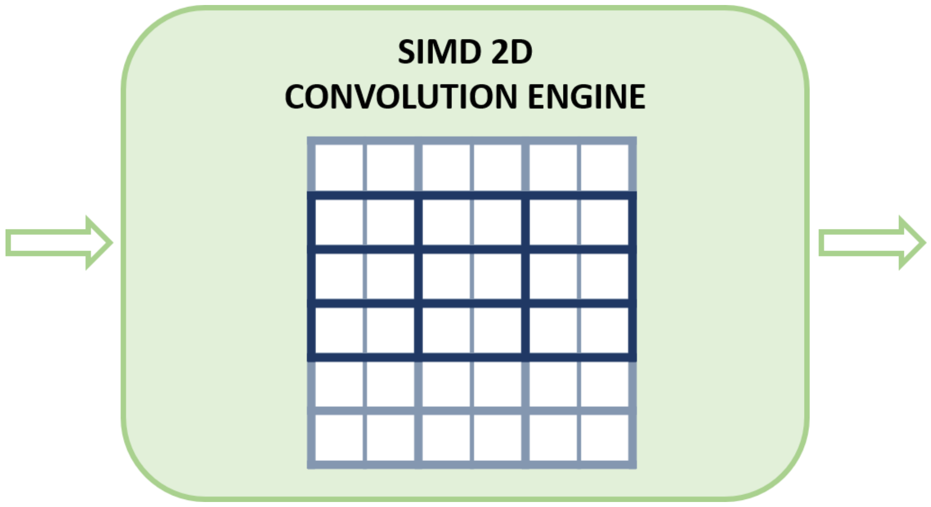
Department of Informatics, Modeling,
Electronics and System Engineering

Efficient SIMD 2D convolution engine for FPGA-based heterogeneous embedded systems



Students: Andrea Migali, Roman Huzyuk, Mario Andrea Sangiovanni

Supervisor: Prof. Stefania Perri

Team Number: xohw20_244

YouTube video link: https://youtu.be/oM3hgEmobe8

Summary

1. INTRODUCTION

The designed system is an Efficient Convolution Engine able to compute 2-D filterings in the space domain taking advantage of the SIMD paradigm. It also offers the possibility to configure, with an appropriate software, the values of the convolution kernel. The IP-Core has been described in VHDL and implemented within an embedded system built on a Nexys4-DDR board. The advantages introduced by the use of this system are an efficient exploitation of the hardware resources available inside the chip and a reduced power consumption.

The calculation operations are performed using a 3x3 kernel, while the input matrices dimensions can be configured during the design phase. Each single input data to be processed is a 16bit binary word: it can be interpreted as a unique word or as two consecutive 8 bit words.

Figure 1 shows the operation that needs to be done in order to obtain a generic filtered data.

p(0,0)	*p(0,1)*	…	…	…
p(1,0)				
p(2,0)				
…				
…				

a)

w(0,0)	*w(0,1)*	*w(0,2)*
w(1,0)	*w(1,1)*	*w(1,2)*
w(2,0)	*w(2,1)*	*w(2,2)*

b)

c)

$$q_{(i,j)} = \sum_{h=-1}^{1} \sum_{k=-1}^{1} p_{(k+i,h+j)} \times w_{(k+1,h+1)}$$

Figure 1: a) input image; b) kernel weights; c) filtered pixel

The proposed IP-Core, thanks to its features, can be adapted to various application areas where convolution operations are required. In particular, one of the main applications that could benefit from a system like this is that of Convolutional Neural Networks (CNNs) which are widely used in the realization of AI systems. In order to allow a wide development of this technology, it is essential to use techniques of energy consumption and hardware cost optimization without sacrificing performances. In this perspective, hardware accelerators realized on state-of-the-art FPGA chips are particularly relevant [1].

The presented design combines the benefits of using an FPGA with the versatility and resources optimization offered by the SIMD paradigm.

CNNs are based on the computation of multiple cascaded filterings: this may require the presence of a different type of calculation circuit for each convolution layer. The SIMD logic allows to overcome this limit by adapting the same circuit structure to convolution operations with different filtering parameters. This is also made possible by the fact that the kernel values can be modified through software in a very short time.

The realized SIMD convolution engine complies with the AXI4 protocol and therefore it can be easily integrated as a custom accelerator IP core within heterogeneous embedded systems.

2. DESIGN

The purpose of this chapter is to describe the hardware structure of the designed embedded system. In this section, the functionalities of each module will be presented.

2.1 Hardware

The system is made up of five components, as shown in figure 2:

- **SIMD Convolution Engine 3x3 IPCore:** is the designed IP-Core which executes convolution computations.
- **MicroBlaze Soft-Processor**: is the system processor which runs the configuration software.
- **AXI DMA:** Performs direct reading/writing accesses to the memory.
- **Memory Interface Generator:** Manages the data transactions between the SoC and the external memory.
- **External DDR memory:** external memory.

