University of Saskatchewan

Lab 1 : FPGA Basic Design

CME433-01

Addi Amaya

Caa746

11255790

Sept 24th, 2021

Contents

Question 1 2
Question 2 2
Question 3 3
Question 4 4
Question 5 6
Basic Optimization 7
Area Optimization 8
Power Optimization 9
Performance Optimization 11
Conclusion 12

Question 1

When using a case statement, you are inferring ROM functions which means that the synthesis tools are sets of registers that can be replaced with the ALTSYNCRAM or LPM_ROM IP Core. These tools for the case statement generate optimized decode logic to quickly select which case statement is valid. Whereas the if/else statement can incorporate priority-encoded logic, but the synthesis requires more space as it's a series of MUXs. Having a MUX for each condition makes the synthesis of the if/else statements more complex in hardware implementation.

Question 2

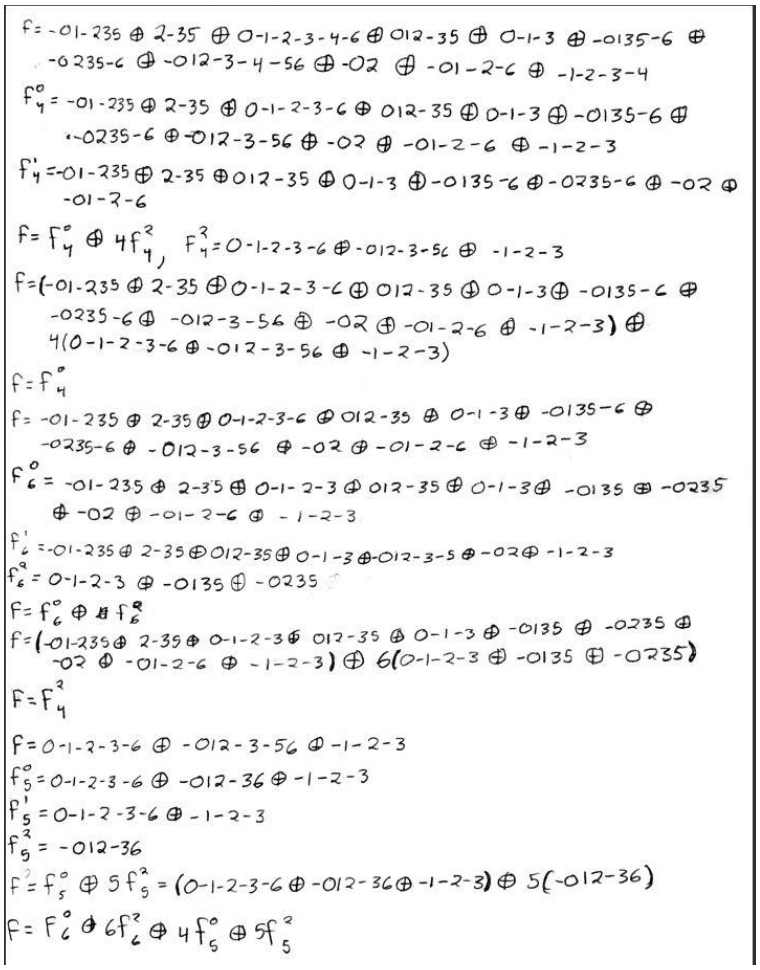
The first step into optimizing your performance and size/area is to determine the purpose of the design.

For example, if your design is a logical applications and combinational functions or adders, counters, accumulators, and comparators, you can configure the LE to a specific operating mode for better performance. LE in normal mode is for general logic and combinational functions. Whereas LE in arithmetic mode is ideal for implementing adders, counters, accumulators, and comparators.

In optimize the size/area of a design, manually configure the carry chain logic to run vertically to allow faster connections through direct link interconnects. Although running vertical may be design specific, the best practice would be to examine how the LE is laid out and in which mode it is in and from there you can determine if the long carry chain would be a better fit vertical or horizontal depending on where the LAB and the M4K memory blocks reside.

Question 3

Using the following walkthrough for the exercise provided in the class it was fairly easy to copy the logic for each condition



$f = -01-235 \oplus 2-35 \oplus 0-1-2-3-4-6 \oplus 012-35 \oplus 0-1-3 \oplus -0135-6 \oplus$
$-0235-6 \oplus -012-3-4-56 \oplus -02 \oplus -01-2-6 \oplus -1-2-3-4$

$f_4^0 = -01-235 \oplus 2-35 \oplus 0-1-2-3-6 \oplus 012-35 \oplus 0-1-3 \oplus -0135-6 \oplus$
$-0235-6 \oplus -012-3-56 \oplus -02 \oplus -01-2-6 \oplus -1-2-3$

$f_4^1 = -01-235 \oplus 2-35 \oplus 012-35 \oplus 0-1-3 \oplus -0135-6 \oplus -0235-6 \oplus -02 \oplus$
$-01-2-6$

$f = f_4^0 \oplus 4f_4^2, \quad f_4^2 = 0-1-2-3-6 \oplus -012-3-56 \oplus -1-2-3$

$f = (-01-235 \oplus 2-35 \oplus 0-1-2-3-6 \oplus 012-35 \oplus 0-1-3 \oplus -0135-6 \oplus$
$-0235-6 \oplus -012-3-56 \oplus -02 \oplus -01-2-6 \oplus -1-2-3) \oplus$
$4(0-1-2-3-6 \oplus -012-3-56 \oplus -1-2-3)$

$f = f_4^0$

$f = -01-235 \oplus 2-35 \oplus 0-1-2-3-6 \oplus 012-35 \oplus 0-1-3 \oplus -0135-6 \oplus$
$-0235-6 \oplus -012-3-56 \oplus -02 \oplus -01-2-6 \oplus -1-2-3$

$f_6^0 = -01-235 \oplus 2-35 \oplus 0-1-2-3 \oplus 012-35 \oplus 0-1-3 \oplus -0135 \oplus -0235$
$\oplus -02 \oplus -01-2-6 \oplus -1-2-3$

$f_6^1 = -01-235 \oplus 2-35 \oplus 012-35 \oplus 0-1-3 \oplus -012-3-5 \oplus -02 \oplus -1-2-3$

