Specification Enforcing Refinement for Convertibility Verification

Partha S Roop, Alain Girault, Roopak Sinha, and Gregor Goessler

Abstract—Protocol conversion deals with the automatic synthesis of an additional component or glue logic, often referred to as an adaptor or an interface, to bridge mismatches between interacting components, often referred to as protocols. A formal solution, called convertibility verification, has been recently proposed, which produces such a glue logic, termed as a converter, so that the parallel composition of the protocols and the converter also satisfies some desired specification. A converter is responsible for bridging different kinds of mismatches such as control, data, and clock mismatches. Mismatches are usually removed by the converter (similar to controllers in supervisory control of Discrete Event Systems (DES)) by disabling undesirable paths in the protocol composition.

This paper formulates a generalization of this convertibility verification problem, by using a new refinement called specification enforcing refinement (SER) between a protocol composition and a desired specification. The existence of such a refinement is shown to be a necessary and sufficient condition for the existence of a converter. We also propose an approach to automatically synthesize a converter if a SER relation exists. The proposed converter is capable of the usual disabling actions to remove undesirable paths in the protocol composition. In addition, the converter can perform forcing actions when disabling alone fails to find a converter to satisfy the desired specification. Forcing allows the generation of control inputs in one protocol that are not provided by the other protocol. Forcing induces state-based hiding, an operation not yet achievable using DES control theory.

Index Terms—Protocol conversion, forced simulation.

I. Introduction

System-on-Chip (SoC) [2] design involves the interconnection of many pre-designed components, called IPs. While, a set of selected IPs may meet the functional requirements, their protocols may not be consistent, leading to several kinds of mismatches. The most common types of such mismatches are control, data, and clock mismatches. Control mismatches happen when the sequencing of control signals between protocols is inconsistent. Data mismatches happen when the data-widths of the two protocols differ and additional buffers are needed to manage loss-less data communication. Clock mismatches are common between IPs having different clock frequencies. The first approach to demonstrate the problem and some informal steps for a solution was proposed in [4]. Many techniques have been proposed since then to solve one or more of these incompatibilities using automated algorithmic techniques [1], [7], [9]. Their goal is to synthesize some additional glue logic, termed as a converter/interface/adaptor (from now on termed converter) to bridge these mismatches. While these techniques were automated, they failed to address several questions. These include how to formally model protocols and their interaction, and when such models are available, how to determine if a converter exists for a given set of protocols? Moreover, once the existence of a converter is determined, how to synthesize it? More recently, a set of formal techniques have been proposed [3], [6], [10], [14], [15] to address these questions. Table I compares these approaches over a range of features. The features listed as columns of Table I are: the modelling language for protocols, the language for describing desired specifications, multiple protocols (two and more than two), type of conversion algorithm, whether the approach can handle uncontrollable events, event buffering, whether data-width mismatches are handled, whether clock mismatches are handled, and finally the type of control action used. Among the proposed techniques, most approaches use Labeled Transition Systems (LTS) to describe both protocols and specifications, except [14] where CTL temporal logic is used for the specification part. Also, except the approach of [3], [15] which use oversampling [5] to bridge clock mismatches, all other techniques ignore clock mismatches.

Central to protocol conversion is the use of a suitable controller that is used to remove undesirable paths in the protocol composition. This is done using the well known idea of disabling from Discrete Event Systems (DES) control theory [11]. Here, a supervisor or controller is synthesized to control a plant so that the controlled system (composition of the controller and the plant) satisfies the desired specification. The role of the controller is to disable all controllable paths that violate the specification while leaving uncontrollable transitions untouched. In this domain, the plant and the specification are described as labelled transition systems (LTS) over an alphabet partitioned into controllable and uncontrollable events. While a converter is like a DES controller, the convertibility verification problem is not identical to DES supervisory control. Firstly, in convertibility verification there is a need to buffer events as an event generated by one protocol may be needed by another protocol at a later time. Secondly, there may be data and clock mismatches between protocols that are specific problems not addressed by DES supervisory control. Yet, both domains need to deal with controllable and uncontrollable events.

Kumar and Nelvagal proposed a formulation mapping the convertibility verification problem to the DES supervisory control problem [6] in a simplistic setting. Passerone et al. [10] developed a game theoretic formulation to solve the convertibility verification problem. Subsequently, in D’Silva et al. [3] a refinement based solution is developed for checking protocol compatibility. Recently, Sinha et al. [14] proposed a module checking based solution to the convertibility verification problem. Unlike earlier approaches, this approach bridges both control and data-width mismatches. Finally, Tivoli et al. [15] proposed a contrasting
solution to the same problem that they termed as adaptor synthesis. This formulation ensures that timing constraints are met, events between protocols are adequately buffered, and that the composition is deadlock free.

