Intel Soc Design Workshop-1

Subham Ball
Dept. of System of Chip Design
Indian Institute Of Technology–Palakkad
152202017@smail.iitpkd.ac.in

***Abstract*—FPGA design using Intel Quartus Prime Design Software and Programming FPGA Design using Labsland remote board access.**

I. INTRODUCTION

Field Programmable Gate Arrays (FPGAs) is semiconductor devices. it is based on a matrix of configurable logic blocks (CLBs) and connected via programmable interconnects. After manufacture, FPGAs can be reprogrammed to meet specific application or feature needs. Here , "Gate" refers to transistors,"Array" refers they are arranged it a larger structure and "Field Programmable" refers the connections between the internal components are re-programmable after manufacture.
To Configure any FPGA there are three main steps are:

I)**Choose**:The building blocks we Chose.
II)**Configure**:How we configure them.
III)**Connect**:How We connect them.

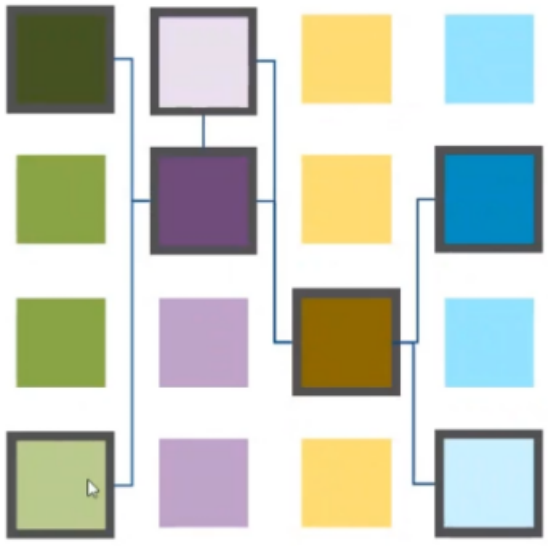


Fig. 1. Building Blocks

II. BASIC COMPONENTS OF FPGA

A. LookUp Table(LUT)

LUT is a basic building block of FPGA,Based on our unique requirements and instructions, the lookup table is essentially our personalised truth table, which is filled with data that are pertinent to our FPGA.we can think LUT as a small piece of RAM which is loaded whenever we turn on the FPGA chip.

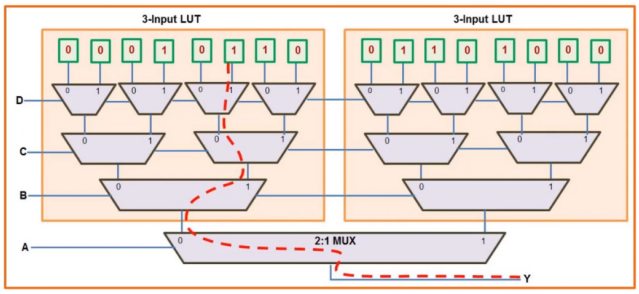


Fig. 2. LookUp Table

Adaptive Logic Module(ALM): ALM have four dedicated registers and 8 input fracturable LUT to aid in better timing closure in register design. Here Fracturable means we can break 8 input to (5+3) or (4+4) or any combination based on our equation. that is why we called it Adaptive Logic Module(ALM).

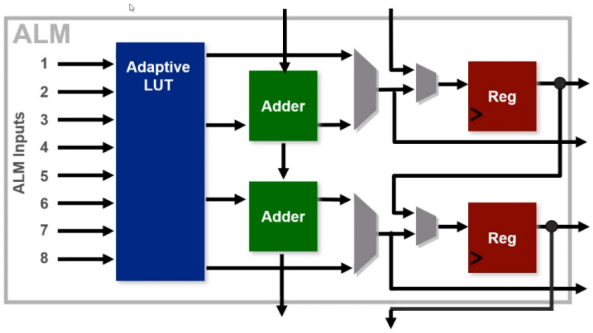


Fig. 3. Adaptive Logic Module

Logic array Block(LAB): A group of 10 ALM that are connected in an array are referred to as a LAB.