$f_6^2 = 0-1-2-3 \oplus -0135 \oplus -0235$

$f = f_6^0 \oplus f_6^2$

$f = (-01-235 \oplus 2-35 \oplus 0-1-2-3 \oplus 012-35 \oplus 0-1-3 \oplus -0135 \oplus -0235 \oplus$
$-02 \oplus -01-2-6 \oplus -1-2-3) \oplus 6(0-1-2-3 \oplus -0135 \oplus -0235)$

$f = f_4^2$

$f = 0-1-2-3-6 \oplus -012-3-56 \oplus -1-2-3$

$f_5^0 = 0-1-2-3-6 \oplus -012-36 \oplus -1-2-3$

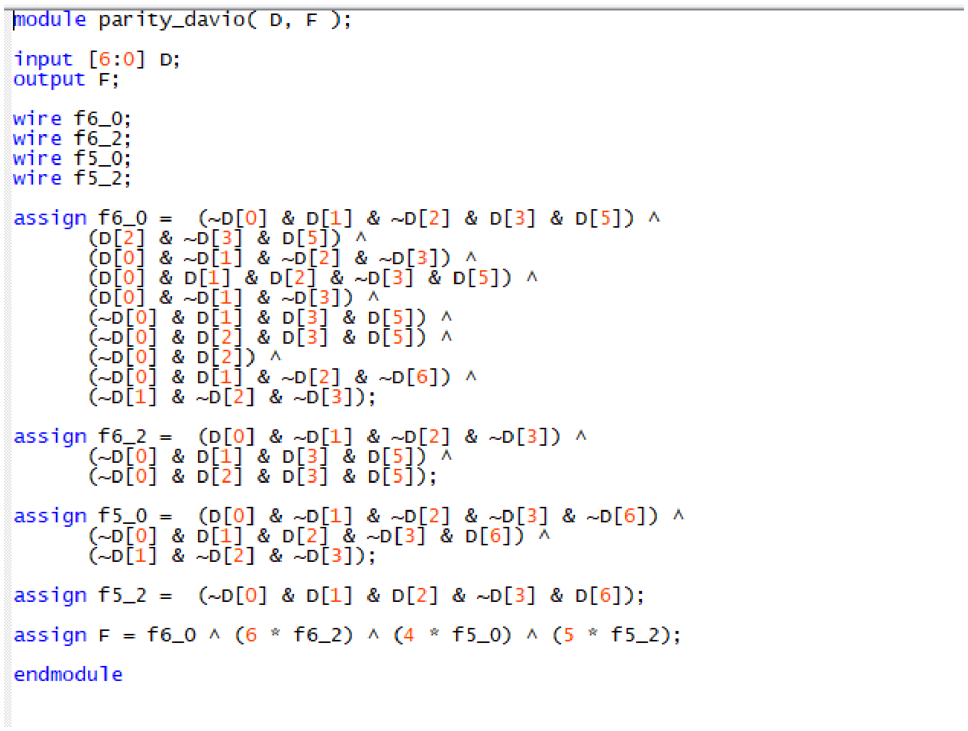
$f_5^1 = 0-1-2-3-6 \oplus -1-2-3$

$f_5^2 = -012-36$

$f = f_5^0 \oplus 5f_5^2 = (0-1-2-3-6 \oplus -012-36 \oplus -1-2-3) \oplus 5(-012-36)$

$f = f_6^0 \oplus 6f_6^2 \oplus 4f_5^0 \oplus 5f_5^2$

From this paper the following module was made from the template

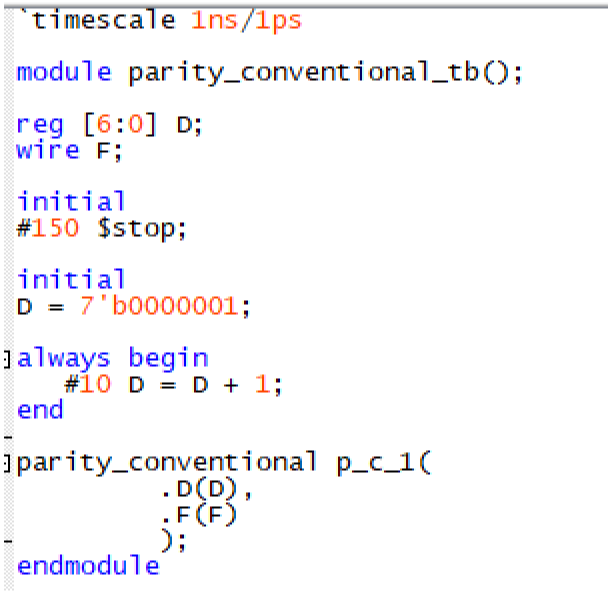


```
module parity_davio( D, F );

input [6:0] D;
output F;

wire f6_0;
wire f6_2;
wire f5_0;
wire f5_2;

assign f6_0 =  (~D[0] & D[1] & ~D[2] & D[3] & D[5]) ^
      (D[2] & ~D[3] & D[5]) ^
      (D[0] & ~D[1] & ~D[2] & ~D[3]) ^
      (D[0] & D[1] & D[2] & ~D[3] & D[5]) ^
      (D[0] & ~D[1] & ~D[3]) ^
      (~D[0] & D[1] & D[3] & D[5]) ^
      (~D[0] & D[2] & D[3] & D[5]) ^
      (~D[0] & D[2]) ^
      (~D[0] & D[1] & ~D[2] & ~D[6]) ^
      (~D[1] & ~D[2] & ~D[3]);

assign f6_2 =  (D[0] & ~D[1] & ~D[2] & ~D[3]) ^
      (~D[0] & D[1] & D[3] & D[5]) ^
      (~D[0] & D[2] & D[3] & D[5]);

assign f5_0 =  (D[0] & ~D[1] & ~D[2] & ~D[3] & ~D[6]) ^
      (~D[0] & D[1] & D[2] & ~D[3] & D[6]) ^
      (~D[1] & ~D[2] & ~D[3]);

assign f5_2 =  (~D[0] & D[1] & D[2] & ~D[3] & D[6]);

assign F = f6_0 ^ (6 * f6_2) ^ (4 * f5_0) ^ (5 * f5_2);

endmodule
```