Table I summarizes the features of all these formal approaches to convertibility verification. A key feature of the existing formal solutions (rows 1 to 5) is that they are based on disabling-based controllers. The last row of the table compares the existing solutions to the solution proposed in this paper. We extend the capability of the controller to allow, not only disabling actions, but also forcing actions [12]. Forcing actions are introduced to solve specific needs of the problem domain, namely the need for state based hiding, which is not possible using conventional disabling-only controllers. We will elaborate on this aspect through a motivating example in the next section.

### II. Example and method overview

Figure 1 shows an overview of the convertibility verification problem. The actual protocols and their composition is shown in Fig. 2. We use CCS [8] style primed and unprimed symbols to indicate outputs and inputs respectively. E.g., in Figure 1, $a$ is produced by the handshake protocol and read by the serial protocol. Event $T$ models an internal tick, and is therefore neither an input nor an output.

![Fig. 1. Overview of convertibility verification](image)

**Table I**

<table>
<thead>
<tr>
<th>Approach</th>
<th>Input (Protocols)</th>
<th>Input (spec.)</th>
<th>Multiple-Processes</th>
<th>Algorithm</th>
<th>Uncontrollable Signal Buffering</th>
<th>Data</th>
<th>Multi-clock</th>
<th>Control Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kumar et al. [6]</td>
<td>LTS, LTS</td>
<td>LTS</td>
<td>√</td>
<td>supervisory control</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Passerone et al. [10]</td>
<td>LTS, LTS</td>
<td>×</td>
<td>×</td>
<td>same-theoretic</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>D’Siva et al. [11]</td>
<td>SPA</td>
<td>×</td>
<td>×</td>
<td>refinement</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Tisvli et al. [12]</td>
<td>LTS</td>
<td>×</td>
<td>×</td>
<td>controlled coverability</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Sinha et al. [13]</td>
<td>LTS, CTL</td>
<td>×</td>
<td>×</td>
<td>model-checking</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>SER Refinement</td>
<td>LTS, LTS</td>
<td>×</td>
<td>×</td>
<td>refinement</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
</tbody>
</table>

**Fig. 2. Handshake and serial protocols and their synchronous parallel composition**

**Specifcation**

Compared to the example in [10], the handshake protocol $P_1$ has an additional initialization step through an input $b$ that is needed prior to establishing a handshake. Once initialized, $P_1$ outputs the signal $a$ in the next instant. It then outputs the signal $b$ after some random number of clock ticks, modelled using self loops labelled by $T$. On the other hand, the serial protocol $P_2$ expects to read input $b$ immediately in the instant following the reception of $a$. The synchronous parallel composition of $P_1$ and $P_2$, noted $P_1∥P_2$, is also depicted in the same figure. We used synchronous parallel composition along with $T$-actions to indicate that a given protocol just delays. Other kinds of products, such as the interleaved parallel of CCS [8], were not considered to avoid non-determinism.

We provide a desired specification as shown in Fig. 3. This specification has a notion of completed transactions through the introduction of marked states (depicted by a double circle). This specification enforces that every input must be preceded by its corresponding output (either in the same or a previous step). Moreover, the transmission and reception of a must precede that of $b$.

**Fig. 3. A desired specification with a marked state**

**Overview of the proposed methodology:** The handshake-serial protocol pair is a mismatched protocol because of the following reasons:

- Initialization input $b$ required by $P_1$ is not provided by $P_2$ at all.
• After an $a$ is output by $P_1$, $P_2$ expects a $b$ immediately while $P_1$ may produce $b$ after producing any number of $T$ outputs.

Given such mismatching protocols and a desired specification, the goal of convertibility verification is to determine the existence of a converter that can bridge the mismatches, so that the overall system with the converter satisfies the desired specification. The converter can buffer inputs and forward them when necessary; it can also disable controllable paths in the composition. In the example of Fig. 2, $P_2$ requires a $b$ immediately after having read the input $a$. Hence, when the protocol composition is in state $(s_1, t_0)$, the converter reads the $a$ produced by $P_1$, and forces $P_2$ to make a $T$ transition during this step. During the next transition, the converter transmits this $a$ to $P_2$, while $P_1$ does a $T$ transition. We therefore say that the converter buffers the event $a$. In addition to buffering $a$, the converter also disables all other transitions of $(s_1, t_0)$.

