Max Pooling for a Convolutional Neural Network

Studenti: Leonardo Cecchelli, Riccardo Xefraj

1 INTRODUCTION

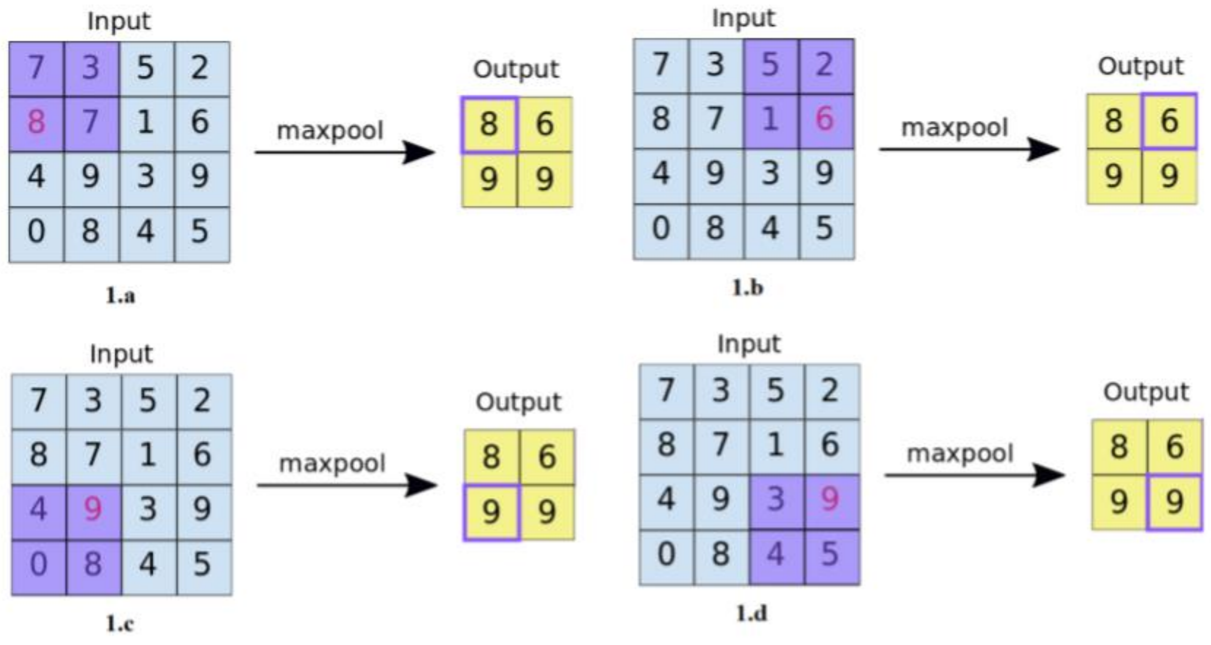
The aim of this project is to create a max pooling layer for a Convolutional Neural Network, a type of neural networks mainly used for voice and object recognition. In the specific the layer we are going to realize is going to reduce the dimensions of the matrixes used by a CNN, reducing so the amount of operations and the memory size needed by the neural network.

Let's now see a brief schematic of the max pooling layer:

- N input channels consisting of one 10x10 matrix each
- N output channels consisting of one 5x5 matrix each

At the beginning each input channel is sent to a max pooling filter that selects a 2x2 portion of the input matrix and extracts the maximum values between the 4 elements selected. This value is stored in the related output matrix and then the filter is moved left towards the next 2x2 subpart of the input matrix. Once the filter reaches the end of the matrix it is moved to the next row of the input matrix and the process is repeated until the entire 10x10 matrix has been scanned. At the end of the entire process the 5x5 output matrix will be filled with all the maximum values extracted from the five 2x2 sub-matrixes of the 10x10 input matrix.

