

University of Erlangen-Nuremberg

Xilinx Open Hardware 2018 Design Contest

Team: xohw18-155

PYNQ-OpenCV

cv2pynq

Author:

Wolfgang BRUECKNER

Supervisor:

Steffen VAAS, M.Sc.

June 30, 2018



Link to YouTube Video:

<https://www.youtube.com/watch?v=nRxe-Nqv0l8>

Contents

1	Introduction	1
2	Design	2
2.1	Hardware	3
2.2	Software	6
2.3	Design Reuse	7
3	Results	8
4	Conclusion	11
	References	12

1 Introduction

This project designs and evaluates an approach for accelerating image processing functions of the Open Source Computer Vision Library (OpenCV). The workload of this computation-intensive tasks is outsourced and parallelized on the programmable logic part of a System on Chip (SoC). Therefore real-time image processing can be achieved on a low-cost hardware platform. This design is integrated into the PYNQ framework, an open-source project using the Python language to accelerate the creation of embedded applications for SoCs. The whole design is integrated into a Python library called *cv2PYNQ*. This library can be easily installed on the PYNQ [5] platform and used by Python programmers.

In the last few years, the importance of computer vision applications has steadily increased as diverse areas of application are developed. Various of these use cases such as robotics, autonomous cars or medical systems require real-time image processing in order to deliver useful results in their domain. An autonomous car, for example, needs a fast lane-tracking system to generate movement instructions and avoid collisions. Conventional low-cost embedded systems are equipped with processing units which cannot deliver the required computing power, especially as the resolution of the input images increases. Through the use of SoCs, hardware accelerators can be harnessed to parallelize and calculate the image processing tasks on the FPGA part. This approach accelerates the computation and reduces the

usage of the main CPU heavily. Another advantage of SoCs is the reduced energy consumption due to the specialized hardware which is a significant feature in mobile applications such as autonomous cars.

The popularity of Python increased during the last few years and even reached the first place in an IEEE survey [1]. Due to its simplicity and high abstraction level, it is ideally suited for rapid application development. The OpenCV-Python library is composed of Python bindings which work as an API-layer to offer the vast range of OpenCV functions in Python. Utilizing Python as the application programming language and accelerating image processing task unnoticed by the user on hardware is thus an ideal combination of a rapid application development while delivering real-time results at the same time. Consequentially, this project used the PYNQ platform from Xilinx because it unifies all the aspects mentioned above. It runs on a ZYNQ-7020 SoC [7], uses Python as the programming language for developers and provides an easy to use web interface through Jupyter-Notebooks.

2 Design

The project was build using *Vivado 2017.4* and *Vivado HLS 2017.4*. It was designed with the need for expansion in mind, so we created a very adaptable architecture. The hardware design uses image processing cores created with *Vivado HLS* and connects them to the *ZYNQ7 Processing System* with an *AXI Direct Memory Access* [2] IP-core. Additionally, the video subsystem from the base-design [6] is included, so the HDMI input and output ports can be used too. See figure 1 on page 4.

The whole functionality is encapsulated into a Python library. This makes the usage as well as tests and evaluation really simple. Just include the library on top of the Python script and leverage the capabilities of the hardware design. Every expansion of the library can be achieved by adding image processing cores in the corresponding subsystem of the *Vivado* project and adapting the *cv2PYNQ* library. The decision which functions to accelerate is based on computational complexity, the portion of parallel computability, achieved speedup and frequency of use. The project currently accelerates the following widely used OpenCV functions for gray-scale images with eight bits per pixel for any resolution up to 1920x1080 pixel.

- Sobel: kernel size = 3 & 5

- Scharr
- Laplacian: kernel size = 1 & 3 & 5
- blur: kernel size = 3
- GaussianBlur: kernel size = 3
- erode: kernel size = 3
- dilate: kernel size = 3
- Canny edge detection

2.1 Hardware

Each function core is configurable and designed with Vivado[®] HLS, a High-Level Synthesis tool which converts a C-like implementation into a ready to use IP-block for the Vivado[®] design environment. This approach accelerates the design process compared to the development in hardware description languages like VHDL. Each core can be tested in a simulation independent of the hardware. The results are compared to the output of the OpenCV functions. After the validation of the results, every image processing core is inserted in the hardware design and connected via the advanced extensible interface bus (AXI) streaming interface for each input and output. See figure 2 on page 5. The image processing cores are designed with the help of the HLS Video Library [3]. It provides ready to use implementations of image filters like Sobel. However, including a separate IP core for every functionality uses loads of hardware resources. That's why the design only includes the common foundations of each filter type. The *Sobel*, *Scharr* and *Laplacian* filter can be implemented by an common *Filter2D* with configurable kernel parameters. The same principle can be applied to the *blur* and *GaussianBlur* filters with a kernel containing fixed-point parameters.

