

# A Design Methodology for the Next Generation Real-Time Vision Processors

Jones Yudi Mori<sup>1,2</sup>(✉), André Werner<sup>1</sup>, Arij Shallufa<sup>1</sup>,  
Florian Fricke<sup>1</sup>, and Michael Hübner<sup>1</sup>

<sup>1</sup> ESIT - Embedded Systems for Information Technology,  
Ruhr-University Bochum, Bochum, Germany  
{Jones.MoriAlvesDaSilva,Andre.Werner-w2m,Arij.Shallufa,  
Florian.Fricke,Michael.Huebner}@rub.de

<sup>2</sup> Department of Mechanical Engineering, University of Brasília, Brasília, Brazil

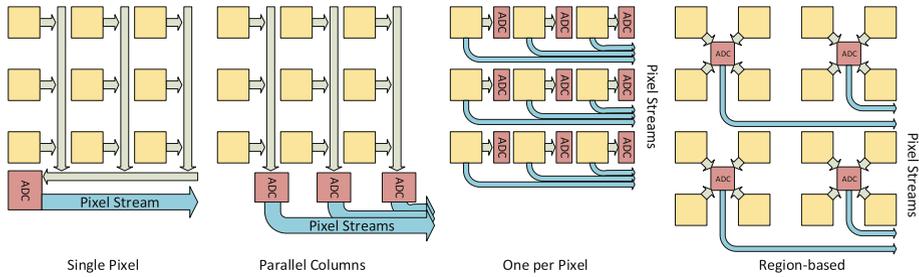
**Abstract.** In this work we present a methodology to design the next generation of real-time vision processors. These processors are expected to achieve high throughput with complex applications, under real-time embedded constraints (time, fault-tolerance, silicon area and power consumption). To achieve these goals, we propose the fusion of two key concepts: the Focal-Plane Image Processing (FPIP) and the Many-Core architectures. We show the concepts and ideas to build-up a methodology able to offer both design space exploration, and a customized programming toolchain for the final architecture. We present implementation details and results for working parts of the framework, and partial results and general comments about the work-in-progress.

**Keywords:** ASIP · Image processing · Processor architecture · Real-time

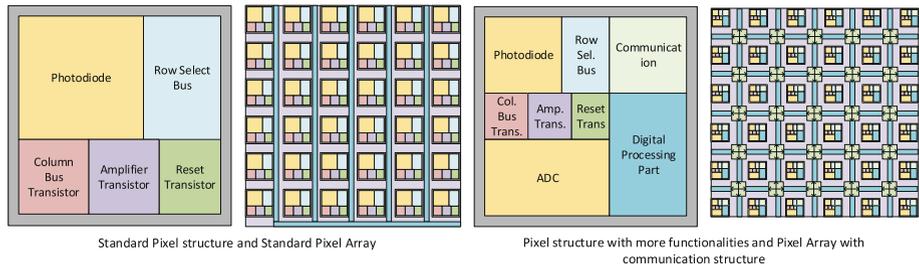
## 1 Introduction

Smart Cameras are special cameras which do not only acquire, compress and transmit images, but are capable of processing them to extract useful information. Complete IP/CV (Image Processing and Computer Vision) applications should be executable in modern Smart Cameras. With the growing of the Internet of Things (IoT) and the CyberPhysical Systems (CPS), a single device will be expected to run several complex applications simultaneously.

A real-time IP/CV system is composed by two main parts: acquisition and processing. The acquisition part is in general a standard cmos sensor array which provides a pixel stream and some synchronization signals. The main problem in standard acquisition systems is the bottleneck in the pixel stream, since the pixels are transmitted one by one [6]. The hardware architectures commonly used in the processing part (DSP concepts, VLIW, SIMD operations) are not able to achieve the constraints in throughput, fault tolerance, silicon area and power consumption [12].



