ECE 5680
Computer-Aided Logical Design and FPGAs
Winter 2021
Project

WAYNE STATE UNIVERSITY

Mostafa AlNaimi
HB5193
04/10/2021

Contents

References

Ref	Document Reference	Owner	Title	Issue/Date
1	https://www.fpga4student.com/2017/08/verilog-code-for-pwm-generator.html	fpga4student	Verilog code for PWM generator	August 2018
2	https://circuitdigest.com/tutorial/what-is-pwm-pulse-width-modulation	Circuit Digest	What is PWM: Pulse Width Modulation	Sep. 2018
3	https://www.allaboutcircuits.com/textbook/semiconductors/chpt-11/pulse-width-modulation/#:~:text=Being%20able%20to%20vary%20their,low%20RPM%20than%20linear%20methods.	All About Circuits	Pulse Width Modulation	Oct. 2020

1. Introduction

In this project, we are implementing a Verilog code on the BASYS 3 FPGA board that will generate a PWM (Pulse Width Modulation) signal with variable duty cycle. The PWM generator will create 10 MHz signal controlled by two debounced buttons on the BASYS 3 board to control the duty cycle of the PWM signal. The first push button will increase the duty cycle by 10% and the second push button will decrease the duty cycle by 10%.

2. Design Overview

PWM stands for Pulse Width Modulation. A PWM signal stays on for a particular time and then stays off for the rest of the period. What makes this PWM signal special and more useful is that we can set for how long it should stay on by controlling the duty cycle of the PWM signal.

The percentage of time in which the PWM signal remains HIGH (on time) is called as duty cycle. If the signal is always ON it is in 100% duty cycle and if it is always off it is 0% duty cycle. The formulae to calculate the duty cycle is shown below.

Duty Cycle =Turn ON time/ (Turn ON time + Turn OFF time)

The below figure represents a PWM signal with 50% duty cycle. As you can see, considering an entire time (on time + off time) the PWM signal stays on only for 50% of the time period.

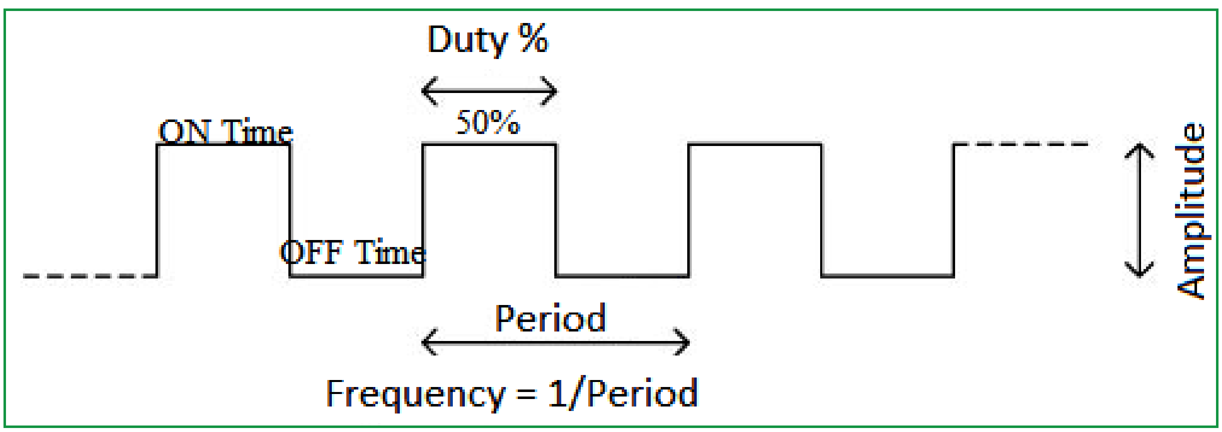


Figure 1: PWM signal with 50% duty cycle

By controlling the Duty cycle from 0% to 100% we can control the “on time” of PWM signal and thus the width of signal. Since we can modulate the width of the pulse, it got its iconic name “Pulse width Modulation”.

The frequency of a PWM signal determines how fast a PWM completes one period. One Period is the complete ON and OFF time of a PWM signal as shown in the above figure.

3. Design Interface

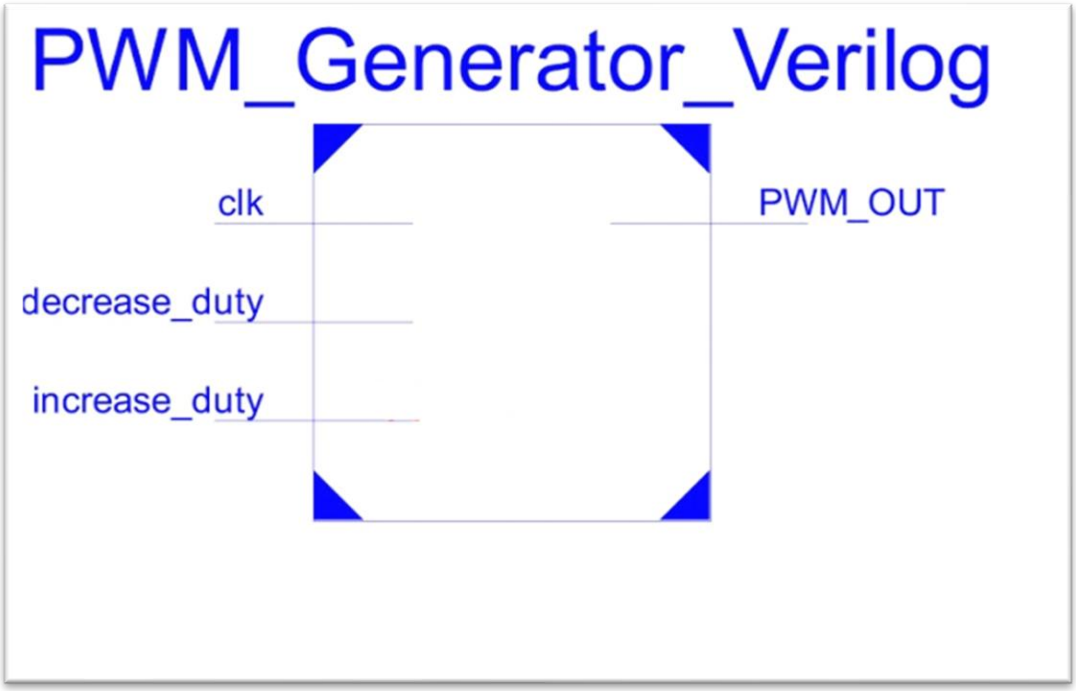


Figure 2: High Level Design interface of the PWM module

4. Verilog Code, Testbench & Constraint file:

The Verilog code for PWM generator with variable duty cycle is taken from the FPGA4student website.

Verilog Code:

```
// fpga4student.com: FPGA Projects, Verilog projects, VHDL projects
// Verilog project: Verilog code for PWM Generator with variable Duty
Cycle
// Two debounced buttons are used to control the duty cycle (step
size: 10%)
module PWM_Generator_Verilog
 (
 clk, // 100MHz clock input
 increase_duty, // input to increase 10% duty cycle
 decrease_duty, // input to decrease 10% duty cycle
 PWM_OUT // 10MHz PWM output signal
    );
 input clk;
 input increase_duty;
 input decrease_duty;
 output PWM_OUT;
 wire slow_clk_enable; // slow clock enable signal for debouncing FFs
```

```
 reg[27:0] counter_debounce=0;// counter for creating slow clock
enable signals
 wire tmp1,tmp2,duty_inc;// temporary flip-flop signals for debouncing
the increasing button
 wire tmp3,tmp4,duty_dec;// temporary flip-flop signals for debouncing
the decreasing button
 reg[3:0] counter_PWM=0;// counter for creating 10Mhz PWM signal
 reg[3:0] DUTY_CYCLE=5; // initial duty cycle is 50%
  // Debouncing 2 buttons for inc/dec duty cycle
  // Firstly generate slow clock enable for debouncing flip-flop (4Hz)
 always @(posedge clk)
 begin
   counter_debounce <= counter_debounce + 1;
   //if(counter_debounce>=25000000) then
   // for running on FPGA -- comment when running simulation
   if(counter_debounce>=1)
   // for running simulation -- comment when running on FPGA
    counter_debounce <= 0;
 end
 // assign slow_clk_enable = counter_debounce == 25000000 ?1:0;
 // for running on FPGA -- comment when running simulation
 assign slow_clk_enable = counter_debounce == 1 ?1:0;
 // for running simulation -- comment when running on FPGA
 // debouncing FFs for increasing button
 DFF_PWM PWM_DFF1(clk,slow_clk_enable,increase_duty,tmp1);
 DFF_PWM PWM_DFF2(clk,slow_clk_enable,tmp1, tmp2);
 assign duty_inc =  tmp1 & (~ tmp2) & slow_clk_enable;
 // debouncing FFs for decreasing button
 DFF_PWM PWM_DFF3(clk,slow_clk_enable,decrease_duty, tmp3);
 DFF_PWM PWM_DFF4(clk,slow_clk_enable,tmp3, tmp4);
 assign duty_dec =  tmp3 & (~ tmp4) & slow_clk_enable;
 // vary the duty cycle using the debounced buttons above
 always @(posedge clk)
 begin
   if(duty_inc==1 && DUTY_CYCLE <= 9)
    DUTY_CYCLE <= DUTY_CYCLE + 1;// increase duty cycle by 10%
   else if(duty_dec==1 && DUTY_CYCLE>=1)
    DUTY_CYCLE <= DUTY_CYCLE - 1;//decrease duty cycle by 10%
 end
// Create 10MHz PWM signal with variable duty cycle controlled by 2
buttons
 always @(posedge clk)
 begin
   counter_PWM <= counter_PWM + 1;
   if(counter_PWM>=9)
    counter_PWM <= 0;
 end
 assign PWM_OUT = counter_PWM < DUTY_CYCLE ? 1:0;
endmodule
// Debouncing DFFs for push buttons on FPGA
module DFF_PWM(clk,en,D,Q);
input clk,en,D;
```

```
output reg Q;
always @(posedge clk)
begin
 if(en==1) // slow clock enable signal
  Q <= D;
end
endmodule
```

Test Bench:

```
`timescale 1ns / 1ps
// fpga4student.com: FPGA Projects, Verilog projects, VHDL projects
// Verilog project: Verilog testbench code for PWM Generator with variable duty cycle
module tb_PWM_Generator_Verilog;
 // Inputs
 reg clk;
 reg increase_duty;
 reg decrease_duty;
 // Outputs
 wire PWM_OUT;
 // Instantiate the PWM Generator with variable duty cycle in Verilog
 PWM_Generator_Verilog PWM_Generator_Unit(
  .clk(clk),
  .increase_duty(increase_duty),
  .decrease_duty(decrease_duty),
  .PWM_OUT(PWM_OUT)
 );
 // Create 100Mhz clock
 initial begin
 clk = 0;
 forever #5 clk = ~clk;
 end
 initial begin
  increase_duty = 0;
  decrease_duty = 0;
  #100;
    increase_duty = 1;
```

```
 #100;// increase duty cycle by 10%
  increase_duty = 0;
 #100;
  increase_duty = 1;
 #100;// increase duty cycle by 10%
  increase_duty = 0;
 #100;
  increase_duty = 1;
 #100;// increase duty cycle by 10%
  increase_duty = 0;
 #100;
  decrease_duty = 1;
 #100;//decrease duty cycle by 10%
  decrease_duty = 0;
 #100;
  decrease_duty = 1;
 #100;//decrease duty cycle by 10%
  decrease_duty = 0;
 #100;
  decrease_duty = 1;
 #100;//decrease duty cycle by 10%
  decrease_duty = 0;
 end
endmodule
```

