Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2008, Article ID 134798, 11 pages
doi:10.1155/2008/134798

Research Article

Modelling Field Bus Communications in Mixed-Signal Embedded Systems

Mohamad Alassir, Julien Denoulet, Olivier Romain, Abraham Suissa, and Patrick Garda

Université Pierre et Marie Curie – Paris 6, EA 2385, BC 252-4 Place Jussieu, 75252 Paris Cedex 05, France

Correspondence should be addressed to Julien Denoulet, julien.denoulet@upmc.fr

Received 1 October 2007; Revised 29 February 2008; Accepted 10 June 2008

Recommended by Christoph Grimm

We present a modelling platform using the SystemC-AMS language to simulate field bus communications for embedded systems. Our platform includes the model of an I/O controller IP (in this specific case an I^2C controller) that interfaces a master microprocessor with its peripherals on the field bus. Our platform shows the execution of the embedded software and its analog response on the lines of the bus. Moreover, it also takes into account the influence of the circuits's I/O by including their IBIS models in the SystemC-AMS description, as well as the bus lines imperfections. Finally, we present simulation results to validate our platform and measure the overhead introduced by SystemC-AMS over a pure digital SystemC simulation.

1. INTRODUCTION

Nowadays, the design of embedded systems is a key issue for the electronics industry [1]. Many ongoing research projects address this issue by increasing the design abstraction level from the circuit to the system level [2]. However, most embedded systems gather analog and digital electronics with embedded processor cores running large amounts of embedded software.

A number of issues are unfortunately discovered after the conception of embedded systems prototypes. These issues include for example coupling between analog and digital functions or unexpected hazard resulting from unexpected low level effects. Of course, discovering such issues at the prototyping step and revising the design and its implementation to get rid of them is time and money consuming. Addressing these issues at the design time requires a mixed-signal model gathering digital and analog functions [3]. A number of approaches exist to provide these mixed-signal models. They are considering different levels of abstraction and different models of computations. These levels and models are simulated with different tools and cosimulation is used to cover the relations between them.

To be more specific, we observe that neither VHDL-AMS nor SystemC are perfectly well suited to model the architecture of these embedded systems. On the one hand, VHDL-AMS is perfect to model analog and digital electronics, but the simulation speed of processor cores is too slow in VHDL for the simulation of embedded software [4]. On the other hand, SystemC is perfect to model digital electronics and processor cores executing embedded software, but it cannot model analog electronics [5].

These observations led to the proposal of SystemC-AMS, for which beta releases of the language and the simulation engine are now available [6]. SystemC-AMS provides, in addition to the SystemC discrete-event simulation kernel, continuous-time models of computations for conservative and nonconservative system modelling [7–9]. Moreover, a specific AMS working group was created by OSCI [10] in 2006 to specify SystemC-AMS [11].

We want to investigate the potential of SystemC-AMS to provide a unified framework for the design of embedded systems. It could give the advantages of multiple levels and multiple computational models in a single language based on an open standard. Of course, as SystemC-AMS is still under specification, it is unclear as of today if this is the most valid approach, or if it provides significant gains over other approaches, but the objective of our work presented in this paper is to contribute to this specification and to provide the

AMSWG with some applications of mixed-signal embedded systems.

Considering the long-term challenge we are addressing and the need for specific performances data to provide a significant insight, we chose a pragmatic step-by-step research strategy, and we begun with a rather simple case of mixed-signal embedded system.

In many different applications, (automotive, industrial control, etc.), embedded systems are typically formed by a set of nodes connected through field busses such as I^{2}C or CAN. The nodes are typically mixed analog and digital SOCs, including one (or several in the near future) processor core(s) and an array of peripheral interfaces. In its most usual form, the node is simply a microcontroller, connected to various sensors or actuators.

In this paper, we show how to model the field bus communications between the nodes of an embedded system. Our methodology is based on three main parts: firstly, a generic architecture of an I/O controller realizing the interface between the processor cores and an I^{2}C field bus; secondly, a generic architecture for the analog part of this interface, and a software tool to get the parameters of this interface from industrial IBIS files; thirdly, a generic model of the bus lines. Hence, our method is generic and it can be applied to any field bus. This paper is based on our previous work on the design and modelling of an I^{2}C bus controller in VHDL [12] and SystemC-AMS [13, 14]. It is an extended revision of the (unpublished) paper presented in [15].

For didactical purposes, the paper is organised around a platform presented in Figure 1. It represents a common situation, where a microcontroller is connected to a System-on-Chip through a field bus. Our model includes discrete-event models for the digital cores, implemented in SystemC, and continuous-time models for the analog interfaces and for the bus lines, implemented in SystemC-AMS. Whereas this methodology is generic, we will present it in the specific case of the I^{2}C and CAN busses.

The paper is divided into three main sections. Section 2 describes the architecture of the digital I/O controller. Section 3 presents the architecture of the analog I/O, the method to include data from an IBIS model into an equivalent SystemC-AMS description of a circuit's I/O buffers, and the electrical model of the bus lines. Section 4 gives several simulation results of field bus communication on the platform and estimates the overhead of the analog parts of the models compared to a totally digital simulation.

2. DIGITAL MODELS

The modelling platform discussed below is presented in Figure 2. It is centered on a field bus that connects four nodes: among them two can be masters on the bus, while the remaining two can only be slaves. The two masters are a 8051 microcontroller in association with a bus controller and a MIPS-based system on Chip. The two slaves are memory devices which will be accessed in the course of simulation by the two masters. Each node is modeled with a mixed SystemC/SystemC-AMS description, where a device's digital core is modeled in SystemC and its analog interface is modeled in SystemC-AMS. This section introduces the digital architecture of the platform components, especially focusing on the bus controller.

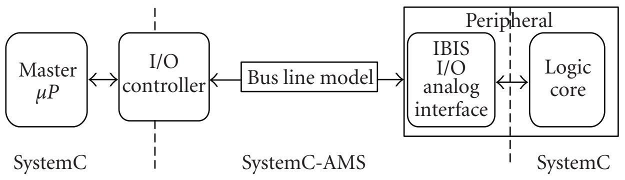

FIGURE 1: General structure of the modelling platform.

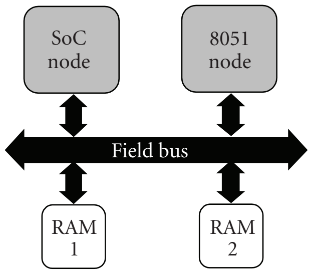

FIGURE 2: Field-bus based modelling platform.