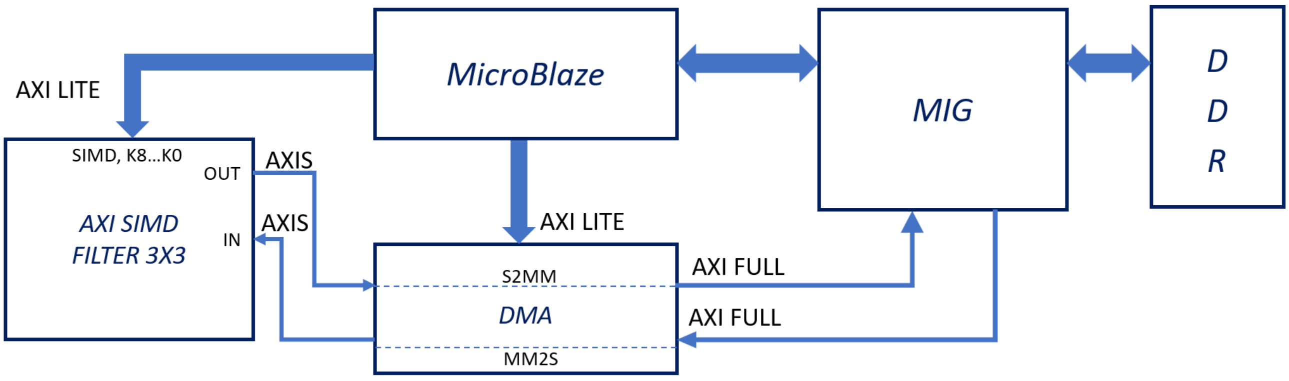


Figure 2 Embedded System

2.1.1 SIMD Convolution Engine 3x3 IP-Core

The designed IP-Core performs a space-domain filtering by operating 3x3 2D convolution using a fully reconfigurable Kernel and exploiting SIMD logic. More specifically, the IP-Core is able to manage 16-bit as well as 8-bit pixels input images. The module is designed with a parametric VHDL code. The two main parameters allow the user to specify input 2D array dimensions.

In order to implement the SIMD logic in a highly efficient way, each part of the circuit has been specifically designed for the purpose. (example: SIMD Multiplier, SIMD Adder).

The general structure of the convolution engine consists of several functional modules, as illustrated in the figure below:

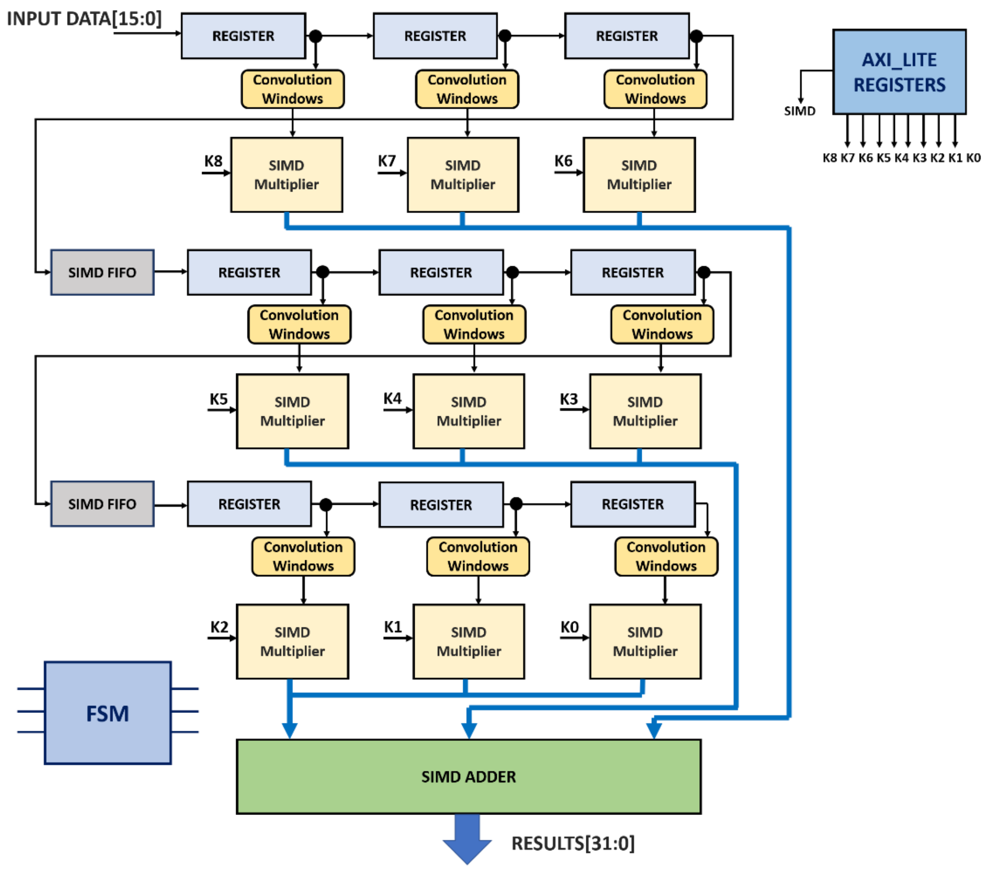


Figure 3 IP-Core internal block diagram

The filter receives one 16-bit input word per clock cycle and it flows inside the buffer, which is able to properly manage the two different SIMD cases. According to the specific configuration, the input data is considered as a single 16-bit word or as two parallel 8-bit words. The 3x3 registers array, along with an auxiliary circuit, builds up the convolution window to be processed. Each pixel is then multiplied by the corresponding kernel weight using a dedicated SIMD multiplier specifically designed for this purpose. The nine obtained products are finally summed by a SIMD adder. The result of this operation is a 32-bit word which can be interpreted as a single data or as two parallel 16-bit words. The behaviour of the whole circuit is supervised by a Finite State Machine.

SIMD Buffer

The SIMD buffer is the first functional module of the IP-Core and its task is to properly reorganize the input pixels in order to execute the correct operations on each of them. This section is necessary since the filter receives an input stream of raster-ordered pixels.

The 16-bit words flow in a structure made up of nine 16-bit registers and two SIMD FIFOs, which is illustrated in figure 4. The output of each register is connected to a multiplexers grid which creates the correct convolution windows to be provided to the multipliers. This circuit is needed to reorder the 8bit sub-words inside each register which would be wrong otherwise. Another task assigned to the same circuit is the zero-padding management.