Question 4

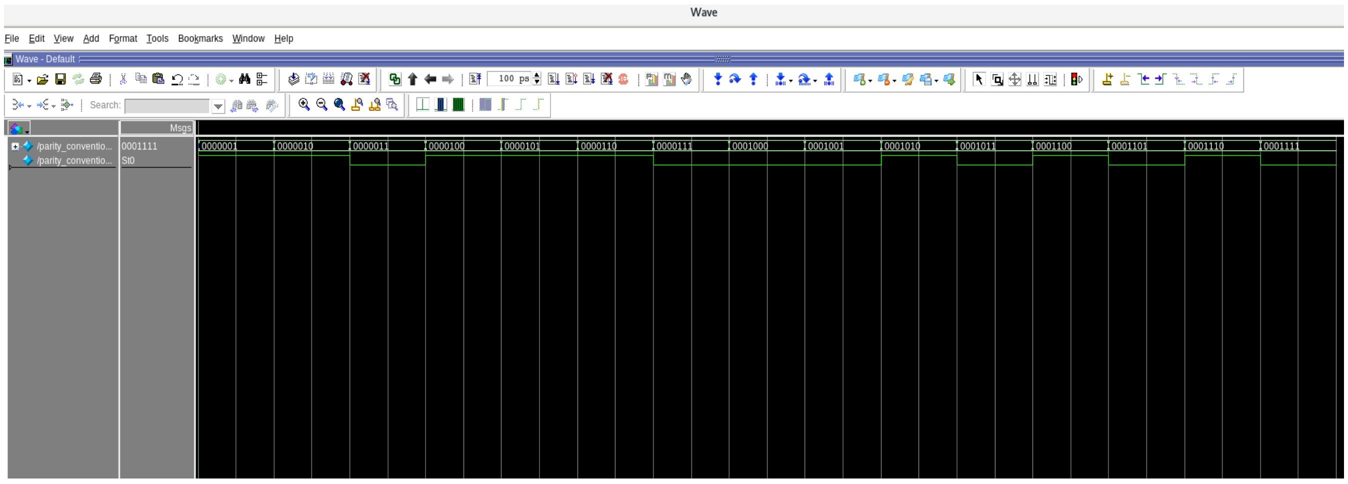
The method that I decided on to test if davio matches the conventional was to make a testbench that just increments the D value and compare the results of both

The testbench that I used for the conventional method was as follows



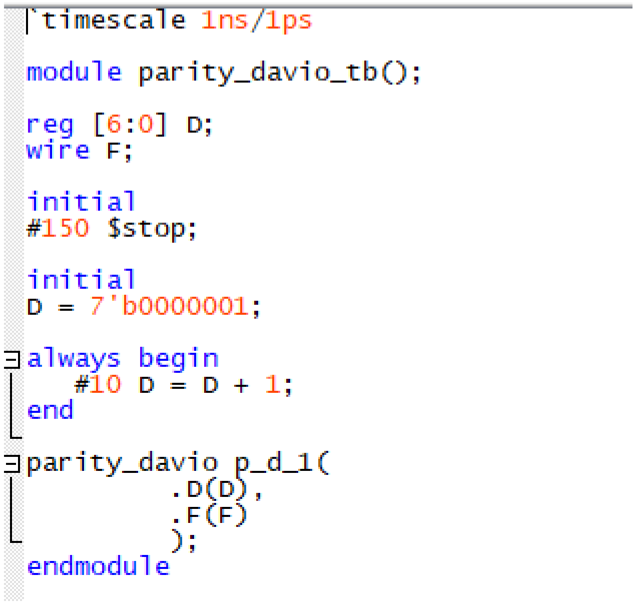
```
`timescale 1ns/1ps

module parity_conventional_tb();

reg [6:0] D;
wire F;

initial
#150 $stop;

initial
D = 7'b0000001;

always begin
    #10 D = D + 1;
end

parity_conventional p_c_1(
            .D(D),
            .F(F)
            );
endmodule
```

Which provided a wave form of



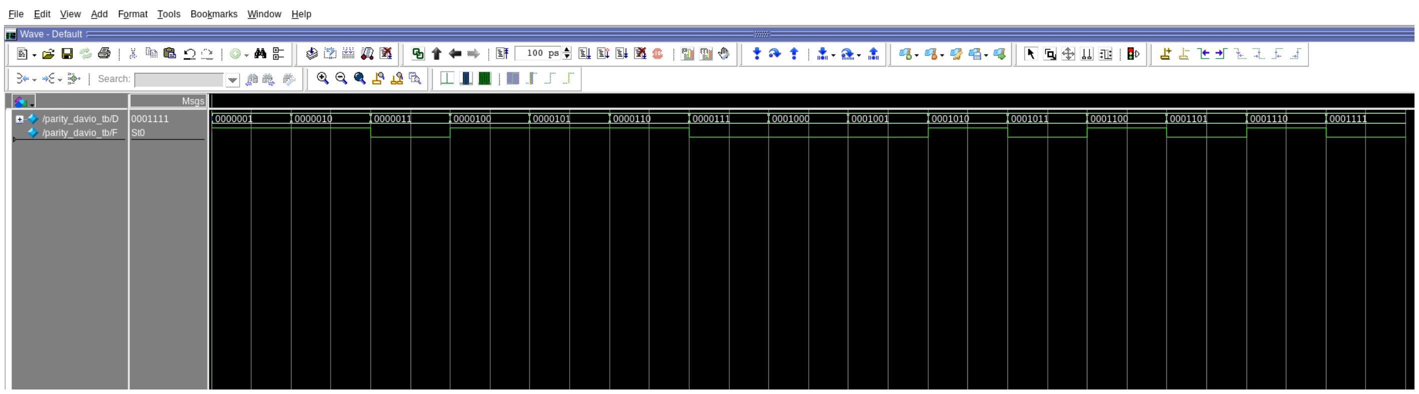
```
/parity_conventio...  0001111  0000001 0000010 0000011 0000100 0000101 0000110 0000111 0001000 0001001 0001010 0001011 0001100 0001101 0001110 0001111
/parity_conventio...  St0
```