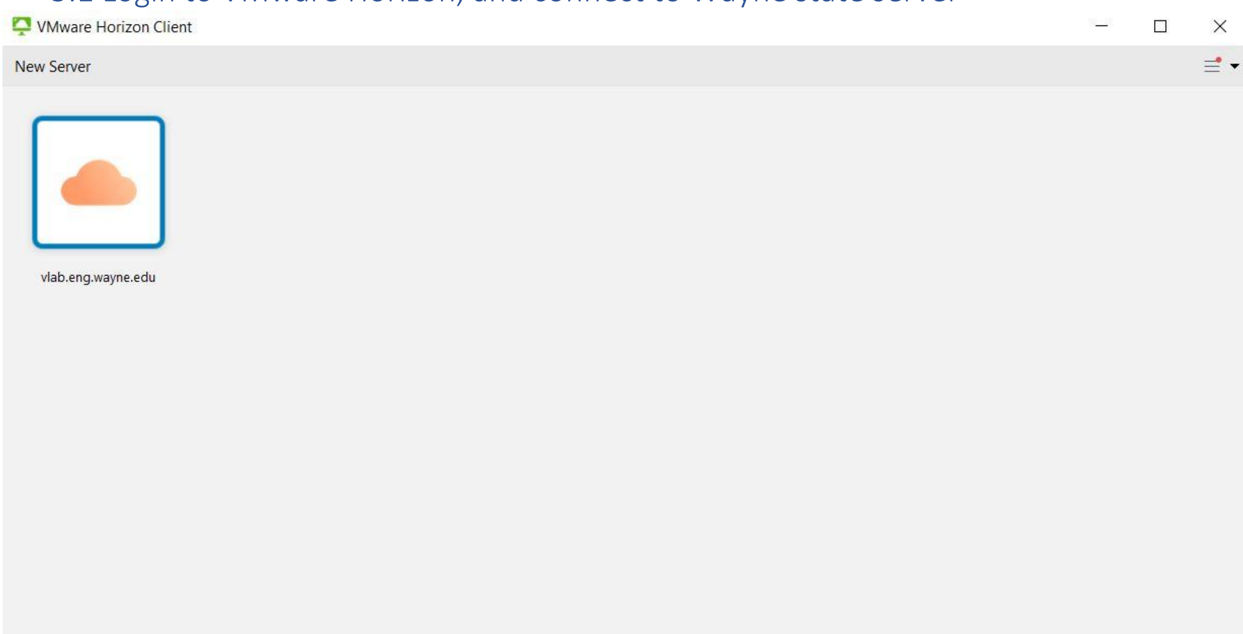
Constraint:

```
set_property direction OUT [get_ports {PWM_OUT}]
set_property direction IN [get_ports {clk}]
set_property direction IN [get_ports {decrease_duty}]
set_property direction IN [get_ports {increase_duty}]
set_property IOSTANDARD LVCMOS18 [get_ports clk]
set_property IOSTANDARD LVCMOS18 [get_ports increase_duty]
set_property IOSTANDARD LVCMOS18 [get_ports decrease_duty]
set_property IOSTANDARD LVCMOS18 [get_ports PWM_OUT]
```

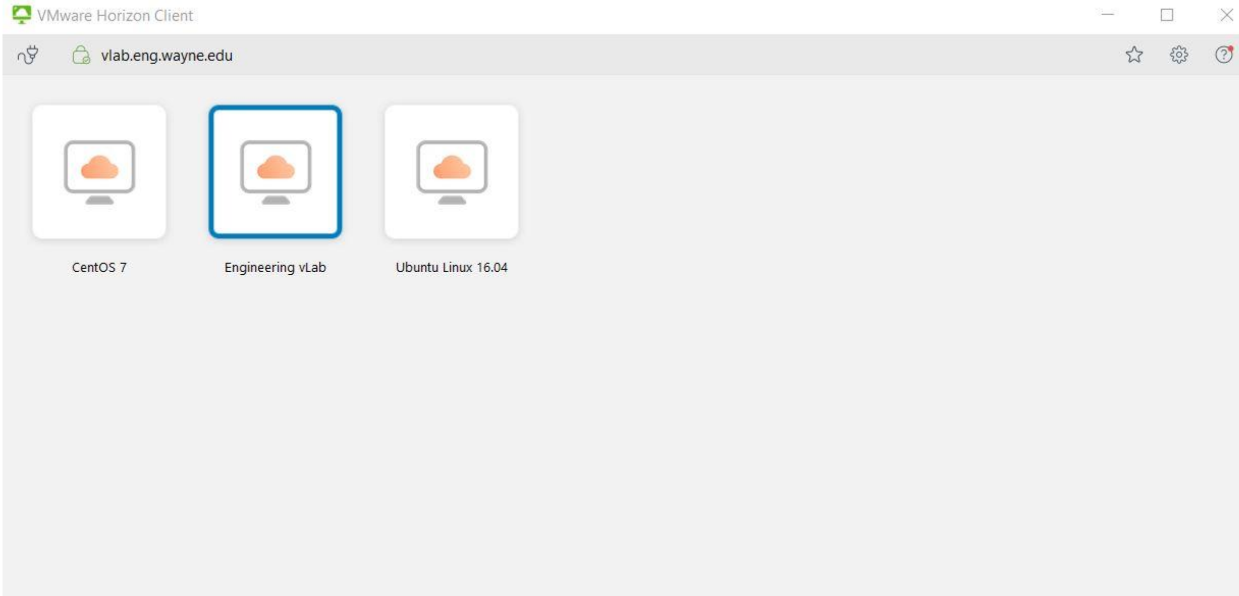
```
set_property DRIVE 12 [get_ports PWM_OUT]
set_property SLEW SLOW [get_ports PWM_OUT]
set_property PACKAGE_PIN W5 [get_ports clk]
set_property PACKAGE_PIN W19 [get_ports increase_duty]
set_property PACKAGE_PIN T17 [get_ports decrease_duty]
set_property PACKAGE_PIN E19 [get_ports PWM_OUT]

#revert back to original instance
current_instance -quiet
```

5. Lab Procedure (Vivado on VM)

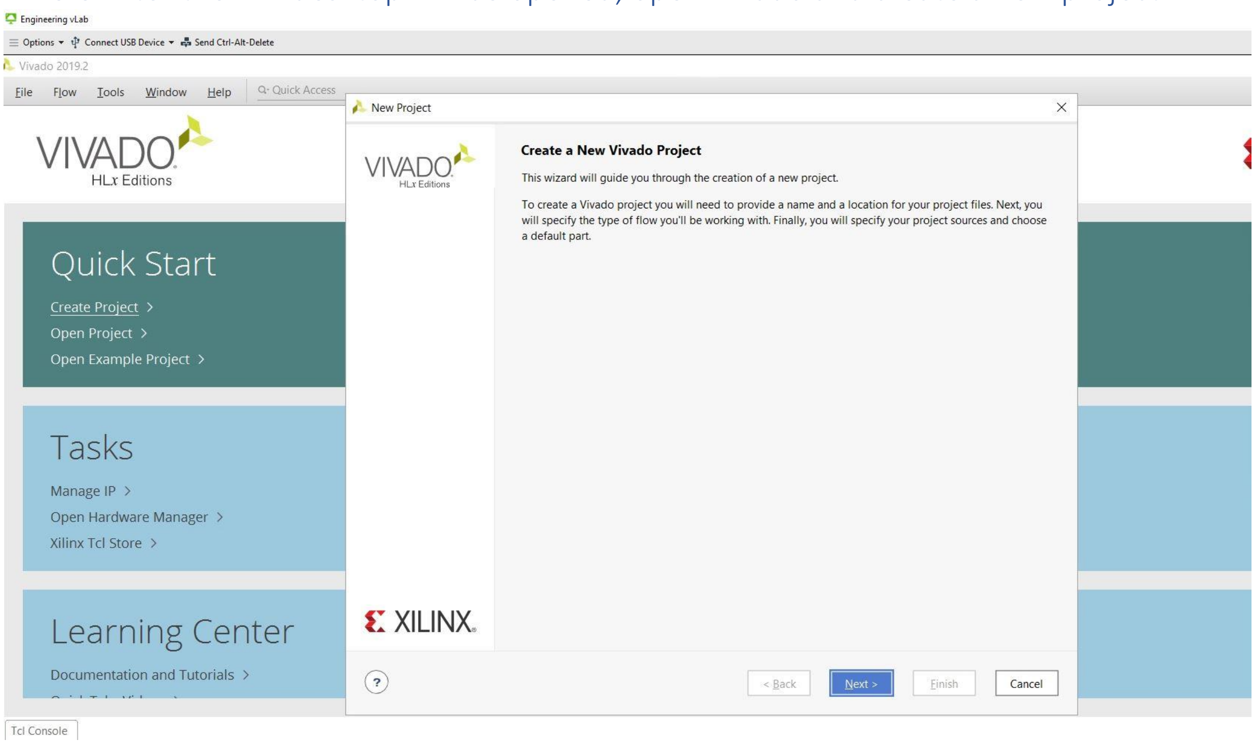
5.1 Login to VMware Horizon, and connect to Wayne state server