Note that by relying only on the usual converter actions (buffering and disabling), we would not be able to generate a converter for the handshake-serial protocol pair. This is because this protocol pair requires a $(b, T)$ input in its initial state, while the specification allows the input of $b$ only after the generation and consumption of $a$ has been completed. Also note that the input $(b, T)$ is not present in the converter’s buffers initially. Thus, using conventional techniques based on DES supervisory control, we would fail to produce a correct converter. Hence, we propose a new way to control the protocol composition using a converter, as shown in Figure 4(a). The converter first forces the transition from $(s_0, t_0)$ to $(s_1, t_0)$ by generating the $(b, T)$ input. These are inputs required by the protocols and are not present in the buffers. Forced inputs are marked within square brackets "[ ]" in the converter to distinguish them from other inputs that are either read from the converters buffers (events generated by the protocols in the past and not been consumed yet), or are directly read from the other protocol in the current instant. Since forced inputs are not produced by any of the protocols and have not been consumed from the buffers, they can be hidden by the composition of the protocols and the converter, thus satisfying the specification.

The composition of the converter with the protocols is shown in Fig. 4(b). Note that the forced transition from $(s_0, t_0)$ to $(s_1, t_0)$ is hidden in the composition (hence labelled by $\tau$). Forcing enables state based hiding, different from global hiding achieved by DES supervisory control (based on the hiding operator of CCS [8]). For instance, in the composed system of Fig. 4(b), the event $b$ is hidden in the initial state cs0, while it is visible later in the state cs3. This is not directly achievable using DES controllers.

Forcing guides a controlled system to a successor state whenever the current state fails to satisfy the requirements of the specification. Like in DES supervisory control, where only controllable transitions may be disabled, only forceable transitions can be forced, and the user must specify a subset of inputs that can be forced. We elaborate on details of forcing in the next section.

We solve the convertibility verification problem as follows. A protocol pair $(P_1, P_2)$ is said to satisfy a specification $S$ when the language accepted by the synchronous parallel composition of the protocols, $L(P_1 \parallel P_2)$ is a subset the specification’s one, $L(S)$. We only look at visible traces when checking this, because of forcing actions. Otherwise, the protocols are mismatching. In this case, we propose a new refinement relation from the composite protocols $P_1 \parallel P_2$ to the specification $S$. We also show that the existence of such a refinement relation is a necessary and sufficient condition for the existence of the converter. Finally, we provide an algorithm to synthesize the converter given such a relation. Our method for protocol conversion is based on DES supervisory control [11] and forced simulation [12]. While we have motivated the proposed method using the case of two protocols, the proposed approach generalizes straightforwardly to an arbitrary number of protocols.

III. Convertibility verification using specification enforcing refinement

A. Preliminaries

Error-free communication between protocols implies that traces in the protocol composition always respect the event sequencing described in the specification. In this section, we define the refinement that enforces a desired specification over a protocol composition to ensure error free communication of the two components. We start by introducing the models of the protocols and of the specification.

We component protocols by using labelled transition systems (LTS).

**Definition 1:** An LTS is a tuple $P = (\Sigma, Q, q_0, \rightarrow, \epsilon_0)$, where $\Sigma$ is the alphabet of actions, which is partitioned into the set of input actions and the set of output actions ($\Sigma = \Sigma_I \cup \Sigma_O \cup \{T\}$), $Q$ is the set of states, $\rightarrow \subseteq Q \times \Sigma \times Q$ is the transition relation, and $q_0 \in Q$ is the initial state. The transition relation is also written as $q \xrightarrow{\sigma} q'$ if and only if $(q, a, q') \in \rightarrow$. The language $L(P)$ is the set of all finite and infinite words generated by the LTS $P$.

Output events are emitted by the components while input events are received. We use primed symbols to represent output events and unprimed symbols to represent input events. Fig. 2 depicts the LTS of the handshake protocol to the left and that of the serial protocol at the top. The handshake protocol awaits event $b$ in state $s_0$ and outputs event $a$ in state $s_1$. We term all the events labeling the outgoing transitions from a given state $q$ as $Label(q) = \{a, [a]q', \epsilon a, q' \}$.

Protocol interaction is defined using their synchronous parallel composition. The parallel composition of handshake and serial protocols is depicted in Fig. 2.

