Hybrid Modeling of Opto-Electrical Interfaces Using DEVS and Modelica

Victorino Sanz*, Shafagh Jafer, Gabriel Wainer**, Gabriela Nicolescu***, Alfonso Urquia*, Sebastian Dormido***
* **Dpto. Infomática y Automática, ETSI Informatica, UNED, Madrid, Spain**
{*vsanz,aurquia,sdormido*}*@dia.uned.es*
** **Dept. Computer Science and Engineering, Carleton University, Ottawa, ON, Canada**
{*sjafer,gwainer*}*@sce.carleton.ca*
***Dépt. Génie Informatique, École Polytechnique de Montréal, Montréal, QC, Canada**
gabriela.nicolescu@polymtl.ca

Keywords: Opto-electrical systems, DEVS, CD++, Modelica.

Abstract

We discuss two implementations of opto-electrical interfaces, their characteristics and functionalities using a hybrid M&S approach. These interfaces consist in a transmitter and a receiver, composed by electrical and optical parts, that translate electrical signals into optical impulses and viceversa. The first implementation, performed using the CD++ modeling environment, represents a discrete-event model of the system following the DEVS formalism. The other implementation uses the Modelica Standard Library to compose the electrical parts as a continuous-time model, and the Modelica DEVSLib library to describe the optical part as a discrete-event system. The obtained simulation results are equivalent in both implementations. They reproduce translation of a continuous-time sinusoid electrical signal into discrete optical impulses in the transmitter, and the opposite process in the receiver. These approaches simplify the development of multidomain hybrid systems and the study of ONoC at a high abstraction level.

1. INTRODUCTION

Opto-electrical interfaces, transmitters and receivers, can be used to translate the information contained in form of electrical current into light and viceversa [2]. These interfaces constitute the basic components of Optical Networks on Chip (ONoC) [5]. A transmitter in such interface receives an electrical current and translates it into optical impulses. The receiver receives the optical impulses and translates them into electrical current again. ONoC interfaces can be used to substitute electrical interconnects between processors in digital integrated circuits [15].

To facilitate the development of such opto-electrical communication systems, several approaches have been proposed to model and simulate the behavior of the circuits in order to better understand the phenomena involved. Depending on the abstraction level used to describe the system, FEM (Finite Elements Method) and FDTD (Finite-Difference Time-Domain) methods have been used to describe the optical behavior at the physical level [14].

On the behavioral and system levels, several authors use the VHDL-AMS hardware description language to describe the components of the system [12, 5, 14]. Also event-driven approaches have been used to describe components at system level, like SystemC [7, 6, 13] and OMNet++ [16].

The objective of our work is to develop models of basic opto-electrical interfaces in CD++ and in Modelica, compare their simulation results and discuss their functionalities. These models show the possibility to describe Systems on Chip (SoC), using the DEVS formalism [18], at a higher abstraction level than other existing techniques.

The description of each domain (electrical and optical) with DEVS models simplifies the multi-domain communication systems. This functionality can be extended in Modelica by describing the electrical part of the model as a continuous-time model, using DAE equations, and obtaining a more detailed hybrid model.

CD++ is a modeling tool that was defined using the DEVS and Cell-DEVS specifications [17]. With CD++, DEVS Atomic models can be programmed and incorporated onto a class hierarchy programmed in C++. Coupled models can be defined using a built-in specification language. CD++ makes use of the independence between modeling and simulation provided by DEVS, and different simulation engines have been defined for the platform.

Modelica is a modeling and simulation language mainly designed for mathematical modeling of physical systems [3]. It also has functionalities to manage discrete events [11].

The Modelica Standard Library contains several models from different domains (electrical, mechanical, thermodynamical, etc.) that can be used in order to build larger models composed by elements of the library [10]. DEVSLib is a Modelica library developed by the authors, that implements the Parallel DEVS formalism allowing the development of new atomic and coupled DEVS models. Models constructed using DEVSLib can be combined with any other Modelica model to compose hybrid continuous-discrete systems. The Dymola modeling environment has been used to develop the DEVSLib library and the Modelica models discussed in this manuscript [1]. The DEVSLib Modelica library is freely distributed and can be downloaded from the web [9].

