Faculty of Engineering

Cairo University

CMPN111

Hardware Documentation Report

Final Assessment

Team: 16

Names:

Lina Ayman Hamza 1170113

Maria Ossama Fathy 4180292

Passant Ahmed Maher 1170030

Shorouk Mohamed Roshdy 1170110

Workload distribution:

i. Project workload distribution

Student	**Parts**
Lina Ayman Hamza	Convolutoin (part1) Integrating the FC Layer with softmax (Part 5)
Shorouk Mohamed Roshdy	Average pool (part4) Integrating average pool, tanh and Conv (Part 5)
Maria Ossama Fathy	Softmax (part3) Modifying the FC_Layer module (Part 5)
Passant Ahmed Maher	Tanh (part2) Modifying The PE module, Testbench of the whole CNN (Part 5)

ii. Report workload distribution

Student	**Parts**
Lina Ayman Hamza	Section II,III
Shorouk Mohamed Roshdy	Section V
Maria Ossama Fathy	Section VI
Passant Ahmed Maher	Section IV

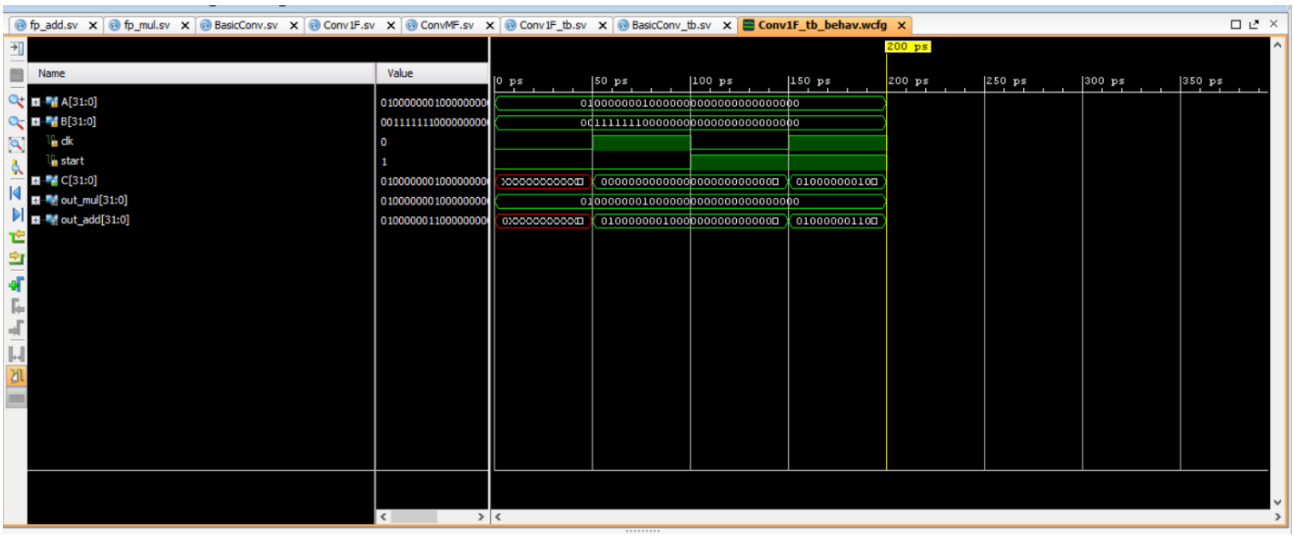
➔ Part 1: Convolution Part Algorithm Explanation:

This Part is the core of the CNN. It takes an input image and performs multiple convolution processes on it. According to the size of the input filter, it performs an element-wise multiplication on a part of the image and assigns the addition of all these element-wise multiplication results to a single pixel of the output image. This process is performed to each pixel of the input image and its surrounding pixels. This process is used in deep learning and recognition of images.

It consists of three layers:

1- **BasicConv.sv:**
   This part contains the fp_mul & fp_add instantiations. It's the layer where I send two-32bit elements **A** & **B**, they get multiplied in fp_mul and then sent to fp_add to accumulate the result on the last output **C.**

   Test bench simulation:



   Comment on output:
   There is an important observation that made a difference to handling the next layer.
   As you can see from the simulation there's always a lost cycle when I need to put 0 in **C (the output)**, and another lost cycle assigning the **out_add** to

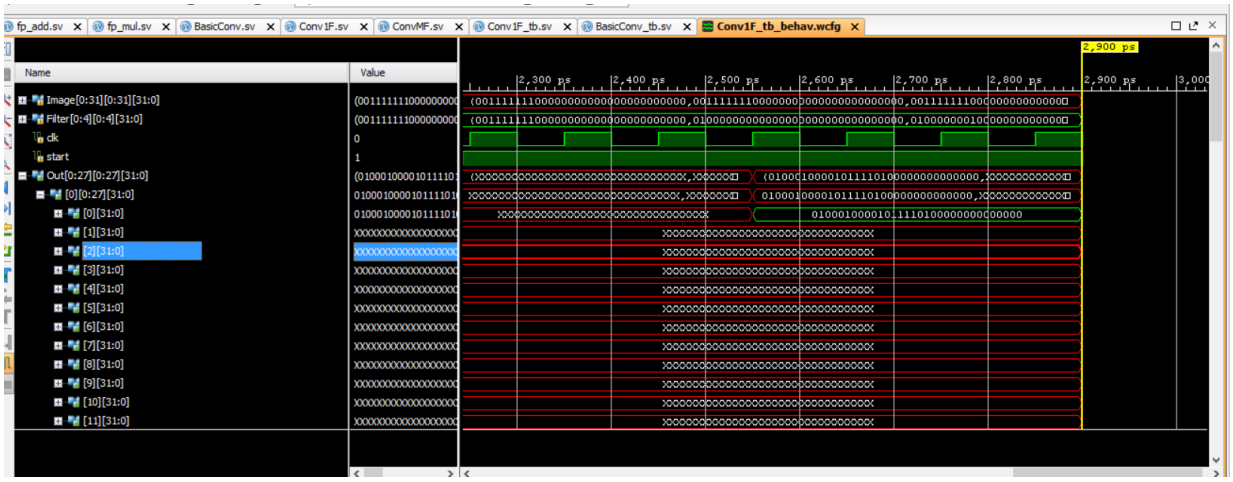
C . So that means that instead of being able to finish accumulating 25 elements multiplication and send the output in 25 cycles I now have the output ready after **27 cycles.**

2- Conv1F.sv:

This is the most important layer that does all the work. It takes as input The **Image** as a 2D array (3D if we count the DATAWIDTH) the dimensions of the image are parameters because the input of the convolution changes each time after the average pool. The second Input is the **Filter** which is always of 5x5 dimension.

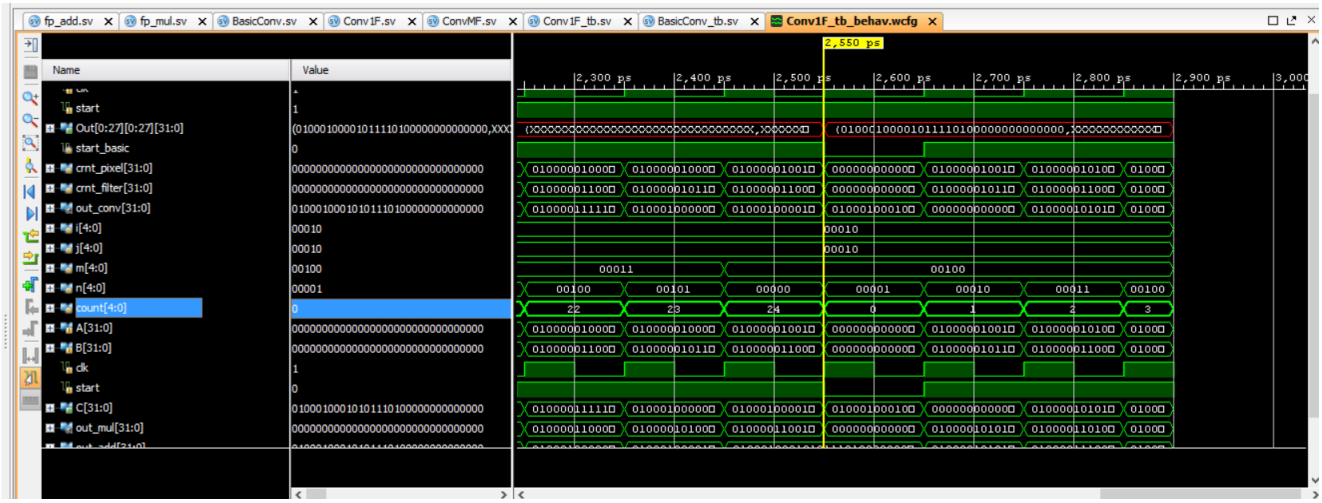
In this layer I use the **BasicConv** module to send **crnt_pixel** & **crnt_filter (32 bits)** to its inputs.
I start from **i=2** & **j=2** (Image row & column indices), I skip 2 rows & 2 columns to have an output of the
IDim-2 X IDim-2 , this is like the center that I want to move up, down, right and left from it to be able to send 25 elements as I'm also checking on **n** & **m** (rows & columns of the filter) sending the **crnt_pixel** & **crnt_filter**.
I also had to make a counter check to only assign the result to the output when my **count ==25** (meaning first 5x5 elements wise multiplication is done), I always assign to the **Out[i-2][j-2]** because the **Out** array is always smaller than the input by 2 rows and 2 columns , this is also when I had to add a separate start signal to send to the **BasicConv** to indicate that I have to reset the accumulation.

<u>NB:</u> I had to stall for one cycle each 25 count in order to write the correct result in the <u>Out</u> array because the output of the <u>BasicConv</u> is late by a clock cycle.

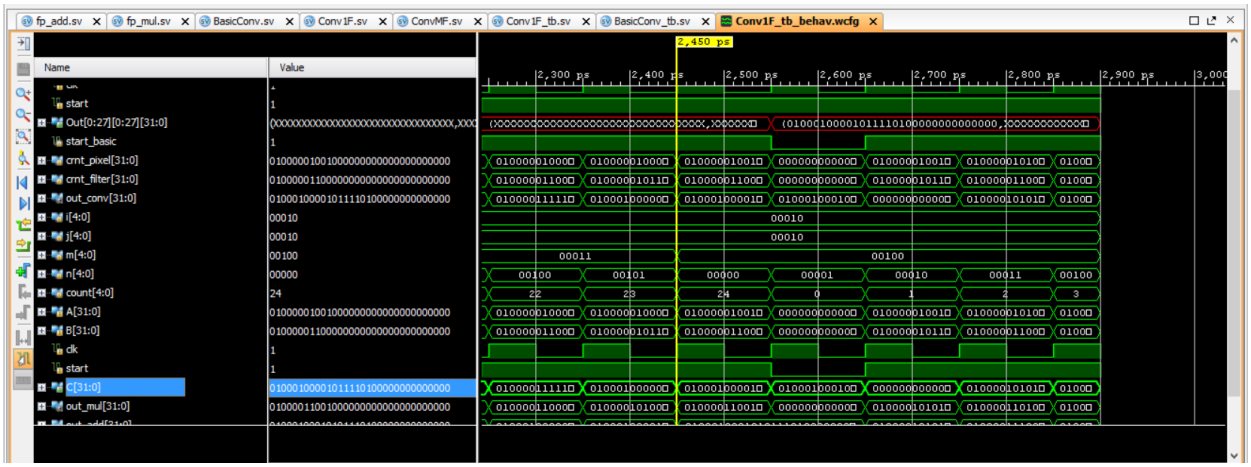


Test Bench simulation:

In this test bench I filled the first 5x5 in the image and I filled a 5x5 filter to test the algorithm of the code.



This is the first output put in the first pixel of the **Out** array right.



Here we can see that the output of **C** is written to **Out** array after stalling for one cycle and resetting **C** (the cycle where count=0)

<u>Comment on output:</u>

The number of cycle for each size of output image we are dealing with in this project

For 28x28: Ideally cycles=28x28x25= 19,600

Actual cycles=28x28x27= 21,168

For 10x10: Ideally cycles=10x10x25= 2,500

Actual cycles=10x10x27= 2,700

For 5x5: Ideally cycles=1x1x25= 25

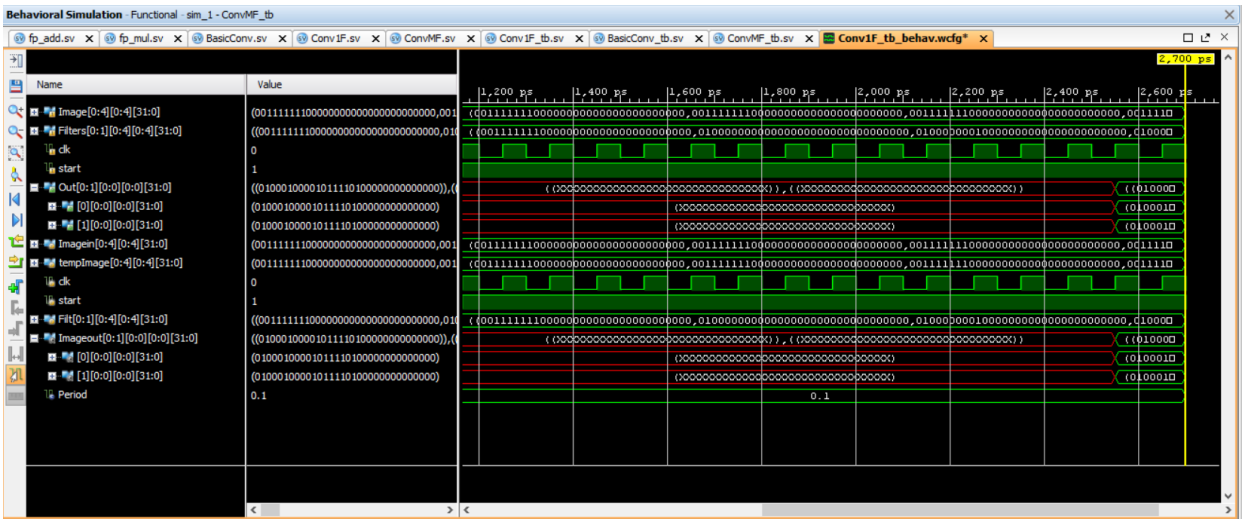
Actual cycles= 1x1x27=27

3- **ConvMF.sv:**

This is the last layer that takes multiple **Filters** and generates a number of parallel instantiations of the previous layer equal to the number of **Filters**.

