# CASSE: a system-level modeling and design-space exploration tool for multiprocessor systems-on-chip

**Conference Paper** · January 2004

DOI: 10.1109/DSD.2004.1333313 · Source: IEEE Xplore



**5 authors**, including:

Victor Reyes
Metropolitan Autonomous University
**12** PUBLICATIONS **68** CITATIONS
SEE PROFILE

Tomás Bautista
Universidad de Las Palmas de Gran Canaria
**41** PUBLICATIONS **202** CITATIONS
SEE PROFILE

Gustavo Marrero Callico
Universidad de Las Palmas de Gran Canaria
**161** PUBLICATIONS **1,031** CITATIONS
SEE PROFILE

Pedro P. Carballo
Universidad de Las Palmas de Gran Canaria
**46** PUBLICATIONS **99** CITATIONS
SEE PROFILE

# CASSE: A System-Level Modeling and Design-Space Exploration Tool for Multiprocessor Systems-on-Chip

Víctor Reyes, Tomás Bautista, Gustavo Marrero, Pedro P. Carballo
*IUMA, Institute for Applied Microelectronics, University of Las Palmas GC, 35017, Las Palmas GC, Spain*
*vreyes@iuma.ulpgc.es*

Wido Kruijtzer
*Philips Research Laboratories, Prof. Holstaan 4, 5656 AA Eindhoven, The Netherlands*
*wido.kruijtzer@philips.com*

## Abstract

*As SoC complexity grows new methodologies and tools for system design and time-effective design space exploration are required. In this paper we introduce a tool called CASSE, what stands for CAmellia System-on-chip Simulation Environment. CASSE is a fast, flexible, and modular SystemC-based simulation environment which aims to be useful for design-space exploration and system-level design at different abstraction levels. The tool uses transaction-level modeling techniques for fast simulations and easy architectural modeling, and bridge the gap to system implementation by a progressive refinement approach.*

*CASSE is being used in the European IST-2001-34410 CAMELLIA project, which focuses on the mapping of innovative smart imaging applications onto an existing video encoding architecture.*

## 1. Introduction

Smart imaging applications combine image and video capturing with the processing and/or interpretation of the scene contents. A good example is a camera that is able to segment a video sequence into objects, track some of them, and raise an alarm if some of these objects show an unusual behaviour. The aim of the European IST-2001-34410 CAMELLIA project is to develop a smart imaging core that can be embedded in a camera [1]. This core should be suitable of supporting automotive and mobile communication applications. In the mobile domain MPEG-4 video compression is also required so the core will be based on an existing MPEG-4 video compression core. The idea is to extend this core with smart imaging (pixel processing) functionality.

Smart imaging is an important step in the direction of ambient intelligence. Ambient intelligence in general and smart imaging in particular are two concepts which are expected to be the driving forces for the consumer electronics industry in the near future [18]. Designing systems-on-a-chip (SoCs) that cope with such application domains is indeed very complex. Complexity arises not only from the increasingly computational requirements of those applications but also from the more and more tough constraints in terms of time-to-market, cost, safety, performance, reliability, etc. Therefore, as SoCs complexity grows methodologies and tools which overcome these drawbacks are needed for helping the system designer.

On one hand, heterogeneous multiprocessor systems-on-chip (MPSoC) are becoming the chosen option to overcome the continuous increase of computational requirements, like these in the smart imaging field. But to take advantage of their multiprocessing possibilities, applications have to be described in a way that their parallelism can be exploited. Exploiting task level parallelism allows these computing-intensive applications to perform in real time. In order to describe a parallel system in an easy and efficient way formal methods or models of computation (MoC) are used. Application modeling based on the Kahn Process Network MoC, which exploit the parallelism at task-level, has been applied successfully in many signal processing applications [10] [11] [12].

On the other hand, methodologies that emphasize re-use and standardized SoC design methods has resulted in the notion of a platform and in the orthogonalization of concerns [2]. A platform structures and standardizes SoC architectures, by regulating the kind of IP blocks that can be used, how they are integrated, and how the system is programmed. The orthogonalization of concerns promotes separating *functionality* and *architecture*, and separating *communication* from *computation*. The

Proceedings of the EUROMICRO Systems on Digital System Design (DSD'04)
0-7695-2203-3/04 $ 20.00 IEEE

function/architecture separation allows the reuse of functions for implementation on different architectures. The communication/computation separation allows the communication infrastructure to evolve without the need to change the computational blocks and thereby enabling IP re-use.