**Definition 2:** Let $P_1 = (\Sigma_1, Q_1, \rightarrow_1, \epsilon_1)$ and $P_2 = (\Sigma_2, Q_2, \rightarrow_2, \epsilon_2)$ be two protocols. The synchronous product of
where \( q_1, q_2 \) are a new refinement relation. We start by defining the notion of a refinement relation. We will subsequently show that this refinement guarantees the existence of a converter. For notational clarity, we represent the existence of a converter. Thanks to the associativity of the synchronous product, our formalization generalizes straightforwardly to an arbitrary number of protocols. The only restriction is that communications between the protocols are point-to-point; formally, for each LTS protocol \( P_1 = (\Sigma_1, Q_1, \rightarrow, q_1^0) \) such that \( \Sigma_1 = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3 \cup \Sigma_4 \) and \( e \in \Sigma_1 \), there exists a unique protocol \( P_2 \) such that \( j \neq i \) and \( e \in \Sigma_2 \) (vice-versa).

B. Refinement Relation

We now provide a solution for convertibility verification using a new refinement relation. We start by defining the notion of satisfaction of a specification for a given protocol composition model.

Definition 4: A model of two interacting protocols, \( M = P_1 \parallel P_2 \), satisfies a specification, denoted \( M \models S \), if and only if \( L(M) \subseteq L(S) \).

Since forcing leads to some actions being hidden in the resulting system, we weaken the classical definition of specification satisfaction of [11] to deal with these hidden transitions in the composition.

Definition 5: An LTS \( M = (\Sigma, Q, \rightarrow, q^0) \) weakly satisfies a specification \( S \), denoted \( M \models_w S \) if and only if \( L([M]) \subseteq L(S) \) where \( L([M]) \) is \( \{ (\alpha | \sigma | L(M)) \} \) and \( \alpha \) is the word obtained by deleting all \( \tau \) actions from the word \( \sigma \).

Now, we introduce a new refinement relation, called specification enforcing refinement (SER), from \( P_1 \parallel P_2 \) to a specification \( S \).

We will subsequently show that this refinement guarantees the existence of a converter. For notational clarity, we represent \( P_1 \parallel P_2 \) as an LTS \( M \) that is equal to the composition of the two protocols: \( M = (\Sigma_1 \times \Sigma_2, Q_1 \times Q_2, \rightarrow, (q_1^0, q_2^0)) \) with \( q^0 = q_1^0 \) if \( (a, b) = (a, b) \) and \( (q_1^0, q_2^0) \) and \( \sigma \) is a shorthand for \( (a, b) \).

Exactly like in the framework of DES supervisory control, we partition the set \( \Sigma_M \) into two subsets: the subset \( \Sigma_{M_c} \) of controllable events and the subset \( \Sigma_{M_u} \) of uncontrollable events. We introduce two additional subsets: the subset \( \Sigma_{M_c} \) of buffered events and the subset \( \Sigma_{M_f} \) of forceable events. While \( \Sigma_{M_c} \) and \( \Sigma_{M_u} \) are static sets (i.e., they don’t change over time), \( \Sigma_{M_f} \) is a dynamic set since it depends on the current set of inputs in the converter’s buffers. The set \( \Sigma_{M_f} \) is obtained by removing all current buffered inputs from the set of controllable inputs \( \Sigma_{M_c} \). This is because buffered inputs have been produced in the environment; hence, they are visible and can’t be hidden through forcing by the converter. Like \( \Sigma_{M_b} \), \( \Sigma_{M_f} \) is also a dynamic set (i.e., its contents change over time).

In the current setting, only outputs are uncontrollable, because the converter can not exert any influence on their generation. We now formally define these sets. We introduce a predicate \( \text{inBff} \) that returns true when a given input event is in the converter’s buffers.

Definition 6: Given an LTS \( M = P_1 \parallel P_2 \), the subset \( \Sigma_{M_c} \) of the controllable events of \( M \) is \( \Sigma_{M_c} = \{ \sigma | \sigma = (a, b) \land a \in \Sigma_M \land b \in \Sigma_M \} \), the subset \( \Sigma_{M_u} \) of uncontrollable events is \( \Sigma_{M_u} = \Sigma - \Sigma_{M_c} \), the subset \( \Sigma_{M_f} \) of buffered events is \( \Sigma_{M_f} = \{ \sigma | \sigma \in \Sigma_{M_c} \land \text{inBff}(\sigma) \} \), and finally, the subset \( \Sigma_{M_f} \) of forceable events is \( \Sigma_{M_f} = \Sigma_{M_c} - \Sigma_{M_u} \).

We now define the new refinement relation.