2.1. Bus controller

We first decided to model our platform around an I^{2}C bus, mainly because this standard provided a simple protocol and an easy-to-implement mixed-signal interface. However, it seemed appropriate to work on a generic structure that could be applied to different protocols with only a minimum of changes. Thus, this subsection presents our controller architecture applied to the case of an I^{2}C bus.

2.1.1. I^{2}C protocol

Historically, the first I^{2}C (Inter-Integrated Circuit) controllers and the I^{2}C protocol were designed by Philips at the beginning of the eighties for television applications [16]. The I^{2}C bus was invented to provide communication on a two-wire bidirectional bus—a serial data (SDA) and clock (SCL)—between a small number of devices (sensors, LCD, microcontroller, etc.). Transfer frequency is up to 100 kbits/s for standard mode, and up to 3.4 Mbits/s in high-speed mode.

Data frame for the standard mode is made of a start bit, a 7 bits address, a Read/Write bit, an acknowledge bit (ACK), and a sequence of data bytes. Each data byte is followed by an acknowledge bit issued by the target device. A stop bit finalizes the transmission (Figure 3). Each bit is transmitted on SDA in conjunction with the SCL clock.

(i) Transfers are initiated by a START condition. It happens when a falling transition occurs on the SDA line while SCL is high.

(ii) Transfers end with a STOP condition. It happens when a rising transition occurs on the SDA line while SCL is high (Figure 4).

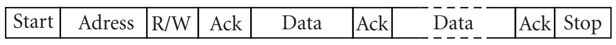

Figure 3: I^{2}C data frame.

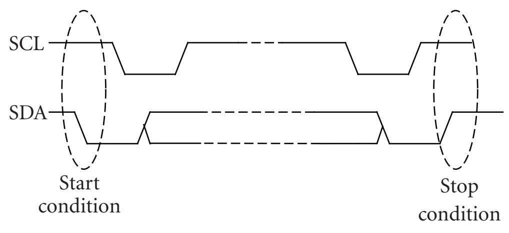

Figure 4: Start and stop condition.

(iii) Data is considered valid during the high state of SCL. Therefore, SDA signal must remain stable during this half period (Figure 5).

(iv) I^{2}C allows multimaster communications and features an arbitration management protocol for such transmissions. Arbitration takes place on the SDA line, while the SCL line is at the high level. Bus control is given to the master which transmits a low level while the others transmit a high level. A master which loses arbitration switches to high impedance its data output stage (Figure 6).

As a specification for our I^{2}C controller model, we chose the PCF8584 designed by Philips/NXP [17].

2.1.2. I^{2}C controller digital block architecture

The architecture of the digital block is divided into three blocks (Figure 7).

(i) A microprocessor interface handles communication with the microprocessor core. It is built around a FIFO that stores the successive requests coming from the microprocessor bus (8051 bus, VCI, AMBA, etc.). When the controller has finished a transaction on the bus, and if a request is stored in the FIFO, the FIFO is read and the interface extracts the information needed by the sequencer (type of operation, address, data) to perform the new communication. The interface also includes an interrupt line connected to the microprocessor core so that it can read a data received from the I^{2}C bus.

(ii) The core of the controller is a sequencer which translates the request from the master into a detailed sequence respecting the I^{2}C protocol (frame generation, byte transmission, or reception, etc.).

(iii) Finally, a signal generator module manages or drives the SCL and SDA bus lines (Figure 8). This block manages the acknowledge generation/detection depending on the operating mode (transmitter or receiver). A shift register either serializes data to be sent to the bus line in transmitting mode or collects information from the bus in reception mode. It also sets the transmission frequency by dividing the system clock with a user-defined constant. The SDA and SCL signals from this block will be interfaced with the analog model presented in Section 3.

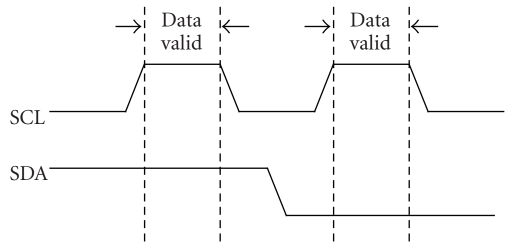

Figure 5: Data validity period.

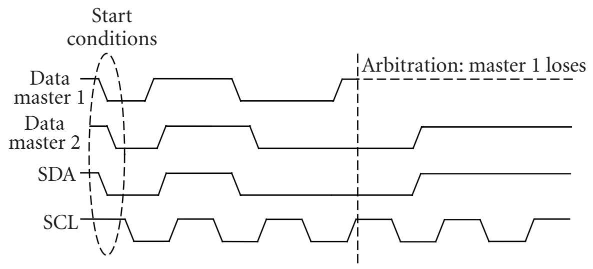

Figure 6: Arbitration procedure with two masters.

2.2. Other devices

To develop the SoC node of the platform, we used a virtual prototyping platform for system-on-chips called SoCLib [18]. SoCLib provides a number of IP SystemC models (including processor cores, VCI interconnect, memory devices, I/O controllers, timers, etc.), to easily simulate an application on a user-assembled SoC or MPSoC architecture built around a virtual component interface (VCI) bus [19].

We added our controller model to the SoCLib library and used it to connect an MIPS processor core to an external field bus. To do so, a SystemC wrapper has been developed to interface our controller to the VCI bus provided by SoCLib. The platform also includes a RAM component used by the MIPS to store its code and data and a TTY terminal for debug purposes. A SoCLib-generated platform synopsis can be found in Figure 9.

We will not detail the remaining models, the I^{2}C RAM, and the 8051 microcontroller [20]. The first of this model is a basic description of a memory component while the second one is a standard description of this well-known circuit.

3. ANALOG MODELLING

We introduce in this section the analog modules of our platform. All these devices were modeled in SystemC-AMS v.0.15. First, we show the analog interface of the bus controller, designed according to the requirements of the I^{2}C protocol. Then, for the slave devices, we present an interface retrieved from an IBIS model of a standard component and show how such IBIS models can be automatically translated into a SystemC-AMS description. Finally, we focus on the transmission lines, introducing a model that takes into account the wires' physical imperfections as well as the mutual influence of adjacent transmission lines.

