Computing and Informatics, Vol. 29, 2010, 901–928

TIMED CHI: MODELING, SIMULATION AND VERIFICATION OF HARDWARE SYSTEMS

Ka Lok Man

Department of Computer Science and Software Engineering
Xian Jiaotong-Liverpool University
Suzhou, China
e-mail: ka.man@xjtlu.edu.cn

Manuscript received 24 September 2007; revised 19 May 2008
Communicated by Elena Gramatová

Abstract. Timed Chi (χ) is a timed process algebra, designed for Modeling, simulation, verification and real-time control. Its application domain consists of large and complex manufacturing systems. The straightforward syntax and semantics are also highly suited to architects, engineers and researchers from the hardware design community. There are many different tools for timed Chi that support the analysis and manipulation of timed Chi specifications; and such tools are the results of software engineering research with a very strong foundation in formal theories/methods. Since timed Chi is a well-developed algebraic theory from the field of process algebras with timing, we have the idea that timed Chi is also well-suited for addressing various aspects of hardware systems (discrete-time systems by nature). To show that timed Chi is useful for the formal specification and analysis of hardware systems, we illustrate the use of timed Chi with several benchmark examples of hardware systems.

Keywords: Formal methods, formal languages, formal semantics, process algebras, real-time systems, formal specification and analysis of hardware systems, software engineering research

1 INTRODUCTION

The goal of developing a *Formal semantics* is to provide a complete and unambiguous specification of the language. It also contributes significantly to the sharing, portability and integration of various applications in simulation, synthesis and formal verification.

Formal methods provide a set of notations that can be used to build mathematical models of systems; and techniques for automatic verification of such models. Over the years, formal methods have been widely and successfully used in a wide range of problems and in practical applications in both academia and industry for the specification and analysis of many different systems. *Formal verification* is intended to prove some properties (e.g. expressed in temporal logic) hold in the system (i.e. a mathematical model) under analysis. Although formal verification has shown to be very useful for analysis of various systems (e.g. hardware circuits), its power is still limited by the complexity of the analysis that grows very large as the size of the systems increases (namely *state space explosion problem*).

Formal languages with a semantics formally defined in *computer science* increase understanding of systems, increase clarity of specifications and help solving problems and remove errors. Over the years, several flavors of formal languages have been gaining industrial acceptance.

Process algebras [1] are formal languages that have formal syntax and semantics for specifying and reasoning about different systems. They are also useful tools for verification of various systems. Generally speaking, process algebras describe the behavior of processes and provide operations that allow to compose systems in order to obtain more complex systems. Moreover, the analysis and verification of systems described using process algebras can be partially or completely carried out by mathematical proofs using equational theory. In addition, the strength of the field of process algebras lies in the ability to use *algebraic reasoning* (also known as equational reasoning) that allows rewriting processes using axioms (e.g. for commutativity and associativity) to a simpler form. By using axioms, we can also perform calculations with processes. These can be advantageous for many forms of analysis.

Process algebras have also helped achieve a deeper understanding of the nature of concepts like observable behavior in the presence of non-determinism, system composition by interconnection of system components modeled as processes in a parallel context, and notions of behavioral equivalence (e.g. bisimulation) of such systems.

Serious efforts have been made in the past to deal with systems (e.g. real-time systems [2, 3] and hybrid systems [6, 4]) in a process algebraic way. Over the years, process algebras have been successfully used in a wide range of problems and in practical applications in both academia and industry for analysis of many different systems.

On the other hand, the need for a formal and well-defined semantics of a hardware description language is widely accepted and desirable for architects, engineers and researchers in the electronic design community.

In the hardware design community, architects use hardware description languages[1] (that are not defined by means of mathematics) like Verilog [15] and

[1] In industry, SystemC [17] and SystemVerilog [18] are widely used for modeling hardware systems. Since they are languages for system-level designs and embedded system

VHDL [16] to model hardware systems. Since these description languages are not formally (i.e. mathematically) defined, the models described using them may be ambiguous. It is worth mentioning that, after these description languages have been widely used, some research works on their formal semantics have been done by other researchers. Hence, we believe that there is a gap between the intuition behind the description languages (given by the developers) and the formal semantics of the description languages defined by the researchers. Currently, the analysis of hardware systems (e.g. modeled in Verilog and VHDL) is mainly addressed by a simulation context. Simulation engineers apply the simulators, that are built based on their semantics (that is not formally defined), to simulate the behavior of such systems. The results are then not always guaranteed to be correct.

The ability of unambiguously specifying (in a mathematical sense) and rigorously analyzing timing properties/constraints is fundamental to design correct hardware systems. A formalism in which hardware behavior and timing properties can be precisely captured is a mandatory prerequisite for designing correct hardware systems. The timed process algebra timed Chi (χ) [7] is such a formalism.

The timed Chi formalism is obtained by means of the simplification of hybrid Chi formalism [4, 13]. Principally, the timed Chi formalism is suited to modeling, simulation, verification and real-time control. Its application domain consists of large and complex manufacturing systems.

The formal semantics of timed Chi is defined by means of deduction rules in a *Structured Operational Semantics* (SOS) style [14] that associates a time transition system with a timed Chi process. A set of axioms/properties of timed Chi is presented for a notion of equivalence (bisimulation). The straightforward syntax and semantics is also highly suited to architects, engineers and researchers from the hardware design community.

Since timed Chi is a well-developed algebraic theory from the field of process algebras with timing, we have the idea that timed Chi is also well-suited for addressing various aspects of hardware systems (discrete-time systems by nature and they are always modeled as finite-state machines)[2]. To show that timed Chi is useful for formal specification and analysis of hardware systems and our idea is correct, in this paper we illustrate the use of timed Chi with some benchmark examples of hardware systems: a multiplexer (MUX), a D flip-flop, an asynchronous arbiter and a simple arbiter (with assertion). Particularly, in this paper we emphasize the formal analysis of timed Chi specifications by means of *mathematical proofs*.

designs (by nature), they are not further discussed. In this paper, we mainly intend to show that timed Chi is useful for addressing various aspects of hardware and compare it with other formalisms/specification languages that are used for model-ling and analyzing hardware.

[2] In this paper, our main interest is to show that timed Chi is useful for addressing various aspects of hardware with timing. As we will see in Subsection 4.3 and Section 7, timed Chi is also well-suited for formal specification and analysis of untimed hardware.

In the rest of this paper, we may usually refer to timed Chi as χ. Also, we may write formal χ specification as χ specification.

1.1 Related Work

1.1.1 Hardware Description Languages

Over the last ten years or so, research works in formal semantics in the electronic design community that have targeted to obtain some applicable opportunity mainly focused on various industrial hardware description languages. Quite often, their definitions were based on *Abstract State Machine* (ASM) specifications, *Denotational Semantics* and rewrite rules [19, 20, 21, 22, 23].

It is generally believed that a SOS provides more intuitive descriptions and that ASM specifications and denotational semantics appear to be less suited to describe the dynamic behavior of processes [34]. Since processes are the basic units of execution within industrial hardware description languages that are used to simulate the behavior of a device or a system, process algebras with a SOS style semantics are more immediate choices for giving formal specifications of hardware systems in the electronic design community.

In the recent years, various formal approaches (based on ASM specifications, deduction rules and denotational semantics) have already been studied and investigated for Verilog and VHDL that can only be considered as theoretical frameworks [19, 20, 21, 22, 23], because they are not directly executable.

In contrast to such formal approaches, specifications described in χ are completely executable (as in many process algebraic specifications). More precisely, the behavior of a specification described in χ can be illustrated by means of transition traces according to χ deduction rules together with the Timed Transition System (TTS) associating to a χ process. Similarly, formal analysis of χ specifications can be performed using χ deduction rules together with the TTS associating to the χ processes (see Subsection 5.4 for details).

1.1.2 Formalisms

On the other hand, the χ formalism is a timed process algebra, and is thus related to the other formalisms with timing[3], for example, Timed CSP [29], TCOZ [30], ATP [3], process algebra with timing from [2] and Timed Action Systems [31]. Some comparisons and related works of the above-mentioned formalisms with timing can already be found in [31] and [32].

Here we discuss the most important concepts and key features of χ in more generality. Also, these concepts and features make χ suitable for formal specification and analysis of various complex systems including hardware systems. The most important concepts and key features of χ are summarized below:

[3] Since there exist many different formalisms with timing, it is not our intention to list them all in this paper.

Ease of modeling. χ is principally designed with a well-defined semantics for modeling purpose. It is especially suited to the specification and analysis of complex systems. This is achieved by means of:

1. the process terms for scoping that integrate abstraction, local variables, local channels and local recursion definitions;
2. the process definition, process instantiation and syntactic extensions that enable process re-use, encapsulation, hierarchical and/or modular composition of processes;
3. the different interaction mechanisms, namely handshake synchronization and synchronous communication that are mainly intended for processes that do not share variables, and shared variables that are mainly intended for interaction between processes in a parallel context.

Ability to use algebraic reasoning. The use of algebraic reasoning for χ specifications will be explained and shown in Subsections 3.3 and 5.4.

In contract to the above-mentioned formalisms, χ has a simulator and has a rich set of back-end verification tool supported (see Section 6 for details).

