Pre-Simulation Symbolic Analysis of Synchronization Issues between Discrete Event and Timed Data Flow Models of Computation

Liliana Andrade*, Torsten Maehne*, Alain Vachoux†, Cédric Ben Aoun*, François Pêcheux*, Marie-Minerve Louërat*
*Sorbonne Universités, UPMC Univ Paris 06, CNRS UMR 7606, LIP6, Paris, France
WWW: http://www-soc.lip6.fr/, E-mail: Liliana.Andrade@lip6.fr
†École Polytechnique Fédérale de Lausanne (EPFL), Laboratoire de Systèmes Microélectroniques (LSM), Switzerland
WWW: http://lsm.epfl.ch/, E-mail: alain.vachoux@epfl.ch

***Abstract*—The SystemC AMS extensions support heterogeneous modeling and make use of several Models of Computation (MoCs) that operate on different time scales in the Discrete Event (DE), Discrete Time (DT), and Continuous Time (CT) domains. The simulation of such heterogeneous models may raise synchronization problems that are hard to diagnose and to fix, especially when considering multi-rate data flow parts. In this paper, we show how to formally analyze the execution of Timed Data Flow (TDF) models including their interaction with the DE domain by converting the synchronization mechanics into a Coloured Petri Net (CPN) equivalent. The developed symbolic execution algorithm for the CPN allows to detect all DE-TDF synchronization issues before simulation and to propose appropriate sample delay settings for the TDF converter ports to make the system schedulable. The presented technique is validated with a case study including a vibration sensor model and its digital front end.**

***Index Terms*—synchronization; multi-domain simulation; SystemC Analog/Mixed-Signal (AMS) extensions; Discrete Event (DE); Discrete Time (DT); Timed Data Flow (TDF); Coloured Petri Net (CPN); Model of Computation (MoC).**

I. Introduction

One of the greatest challenges in today's microelectronics industry consists undoubtly in the flawless design of More-than-Moore embedded systems that soundly interact with their surrounding analog environment. Such complex heterogeneous systems that mix digital and analog electronics as well as mechanical, thermal, optical, RF domains and software can not anymore be conceived as the late juxtaposition of parts known to work individually. The ability to model and simulate these digital-centric multidiscipline systems as a whole with Models of Computation (MoCs) that soundly cooperate and synchronize becomes the keystone of successful design methodologies. In order to reduce architecture exploration issues and design errors, virtual prototyping at the highest possible level of abstraction has become inevitable, especially for systems featuring a tight coordination between the computational and physical elements.

On the one hand, SystemC [1], based on a mature Discrete Event (DE) simulation kernel, is a well known solution for rapid system prototyping at the Register Transfer, bit-cycle-accurate and Transaction levels. On the other hand, the available SystemC Analog/Mixed-Signal (AMS) extensions [2], [3] allow the modeling of analog behaviors that may be coupled with digital ones. The preferred MoC for SystemC AMS is Timed Data Flow (TDF), which introduces time-stamped samples based on the timeless Synchronous Data Flow (SDF) theory [4]. TDF can efficiently be used to capture oversampled real or complex signal processing behaviors with feedback loops or delays. The specific representation of time in the TDF MoC and the use of converter ports to interface with the DE MoC yield time synchronization relationships that must be dealt with very carefully to avoid causality issues.

Several approaches have been presented in order to integrate untimed SDF MoCs in the DE domain [5], [6]. These approaches, despite describing the methods by which the SDF MoCs are synchronized with the DE MoC, do not address the time-related causality issues that could arise during the MoC interactions. Ptolemy II [7] includes a timed extension of the SDF MoC [8], which matches the SystemC AMS TDF MoC semantics. A rule is proposed in [8] to ensure the internal time-related causality of timed SDF models by adding latency tokens. In this paper, we do not consider this rule, because the SystemC AMS LRM does not impose that TDF models have to be fully causal for simulation. However, we demystify the causality issues that may arise when TDF models are connected to the "outside world" described by the DE MoC.

II. Discrete Event-Timed Data Flow Synchronization Issues

The embedding of TDF clusters of interconnected TDF modules into DE models is supported by *TDF converter input and output ports*. They can be directly connected to DE signals that can then be read in, or written from, TDF modules. DE signals that are read are sampled, while those that are written generate DE events. To that end, TDF clusters are interpreted as DE processes that are executed at the first delta cycle of the DE simulation cycle to provide the context for the TDF modules' potential DE read/write operations via TDF converter ports. These DE processes activate the

This work is supported by the European project CATRENE CA701 H-INCEPTION.

