Analog Property Checkers: A DDR2 Case Study

Kevin D. Jones[1], Victor Konrad[1] and Dejan Ničković*[2]

[1] Rambus Inc., 4440 El Camino Real, 94040 Los Altos, USA
[kdj | vkonrad]@rambus.com
[2] Verimag, Université de Grenoble, 2 Av. de Vignate, 38610 Gières, France
nickovic@imag.fr

Abstract. The formal specification component of verification can be exported to simulation through the idea of property checkers. The essence of this approach is the automatic construction of an observer from the specification in the form of a program that can be interfaced with a simulator and alert the user if the property is violated by a simulation trace. Although not complete, this lighter approach to formal verification has been effectively used in software and digital hardware to detect errors. Recently, the idea of property checkers has been extended to analog and mixed signal systems.
In this paper, we apply the property-based checking methodology to an industrial and realistic example of a DDR2 memory interface. The properties describing the DDR2 analog behavior are expressed in the formal specification language STL/PSL in form of assertions. The simulation traces generated from an actual DDR2 interface design are checked with respect to the STL/PSL assertions using the AMT tool. The focus of this paper is on the translation of the official (informal and descriptive) specification of two non-trivial DDR2 properties into STL/PSL assertions. We study both the benefits and the current limits of such approach.

1 Introduction

The formal verification of digital (and other finite state) systems has been based on the decision procedures which often involve model-checking temporal logic formulae. Temporal logic [MP95] is a rigorous specification formalism that is used to describe desired behaviors of the system. The fact that logics such as LTL or CTL can be efficiently translated into corresponding automata [VW86,SB00,GPVW95,GO01] has facilitated their integration into main verification tools. An adaptation of formalisms based on temporal logics and regular expressions to the needs of the hardware industry has been done through standard specification languages PSL [HFE04] and SVA [Acc04].

Similar verification methods have been introduced in the analog and mixed signal domain with the advent of *hybrid automata* [MMP92], which serve as a model to describe systems with continuous dynamics with switches, and the algorithms for the exhaustive exploration of their search space. While certain progress has been made recently in that field [ADF+06], scalability remains an important issue for the exhaustive verification of hybrid systems, due to the explosion of the underlying state space. Consequently, this verification method can be used nowadays to reason about small critical

* Intern at Rambus, Inc. during this work

analog and mixed signal blocks containing up to a dozen continuous variables. Moreover, property-based verification of hybrid systems is only at its beginning [FGP06].

The preferred analog validation method remains simulation-based testing, combined with a number of common analysis techniques (frequency-domain analysis, statistical measures, parameter extraction, eye detection etc.) The validation tools are specific to the class of properties checked, and include waveform analyzers and calculators, measuring commands as well as manually written scripts. These solutions are often ad-hoc and may require considerable user effort, and in the case of scripts, reusability becomes an issue.

The gap between formal verification and standard simulation analysis of analog systems can be reduced by introducing formal specifications into the domain of simulation. This approach relies on an automatic construction of an observer, also called a property checker, from the formula. This checker takes the form of a program that can be interfaced with the simulator and alert the user if the property is violated by a simulation trace. This method is not complete, but can be effectively used to catch "bugs" in the system. It can be more reliable and efficient than the visual inspection of simulation traces, or manual construction of property observers. This procedure, often related to as *lightweight verification*, has been successfully integrated into the validation flow of software and hardware frameworks, and temporal logic has been used as the specification language in a number of property checking tools, including TemporalRover [Dru00], FoCs [ABG+00], Java PathExplorer [HR01] and MaCS [KLS+02].

The extension of property-based checkers to analog and mixed signal systems has been proposed in [MN04,NM07,N08], with the introduction of the formal specification language STL/PSL, based on the dense-time temporal logic MITL [AFH96], and it allows to relate temporal behavior of continuous waveforms via their *static abstractions*. The properties expressed in STL/PSL can be checked against analog simulation traces with the tool AMT [NM07,N08]. A similar approach for checking PSL properties of discrete time analog and mixed signal systems was proposed in [AZDT07]. The authors of [JHP+07] describe a framework based on PSL extended with analog operators, which is targeted at checking mixed signal interface properties. In [DC05], the authors introduce the ANACTL logic, an analog extension of CTL, used to check properties of a finite state machine, which represents a set of discretized and bounded transient simulation traces.

In this paper, we study the framework of property checkers in the analog domain and its applicability to real-world industrial examples. We present a case study where we translate two non-trivial properties of a DDR2 memory interface [Jed06] in STL/PSL and use the monitoring tool AMT to check the specification against the simulation waveforms. DDR2 memory is a natural candidate for this case study as it contains a number of timing relations between different analog signals. We are particularly interested in the expressiveness of STL/PSL with respect to the class of properties informally described in the official DDR2 specification document.

The rest of this document is organized as follows: in Section 2 we present the STL/PSL specification language. Section 3 describes some non-trivial properties of the DDR2 memory component and their formalization and translation into STL/PSL. The

experimental results are reported in Section 4.2 followed by a discussion about the results and the conclusions (Section 5).

2 STL/PSL Specification Language

The specification of properties of continuous waveforms requires an adaptation of the semantic domain and the underlying logic. Let the time domain $\mathbb{T}$ be the set $\mathbb{R}_{\geq 0}$ of non-negative real numbers. We consider finite length signals ξ over an abstract domain $\mathbb{D}$, which are partial functions $\xi : \mathbb{T} \rightarrow \mathbb{D}$ whose domain of definition is $I = [0, r), r \in \mathbb{Q}_{>0}$. The length of the signal ξ is r, and is denoted with $|\xi| = r$. We restrict our attention to two particular types of signals, Boolean signals with $\mathbb{D} = \mathbb{B}$ and continuous signals with $\mathbb{D} = \mathbb{R}$. We denote by $\pi_p(\xi)$ the projection of the signal ξ on the dimension with domain $\mathbb{B}$ that corresponds to the proposition p (likewise, $\pi_s(\xi)$ denotes projection of the signal ξ on the dimension with domain $\mathbb{R}$ corresponding to the continuous variable s).

