# Hardware synthesis of complex standard interfaces using CAL dataflow descriptions

Richard Thavot*, Romuald Mosqueron*, Julien Dubois†, Marco Mattavelli*
*Ecole Polytechnique Fédérale de Lausanne (EPFL), GR-LSM, CH 1015 Lausanne, Switzerland
Email: *firstname.name@epfl.ch*
†Université de Bourgogne, Laboratoire LE2I, 21000 Dijon, France
Email: *firstname.name@u-bourgogne.fr*

## Abstract

*This paper presents a contribution to the development of rapid prototyping tools based on data-flow description. In this context, the efficiency of automatic translator tools from the data-flow description to C and/or HDL are presented using two design cases. Moreover, this paper presents the novel concept of the automatic synthesis of interfaces based on dataflow description. Such "generic" interfaces include an embedded microprocessor, which enables using a vide variety of interfaces already available as optimized libraries from the FPGA manufacturers.*

*The different design cases described have been tested and validated on different platforms. The results of the work show the flexibility and generality of the proposed wrapper methodology that is described in the paper.*

## 1. Introduction

Nowadays, in the fields of embedded system, heterogeneous platforms are more and more used to implement complex processing applications. Due to the increase of the application complexity, it becomes more and more difficult to develop and to optimize algorithms mapped on heterogeneous platforms. Several steps need to be completed, and several issues need to be addressed such as: 1) which portion of the algorithm will result to be a more efficient implementation on which component of the heterogeneous platform?, 2) which partitioning schemes respect the constraint of the application, 3) and how to convert the selected section of the algorithm into the language compatible with the corresponding component. Several works are already ongoing and aims at addressing those issues.

The first issue can be addressed by using an appropriate language, which is the CAL dataflow language in that case. A dataflow language allows for getting a visual programming because the application could be represented by a graph. Compared with the languages that use other paradigms, dataflow programming allows for modeling on a high level of abstraction. The CAL language is based on the actor model of computation. It provides many features to facilitate systems modeling.

The second issue can be addressed by the existence of a tool environment that supports different design space exploration stages and yields efficient mapping and partitioning of the high level algorithm specification on each component of the heterogeneous platform. An essential element is the inclusion, at the level of the unified computation model, of the architectural components of the heterogeneous platforms and of native library/IP components.

A dataflow model expressed in CAL is composed of a set of independent "actors", which consume data tokens from channels, process these tokens, and then bring new tokens on channels. The channels are represented by directing edges to represent the direction of the communication between two actors. CAL language facilitates explaining explicitly how the full design will be partitioned on different chips/cores as well as the type of communication between them.

Generic wrappers have been designed to easily convert the model with the appropriate interface. A generic structure must be able to adapt to every communication controller. These wrappers could be connected with its sub-layer to every IP-interface defined within the component library. Two wrappers are necessary: one for communication interfaces and the other for memories.

In this paper, a description of the wrapper for hardware component is provided, with focus on the case of an FPGA with a soft-core. The paper shows that the wrapper is generic and can be used with many communication interfaces.

The remainder of this paper is organized as follows: Section 2 presents the main objective, the language and the methodology to quickly prototype new architecture. Section 3 presents the efficiency of the CAL converters on two design cases (an MPEG-4 simple profile

decoder and a bar code decoder). Section 4 shows the utilization of the interface wrappers. In section 5, a presentation of some interface wrappers is provided. Finally, section 6 concludes the paper.

## 2. Objectives and principles

This section describes the main objectives behind the work of this paper, focusing on the location of the wrappers in the design flow. A presentation of the CAL language [1] and its properties are made to explain why this language has been chosen. The methodology and the interface management are explained to sense the real advantage of the work.

### 2.1. CAL language.

A dataflow model expressed in CAL is composed of a set of independent "actors" [1], [2] and their connection structure which build a network (see Figure 1). An actor is a standalone entity which has its own internal state represented by a set of state variables. It performs computations by executing actions and it must have, at least, one action to do computations. An action execution is modeled as an atomic component which means that no other action, of the same actor, can execute at the same time. Moreover, an action is executed based on the internal state of the actor and depending on the availability and values of tokens at the input ports. The "actor" has a set of input and output ports through which it communicates with other actors by passing data tokens. In summary, an "actor" may consume tokens from inputs, may change the internal state and may produce tokens at the outputs.

