W. Karl, J. Keller (Eds.): PARS 2017
Proc. 27th PARS-Workshop

An Image Processing Operator Language for Design and Synthesis of Smart Camera Architectures

Christian Hartmann [1] Konrad Häublein [1] Benjamin Pfundt [1] Marc Reichenbach [1]
Dietmar Fey [1]

Abstract: Recent trends showed a rise of heterogeneous hardware architectures for image processing applications. Due to the usage of these camera systems in the embedded field, the reduction of area and power consumption became essential. Standard CPUs are not suitable in the embedded field, because of their lavish commerce regarding power and area consumption. Embedded applications have strict constraints regarding these parameters. Therefore, optimized and specialized hardware is required resulting in a heterogeneous system architecture. Designing such a system is a challenging and error-prone task. In the design process, software and hardware skills are needed. Programming skills in different programming and design languages are necessary. For reducing the complexity a common language which can easily be mapped on different hardware architectures combined with a synthesis framework is needed. With the Image Processing Operator Language (IPOL) the description of heterogeneous systems with one language become possible. The synthesis framework called Image Processing Architecture Synthesis (IPAS) completes the domain-specific language (DSL) as an underlying mapping methodology.

Keywords: image processing, system design, compiling tools, domain-specific language, UML

1 Introduction

Due to the growth of heterogeneity in image processing systems, the complexity and the number of different programming and description languages rise. A trade-off between the design effort and system efficiency in terms of processing time, area and power consumption has to be found. By using traditional development tools the effort for efficient designs is too high. Thus companies are animated to choose a less efficient way, because of lack in time, personnel and money. Reducing the design effort for heterogeneous systems increases the acceptance in companies for the usage of heterogeneous architectures and this will lead to more efficient systems. It opens up the possibility for more image processing systems in the embedded field. Embedded image processing applications hold strict constraints regarding heat transfer, area and power consumption. Reducing the effort of designing image processing systems will also lead to an advantage in time to market. Also large projects of heterogeneous systems with groups of people using different programming and development background such as GPU programming, micro-controller and digital IC-design become feasible by using a common description language. Examples of such projects can be found in the automotive field.

[1] Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Martensstr. 3, 91058 Erlangen, {christian.hartmann, kondrad.haeublein, benjamin.pfundt, marc.reichenbach, dietmar.fey}@fau.de

For such projects the domain specific language IPOL serves as an interface and common description language for creating an efficient hardware architecture. Hence IPOL was created with the IPAS framework as synthesis methodology. IPOL is a DSL which is used for mapping the common structures of an image processing application on different hardware architectures by using the IPAS framework. For finding a suitable hardware architecture for an image processing application the IPOL provides an opportunity of defining optimization targets and constraints such as system accuracy, processing time, area and power consumption. The optimization and mapping will be done by the synthesis tool called IPAS.

2 Related Work

Domain-specific languages for image processing applications are unoriginal and not new. Frameworks and DSLs for describing image processing applications for different hardware platforms with one description language do already exist. A legitimate question is about the need of a new DSL with underlying mapping methodology. One aspect is the missing optimization possibilities for *system parameters* such as accuracy, processing time, area and power consumption the existing tools do not support. The framework HIPAcc introduced in [RHR+15] provides a C/C++ library for the description of image processing applications. These applications are able to run on GPUs, CPUs and Xilinx FPGAs by using Vivado HLS, [HLS17]. However, there is no optimization logic regarding *system parameters*. The Vivado HLS back-end is too limited for a suitable mapping. The resulting circuits are not resource aware and do not consider specialized hardware structures such as the *Full Buffering*, [SRF12]. A second framework such as CubeGen, introduced in the work of [CDAC13] is a rapid prototype platform for testing, evaluating and compiling the functionality of image processing applications on different hardware architectures. However, this framework does only work for CPUs by utilizing soft-cores such as Microblaze or NIOSII. CubeGen tries to distribute image processing applications on whole processor cores. This can be highly inefficient for image processing algorithms with a local memory access pattern. A further interesting approach is the Preesm framework introduced in [PDH+14], it optimizes image processing applications for digital signal processors from Texas Instruments. However, as for CubeGen Preesm does not provide compareable results for low power and low area applications such it can be achieved with application specific heterogeneous hardware architectures. Also a product dependency on Texas Instruments products exists with Preesm. Another weakness of existing approaches is the strong limitation in stencil codes. Frameworks such as HIPAcc, Streamit [Str17] and Darkroom, [HBD+14], are not able to support global processing algorithms such as the Hough transformation [KA95] and SURF algorithm in [HQ11]. They do not support a whole image processing pipeline with local and global processing algorithms. They just support algorithms with a local data access pattern. That limits the effort of these tools. The third aspect is the missing flexibility and extendability of the frameworks. In case of HIPAcc or Darkroom the user is not able of using their own image processing operators. Only a very limited number of predefined existing image processing algorithms can be used. The last argument for defining a new image processing DSL with underlying mapping methodol-