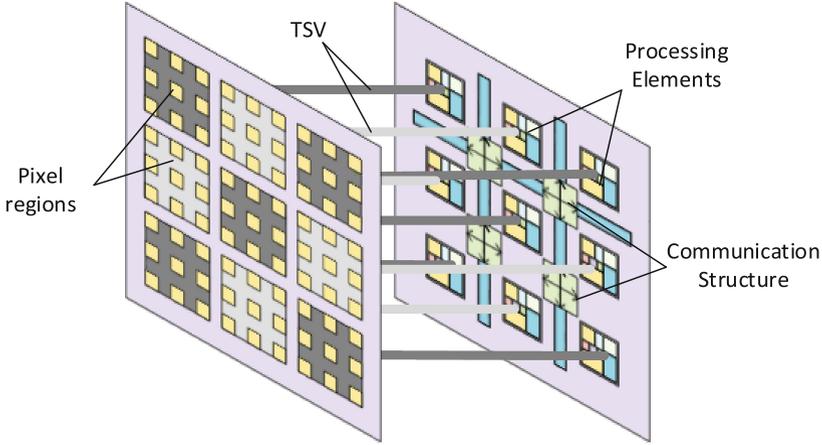
**Fig. 1.** Readout schemes in a Pixel Array.



**Fig. 2.** Fill-factor reduction when adding more functionalities to the image sensor.

To eliminate (or, at least, to minimize) the acquisition bottleneck problem, two main solutions can be found in the literature. The first one is to replicate the amplifier and ADC (analog to digital converter), adding one pair for each row (or for each column). The second solution is to have one pair (amplifier/ADC) per pixel [16]. The last option allows for full acquisition parallelism, however it is too expensive for standard cameras, being used only in scientific/industrial custom applications. To better explore the interface acquisition/processing, we propose a different way to acquire the pixels: to add the pair amplifier/ADC for regions of the sensor array. As will be explained later in the text, the configuration we propose offers several advantages in the envisioned architecture. In Fig. 1 we can see the four types of acquisition systems discussed here.

A different camera concept was created some years ago, with the aim of achieving high throughput: the Focal-Plane Image Processing (FPIP) concept is based on inserting small processing elements (PEs) close to the pixel sensors, minimizing the transmitting paths and allowing also for parallel acquisition [2, 11]. In Fig. 2 we can see the structure of single standard pixel. The fill-factor is the percentage of the chip area which is photosensitive. As can be seen in the picture, the addition of an ADC and a PE to the pixel area would reduce considerably the fill-factor, and by consequence, the image quality would be degraded. In addition, due to the limited area available, the PEs found in the literature are mostly analog filters and/or small digital ones, and do not offer too much flexibility. Figure 2 shows issues related to the communication structure which



**Fig. 3.** Envisioned hardware configuration: a 3D integration of acquisition and processing parts using TSV technology.

must be present to integrate the PEs. This structure would contribute to reduce even more the sensor’s fill-factor.

We propose a hardware configuration to integrate the acquisition and the processing parts in a more interesting architecture. Figure 3 shows the concept. The sensor array has spatially distributed pairs amplifier/ADC, each pair being responsible for a region of the image. Using the Through Silicon Vias (TSV) technology, the outputs of the ADCs are sent to an underlying processing layer. The processing layer is a manycore architecture composed by distributed pixel registers which receive the pixel stream from the ADCs. A PE is responsible to process each region, and a communication infrastructure allows for data exchange among them. With this configuration, both the acquisition and the processing parts can explore a higher amount of parallelism.

The design of such architecture is not easy, since the PEs and the communication structure must be developed with focus in the application and in the embedded hardware constraints (silicon area, power consumption, thermal distribution and so on). In addition, software related issues must be solved. The programming model must be able to explore the parallelism in the applications, considering the spatial distribution of PEs, the distributed input streams, and the synchronization and data exchange issues.

In this project we show a methodology to help the design of such system, and also to provide an efficient parallel programming model. This work provides a general overview of subprojects (complete and work-in-progress) integrated as a design methodology. Considering this, no exclusive section about literature review is provided. However, before explaining each subproject, we provide specific motivation and state-of-art. In Sect. 2 we show in details how the framework was conceived. Finally, in Sect. 3, we have a discussion about the issues and next steps of the project.

