System-Level Modeling and Simulation of Networked PROFINET IO Controllers

Liyuan Zhang, Martin Streubühr, Michael Glaß, Jürgen Teich
Hardware/Software Co-Design
University of Erlangen-Nuremberg
{liyuan.zhang,streubuehr,glass,teich}@cs.fau.de

Andreas v. Schwerin, Kai Liu
Industry Sector
Siemens AG
{andreas.schwerin,kai-liu}@siemens.com

Abstract

In the industrial automation domain, an embedded system must not only fulfill its functionality, but also has to satisfy additional requirements, particularly with respect to real-time aspects. Given their often distributed fashion, a key aspect of industrial automation systems is the communication infrastructure. PROFINET IO modifies traditional Ethernet to fulfill the challenging requirements with respect to real-time guarantees and high timing precision. In this paper, we present a system-level modeling and simulation approach of modular networked PROFINET IO controllers. This includes the modeling of PROFINET IO core components such as controllers and switches, testing components such as traffic generators and monitors, as well as the composition of these components for the purpose of applying functional and timing simulation of a complete network.

1. Introduction

Driven by the rapid development of microelectronics technology, modern embedded systems continuously increase in terms of functionality and design complexity. The traditional Register Transfer Level (RTL) design methodologies can hardly handle the growing design difficulty and time-to-market pressure. In contrast to RTL design, Electronic System Level (ESL) [BMP07] design methodologies cope with these challenges by introducing higher abstraction levels. For example, a complete embedded system can be described by an executable specification at the system-level. Low-level descriptions such as RTL can be automatically generated using high-level synthesis, see [GHP*09]. As a major benefit, system-level abstraction allows evaluation of design decisions to be taken much earlier in the design flow. This results in reduced development cost and shortened design cycles.

In the industrial automation domain, an embedded system must not only fulfill its functionality, but also has to satisfy additional requirements like real-time guarantees, determinism, reliability, etc. Industrial automation systems are typically implemented in a distributed fashion. Thus, the
mentioned requirements particularly hold for the communication infrastructure. Under the term *Industrial Ethernet*, several extensions for traditional Ethernet have been proposed that aim at achieving the requirements of the industrial domain, see also [JSW07]. Currently, there are several Industrial Ethernet implementations on the market. One of them is PROFINET IO, an open industrial Ethernet standard, see for example [PM08]. By deploying PROFINET IO in the automation field, standard Ethernet technology is extended with capabilities for *real-time* (RT) and *isochronous real-time* (IRT) communication.

In [KSH11], the modeling of individual PROFINET IO controllers is shown. Here, we present a system-level modeling and both functional and timing simulation approach for *modular networked* PROFINET IO controllers: The novel modular model enables a seamless composition of controllers with configurable port numbers. Moreover, focus is put on combining the individual controllers to arbitrary and complex communication topologies. The topologies include the modeling of PROFINET IO core components like controllers and switches as well as testing components like traffic generators and monitors. The ESL design methodology employed in this paper is based on SystemC [GLMS02]. It uses actor-oriented design and *Models of Computation* (MoC) for describing the functionality of the system. Additionally to traditional data-driven Ethernet, PROFINET IO is time-driven with real-time constraints imposed at system level. Since we focus on the modeling of *networked* PROFINET controllers, key aspects are properly configured and synchronized controllers as well as global schedules for isochronous real-time (IRT) messages.

The remainder of the paper is outlined as follows: Related work is discussed in Section 2. In Section 3, fundamentals for actor-oriented modeling in ESL design approaches are introduced. An overview for PROFINET IO is given before the system-level model of a modular PROFINET IO controller is introduced, in Section 4. In Section 5, we first discuss the benefits of our proposed modular design methodology before all the essential features such as clock synchronization, topology detection, etc. modeled with this methodology are introduced. Section 6 concludes the paper.

2. Related Work

In [FHT06], a library for modeling and simulating actor-oriented models termed *SysteMoC* is presented. SysteMoC is based on SystemC, the de facto standard for system-level modeling, and adds a Model of Computation to develop an analyzable executable specification language. Other approaches similar to SysteMoC are for example the California Actor Language (CAL) [EJ03] and the Extended Codesign Finite State Machines (ECFSMs) [SVSL00].

In [KSH11], a single PROFINET IO controller is modeled at system-level using SysteMoC. The model is restricted as follows: (a) Only the functionalities on the data link layer are modeled. (b) The system is assumed to be synchronized. (c) The yellow transmission interval, a technique to maximize bandwidth usage, is neglected. However, these restrictions are not sound in the context of a complete network simulation since the model enables to analyze a steady-state behavior only. To overcome these drawbacks, the work at hand extends the model of [KSH11] by (a) a modular controller design, (b) PTCP-based synchronization, (c) ARP-based topology detection, and (d) a high-level model of the yellow interval.
3. Actor-oriented Modeling Fundamentals

In actor-oriented models, actors, which encapsulate the system functionalities, are potentially executed concurrently and communicate over dedicated abstract channels. Thereby, actors produce and consume data (so-called tokens), which are transmitted by those channels. Actor-oriented models may be represented as bipartite graphs, consisting of channels \( c \in C \) and actors \( a \in A \).