ogy is that the existing frameworks do not support the communication between the different components of a heterogeneous system. The bus protocol support for buses such as AXI or I2C is missing for connecting different hardware components. But these protocols are needed for supporting a real heterogeneous image processing system. Therefore we created the image processing DSL called IPOL with the underlying mapping methodology called IPAS.

3 The Example Application

This section introduces the test application used as an example, in order to illustrate the synthesis concept. The image processing application is a circle centroid detection, implemented by the Sobel-Bresenham algorithm. This kind of application is often used in the industry automation field for marker and position detection. In non-industrial fields it can be used as object detection for sport events, eye-tracking or automated landing of drones. The Sobel-Bresenham centroid detection algorithm covers the following processing scheme for detecting the centroid point of a circle. The first stage is a Sobel edge detection algorithm used for shape detection of the circle which is done by detecting the edge points. Coherent edge points are forming the circle's shape. The second result of the Sobel algorithm is the gradient's direction for each edge point. An illustration of the algorithm is shown in Figure 1.

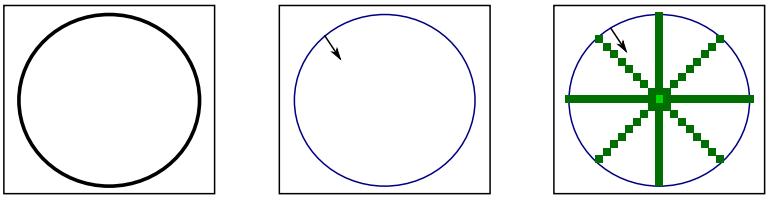

Fig. 1: The algorithm from left to right: Input image, *Canny* with gradient direction, *Bresenham* with the search path in gradient direction

The left Figure shows the input image, the central part shows the result of the *Sobel* edge detector. The pointer in the central Figure is the gradient direction exemplary for one edge point position. The result of the edge detector is always the edge point with its gradient direction. The Sobel algorithm is followed by the Bresenham algorithm and starts the processing for each detected edge point in the direction of the gradient. By touching every pixel in gradient direction a counter at each pixel recognizes the total number of touches. The pixel with the highest counter value represents the centroid point. This is shown in Figure 1 on the right. Exemplary eight edge points are used for determining the centroid point colored in light green. The advantage of this algorithm is the flexibility in different circle sizes. The algorithm can work with every circle size.

4 Structure of IPOL and the Mapping Methodology of IPAS

The last section introduced the image processing application. This section shows how such image processing applications can be described in the IPOL DSL. The image processing

operator language (IPOL) is used as formal description language with its structure and the synthesis design flow called image processing architecture synthesis (IPAS), [HHRF16]. IPOL is an XML-based meta-language for image processing applications. Every image processing application follows a standard processing scheme - (1) data acquisition, (2) local preprocessing, (3) global post processing and at the end a (4) data sink for display or storage. These parts are distributed in several so-called image processing operators. Image processing operators can be image processing algorithms such as Sobel, Canny, Hough transformation etc. or hardware elements such as image sensors for the data acquisition and monitors for the data representation. Numerous image processing operators combine to an operator chain is called image processing pipeline. All these elements can be described by IPOL.

An example pipeline comprising the Sobel-Bresenham application is shown in Figure 2. Active elements such as operators are described inside the *image_operators* tag. Operators of an image processing application can be described in two ways. For well-known operators such as the Sobel and the Bresenham, predefined mapping modules are provided by a library inside the IPAS framework. They can be addressed by using the *name* in the *op_object name* part e.g. $< op\_object\ name ='' Sobel'' >$. The communication between the image processing operators is managed in the *connection* part.

