Dataflow design of a co-processor architecture for image processing

Richard Thavot*, Romuald Mosqueron*, Mohammad Alisafaee*,
Christophe Lucarz*, Marco Mattavelli*, Julien Dubois†, Vincent Noel‡
*Ecole Polytechnique Fédérale de Lausanne (EPFL), GR-LSM, CH 1015 Lausanne Switzerland
Email: richard.thavot@epfl.ch
†Université de Bourgogne, Laboratoire LE2I, 21000 Dijon, France
‡AKAtech SA, Ecublens, Switzerland

***Abstract*— This paper presents the comparison of two design methodologies applied to the design of a co-processor dedicated to image processing. The first methodology is the classical development based on specifying the architecture by directly writing a HDL model using VHDL or Verilog. The second methodology is based on specifying the architecture by using a high level dataflow language followed then by direct synthesis to HDL. The priciple of developing a dataflow description consists on defining a network of autonomous entities called actors, which can communicate only by sending and receiving data tokens. Each entity in the process of consuming and generating data tokens performs completely independent and concurrent processing. A heterogeneous platform composed by a SW processor and the designed HW co-processor is used to compare the results of the designs obtained by the two different methodologies. The comparison of the results shows that the implementations based on the dataflow methodology, not only can be completed with an important reduction of design and development time, but also enable efficient re-design iterations capable of achieving performances, which are comparable in efficiency to design obtained by hand written HDL.**

I. Introduction

Since the 70's, the design of digital systems has known a continuous evolution. The technology used to realize silicon has been continually improving and thus the number of transistors available for digital design on the same silicon area has been increased. On the same line the complexity of algorithms and of circuits has followed a similar trend. Nowadays, the circuits complexity available on a chip has also generated another phenomenon. Several processing units such as processors, FPGAs and DSPs are available on the same heterogeneous platform. Nowadays, one of the major challenges is how to exploit all the processing resources available on such platforms for implementing complex applications, but using only limited developments and design resources. In other words, design productivity and efficient usage of the platform processing resources are the fundamental challenges of current and next generation designs. VHDL was introduced 20 years ago to simplify the design of the logic circuits by raising the abstraction layer avoiding the designer to work at the gate level. The dataflow methodology described in this paper, is based on CAL language[1] and on the synthesis of HDL directly from the dataflow model abstraction layer. Its introduction has exactly the same objective: raise the abstraction layer of a design. Thus, the designer should not have to care about most of the low level implementation issues present in VHDL or Verilog, but rather focus on higher level architectural issues such as how efficiently dataflow trough the different architecture components and how to partition and map the algorithm/processing elements on the different components of actual heterogeneous platforms. The aim of this work is to show how CAL design methodology can lead to efficient hardware designs within a shorter development time and lower design resource usage than classical HDL methods. In doing so, the paper introduces the essential concepts and elements of the new dataflow methodology based on writing networks of CAL actors [1], [2], [3] and compares the results of a design case using the new approach and the classical development at VHDL level.

A heterogeneous platform composed by a SW processor and a HW co-processor is used for the comparison. An image processing application partitioned into the SW and HW components is designed using CAL and finally compared to the implementation obtained by a classical HDL approach [4] [5] [6]. The main novelty of the approach is the possibility of specifying both SW and HW components, using the same language CAL, and then to generate automatically VHDL or Verilog at RTL level. Different versions of CAL models have been developed in order to explore the achievable performances of the automatic HDL generation tool.

The paper is organized as follows. Section II presents the dataflow concept and the "modus operandi" of the CAL language. Then, the heterogeneous platform (i.e. the smart camera) and the co-processor unit are presented in details in section III. The CAL dataflow model of the co-processor is presented in section IV. The implementation results of the different versions of the CAL dataflow models and the comparison with the hand written version in HDL are presented in section V. Finally, conclusions and future works are reported in section VI.

II. Modeling dataflow systems using CAL

There are many applications that fit well the semantics of dataflow systems. An example is multimedia systems with flowing streams of data within processing blocks. Developing true dataflow models of such systems using general purpose

programming languages or hardware description languages is possible. However, the genericity of concepts and operators of these languages make the description of the models more complicated. This implies that the models are harder and more time-consuming to create and manipulate. It may be better if they were modelled directly using a specific dataflow language.