Test bench simulation:

For simplicity in this test bench and since I'm sure now that the 2nd layer is working properly I gave this test bench an Image of size 5x5 and two filters so that the output is 1x1x2



The output is ready after 27 cycles as shown .

Comment on output:

The number of cycle will be the same as the previous layer because this layer achieves **parallelism** meaning that it assigns the two images (in this case) or any number of images in parallel, so the clock cycles for this layer will be as follow:

For 28x28xn: Ideally cycles=28x28x25= 19,600

Actual cycles=28x28x27= 21,168

For 10x10xn: Ideally cycles=10x10x25= 2,500

Actual cycles=10x10x27= 2,700

For 1x1xn: Ideally cycles=1x1x25= 25

Actual cycles= 1x1x27=27

Synthesis of the 3rd layer:

```
Report RTL Partitions:
+------+--------------+-------------+----------+
|      |RTL Partition |Replication  |Instances |
+------+--------------+-------------+----------+
|1     |Conv1F__GBM0  |            6|     32036|
|2     |Conv1F__GBM1  |            6|      4960|
|3     |Conv1F__GBM2  |            6|     10912|
|4     |Conv1F__GBM3  |            6|     20832|
|5     |Conv1F__GBM4  |            6|     14880|
|6     |Conv1F__GBM5  |            6|     17856|
|7     |Conv1F__GBM6  |            6|      9147|
|8     |Conv1F__GBM7  |            6|      4676|
|9     |Conv1F__GBM8  |            6|     11189|
|10    |Conv1F__GBM9  |            6|      5230|
+------+--------------+-------------+----------+
```

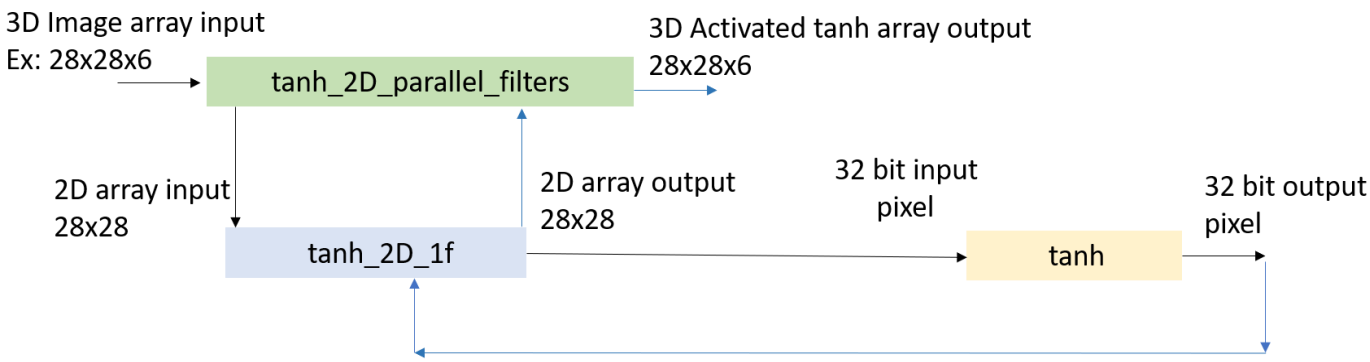
```
Report Cell Usage:
+------+--------+-------+
|      |Cell    |Count  |
+------+--------+-------+
|1     |BUFG    |      1|
|2     |CARRY4  |    294|
|3     |DSP48E1 |     12|
|4     |LUT1    |    325|
|5     |LUT2    |    762|
|6     |LUT3    |   1860|
|7     |LUT4    |   2070|
|8     |LUT5    |   4036|
|9     |LUT6    | 189336|
|10    |MUXF7   |  62214|
|11    |FDRE    | 151446|
|12    |FDSE    |     60|
|13    |LD      |     48|
|14    |IBUF    |  37570|
|15    |OBUF    | 150528|
+------+--------+-------+

Report Instance Areas:
+------+---------------------+-------------+-------+
|      |Instance             |Module       |Cells  |
+------+---------------------+-------------+-------+
|1     |top                  |             | 600562|
|2     |  \genblk1[0].conv   |Conv1F       |  68691|
|3     |    conv             |BasicConv_17 |    974|
|4     |      add            |fp_add_18    |    637|
|5     |      mul            |fpmul_19     |    252|
|6     |  \genblk1[1].conv   |Conv1F_0     |  68690|
|7     |    conv             |BasicConv_14 |    974|
|8     |      add            |fp_add_15    |    637|
|9     |      mul            |fpmul_16     |    252|
|10    |  \genblk1[2].conv   |Conv1F_1     |  68690|
|11    |    conv             |BasicConv_11 |    974|
|12    |      add            |fp_add_12    |    637|
|13    |      mul            |fpmul_13     |    252|
|14    |  \genblk1[3].conv   |Conv1F_2     |  68689|
|15    |    conv             |BasicConv_8  |    974|
|16    |      add            |fp_add_9     |    637|
|17    |      mul            |fpmul_10     |    252|
|18    |  \genblk1[4].conv   |Conv1F_3     |  68690|
|19    |    conv             |BasicConv_5  |    974|
|20    |      add            |fp_add_6     |    637|
|21    |      mul            |fpmul_7      |    252|
|22    |  \genblk1[5].conv   |Conv1F_4     |  68689|
|23    |    conv             |BasicConv    |    974|
|24    |      add            |fp_add       |    637|
|25    |      mul            |fpmul        |    252|
+------+---------------------+-------------+-------+

Finished Writing Synthesis Report : Time (s): cpu = 00:06:
```

Utilization of the 3rd layer:

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	197015	0	53200	370.33
LUT as Logic	197015	0	53200	370.33
LUT as Memory	0	0	17400	0.00
Slice Registers	151554	0	106400	142.44
Register as Flip Flop	151506	0	106400	142.39
Register as Latch	48	0	106400	0.05
F7 Muxes	62214	0	26600	233.89
F8 Muxes	0	0	13300	0.00

➔ Part 2: Tanh Activation Function

❖ Block diagram:



The hyperbolic tangent activation function for a 3D array input represented (image_dimension* image_dimension * no. filters) is applied on 3 modules explained in the following figure.

tanh	tanh_2D_1f	tanh_2D_parallel_filters
This module takes a 32-bit input pixel in float point IEEE format and applies the hyperbolic tangent activation function then produces 32-bit activated pixel.	This module takes a 2D array (image) as input and applies the tanh module to two pixels at a time. Parallelism: within activating the image.	This module takes a 3D array as input and applies the tanh module to two pixels in an all filters at the same time. (2D Parallel Tanh Layer). 2D Parallelism: All images are activated in parallel, also 2 pixels of each image are activated in parallel

1) Tanh module:

→The tanh module applies 4 terms of the taylor expansion series.

→ the coefficients $\frac{-1}{3}, \frac{2}{15}$ and $\frac{-17}{315}$ are converted to IEEE 754 using an online converter.

An example of how this module works: assume x is the input

➔ x is entered also to the first multiplier as flpA and flpB to get x^2

➔ then the output of this multiplier (x^2) is entered as the flpA to another multiplier and x is entered as flpB, the output of this multiplier is x^3

➔ then x^3 → flpA and $\frac{-1}{3}$ → flpB and then the output is $\frac{-1}{3}x^3$ is entered as flpA and x is entered as flpB in an adder.

A) Testbench simulation:

This testbench consists of 6 test cases to verify the tanh module's output.

Each case takes one cycle as the input is on the rising edge and the output on the falling edge.

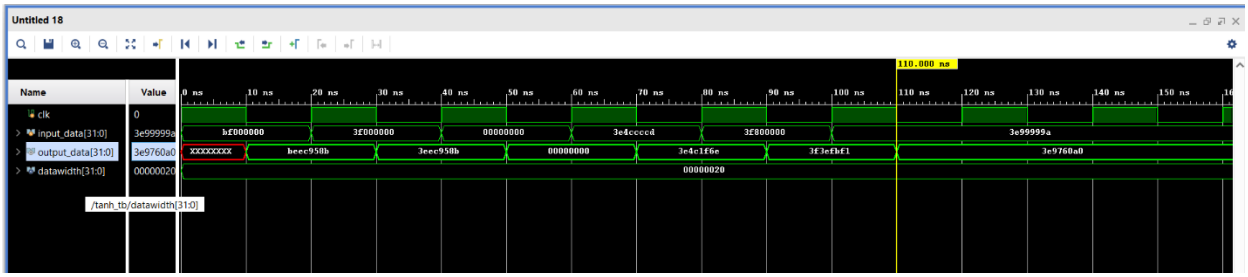
Therefore the total number of cycles= 6.

The input values are:

➔ - 0.5, 0.5, 0, 0.2, 1, 0.3
➔ BF000000, 3F000000, 00000000,3E4CCCCD,3F800000,3E99999A,

The expected output should be → -0.462, 0.462, 0 , 0.197, 0.746, 0.29

➔ beec958b, 3eec958b, 00000000, 3e4c1f6e, 3f3efbf1, 3e9760a0



B) Synthesis and Utilization:

1. Utilization : ..\Downloads\tanh_utilization.txt

```
1. CLB Logic
------------

+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| CLB LUTs*               | 2669 |     0 |    242400 |  1.10 |
|   LUT as Logic          | 2669 |     0 |    242400 |  1.10 |
|   LUT as Memory         |    0 |     0 |    112800 |  0.00 |
| CLB Registers           |   56 |     0 |    484800 |  0.01 |
|   Register as Flip Flop |   32 |     0 |    484800 | <0.01 |
|   Register as Latch     |   24 |     0 |    484800 | <0.01 |
| CARRY8                  |   66 |     0 |     30300 |  0.22 |
| F7 Muxes                |    2 |     0 |    121200 | <0.01 |
| F8 Muxes                |    0 |     0 |     60600 |  0.00 |
| F9 Muxes                |    0 |     0 |     30300 |  0.00 |
+-------------------------+------+-------+-----------+-------+
```

```
|6     |  \genblk1[0].multiplier10  |fpmull_2  |    43|
|7     |  \genblk1[0].multiplier11  |fpmull_3  |   103|
|8     |  \genblk1[0].multiplier12  |fpmull_4  |    44|
|9     |  \genblk1[0].multiplier13  |fpmull_5  |    56|
|10    |  \genblk1[0].multiplier14  |fpmull_6  |    74|
|11    |  \genblk1[0].multiplier15  |fpmull_7  |    82|
|12    |  \genblk1[0].multiplier2   |fpmull_8  |    44|
|13    |  \genblk1[0].multiplier3   |fpmull_9  |    91|
|14    |  \genblk1[0].multiplier5   |fpmull_10 |    36|
|15    |  \genblk1[0].multiplier6   |fpmull_11 |    83|
|16    |  \genblk1[0].multiplier7   |fpmull_12 |    52|
|17    |  \genblk1[0].multiplier8   |fpmull_13 |    61|
+------+----------------------------+----------+------+

---------------------------------------------------------------------------
Finished Writing Synthesis Report : Time (s): cpu = 00:00:27 ; elapsed = 00:00:28 . Memory (MB): peak = 855.645 ; gain = 643.117
---------------------------------------------------------------------------
```

```
+-+--------------+------------+----------+
No constraint files found.
---------------------------------------------------------------------------
Start RTL Component Statistics
---------------------------------------------------------------------------
Detailed RTL Component Info :
+---Adders :
      2 Input      25 Bit       Adders := 3
      3 Input      25 Bit       Adders := 6
      3 Input       8 Bit       Adders := 25
      4 Input       8 Bit       Adders := 3
      2 Input       8 Bit       Adders := 3
+---XORs :
      2 Input       1 Bit         XORs := 13
+---Muxes :
      2 Input      32 Bit        Muxes := 35
      2 Input      24 Bit        Muxes := 15
      2 Input      23 Bit        Muxes := 16
      2 Input       8 Bit        Muxes := 159
     23 Input       8 Bit        Muxes := 6
      2 Input       1 Bit        Muxes := 147
---------------------------------------------------------------------------
Finished RTL Component Statistics
---------------------------------------------------------------------------
---------------------------------------------------------------------------
Start RTL Hierarchical Component Statistics
---------------------------------------------------------------------------
Hierarchical RTL Component report
Module fpmull
Detailed RTL Component Info :
+---Adders :
      3 Input       8 Bit       Adders := 1
+---XORs :
      2 Input       1 Bit         XORs := 1
+---Muxes :
      2 Input      32 Bit        Muxes := 2
      2 Input      23 Bit        Muxes := 1
Module fp_add
Detailed RTL Component Info :
+---Adders :
      2 Input      25 Bit       Adders := 1
      3 Input      25 Bit       Adders := 2
      3 Input       8 Bit       Adders := 4
      4 Input       8 Bit       Adders := 1
      2 Input       8 Bit       Adders := 1
+---Muxes :
      2 Input      32 Bit        Muxes := 3
      2 Input      24 Bit        Muxes := 5
      2 Input      23 Bit        Muxes := 1
      2 Input       8 Bit        Muxes := 53
     23 Input       8 Bit        Muxes := 2
      2 Input       1 Bit        Muxes := 49
---------------------------------------------------------------------------
Finished RTL Hierarchical Component Statistics
```