**Definition 1 (Network Graph)** A network graph is a directed bipartite graph \( G_n = (A, C, E) \), containing a set of actors \( A \), a set of channels \( C \), and directed edges \( E \subseteq (C.O \times A.I) \cup (A.O \times C.I) \) between actor output ports \( A.O \) and channel input ports \( C.I \), as well as channel output ports \( C.O \) and actor input ports \( A.I \).

**Definition 2 (Channel)** A channel is a tuple \( c = (I, O, n, d) \) containing channel ports partitioned into a set of channel input ports \( I \) and a set of channel output ports \( O \), its buffer size \( n \in N_\infty = \{1, 2, 3, ..., \infty\} \), and also a possibly empty sequence \( d \in D^* \) of initial tokens, where \( D^* \) denotes the set of all possible finite sequences of tokens.

We use SysteMoC, a SystemC based library for modeling and simulating actor-oriented models. The basic SysteMoC model uses a FIFO channel to present unidirectional point-to-point connection between an actor output port and an actor input port. Actors are only permitted to communicate with each other via channels, to which the actors are connected by ports. In a SysteMoC actor, the communication behavior is separated from its functionality, which is a collection of functions that can access data on channels via ports.

**Definition 3 (Actor)** An actor is a tuple \( a = (I, O, F, R) \) containing actor ports partitioned into a set of actor input ports \( I \) and a set of actor output ports \( O \), the set of functions \( F \) and the Finite State Machine (FSM) \( R \).

The functions encapsulated in an actor are partitioned into actions and guards and are driven by the finite state machine (FSM) \( R \) that also represents the communication behavior of the actor (i.e., the number of tokens consumed and produced in each actor activation). An action produces...
outputs, which are used by the firing FSM to generate tokens for the FIFO channels connected to the actor output ports. A guard just returns a Boolean value and does not have the ability to change the functionality state.

**Definition 4 (Actor FSM)** The actor FSM is a tuple $R = (Q, q_0, T)$ containing a finite set of states $Q$, an initial state $q_0 \in Q$, and a finite set of transitions $T$.

**Definition 5 (Transition)** In an FSM $R = (Q, q_0, T)$, a transition is a tuple $t = (q, k, f, q') \in T$ containing the current state $q \in Q$, an activation pattern $k$, the associated action $f \in a.F$, and the next actor state $q' \in Q$. The activation pattern $k$ is a Boolean function that decides if transition $t$ can be taken or not.

A graphical representation of a SysteMoC actor is given in Figure 1. The actor *Store-and-forward* models processing of a data frame using a store and forward paradigm. It contains one input port $in$ and one output port $out$. Thereby, tokens read from $in$ and written to $out$ represent entire data packets. The FSM of the actor uses two transitions representing the two phases: (1) storing and processing the frame; (2) forwarding the packet. The activation pattern $in(1)$ checks if there are enough tokens on the input queue. The activation pattern $waitFor(forwardEvent)$ is used to model execution delay by waiting for a delay that depends on the frame size. Consequently, the actor is either in a state *ready* to accept incoming frames or in a state waiting for the processing delay of the currently processed frame.

### 4. System-Level PROFINET IO Model

The proposed system consists of multiple IO controller models, which are developed on the system-level in a modular fashion. In this section, the fundamental aspects of PROFINET IO are briefly introduced. Afterwards, a single system-level IO controller model is presented from a modular design perspective.
4.1. PROFINET IO overview

PROFINET IO² modifies standard Ethernet to implement Real-Time (RT) and Isochronous Real-Time (IRT) communication. In IRT, cycle times are below 1 ms with a jitter below 1 µs [IEC10]. These values are required for clock synchronized applications in the automation domain like controlling the position and acceleration of a robotic arm (motion control). PROFINET IO separates the data exchange process into communication cycles, see Figure 2, with each communication cycle being further divided into intervals for different kinds of transmissions: The red interval is used for IRT communication within a subnet with maximum precision in a determined order that is specified offline. As a consequence, IRT communication requires a fixed network topology that must be determined during project planning. In the red interval, no communication other than the scheduled one is allowed. Within the orange interval, all Real-Time (RT) frames are transmitted without an a priori schedule. During the green interval, all Non-Real-Time (NRT) frames are transmitted according to their priorities. The yellow interval is used to guarantee that no frame will block the transmission of IRT frames in the next red interval. Only the frames that are legitimate for transmission, i.e., the transmission of the frame ends within the current yellow interval are sent.