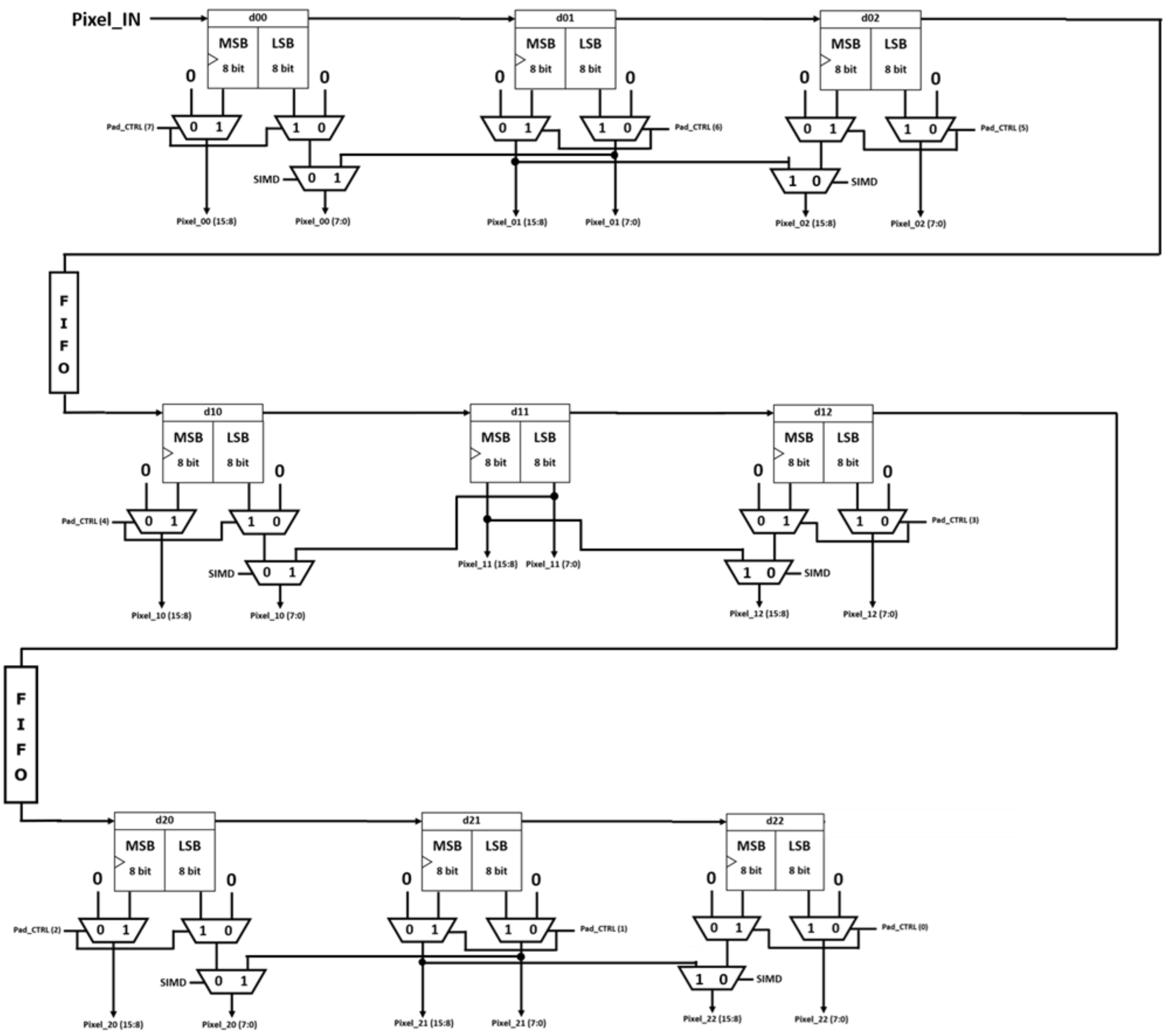


Figure 4 SIMD buffer Structure with registers and SIMD FIFOs

SIMD Binary Multiplier

The Binary Multiplier has the task of computing the product between one input word and the proper weight of the convolution kernel. This circuit has been designed and optimized to implement the SIMD logic properly exploiting the fact that the kernel is the same in both SIMD cases. The weights of the kernel are 2’s complement words while the input pixels are unsigned.

The product is performed using AND logic gates and appropriate bit-shifting operations. The obtained partial products are then summed using an adder tree implemented with fast carry chains.

SIMD Adder

The final adder receives nine convolution products and computes the value of the filtered pixel. The structure is realized using fast carry chain along with a certain number of multiplexers needed to distinguish the two SIMD operations.

Finite State Machine

The finite state machine manages the input and output AXI4 pixels streams, controls the zero padding signals and moves other signals to guarantee the proper functionality of the system.

Configuration Interface

The reconfigurable values of the kernel and the selection of the SIMD behaviour are implemented using slave registers which are accessible through an AXI4-Lite write interface. This is shown in the following figure:

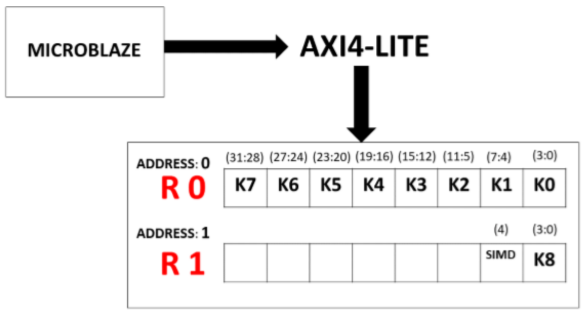


Figure 5 AXI4_lite write Interface for kernel and SIMD configuration

2.1.2 DMA

The Xilinx DMA IP-Core [2] is used to manage the data stream transmitted and received from the memory. This operation is implemented through the two DMA channels: Memory-Mapped to Stream and Stream to Memory-Mapped. This IP-Core has been used in order to perform reading and writing operations without involving the Microblaze so that it is able to proceed running the software.

The read channel is configured to read 32-bit words from the external memory and to transfer 16-bit words to the Convolution Engine IP-Core. The write channel sends 32-bit results to the DDR Memory.

2.1.3 Memory Interface Generator

The task assigned to the Xilinx MIG IP-Core [4] is to receive input data from the AXI-Stream Interface and adapt them to be sent through the DDR Memory interface. As an example, the MIG module generally works at the higher frequency which is typically used for the external DDR memory.

2.1.4 MicroBlaze Soft-Processor

The MicroBlaze soft-processor [3] has been used as part of the processing system in the designed embedded system. It coordinates the operations performed by each module of the system running a specific software. The processor configures every channel of the DMA in terms of input and output data addresses and number of data to be processed. It also provides kernel and SIMD values to the Convolution Engine exploiting its AXI-Lite interface.

2.2 Software

All the configuration operations are described in a C software that is executed by the MicroBlaze processor. The C code must include at least the following steps:

- Convolution Engine IP-Core configuration: writing of nine kernel values and one SIMD bit
- DMA channels Reset
- S2MM Channel Activation: channel enabling, destination address writing, number of bytes to transfer
- MM2S Channel Activation: channel enabling, source address writing, number of bytes to transfer

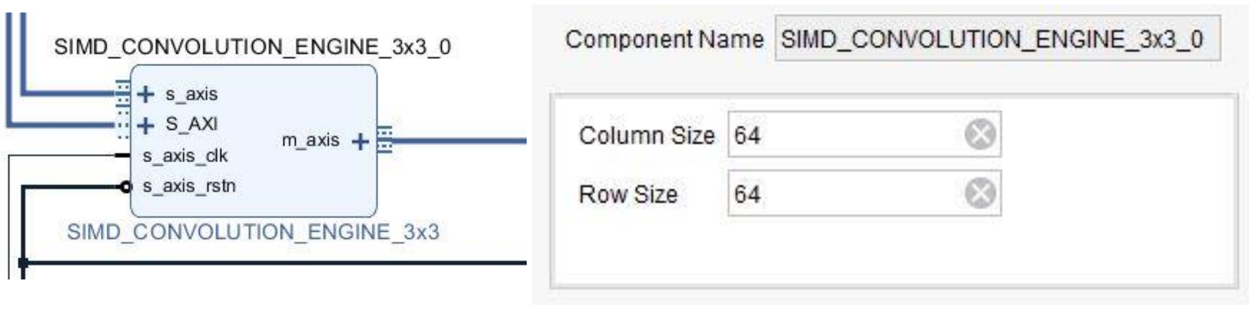
The code can be optimized using the writing/reading functions included in Xilinx's "*xil_io*" and "*xil_printf*" libraries. As an example, writing operations in the written test code have been implemented using the Xil_Out32 function.