A description of the modeled system, the basic opto-electrical communication system, is given in the next section. The model constructed using CD++ is described in Section 3., and the Modelica model is described in Section 4. At the end, a discussion of the experiment performed and the results obtained is included, together with some conclusions and ideas for future work.

2. COMMUNICATION BETWEEN THE OPTO-ELECTRICAL INTERFACES

The opto-electrical communication system is composed by a transmitter that sends optical information to the receiver, which recovers it and generates a current. These components are shown in Fig. 1.

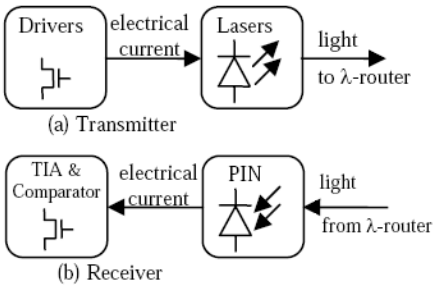


Figure 1. Basic opto-electrical interfaces.

The transmitter is composed by two components: a driver and a laser. The driver receives external information with the form of an electrical current, modulates it, and sends it to the laser. The laser receives the modulated current and generates optical impulses.

The receiver model is also defined by two components: a photodiode and a transimpedance amplifier (TIA). The photodiode receives the optical impulses and translates them into electrical current. The TIA amplifies the generated current and translates it into an electrical current.

The behavior of each of these components can be described using DEVS formalism as shown in [4]. Components are basically described as processors, that receive a message, perform a process to the information contained in the message and, depending on the defined behavior, send an output message containing the information processed. This output will be received as input signal by the next component. The models developed and discussed in this manuscript are based in the mentioned DEVS formalization.

3. CD++ MODEL

Listing 1 describes the coupled model defined for the opto-electrical communication system using CD++.

The transmitter is modeled as a coupled model composed by two models, Driver and Laser. The coupled model simply defines the models that it includes and connects their input/output ports to each other.

```
[top]
components : Transmitter
components : Receiver
in : indata
out : outdata
Link : indata data1@Transmitter
Link : light@Transmitter light1@Receiver
Link : data2@Receiver outdata
[Transmitter]
components : laser@Laser
components : driver@Driver
out : light
in : data1
Link : data1 data@driver
Link : current@driver current@laser
Link : light@laser light
[Receiver]
components : photodiode@Photodiode
components : aTIA@TIA
in : light1
out : data2
Link : light1 light@photodiode
Link : current@photodiode current@aTIA
Link : data@aTIA data2
[Transmitter]
processingTime : 00:00:02:00
[Receiver]
processingTime : 00:00:02:00
[laser]
processingTime : 00:00:30:00
[photodiode]
processingTime : 00:00:30:00
[aTIA]
processingTime : 00:00:02:00
[driver]
processingTime : 00:00:02:00
.
```

Listing 1. Transmitter-Receiver model definition file.

The Driver component accepts electrical voltage as input, performs modulation on it, and sends it out. As shown on the model definition file (Listing 1) the processing time of this component is set to 2 time units. This parameter can be changed easily to reflect different scenarios. In CD++, the input can be read from an ”event” file. The entries of the event file will be read one by one and inserted into the Driver component via its data input port. Once, input is received, it stays in the component by the time specified as processing time so that required processing is performed on the data. The modulation process that is performed by the Driver is varying the amplitude of the input carrier. The Driver reduces the amplitude of the voltage by dividing it by A to produce a modulated data that is the electrical current (performed by the outputFunction). After that, the output of the component is sent out through the ”current” output port. The output will serve as the input to the Laser component. Listing 2 includes a fragment of the Driver component.

```
/***********************************************
* Function Name: Driver
* Description: constructor
***********************************************/
Driver::Driver( const string &name )
: Atomic( name )
, data( addInputPort( "data" ) )
, current( addOutputPort( "current" ) )
, processingTime (0,0,2,0)
{
  string time1(
           MainSimulator::Instance().getParameter(
                 description(), "processingTime"));

  if( time1 != "" )
    processingTime = time1 ;
}
/***********************************************
* Function Name: externalFunction
* Description: the Driver receives data
***********************************************/
Model &Driver::externalFunction(
                        const ExternalMessage &msg)
{
  if ( this->state() == passive){
    if (msg.port() == data){
      DATA = msg.value();
      holdIn(active, processingTime);
    }
  }
  return *this ;
}
/***********************************************
* Function Name: internalFunction
***********************************************/
Model &Driver::internalFunction(
                           const InternalMessage &)
{
  passivate();
  return *this;
}
/***********************************************
* Function Name: outputFunction
***********************************************/
Model &Driver::outputFunction(
                        const InternalMessage &msg)
{
 //The modulation process.(Reducing the Amplitude)
   DATA = DATA / A;
   sendOutput( msg.time(),current, DATA);
   return *this;
}
```