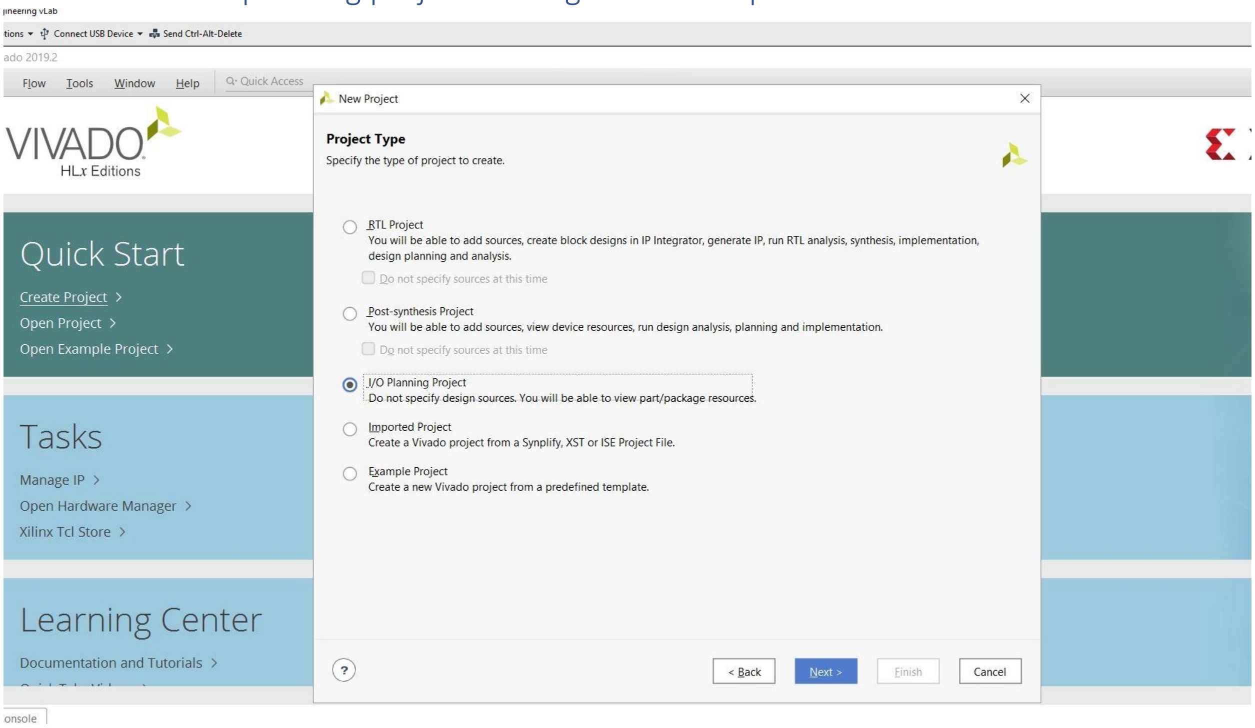
5.2 Go to Engineering vLab



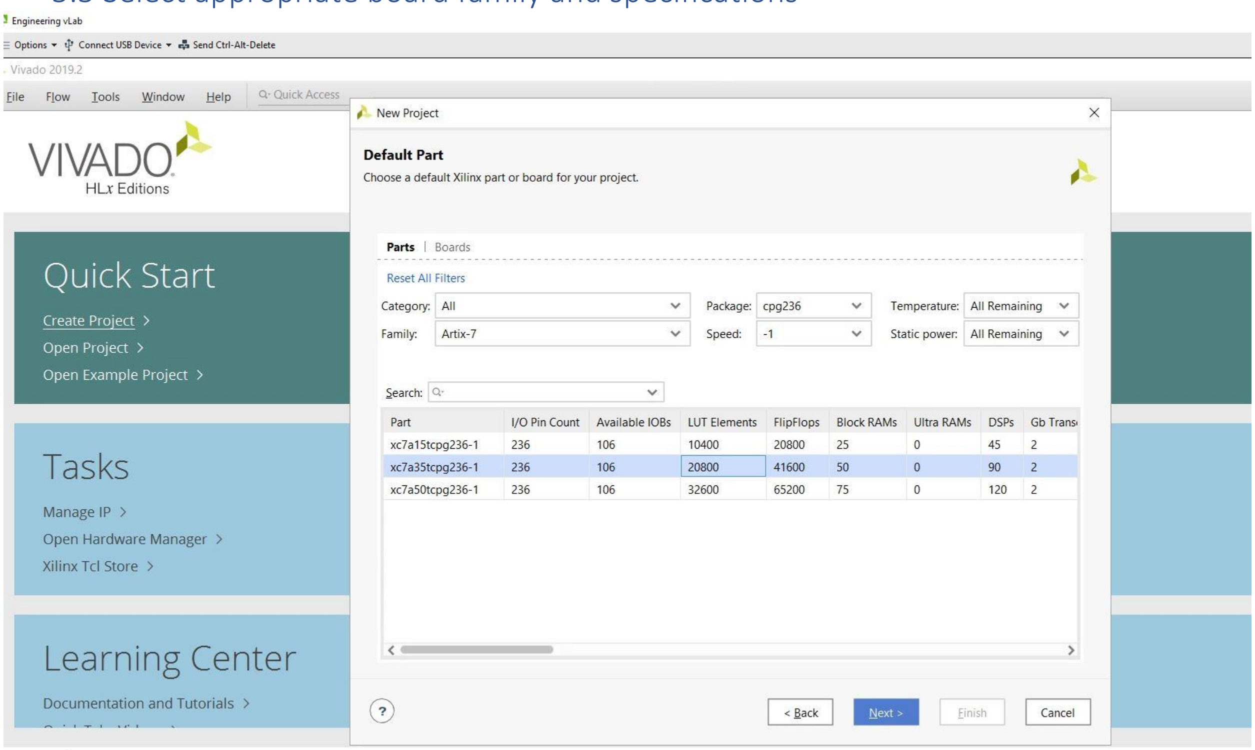
5.3 After the VM desktop will be opened, open Vivado and create a new project



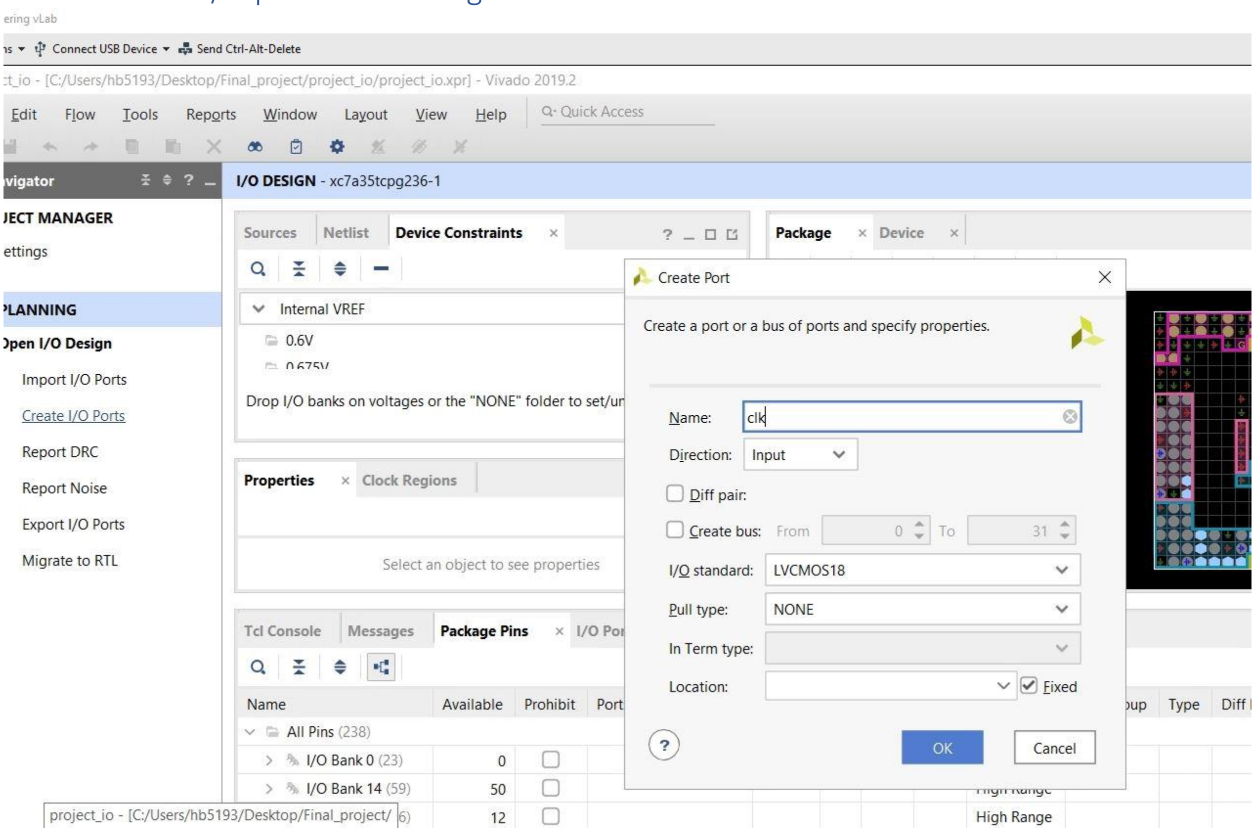
5.4 Create IO planning project to assign hardware ports



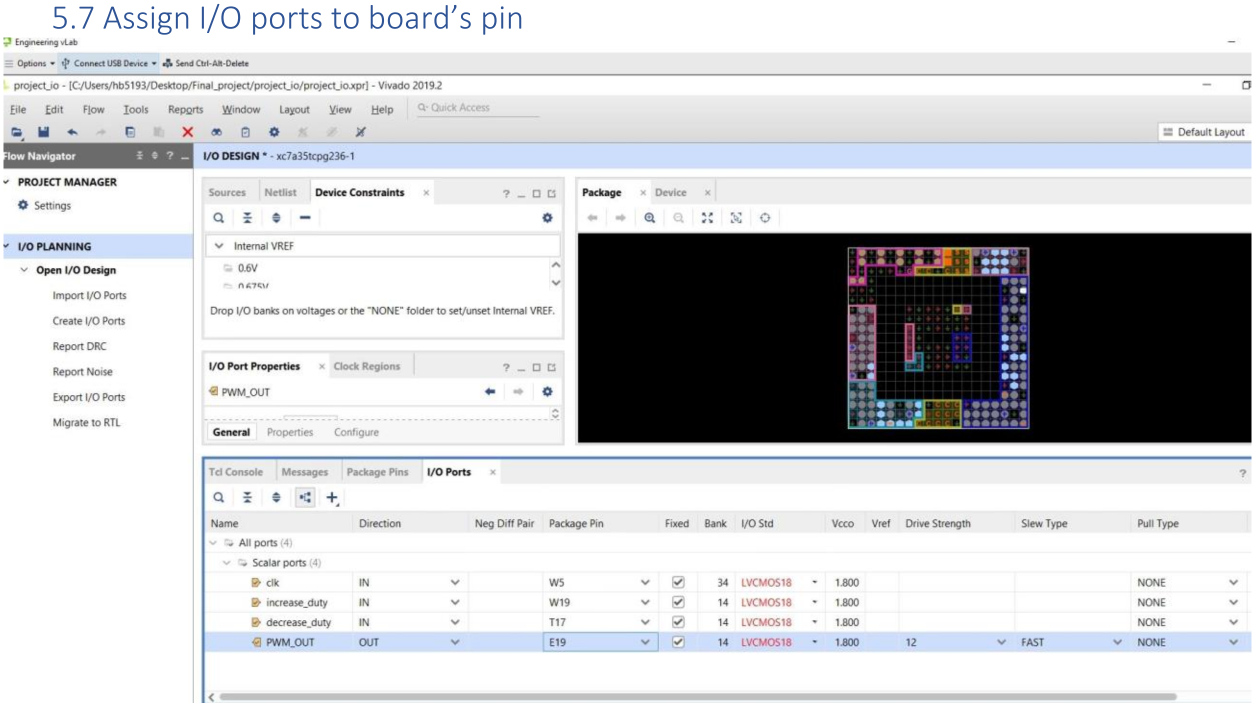
5.5 Select appropriate board family and specifications



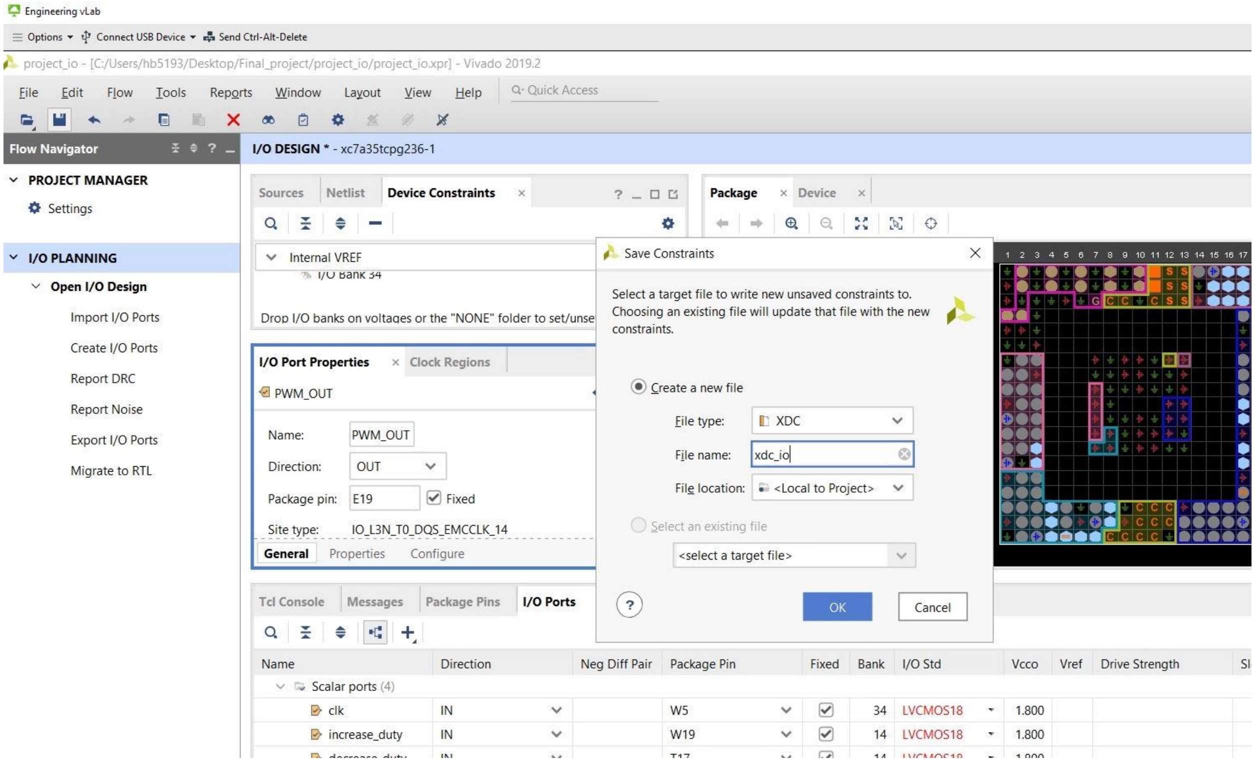
5.6 Create I/O ports to be assigned to the board