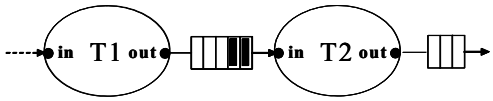
One of the important aspects of the CAMELLIA project is centred on the efficient mapping of the smart imaging functionality onto an enhanced video compression core. Thereby, mapping techniques which allow design-space exploration in order to find out the best HW/SW partitioning of a demanding application into a constrained architectural platform are crucial for shortening the development cycle and, hence, for facing with guarantees the increasing complexity of nowadays applications.

We present in this paper a tool called CASSE which supports system-level design and design-space exploration of multiprocessor system-on-chip at different levels of abstraction. System-level design and design-space exploration of complex systems require fast simulations and easy modeling with a certain level of accuracy. Fast simulation can be achieved by designing at high levels of abstraction. Transaction-level modeling (TLM) has been promoted as the next modeling abstraction above RTL [5]. TLM is intended to make the system modeling easier by reducing the amount of details that the designer must handle. The aim of transaction-level modeling is to achieve increased simulation speeds, while keeping enough accuracy for system analysis and verification.

### 1.1. Methodology overview

CASSE applies a Y-chart based methodology [3] [9]. Our starting point is a functional model in terms of process networks based on the Kahn Process Network (KPN). Applications are decomposed in concurrent tasks which communicate with each other using a KPN-derived protocol for inter-task communication. This protocol is similar as the ones found in [10] [11]. The mapping of this functional model comprises an assignment of the tasks towards a specific architectural platform taking into account its costs and constraints, such that the tasks can be implemented efficiently in hardware and software. Our architectural models follow an interface-based design approach [4] where the communication among architectural components is based on predefined interfaces and on an inter-component communication protocol. Once the mapping phase is completed performance simulations are performed in order to obtain simulation traces and metrics which can help in the decision process. If requirements are not fulfilled the system designer could iteratively modify the mapping, the architecture model or even the process network structure. If the requirement are fulfilled the system is ready for implementation. CASSE uses for the system implementation a progressive refinement approach.

CASSE is a fast, flexible, and modular SystemC-based simulation environment which aims to be useful for design-space exploration (DSE) and system-level design at different abstraction levels. The CASSE simulator has been completely developed using SystemC/C++. Similar approaches have been followed in [6] [7] and [8]. In [8] a SystemC based methodology for architectural exploration of SoC is also presented. In [6] [7] an automatic component integration methodology which allows re-use of predefined component is described. Unlike in those previous works we emphasize more on a seamless KPN-derived protocol refinement from system-level to implementation, and a dynamic instantiation of mapping alternatives for DSE.




```
T2: while (true) {
        acquireData(in);
        load(in,t);
        releaseRoom(in);
        y = f(t);
        acquireRoom(out);
        store(out,y);
        releaseData(out);
}
```

**Figure 1. Task communication with ITCP**

The paper is organized as follows: Section 2 introduces more in detail the CASSE simulator. The KPN-derived inter-task communication protocol called ITCP is introduced in section 2.1. Likewise, the inter-component communication protocol called ICCP is presented in section 2.2. Section 2.3 covers the simulator structure and main functionality. Architectural refinement also covered by CASSE is described in section 2.4. In section 3 the CAMELLIA case study is introduced. Finally, in section 4, we present our conclusions.

## 2. CASSE simulator overview

### 2.1. The inter-task communication protocol

CASSE uses a KPN-derived protocol for describing applications called ITCP. Applications are described as process networks, where processes (or tasks) execute concurrently and communicate with each other by means of point-to-point channels. Although execution inside every task is sequential, all tasks in the process network execute concurrently and, therefore, communication

among them has to be synchronized to inform the tasks about the presence/absence of data and buffer space in the channels.