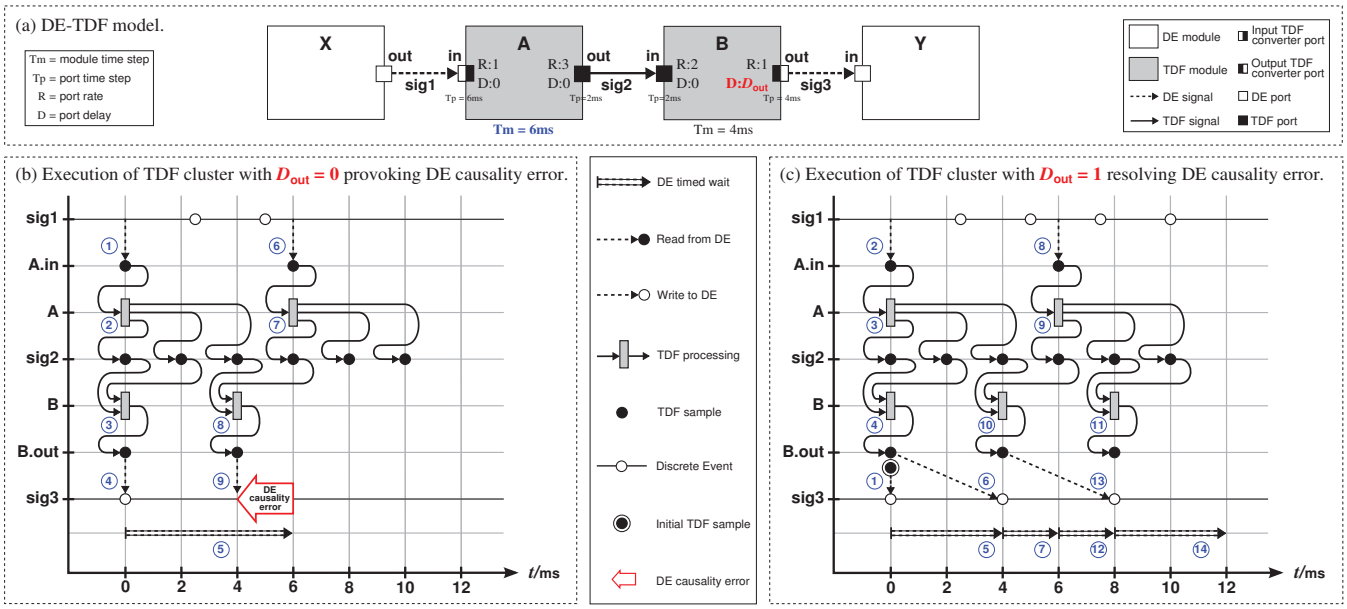

Fig. 1. Transient simulation of a TDF cluster with DE-TDF synchronization.

TDF modules' processing functions according to the finite static schedule, which can be obtained using the SDF theory [4]. Once the finite schedule has been executed, the TDF cluster has returned in its initial state and can be restarted after a *timed wait* DE statement, which aligns the DE/TDF time scales on the beginning of the next cluster period Tc.

It is important that the global advance of DE time during simulation of a mixed DE-TDF model is monotonically increasing. A valid static scheduling must guarantee that the DE events generated by a TDF cluster cannot happen earlier than the current DE time. However, common scheduler implementations for the SDF MoC and the TDF MoC are not formally addressing this issue. In consequence, synchronization problems can only be identified during simulation at which point not enough useful information is available to propose a way to fix the cause.

To illustrate the problem, let us consider, as a simple example, the mixed DE-TDF model of Fig. 1(a). The TDF cluster consists of two TDF modules **A** and **B**. The module **A** (**B**) is connected to the DE module **X** (**Y**) through the DE signal **sig1** (**sig3**). The modules **A** and **B** are connected through the TDF signal **sig2**. The signal arrows indicate the respective input and output ports. Ports **A.in** and **B.out** are TDF converter ports, while ports **A.out** and **B.in** are regular TDF ports. The required time step definition is here done for the time step of module **A**, $Tm_{,\mathsf{A}}$, which is set to 6 ms. The port rates are also defined as follows: 3 samples for **A.out**, 2 samples for **B.in**, and 1 sample for **A.in** and **B.out**. A valid TDF schedule (without considering DE-TDF interactions) is then **ABABB** with a module time step for **B** $Tm_{\mathsf{B}} = 4$ ms. The cluster period Tc is therefore equal to 12 ms and the cluster start time tc is equal to $n \cdot Tc$, where $n = 0, 1, \ldots$. If we now include the interactions with the DE part of the model, we can determine that the DE signal **sig1** must be sampled every $Tm_{\mathsf{A}} = 6$ ms, while the DE signal **sig3** will have an event every $Tm_{\mathsf{B}} = 4$ ms.

