EE 599

Homework 1

Ruizhi Zhang

8230108665

Link:

https://github.com/ruizhiz/-EE-599\_Ruizhi\_Zhang\_8230108665.git

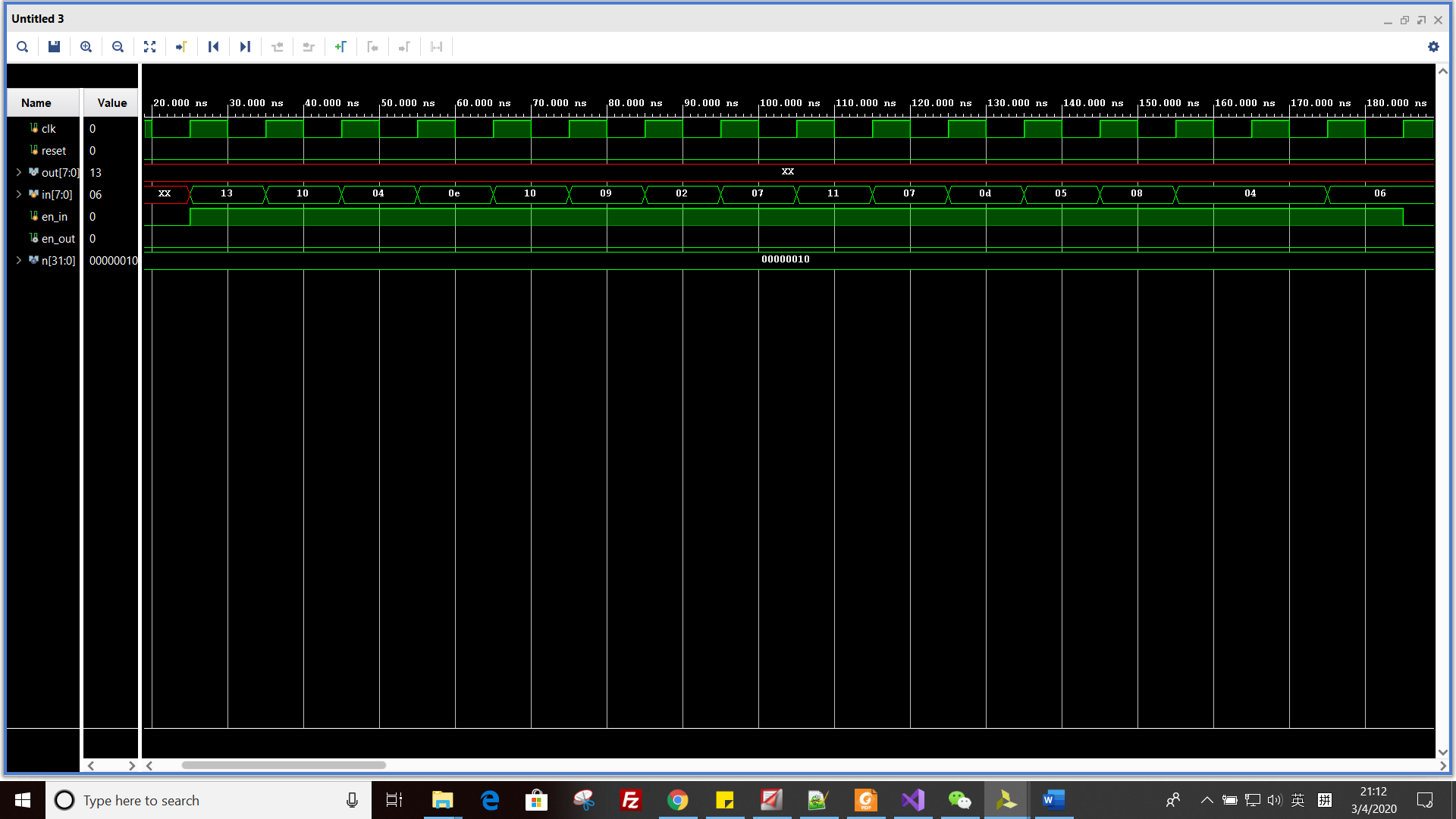
**Odd-even transposition sort**

The Odd\_even.v and Odd\_even\_tb.v have been attached.

We do not use BRAM. All the data get from input directly, and then stored into registers inside circuits. Finishing comparing with each other, all of them will be output one by one.

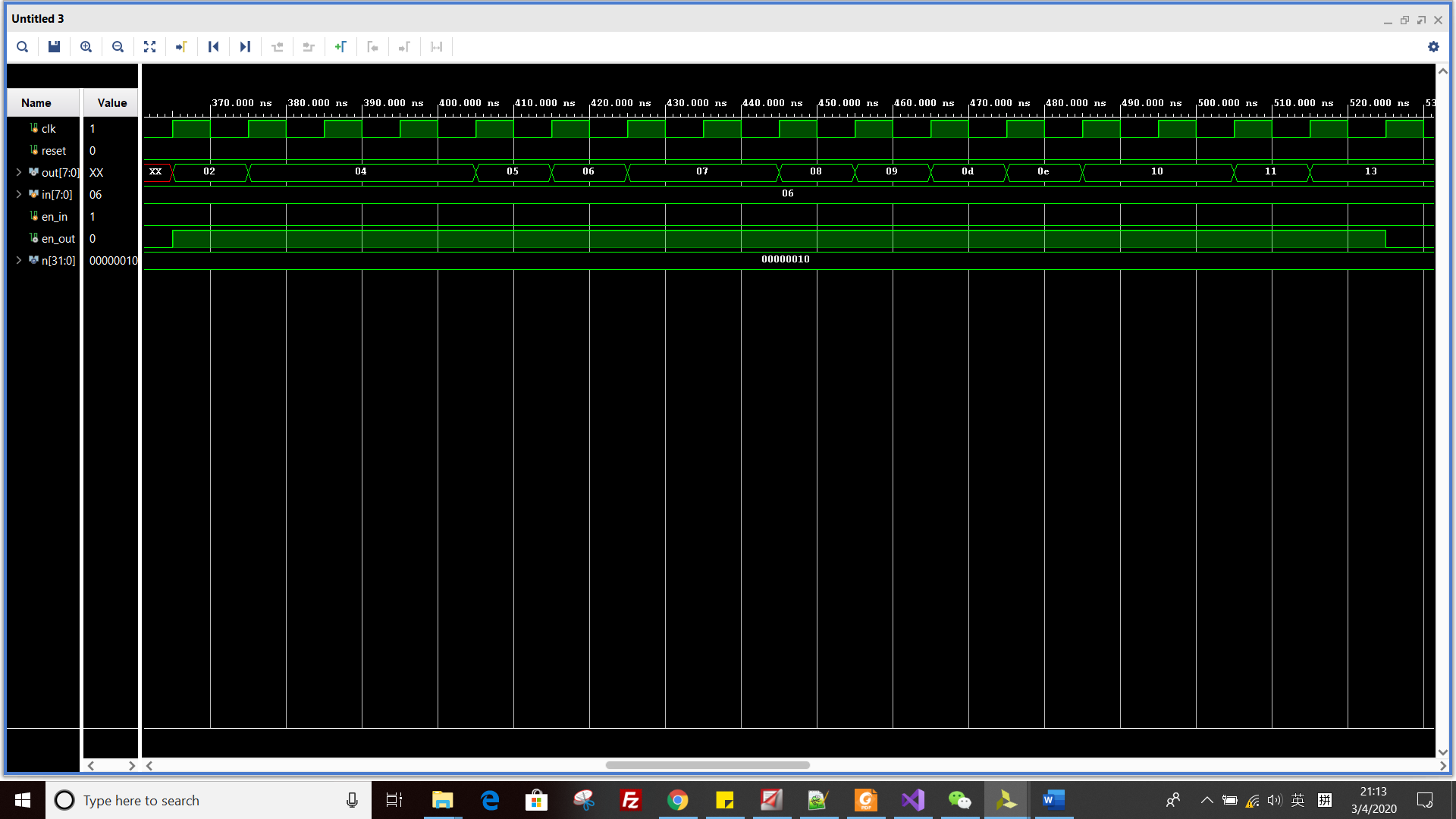
1. **simulation**

In Odd\_even\_tb.v, the input is generated by *$urandom%20;,* which means the input is [0, 20] unsigned random number.



*Picture 1: the waveform of input for 16 elements*.

Based on the picture above, the input is 13, 10, 04, 0e, 10, 09, 02, 07, 11, 07, 0d, 05, 08, 04, 04, 06. All of them are Hexadecimal.



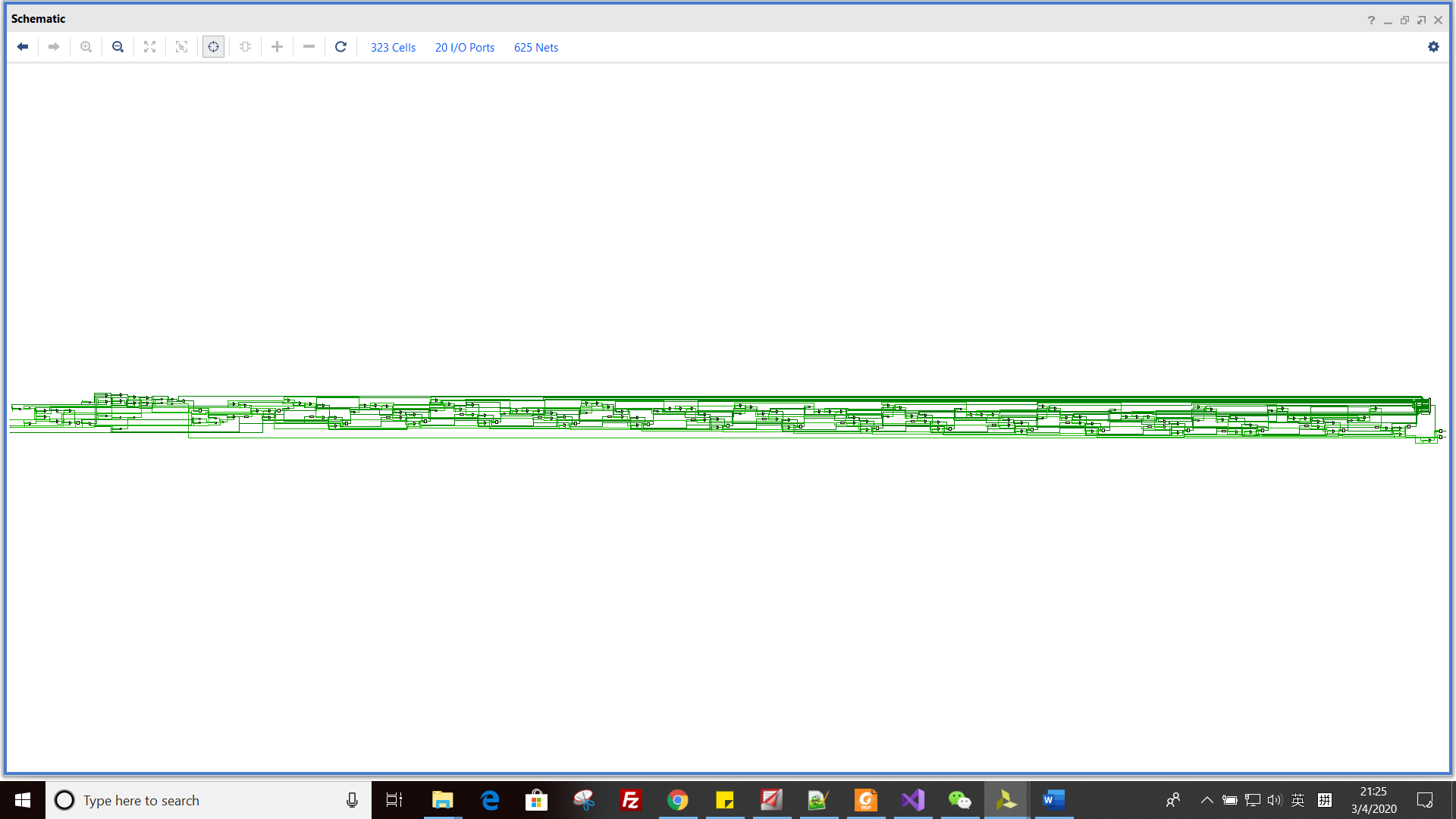
*Picture 2: the waveform of output for 16 elements*

The output order is 02, 04, 04, 04, 05, 06, 07, 07, 08, 09, 0d, 0e, 10, 10, 11, 13. All of them are Hexadecimal.

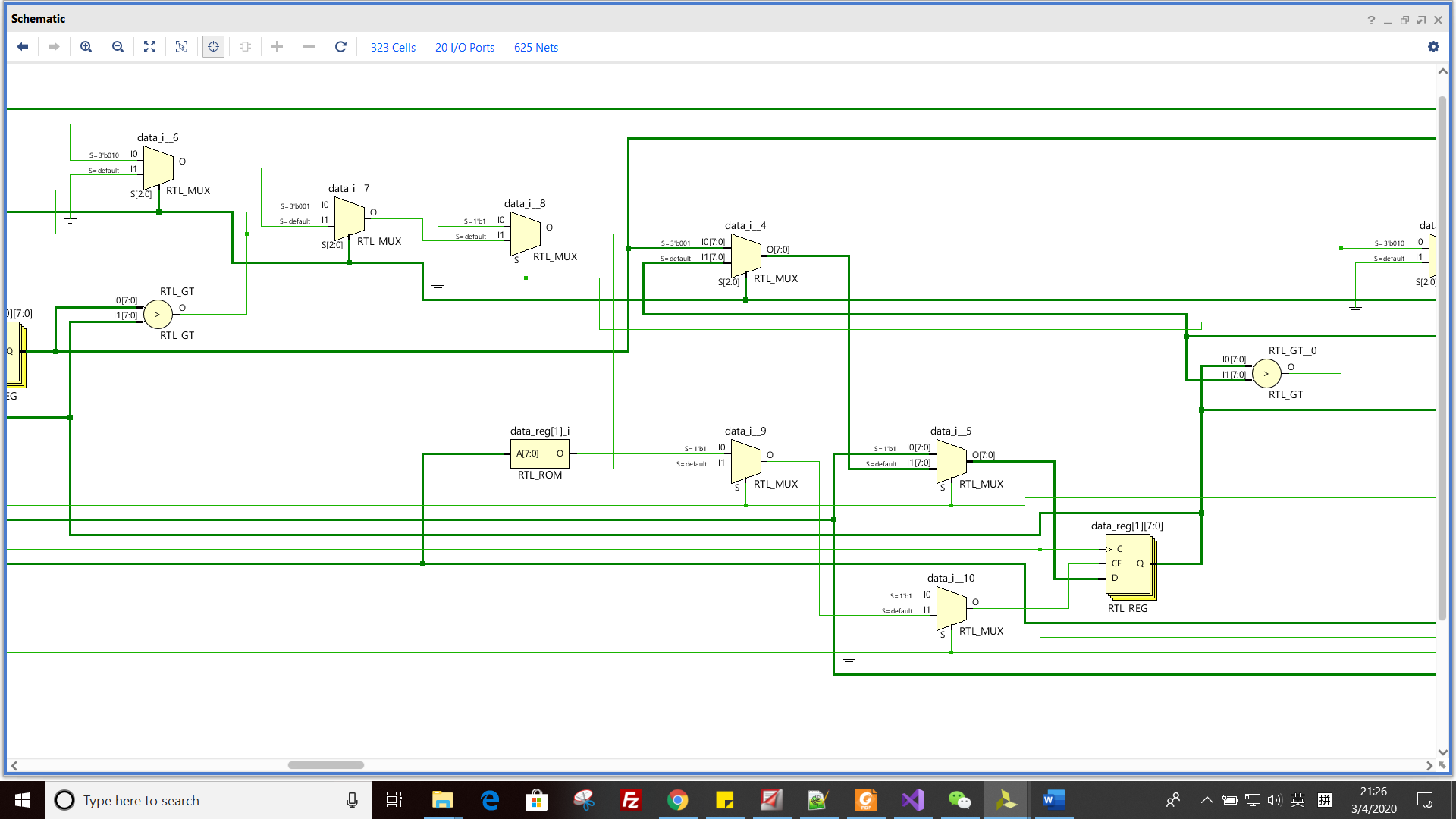
As we can see, the input has been sorted.