In the part *system_parameters* the requirements of the image processing application are defined. Application-specific test cases can be created in the *system_testcase* part. A realistic test environment will be formed by using the *system_testcase*. The *synthesis* part at the end of Figure 2 is a reserved place for the results produced by the mathematical, simulation and implementation layer. These layers will be introduced in a later section. In the following a more detailed description of the elements inside the *image_operators* and *chain* parts will be given.

image_operator are members of the image processing application such as sensors, monitors or implementations of image processing algorithms.

type operator types differ between *Sensor*, *Monitor* and *Operator*.

op_class image processing algorithms are distributed in classes for example the Sobel, Canny, LoG and Roberts are all edge detectors. This information can be used for comparing the members of an *op_class* and find the best solution for the application.

op_object The name of the image processing operator, it will be used for referencing the (predefined) implementation. It also contains the definitions of input and output ports and parameters for the implementation. The parameters can influence the shape of the image processing operator (e.g. for local operators the mask size).

connect describes the communication between the image processing operators. The parameters *op_out* refer to the sending operator, *op_in* to the receiving operator. Multiple ports of each operator can be defined by using *port_out* and *port_in* with different ports for the same operator.

When reading the introduction about IPOL and how to describe an image processing application in IPOL, you may think about the need of learning such an uncomfortable XML

```
<ipol>
<system_parameters>
  <accuracy val="20" unit="db">
  <power_consumption val="10" unit="W" />
  <image_size pwidth="1920" pheight="1080"/>
</system_parameters>

<system_testcase>
  <object_equation>y=5*x+4</object_equation>
  <domain var="x" lower="100" upper="500"/>
</system_testcase>
...

  <image_operator id ="1">
    <type>Operator</type>
    <op_class>Edge</op_class>
    <op_object name="Sobel">
      <generic>
        <mask>3x3</mask>
      </generic>
      <ports>
        <p name="edge" width="1" direction="out"/>
        <p name="dx" width="32" direction="out"/>
        <p name="dy" width="32" direction="out"/>
      </ports>
    </op_object>
  </image_operator>

  <image_operator id ="2">
    <type>Operator</type>
    <op_class></op_class>
    <op_object name="Bresenham">
      <generic>
        <steplength>50</steplength>
      </generic>
      <ports>
        <p name="edge" width="1" direction="in"/>
        <p name="dx" width="32" direction="in"/>
        <p name="dy" width="32" direction="in"/>
      </ports>
    </op_object>
  </image_operator>
  ...

<chain>
  <connect op_out="0" op_in="1" port_out="0" port_in="0"/>
  <connect op_out="1" op_in="2" port_out="edge" port_in="edge"/>
  <connect op_out="1" op_in="2" port_out="dx" port_in="dx"/>
  <connect op_out="1" op_in="2" port_out="dy" port_in="dy"/>
  <connect op_out="2" op_in="3" port_out="0" port_in="0"/>
</chain>

<synthesis>
  ...
</synthesis>
</ipol>
```

Fig. 2: Example of an operator chain in the IPOL

based language. The answer is very simple, you don't have. IPOL was developed to simplify the design process of image processing applications by using high level modeling methods such as SysML, [CLY+15]. The advantage of using an XML-based language is that most modeling tools have an XML-based representation of their UML or SysML diagrams. This representation can easily be transferred in an IPOL-conform language.

4.1 The IPOL Instruction Set

This subsection introduces a methodology for creating custom image processing operators. It shows the instruction set for custom operators in the IPOL language and their utilization. The definition of new custom operators can be done in the *image_operator_class* part of the IPOL language and can be referenced from the *image_operator* by using the same operator

name. Figure 3 shows the definition of a Sobel operator with a 3×3 convolution mask and derivation in x- direction by using the instruction set for custom operators.

```
<image_operator_class>
<name>Sobel</name>
<class>Edge</op_class>
<generic>
  <mask>3x3<mask>
</generic>
<op_description>
  <![CDATA[
    SET address startaddress
    ADD address1 address width
    ADD address2 address1 width
    ADD address3 address2 width
    ADD address4 address3 width
    ADD address5 address4 width

    LOAD p1 portin address  #I(x+0,y+0)
    LOAD p2 portin address1 #I(x+0,y+1)
    LOAD p3 portin address2 #I(x+0,y+2)
    LOAD p4 portin address3 #I(x+2,y+0)
    LOAD p5 portin address4 #I(x+2,y+1)
    LOAD p6 portin address5 #I(x+2,y+2)

    ADD x1 p1 p2
    ADD x1 x1 p3
    SUB x1 x1 p4
    SUB x1 x1 p5
    SUB x1 x1 p6

    STORE x1 address
    ]]>
  </op_description>
</image_operator_class>
```