A. CAL language

CAL Actor Language is a language based on the Actor model of computation for dataflow systems. It provides many natural concepts to facilitate modeling of those systems [1]. A dataflow model expressed in CAL is composed by a set of independent "actors" and their connection structure. It makes a network of actors. An actor is a stand alone entity which has its own internal state represented by a set of state variables and it performs computations by firing actions. It has a set of input and output ports through which it communicates with other actors by passing data tokens. An actor must have, at least, one action to do computations. Actions execute (or fire) based on the internal state of the actor and depending on the availability and values of tokens at the input ports. An action may consume tokens from inputs, may change the internal state of the actor, and may produce tokens at the outputs. Action execution is modeled as an atomic component which means that no other action, of the same actor, can execute while an action is executing or interrupting any executing action. CAL provides scheduling concepts to control the executions order of actions inside an actor. CAL actors can be combined into a network of actors to build larger systems called network of actors. This is achieved by connecting the input and output ports of actors together to define the communication structure of the model (Figure 1). These communication channels are constituted by FIFOs, which in the CAL computation model have infinite size. Using CAL, designers can only focus on the model-

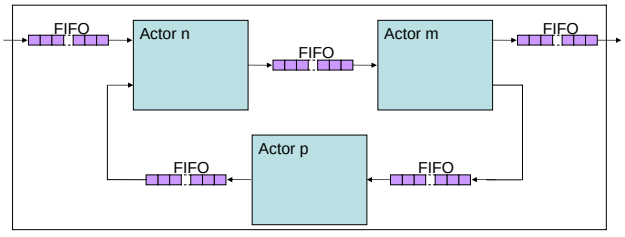

Fig. 1. A dataflow network with actors connected by FIFO channels.

ing of the dataflow system (actors and their communication topology) and do not need to care much about the low level of details to implemente the communication between actors (i.e. message passing protocols, queues, ...). The underlying computation model, simulation and synthesis system take care of all communication driven issues. However, it also provides to the designer the control over communication parameters such as length of queues and types of exchanged data. In this paper, we focus only on the issues related to the development of a CAL model of HW accelerators and on the results of the implementations.

B. Workflow for CAL-based designs

One of the current challenges of designing embedded systems composed by mixed SW and HW components is the difficulty and the design efforts needed for specifying, modeling and implementing complex signal processing systems on a heterogeneous platform. CAL addresses this issue by unifying the hardware and software design and implementation process in a single flow. In a CAL-based design flow, the whole system is modeled and implemented in CAL. After that, designers can decide on HW/SW partitioning for the final implementation. A subset of the model can be used to generate synthetizable HDL code. The generated code can also be combined with existing HDL designs. Software can be generated in a similar manner based on the partitioning decision[3]. In such workflow, the partitioning between hardware and software can be easily modified since the same source is used for generating both parts. Figure 2 shows the complete CAL design flow for the implementation of a smart camera platform. It is composed of a general-purpose processor for running SW modules and a FPGA platform including specialized hardware.

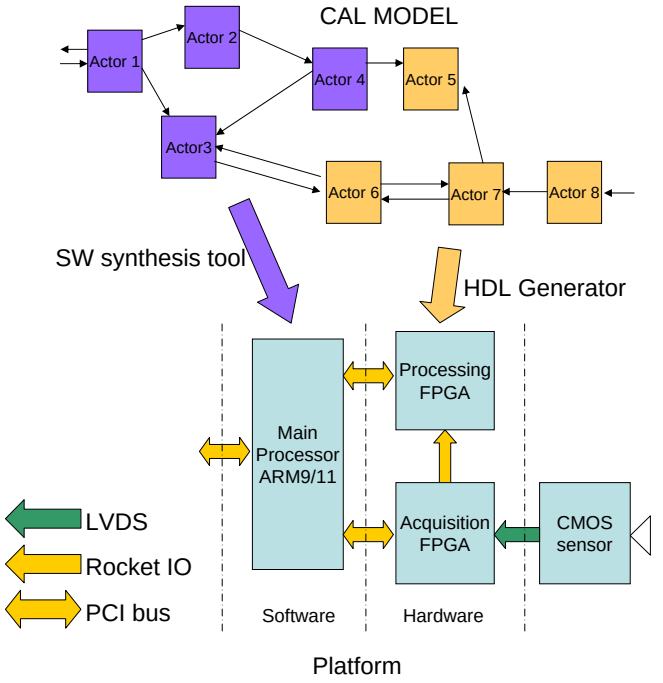

Fig. 2. Graphic representation on how partitions of the CAL dataflow models can be mapped on a heterogeneous SW and HW platform using synthesis tools.