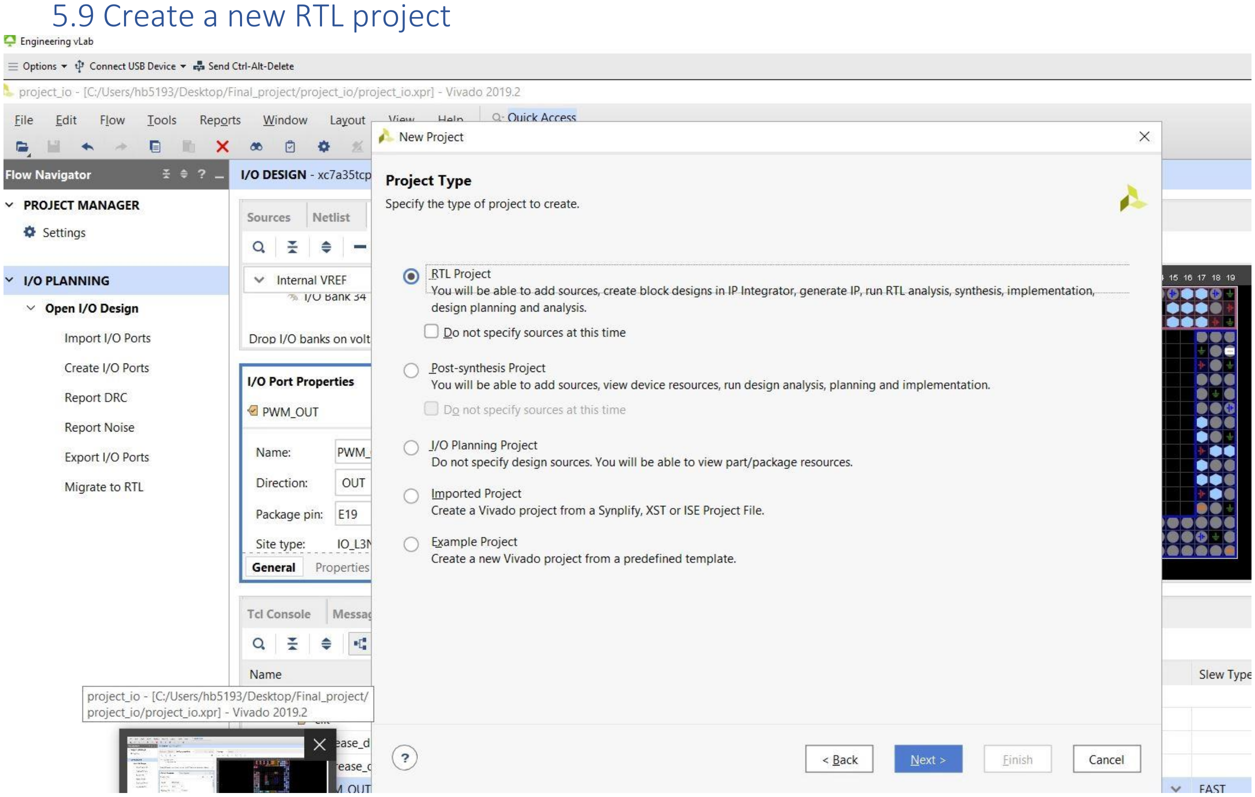
5.7 Assign I/O ports to board's pin



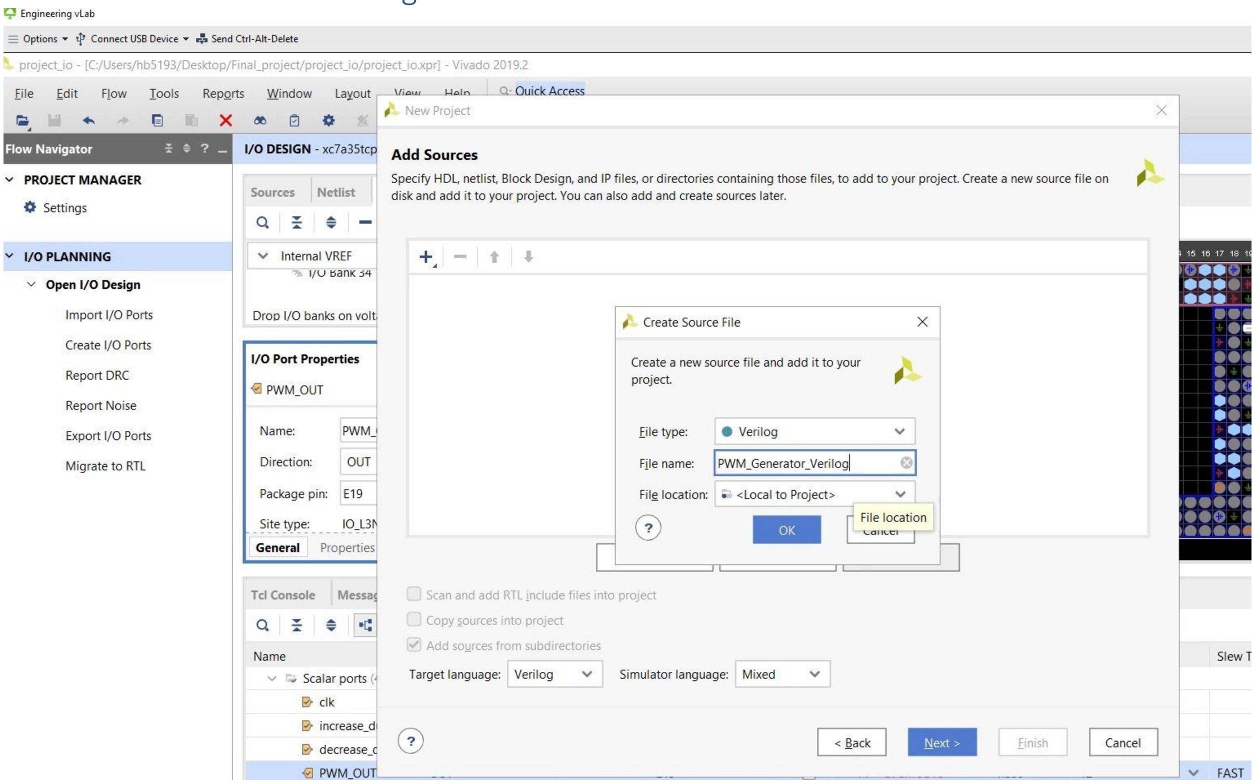
5.8 Save the constraint file to be used in the Verilog project



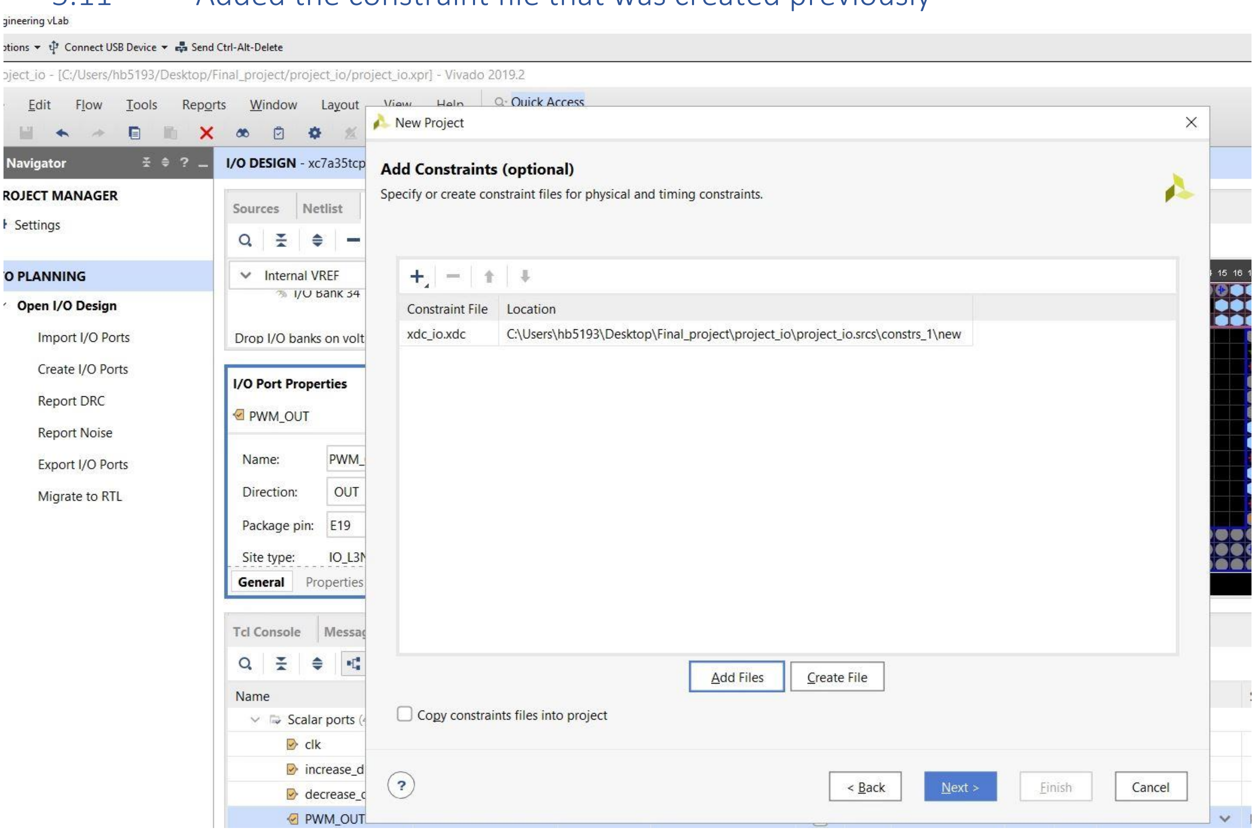
5.9 Create a new RTL project



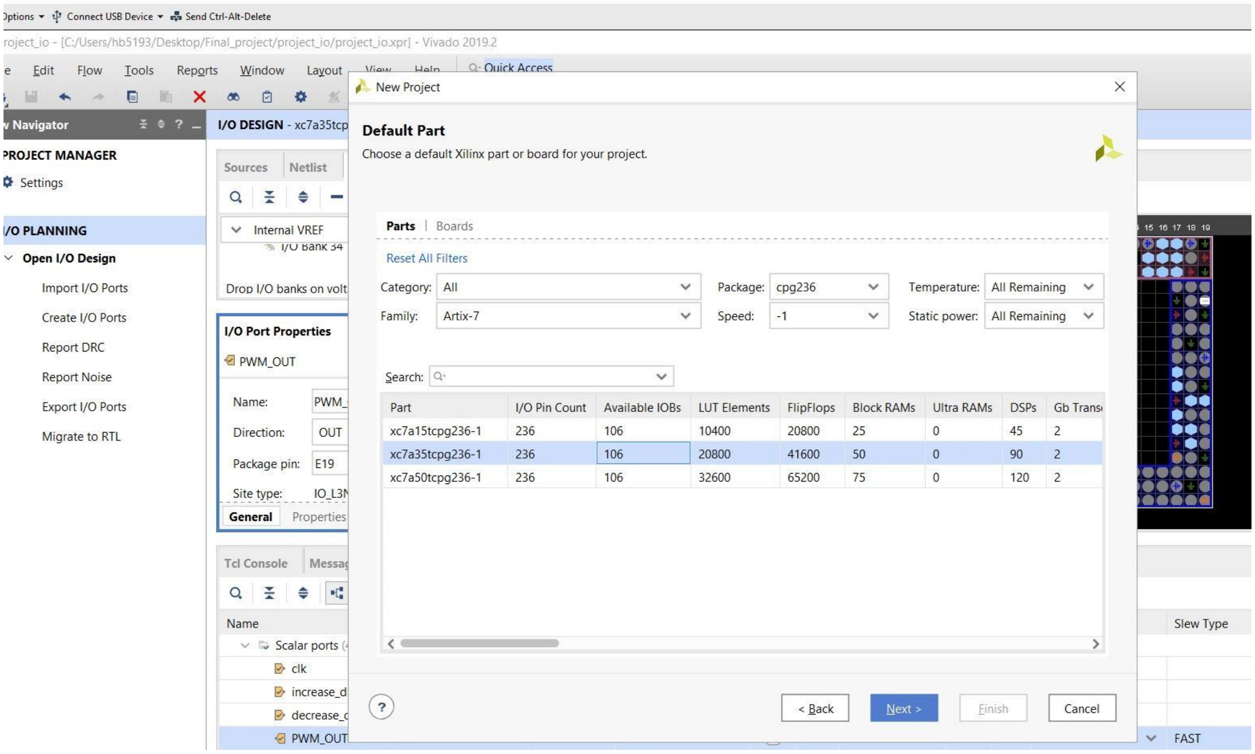
5.10 Create a Verilog module source file



