Application of Multi-domain and Multi-language Cosimulation to an Optical MEM Switch Design

Abstract

This paper presents the applicability of a cosimulation methodology based on an object-oriented simulation environment, to multi-domain and multi-language systems design. This methodology start with a system model given as a netlist of heterogeneous components and enables the systematic generation of simulation models for multi-domain and multi-language heterogeneous systems. For experiments, we used a complex multi-domains application: an optical MEM switch.

1. Introduction

Today's embedded systems are getting more and more heterogeneous including multi-domains (electronic, mechanical, and optical) components. Designing such systems is challenging in two aspects: heterogeneous design languages and high complexity of design to integrate heterogeneous components. Due to the heterogeneity of such systems, they are described using different domain-specific languages (e.g. C language and HDL for software and hardware design and MATLAB models of electro-mechanical sub-systems, etc.). To cope with the design complexity of refining heterogeneous components and their integration, the design process has a top-down flow from a high-level specification down to a low-level implementation. In this case, the validation of system design at any stage of the design flow is challenging. The key issue is the coordination between the different sub-systems that manipulate different communication concepts and are described using different design languages.

Recently optical MEMS's (micro-electro-mechanical systems) have been transitioning from abstract ideas to marketable products such as switching, scanning, projection, printing, etc. All optical-switches are becoming increasingly popular due to the many advantages that they propose over typical optical fiber optic switches (e.g. small, fast, reliable, and eventually will be inexpensive to produce) [1]. Today, MEMS become sub-systems of heterogeneous systems (e.g. complex controls are in charge of processors, communicating with MEMS through analogical hardware).

In this paper, we present a method of co-simulating such heterogeneous systems that include software processors, analog hardware and MEM sub-systems.

2. Related Work

Cosimulation is currently very popular for electronic systems design. Most of existing work presents methodologies for digital hardware-software cosimulation, e.g. N2C [2], Seamless CVE [3]. Ptolemy [4]

provides cosimulation for heterogeneous systems where heterogeneity concerns models of computation, e.g. process models, SR, FSMs, event-driven models, etc.

For MEMS design, academic research has created specialized tools for MEM modeling and simulation [5], [6], and has bridged the gap between CAD and foundry facilities. Chatoyant [7] is also a CAD tool for optical MEMS.

3. Conceptual framework

This section presents the conceptual framework of our cosimulation tool based on SystemC [8]. More specifically, we present the specification and execution models that we use for heterogeneous systems design.

3.1. *Specification models for heterogeneous systems*

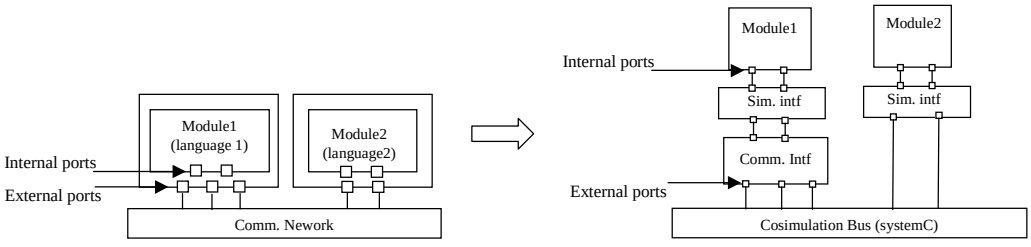
We represent systems as an ensemble of hierarchical modules communicating through communication channels. To separate communication and behavior, we use a concept of wrapper. To be specific, we use a wrapper for a module (1) described in an environment different from SystemC; (2) its communication protocol/abstraction level is different from the rest of the system. A module wrapper is an ensemble of two kinds of ports:

- internal ports (specific to the modules)
- external ports (connected to the communication channels)

Figure 1.a illustrates an example of a heterogeneous system specification. The wrapper concept enables the designer to easily specify the heterogeneous systems. For instance, to specify an optical module that is connected to the communication network, the interface of the optical module is specified as the internal port specification and that of the communication network as the external port specification.