The STL/PSL logic is an extension of MITL [AFH96] and STL [MN04] logics, using layers in the fashion of PSL [HFE04]. The *analog layer* allows to reason about continuous signals and the *temporal layer* relates the temporal behavior of input traces. The "communication" between the two layers is done via *static abstractions* that partition the continuous state space according to some (in)equality constraints on the continuous variables. The STL/PSL properties are targeted at the *lightweight verification* over *finite* traces, so the language adopts the finitary interpretation in the spirit of PSL, with *strong* and *weak* forms of the temporal operators[1]. The *analog layer* of STL/PSL is defined by the following grammar:

$$\phi :== s \mid \texttt{shift}(\phi, \texttt{k}) \mid \phi_1 \star \phi_2 \mid \phi \star \texttt{c} \mid \texttt{abs}(\phi)$$

where s belongs to a set $S = \{s_1, s_2, \ldots, s_n\}$ of continuous variables, $\star \in \{\texttt{+}, \texttt{-}, \texttt{*}\}$, $\texttt{c} \in \mathbb{Q}$ and $\texttt{k} \in \mathbb{Q}^+$.

The semantics of the analog layer of STL/PSL is defined as an application of the analog operators to the input signal ξ:

$$\begin{array}{ll} s[t] & = \pi_s(\xi)[t] \\ \texttt{shift}(\phi, \texttt{k})[t] & = \phi[t + \texttt{k}] \\ (\phi_1 \star \phi_2)[t] & = \phi_1[t] \star \phi_2[t] \\ (\phi \star \texttt{c})[t] & = \phi[t] \star \texttt{c} \\ \texttt{abs}(\varphi)[t] & = \begin{cases} \phi[t] & \text{if } \phi[t] \geq 0 \\ -\phi[t] & \text{otherwise} \end{cases} \end{array}$$

The *temporal layer* of STL/PSL contains both *future* and *past* operators and is defined as follows:

$$\begin{array}{c} \varphi :== p \mid \phi \circ \texttt{c} \mid \texttt{not}\ \varphi \mid \varphi_1\ \texttt{or}\ \varphi_2 \mid \\ \varphi_1\ \texttt{until!}\ I\ \varphi_2 \mid \varphi_1\ \texttt{until}\ I\ \varphi_2 \mid \varphi_1\ \texttt{since}\ I\ \varphi_2 \\ \texttt{rise}(\varphi) \mid \texttt{fall}(\varphi) \end{array}$$

[1] The strong form of an operator requires the terminating condition to occur before the end of the signal, while the weak form makes no such requirements. In PSL for example, `until!` and `until` represent the strong and the weak form of the until operator, respectively.

where p belongs to a set $P = \{p_1, p_2, \ldots, p_n\}$ of propositional variables, `a,b,c` $\in \mathbb{Q}$, $\circ \in \{$`>,>=,<,<=,==`$\}$ and I is an interval of type (a, b), $(a, b]$, $[a, b)$, $[a, b]$, (a, ∞) or $[a, \infty)$, where a, b are rationals with $0 \leq a < b$.

The satisfaction relation $(\xi, t) \models \varphi$, indicating that signal ξ satisfies φ at time t is defined inductively as follows:

$$
\begin{array}{ll}
(\xi,t) \models p & \text{iff } \pi_p(\xi)[t] = \textsc{True} \\
(\xi,t) \models \phi \circ \texttt{ c} & \text{iff } \phi[t] \circ \texttt{c} \\
(\xi,t) \models \texttt{not } \varphi & \text{iff } (\xi,t) \not\models \varphi \\
(\xi,t) \models \varphi_1 \texttt{ or } \varphi_2 & \text{iff } (\xi,t) \models \varphi_1 \text{ or } (\xi,t) \models \varphi_2 \\
(\xi,t) \models \varphi_1 \texttt{ until! } I\ \varphi_2 & \text{iff } \exists t' \in t + I \text{ st } (t' < |\xi| \text{ and } (\xi,t') \models \varphi_2) \text{ and} \\
 & \quad \forall t'' \in (t,t')\ (\xi,t'') \models \varphi_1 \\
(\xi,t) \models \varphi_1 \texttt{ until } I\ \varphi_2 & \text{iff } \exists t' \in t + I \text{ st } (t' \geq |\xi| \text{ or } (\xi,t') \models \varphi_2) \text{ and} \\
 & \quad \forall t'' \in (t,t')\ (\xi,t'') \models \varphi_1 \\
(\xi,t) \models \varphi_1 \texttt{ since } I\ \varphi_2 & \text{iff } \exists t' \in t - I \text{ st } t' \geq 0 \text{ and } (\xi,t') \models \varphi_2 \text{ and} \\
 & \quad \forall t'' \in (t,t')\ (\xi,t'') \models \varphi_1 \\
(\xi,t) \models \texttt{rise(}\varphi\texttt{)} & \text{iff } ((\xi,t) \models \varphi \text{ and } \exists\ t' \in [0,t) \text{ st} \\
 & \quad \forall t'' \in (t',t)\ (\xi,t'') \not\models \varphi) \text{ or} \\
 & \quad ((\xi,t) \not\models \varphi \text{ and } \exists\ t' > t \text{ st} \\
 & \quad \forall t'' \in (t,t')\ (\xi,t'') \models \varphi) \\
(\xi,t) \models \texttt{fall(}\varphi\texttt{)} & \text{iff } ((\xi,t) \not\models \varphi \text{ and } \exists\ t' \in [0,t) \text{ st} \\
 & \quad \forall t'' \in (t',t)\ (\xi,t'') \models \varphi) \text{ or} \\
 & \quad ((\xi,t) \models \varphi \text{ and } \exists\ t' > t \text{ st} \\
 & \quad \forall t'' \in (t,t')\ (\xi,t'') \not\models \varphi)
\end{array}
$$

An STL/PSL specification φ_{prop} is an STL/PSL temporal formula. The signal ξ satisfies the specification φ_{prop}, denoted by $\xi \models \varphi_{\text{prop}}$, iff $(\xi, 0) \models \varphi_{\text{prop}}$.