III. Smart Camera Platform

In this work, the test platform is a "smart camera" based on an embedded HW/SW co-processor designed and developed in a previous work [4]. Cameras with embedded co-processors enable the implementation of more powerful processing due

to the high degree of flexibility and to the clear task separation between the different units. The efficiency and the processing tasks have been tested and validated implementing a real application. This application is the detection and decoding of bar codes in a postal sorting application [5], described in more details hereafter. The whole co-processor has been specified and designed manually in VHDL. The high level of performance of the co-processor have been obtained exploiting the potential parallelism at the different stages of the processing. The communication and task controllers are rather complex due to the large variety of implemented functionalities. Therefore, obtaining an efficient model for direct HW synthesis by means of a high level dataflow description represents a real challenge. The system infrastructure of the SW and HW platform is presented below. The platform is composed of an embedded frame-grabber and is equipped, at different levels, of a processing unit for the image captured by the sensor. Figure 3 illustrates the main architectural components of the smart camera with embedded co-processor (Xilinx FPGAs) and processor (Nexperia). Two FPGAs are used to acquire and pre-process the image coming from the camera sensor. The main processor is in charge of the high-level processing tasks. The co-processor deals with the acquisition, pre-processing tasks specific to the application, and the lower-level tasks. These latter are characterized by processing regularity and a high level of parallelism. Figure 3 illustrates the main architectural components of the smart camera with the embedded co-processing stage.

In this section, the platform communication infrastructure is

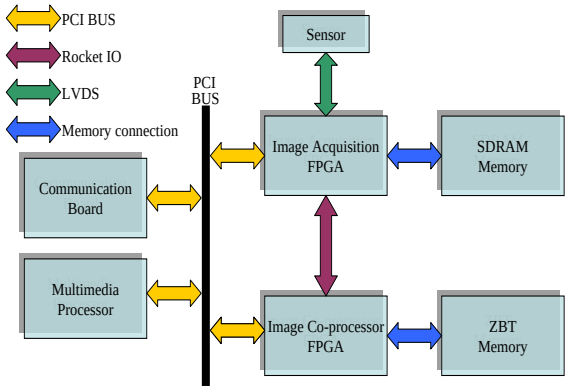

Fig. 3. Simplified smart camera description.

described showing that when building a CAL dataflow model it is not necessary to redefine the implementation of the existing hard-wired communication interfaces and busses.

A. The communication between the different platform components

The communication infrastructure in data dominated systems is an essential part in the development of an embedded system. The design choices aim at obtaining the largest achievable bandwidth between the four components of the platform. In developing the CAL dataflow model, ”Core” or ”drivers” already written by the component vendors (Xilinx and Nexperia in this case) have been used. In this way, a reduction of the development time and resources were achieved. The components of the communication infrastructure are:

- Sensor => acquisition FPGA,
- Acquisition FPGA => pre-processing FPGA,
- Acquisition FPGA => processor,
- Pre-processor FPGA => processor.

The communication between the sensor and the acquisition part is specific for each image sensor. Consequently, the interface must be built accordingly. The connection between the acquisition part and the processing part is standard and independent from the sensor. In the described design case, the connection between the co-processor and the processor is implemented with a standard PCI bus. Hence, the co-processor is independent from the processor and could be used as embedded IP with any PCI system. The co-processor architecture can achieve full data rate transfer on the PCI bus. The communication between the two FPGAs is either a PCI communication (same bus and the processor master this communication channel) or a RocketI/O connection (serial high-speed connection specific to Xilinx component).
In FPGAs, all communications are built with the Core Generator tool of Xilinx, which yields optimized designs. In the processor, a driver is included in the dedicated component library. Thus, it is not necessary to redefine and to implement again these parts of each component in the CAL model.
For such reasons, only the core architecture of the co-processor FPGA is addressed in this design case study. For performance reasons, the pixels of the image from the sensor are transferred in words of 32 bits.

B. Co-processor description

The Co-processor is composed of four components as indicated in Figure 4. These components are:

- The interface with the configuration memory
- The co-processor manager
- The external memory controller
- The processing modules

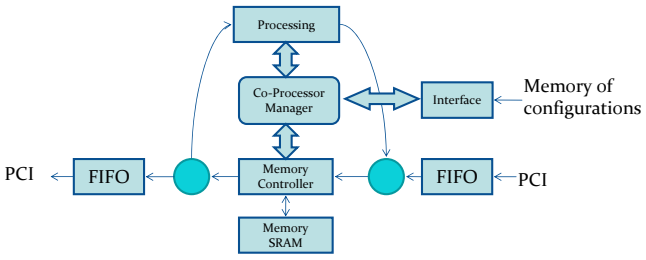

Fig. 4. High level architecture of the co-processor.

The interface connects the configuration memory and the co-processor manager. The configuration defines: the image size, the processing tasks and the start address in the external

memory where the results of the image processing are stored. The co-processor manager deals with the whole system by controlling all the components. The memory manager accesses the external memory to acquire an image according to the processing. The processing module performs the actual processing according to the desired configuration.
The processing modules that have been developed by writing VHDL are:

- Median filter 3x3,
- Transpose,
- Adaptive local binarization,
- High pass filter 11x1,
- Dilation 31x1,
- Sub sample by 4 in width and by 4 in height.