2. Synthesis: ..\Downloads\tanh_synthesis.txt

```
| |RTL Partition |Replication |Instances |
+-+--------------+------------+----------+
+-+--------------+------------+----------+
---------------------------------------------------------------------------
Start ROM, RAM, DSP and Shift Register Reporting
---------------------------------------------------------------------------

DSP:
+------------+---------------+---------------+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+
|Module Name | DSP Mapping   | Neg Edge Clk  | A Size  | B Size  | C Size  | D Size  | P Size  | AREG    | BREG    | CREG    | DREG    | ADREG   | MREG    | PREG    |
+------------+---------------+---------------+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+
|fpmull      | A*B           | No            | 25      | 18      | 48      | 25      | 48      | 0       | 0       | 1       | 1       | 1       | 0       | 0       |
|fpmull      | (PCIN>>17)+A*B | No            | 25      | 0       | 31      | 25      | 31      | 0       | 0       | 1       | 1       | 1       | 0       | 0       |
|fpmull      | A*B           | No            | 25      | 18      | 48      | 25      | 48      | 0       | 0       | 1       | 1       | 1       | 0       | 0       |
|fpmull      | (PCIN>>17)+A*B | No            | 25      | 8       | 31      | 25      | 31      | 0       | 0       | 1       | 1       | 1       | 0       | 0       |
|fpmull      | A*B           | No            | 25      | 18      | 48      | 25      | 48      | 0       | 0       | 1       | 1       | 1       | 0       | 0       |
|fpmull      | (PCIN>>17)+A*B | No            | 25      | 0       | 31      | 25      | 31      | 0       | 0       | 1       | 1       | 1       | 0       | 0       |
|fpmull      | A*B           | No            | 25      | 18      | 48      | 25      | 48      | 0       | 0       | 1       | 1       | 1       | 0       | 0       |
|fpmull      | (PCIN>>17)+A*B | No            | 25      | 8       | 31      | 25      | 31      | 0       | 0       | 1       | 1       | 1       | 0       | 0       |
|fpmull      | A*B           | No            | 25      | 18      | 48      | 25      | 48      | 0       | 0       | 1       | 1       | 1       | 0       | 0       |
|fpmull      | (PCIN>>17)+A*B | No            | 25      | 0       | 31      | 25      | 31      | 0       | 0       | 1       | 1       | 1       | 0       | 0       |
|fpmull      | A*B           | No            | 25      | 18      | 48      | 25      | 48      | 0       | 0       | 1       | 1       | 1       | 0       | 0       |
|fpmull      | (PCIN>>17)+A*B | No            | 25      | 8       | 31      | 25      | 31      | 0       | 0       | 1       | 1       | 1       | 0       | 0       |
|fpmull      | A*B           | No            | 25      | 18      | 48      | 25      | 48      | 0       | 0       | 1       | 1       | 1       | 0       | 0       |
|fpmull      | (PCIN>>17)+A*B | No            | 25      | 0       | 31      | 25      | 31      | 0       | 0       | 1       | 1       | 1       | 0       | 0       |
|fpmull      | A*B           | No            | 25      | 18      | 48      | 25      | 48      | 0       | 0       | 1       | 1       | 1       | 0       | 0       |
|fpmull      | (PCIN>>17)+A*B | No            | 25      | 8       | 31      | 25      | 31      | 0       | 0       | 1       | 1       | 1       | 0       | 0       |
|fpmull      | A*B           | No            | 25      | 18      | 48      | 25      | 48      | 0       | 0       | 1       | 1       | 1       | 0       | 0       |
|fpmull      | (PCIN>>17)+A*B | No            | 25      | 0       | 31      | 25      | 31      | 0       | 0       | 1       | 1       | 1       | 0       | 0       |
|fpmull      | A*B           | No            | 25      | 18      | 48      | 25      | 48      | 0       | 0       | 1       | 1       | 1       | 0       | 0       |
|fpmull      | (PCIN>>17)+A*B | No            | 25      | 8       | 31      | 25      | 31      | 0       | 0       | 1       | 1       | 1       | 0       | 0       |
|fpmull      | A*B           | No            | 25      | 18      | 48      | 25      | 48      | 0       | 0       | 1       | 1       | 1       | 0       | 0       |
|fpmull      | (PCIN>>17)+A*B | No            | 25      | 0       | 31      | 25      | 31      | 0       | 0       | 1       | 1       | 1       | 0       | 0       |
+------------+---------------+---------------+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+

Note: The table shows the DSPs inferred at the current stage of the synthesis flow. Some DSP may be reimplemented as non DSP primitives later in the synthesis flow. Multiple instan
---------------------------------------------------------------------------
Finished ROM, RAM, DSP and Shift Register Reporting
---------------------------------------------------------------------------
No constraint files found.
---------------------------------------------------------------------------
Start Area Optimization
---------------------------------------------------------------------------
---------------------------------------------------------------------------
```

2- Tanh_2D_1f module

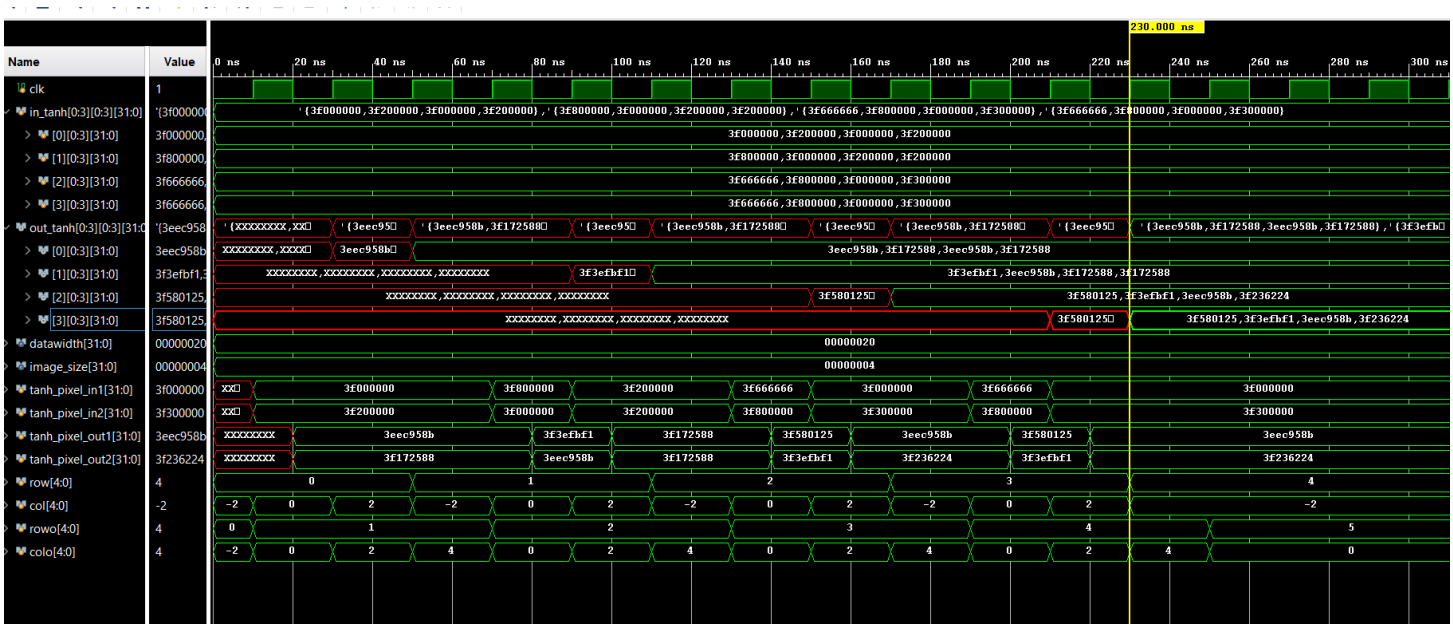
This module creates two instances of the tanh module to activate two pixels in parallel.

A) testbench simulation:

This testbench consists of a 4x4 array (image) that is activated 2 pixels per cycle.

Input → in_tanh , Output → out_tanh

Total number of cycles= 11.



Inputs:

$3f000000$	$3f200000$	$3f000000$	$3f200000$
$3f800000$	$3f000000$	$3f200000$	$3f200000$
$3f666666$	$3f800000$	$3f000000$	$3f300000$
$3f666666$	$3f800000$	$3f000000$	$3f300000$

Expected output:

$3eec958b$	$3f172588$	$3eec958b$	$3f172588$
$3f3efbf1$	$3eec958b$	$3f172588$	$3f172588$
$3f580125$	$3f3efbf1$	$3eec958b$	$3f236224$
$3f580125$	$3f3efbf1$	$3eec958b$	$3f172588$

B) Synthesis and Utilization:

1. Utilization:

1. CLB Logic

Site Type	Used	Fixed	Available	Util%
CLB LUTs*	27089	0	242400	11.18
LUT as Logic	27089	0	242400	11.18
LUT as Memory	0	0	112800	0.00
CLB Registers	25432	0	484800	5.25
Register as Flip Flop	25384	0	484800	5.24
Register as Latch	48	0	484800	<0.01
CARRY8	126	0	30300	0.42
F7 Muxes	2244	0	121200	1.85
F8 Muxes	896	0	60600	1.48
F9 Muxes	0	0	30300	0.00

2-Synthesis:

```
Module tanh_2D_1f
Detailed RTL Component Info :
+---Adders :
       2 Input     32 Bit       Adders := 1
       2 Input      6 Bit       Adders := 1
       2 Input      5 Bit       Adders := 6
+---Registers :
                   32 Bit    Registers := 786
                    5 Bit    Registers := 4
+---Muxes :
       2 Input      5 Bit        Muxes := 2
       2 Input      1 Bit        Muxes := 28
Module fpmull__1
Detailed RTL Component Info :
+---Adders :
       3 Input      8 Bit       Adders := 1
+---XORs :
       2 Input      1 Bit         XORs := 1
+---Muxes :
       2 Input     32 Bit        Muxes := 2
       2 Input     23 Bit        Muxes := 1
Module fpmull__2
Detailed RTL Component Info :
+---Adders :
       3 Input      8 Bit       Adders := 1
+---XORs :
       2 Input      1 Bit         XORs := 1
+---Muxes :
       2 Input     32 Bit        Muxes := 2
       2 Input     23 Bit        Muxes := 1
```

DSP:

Module Name	DSP Mapping	Neg Edge Clk	A Size	B Size	C Size	D Size	P Size	AREG	BREG	CREG	DREG	ADREG	MREG	PREG
fpmull	A*B	No	25	18	48	25	48	0	0	1	1	1	0	0
fpmull	(PCIN>>17)+A*B	No	25	8	31	25	31	0	0	1	1	1	0	0
fpmull	A*B	No	25	18	48	25	48	0	0	1	1	1	0	0
fpmull	(PCIN>>17)+A*B	No	25	8	31	25	31	0	0	1	1	1	0	0
fpmull	A*B	No	25	18	48	25	48	0	0	1	1	1	0	0
fpmull	(PCIN>>17)+A*B	No	25	8	31	25	31	0	0	1	1	1	0	0
fpmull	A*B	No	25	18	48	25	48	0	0	1	1	1	0	0
fpmull	(PCIN>>17)+A*B	No	25	8	31	25	31	0	0	1	1	1	0	0
fpmull	A*B	No	25	18	48	25	48	0	0	1	1	1	0	0
fpmull	(PCIN>>17)+A*B	No	25	8	31	25	31	0	0	1	1	1	0	0
fpmull	A*B	No	25	18	48	25	48	0	0	1	1	1	0	0
fpmull	(PCIN>>17)+A*B	No	25	8	31	25	31	0	0	1	1	1	0	0
fpmull	A*B	No	25	18	48	25	48	0	0	1	1	1	0	0
fpmull	(PCIN>>17)+A*B	No	25	8	31	25	31	0	0	1	1	1	0	0
fpmull	A*B	No	25	18	48	25	48	0	0	1	1	1	0	0
fpmull	(PCIN>>17)+A*B	No	25	8	31	25	31	0	0	1	1	1	0	0
fpmull	A*B	No	25	18	48	25	48	0	0	1	1	1	0	0
fpmull	(PCIN>>17)+A*B	No	25	8	31	25	31	0	0	1	1	1	0	0
fpmull	A*B	No	25	18	48	25	48	0	0	1	1	1	0	0
fpmull	(PCIN>>17)+A*B	No	25	8	31	25	31	0	0	1	1	1	0	0
fpmull	A*B	No	25	18	48	25	48	0	0	1	1	1	0	0
fpmull	(PCIN>>17)+A*B	No	25	8	31	25	31	0	0	1	1	1	0	0
fpmull	A*B	No	25	18	48	25	48	0	0	1	1	1	0	0
fpmull	(PCIN>>17)+A*B	No	25	8	31	25	31	0	0	1	1	1	0	0