Other standard operators such as strong and weak versions of *always* and *eventually*, as well as their past counterparts *historically* and *once* can be derived from the basic ones. Note that the syntax and semantics of STL/PSL differ from [NM07] in several aspects. The *until* operator has the strict semantics as originally proposed in [AFH96] and the past operators as well as events (detection of rising and falling edges of a signal) have been added to the language[2].

The STL/PSL language contains some additional constructs that simplify the process of property specification. Each top-level STL/PSL property is declared as an *assertion*, and a number of assertions can be grouped into a single logical unit in order to monitor them together at once. We also add a definition directive which allows the user to declare a formula and give it a name, and then refer to it as a variable within the assertions. The Boolean and analog variables are typed (prefixes `b:` and `a:`, respectively). The extended STL/PSL is defined with the following production rules

```
stl_psl_prop :==
  vprop NAME {
    { define_directive } { assert_directive }
```

[2] The underlying changes that were done to support these extensions are out of scope in this paper, see [N08] for more details

}

```
define_directive :==
  define b:NAME := stl_psl_property
  | define a:NAME := analog_expression

assert_directive :==
  NAME assert : stl_psl_property
```

where `stl_psl_property` and `analog_expression` correspond to φ and ϕ defined above, respectively.

3 Translation of DDR2 Properties to STL/PSL Assertions

The subject of this case study is a DDR2 memory interface developed at Rambus. DDR2 presents a number of features that make it a good candidate for property-based monitoring approach. The memory interface acts as a bus between the memory and other components in the circuit and exhibits the communication of digital data implemented at the analog level. Hence, the correct functioning of a DDR2 memory interface largely depends on the appropriate timing of different signals within the circuit. In this section, we describe two typical DDR2 properties, one specifying the correct alignment between analog signals, and the other reasoning about time accumulation error in the clock signal. We focus on different steps needed for translating these informally described properties into an STL/PSL specification.

The simulation traces provided by Rambus are from a DDR2-1066 interface with single-ended data strobe, but there are no written specification documents for this particular design setting. Instead, we used the specification for DDR2-400 and DDR-800 from the official document.

3.1 Data and Data Strobe Alignment Property

In DDR2, the data access is controlled by a single-ended or differential data strobe signal, which acts as an asynchronous clock. The official JEDEC DDR2 specification describes, amongst others, a number of properties that involve timing relationship between events that happen in data and data strobe signals. In this section, we are particularly interested in a property that defines the correct alignment between the data and data strobe signals. The case study considers the specification parameters for the single-ended data strobe DDR2-400 memory interface, which is part of the JEDEC standard.

The DDR2 specification defines a number of thresholds, shown in Table 1. The temporal relationship between data signal DQ and data strobe signal DQS is defined with respect to different crossings of these thresholds.

The general definition of the alignment of data DQ and data strobe DQS signals is illustrated in Figure 1. The proper alignment between the two signals is determined by two values, the *setup* time tDS and *hold* time tDH. The setup and hold times of DQ and DQS are checked on both their *falling* and *rising* edges. For the sake of simplicity,

Threshold	Value (V)
V_{DDQ}	1.8
$V_{IH(AC)_{min}}$	1.25
$V_{IH(DC)_{min}}$	1.025
V_{REF}	0.9
$V_{IL(DC)_{max}}$	0.775
$V_{IL(AC)_{max}}$	0.65

Table 1. Threshold values for DQ and DQS

we only consider the specification of the property for the setup time at the signals' falling edge and the other cases are similar and symmetric.

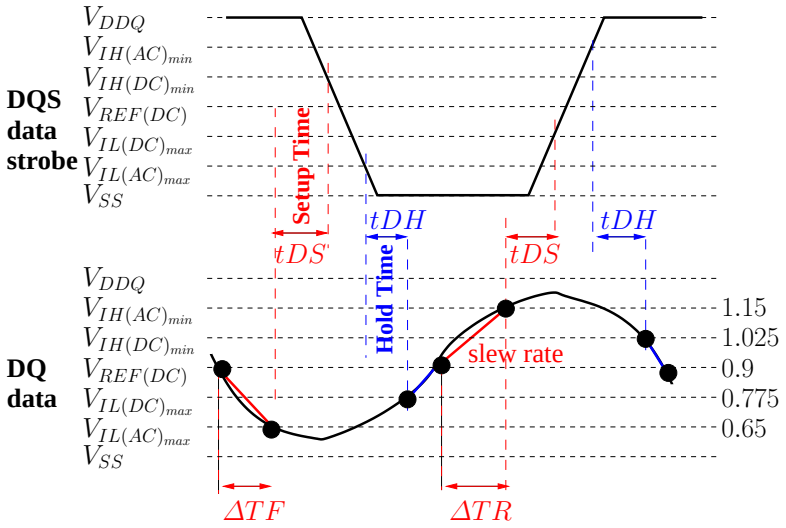

Fig. 1. Data DQ and data strobe DQS alignment

Informally, the setup property at the falling edge requires that whenever DQS crosses the $V_{IH(DC)_{min}}$ threshold from above, the previous crossing of $V_{IL(AC)_{max}}$ by the signal DQ from above should precede it by at least a period of time of tDS. This property is formalized in STL/PSL as follows

```
define b:dqs_above_vihdcmin := (a:DQS >= 1.025);
define b:dqs_above_vilacmax := (a:DQ >= 0.65);

always (fall(b:dqs_above_vihdcmin)
  -> historically[0:tDS] not fall(b:dq_above_vilacmax));
```