Here we can see an example (taken from the project specification doc) of a max pooling filter applied to a 4x4 input matrix:

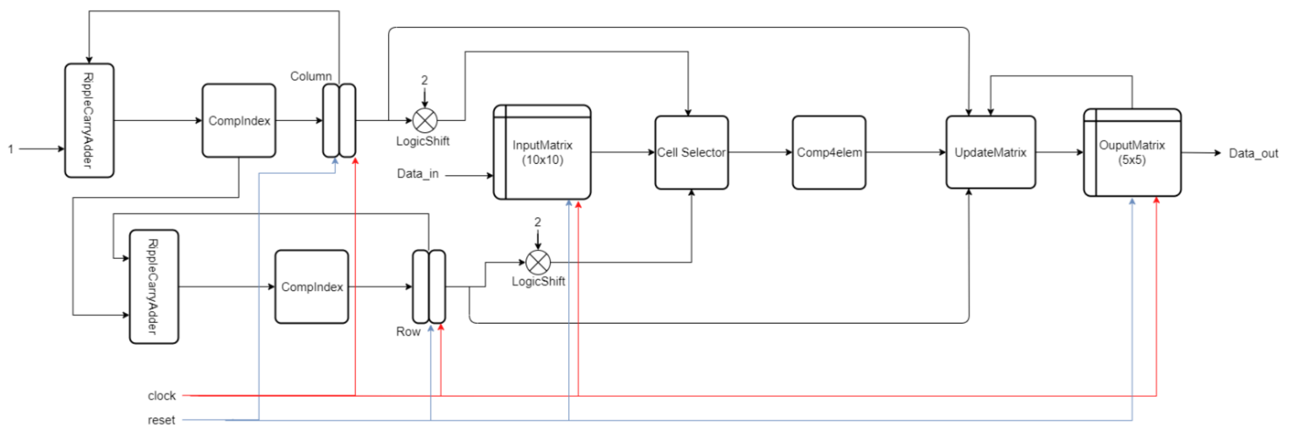


A project requirement is that each element of the output matrix has to be computed in one clock cycle. Therefore, in our case all the output matrixes will be completely filled after 25 clock cycles.

2 IMPLEMENTED ARCHITECTURE

In this paragraph we will have a look to how we designed the architecture of the hardware that is going to implement our max pooling layer.

Here is a simplified block diagram of the overall architecture:



We can identify 2 parts of the architecture: the part that manages the indexes of the matrixes (both input and output one) and the other part that applies the max pooling filter and update the output matrix.

We called the first part *RowColumnIndexManager* and is composed by the following blocks:

1. **RippleCarryAdder:** There are two instances of these blocks and they are used to increment sequentially the column and row indexes.
2. **CompIndex:** This is a comparator used to check if an index (both column and row) is still in the correct number range (from 0 to 4), in case an index value goes beyond 4 it is reset to 0. This way we can scan the entire 5x5 *OutputMatrix* without exceeding his dimension.
3. **LogicShift:** Since we want to scan the 10x10 *InputMatrix* by selecting a 2x2 blocks, we basically multiply by a factor of 2 each index. This way we are sure that in the *CellSelector* we are not going to select elements of previous 2x2 blocks. We called this part of the circuit *LogicShift* since the operation we described is a logic shift towards left.

The second part instead is composed by these blocks:

1. **Cell selector:** Takes as input the row and column indexes in order to select the correct 2x2 portion of the input matrix. Then it outputs an array that contains the four elements of the matrix.
2. **Comp4elem:** This part of the circuit takes the 4 elements from the *Cell Selector* and outputs the element with the maximum value.
3. **UpdateMatrix:** In this final block the value that comes from the *Comp4elem* block is stored in the correct position inside of the 5x5 *OutputMatrix* using the row and column indexes coming from the *RowColumnIndexManager*.

3 TEST-PLAN

There are three possible scenarios that should be tested for one input matrix:

- Matrix with one block with all positive numbers (>= 0)
- Matrix with one block with all negative numbers (< 0)
- Matrix with one block with mixed positive and negative numbers

We have also to check the correctness of our circuit given as input 2 or more matrixes.