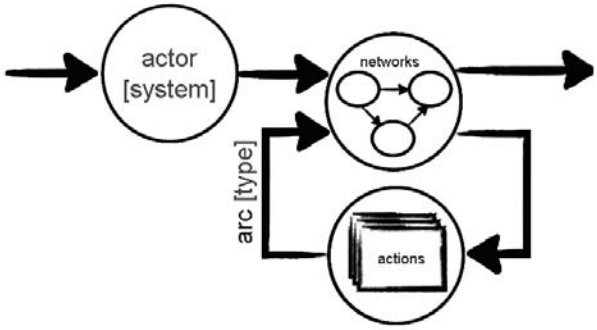


**Figure 1: CAL principles**

CAL provides scheduling concepts to control the executions order of actions inside an actor. CAL allows for combining a network of actors to build hierarchical systems. The network is achieved by connecting the input and output ports of actors together to define the communication structure of the model. Moreover the communication channels are constituted by FIFOs.

By using CAL, designers can only focus on the modeling of the dataflow system and do not need to care much about the low level of details to implement the communication between actors. Also, CAL provides the designer with the control over communication parameters such as length of queues and types of exchanged data. When a dataflow model is developed, it can be simulated using the Opendataflow simulator [3] to check for the correct functionality.

### 2.2. Objectives

The main objective is to define and develop a methodology for a unified specification formalism for software and hardware components to be mapped onto heterogeneous multi-component embedded platforms, using a synthesizable high-level dataflow formalism, based on the CAL language, which is capable of specifying and modeling both software and hardware components. In a CAL-based design flow, the whole system is modeled and implemented in CAL. Moreover the partitioning between hardware and software can be easily modified since the same source is used for generating both parts.

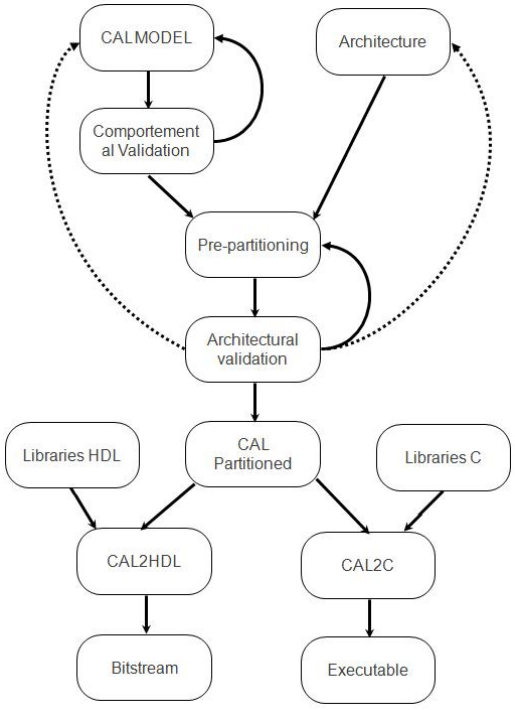


**Figure 2: Data flow based on CAL description**

Figure 2 details the design flow: a CAL program which is the global algorithm is the first input, while the second input is the architecture of the

heterogeneous platform. First, the CAL program is validated with a behavioural simulation.

Then, the second step is the pre-partitioning. This step defines, in accordance with the constraints and the architecture (component and interface), which actors will be the best placement, either hardware or software. During this step, simulations are made to find the correct partition, with insertion of wrappers and actors which represent the behaviour of the interfaces included into the architecture. In the case where no partition is found, a request to rewrite the CAL program or change the architecture constraints is made to the designer. Once the architectural validation is obtained, a phase of writing a CAL model, called CAL partitioned, is executed specifically after the validation. The partitioned CAL is the result of the partitioning where attributes are added to correspond with the component (SW or HW). This is in this part that the actors "wrappers" are included into the design. In accordance with the interface present in the platform, the wrapper is defined and the sub-layer (peripheral wrapper) is parameterized to correspond with the IP-interface during the code translation. Nevertheless, on the hardware side, the interface integration is much harder to put in place because it has no operating system (OS) to manage interfaces automatically instead of the software side. Moreover many edges within a chip can use the same physical path (or peripheral) to access actors within another chip. An example of a partition is shown figure 3.