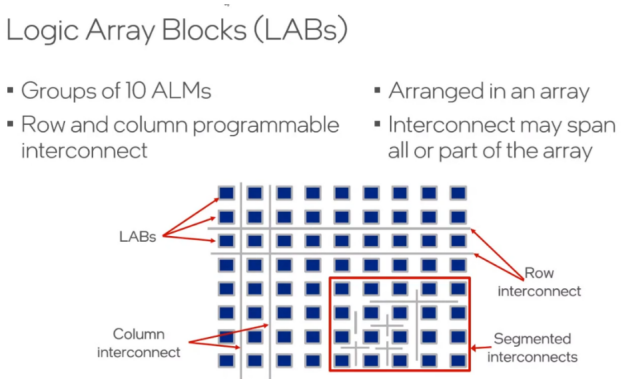


Fig. 4. Logic Array Block

B. Memory Resources

The memory is present inside FPGA we called it on-chip RAM block. it can be configure as RAM or ROM. we can configure this block into 4 ways
1)Single port RAM
2)Dual port RAM
3)Single port ROM
4)Dual port ROM

C. Digital Signal Processing(DSP)

it includes Accumulator and Multiplier .Using these DSP resources, we can perform any mathematical operations, including add, subtract, multiply, and more

III. Benefits of FPGA

Some important benefits are :
I) Re-configurable & flexible.
II) Product longevity
II) Reduced Time to market
IV) Market size optimized

IV. Characteristic of FPGA

Some important Characteristic are :
I) Flexible and multi configure re-programmable.
II) Truly parallel in nature
II) custom Hardware functionality
IV) speed and reliability

V. FPGA Application

FPGA is Reprogrammable, that is why FPGAs are an ideal fit for many differnt market such as:
I)Aerospace Defense
II)Automotive
III)High Performance Computing and Data Storage
IV)Wired and Wireless Communications, etc.

VI. Intel FPGAs

1) Intel Max: It is introduced 2014,55nm size ,Delivering lowest power,single chip, non volatile,lowest cost programmable logic device used for the optimal set of system components.

2) Intel Cyclone: It is of 28nm size,optimized for balanced power and bandwidth for cost sensitive application.

3) Intel Arria: It is of 20nm size,mid range power it also consists processor inside, used in data centers, networking etc.it deliver more than a speed grade faster core performance and consume up to 40 percent lower power than previous generation FPGAs.

4) Intel STRATIX: It is of 14nm size,deliver advantages in performance, power efficiency,system integration etc. optimized for high band-width. used in Data Center Acceleration ,cognetive Computing etc.

5) Intel AGILEX: It is built with advanced 10 nm superFin technology,optimized for high band-width,used in network Communications Data Centers etc.

Any FPGA contains mainly 5 resources ALM, MLAB, DSP, Clocking resourcse,IO resources

VII. General FPGA Design Process

This process includes

1) HDL code created in the Verilog, VHDL, and System Verilog programming languages
2) Run simulations to determine whether the functionality of the code we've developed is correct or not.
3) perform synthesis means converting our RTL code into gate level net-list.
4) We do operations like placing and routing using the software Intel Quartus Prime Lite, and we perform some timing analysis to make sure that the device is not experiencing any hold and setup time violations.
5) The FPGA receives a created sof file.

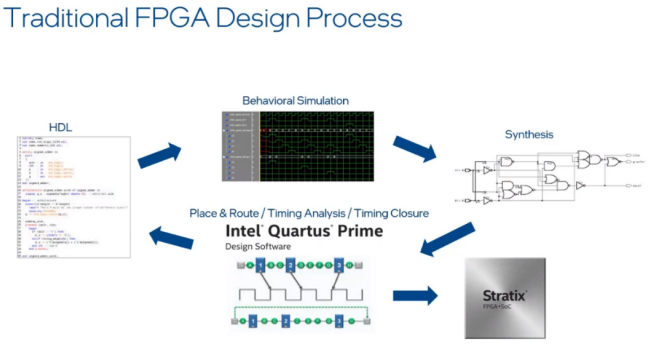


Fig. 5. FPGA Design process

VIII. IMPLEMENTATION

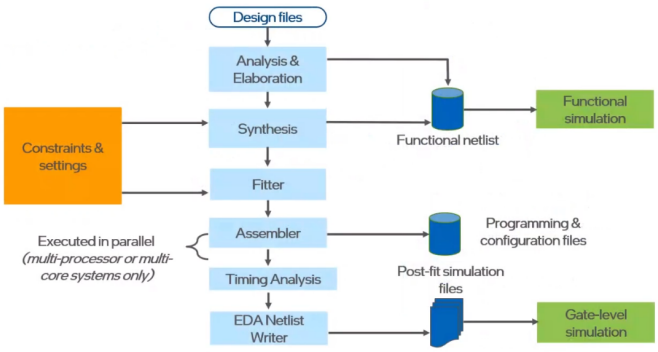


Fig. 6. Tool Flow Process

1) Design files: Develop the code based on the Specification. In this experiment we used max10(10M50DAF484C7G) and The code for the full adder is given in the Figure7