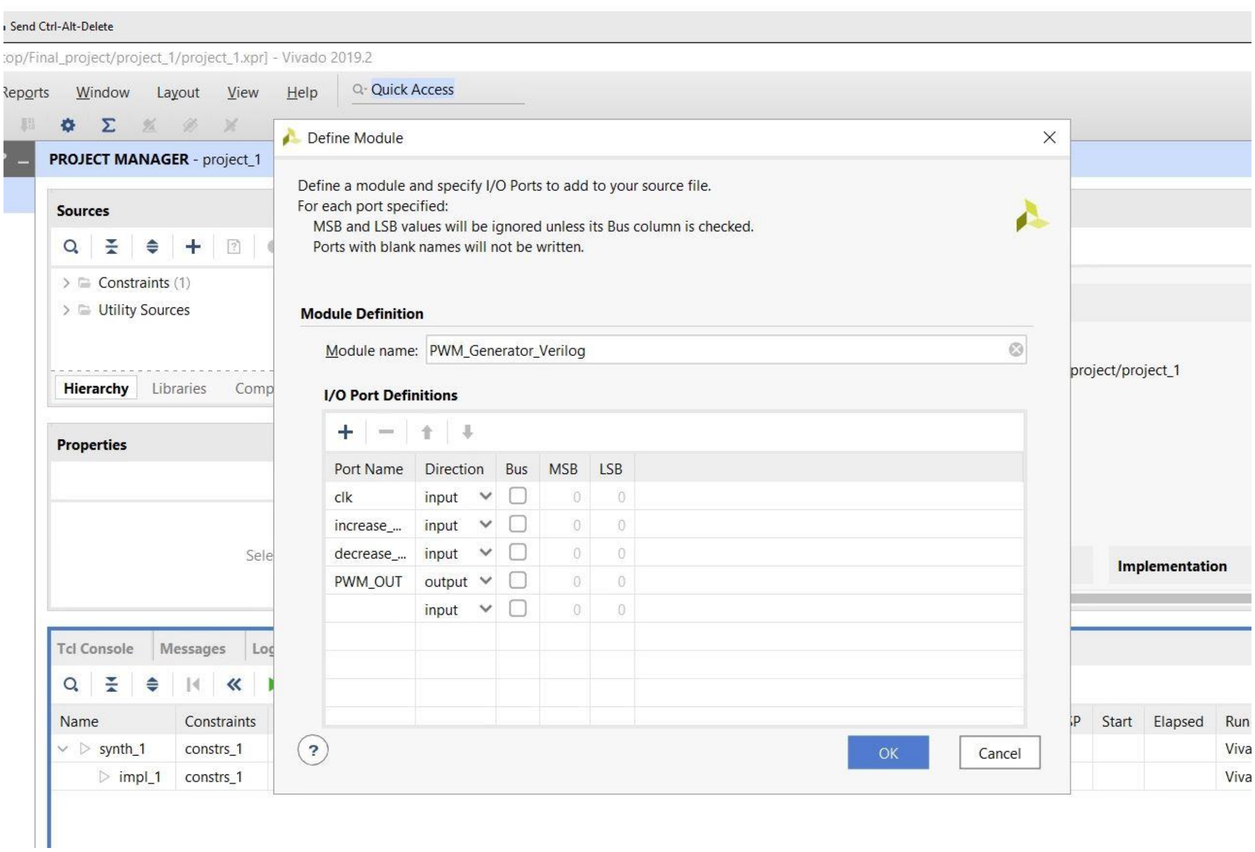
5.11 Added the constraint file that was created previously



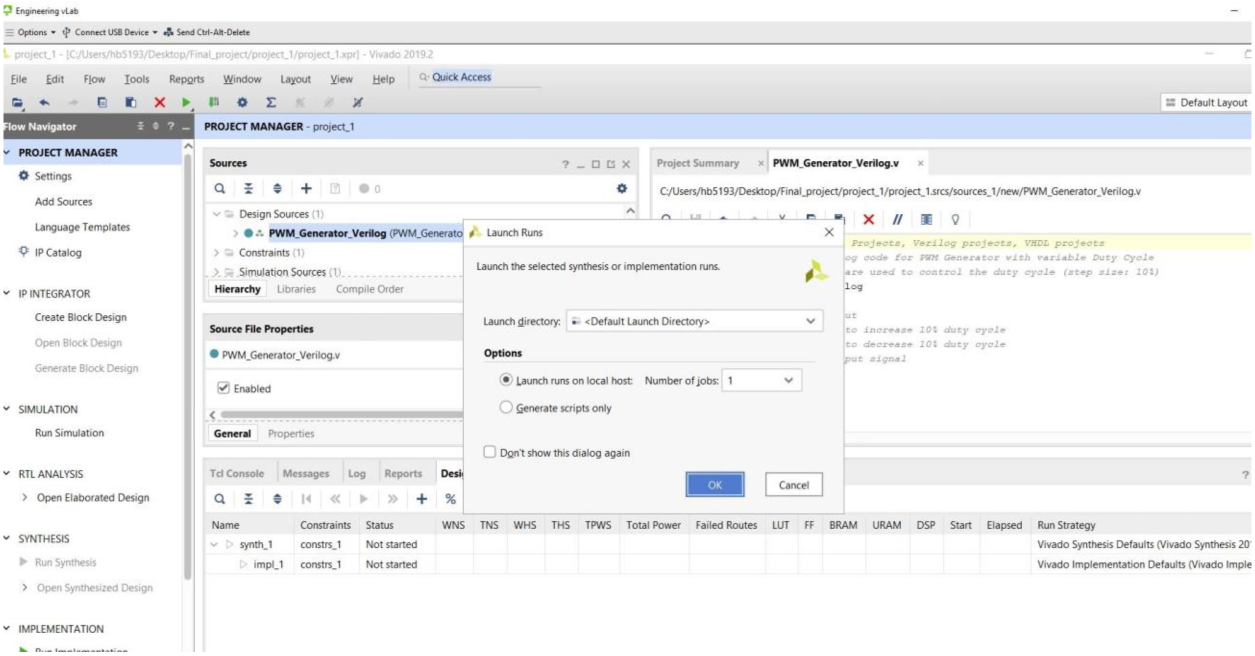
5.12 Choose the board family and specifications



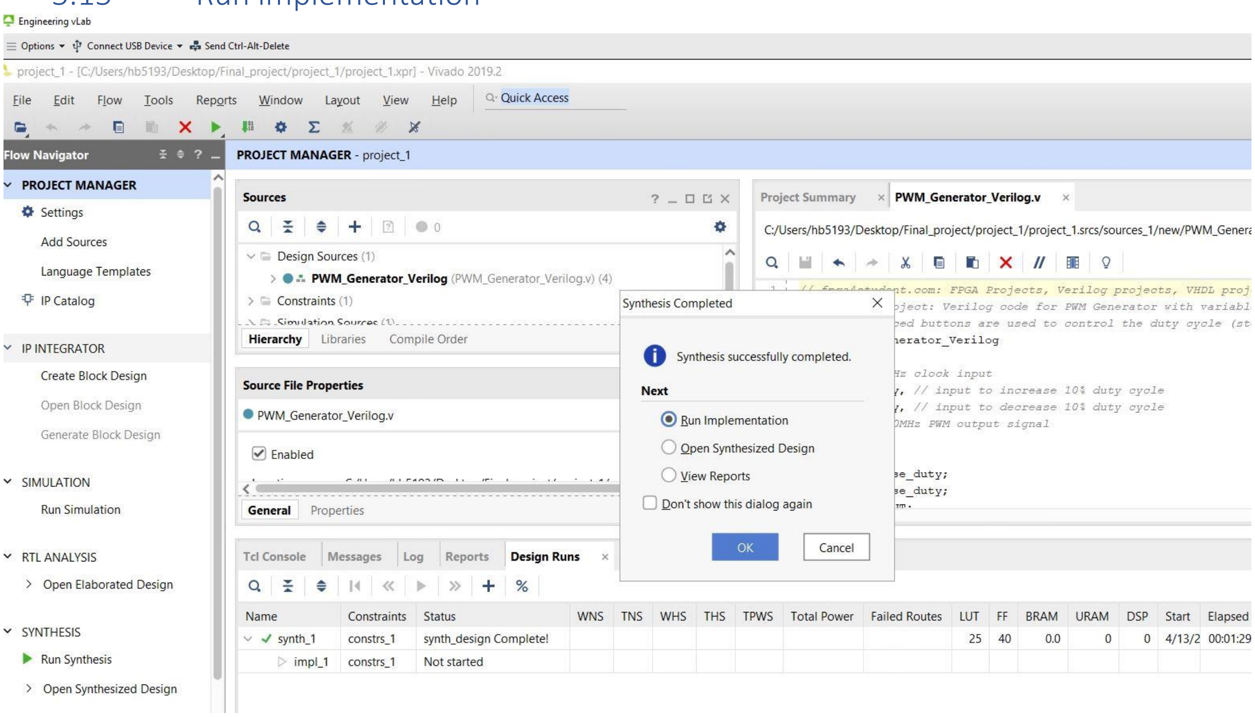
5.13 Create the high level module I/O ports



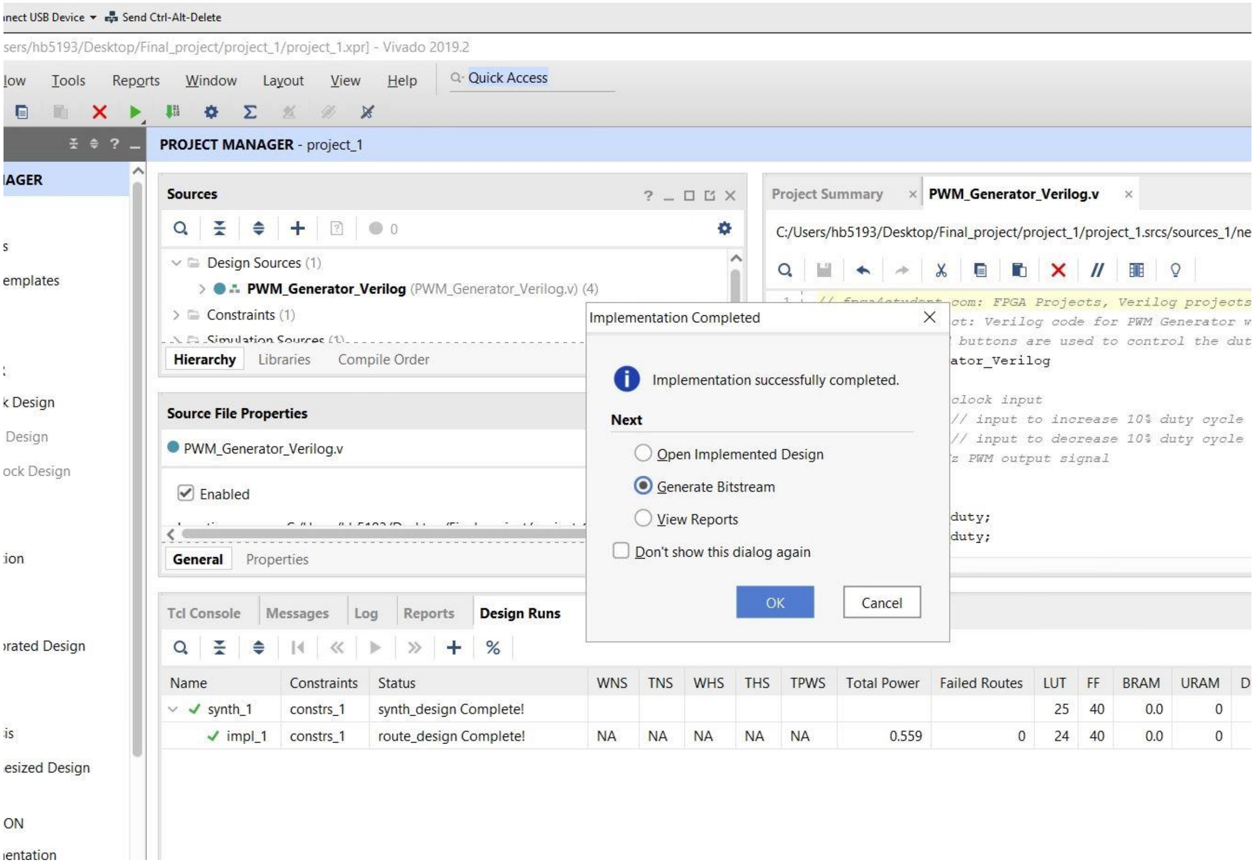
5.14 Copy the PWM Verilog code from FPGA4student website, and run synthesis