Fig. 3: IPOL custom code for image processing algorithms

The custom code has a similar structure to common assembler code. Its compiler is part of the IPAS framework. For custom code following instructions are supported by the IPOL language: LOAD, STORE, SET, ADD, SUB, MUL, DIV, POW, SIN, COS, IF. With this instruction set a custom design for small image processing operators or an extension of existing image processing operators can be done. It makes the design process more flexible and constrained in a wider manner. An image processing application can use more algorithms than just well-known algorithms predefined in the libraries of the IPAS framework.

4.2 The IPAS Synthesis

Using the IPOL language for describing an image processing application, predefined or custom-created algorithms can be used. These two ways are already introduced in the last subsection. For mapping an image processing application described in IPOL on a heterogeneous hardware architecture a methodology is necessary. Hence a synthesis design flow called IPAS [HHRF16] will be introduced in this subsection. IPAS stands for Image Processing Architecture Synthesis. The IPAS framework consisting of four layers at different abstraction levels which are: the application layer, the mathematical layer, the simulation layer and the implementation layer. Figure 4 illustrates the top-down design synthesis process of the IPAS framework by using the IPOL language as input and output of every layer. IPOL can be seen as a kind of interface between the layers. Every layer stores its synthesis results in the IPOL. As shown in Figure 4 IPAS uses different languages at the layers. The application layer uses SysML and XML, the mathematical layer describes the applications by using Wolfram Mathematica, the simulation layer takes the languages