Input and Expected Output

On the left we have our input matrixes on the right the expected solution of our circuit

	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	10
2	2	3	4	5	6	7	8	9	10	11
3	3	4	5	6	7	8	9	10	11	12
4	4	5	6	7	8	9	10	11	12	13
5	5	6	7	8	9	10	11	12	13	14
6	6	7	8	9	10	11	12	13	14	15
7	7	8	9	10	11	12	13	14	15	16
8	8	9	10	11	12	13	14	15	16	17
9	9	10	11	12	13	14	15	16	17	18

2	4	6	8	10
4	6	8	10	12
6	8	10	12	14
8	10	12	14	16
10	12	14	16	18

	0	1	2	3	4	5	6	7	8	9
0	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14
1	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13
2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12
3	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11
4	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10
5	0	-1	-2	-3	-4	-5	-6	-7	-8	-9
6	1	0	-1	-2	-3	-4	-5	-6	-7	-8
7	2	1	0	-1	-2	-3	-4	-5	-6	-7
8	3	2	1	0	-1	-2	-3	-4	-5	-6
9	4	3	2	1	0	-1	-2	-3	-4	-5

-4	-6	-8	-10	-12
-2	-4	-6	-8	-10
0	-2	-4	-6	-8
2	0	-2	-4	-6
4	2	0	-2	-4

We can see that at least one matrix respects all the previous illustrated scenarios. Each colour represents the cell that should be calculated every clock cycle according to the given documentation.

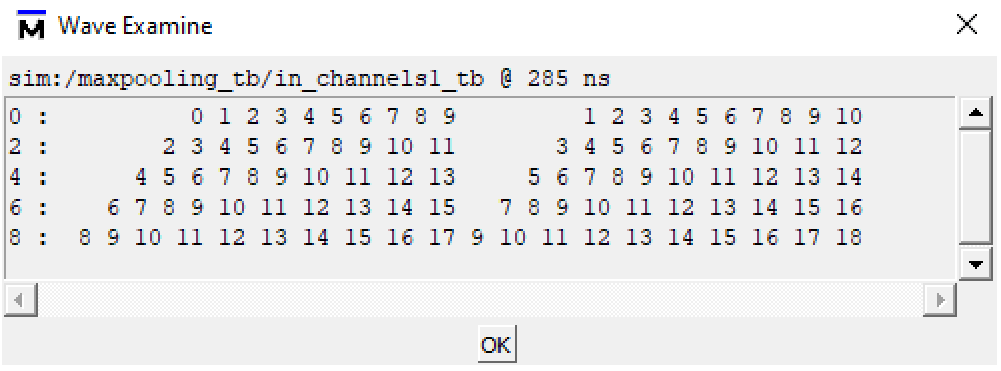
Java function to verify the correctness of our results

DimPool at the variable should be assigned 2 according to our design

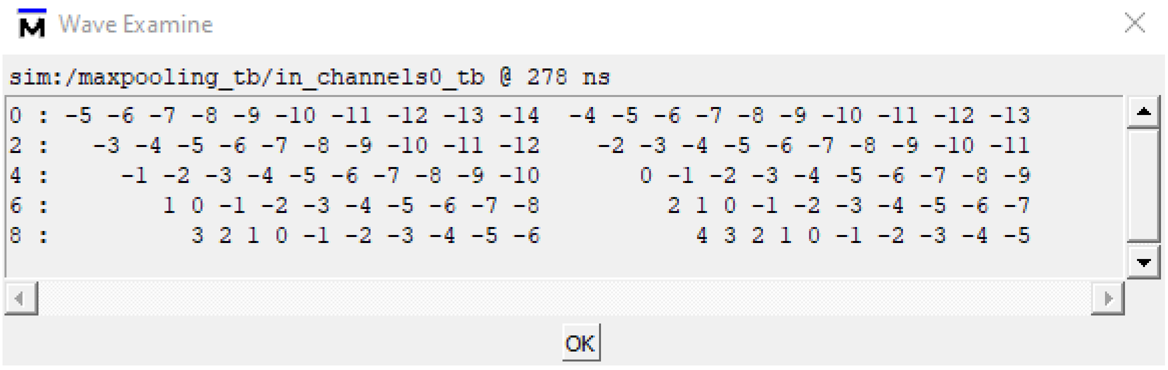
```java
    public static int[][] maxPooling(int[][] Matrice, int DimPool)
    {

        int[] ValoriDaConfrontare=new int[DimPool*DimPool];
        int rowSize=Matrice.length;
        int columnSize=Matrice[0].length;
        int[][] Pool=new int[rowSize/DimPool][columnSize/DimPool];
        int indiceValori=0;
        for(int i=0;i<rowSize;i+=DimPool)
            {
                for(int j=0;j<columnSize;j+=DimPool)
                {
                    for(int sottoRiga=i;sottoRiga<i+DimPool;sottoRiga+=1)
                    {
                         for(int sottoColonna=j;sottoColonna<j+DimPool;sottoColonna+=1)
                         {

                             ValoriDaConfrontare[indiceValori]=Matrice[sottoRiga][sottoColonna];
                             indiceValori++;
                         }
                    }
                    int max=ottieniMassimo(ValoriDaConfrontare);
                    Pool[i/DimPool][j/DimPool]=max;
                    indiceValori=0;
                }
            }
       return Pool;
    }
```