Definition 7: Let \( M = (\Sigma_M, Q_M, \rightarrow, q^0_M) \) and \( S = (\Sigma_S, Q_S, \rightarrow, q^0_S) \) be the LTS of the protocol composition and the specification respectively. A relation \( R \subseteq \Sigma_M \times Q_M \times \Sigma^* \) is called a specification enforcing refinement (SER) from \( M \) to \( S \), if the three conditions below hold. The notation \( q^0_M \rightarrow^R q^0_S \) is used as a shorthand for \( (q^0_M, q^0_S, s) \in R \), where \( s \) is any word over \( \Sigma_M \) whose maximum length is bounded by \( |q^0_M| \).

1. \( q^0_M \) and \( q^0_S \) are directly related if \( q^0_M \) has at least one transition having the same label as a transition from \( q^0_S \) (a matching transition pair). Moreover, for the matching transition pair \( q^0_M \rightarrow q^0_S \) in \( M \) and \( q^0_M \rightarrow q^0_S \) in \( S \), the successor states \( q^0_M \) and \( q^0_S \) are also related via some forcing sequence \( s \in \Sigma_M \). In this case, \( q^0_M \rightarrow^R q^0_S \).

2. \( q^0_M \) and \( q^0_S \) are related via some forcing sequence \( s \). If there exists a successor state \( q^0_M \) in \( M \) such that \( q^0_M \) is reachable from \( q^0_M \) via a forcing sequence \( s \) where \( q^0_M \) and \( q^0_S \) are related via \( R \). In this case, \( q^0_M \rightarrow^R q^0_S \).

These possibilities are formalized in the first two conditions of Definition 7. In addition, the start states are required to be related via some forcing sequence \( s \), which corresponds to the third condition.

Between the handshake-serial example in Fig. 2 (M) and the specification shown in Fig. 3 (S), an SER exists as follows:

\[
R = \{(q_0, t_0), q_0, [b, T], (q_1, t_0), q_0, \varepsilon, (q_2, t_0), q_2, \varepsilon, (q_2, t_1), q_3, \varepsilon, (q_0, t_0), q_1, \varepsilon)\}
\]

In general, there may be many SER relations between \( M \) and \( S \). For example, \( R' = \{(q_0, t_0), p_0, \varepsilon, \} \) is also a valid SER relation between \( M \) and \( S \).
C. Marked compatibility
The goal of convertibility verification is to determine the conditions under which a suitable converter between M and S exists. Note that an SER refinement between M and S alone doesn’t guarantee the existence of such a converter. For example, consider the model M of the composite protocols as shown in Fig. 2 and the specification as shown in Fig. 3. We can define an SER \( R' = \{(s_0, t_0), p_0, \varepsilon\} \). The corresponding trivial converter will just enable the self-loop transition in the initial states of the two protocols. However, such a converter doesn’t ensure completed transactions in the protocols. Marked states (e.g., state \( q_1 \) in Fig. 3) in the specification are used to represent completed transactions.

To prevent synthesis of trivial converters, we define marked compatible SERs.

**Definition 8:** Let \( R \) be an SER relation between M and S. A path \( q_M \rightarrow q_{M_1} \rightarrow q_{M_2} \cdots \rightarrow q_{M_n} \) in M is a compatible path to a path \( q_S \rightarrow q_{S_1} \rightarrow q_{S_2} \cdots \rightarrow q_{S_m} \) in S if \( (q_{M_i}, q_{S_i}, s_i) \in R \) for some \( s_i \in \Sigma_{MC} \) and for all i \( \in \{1..n\} \): \( (q_{M_i}, q_{S_i}, s_i) \in R \) for some \( s_i \in \Sigma_{MC} \).

**Definition 9:** An SER relation \( R \) between M and S is marked compatible if for every \( (q_M, q_S, s) \in R \) there exists \( (q_M', q_S', s') \in R \) such that there exists a path from \( q_M \) to \( q_{M}' \) and a compatible path from \( q_{S} \) to \( q_{S}' \) in \( \Sigma_{MC} \), i.e., \( q_S' \) is a marked state of S.

**Definition 10:** Let M and S be a model of protocol composition and a specification, respectively. M \( \preceq_{SER} \) S if there exists a specification enforcing refinement from M and S that is marked compatible.

D. Converters
We now synthesize converters between protocols. A converter is an LTS whose role is to appropriately guide the protocols. In our framework, a converter can perform the following action.

1. Disabling a transition of the protocols: Remove undesirable communication paths that violate the specification. This operation is identical to the controllers in DES [11]. The disabled transitions must be controllable.
2. Forcing a transition of the protocols: Automatically guide the protocols to a successor state from its current state so that the future state is consistent with a specification state. This is done by generating the suitable forceable inputs on a path.
3. Buffering a communication between the protocols: If a given input generated by one of the protocols cannot be consumed by the receiving protocol, a converter can buffer this event so that it can be forwarded in the future.