Now, the testbench that I used for the Davio was similar to that of the conventional



```
`timescale 1ns/1ps

module parity_davio_tb();

reg [6:0] D;
wire F;

initial
#150 $stop;

initial
D = 7'b0000001;

always begin
    #10 D = D + 1;
end

parity_davio p_d_1(
            .D(D),
            .F(F)
            );
endmodule
```

And the wave for that come from it was as follows



```
/parity_davio_tb/D   0001111   0000001 0000010 0000011 0000100 0000101 0000110 0000111 0001000 0001001 0001010 0001011 0001100 0001101 0001110 0001111
/parity_davio_tb/F   St0
```

As you can see, both wave forms are one to one which means that they both provide the same logic

Question 5

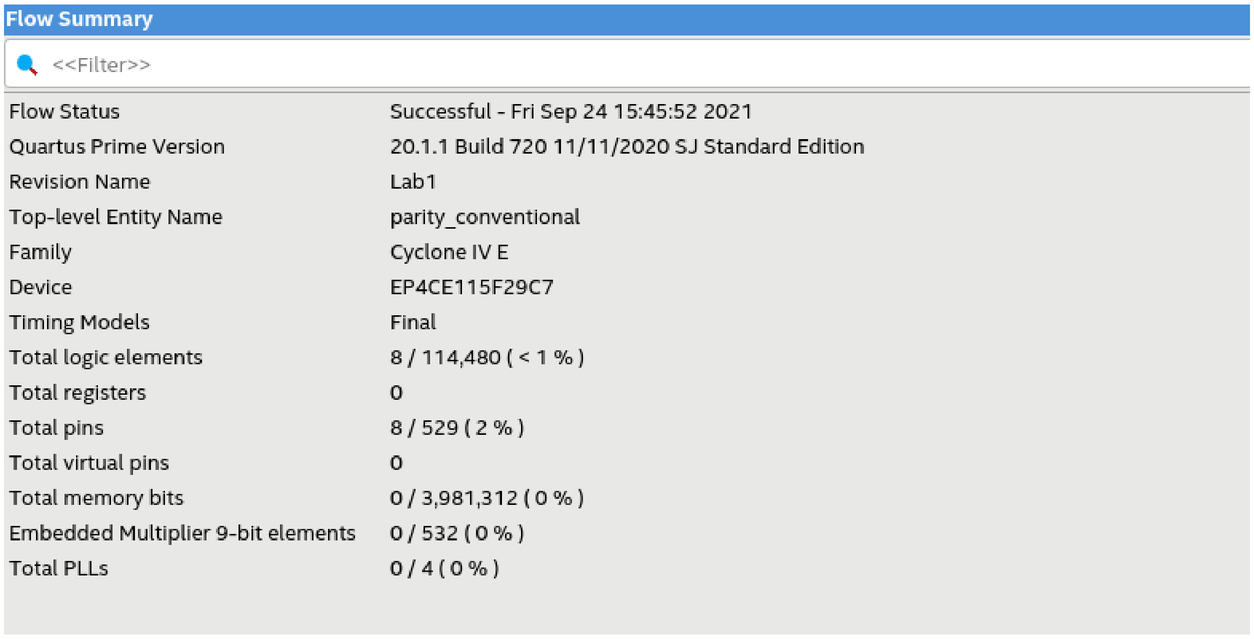
Area Optimization – Reduce the device area required to implement the design at the cost of performance

Power Optimization – Optimize synthesis for low power but will reduce speed performance

Performance Optimization – Optimizes speed performance at the cost of synthesis of run time and increased device resources

Basic Optimization

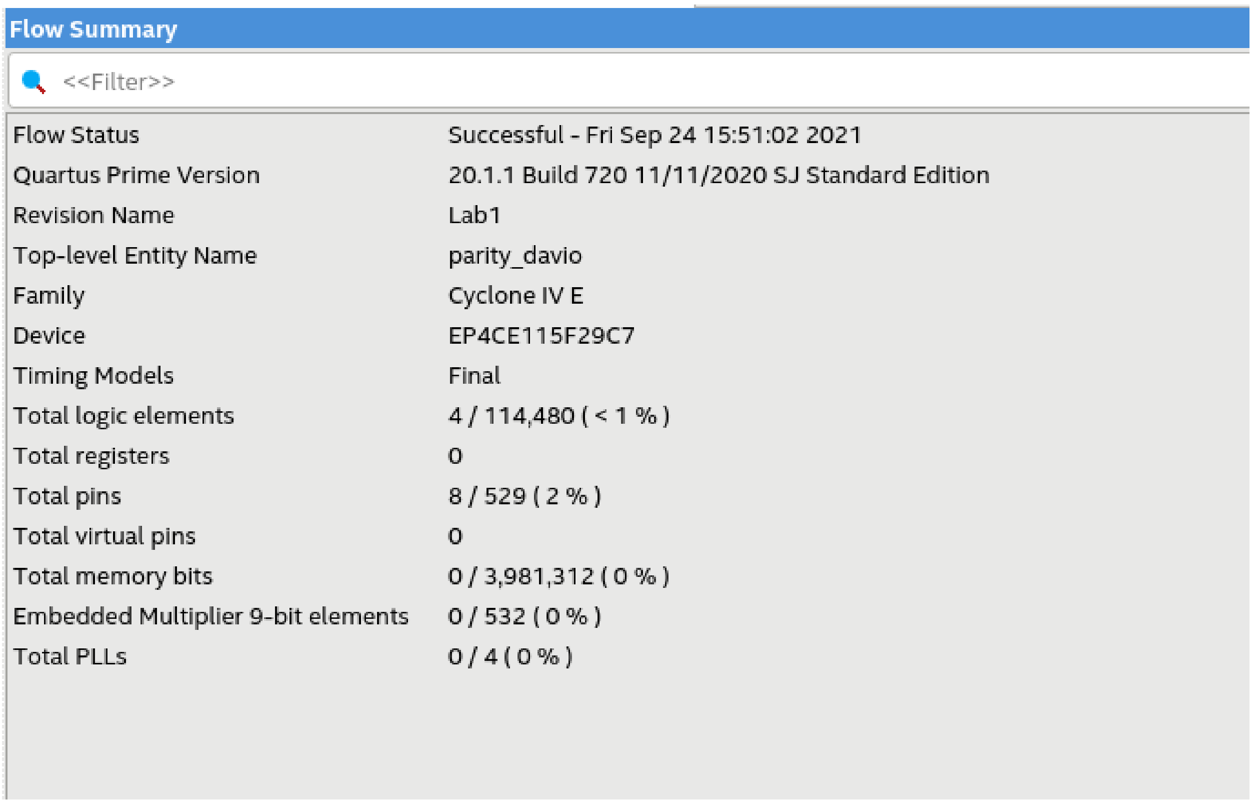
The following will be the basic optimization for the conventional and davio's method, respectively.