Listing 2. Driver component source code.

On the other hand, the Laser component accepts the modulated current from the Driver as input (by invoking the external transition function), senses the current and generates impulses as output and sends it out (done by the output function). The Laser component source code is reflected in Listing 3.

Since, DEVS is discrete, the continuous electrical current is translated into a set of discrete inputs. The Transmitter implemented in CD++ generates the same results as if it was built in Modelica. Thus, the event file has discretized information of the input to the Transmitter, and the output file presents

```
/***********************************************
* Function Name: Laser
* Description: constructor
***********************************************/
Laser::Laser( const string &name )
: Atomic( name )
, current( addInputPort( "current" ) )
, light( addOutputPort( "light" ) )
, processingTime (0,0,30,0)
{
   string time1(
           MainSimulator::Instance().getParameter(
                description(), "processingTime"));

   if( time1 != "" )
      processingTime = time1 ;
}
/***********************************************
* Function Name: externalFunction
* Description: the Laser receives data
***********************************************/
Model &Laser::externalFunction(
                        const ExternalMessage &msg)
{
  if ( this->state() == passive){
    if (msg.port() == current){
      DATA = msg.value();//extract DATA from msg
      holdIn(active, processingTime);
    }
  }
  return *this;
}

/***********************************************
* Function Name: internalFunction
***********************************************/
Model &Laser::internalFunction(
                           const InternalMessage &)
{
  passivate();
  return *this;
}
/***********************************************
* Function Name: outputFunction
***********************************************/
Model &Laser::outputFunction(
                       const InternalMessage &msg)
{
  float temp = 0.94;
  if (DATA == temp && I == 0){
    I = 1;
    sendOutput( msg.time(),light, 2);
  } else if (DATA==temp && I==1){
      I =0; //filter out the second 0.9
      sendOutput( msg.time(),light, 1);
  } else sendOutput( msg.time(),light, 1);
      return *this;
}
```

Listing 3. Laser component source code.

00:00:00:00	data	0	00:00:32:000	light	1
00:01:00:00	data	0.17	00:01:32:000	light	1
00:02:00:00	data	0.34	00:02:32:000	light	1
00:03:00:00	data	0.5	00:03:32:000	light	1
00:04:08:00	data	0.64	00:04:40:000	light	1
00:05:00:00	data	0.77	00:05:32:000	light	1
00:06:00:00	data	0.87	00:06:32:000	light	1
00:07:00:00	data	0.94	00:07:32:000	light	2
00:08:00:00	data	0.98	00:08:32:000	light	1
(a)			(b)		

Table 1. a) Input data of the Transmitter in the form of discrete electric current; b) Output of the Transmitter in the form of optical impulse.

the optical impulses generated by the Laser component. The event and output file are presented in Table 1.

As seen in Listing 1, at time 00:07:00:00 an input of 0.94 (cross function value with amplitude of one) is received, it took 32 time units for the Transmitter to process this input and translates it into optical impulse via the Laser component. The second input of 0.94 must be filtered out. This is done in the source code of the Laser component.

The receiver coupled model is decomposed of two atomic components: Photodiode and TIAComparator. The Receiver receives the optical impulses, passes them through the Photodiode component where they get translated into current variation and finally the output of the Receiver appears as current at the output port of the Receiver. The Photodiode component receives optical impulses through its input port and translates them into current for the specific processing time which can be easily modified. On the other hand, the TIAComparator receives the output from the Photodiode as its input and amplifies this current and generates voltage. The source code associated with each of these atomic components is reflected in Listings 4 and 5.