ITCP is a protocol for inter-tasks communication and synchronization. This protocol defines a group of primitives and their behavior, but it is not focused to any particular implementation. The protocol is architecture-independent and can be implemented on both hardware and software. To be as generic as possible the protocol treats separately the data access and the data synchronization. With ITCP applications are represented as a network (or graph) of concurrent tasks which are connected together by unidirectional channels. These channels behave as FIFOs. Its sizes can be selected individually by the system designer. Tasks access the channels using ports. These ports implement the ITCP primitives. We provide two classes of ports: input and output ports. As mentioned before, the protocol primitives are also split in two classes: data transport and data synchronization. There are two data transport primitives: read a data from the channel (*load*), and store a data into the channel (s*tore*). Likewise, there are four data synchronization primitives: two for testing and two for update purposes. The test primitives are: check if there is data available in the channel for reading (*acquireData*) and check if there is room in the channel for writing (*acquireRoom*). The update primitives are: update the number of data items available in the channel for reading (*releaseData*) and update the room available in the channel for writing (*releaseRoom*). The l*oad*, a*cquireData* and *releaseRoom* primitives are implemented in the input ports, and the *store*, ac*quireRoom* and *releaseData* primitives are implemented in the output ports. Besides the protocol primitives, ITCP defines also primitives to interconnect ports and channels.

Communication between two tasks using the inter-tasks communication protocol, see figure 1, is performed as follows. First T2 claims access rights for reading data from input port 'in' using the blocking '*acquireData*' primitive, after which the data can be read and the buffer space freed at the input port ('*load*' and '*releaseRoom*' primitives). After computing the result *y*, we first claim access rights for storing data (room) from the output 'port' using the blocking '*acquireRoom*' primitive, after which the data is written and the buffer space filling updated at the output port ('*store*' and '*releaseData*' primitives).

## 2.2. The inter-component communication protocol

The inter-component communication protocol is used for communication among architectural components. ICCP is an abstract protocol which can be used for modeling device transaction level protocols such as VCI [16], OCP [15] or AXI [17]. In our case we model the DTL protocol [19], part of the Philips Nexperia platform. DTL is a point-to-point communication protocol similar to the other well-known protocols mentioned above. The protocol defines a point-to-point interface between two communication entities called Initiator and Target. The Initiator acts as the master performing the communication requests. Likewise, the Target acts as a slave responding the requests sent by the Initiator. This interface has been modeled at the transactional-level using SystemC Master/Slave channels [13], and at the bit-true level using SystemC signals.

At the transactional level each channel of the interface transports different information belonging to a group of signals of the protocol. In order to be as accurate as possible and still keeping the abstraction level, four different channels or group of signals have been defined in the interface. These four channels are a *request* channel, a *response* channel, a *write* channel and a *read* channel. The *request* channel includes all the information needed for carrying out the transaction (i.e. base address, read or write operation, number of data to read or write, and other essential information needed for the protocol). The *response* channel includes the information corresponding to the status of the transaction (i.e. *done*, *error*, or *reply*). The *write* channel includes the data to be written into the Target, and the *read* channel includes the data to be read from the Target.

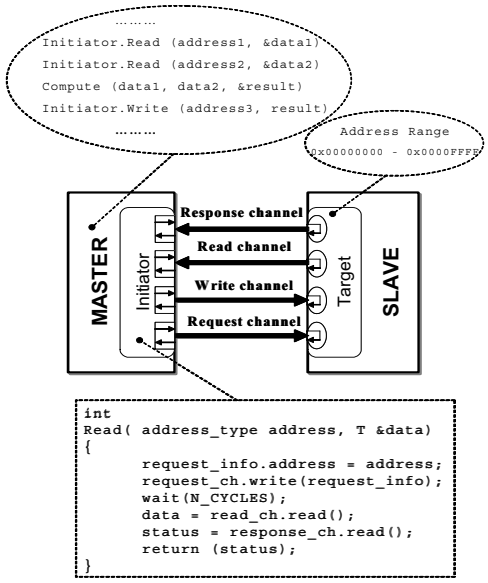



**Figure 2. Initiator and Target communication**

In ICCP the Initiator entity implements two basic methods, the read and the write method, for initiating the communication. Communication starts on the processing elements by executing those methods on the Initiator

modules. Likewise, the Target modules act as slaves simply executing the commands coming from the channels. The Target entity implements a register file interface to access the data and has a programmable address range. Both the Initiator and Target entities can be parameterized with different communication latency in their operations in order to model different system behavior and protocol standards. Communication between two components using the ICCP protocol is shown in figure 2.

