Energy Efficient Implementation of Convolutional Neural Network on FPGA

EITF35: Introduction to structured VLSI design

Fabian Eklund Sigleifs
Cina Arjmand

Lunds Tekniska Högskola
Sweden
November 2022

Contents

1 Introduction

The purpose of this project is to develop a convolutional neural network hardware implementation capable of classifying rectangles and circles from a set of input images containing one of these shapes. Hardware descriptions written in VHDL are implemented in Xilinx Vivado with efficiency considerations in terms of both timing and power. The design consists of three distinctive layers: convolutional, rectified linear unit (ReLu) and maxpooling, which are performed in said order to obtain the final result. The design was implemented based on a pre-trained high-level network architecture with weights trained on a set of one thousand 64x64 bit images.

2 Convolutional neural networks

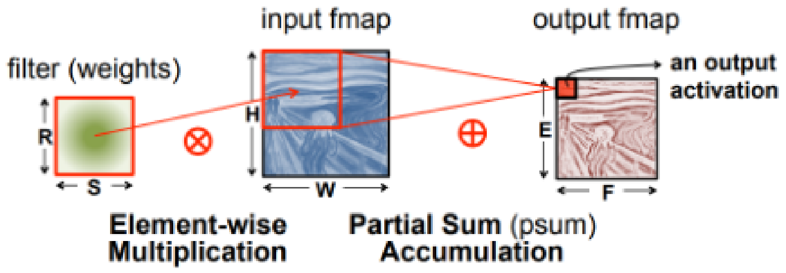


Figure 1: Overview of image convolution.

The principle of convolutional neural networks is to perform feature detection on input images by means of convolution operations using a set of filters, each filter normally corresponding to a given feature to be detected [1]. In discretized form, the convolution operation corresponds to an elementwise multiplication of a filter matrix with the matrix form of the input image (in the case of this lab, the input image is a single-channel binary matrix). If the filter matrix is smaller than the input image, the filter is traversed repeatedly across each point in the image with a given stride length to produce a 2-dimensional cross-correlation of the matrices.

3 Design specifications

The design specifications for this project is to design a convolutional network to detect if a 64x64 bit image displays a circle or a rectangle. The overview of the design requirements can be seen in figure 2. The FPGA to be used is a Nexys 4 Artix-7 and the power consumption of the implementation should not exceed 1 W in total. The clock frequency of the board is 100 MHz which requires the design to meet this timing constraint. As seen in the figure 2 the output of the ReLu function is a 61x61 image which is passed into the pooling layer and subsequently converted into a 4x4 output matrix. The 4x4 output matrix of the maxpooling layer is then passed through a fully connected layer, another ReLu layer, a second fully connected layer, and finally a sigmoid activation function to produce the binary classification output corresponding to circle (0) or square (1).

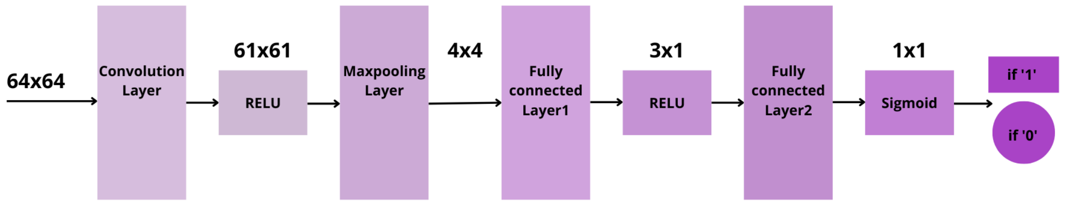


Figure 2: Design specifications of the convolutional neural network.

4 Hardware implementation

In the following section the hardware implementation is explained in detail. The convolutional, ReLu, maxpooling and fully connected layers are discussed in the order in which the data passes through them in the design.

4.1 Convolutional layer

The convolutional layer performs bit-wise multiplication of input image data read from 4 separate RAM cells and the 4x4 weight matrix. The data is selected from the RAM so that it comes in the correct order for a 4x4 sliding window with stride equal to one and zero padding. The hardware implemented a pipeline structure to enable four convolutions to be calculated simultaneously, thus further helping to achieve the design requirements of timing and power. The four RAMs in figure 3 are initialized so that the first RAM contains addresses 1,5,9,13... and the second contains addresses 2,6,10,14 and so on. This particular arrangement of the image data is performed beforehand in order to enable pipelined processing in an efficient manner. After the multiplication has been performed the values are stored in registers until a total of 16 values has been collected in each register; at this time the valid signal goes high and the correct sum can be read on the SUM1-4 outputs. These four sums correspond to four distinct filter convolutions, and four points of the 61x61 output image. This dataflow is summarized in the schematic in figure 4.