As shown in the source code of the Photodiode model (Listing 4), the external transition function is invoked when LIGHT is input is received at the input port of this component. Then, a processing time of 30 time units is spent at this component until the output can be generated. The process of sensing the optical impulses (the input) and generating the associated current takes place at the output function where current is sent out of this component.

The source code of the TIAComparator model (Listing 5) shows how the external transition function is invoked when current is received at the input port of the model. After this, the input is amplified (which takes 2 time units for this action to complete), the output is generated and sent out by the output function. The amplification process is performed by increasing the amplitude of the input to produce voltage by multiplying the received current by an amplitude (performed at the outputFunction). In fact, the TIAComparator does the

```
/*******************************************
* Function Name: Photodiode
* Description: constructor
*******************************************/
Photodiode::Photodiode( const string &name )
: Atomic( name )
, light( addInputPort( "light" ) )
, current( addOutputPort( "current" ) )
, processingTime (0,0,30,0)
{
  string time1(
           MainSimulator::Instance().getParameter(
                description(), "processingTime")) ;

  if( time1 != "" )
    processingTime = time1 ;

}
/*******************************************
* Function Name: externalFunction
* Description: the Photodiode receives data
*******************************************/
Model &Photodiode::externalFunction(
                        const ExternalMessage &msg)
{
  if ( this->state() == passive){
    if (msg.port() == light){
      //extract DATA from msg
      LIGHT = static_cast < int > (msg.value());
      holdIn(active, processingTime);
    }
  }
  return *this ;
}
/*******************************************
* Function Name: internalFunction
*******************************************/
Model &Photodiode::internalFunction(
                          const InternalMessage &)
{
  if (I>=0 && I<=8 )
    holdIn(active, processingTime);
  else
    passivate();
  return *this;
}
/*******************************************
* Function Name: outputFunction
*******************************************/
Model &Photodiode::outputFunction(
                       const InternalMessage &msg)
{
  if (LIGHT == 1){
    I = 10;
    sendOutput( msg.time(),current, LIGHT);
  }else if (LIGHT == 2){
    if (I>=0 && I<=8)
      DATA = (10 - (I+1))/10;
    else
      DATA = 0;
    sendOutput( msg.time(),current, DATA);
    I++;
  }
  return *this;
}
```

Listing 4. Photodiode model definition file.

```
/*********************************************
* Function Name: TIA
* Description: constructor
*********************************************/
TIA::TIA( const string &name )
: Atomic( name )
, data( addOutputPort( "data" ) )
, current( addInputPort( "current" ) )
, processingTime (0,0,1,0)
{
  string time1(
          MainSimulator::Instance().getParameter(
                description(), "processingTime"));

  if( time1 != "" )
    processingTime = time1 ;

}
/*********************************************
* Function Name: externalFunction
* Description: the TIA receives data
*********************************************/
Model &TIA::externalFunction(
                        const ExternalMessage &msg)
{
  if ( this->state() == passive){
    if (msg.port() == current){
      DATA = msg.value();//extract DATA from msg
      holdIn(active, processingTime);
    }
  }
  return *this ;
}
/*********************************************
* Function Name: internalFunction
*********************************************/
Model &TIA::internalFunction(
                          const InternalMessage &)
{
  passivate();
  return *this;
}
/*********************************************
* Function Name: outputFunction
*********************************************/
Model &TIA::outputFunction(
                        const InternalMessage &msg)
{
//perform amplification by increasing the
//amplitude of the carrier.
  DATA = DATA * A;
  sendOutput( msg.time(),data, DATA);

  return *this;
}
```

Listing 5. TIAComparator model definition file.

reverse action of the Driver component.

4. MODELICA MODEL

In the Modelica model, the interfaces, the transmitter and the receiver, are coupled DEVS models constructed using the DEVSLib Modelica library.

Due to the current Modelica modeling capacities, the electrical components have been modeled with a continuous-time model, using the electrical components of the Modelica Standard Library. The system modeled with Modelica is shown in Fig. 2.