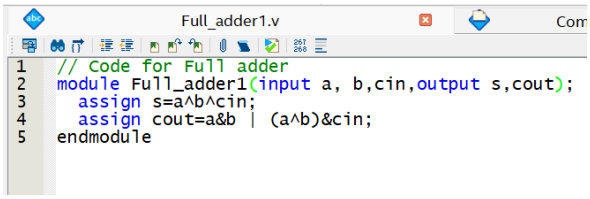


Fig. 7. Full Adder Design - verilog code

2) Analysis and Elaboration: If there are no syntax or semantic mistakes, the tool will output the RTL depicted in Figure 6. This technique verifies our written code for any errors.

Additionally, this process will produce a flow summary that includes the family and device we are using, as well as the total number of pins and registers (among other things) that are unknown because we haven't yet connected. Figure 9 displays the flow summary produced by this analysis and elaboration process.

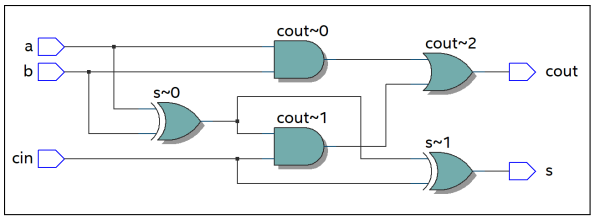


Fig. 8. RTL View

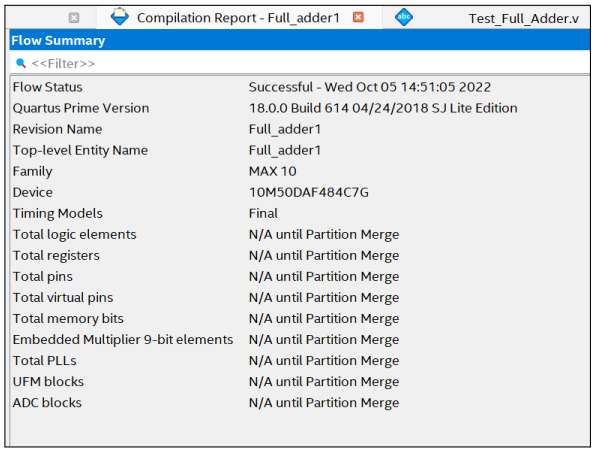


Fig. 9. Flow Summary

3) Functional Simulation: In this step, we must confirm that the functionality of the code is working as intended, which in this experiment implies whether or not the entire adder is correctly adding data. To this end, a test bench is developed, and it is shown in Figure 10. Following that, we

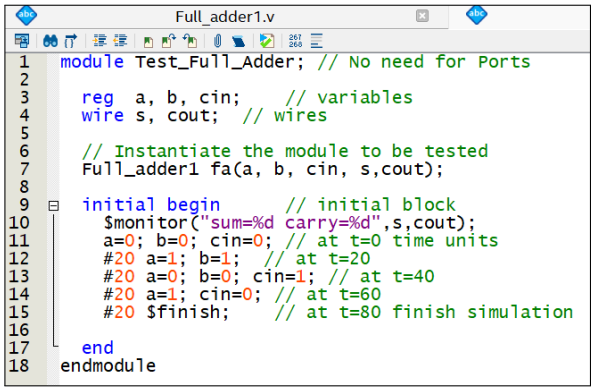


Fig. 10. Test Bench - Verilog Code

perform analysis and elaboration once more to ensure that no syntax or semantic errors are present in the test bench. Now, using modelsim to simulate the above test bench, generate a wave form, as shown in figure 11, which is used to verify functionality.