**Part 1: 16 elements**

1. **Schematics.**



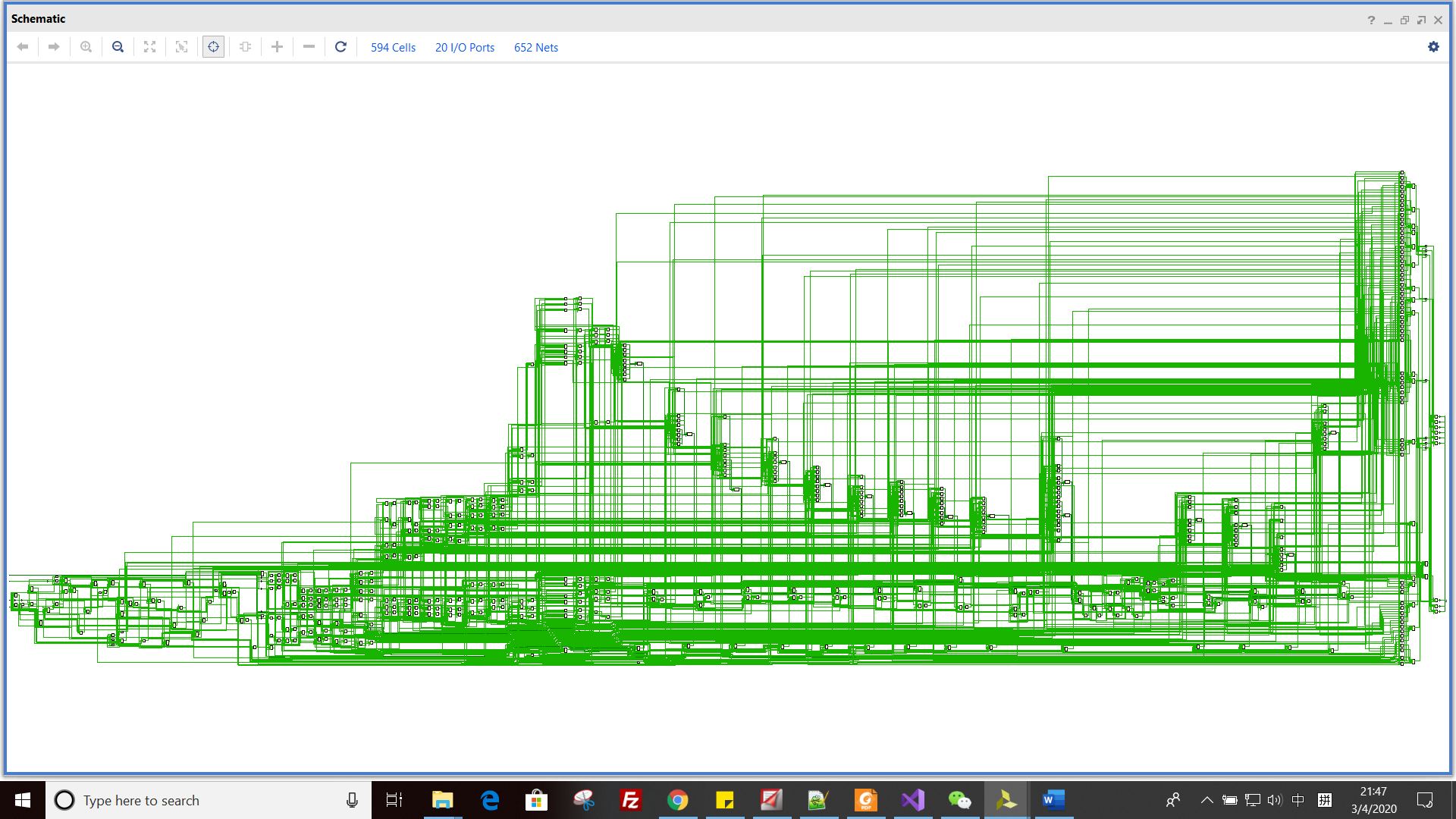
*Picture 3: the schematics of our design*



*Picture 4: the basic unit of our design*

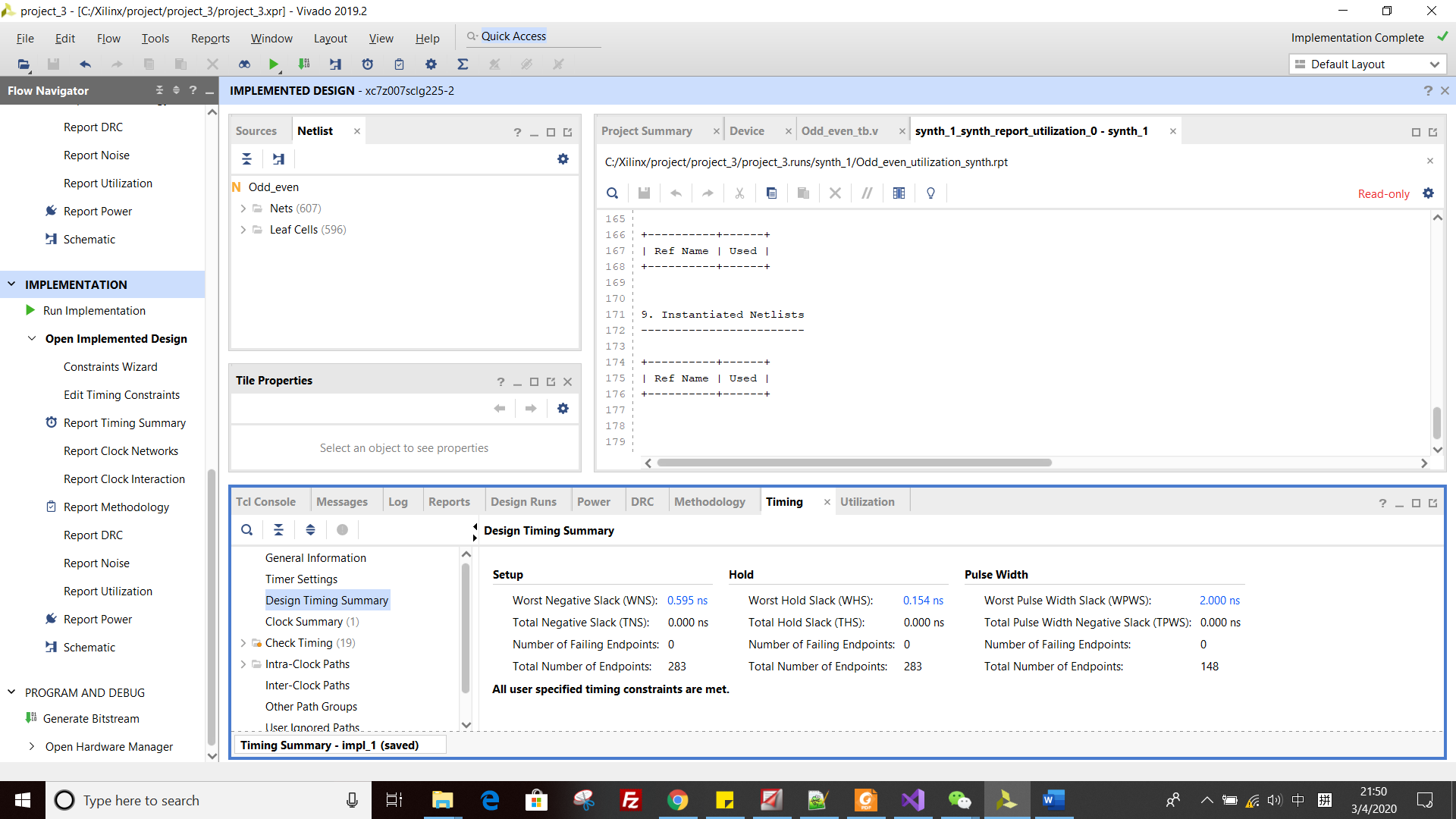
Because our design has 16 elements, they compare to each other. So, we have 16 above basic units in total, them combine to be picture 3.