In addition to this, complementary sections can be added to the code. More specifically, it could be useful to realize a for loop to write in the memory a specific data pattern to be processed.
Another functionality that can be implemented through a for loop is the reading of the obtained results and their visualization. A proper C software could be used to exploit the IP-Core capability to perform consecutive operations on the same matrix taking the 8-bit operation output and sending it back as an input for the 16-bit operation.

2.3 Design Reuse

During design phase, in order to maximize the versatility and flexibility of the system, some of the modules were described through parametric VHDL code. This allowed us to obtain an IP-Core that can be easily adapted to different application contexts which may require data matrices of different sizes. Moreover, the VHDL codes related to some sub-modules can be extracted from the system and reused by specifying, according to needs, the values of various parameters.

In particular, when using the IP-Core within an embedded system, as illustrated in figure 6, it is possible to configure two parameters related to the input data: *Row Size* and *Column Size*.



Component Name SIMD_CONVOLUTION_ENGINE_3x3_0

Column Size 64

Row Size 64

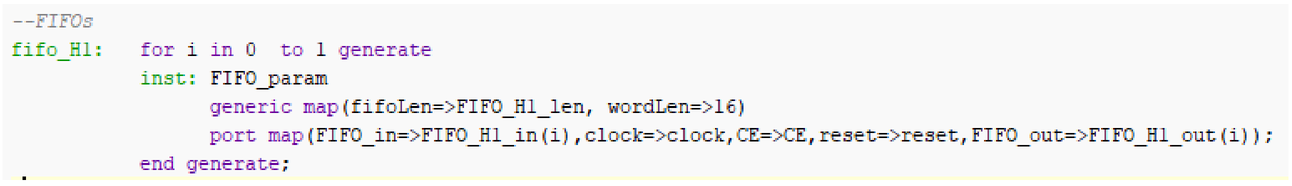
Figura 6 IP-Core Component overview

Among the parametric VHDL modules available in the design we have: “SIMD_Sum” and “FIFO_param”:

- **SIMD_Sum**: it is a binary SIMD adder that receives two input words of N bits and calculates their sum by interpreting them as (N)-bits single words or (N/2)-bits parallel words. This type of binary adder can be useful in all those applications that require two different degrees of parallelism and a configurable word length.
- **FIFO_param**: another fundamental component in convolution operations is the buffer which contains registers and FIFO structures. The design features easily reusable and configurable

FIFOs. The VHDL code can be adapted to fit specific needs in terms of depth of FIFOs and length of the single binary word.

All the parameters just described can be configured during the design phase by acting on the VHDL code. Figure 7 shows an example of it.



```
--FIFOs
fifo_H1:    for i in 0  to 1 generate
            inst: FIFO_param
                  generic map(fifoLen=>FIFO_H1_len, wordLen=>16)
                  port map(FIFO_in=>FIFO_H1_in(i),clock=>clock,CE=>CE,reset=>reset,FIFO_out=>FIFO_H1_out(i));
            end generate;
```

Figure 7 Example of utilization of a parametric FIFO module

The system also offers the possibility to reconfigure some parameters via software, while the system is running, between a computation and the other. In particular, it is possible to modify the kernel coefficients and the degree of parallelism to be used. This feature is implemented through two slave registers accessible in writing mode with an AXI4-Lite interface. Functionalities like these are in great demand as multiple cascaded elaborations are often performed with different kernel weights.

The output results from each filtering operation may have different dimensions and therefore may require different processing circuits. Using the designed IP-Core, this operation can be easily performed exploiting SIMD logic and therefore using a single calculation circuit. In this way, resources utilization and power dissipation are optimized.

As far as the input and output data interfaces are concerned, they are designed according to the AXI4-Stream protocol logic. This gives more flexibility to the module and makes it easy to integrate it into various complex embedded systems.

The project repository is available at: https://github.com/AMigali/SIMD_CE

3. RESULTS

3.1 Challenges

The proposed Convolution Engine has been designed without using any third-party IP-Core or any ready-made VHDL module. Each component has been designed specifically for the particular application.