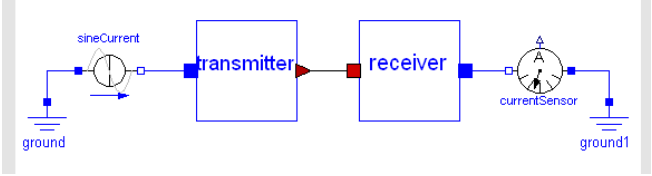


Figure 2. Basic opto-electrical communication system modeled with Modelica.

The transmitter contains the *driver* and the *laser* models, as it is shown in Fig. 3. The current received in the transmitter, through the electrical port, is passed to the *driver* model. To simplify the model, the received current is not modified by the *driver*. This situation can be easily changed to model different modulations and polarization of the electrical signal, including them in the *driver* model.

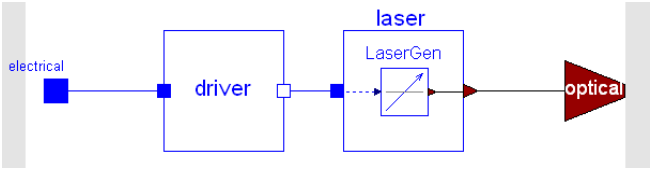


Figure 3. Opto-electrical transmitter modeled with Modelica

The *laser* receives the electrical current from the driver. It is composed by the *LaserGen* model, which translates the continuous electrical current into discrete-events, that represent the optical impulses.

The *LaserGen* model has been build using the CrossUP model included in the DEVSLib Modelica library, and shown in Listing 6. This model receives a continuous-time input signal(i.e., the electrical current) and a reference value, and generates a discrete-time event every time the signal crosses the reference value in upwards direction. This behavior represent the generation of an optical impulse every time the electrical current reaches a given value.

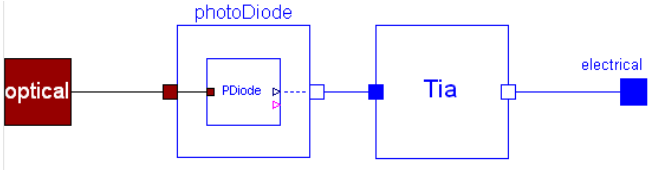


Figure 4. Opto-electrical receiver modeled with Modelica

The description of the receiver model is shown in Fig. 4. It

```
model CrossUP
  parameter Real Value = 1
        "Cross value";
  parameter Real Threshold = 1e-010
        "Detection value threshold";
  parameter Integer EType = 1
        "Type of the event to generate";
  Interfaces.outPort outport;
  Modelica.Blocks.Interfaces.RealInput u;
protected
  stdEvent e;
  Real x = der(u);
algorithm
  when (u > Value-Threshold) and x > 0
    and time > 0 then
    e.Type := EType;
    e.Value := u;
    e.Port := 1;
    sendEvent(outport.queue, e);
    outport.event := pre(outport.event) + 1;
  end when;
end CrossUP;
```

Listing 6. CrossUP model source code.

```
model DiCo
  Interfaces.inPort inport;
  RealOutput y;
  BooleanOutput change;
protected
  Boolean externalEvent( start = false);
  Integer receivedEvents;
  Integer eventType;
  stdEvent e;
  Boolean init( start = true);
algorithm
  when initial() and not(time>0) and pre(init) then
    inport.queue := CreateQueue();
    init := false;
  end when;
  when pre(externalEvent) then
    e := DEVSLib.SRC.getEvent(inport.queue);
    y := e.Value;
    change := not change;
    eventType := e.Type;
  end when;
equation
  receivedEvents= -integer(inport.event);
  externalEvent=(receivedEvents>pre(receivedEvents))
             or (receivedEvents<pre(receivedEvents));
end DiCo;
```

Listing 7. DiCo model source code.

is composed by the photodiode and the TIA amplifier. The optical impulses are received in the optical input port. These impulses are sent to the *photoDiode* model that translates each impulse in a current variation. The generated current is sent to the *Tia* model, which sends it through the electrical output port.