1. **Synthesis.**

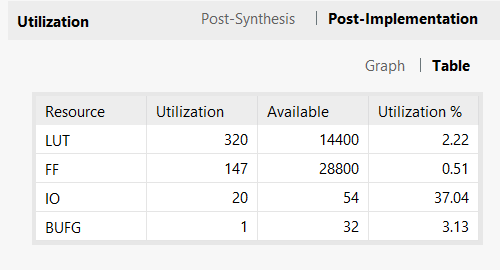


*Picture 5: synthesis design*

1. **Resource estimation and timing estimation**



*Picture 6: timing estimation*



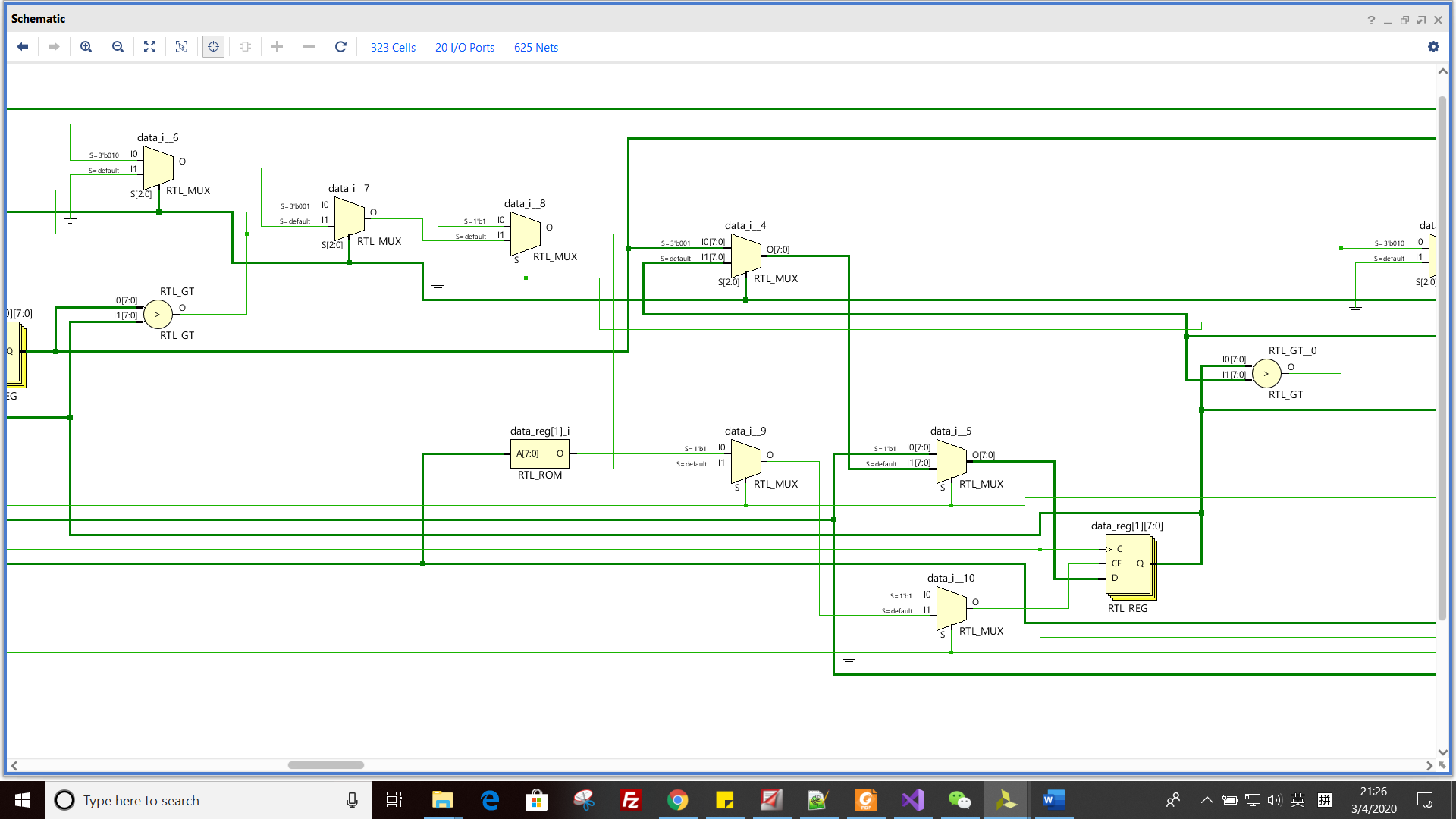
*Picture 7: resource estimation*

**Part 2: 32 elements**

1. **Schematics.**



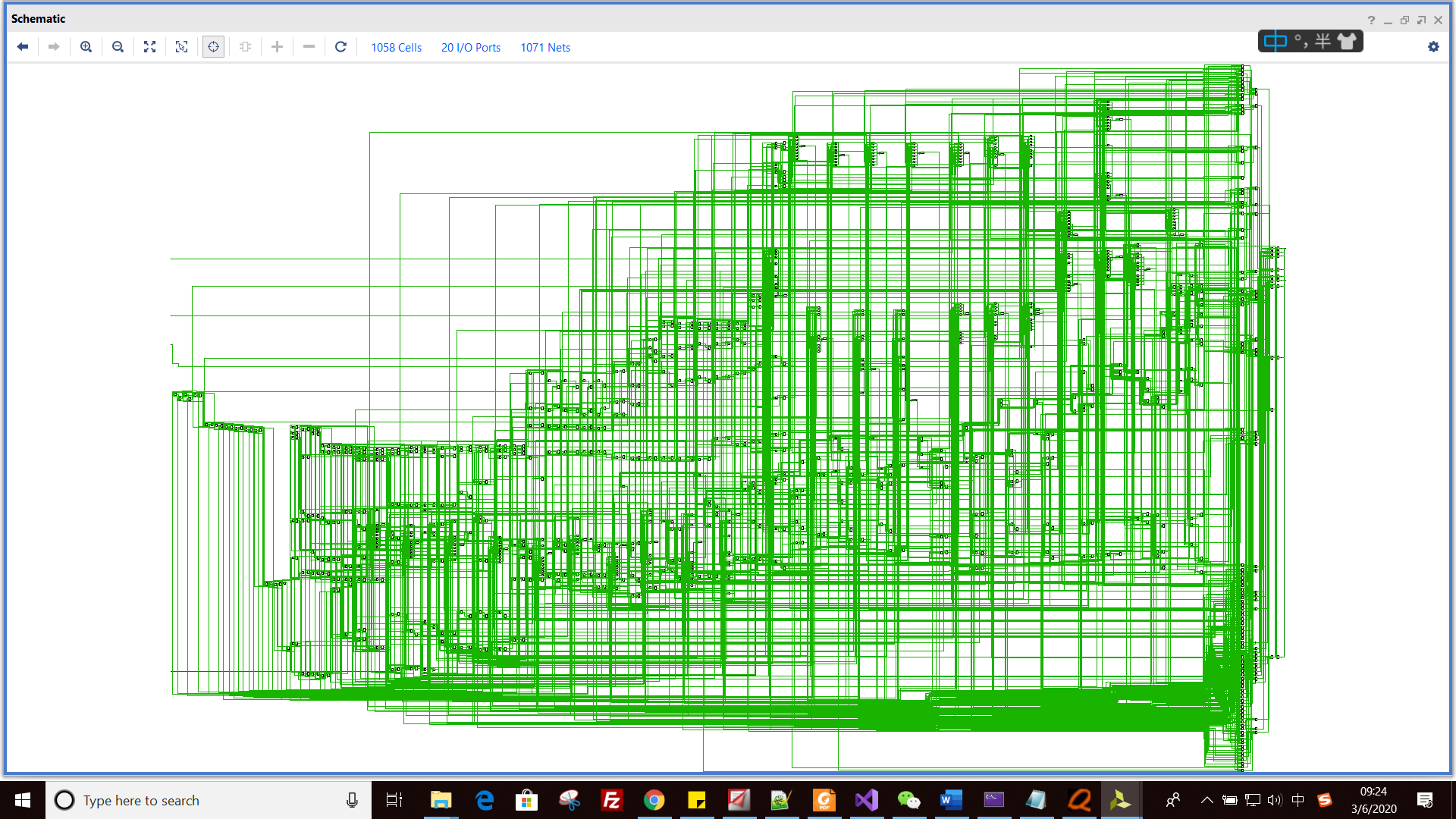
*Picture 8: the schematics of our design*



*Picture 9: the basic unit of our design*

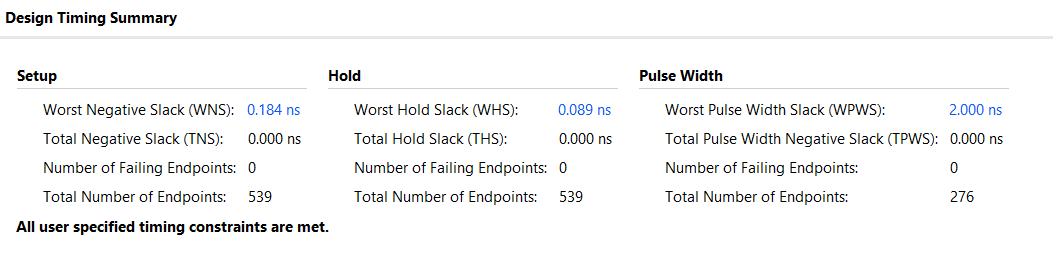
Because our design has 32 elements, they compare to each other. So, we have 32 above basic units in total, them combine to be picture 3.

1. **Synthesis.**

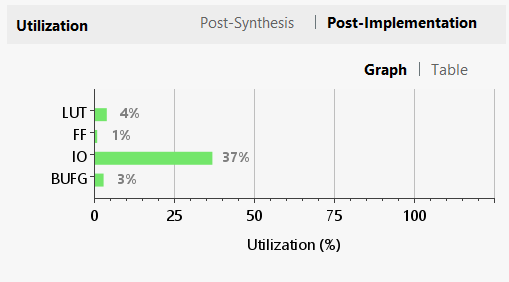


*Picture 10: synthesis design*

1. **Resource estimation and timing estimation**



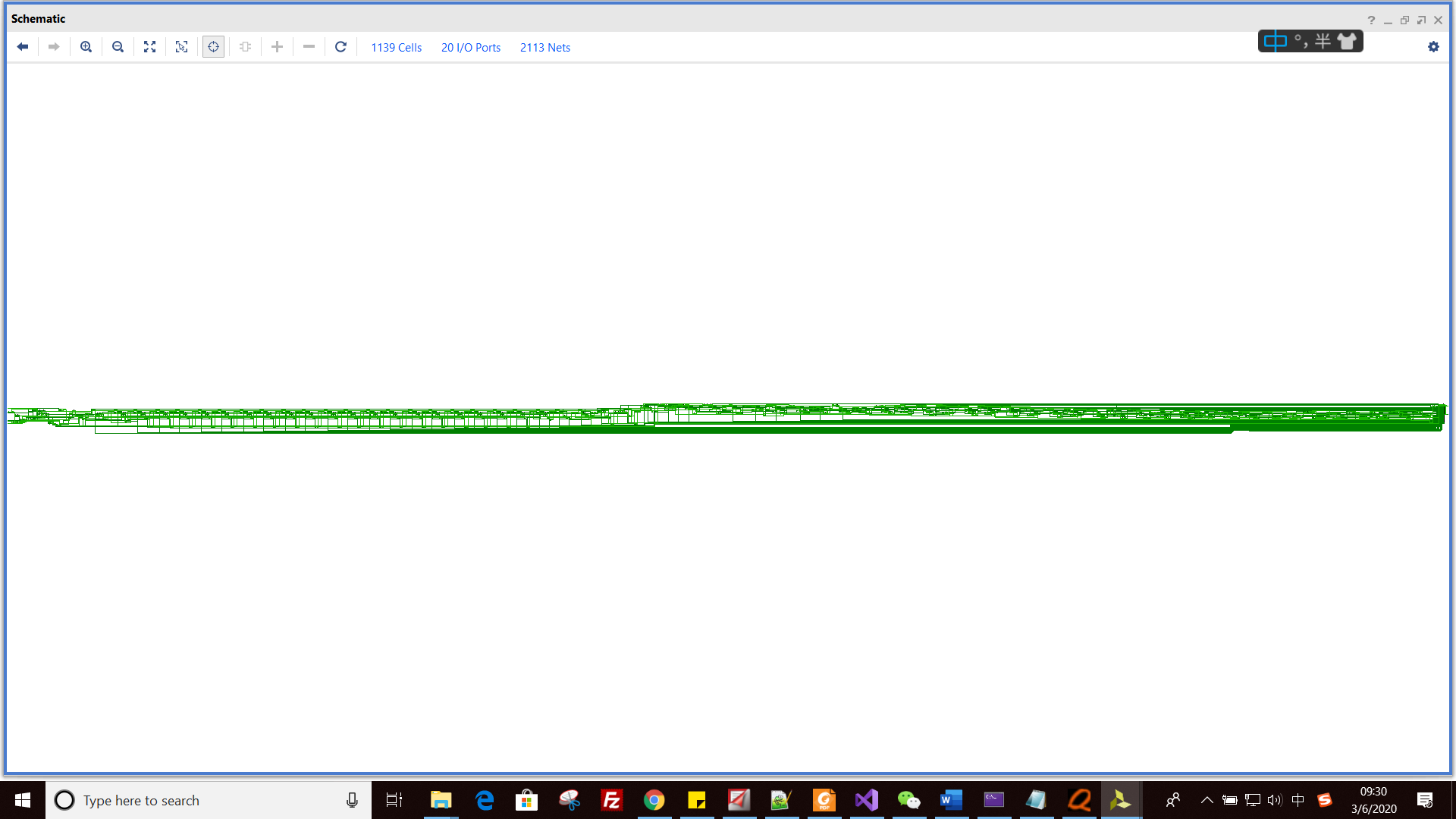
*Picture 11: timing estimation*



*Picture 12: resource estimation*

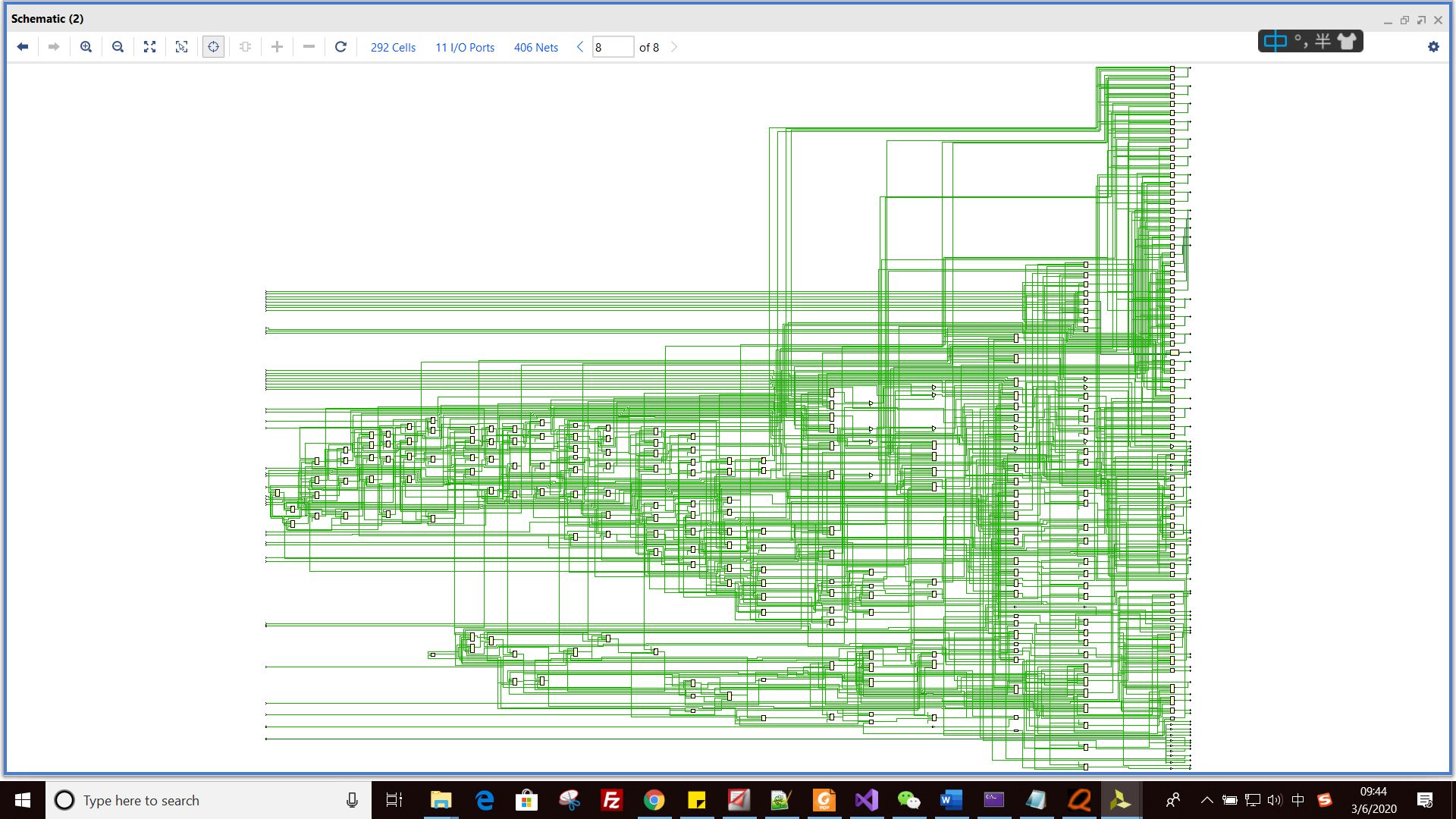
**Part 3: 64 elements**

1. **Schematics.**



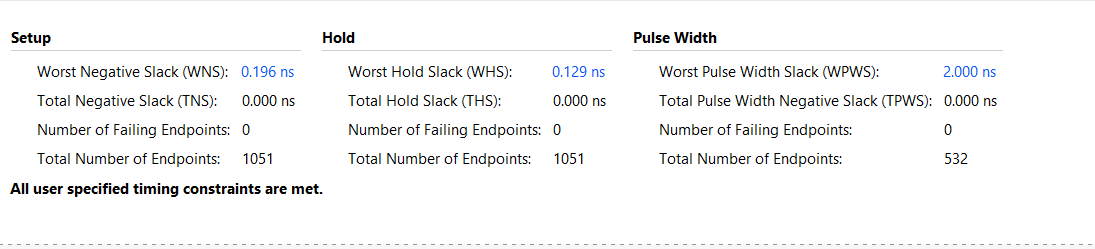
*Picture 13: the schematics of our design*