Flow Summary

<<Filter>>

Flow Status	Successful - Fri Sep 24 15:45:52 2021
Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Standard Edition
Revision Name	Lab1
Top-level Entity Name	parity_conventional
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	8 / 114,480 (< 1 %)
Total registers	0
Total pins	8 / 529 (2 %)
Total virtual pins	0
Total memory bits	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)
Total PLLs	0 / 4 (0 %)



Flow Summary

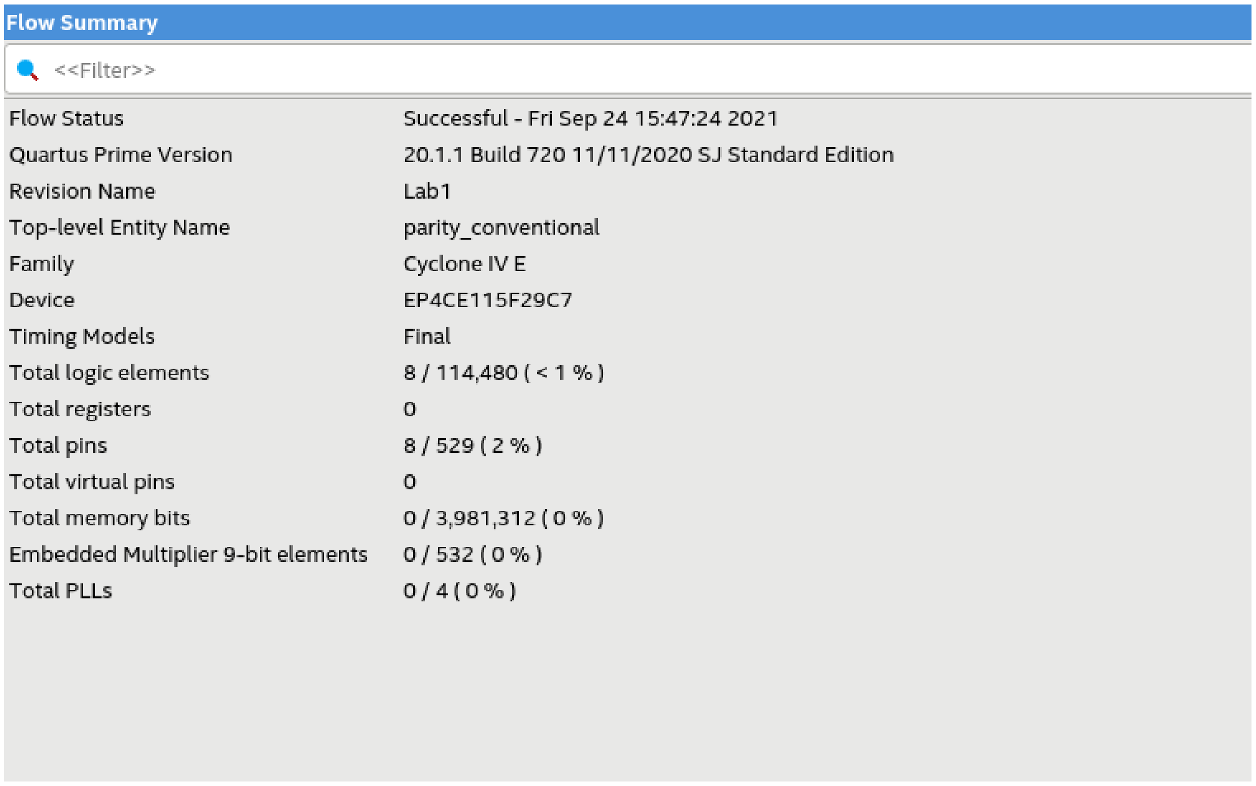
<<Filter>>

Flow Status	Successful - Fri Sep 24 15:51:02 2021
Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Standard Edition
Revision Name	Lab1
Top-level Entity Name	parity_davio
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	4 / 114,480 (< 1 %)
Total registers	0
Total pins	8 / 529 (2 %)
Total virtual pins	0
Total memory bits	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)
Total PLLs	0 / 4 (0 %)

As you can see, the only difference is the number of total logic elements that were used. The CPU time for the conventional method was 27 seconds and the elapsed time 14 seconds. Whereas the CPU time for Davio's method was 27 seconds and the elapse time was 15 seconds.