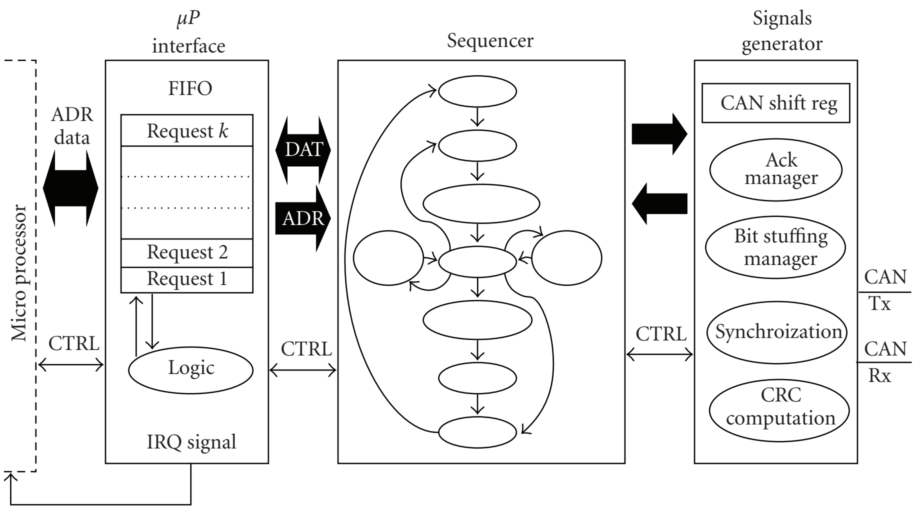

Figure 7: Controller digital block architecture.

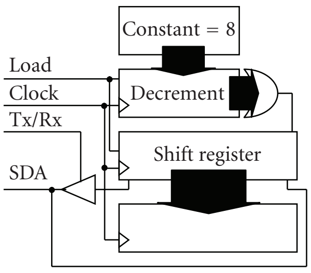

Figure 8: SDA line manager.

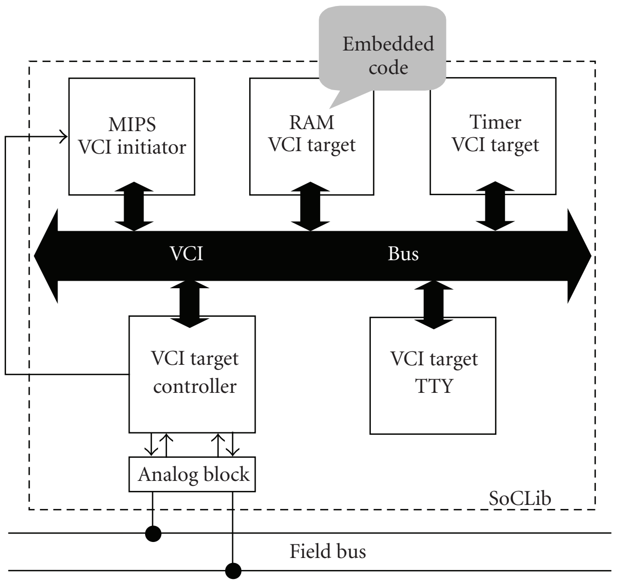

Figure 9: SoCLib simulation platform.

3.1. Bus controller

Specifications of the I^{2}C protocol indicate that the devices must have open-drain or open-collector outputs (depending on technology) in order to perform a wired-AND function to manage multimaster mode. Both lines also feature a pullup resistor to VCC (Figure 10(a)).

We modeled this analog interface with the SystemC-AMS 0.15 library, by instantiating linear elements (resistors, capacitors, voltages sources, etc.). Figure 10(b) represents the interface model for the SDA line. It translates the logic levels sent by the digital block to voltages across the bus lines. The SCL model is similar.

In this model, the transistor is represented with an interrupter and a resistor as no transistor model is available in SystemC-AMS 0.15. A 20 pF capacitor is used to manage the rising and falling times of the SDA signal. Finally, a pullup resistor sets the line at a high state when no command is applied on the bus. To read a value coming from the bus, a threshold detection is applied to the SDA line and it provides a logical value to the digital block.

Algorithm 1 gives a SystemC-AMS sample code of the analog interface for either line of the bus. The electrical parameters of the devices (including the output transistor) were chosen to be in accordance with specifications found in NXP's PCF8584 controller datasheet [17].

3.2. IBIS to SystemC-AMS conversion

In order to accurately model the analog behaviour of the components connected on the bus, and integrate real parameters of industrial components to our SystemC-AMS models, we developed an IBIS-to-SystemC-AMS conversion tool. It parses a standard IBIS source file to produce a SystemC-AMS module we can directly instantiate in our platform. This section briefly presents the IBIS standard and then shows how these models can be used in our simulation platform.

3.2.1. IBIS specifications

The input/output buffer information specification (IBIS) [21] standard was originally introduced to give a behavioural

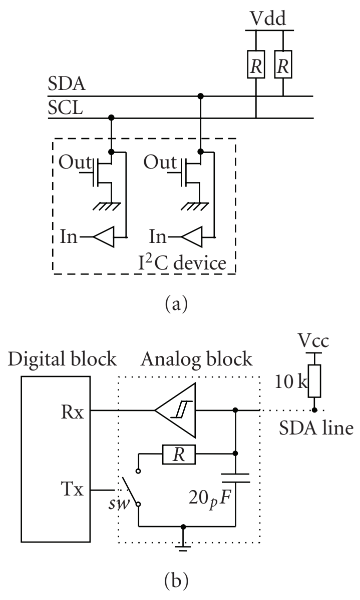

Figure 10: (a) I^{2}C electrical scheme (b) SystemC-AMS model.

description of a circuit's inputs/outputs. This behaviour is based on voltage/current curves obtained from measurements or full simulation. It especially takes into account the influence of the chip's package on the I/O signal form.

An IBIS model is presented as a text file which includes for each input or output pad an R, L, C equivalent circuit modelling the influence of the package as well as another circuit representing the response of the pad to a rising or falling edge transition.

Figure 11 presents a model of an output buffer (input buffers are modeled in a similar way). The voltage/current curves included in the IBIS file give the response of the CMOS inverter to a logic transition. However, the IBIS standard specifies that this response is equivalent to another R, L, C circuit. As a result, a SystemC-AMS module of the electrical scheme presented in Figure 12 can accurately model the behaviour of the output buffer. The figure also shows how the module can be interfaced with a SystemC description of the logic core of the component.

3.2.2. IBIS-to-SystemC-AMS conversion

To add IBIS models in our SystemC-AMS simulation platform, we developed an IBIS file analyzer program which parses the IBIS file to extract the I/O circuits useful information and then automatically creates a SystemC-AMS module using these parameters.