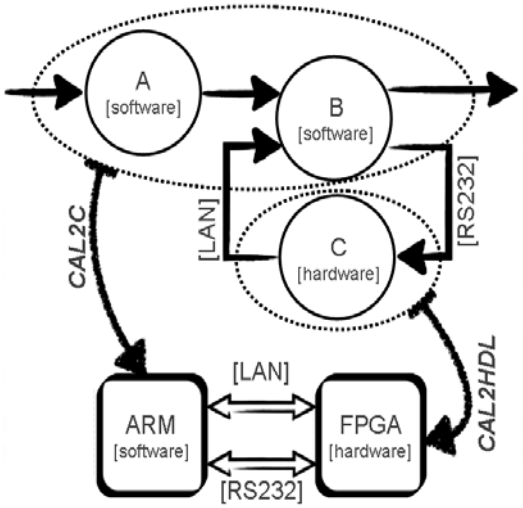


**Figure 3: Example of partitioning**

Once the choice of the actors is made for the partition, each part is translated into the correct language by two tools: CAL2HDL [4] for FPGAs and CAL2C for processors [5].

The final step is to synthesize and to implement by calling and adding specific IPs that are present in the component's library. For instance a bitstream is created for an FPGA. The ultimate goal is to automate all the previous steps. However, up till the time this paper is written, only the definition of the wrappers has been performed.

.

## 2.3. The global methodology and interface management.

One of the major key issues in design conception is to provide rapid prototyping methodology to define and to validate a system architecture that reaches the application's requirements. The CAL description and the related implementation tools enable to describe and to implement a completed processing chain.

The processing part, as explained in the previous section, can be split into two partitions SW and HW that can be automatically translated to C and VHDL codes respectively. The efficiency of the automatic translation will be discussed in section 5 with two existing design cases.

Two specific classes of actors can be defined to represent respectively the external interfaces and the external memory. The two resulting models can be used at different stages of the design's validation. The model can be used to validate:

- the functional CAL,
- the CAL description obtained after merging with the architecture definition

For instance, the external interfaces can be described with a simple description:

- bandwidth,
- temporal interruption (period or randomly generated).

Obviously, the model can be completed to be more conformant to the real physical interfaces.

The external interfaces are directly exchanged with the physical link (for instance LAN, RS232...).

A completed automatic implementation requires handling with the control of the different interfaces. A technological solution is proposed for the two partitions.

The processor in charge of the SW partition can easily handle the control of the interface with a C driver. For the HW partition, a controller, as well as the driver, must also be generated to connect the interface with the HW partition.

A unique wrapper structure, described in CAL language, is proposed to handle the interfaces. These latter enables a large variety of interfaces to be integrated. It is composed of a generic part to handle with interface interconnections and an adapter specific

to each physical interface. The structure of the adapter is based on a micro-controller (Xilinx-Microblaze or Altera-NiosII), which nowadays proposes a variety of interface controllers. The wrapper does not modify the performances of the micro-controller's solution.

For instance, as presented in figure 4, the different wrapper should be added for each external interface.

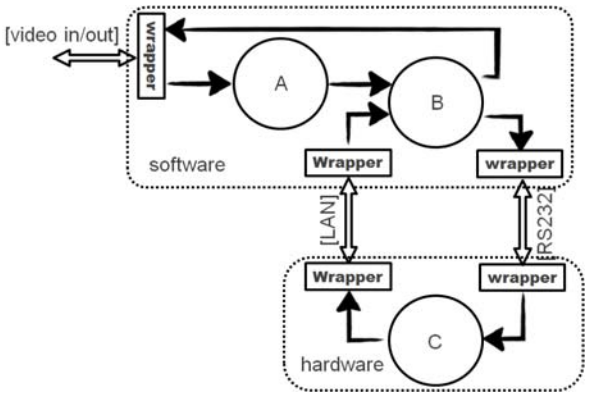


**Figure 4: Example of partitioning with the addition of the communication wrapper.**

A validation test is made with the partition of actors or network in both SW and HW component. The insertion of actors "wrappers" is obligatory, for this real simulation, as well as the insertion of the interface's models. Video in/out is one of this model defined by parameters near the physical constraints. The simulation made at this step is the nearest of the real situation. LAN and RS232 are also defined between the two wrappers with their constraints. This stage is the pre-partitioning one, which is one of the most important into the rapid-prototyping phase.

For the SW partition, a C driver is generated with the CAL2C tool. For the HW partition, a wrapper with a specific adapter, based on embedded micro-controller is generated. The adapter is dedicated to a specific interface due to the associated micro-controller code, which is also automatically generated.