4.2. Modular PROFINET IO Controller

The actor-oriented high-level model of a dual-port PROFINET controller is depicted in Figure 3. An inbound frame is received by an actor RX. Then, an actor TimeStamp generates a timestamp for the frame and forwards it to the actor Demux. Demux transmits the frame either to the actor RED Relay or MAC Relay. Demux hereby transmits the frames according to the rules concerning the message type, i.e., NRT, RT, and IRT, and the current communication interval, see Figure 2. The switching of the communication interval is controlled by the actor Scheduler, according to the predefined communication cycle. After the frame is processed by the RED Relay or MAC Relay, the following actor Mux transfers the frame to the actor Switch Fabric, where the switching operation is performed. Then, the frame is transmitted to the actor Queue Handler. The Queue Handler processes the inbound frame in two ways: (a) It can (a) send the frame immediately to the transmitter, if the frame is in its corresponding interval and is allowed by the protocols, e.g., sending a frame in the yellow interval. (b) It can buffer the frame in the sending queues according to the frame’s type and priority, cf. IEEE 802.1Q. The actor TX models the sending of the data frame on the actual physical medium.

In [KSH +11], one centralized RED Relay is used to regulate the IRT traffic per IO controller. Moreover, in order to ease the modeling effort, a multiplex FIFO is used to route the frames from all the ingress ports to the centralized RED Relay or MAC Relay. This centralized approach sequentializes parallel processing for IRT messages and, hence, induces significant inaccuracies in the timing simulation. In this work, we focus on a modular design by partitioning the functionalities of the PROFINET IO controller into three modules, see Figure 3: (1) The Switch Processing Unit (SPU) includes (a) the centralized MAC Relay to maintain the MAC address table for the controller, (b) the Scheduler to control the changing of transmission intervals, and (c) the Switch Fabric to conduct the frames to their destination ports. (2) The Port Processing Unit (PPU) encapsulates

---

¹The waitFor mechanism can be implemented using SystemC’s dynamic sensitivity or using the event channel mechanism as suggested in [KSH+11].
²In this paper, the model follows the PROFINET IO standard version 2.2. [PRO07a]
all essential functions concerning one PROFINET communication port, i.e., two TimeStamp units, the RED Relay, and the Queue Handler. (3) The PHY includes a transmitter (TX) and a receiver (RX), corresponding to the interface between data link layer and physical layer.

5. Networked PROFINET IO Controllers

Section 4.2 introduced the proposed modular modeling approach that subdivides an IO controller’s functions into three modules. This modular design has three main benefits: (a) it allows creating complex topologies by simply adding more port modules (PPUs) and PHYs; (b) the modules are re-usable; and (c) finding modeling errors is easier. Figure 4 depicts an example where a four-port IO controller is employed to build a star topology with three dual-port controllers. In the following subsections, essential functionalities for networked IO controllers are introduced. We focus on how these functions are realized with the proposed modeling approach.

5.1. Topology Detection

As described in Section 4.1, NRT frames, i.e., standard Ethernet messages, are processed in the MAC Relay. They are forwarded to the corresponding ports according to the protocol defined in IEEE 802.1D [IEE04]. Since the standard Ethernet frames do not have a priori knowledge of the network topology, an ARP-based (Address Resolution Protocol [Dav82]) topology detection method is implemented in the proposed IO controller model. As the IO controller models the data link layer, the ARP is used to solve the problem of topology detection based on matching MAC addresses. If a controller receives a message with an unknown MAC address as its forwarding destination, the controller will: (1) buffer the message; (2) send ARP broadcast message; (3) receive the response message and update its MAC table; and (4) forward the buffered frame.
A star topology is modeled by a four-port PROFINET IO controller and three dual-port PROFINET IO controllers.

For example in Figure 4, the network is configured as a start topology. During the network startup phase, every IO controller does not possess any information about network topology and the MAC table maintained by the MAC Relay is empty. If a controller, for example Controller 1, sends a message to Controller 3, the initial knowledge Controller 1 has is only the MAC address of Controller 3. Therefore, if the MAC Relay of Controller 1 looks up the MAC address of Controller 3 in its MAC table, there is no existing record. If this situation occurs, Controller 1 sends an ARP broadcast message, which carries the MAC address of Controller 3 in the target hardware address field. Consequently, the central node in the topology, i.e., Controller 2, receives the broadcast message. First, Controller 2 updates its MAC table to add a new record for the MAC address of Controller 1 (MAC address learning). Then, Controller 2 checks if its MAC address matches the MAC address carried in the broadcast message. If not, Controller 2 has to forward this message to all available ports. Finally, Controller 3 receives the broadcast message and sends an answer back to Controller 2. Controller 2 updates its MAC table for the MAC address of Controller 3 and forwards the answer to Controller 1. After Controller 1 receives the message, a data link is established.