The photodiode model is composed by a *PDiode* that performs the translation from optical impulses into electrical signals. The Pdiode model has been composed using the DiCo (Discrete-to-Continuous) model included in the DEVSLib Modelica library, and shown in Listing 7. The DiCo model translates series of events into a real piecewise-constant signal, with values equal to the values of the received events. Each time the *PDiode* model receives an optical impulse it raises the generated current to the value of the received optical impulse. Bigger optical impulses generate higher current variations. This action simulates the excitation of the photodiode. After the optical impulse is received, the generated current decreases, because of the lack of optical excitation in the photodiode.

In order to simplify the system complexity, the *Tia* model does not modify the electrical signal. However, as in the case of the *driver* model, the *Tia* model can be modified using Modelica components, or equations.

5. EXPERIMENT AND RESULTS

As mentioned before, a simple example of opto-electrical communication system has been constructed using the previously described transmitter and receiver models. Since, DEVS is discrete, in CD++ the continuous electrical current is translated into a set of discrete inputs using quantized DEVS models. The input can be read from an external "event" file, which are transmitted to the driver component via its data input port. As shown on the model file (Fig. 1) the processing time of the driver model is set to 2 seconds. This parameter can be changed easily to reflect different scenarios.

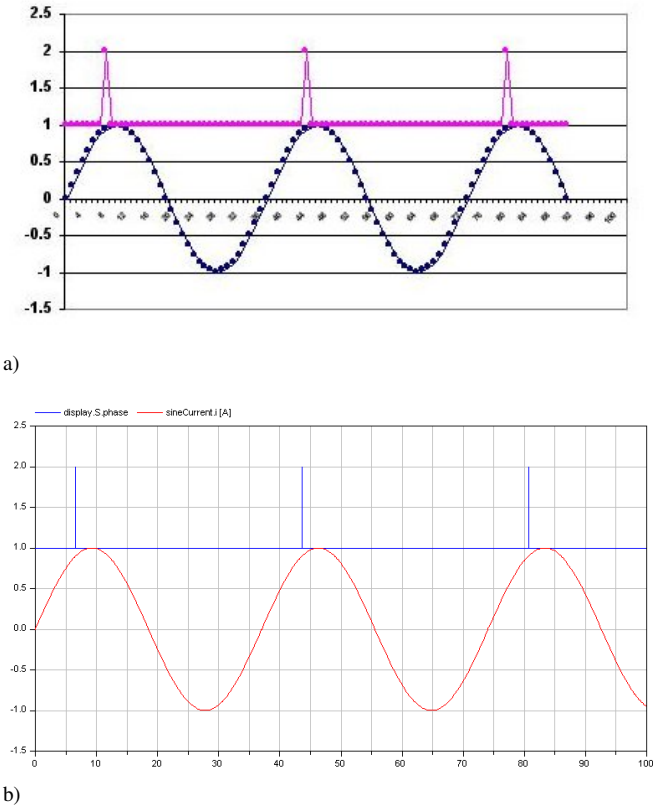
On the other hand, the input of the Modelica model is generated by a current source, available in the Modelica Standard Library. This current source generates a continuous-time sinusoid current. The sine current activates the optical generation in the transmitter, generating impulses.

In the Modelica model, the *LaserGen* has been configured as a cross-function with value 0.9, because the input current corresponds to a sine input with amplitude 1. The translation of the input current into optical impulses is shown in Fig. 5.

These impulses are received by the receiver and translated into current again. In the Modelica model, the derivate of the generated current is set to -0.5 to reproduce the lack of excitation. To monitor the generated current in the Modelica receiver, a current sensor has been included, also from the Modelica Standard Library. The reception of the optical impulses and its translation into electrical current is shown in Fig. 6.

The simulation results of the complete system are shown in Fig. 7. It can be noticed that no communication delays have been included in the models, but it will be easy to include them as additional atomic DEVS models.