## 2 Design Methodology

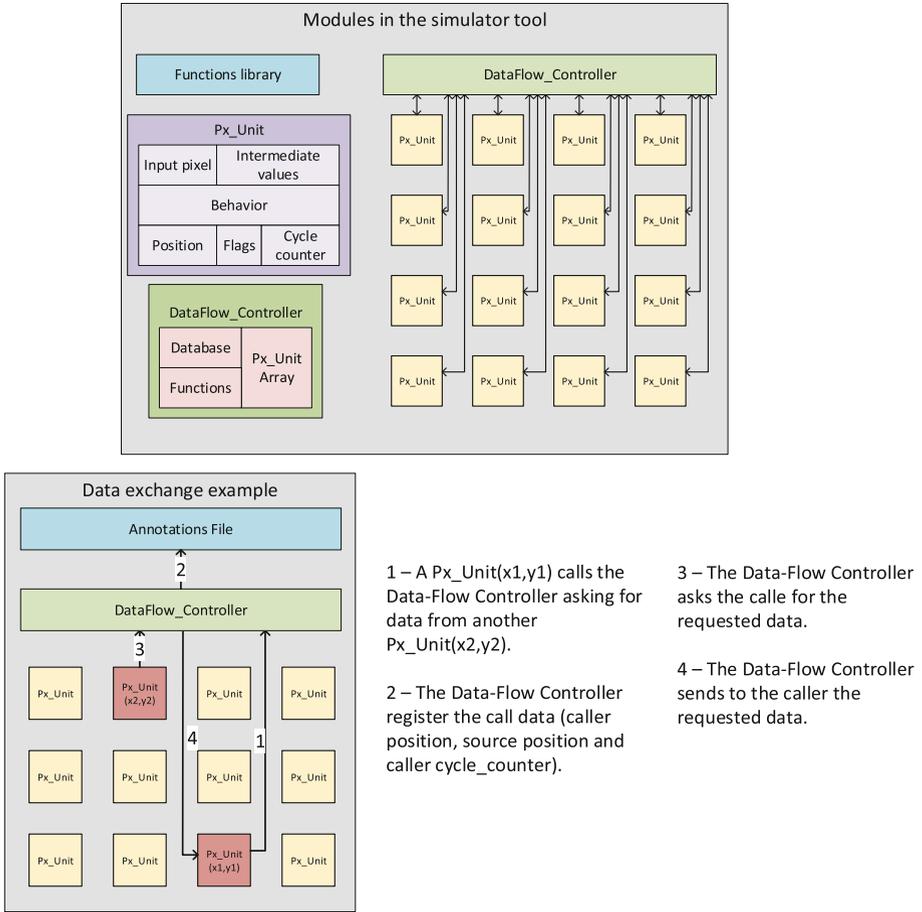
In this section we discuss the concepts and ideas behind the proposed methodology. We start the analysis with a SystemC/TLM2.0 simulator used to analyze communication patterns for different IP/CV algorithms. This analysis is used to determine geometric constraints and a rough structure for the architecture. After that, we analyze the IP/CV application domain, in order to show how the domain-specific characteristics shape and constrain the design space. After that, we show the development of some tools which help in the design decision making. The first tool performs a static analysis over the application's source code, in order to determine the algorithm's structure. Then another tool generates a SystemC/TLM2.0 model which estimates parameters to help the design of the PEs microarchitectures.

### 2.1 High-Level Analysis of Communication Patterns

To determine a starting point for the architecture design, we developed a simulation tool based on SystemC/TLM2.0 models. This tool should be able to extract communication patterns from IP/CV algorithms simulations. Figure 4 shows the main modules of the developed tool: *Px-Unit*, *Functions Library*, and *Data-Flow Controller*, described as follows:

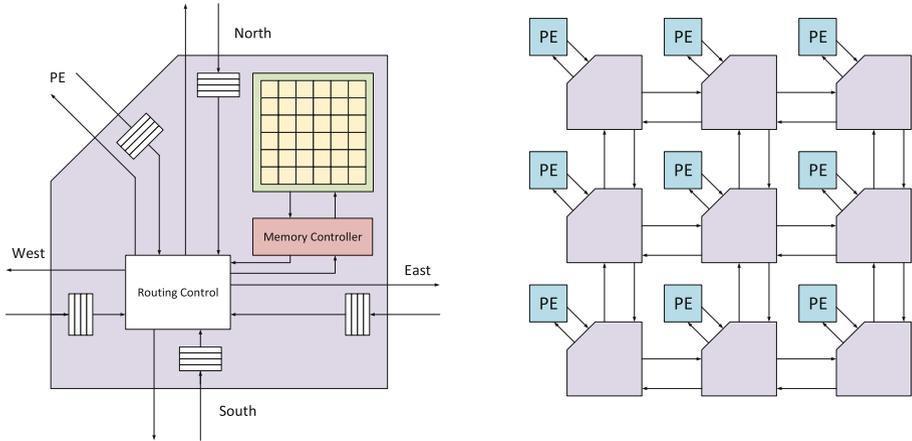
- *Px-Unit*: It is a module that represents an unit with the following content: the input pixel value; the pixel position in the image; pixel values in intermediate images; status flags; and the image processing algorithm behavior described using the *Functions Library*. It also has a cycle counter (similar to a Program Counter in processor architectures) responsible for synchronizing the *Px-Unit* with the *Data-Flow Controller*.
- *Functions Library*: It was created to implement all operations needed to compute each output pixel. It covers since simple arithmetic operations until calls for memory access. The algorithm behavior of *Px-Units* is implemented using these functions, as a common programming library.
- *Data-Flow Controller*: It is a module created to instantiate an array of *Px-Units* representing the Pixel Array. It is responsible for writing the input values inside each *Px-Unit*. When a *Px-Unit* needs a pixel value from another one, it calls access functions from the *Data-Flow Controller* which will register this call (the instant it occurs, the caller and the callee, etc.). After registering the call, the controller takes the value from the source unit and gives to the caller unit.