C. Bar code reading application

The postal sorting is a real-world example chosen to show the processing possibilities and the achieved level of parallelism of the system. The goal of this application is to detect and decode bar codes on letters, as shown in Figure 5, to enable automatic sorting at different stages of the logistic postal letter handling. This application is a good example of usage of all the processing possibilities of this heterogeneous platform. Some processing tasks are implemented by the co-processor and the others by the processor.
This section describes how the bar code is decoded. Different

Fig. 5. Application of the reading of a bar code.

processing steps are necessary to complete the process. In sequence, a bar code is detected by applying: a transposition, a high pass filter, a dilation, a sub-sampling, a "blobbing" and finally a decoding on the area where the bar code has been detected. The first processing stage is a transposition. The transposition rotates the image acquired line by line vertically of 90 degrees. As described in [6], a transposition is necessary because the other processing stages are specific to a horizontal reading. The first processing is a high pass filter. The high pass filter deletes the background and raises the white bar code. The resulting image is stored into the memory, but two others processing tasks are built into the FPGA co-processor. The two pre-detection tasks are the dilation and the sub-sampling. The first step dilates the bars of the bar code to build white areas. This step is necessary later to correctly identify the bar code location within the image. The second step reduces the image size without changing its content relevance. Then, the small image obtained is sent to the processor using less bandwidth on the PCI bus. These four tasks are all executed by the co-processor. The "blobbing" executed by the processor consists in locating the two or three largest "white" areas in the small image which provide the location of the bar codes. A command is sent to the co-processor to send only the regions determined by the blobbing. These regions are taken into the image without background stored previously into the memory. In order to decode the bar code, the processor executes several 1-D FFT on lines oriented in different directions. In reality, only the part delimited by the rectangular region is read as illustrated in Figure 5.

IV. Co-processor dataflow model

In this work, we want to compare the implementation of a subset of the smart camera platform in VHDL with its equivalent implementation in CAL. Two versions of the coprocessor implemented in CAL are used for the comparison. The first design is a exact transposition of the VHDL architecture into a CAL dataflow architecture. The comparison of this design with the VHDL version provides the information on how the CAL toolset is efficient in generating HDL code from a given dataflow architecture. In the second design, the coprocessor is a complete redesign of new architecture at CAL abstraction layer providing the same functionality of the VHDL model and the original design. Such new design exploits the properties of the CAL to HDL compiler toolset to reduce the on-chip area of the original design. Developing a CAL design by successive architectural refinements can be performed much more quickly compared to modifying a HDL design.

A. CAL co-processor design of the handwritten HDL architecture

This section describes the architecture of the original co-processor design (illustrated in Figure 6) and how it has been transposed into the CAL dataflow model. As shown in

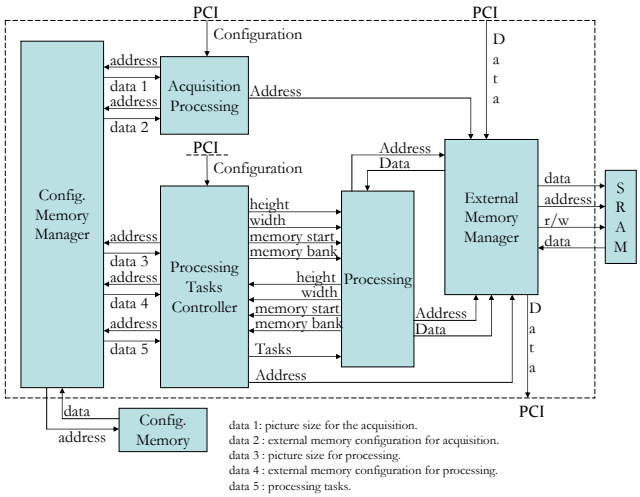

Fig. 6. CAL Dataflow HDL architecture of the co-processor.

the picture, "Acquisition Processing", "Configuration Memory Manager", "Processing Task Controller", "Processing" and "External Memory Manager" are architectural components that have been fully specified in CAL.
Processing tasks have been created to support the described

application. Such processing tasks written in CAL are : High Pass Filter, Dilation, and Transpose. Their implementations in CAL respect the same architecture of the original design with just minor modifications to I/O ports. These actors are not explained hereafter.