## 2.3. CASSE structure and functionality

CASSE is structured in three different layers, as shown in figure 3:

- Front-end layer. This layer feeds to the back-end layer information about the application, architecture and mapping selection needed to perform its operations. The front-end layer is controlled by the user and generates two classes of information: the user libraries and the description files. There are two different user libraries: the tasks user library which contains the ITCP-compliant tasks composing the application, and the component user library which contains ICCP-compliant components which can be used to model a more accurate architecture. The description files are simple plain text files describing the process network structure (*task-graph* file), the architectural platform (*architectural* file) and the mapping (*mapping* file).
- Back-end layer. This layer is composed of the simulator core, the system libraries, the trace collector and the parser. System libraries implement components, interfaces and protocols for both architectural modeling and process network modeling. The parser interprets the description files provided by the upper layer to instantiate and bind components from the corresponding system libraries. The simulator core is in charge of performing several operations like process network modeling, architectural modeling, and mapping selection. The output of the simulator core is an executable model which is simulated using the kernel layer. The trace collector module gathers the system metrics and the application trace information produced during simulation.
- Kernel layer. The kernel layer is in charge of carrying out the system simulations (i.e. functional simulations of process networks and performance simulation of architectural models). As kernel layer CASSE builds upon the standard SystemC simulation kernel. The SystemC class library and simulation kernel extend the C++ language to enable the modeling of digital systems. SystemC provides a threaded event-driven simulator, modules and ports for representing structure, and interfaces and channels to describe communication [14].

**2.3.1. Functional simulation of process networks.** CASSE can dynamically model and simulate a process network. The process network has to be compliant with the inter-task communication protocol described in section 2.1. The simulator core uses the following elements for the functional simulation of a process:

- ITCP system-library. This library implements the ITCP protocol, that is, the primitives, ports and interfaces.
- Process-Network system-library. This library contains the following elements:

*Task container.* Every task in the process network has to be bound to a task container. A task container can be parameterized on the number of ITCP interfaces (which must be equal to the number of task ports). In turn, each port in the task will be bound to a single interface which provide access to the channels, see figure 4.

*ITCP channel.* This element implements a circular buffer channel with all the required functionality to be compliant with the protocol.

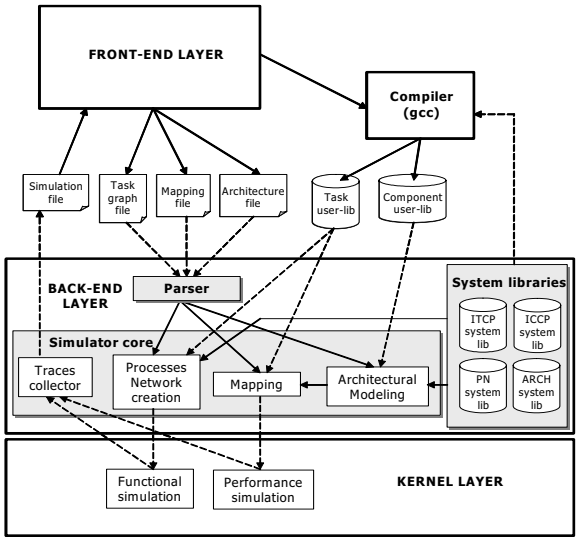



**Figure 3. The CASSE simulator structure**

- Tasks user library. The tasks user library contains the tasks which compose the application. These tasks have to be ITCP-compliant and, therefore, use the ports and primitives described in section 2.1.
- The '*task-graph'* input text file. This file describes the process network structure in terms of number of tasks, number of ports per task, number of channels and their interconnections. This file is used to instantiate dynamically tasks, task containers, interfaces and channels from the corresponding libraries. Moreover, tasks are bound to containers and channels to interfaces according to the structure described in this file.

Once the process network has been created and configured, the simulation can start. Process network simulations are done completely at the untimed functional level (UTF). Functional correctness of the process network can be checked and valuable information in terms of inter-task communication load can be gathered at this level.