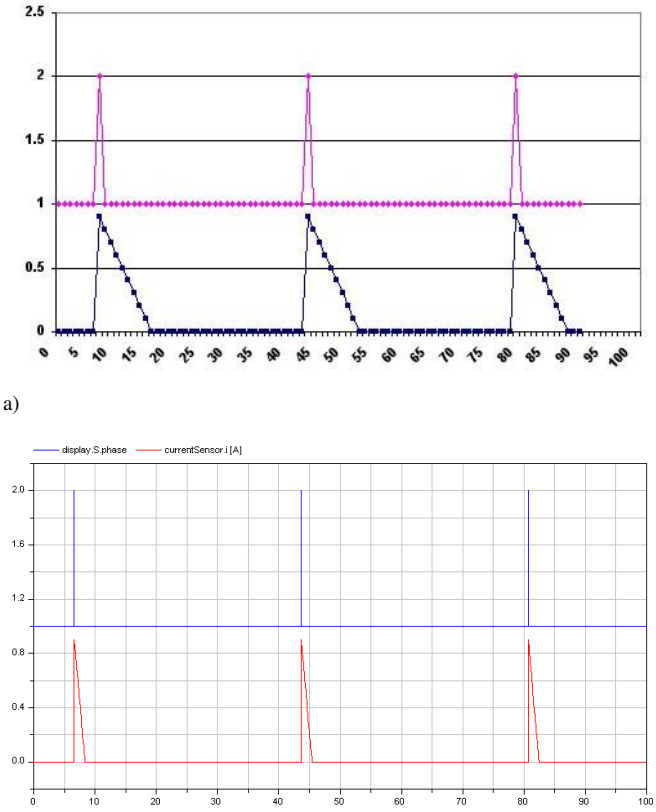
It can be noticed that the obtained results are very similar in both models.



a)

b)

Figure 5. Sinusoid electrical current transformed into optical impulses, modeled with: a) CD++ and; b) Modelica.



a)

b)

Figure 6. Optical impulses translated into current by the receiver, modeled with: a) CD++ and; b) Modelica.

6. CONCLUSIONS

The DEVS formalism is suitable to describe multiple kind of systems, and specially multidomain systems. Two different implementations of a basic opto-electrical communication system, using the DEVS formalism, have been developed, obtaining equivalent results in both cases. In the case of the Modelica implementation, it can be noticed that the development of hybrid multidomain systems is simplified by the use of already available libraries of components. The developed components and models can be reused to implement larger systems, and study the ONoC at a high abstraction level.

Future work ideas include the development of more acurate models of the opto-electrical interfaces. Also, a library of opto-electrical components can be developed to simplify the development and study of this kind of systems.

REFERENCES

[1] Dynasym AB. Dymola Dynamic Modeling Laboratory User's Manual. http://www.dymola.com/, 2006.

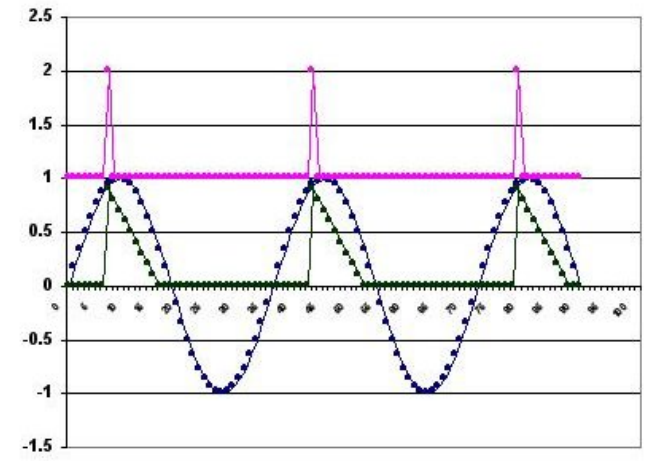
[2] G. P. Agrawal. *Fiber-Optic Communication Systems (2nd ed.)*. Wiley-Interscience, New York, NY, USA, 1997.

[3] Modelica Association. Modelica Language Specification. http://www.modelica.org/documents, 2007.