The dataflow description of the co-processor is reported in Figure 6. The Acquisition Processing actor receives its configuration by the processor via the PCI bus. Then, it produces two addresses which are used to retrieve data in the configuration memory. When it receives *data 1* and *data 2* coming from the configuration memory, it produces addresses to store the image into the external memory. *data 1* is the image size and *data 2* contains the information about the localization of the read/write operation in the external memory (addresses). When the acquired image is completely saved in the SRAM, the Processing Tasks Controller actor receives its configuration data sets. Like the Acquisition Processing actor, it fetches *data 3, data 4* and *data 5* into the configuration memory. Then, it sends the list of the selected tasks to the actor processing and their associated configurations. Afterwards, the Processing Tasks Controller actor waits for the updated configurations produced by "Processing" actors. If number of processing tasks are realized, the resulting image is sent via the PCI bus. The Configuration Memory Manager and External Memory Manager actors are only used to switch the data or the addresses towards the right actors.

The subsections below explains in details behavior of the different actors illustrated in Figure 6.

1) "Acquisition processing" actor: The actor "Acquisition Processing" is illustrated in Figure 7. Its function is to interpret the configuration sent by the processor via the PCI bus. These configuration data sets are stored temporarily into the Config. Memory. Configuration data are the size of the image and its place into the physical memory. In details, the "Acquisition Task Convertor" actor selects the right configuration data in the Config. memory thanks to the "Data Requester" actors. The "memory start convertor" and "Picture Size Convertor" actors send the following tokens to the "Address Generator" actor : *Height, Width, Start Address* and *Memory Bank*. This latter actor generates also addresses used to store the image into the external memory.

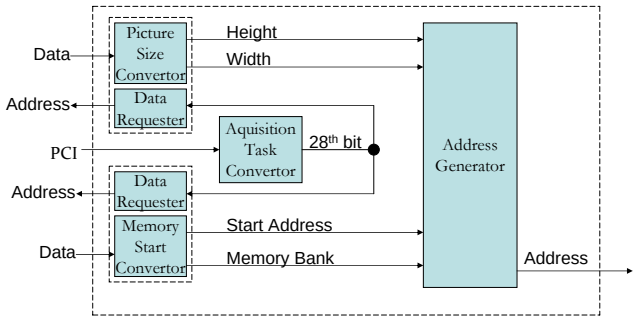

Fig. 7. Dataflow model of the "Acquisition Processing".

2) "Processing tasks controller" actor: The configuration of the processing which is sent by the processor is obtained with the same principle than the previous actor. These configurations are: size of the resulting image, place in the physical memory of the result, list and number of processing tasks.

The four "Processing Tasks Manager" actors manage correctly the *Height, Width, Address Start* and *Memory Bank* parameters in function of the number of processing tasks. These actors communicate with the actor "Processing", described hereafter, via wires "underway" and "new". "underway" is the current value of each parameter and "new" the new parameters sent by "Processing" actors. Once processing tasks are finished, the final step is to send a resulting image to the processor. This last step is processed by the "Processing Manager" and "Address Generator" with the last configuration values.

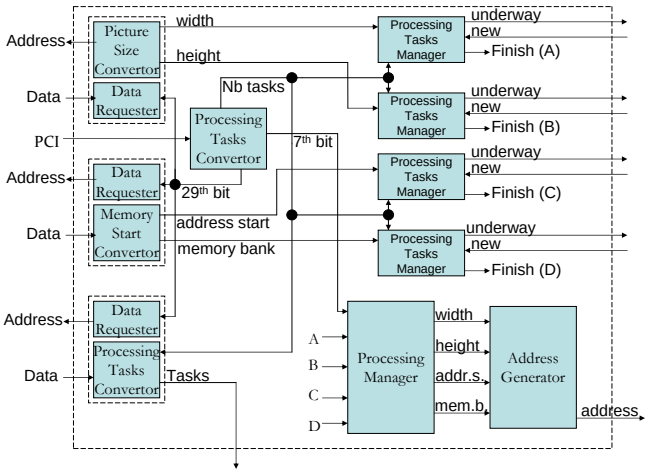

Fig. 8. Dataflow model of the "Processing Tasks Controller".

3) "Processing" network: "Processing" is a network of actors that performs various processing tasks on the images (Figure 9). As explained above, this network receives the current parameters into "Processing Manager" actors. Only one "Processing manager" actor reacts in function of the selected processing tasks. After, the right "Processing" actor (or network) receives the parameters and execute its task. When the processing is finished, parameters are updated and sent to the "Processing Task Controller". Each processing operation in this network (denoted by "Processing N") can be a single actor or a network of actors. In this design there are three processing operations: Transpose, High pass filter, and Dilation. These operation are partitioned into multiple actors which allow to perform actions in parallel.