Fig. 1(b) shows the DE-TDF synchronization for the case where all TDF port delays D_{P} are set to zero using a graphical notation that we developed. The numbers in circles identify a *simulation trace*, i.e., the simulation steps of TDF module activations and DE-TDF synchronization, without actually considering any particular behavior. Two kinds of time lines are represented: the ones having white circles denote a DE time line, while the ones having black circles and grey boxes denote a TDF time line. The position of a grey box indicates when a TDF module is activated and the solid arcs indicate its consumption and production of samples, in that order. For the sake of simplicity, the trace details of the TDF ports and related TDF signals are omitted. The dashed arcs denote either the sampling of a DE signal or the generation of an event on a DE signal. The double dashed arrows indicate the advance of DE time by a timed wait. We can see in Fig. 1(b) that the simulation trace aborts at step ⑨ due to a causality error: the sample produced by the TDF module **B** on **B.out** is supposed to generate a DE event on signal **sig3** at 4 ms, while the DE time already advanced to 6 ms at step ⑥. Fig. 1(c) then shows that having a port delay of 1 sample on port **B.out** fixes the DE-TDF synchronization issue, since the DE events are postponed by one port time step of 4 ms.

The discussed example shows how easily causality problems arise in multi-rate TDF models due to DE-TDF synchronization that are not as obvious as feedback loops. The graphical notation we used in Figs. 1(b) and 1(c) is very helpful to understand the TDF and the DE-TDF synchronization semantics in the first place and later diagnose any causality problem. This approach however reaches its limits when the DE-TDF model has a more complex topology, important rate

differences, many delays, and feedback loops. The next sections propose our formalization of the TDF-DE synchronization that enables to detect such problems as early as the computation of the static schedule of the TDF cluster(s). Therefore, proper diagnostics and a proposal for resolving causality issues can be provided to the user before simulation actually starts.

III. CPN-based Representation of the DE and TDF Synchronization Interactions

Coloured Petri Nets (CPNs) are a DE modeling language combining the graphical notation of Petri Nets with functional expressions for constructing models of concurrent systems and analyzing their properties [9]. It contains *places* drawn as ellipses, for representing states; *transitions* drawn as rectangles, for representing actions; *directed arcs* for linking places and transitions; and *textual inscriptions* written in the CPN ML language, for defining data types, variables, priorities, and functions useful during the CPN simulation. When a CPN includes timing information, it is called a *timed CPN*. We have developed an equivalent representation of TDF clusters and their interactions with the DE domain using these *timed CPN*. It facilitates the comprehension of the TDF simulation, the detection of time inconsistencies, and the proposition of solutions for the synchronization issues presented in Section II.

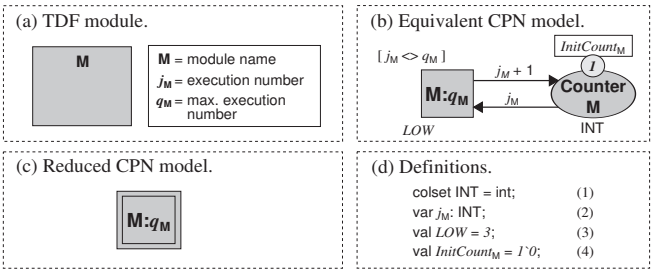

Fig. 2. Construction of an equivalent CPN for TDF modules.