3- Tanh_2D_parallel_filters module

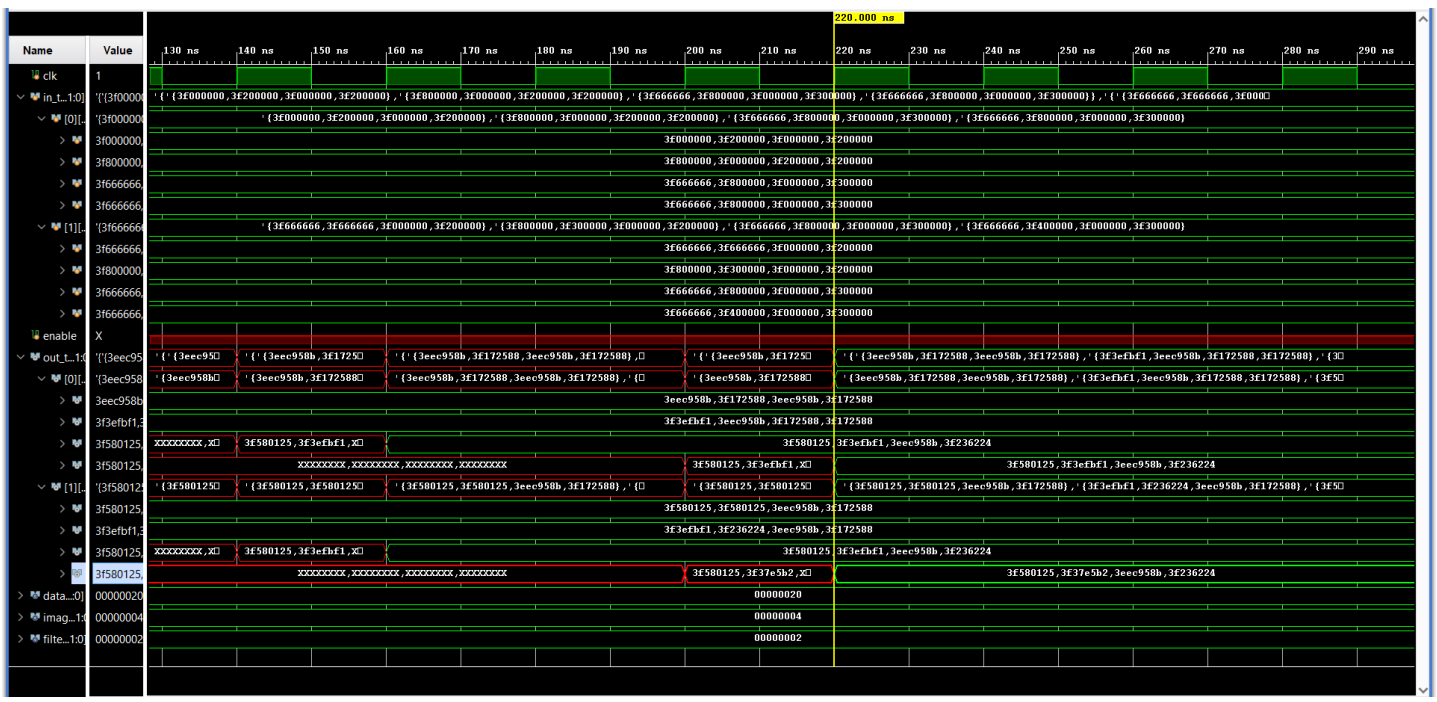
This module activates multiple images at the same time where each image is also being activated two pixels at a time (2D parallel layer).

→ This module creates a number of tanh_2D modules that is equal to the number of filters needed and iterates on the 3D input array to send a 2D array [i] to the instantiated tanh_2D objects.

A) Testbench simulation:

This testbench consists of 2x4x4 array that is activated 2 pixels per cycle and both filters are in parallel.

Total number of cycles for 2x4x4 input= Total number of cycles for one image 4x4 = 11 cycles.



B) Synthesis and Utilization:

1. Utilization:

1. CLB Logic

Site Type	Used	Fixed	Available	Util%
CLB LUTs*	175082	0	242400	72.23
LUT as Logic	175082	0	242400	72.23
LUT as Memory	0	0	112800	0.00
CLB Registers	152512	0	484800	31.46
Register as Flip Flop	152224	0	484800	31.40
Register as Latch	288	0	484800	0.06
CARRY8	756	0	30300	2.50
F7 Muxes	24	0	121200	0.02
F8 Muxes	0	0	60600	0.00
F9 Muxes	0	0	30300	0.00

2. Synthesis:

Report RTL Partitions:

	RTL Partition	Replication	Instances
1	tanh	2	35468
2	tanh_2D_1f__GCBM0	1	28034
3	tanh_2D_1f__GCBM1	1	21613
4	tanh_2D_1f__GCBM2	1	18827
5	tanh_2D_1f__GCBM3	1	23398
6	tanh_2D_1f__GCBM4	1	30943
7	tanh_2D_1f__GCBM5	1	25112
8	tanh_2D_1f__GCBM6	1	26959
9	tanh_2D_1f__GCBM7	1	10131
10	tanh_2D_1f__GCBM8	1	25344
11	tanh_2D_1f__GCBM9	1	27305
12	tanh_2D_1f__GCBM10	1	11928
13	tanh_2D_1f__GCBM11	1	13546
14	tanh_2D_1f__GCBM12	1	33944
15	tanh_2D_1f__GCBM13	1	15303
16	tanh_2D_1f__GCBM14	1	19892
17	tanh_2D_1f__GCBM15	1	22439
18	tanh_2D_1f__GCBM16	1	18501
19	tanh_2D_1f__GCBM17	1	26288
20	tanh_2D_1f__GCBM18	1	9833
21	tanh_2D_1f__GCBM19	1	26803
22	tanh_2D_1f__GCBM20	1	19896
23	tanh_2D_1f__GCBM21	1	24566
24	tanh_2D_1f__GCBM22	1	28463
25	tanh_2D_1f__GCBM23	1	20782
26	tanh_2D_1f__GCBM24	1	22823
27	tanh_2D_1f__GCBM25	1	13451
28	tanh_2D_1f__GCBM26	1	32758
29	tanh_2D_1f__GCBM27	1	21135
30	tanh_2D_1f__GCBM28	1	30656
31	tanh_2D_1f__GCBM29	1	27025
32	tanh_2D_1f__GCBM30	1	20855
33	tanh_2D_1f__GCBM31	1	17457
34	tanh_2D_1f__GCBM32	1	11836
35	tanh_2D_1f__GCBM33	1	25758
36	tanh_2D_1f__GCBM34	1	19693
37	tanh_2D_1f__GCBM35	1	31693
38	tanh_2D_1f__GCBM36	1	28350
39	tanh_2D_1f__GCBM37	1	8943
40	tanh_2D_1f__GCBM38	1	30368
41	tanh_2D_1f__GCBM39	1	11758
42	tanh_2D_1f__GCBM40	1	22333
43	tanh_2D_1f__GCBM41	1	33774
44	tanh_2D_1f__GCBM42	1	8737
45	tanh_2D_1f__GCBM43	1	11074
46	tanh_2D_1f__GCBM44	1	14725

Tanh_parallel_Synth_Report - Notepad

File Edit Format View Help

29	tanh_2D_1f__GCBM27	1	21135
30	tanh_2D_1f__GCBM28	1	30656
31	tanh_2D_1f__GCBM29	1	27025
32	tanh_2D_1f__GCBM30	1	20855
33	tanh_2D_1f__GCBM31	1	17457
34	tanh_2D_1f__GCBM32	1	11836
35	tanh_2D_1f__GCBM33	1	25758
36	tanh_2D_1f__GCBM34	1	19693
37	tanh_2D_1f__GCBM35	1	31693
38	tanh_2D_1f__GCBM36	1	28350
39	tanh_2D_1f__GCBM37	1	8943
40	tanh_2D_1f__GCBM38	1	30368
41	tanh_2D_1f__GCBM39	1	11758
42	tanh_2D_1f__GCBM40	1	22333
43	tanh_2D_1f__GCBM41	1	33774
44	tanh_2D_1f__GCBM42	1	8737
45	tanh_2D_1f__GCBM43	1	11074
46	tanh_2D_1f__GCBM44	1	14725


```
No constraint files found.
-----------------------------------------------------------------------------
Start RTL Component Statistics
-----------------------------------------------------------------------------
Detailed RTL Component Info :
+---Adders :
            2 Input     32 Bit       Adders := 6
            2 Input     25 Bit       Adders := 36
            3 Input     25 Bit       Adders := 72
            3 Input      8 Bit       Adders := 300
            4 Input      8 Bit       Adders := 36
            2 Input      8 Bit       Adders := 36
            2 Input      6 Bit       Adders := 6
            2 Input      5 Bit       Adders := 36
+---XORs :
            2 Input      1 Bit         XORs := 156
+---Registers :
                        32 Bit    Registers := 4716
                         5 Bit    Registers := 24
+---Muxes :
            2 Input     32 Bit        Muxes := 420
            2 Input     24 Bit        Muxes := 180
            2 Input     23 Bit        Muxes := 192
            2 Input      8 Bit        Muxes := 1908
           23 Input      8 Bit        Muxes := 72
            2 Input      5 Bit        Muxes := 12
            2 Input      1 Bit        Muxes := 1932
-----------------------------------------------------------------------------
Finished RTL Component Statistics
-----------------------------------------------------------------------------
```

➔ Part 3:SoftMax activation function

the SoftMax activation function gives us a discrete probability distribution for every possible class and it's applied to the last layer in the CNN

- ➔ We will discuss the idea behind the probability and see if it's really useful or if there is a way to know the closet value to the class without calculating the probability and only by using the exponential function

1- softmax_fn:

- ➔ this module finds the exponential of one number at a time and since all the numbers that will enter to this module are between -1 and 1 the **Taylor function** will be perfect to use it here and will has a small amount of error that can be neglected.

- ➔ **the Taylor function:** $\mathbf{e^x} = 1 + x + \frac{x^2}{2} + \frac{x^3}{3} + \frac{x^4}{24} + \frac{x^5}{120} + \frac{x^6}{720}$

A screenshot from the simulation:

The inputs enter on the positive edge and the output is computed on the negative edge

$\rightarrow \mathbf{e^0} = 0x3f800000$

$\rightarrow \mathbf{e^1} = 0x402df49d$

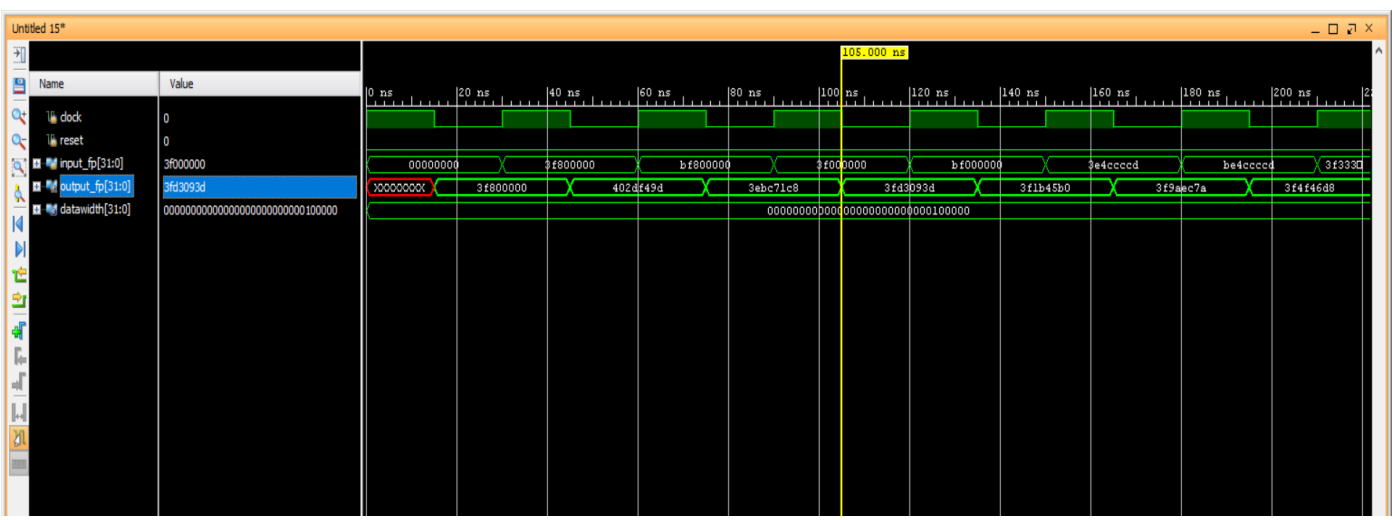
$\rightarrow \mathbf{e^{-1}} = 0x3ebc71c8$

$\rightarrow \mathbf{e^{0.5}} = 0x3fd3093d$

$\rightarrow \mathbf{e^{-0.5}} = 0x3f1b45b0$

$\rightarrow \mathbf{e^{0.2}} = 0x3f9aec7a$

$\rightarrow \mathbf{e^{0.6}} = 0x3f4f6d8$



➔ *Utilization:*

1. CLB Logic

Site Type	Used	Fixed	Available	Util%
CLB LUTs*	3577	0	242400	1.48
LUT as Logic	3577	0	242400	1.48
LUT as **Memory**	0	0	112800	0.00
CLB Registers	80	0	484800	0.02
Register as Flip Flop	32	0	484800	<0.01
Register as Latch	48	0	484800	<0.01
CARRY8	104	0	30300	0.34
F7 Muxes	5	0	121200	<0.01
F8 Muxes	0	0	60600	0.00
F9 Muxes	0	0	30300	0.00

[reports ultrasonic\utilization report softmax]

➔ *Synthesis:*

Report RTL Partitions:

RTL Partition	Replication	Instances

Start ROM, RAM, DSP **and** Shift Register Reporting

DSP:

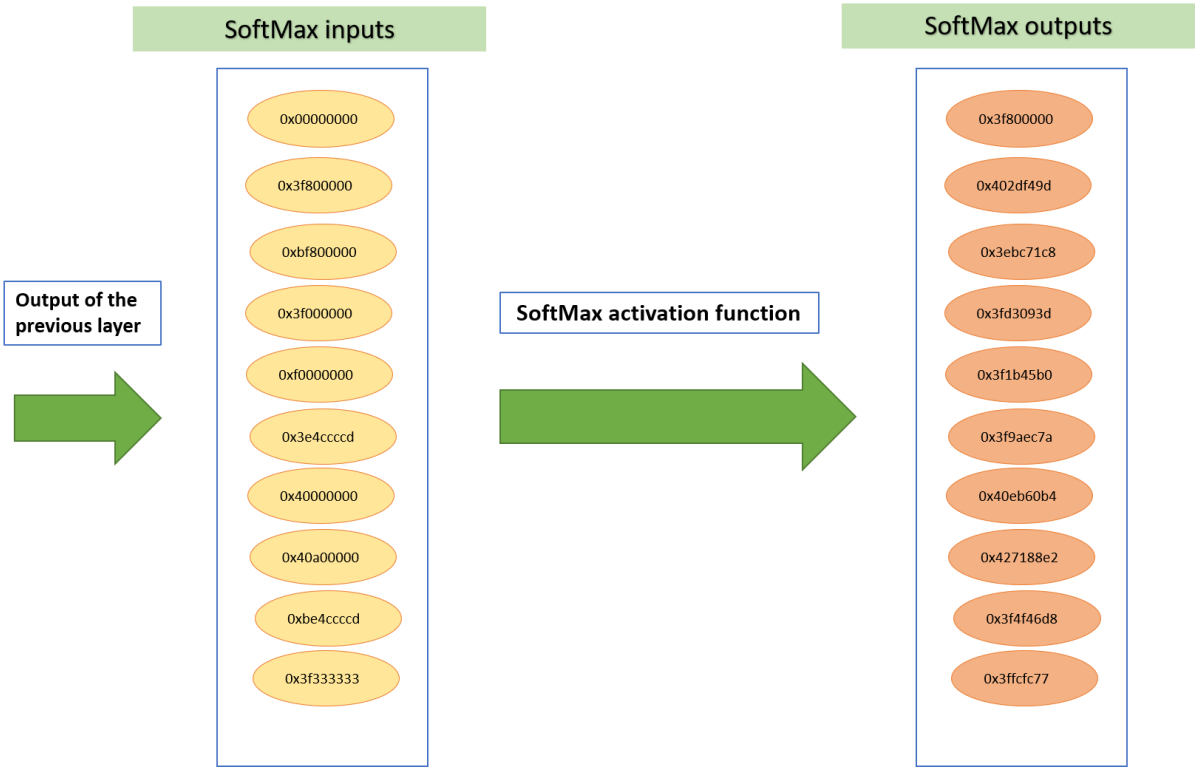
Module Name	DSP Mapping	**Neg** Edge Clk	A Size	B Size	**C** Size	D Size	P Size	AREG	BREG	CREG	DREG	ADREG	MREG	PREG
fpmull	A*B	No	25	18	48	25	48	0	0	1	1	1	0	0
fpmull	(PCIN>>17)+A*B	No	25	8	31	25	31	0	0	1	1	1	0	0
fpmull	A*B	No	25	18	48	25	48	0	0	1	1	1	0	0
fpmull	(PCIN>>17)+A*B	No	25	8	31	25	31	0	0	1	1	1	0	0
fpmull	A*B	No	25	18	48	25	48	0	0	1	1	1	0	0
fpmull	(PCIN>>17)+A*B	No	25	8	31	25	31	0	0	1	1	1	0	0
fpmull	A*B	No	25	18	48	25	48	0	0	1	1	1	0	0
fpmull	(PCIN>>17)+A*B	No	25	8	31	25	31	0	0	1	1	1	0	0
fpmull	A*B	No	25	18	48	25	48	0	0	1	1	1	0	0
fpmull	(PCIN>>17)+A*B	No	25	8	31	25	31	0	0	1	1	1	0	0
fpmull	A*B	No	25	18	48	25	48	0	0	1	1	1	0	0
fpmull	(PCIN>>17)+A*B	No	25	8	31	25	31	0	0	1	1	1	0	0
fpmull	A*B	No	25	18	48	25	48	0	0	1	1	1	0	0
fpmull	(PCIN>>17)+A*B	No	25	8	31	25	31	0	0	1	1	1	0	0
fpmull	A*B	No	25	18	48	25	48	0	0	1	1	1	0	0
fpmull	(PCIN>>17)+A*B	No	25	8	31	25	31	0	0	1	1	1	0	0
fpmull	A*B	No	25	18	48	25	48	0	0	1	1	1	0	0
fpmull	(PCIN>>17)+A*B	No	25	8	31	25	31	0	0	1	1	1	0	0

[reports ultrasonic\synthesis report softmax]

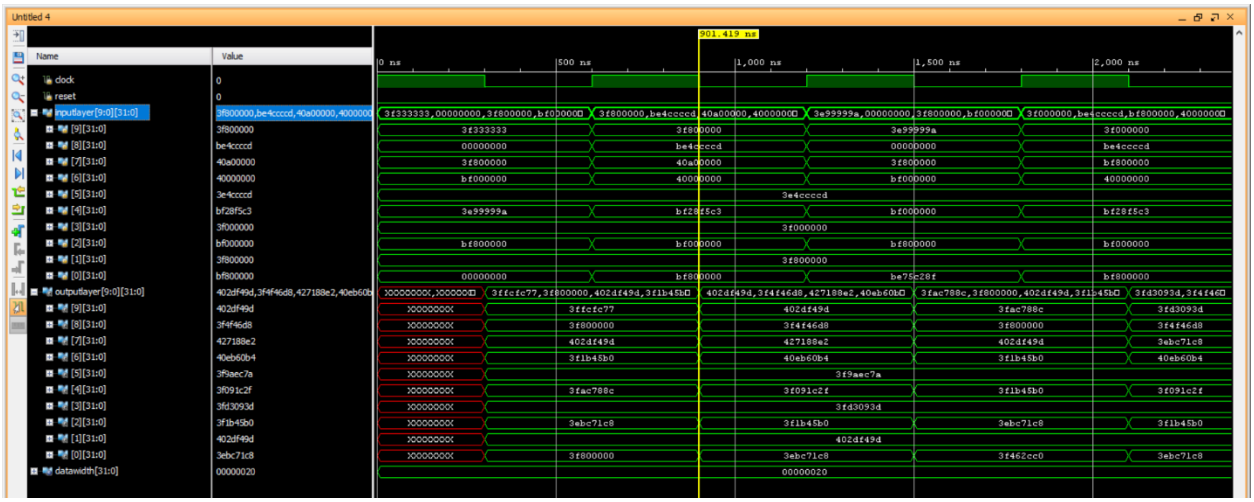
2- *softmaxLayer:*

In this module we will find the exponential of the 10 inputs in parallel in 1 clock cycle at the falling edge to find the positive value of all the inputs

the flow of the softmaxLayer module:



In this screenshot this is a part of the test bench for 4 cycles for 4 sets of inputs (4x10) were we have the 10 exponential values in 1 clock cycle at the falling edge



➔ *Utilization:*

Site Type	Used	Fixed	Available	Util%
CLB LUTs*	35470	0	242400	14.63
LUT as Logic	35470	0	242400	14.63
LUT as Memory	0	0	112800	0.00
CLB Registers	800	0	484800	0.17
Register as Flip Flop	320	0	484800	0.07
Register as Latch	480	0	484800	0.10
CARRY8	980	0	30300	3.23
F7 Muxes	50	0	121200	0.04
F8 Muxes	0	0	60600	0.00
F9 Muxes	0	0	30300	0.00

[reports ultrasonic\utilization report softmaxLayer]

➔ *Synthesis:*

Module Name	DSP Mapping	Neg Edge Clk	A Size	B Size	C Size	D Size	P Size	AREG	BREG	CREG	DREG	ADREG	MREG	PREG
fpmul	A*B	No	25	18	48	25	48	0	0	1	1	1	0	0
fpmul	(PCIN>>17)+A*B	No	25	8	31	25	31	0	0	1	1	1	0	0
fpmul	A*B	No	25	18	48	25	48	0	0	1	1	1	0	0
fpmul	(PCIN>>17)+A*B	No	25	8	31	25	31	0	0	1	1	1	0	0
fpmul	A*B	No	25	18	48	25	48	0	0	1	1	1	0	0
fpmul	(PCIN>>17)+A*B	No	25	8	31	25	31	0	0	1	1	1	0	0
fpmul	A*B	No	25	18	48	25	48	0	0	1	1	1	0	0
fpmul	(PCIN>>17)+A*B	No	25	8	31	25	31	0	0	1	1	1	0	0
fpmul	A*B	No	25	18	48	25	48	0	0	1	1	1	0	0
fpmul	(PCIN>>17)+A*B	No	25	8	31	25	31	0	0	1	1	1	0	0
fpmul	A*B	No	25	18	48	25	48	0	0	1	1	1	0	0
fpmul	(PCIN>>17)+A*B	No	25	8	31	25	31	0	0	1	1	1	0	0
fpmul	A*B	No	25	18	48	25	48	0	0	1	1	1	0	0
fpmul	(PCIN>>17)+A*B	No	25	8	31	25	31	0	0	1	1	1	0	0
fpmul	A*B	No	25	18	48	25	48	0	0	1	1	1	0	0
fpmul	(PCIN>>17)+A*B	No	25	8	31	25	31	0	0	1	1	1	0	0
fpmul	A*B	No	25	18	48	25	48	0	0	1	1	1	0	0
fpmul	(PCIN>>17)+A*B	No	25	8	31	25	31	0	0	1	1	1	0	0
fpmul	A*B	No	25	18	48	25	48	0	0	1	1	1	0	0
fpmul	(PCIN>>17)+A*B	No	25	8	31	25	31	0	0	1	1	1	0	0

[reports ultrasonic\synthesis report softmaxLayer]

The division:

- **After getting the exponential of the 10 inputs, the SoftMax after that used to find the percentage of each input by diving each exponential output by the sum of all the exponential outputs done in parallel in order to find the greatest probability among them**

But since dividing numbers by the same amount will hold the same ratio between them, we will be able to find the index of the greatest number among them SoftMax input then this index (between 0 to 9) will be the digit in the input picture

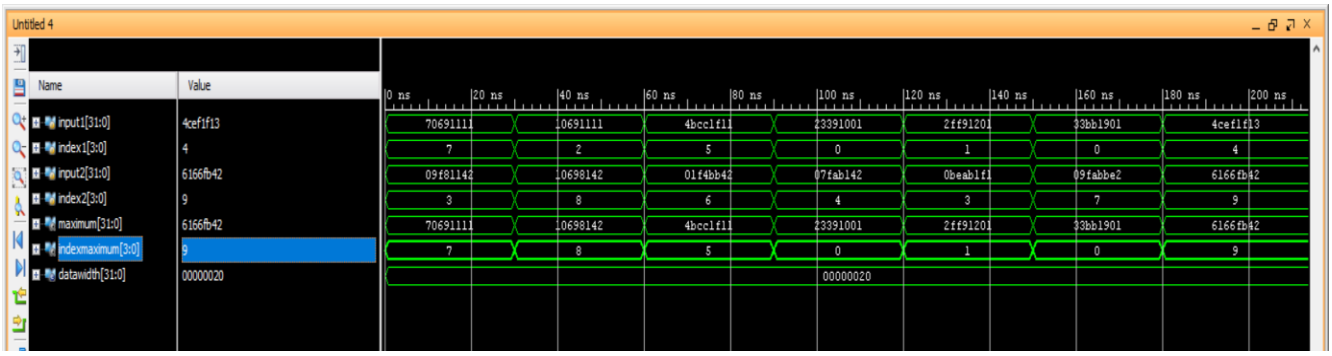
Therefore, we will find the greatest number by applying a special algorithm for the IEEE754 in 2 modules:

1st module: max

This module finds the maximum between two numbers (asynchronous) by computing the greater **exponent [30:23] bits** without taking the sign bit into consideration: as we already have only positive numbers (output of exponential)
And if the exponents are equal, we will compute the greater **mantissa [22:0] bits**

screenshot from simulation:

- **[0 ns] 1st exponent:1110 0000 and 2nd exponent: 0001 0011**
  The biggest number in the 1st number
- **[30 ns] 1st exponent:0010 0000 and 2nd exponent:0010 0000**
  In this case we will have to find the biggest mantissa
  1st mantissa: 11010010001000100010001
  2nd mantissa:11010011000000101000010
  The biggest number is the 2nd number