Area Optimization

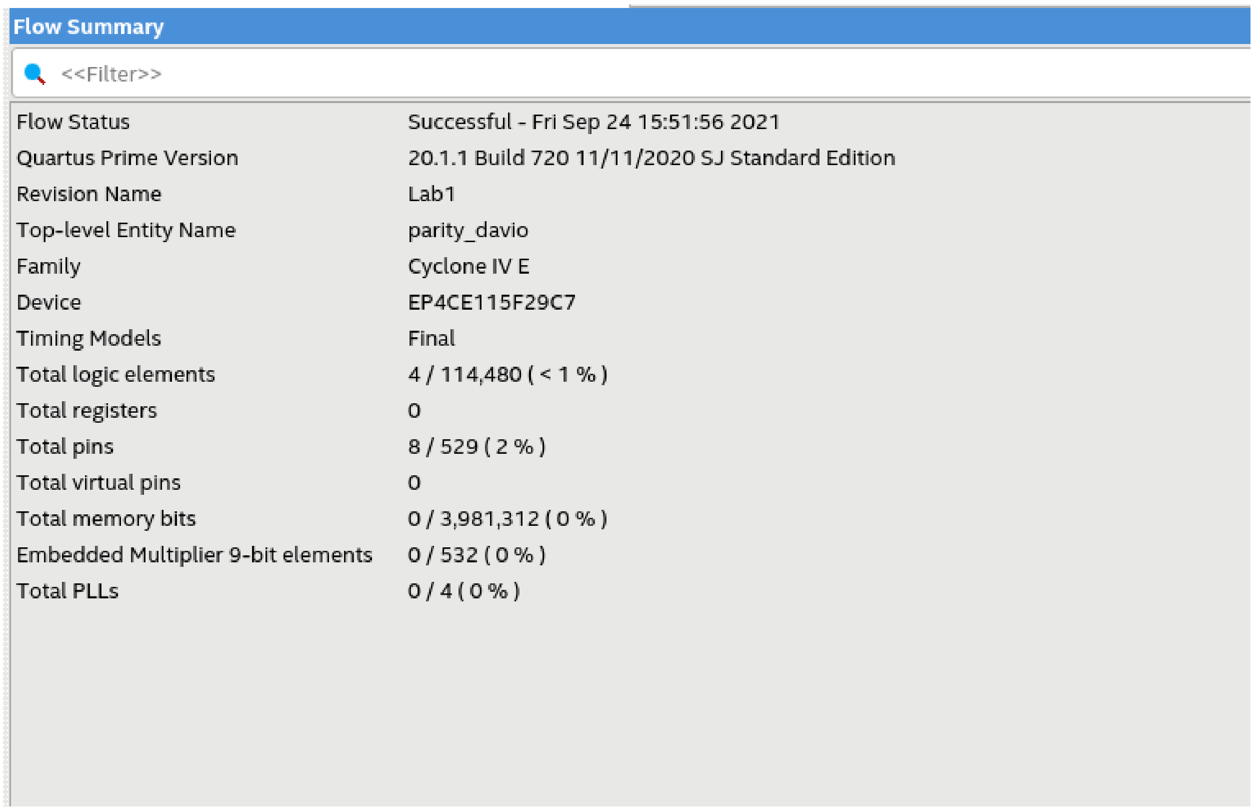
Now for the area optimization option provided by Quartus. The first picture will be conventional, and the following will be Davio's.



Flow Summary

<<Filter>>

Flow Status	Successful - Fri Sep 24 15:47:24 2021
Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Standard Edition
Revision Name	Lab1
Top-level Entity Name	parity_conventional
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	8 / 114,480 (< 1 %)
Total registers	0
Total pins	8 / 529 (2 %)
Total virtual pins	0
Total memory bits	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)
Total PLLs	0 / 4 (0 %)


Flow Summary

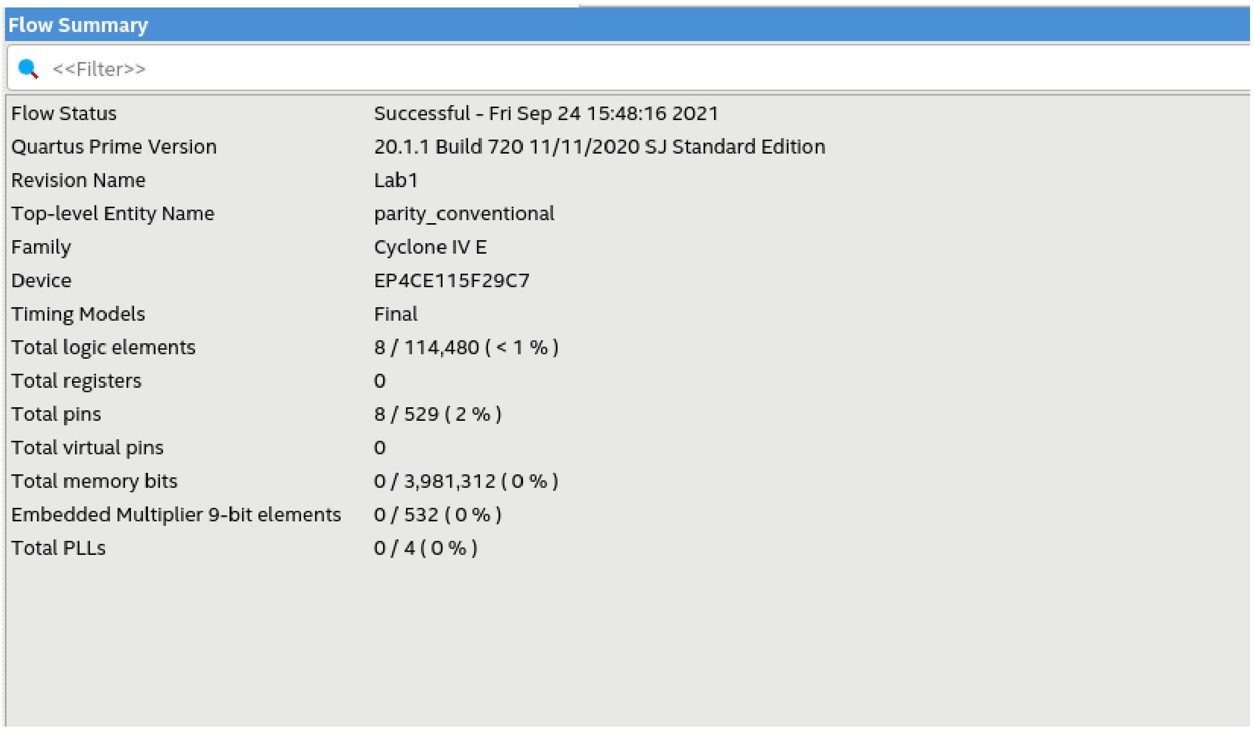
<<Filter>>

Flow Status	Successful - Fri Sep 24 15:51:56 2021
Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Standard Edition
Revision Name	Lab1
Top-level Entity Name	parity_davio
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	4 / 114,480 (< 1 %)
Total registers	0
Total pins	8 / 529 (2 %)
Total virtual pins	0
Total memory bits	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)
Total PLLs	0 / 4 (0 %)