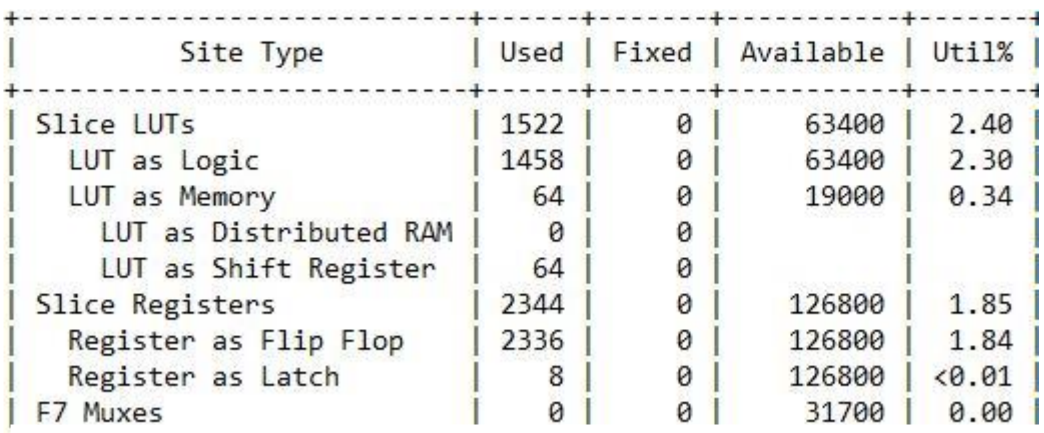
The description of custom VHDL modules, able to operate according to the SIMD logic and able to manage input data of different sizes, was the main challenge faced during the design phase.

A module that required particular attention from this point of view is the Pixel Buffer. The problem that this section of the circuit has to solve is the construction of the convolution windows. This operation must be handled differently, according to the SIMD accuracy used. The correct data management required the design of a particular sub-module usually not present in the classic architecture of a pixel buffer.

Another challenge faced during the design concerns the realization and optimization of the input data and kernel weights multiplication circuit. During the development phase, in fact, various architectures were tested until an efficient and reliable structure was obtained. Moreover, the whole design process was done trying to minimize the number of utilized hardware resources.

3.2 Resources Utilization

The table below shows the data obtained by implementing the circuit with rows and columns parameters equal to 64. The data refers only to the IP-Core, analyzed outside of the embedded system.

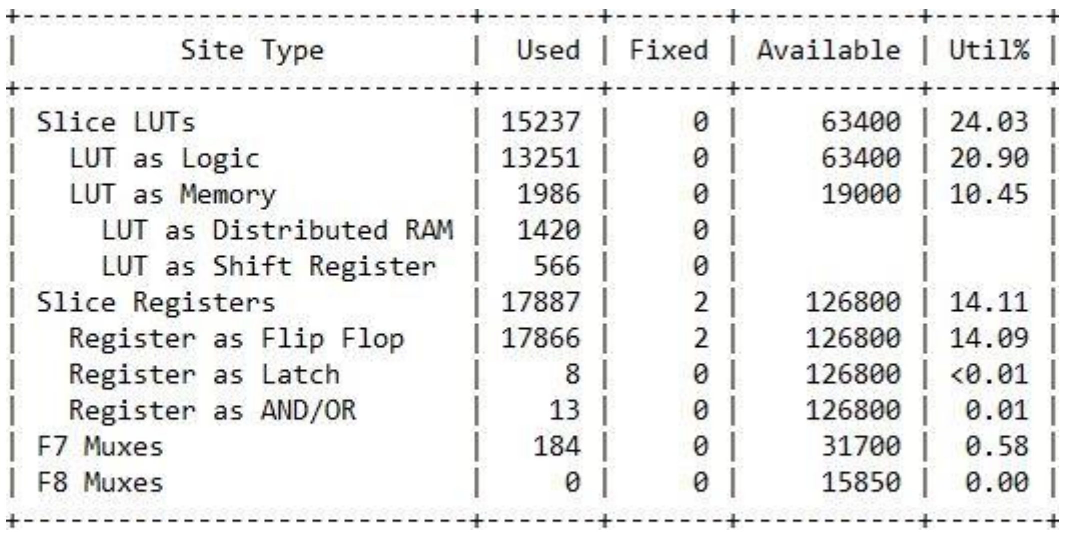


```
+----------------------------+------+-------+-----------+-------+
|          Site Type         | Used | Fixed | Available | Util% |
+----------------------------+------+-------+-----------+-------+
| Slice LUTs                 | 1522 |     0 |     63400 |  2.40 |
|   LUT as Logic             | 1458 |     0 |     63400 |  2.30 |
|   LUT as Memory            |   64 |     0 |     19000 |  0.34 |
|     LUT as Distributed RAM |    0 |     0 |           |       |
|     LUT as Shift Register  |   64 |     0 |           |       |
| Slice Registers            | 2344 |     0 |    126800 |  1.85 |
|   Register as Flip Flop    | 2336 |     0 |    126800 |  1.84 |
|   Register as Latch        |    8 |     0 |    126800 | <0.01 |
| F7 Muxes                   |    0 |     0 |     31700 |  0.00 |
```

Table 1 IP-Core utilization report

IP-Core only occupies about 4% of the total resources available in the FPGA chip used (Artix7 on Nexys4-DDR board), thus leaving a lot of space to the rest of the system.