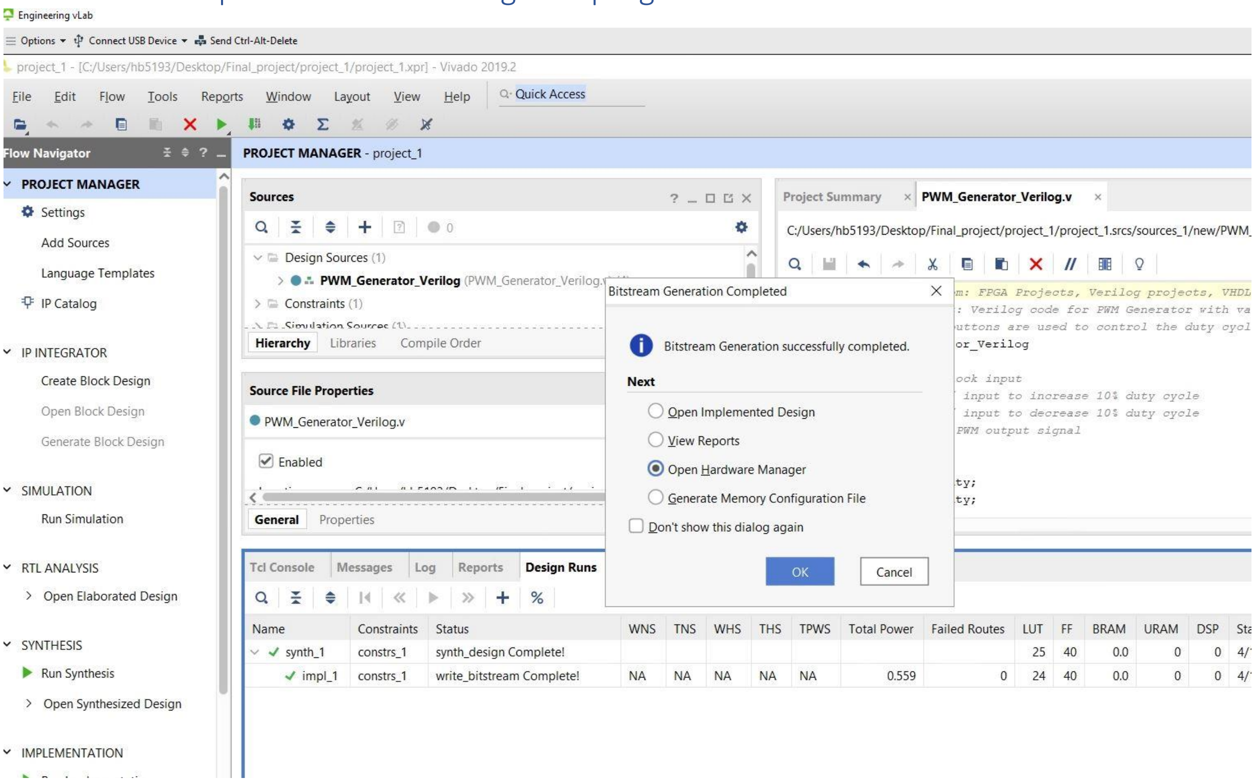
5.15 Run implementation



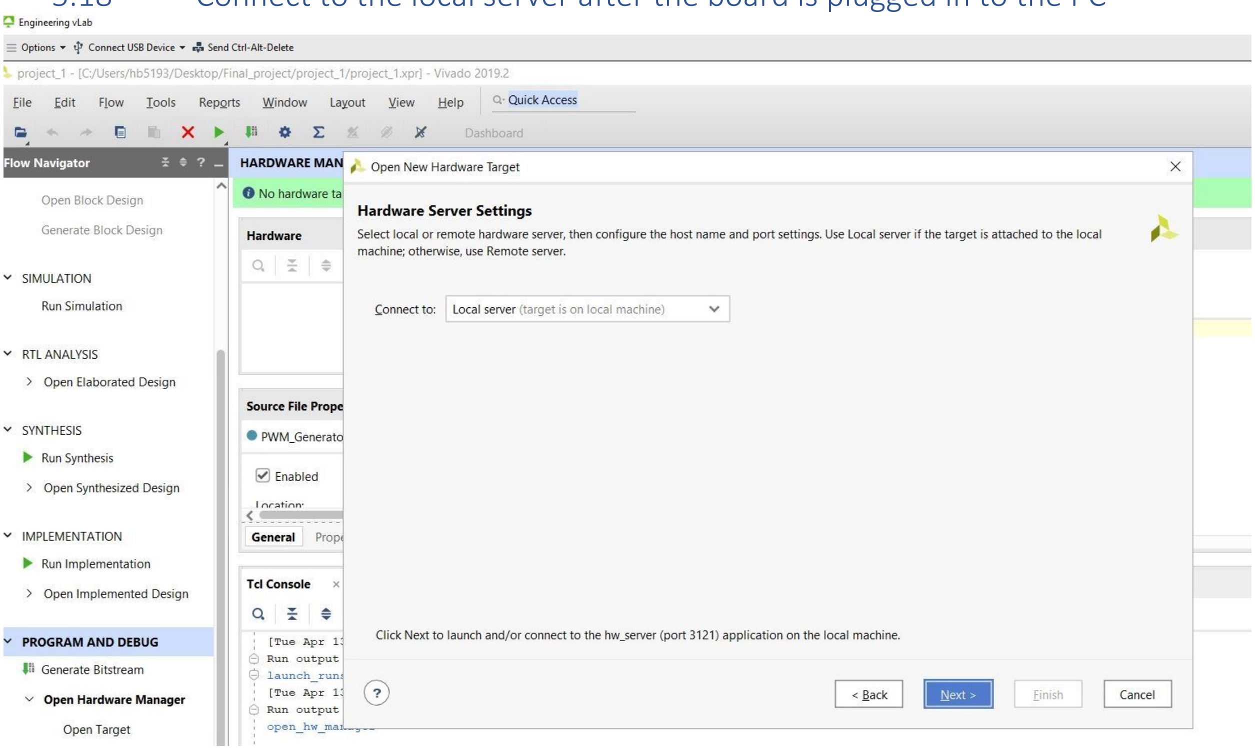
5.16 After successful design build, generate Bitstream



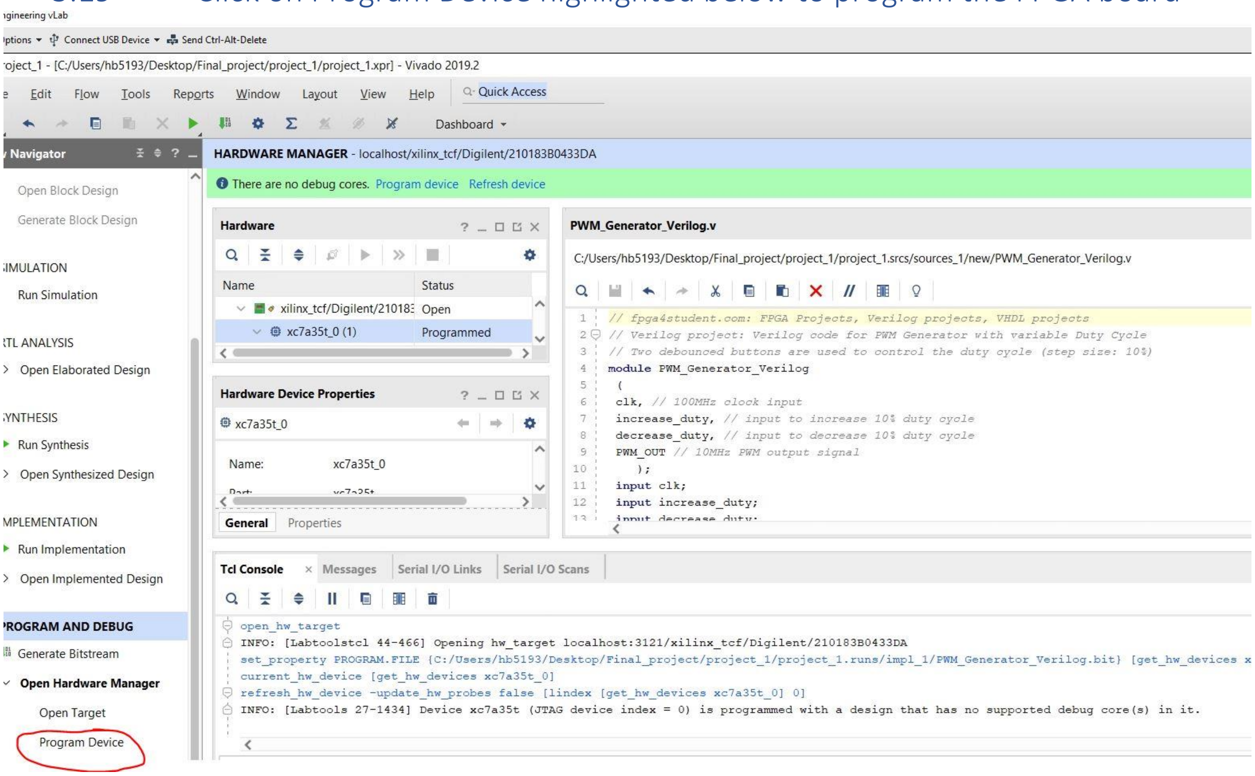
5.17 Open hardware Manager to program the BASYS 3 board



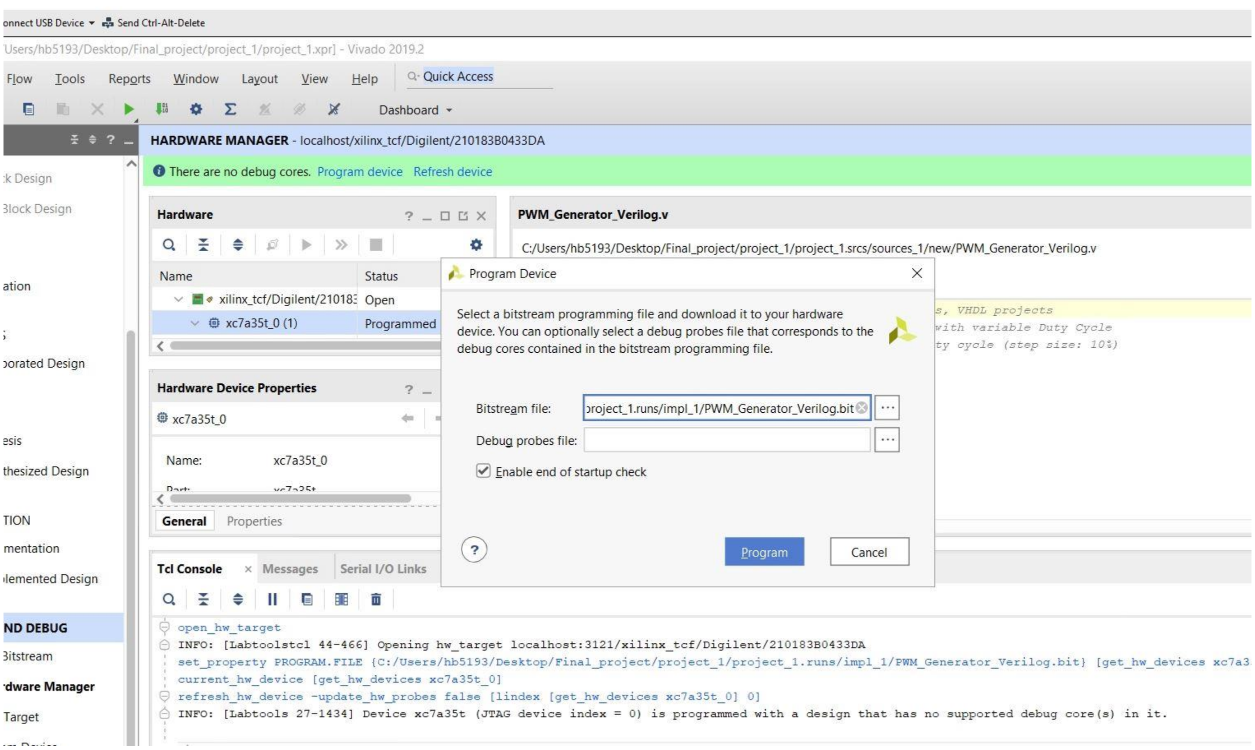
5.18 Connect to the local server after the board is plugged in to the PC