4 TEST BENCH

Input Matrixes

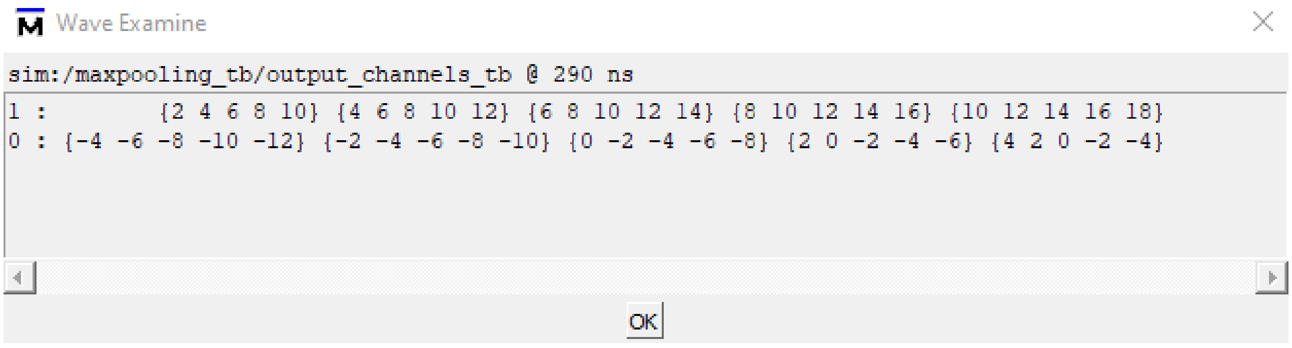


```
sim:/maxpooling_tb/in_channels1_tb @ 285 ns
0 :         0 1 2 3 4 5 6 7 8 9        1 2 3 4 5 6 7 8 9 10
2 :       2 3 4 5 6 7 8 9 10 11      3 4 5 6 7 8 9 10 11 12
4 :     4 5 6 7 8 9 10 11 12 13    5 6 7 8 9 10 11 12 13 14
6 :   6 7 8 9 10 11 12 13 14 15  7 8 9 10 11 12 13 14 15 16
8 :  8 9 10 11 12 13 14 15 16 17 9 10 11 12 13 14 15 16 17 18
```



```
sim:/maxpooling_tb/in_channels0_tb @ 278 ns
0 : -5 -6 -7 -8 -9 -10 -11 -12 -13 -14  -4 -5 -6 -7 -8 -9 -10 -11 -12 -13
2 :   -3 -4 -5 -6 -7 -8 -9 -10 -11 -12    -2 -3 -4 -5 -6 -7 -8 -9 -10 -11
4 :     -1 -2 -3 -4 -5 -6 -7 -8 -9 -10       0 -1 -2 -3 -4 -5 -6 -7 -8 -9
6 :        1 0 -1 -2 -3 -4 -5 -6 -7 -8         2 1 0 -1 -2 -3 -4 -5 -6 -7
8 :          3 2 1 0 -1 -2 -3 -4 -5 -6           4 3 2 1 0 -1 -2 -3 -4 -5
```

