5]), .Y (n_0));
  CLKINVX1 g767(.A (counter[7]), .Y (n_21));
  DFFRX1 \counter_reg[1] (.RN (n_34), .CK (clk), .D (n_8), .Q
    (counter[1]), .QN (n_3));
  DFFRX1 \counter_reg[3] (.RN (n_34), .CK (clk), .D (n_23), .Q
    (counter[3]), .QN (n_19));
  DFFRX1 \counter_reg[5] (.RN (n_34), .CK (clk), .D (n_30), .Q
    (counter[5]), .QN (n_2));
  DFFRX1 \counter_reg[0] (.RN (n_34), .CK (clk), .D (n_1), .Q
    (counter[0]), .QN (n_1));
  DFFRX1 \counter_reg[2] (.RN (n_34), .CK (clk), .D (n_15), .Q
    (counter[2]), .QN (n_14));
  DFFRX1 \counter_reg[4] (.RN (n_34), .CK (clk), .D (n_28), .Q
    (counter[4]), .QN (n_25));
endmodule
```

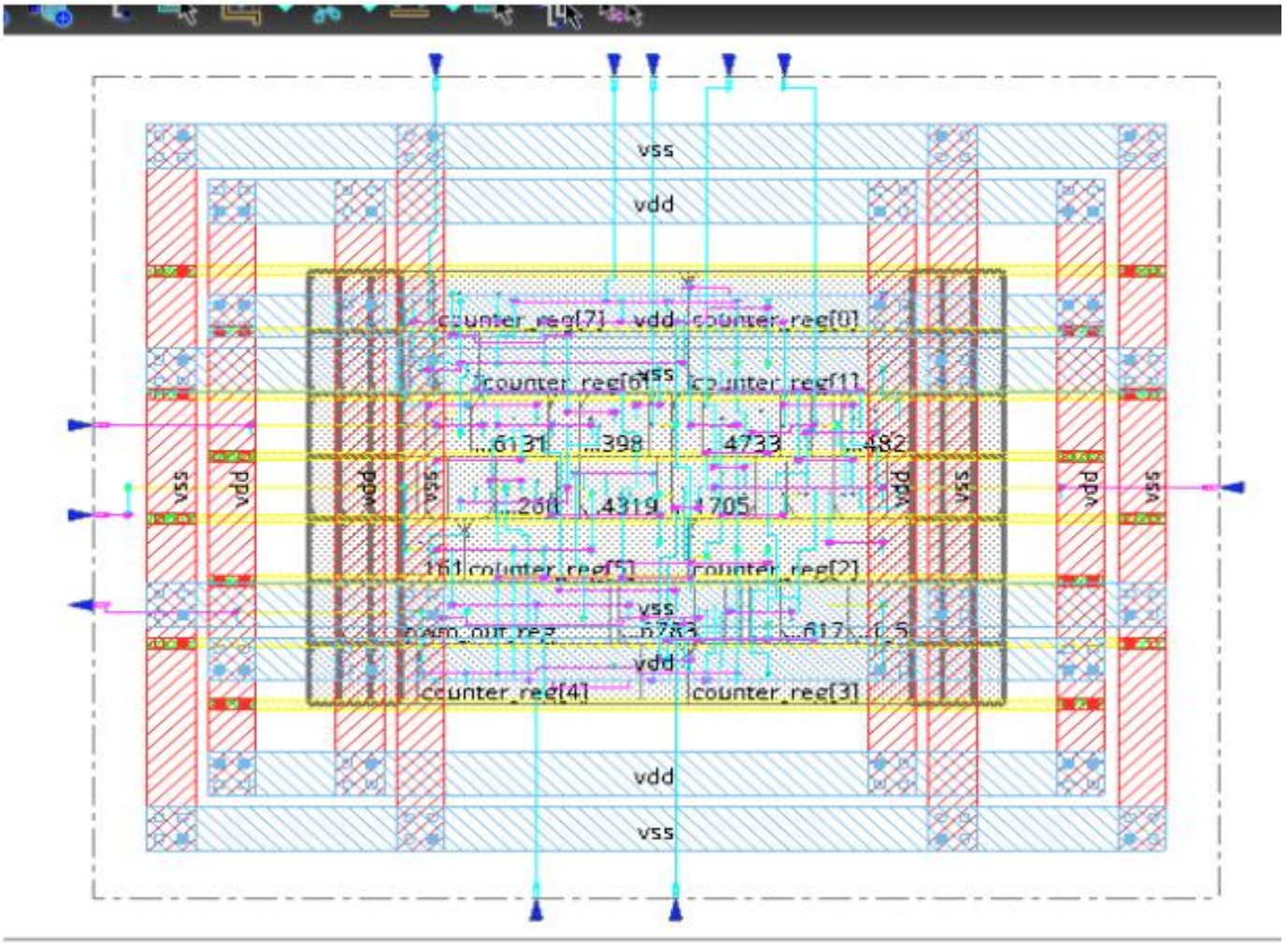
AREA REPORT:

```
====================================================================
  Generated by:           Genus(TM) Synthesis Solution 20.11-s111_1
  Generated on:           May 03 2025  04:11:30 pm
  Module:                 pwm_generator
  Technology library:     slow
  Operating conditions:   slow (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
====================================================================

   Instance    Module  Cell Count  Cell Area  Net Area   Total Area   Wireload
-------------------------------------------------------------------------------
pwm_generator                 38    331.522     0.000      331.522 <none> (D)
  (D) = wireload is default in technology library
```

POWER REPORT:

```
Instance: /pwm_generator
Power Unit: W
PDB Frames: /stim#0/frame#0
  ------------------------------------------------------------------------
    Category         Leakage     Internal    Switching        Total    Row%
  ------------------------------------------------------------------------
      memory     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
    register     1.12940e-06  8.63742e-05  1.33749e-05  1.00878e-04  82.48%
       latch     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
       logic     4.66342e-07  1.03961e-05  4.49658e-06  1.53591e-05  12.56%
        bbox     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
       clock     0.00000e+00  0.00000e+00  6.07500e-06  6.07500e-06   4.97%
         pad     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
          pm     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00   0.00%
  ------------------------------------------------------------------------
    Subtotal     1.59574e-06  9.67703e-05  2.39465e-05  1.22313e-04 100.01%
  Percentage           1.30%       79.12%       19.58%      100.00% 100.00%
  ------------------------------------------------------------------------
```

LAYOUT :


vss
vdd
counter_reg[7]
vdd
counter_reg[0]
counter_reg[6]
vss
counter_reg[1]
...6131
...398
4733
...482
...2618
4319
705
counter_reg[5]
counter_reg[2]
vss
vdd
counter_reg[4]
counter_reg[3]
vdd
vss
vss
ppa
ppa
vss
vdd
vss
ppa
vss

Marks assigned	Excellent (16-20)	Good (15-10)	Satisfactory (<10)	
Conduct of Experiment(20)				
Analyze the problem and develop programming constructs (15)				
Completeness of the experiment (5)				
Observation /Record (30)				
Interpretation of the findings (15)				
Simulation and Synthesis(5)				
Adherence to record submission deadline (5)				
Presentation and completion of record (5)				
VIVA (10)				
Ability to recall the theoretical concepts				
Total				

RESULT:

Thus, the Design, simulation, synthesis and layout generation of Pulse Width Modulation Generation were done successfully.