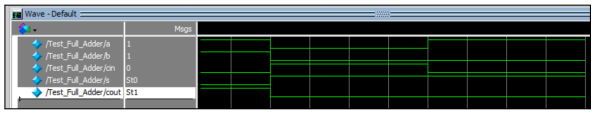


Fig. 11. Wave Form

4) Pin Planner: Before proceeding to the synthesis and fittering, Pin Planner is required to connect input and output to device pins such as a,b,cin are connected to switches and outputs cout and sum are connected to leds. Figure 12 depicts the connection.

The added pins are shown in brown on the board in Figure 13.

Node Name	Direction	Location	I/O Bank	/REF Group	'O Standard	Reserved	rent Stren	Slew Rate	fferential P.	:t Preserva
a	Input	PIN_C10	7	B7_N0	2.5 V...ault)		12mA...ult)			
b	Input	PIN_C11	7	B7_N0	2.5 V...ault)		12mA...ult)			
cin	Input	PIN_D12	7	B7_N0	2.5 V...ault)		12mA...ult)			
cout	Output	PIN_A8	7	B7_N0	2.5 V...ault)		12mA...ult)	2 (default)		
s	Output	PIN_A9	7	B7_N0	2.5 V...ault)		12mA...ult)	2 (default)		

Fig. 12. Pin Planning

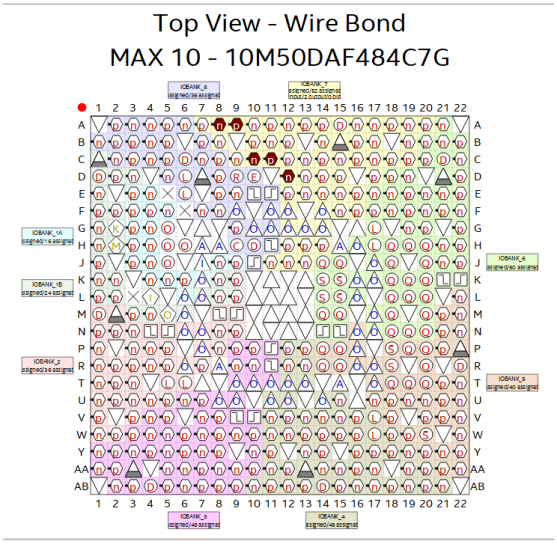


Fig. 13. MAX 10(Top view)

5) Synthesis: The technology map viewer shown in Figure 14 while doing analysis and synthesis shows a circuit with combinational cells in which the gate level is concealed.

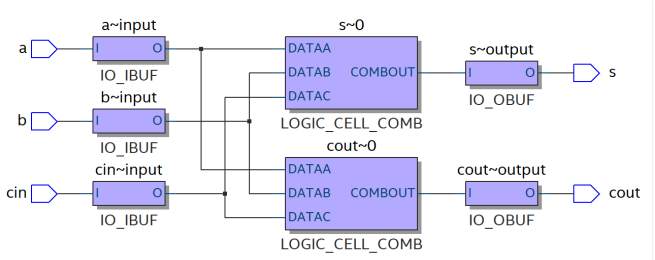
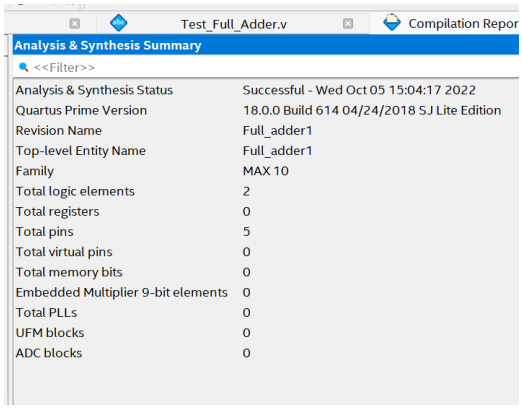


Fig. 14. Technology Map View

When you right-click that combinational cell and choose properties, the gate level viewer shown in Figure 8 is displayed.

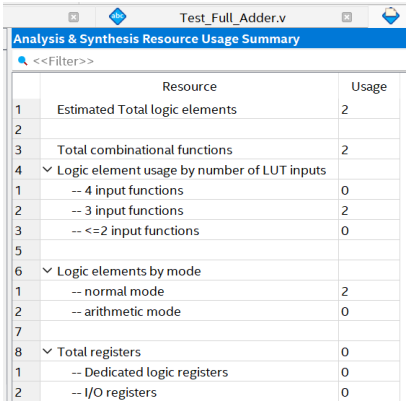
This procedure will provide a summary that includes details on the pins we utilised, the logic components that were used, and other information that is displayed in the flow summary shown in Figure 15. The proportion of utilisation will be revealed in the figure after the fitting process is complete. We utilised 5 pins altogether, 3 for inputs and 2 for outputs.