Figure 4 shows how the data exchange among two *Px-Units* is performed. The annotations file is the output obtained after a simulation. In the *Data-Flow Controller* block, several different parameters can be configured. We performed simulations for different topologies to determine the communication needs, bottlenecks and possible solutions. These are only rough topology models, considering the relative position among Pixels and PEs.



**Fig. 4.** Main modules of the simulation tool, and a sample data exchange among two *Px-Units*.

There are two main features to determine in this part: (a) if the Pixels will be stored internally or externally to the PE; (b) how the communication among PEs will be organized. If we choose the alternative to store Pixels internally to the PEs, all data requests from outside would cause the PE to stop processing the IP/CV algorithm, to deal with the communication tasks. Considering that each is surrounded by at least 4 neighbors, depending on the algorithm, the amount of data requests can lead the system to fail in achieving the desired throughput. From now on, we assume the Pixels in a location external to the PEs (a Register File, a Scratch-Pad Memory, etc.). Also, as the requests can come at the same time from different directions, more than one access should be possible at the same time. Considering the communication among the PEs, the action of accessing a Pixel value should be transparent to the PE. In this work



**Fig. 5.** Selected system's topology: mesh Network-on-Chip with special routers.

we consider Networks-on-Chip (NoCs) as good alternatives to solve our communication needs. NoCs are scalable, and as we have homogeneous communication patterns, their performance will be efficient enough for most applications.

Figure 5 shows the topology which offers the highest throughput. It is similar to a standard Mesh NoC [13], but with special Routers able to store internally the Pixels of an image region. In addition, this topology fits the geometric constraints defined by the Pixel Array with Region-based readout.

## 2.2 Analysis of the Application Domain

In the parallel processing domain, one of the most important goals is to identify and explore the maximum amount of parallelism possible [9]. Looking to the IP/CV algorithms, and considering the spatial distribution of PEs over the image area, we can identify a coarse-grained parallelism. In the IP/CV domain, the OpenCV library [1] is one of the most used collection of algorithms. It is used for educational, industrial and scientific purposes, and can be considered as a informal standard.

With the increasing number of complex IP/CV commercial applications, the industry identified the need for an IP/CV standard *de facto*. The Chronos Group [8] released in 2014 the first version of the OpenVX standard. OpenVX is a set of rules and design patterns created to describe IP/CV applications. Similar to other standards, like OpenCL and OpenGL, the OpenVX actuates as a *frontend* for application's description. The *backend* should be created by each hardware manufacturer, accordingly to its architecture's characteristics [8].

OpenVX defines a programming model based on graphs, composed by nodes and links. Each node is a complete IP/CV algorithm (filtering, motion detection, arithmetic operations among images, and so on) Fig. 6. In general, both input and output of a node are images. In our approach we consider that each PE is

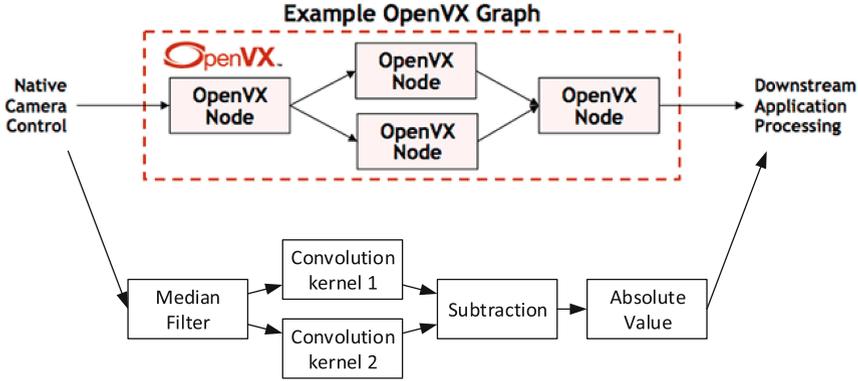


Fig. 6. OpenVX Graph of a simple application.