➔ *Utilization:*

Site Type	Used	Fixed	Available	Util%
CLB LUTs*	22	0	242400	<0.01
LUT as Logic	22	0	242400	<0.01
LUT as Memory	0	0	112800	0.00
CLB Registers	0	0	484800	0.00
Register as Flip Flop	0	0	484800	0.00
Register as Latch	0	0	484800	0.00
CARRY8	1	0	30300	<0.01
F7 Muxes	0	0	121200	0.00
F8 Muxes	0	0	60600	0.00
F9 Muxes	0	0	30300	0.00

reports ultrasonic\utilization report max

➔ *Synthesis:*

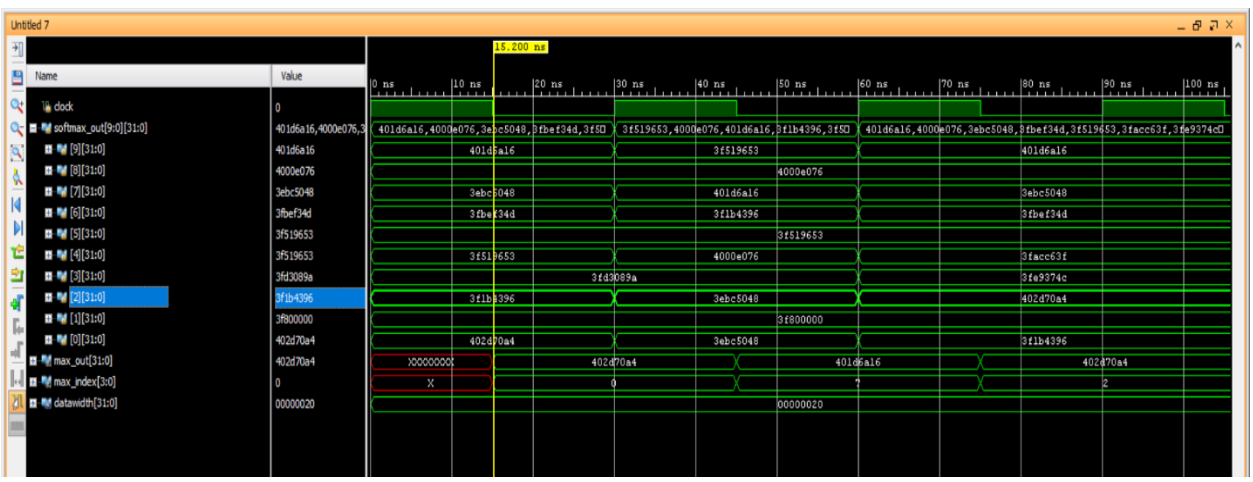
```
+------+---------+-------+------+
|      |Instance |Module |Cells |
+------+---------+-------+------+
|1     |top      |       |   153|
+------+---------+-------+------+
-------------------------------------------------------------------------------
Finished Writing Synthesis Report : Time (s): cpu = 00:00:11 ; elapsed = 00:00:12 . Memory (MB): peak = 672.145 ; gain = 459.719
-------------------------------------------------------------------------------
Synthesis finished with 0 errors, 0 critical warnings and 0 warnings.
Synthesis Optimization Runtime : Time (s): cpu = 00:00:09 ; elapsed = 00:00:09 . Memory (MB): peak = 672.145 ; gain = 437.699
Synthesis Optimization Complete : Time (s): cpu = 00:00:11 ; elapsed = 00:00:12 . Memory (MB): peak = 672.145 ; gain = 459.719
INFO: [Project 1-571] Translating synthesized netlist
INFO: [Netlist 29-17] Analyzing 72 Unisim elements for replacement
INFO: [Netlist 29-28] Unisim Transformation completed in 0 CPU seconds
INFO: [Project 1-570] Preparing netlist for logic optimization
INFO: [Opt 31-138] Pushed 0 inverter(s) to 0 load pin(s).
INFO: [Project 1-111] Unisim Transformation Summary:
  A total of 72 instances were transformed.
  IBUF => IBUF (IBUFCTRL, INBUF): 72 instances

INFO: [Common 17-83] Releasing license: Synthesis
11 Infos, 0 Warnings, 0 Critical Warnings and 0 Errors encountered.
synth_design completed successfully
synth_design: Time (s): cpu = 00:00:10 ; elapsed = 00:00:11 . Memory (MB): peak = 713.504 ; gain = 491.715
report_utilization: Time (s): cpu = 00:00:00 ; elapsed = 00:00:00.059 . Memory (MB): peak = 713.504 ; gain = 0.000
INFO: [Common 17-206] Exiting Vivado at Mon Jun 08 12:50:11 2020...
```

reports ultrasonic\vivado synthese max

➔ *2nd module: comparingSoftmax*

In this module we give it 10 numbers (the exponential output) and then it gives us the maximum number and its index in 1 clock cycle at the negative edge
This index is the number of the digit in the picture and by this procedure we saved space in the FPGA for other activation function or maybe more precision if we needed to improve the CNN

In this screenshot:
1st cycle: $e^{1}, e^{0}, e^{-0.5}, e^{0.5}, e^{0}, e^{-0.2}, e^{0.4}, e^{-1}, e^{0.7}, e^{0.9}$
and the maximum index is zero → e^{1}
→this means the digit in the picture is zero
2nd cycle: $e^{-1}, e^{0}, e^{-1}, e^{0.5}, e^{0.7}, e^{-0.2}, e^{-0.5}, e^{0.9}, e^{0.7}, e^{-0.2}$
and the maximum index is 7 → $e^{0.9}$
→this means the digit in the picture is seven



➔ *Utilization:*

Site Type	Used	Fixed	Available	Util%
CLB LUTs*	540	0	242400	0.22
LUT as Logic	540	0	242400	0.22
LUT as Memory	0	0	112800	0.00
CLB Registers	36	0	484800	<0.01
Register as Flip Flop	36	0	484800	<0.01
Register as Latch	0	0	484800	0.00
CARRY8	36	0	30300	0.12
F7 Muxes	0	0	121200	0.00
F8 Muxes	0	0	60600	0.00
F9 Muxes	0	0	30300	0.00

reports ultrasonic\utilization report comparingSoftmax

➔ *Synthesis:*

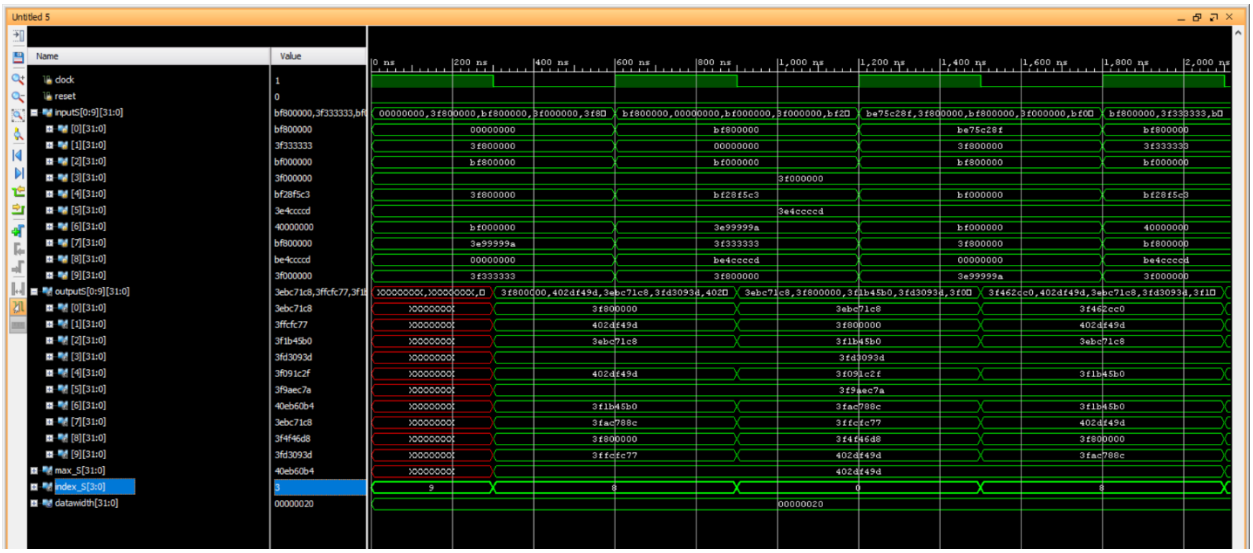
```
Report Cell Usage:
+------+-------+------+
|      |Cell   |Count |
+------+-------+------+
|1     |BUFG   |     1|
|2     |CARRY8 |    36|
|3     |LUT1   |     1|
|4     |LUT2   |     2|
|5     |LUT3   |     9|
|6     |LUT4   |   228|
|7     |LUT5   |   281|
|8     |LUT6   |   137|
|9     |FDRE   |    36|
|10    |IBUF   |   321|
|11    |OBUF   |    36|
+------+-------+------+

Report Instance Areas:
+------+---------+-------+------+
|      |Instance |Module |Cells |
+------+---------+-------+------+
|1     |top      |       |  1088|
+------+---------+-------+------+
-----------------------------------------------------------------------------
Finished Writing Synthesis Report : Time (s): cpu = 00:00:13 ; elapsed = 00:00:14 . Memory (MB): peak = 767.574 ; gain = 554.938
-----------------------------------------------------------------------------
Synthesis finished with 0 errors, 0 critical warnings and 0 warnings.
```

reports ultrasonic\vivado synthese comapringSoftmax

softmax_max_index:

this module is the integration of all previous modules it takes the 10 neurons of the fully connected layer and gives the exponential of the 10 inputs and it computes the maximum value and its index in the inputs to map it to the desired digit

screenshot from the testbench were the index_S is the index of the greatest number and this module takes only 1 clock cycle and gives the output on the falling edge



➔ *Utilization:*

Site Type	Used	Fixed	Available	Util%
CLB LUTs*	36012	0	242400	14.86
LUT as Logic	36012	0	242400	14.86
LUT as Memory	0	0	112800	0.00
CLB Registers	800	0	484800	0.17
Register as Flip Flop	320	0	484800	0.07
Register as Latch	480	0	484800	0.10
CARRY8	1016	0	30300	3.35
F7 Muxes	50	0	121200	0.04
F8 Muxes	0	0	60600	0.00
F9 Muxes	0	0	30300	0.00

..\Desktop\reports ultrasonic\utilization\utilization softmax_max_index

➔ *Synthesis:*

Module Name	DSP Mapping	Neg Edge Clk	A Size	B Size	C Size	D Size	P Size	AREG	BREG	CREG	DREG	ADREG	MREG	PREG
fpmul	A*B	No	25	18	48	25	48	0	0	1	1	1	0	0
fpmul	(PCIN>>17)+A*B	No	25	8	31	25	31	0	0	1	1	1	0	0
fpmul	A*B	No	25	18	48	25	48	0	0	1	1	1	0	0
fpmul	(PCIN>>17)+A*B	No	25	8	31	25	31	0	0	1	1	1	0	0
fpmul	A*B	No	25	18	48	25	48	0	0	1	1	1	0	0
fpmul	(PCIN>>17)+A*B	No	25	8	31	25	31	0	0	1	1	1	0	0
fpmul	A*B	No	25	18	48	25	48	0	0	1	1	1	0	0
fpmul	(PCIN>>17)+A*B	No	25	8	31	25	31	0	0	1	1	1	0	0
fpmul	A*B	No	25	18	48	25	48	0	0	1	1	1	0	0
fpmul	(PCIN>>17)+A*B	No	25	8	31	25	31	0	0	1	1	1	0	0
fpmul	A*B	No	25	18	48	25	48	0	0	1	1	1	0	0
fpmul	(PCIN>>17)+A*B	No	25	8	31	25	31	0	0	1	1	1	0	0
fpmul	A*B	No	25	18	48	25	48	0	0	1	1	1	0	0
fpmul	(PCIN>>17)+A*B	No	25	8	31	25	31	0	0	1	1	1	0	0
fpmul	A*B	No	25	18	48	25	48	0	0	1	1	1	0	0
fpmul	(PCIN>>17)+A*B	No	25	8	31	25	31	0	0	1	1	1	0	0
fpmul	A*B	No	25	18	48	25	48	0	0	1	1	1	0	0
fpmul	(PCIN>>17)+A*B	No	25	8	31	25	31	0	0	1	1	1	0	0

..\Desktop\reports ultrasonic\synthesis\synthesis report softmax_max_index

➔ Part 4: Average Pool Layer

This stage of the CNN reduces the data/ pixels of the input image by calculating the average of pixels for each 2x2 window, with a stride of 2. This part was implemented using 3 modules (3 Layers), to handle each dimension of the input (3D input).
Note: All modules work on the rising edge of the clock.

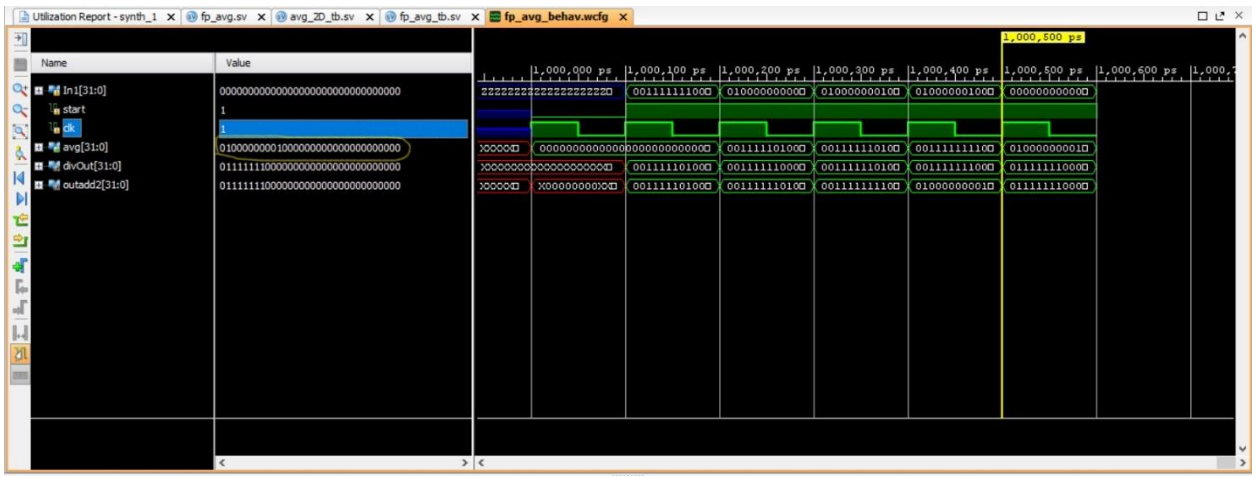
a. Fp_avg Module (1st Layer)

The 1st module has 1 input (1 pixel) in addition to a clock, and a start bit, and 1 output. It uses **fp_mul** & **fp_add** modules (Previously implemented).