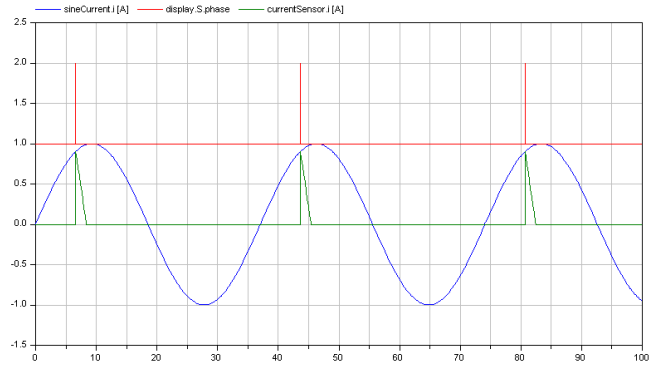
[4] M. Biere, L. Gheorghe, G. Nicolescu, I. O'Connor, and G. Wainer. Towards the high-level design of optical networks-on-chip. formalization of opto-electrical interfaces. *Electronics, Circuits and Systems, 2007. ICECS 2007. 14th IEEE International Conference on*, pages 427–430, Dec. 2007.

[5] M. Brière, L. Carrel, T. Michalke, F. Mieyeville, I. O'Connor, and F. Gaffiot. Design and behavioral modeling tools for optical network-on-chip. In *DATE '04: Proceedings of the conference on Design, automation and test in Europe*, page 10738, Washington, DC, USA, 2004. IEEE Computer Society.

[6] M. Brière, B. Girodias, Y. Bouchebaba, G. Nicolescu, F. Mieyeville, F. Gaffiot, and I. O'Connor. System level assessment of an optical noc in an mpsoc platform. In *DATE '07: Proceedings of the conference on Design, automation and test in Europe*, pages 1084–1089, San Jose, CA, USA, 2007. EDA Consortium.



a)



b)

Figure 7. Opto-electrical communication system, modeled with: a) CD++ and; b) Modelica.

[7] Matthieu Briere, Emmanuel Drouard, Fabien Mieyeville, David Navarro, Ian O'Connor, and Frederic Gaffiot. Heterogeneous modelling of an optical network-on-chip with systemc. In *RSP '05: Proceedings of the 16th IEEE International Workshop on Rapid System Prototyping*, pages 10–16, Washington, DC, USA, 2005. IEEE Computer Society.

[8] Francois E. Cellier and Ernesto Kofman. *Continuous System Simulation*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[9] http://www.euclides.dia.uned.es/.

[10] Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. Wiley-IEEE Computer Society Pr, 2003.

[11] Sven Erik Mattsson, Martin Otter, and Hilding Elmqvist. Modelica Hybrid Modeling and Efficient Simulation. In *Proceedings of the 38th IEEE Conference on Decision and Control*, pages 3502–3507, 1999.

[12] F. Mieyeville, M. Brière, I. O'Connor, F. Gaffiot, and G. Jacquemod. A vhdl-ams library of hierarchical optoelectronic device models. *Languages for system specification: Selected contributions on UML, systemC, system Verilog, mixed-signal systems, and property specification from FDL'03*, pages 183–199, 2004.

[13] I. O'Connor, F. Tissafi-Drissi, D. Navarro, F. Mieyeville, F. Gaffiot, J. Dambre, M. de Wilde, D. Stroobandt, and M. Briere. Integrated optical interconnect for on-chip data transport. *Circuits and Systems, 2006 IEEE North-East Workshop on*, pages 209–209, June 2006.

[14] Ian O'Connor. Optical solutions for system-level interconnect. In *SLIP '04: Proceedings of the 2004 international workshop on System level interconnect prediction*, pages 79–88, New York, NY, USA, 2004. ACM.

[15] Behzad Razavi. *Design of Integrated Circuits for Optical Communications*. McGraw-Hill, New York, NY, USA, 2002.

[16] A. Shacham, K. Bergman, and L.P. Carloni. On the design of a photonic network-on-chip. *Networks-on-Chip, 2007. NOCS 2007. First International Symposium on*, pages 53–64, May 2007.

[17] Gabriel Wainer. CD++: A Toolkit to Develop DEVS Models. *Softw. Pract. Exper.*, 32(13):1261–1306, 2002.

[18] Bernard P. Zeigler, Tag Gon Kim, and Herbert Praehofer. *Theory of Modeling and Simulation*. Academic Press, Inc., Orlando, FL, USA, 2000.