## 3. Effectiveness of CAL2C and CAL2HDL

The translator has been tested with two different real applications. The first is the MPEG-4 SP decoder [4], [5], [6] [7], while the second is the code bar decoding [8], [9], [10], [11] in postal sorting. Both applications focus on the flexibility of a CAL description and the interests of high level of abstraction for a complete application description.

### 3.1. First design case: the MPEG-4 SP decoder

MPEG-4 is a suite of standards which has many "parts", where each part standardizes various entities related to multimedia, such as audio, video, and file formats. MPEG-4 contains a number of features that allow it to compress video much more effectively than older standards and to provide more flexibility. Figure 5 shows the MPEG-4 decoder which has been described via a dataflow model using CAL.

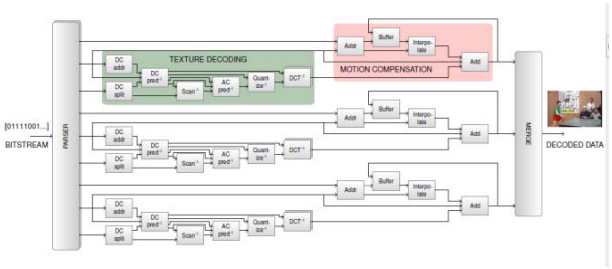


**Figure 5: MPEG-4 SP decoder described in CAL**

This decoder model is composed of three distinct parts. The first includes the parser and merger actors. The parser cuts the bitstream video in Y, Cr, Cb streams and theirs associated motion vectors. The merger recomposes the video picture. The second part is used to decode the texture, then the third part computes the motion compensation on the decoded texture. The MPEG-4 SP dataflow description is composed of 42 actor instantiations. Figure 6 compares the entire MPEG-4 SP decoder written in CAL and the same decoder directly describe in HDL files. This graph shows a big advantage in term of development time and code size description for the MPEG-4 SP decoder CAL description compared with the manually-written description (normalize to 1). This graph also shows an advantage for CAL description in term of area used by the FPGA and the throughput.

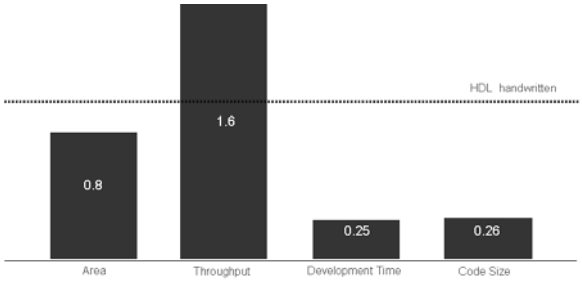


**Figure 6: Comparison of hardware performances between CAL generation and HDL handwrite for the MPEG-4 SP Decoder**

The entire MPEG-4 decoder can be generated from the CAL description using CAL2C [6]. Table 1 shows the different throughput performances between three MPEG-4 SP decoder code [4].

| MPEG4 SP decoder | Speed kMB/S | Code size kSLOC |
|---|---|---|
| CAL simulator | 0.015 | 3.4 |
| Cal2C | 2 | 10.4 |
| Cal2HDL | 290 | 4 |

**Table 1: Performances of the MPEG-4 SP decoder described in CAL, generate C, and generated HDL**

### 3.2. Second design case: the code bar decoder

The goal of this application is to detect and decode bar codes on letters to enable automatic sorting at different stages of the logistic postal letter handling.

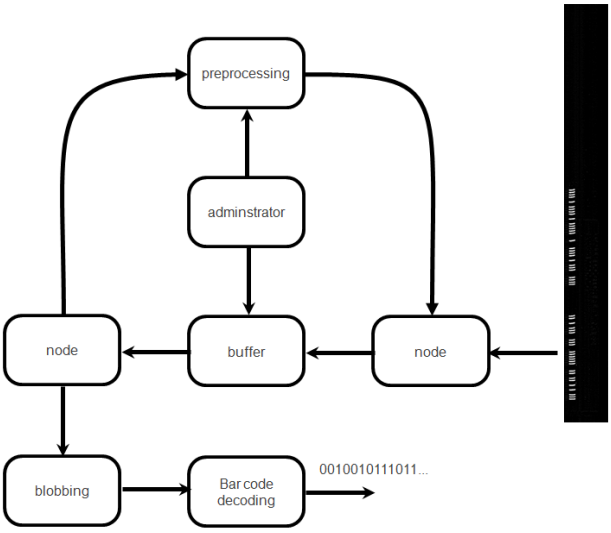


**Figure 7: Code bar decoder describes in CAL.**

Figure 7 shows the code bar decoder which has been described in CAL. This design has three distinct parts, which are the preprocessing, two processing (blobbing and code bar decoder), and then the manager stage. The first part computes some filters to improve the picture quality and highlights useful area. Both processing compute algorithms on the picture to find the proper area and decode it. The third part manages correctly the flow inside the design.