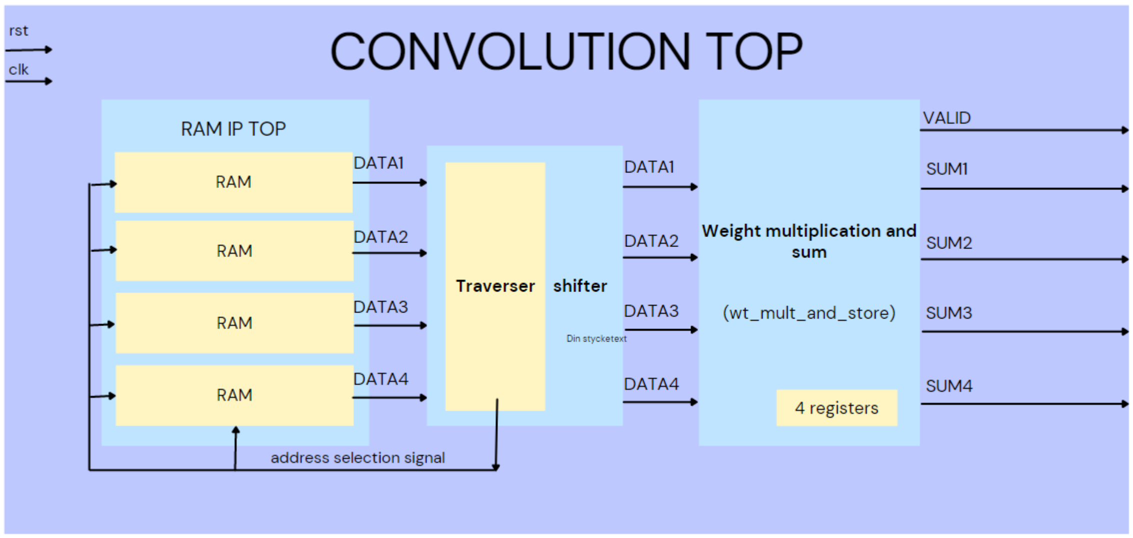


Figure 3: Diagram showing the design blocks for convolution top

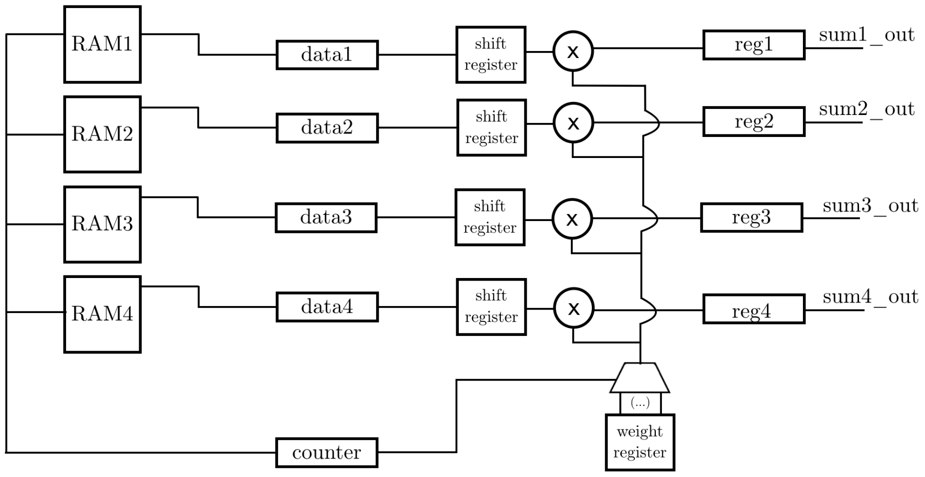


Figure 4: Schematic of convolutional layer implementation.

4.2 ReLu layer

In the ReLu layer the result from the convolution layer is passed through the function displayed in figure 6. As can be seen in the figure, all negative input values are set to zero and the remaining values are passed through unmodified. A block diagram of the ReLu hardware implementation is included in figure 7.

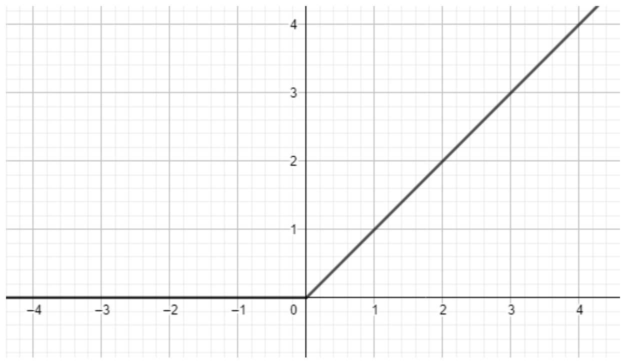


Figure 5: The ReLu activation function.