1.1.3 Formal Verification Techniques and Tools

Over the years, model checking, SAT-based verification and theorem proving techniques and tools [39, 40, 41] have been successfully and widely used for the formal verification of hardware systems in industry.

χ is not only a simulation language with a semantics formally defined, but can also be purportedly used for formal verification. In principal, χ is a formalism that can be used for specifying concurrent/distributed systems, finite-state systems and real-time systems (as in major hardware description languages) and χ can serve as a single-formalism-multi-solution. This means that we can translate a χ specification to the input languages of several verification tools (e.g. CADP [35], SPIN [36] and UPPAAL [37]) and it can be verified in those verification tool environments. For instance, safety properties of concurrent systems specified in χ can be verified by translating those systems to PROMELA [36], which is the input language of the SPIN Model Checker.

It is worth mentioning that several translations (between χ and some formalisms) have been already automated and the correctness of such translations have been carefully studied at the semantical level (see Section 6 for details). It is not hard to see that χ can also be translated to the specification/input languages of several theorem provers (e.g. ACL2 [41]). After having defined such translations, different theorem provers can also be used as verification engines for χ specifications. Note that a set of axioms/properties for χ was presented and the notion of stateless bisimilarity was proved to be a congruence [1] with respect to all χ operators in [7]. As a consequence, algebraic reasoning is facilitated, since it is allowed to replace equals by equals in any context (i.e. χ is compositional operationally). To the best of our understanding, such a congruence property has not been specifically studied/proved for the specification/input languages used for the major theorem provers.

Furthermore, an introduction paper of our research work in this direction can be found in [44][4].

1.2 Paper Organization

This paper is organized as follows. In Section 2, we give a brief overview of the χ formalism. Through some simple examples, Section 3 shows that the deduction rules of χ can ensure the correctness of specifications and can help modelers make correct specifications. Some samples (modeling several benchmark hardware systems) of the applications of χ are shown in Section 4. A more detailed account of the formal syntax and formal semantics of χ is provided in Section 5. In the same section, a complete mathematical proof for the analysis of the assertion of a simple arbiter is given. A variety of approaches that can be used for the analysis of the formal specifications described in χ is presented in Section 6. Finally, concluding remarks are made in Section 7.

2 χ FORMALISM

χ is such a rich formalism and presenting the complete formal syntax and formal semantics of it is far beyond the scope of this paper. Hence, in this section, we informally present just a small part of χ, disregarding features[5] that may not be relevant for the use in this paper.

Nevertheless, Subsection 5.2 gives a more detailed account of the formal semantics of χ. Again, in what follows, we refer to this small part of timed χ as χ. For an extensive treatment of χ, the reader is referred to [7].

2.1 Data Types, Time Model, Synchronization and Communication Model

Data types: χ is statically strongly typed. Every variable has a type which defines the allowed values of that variable and the allowed operations on that variable. The basic types are natural numbers, integers, real numbers, Booleans, strings and enumerations. Type constructors operate on existing types to create structured types. χ uses type constructors to create sets, lists, array tuples, record

[4] This is a very short paper. It only presented the syntax and semantics of χ in a very informal way. Also, no deduction rules of χ and their usefulness were discussed and no analysis example was given in such a paper.

[5] For instance, recursive definitions, operators used for scoping and communication, process definition and process instantiation are not treated in this paper.

tuples, dictionaries, functions and distributions (for stochastic models). Channels also have a type that indicates the type of data that is communicated via the channel. Pure synchronization channels, that do not communicate data, are of the predefined type *void*.

Time model: The time in χ is dense. So, timing is measured on a continuous time scale. χ has a strong time determinism principle. This means that passage of time cannot result in making a choice between the two operands of the choice. Also, the maximal progress (a process can delay only if it cannot do anything else) is not implicit in χ.

Synchronization and communication model: In χ, the synchronization and communication mechanism are based on CSP [33]. This means that, although a channel can be used in any number of processes, synchronization or communication always occurs on a point-to-point basis, i.e. synchronization or communication always occurs between exactly two processes. Also, this synchronization or communication mechanism is widely used in the hardware design community for modeling asynchronous circuits.

2.2 χ Specification

A χ specification (restricted to the use in this paper) is of the following form:

$$\begin{array}{l} \langle\ \mathsf{disc}\ s_1, \ldots, s_k \\ ,\ \mathsf{chan}\ h_1, \ldots, h_l \\ ,\ i \\ \mid p \\ \rangle \end{array}$$

where

- $s_1, \ldots, s_k$ denote the discrete variables;
- $h_1, \ldots, h_l$ denote the channels;
- i denotes an initialization predicate that restricts the allowed values of the variables initially;
- p is a process term defining the behavior of the specification.

Notice that the keywords disc and chan are omitted where there are no discrete variable declarations and are no channel declarations, respectively. Also, the initialization predicate may be omitted, indicating a predicate that always holds.

The set P of process terms $p \in P$ (for the use in this paper) is defined according to the following grammar:

$$\begin{array}{rl} p ::= & \mathrm{skip} \mid \mathbf{x}_n := \mathbf{e}_n \mid \delta \mid \bot \mid \Delta d \mid [p] \mid u \curvearrowright p \mid p;\, p \\ \mid & b \rightarrow p \mid p \mathbin{[\!]} p \mid p \parallel p \mid h \,!!\, \mathbf{e}_n \mid h \,??\, \mathbf{x}_n \mid \partial_A(p) \mid *p \\ \mid & \llbracket\ \mathsf{disc}\ \mathsf{s}_k, \mathsf{chan}\ \mathbf{h}_m, i \mid p\ \rrbracket \end{array}$$

Here, $\mathbf{x}_n$ and $\mathbf{e}_n$ are a list of variables $x_1, \ldots, x_n$ (i.e. $\mathbf{x}_n$) and a list of expressions $e_1, \ldots, e_n$ (i.e. $\mathbf{e}_n$), respectively. $d \in \mathbb{R}_{\geq 0}$, b denotes a guard (i.e. a Boolean expression) and u represents a predicate. h is a channel and A represents a set of actions. Moreover, $\mathbf{s}_k$, $\mathbf{h}_m$ and i are local discrete variables, local channels and an initialization predicate predicate, respectively.

In χ, it is allowed to use common arithmetic operators (e.g. $+, -$), relational operators (e.g. $=, \geq$) and logical operators (e.g. $\wedge, \vee$) as in mathematics to construct expressions over variables.

The operators are listed in descending order of their binding strength as follows: $\{ \rightarrow , \curvearrowright \}$, ; , $\{ \|, [\!] \}$. The operators inside the braces have equal binding strength. In addition, operators of equal binding strength associate to the right, and parentheses may be used to group expressions. For example, $p;\ q;\ r$ means $p;\ (q;\ r)$, where $p, q, r \in P$.

2.3 Concise Explanation of the Syntax

Atomic Process Terms

The atomic process terms of χ are undelayable process term constructors that cannot be split into smaller process terms. They are:

1. The *skip* process term skip . It can only perform an internal action τ to termination.
2. The *multi-assignment* process term $\mathbf{x}_n := \mathbf{e}_n$. It assigns the values of expressions $e_1, \ldots, e_n$ to variables $x_1, \ldots, x_n$, respectively, in an atomic way.
3. The *deadlock* process term δ. It cannot perform any actions or delays, but it is consistent (in the state).
4. The *inconsistent* process term $\perp$. It is inconsistent in all states.
5. The *send* process term $h\,!!\,\mathbf{e}_n$. It sends the values (must be defined) of expressions $e_1, \ldots, e_n$ via channel h by means of internal send actions.
6. The *receive* process term $h\,??\,\mathbf{x}_n$. It receives values (of size n) via the channel h and assigns them to the variables $x_1, \ldots, x_n$ by means of internal receive actions.

Operators

Atomic process terms can be combined using the following operators. The operators are:

1. The *delay operator* Δd denotes a process term that first delays for d time units, and then terminates by means of the internal action τ.
2. The *signal emission operator* $u \curvearrowright p$, where u denotes a predicate, behaves as p for those states where u holds. The process term is inconsistent in the state for which u does not hold.

3. The *sequential composition* of process terms p and q (i.e. $p;\ q$) behaves as process term p until p terminates, and then continues to behave as process term $q \in P$.
4. By means of the *any delay operator* $[p]$, delay behavior of arbitrary duration can be specified. The resulting behavior is such that arbitrary delays are allowed. As a consequence, any delay behavior of p is neglected. The action behavior of p remains unchanged.
5. The *guarded process term* $b \rightarrow p$ can perform whatever actions p can perform under the condition that the guard b (a Boolean expression) evaluates to true in the current state. The guarded process term can delay according to p under the condition that the guard b holds. The guarded process term can perform arbitrary delays under the condition that the guard b does not hold.
6. The *alternative composition* of process terms p and q (i.e. $p \parallel q$) allows a non-deterministic choice between different actions of the process term either p or q. With respect to time behavior, the participants in the alternative composition have to synchronize.
7. The *parallel composition* of process terms p and q (i.e. $p \| q$) executes p and q concurrently in an interleaved fashion with the possibility of synchronization or communication (in a CSP based) between p and q. Also, with respect to time behavior, the participants in the parallel composition have to synchronize.
8. The *encapsulation operator* $\partial_A(p)$ is introduced to block the actions that p can perform from the set A.
9. The *repetition* process term $*p$ represents the infinite repetition of process term p.
10. The *modeling scope operator* process term (used for hierarchical modeling) $[\![\, \mathsf{disc}\ \mathbf{s}_k,\ \mathsf{chan}\ \mathbf{h}_m,\ i \mid p\,]\!]$ is used to declare a scope consisting of local discrete variables $\mathbf{s}_k$ (as an abbreviation of $s_1, \ldots, s_k$), local channels $\mathbf{h}_m$ (as an abbreviation of $h_1, \ldots, h_m$) and initialization predicate i.