Analysis & Synthesis Status	Successful - Wed Oct 05 15:04:17 2022
Quartus Prime Version	18.0.0 Build 614 04/24/2018 SJ Lite Edition
Revision Name	Full_adder1
Top-level Entity Name	Full_adder1
Family	MAX 10
Total logic elements	2
Total registers	0
Total pins	5
Total virtual pins	0
Total memory bits	0
Embedded Multiplier 9-bit elements	0
Total PLLs	0
UFM blocks	0
ADC blocks	0

Fig. 15. Flow Summary(After synthesis)

6) Fitter: The process of putting up the logic blocks is done through placement and routing, and after that is complete, connecting the blocks we have already placed is done by routing. Following completion of this operation, the flow summary created, shown in Figure 16, reveals that there is a percentage



	Resource	Usage
1	Estimated Total logic elements	2
2		
3	Total combinational functions	2
4	Logic element usage by number of LUT inputs	
1	-- 4 input functions	0
2	-- 3 input functions	2
3	-- <=2 input functions	0
5		
6	Logic elements by mode	
1	-- normal mode	2
2	-- arithmetic mode	0
7		
8	Total registers	0
1	-- Dedicated logic registers	0
2	-- I/O registers	0

Fig. 16. Flow Summary(After Fitter)

use of logic components, pins, etc. In the experiment, we utilised a total of 5 pins, and the device has 360 total pins, therefore we used 1% of the total number of pins accessible. The resource overview also describes how to use the materials.

7) Assembler: The software files that we utilise will be generated after we complete this Assembler stage. The FPGA is programmed using this .sof file, also known as an SRAM Object File.

8) Timing Analysis: we perform some timing analysis to make sure that the device is not experiencing any hold and setup time violations.shown in Figure 17

Test_Full_Adder.v | Compilation R

Timing Analyzer Summary

<<Filter>>

Quartus Prime Version	Version 18.0.0 Build 614 04/24/2018 SJ Lite Edition
Timing Analyzer	Legacy Timing Analyzer
Revision Name	Full_adder1
Device Family	MAX 10
Device Name	10M50DAF484C7G
Timing Models	Final
Delay Model	Combined
Rise/Fall Delays	Enabled

Fig. 17. Timing Analysis Summary

IX. PROGRAMMING SWITCH TO LED DESIGN INTO CYCLONE(5) USING LABSLAND

Apply the preceding stages, or when we have code with correct functionality, click the Compile and design process, and Intel quartus prime lite creates the programming files. The following steps are taken:

1) project creating: Set the family to cyclone(5) and the device to 5CSEMA5F31C6 to create the new project.

2) Implementation: we Use the verilog code for the switch to led shown in Figure 18 to implement the above mentioned procedures. Additionally, manually entering pins into the pin planner is difficult.

```
module switch_to_led(SW, LEDR); //create module switch_to_led
  input [9:0] SW; // input declarations: 10 switches
  output [9:0] LEDR; // output declarations: 10 red LEDs
  assign LEDR = SW; // connect switches to LEDs
endmodule
```

Fig. 18. Switch to led - verilog code

When there are more pins needed for that purpose, we use the TC Console, which does the pin planning automatically using the code shown in Figure 19 The next step is the design and compilation process, which produces the programming files.

```
set_location_assignment PIN_Y21 -to LEDR[9]
set_location_assignment PIN_W21 -to LEDR[8]
set_location_assignment PIN_W20 -to LEDR[7]
set_location_assignment PIN_Y19 -to LEDR[6]
set_location_assignment PIN_W19 -to LEDR[5]
set_location_assignment PIN_W17 -to LEDR[4]
set_location_assignment PIN_V18 -to LEDR[3]
set_location_assignment PIN_V17 -to LEDR[2]
set_location_assignment PIN_W16 -to LEDR[1]
set_location_assignment PIN_V16 -to LEDR[0]
set_location_assignment PIN_AF16 -to SW[9]
set_location_assignment PIN_AE16 -to SW[8]
set_location_assignment PIN_AG16 -to SW[7]
set_location_assignment PIN_AH17 -to SW[6]
set_location_assignment PIN_AH18 -to SW[5]
set_location_assignment PIN_AJ16 -to SW[4]
set_location_assignment PIN_AJ17 -to SW[3]
set_location_assignment PIN_AJ19 -to SW[2]
set_location_assignment PIN_AK19 -to SW[1]
set_location_assignment PIN_AK18 -to SW[0]
set_location_assignment PIN_AF14 -to CLOCK_50
```