The first step for the construction of an equivalent CPN is the representation of TDF modules. As is shown in Fig. 2(b), the TDF module execution action is represented by a transition with three defined textual inscriptions: a name **M:q_M** to identify the transition; a guard [j_M<>q_M], which represents a Boolean expression used to evaluate whether the transition is enabled; and a priority defined in (3) of Fig. 2(d), which restricts the transition occurrence. Moreover, the TDF module's current execution number j_M is stored in a place with three defined textual inscriptions: a name **Counter M**, which identifies the place; a color set defined in (1) of Fig. 2(d), which indicates the token data type that can be contained therein; and an initial marking defined in (4) of Fig. 2(d), which specifies the initial tokens of the place. Tokens respect a multiset notation consisting of a back-quote operator " ` ", which takes an integer as left argument specifying the number of appearances of the element provided as right argument. When the initial marking is composed of several tokens, they are separated using the operator "**++**". The circle, which joins the place to the initial marking, indicates the number of tokens contained in the place. To complete the TDF module representation, directed arcs link the defined transition and the defined place. Textual inscriptions next to arcs indicate that the j_M value is incremented when the **M:q_M** transition is fired. In order to simplify the graphical notation, Fig. 2(c) proposes the reduced CPN used for representing the TDF modules.

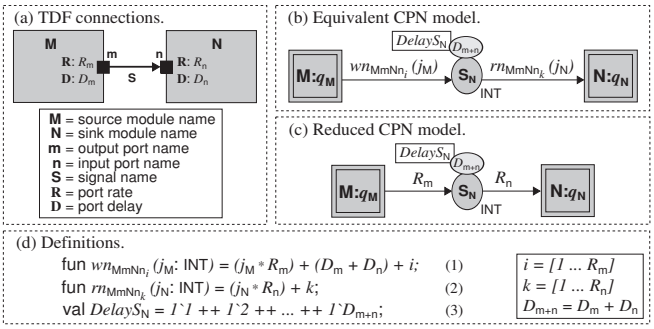

Fig. 3. Construction of an equivalent CPN for TDF connections.

The second step for the construction of an equivalent CPN is the representation of the TDF connections. As is shown in Fig. 3(b), the TDF signal **S** is represented using a place with three inscriptions: the name **S_N** (signal **S** connected to an input port of module **N**); the color set defined in (1) of Fig. 2(d); and the initial marking defined in (3) of Fig. 3(d), indicating the multiset of delay tokens associated to the TDF signal. The delay tokens number of the place **D_{m+n}** is the addition of the delay attributes associated to the interconnected ports. The transition representing the producer module **M:q_M** is linked to the **S_N** place using a directed arc with the function inscription defined in (1) of Fig. 3(d); it calculates the token identifier (integer indicating the token position in the **S_N** place) to be produced when **M:q_M** is fired. Similarly, the **S_N** place is linked to the transition representing the consumer module **N:q_N**, using a directed arc with the function inscription defined in (2) of Fig. 3(d); it calculates the token identifier to be consumed when **N:q_N** is fired. In multi-rate models, the number of arcs linked among transitions and places, are determined by the involved port rates (in the equations, it is represented using the i and k index). To simplify the graphical notation, Fig. 3(c) proposes the reduced CPN used for representing the TDF connections.

The third step for the construction of an equivalent CPN model is the representation of the input and output converter ports. This representation adds the timing information required to synchronize the read and write operations among the DE and DT domains. Fig. 4(b) shows the equivalent CPN for an input converter port at time t: the **S read ops. list** place stores the reading synchronization events defined in (8) of Fig. 4(d) for one cluster period; the **Enable S read op.** transition enables a read operation; the **S read op. enabled** place stores the read operation that is enabled; the **Counter S** place stores the number of read operations executed; the **Read S** transition represents the DE read operation; and the **M.m** place stores the available tokens, which can be consumed by the **M:q_M** transition. Similarly, Fig. 5(b) shows the equivalent CPN for an output

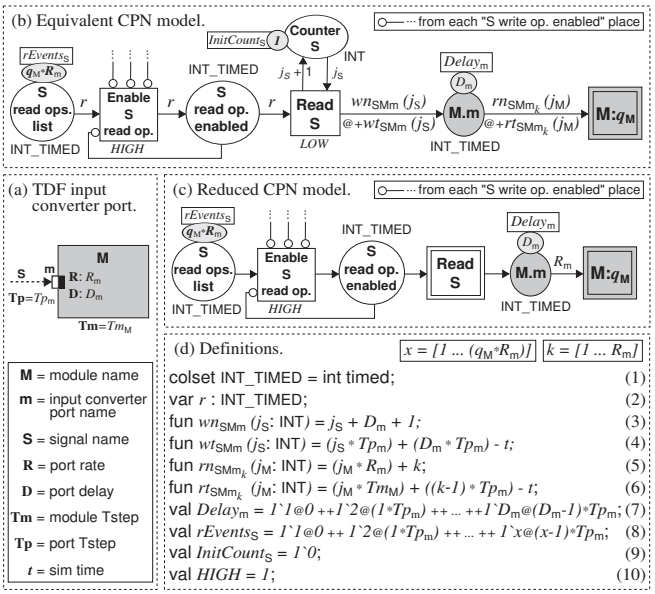

Fig. 4. Construction of equivalent CPN for TDF input converter ports.