5.2. Timing Synchronization

One crucial system requirement of PROFINET IO network is that all IO controllers must be synchronized. PROFINET IO defines the PTCP (Precision Transparent Clock Protocol) [PRO07b] to share the time reference among IO controllers. PTCP is similar to the IEEE 1588 PTP (Precision Time Protocol) standard [IEE08]. According to PTCP, the clocks in IO controllers are organized in a master-slave synchronization hierarchy with a grandmaster clock at top that determines the reference time for the complete system. The synchronization is accomplished by exchanging PTCP timing messages between slaves and their master. Slave clocks use the timing information to adjust their clocks, i.e., keep synchronized with their timing masters.

PTCP-based timing synchronization is depicted in Figure 5. We assume the master-slave synchronization hierarchy is given a priori. The local port in each IO controller is connected with an application-level module that presents a PTCP application, i.e., PTCP master, PTCP bridge, or PTCP slave. In order to correct the timing error between a slave and its master. Three parameters
Figure 5: Modular modeling of PTCP-based synchronization.

are measured:

1. Since a slave relies on the timestamps generated from its local clock to adjust the time provided by its master, it is important to measure the Clock Drift between slave and its master first. Two timing messages with increased sequence number are sent from the master to the slave with timestamps $T_{m2}$ and $T_{m1}$. The slave receives these two messages at $T_{s2}$ and $T_{s1}$. The clock drift is defined as

$$R_{drift} = \frac{T_{m2} - T_{m1}}{T_{s2} - T_{s1}}.$$  \hfill (1)

2. PTCP uses the peer delay mechanism to measure the port-to-port propagation delay between two ports. The average Link Delay is calculated as

$$D_{link} = \frac{(T_4 - T_1) - (T_3 - T_2)}{2}.$$  \hfill (2)

3. PTCP defines the Bridge Delay $D_{bridge}$ (residence time) as the time duration that a frame used to travel from a controller’s ingress port to its egress port, is defined as $D_{bridge} = T_6 - T_5$. Finally, the Timing Error $T_e$ between a slave clock and its master clock is calculated using

$$T_e = T_s - (D_{link1} + D_{bridge} + D_{link2}) - T_m.$$  \hfill (3)

5.3. Optimising Bandwidth Usage

The transition from the green interval to the reserved red interval is preceded by a yellow interval in which an IRT-suitable switch accepts only frames that can be completely transported before the start of the next reserved interval. If the forwarding of these frames before the start of the next reserved interval is not assured, these frames are stored temporarily and sent in the next green interval [PRO09]. Figure 6 shows an example of yellow interval transmission to optimize bandwidth usage, at same time assuring that no IRT frames are blocked in the next red interval.
Figure 6: Example sequence of frames transmitted in a yellow interval.

In (a), frame 61 (with priority 6) has arrived and is currently transmitted in the green interval. Before the end of this transmission, four frames arrive with the sequence 51, 41, 31, 52, 32, see (b). New frames are queued according to their priorities. After the transmission of frame 61, the yellow interval transmission filter is invoked. This filter checks which frame is suitable to be sent in the remaining time frame, i.e., the transmission delay for the evaluating frame shorter than the remaining time budget. According to this algorithm, in the example only frame 51, see (c), and frame 41, see (d), are legitimate for transmission.

We modeled the yellow interval defined in the PROFINET standard version 2.2. Therefore, during the yellow transmission interval, a queued frame is evaluated as a whole: If the transmission delay for sending this frame is greater than the available yellow interval time budget, this frame will not be sent.

6. Conclusions

In this paper, we presented a system-level modeling and simulation approach of modular networked PROFINET IO controllers. First, we introduced the fundamentals of an existing actor-oriented modeling approach. Then, we showed the modular design of PROFINET IO controllers. This modular design allows to build complex topologies and to simulate entire PROFINET IO networks at system-level. Finally, we focused on the modeling of the essential functionalities for networked IO controllers, i.e., synchronization and topology detection, to enable functional and timing simulation. As future work, we model the PROFINET IO controller according to the next IEC standard version 2.3. In the new standard, some functionalities will be modified, e.g., segment-
tation of frames during the yellow interval. Another aspect left out is the evaluation of simulation performance, which will also be covered in future work.

7. Acknowledgements

The work has been partially supported by EFRE funding from the Bavarian Ministry of Economic Affairs (Bayerisches Staatsministerium für Wirtschaft, Infrastruktur, Verkehr und Technologie) as a part of the ‘ESI Application Center’ project.

References