4) "External Memory Manager" and "Configuration Memory Manager" actors: The "External Memory Manager" actor implements two functionality: read and write. The write function generates the addresses and the data. The read function generates the addresses and the data according to addresses shunting. The actor "Address Convertor" provides the information if the address is in read mode or in write mode. The "Configuration Memory Manager" actor implements a similar

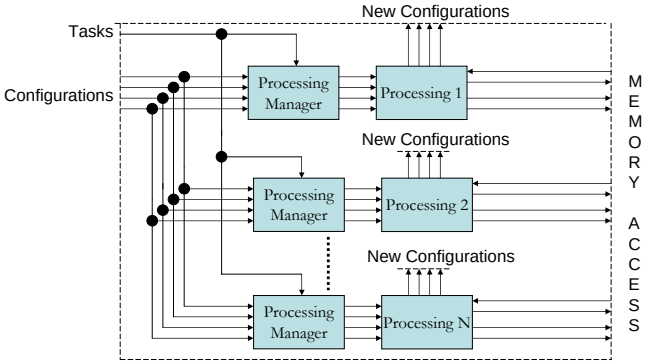

Fig. 9. Dataflow model of the "Processing".

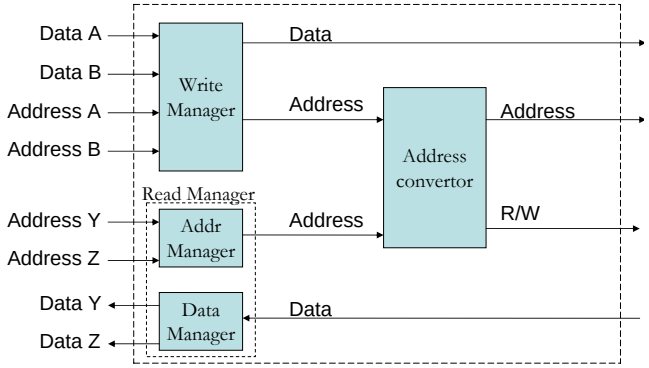

Fig. 10. Dataflow model of the "memory manager".

functionality which is the read part of the "memory manager" actor.

B. Redesign of the coprocessor

Beside the transposition of the architecture written from VHDL in CAL, described in the previous sections a new architecture has been completely redesigned directly in CAL with no more correspondences with the original VHDL reference architecture (Figure 11). Groups of actors that operate sequentially and do not benefit much from parallelism are merged into a larger actor. Such actors have almost the same functionality of all actors of the transposed architecture. In the new design, actors "Acquisition Processing", "Configuration Memory Manager", and "Processing Task Controller" are merged into a single actor, "Controller". This actor deals with commands sent via the PCI bus, reads and interprets configuration data stored in the configuration memory and controls the processing actors. It initiates appropriate processing operations and collects and stores results of processing. The actor "Memory Controller" provides the interface to the SRAM memory and arbitrates access requests made by various sources. A processing network includes three processing operations: High Pass Filter, Dilation, and Transpose. As above, each of these networks are redesigned by merging some of their actors into larger actors and thus the number of actors in each network is reduced compared to the original transposed design described above. However, each processing operation implements the same functionality and it is equivalent to the original design. One objective of the new data flow redesign, where actors are merged into larger actors is to achieve better tradeoffs between area and throughput. Such possibility is achievable at the level of the CAL design. Since each actor when converted to HDL has an area overhead for reset circuitry, internal finite-state-machine, input and output token queues and circuitry for the communication channels, a non negligible resource overhead is required when too many actors are instantiated. If such actors do not usually work in parallel there is no throughput penalty in merging them and saving silicon area. This usually holds when dealing with many small actors with a small number of actions. On the other hand, due to the fact that actions of an actor cannot be executed in parallel, it must be considered that throughput could decrease as the number of actions in an actor increases. This suggests that for those parts of the system that mostly work in sequence and do present little or no parallelism, it is better to use a fewer number of actors to save resources. For parts with a relevant amount of parallelism is it better to use a higher number of actors to achieve higher throughput. The fact that such trade-offs can be developed at CAL level using a compact high level representation, constitutes a very attractive feature of CAL based design.

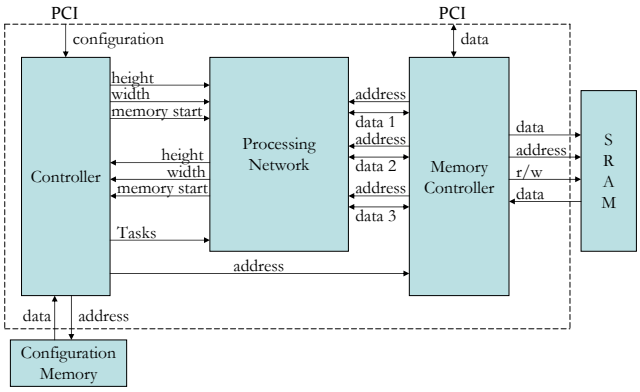

Fig. 11. New CAL Dataflow architecture.