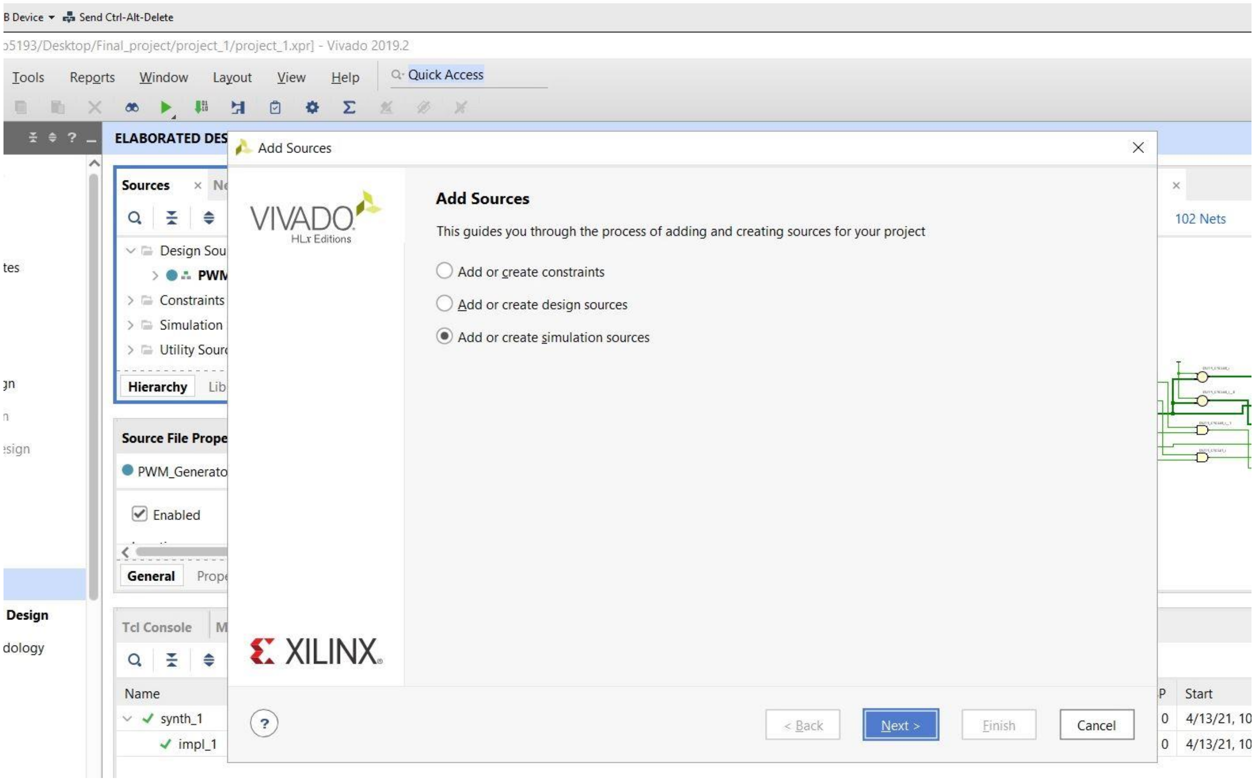
5.19 Click on Program Device highlighted below to program the FPGA board



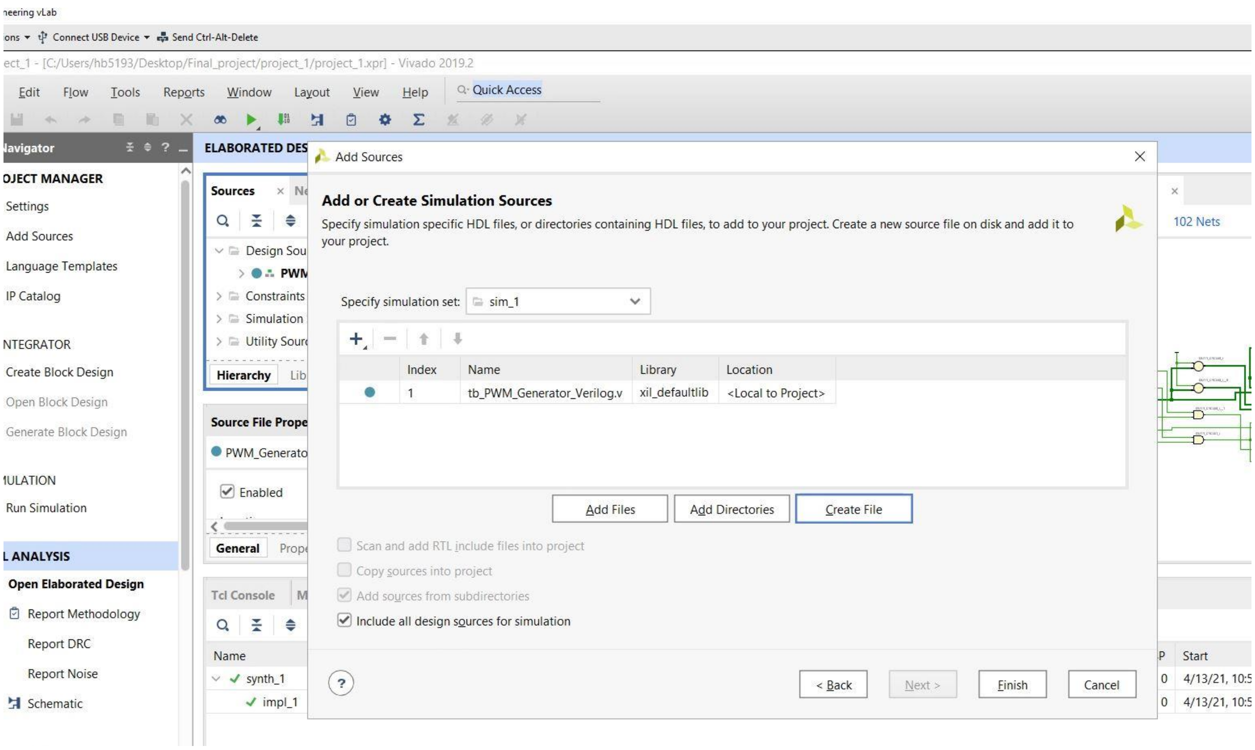
5.20 Select the bitstream file to program the device



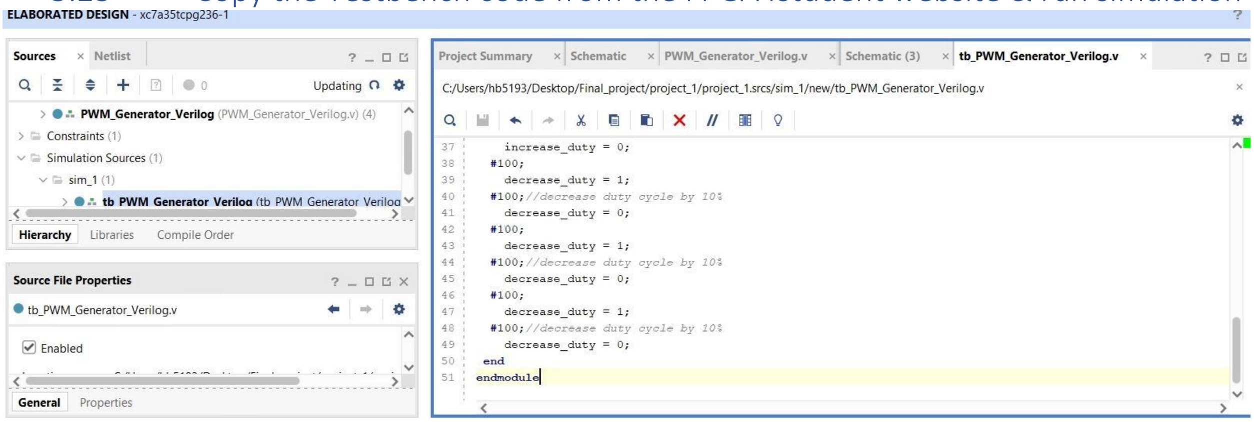
5.21 Add a simulation source file to get the simulation waveform