Again, the only difference in the flow summary is the number of total logic elements used. The CPU time for conventional was 27 seconds and the elapsed time was 15 seconds. Whereas the CPU time for Davio was 27 seconds and the elapsed time of 13 seconds

Power Optimization

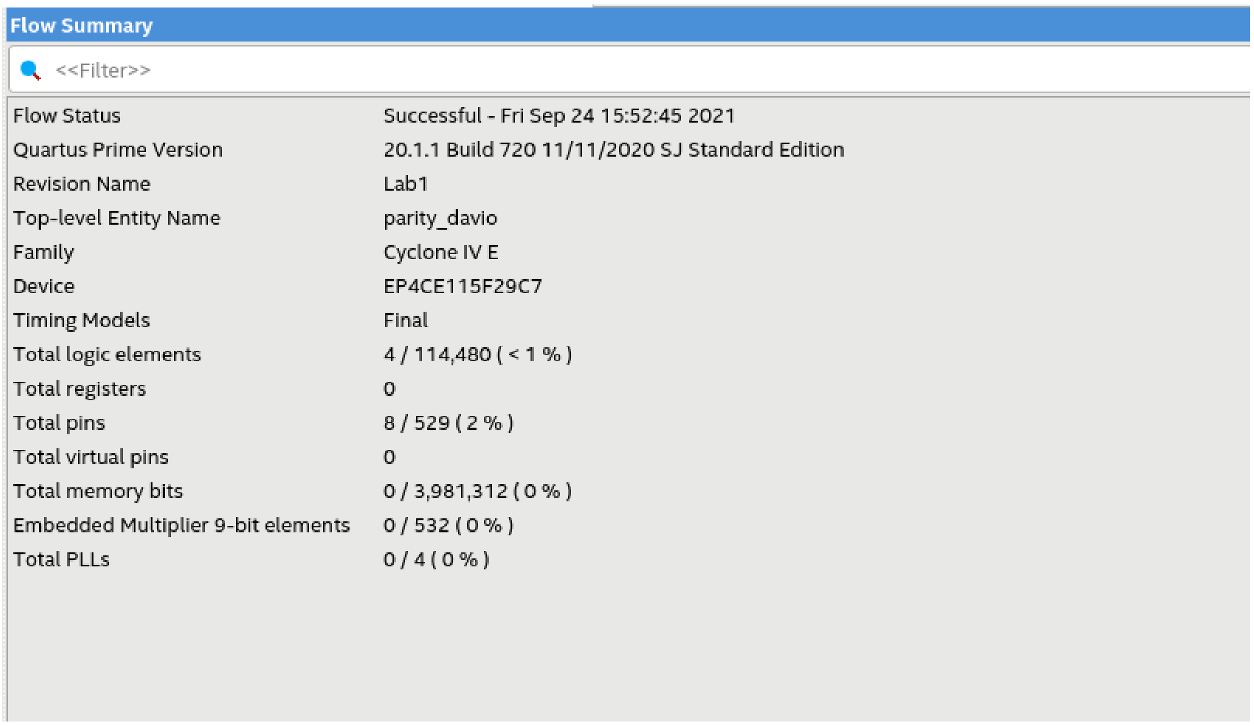
The next set of photos will be with the Power optimization in Quartus. The first picture will be the conventional and the second will be for Davio's.



Flow Summary

<<Filter>>

Flow Status	Successful - Fri Sep 24 15:48:16 2021
Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Standard Edition
Revision Name	Lab1
Top-level Entity Name	parity_conventional
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	8 / 114,480 (< 1 %)
Total registers	0
Total pins	8 / 529 (2 %)
Total virtual pins	0
Total memory bits	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)
Total PLLs	0 / 4 (0 %)