The above property is, as one can see, naturally expressed in STL/PSL, but unfortunately, it does not present the full reality. In fact, setup time tDS is not a constant value, but rather varies during the simulation according to the slew rates (slopes) of DQ and DQS signals. For example, when DQ and DQS fall more sharply, the required tDS increases. Setup time tDS is defined as the sum of a (constant) *base term* $tDS(base)$ and a (variable) *correction term* ΔtDS

$$tDS = tDS(base) + \Delta tDS$$

The setup base term $tDS(base)$ is equal to $150ps$ for the single-ended DDR2-400. The correction term ΔtDS is a value that varies according to the slew rates of DQ and DQS, with the setup slew rate of a falling signal being defined as

$$sr = \frac{V_{REF} - V_{IL(AC)_{max}}}{\Delta TF} \tag{1}$$

where ΔTF is the time that the signal spends between $V_{REF(DC)}$ and $V_{IL(AC)_{max}}$. As we can see, the falling setup slew rate sr of a signal can be deduced from ΔTF.

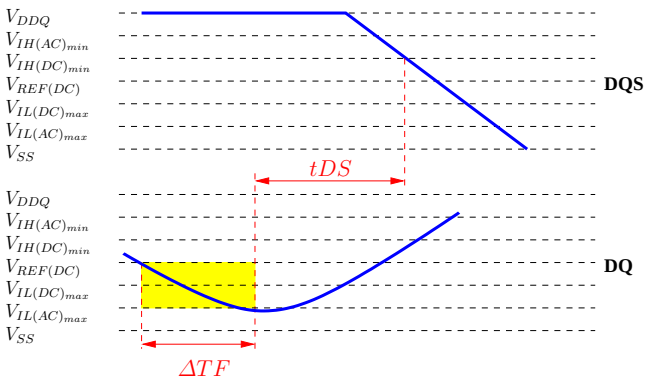

Fig. 2. DQ/DQS falling setup time tDS and the relation between slew rate and ΔTF

In order to extract the setup correction term ΔtDS from the actual slew rates of DQ and DQS (sr_{DQ} and sr_{DQS}), we can use a specification table from [Jed06], partially reproduced in Table 2. According to the JEDEC specification, ΔtDS corresponding to the slew rates not listed in Table 2 should be linearly interpolated. Consequently, we can apply the following sequence of computations in order to determine the correct value of tDS at any time

1. Measure ΔTF, the time that the signal remains within the setup falling slew rate region
2. Infer the *setup falling slew rate* value from ΔTF
3. Interpolate the *correction term* from the slew rate specification table

		DQS Single-Ended Slew Rate tDS			
		2V/ns	1.5V/ns	1V/ns	0.9V/ns
DQ	2V/ns	**188**	167	125	
Single-Ended	1.5V/ns	146	125	83	81
Slew Rate	1V/ns	63	42	0	-2
tDS	0.9V/ns		31	-11	13

Table 2. Correction terms for setup time

4. Add the correction term to the base term in order to obtain tDS

To summarize, tDS is a value that varies during the simulation as a function of slew rates of DQ and DQS ($tDS = f(sr_{DQ}, sr_{DQS})$). The problem is that STL/PSL cannot capture parametrized time bounds and therefore we have to use approximation in order to express a similar alignment property that still preserves some guarantees. We can subdivide the domain of slew rates (say $S = [sr_{min}, sr_{max}]$) into n regions $R_1, \ldots, R_n$. For each pair (R_i, R_j) of DQ/DQS slew rate regions, we assign a separate constant setup time tDS_{ij}. Instead of one property, we will have $n \times n$ properties of the form:

– "whenever DQS crosses the $V_{IH(DC)_{min}}$ threshold from above, DQ slew rate sr_{DQ} is in R_i and DQS slew rate is in R_j, the previous crossing of $V_{IL(AC)_{max}}$ by the signal DQ from above should precede it by at least a period of time of tDS_{ij}."

The proper constant value for tDS_{ij} for a pair of slew rate regions (R_i, R_j) can be chosen in two different manners. The first solution consists in computing tDS_{ij} from the maximum correction term for the DQ and DQS slew rates that are in the R_i and R_j regions, respectively. This corresponds to an over-approximation of the original specification, and if this property is violated, we don't know if it is a real failure or a false alarm. On the other hand, the satisfaction of the over-approximated property implies that the original property holds too. Conversely, the computation of tDS_{ij} from the minimum correction term defined for the slew rates in the pair of regions (R_i, R_j) yields to an under-approximation of the original property. If the new property is falsified, we know that it corresponds to a real violation, while if it passes, we cannot say whether we are indeed safe.

As an example, consider the highlighted range of Table 2, which we call the "top-left" range, where the setup falling slew rates of DQ and DQS are between 1 and 2 V/ns. For the conservative approximation of tDS, with slew rates falling in that range, we choose the worst-case ΔtDS as the correction term, that is $188ps$. Hence, the approximated falling setup time tDS_{TL} for all DQ and DQS with falling slew rates between 1 and $2V/ns$ would be equal to $tDS_{TL} = 150 + 188 = 338ps$.