1. **Synthesis.**

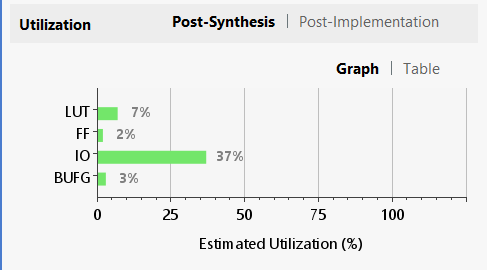


*Picture 14: synthesis design*

1. **Resource estimation and timing estimation**



*Picture 15: timing estimation*



*Picture 16: resource estimation*

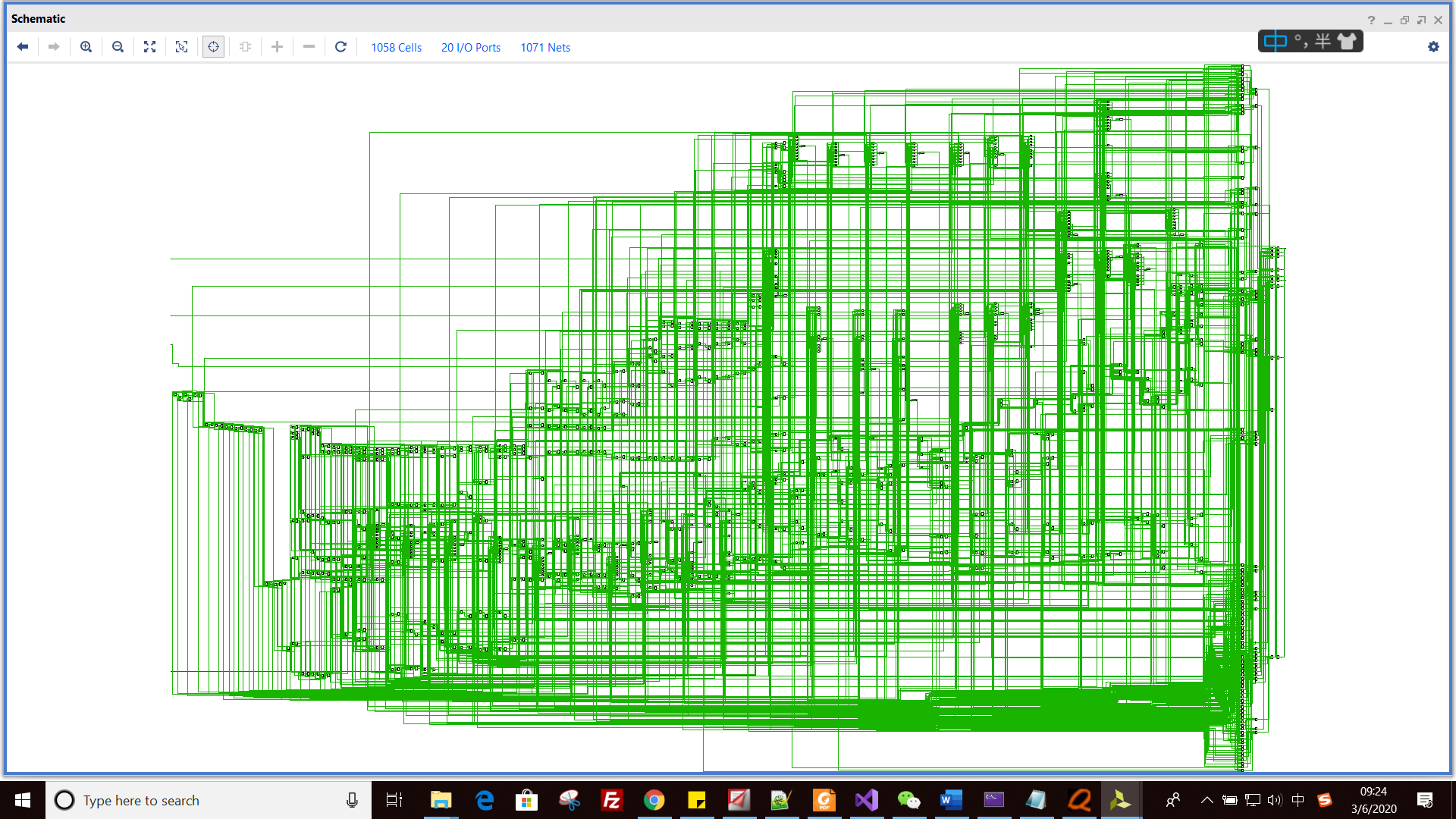
**Part 4: 128 elements**

1. **Schematics.**



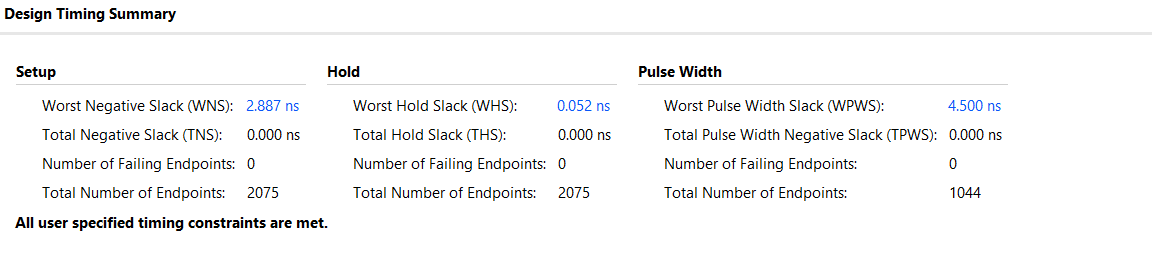
*Picture 17: the schematics of our design*

1. **Synthesis.**

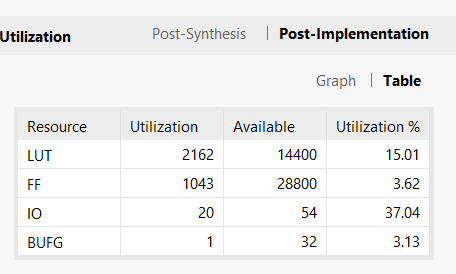


*Picture 14: synthesis design*

1. **Resource estimation and timing estimation**



*Picture 15: timing estimation*



*Picture 16: resource estimation*

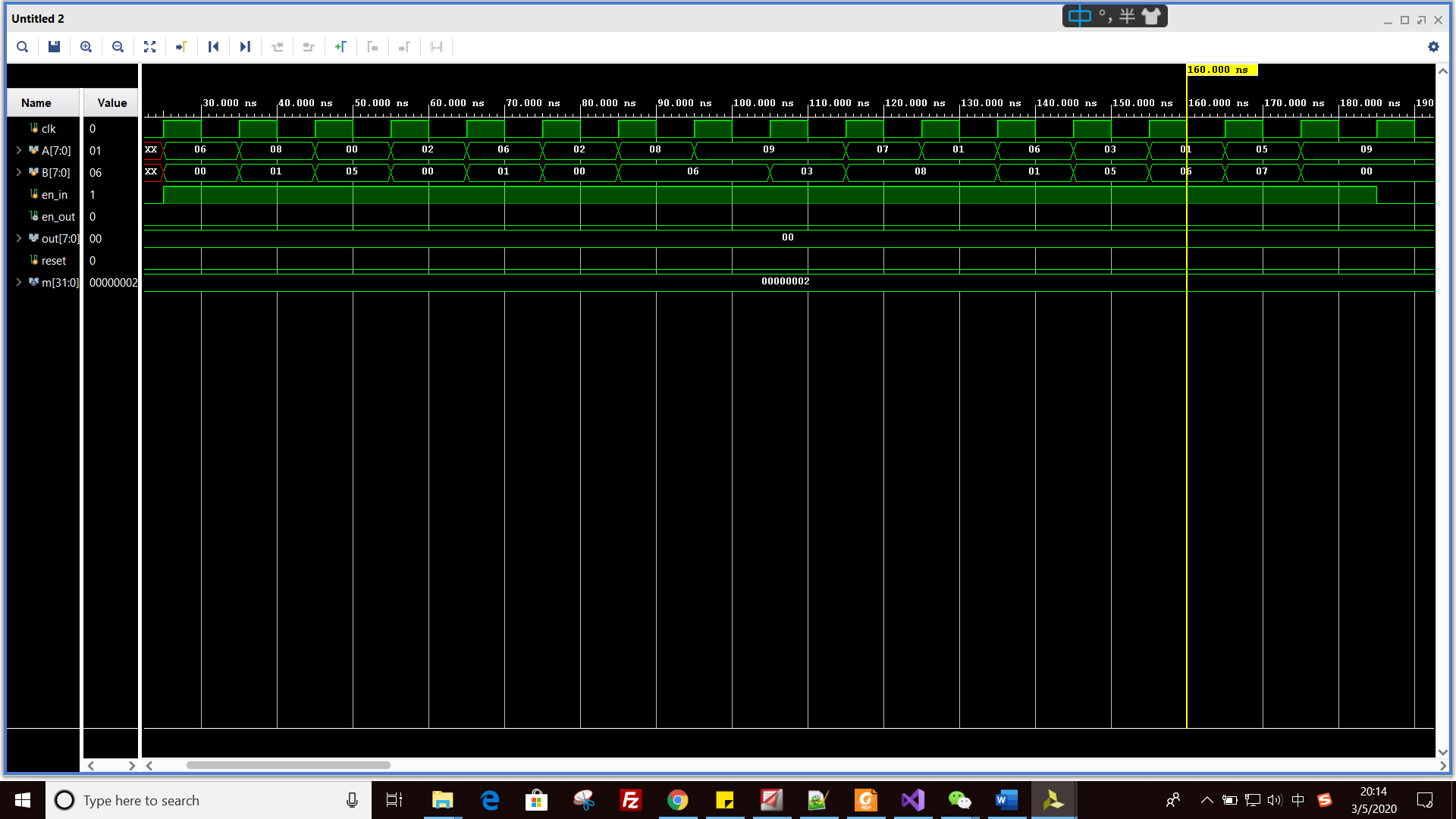
**Dense Matrix – Matrix Multiplication**

M\_mult\_1.v, adder.v, multiply.v, MulandAddTree.v and M\_mult\_tb.v have been attached.

M\_mult\_2.v is used to test the largest number of parallel MulandAddTrees.

BRAM is not used in this design. All the matrix data get from input in several clock cycle.

1. **Simulation**



*Picture 17: the waveform of input*

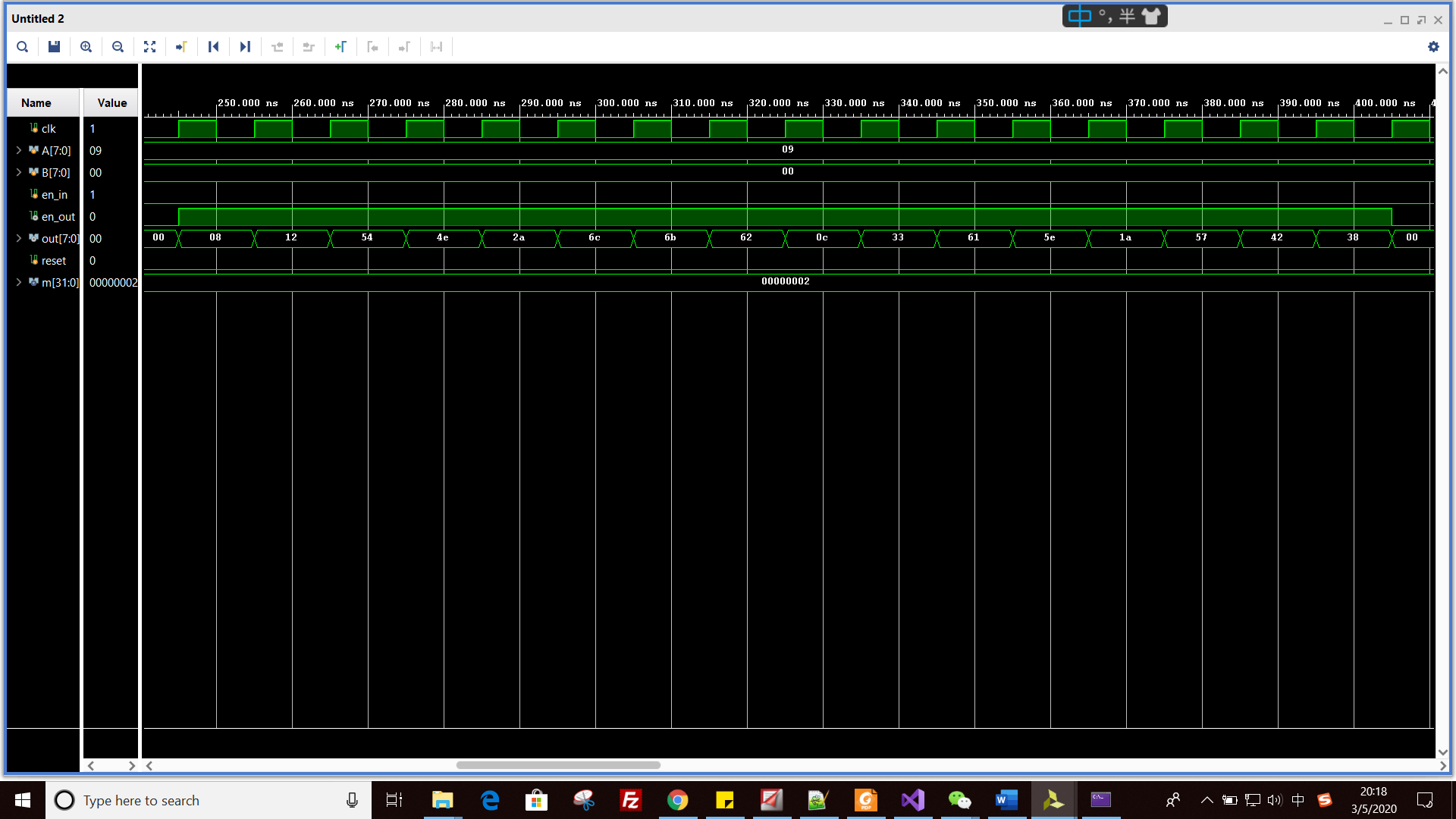
The input is generated by A <= $urandom%10; B <= $urandom%10; Both A and B are random unsigned int from 0 to 10.

As we can see, the Matrixes are:

|  |  |  |  |
| --- | --- | --- | --- |
| 00 | 01 | 03 | 05 |
| 01 | 00 | 08 | 06 |
| 05 | 06 | 08 | 07 |
| 00 | 06 | 01 | 00 |

A: B:

|  |  |  |  |
| --- | --- | --- | --- |
| 06 | 08 | 00 | 02 |
| 06 | 02 | 08 | 09 |
| 09 | 07 | 01 | 06 |
| 03 | 01 | 05 | 09 |



*Picture 18: waveform of output*

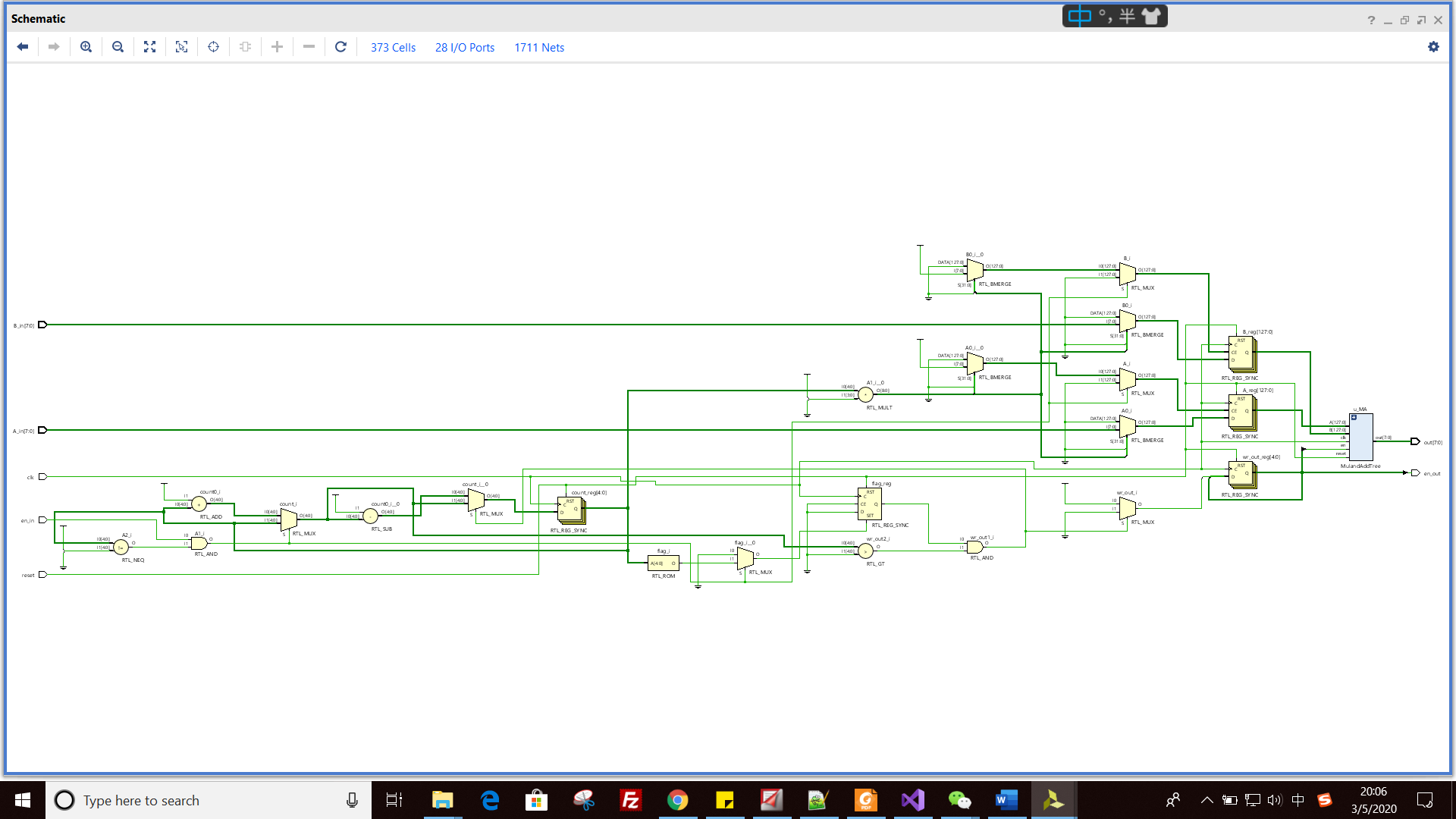
As we can see, the result Matrix is:

Out:

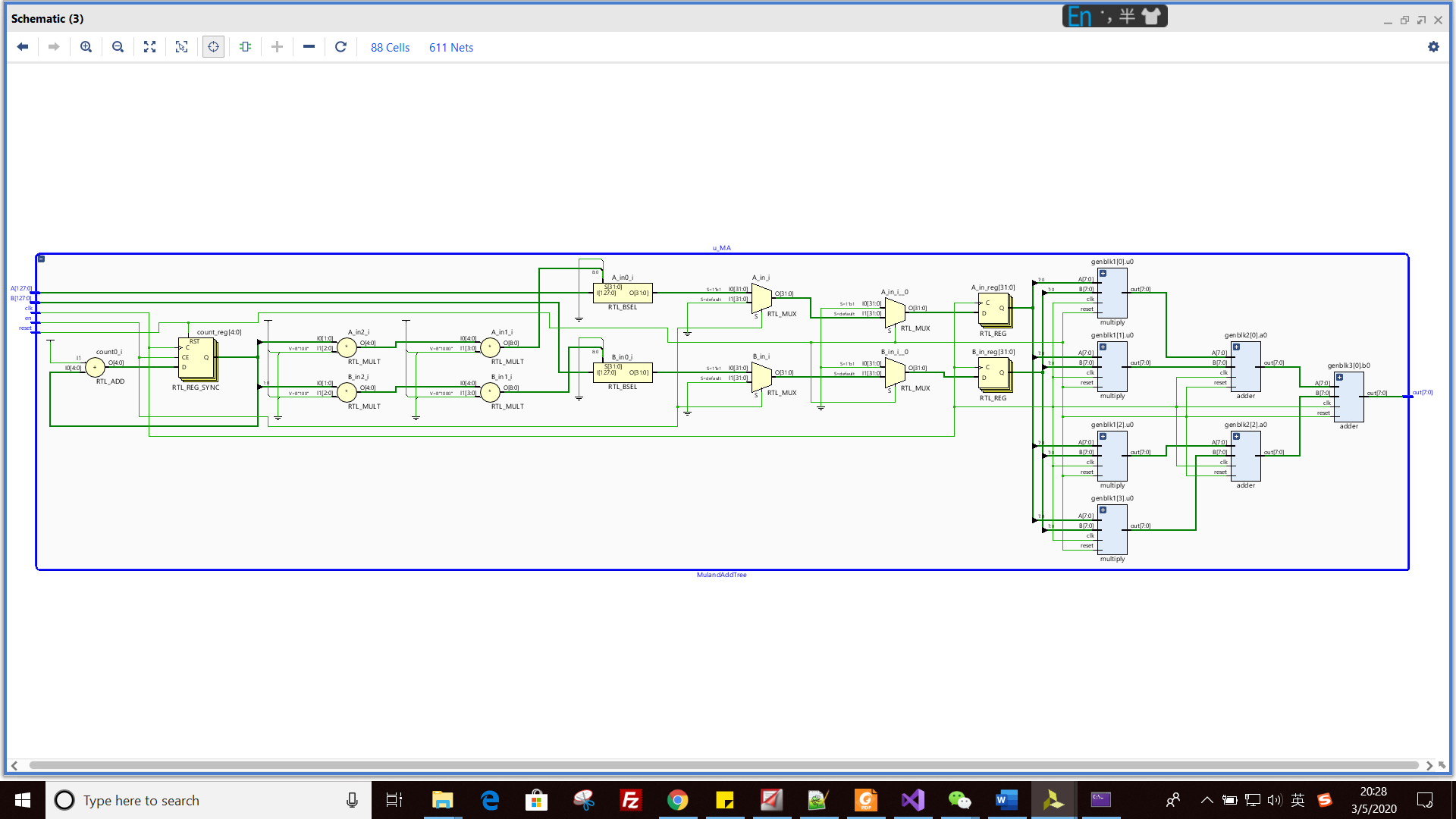
|  |  |  |  |
| --- | --- | --- | --- |
| 8 | 12 | 54 | 4e |
| 2a | 6c | 6b | 62 |
| 0c | 33 | 61 | 5e |
| 1a | 57 | 42 | 38 |

This result is what we expected.

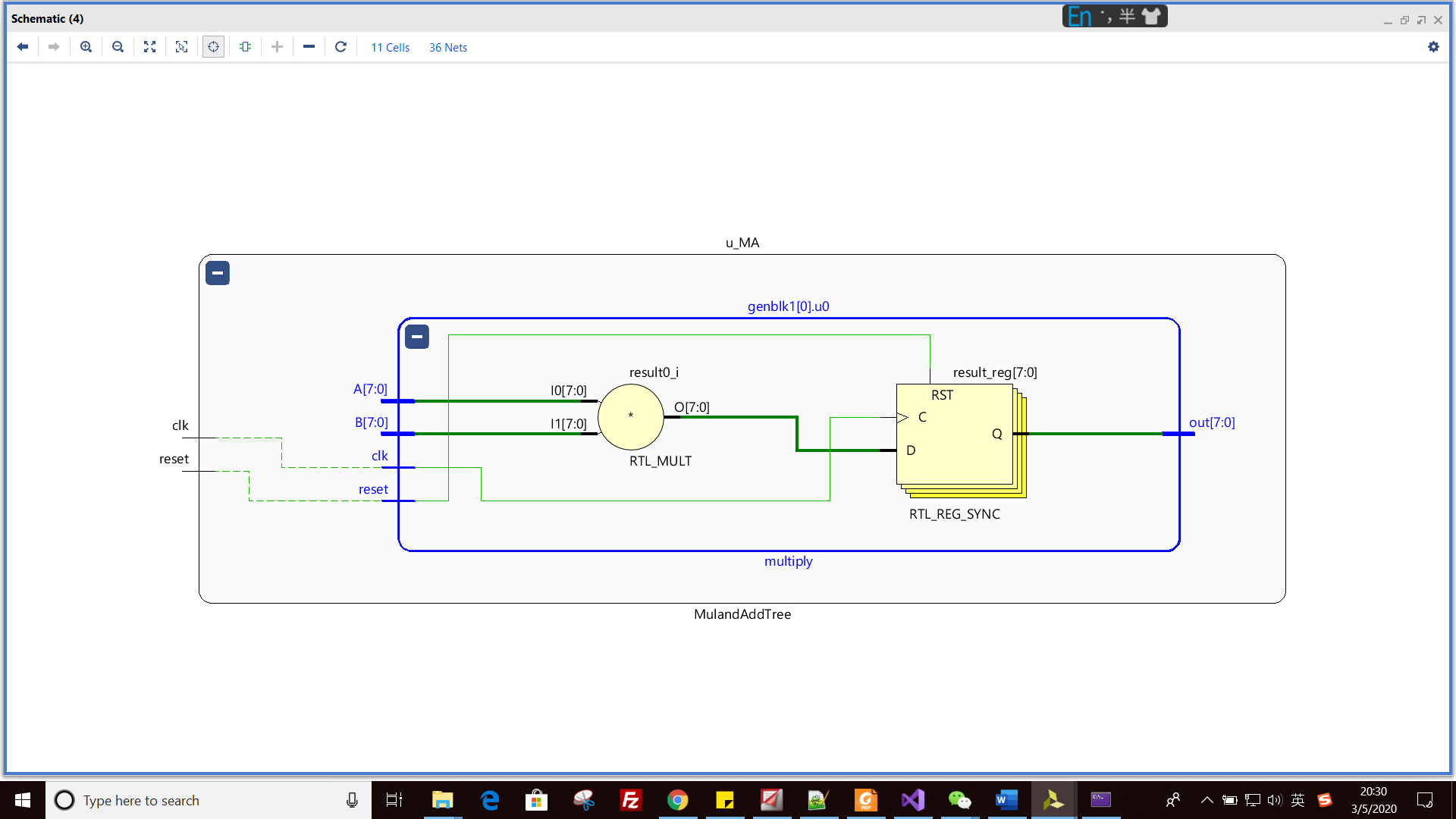
1. **Schematics:**



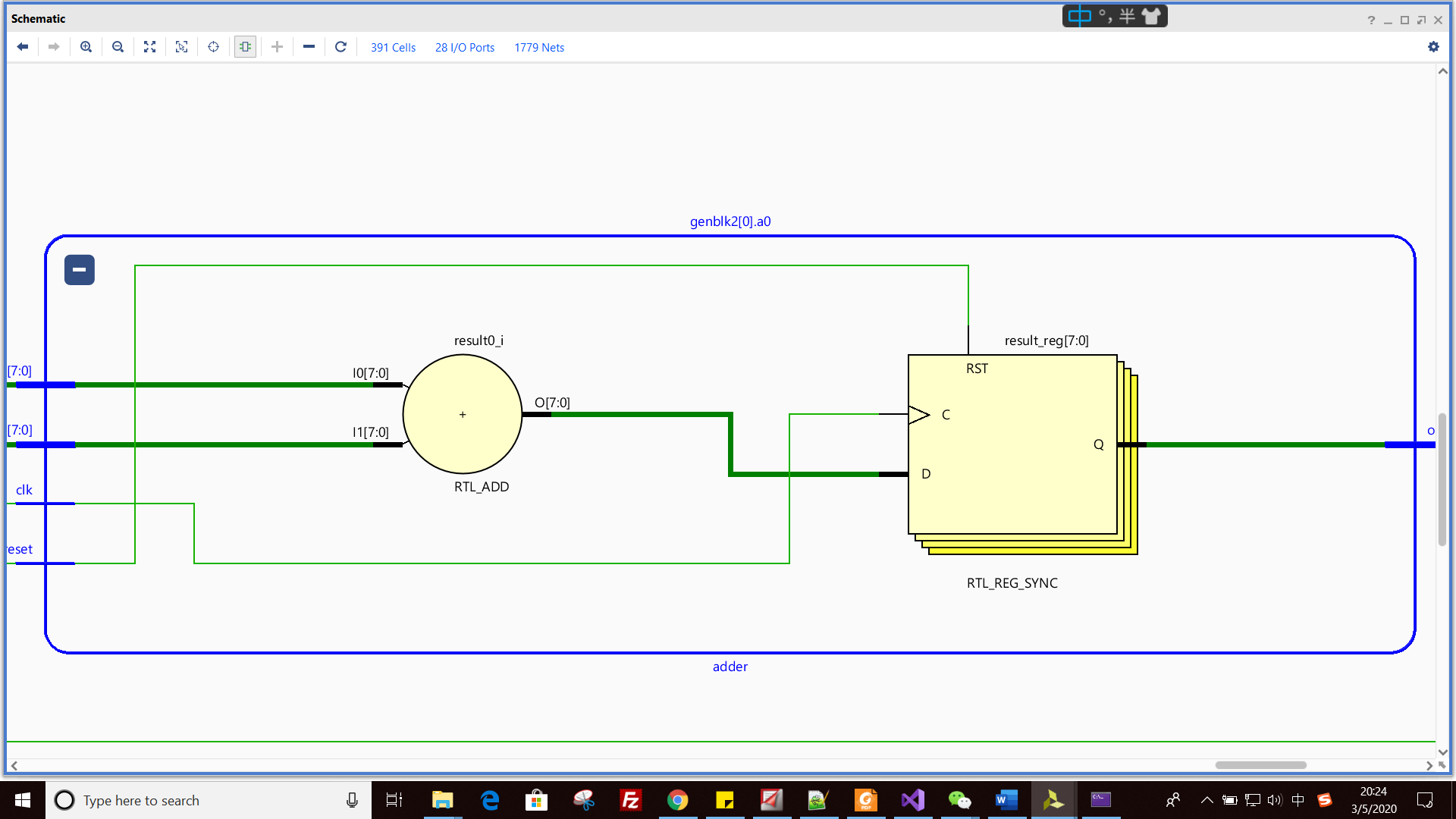
*Picture 19: the schematics of our design*