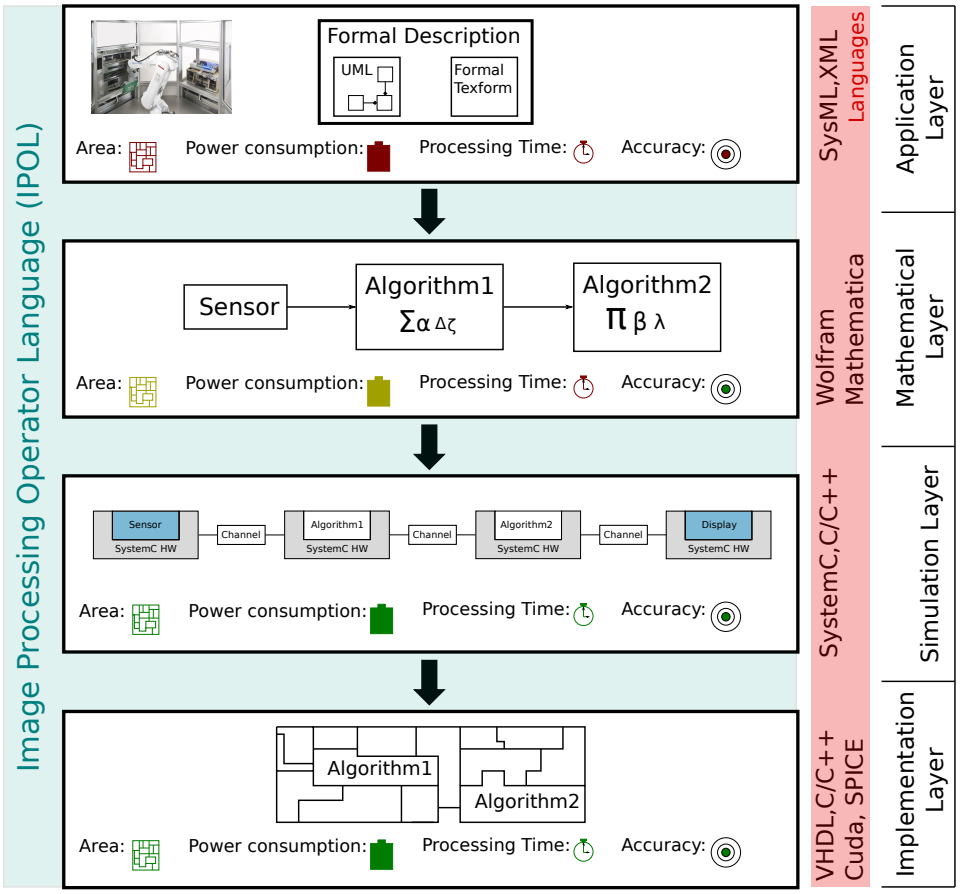

Fig. 4: IPAS: Holistic top-down design flow for image processing [HHRF16]

SystemC and C/C++ and the implementation layer uses VHDL, Verilog, C/C++, Cuda and SPICE. With the IPOL description we are able to describe an image processing application in one language which is the IPOL an XML-based DSL. A XML parser [Xer17] is used to detect IPOL syntax inside a XML description. The XML parser is coupled to a XSD and XSLT scheme to read/write information out/in the IPOL file. In further steps the IPAS framework uses uses compilation rules written in XSLT to transform IPOL code to C/C++, Cuda, Mathematica or a VHDL description. XSLT is used for compiling IPOL described applications to a layer-specific code. For the transformation of the IPOL code in executable code a predefined subset of compiling rules must exist. Figure 5 shows the methodology of the image processing transformation. In Figure 5 a custom defined-operator written in IPOL will be transfered to Wolfram Mathematica and C code. In this case the executable code can be used in the mathematical and simulation layer. Rules for other instructions or well-known image processing operators will be transferred by using the same principle. The transformation rules of IPOL to result in an executable code is the first part of the IPAS synthesis. The second part of the IPAS synthesis is the evaluation of the layer results.

Before a layer uses the transformation for generating executable code the synthesis results of the upper layer will be evaluated for making design decisions. The IPAS layers will be executed in the following order, the mathematical layer, simulation layer and implementation layer. That means the simulation layer evaluates the results of the mathematical layer before executable code will be created from the implementation layer.

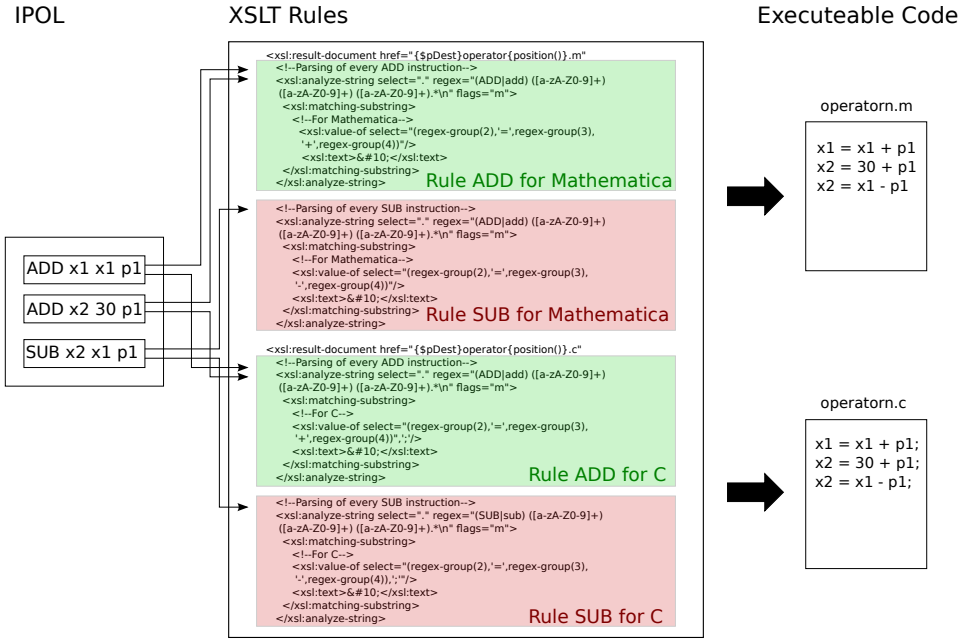

Fig. 5: IPAS: transformation rules

5 Results

For an executable system a hardware implementation is necessary. Therefore, the implementation layer of the IPAS framework is responsible. Specialized hardware such as the *Generic-IP-Library* which is a generic VHDL implementation of kernel based processing are included. This set is useful for supporting a broad pallet of specialized and efficient hardware components. Also back ends for standard processing units such as an ARM Cortex A9, Intel Core I7 and Kepler K1 are part of the implementation layer and can be addressed by using the IPOL. This section introduces the implementation layer by an example image processing application. The example application is the Sobel-Bresenham application which was already introduced in Section 3. The test input is shown in Figure 6.