In order to determine the falling slew rates of DQ and DQS, we need to detect how much time these signals remain in their falling slew region (between $V_{REF(DC)}$ and $V_{IL(AC)_{max}}$ crossing $V_{REF(DC)}$ from above). We can detect when the signal is within the falling slew region with the following properties

```
define b:dq_in_fsr :=
  ((a:DQ <= 0.9) and (a:DQ >= 0.65)) since (a:DQ >= 0.9)

define b:dqs_in_fsr :=
  ((a:DQS <= 0.9) and (a:DQS >= 0.65)) since (a:DQS >= 0.9)
```

which hold if the signal is in the falling slew region, as shown in Figure 3.

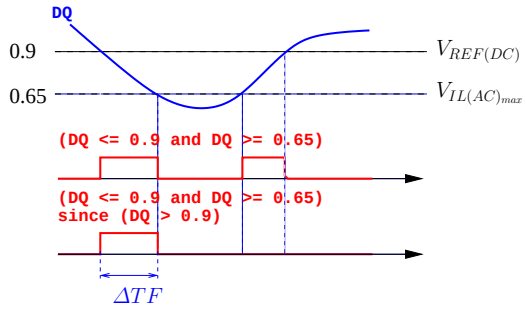

Fig. 3. Falling slew region and ΔTF

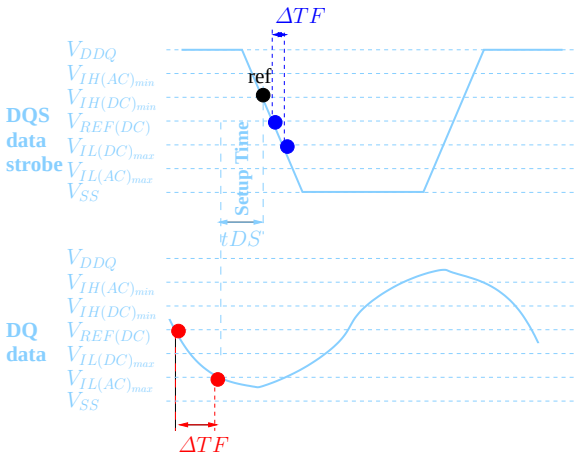

Fig. 4. Relation between the reference point and the corresponding ΔTF of DQ and DQS

Note that according to equation (1), DQ and DQS have their slew rates in the range between 1 and $2V/ns$ if their respective ΔTF is between 125 and $250ps$. Moreover, the

value of tDS is determined at the crossing of $V_{REF(DC)}$ by DQS from above (point **ref** in Figure 4) with respect to the previous falling setup slew rate of DQ and the next falling setup slew rate of DQS, as shown in Figure 4. Hence, the falling slew rates of DQ and DQS are in the range between 1 and $2V/ns$ if the following formulae hold

```
define b:dq_slew_rate_in_1_2 := not b:dq_in_fsr since
  (b:dq_in_fsr since[125:250] rise(b:dq_in_fsr));

define b:dqs_slew_rate_in_1_2 := not b:dqs_in_fsr until
  (b:dqs_in_fsr until[125:250] fall(b:dqs_in_fsr));

define b:top_left_region :=
    b:dq_slew_rate_in_1_2 and b:dqs_slew_rate_in_1_2;
```

Finally, the main property for the falling setup time, provided that DQ and DQS falling slew rates are in the range between 1 and $2V/ns$, is expressed as

```
define b:dqs_above_vihdcmin := (a:DQS >= 1.025);
define b:dqs_above_vilacmax := (a:DQ >= 0.65);

always ((fall(b:dqs_above_vihdcmin) and b:top_left_region)
  -> historically[0:338] not fall(b:dq_above_vilacmax));
```

with similar properties that have to be written for each range of DQ and DQS slew rates.

3.2 Jitter Property

An important class of DDR2 properties involves the jitter of the clock signals. The differential clock is composed of two signals, CK and CKB and the clock period tCK is defined as the time elapsed between two consecutive crossings of a rising CK and a falling CKB, as show in Figure 5. The average clock period $tCK(avg)$ is computed over 200 consecutive clock periods. Finally, the differential clock accumulation error over n periods is the difference between n actual periods and n average clock periods. The acceptable accumulation error over n clock periods is defined in Figure 6 taken from the DDR2 official specification document.

The purpose of this example is to express accumulation error properties using STL/PSL. In order to be able to specify time accumulation error between n consecutive events, we need a "counting" operator in the spirit of regular expressions. Consequently, for the purpose of the case study we define an ad-hoc operator `next_rise[n]I(phi)`. Intuitively, this operator holds at time t, if and only if the n^{th} consecutive rising edge of `phi` happens within the interval $t' \in t \oplus I$ (see Figure 7). Formally, the semantics of the operator is defined as follows

$(\xi, t) \models$ `next_rise[n]I(phi)` iff $\exists t_1, \ldots t_n$ st $t \leq t_1 < \ldots < t_n$ and
$t_n \in t \oplus I$ and $\bigwedge_{i=1}^{n} (\xi, t_i) \models$ `rise(phi)` and
$\bigwedge_{i=1}^{n-1} \forall t' \in (t_i, t_{i+1})\ (\xi, t') \not\models$ `rise(phi)`