Figure 8 compares two architectures, one is the CAL-generated, while the other is the manually-written CAL description (normalize to 1). The results show that the development time and the code size of the description have a factor of four compared with the handwritten. The figure also shows that in term of area the difference is more or less equivalent but the throughput change dramatically according to the used description.

Furthermore, this comparison shows that the application can even run with half the throughput.

These two design cases show the effectiveness of the CAL translators (CAL2HDL and CAL2C). Therefore, the HW/SW partitioning can be delayed to the last conception stage of the design flow. Consequently, the automatic insertions of interface controllers represent a key-point to propose efficient rapid prototyping tools.

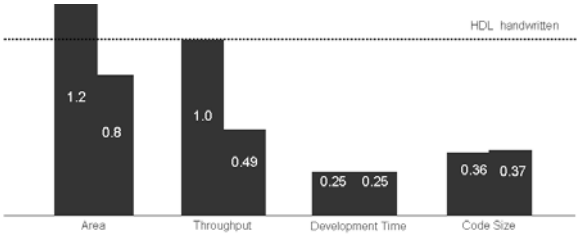


**Figure 8: Comparison of hardware performances between two CAL generation and HDL handwrite for the Bar code decoder [7].**

The results are more or less the same performance in order of the description CAL used, particularly the area's performance. This latter is closed to the handwritten description. But the development time and the size of the description has been improved by about a factor of four.

## 4. A CAL wrapper for implementation of interface controllers

A wrapper is used to connect directly an interface without its denomination. The wrapper must be able to easily connect as many devices as possible. In the partitioned CAL, only the parameters change to configure the right adapter. Figure 9 represents the overview of the wrapper on the hardware side.

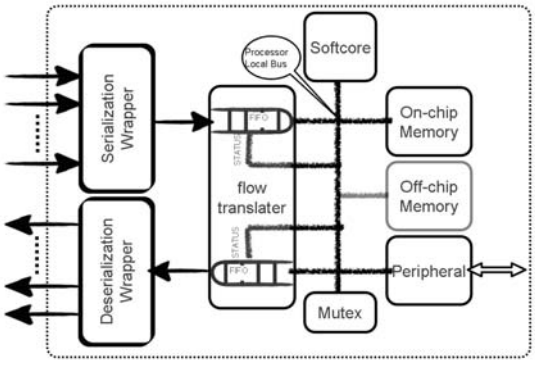


**Figure 9: Wrapper overview**

Several “arcs” can use one wrapper, if only one interface can be used for many tasks or transfers. The wrapper must be able to serialize data from channels and must be able to de-serialize the data to the proper channel.

### 4.1. Serialization and de-serialization process

Both wrappers are used to convert a token from any channel into a sequence of words transmitted across a network connection link. Both wrappers are described in CAL for simplification. This allows automatic generation of the wrapper according to the number of “arcs” connected to the device. Even if it is written in CAL, it remains completely transparent because it will be instantiated during the pre-partitioning and the partitioning steps. Another advantage of making a CAL description is that the future improvements will be easier. The serialization wrapper should add data such that the de-serialization wrapper is able to redistribute the tokens on the proper channels. Moreover, the arcs connected to the wrapper are not necessarily active at the same time, and then the wrapper must be able to adjust its consumption and its production of tokens automatically according to the active arcs. To solve this problem of random token presence on the arcs, the actions should describe all the possibilities of consumption.

### 4.2. Generic device connection

The generic device connection allows connecting any communication interface with the serialization and the de-serialization wrappers. To make it easily realizable, a soft-core microprocessor is required; microBlaze for Xilinx and/or NiosII for Altera. The advantage of using a microprocessor is that most of the IPs to dedicate communication interfaces have already been created and optimized.