*Picture 20: MulandAddTree*



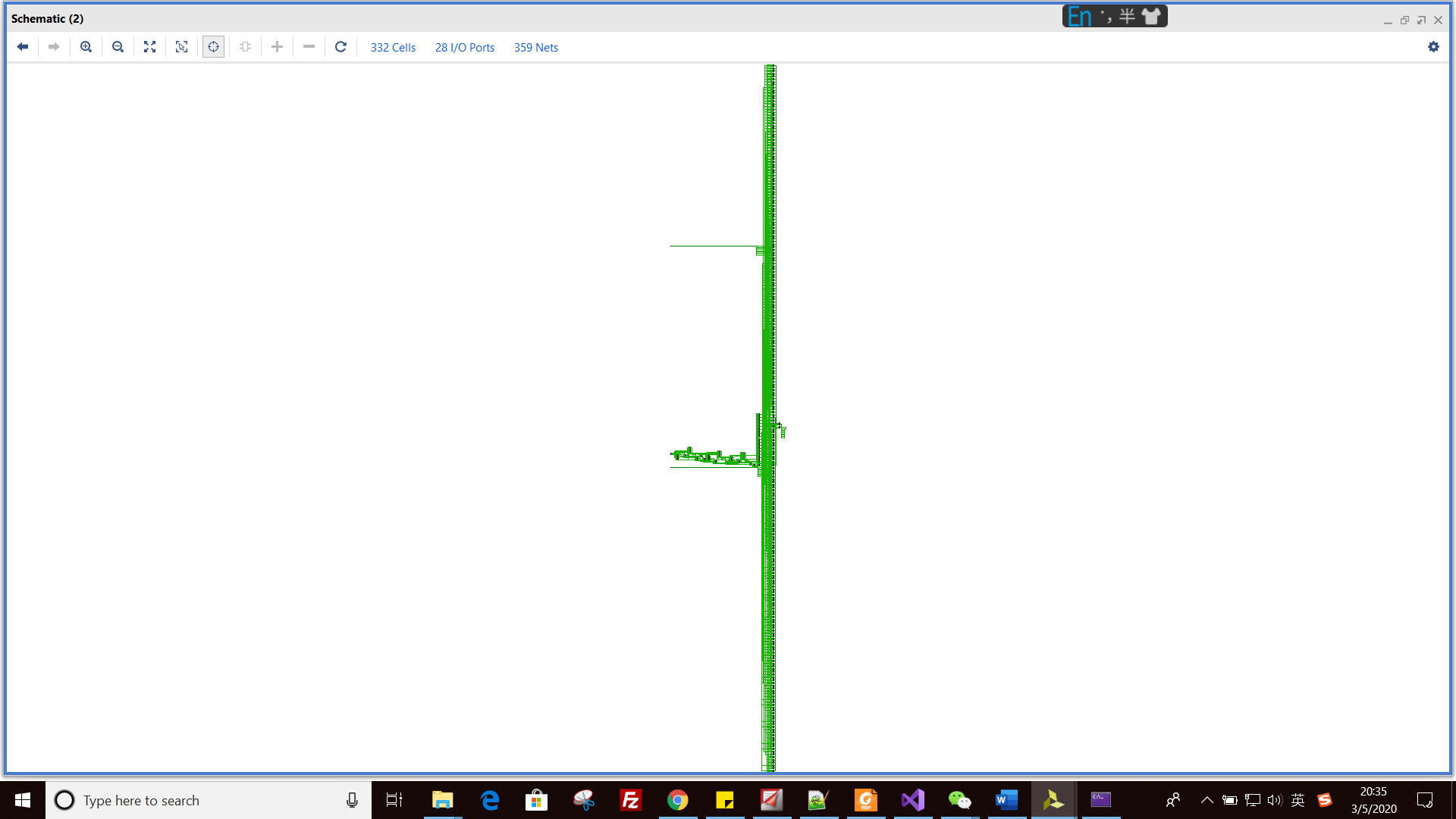
*Picture 21: multiply*



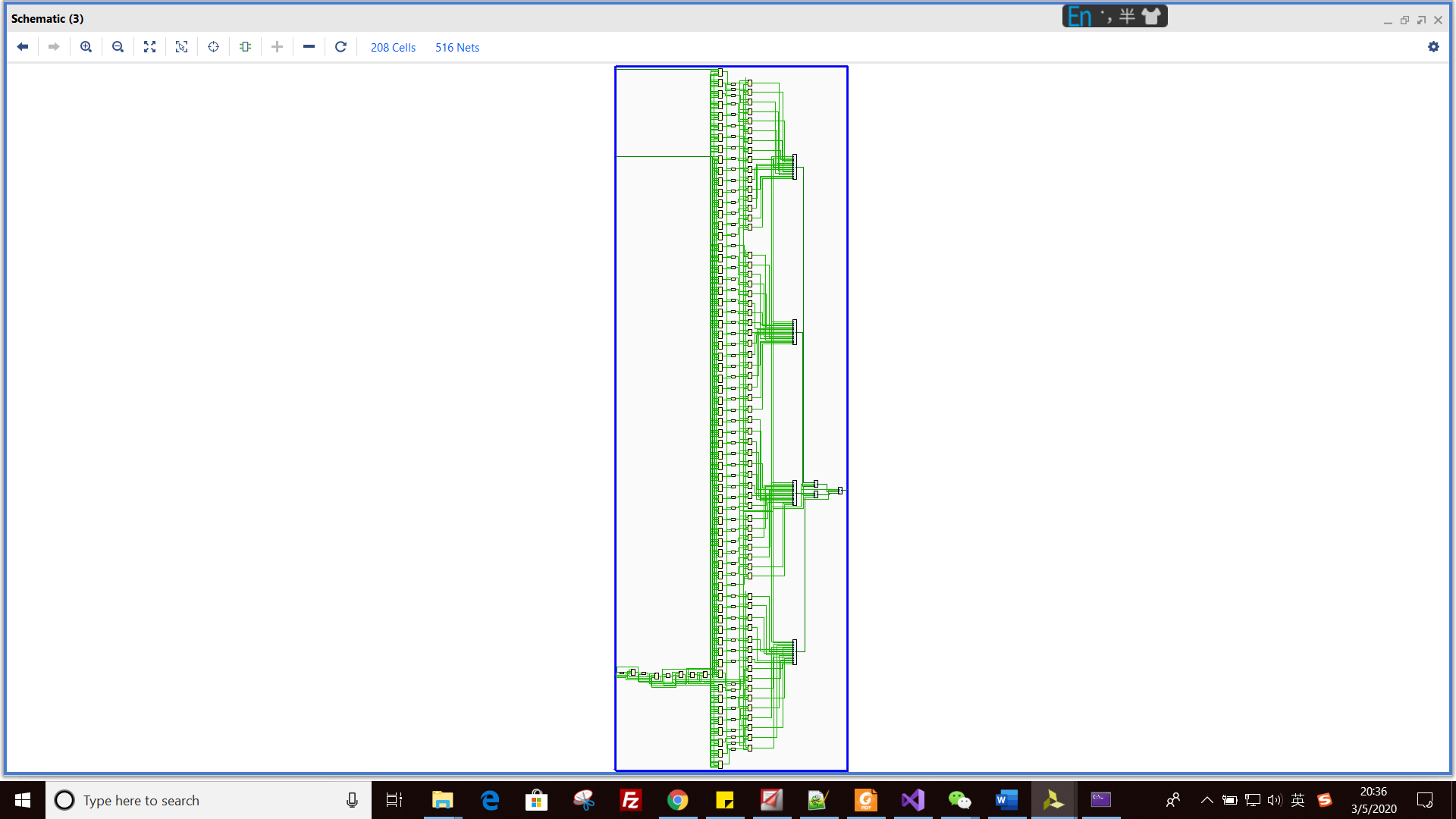
*Picture 22: adder*

**Part 1: 4\*4**

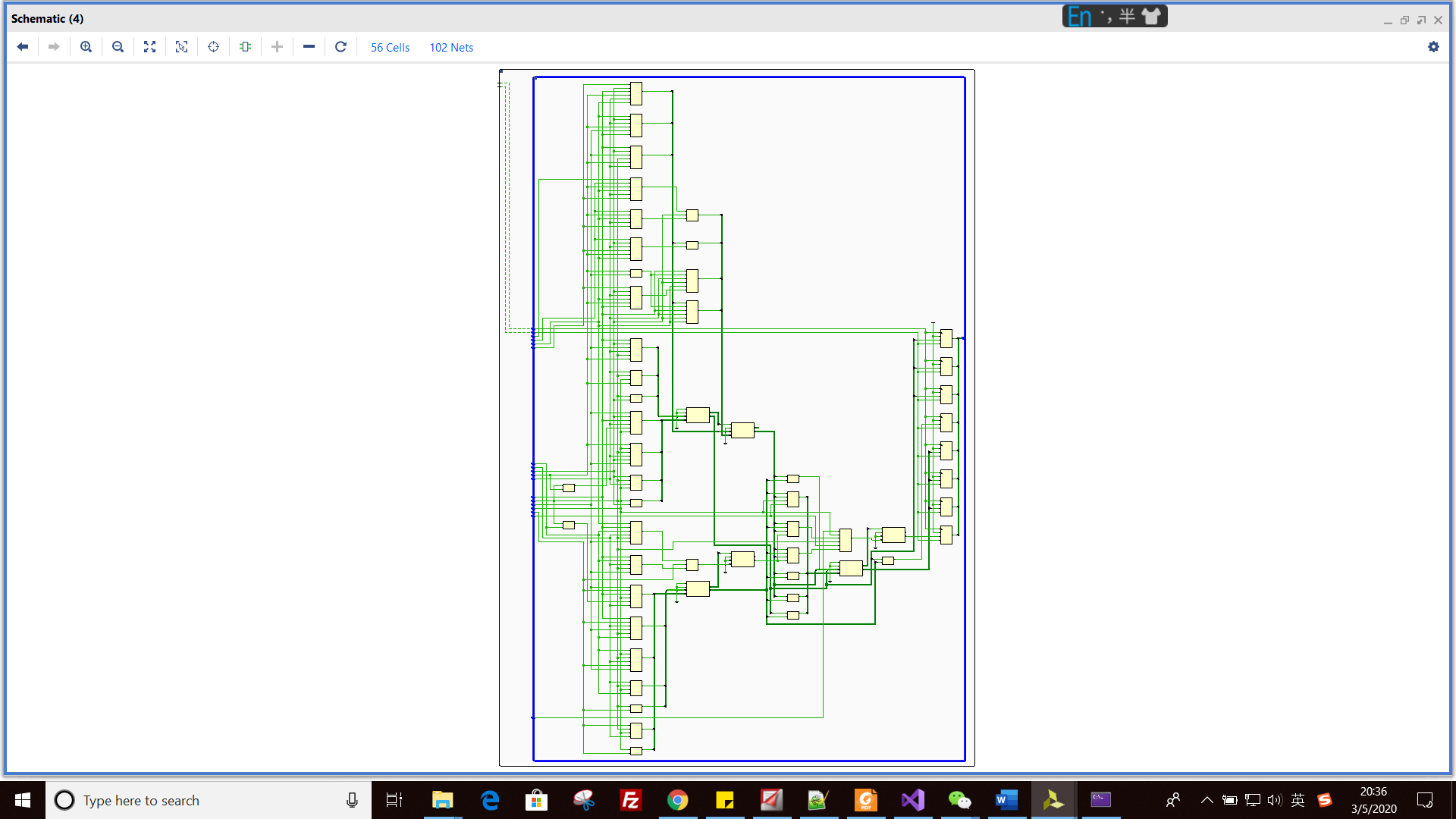
1. **Synthesis**



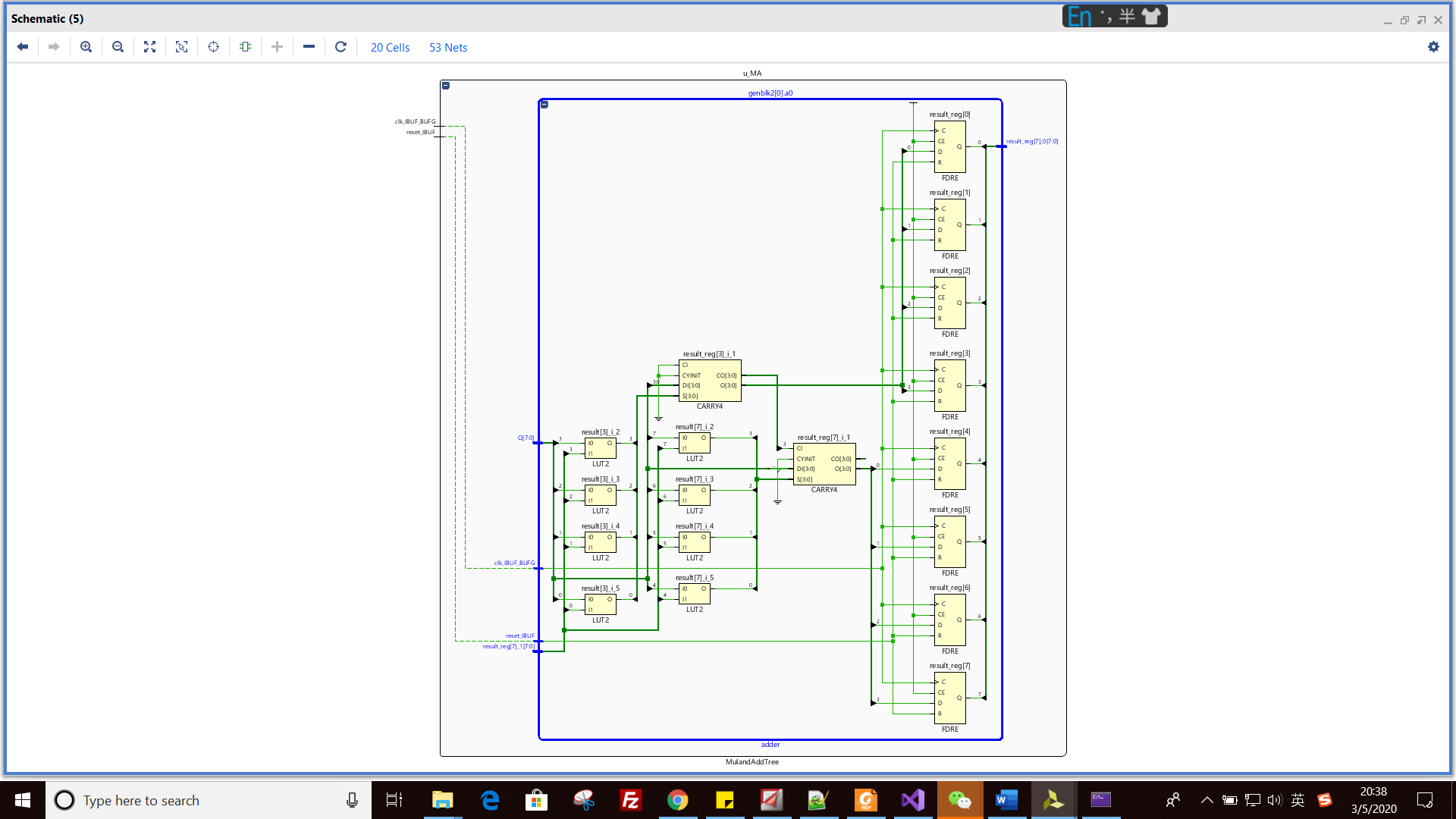
*Picture 23: whole schematics of design*