2.4 Formal Semantics of χ

This subsection informally describes the formal semantics of χ. It is defined by means of deduction rules in SOS style that associate a time transition system with a χ process. Three different kinds of transition relations are defined, namely:

1. one associated with termination transition;
2. one associated with action transition (for discrete action);
3. one associated with time transition (delay behavior).

3 CORRECTNESS OF χ SPECIFICATIONS

As we already mentioned in Section 2 the formal semantics of χ is defined by means of deduction rules in SOS style. These deduction rules ensure the correctness of

χ specifications and can help modelers make correct specifications. In this section, for illustration purposes, we define some deduction rules (only for the use in this section to give the first impression of such deduction rules to the reader) and show their use through some toy examples in χ.

For those who have computer science background, this section can be left out.

3.1 Deduction Rules

Here, we define several deduction rules for atomic process term: multi-assignment; and operators: sequential composition, alternative composition and parallel composition by leveraging "very high" abstraction to the original deduction rules for such atomic process term and operators given in [7].

For the set P of process terms $p \in P$ (for the use in this subsection), we have:

$$p \ ::= \quad \mathbf{x}_n := \mathbf{e}_n \quad | \quad p;\, p \quad | \quad p \,[\!]\, p \quad | \quad p \parallel p.$$

We further define the following deduction rules:

$$\frac{}{\mathbf{x}_n := \mathbf{e}_n \to \checkmark}\,1 \qquad \frac{p \to \checkmark}{p;\, q \to q}\,2 \qquad \frac{p \to \checkmark}{p \,[\!]\, q \to \checkmark}\,3$$

$$\frac{p \to \checkmark}{q \,[\!]\, p \to \checkmark}\,4 \qquad \frac{p \to \checkmark}{p \parallel q \to q}\,5 \qquad \frac{p \to \checkmark}{q \parallel p \to q}\,6.$$

The above deduction rules (of the form $\frac{premise}{conclusion}$) have two parts: on the top of the bar we put *premise* of the rule, and below it the *conclusion*. If the premise holds, then we infer that the conclusion holds as well. Moreover, $\to$ and $\checkmark$ are used to represent a transition and a terminated process, respectively.

Rule 1 states that $\mathbf{x}_n := \mathbf{e}_n$ can always perform a transition to a terminated process (i.e. successful termination). The sequential composition of the process terms p and q (i.e. $p;\, q$) behaves as process term p until p terminates, and then continues to behave as process term q (see Rule 2). The effect of applying the alternative operator to the process terms p and q (i.e. $p \,[\!]\, q$) is that the execution of a transition by either one of them results in a definite choice as shown in Rules 3 and 4. The parallel composition of the process terms p and q (i.e. $p \parallel q$) has as its behavior with respect to transitions the interleaving of the behaviors of p and q (see Rules 5 and 6).

3.2 Running Examples

Using the above deduction rules, for instance, we can prove that:

1. Process term $\mathbf{x}_n := \mathbf{e}_n;\, (\mathbf{x}'_n := \mathbf{e}'_n \,[\!]\, \mathbf{x}''_n := \mathbf{e}''_n)$ can terminate successfully after a finite number of transitions.

- ***Proof:*** According to Rule 1, $\mathbf{x}_n := \mathbf{e}_n$ can always perform a transition to a terminated process. Due to this, we can apply Rule 2 to obtain $\mathbf{x}_n := \mathbf{e}_n;\ (\mathbf{x}'_n := \mathbf{e}'_n \mathbin{[\!]} \mathbf{x}''_n := \mathbf{e}''_n) \rightarrow \mathbf{x}'_n := \mathbf{e}'_n \mathbin{[\!]} \mathbf{x}''_n := \mathbf{e}''_n$. Using Rule 1 together with either Rule 3 or Rule 4, we can further have $\mathbf{x}'_n := \mathbf{e}'_n \mathbin{[\!]} \mathbf{x}''_n := \mathbf{e}''_n \rightarrow \checkmark$.

2. Process term $(\mathbf{x}_n := \mathbf{e}_n \parallel \mathbf{x}'_n := \mathbf{e}'_n);\ \mathbf{x}''_n := \mathbf{e}''_n$ cannot terminate successfully in two transitions.

   - ***Proof:*** We assume to have $(\mathbf{x}_n := \mathbf{e}_n \parallel \mathbf{x}'_n := \mathbf{e}'_n);\ \mathbf{x}''_n := \mathbf{e}''_n \rightarrow \mathbf{x}''_n := \mathbf{e}''_n \rightarrow \checkmark$ according to Rules 1 and 2. This means that we must have the transition $\mathbf{x}_n := \mathbf{e}_n \parallel \mathbf{x}'_n := \mathbf{e}'_n \rightarrow \checkmark$ as a premise necessarily. However, this is not possible due to Rules 5 and 6.

3.3 Properties

We can also deduce some properties (that add to the level of confidence one has with respect to the correctness of the formal semantics) for all specifications that can be generated by the set P of process terms according to the deduction rules as defined in Subsection 3.1. For instance, we can have the following properties for equivalence:

- $p \mathbin{[\!]} q = q \mathbin{[\!]} p$ and $p \parallel q = q \parallel p$ (so-called commutativity property);
- $p@(q@r) = (p@q)@r$, for $@ \in \{;\ , [\!], \|\}$ (so-called associativity property).

Using properties, we can rewrite a χ specification to a simpler form. This can be advantageous for many forms of analysis. As we already mentioned, in the field of process algebras, this is so-called algebraic reasoning.

4 FORMAL χ SPECIFICATION OF HARDWARE SYSTEMS

This section presents some samples (modeling several benchmark hardware systems) of the applications of χ.

4.1 MUX

In hardware systems, a multiplexer (MUX) is a device that encodes information from two or more data input into a single output (i.e. multiplexers function as multiple-input and single-output switches).

A multiplexer shown in Figure 1 has two inputs (a and b) and a selector (sel) that connects a specific input to the single output (y). Below is the χ specification of such a MUX:

```
⟨ disc a, b, sel, y
, a = true, b = false, sel = true, y = true
| ∗(MUX ∥ SEL ∥ A ∥ B)
⟩,
```

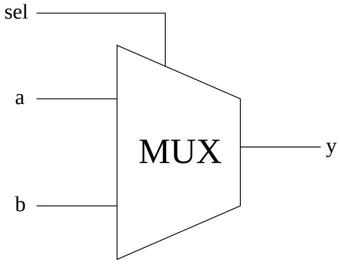

Fig. 1. A MUX

$$\mathrm{MUX} \approx \mathrm{sel} \to \mathrm{y} := \mathrm{a} \;[\!]\; \neg\mathrm{sel} \to \mathrm{y} := \mathrm{b},$$

$$\mathrm{SEL} \approx \Delta 3;\; (\neg\mathrm{sel}^{-} \to \mathrm{sel} := \mathrm{true} \;[\!]\; \mathrm{sel}^{-} \to \mathrm{sel} := \mathrm{false}),$$

$$\mathrm{A} \approx \Delta 6;\; (\neg\mathrm{a}^{-} \to \mathrm{a} := \mathrm{true} \;[\!]\; \mathrm{a}^{-} \to \mathrm{a} := \mathrm{false}),$$

$$\mathrm{B} \approx \Delta 9;\; (\neg\mathrm{b}^{-} \to \mathrm{b} := \mathrm{true} \;[\!]\; \mathrm{b}^{-} \to \mathrm{b} := \mathrm{false}).$$

The complete system is modeled by a repetition of the parallel composition of process terms MUX, SEL, A and B. The process term SEL assigns the selected input (either a or b) to the output y. The process terms SEL, A and B model the behavior of the variables sel, a and b with the frequency of 3, 6 and 9 time units (swapping the values between "true and false") respectively.

Note that all variables used in the χ specification are the type of Boolean variables and x^{-} denotes the value of variable x before execution of an discrete action.

4.2 A D Flip-Flop

D flip-flops are among the basic building blocks of *Register-Transfer Level* (RTL) designs. A D flip-flop has a clock input (clk) in the sensitivity list, a data input (d) and a data output (Q).

When a positive or negative edge occurs in the clock signal (which means that $(\mathrm{clk} \wedge \neg\mathrm{clk}^{-}) \vee (\mathrm{clk}^{-} \wedge \neg\mathrm{clk})$), the value of input port d is assigned to output port Q.

Figure 2 depicts such a D flip-flop.