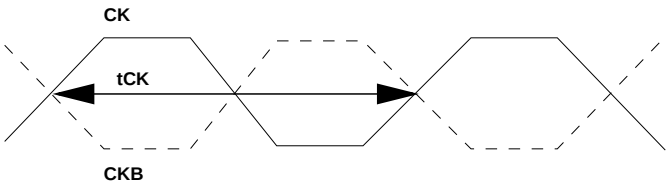

Fig. 5. Differential clock period

Parameter	Symbol	DDR2-667		DDR2-800		Units	Notes
		min	max	min	max		
Clock period jitter	tJIT(per)	- 125	125	-100	100	ps	35
Clock period jitter during DLL locking period	tJIT(per,lck)	- 100	100	-80	80	ps	35
Cycle to cycle clock period jitter	tJIT(cc)	- 250	250	-200	200	ps	35
Cycle to cycle clock period jitter during DLL locking period	tJIT(cc,lck)	- 200	200	-160	160	ps	35
Cumulative error across 2 cycles	tERR(2per)	- 175	175	-150	150	ps	35
Cumulative error across 3 cycles	tERR(3per)	- 225	225	-175	175	ps	35
Cumulative error across 4 cycles	tERR(4per)	- 250	250	-200	200	ps	35
Cumulative error across 5 cycles	tERR(5per)	- 250	250	-200	200	ps	35
Cumulative error across n cycles, n = 6 ... 10, inclusive	tERR(6-10per)	-350	350	-300	300	ps	35
Cumulative error across n cycles, n = 11 ... 50, inclusive	tERR(11-50per)	-450	450	-450	450	ps	35
Duty cycle jitter	tJIT(duty)	- 125	125	-100	100	ps	35

Fig. 6. Jitter accumulation error specification

Now, we can specify the property that defines the acceptable accumulation error over n clock periods (we set $n = 3$ for this example).

We first need to detect clock periods and we use the property below in order to achieve this goal. The `rise` operator is needed in order to consider only differential crossings of CK and CKB when CK is rising and CKB falling, as shown in Figure 8. We use the STL/PSL `define` construct to declare a Boolean signal (as a variable with a name) that corresponds to a temporal property. The defined signal can be reused in other properties as a variable.

```
define clk_period_start := rise (CK - CKB >= 0);
```

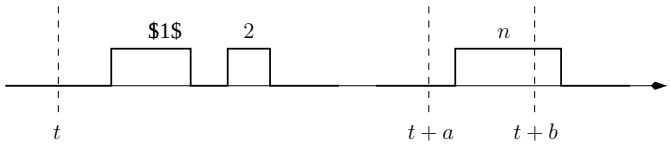

Fig. 7. Next rise operator

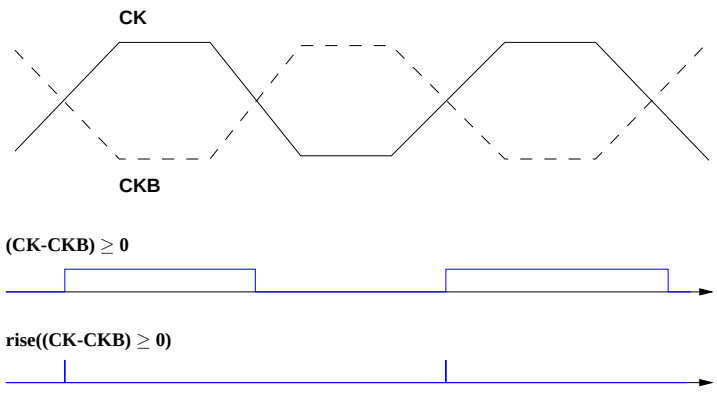

Fig. 8. Detection of the rising edge of the differential clock period

The property in STL/PSL that specifies the allowed accumulation error over 3 cycles is expressed as follows

```
always (clk_period_start ->
  next_rise[3][3*tCK(avg)-175:3*tCK(avg)+175]
    clk_period_start);
```

The average clock period $tCK(avg)$ varies in time and in STL/PSL we can currently define only fixed time bounds. In order to resolve this problem, we had to use the existing measures of the minimum and maximum average differential clock periods for the given simulation traces, obtaining the values $tCK(avg)_{min} = 1876ps$ and $tCK(avg)_{max} = 1877ps$. We used these values in order to determine fixed time bounds in a conservative way $[3 * tCK(avg)_{max} - 175 : [3 * tCK(avg)_{min} + 175] = [5456 : 5803]$. Finally, we could write the following property for the differential clock accumulation error over 3 cycles.

```
always (clk_period_start ->
  next_rise[3][5456:5803] clk_period_start);
```

4 Experimental Results

In this case study, we considered a single-ended DDR2-1066 memory interface, which is not yet a JEDEC standard. Hence the exact specification parameters could not be obtained for that particular version of the DDR2 memory, and we used instead the official specification parameters for the single-ended DDR2-400 presented in Section 3, assuming that these parameters would be conservative enough. The simulation traces (see Figure 9) contained about 180,000 samples for each signal.

For the case study evaluation, we used the AMT stand-alone tool. AMT takes as input an STL/PSL specification and analog/mixed signal traces. The tool translates the specification into an interpreted program (see [NM07,N08] for a presentation of translation

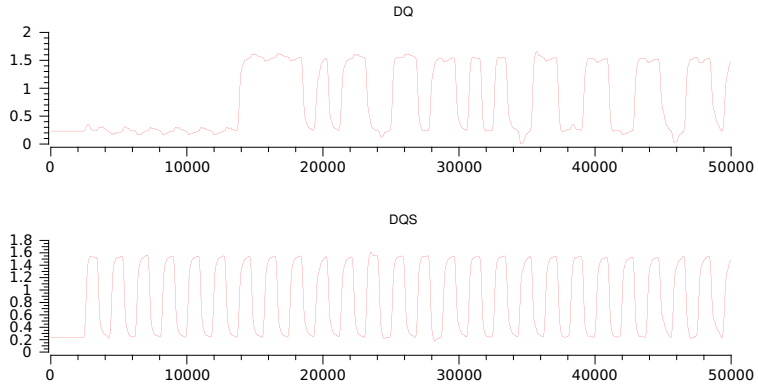

Fig. 9. Fragment of a data DQ and data strobe DQS simulation trace

algorithms) that checks whether the assertions are satisfied with respect to the simulation waveforms. The architecture of AMT is shown in Figure 10. The tool offers two evaluation modes, *offline*, where the input traces are validated after the simulation, and *incremental*, where the property evaluation can run in parallel with the simulation via a communication through a TCP/IP connection and try to early determine the satisfaction of the formula[3]. We used the offline mode for this case study because the DDR2 simulation traces were already available.