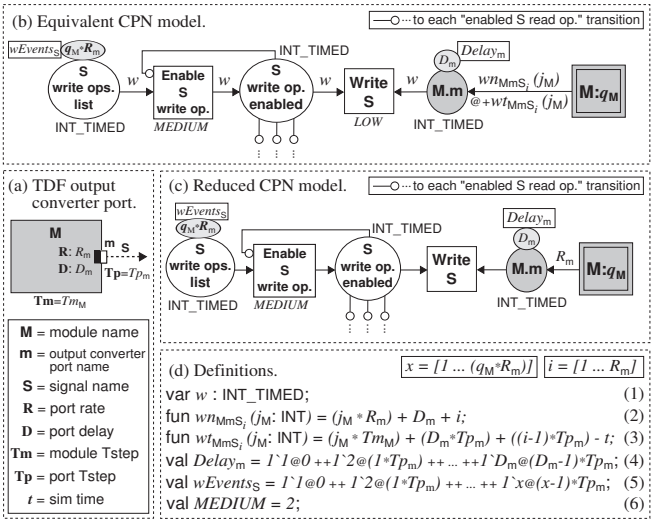

Fig. 5. Construction of equivalent CPN for TDF output converter ports.

converter port at time t: the **S write ops. list** place stores the writing synchronization events defined in (5) of Fig. 5(d) for one cluster period; the **Enable S write op.** transition enables a write operation; the **S write op. enabled** place stores the write operation that is enabled; the **M.m** place stores the available tokens, which can be written in the DE domain; and the **Write S** transition represents the DE write operation.

A new color set, defined in (1) of Fig. 4(d) is associated to some of the defined places: it indicates that the stored tokens contain not only an identifier, but also a time stamp indicating when they can be consumed; this time stamp is added to the token using the operator "@". Three transition priority levels (*HIGH*, *MEDIUM* and *LOW*) are defined in (10) of Fig. 4(d), (6) of Fig. 5(d) and (3) of Fig. 2(d). The *HIGH* priority transitions are reserved to enable the DE read operations, the *MEDIUM* ones to enable the DE write operations, and the *LOW* ones to enable the TDF executions. The initial marking of the **M.m** places is present when the involved converter ports have delay attributes associated. The arc functions defined in (3–6) of Fig. 4(d) and (2–3) of Fig. 5(d) calculate the identifier and the time stamp of the tokens that should be consumed and produced for a current time t. The formulation of all functions used as arc inscriptions in the model has been derived from the TDF execution rules defined in the SystemC AMS standard [2].

To complete the converter port representation, inhibitor arcs link some places and transitions. The execution of each **Enable S read op.** transition is inhibited by the **S read op. enabled** place directly linked to it, and by each **S write op. enabled** place present in the model, as is shown in Fig. 4(b). The execution of each **Enable S write op.** transition is inhibited by the **S write op. enabled** place directly linked to it, as is shown in Fig. 5(b). To simplify the graphical notation, Fig. 4(c) and Fig. 5(c) propose the reduced CPN used for representing the input and output converter ports.

By combining the transformation rules for the individual TDF model elements, any TDF cluster can be represented as a CPN. As an example, Fig. 6 shows the equivalent CPN model for the DE-TDF model in Fig. 1(a). Using CPN Tools [10], we validated the semantical correctness of the CPN representation of the DE-TDF synchronization interactions and that the execution of the resulting CPN model yields a legal simulation trace respecting all semantics of the TDF simulation as shown in Figs. 1(b) and 1(c).

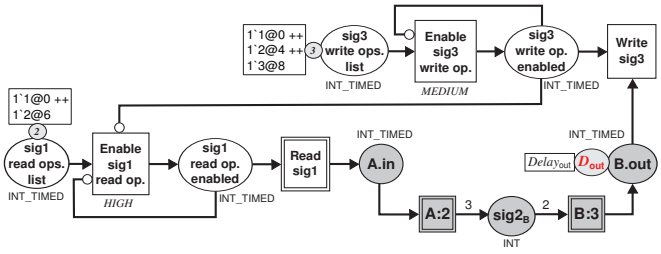

Fig. 6. Equivalent CPN model of the DE-TDF model in Fig. 1(a).