Output Matrixes

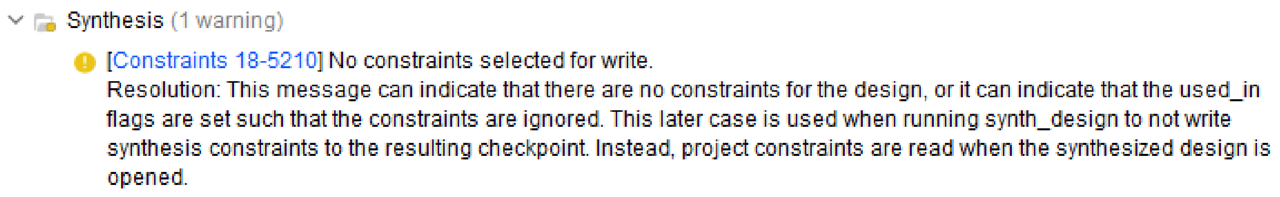


```
sim:/maxpooling_tb/output_channels_tb @ 290 ns
1 :        {2 4 6 8 10} {4 6 8 10 12} {6 8 10 12 14} {8 10 12 14 16} {10 12 14 16 18}
0 : {-4 -6 -8 -10 -12} {-2 -4 -6 -8 -10} {0 -2 -4 -6 -8} {2 0 -2 -4 -6} {4 2 0 -2 -4}
```

This testbench works as we expected from the Test-Plan

5 VIVADO IMPLEMENTATION

5.1 WARNINGS MESSAGES



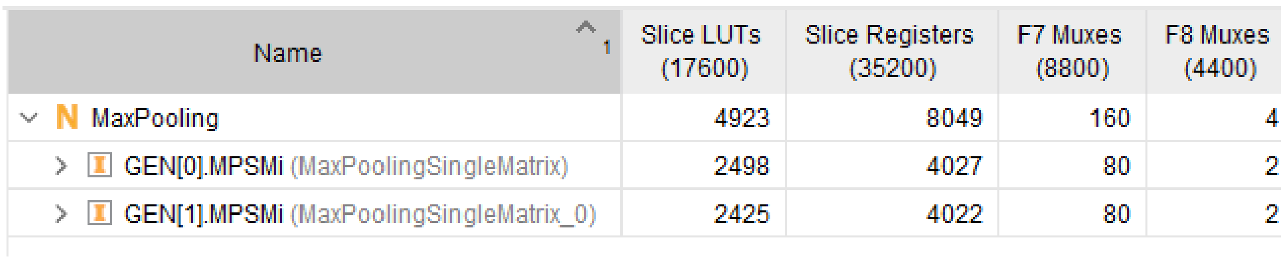
Synthesis (1 warning)

[Constraints 18-5210] No constraints selected for write.
Resolution: This message can indicate that there are no constraints for the design, or it can indicate that the used_in flags are set such that the constraints are ignored. This later case is used when running synth_design to not write synthesis constraints to the resulting checkpoint. Instead, project constraints are read when the synthesized design is opened.

This warning message appears because we have not specified any write constraints in our design