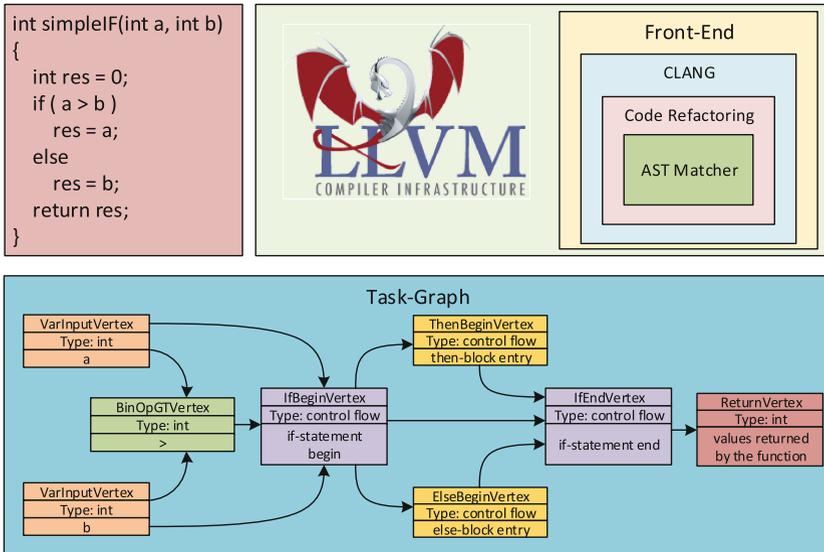
responsible for computing the output values of a sub-image. This means that the algorithms in each node must be executed by all the PEs simultaneously. We define here the concept of Core-Code: the code necessary to generate a single output pixel. Each OpenVX node can be mapped to a Core-Code. This means that each PE will execute the Core-Code repeatedly for all Pixels in its own region. Each PE operate independently from the other when performing the same node, what means that the only synchronization among PEs occurs when they start/finish a node [12].

### 2.3 Static Analysis of Application’s Core-Codes

In the last section we determined the Core-Codes as the codes to be executed in each PE. A complete IP/CV application is composed by several OpenVX nodes, what means several Core-Codes. The PEs must be designed to be able to process the sequence of nodes efficiently. There are several works in the literature regarding the design of Application Specific Instruction-Set Processors (ASIP). An ASIP is a processor architecture specially tuned for an application domain. An ASIP should be able to provide a medium term between the flexibility (programmability) of common General Purpose Processors (GPPs), and the efficiency of direct hardware implementations.

A straightforward approach was used to determine the PE’s microarchitecture from the application’s Core-Codes. A tool for graph generation from the Core-Codes was created. This tool is based on the Clang compiler from the LLVM project [10]. The starting point of the analysis is the Abstract Syntax Tree (AST), which is syntactically, semantically and type-checked. However, no optimization is performed in this step, what means that our systems depends on code-quality. To avoid issues, a library of nodes is available for OpenVX descriptions.

By the time of this work was written, this tool had some restrictions: pointer variables and arrays not supported; jump operations not supported; the function