A χ specification is given as follows:

$$\begin{array}{l} \langle\; \mathsf{disc}\; \mathrm{d}, \mathrm{Q}, \mathrm{clk} \\ ,\; \mathrm{d} = \mathrm{true}, \mathrm{Q} = \mathrm{true}, \mathrm{clk} = \mathrm{false} \\ \mid\; *(\mathrm{DFF} \parallel (\mathrm{CLK_a} \;[\!]\; \mathrm{CLK_7})) \\ \rangle, \end{array}$$

$$\mathrm{DFF} \approx (\mathrm{clk} \wedge \neg\mathrm{clk}^{-}) \vee (\mathrm{clk}^{-} \wedge \neg\mathrm{clk}) \to \mathrm{Q} := \mathrm{d},$$

$$\mathrm{CLK_a} \approx [\mathrm{SWITCH};\; \mathrm{INPUT}],$$

$$\mathrm{CLK_7} \approx \Delta 7;\; \mathrm{SWITCH};\; \mathrm{INPUT},$$

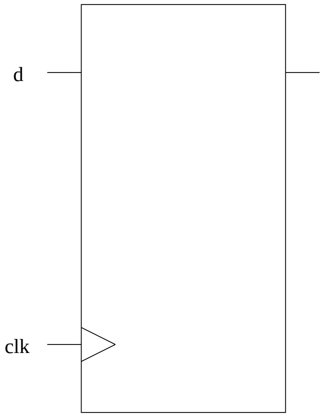

Fig. 2. A D flip-flop

$$\mathrm{SWITCH} \approx \neg \mathrm{clk}^{-} \to \mathrm{clk} := \mathrm{true} \,[\!]\, \mathrm{clk}^{-} \to \mathrm{clk} := \mathrm{false},$$

$$\mathrm{INPUT} \approx \mathrm{d} := \mathrm{true} \,[\!]\, \mathrm{d} := \mathrm{false}.$$

In the χ specification, clk, d and Q are modeled by Boolean variables. The complete system is modeled by a repetition of the parallel composition of process terms DFF and a choice between process terms $\mathrm{CLK_a}$ and $\mathrm{CLK_7}$. The process term DFF describes the behavior of the D flip-flop. When a positive or negative edge occurs in clk, the value of d is assigned to Q. Otherwise, it performs an arbitrary delay.

The process terms $\mathrm{CLK_a}$ and $\mathrm{CLK_7}$ model the behavior of the clock clk with the frequency of arbitrary and 7 time units respectively. The switching from positive edge to negative edge or vice verse is modeled by the process term SWITCH. A non-deterministic choice of the clock frequency (any value) for at most 7 time units is assigned to clk by means of the alternative composition of $\mathrm{CLK_a}$ and $\mathrm{CLK_7}$ (i.e. $\mathrm{CLK_a} \,[\!]\, \mathrm{CLK_7}$).

In the process term $\mathrm{CLK_a}$, the any delay operator [ ] is needed to apply to SWITCH; INPUT, because otherwise $\mathrm{CLK_a} \,[\!]\, \mathrm{CLK_7}$ may not delay together for at most 7 time units (as already explained in Section 2 that χ a has strong time determinism principle). For simulation purposes, a test-bench of d (assigning a value "true" or "false" to d arbitrarily) is given by the process term INPUT.

4.3 An Asynchronous Arbiter

Asynchronous arbiter circuits are standard hardware verification benchmark circuits. An arbiter circuit controls the exclusive access of one out of a number possibly competing processes to a shared resource.

Figure 3 shows an (untimed) asynchronous arbiter (taken from [24]) such that two clients (client-1 and client-2) complete for a shared resource. Each client sends a request (a number 1 for client-1 and a number 2 for client-2) for the resource to the arbiter via an individual channel (a and b).

The arbiter chooses non-deterministically between clients with pending requests, and then sends the number of the selected client-(1 or 2) via another channel (c) to the environment.

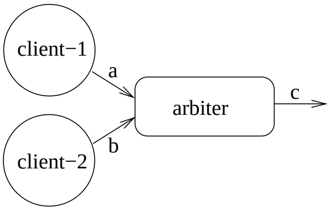

Fig. 3. An asynchronous arbiter

A χ specification is given as follows:

$\langle$ disc x
, chan a, b, c
, x = 0
| $*(\partial_A(\mathrm{CLI}_1 \parallel \mathrm{CLI}_2 \parallel \mathrm{ARB}))$
$\rangle$,

$$\mathrm{CLI}_1 \approx \mathrm{a}\,!!\,1, \quad \mathrm{CLI}_2 \approx \mathrm{b}\,!!\,2,$$

$$\mathrm{ARB} \approx (\mathrm{a}\,??\,\mathrm{x} \;[\!]\; \mathrm{b}\,??\,\mathrm{x});\; \mathrm{c}\,!!\,\mathrm{x}.$$

The process terms CLI_1, CLI_2 and ARB model the behavior of the client-1, client-2 and arbiter, respectively, as described above. The encapsulation operator is applied to $\mathrm{CLI}_1 \parallel \mathrm{CLI}_2 \parallel \mathrm{ARB}$ to block some undesired internal send and receive actions (specified in the set A) via channels a and b. This means that only successful communication actions via channels a and b are allowed.

4.4 Remarks

The examples given in this section are used to show the applicability of χ for hardware system modeling and develop an insight into the essential (χ) features needed for specifying hardware systems. For simplicity of reasoning, concrete delays are used in most of the examples. Nevertheless, arbitrary delay and delay of any duration can be easily modeled in χ using the any delay operator ([ ]) and/or by means of the predefined reserved global variable `time` (see Section 5 for details). For instance, according to the semantics of χ, the process term [skip] can perform an arbitrary delay and even a delay forever. However, as soon as the process term skip performs the internal action τ (but it does not have to) to termination, the arbitrary delay behavior of [skip] is neglected.

In order to illustrate our work clearly, only simple hardware systems in χ are given in this section. Nevertheless, the use of χ is generally applicable to all sizes and levels of hardware systems. It is important to note that over the years, timed Chi, hybrid Chi and their predecessors are being/have been widely used in industry for modeling and analyzing complex systems (for example, we refer to [9, 10, 11, 12]).

5 FORMAL ANALYSIS OF A χ SPECIFICATION

As we have shown in Section 4 χ can be used to formally represent hardware system designs. Therefore, for illustration purposes, we formally analyze (an assertion of) a simple arbiter described in χ.

5.1 A simple arbiter

In general, the role of an arbiter is to grant access to the shared resource by raising the corresponding *grant* signal and keeping it that way until the *request* signal is removed. A test for such an arbiter can be generated by an assertion as follows:

$$\text{“}\mathit{assertion} : \mathit{grant} \wedge \mathit{request}\text{”}.$$

If the assertion holds, this means that the arbiter work as expected. Using common (temporal) logics (e.g. CTL [39] and LTL [36]), such an assertion can be expressed (and considered) as a liveness property of the arbiter. As mentioned already in Section 1, we aim to analyze the arbiter by means of a complete mathematical proof in this paper. Nevertheless, Section 6 includes a few guidelines for verifying properties of χ specifications using temporal logics.

5.2 Analysis Approach

We can formally analyze (the assertion of) the arbiter described above by means of a complete mathematical proof via transition traces according to deduction rules defined for χ. In order to show such a formal proof, we need to provide deduction rules for some χ atomic process terms and operators in this subsection.

It is not our intention (also due to the reason of space) to give all χ deduction rules defined in [7]. In this subsection, we define a minimum set of deduction rules for atomic process terms and operators (in a restricted setting that is needed for the formal proof in the following subsection) by leveraging “*very high*” abstraction and simplification to the original deduction rules for such atomic process terms and operators defined in [7]. Also, this restricted setting of deduction rules does not support any synchronization/communication mechanism. So, the set of channels is not assumed and is not used in this restricted setting of deduction rules.

Data Types

In order to define the deduction rules, we need to make some assumptions about the data types:

1. Let Var denote the set of all variables $(x_0, \ldots, x_n, \texttt{time})$. Besides the variables $x_0, \ldots, x_n$, the existence of the predefined reserved global variable `time` which denotes the current time, the value of which is initially zero, is assumed. This variable cannot be declared.
2. Let Value denote the set of all possible values $(v_0, \ldots, v_m, \bot)$ that contains at least all Reals, all Booleans and $\bot$, where $\bot$ denotes the "*undefinedness*".
3. We then define a *valuation* as a partial function from variables to values. Syntactically, a valuation is denoted by a set of pairs $\{x_0 \mapsto v_0, \ldots, x_n, \mapsto v_n, \texttt{time} \mapsto t\}$, where $x_i \in$ Var represents a variable and $v_i \in$ Value its associating value; and $t \in \mathbb{R}_{\geq 0}$.
4. Further to this, the set of all valuations is denoted by Σ.

Convention

To allow more intuitive specifications, a more user-friendly syntax for a χ specification was given in Subsection 2.2. Such a χ specification is just an abbreviation for the set of χ processes described using formal χ syntax (see [7] for details). For the formal proof in the following subsection, we also need to present the formal χ syntax.

For simplicity, formal χ syntax presented here is also in a sort of restricted setting, which is limited to the use and the need of the formal proof in the following subsection. For the complete formal χ syntax, we refer to [7].