**2.3.2. Performance simulation of system architectural models.** CASSE provides a modular (plug and play) approach for the architectural modeling of a platform composed of generic components. Complete system architecture models can be created from scratch and known architectural platforms can be emulated using the generic available components by configuring them accordingly. This architectural platform is composed of a configurable number of processing elements and storage elements interconnected together by a configurable communication network, see figure 5. All these elements communicate together using the inter-component communication protocol (ICCP) described in section 2.2. In order to be modular enough the architectural platform does not contain any functionality in its creation, but the functionality is assigned to the platform during the mapping phase. The simulator core uses the following elements to model the system architecture:

- Architecture system library. The components in this library are:

*Processing elements.* These elements perform the computational tasks in the systems, and can model both software and hardware components. Each processing element is composed of a *multi-task container* and a *protocol translator wrapper*. As above mentioned, functionality is assigned to the architectural platform by mapping tasks into the processing elements, but more specifically, tasks are mapped into the *multi-task container*. More than one task can be mapped into the same container, what calls for a *mutual-exclusion unit* (MEU) for accessing the shared resources. Hence, processing elements are well suited for modeling computations ranging from multitasking software running on a CPU to single-task hardware elements. Tasks communicate using the high-level ITCP protocol, but in turn the lower level ICCP protocol is used for communication in the architectural platform. This calls for a protocol translation from ITCP into ICCP. This is carried out by the *protocol translator wrapper* (PTW) which is highly configurable and able to implement multiple data transport and synchronization schemes.

*Storage elements.* These components model memory elements which can be parameterized in its size. Moreover, the storage elements are also configurable in terms of number of Targets interfaces.

*Communication network.* The communication network consists of a number of configurable network components. A network component is a shared-bus based interconnection with an arbiter module attached. Currently, two different arbiter modules are provided: a static priority policy and a round-robin priority policy. The network components are used to interconnect processing and storage elements. Each network component is configurable in terms of number of Initiator and Target interfaces, communication latency, and address-range for each Target module.

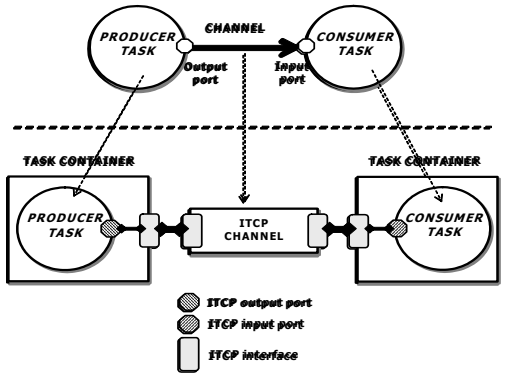



**Figure 4. A process network model in CASSE**

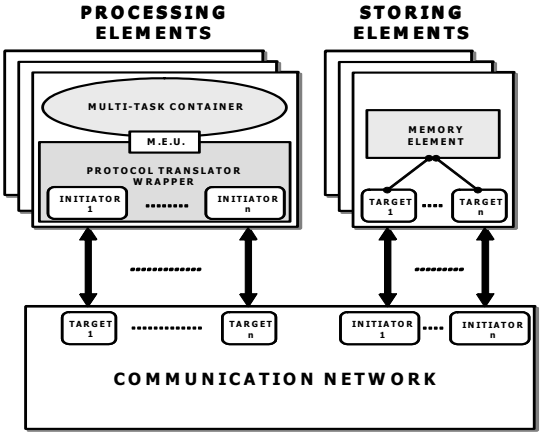



**Figure 5. The architecture platform template**

- The 'architectural' input text file. This file describes the composition and structure of the architectural platform. The file contains information about the number of components of each type and their interconnections, as well as the configuration of each individual component. The simulator core instantiates automatically those components from the architecture system-library and bind them together following the rules described in the input file.

During the mapping phase the CASSE simulator core performs a series of steps according the information provided in the *'mapping'* input text file. This *'mapping'* file contains all the necessary information to assign tasks and channels of the process network to specific processing and storage elements of the architectural platform. Of course, both the process network and the architectural platform have to be created before starting the mapping process. Once the mapping is performed the simulator core starts the simulation. This simulation is at the bus-cycle accurate level (BCA) for communication and at the timed functional level (TF) for computation. Communication latency can be specified in the Initiator and Target modules of the architecture platform as desired. The modeling of computation delays is by means of manual code annotation in the application tasks.