The microprocessor architecture is a RISC soft-core architecture, which is implemented entirely in the programmable logic. It holds between 900 and 2600 “logic cells” and can reach a frequency of 80 to 180 MHz depending on the platform and options. A microprocessor is connected on a multi-master bus to access the IP slaves.

The proposed architecture is composed of one On-chip memory and if necessary one external memory. The architecture has a peripheral controller that is either built by the manufacturers or self-described. The architecture has a flow translator which converts the tokens into a pile of data and vice versa. Both piles of data are accessible from two different processor local bus addresses as the status of both FIFO.

This conversion is made thanks to FIFOs for both directions. The wrapper adapter has also the possibilities of checking both FIFO statuses. Each component is accessible by the Processor Local Bus which is managed by the soft-core. A hardware Mutex has been placed on the Processor Local Bus, then the soft-core is able to supervise the critical section. The soft-core uses robust and light-weight µOS which are based on the pre-emptive real time multitasking operating system kernel for microprocessors. This type of µOS allows having several concurrently running tasks called “threads” and allows having event flags, which suspend or run the thread according to their status.

## 5. Implementations and experimental results

The wrappers are used in a simple CAL example (at the level of partitioned CAL model), which sends and receives data from and to the FPGA. The examples follow the steps from partitioned CAL to bitstream generation. These examples have been tested and validated on different platforms.

To verify the flexibility and the smooth operation of the generic hardware interface model; several implementations with communication bus has been performed.

### 5.1 Ethernet link

In this subsection a peripheral that is dedicated for the Ethernet protocol is tested. To compare the flexibility of the wrapper interface, the system is implemented on two different FPGAs from two different manufacturers. The first example has been designed targeting the Altera family associated with an SMSC component. The same case can be applied to the Xilinx family. Nevertheless, Virtex 5 from Xilinx includes a specific core dedicated for Ethernet communication, which is the second case.

Both designs need an external memory to use the light-weight implementation of the stack TCP/IP. The Internet Protocol Suite is the set of communication protocols used for the Internet and other similar networks. The light-weight TCP/IP implementation is to reduce the RAM usage while still having a full scale TCP. This makes light-weight TCP/IP suitable for usage in embedded systems with tens of kilobytes of free RAM and rooms for around 40 kilobytes of code ROM.

### 5.1.1. Ethernet on Cyclone II

The microprocessor that is used in the cyclone II is the NIOS II, which is a 32-bit embedded-processor architecture designed specifically for the Altera family of FPGAs [12]. This microprocessor is connected on a local processor bus named "Avalon". "Avalon" interfaces simplify system design by allowing the designer to easily connect components into an FPGA. The Avalon interface family defines interfaces for usage in both high-speed streaming and memory-mapped applications. In the studied case, the peripheral controller connected to this bus is the SMSC LAN91C111 device controller [13]. This external device is designed to facilitate the implementation of a third generation of Fast Ethernet connectivity solutions for embedded applications. The LAN91C111 is a mixed signal Analog/Digital device that implements the AMC and PHY portion of the CSMA/CD protocol at 10 and 100 Mbps. The design will also minimize data throughput constraints utilizing 32-bit, 16 bit or 8-bit bus Host interface in embedded applications. The internal buffer size is 8 Kbytes, which is the total chip storage for transmitting and receiving operations.

### 5.1.2. Ethernet link on Virtex 5

The microprocessor that is used with the virtex 5 is the microblaze, which is also a 32-bit Harvard RISC architecture, optimized for the Xilinx FPGA families [14]. The microblaze is connected on processor local bus named "PLB". The processor local bus PLB is a high-performance bus that provides a standard interface between the processor cores and the integrated bus controllers. The PLB supports reading and writing data transfers between master and slave devices. These latter are equipped with a PLB bus interface and connected through PLB signals. The device controller connected to the PLB is the TEMAC (Tri-Mode Ethernet MAC). The TEMAC is a configurable core ideally suited for using in networking equipment such as switches and routers. The customizable TEMAC core enables system designers to implement a broad range of integrated Ethernet designs, from low cost 10/100 Ethernet to higher performance 1 Gigabit ports. The TEMAC core is designed to the *IEEE 802.3* specification and operates in 1000 Mbps, 100 Mbps, and 10 Mbps modes. In addition, it supports both half and full duplex operation. The Xilinx Tri-Mode Ethernet MAC, combined with the Ethernet 1000BASE-X PCS/PMA or SGMII core, provides a complete and highly flexible solution for the implementation of Ethernet Link and Physical layers.