Formal Syntax

A χ process is a pair $\langle P, \Sigma \rangle$ and we use the convention $\langle p, \sigma \rangle$ to write a χ process, where $p \in P$ (as defined in Subsection 2.2) and $\sigma \in \Sigma$.

Formal Semantics

We give a formal semantics to the formal syntax defined above for χ, by constructing a kind of time transition system (TTS), for each process term and each possible valuation of variables.

Definition 1. The set of actions A_τ contains at least $aa(\mathbf{x}_n, \mathbf{v}_n)$ and τ, where $aa(\mathbf{x}_n, \mathbf{v}_n)$ is the assignment action (i.e. the values of $v_1, \ldots, v_n \in$ Value are correspondingly assigned to variables $x_1, \ldots, x_n \in$ Var in an atomic way) and τ is the internal action. The set A_τ is considered as a parameter of χ that can be freely instantiated.

Definition 2. We give a formal semantics for χ processes in terms of a time transition system (TTS), and define the following transition relations on processes of χ:

- $\_ \overset{\_}{\to} \langle \checkmark, \_ \rangle \subseteq (P \times \Sigma) \times A_\tau \times \Sigma$, denotes termination, where $\checkmark$ is used to indicate a successful termination, and $\checkmark$ is not a process term;
- $\_ \overset{\_}{\to} \_ \subseteq (P \times \Sigma) \times A_\tau \times (P \times \Sigma)$, denotes action transition;
- $\_ \overset{\_}{\longmapsto} \_ \subseteq (P \times \Sigma) \times \mathbb{R}_{\geq 0} \times (P \times \Sigma)$, denotes time transition (so-called delay).

For $p, p' \in P$; $\sigma, \sigma' \in \Sigma$, $a \in A_\tau$ and $d \in \mathbb{R}_{>0}$, the three kinds of transition relations can be explained as follows:

1. Firstly, a termination $\langle p, \sigma \rangle \overset{a}{\to} \langle \checkmark, \sigma' \rangle$ is that the process executes the action a followed by termination.
2. Secondly, an action transition $\langle p, \sigma \rangle \overset{a}{\to} \langle p', \sigma' \rangle$ is that the process $\langle p, \sigma \rangle$ executes the action a starting with the current valuation σ and by this execution p evolves into p', where σ' represents the accompanying valuation of the process after the action a is executed.
3. Thirdly, a time transition $\langle p, \sigma \rangle \overset{d}{\longmapsto} \langle p', \sigma' \rangle$ is that the process $\langle p, \sigma \rangle$ may idle during a d time units and then behaves like $\langle p', \sigma' \rangle$.

Deduction Rules

We define several deduction rules (for some atomic process terms and operators) that are specifically used and needed in the proof of the following subsection.

In the deduction rules below, we assume that the value of variables occurring in a process term are not allowed to change in action transitions, unless their changes are explicitly specified, for example by means of assigning a new value to such a variable.

Multi-assignment

$$\frac{}{\langle \mathbf{x}_n := \mathbf{e}_n, \sigma \rangle \xrightarrow{aa(\mathbf{x}_n, \bar{\sigma}(\mathbf{e}_n))} \langle \checkmark, \sigma[\bar{\sigma}(\mathbf{e}_n)/\mathbf{x}_n] \rangle}\ 7$$

By means of a multi-assignment (see Rule 7), the values of $e_1, \ldots, e_n$ are assigned to $x_1 \ldots, x_n$. Notice that $\sigma[\bar{\sigma}(\mathbf{e}_n)/\mathbf{x}_n]$ denotes the update of valuation σ such that the new values $\bar{\sigma}(e_1), \ldots, \bar{\sigma}(e_n)$ are assigned to $x_1, \ldots, x_n$, respectively, in an atomic way.

Delay

$$\frac{n = 0}{\langle \Delta n, \sigma \rangle \overset{\tau}{\to} \langle \checkmark, \sigma \rangle}\ 8 \qquad \frac{d \leq n}{\langle \Delta n, \sigma \rangle \overset{d}{\longmapsto} \langle \Delta(n - d), \sigma \rangle}\ 9$$

Rule 8 is used to model a delay of 0 duration by means of performing the internal τ action. Rule 9 states that a delay process term can perform a delay which is smaller than or equal to the value of the argument of the delay process term, and has no effect on the valuation.

Sequential composition

$$\frac{\langle p,\sigma\rangle \xrightarrow{a} \langle \checkmark,\sigma'\rangle}{\langle p;\ q,\sigma\rangle \xrightarrow{a} \langle q,\sigma'\rangle}\ 10 \qquad \frac{\langle p,\sigma\rangle \xrightarrow{a} \langle p',\sigma'\rangle}{\langle p;\ q,\sigma\rangle \xrightarrow{a} \langle p';\ q,\sigma'\rangle}\ 11$$

$$\frac{\langle p,\sigma\rangle \stackrel{d}{\longmapsto} \langle p',\sigma'\rangle}{\langle p;\ q,\sigma\rangle \stackrel{d}{\longmapsto} \langle p';\ q,\sigma'\rangle}\ 12$$

The process term q is executed after (successful) termination of the process term p as defined by Rules 10, 11 and 12.

Guard

$$\frac{\langle p,\sigma\rangle \xrightarrow{a} \langle \checkmark,\sigma'\rangle, \sigma \models b}{\langle b \rightarrow p,\sigma\rangle \xrightarrow{a} \langle \checkmark,\sigma'\rangle}\ 13 \qquad \frac{\langle p,\sigma\rangle \xrightarrow{a} \langle p',\sigma'\rangle, \sigma \models b}{\langle b \rightarrow p,\sigma\rangle \xrightarrow{a} \langle p',\sigma'\rangle}\ 14$$

If b evaluates to *true* in σ (denoted by $\sigma \models b$), for termination and action transition, the guarded process term (i.e. $b \rightarrow p$) behaves as process term p (see from Rule 13 to Rule 14).

Parallel composition

$$\frac{\langle p,\sigma\rangle \xrightarrow{a} \langle \checkmark,\sigma'\rangle}{\langle p \parallel q,\sigma\rangle \xrightarrow{a} \langle q,\sigma'\rangle}\ 15 \qquad \frac{\langle q,\sigma\rangle \xrightarrow{a} \langle \checkmark,\sigma'\rangle}{\langle p \parallel q,\sigma\rangle \xrightarrow{a} \langle p,\sigma'\rangle}\ 16$$

$$\frac{\langle p,\sigma\rangle \xrightarrow{a} \langle p',\sigma'\rangle}{\langle p \parallel q,\sigma\rangle \xrightarrow{a} \langle p' \parallel q,\sigma'\rangle}\ 17 \qquad \frac{\langle q,\sigma\rangle \xrightarrow{a} \langle q',\sigma'\rangle}{\langle p \parallel q,\sigma\rangle \xrightarrow{a} \langle p \parallel q',\sigma'\rangle}\ 18$$

$$\frac{\langle p,\sigma\rangle \stackrel{d}{\longmapsto} \langle p',\sigma'\rangle,\ \langle q,\sigma\rangle \stackrel{d}{\longmapsto} \langle q',\sigma'\rangle}{\langle p \parallel q,\sigma\rangle \stackrel{d}{\longmapsto} \langle p' \parallel q',\sigma'\rangle}\ 19$$

The parallel composition of the process terms p and q (i.e. $p \parallel q$) has as its behavior with respect to action transitions the interleaving of the behaviors of process terms p and q (see Rules from 15 to 18).

If both process terms p and q can perform the same delay, then the parallel composition of process terms p and q (i.e. $p \parallel q$) can also perform that delay, as defined by Rule 19.

Repetition

$$\frac{\langle p, \sigma\rangle \xrightarrow{a} \langle \checkmark, \sigma'\rangle}{\langle *p, \sigma\rangle \xrightarrow{a} \langle *p, \sigma'\rangle}\ 20 \qquad \frac{\langle p, \sigma\rangle \xrightarrow{a} \langle p', \sigma'\rangle}{\langle *p, \sigma\rangle \xrightarrow{a} \langle p'; *p, \sigma'\rangle}\ 21$$

$$\frac{\langle p, \sigma\rangle \overset{d}{\longmapsto} \langle p', \sigma'\rangle}{\langle *p, \sigma\rangle \overset{d}{\longmapsto} \langle p'; *p, \sigma'\rangle}\ 22$$

Rule 20 states that a repetition process term $*p$ repeats itself if its argument (i.e. p) terminates. Rule 21 states that if the process term p can perform an action transition or time transition, then the repetition process term $*p$ can also perform that action transition or time transition followed by its repetition (i.e. $p'; *p$).

χ Properties

To apply algebraic reasoning to ease the formal proof in the following subsection, we list several χ properties that are used in such a formal proof. For $p, q, r \in P$, we have:

1. false $\rightarrow p =$ true, in case a process term is guarded by a false predicate, process term false $\rightarrow p$ can perform any time transition, hence equals a true predicate.
2. $(p;\ q);\ r = p;\ (q;\ r)$, the sequential composition is associative.
3. $p \parallel q = q \parallel p$, the parallel composition is commutative.
4. $(p \parallel q) \parallel r = p \parallel (q \parallel r)$, the parallel composition is associative.