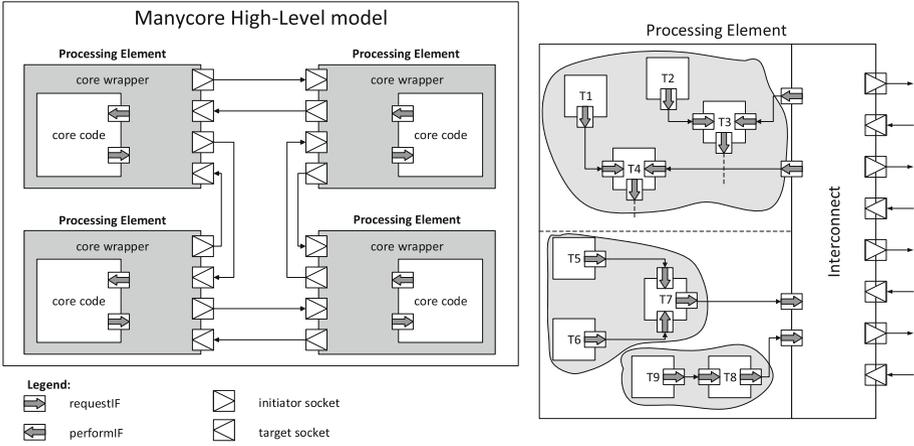


**Fig. 7.** Graph generator based on LLVM: sample example for a simple function.

may have only one return statement, on its end. The graphs generated have some properties, like: parameters and literals are inputs; vertices are basic operations of the C programming language; edges represent dependencies, data and control flow; function return value is the output. matcher, to extract the algorithm's structure and create a the graph. At this moment, this tool is able to handle most of the ANSI C language specification. For each Core-Code in an application, a new graph is generated. Figure 7 shows the graph generated for a sample Core-Code.

## 2.4 Parameterizable SystemC/TLM2.0 Simulator

This tool receives the graphs from the last Section and generates a SystemC/TLM2.0 representation of the manycore architecture. By using a library of SystemC blocks, the graphs are rebuilt and grouped in high-level models of the PEs. A *Core wrapper* handles TLM communication between partners, the transaction object contains information about the dependencies and the target generates results for the dependencies. Some advantages in the use of TLM are: PE model is separated into communication and functionality; few functions make the model easy to understand; sparsely connections for dependencies, because they are only for external communications needed. Figure 8 shows the TLM models for the PEs (with the graph inside) and the manycore organization. The simulations allow to extract in more details informations regarding the communication patterns among PEs, and also timing and resources needs inside the PEs.

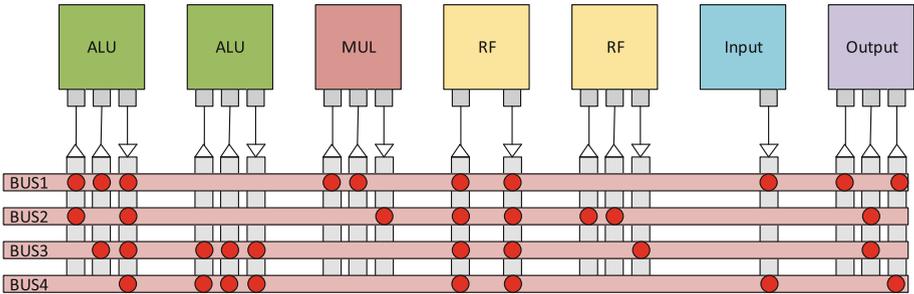


**Fig. 8.** TLM models of the Manycore architecture: communication and spatial distribution, and graph-based Core-Codes in each PE.