Flow Summary

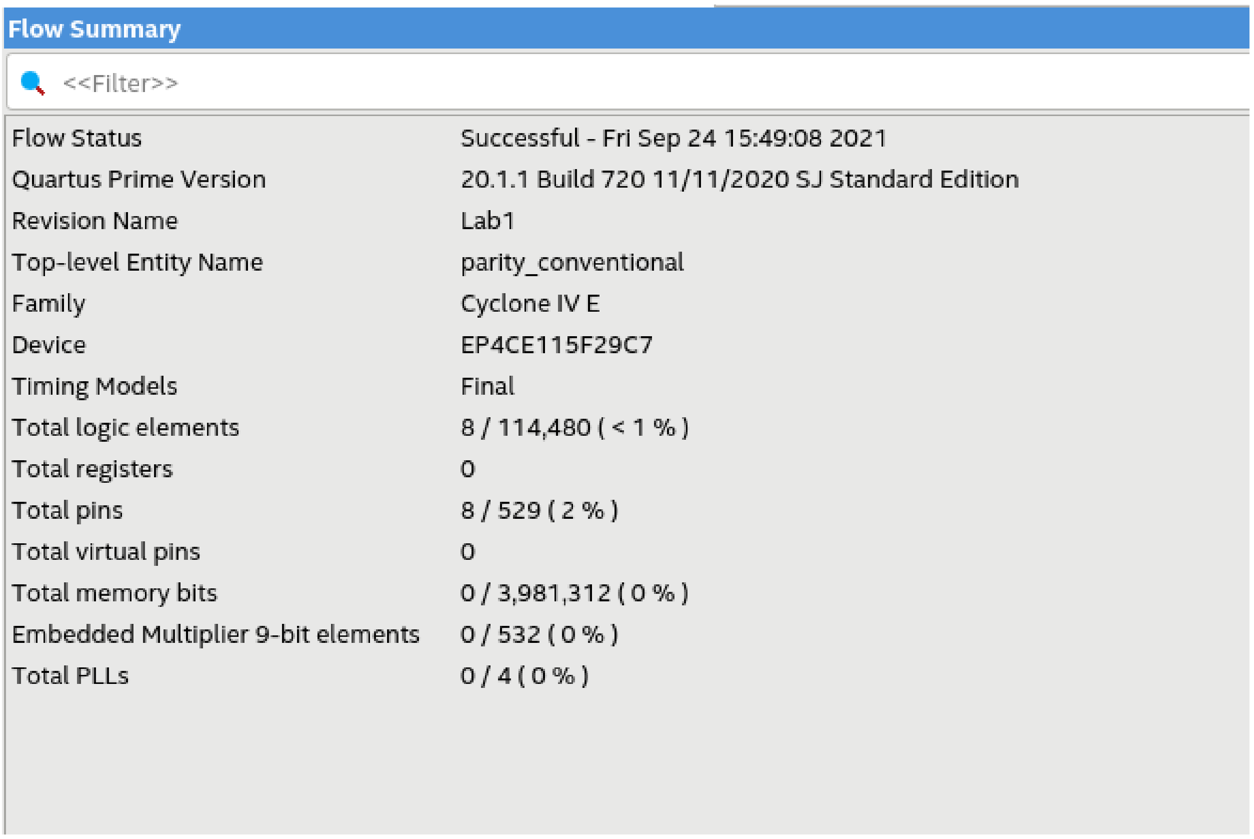
<<Filter>>

Flow Status	Successful - Fri Sep 24 15:52:45 2021
Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Standard Edition
Revision Name	Lab1
Top-level Entity Name	parity_davio
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	4 / 114,480 (< 1 %)
Total registers	0
Total pins	8 / 529 (2 %)
Total virtual pins	0
Total memory bits	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)
Total PLLs	0 / 4 (0 %)

The only difference between the two is again the number of total logic elements used. The CPU time for the conventional method was 33 seconds with an elapsed time of 18 seconds. The CPU time for Davio's method was 34 seconds with an elapsed time of 19 seconds.

Performance Optimization

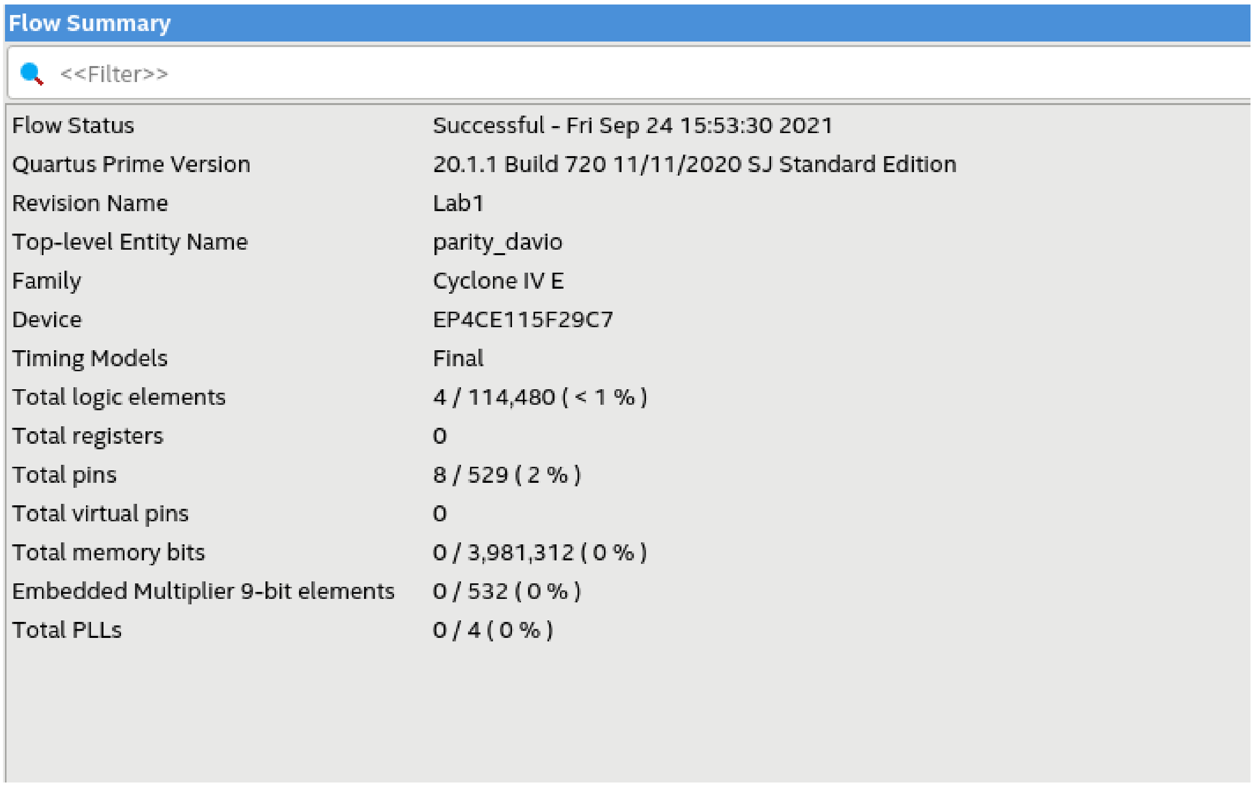
The last optimization that was used was the performance optimization provided by Quartus. Once again, it will go the conventional then Davio's



Flow Summary

<<Filter>>

Flow Status	Successful - Fri Sep 24 15:49:08 2021
Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Standard Edition
Revision Name	Lab1
Top-level Entity Name	parity_conventional
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	8 / 114,480 (< 1 %)
Total registers	0
Total pins	8 / 529 (2 %)
Total virtual pins	0
Total memory bits	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)
Total PLLs	0 / 4 (0 %)



Flow Status	Successful - Fri Sep 24 15:53:30 2021
Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Standard Edition
Revision Name	Lab1
Top-level Entity Name	parity_davio
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	4 / 114,480 (< 1 %)
Total registers	0
Total pins	8 / 529 (2 %)
Total virtual pins	0
Total memory bits	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)
Total PLLs	0 / 4 (0 %)

Lastly, the only difference is the total logic elements. The CPU time for the conventional method was 32 seconds and the elapsed time was 19 seconds. The CPU time for Davio's method was 33 seconds and the elapsed time was 18 seconds.

Conclusion

As a conclusion, I am aware what each optimization does, and it seems that Quartus has already optimized the design to its fullest content to the point that changing the optimization configurations do little to nothing to the entire design. The only major difference that I could find between each optimization mode was the CPU and elapsed time, which was mentioned before. As designs become more complicated and complex, the usefulness of the different optimization modes starts to show.