During simulation the simulator core collects relevant information and dumps it into text files for later inspection. The simulator gathers all the simulation information using special tracing classes which are transparent to the system designer. The information gathered is: the total number of execution cycles, communication load per interface (in bytes), communication load per network component (in bytes), average communication latency per interface (in cycles), and maximum communication latency per interface (in cycles). Moreover, information regarding the inter-task communication protocol performance can also be collected. Synchronization rate and average number of cycles waiting for data or room per tasks is stored in order to analyze the protocol implementation. Furthermore, the simulator core can trace information regarding the computational load per processing element if latencies are annotated in the tasks descriptions.

If after analyses of the trace information the system designer finds out that the chosen mapping does not fulfill the requirements, then both a new mapping selection and/or architectural modifications can be created without the need of recompiling the system. The fact that different architectures can be instantiated without recompilation allows driving the simulator through scripts (batch processing) more easily and hence allows a more extensive exploration of the design space (parameters sweeps).

### 2.4. System refinement

CASSE is not only useful for system analysis at the highest abstraction levels but also can bridge the existing gap between system modeling and system implementation. The approach followed to refine from an architectural model to a synthesizable architecture is twofold:

- Interface refinement. As mentioned in section 2.1, the ICCP protocol is also implemented at a bit-true level by using SystemC signals. This means that architectural components might communicate with each other at a cycle-true/bit-true level by simply replacing the transactional-level Initiator and Target modules with their equivalent bit-true versions. Both the Initiator and Target entities have been also implemented at the RTL level using synthesizable SystemC.
- Component refinement. As mentioned above, a components user library can be used to plug external components (designed by the user) into the architecture platform. Such components can be instantiated during the architectural modeling phase just like any other generic component. These user-designed components can be SystemC modules designed at both behavioral or RTL level with the only requisite of being ICCP compliant. Therefore, refinement is achieved by simply replacing the generic processing elements which execute functional tasks, with external components which, for instance, can model completely accurate hardware (co)processors.

CASSE also supports co-simulation: Untimed functional models using transaction-level communication with SystemC Master/Slave channels (TLM/SC-MS) and cycle-accurate models using bit-true communication with SystemC signals (CA/BT) can be connected using *abstraction-level adaptors* components (i.e. adaptors from TLM/SC-MS to CA/BT, and vice versa), which are included in the system libraries.

## 3. The CAMELLIA case study

The target smart imaging system architecture is based on the integration of coprocessors with smart imaging functionality in an existing video encoder, which consists of highly configurable dedicated hardware accelerators for video encoding (one of these hardware accelerators is a motion estimator coprocessor), a central CPU, a memory interface, and input and output interfaces.

As an example, one of the automotive domain applications identified for CAMELLIA, the *low-speed obstacle detection application* (LSOD), is disclosed here. This application is composed of a "high-level" algorithm (HLA) which combines the output of several vehicle detection "medium-level" algorithms (MLA) in order to obtain an exact detection and localization of vehicles. These medium-level algorithms use low-level operations (LLA) for pixel processing (e.g. kernel filtering, morphological and arithmetic operations, etc). The medium-level algorithms used in LSOD are: shadow detection, edges detection, rear lights detection, symmetry detection and motion segmentation.

With the classification of algorithms described above, a general approach for mapping an application onto the architecture can be derived. As the main characteristic of the high-level algorithm is its sequential nature and the limited computational requirements, this kind of

algorithms are candidates for execution on the embedded CPU. Low-level algorithms are associated with a high amount of inherent parallelism and relatively simple operations. Hence, they are well-suited for being mapped on coprocessors. Medium-level algorithms can be split into a pixel processing part to be mapped onto coprocessors and a control part to be mapped onto the embedded CPU. CASSE is used to analyze different partition alternatives for these medium-level algorithms. Depending on the partition chosen different processing granularity (i.e. synchronization rate) is derived. The processing granularity has a significant impact in both the communication load and the storage needed. For a particular HW/SW partition, in addition to this processing granularity impact, there exist many architectural alternatives which could also significantly influence on the final performance. These architectural alternatives (e.g. number and organization of memories, number of busses, etcetera) and even system parameters regarding communication issues can be also explored using CASSE.