As far as the entire embedded system is concerned, data on resources utilization are shown in Table 2:



```
+----------------------------+-------+-------+-----------+-------+
|          Site Type         |  Used | Fixed | Available | Util% |
+----------------------------+-------+-------+-----------+-------+
| Slice LUTs                 | 15237 |     0 |     63400 | 24.03 |
|   LUT as Logic             | 13251 |     0 |     63400 | 20.90 |
|   LUT as Memory            |  1986 |     0 |     19000 | 10.45 |
|     LUT as Distributed RAM |  1420 |     0 |           |       |
|     LUT as Shift Register  |   566 |     0 |           |       |
| Slice Registers            | 17887 |     2 |    126800 | 14.11 |
|   Register as Flip Flop    | 17866 |     2 |    126800 | 14.09 |
|   Register as Latch        |     8 |     0 |    126800 | <0.01 |
|   Register as AND/OR       |    13 |     0 |    126800 |  0.01 |
| F7 Muxes                   |   184 |     0 |     31700 |  0.58 |
| F8 Muxes                   |     0 |     0 |     15850 |  0.00 |
+----------------------------+-------+-------+-----------+-------+
```

Table 2 Embedded System utilization report

Most of the resources are used by the MicroBlaze soft-processor, DMA and MIG.

3.3 Performances

The system has been implemented with a working frequency of f=100MHz. According to the timing data obtained from the analysis, the maximum achievable frequency is equal to:

$$F_{MAX} = \frac{1}{(Actual\_Period\ -\ WNS)} = \frac{1}{(10 * 10^{-9} - 1.31 * 10^{-9})} \cong\ 115\ MHz$$

The time required to filter a $Row\_Size * Column\_Size$ matrix, in the case of 16 bit precision, is equal to:

$$T_{elab} =\ (Latency\_IN + Row\_Size * Column\_Size + Latency\_OUT) * Tclk$$

where $Latency\_IN = Column\_Size + 1$ and $Latency\_OUT = 7$

(In the 8-bit precision case, the above formulas need to be modified using $Column\_Size/2$ instead of $Column\_Size$)

This time duration was compared with the $Telab\_sw$ calculation time required by a software convolver for the same operation. Results are shown in table 3. This gives a first indication of the actual acceleration of operations:

DIMENSIONS (columns*row)	Telab (IP-Core)	Telab_sw (Matlab script)
64*64	41.680 µs	62.500 ms
640*480	3.078 ms	125.000 ms
1920*1080	20.755 ms	171.000 ms

Table 3 IP-Core and Software convolver elaboration time comparison for various image sizes

The following images were obtained by performing a Laplacian kernel filtering through the designed module:

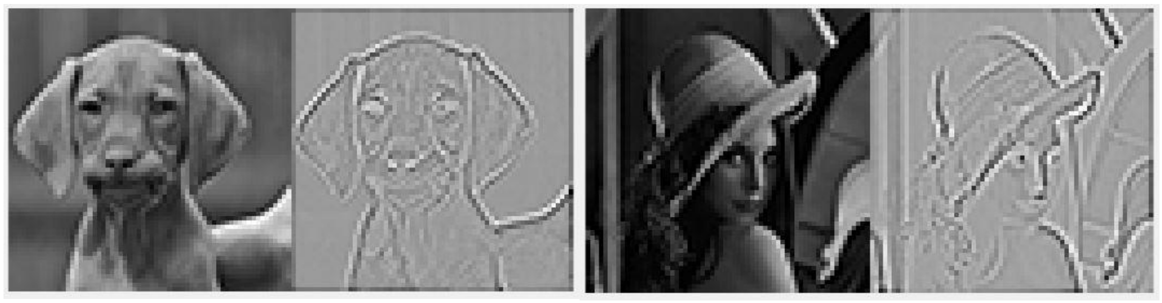
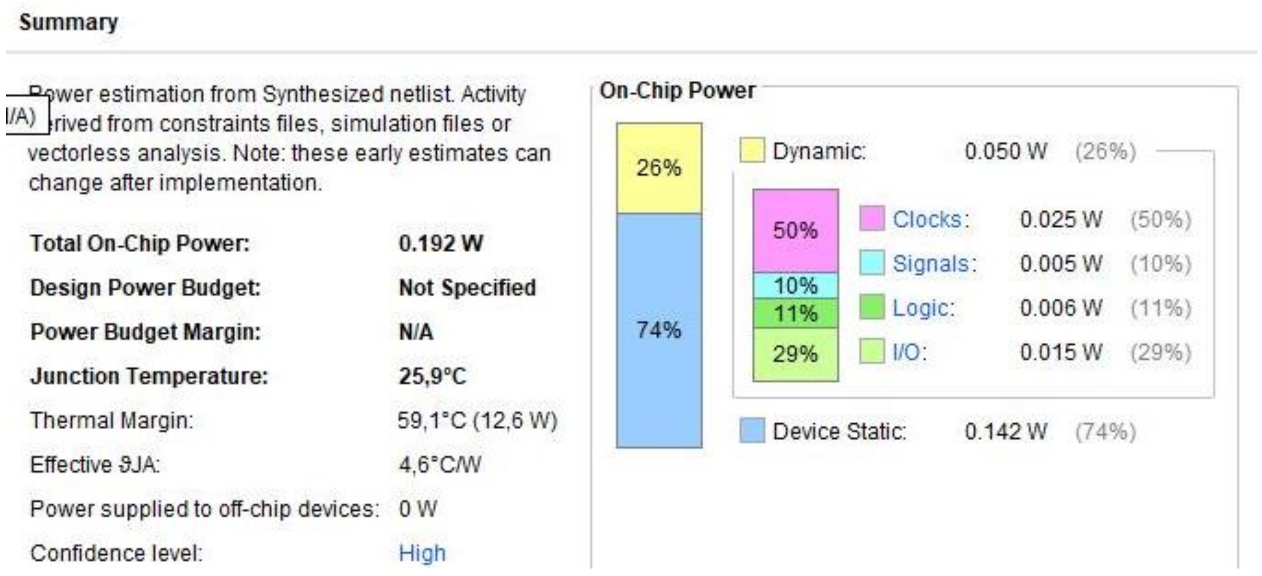


Figure 8 Examples of images filtered using the designed SIMD convolution engine with 16 bit precision

3.4 IP-Core Power Consumption

The power dissipation analysis was performed on the individual IP-Core outside of the system. The results obtained are as follows:



Summary

Power estimation from Synthesized netlist. Activity I/A) rived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	**0.192 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**25,9°C**
Thermal Margin:	59,1°C (12,6 W)
Effective ϑJA:	4,6°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	High

On-Chip Power

🟨	Dynamic:	0.050 W	(26%)
🟪	Clocks:	0.025 W	(50%)
🟦	Signals:	0.005 W	(10%)
🟩	Logic:	0.006 W	(11%)
🟩	I/O:	0.015 W	(29%)
🟦	Device Static:	0.142 W	(74%)

Figure 1 IP-Core power dissipation summary

4. CONCLUSIONS

The presented IP-Core was designed mainly for the Image and Video Processing field and in particular for applications concerning CNNs. Realizing a system able to work in this application field puts the designer in front of some problems related to the strict specifications that must be respected. In fact, to obtain optimal results in a very short time, it is necessary to design systems able to work efficiently from the point of view of hardware resources and power consumption. In addition, it is essential to ensure the simplicity of integration of the IP-Core within more complex embedded systems.

Both these targets have been met. In particular, the Single-Instruction-Multiple-Data (SIMD) logic made it possible to create a single circuit adaptable to different situations and particularly suitable for cascaded filtering operations. Moreover, the convolution engine complies with the AXI4-Stream and AXI4-Lite protocols which allow it to easily communicate with other modules inside a generic embedded system.

One of the main challenges faced during the system development was the design of the individual functional modules according to the SIMD logic. The achievement of this result has made the calculation structure particularly versatile and suitable for all those situations that require cascaded filterings with different degrees of parallelism.

The correct functionality of the whole system, with both the 8-bit and 16-bit degree of parallelism, has been verified with several on-board tests whose results were compared with Matlab produced ones.

The initial goals of creating an efficient Convolution Engine capable of optimizing the use of resources and power consumption have been achieved. Some possible future improvements could involve the management of a configurable kernel size and a possible extension of the degrees of parallelism supported by the various SIMD modules.

5. REFERENCES

[1] V. Sze, Y. Chen, T. Yang and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," in *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295-2329, Dec. 2017, doi: 10.1109/JPROC.2017.2761740.

[2] Xilinx, "AXI DMA LogiCORE IP, PG021 (v7.1)," 2019. Available: https://www.xilinx.com/support/documentation/ip_documentation/axi_dma/v7_1/pg021_axi_dma.pdf

[3] Xilinx, "MicroBlaze Processor, UG984 (v2020.1)",2020. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2020_1/ug984-vivado-microblaze-ref.pdf

[4] Xilinx, "Zynq-7000 AP SoC and 7 Series Devices Memory Interface Solutions v4.0, UG586", 2016. Available: https://www.xilinx.com/support/documentation/ip_documentation/mig_7series/v4_0/ug586_7Series_MIS.pdf