### 2.5 Processing Element Parameter Estimation

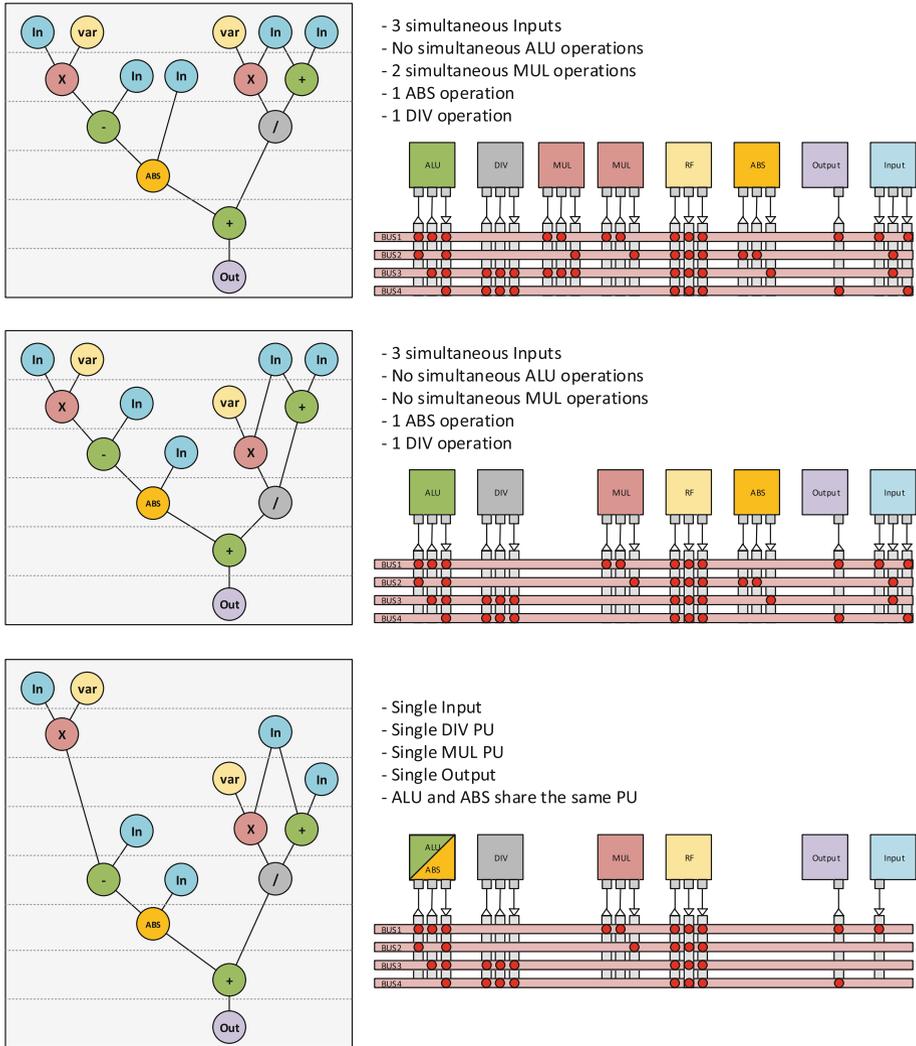
As explained previously, due to the design constraints and restrictions in area, power consumption and speed, the PEs must be specialized for the application domain we are exploring. Flexibility for changes after the chip design is important to allow for new application’s implementation over the architecture. However, specialization and flexibility are quite antagonic.

[9] states that one of the key concepts to achieve high efficiency in IP/CV processing is the parallelism exploration. IP/CV applications have different levels of parallelism. In Sect. 2.2 we explored the coarsest level, by dividing the image into regions allocated to each PE. Regarding the PE’s microarchitecture, solutions exploring the Instruction Level Parallelism (ILP) are among the most efficient.



**Fig. 9.** Sample Transport Triggered Architecture processor.

From the literature we could identify that several different architectures have been tested for IP/CV algorithms, each one with different efficiency, advantages and disadvantages. Architectures such as VLIW (Very Large Instruction Width) are strong candidates to be used in our RISC system, considering that they offer a throughput higher than the common RISC architectures, and are more flexible in comparison with embedded GPUS [7]. Hardware accelerators, like the ones generated by the LegUp framework [3] are also interesting and can be quite fast, however there is a lack of flexibility. Once the accelerators are defined,



**Fig. 10.** From the graph to the TTA configuration: same application, different hardware implementations.

new applications maybe would not benefit from these already implemented accelerators, resulting in bad efficiency.

ASIPs (Application Specific Instruction Set Processors) are considered as well-balanced solutions for embedded systems. They offer some advantages from GPPs (e.g. programmability) and from hardware accelerators (e.g. special complex instructions). In [14] the authors suggest a methodology based on graphs to help the design of processor architectures. [5] shows a method based on profiling and microarchitecture's templates to ASIP customization. The design of ASIPs (and the design of processors in general) is a well-studied field, however, most of the methodologies available relay too much on the designer's knowledge about both application and hardware features.

In our project, we proposed a simple and straightforward method to define the PE's microarchitecture given the application's Core-Code. We selected the use of Transport Triggered Architectures (TTAs) as the standard basis. TTAs are a superset of the VLIW architectures, with some special characteristics [4]. Figure 9 shows the general idea of a TTA processor. It is composed by several processing units (PUs) interconnected through one or more buses. The type of PUs determine the amount of ILP possible to be explored (as in a VLIW processor) and deep bypasses can be configured depending on the number of buses used.