5.22 Create the testbench file



5.23 Copy the Testbench code from the FPGA4student website & run simulation



6. Project Test Results

6.1 Power & Utilization

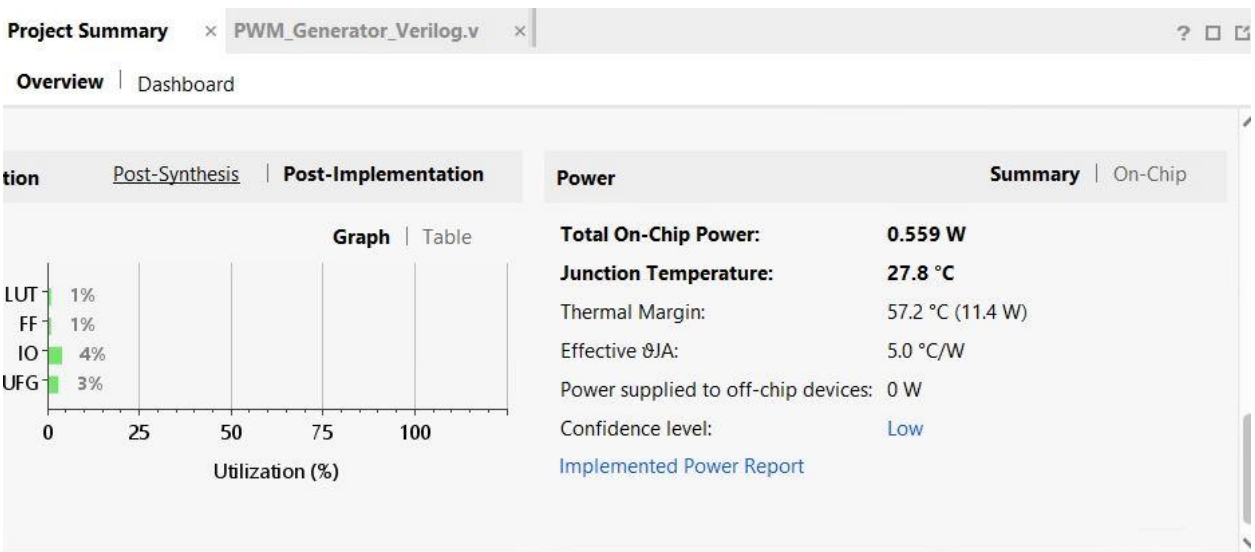


Figure 3: Post-Implementation Utilization

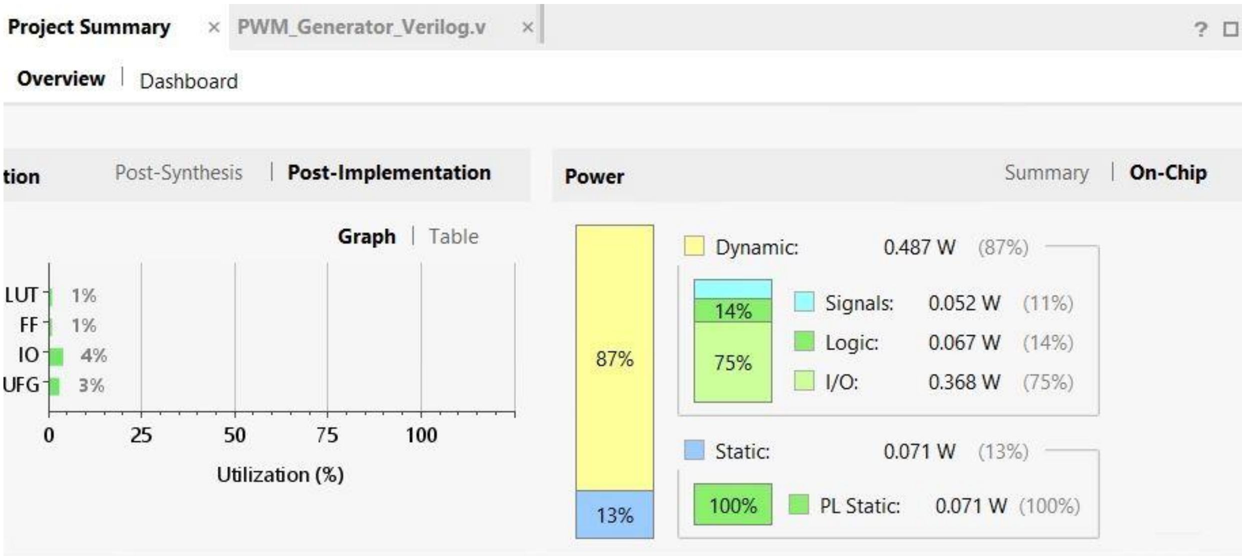


Figure 4: Power Summary

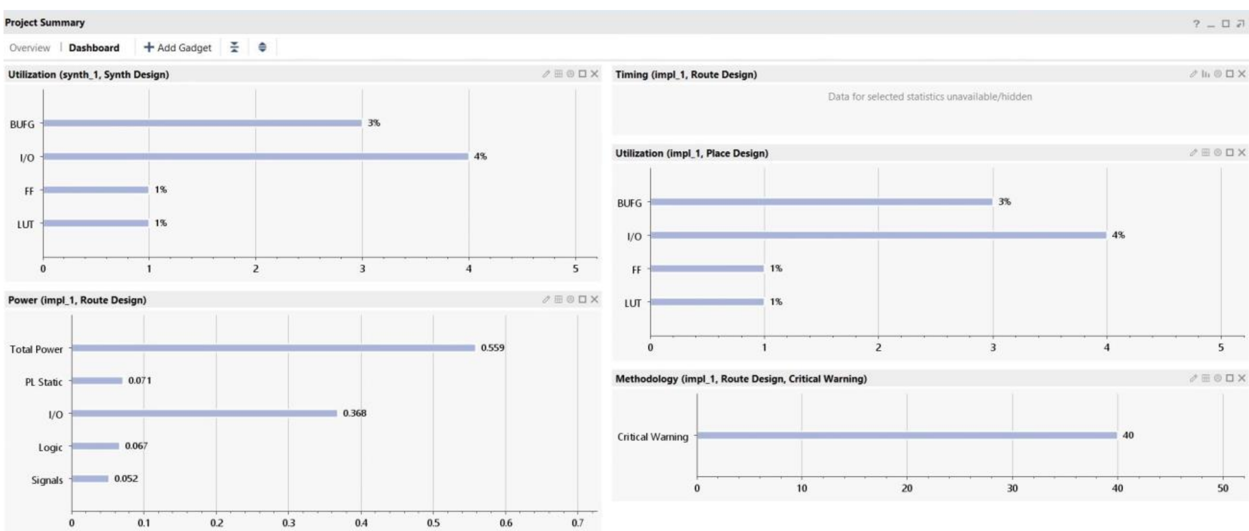


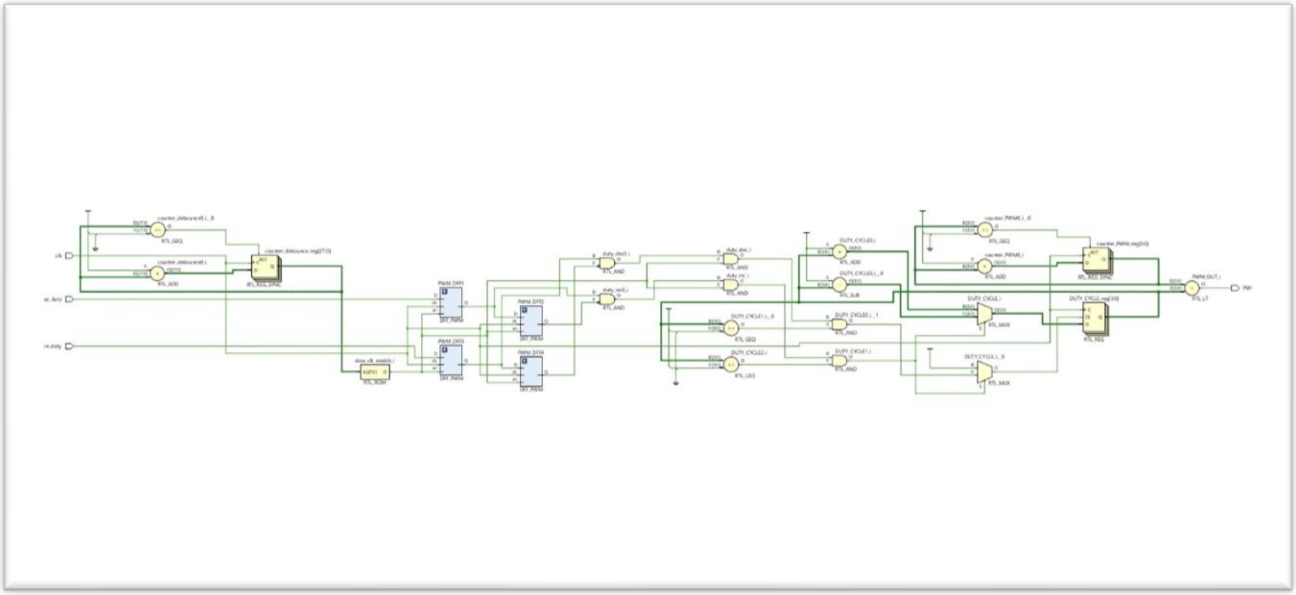
Figure 5: Detailed Summary

6.2 Timing Results

Stage	Time
Synthesis design Time	1.02 minutes
Implementation Design Time	2.10 minutes
Write Bitstream Time	51 seconds

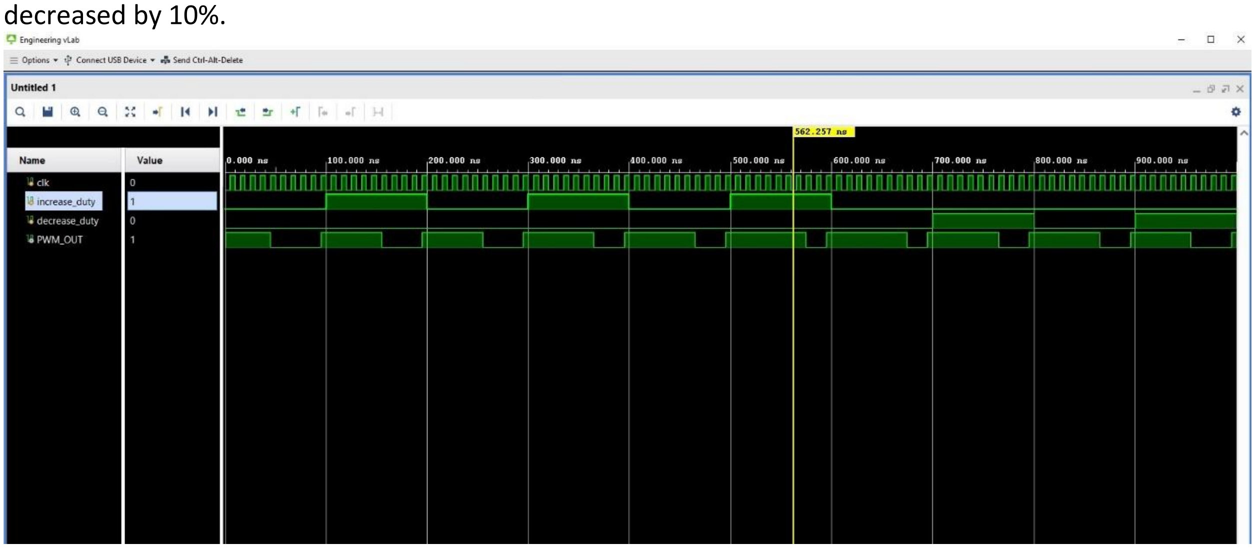
6.3 RTL Detailed Design

This is a **sequential** design since the output is driven by a clock and it depends not only on the current value of the input.



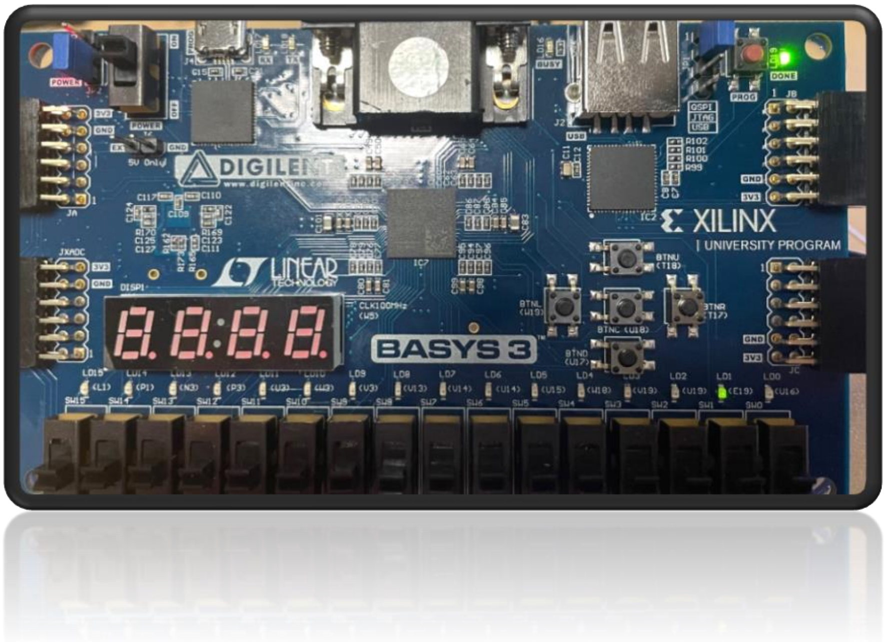
6.4 Simulation Waveform Results

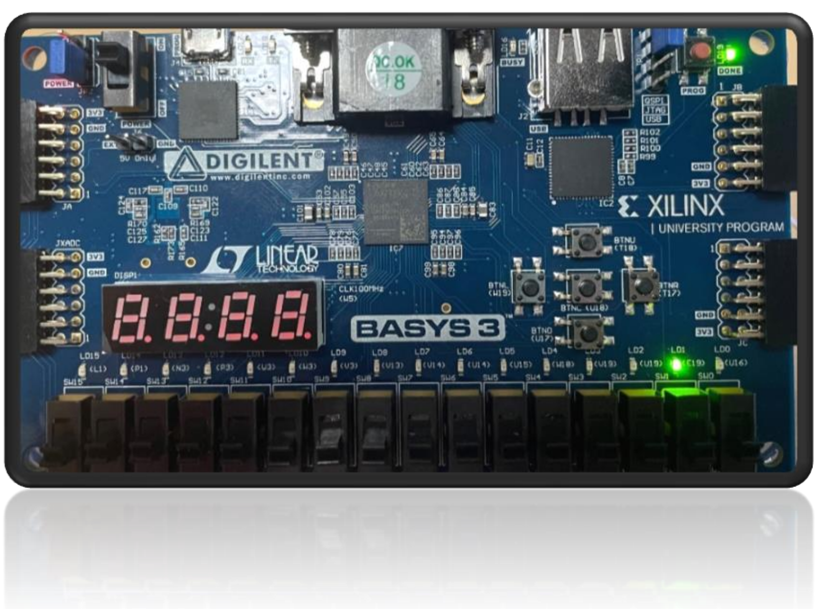
We could see the clock is generated at 10MHz, while every time the increase duty will be set to 1, the duty cycle will be increased by 10%. Every time the decrease duty will be set to 1, the duty cycle will be decreased by 10%.



6.5 Hardware (BASYS 3) results:

Each time the (U18) push button is pressed, the duty cycle of the PWM signal output will increased by 10% on the output LED (E19). Also, each time the (T17) push button is pressed, the duty cycle will be decreased by 10%. The first picture is shown the PWM signal that's decreased by 30% when presseing the push button 3times. The second picture is shown the PWM signal is increased by 30% as you could see the LED light is more intenese.





7. Conclusion

This project is considered a fairly small project, but the usage of the PWM signal generator is very important in the embedded system design technologies. Uses for PWM vary widely, it is the heart of Class D audio amplifiers, by increasing the voltages you increase the maximum output, and by selecting a frequency beyond human hearing (typically 44Khz) PWM can be used. Another popular application is motor speed control. Motors as a class require very high currents to operate. Being able to vary their speed with PWM increases the efficiency of the total system by quite a bit. PWM is more effective at controlling motor speeds at low RPM than linear methods.