Since the average pool layer always calculates the average of 4 pixels, **fp_mul** was modified to multiply the input pixel by ¼. This means that each 4 outputs of the **fp_mul** should be accumulated to have the final average result of the 2x2 window.
Noting that: $\frac{A+B+C+D}{4} = \frac{A}{4} + \frac{B}{4} + \frac{C}{4} + \frac{D}{4}$

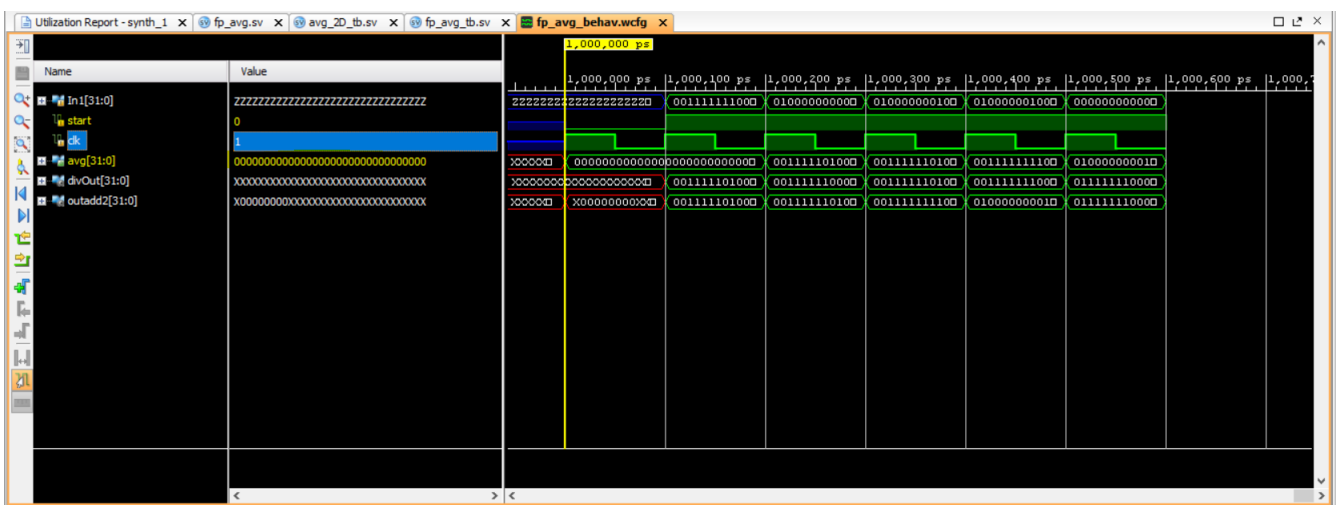
1. The module divides the input pixel over 4 (using **fp_mul**)
2. It then accumulates the output of **fp_mul** (using **fp_add**)
3. The accumulation is reset if the start bit = 0.

- **Simulation of The Test Bench**
  - A simple test bench was created **(fp_avg_tb)** to show the results & clock cycles of the **fp_avg** module.
  - It takes 4 inputs & accumulates them. Inputs are: 1,2,3, & 4 in IEEE-754 format.
  - 1st Rising edge in the screenshot is to initialize the



accumulation. (start=0).

  - Starting from the 2nd rising edge the inputs are added.
  - At the last rising edge, the result (avg) is stored which equals 2.5 (Highlighted in yellow).



Important Observation: Typically, the **fp_avg** should take 4 cycles to calculate the average of the 4 pixels - aside from another where start=0 -, however, during testing, it was noticed that the accumulation is always calculated first (the always block is done before the instantiation), this means that the data is ready in the 4th cycle, but will be assigned to the output in the 5th cycle. So the whole operation takes 5 cycles + the initialization cycle. (Total of 6 cycles)

- **Synthesis Report**

```
Module fp_avg
Detailed RTL Component Info :
+---Registers :
                    32 Bit    Registers := 1
Module fp_mul
Detailed RTL Component Info :
+---Adders :
        2 Input      8 Bit       Adders := 1
Module fp_add
Detailed RTL Component Info :
+---Adders :
        2 Input     25 Bit       Adders := 1
        3 Input     25 Bit       Adders := 2
        3 Input      8 Bit       Adders := 4
        4 Input      8 Bit       Adders := 1
        2 Input      8 Bit       Adders := 1
+---Muxes :
        2 Input     32 Bit        Muxes := 3
        2 Input     24 Bit        Muxes := 5
        2 Input     23 Bit        Muxes := 1
        2 Input      8 Bit        Muxes := 53
       23 Input      8 Bit        Muxes := 2
```

```
Report Cell Usage:
+------+-------+------+
|      |Cell   |Count |
+------+-------+------+
|1     |BUFG   |     1|
|2     |CARRY8 |    22|
|3     |LUT1   |    61|
|4     |LUT2   |    82|
|5     |LUT3   |    72|
|6     |LUT4   |    44|
|7     |LUT5   |   191|
|8     |LUT6   |   327|
|9     |MUXF7  |     2|
|10    |FDRE   |    32|
|11    |LD     |     8|
|12    |IBUF   |    34|
|13    |OBUF   |    32|
+------+-------+------+
```

- **Utilization Report**

```
1. CLB Logic
------------

+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| CLB LUTs*               |  680 |     0 |    242400 |  0.28 |
|   LUT as Logic          |  680 |     0 |    242400 |  0.28 |
|   LUT as Memory         |    0 |     0 |    112800 |  0.00 |
| CLB Registers           |   40 |     0 |    484800 | <0.01 |
|   Register as Flip Flop |   32 |     0 |    484800 | <0.01 |
|   Register as Latch     |    8 |     0 |    484800 | <0.01 |
| CARRY8                  |   22 |     0 |     30300 |  0.07 |
| F7 Muxes                |    2 |     0 |    121200 | <0.01 |
| F8 Muxes                |    0 |     0 |     60600 |  0.00 |
| F9 Muxes                |    0 |     0 |     30300 |  0.00 |
+-------------------------+------+-------+-----------+-------+
```

b. AvgLayer Module (2nd Layer)

This layer takes the 2D array of pixels (Pixels of the image) as an input, and calculates output pixels of the image after averaging each 2x2 window. So if 2D array is NXN, the resulting 2D array will be of $\frac{N}{2} X \frac{N}{2}$. It uses the **fp_avg** module.

The layer loops over the 2D array with 2 indices for the rows & columns, and it uses a counter to assign the output - average of the 2x2 window – in its location in the output 2D array (using another 2 indices).

- **Sequence of the Implementation**

1. All used registers (reg) are initialized when the start bit = 0, including the start bit of **fp_avg** (to initialize accumulation).
2. When start = 1, the module starts looping over the input array, it sends a new pixel to **fp_avg** each cycle.

During these loops, things get a little tricky, since the output of **fp_avg** (the average of a window) is ready at the 5th cycle as mentioned in the previous observation, the same concept will be applied in this layer. Which means that the assigning of **avg** will be correctly assigned in the 6th cycle in this module (Since the always block is done before the instantiation, as mentioned). In the 5th cycle, the **fp_avg** start bit will be 0 (to reset the accumulation), also to avoid wrong assigning of the pixels, no pixels from the input array will be sent to **fp_avg** (because this is the cycle where we reset the accumulation).

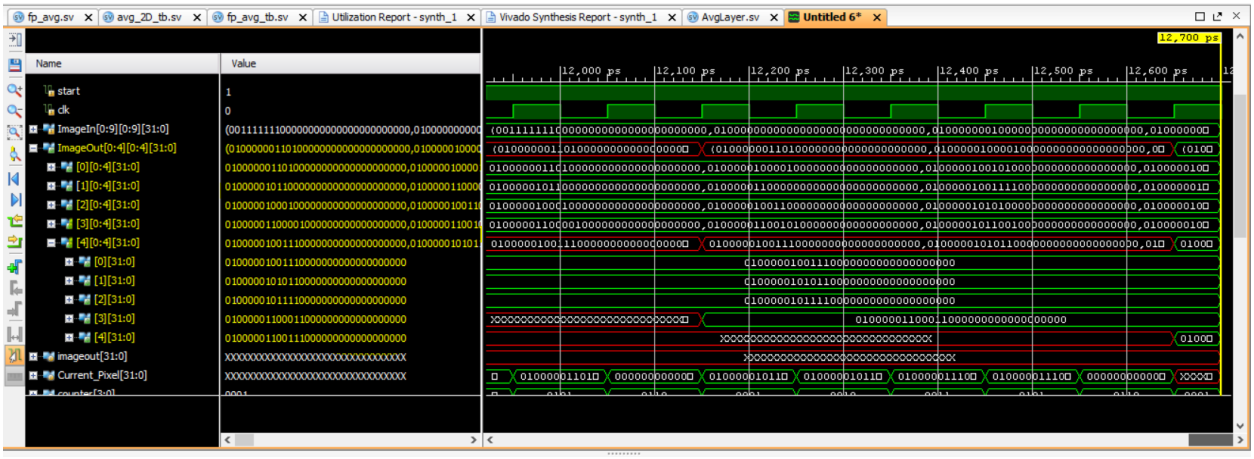
So the handled cycles in the code are the 5th, and 6th cycles. **So:**

5th Cycle: Sent pixel - to **fp_avg** - is 0. Accumulation is reset (by the start bit)

6th Cycle: Average of the window is assigned to the output array (Image after averaging). The counter used to count the cycles will be reset.

- **Simulation of the Test Bench**

Since each pixel of the 2D array is assigned in a cycle, we have number of cycles that = NxN cycles (where N is the dimension of the image). Also, as mentioned above, there is an extra cycle in **fp_avg** where the



final output is ready (the average of a 2x2 window). So we have another $\frac{NxN}{4}$ cycles, adding another cycle for the last output to be assigned in the output image (the 6th cycle for the last averaging), and also the very 1st cycle where we reset all registers.

So, **total number of cycles = (**NxN) + $\frac{NXN}{4}$ + 2

The Average Pool Layer in this CNN takes either a 28x28 image or 10x10 image. A test bench was implemented on the 10x10 case. **(avg_2D_tb)**

So, **total number of cycles in this case** = (10x10) + $\frac{10x10}{4}$ + 2 = 127 Cycles

The period of the clock here =100 ps, so the simulation should stop at 12,700 ps for the output image to be ready (Highlighted in yellow).

For simplicity & easy tracing for the module, the pixels of the input image in this test bench were from 1 to 25 repeated 4 times (in IEEE-754 format).

So the 2D array would be something similar to : $\begin{bmatrix} 1 & \cdots & 10 \\ \vdots & \dots\ 25\ \dots & \vdots \\ 15 & \dots & 25 \end{bmatrix}$

In this case, the last pixel of the output image would = $\frac{14+15+24+25}{4} = 19.5$

We can see that ImageOut [4] [4] = the IEEE-754 format of 19.5, meaning that the sequence is efficiently working. So, the module is working correctly.

- **Synthesis Report**

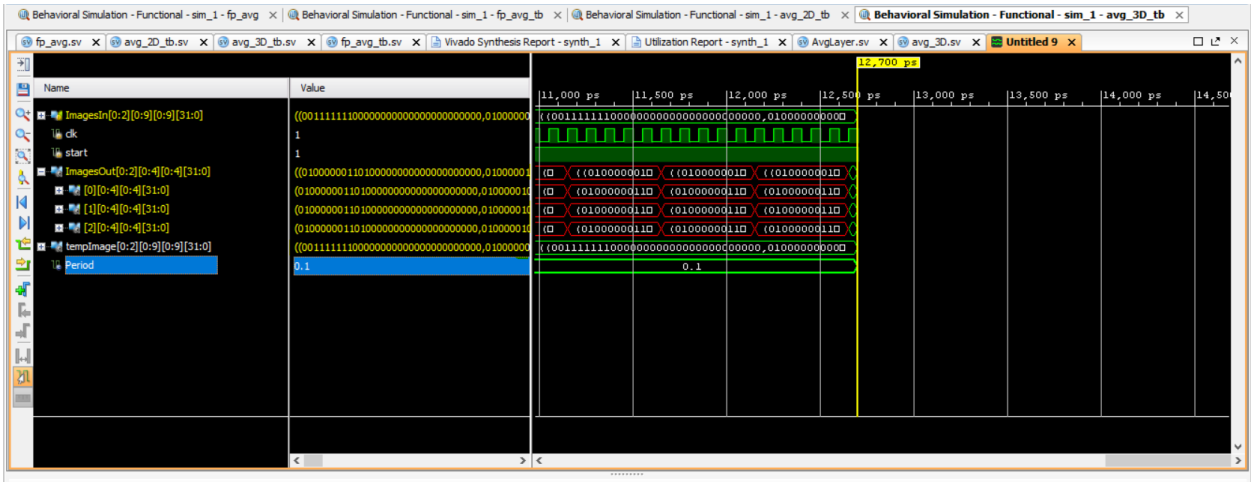
```
Detailed RTL Component Info :
+---Adders :
       2 Input     25 Bit       Adders := 1
       3 Input     25 Bit       Adders := 2
       2 Input      8 Bit       Adders := 2
       3 Input      8 Bit       Adders := 4
       4 Input      8 Bit       Adders := 1
       2 Input      5 Bit       Adders := 9
       2 Input      4 Bit       Adders := 1
       2 Input      2 Bit       Adders := 1
+---Registers :
                   32 Bit    Registers := 27
                    5 Bit    Registers := 6
                    2 Bit    Registers := 1
                    1 Bit    Registers := 2
```

```
Hierarchical RTL Component report
Module AvgLayer
Detailed RTL Component Info :
+---Adders :
       2 Input      5 Bit       Adders := 9
       2 Input      4 Bit       Adders := 1
       2 Input      2 Bit       Adders := 1
+---Registers :
                   32 Bit    Registers := 26
                    5 Bit    Registers := 6
                    2 Bit    Registers := 1
                    1 Bit    Registers := 2
+---Muxes :
       2 Input     32 Bit        Muxes := 5
       2 Input      5 Bit        Muxes := 4
       2 Input      4 Bit        Muxes := 111
       4 Input      4 Bit        Muxes := 1
       2 Input      3 Bit        Muxes := 1
       2 Input      1 Bit        Muxes := 128
       4 Input      1 Bit        Muxes := 26
       3 Input      1 Bit        Muxes := 1
```

```
Report Instance Areas:
+------+--------------+-------+------+
|      |Instance      |Module |Cells |
+------+--------------+-------+------+
|1     |top           |       |  7917|
|2     |  AveragePool |fp_avg |   777|
|3     |    add2      |fp_add |   381|
+------+--------------+-------+------+
```