Fig. 19. Pin Planner

3) programming the file in FPGA using Labsland: we use labsland website which is virtual lab. we upload the .sof file before programming the device. the output of the above mentioned switch to let device is displayed in figure 20 and figure 21. our main goal of the switch to led design is that when we click any switch the associated led is glow.

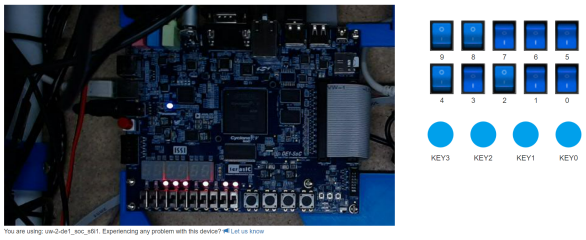


Fig. 20. Output Screenshot 1

In Figure 20, the switches that are turned on are 0,1,3,5,6,7, and in Figure 21, the switches that are turned on are 0,1,3,5,6,7,9, and the matching lights are glowing, as seen in Figures 20 and 21. We programmed the gadget in labsland without utilising the hardware FPGA chip.

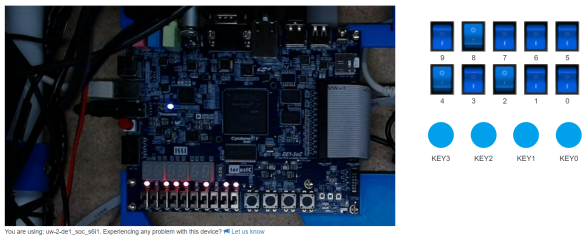


Fig. 21. Output Screenshot 2

X. PROGRAMMING INEQUALITY DETECTOR DESIGN INTO CYCLONE (5) USING LABSLAND

Apply the same process which we discuss in section IX. we use the veriog code for the Inequality detector is shown in figure 22.

```
// Code your design here

module Inequality_checker ( input [1:0] A, input [1:0] B, output z );
    assign z= (A=B)?0:1;

endmodule
```

Fig. 22. Inequality Checker - verilog code

and figure 23 shown the gate level view of the inequality detector design.

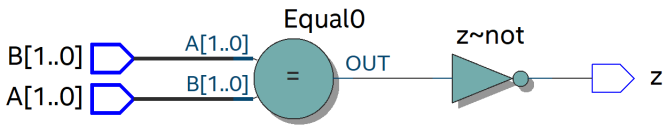
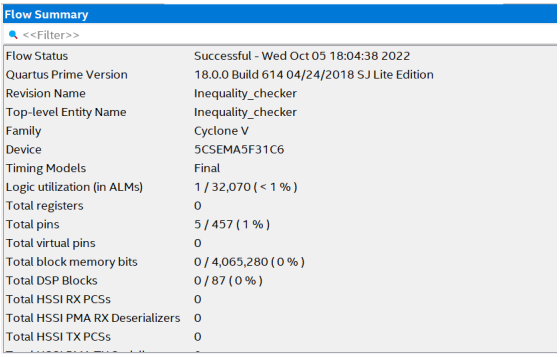


Fig. 23. Gate Level View

performing this process in the labsland we saw the flow summary generated which is shown in the Fig.24. and the



Flow Summary

<<Filter>>

Flow Status	Successful - Wed Oct 05 18:04:38 2022
Quartus Prime Version	18.0.0 Build 614 04/24/2018 SJ Lite Edition
Revision Name	Inequality_checker
Top-level Entity Name	Inequality_checker
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	1 / 32,070 (< 1 %)
Total registers	0
Total pins	5 / 457 (1 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	0 / 87 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0

Fig. 24. Flow Summary

and the figure 25 shown the output of the Inequality detector program , here we gave the two input are same that is why light is not glow.

Fig. 25. Output Screenshot -1

figure 26 shown the output of the Inequality detector program , here we gave the two input are not equal that is why light is glow.

Fig. 26. Output Screenshot -2