5.3 Formal Specification of the Arbiter

Below is a χ specification (in formal χ syntax) of the simple arbiter as described in Subsection 5.1:

$\langle$ INIT; $*(\text{ARB} \parallel \text{CLK} \parallel \text{ASSER}), \sigma\ \rangle$, where

$$\begin{array}{cl}
\text{INIT} & \approx \mathit{clk}, \mathit{grant}, \mathit{request} := \text{false}, \text{false}, \text{false} \\
\text{ARB} & \approx \text{R}_1;\ \text{G};\ \text{R}_0 \\
\text{R}_1 & \approx \Delta 4;\ \mathit{request} := \text{true} \\
\text{G} & \approx \Delta 4;\ \mathit{grant} := \text{true} \\
\text{R}_0 & \approx \Delta 4;\ \mathit{request} := \text{false} \\
\text{CLK} & \approx \Delta 5;\ \mathit{clk_c}, \mathit{clk} := \mathit{clk}, \neg \mathit{clk} \\
\text{ASSER} & \approx \neg \mathit{clk_c}^- \wedge \mathit{clk}^- \wedge \mathit{grant}^- \wedge \\
 & \quad \mathit{request}^- \rightarrow t := \mathtt{time}
\end{array}$$

$$\sigma = \{\mathit{clk} = \mathit{clk_c} = \mathit{grant} = \mathit{request} = t \mapsto \bot, \mathtt{time} \mapsto 0\}.$$

The χ specification of the arbiter is a sequential composition of the process terms INIT and the repetition of the parallel composition of process terms ARB, CLK and ASSER:

- INIT – It assigns the initial values to variables *clk*, clk_c, *grant* and *request* (i.e. the initialization).
- ARB – It models the change of behavior of variables *clk*, clk_c, *grant* and *request* according to time.
- CLK – It models the behavior of a clock (i.e. *clk*) which swaps the values between "false" and "true" every 5 time units.
- ASSER – It expresses the assertion for the arbiter (as indicated above). Also, it models the fact that the test of the assertion is executed whenever there is a positive change in *clk*. When this happens, the current time is assigned to the variable t.

In the process term CLK, variable clk_c is introduced (as a copy of clk_c) to save the temporary value of *clk*, which is used to model event change on the variable *clk* (i.e. event controls).

5.4 Formal Proof Via Transition Traces

To increase the readability of the following formal proof:

- We often apply the commutativity property and associativity property of the parallel composition without explicitly referring to the deduction rules and such properties.
- Also, we do not specifically mention which assignment actions are used in the action transitions. We just mention them as some actions a.
- Several unimportant brackets are introduced to group expressions, which may help the reader follow the proof in a more intuitive way.
- We need to keep in mind that the process term $\neg clk_c^- \wedge clk^- \wedge grant^- \wedge request^- \rightarrow t := \texttt{time}$ (i.e. the process term ASSER) performs arbitrary delay whenever the guard $\neg clk_c^- \wedge clk^- \wedge grant^- \wedge request^-$ does not hold in the state (i.e. the application of the property: false $\rightarrow p =$ true). We do not mention this again in the formal proof.
- In the following proof, we only consider the maximum duration for a possible time transition and the transitions of intermediate time points for such a time transition are not shown. For example, we only show $\langle \Delta 5, \sigma\rangle \overset{5}{\longmapsto} \langle \Delta 0, \sigma\rangle$ and not $\langle \Delta 5, \sigma\rangle \overset{t_i}{\longmapsto}, \ldots, \overset{t_j}{\longmapsto} \langle \Delta 0, \sigma\rangle$ for some $t_i, t_j \in \mathbb{R}_{>0}$ such that $t_i + \ldots + t_j = 5$.

Formal Proof

1. We start with the process below:

$$\langle \text{INIT};\ *(\text{ARB} \parallel \text{CLK} \parallel \text{ASSER}), \sigma\rangle.$$

2. Applying Rules 7 and 10, we obtain:

   $\langle \mathrm{INIT};\ {*}(\mathrm{ARB} \parallel \mathrm{CLK} \parallel \mathrm{ASSER}), \sigma\rangle \xrightarrow{a} \langle {*}(\mathrm{ARB} \parallel \mathrm{CLK} \parallel \mathrm{ASSER}), \sigma_1\rangle$, where $\sigma_1 = \{\mathit{clk} = \mathit{grant} = \mathit{request} \mapsto \text{false}, \mathit{clk}_c \mapsto \bot, t \mapsto \bot, \mathtt{time} \mapsto 0\}$.

3. Due to Rules 22, 20, 8 and 17, the process has to perform a time transition of 4 time units and then to execute the internal action τ as follows:

   $\langle {*}(\mathrm{ARB} \parallel \mathrm{CLK} \parallel \mathrm{ASSER}), \sigma_1\rangle \overset{4}{\longmapsto} \xrightarrow{\tau} \langle (\mathit{request} := \text{true};\ \mathrm{G};\ \mathrm{R}_0 \parallel \Delta 1;\ \mathit{clk}_c, \mathit{clk} := \mathit{clk}, \neg \mathit{clk} \parallel \mathrm{ASSER});\ {*}(\mathrm{ARB} \parallel \mathrm{CLK} \parallel \mathrm{ASSER}), \sigma_2\rangle$, where $\sigma_2 = \{\mathit{clk} = \mathit{grant} = \mathit{request} \mapsto \text{false}, \mathit{clk}_c \mapsto \bot, t \mapsto \bot, \mathtt{time} \mapsto 4\}$.

4. Followed by Rules 22, 10 and 7, we have:

   $\langle (\mathit{request} := \text{true};\ \mathrm{G};\ \mathrm{R}_0 \parallel \Delta 1;\ \mathit{clk}_c, \mathit{clk} := \mathit{clk}, \neg \mathit{clk} \parallel \mathrm{ASSER});\ {*}(\mathrm{ARB} \parallel \mathrm{CLK} \parallel \mathrm{ASSER}), \sigma_2\rangle \xrightarrow{a} \langle (\mathrm{G};\ \mathrm{R}_0 \parallel \Delta 1;\ \mathit{clk}_c, \mathit{clk} := \mathit{clk}, \neg \mathit{clk} \parallel \mathrm{ASSER});\ {*}(\mathrm{ARB} \parallel \mathrm{CLK} \parallel \mathrm{ASSER}), \sigma_3\rangle$, where $\sigma_3 = \{\mathit{clk} = \mathit{grant} \mapsto \text{false}, \mathit{request} \mapsto \text{true}, \mathit{clk}_c \mapsto \bot, t \mapsto \bot, \mathtt{time} \mapsto 4\}$.

5. Using Rules 12 together with 19, 17, 10 and 8, we get:

   $\langle (\mathrm{G};\ \mathrm{R}_0 \parallel \Delta 1;\ \mathit{clk}_c, \mathit{clk} := \mathit{clk}, \neg \mathit{clk} \parallel \mathrm{ASSER});\ {*}(\mathrm{ARB} \parallel \mathrm{CLK} \parallel \mathrm{ASSER}), \sigma_3\rangle \overset{1}{\longmapsto} \xrightarrow{\tau} \langle (\Delta 3;\ \mathit{grant} := \text{true};\ \mathrm{R}_0 \parallel \mathit{clk}_c, \mathit{clk} := \mathit{clk}, \neg \mathit{clk} \parallel \mathrm{ASSER});\ {*}(\mathrm{ARB} \parallel \mathrm{CLK} \parallel \mathrm{ASSER}), \sigma_4\rangle$, where $\sigma_4 = \{\mathit{clk} = \mathit{grant} \mapsto \text{false}, \mathit{request} \mapsto \text{true}, \mathit{clk}_c \mapsto \bot, t \mapsto \bot, \mathtt{time} \mapsto 5\}$.

6. Similarly, applying Rules 11, 15 and 7, we obtain:

   $\langle (\Delta 3;\ \mathit{grant} := \text{true};\ \mathrm{R}_0 \parallel \mathit{clk}_c, \mathit{clk} := \mathit{clk}, \neg \mathit{clk} \parallel \mathrm{ASSER});\ {*}(\mathrm{ARB} \parallel \mathrm{CLK} \parallel \mathrm{ASSER}), \sigma_4\rangle \overset{a}{\longmapsto} \langle (\Delta 3;\ \mathit{grant} := \text{true};\ \mathrm{R}_0 \parallel \mathrm{ASSER});\ {*}(\mathrm{ARB} \parallel \mathrm{CLK} \parallel \mathrm{ASSER}), \sigma_5 \rangle$, where $\sigma_5 = \{\mathit{grant} \mapsto \text{false}, \mathit{clk} \mapsto \text{true}, \mathit{request} \mapsto \text{true}, \mathit{clk}_c \mapsto \text{false}, t \mapsto \bot, \mathtt{time} \mapsto 5\}$.

7. By Rules 12, 19, 17, 10 and 9, we achieve:

   $\langle (\Delta 3;\ \mathit{grant} := \text{true};\ \mathrm{R}_0 \parallel \mathrm{ASSER});\ {*}(\mathrm{ARB} \parallel \mathrm{CLK} \parallel \mathrm{ASSER}), \sigma_5\rangle \overset{3}{\longmapsto} \xrightarrow{\tau} \langle (\mathit{grant} := \text{true};\ \mathrm{R}_0 \parallel \mathrm{ASSER});\ {*}(\mathrm{ARB} \parallel \mathrm{CLK} \parallel \mathrm{ASSER}), \sigma_6\rangle$, where $\sigma_6 = \{\mathit{grant} \mapsto \text{false}, \mathit{clk} \mapsto \text{true}, \mathit{request} \mapsto \text{false}, \mathit{clk}_c \mapsto \text{false}, t \mapsto \bot, \mathtt{time} \mapsto 8\}$.