E. Converter Synthesis from an SER Relation
Converters can be derived automatically once an SER is established between M and S. Given an SER \( R \), the states of the converter \( Q_C \) are exactly the elements of R. We now formalize the relationship between an SER relation and a corresponding converter. We start by defining precisely-forced SERs so as to ensure that forcing is always performed in a unique fashion from any forced state.

**Definition 11:** An SER relation \( R \) is precisely-forced iff \( (q_M, q_S, s_1) \in R \) and \( (q_M, q_S, s_2) \in R \) implies that \( s_1 = s_2 \). Given an SER \( R \), a precisely-forced SER \( R' \) can be automatically derived from \( R \) and always exists. From a precisely-forced SER, we now build a converter that derives from it:

**Definition 12:** Let \( R \) be a precisely-forced SER between a model M and a specification S. The converter derived from R is the LTS \( C_R = (\Sigma_M \cup \Sigma_M', R, \varepsilon, -c, (q_M^R, q_S^R, s)) \), where \( \Sigma_M' = \{\{\sigma, \sigma' \in \Sigma_M \} \wedge |\sigma| = |\sigma'| \} \) and \( |\sigma'| \) denotes the forced action over event \( \sigma \), and \( -c \) is defined by the following two rules:

- **[Matched-event]:** If \( (q_M, q_S, \varepsilon) \in R \wedge (q_M', q_S', s') \in R \wedge q_M \overset{\sigma}{\rightarrow} q_M' \wedge q_S \overset{\sigma}{\rightarrow} q_S' \), then \((q_M, q_S, \varepsilon) \overset{\sigma}{\rightarrow} (q_M', q_S', s') \).
- **[Forced-event]:** If \( (q_M, q_S, \alpha, s) \in R \wedge q_M \overset{\sigma}{\rightarrow} q_M' \), then \((q_M, q_S, \alpha, s) \overset{\sigma}{\rightarrow} (q_M', q_S', s) \).

For the handshake-serial protocol pair shown in Fig. 2 and the specification S shown in Fig. 3, a converter generated by our approach is shown in Fig. 4(a). It first forces the transition from \( (s_0, t_0) \) to \( (s_1, t_0) \) by generating the events \( \{b, T\} \). Subsequently, it reads the a produced by \( P_1 \) and forwards it while allowing \( P_2 \) to remain in its initial state through a T transition. It then forwards the buffered a to the \( P_2 \) while allowing \( P_1 \) to remain in its current state through a T transition. Note that there is also the choice of directly allowing the \( (a', a) \) transition in the state \( c \) instead of first buffering a and later forwarding a. Hence, the generated converter keeps all possibilities.

Having established the relationship between a given SER and the associated converter, we now define well-formed converters. Well-formed converters ensure that protocols always complete their transactions.

**Definition 13:** Let \( R \) be a SER between a model M and a specification S. A converter \( C = (\Sigma_C, Q_C, -c, C^R_C) \) derived from \( R \) is said to be well-formed if the two following conditions hold:

- **[Forced-alone]:** For all \( q, q' \in Q_C \) and \( \alpha \in \Sigma_C \) such that \( q \overset{\alpha}{\rightarrow} q' \), \( \overset{\alpha}{\not\rightarrow} q'' \), and \( \overset{\alpha}{\not\rightarrow} q''' \), for some \( q'', q''' \in Q_C \) and some \( \sigma, \beta \in \Sigma_C \), then \( \sigma = \alpha \), \( q'' = q''' \), \( \beta = \alpha \), and \( q'' \not\rightarrow q''' \).
- **[Marked-path]:** For any state \( q \in Q_C \), there always exists a path to a state \( q' \in Q_C \) such that \( q' = (q_M, q_S, s) \) and \( q_S \in Q_S^m \).

The state graph of a well-formed converter has only one successor for states where forcing is performed. Other states may have more than one successor. Moreover, from every state of a well formed converter, a marked state can always be reached. A state in the converter is called a marked state if the corresponding component of R is of the form \((q_M, q_S, s)\) such that \( q_S \) is a marked state. It is easy to note that any converter derived from a deterministic and marked compatible SER is always well-formed.

In our framework, event buffering is achieved thanks to the state space of the converter. This is the case of the event \( a \) in the converter of Figure 4.

**Lemma 1:** Let \( R \) be a precisely-forced SER between a model M and a specification S, and let \( C_R \) be the converter derived from \( R \). If \( R \) is marked compatible, then \( C_R \) is well-formed.