4.1 Methodological Evaluation

Property-based monitoring of analog and mixed-signal behaviors is a novel approach and it is worth discussing some methodological aspects related to this case study. The process started by investigating the validation methods that are currently used by analog designers and understanding what are the actual difficulties that they encounter in checking the correctness of their designs. The next step required to identify the type of application whose validation is not fully covered by existing tools and that could benefit from assertion-based monitoring techniques, which led us to consider the DDR2 memory interface. With the help of analog designers we were able to study in detail different properties that are defined in the official DDR2 specification, and consequently understand how to translate them into STL/PSL assertions. This preparation process of the case study is difficult to quantify although it clearly took orders of magnitude more time than the actual writing and evaluation of the assertions that describe DDR2 properties. Despite the length of this pre-processing, it was a crucial step in understanding relevance, strengths and weaknesses of the property-based analog monitoring framework.

[3] Relative time and memory requirements are compared and analyzed in [NM07]

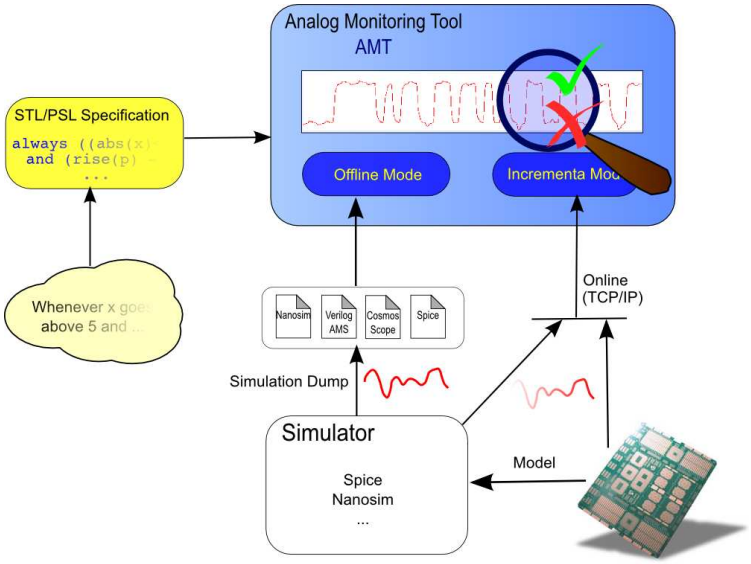

Fig. 10. Architecture of STL/PSL Property Checkers

4.2 Experimental Evaluation

The translation of the alignment property into a set of STL/PSL assertions started by splitting the main property into 4 different ranges, taking an over-approximated tDS value for each slew rate range. The evaluation of each property took about 7 seconds. Since some of the over-approximating properties were shown to be false, we decomposed them further through 3 iterations into a total of 7 properties before being able to show that the simulation traces satisfy the specification. The properties were refined manually and this proved to be a tedious task.

The jitter property was evaluated with the accumulation error specified over a varying number of clock periods. Table 4.2 shows the time required for the evaluation of the property wit respect to the different numbers of clock periods considered.

n	time (s)
1	1.8
2	1.8
5	1.9
10	2.1
20	2.6
50	4.8

Table 3. Evaluation time for the jitter property - n is the number of consecutive clock periods

5 Future Work and Conclusion

The DDR2 case study presents, to the best of our knowledge, a first attempt to apply assertion-based verification framework to a realistic industrial example in a systematic way. The importance of this case study lies in the fact that it exposes the relevance and the level of maturity of assertion-based methodology in the context of analog validation.

The case study showed that an important class of non-trivial properties describe event-based timing relationships between analog signals, which can be in general naturally expressed in a specification language such as STL/PSL. Since assertion checking remains a "lightweight" simulation-based validation technique, it fits well with the current practice of analog designers. We believe that this methodology can provide an extra set of useful checks on simulation traces, which are already generated by the designers for their own purposes. Moreover, in the analog domain it often takes orders of magnitude longer to produce simulation traces than to check assertions. Consequently, the overhead induced by property monitors with respect to simulation time remains low, while it can provide another level of confidence in the correct functioning of the underlying design. In our opinion, the general idea of simulation-based checking of properties to find potential bugs may be successfully adapted from digital to analog and mixed-signal domain and integrated into the analog validation flow in a reasonably-near future.

The DDR2 case study also revealed some weaknesses in the current state of analog property checking, providing useful guidelines for further development and optimization of this methodology. For instance, the timing relationship between analog signals can be more complex than what STL/PSL (and MITL) can express. This problem has been exposed by both the DDR2 data vs. data strobe alignment and the jitter properties. For the former, we had to use approximate techniques in order to show that the alignment between data and data strobe signals was correct. Consequently, the resulting specification turned out to be quite complex to write. The jitter property required introducing a novel temporal operator that allows to reason about the relative timing distance between n consecutive events. Another difficulty is related to the fact that STL/PSL is based on a temporal logic, a formalism that remains esoteric to analog designers[4]. Consequently, we should consider identifying some common properties encountered by analog designers, and use parameterized templates to "hide out" the temporal logic details.

We present here some directions for future work based on different observations made during the evaluation of the case study:

Parameterized time bounds: the DDR2 case study exposed that STL/PSL temporal operators with constant time bounds may not be sufficient to describe some realistic relations between analog signals. The temporal relations between "events" in input signals require more flexibility, such as time bounds that are functions of parameters that vary during the simulation.