*Picture 24: MulandAddTree*

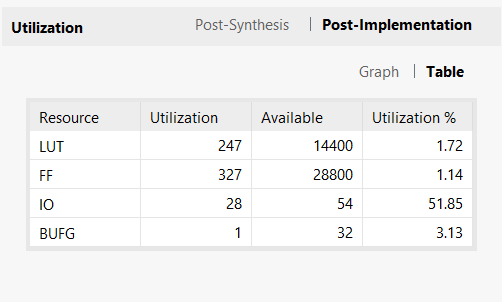


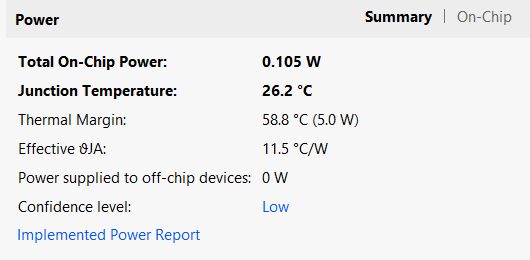
*Picture 25: multiply*

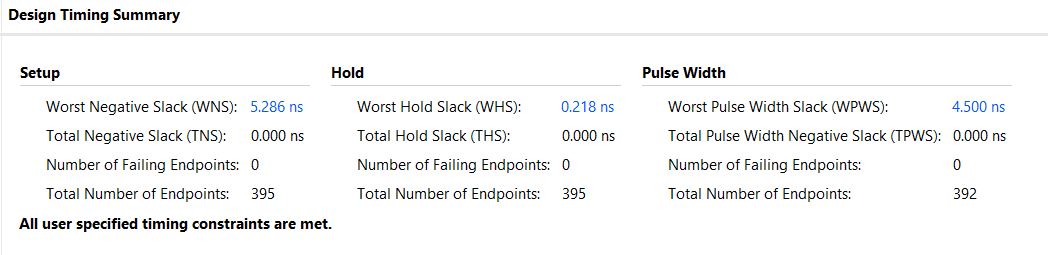


*Picture 26: adder*

1. **estimation reports**

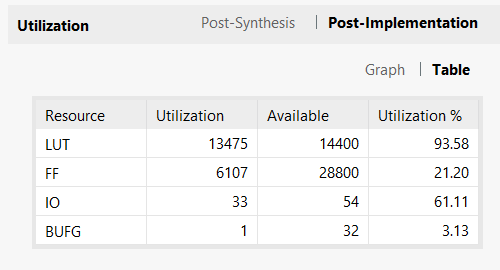






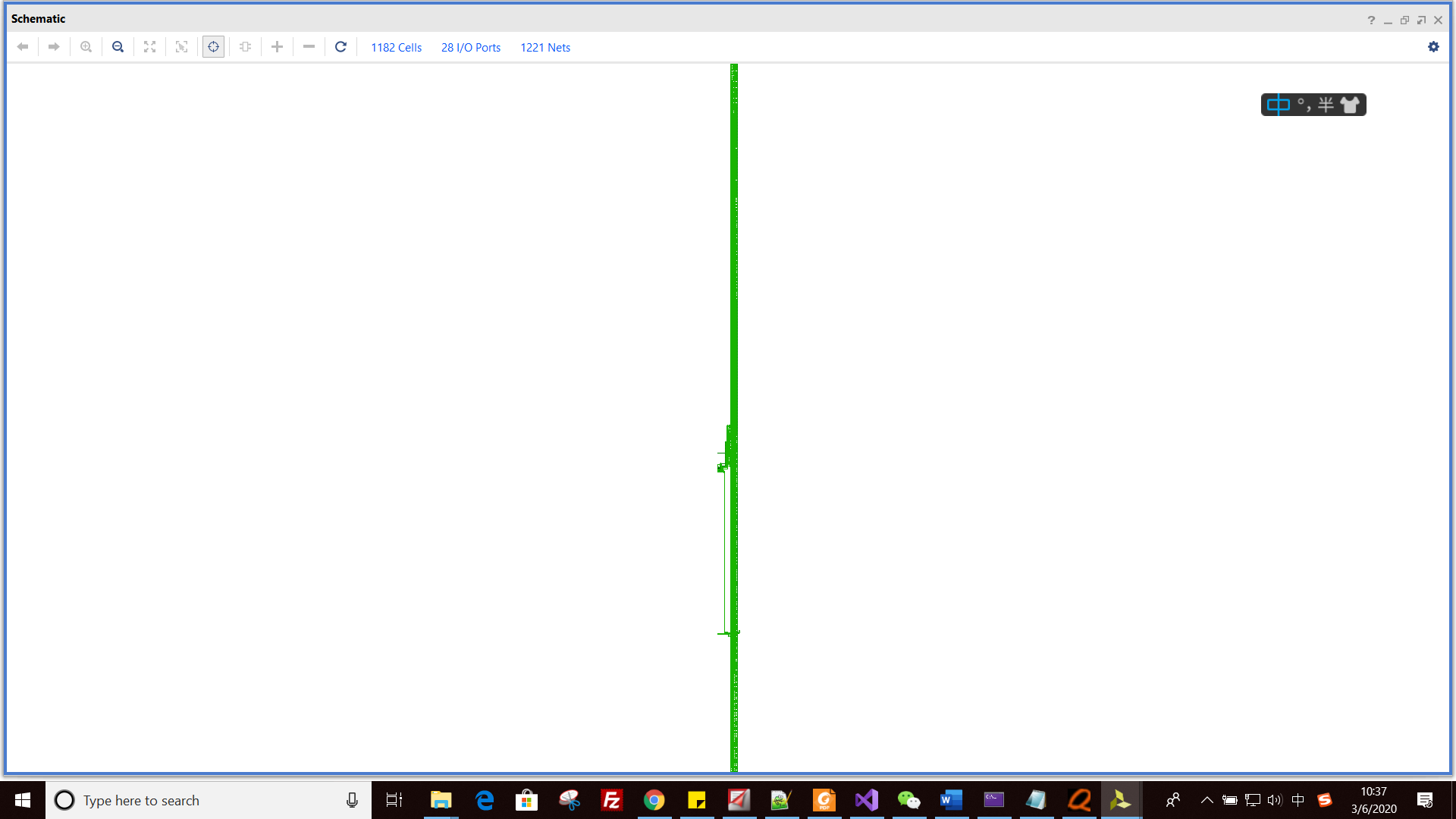
**5. the largest number of parallel MulandAddTrees.**

The largest number of parallel MulandAddTrees is 256, M\_multi\_2.v is used to do the test. The following table is our resource utilization.

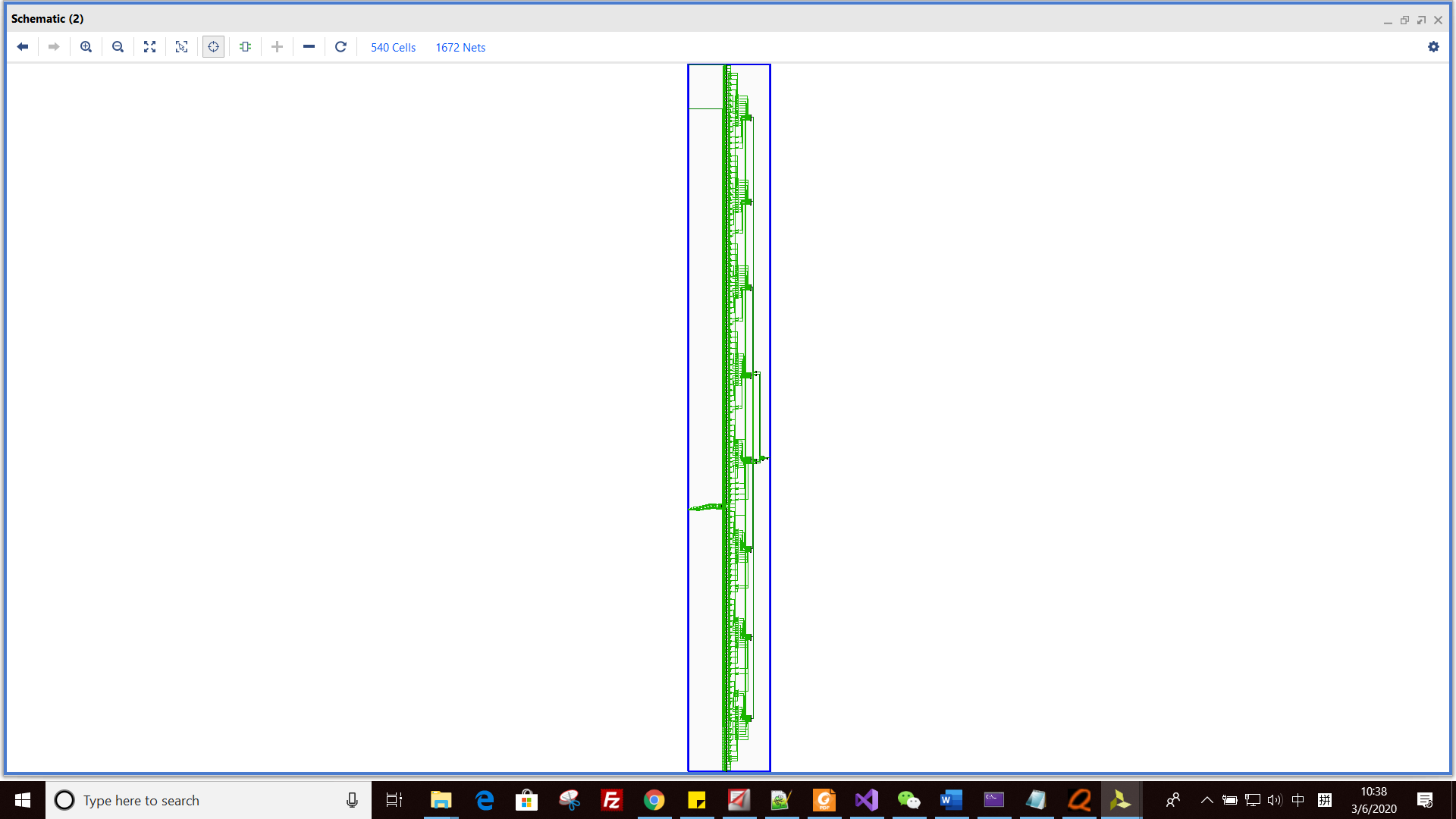


**Part 2: 8\*8**

1. **Synthesis**



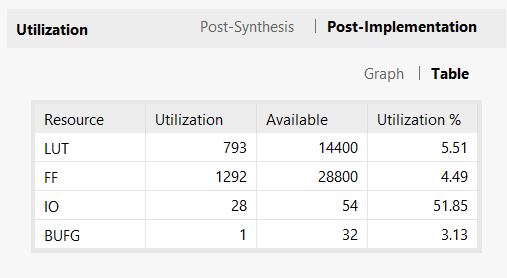
*Picture 27: whole schematics of design*

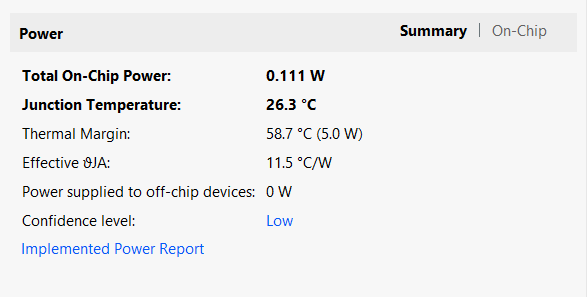


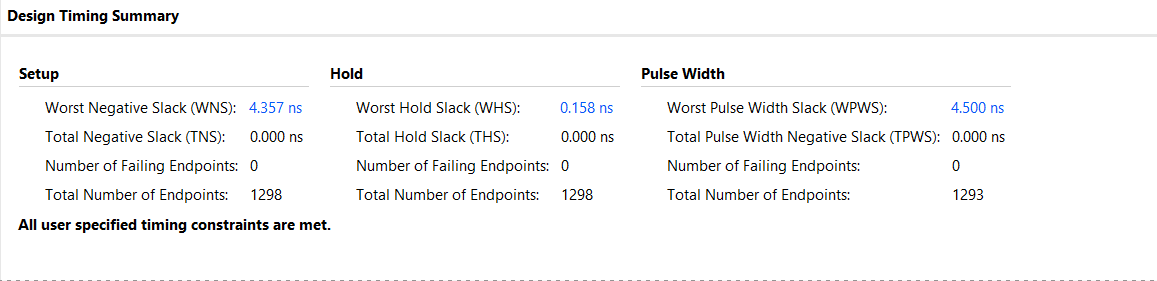
*Picture 28: MulandAddTree*

For multiply and adder module, their schematics same as 4\*4 ones.