4.3 Maxpooling layer

In the maxpooling layer a 15x15 maxpooling filter is applied to the output of the ReLu, which is a 61x61 matrix. Since the 15x15 filter is not evenly divisible across the 61x61 output, the last row and column of the convolution output is excluded from the maxpooling operation. The basic functionality of maxpooling is shown in figure 6. For every 15x15 window of values, the maximum among these is passed as an output point. Since 16 of these filter windows can fit inside the 60x60 matrix, the output of this layer will be a 4x4 matrix. The maxpooling filter is used to downsample the result from the convolution and ReLu, thus making the system less susceptible to noise or faulty indata. For efficiency, the maxpooled value for each individual 15x15 square of the convolution output image is calculated as soon as that square has been completed, using the implementation in figure 7. For each calculated sum, only the maximum sum value belonging to a given square is stored, while the rest are discarded.

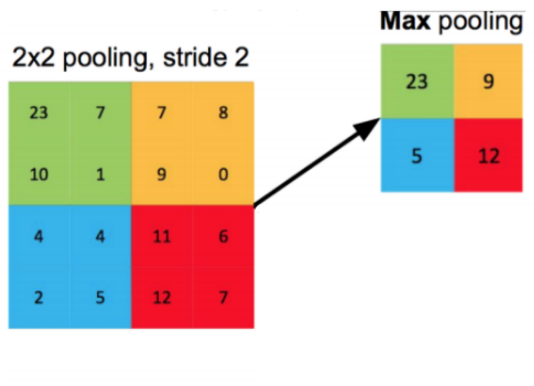


Figure 6: Maxpooling example for a 4x4 matrix.

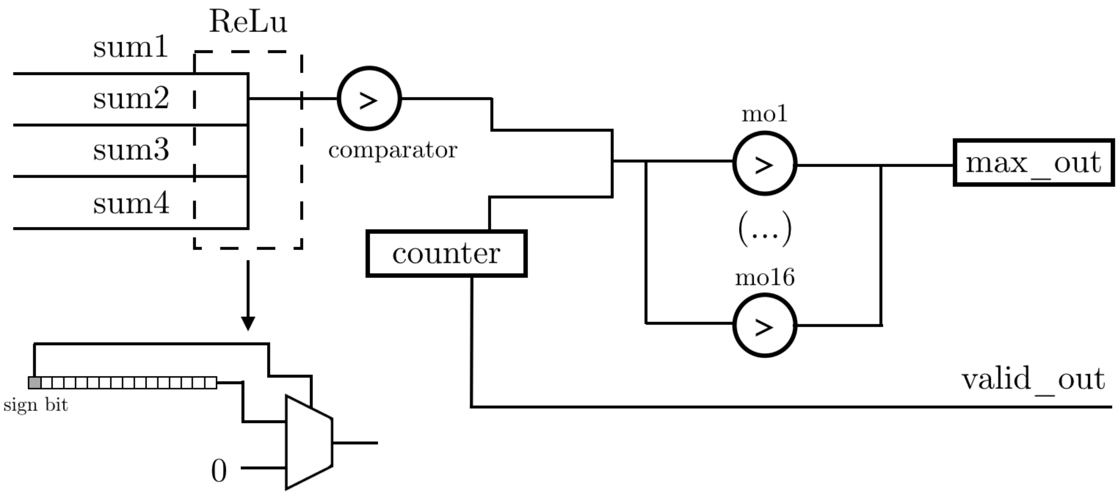


Figure 7: Schematic of maxpooling layer implementation. The registers mo1-16 store the maximum value for each square of the 4x4 output.

4.4 Fully connected layer

The signal from the maxpool layer then passes into the fully connected layer. Here the signal is multiplied with three different 4x4 matrices named wt1, wt2 and wt3, see figure 8. After the multiplication with the weights has been performed, the signal is passed to another ReLu function similar to the one described in section 4.2.

Afterwards, the 3x1 signal from the ReLu is multiplied with a 1x3 weight vector. The elements of this output vector from this multiplication are then summed together with a bias value and passed to a comparator which checks if the value is smaller or larger than zero. If the value is less than zero, a logic zero is passed as output indicating that a circle has been detected; otherwise, it will output a one to indicate that a rectangle has been detected.

