# Hardware/Software co-design of a key point detector on FPGA

H. Djakou Chati[1], F. Mühlbauer[1], T. Braun[2], C. Bobda[1] and K. Berns[2]
[1] Self-Organizing Embedded System Group
[2] Robotic Research Lab
Kaiserslautern University of Technology, Germany
{h_djakou,muehlbauer,braun,bobda,berns}@informatik.uni-kl.de

## Abstract

*The design and implementing of a key point detector on embedded reconfigurable hardware is investigated. The major challenges are efficient hardware/software partitioning of the key point detector algorithm, data flow management as well as efficient use of memory, bus and processor. We present a modular and manual hardware/software co-design, with its implementation on a Xilinx XUP-Virtex II Pro board co-design to solve these issues.*

## 1. Introduction

Key point detector algorithms have shown substantial success in a large diversity of applications. In embedded environment, like in robot navigation or in automotive, systems using key point detectors have to work in real time while maintaining a low power at the same time. Furthermore, today's systems must be able to adapt themselves to changing environmental and operational conditions. Embedded systems are no longer being used as simple controllers and the processors commonly used in these systems, like DSPs or ASICs cannot provide the requirements (performance, low power, and adaptivity) needed. The computational power needed to satisfy actual embedded systems like video or image processing have accelerated the emergence of Systems On Programmable Chip (SoPC). Here, an alternative for increasing the processing power of a system is to combine processor(s) with customized user hardware which is offered by FPGAs in a dynamic way.

This work presents an efficient hardware/software co-design of a key point detector in FPGA. The architecture is realized under consideration of the image stream and an efficient choice of resources (processor, memories, and custom hardware modules). In the resulting SoPC, a processor and a set of dedicated hardware accelerator nodes cohabits. The focus here resides in building a viable data flow management as well as an efficient use of memories, bus and processor features to speed up the computation of the key points. The prototype realized on the Xilinx XUP-Virtex II Pro FPGA board yields considerable improvement compared to the pure software solution.

## 2. Key Point Detection

Key point detection is an approach used in the field of Computer Vision to detect points of interest so-called *features* in images which also can be assigned to objects. The goal is the re-identification of features within new images e.g. captured by a camera. Such algorithms are used e.g. for generation of panoramas, object recognition and tracking or also visual navigation of robots. The main problem is to deal with the wide area of errors like affine distortions, deformations, change in 3D viewpoint, illumination changes, image noise, etc. The most robust key point detector algorithm is the SIFT (Scale Invariant Feature Transform) algorithm described by Lowe in [3]. It can extract distinctive invariant features respectively descriptors, which can be used to perform reliable matching of objects or scenes between different images or viewpoints. For extracting feature four filter stages are necessary:

**scale-space peak selection:** The Gaussian function is used to build up a pyramid (octave) of blurred images (scales). These scales are subtracted (Difference-of-Gaussian) to identify stable points (peaks).

**key point localization:** In this stage local extrema are search examining adjacent scales. The resulting peaks are filtered and only stable key points, invariant to e.g. noise or illumination changes, are kept.

**orientation assignment:** To each key point one or more orientations are assigned based on its gradient.

**key point descriptor:** The final stage builds an image descriptor for each key point considering gradients in its local neighborhood.

These calculations are very expensive and most implementations are running on desktop computers. Some work has already be done to increase performance by utilizing a

0-7695-2940-2/07 $25.00 © 2007 IEEE
DOI 10.1109/FCCM.2007.61

GPU (graphic processor in desktop computers) or by simplifying the algorithm through approximation [4, 1, 2].

Timing measurements result in stages which contain highly parallelizable parts consume about 70 percent of computation time. These parts are suitable for implementation in hardware to speed up overall performance.

## 3. Architecture and HW/SW Partitioning

Blurring images is a convolution operation which can be implemented very efficiently in hardware using so-called sliding windows. Therefor the image pixels are streamed, means shifted through a buffer (size is two image lines + 3 for a $3 \times 3$ convolution matrix) and so all 9 necessary multiplications can be done in parallel. Compared to a processor which has to load, calculate and store each pixel value for each convolution, the hardware solution has only an initial delay and then can perform one pixel per clock.

The Difference-of-Gaussian and the identification of key points is also a task well suitable for hardware. Executing several convolutions sequentially (as in stage 1) results in pixel streams with different delays, but calculating differences needs sychronious data. Our solution is to only use one big matrix (sliding window) for the convolutions, so all scales can be produced synchronously.

Pixels with low contrast or intensity are filtered out and for further processing the remaining data is examined for extrema. To detect one, 26 comparisons (8 in same scale and $9 + 9$ in adjacent scales) of pixel values in the immediate neighborhood are necessary which are very cheap implemented in hardware. About 50 percent of the pixels are rejected. Next, key points with edge response are sorted out. Therefor the local principal curvatures of the image resolution around a point is calculated as the ratio of eigenvalues of the $2 \times 2$ Hessian matrix of the image intensity at that point. Another reject criterion is the following: When an image is blurred many times, points in the original image could drift away from their initial position. In our implementation only drifts by one step are recognized because this is the case in about 80 percent. The remaining key points are considered to be stable. This task is more suitable for an implementation in software but therefor pixel values of the scales (stage 1) are necessary and have to be buffered meanwhile. For this reason this task was also implemented in hardware and data is stored into a dual ported memory.

Connected to the other port a processor performs the last stage namely building the feature descriptors. The magnitude and orientation of each image point was calculated, in parallel to the till now described processing chain, based on its gradient using a pipelined arithmetic unit and stored into a big memory. Now the processor reads a key points one by one from the dual ported memory and fetches the needed gradients (up to 40 columns/raws relative to the key point location).

The processor and the dedicated hardware modules run in different clock domains and so the processing speed is adjusted to each another. To inform the processor about the amount of data already stored in the memories, the feature extending the instruction set is used and the fill level of the memories are send over direct communication channels to the processor.

## 4. Results and Conclusion

The processor of the implemented design runs with 100 MHz and for a $320 \times 240$ sized image $0.8\ ms$ are consumed.

In this work, we presented the use of System On Programmable Chip to implement an efficient embedded Hardware/Software solution to a key point detector algorithm. The efficiency is achieved by using a viable hardware/software partitioning and data streaming approach between modules.

The adaptivity of the system can be achieved by exchanging some custom modules at run-time. In a separate bachelor thesis a framework was developed that allows the integration of run-time swappable modules in a SoPC. In this framework, a PowerPC is used to trigger the partial reconfiguration of from within the FPGA using the ICAP-port. The system was successfully tested with a video streaming application in which the convolution filters are swapped at run-time.

Our next step consists of integrating our work into the developed framework in order to swap some modules in and out according to the parameters of the convolution sequence. We also intend to deploy the complete system in a mobile robot to allow navigation using the SIFT-algorithm and the FPGA as computing platform.

## References


[1] M. Grabner, H. Grabner, and H. Bischof. Fast approximated sift. *In Proceedings of Asian Conference on Computer Vision*, 2006.

[2] Y. Ke and R. Sukthankar. Pca-sift: a more distinctive representation for local image descriptors. *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, pages II–506–II–513 Vol.2, 2004.

[3] D. G. Lowe:. Distinctive image features from scale-invariant key points. *In International Journal of Computer Vision*, 2004.

[4] M. P. Sudipta N Sinha, Jan-Michael Frahm and Y. Genc. Gpu-based video feature tracking and matching. *EDGE 2006, workshop on Edge Computing Using New Commodity Architectures*, May 2006.