Based on the CPN execution rules, we developed an algorithm, which extracts from the simulation trace the static schedule and in case of causality errors provides the necessary delay changes to make the TDF module computable. This algorithm and its application to the example in Fig. 6 are presented in the next section.

IV. Pre-simulation Symbolic Analysis

In order to detect the causality problems and construct the schedule required for planning the TDF cluster simulation, the algorithm shown in Listing 1 has been developed. It takes as arguments: the net structure on which the analysis is performed, the schedule structure where the execution order is stored, and the T_net cluster period. Before starting, the net is assumed computable, the final state is supposed not reached and the time is initialized (Listing 1, lines 2–4). The first called function (Listing 1, line 6) is responsible for firing all enabled transitions

```
1  bool analyze_computability (net, schedule, T_net) {
2     bool computable_net = true;
3     bool final_state = false;
4     double t = 0;
5
6     fire_enabled_transitions (net, schedule, computable_net, t, T_net);
7     final_state = verify_final_net_state (net, computable_net);
8     while (!final_state) {
9        computable_net = false;
10       unlock_net_and_determine_delay_changes (net, t);
11       fire_enabled_transitions (net, schedule, computable_net, t, T_net);
12       final_state = verify_final_net_state (net, computable_net);
13    }
14    if (!computable_net) print_required_delay_changes (net);
15    return computable_net;
16 };
```

Listing 1. Algorithm to analyze the CPN computability.

```
1  void fire_enabled_transitions (net, schedule, computable_net, t, T_net) {
2     bool disabled_net = false;
3     double tmin = 0;
4
5     while (!disabled_net) {
6        fire_enabled_high_priority_transitions (net);
7        fire_enabled_medium_priority_transitions (net);
8        if (!computable_net)
9           fire_enabled_low_priority_transitions (net);
10       else
11          fire_and_push_enabled_low_priority_transitions (net, schedule);
12       disabled_net = true;
13       tmin = search_minimum_tstamp (net);
14       if ((tmin > t) && (tmin != T_net)) {
15          t = tmin;
16          disabled_net = false;
17       }
18    }
19 };
```

Listing 2. Algorithm to fire the CPN enabled transitions.

and adding to the schedule, the order and the DE times at which the low priority transitions are fired. Once the net is locked (transitions are disabled), it is necessary to check if the net's final state is reached (Listing 1, line 7); it occurs when the initial synchronization events have been consumed from the **S read ops. list** and **S write ops. list** places, when the **M:q_M** transitions have been fired q_M times, and when the initial number of samples is available in the **M:q_M** and **S_N** places. If the final state is directly reached, the function returns *true* indicating that the schedule is complete and that no causality problems were found. Otherwise, the net is marked as not computable, and the delay changes required to solve the causality problems are determined (Listing 1, lines 9–10).

The causality problems in the TDF clusters occur when a DE write operation is required; and the sample to be written in the DE domain, has not yet been generated by a TDF output converter port. In the CPN, the detection of this problem corresponds to identifying the locked **Write S** transition, because its **S write op. enabled** connected place has one token indicating that the write operation should be realized at time t; and its **M.m** connected place has no token to be consumed at time t. Accordingly, the objective of the function defined in Listing 1, line 10, is to detect the locked **Write S** transition, to delete the token contained in its **S write op. enabled** connected place, and to increase the delay attribute associated to the **M.m** connected place. After these modifications, the net is unlocked and the execution continues until a new lock case is found (Listing 1, line 11). The net analysis has to be performed while the net has not yet reached its final state. If the final state is reached and the net is marked as not computable, the delay changes required to solve the causality problems are presented (Listing 1, line 14).

The algorithm responsible of the transition firing and the schedule creation is shown in Listing 2. Following the CPN execution rules, the transitions are fired according to the defined priority levels. If the net is computable, the low priority transitions are added to the schedule, as they represent the TDF module executions and their DE interactions (Listing 2, line 11). Once the net is not computable, the construction of the schedule halts: the low priority transitions are fired, but they are not added to the schedule (Listing 2, line 9). As long as time t is different to the T_net period and no more transitions are enabled, the time t is increased to the the minimum time stamp value contained in the net places (Listing 2, lines 13–17). As the time increase enables new transitions, the algorithm will be re-executed (Listing 2, lines 5–18).