The following shows the parameter set for the experiment: Algorithm: Bresenham; Image size: 1280×720; Pixel width: 24 bit; Sobel Mask: 3×3

First we start with mapping the preprocessing operator and stepping through the whole application by mapping every part of the application on different hardware components.

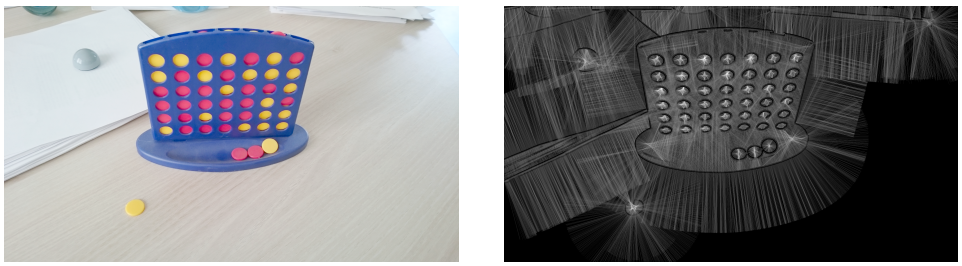

Fig. 6: Sobel-Bresenham test image. Input on the left and output on the right side

We will measure the power consumption and *pixel delay*. The pixel delay is the processing time which is needed by the architecture for one single pixel.

5.1 Generic-IP-Library (FPGA back-end)

A low power preprocessing hardware back-end of the IPAS is the IP-Library. The IP-Library is a generic VHDL library for local preprocessing algorithms on FPGAs. Stencil codes are implemented according to the full buffering scheme [SRF12] in order to avoid time-consuming external memory accesses. From the *Sobel-Bresenham* application, only the Sobel operator is implemented and tested by the *Generic-IP-Library*. For designing a filter with the *Generic-IP-Library*, predefined and generic blocks of the VHDL library will be instanced and connected. The blocks are scalable in functional parameters and can be recycled for similar filters. Sobel, Canny, Gaussian blur and LoG are all using a convolution mask. This convolution mask is a generic block included in the IP-Library and can be accessed by using the IPOL language. For the experimental results the Sobel edge detector implemented by the *Generic-IP-Library* was tested on the Zynq7020 platform. The results are listed in the following table.

Architecture	Custom Digital Design (Zynq7020)
Power Consumption	1.5 W
Pixel delay	1.8 ns

Compared to standard PCs, the power consumption of the IP-Library is very low. The *Generic-IP-Library* can be used on FPGAs and does not need a realization in silicon.

5.2 ARM Processing

Image processing applications can include more than preprocessing algorithms. In this Subsection a hardware back-end which supports a higher range of algorithms will be introduced. In the IPAS implementation layer a back-end for embedded CPUs was created; it supports the series of ARM Cortex cores. For our tests we chose the ARM Cortex A9. All parts of the image processing pipeline except the sensor are supported by the ARM core.

The main difference between the ARM architecture and the libraries as mentioned before is the memory hierarchy. The IP library supports stencil-code based processing because of the limited amount of on-chip-memory. The ARM architecture uses a L1 and L2 cache and thus brings a better performance on random memory access. In that case the ARM Cortex A9 is able to support global image processing algorithms.

Architecture	ARM Cortex A9
Power Consumption	4 W
Pixel delay	11.5 μs

With its low power consumption and small size the ARM Cortex A9 can be found in the embedded systems field and in mobile devices. The result show a high delay for one pixel and a moderate power consumption.

5.3 Embedded GPU Tegra K1

The second general processing platform architecture which is supported by the IPAS implementation layer is the Tegra K1 GPU. The Tegra K1 is a Nvidia SoC with 192 Kepler Cores and 2 GB RAM. With its low power and area consumption the Tegra K1 can be used in the embedded field. In contrast to the ARM architecture the Kepler architecture does not have a single powerful processor core. It has many small cores which are suitable for parallel processing. Such as the ARM core the Tegra K1 supports the same parts of the application. For the example application the Tegra K1 has the following results.