### 5.2 PCI link

In this subsection a peripheral that is dedicated for the PCI protocol is tested. The proposed implementation is based on the wrapper that is explained in section 4. This implementation is compared with the direct implementation of a specific PCI controller.
As described in section 4, the wrapper is based on a microblaze microprocessor. The target technology is the Virtex 2 Pro FPGA from Xilinx. The peripheral connected on the PLB bus is the PLBV46 PCI Full Bridge. This one provides full bridge functionality between the Xilinx PLB and a 32-bit Revision 2.2 compliant Peripheral Component Interconnect (PCI) bus.
The proposed implementation is obtained with the automatic translation of the wrapper dedicated to this PCI configuration. The obtained interface performances have been compared to a reference design, which is based on the Initiator/Target PCI core form Xilinx. This module is a pre-implemented and fully tested module for Xilinx FPGAs. The pinout for each device and the relative placement of the internal logic are predefined. Critical paths are controlled by constraints files to ensure predictable timing. The Xilinx IP cores for PCI technologies operate at maximum throughput, with zero wait-state bursts. The core is directly connected with a minimal wrapper (serialization and de-serialization). The performance is identical for both designs, and therefore represents another important step to obtain an efficient rapid prototyping tool based on CAL dataflow description.

## 6. Conclusion

### 6.1. Results

This paper shows that the generic wrapper methodology provides a high-degree of flexibility and robustness. The proposed methodology is expected to enhance the design-flow of typical embedded systems. In case of observing a lack of performance, a designer may design his/her own peripheral Intellectual Property, and then modify the generic peripheral wrapper. Moreover, the proposed wrapper methodology enables easy changing of interfaces.

### 6.2. Future work

The serialization and the de-serialization wrappers suffer deadlocks if the FIFOs sizes are not chosen correctly according to the design. These deadlocks will

be corrected by feedback information between the two wrappers.

The same methodology for the memory device will be implemented. The difference between a memory device and a standard interface is that memories have a two-flow access, addresses and data. Moreover, the data channel is bidirectional. Thus memories will be described with four channels (two flows in both directions) as shows in the figure 10.

In addition, if many actors want access to the memory via the serialization wrapper (and the de-serialization wrapper for the data read), the soft-core must drive the tokens properly.

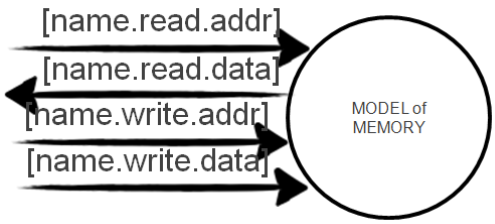


**Figure 10: CAL description for memories.**

## 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] 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.

[3] “Open DataFlow Sourceforge Project” http://opendf.sourceforge.net/.

[4] Jorn W. Janneck, Ian D. Miller, Dave B. Parlour. “Profiling Dataflow Programs”, Reconfigurable Video Coding and Processing (ICME08), Germany, Hannover, June 2008

[5] Ghislain Roquier, Matthieu Wipliez, Mickaël Raulet, Jörn W. Janneck, Ian D. Miller, David B. Parlour, “automatic software synthesis of dataflow program: an MPEG-4 Simple Profile decoder case study”, Signal Processing Systems, 2008. SiPS 2008. IEEE Workshop on, Washington,USA (2008).

[6] 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.

[7] C. Lucarz, M. Mattavelli, J. Thomas-Kerr, and J. Janneck, “Reconfigurable Media Coding: a new specification model for multimedia coders,” in Proceedings of SIPS’07, Oct. 2007.

[8] Richard Thavot , Romuald Mosqueron , Mohammad Alisafaee , Christophe Lucarz , Marco Mattavelli , Julien Dubois , Vincent Noel, ”Dataflow design of a co-processor architecture for image processing”» , Workshop on Design and Architectures for Signal and Image Processing (DASIP 2008), Bruxelles , Belgium, November 2008.

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

[10] 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.

[11] 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

[12] “Nios II Processor Reference Handbook” http://www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf

[13] “10/100 Non-PCI Ethernet Single Chip MAC+PHY” http://www.smsc.com/main/datasheets/91c111.pdf

[14] “MicroBlaze Processor Reference Guide” http://www.xilinx.com/support/documentation/sw_manuals/mb_ref_guide.pdf