Using an implementation of these algorithms in C++, the model in Fig. 6 is analyzed for two scenarios. When $D_{out} = 0$, the causality problems are detected, the schedule is not valid, and the delay changes are proposed. When $D_{out} = 1$, the CPN directly reaches its final state and the schedule is constructed, indicating the execution order and the DE times at which the TDF modules and their interactions with the DE domain should be performed. The results are summarized in Table I.

TABLE I
ANALYSIS RESULTS OF THE CPN MODEL SHOWN IN FIG. 6

Initial Delays	Schedule	Computability	Proposed Changes
$D_{out} = 0$	0 ms – **read sig1, A, B, write sig3**	*false*	$D_{out} = 1$
$D_{out} = 1$	0 ms – **write sig3, read sig1, A, B** 4 ms – **write sig3** 6 ms – **read sig1, A, B, B** 8 ms – **write sig3**	*true*	none

V. CASE STUDY

In order to validate the proposed technique, a vibration sensor and its digital front end has been modeled using the SystemC AMS TDF MoC. As is shown in Fig. 7(a), it has a TDF cluster composed of a source **SRC**, which generates the vibration signal; a **SENSOR** element, which takes the acceleration caused by the vibration and generates a speed-proportional voltage signal; a Programmable Gain Amplifier **PGA**, which amplifies the input voltage using the **kin** gain factor; an Analog to Digital Converter **ADC**, which digitizes the amplified voltage; a TDF to DE converter **TDF2DE**, which forwards the digitized value to the DE domain; and an amplitude estimator **AAVG**, which calculates the absolute average amplitude of the received samples. The **kin** gain factor, read by the PGA, is calculated by a gain controller **CTRL**

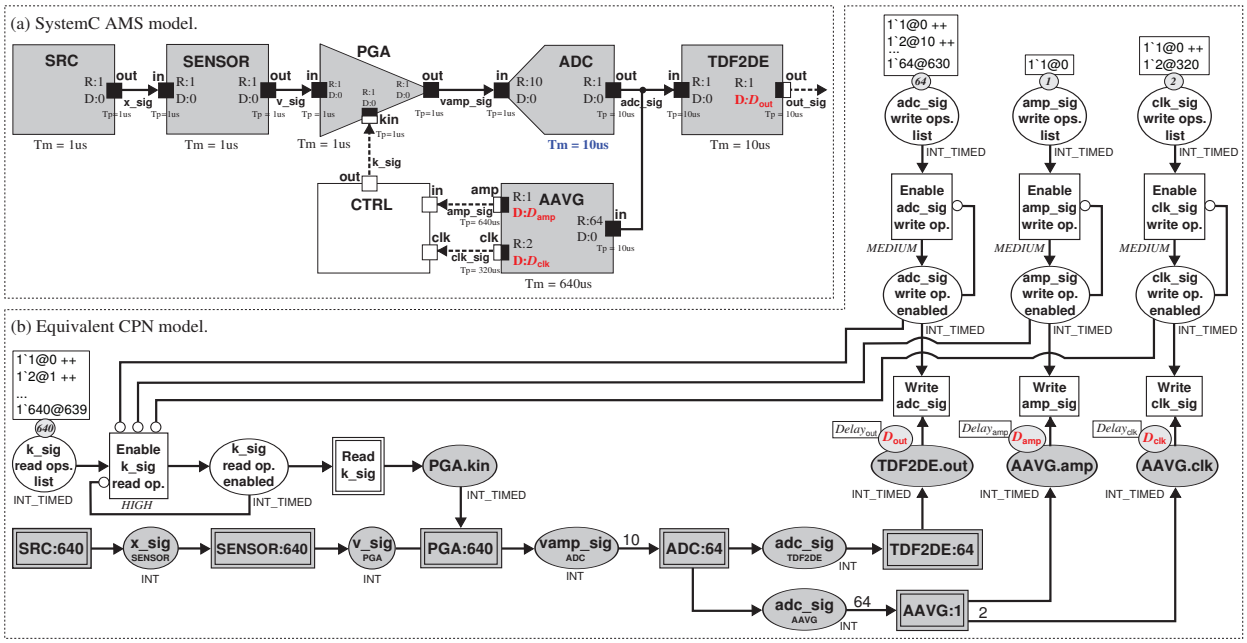

Fig. 7. Vibration sensor model.

modeled using the SystemC DE MoC. The function of this controller is to vary the gain according to the estimated amplitude received at its input.