3.2. Simulation models for heterogeneous systems

Due to the difference between the internal and external ports specification, the heterogeneous system specification may not be executable. To obtain simulation model for a heterogeneous specification, interfaces adapting the different simulators or communication protocols/levels have to be generated. These interfaces implement in fact the simulation models of the modules wrappers. In our approach, the simulation model of the wrappers consists in two types of interfaces:: (1) simulator interface - that adapt different simulators; (2) communication interface – that adapt different communication protocols/levels. Figure 1 presents an example of heterogeneous spsecification and its corresponding simulation model.



(a) Specification model (b) Simulation model

Fig. 1. System specification and corresponding simulation model

A simulator interface is required when the module behavior is simulated by a different simulator than the cosimulation bus in SystemC. It is specific to the used simulator.

A communication interface has a generic architecture composed of three basic elements:
- **module adapter** (MA) that provides the internal ports (e.g. FIFO port) with required communication services (e.g. fifo_write, fifo_read, etc.). It performs also data conversion and channel resolution. The channel resolution is required since the number of internal ports can be different from that of corresponding external ports.
- **internal communication media** (ICM) to transfer data between module adapter and channel adapter. ICM can be an RPC (remote procedural call) relationship (i.e. the MA calls RPCs provided by channel adapters).
- **channel adapter** (CA) that enables the module to access the external channel. To do that, after receiving a channel access request (via MA) from the module behavior, it uses channel communication services (e.g. read_hs in the case of hand-shake communication network, etc.). To each external port, a channel adapter is assigned.

To illustrate how these elements compose a communication interfaces, figure 2 shows the structural details of the communication interface of module 1 in figure 1.b.

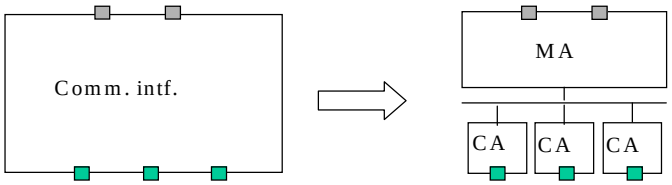


Fig. 2. An example of internal structure of communication interface

As shown in the figure, the communication interface has a module adapter that has two internal port connections and three channel adapters each of which has an external port connection. In the figure, the internal communication media is shown as a set of arrows each of which can be an RPC relationship.

3.3. Generation of simulation models from specification models

Simulation models are generated automatically using a library containing simulator interfaces and modules/channels adapters composing communication interfaces. The flow is illustrated in figure 3. As shown in the figure, given a specification model, we find a module to which a simulator interface and/or a communication interface are/is required. For each of such modules, we find a simulator interface specific to the module and/or compose a communication interface using a module adapter specific to the module and channel adapters that are specific to the communication channels connected to the module.