- **Utilization Report**

```
1. CLB Logic
------------

+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| CLB LUTs*               | 2516 |     0 |    242400 |  1.04 |
|   LUT as Logic          | 2516 |     0 |    242400 |  1.04 |
|   LUT as Memory         |    0 |     0 |    112800 |  0.00 |
| CLB Registers           |  923 |     0 |    484800 |  0.19 |
|   Register as Flip Flop |  915 |     0 |    484800 |  0.19 |
|   Register as Latch     |    8 |     0 |    484800 | <0.01 |
| CARRY8                  |   22 |     0 |     30300 |  0.07 |
| F7 Muxes                |  258 |     0 |    121200 |  0.21 |
| F8 Muxes                |   96 |     0 |     60600 |  0.16 |
| F9 Muxes                |    0 |     0 |     30300 |  0.00 |
+-------------------------+------+-------+-----------+-------+
```



c. Avg_3D Module (3rd Layer)

This layer is the last in the Average Pool part. It has a 3D array input (Multiple images), and produces another 3D array with the same number of images, but with rows and columns reduced to half (As they're averaged).
The implementation of this layer is simple. It loops over the images- which are 2D arrays -, and sends an image to the **AvgLayer** module to do its averaging.

- **Simulation of The Test Bench**

A test bench is created **(avg_3D_tb)** where the input for the module is of dimensions 3x10x10 (3 Images, each are 10x10). In the CNN, 16 or 5 images will be sent to the module, but the same concept is applied regardless of the number of the images. So, if the algorithm works in this case, it should work efficiently in any other case.

Since the correctness of the output values for each image was tested in the 2nd, this test bench is just to check if the layer assigns each image correctly to the 2nd layer. We can see that by the end of the cycles, all 3 output images were assigned successfully.

- ✓ The 3rd layer assigns the 3 images at the same time achieving **Parallelism.** Which means that 3rd layer takes the same number of cycles that the 2nd layer. As seen in the previous screenshot, all output images are ready after 127 cycles (127,000 ps) (Highlighted in yellow).

- **Synthesis Report**

```
Report Cell Usage:
+------+-------+------+
|      |Cell   |Count |
+------+-------+------+
|1     |BUFG   |     1|
|2     |CARRY8 |    66|
|3     |LUT1   |   169|
|4     |LUT2   |   264|
|5     |LUT3   |   300|
|6     |LUT4   |   237|
|7     |LUT5   |   636|
|8     |LUT6   |  6198|
|9     |MUXF7  |   774|
|10    |MUXF8  |   288|
|11    |FDRE   |  2721|
|12    |FDSE   |    24|
|13    |LD     |    24|
|14    |IBUF   |  9602|
|15    |OBUF   |  2400|
+------+-------+------+
```

```
Module fp_avg
Detailed RTL Component Info :
+---Registers :
                    32 Bit    Registers := 1
Module AvgLayer
Detailed RTL Component Info :
+---Adders :
       2 Input      5 Bit       Adders := 9
       2 Input      4 Bit       Adders := 1
       2 Input      2 Bit       Adders := 1
+---Registers :
                    32 Bit    Registers := 26
                     5 Bit    Registers := 6
                     2 Bit    Registers := 1
                     1 Bit    Registers := 2
+---Muxes :
       2 Input     32 Bit         Muxes := 5
       2 Input      5 Bit         Muxes := 4
       2 Input      4 Bit         Muxes := 111
       4 Input      4 Bit         Muxes := 1
       2 Input      3 Bit         Muxes := 1
       2 Input      1 Bit         Muxes := 128
       4 Input      1 Bit         Muxes := 26
       3 Input      1 Bit         Muxes := 1
```

- **Utilization Report**

```
1. CLB Logic
------------

+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| CLB LUTs*               | 7498 |     0 |    242400 |  3.09 |
|   LUT as Logic          | 7498 |     0 |    242400 |  3.09 |
|   LUT as Memory         |    0 |     0 |    112800 |  0.00 |
| CLB Registers           | 2769 |     0 |    484800 |  0.57 |
|   Register as Flip Flop | 2745 |     0 |    484800 |  0.57 |
|   Register as Latch     |   24 |     0 |    484800 | <0.01 |
| CARRY8                  |   66 |     0 |     30300 |  0.22 |
| F7 Muxes                |  774 |     0 |    121200 |  0.64 |
| F8 Muxes                |  288 |     0 |     60600 |  0.48 |
| F9 Muxes                |    0 |     0 |     30300 |  0.00 |
+-------------------------+------+-------+-----------+-------+
```

✓ Part 5: Non-pipelined LeNet Convolution Neural Network

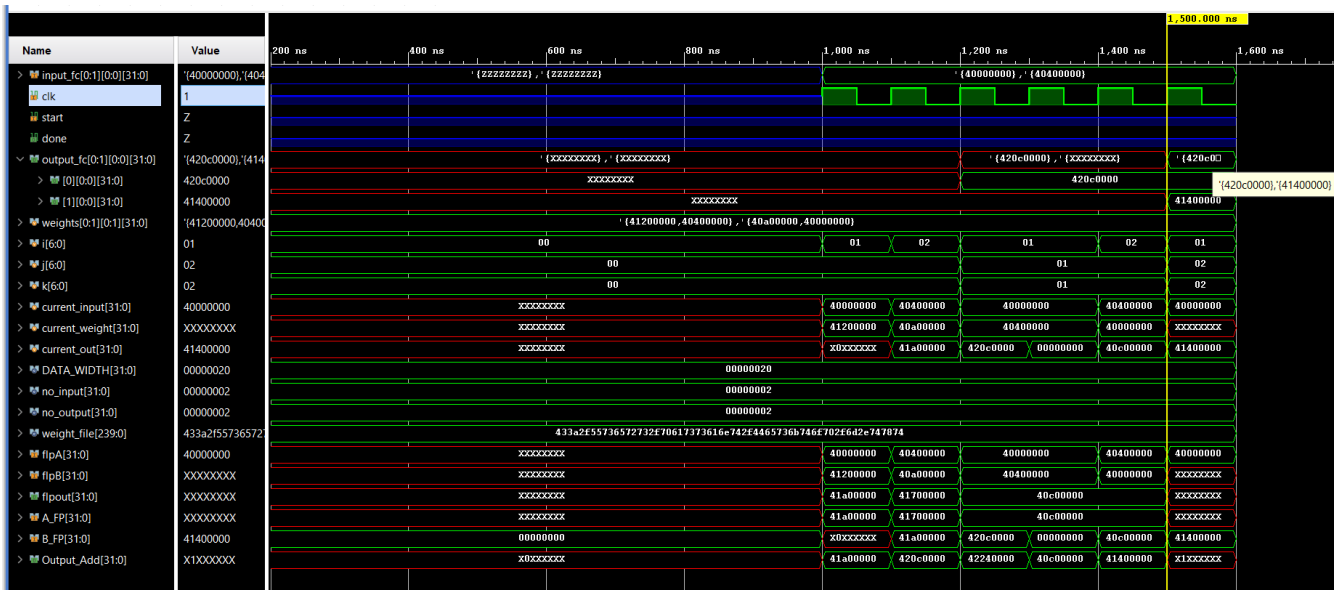
In this part we did some modifications in some modules to reduce the utilization since the convolution layers have a huge utilization percentage therefore we changed the FC Layer module and the PE module by making them work sequentially not in parallel. i.e. the FC layer before modification takes number of cycles = number of columns of the weights matrix (column multiplication), but after modifying the mentioned modules the number of cycles needed is equal to the number of weights =inputs x outputs.

Here is a simulation of the FC Layer module using :

Input values : $\begin{matrix} 2 \\ 3 \end{matrix}$

Weights : $\begin{matrix} 10 & 3 \\ 5 & 2 \end{matrix}$

therefore the outputs should be : {35 , 12} → {420C0000, 41400000}



- ✓ We also made 2 modules to store the contents of a text file in 3D or a 2D generic array.

C:/Users/passant/Downloads/synth/project_7/project_7.srcs/sources_1/new/read.sv

```
`timescale 1ns / 1ps
//This module stores a text file into a 2D array of dim  xdim
module read(out);
parameter file="C:/Users/passant/Desktop/m.txt",
dim1=2,dim2=2 ;
reg [31:0] mem [dim1][dim2];
output reg [31:0] out [dim1][dim2];
initial begin
    $readmemh(file,mem);
end

assign out =mem;


endmodule
```

The figure above shows the (read) module that stores into a 2D array

C:/Users/passant/Downloads/synth/project_7/project_7.srcs/sources_1/new/read3D.sv

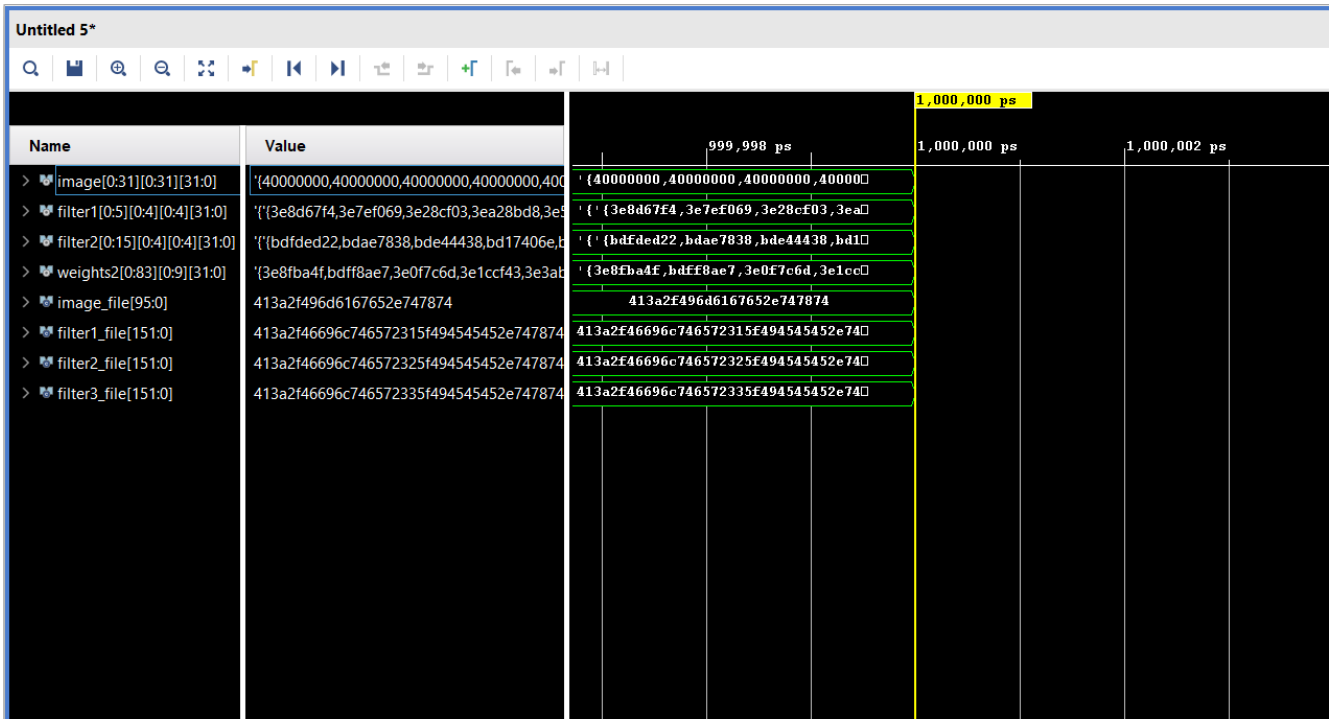
```


//This module stores a text file into a 3D array of dim3  xdim1 xdim2
module read3D(out);
parameter file="A:/Filter1_IEEE.txt",
dim1=2,dim2=2,dim3=6 ;

output reg [31:0] out [dim3] [dim1][dim2];
genvar i;

generate for(i=0;i<dim3; i=i+1)
begin

read #(file,dim1,dim2)
reading2D(out[i]);

end

endgenerate
```

The figure above shows the (read3D) module that stores into a 3D array using the (read) module

✓ Verifying that the read/read3D modules are working correctly:

in the figure below is the pixels of the input image stored in a 32x32 array.

Filter 1 stored in a 5x5x6 array, Filter 2 stored in a 5x5x16 array , Filter 3 stored in a 5x5x120 array, Weight file 1 stored in a 120x84x1 array (but since the number of bits exceeds the vivado simulation limit it didn't appear in the wave), and Weight file 2 stored in a 84x10x1 array



- **Utilization & Synthesis Reports**

➔ Reports for this part are attached in .txt files in “Synth. & Ut. Reports” Folder, named: “LeNet – Synth. txt” & “LeNet – Ut .txt”.

➔ Simulation of this part ran successfully (No errors that prevented running), but we faced some logical errors relating to the final output.