8. Again, using Rules 11, 17, 10 and 7, we obtain:

   $\langle (\mathit{grant} := \text{true};\ \mathrm{R}_0 \parallel \mathrm{ASSER})\ ;\ {*}(\mathrm{ARB} \parallel \mathrm{CLK} \parallel \mathrm{ASSER}), \sigma_6\rangle \xrightarrow{a} \langle (\mathrm{R}_0 \parallel \mathrm{ASSER});\ {*}(\mathrm{ARB} \parallel \mathrm{CLK} \parallel \mathrm{ASSER}), \sigma_7\rangle$, where $\sigma_7 = \{\mathit{grant} \mapsto \text{true}, \mathit{clk} \mapsto \text{true}, \mathit{request} \mapsto \text{false}, \mathit{clk}_c \mapsto \text{false}, t \mapsto \bot, \mathtt{time} \mapsto 8\}$.

9. At this stage, the guard $\neg \mathit{clk}_c^- \wedge \mathit{clk}^- \wedge \mathit{grant}^- \wedge \mathit{request}^-$ in the process term ASSER holds. So, by means of applying Rules 11, 16, 13 and 7, we have:

   $\langle (\mathrm{R}_0 \parallel \mathrm{ASSER});\ {*}(\mathrm{ARB} \parallel \mathrm{CLK} \parallel \mathrm{ASSER}), \sigma_7\rangle \xrightarrow{a} \langle \mathrm{R}_0;\ {*}(\mathrm{ARB} \parallel \mathrm{CLK} \parallel \mathrm{ASSER}), \sigma_8 \rangle$, where $\sigma_8 = \{\mathit{grant} \mapsto \text{true}, \mathit{clk} = \text{true}, \mathit{request} \mapsto \text{true}, \mathit{clk}_c \mapsto \text{false}, t \mapsto 8, \mathtt{time} \mapsto 8\}$.

10. Following the same fashion, more transition traces can be performed according to the deduction rules.

5.5 Achievement

In σ_8, the variable t is mapped to the value of the current time (when the property is checked). This also means that the property holds, i.e. the arbiter worked as expected at least for *"one time"*.

5.6 Hardware Description of the Arbiter

For those who are familiar with hardware description languages, the formal χ specification of the arbiter presented in Subsection 5.3 can be regarded as the mathematical model of the below hardware description of the arbiter in Verilog.

```
module assert();
reg clk, grant, request;
time current_time;
initial begin
   clk = 0;
   grant = 0;
   request = 0;
   #4 request = 1;
   #4 grant = 1;
   #4 request = 0;
   #4 $finish;
end
always #5 clk = ~ clk;
always @ (posedge clk)
begin
if (grant == 1 && request == 1) begin
    current_time = $time;
    $display {``working as expected'');
    end
end
endmodule
```

6 OTHER ANALYSIS TECHNIQUES FOR χ

Using the deduction rules and properties of χ to analyze χ specifications may not be intuitive to those who have not a strong computer science background, because rewriting of the specifications (based on the properties) and formal reasoning (based on the deduction rules) have to be made. Nevertheless, this analysis approach (by means of deduction rules and properties) is one of many analysis possibilities offered by χ.

Hence, in this section, we survey various approaches that can be effectively used for the analysis of hardware systems described in χ (for different analysis purposes).

- In process algebras, *linearization* is a transformation of a recursive specification into a linear representation, i.e., a kind of normal form that is convenient for many forms of analysis. Note that these linear representations are expressed as recursive specifications as well, but they use only a small subset of the full process algebra. In general, such linear representations can also be considered very compact representations of a possibly infinite state space. The original recursive specification and its transformation are required to be bisimilar, which ensures that the relevant specification properties are preserved. Some algorithms for linearization of hybrid Chi have already been developed (see [25, 26] for details). These algorithms can be reasonably easily adopted for χ.
- We can use χ tools for simulation and verification of χ specifications. Below is a summary for χ tools:

  χ **simulator:** a simulator for χ specifications was built (based on the formal semantics of χ).

  χ **translators:** automatic translation tools, which convert χ specifications to the corresponding models/specifications in mCRL (their AUT formats can be verified by CADP), PROMELA (the input language of SPIN) and timed automata (the input language of UPPAAL). In addition:

  1. The correctness proof of the translations from (a subset of) χ to PROMELA and from χ to timed automata were given in [5] and [8] respectively. It is proved that any transition of a χ specification can be mimicked by a transition in the corresponding PROMELA model and timed automaton model, which indicates that translations as defined are correct. These also mean that relevant properties of χ specifications are preserved in the translations.
  2. Due to the translation from a χ to the PROMELA is correct, property (e.g. safety and liveness) expressed in temporal logic LTL (which is the type of temporal logic used for verification in SPIN) can be verified in the translated χ specification in PROMELA using SPIN. If the property holds, this also implies/relating back that such a property (regardless of the form of representation) should hold in the original χ specification because of the preservativity of the translation.
  3. Similar approach can be applied to verify properties expressed in TLTL [37] in a χ specification via the formal translation from χ to timed automata using UPPAAL.

  Availability: simulator, translation tools and manuals of χ can be found in [38].

7 CONCLUSIONS

χ can be reasonably and effectively used to give formal specifications of hardware systems and possibly to analyze them as indicated in this paper.

In our opinion, with respect to common hardware description languages (e.g. VHDL and Verilog), χ can precisely describe the behavior of hardware systems in a complete mathematical way. Prior to χ, from literature, several process algebra based formalisms (e.g. CHP [24]) were used to give formal specifications of hardware systems. In addition to those formalisms, χ enhances strength for formal specification, because χ has a comprehensive set of operators that enables process re-use, encapsulation, hierarchical and/or modular composition of processes, etc.

Furthermore, χ has a rich set of support tools. Referring to the χ specifications provided in Section 4, we can simulate the dynamic behavior of the MUX and D flip-flop using the χ simulator. For the asynchronous arbiter, we can translate such a χ specification to the corresponding PROMELA model using the χ automatic translation tool and then apply the model checker SPIN to verify some safety properties (e.g. whether the mutual exclusion is valid) on the translated χ specification in PROMELA. A complete mathematical proof for the assertion of the simple arbiter was already given in Subsection 5.4.

Recently, several other timed process algebras have been developed (e.g. $SystemC^{\mathbb{FL}}$ [42, 43] and PAFSV [45]) that can also be used for formal specification and analysis of hardware systems. Our future work will focus on a comparative study between χ and such other timed process algebras.

8 AVAILABILITY

For research purposes, we would be pleased to receive interesting case studies on formal specification and analysis of hardware systems from anyone working in this area.

For more information, please send mail to pafesd@gmail.com or visit PAFESD website `http://digilander.libero.it/pafesd/`.

Acknowledgments

K. L. Man wishes to thank Jos Baeten, Bert van Beek, Mohammad Mousavi, Koos Rooda, Ramon Schiffelers, Pieter Cuijpers, Michel Reniers, Kees Middelburg, Uzma Khadim and Muck van Weerdenburg for many stimulating and helpful discussions (focusing on process algebras for distinct systems) in the past few years.

He is indebted to Andrea Fedeli for his significant contribution to the draft version of this paper. Also, he would like to thank Michel Schellekens and Menouer Boubekeur for many stimulating and helpful discussions.

REFERENCES

[1] Baeten, J. C. M.—Weijland, W. P.: Process Algebra. Volume 18 of Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, Cambridge, United Kingdom 1990.

[2] Baeten, J. C. M.—Middelburg, C. A.: Process Algebra With Timing. EATCS Monographs Series, Springer-Verlag,2002.

[3] Nicollin, X.—Sifakis, J.: The Algebra of Timed Processes ATP: Theory and Application. Information and Computation, Vol. 114, 1994, No. 1, pp. 131–178.

[4] Man, K. L.—Schiffelers, R. R. H.: Formal Specification and Analysis of Hybrid Systems. Ph. D. Thesis, Eindhoven University of Technology, The Netherlands 2006.

[5] Trcka, N.: Silent Steps in Transition Systems and Markov Chains. Ph. D. Thesis, Eindhoven University of Technology, The Netherland, 2007.

[6] Cuijpers, P. J. L.—Reniers, M. A.: Hybrid Process Algebra. Journal of Logic and Algebraic Programming, Vol. 62, 2005, No. 2, pp. 191–245.

[7] van Beek, D. A.—Man, K. L.—Reniers, M. A—Rooda, J. E.—Sschiffelers, R. R. H.: Syntax and Semantics of Timed Chi. Technical Report CS-Report 05-09, Eindhoven University of Technology, Department of Computer Science, The Netherlands 2005.

[8] Bortnik, E. M.—van de Mortel-Fronczak, J. M.—Rooda, J. E.: Translating Chi Models to UPPAAL Timed Automata. Technical Report SE Report 2007-06, Eindhoven University of Technology, Department of Mechanical Engineering, The Netherlands 2007.