Figure 10 shows an automatic method to determine the best TTA configuration. TCE is a framework for design space exploration of TTA processors [15]. It allows the designer to select all the configurations (mainly the number and types of PUs, and the number of buses). It also generates RTL descriptions and a LLVM based compiler for each configuration desired. Our method is based on a mixed ALAP/ASAP (As Last As Possible/As Soon As Possible) analysis of the Core-Code's graphs. For each graph possibility we generate a set of possible solutions in the TCE environment, comparing the results with the design constraints. This cycle is repeated until the design meets the design fits under the design constraints.

### 3 Discussion and Conclusion

In this work we discussed the concepts and ideas behind the development of the next generation vision processors. A methodology for the design of such processors was explained and some parts were detailed.

The project presented in this work is currently under development. Many issues are still not solved, but the results achieved until now are promising. The design methodology is already able to generate a rough model of the many-core vision processor. Optimizations must be done in all subprojects and the development of the Analog part (Acquisition and Readout) is not under development yet.

**Acknowledgment.** The authors would like to acknowledge CAPES Foundation/Brazilian Ministry of Education (Science without Borders Program, Grant Process Nr. 9054-13-8) and the support received from the University of Brasilia.

## References

1. OpenCV: Open source computer vision. Technical report. [www.opencv.org](http://www.opencv.org)
2. El Gamal, A., Fowler, B.A., Yang, D. X.: Pixel-level processing: why, what, and how? In: Proceedings of the SPIE, Sensors, Cameras, and Applications for Digital Photography, vol. 3650 (1999)
3. Canis, A., Choi, J., Aldham, M., Zhang, V., Kammoona, A., Czajkowski, T., Brown, S.D., Anderson, J.H.: LegUp: an open-source high-level synthesis tool for FPGA-based processor/accelerator systems. *ACM Trans. Embed. Comput. Syst. (TECS)* **13**(2), 24 (2013)
4. Corporaal, H.: *Microprocessor architectures: from VLIW to TTA* (1997)
5. Eusse, J., Williams, C., Leupers, R.: Coex: a novel profiling-based algorithm/architecture co-exploration for ASIP design. In: 2013 8th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), pp. 1–8, July 2013
6. Fossum, E.R., Kemeny, S.: Camera on a chip. In: *The World and I*, pp. 178–185 (1996)
7. Hoozemans, J., Wong, S., Al-Ars, Z.: Using VLIW softcore processors for image processing applications. In: Proceedings of the 15th International Conference on Systems, Architectures, Modeling and Simulation (SAMOS) (2015)
8. Openvx 1.01 specification. Technical report (2015). <https://www.khronos.org/openvx/>
9. Kehtarnavaz, N., Gamadia, M.: Real-time image and video processing: from research to reality. *Synth. Lect. Image Video Multimedia Process.* **2**(1), 1–108 (2006)
10. Lattner, C., Adve, V.: LLVM: a compilation framework for lifelong program analysis & transformation. In: 2004 International Symposium on Code Generation and Optimization, CGO 2004, pp. 75–86. IEEE (2004)
11. Mori, J., Huebner, M.: A high-level analysis of a multi-core vision processor using systemC and TLM2.0. In: 2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig), pp. 1–6, December 2014
12. Mori, J.Y., Llanos, C., Huebner, M.: A framework to the design and programming of many-core focal-plane vision processors. In: 2015 International Conference on Embedded and Ubiquitous Computing (2015)
13. Sepulveda, M., Diguët, J.-P., Strum, M., Gogniat, G.: NoC-based protection for SoC time-driven attacks. *IEEE Embed. Syst. Lett.* **7**(1), 7–10 (2015)
14. Trajkovic, J., Gajski, D.D.: Custom processor core construction from C code. In: 2008 Symposium on Application Specific Processors, SASP 2008, pp. 1–6. IEEE (2008)
15. Viitanen, T., Kultala, H., Jaaskelainen, P., Takala, J.: Heuristics for greedy transport triggered architecture interconnect exploration. In: 2014 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES), pp. 1–7, October 2014
16. Zarandy, Á.: *Focal-Plane Sensor-Processor Chips*. Springer, New York (2011)



<http://www.springer.com/978-3-319-30480-9>

Applied Reconfigurable Computing  
12th International Symposium, ARC 2016 Mangaratiba,  
RJ, Brazil, March 22-24, 2016 Proceedings  
Bonato, V.; Bouganis, C.; Gorgon, M. (Eds.)  
2016, XIV, 370 p. 195 illus. in color., Softcover  
ISBN: 978-3-319-30480-9