V. Results and Performances comparison

In this section, the results of the synthesis in HDL and RTL of the models of the architecture developed in CAL dataflow language are reported and compared with the hand written model.

A. Results of dataflow language

When the dataflow model has been developed, it is simulated using the Opendataflow simulator [7] to check for the correct functionality. Its equivalent HDL code is generated using the tool described in [2] and then it is synthesized. The results of the synthesis is reported in Tables I and II for the original and revised design. Several variant configurations for the coprocessor have been tested to explore the performance

of the conversion tool and the appropriateness of the modeling methodology. Initially, a version of the co-processor without image processing functions is reported. This includes only functionality required for scheduling and dataflow controls which are highly adapted to CAL framework. As a result, both design have better area performance than the VHDL design while both preserve the same processing throughput.
In the redesigned coprocessor, the hardware overhead is reduced compared to the original design. This is due to the lower number of actors in the CAL model, since each actor instantiation implies overheads in the synthesized hardware (i.e. fifos and handshake protocols for each data token connection).
In terms of frequency and processing time, the maximum achievable frequencies for the original and the revised design are 90 MHz and 100 MHz respectively. The processing time at these frequencies for a test image of 1712×180 is about 0.85 ms and 0.77 ms. Both designs present a processing time of 1.54 ms at 50 MHz which is the working frequency for the example application.
Other scenarios experimented in this work include the usage of the processing network. In one case it includes only the high pass filter and in the other it includes the complete processing tasks (high pass filter, transpose, and dilation). Processing tasks introduce additional delay and as result the throughput is reduced compared to the scenario with no processing. The same considerations can be seen with the whole processing used in the design case application. For the original design the clock frequency is the same for the three scenarios. The revised design has higher clock frequency in the two first scenarios and has a frequency equal to the original design for the scenario with full processing. Nevertheless, all clock frequencies are completely in accord with the specification of the application. Processing delays for the original design are 3.57 ms at 50 MHz and 1.92 ms at 90 MHz and for the revised design are 7.30 ms at 50 MHz and 4.06 ms at 90 MHz. Throughput comparison between the two designs shows that both achieve the same performance in the first comparison scenario, but the original design performs better in scenarios with processing tasks included. The reason is that in the first scenario all tasks are performed sequentially and we do not benefit from having actors working in parallel. For processing operations, the potential parallelism is high so the original design which presents a large number of actors working in parallel yields a higher throughput at the cost of larger area.

B. Performances comparison

Table I, Table II and Table III reports the results of the two methodologies in terms of number of occupied slices, slice number of Flip Flops, number of four input LUTs, frequency and throughput. In these tables, the size of files that describe the same elements in VHDL and CAL is also reported.
Table I reports the results of the co-processors implemented in CAL. Table II reports the results of hand written VHDL as described in [6]. It has to be be noticed that the PCI core interface is not taken into account by the results reported in the tables. The results show that all along the development of the dataflow CAL model the hardware resources used in the FPGAs are also lower or nearly the same in the revised design than the one necessary by the handwritten VHDL design. A second interesting point is that the code size of CAL is by far smaller than the code size of VHDL with a factor ranging from 3 up to 10. However, even if such factors are already an excellent result, it can be noticed that supplying CAL with a library of basic functions similar to what in VHDL are Concatenation(),Bitselect() and similar low level library functions, such compactness factor can largely further improve. Moreover, CAL code is better structured and results are much easier to understand and analyze than an equivalent VHDL or Verilog design.
In terms of maximum achievable frequency, the hand written architecture remains better and consequently the achievable data throughput at the maximum frequency is higher. However, considering that the application requires processing at 50 MHz, when the two architecture work at the same frequency the throughput for the given application example is the same for both the original design and the revised design. However, in terms of HW resources, the revised design achieves better results compared to the original design and the handwritten design. Another important point is the reduction of the development time in CAL by a factor of about four compared to the hand written coding. Thus, with these results it is easy to say that the development time in dataflow is much faster than the one of a standard HDL development language.
Figure 12 summarizes the main components of a design: platform resource usage, design productivity and performance. For this application example size area is reduced in one of the case (redesigned CAL), the development time is considerably reduced by at least a factor four and the processing/data throughput is approximately the same for both methodologies. Moreover, the code size written for the same application is reduced by a factor of 3.