Every image core has a similar interface consisting of an input and an output AXI-Stream for the image data and an additional AXI-Lite interface for the parameters. Each parameter gets a corresponding address offset in the AXI-Lite interface.

```
#pragma HLS INTERFACE s_axilite port=rows bundle=CONTROLBUS
                        offset=0x14 clock=AXI-LITE_clk
```

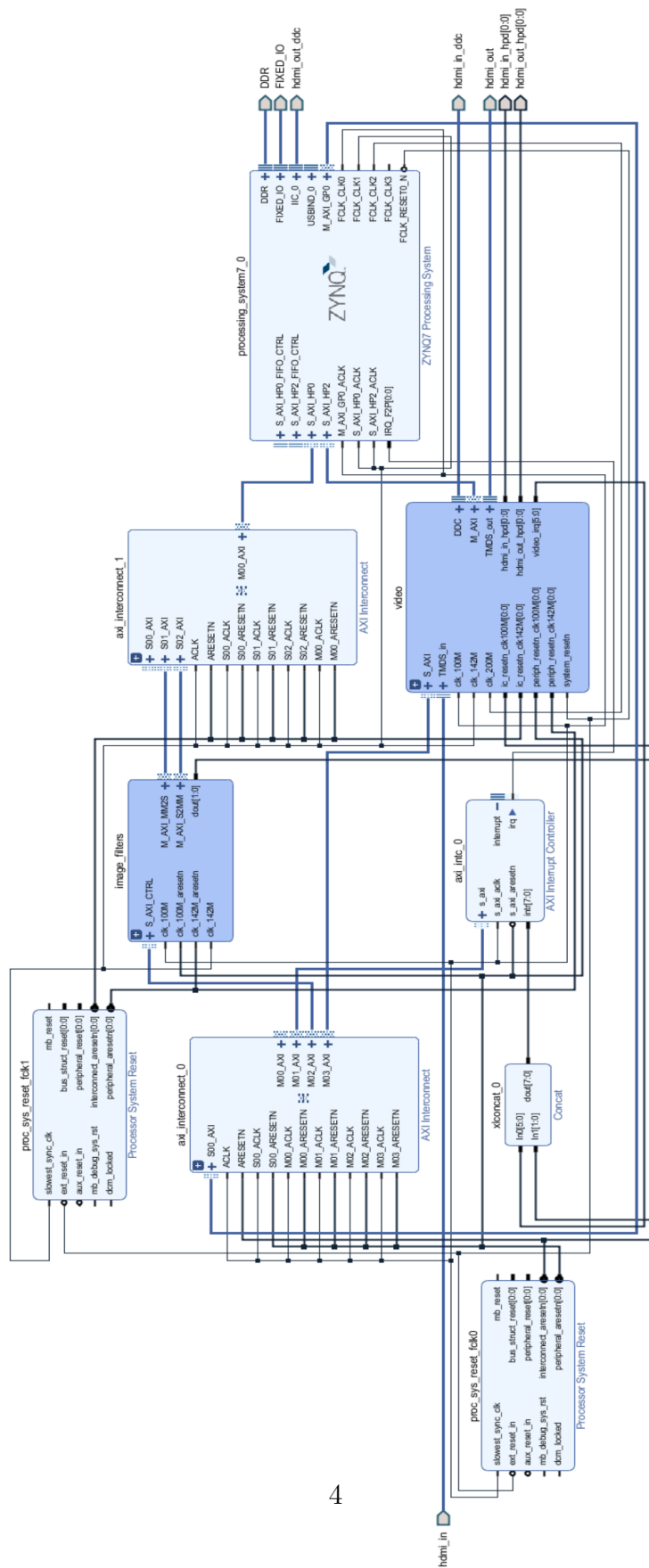


Figure 1: Hardware Block Design Overview

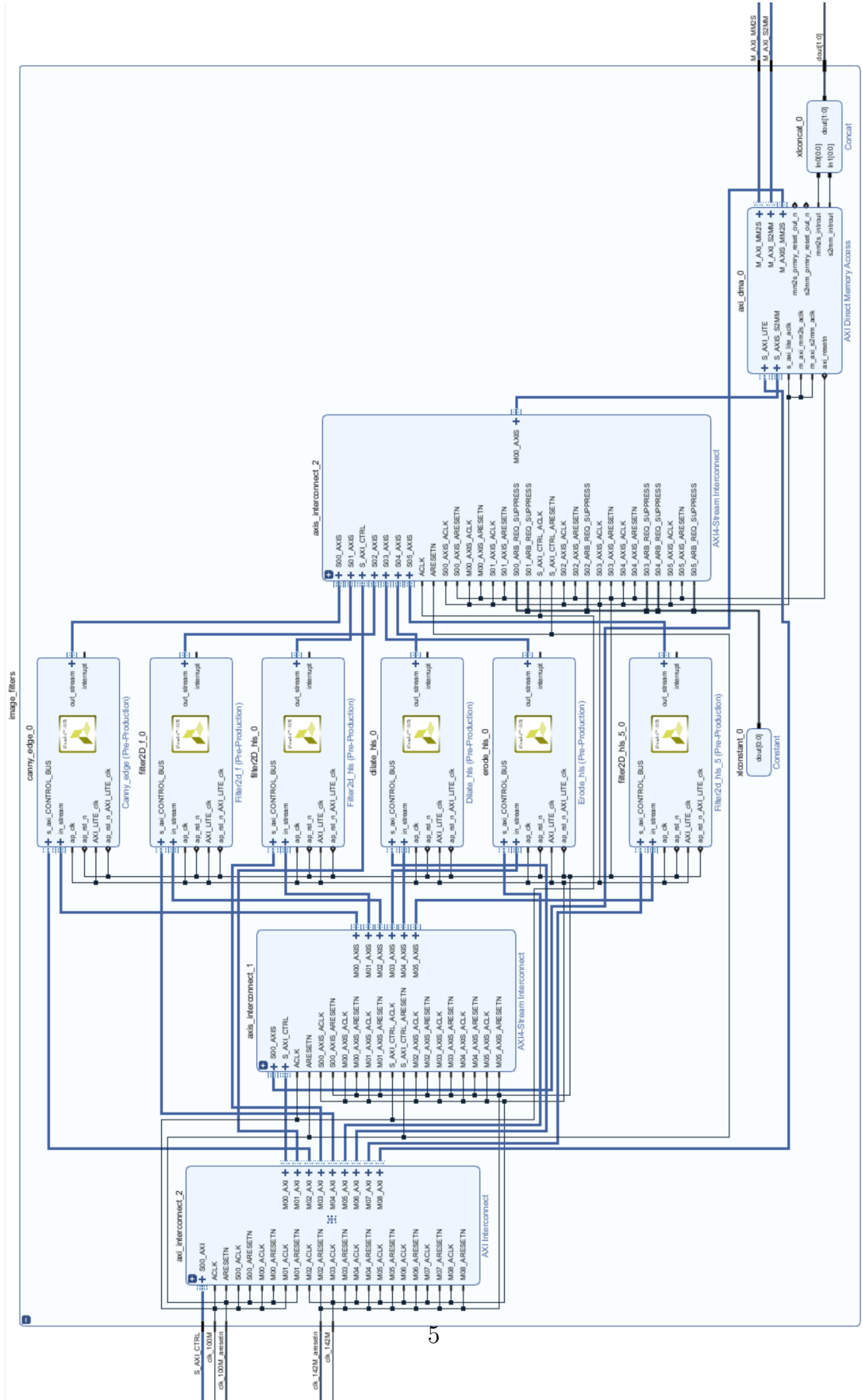


Figure 2: Image Filters Block Design

The hardware architecture takes advantage of direct memory access (DMA) via AXI provided by Xilinx [2]. It provides high-bandwidth memory access to the DRAM and uses shared memory regions to exchange data between the processors and the programmable logic. The design only uses one DMA because only one image processing core can be used by the Python programmer at one point in time. This fact heavily reduces the resource usage compared to a design with one DMA for each core. Compare with table 5 on page 10.