Following the methodology presented in this paper we first have decomposed the low-speed obstacle detection (LSOD) application into several parallel processes which communicate with each other using the described ITCP protocol. The correctness of the process network is checked by functional simulations. In addition, the target system architecture is modeled using the simulator at the highest level of abstraction. According to the application composition (i.e. HLA, MLAs, and LLAs) and the project requirements (i.e. the need of reusing an existing platform), the mapping of the process network into the modeled architecture is performed as follows. The high-level algorithm and the more control-oriented part of the medium-level algorithms are combined together in a task, which fits well to be mapped onto the CPU component of the architecture. All the low-level operations are combined together in a pixel-processing task, which is mapped onto a smart imaging component. Likewise, the pixel processing part of the motion segmentation medium-level algorithm is distinguished as an independent task, which is mapped onto the existing motion estimator component. Hence, the best communication network and memory organization instance for the selected HW/SW partition is chosen by an iterative performance simulation process. Hereunder, an example of the target system architecture with two busses and two shared memories is presented, see figure 6.

Communication load and latencies per processing elements, as well as the impact of the synchronization rate, is obtained during the performance simulations. In figure 7 the communication load produced per processing element on the network component 2, corresponding to the figure 6 example, is shown. In figure 8 the number of synchronization calls per processing element is shown. These numbers correspond to a 25 frames simulation of the LSOD application. Among others, these results guide the tuning of parameters related with both the data transport and the synchronization (i.e. burst size, priorities, channel size, channel allocation, processing granularity, number of busses, etcetera).

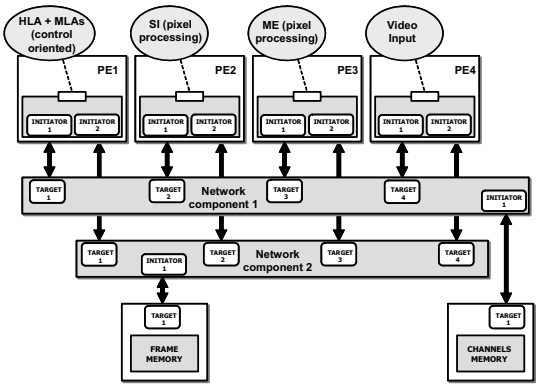



**Figure 6. Mapping of the LSOD application onto a system architecture instance**

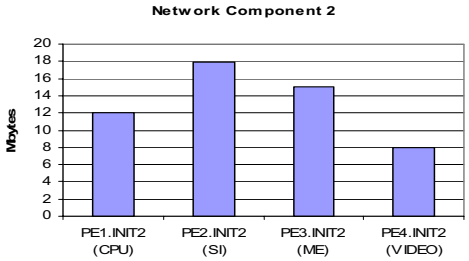



**Figure 7. Communication load on the Network Component 2**

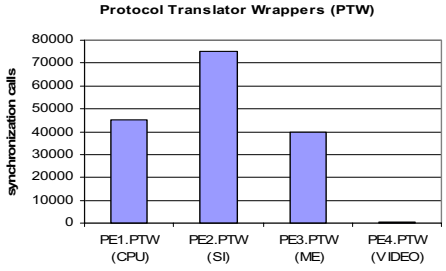



**Figure 8. Synchronization rate per processing elements**

In addition, this system architectural model is being used to verify refined models (RTL SystemC) of both a smart imaging and a motion estimator coprocessor, which are being developed during the project. System verification is performed by replacing the processing

elements where the corresponding functional tasks are mapped, for these more accurate coprocessors.

## 4. Conclusions

In this paper a SystemC-based simulator called CASSE is presented. CASSE provides fast simulations and easy architectural modeling by using transaction-level modeling techniques. Moreover, CASSE provides a seamless KPN-derived protocol refinement to cover from application to system implementation. Simulations at different abstraction levels are covered by CASSE, from completely untimed functional simulations of process networks to BCA/CA performance simulations of architectural models. Co-simulation with elements at different abstraction levels is also possible.

The work presented in this paper is being used in the European CAMELLIA project to derive an adequate HW/SW partition and an optimal communication infrastructure for the building blocks in the target CAMELLIA smart imaging system.