An IBIS file is made of several sections that first present general information about the chip, display the R, L, C values of the circuit's package, enumerate the different I/O pins, and specify their type (input, output, GND, etc.), then they describe the behaviour of each I/O type found in the chip. A sample of an IBIS file is presented in Algorithm 2. Each section is specifically introduced by a keyword (such as [Package] or [Pin] in the file sample). The aim of our tool

```
SCA_SDF_MODULE(ADC){    // Threshold Detector
   sca_sdf_in<double> in;
   sc_out<bool> out;
   ⋮
   void init(){}
   void sig_proc(){ if(in.read() > 2.5) out = 1;
                    else out = 0;}
   ⋮
   SCA_CTOR(ADC){}
};
   ⋮
SCA_SDF_MODULE(data_conv){
   sca_sdf_in<double> portin;
   sc_out<double> portout;
   ⋮
   double portout_;
   void init(){ portout = 1.0e3;}
   void sig_proc(){ portout_ = portin.read();
                    portout = portout_ * 3.0e2;}
   ⋮
   SCA_CTOR(data_conv){}
};
   ⋮
SC_MODULE(SDA_Mgr){
   sc_in<bool> sda_in;      // From digital block
   sc_out<bool> sda_out;  // To digital block
   sca_sdf_signal<double> sda_sdf; // To I2C Bus
   ⋮
   sc_signal<double> r_value;
   sc_signal<bool> sig_;
   sca_elec_port out;    // Output port
   sca_elec_port gnd;    // Ground port
   sca_elec_node y;      // Internal node
   ⋮
   sca_sc2r *r1;               sca_c *c1;
   sca_rswitch *sw1;           sca_vd2sdf *conv1;
   data_conv *data_conv1;      can *can1;
   ⋮
   SC_CTOR(SDA_Mgr){
   ⋮
     ADC1 = new ADC("ADC1");    // Threshold
          ADC1->in(sda_sdf); ADC1->out(sda_out);
   ⋮
       sw1 = new sca_rswitch("sw1");    // Switch
       sw1->off_val = true; sw1->p(y);
       sw1->n(gnd); sw1->ctrl(sda_in);
   ⋮
     r1 = new sca_sc2r("r1");    // Resistor
         r1->ctrl(r_value); r1->p(out); r1->n(y);
   ⋮
     data_conv1 = new data_conv("data_conv1");
       data_conv1->portin(sda_sdf);
       data_conv1->portout(r_value);
   ⋮
     c1 = new sca_c("c1");    // Capacitor
       c1->value = 20.0e − 12;
       c1->p(out); c1->n(gnd);
   ⋮
     conv1 = new sca_vd2sdf("conv1"); // Output voltage
       conv1->p(out); conv1->n(gnd);
       conv1->sdf_voltage(sda_sdf);
     }
};
```

Algorithm 1: I^{2}C controller analog block code sample.

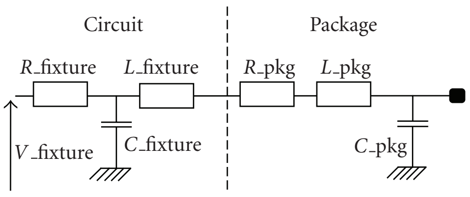

Figure 11: IBIS output model.

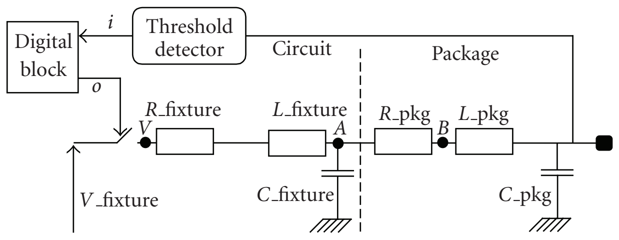

Figure 12: IBIS equivalent circuit to model in SystemC-AMS.

```
[Package]
| variable      typ         min         max
R_pkg           0.20        100.00 m    0.40
L_pkg           7.00 nH     4.10 nH     10.00 nH
C_pkg           1.50 pF     1.00 pF     3.00 pF
|
|************Pin Defintions***********
[Pin]           signal_name     model_name
|
1               A0              InputModel1
2               A1              InputModel1
3               A2              InputModel1
4               GND             GND
5               SDA             IOModel
6               SCL             InputModel2
7               WP              InputModel3
8               VCC             POWER
```

Algorithm 2: IBIS text file sample.

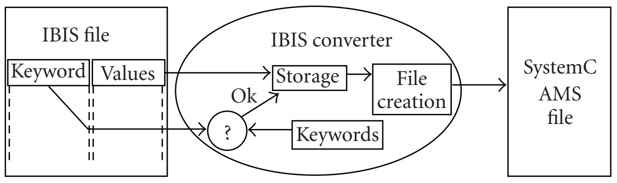

Figure 13: IBIS-to-SystemC-AMS converter tool synopsis.

is to scan the IBIS file, look for those keywords, and extract useful information, in particular the number of each I/O type as well as each type's electrical characteristics, that is, the value of the R, L, C elements.

The synopsis of our IBIS-to-SystemC-AMS converter is presented in Figure 13. When running, it performs a scanning phase followed by a generation phase.

When it parses the IBIS file, the program first counts the number of different I/O models for the circuit as well as their number of occurrences. Then, for each I/O model, it extracts and stores the value of the R, L, C elements. To do so, the program compares the first word of each line of the IBIS file with a set of predefined keywords (e.g., "R_pkg" for the package resistance). If it matches with one of the keywords, it saves the value of the element (which, in an IBIS file, is usually written next to the keyword).

As an illustration, in the IBIS text file sample (Algorithm 2), the program will count 8 I/O, among them three identical (A0, A1, A2). It will also notice the R_pkg, L_pkg and C_pkg keywords and save the values located next to these keywords (resp., 0.2 Ω, 7 nH, and 1,5 pF).

When the scanning phase is completed, the program generates the SystemC-AMS file. This file includes one top level module with the correct number of I/Os. Each I/O is instantiated as a separate module inside the top level component and can be directly associated with a SystemC digital simulation model (see Figure 1): on one hand, a signal on an output pin is produced with the help of a switch component, commanded by the digital model, on the other hand, a voltage on an input pin is converted by a threshold detector to generate a boolean value that is sent to the digital model. Obviously, each I/O module models the electrical circuits presented above, with the extracted values of the R, L, C components. Algorithm 3 presents a sample of an in-out pin module created from an IBIS file specification.

This tool was used to generate a SystemC-AMS model out of the IBIS description of the CAT24WC0 2 Kbit I^2C EEPROM chip from catalyst semiconductors [22]. This model serves as the analog interface of both slave memory devices featured in the I^2C platform.

3.3. Transmission line model

The bus lines feature a number of imperfections and can suffer from any kind of signal integrity perturbation. This section presents how the electrical model of the bus lines can be included in our modelling platform.