2.2 Software

The project comes with the *cv2PYNQ* library. This was created according to the corresponding *readthedocs* page [4]. Each library contains a *steup.py* file containing all the information needed for installation. This includes the version number, author, and other required Python libraries. The *MANIFEST.in* is executed during the installation process and copies the bitstreams for the programmable logic as well as the TCL files describing the design. In the *cv2pynq* folder are the *__init__.py* and the *cv2pynq.py* files located. The former accommodates the function definitions which check the given parameters and includes the video subsystem from the overlay, the latter implements the interface to the actual hardware. *cv2pynq.py* defines a class for each image processing core which inherits from the *DefaultIP* class and is associated with the hardware IP block. This translates the parameters defined in HLS to class properties. Each common functionality like the *Filter2D* is encapsulated in a separate function. They are called by the filter functions after parameterizing the image processing cores according to their requirements. In order to save the time for the allocation of contiguous memory arrays needed if the source or destination is not from that type, the library defines a wrapper class called *ContiguousArrayCv2pynq*. This class overwrites the *nbytes* attribute of the arrays which can be set to a custom value. This allows to adapt the value to the given image size and transferring the data through the PYNQ DMA function, which checks for that special attribute.

2.3 Design Reuse

All parts of the design have been made available (Open Source) for reuse. The *cv2OYNQ* library is uploaded to GitHub¹ and the Vivado project is also available on GitHub². The project encourages to expand the library and contribute to the repository. Furthermore, the library comes with a Jupyter-Notebook to demonstrate its usage and capabilities. It is automatically copied to the *cv2PYNQ* folder of the home directory after installation.

¹cv2PYNQ: <https://github.com/wbrueckner/cv2pynq>

²cv2PYNQ-The project:
<https://github.com/wbrueckner/cv2PYNQ-The-project-behind-the-library>

3 Results

The Library is able to accelerate a significant speedup compared to the default software implementation of OpenCV functions. The higher the computing effort gets, the higher is the speedup of the hardware. A similar speedup

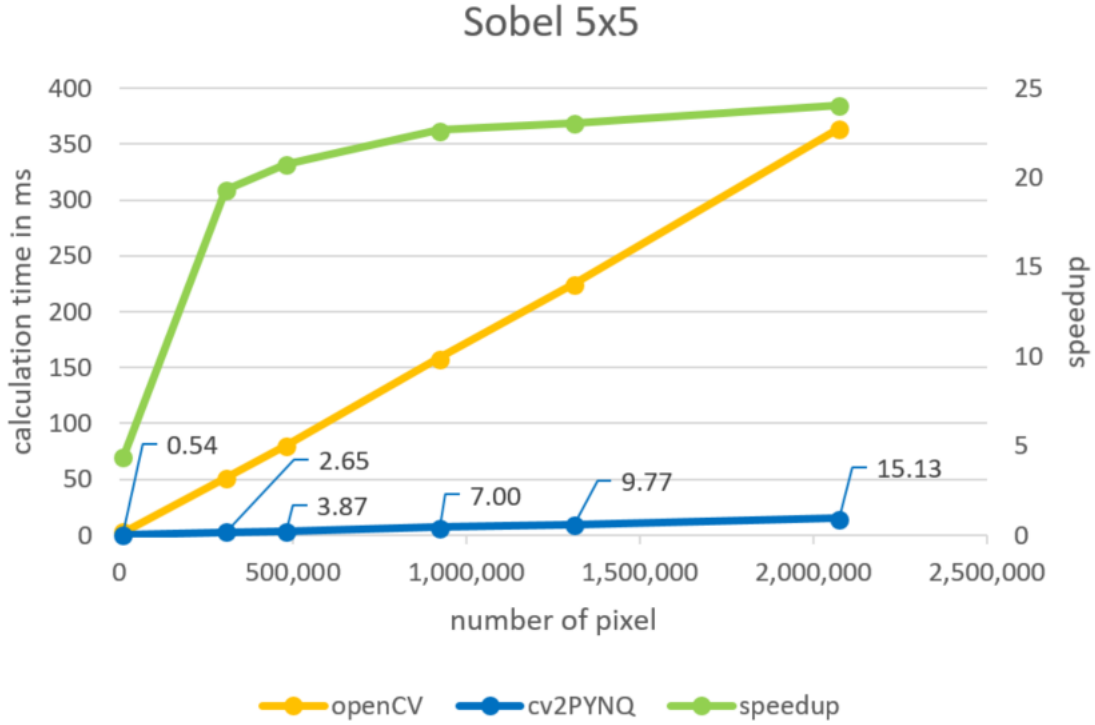


Figure 3: Sobel 5x5 computation time in Software and Hardware

can be achieved for the Canny edge detector. The other supported filters are still significantly accelerated, but the speedup is shrinking as the complexity reduces. The image processing cores take 16 ms for each 1080p image, independent of the function. This time is due to the bandwidth limitations of the DMA and the AXI high-performance interfaces to the processing system.