1. **estimation reports**

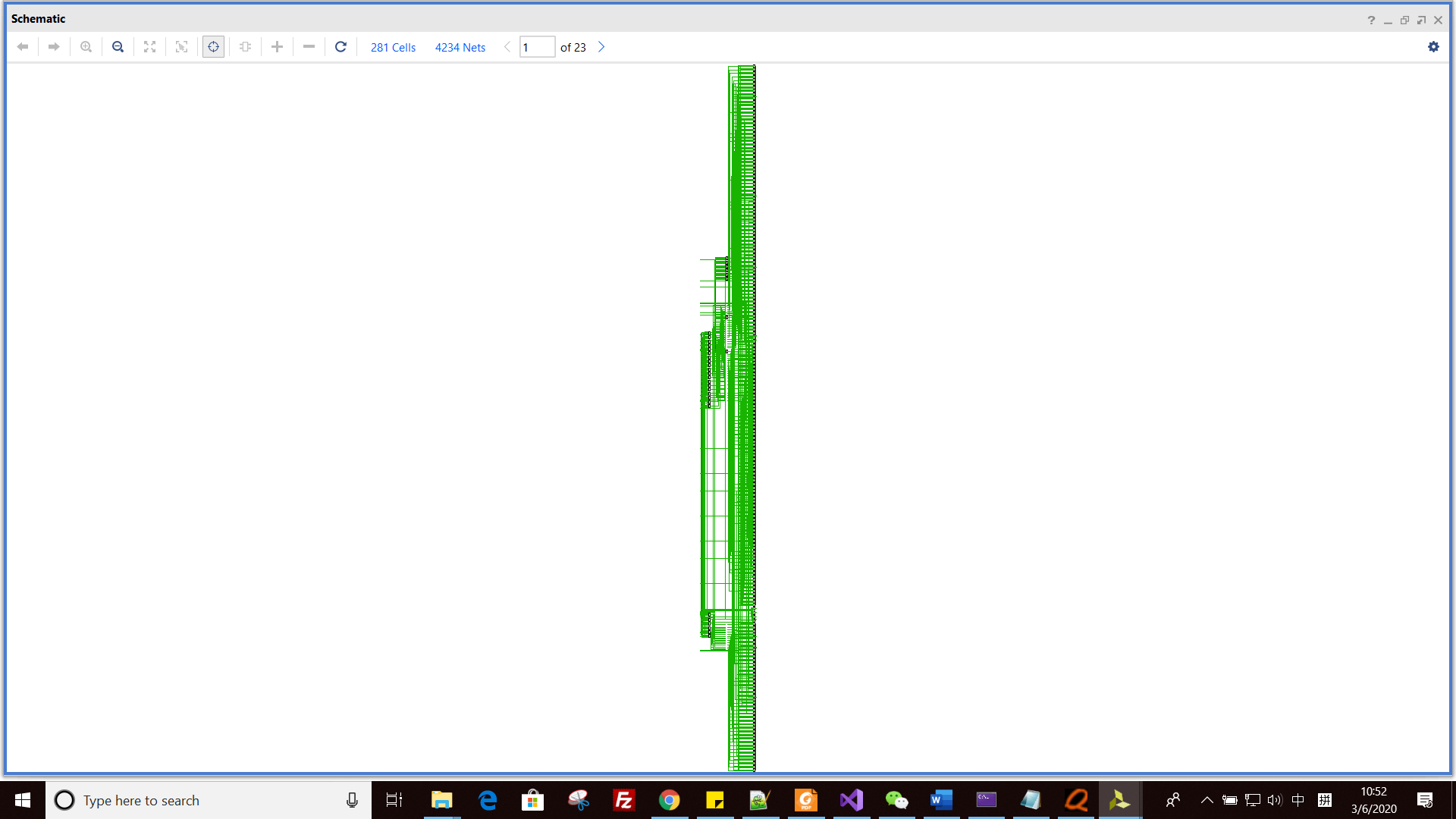




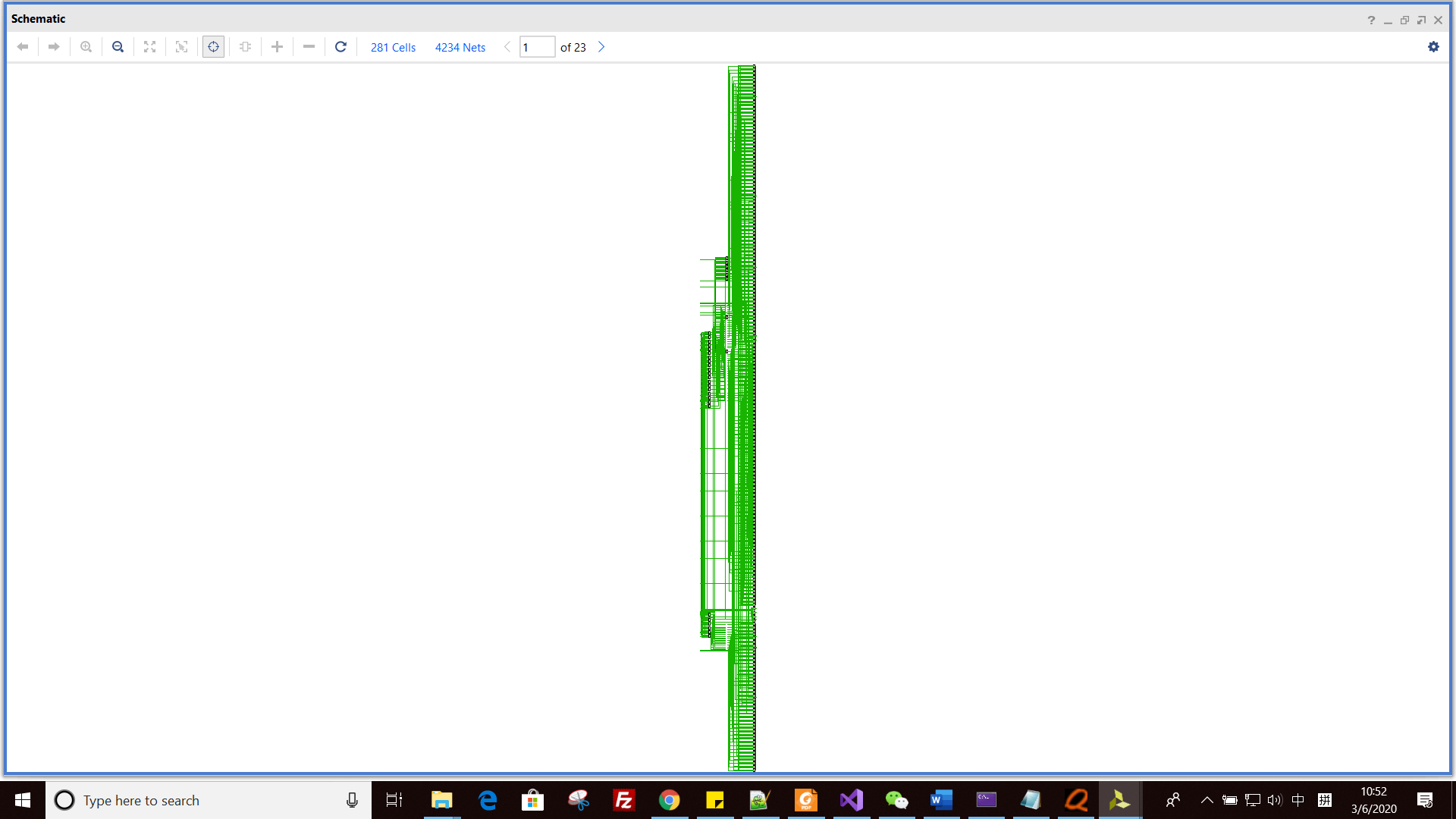


**Part 3: 16\*16**

1. **Synthesis**



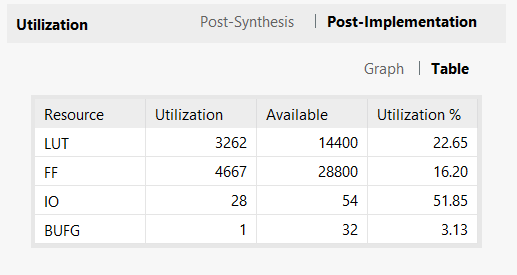
*Picture 29: whole schematics of design*

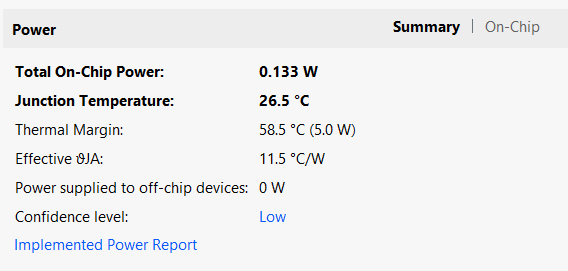


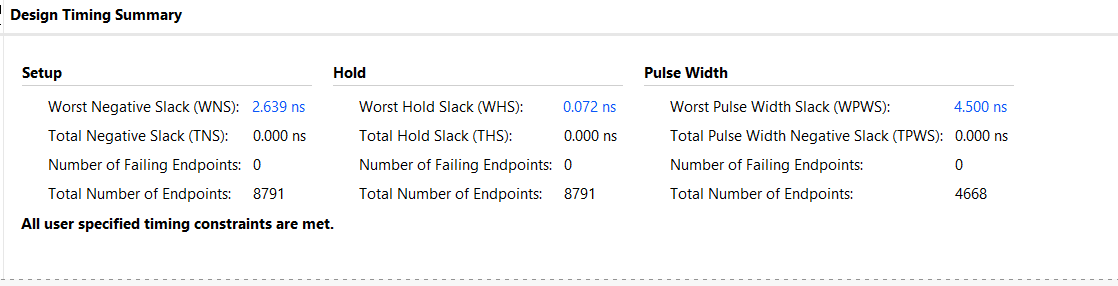
*Picture 30: MulandAddTree*

For multiply and adder module, their schematics same as 4\*4 ones.

1. **estimation reports**

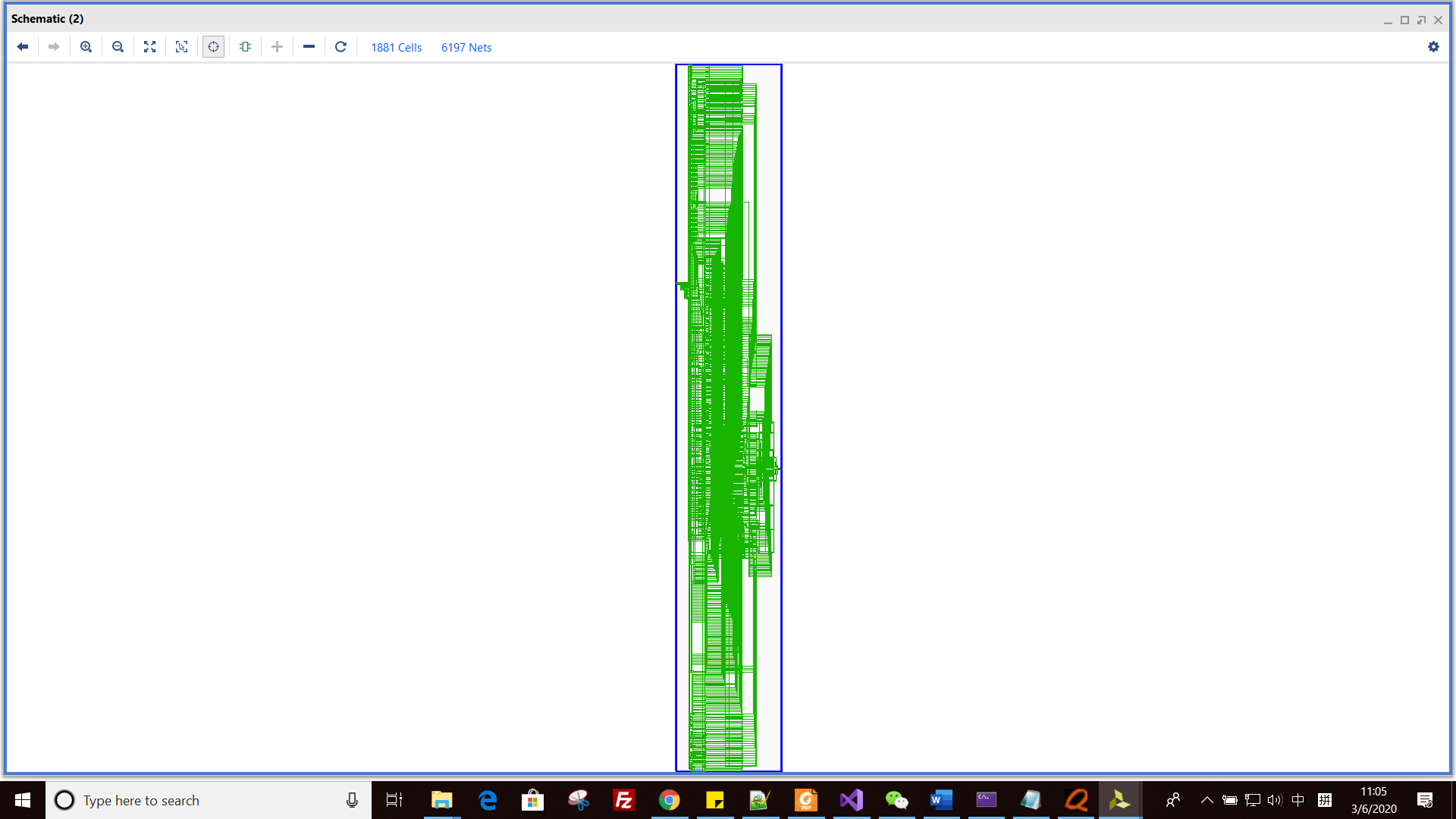




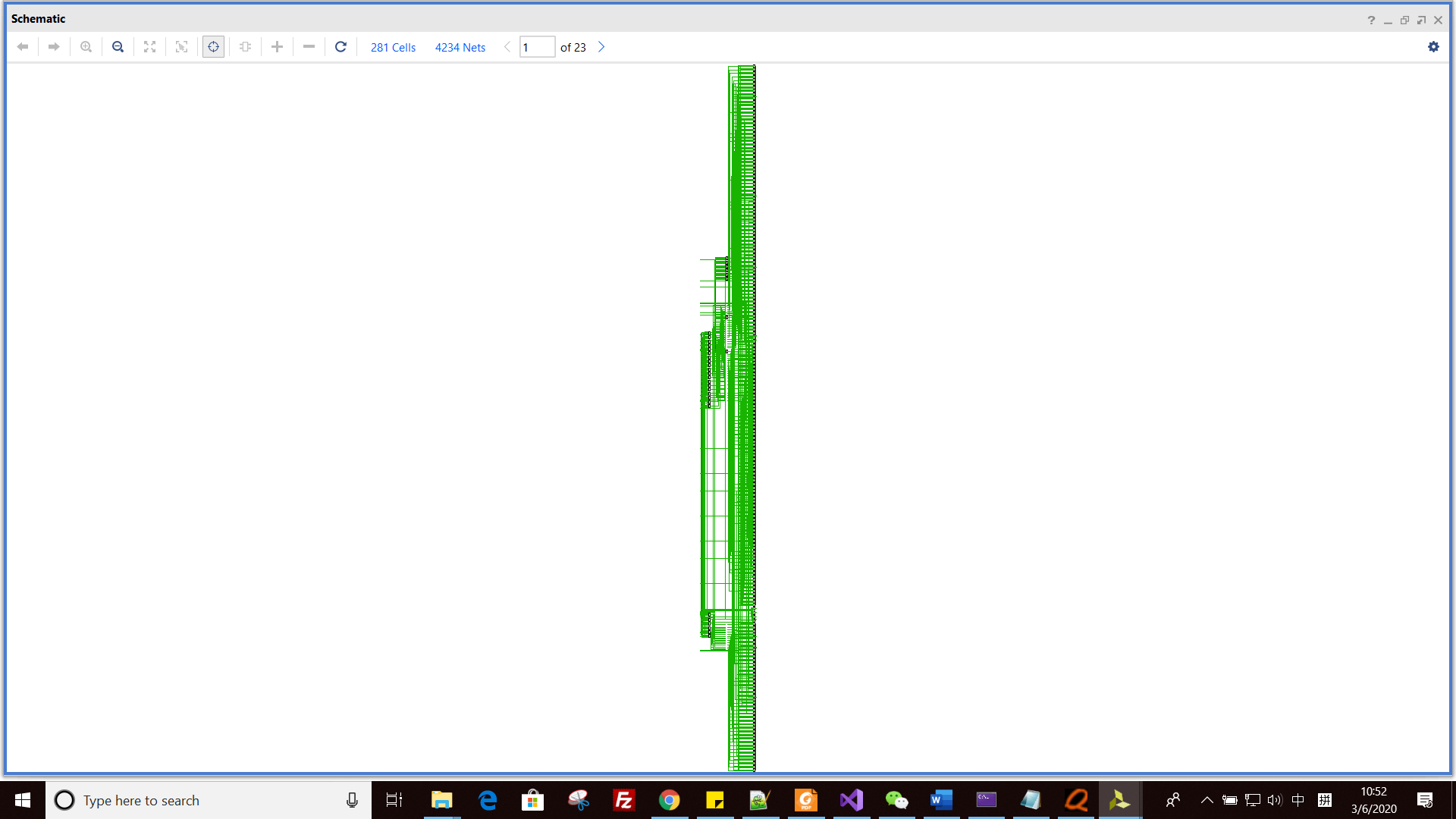


**Part 4: 32\*32**

1. **Synthesis**



*Picture 31: whole schematics of design*



*Picture 30: MulandAddTree*

For multiply and adder module, their schematics same as 4\*4 ones.

1. **estimation reports**

the 32 \* 32 matrixs cannot be implement on our FPGA, because the FPGA do not have enough resource.