So far, we have considered the bus lines as perfect conductors immune to any kind of interference. Still, the nature of the wires as well as their geometry can modify the behaviour of the transmitted signals and, of course, so does the external environment. These imperfections are represented with R, L, C elements that we integrate in our platform. Our line model below (Figure 14) is based on [23].

Each bus line includes the wire resistance and also an inductance that represents the wire self-inductance. The value of this inductance is given by the following formula, where l is the wire length and d the wire diameter in centimetres:

$$L = 0.002l\left(\ln\left(\frac{4l}{d}\right) - 0.75\right)\ [\mu\text{H}]. \tag{1}$$

In the situation of two (or more) adjacent lines, which is likely to appear in the case of the I^2C bus, a mutual coupling phenomenon appears. It is represented by a parallel

```
struct IBIS_INOUT_PIN : sc_module
{
  sc_out<bool>   i; // Input signal sent to digital block
  sc_in<bool>    o; // Output signal sent by digital block
⋮
  sca_elec_port   pin; // Inout Pin
⋮
  sca_sdf_signal<double> in_sdf; // Pin Voltage
⋮
  sca_elec_ref    gnd;      // Electrical Ground
  sca_elec_node B;         // Node between R_pkg & L_pkg
  sca_elec_node A;         // Node between R_pkg & L_fix
  sca_elec_node V;         // Node between R_fix & switch
⋮
  // Declaration of IBIS RLC elements
  sca_r *R_pkg, *R_fixture; sca_c *C_pkg, *C_fixture;
  sca_l *L_pkg; sca_vconst *V_fixture;
  sca_rswitch *sw1;    sca_vd2sdf *conv_pin;
⋮
  // Threshold converts input voltage to bool value
  THRESHOLD_DETECTOR *thd_in;
⋮
  SC_HAS_PROCESS(IBIS_INOUT_PIN);
⋮
  // Constructor: RLC values retrieved from IBIS file
  IBIS_INOUT_PIN( sc_module_name insname,
        double R_pkg_value, double C_pkg_value,
        double L_pkg_value, double C_fixture_value,
        double R_fixture_value, double V_fixture_value )
     {
        // PACKAGE ELEMENTS INSTANCIATION
           C_pkg->value = C_pkg_value;
           C_pkg->p(pin);  C_pkg->n(gnd);
⋮
           L_pkg->value  = L_pkg_value;
           L_pkg->p(B);     L_pkg->n(pin);
⋮
           R_pkg->value  = R_pkg_value;
           R_pkg->p(A);     R_pkg->n(B);
⋮
        // PACKAGE ELEMENTS INSTANCIATION
           C_fixture->value = C_fixture_value;
           C_fixture->p(A); C_fixture->n(gnd);
⋮
           R_fixture->value = R_fixture_value;
           R_fixture->p(Vfix); R_fixture->n(B);
⋮
           V_fixture->value = V_fixture_value;
           V_fixture->p(Vfix); V_fixture->n(gnd);
⋮
        // Switch controlled by digital block
           sw1->p(V); sw1->n(gnd);
           sw1->ctrl(o); sw1->off_val = true;
⋮
        // Threshold: Pin Voltage is sent to digital
        // block as bool value
           thd_in->threshold = 1.1;
           thd_in->in (in_sdf); thd_in->out(i);
           };
     }
```

Algorithm 3: SystemC-AMS code sample generated from IBIS specification.

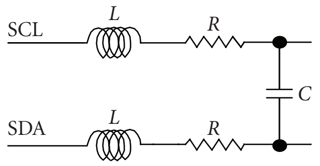

Figure 14: Transmission line model.

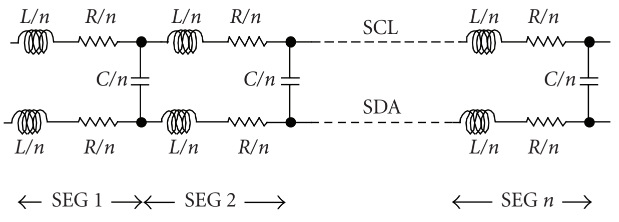

Figure 15: Transmission line model in n segments.

capacitance whose value depends on the diameter d of the wires and the distance D between them. The mutual capacitance is given by the following formula:

$$C = \frac{\pi 0.0885}{\cosh^{-1}(D/d)}\ [\text{pF/cm}]. \tag{2}$$

In a first step, we considered that the mutual inductance between the I^2C wires is negligible.

The length of the bus lines, for instance in an automotive context, can be quite important, up to several metres. Therefore, the global line model of Figure 14 can be divided into several segments, such as in Figure 15. With this representation, we can take into account a distance variation between two bus lines, affect only a part of the bus with a perturbation, or simply use segments with different lengths to respect the node topology in the system (Figure 16).

Our SystemC-AMS line model comes in the form of one top module that instanciates n segment submodules. The length of each segment can be parameterized in the submodule constructor (Algorithm 4).

4. PLATFORM SIMULATION

We present in this section the simulation of our field-bus-based platform, summarized in Figure 17. The SystemC and SystemC-AMS description allow us to cosimulate the MIPS or the 8051 embedded software with the hardware behaviour of the components, whether analog or digital. This section first gives some information about the code that is executed during simulation, then shows its impact on the digital blocks of the platform, especially the bus controllers. We focus next on the analog response on the bus, presenting the influence of the transmission line on the shape of the signal. Finally, we provide simulation performances for our mixed platform.

4.1. Embedded code

The aim of our simulation is to validate the functionality of the bus controller by testing all the protocol features.

```
struct LINE_I2C:sc_module
{
    sca_elec_port scl1, scl2;       // Segment ends for SCL
    sca_elec_port sda1, sda2;     // Segment ends for SDA
    sca_elec_node lr1, lr2;         // Node between L and R
⋮
    double L_value, R_value, C_value; // Elements value
⋮
    sca_r *R_scl, *R_sda;
    sca_l *L_scl, *L_sda;
    sca_c *C_cpg;
⋮
    SC_HAS_PROCESS(LINE_I2C);
⋮
    LINE_I2C(sc_module_name insname, double length)
        {    // Length in cm
            L_value = 7.8e − 9*length;
            R_value = 0.5*length;
            C_value = 0.1e − 12*length;
⋮
        // Elements Instanciation
⋮
            L_scl->p(scl1); L_scl->n(lr1);
            L_scl->value = L_value;
⋮
            R_scl->p(lr1); R_scl->n(scl2);
            R_scl->value = R_value;
⋮
            L_sda->p(sda1); L_sda->n(lr2);
            L_sda->value = L_value;
⋮
            R_sda->p(lr2); R_sda->n(sda2);
            R_sda->value = R_value;
⋮
            C_cpg->p(scl2); C_cpg->n(sda2);
            C_cpg->value = C_value;
        };
    };
```

Algorithm 4: Line segment SystemC-AMS code sample.