Architecture	Kepler K1
Power Consumption	12 W
Pixel delay	48 ns

The low volume, power consumption and the low processing time of the Tegra K1 makes an effort in the embedded field possible.

6 Conclusion

In this paper we introduced a novel DSL for image processing applications. The tests show the necessity of supporting heterogeneous systems. All of the introduced components are not able to provide a suitable solution for the whole application on its own. A combination of the components is needed. That leads to a heterogeneous system which is supported by the IPOL DSL combined with the IPAS framework. In contrast to most frameworks and DSLs in that field, IPOL does not support just stencil-code based algorithms; also global processing algorithms can be described by using known predefined algorithms from the library or own custom designed algorithms. Thus, the IPOL language is not strictly limited. It means all algorithms of the image processing domain can be supported. The underlying methodology called IPAS makes a mapping on different hardware components and the creation of a heterogeneous system feasible. The usage of the IPOL language and IPAS framework lowers the project effort with heterogeneous systems and provides efficient solutions in an easier way.

References

[CDAC13] H. Chenini, J. P. Derutin, R. Aufrere, and Roland Chapuis. Parallel embedded processor architecture for FPGA-based image processing using parallel software skeletons. pages 1 – 23, 2013.

[CLY+15] C. Chang, C. Lu, W. P. Yang, W. C. Chu, C. Yang, and P. Hsiung. A SysML Based Requirement Modeling Automatic Transformation Approach. In *Computer Software and Applications Conference Workshops (COMPSACW), 2014 IEEE 38th International*, pages 474 – 479, Vasteras, Sweden, 2015. IEE.

[HBD+14] J. Hegarty, J. Brunhaver, Z. DeVito, J. Ragan-Kelly, N. Cohen, S. Bell, A. Vasilyev, M. Horowitz, and P. Hanrahan. Darkroom: Compiling High-Level Image Processing Code into Hardware Pipelines. *Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2014*, pages 144:1 – 144:11, 2014.

[HHRF16] C. Hartmann, K. Häublein, M. Reichenbach, and D. Fey. IPAS: a design framework for analysis, synthesis and optimization of image processing applications for heterogenous computing architectures. *Journal of Real-Time Image Processing*, pages 1–16, 2016.

[HLS17] Xilinx Vivado HLS. {`http://www.xilinx.com/products/design-tools/vivado.html`}, Url = http://www.xilinx.com/products/design-tools/vivado.html, Timestamp = 2016.02.24, 2017.

[HQ11] Z. Huijuan and H. Qiong. Fast image matching based-on improved SURF algorithm. In *Electronics, Communication and Control*, pages 1460 – 1463, Ningbo, China, 2011. IEEE.

[KA95] D. J. Kerbyson and T. J. Atherton. Circle Detection Using Hough Transform Filters. In *Image Processing and its Applications*, pages 370 – 374, Edinburgh, Scotland, 1995. IEEE.

[PDH+14] M. Pelcat, K. Desnos, J. Heulot, C. Guy, J-F. Nezan, and S. Aridhi. Preesm: A dataflow-based rapid prototyping framework for simplifying multicore DSP programming. In *European Embedded Design in Education and Research Conference*, pages 30 – 40, Milano, Italy, 2014. IEEE.

[RHR+15] O. Reiche, K. Häublein, M. Reichenbach, F. Hannig, J. Teich, and D. Fey. Automatic Optimization of Hardware Accelerators for Image Processing. In *Heterogeneous Architectures and Design Methods for Embedded Image Systems (HIS 2015)*, pages 10 – 15, Grenoble, France, 2015. arXiv.

[SRF12] M. Schmidt, M. Reichenbach, and D. Fey. A Generic VHDL Template for 2D Stencil Code Applications on FPGAs. In *International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW)*, pages 180 – 187, Shenzen, China, 2012. IEEE.

[Str17] Streamit. `http://groups.csail.mit.edu/cag/streamit`, 2017.

[Xer17] Xerces. `https://xerces.apache.org/xerces-c`, 2017.