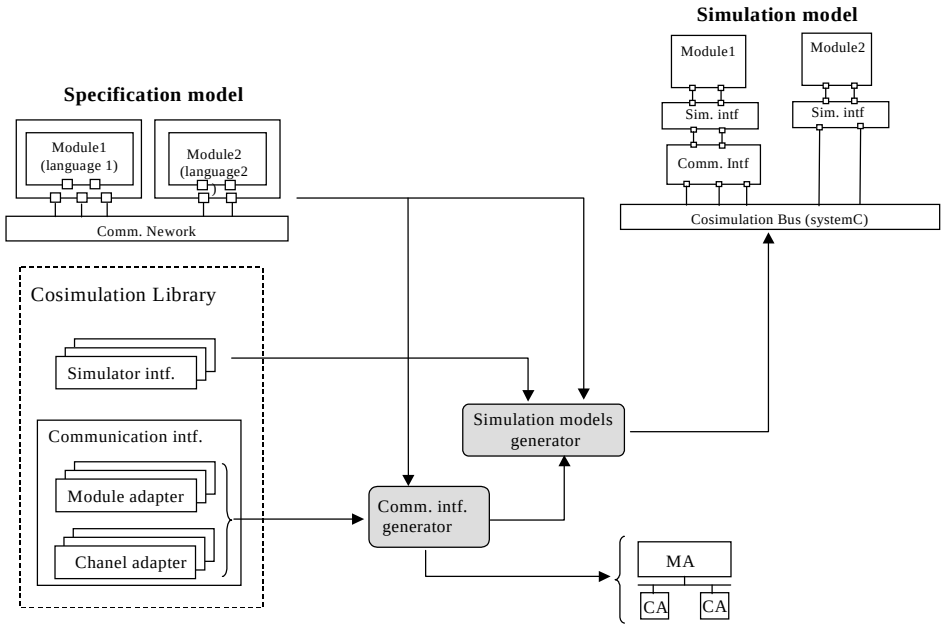


Fig. 3. Flow for the generation of simulation models

The details of the generation of the simulation model in the case of optical switch design will be given in Section 5 and 6.

4. Optical MEM Switch

We applied the presented concepts to the design of a complex heterogeneous system: the optical MEM switch. The global view of the system is presented in figure 4. The system is composed of three sub-systems:

(1) The control sub-system that will be finally mapped on a processor (control sub-system in figure 4). It calculates electronic orders (voltage) that commands motion of the mechanical mirrors composing the optical sub-system.

(2) The electro-mechanical converter that transforms the voltage from the output of the control sub-system into mechanical orders, in terms of distance, for each of the mirrors.

(3) The optical sub-system that is composed of a 2x2 mirror array, two light generators (G1, G2 in figure 4) and two beams detectors (D1, D2 in figure 4). Four lenses (L1-L4) are used to change the intensity and phase of the light.

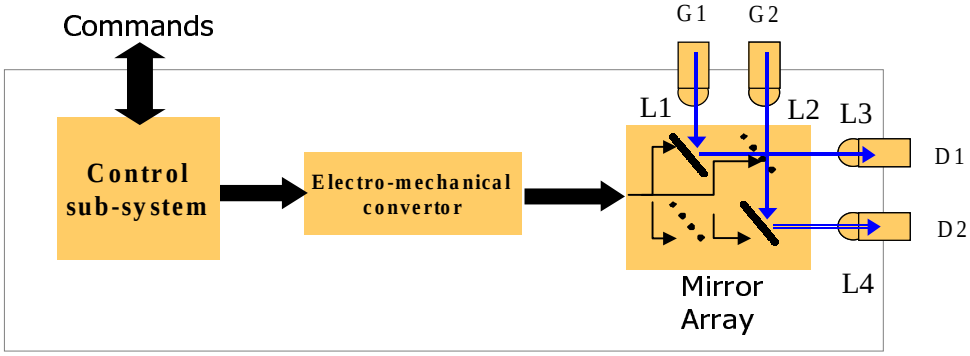


Fig. 4. Optical MEM switch

The switching operation of MEM optical switch is achieved through the mechanical motion of mirrors steering the data path from the inputs (G1 and G2 in figure 4) to the desired outputs (D1 and D2 in figure 4). This mechanical motion is controlled by the control sub-system.

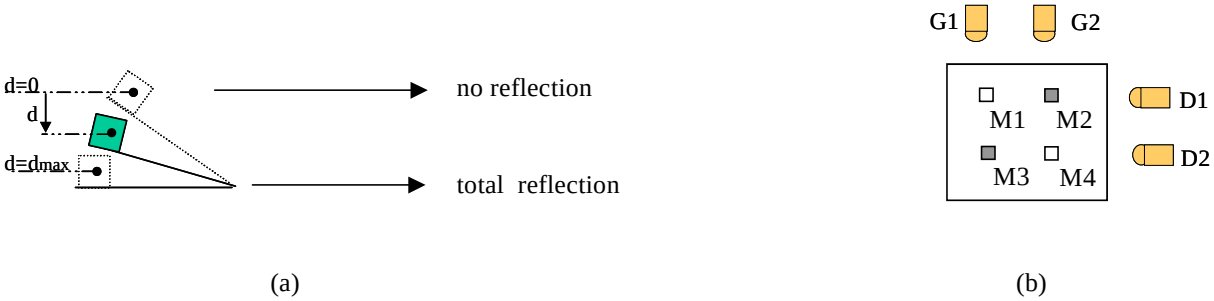
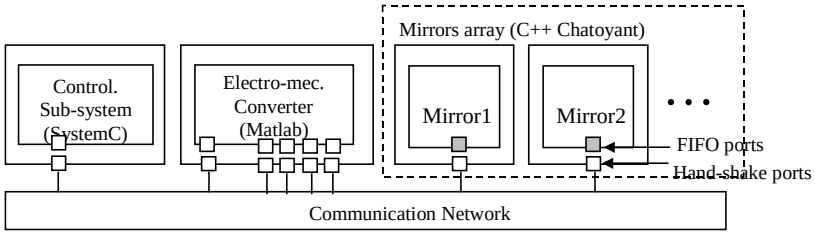