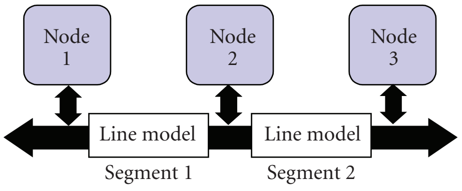

Figure 16: Line model and system topology.

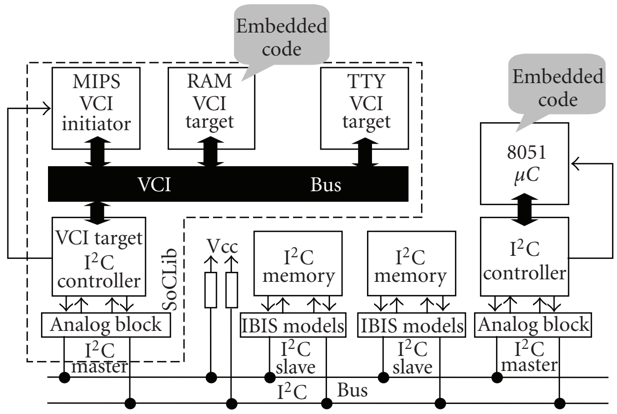

Figure 17: I^2C platform.

```
        //          Write in RAM(slave address = 0x50)
        //          the value 0x4E in cell 0xF1
⋮
main: MOV        R0, #A0 // Put slave address in R0 register
      MOV        A, #B4 // Put cell address in A register
      MOVX      @RO, A // Send R0 + A to I2C IP
      MOV        R0, #A0 // Put slave address in R0 register
      MOV        A, #4E // Put data byte in A register
      MOVX      @RO, A // Send R0 + A to I2C IP
```

Algorithm 5: 8051 assembler code sample—byte writing operation.

Therefore, the embedded code found on the two master devices mainly consists in bus access operations.

The following assembler code (Algorithm 5) is compiled and stored in the internal ROM of the 8051 microcontroller. When it runs, the program performs an I^2C write operation: the data byte 4Eh is written on the B4h cell address of the slave RAM device.

This write operation takes place in two stages: first the IP sends the cell address to the slave device (the A0 h value is made of the 50 h 7-bit slave address and the R/W bit set to 0), then it writes the data byte to this same slave device.

Similarly, Algorithm 6 gives a sample of the MIPS code, featuring the assembler functions used to request the controller IP a read or a write on the I^2C bus, as well as the interrupt subroutine which is called when a read data has been received from the I^2C bus and is available for the MIPS master.

4.2. *Digital simulation*

When running, the codes found in the SoC and the microcontroller send requests and therefore activate their respective I^2C controller. In particular, we can see in Figure 18 a sequence featuring a write operation requested by the 8051 to one of the RAM devices followed by an MIPS-commanded read request of the same RAM cell address.

The SystemC chronogram shows the 8051 positioning the 4 Eh byte on its data bus and the F1 h value on its address bus, resulting on a first transmission on the I^2C bus, then the SoC bus controller performing a second transmission to retrieve the data from the RAM component. The chronogram shows the digital behaviour of the SDA bus line (featured on the last row), as well as the SDA commands sent by each component, indicating which device has control of the bus at any given time. We can also notice that an IRQ

```
// Interrupt Subroutine
void SwitchOnIt(int it)
{
   int i;
⋮
// Identify the active interrupt of highest priority
   for (i = 0; i < 8; i++) if (it&(1 ≪ i)) break;
⋮
   switch (i)
   {
      case 0: break;        // SwIt 0
      case 1: break;        // SwIt 1
      case 2: read_interface(); break; // It 0
      case 3: break;        // It 1
      case 4: break;        // It 2
      case 5: break;        // It 3
      case 6: break;        // It 4
      case 7: break;        // It 5
      default: break;
   }
}
⋮
// Main Program
int main(void)
{
   write_byte(0x50, 0x57, 0x69);
   // Parameters = Slave Adr (stored in $4 register)
   //              Cell Adr (stored in $5 register)
   //              Data Byte (stored in $6 register)
⋮
   read_byte(0x51, 0xf1);
   // Parameters = Slave Adr (stored in $4 register)
   //              Cell Adr (stored in $5 register)
⋮
   while (1);
   return 0;
}
⋮
// Byte Writing ASM Code Sample
write_byte:
   la   $3,0xc0000000   // Load VCI target (I2C IP)
                        // address to $3 register
   sll  $4, $4, 9       // Slave adr in high part of register
   or   $4, $4, $5      // Cell adr in low part of register
   sh   $4, 0 ($3)      // (save half-word)
                        // send target + cell adr to IP
   sb   $6, 1 ($3)      // Send data byte to IP
   nop
   j    $31             // return from subroutine
.end write_byte
// Byte Reading ASM Code Sample
read_byte:
⋮
   la   $3,0xc0000000   // Load VCI target (I2C IP)
                        // address to $3 register
   sll  $4, $4, 9       // Slave adr in high part of register
   ori  $4, $4, 0x100   // Set R/W Bit to Read
   or   $4, $4, $5      // Cell adr in low part of register
   sh   $4, 0 ($3)      // save half-word ->
   nop                  // send target + cell adr to IP
   j    $31             // return from subroutine
```

Algorithm 6: MIPS code sample.

signal is generated when the 4Eh value is available in the MIPS I^{2}C controller.

4.3. Analog simulation

The chronogram in Figure 19 represents the analog behaviour of the I^{2}C bus lines during a write transmission issued by the MIPS-based SoC. It does not take into account the transmission line model. On the first row of the chronogram, the slave address (50 h) is first sent on the bus, then on the second row, after an acknowledge from this slave device, the master transmits the cell address (B4 h) it wants to write into. Finally, on the third row, after another acknowledge from the slave, the data byte (4 Eh) is sent to the slave, followed by a last acknowledge from the slave and a stop bit, signaling the end of transmission.

The chronogram also displays the shape of the analog signal. The rising and falling times that can be seen depend on the analog interface of the component that has control of the bus when a particular bit is transmitted. In the case of the chronogram in Figure 19, most of the bits are issued by the 8051 bus controller interface, except for the acknowledges which are issued by the IBIS-converted interface of the RAM device.