[9] van Beek, D. A.—van den Ham, A.—Rooda, J. E.: Modelling and Control of Process Industry Batch Production Systems. 15th TriennialWorld Congress of the International Federation of Automatic Control, Barcelona, Spain 2002.

[10] van Beek, D. A.—Rooda, J. E.: Languages and Applications in Hybrid Modelling and Simulation: Positioning of Chi. Control Engineering Practice, Vol. 8, 2000, No. 1, pp. 81–91.

[11] Braspenning, N. C. W. M.—Bortnik, E. M.–van de Mortel-Fronczak, J. M.—Rooda, J. E.: Model-Based System Analysis Using Chi and UPPAAL: An Industrial Case Study. Computers in Industry, Vol. 59, 2008, No. 1, pp. 41–54.

[12] Baeten, J. C. M.—van Beek, D. A.—Cuijpers, P. J. L.—Reniers, M. A.—Rooda, J. E.—Schiffelers, R. R. H.—Theunissen, R. J. M.: Model-Based Engineering of Embedded Systems Using the Hybrid Process Algebra Chi. ENTCS, 2007.

[13] van Beek, D. A.—Man, K. L.—Reniers, M. A.—Rooda, J. E.—Schiffelers, R. R. H.: Syntax and Consistent Equation Semantics of Hybrid Chi. Journal of logic and algebraic programming, Vol. 68, 2006, No. 1-2, pp. 129–210.

[14] Plotkin, G. D.: A Structural Approach to Operational Semantics. Technical Report DIAMI FN-19, Aarhus University, Department of Computer Science, Denmark 1981.

[15] IEEE standard for Verilog hardware description language. IEEE Std 1364-2005 (revision of IEEE Std 1364-2001), IEEE Computer Society 2005.

[16] IEEE standard for VHDL language reference manual. IEEE Std 1076-2000 (incorporates IEEE Std 1076-1993 and IEEE Std 1076a-2000), IEEE Computer Society 2000.

[17] IEEE standard for SystemC language reference manual. IEEE Std 1666TM-2005, IEEE Computer Society 2005.

[18] IEEE standard for SystemVerilog – unified hardware design, specification and verification language. IEEE Std 1800TM-2005, IEEE Computer Society, 2005.

[19] Schneider, G.—Qiwen, X.: Towards an Operational Semantics of Verilog. UNU/IIST Report No. 147, International Institute for Software Technology, United Nations University, Macau 1998.

[20] Schneider, G.—Qiwen, X.: Towards a Formal Semantics of Verilog Using Duration Calculus Towards an Operational Semantic. Formal Techniques for Real-Time and Fault Tolerant Systems (FTRTFT '98), Lecture Notes in Computer Science, Springer-Verlag 1998.

[21] Breuer, P. T.—Delgado Kloos, C.: Formal Semantics for VHDL. Kluwer Academic Publishers 1995.

[22] Sasaki, H.—Mizushima, K.—Sasaki, T.: A Formal Semantics for Verilog-VHDL Simulation Interoperability by Abstract State Machine. Proceedings of the Conference on Design, Automation and Test in Europe, Munich, Germany 1999.

[23] Huibiao, Z.—Jifeng, H.: A DC-Based Semantics for Verilog. UNU/IIST Report No. 183, International Institute for Software Technology, United Nations University, Macau 2000.

[24] Salaun, G.—Serwe, W.: Translating Hardware Process Algebras Into Standard Process Algebras – Illustration with CHP and LOTOS. Proceedings of the International Conference on Integrated Formal Methods, Eindhoven, The Netherlands 2005.

[25] Baeten, J. C. M.—van Beek, D. A.—Rooda, J. E.: Handbook of Dynamic System Modeling (Chapter Process Algebra). CRC Press LLC, 2006.

[26] Khadim, U.—van Beek, D. A.—Cuijpers, P. J. L.: Linearization of Hybrid Chi Using Program Counters. Technical Report CS-Report 07-18, Eindhoven University of Technology, Department of Computer Science, The Netherlands 2007.

[27] Bortnik, E. M.—Ttrcka, N.—Wijs, A. J.—Luttik, B.—van de Mortel-Fronczak, J. M.—Baeten, J. C. M.—Fokkink, W. J.—Rooda, J. E.: Analyzing a Chi Model of a Turntable System Using SPIN, CADP and UPPAAL. Journal of Logic and Algebraic Programming, Vol. 65, 2005, No. 2, pp. 51–104.

[28] Bortnik, E. M.—van Beek, D. A.—van de Mortel-Fronczak, J. M.—Rooda, J. E.: Verification of Timed Chi Models Using UPPAAL. Proceedings of the 2nd International Conference on Informatics in Control, Robotics and Automation, Barcelona, Spain 2005.

[29] Davies, J.—Schneider, S.: A Brief History of Timed CSP. Theoretical Computer Science, Vol. 138, 1995, pp. 183–235.

[30] Mahony, B.—Dong, J. S.: Blending Object-Z and Timed CSP: An Introduction to TCOZ. Proceedings of the 20th International Conference on Software Engineering, Kyoto, Japan 1998.

[31] Westerlund, T.—Plosila, J.: Formal Timing Model for Hardware Components. Technical Report 640 (tWePl04a), Turku Centre for Computer Science, Finland 2004.

[32] Baeten, J. C. M.: A Brief History of Process Algebra. Technical Report CS-Report 04-02, Eindhoven University of Technology, Department of Computer Science, The Netherlands 2004.

[33] Hoare, C. A. R.: Communicating Sequential Processes. Communications of the ACM, Vol. 21, 1978, No. 8, pp. 666–467.

[34] Aceto, L.—Fokkink, J.—Verhoef, C.: Structural Operational Semantics. Handbook of Process Algebra, Elsevier 2001.

[35] Fernandez, J. C.—Garavel, H.—Kerbrat, A.—Mounier, L.—Mateescu, R.—Sighireanu, M.: CADP – A Protocol Validation and Verification Toolbox. Proceedings of the 8th Conference on Computer Aided verification, Volume 1102 of Lecture Notes in Computer Science, 1996, pp. 437–440.

[36] Holzmann, G. J.: The SPIN Model Checker. Addison-Wesley 2003.

[37] Larsen, K. G.—Pettersson, P.—Yi, W.: UPPAAL in a Nutshell. Journal on Software Tools for Technology Transfer, Vol. 1, 1997, No. 1-2, pp. 134–152.

[38] Chi Tools and Manuals. Available on `http://se.wtb.tue.nl/sewiki/chi/`.

[39] NUSMV Model Checker. Available on `http://nusmv.irst.itc.it/`.

[40] ACL2. Available on `http://www.cs.utexas.edu/moore/acl2/`.

[41] ISABELLE. Available on `http://isabelle.in.tum.de/`.

[42] Man, K. L.: SystemCFL: Formalization of SystemC. IEEE Proceedings of the Mediterranean Electrotechnical Conference, Dubrovnik, Croatia 2004.

[43] Man, K. L.: SystemCFL: Formal specification and analysis of hardware/software codesigns. Journal of The World Scientific and Engineering Academy and Society Transactions on Circuits and Systems, Vol. 3, 2006, No. 5, pp. 361–368.

[44] Man, K. L.—Schellekens, M. P.: Mathematical Modelling of Digital Hardware Systems in Timed Chi. Proceedings of the 26th IASTED International Conference on Modelling, Identification and Control, Innsbruck, Austria 2007.

[45] Man, K. L.—Boubekeur, M.—Schellekens, M. P.: Process Algebraic Approach to SystemVerilog. Proceedings of the IEEE Canadian Conference on Electrical and Computer Engineering, Vancouver, British Columbia, Canada 2007.

Ka Lok Man holds a Dr. Eng. degree in electronic engineering from Politecnico di Torino (Italy) and a Ph. D. degree in computer science from Technische Universiteit Eindhoven (The Netherlands). From June 1999 through October 2000, he was a research assistant at the Electronic Design Automation Group (EDA), Department of Electrical and Computer Engineering, Politecnico di Torino (Italy). From September 1999 through September 2000, he was a visiting researcher at the University of Colorado at Boulder, USA. From October 2000 through January 2002, he was a researcher in the Research and Development Centre of STMicroelectronics, Agrate (Milan, Italy) Currently, he is a senior researcher at the Centre for Efficiency-Oriented Languages (CEOL), Department of Computer Science, University College Cork, Ireland. His research interests include logic synthesis, formal verification and low power design methodologies for digital integrated circuits and systems, formal specification and analysis of hardware, analog and mixed-signal systems, formalisation of SystemC and SystemVerilog designs, formal methods, process algebras and formal analysis of real-time, hybrid and embedded systems. On the above-mentioned topics, he has authored or co-authored about 50 refereed publications. Over the years, he has received several paper awards at IEEE conferences, research awards and grants from the Italian government, IEEE and industries. Also, he has been a committee member, reviewer, session chair and special session organiser-chairman of different IEEE, IASTED, IAENG and WSEAS conferences. Currently, he is a committee member and/or reviewer of the International Journal of Engineering Letters, the IAENG International Journal of Computer Science and the IBSU Scientific Journal.