Fig. 5. Mechanical motion of mirrors according to the mechanical orders

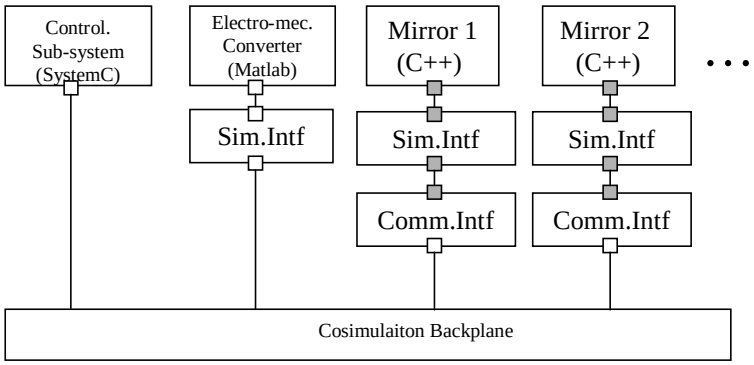
To perform switching operation, the control sub-system controls the reflection of the mirrors. As shown in figure 5.a, depending on the positions of mirrors, the mirrors may reflect totally (d=0), partially or to not reflect their inputs ($d = d_{max}$). Figure. 5.b shows four mirrors that have two inputs from two beam generators (G1 and G2) and two outputs to two light detectors (D1 and D2). In the figure, the control sub-system controls the reflection of each of mirrors in the manner shown in figure. 5.a.

We specified the control sub-system in SystemC and the electro-mechanical part in Matlab. For the behavior of optical devices (mirrors, lens, beam generators and detectors), we used C++ models from the libraries of Chatoyant [7].

The system specification is presented in figure 6.a. For the simplicity of the explanation, only two mirrors of the mirror array are shown in the figure. Modules in the system communicate through communication channels that encapsulate simple hand-shake protocols. Each mirror module receives control data (i.e. the reflection command) from the electro-mechanical converter with a FIFO communication protocol. To specify the interface of the optical modules and that of the communication channels, module wrappers have internal and external ports. As shown in figure 6.a, the internal ports specific to the module, are FIFO ports and the external ports connected to the communication channels are simple hand-shake ports.



(a) System specification model



(b) Simulation model

Fig. 6. Specification and simulation models for the optical MEM switch

5. Cosimulation of the Optical MEM Switch

5.1. Systematic Generation of the Simulation Model

To validate the system, the simulation model of the optical MEM switch is generated. Figure 6.b shows a generated simulation model of the system specification in figure 6.a.

- **Simulator interface**

Since Matlab-Simulink is used to model the electro-mechanical converter sub-system, to adapt Matlab-Simulink simulator to the SystemC cosimulation back plane, the corresponding simulator interface is generated. The simulator interface corresponding to each C++ Chatoyant model is very simple, consisting in standard SystemC interfaces that wraps C++ codes. To generate the simulator interfaces, we need necessary information from the system specification as follows.

- Reference to the behavior of the models
- Internal port specification
    - e.g. inputs/outputs and the correspondent communication protocols, data types, etc.