Our platform also allows us to test the multimaster arbitration of the I^{2}C bus as it is implemented in our model. In the chronogram shown in Figure 20, both masters send a start bit at the same moment. The SDA bus line follows the two commands as long as they are identical, and when they differ, the master which sends a 0 bit (in this case the MIPS) takes control of the bus and the 8051 stops transmitting on SCL and SDA. Also of interest is the wired-AND function performed on the SDA and SCL lines of the bus. This shows that our mixed-signal controller model successfully performs electrical multimaster arbitration.

Finally, Figure 21 presents the signal behaviour if we take into account the line imperfections with the model presented in Section 4. We can clearly see the effect of the wire inductance and the mutual capacitance.

4.4. Simulation performances

Table 1 presents simulation speeds for various configurations of the simulation platform presented in Section 4. The measurements were performed on a Linux workstation equipped with a Pentium M processor running at 1.73 GHz, a L2 cache of 2 Mb, and 1 Gb of SDRAM. Sampling period for the analog models is 100 nanoseconds.

Our aim is to measure the increase of simulation time between a purely digital simulation in SystemC of a platform, and a mixed SystemC/SystemC-AMS simulation of the same platform. We first measure this overhead in the case of a simplified platform made of a microcontroller node and a slave RAM, first without and then with the transmission line model. Results show that for the 8051 platform, the overhead due to the AMS parts of the model is relatively important, though simulation is still very fast.

In the second case that is the full 4-node platform including the transmission line model, the impact is less

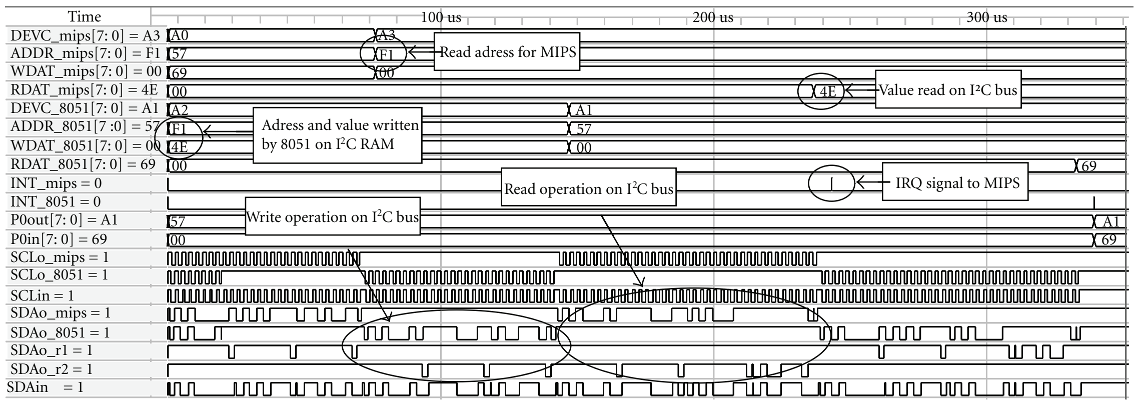

Figure 18: Simulation with 8051 and MIPS master.

Table 1: I^{2}C Platform simulation performance in cycles/s.

Application	SystemC platform	Sys.C + Sys.C-AMS equivalent plarform	Overhead
8051 node + RAM with IBIS	149 300	106 400	29%
8051 node + RAM with IBIS + line model	149 300	87 500	41%
4-node platform without line model	25 800	23 500	9%
Complete I^{2}C platform (Figure 19)	25 800	19 500	24%

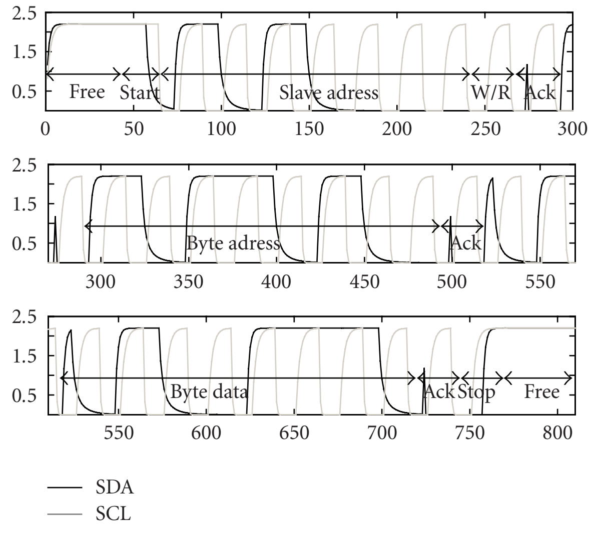

Figure 19: Analog simulation—byte writing operation.

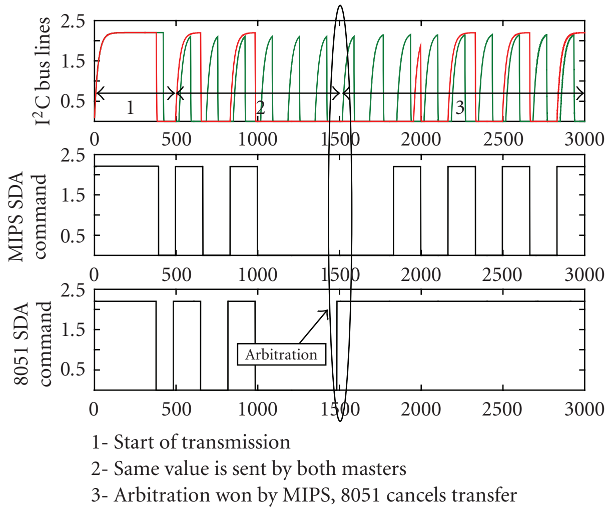

Figure 20: Multimaster mode arbitration.

prominent, mainly because the proportion of purely SystemC components is more important. In this latter configuration, analog simulation of the I^{2}C bus is performed only 9% slower than a digital-only simulation if we do not include the line model, and 24% if we add a line segment model between two nodes.

5. CONCLUSION

In this paper, we showed how to model field-bus communications in the context of embedded systems. We described a heterogeneous model that features three main parts. Firstly, an I/O bus controller interfaces the processor cores with the field bus. Secondly, an analog part represents the interface of the I/O controller with the bus lines. To accurately model the analog behaviour of the components, we introduced an IBIS-to-SystemC-AMS converter to derive the analog description of the field bus nodes analog interface from industrial circuits. Thirdly, we modelled the bus lines as an array of segments that take into account the electrical imperfections of the bus lines.