The particularity of this model is that its multi-rate TDF cluster becomes part of a closed loop including a path through the DE domain. As the TDF cluster itself contains no loops, it could be assumed that port delay assignments are not necessary to calculate a valid schedule [3], but this is only valid for single-rate TDF models. Following the approach from Sections III and IV, the equivalent vibration sensor model in Fig. 7(b) is constructed and analyzed for the delay attributes set to $D_{\text{out}} = 0$, $D_{\text{amp}} = 0$, and $D_{\text{clk}} = 0$. As causality problems are detected during the application of the algorithm shown in Listing 1, the model is not considered computable, the schedule cannot be defined, and delay attribute changes are proposed. These determined delay attributes changes, $D_{\text{out}} = 1$, $D_{\text{amp}} = 1$, and $D_{\text{clk}} = 2$, are required to solve the causality problems in the model. Using this information, the user can modify the model, and restart the elaboration for finding the schedule including the order at which the TDF modules and its DE interactions should be executed before the actual simulation starts. If, in comparison to our approach, the unmodified TDF model is run in the Fraunhofer SystemC-AMS proof-of-concept simulator, the errors will be detected one by one during simulation. Thus, the user needs to perform three complete simulations to determine all required delay attribute changes.

VI. Conclusions

In this paper, we demonstrated that the causality problems arising in multi-rate TDF clusters interacting with the DE domain can be detected and resolved before simulation. We also showed that our approach of analyzing an equivalent CPN constructed from a TDF cluster for these problems yields a valid schedule for causal TDF clusters. In addition to the order of the TDF module activations, this schedule also includes all necessary timed wait and read/write DE operations. This is not the case when a pure SDF schedule algorithm is applied [4]. The vibration sensor case study not only validated our approach, but it also showed that multi-rate clusters, which include DE loops, require port delay assignments to calculate a valid schedule. Future work will target the conception of a simulator prototype that integrates the proposed analysis techniques while always respecting the SystemC AMS standard.

References

[1] IEEE Computer Society, *1666-2011 IEEE Standard SystemC Language Reference Manual*, IEEE, Jan. 9, 2012, 614 pp.

[2] M. Barnasconi *et al.*, Eds., *Standard SystemC AMS extensions 2.0 Language Reference Manual*, Accellera Systems Initiative, Mar. 19, 2013, 212 pp.

[3] M. Barnasconi *et al.*, *SystemC AMS extensions User's Guide*, Open SystemC Initiative (OSCI), Mar. 8, 2010, 166 pp.

[4] E. Lee and D. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing", *IEEE Trans. Comput.*, vol. C-36, no. 1, pp. 24–35, 1987. DOI: 10.1109/TC.1987.5009446.

[5] H. D. Patel and S. K. Shukla, "Towards a heterogeneous simulation kernel for system-level models: a SystemC kernel for synchronous data flow models", *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 23, no. 8, pp. 1261–1271, 2005. DOI: 10.1109/TCAD.2005.850819.

[6] F. Herrera and E. Villar, "A Framework for Heterogeneous Specification and Design of Electronic Embedded Systems in SystemC", *ACM Trans. Des. Autom. Electron. Syst.*, vol. 12, no. 3, 22:1–22:31, 2008. DOI: 10.1145/1255456.1255459.

[7] C. Brooks *et al.*, "Heterogeneous Concurrent Modeling and Design in Java (Volume 1: Introduction to Ptolemy II)", Dept. EECS, Univ. California, Berkeley, Tech. Rep. UCB/EECS-2008-28, 2008.

[8] C. Fong, "Discrete-Time Dataflow Models for Visual Stimulation in Ptolemy II", Dept. EECS, Univ. California, Berkeley, Tech. Rep. UCB/ERL M01/9, 2000.

[9] K. Jensen and L. M. Kristensen, *Coloured Petri Nets. Modelling and Validation of Concurrent Systems*. Springer, 2009.

[10] K. Jensen *et al.*, *CPN Tools 4.0.0*, AIS Group, TU Eindhoven, NL, 2013. [Online]. Available: http://cpntools.org/.