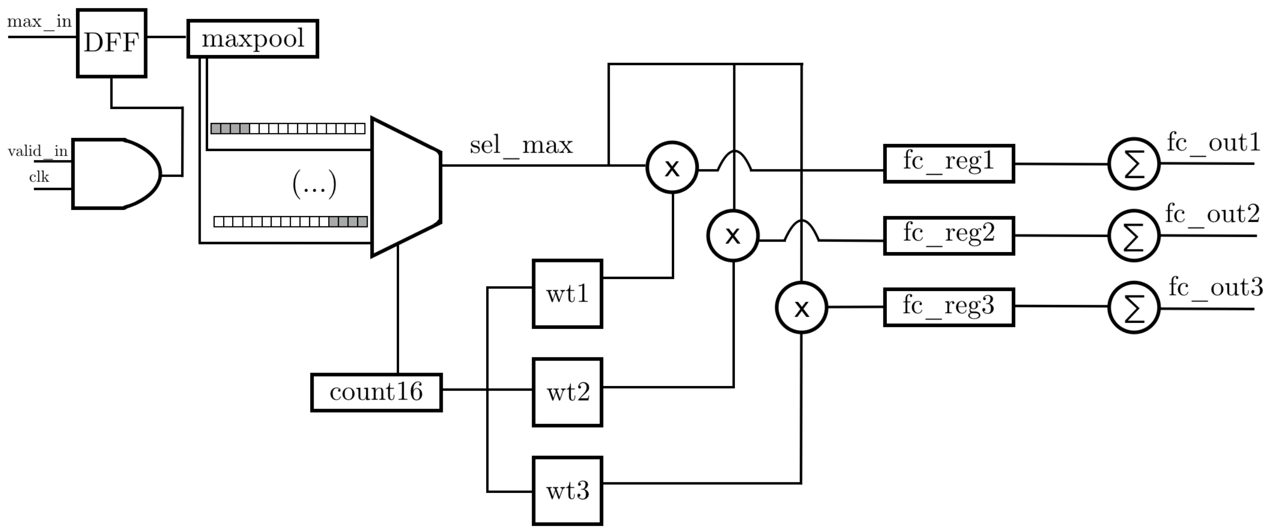


Figure 8: The weight multiplication in the connected layer.

5 Results

The following section summarizes the results of timing, power and resource utilization reports on post-synthesis simulations of the design performed in Xilinx Vivado.

5.1 Timing

The number of passed clock cycles until the various functions are finished can be seen in Table 1. Since a 100 Mhz clock is used, this results in 15 382 and 23 062 clock cycles respectively until the two maxpooling and fully connected processing layers finish. The number of pictures that can be calculated per second will be equal to 4 336.

Function	**Time in ns until function is finished**
Convolution, ReLu and maxpooling	153 815
Fully connected and sigmoid	230 615

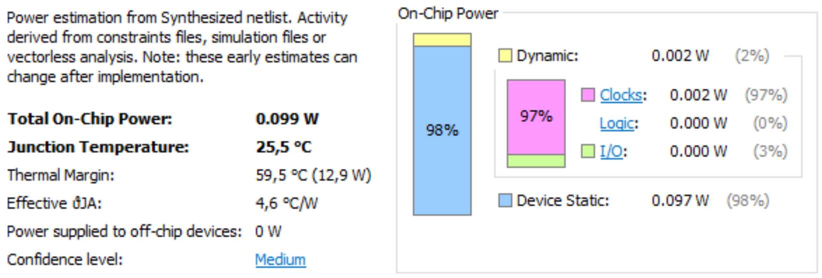
Table 1: Number of clock cycles until different functions finish.

5.2 Power and Utilization

The overall FPGA utilization can be seen in Table 2. From Figure 9 it can also be noted that the power usage is 2 mW for the dynamic power and 97 mW for the static.

Resource	**Utilization**	**Available**	**Utilization %**
LUT	256	63 400	0.40
LUTRAM	256	19 000	1.35
FF	256	126800	0.20
IO	2	210	0.95

Table 2: The utilization report for the synthesis in Vivado.


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:	**0.099 W**
Junction Temperature:	**25,5 °C**
Thermal Margin:	59,5 °C (12,9 W)
Effective ϑJA:	4,6 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

On-Chip Power

Dynamic:	0.002 W	(2%)
Clocks:	0.002 W	(97%)
Logic:	0.000 W	(0%)
I/O:	0.000 W	(3%)
Device Static:	0.097 W	(98%)

Figure 9: Post-synthesis power estimation of the design.

6 Discussion

The final design has relatively low resource utilization and only consumes 2 mW of dynamic power. Timing requirements enable a very high processing rate of over 4 000 images per second. Considering these results, the neural network performs its task with high efficiency, even for what is a very simple image processing task. More complicated tasks, such as larger images, greater number of features, and more detailed features (requiring larger convolution filters), would require the a larger scale of design and naturally a greater power and resource utilization. Nevertheless, by utilizing principles of hardware efficient design and pipelining, the specified network has been succesfully realized in a resource- and energy efficient implementation.

References

[1] Ohlsson, Mattias; Edén, Patrik. 2021. *Introduction to Artificial Neural Networks and Deep Learning*. Lund University Media-Tryck.