Simulations showed the successful operations of the I^{2}C field bus. The SystemC-AMS simulation times showed different overheads over digital SystemC: 40% for a 8051 to

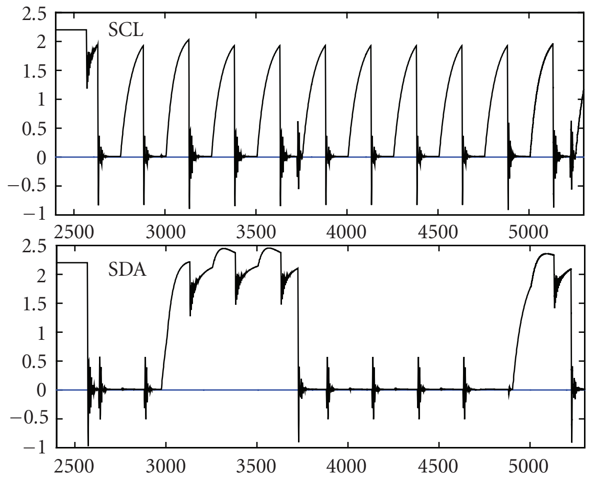

Figure 21: Bus lines simulation with transmission line model.

RAM I^{2}C communication, and around 25% for an MIPS-based SoC plus 8051 system. This overhead is acceptable for a mixed-signal simulation.

The benefits of our approach are that it includes analog and digital models in a single environment. It will therefore allow to simulate interactions between analog and digital functions that are usually discovered on the hardware prototypes.

As a next step, we intend to show that our approach is generic by applying it to other field busses such as CAN. Further, this SystemC-AMS simulation platform stresses the interest of this language as a unique tool from the specification to the verification of AMS embedded systems, gathering together software execution with analog and digital behaviour.

REFERENCES

[1] M. Chiodo, D. Engels, P. Giusto, et al., "A case study in computer-aided co-design of embedded controllers," *Design Automation for Embedded Systems*, vol. 1, no. 1-2, pp. 51–67, 1996.

[2] J. Liu, X. Liu, and E. A. Lee, "Modeling distributed hybrid systems in Ptolemy II," in *Proceedings of the American Control Conference (ACC '01)*, pp. 4984–4985, Arlington, Va, USA, June 2001.

[3] A. Vachoux, C. Grimm, R. Kakerow, and C. Meise, "Embedded mixed-signal systems: new challenges for modeling and simulation," in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS '06)*, pp. 991–994, Island of Kos, Greece, May 2006.

[4] J. Oudinot, J. Ravatin, and S. Scotti, "Full transceiver circuit simulation using VHDL-AMS," in *Proceedings of Forum on Specification and Design Languages (FDL '02)*, pp. 29–33, Marseille, France, September 2002.

[5] T. Grötker, S. Liao, G. Martin, and S. Swan, *System Design With SystemC*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002.

[6] A. Vachoux, C. Grimm, and K. Einwich, "SystemC-AMS requirements, design objectives and rationale," in *Proceedings of Design, Automation and Test in Europe Conference and Exhibition (DATE '03)*, pp. 388–393, Munich, Germany, March 2003.

[7] A. Vachoux, C. Grimm, and K. Einwich, "Extending SystemC to support mixed discrete-continuous system modeling and simulation.," in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS '05)*, vol. 5, pp. 5166–5169, Kobe, Japan, May 2005.

[8] A. Vachoux, C. Grimm, and K. Einwich, "Towards analog and mixed-signal SOC design with systemC-AMS," in *Proceedings of the 2nd International Workshop on Electronic Design, Test & Applications (DELTA '04)*, pp. 97–102, Perth, Australia, January 2004.

[9] A. Vachoux, C. Grimm, and K. Einwich, "Analog and mixed signal modelling with SystemC-AMS," in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS '03)*, vol. 3, pp. 914–917, Bangkok, Thailand, May 2003.

[10] Open SystemC Initiative, http://www.systemc.org/.

[11] OSCI Analog/Mixed-signal Working Group (AMSWG), http://www.systemc.org/apps/group_public/workgroup.php?wg_abbrev=amswg.

[12] O. Romain, T. Cuénin, and P. Garda, "Design and verification of mixed-signal I/O IPs: an 12C bus controller," in *Proceedings of IEEE International Symposium on Industrial Electronics (ISIE '04)*, vol. 1, pp. 77–81, IEEE Press, Ajaccio, France, May 2004.

[13] M. Alassir, J. Denoulet, O. Romain, and P. Garda, "A SystemC AMS model of an 12C bus controller," in *Proceedings of IEEE International Conference on Design and Test of Integrated Systems in Nanoscale Technology (DTIS '06)*, pp. 154–158, IEEE Press, Tunis, Tunisia, September 2006.

[14] M. Alassir, J. Denoulet, O. Romain, and P. Garda, "Modeling I^{2}C communication between SoCs with SystemC-AMS," in *Proceedings of IEEE International Symposium on Industrial Electronics (ISIE '07)*, pp. 1412–1417, IEEE Press, Vigo, Spain, June 2007.

[15] M. Alassir, J. Denoulet, G. Vasilescu, O. Romain, R. Arnaud, and P. Garda, "Modeling field bus communications for automotive applications," in *Proceedings of the Forum on specification and Design Languages (FDL '07)*, Barcelona, Spain, September 2007.

[16] Philips Semiconductors, The I^{2}C-Bus Protocol Specification, Document Order Number: 9398 393 40011, January 2000, http://www.nxp.com/acrobat/literature/9398/39340011_21.pdf.

[17] Philips Semiconductors, PCF 8584, I2C bus controller, http://www.nxp.com/acrobat/datasheets/PCF8584_4.pdf.

[18] SoCLIB, "A modelisation & simulation plat-form for system on chip," 2003, http://www.soclib.fr/.

[19] VSI Alliance, Virtual Component Interface Standard—version 2 (OCB 2 2.0), http://www.vsi.org/index.htm.

[20] Intel, 80C51 Single-Chip 8-bit Microcontroller datasheet, ftp://download.intel.com/MCS51/datashts/27050108.pdf.

[21] IBIS (I/O Buffer Information Specification), Version 4.2, June 2006, http://vhdl.org/pub/ibis/ver4.2/ver4_2.pdf.

[22] CAT24WC01/02/04/08/16, 1K/2K/4K/8K/16K-Bit Serial EEPROM datasheet, Catalyst Semiconductors, http://www.catsemi.com/datasheets/24WC.

[23] G. Vasilescu, *Electronic Noise and Interfering Signals: Principles and Applications*, chapter 12, Springer, New York, NY, USA, 2005.