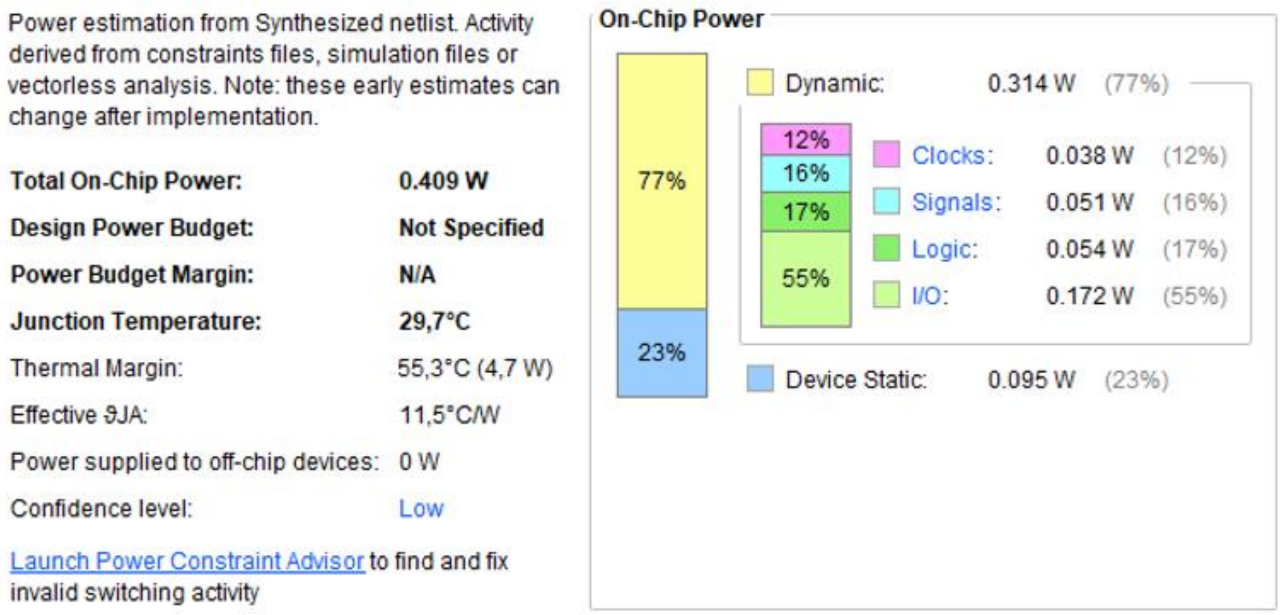
5.2 Device Utilization

Since our design has an excessive number of input and output pins (10*10*32 for every input matrix) the synthesis has been done “out of context” giving use this utilization



Name	Slice LUTs (17600)	Slice Registers (35200)	F7 Muxes (8800)	F8 Muxes (4400)
MaxPooling	4923	8049	160	4
GEN[0].MPSMi (MaxPoolingSingleMatrix)	2498	4027	80	2
GEN[1].MPSMi (MaxPoolingSingleMatrix_0)	2425	4022	80	2

Power consumption:



Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	**0.409 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**29,7°C**
Thermal Margin:	55,3°C (4,7 W)
Effective ϑJA:	11,5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

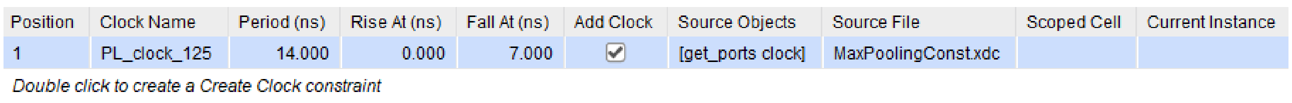
Dynamic:	0.314 W	(77%)
Clocks:	0.038 W	(12%)
Signals:	0.051 W	(16%)
Logic:	0.054 W	(17%)
I/O:	0.172 W	(55%)
Device Static:	0.095 W	(23%)

5.3 Clock Frequency information

We have chosen a clock frequency, so we don’t have a T set up violation according to this equation:

TCLK≥TSUP+ TP-logic+ TC-Q

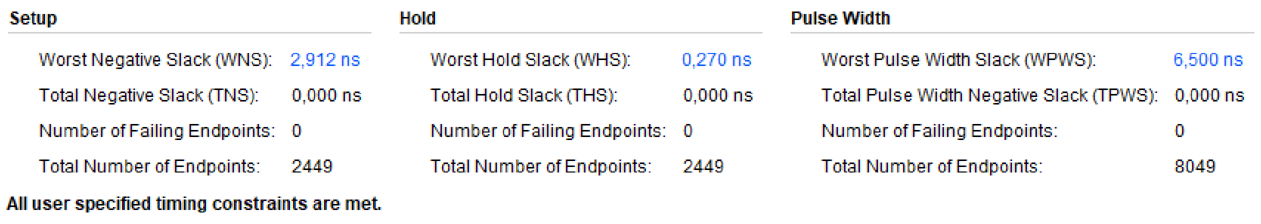
We also gave a little margin to the clock frequency to ensure that our design will result implementable (the synthesis doesn’t take note about the cables latencies).



Position	Clock Name	Period (ns)	Rise At (ns)	Fall At (ns)	Add Clock	Source Objects	Source File	Scoped Cell	Current Instance
1	PL_clock_125	14.000	0.000	7.000	✓	[get_ports clock]	MaxPoolingConst.xdc		

Double click to create a Create Clock constraint

So, we obtained the following results:



Setup

Worst Negative Slack (WNS):	2,912 ns
Total Negative Slack (TNS):	0,000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	2449