- **Communication interface**

The communication interfaces that adapt the communication specific to the mirrors model to the rest of the system are also generated. This is made by instantiating the three components, module adapter, channel adapter and internal communication media described in Section 3.2, from the cosimulation library. Figure 7 shows some code sections of module adapter and channel adapter of the mirror module wrapper. The module adapter of mirror module provides the module with simple FIFO communication primitives (e.g. write_fifo in the figure) called by the mirror module behavior. The channel adapter calls the communication primitives (e.g. read_hs) of the handshake communication channel. The module adapter and the channel adapter communicate through RPC (remote procedural call).

```
1.   Template <class T>;
2.   class MA_fifo<T>
3.   : public sc_module;
4.   public fifo_if //decl
5.   {public : sc_port<T> to_CA;
6.    private : fifo fifo_inst;}
7.   // fifo i/f implementation
8.   Void Write_fifo(T data){
9.     while(fifo_inst.getsize()>=size_max}
10.    {wait();}
11.  fifo_inst.push(data);
12.  }
13.  …
```

(a) Module Adapter

```
1.   Template <class T>;
2.   Class  CA_hand_shake<T>
3.   : public sc_module;
4.   public hs_i/f // decl
5.   {public : sc_port<T> to_CA;
6.   // hand_shake i/f implementation
7.   void read_hs(T data){
8.   get_hs(T_data)
9.   //only chanel primitive call
10.  }
11.  …
```

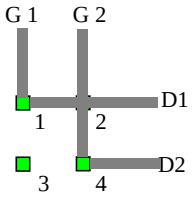
(a) Channel Adapter

Fig. 7. Pseudo-codes for module and channel adapters

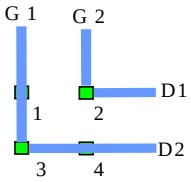
5.2. Cosimulation results

For the experiment we used a 2x2 mirror array that have inputs from two beams generators and outputs to two lights detectors (see figure 5.b).

We run the cosimulation of the MEM optical system. In the experiment, initially, the beam from G1 is detected by D1 and the beam from G2 is detected by D2. Figure 8.a shows the initial mirror configuration where two mirrors M1 and M4 are reflecting light from G1 to D1 by mirror M1 and from G2 to D2 by mirror M4. In the experiment, we simulate the execution of control commands that change the initial mirror configuration in figure 8.a to the mirror configuration of figure 8.b, where mirror M2 reflects the beam from G2 to D1 and mirror M3 reflects the beam from G1 to D2.



(a) Initial mirror configuration

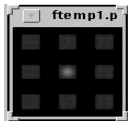


(b) After executing the control commands

Fig. 8. Change of mirrors configuration in the experiments

Each of beam generators and each of detectors can generate/detect upto nine beams. For a better examination of the results, we parameterize G1 and G2 differently: G1 generates one of nine beams, and G2 generates four of nine beams. Figure 9 a and b show the generated beams by G1 and G2. In the figures, nine

rectangles represent nine possible beam positions and dots represent generated beams. Figure 9.a shows that G1 generates one beam centered in the figure, G2 four beams at the positions as shown in the figure.



a) G 1

b) G2

Fig. 9. Outputs for the beam generators

The control sub-system sends commands to the mirrors by changing the electronic voltage assigned to each mirror. The electro-mechanic converter converts the electronic commands to the mechanic commands, i.e. the distance of mirror movement. Table 1 shows the voltage levels of electronic orders of the control sub-system and their correspondent mechanical orders converted in terms of distance for the mirror movement (as shown in figure 5.a). As shown in the table, to change the mirror configuration from total reflecting to non-reflecting, or vice versa, the mirror needs to be moved 400 μm by the commands of the control sub-system. To do that, the control sub-system gives eleven steps of command as shown in Table 1.

Voltage	Distance (μm)
0.0	0
12.7795	20
17.3641	40
20.3459	60
22.4453	80
23.9098	100
24.8782	120
25.4335	140
25.6295	160
25.6309	162
>25.6309	400

Table 1. Mechanical/electrical orders for data path steering

The evolution of data path steering, i.e. beam reflection by the mirrors, during the simulation is illustrated in figure 10. Each line of images corresponds to each of two detectors composing the optical sub-system. Initially, mirrors M1 and M4 steer the data path, by reflecting totally the beam received from G1 to D1 and the four beams from G2 to D2, respectively. Note that, in that case, mirrors M2 and M3 are not reflecting any beams. We can remark that at the first simulation step, D1 and D2 detect the outputs of G1 and G2, respectively, which are totally reflected by mirrors M1 and M4. During the simulation, mirrors change gradually their position according to the commands sent by the control part (in the Matlab simulator). For instance, at the second step of control commands, M4 changed its position and reflected partially - three of its four input beams. Consequently, D2 detected parts of the light generated by G2 (three beams). At the end of simulating eleven steps of command, mirrors M3 and M2 steer the data path, reflecting totally their inputs from G1 and G2, respectively. In this case, mirrors M1 and M4 are not reflecting any beams.