The design uses a 100 MHz clock for the AXI-Lite interfaces and a 142 MHz clock for the image processing cores and the DMA. It uses less than 50% of each Resource except the DSPs with 51%. With less than half of the chip occupied, there is plenty of space for expansions.

Unfortunately, the Canny edge detector implementation in hardware delivers a slightly different result than OpenCV. This is based in the fact that the streaming implementation cannot execute the last step of hysteresis thresholding in a recursive manner. Every user has to decide for himself if the results are sufficient. The *blur* and *GaussianBlur* filters use fixed-point kernel parameters leading to a maximal difference of one bit in the result. All the other filters deliver exactly the same result as OpenCV.

Vivado HLS generates IPs containing the same module names. During the synthesis process of Vivado, this leads to redefinitions and may lead to unpredictable results. Thus, we created a Python script called *makeHLScode-Unique.py* located in the *ip/HLS* folder. This script replaces the names by adding the project name and therefore, make them unique within the project.

Resource	Utilization	Available	Utilization %
LUT	26280	53200	49.40
LUTRAM	1448	17400	8.32
FF	37529	106400	35.27
BRAM	34	140	24.29
DSP	113	220	51.36
IO	22	125	17.60
BUFG	3	32	9.38
MMCM	2	4	50.00

Figure 4: Resource usage of the design on a Zynq 7020












▼  image_filters (imag...	15744	19354	77	0	6040	14689	1055	6964	24	95
>  axi_dma_0 (desi...	1449	2043	1	0	599	1335	114	867	5	0
>  axi_interconnect...	659	1806	0	0	548	659	0	371	0	0
>  axis_interconnec...	936	1772	4	0	533	740	196	348	0	0
>  axis_interconnec...	649	1440	1	0	395	481	168	304	0	0
>  canny_edge_0 (...)	3530	3251	2	0	1234	3511	19	1262	9.5	32
>  dilate_hls_0 (de...	1304	1456	0	0	471	1227	77	571	1.5	8
>  erode_hls_0 (de...	1305	1456	0	0	509	1228	77	563	1.5	8
>  filter2D_f_0 (desi...	1648	2015	34	0	600	1493	155	808	1.5	17
>  filter2D_hls_0 (d...	1653	1680	27	0	569	1567	86	720	1.5	15
>  filter2D_hls_5_0 ...	2612	2435	8	0	868	2449	163	1120	3.5	15

Figure 5: Resource usage of the image filters

4 Conclusion

The library can be used by every Python programmer and therefore improves the capabilities of the *PYNQ* platform in an easy to use manner. We plan to expand the capabilities in the next few weeks so that the scope of the library includes even more use cases.

Only images in the contiguous memory can be processed at the speed mentioned above. If the image is located in the normal RAM, two copy processes must be added to move it into an input buffer and copy the results to a new array. This leads to a significant reduction of the speedup and Python programmers must be educated about that fact. This is done by a Jupyter-Notebook delivered with the library.

If the programmable logic resources should be restricted to contain all the functionalities added in the future, several hardware designs can be generated to expand the utility of this project. In this case, functionalities commonly used together must be included into one design to reduce the necessity to reload the bitfile of the programmable logic.

The library was consciously designed without the use of SDSoC environment because it demands an additional license.



References

- [1] The 2017 top programming languages. <https://spectrum.ieee.org/computing/software/the-2017-top-programming-languages>. Accessed: 2018-02-27.
- [2] Axi dma controller. https://www.xilinx.com/products/intellectual-property/axi_dma.html. Accessed: 2018-02-27.
- [3] Hls video library. <http://www.wiki.xilinx.com/HLS+Video+Library>. Accessed: 2018-06-25.
- [4] How to package your python code. <https://python-packaging.readthedocs.io/en/latest/index.html>. Accessed: 2018-04-12.
- [5] Pynq. <http://www.pynq.io/home.html>. Accessed: 2018-02-27.
- [6] Pynq github repository. <https://github.com/Xilinx/PYNQ>. Accessed: 2018-06-25.
- [7] Zynq-7000 soc product advantages. <https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html>. Accessed: 2018-04-25.