Hold

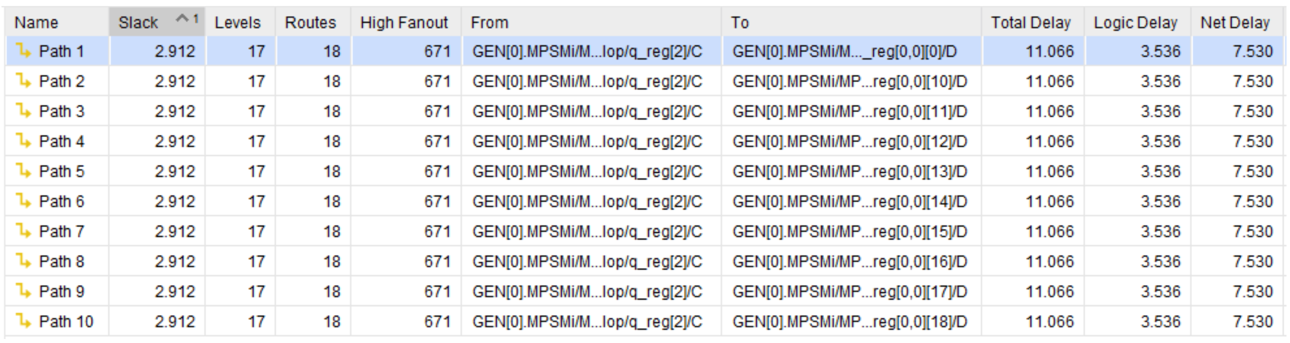
Worst Hold Slack (WHS):	0,270 ns
Total Hold Slack (THS):	0,000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	2449

Pulse Width

Worst Pulse Width Slack (WPWS):	6,500 ns
Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	8049

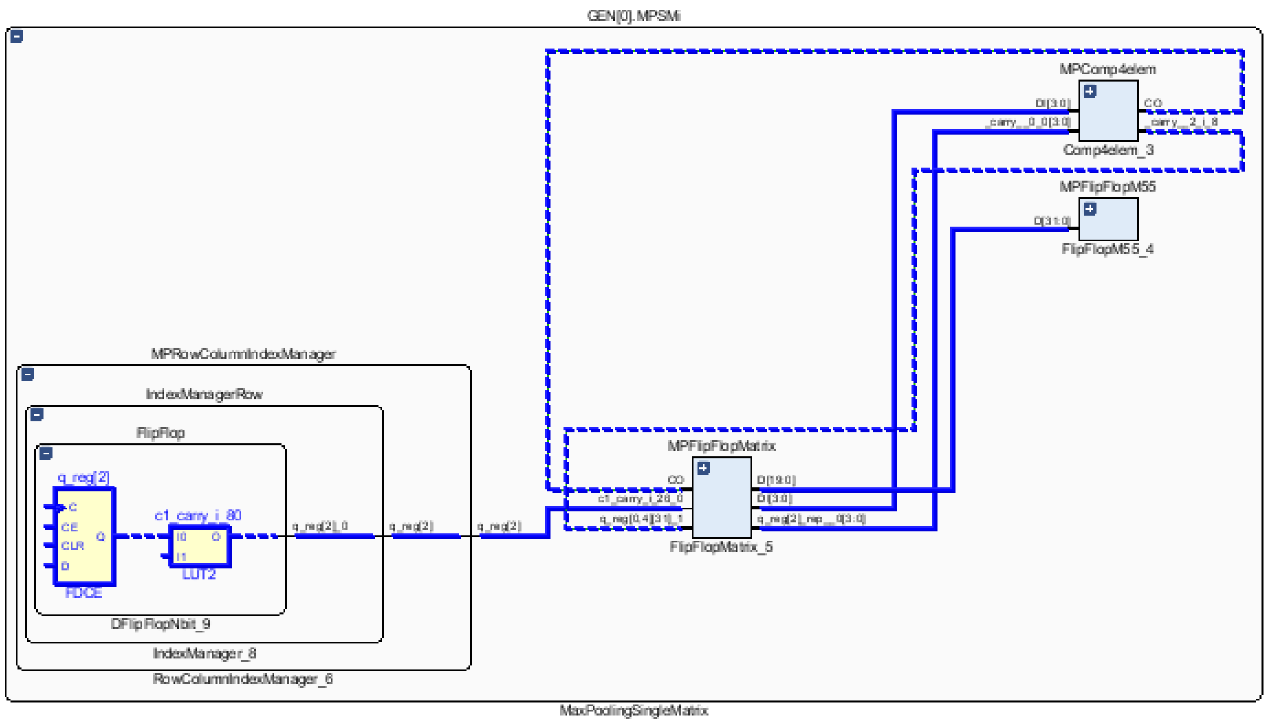
All user specified timing constraints are met.