The simulation time is about 30 seconds. This enabled a fast validation of the overall system functionality, before its implementation in a final architecture.

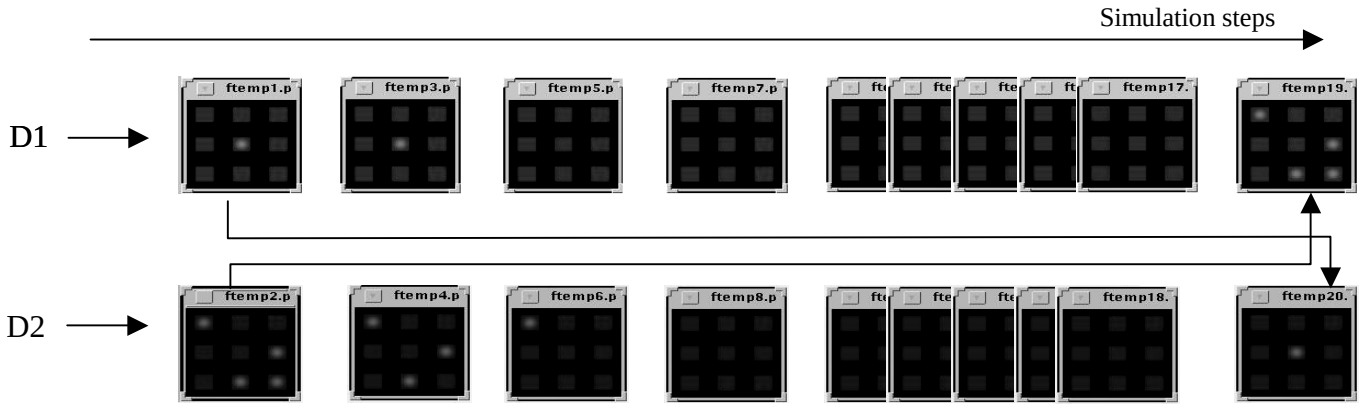


Fig..10. Results of cosimulation of the optical MEM switch

6. Conclusion

This paper has shown the applicability of a multi-domain and multi-language cosimulation methodology. This methodology enables the automatic generation of simulation models for the homogeneous specifications where the different modules may use different communication concepts or may be described in different languages. We applied the methodology for the cosimulation to a complex heterogeneous multi-domains application: an optical MEM switch.

References

[1] Wu, M.C., “Micromachining for Optical and Optoelelctronic Systems”, Proc. of the IEEE, Vol. 85 No. 11, Nov. 1997

[2] Coware, Inc., “N2C”, available at http://www.coware.com/cowareN2C.html

[3] R. Klein. 1996. Miami “A Hardware Software Co-Simulation Environment”, from RSP’96. IEEE CS Press. Pages 173-177.

[4] S. Lee and J.M. Rabaey “A hardware software cosimulation environment”, International Workshop on Hardware- Software Codesign”, Cambridge, oct. 1993.

[5] Senturia, S.D., “CAD for Microelectromechanical Systems”, Tranducers’95, June 25-29, 1995, Stockholm, Sweden, vol. 2.

[6] Wilson, N.M., et al., “A Heterogeneous Environment for Computational Prototyping and Simulation Based Design of MEMS devices”, SISPAD98, Leaven, Belgium, Sept., 1998.

[7] T. Kursweg, J. Martinez, S. Levitan, P. Marchand, D. Chairulli, “Dynamic Simulation of Opitcal MEM Switches”, DTIP, France, april, 2001.

[8] SystemC Consortium, “SystemC Version 2.O” available at http://wwww.systemc.org