## 5. Acknowledgements

This work is sponsored by the European Commission in the IST-2001-34410 CAMELLIA project. The authors would like to thank their colleagues and partners involved in the project for their contributions and fruitful discussions.

## 6. References

[1] CAMELLIA webpage: www.iuma.ulpgc.es/camellia/
[2] K. Keutzer, S. Malik, R. Newton, J. Rabaey and S. Sangiovanni-Vicentelli, "System Level Design: Orthogonalization of Concerns and Platform-Based Design", in Proc. IEEE Transactions on Computer-Aided Design of Circuits and Systems, Vol. 19, No. 12, December 2000
[3] B. Kienhuis, E. Deprettere, K. Vissers and P. van der Wolf, "An Approach for Quantitative Analysis of Application-Specific Dataflow Architectures", in Proc. 11-th Int. Conf. on Application-specific Systems, Architectures and Processors, Zurich, Switzerland, July 14-16 1997
[4] J.A. Rowson and A. Sangiovanni-Vicentelli, "Interface-Based Design", in Proc. Of Design Automation Conference, Anaheim, California, June 1997
[5] Lukai Cai and Daniel Gajski, "Transaction Level Modeling: An Overview", in CODES+ISSS'03, California, USA, October 2003
[6] W. Cesario, D. Lyonnard, G. Nicolescu, Y. Paviot, S. Yoo, A.A. Jerraya, L. Gauthier and M. Diaz-Nava, "Multiprocessor SoC Platforms: A Component-Based Design Approach", in IEEE Design & Test of Computers, December 2002
[7] M-A. Dziri, W. Cesario, F. Wagner, and A.A. Jerraya, "Unified Component Integration Flow for Multi-Processor SoC Design and Validation", in Proc. DATE'04, Paris, February 2004
[8] T. Kogel, A. Wieferink, R. Leupers, G. Ascheid, H. Meyr, D. Bussaglia, M. Ariyamparambath, "Virtual Architecture Mapping: A SystemC based Methodology for Architectural Exploration of System-on-Chip Designs", in Int. Workshop on Systems, Architecture, Modeling and Simulation, samos, Greece, July 2003
[9] P. Lieverse, T. Stefanov, P. van der Wolf, E. Deprettere, "System Level Design with Spade: and M-JPEG Case Study", in Proc. ICCAD'2001, November 2001, San Jose, CA
[10] A. Nieuwland, J. Kang, O.P. Gangwal, R. Sethuraman, N. Busa, K. Goossens, R. Peset Llopis, and Paul Lippens, "C-HEAP: A Heterogeneous Multi-processor Architecture Template and Scalable and Flexible Protocol for the Design of Embedded Signal Processing Systems", *in Design automation for Embedded Systems*, Vol 7(3): 229–266, 2002, Kluwer
[11] M.J. Rutten, J.T.J. van Eijndhoven, and E.-J.D. Pol, "Eclipse: Heterogeneous Multiprocessor Architecture for Flexible Media Processing", in *Workshop on Parallel and Distributed Computing in Image Processing, Video Processing, and Multimedia (PDIVM'2002)*, Fort Lauderdale, USA, 2002, pp. 39–50
[12] T. Stefanov, P. Lieverse, E. Deprettere, P. van der Wolf, "Y-Chart Based System Level Performance Analysis: An M-JPEG Case Study", in *Proc. of the Progress Workshop*, 2000
[13] Functional Specification for SystemC 2.0.1, April 2002, http://www.systemc.org
[14] SystemC 2.0.1 Language Reference Manual
[15] Open Core Protocol Specification-v1.0. http://www.sonic.com, October 1999
[16] Virtual Component Interface Standard Version 2. On-Chip Bus DWG (OCB 2.2.0), http://www.vsi.org, April 2001
[17] ARM. AMBA AXI Protocol Specification, June 2003
[18] Harwig, Rick and Emile Aarts, "Ambient Intelligence: Invisible Electronics Emerging", in Proc. of the 2002 International Interconnect Technology Conference, San Francisco, pp. 3-5
[19] P. Klapproth, "Architectural Concept for IP re-use", in VLSI ASP DAC, December 2002