Critical paths:



Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay
Path 1	2.912	17	18	671	GEN[0].MPSMi/M...lop/q_reg[2]/C	GEN[0].MPSMi/M..._reg[0,0][0]/D	11.066	3.536	7.530
Path 2	2.912	17	18	671	GEN[0].MPSMi/M...lop/q_reg[2]/C	GEN[0].MPSMi/MP...reg[0,0][10]/D	11.066	3.536	7.530
Path 3	2.912	17	18	671	GEN[0].MPSMi/M...lop/q_reg[2]/C	GEN[0].MPSMi/MP...reg[0,0][11]/D	11.066	3.536	7.530
Path 4	2.912	17	18	671	GEN[0].MPSMi/M...lop/q_reg[2]/C	GEN[0].MPSMi/MP...reg[0,0][12]/D	11.066	3.536	7.530
Path 5	2.912	17	18	671	GEN[0].MPSMi/M...lop/q_reg[2]/C	GEN[0].MPSMi/MP...reg[0,0][13]/D	11.066	3.536	7.530
Path 6	2.912	17	18	671	GEN[0].MPSMi/M...lop/q_reg[2]/C	GEN[0].MPSMi/MP...reg[0,0][14]/D	11.066	3.536	7.530
Path 7	2.912	17	18	671	GEN[0].MPSMi/M...lop/q_reg[2]/C	GEN[0].MPSMi/MP...reg[0,0][15]/D	11.066	3.536	7.530
Path 8	2.912	17	18	671	GEN[0].MPSMi/M...lop/q_reg[2]/C	GEN[0].MPSMi/MP...reg[0,0][16]/D	11.066	3.536	7.530
Path 9	2.912	17	18	671	GEN[0].MPSMi/M...lop/q_reg[2]/C	GEN[0].MPSMi/MP...reg[0,0][17]/D	11.066	3.536	7.530
Path 10	2.912	17	18	671	GEN[0].MPSMi/M...lop/q_reg[2]/C	GEN[0].MPSMi/MP...reg[0,0][18]/D	11.066	3.536	7.530

Schematic representation of critical path:



6 Conclusion

6.1 Final result

As we have seen from the previous paragraph the biggest concern for us was finding a suitable clock value. During the synthesis process we came up with a higher value of the frequency (100 MHz) but once we went through the implementation phase, we discovered that the Slack was negative. This was due to the fact that our circuit has a lot of wires (since we are dealing with matrixes) that introduce a lot of delay, something that is not considered during the synthesis process. Therefore, we downgraded the clock to around 70 MHz in order to achieve a positive slack and have no setup time violations.

For what concern power consumption, we can see that the I/O block is the one with the higher consumption rate, which was expected. A thing to note is that the data extracted from the Vivado Tool are related to a max pooling layer with just 2 channels, so two 10x10 matrixes as inputs and two 5x5 matrixes as outputs. If we want to synthetize a max pool layer with more than 2 channels the only thing to do is change the parameter *Nchannels* inside of the vhd file called *MaxPooling.vhd*. Of course, by doing that, the power consumption data will be very different since we are increasing or decreasing the number of inputs and outputs to drive.

6.2 Possible applications

As we said in the introduction, a max pooling layer is a part of Convolutional Neural Networks, whose main applications are mainly related to image recognition. Although there are also other important usage of this type of neural networks such as:

1. **Video analysis:** Video is more complex than images since it has another (temporal) dimension. One approach is to treat space and time as equivalent dimensions of the input and perform convolutions in both time and space. In this case a max pooling layer is very useful to reduce the amount of data that needs to be processed.
2. **Drug discovery and health risk assessment:** CNNs have been used in drug discovery. Predicting the interaction between molecules and biological proteins can identify potential treatments. More over CNNs can be naturally tailored to analyse large collection of time series data representing one week of human physical activity streams together the clinical data.
3. **Language processing:** CNNs have been very useful in semantic parsing and query retrieval, two fundamental operations when processing natural language speech.