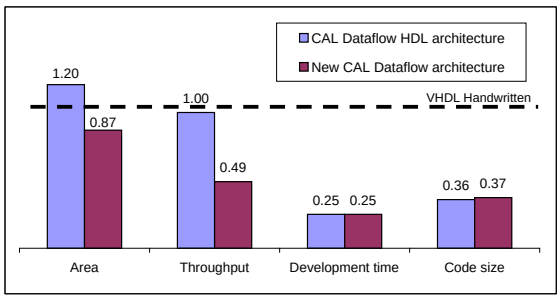

Fig. 12. Comparison to VHDL reference design.

VI. CONCLUSION & FUTURE WORKS

In this paper two design methodologies, a classical approach with handwritten HDL and a CAL dataflow development, are compared. The performance of dataflow approach and the performance of the conversion from CAL to HDL are evaluated. The results reported in section V are very promising. Beside a slight reduction of the maximum achievable frequency, other

	Number of occupied slices	Number of slice Flip Flops	Number of 4 input LUTs	Code size (kbyte)	Frequency MAX (MHz)	Processing Time(ms)/Throughput (image 1712x180/s)	
						At max Frequency	At 50 MHz
Without processing module	949	1120	887	25.7	≈90	0.85/1176	1.54/649
High pass filter only	2193	1999	1846	39.0	≈90	0.85/1176	1.54/649
All processing modules	3127	2930	3956	65,7	≈90	1.92/520	3.57/280

Table I: CAL Dataflow HDL architecture results

	Number of occupied slices	Number of slice Flip Flops	Number of 4 input LUTs	Code size (kbyte)	Frequency MAX (MHz)	Processing Time(ms)/Throughput (image 1712x180/s)	
						At max Frequency	At 50 MHz
Without processing module	469	469	549	13.7	≈100	0.77/1299	1.54/649
High pass filter only	1,220	1,170	1,621	34.2	≈100	2.31/433	4.61/217
All processing modules	2,276	1,991	3,079	68.3	≈90	4.06/246	7.3/137

Table II: Dataflow revised design results

	Number of occupied slices	Number of slice Flip Flops	Number of 4 input LUTs	Code size (kbyte)	Frequency MAX (MHz)	Processing Time(ms)/Throughput (image 1712x180/s)	
						At max Frequency	At 50 MHz
Without processing module	1,171	902	1,790	145.0	≈125	0.61/1,639	1.54/649
High pass filter only	1,639	2,779	2,376	162.0	≈125	0.61/1,639	1.54/649
All processing modules	2,623	3,855	3,740	182.9	≈125	1.37/730	3.57/280

Table III: Hand written VHDL results

design results are improved, particularly in terms of HW resources for the revised design and throughput for the original design. Moreover, development time and code size in both CAL model examples are reduced of a relevant factor, enabling interesting redesign iteration options.

Several improvements could be further applied to the methodology tested so far. The first is certainly to include a library of basic low level function to ease CAL code writing and considerably reduce the size of the code. The second is to continue the improvement and the optimization of the HDL generation tool as well as implementing extensions of the OpenDF framework functionality.

REFERENCES

[1] Johan Eker and Jorn Janneck, "CAL Language Report", Tech.Rep.ERL Technical Memo UCB/ERL M03/48, University of Califonia at Berkeley, Dec. 2003.

[2] Jörn W. Janneck, Ian D. Miller, Dave B. Parlour, Marco Mattavelli, Christophe Lucarz, Matthieu Wipliez, Mickal Raulet, and Ghislain Roquier, Translating dataflow programs to efficient hardware: an MPEG-4 simple profile decoder case study, in Design Automation and Test in Europe (DATE), Munich, Germany, 2008.

[3] Christophe Lucarz, Marco Mattavelli, Matthieu Wipliez, Ghislain Roquier, Mickaël Raulet, Jörn W. Janneck, Ian D. Miller and David B. Parlour, "Dataflow/Actor-Oriented language for the design of complex signal processing systems", Workshop on Design and Architectures for Signal and Image Processing (DASIP08), Bruxelles, Belgium, November 2008.

[4] J. Dubois, M. Mattavelli, "Embedded co-processor architecture for CMOS based image acquisition", IEEE International conference of Image Processing (ICIP03), Volume 2, pp.591–594, 2003

[5] R. Mosqueron, J. Dubois M. Mattavelli "High Performance Embedded co-processor Architecture for Cmos Imaging Systems", Workshop on Design and Architectures for Signal and Image Processing (DASIP07), Grenoble, France, November 2007

[6] R. Mosqueron, J. Dubois and M. Mattavelli, "Smart camera with embedded co-processor : a postal sorting application", Optical and Digital Image Conferance (SPIE08), Proceeding Volume 7000, Strasbourg, France, April 2008.

[7] "open DataFlow Sourceforge Project" http://opendf.sourceforge.net/.