**Proof:** Let \( C_R = (\Sigma_C, Q_C, -c, C^R_C) \) be the converter derived from \( R \). Proof of Condition [Forced-alone]. Let \( q, q' \in Q_C \) and \( \alpha \in \Sigma_C \) such that \( q \overset{\alpha}{\rightarrow} q' \). We prove by contradiction that \( \overset{\alpha}{\not\rightarrow} q'' \in Q_C \) and \( \overset{\alpha}{\not\rightarrow} q''' \in Q_C \) such that \( q \overset{\alpha}{\rightarrow} q' \). Since \( \alpha \) is a forced action, Rule [Forced-event] implies that \( q = (q_M, q_S, \alpha, s) \in R \). Furthermore, since \( q \overset{\alpha}{\rightarrow} q' \), Rule [Matched-event] implies that
Lemma 1. Also, Definition 12. Since the trace set of the converter, we see that all events are deterministic. Thus, the specification respectively. There exists a well-formed converter.

Proof of Condition [Marked-path]: Direct consequence of Definition 9.

We now define the product operation for composing a converter $C$ with a pair of protocols represented as an LTS $M$.

Definition 14: Let $C = \langle \Sigma_C, Q_C, R_C, \tau_C, q_C^0 \rangle$ and $M = \langle \Sigma_M, Q_M, R_M, \tau_M, q_M^0 \rangle$ be a converter and a model respectively. The forced composition $C \bowtie M$ is defined by the following two rules:

- **[Taut-trans]:** $(q_C, q_M) \xrightarrow{\alpha} (q'_C, q'_M)$ if $q_C \xrightarrow{[\alpha]} q'_C$ and $q_M \xrightarrow{\alpha} q'_M$ for some $\alpha \in \Sigma_M$.
- **[Event-trans]:** $(q_C, q_M) \xrightarrow{\tau} (q'_C, q'_M)$ if $q_C \xrightarrow{\tau} q'_C$ and $q_M \xrightarrow{\tau} q'_M$ for all $\tau \in \Sigma_M$.

Lemma 2: The forced composition $C \bowtie M$ is deterministic if both $C$ and $M$ are deterministic and if the converter $C$ is well-formed.

Proof: The proof follows directly from the fact that both $C$ and $M$ are deterministic and from the Rules [Taut-trans] and [Event-trans].

The next result states that a marked compatible SER relation between $M$ and $S$ is a necessary and sufficient condition for the existence of a correct converter.

Theorem 1: Let $M$ and $S$ be deterministic LTSs of the model and the specification respectively. There exists a well-formed and deterministic converter $C$ such that $C \bowtie M \models w$ if and only if $M \leq_{\text{SER}} S$.

Proof: Sufficient Condition: The proof is constructive. Given $M \leq_{\text{SER}} S$, there exists a precisely-formed and marked compatible SER $R$. We can construct a converter $C$ using Definition 12. Since $R$ is marked compatible $C$ is well formed (Lemma 1). Also, $C$ is deterministic since both $M$ and $S$ are deterministic. Thus, $C \bowtie M$ is also deterministic. Now it is easy to see that all $\tau$-projected traces of $C \bowtie M$ are contained in the trace set of $S$.

Necessary Condition: Given a well-formed converter $C$ such that $C \bowtie M \models w$, we need to prove that $M \leq_{\text{SER}} S$. Since $L(C \bowtie M) \subseteq L(S)$ and as $C$ is well-formed and $C$, $M$ and $S$ are deterministic, this result follows.

IV. Prototype tool and results

A local, on-the-fly tableau construction algorithm, similar to [13], is used for converter synthesis. The algorithm employs two tableau rules, for disabling and forcing, and some termination conditions. The worst-case complexity of the algorithm is $O(|Q_M|^2 |Q_S|^2 2^{2 |\Sigma_M|})$ where $|Q_M|$ and $|Q_S|$ are the sizes of the state sets of $M$ and $S$, and $|\Sigma_M|$ is the size of the event set of $M$.

The algorithm is intuitively described by using the tableau shown in Fig. 5 (generated for the handshake-serial example). The inputs to the algorithm are the initial states $(s_0, t_0)$ and $q_0$ of the model and the specification, and an empty set of buffered events. These inputs form the initial assertion $A_1$ of the tableau, which is recursively broken down into sub-assertions using tableau rules. An assertion returns success if its sub-assertions return success. An infinite resolution of assertions into sub-assertions is prevented by termination condition that check if there is an existing assertion (ancestor) which is identical to a newly created assertion (current assertion). If such is the case, the current assertion is not resolved into further sub-assertions (the assertion is completed). For example, in Fig. 5, the assertions $A_2$, $A_5$ and $A_8$ are not resolved further because they are identical to previously processed assertions $A_1$, $A_6$ and $A_1$ respectively. $A_2$ returns failure because the path from $A_1$ to $A_2$ does not contain any assertion corresponding to the marked state $q_1$, whereas $A_8$ returns success because the path from $A_1$ to $A_8$ contains an assertion $A_7$ that corresponds to $q_1$. $A_5$ returns success because an identical assertion $A_6$ has already returned success.

The algorithm exits when the initial assertion ($A_1$) returns success (or failure) to indicate that a successful (or failed) tableau has been generated. Each (non-completed) assertion in a successful tableau corresponds to a unique state in the converter. For example, each (non-completed) assertion in the tableau shown in Fig. 5 corresponds to a unique state in the converter shown in Fig. 4(a), and the initial assertion $A_1$ corresponds to the initial state of the converter.

![Fig. 5. Tableau for the handshake-serial example](image)

Implementation Results

Tab. II shows a set of results obtained by executing the SER algorithm over some well-known conversion problems described in literature [3, 6, 7, 10, 13]. Each entry in the table describes the protocols and specification involved and the types converters obtained by using classical approaches and the SER conversion algorithm (D-disabling, DF-disabling and forcing). Problem 1 is the handshake-serial problem presented in [10] while problem 1A is an extension of problem 1 that involves an extra input transition in the handshake protocol (see Fig. 2). Problems 2, 3, 4 and 5, that are taken from other articles on protocol conversion, are extended in similar fashion to problems 2A, 3A, 4A and 5A. While classical techniques can handle problems 1, 2, 3, 4 and 5 only, we could generate converters for these problems as well as their variants 1A, 2A, 3A, 4A and 5A. Although the implementation does not address the question of finding optimal converters (those with minimum number of forcing steps), it can generate all possible converters. The algorithm not only finds converters where previous approaches fail, it also preserves the full design space by finding all possible solutions.

V. Conclusions

Protocol conversion is required while creating a complex system (such as a System-on-Chip) from pre-designed components (called IPs) which have mismatching communication protocols. Convertibility verification automatically determines if a suitable glue-logic, called a converter, exists to bridge such mismatches. Converters are inspired by controllers from DES supervisory control theory and hence bridge mismatches through disabling
of undesirable communication paths while also performing additional control actions such as event buffering. This paper presents a more generalized converter synthesis technique that performs forcing of actions in addition to the conventional disabling. Forcing actions are used to hide extra control sequences that are required by the protocols but not by the desired specification. Forcing induces state-based hiding that is not possible using standard hiding operators in DES supervisory control.

We have proposed a new refinement relation, called specification enforcing refinement (SER), between a given protocol composition and a desired specification. We have also shown that the existence of this relation is a necessary and sufficient condition for the existence of a suitable converter that enforces the desired specification over the protocols. The proposed approach generalizes existing approaches to convertibility verification, and we have demonstrated it by finding converters for many protocol mismatches that can’t be bridged using existing techniques. Future work will involve extending the formulation to handle data-width and clock mismatches, and finding optimal converters.

References


<table>
<thead>
<tr>
<th>Problem</th>
<th>Specification/Properties</th>
<th>Classical converter types</th>
<th>SER converter types</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Handshake-serial [10]</td>
<td>I/O sequencing</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>1A. Adapted handshake-serial</td>
<td>I/O sequencing</td>
<td>-</td>
<td>DF</td>
</tr>
<tr>
<td>2. ABP receiver, NS sender [7]</td>
<td>No packet loss</td>
<td>D</td>
<td>D, DF</td>
</tr>
<tr>
<td>2A. Adapted ABP receiver, NS sender</td>
<td>No packet loss</td>
<td>-</td>
<td>D, DF</td>
</tr>
<tr>
<td>3. ABP sender, NS receiver [6]</td>
<td>No packet loss</td>
<td>D</td>
<td>D, DF</td>
</tr>
<tr>
<td>3A. Adapted ABP sender, NS receiver</td>
<td>No packet loss</td>
<td>-</td>
<td>D, DF</td>
</tr>
<tr>
<td>4A. Adapted Handshake-Pipeline</td>
<td>Correct data exchange</td>
<td>-</td>
<td>D, DF</td>
</tr>
<tr>
<td>4A. Adapted Producer-Consumer</td>
<td>No over/under flows</td>
<td>-</td>
<td>D, DF</td>
</tr>
</tbody>
</table>

TABLE II IMPLEMENTATION RESULTS