[4] It might be the case that the verification task will be carried out by digital designers at the system integration phase, which will make the "cultural" problems less severe. However, this observation opens the question of what properties are most beneficial to integration within the property-based monitoring approach.

Tighter integration with simulators: property-based analog checking approach would be more appealing to designers if the specification and monitoring process were embedded in the standard design languages and simulators. In the digital world, the assertions are often integrated into Verilog or VHDL code and are inserted at the points where the property should be checked. A tighter integration of analog and mixed signal assertions into the current design flow would consist of the following steps:

1. Standardization of the language, a step that could convince EDA companies to consider integrating assertion-based AMS validation methodology into their tools, and would encourage designers to use such assertions in their designs. STL/PSL follows this direction as it extends the existing standard PSL constructs. Due to the importance of the SVA specification language in the digital domain, we would also need to consider analog and mixed-signal extensions of SVA.
2. Integration of assertions into VERILOG-AMS and VHDL-AMS code. Designers prefer inserting assertions at the points in their design which they want to check, than having a separate tool rather used solely for specification and evaluation of the properties. This tight integration would bring other benefits, such as the possibility to use existing VERILOG/VHDL-AMS constructs within the assertions (better detection of threshold crossing using `@cross`, express richer properties using derivatives and integrals, etc.). Finally, property monitors would be embedded into the simulation process, and could stop it when an assertion is violated and hence save simulation time.

Automatic parameter extraction: the interaction with analog designers revealed that the verification with respect to the existing specification is not the only interesting question that can be asked about an analog design. In fact, the specification parameters such as time relationship between different signals are often not known in advance. Such parameters are rather extracted from the simulation traces, and the specification is completed only after simulating a model of the design. We would like to express properties without specifying the time bounds, for example `always (rise(b:p) -> eventually![?] b:q)`, asking the following question: given a set of simulation traces, what are the minimum and maximum time bounds, if any, such that the the property is satisfied. In formal methods community, this problem is known as model measuring, and has been considered in the context of parametric temporal logics in [AELP99].

Integration with test generation: an interesting direction of research would be to combine the property-based analog checkers approach with techniques for automatic generation of simulation traces, such as those studied in [ND07a,ND07b]. The combined simulation generation and checking flow could make the analog validation more automatic.

More comprehensive examples: the case study carried out in this paper pointed out the classes of analog properties that are natural to express in a specification language like STL/PSL, but more importantly helped us to identify possible extensions of the language that would increase its expressiveness and make the specification process easier to the analog designer. Applying the property-based validation methodology to other industrial analog and mixed-signal design examples

would provide additional useful information about the robustness of this approach and guide our future work on extending the specification language.

Acknowledgments We would like to thank Tom Giovannini from Rambus, Inc. for his detailed explanations of the DDR2 specification and for providing us with simulation traces. We would also like to thank Oded Maler from Verimag for discussions on the STL/PSL language and its extensions.

References

[ABG+00] Y. Abarbanel, I. Beer, L. Glushovsky, S. Keidar, and Y. Wolfsthal. FoCs: Automatic Generation of Simulation Checkers from Formal Specifications. In *Proc. CAV'00*, pages 538–542. LNCS 1855, Springer, 2000.

[Acc04] Accelera Standard, SystemVerilog 3.1a Language Reference Manual

[ADF+06] E. Asarin, T. Dang, G. Frehse, A. Girard, C. Le Guernic and O. Maler, Recent Progress in Continuous and Hybrid Reachability Analysis, *CACSD*, 2006.

[AELP99] R. Alur, K. Etessami, S. La Torre and D. Peled, Parametric Temporal Logic for "Model Measuring" *ICALP'99*, 159–168, 1999.

[AFH96] R. Alur, T. Feder, and T.A. Henzinger, The Benefits of Relaxing Punctuality, *Journal of the ACM* **43**, 116–146, 1996.

[AZDT07] G. Al Sammane, M.H. Zaki, Z.J. Dong and S. Tahar, Towards Assertion Based Verification of Analog and Mixed Signal Designs Using PSL, *FDL'07*, 2007.

[DC05] T.R. Dastidar and P.P Chakrabarti, Verification System for Transient Response of Analog Circuits Using Model Checking, *VLSID'05*, 195–200, 2005.

[Dru00] D. Drusinsky. The Temporal Rover and the ATG Rover. In *Proc. SPIN'00*, pages 323–330. LNCS 1885, Springer, 2000.

[FGP06] G. Fainekos, A. Girard and G. Pappas Temporal Logic Verification Using Simulation In *Proc. FORMATS'06*, pages 171–186. LNCS 4202, Springer, 2006.

[GPVW95] R. Gerth, D.A. Peled, M.Y. Vardi and P. Wolper, Simple On-the-fly Automatic Verification of Linear Temporal Logic, *PSTV*, 3–18, 1995.

[GO01] P. Gastin and D. Oddoux, Fast LTL to Büchi Automata Translation, *CAV'01*, 53–65, LNCS 2102, 2001.

[HFE04] J. Havlicek, D. Fisman and C. Eisner, Basic results on the semantics of Accellera PSL 1.1 foundation language, *Technical Report 2004.02*, Accelera, 2004.

[HR01] K. Havelund and G. Rosu. Java PathExplorer - a Runtime Verification Tool. In *Proc. ISAIRAS'01*, 2001.

[Jed06] JEDEC Standard, JESD79-2C DDR2 SRAM Specification

[JHP+07] A. Jesser, S. Lämmermann, A. Pacholik, R. Weiss, J. Ruf, W. Fengler, L. Hedrich, T. Kropf and W. Rosenstiel, Analog Simulation Meets Digital Verification - A Formal Assertion Approach for Mixed-Signal Verification SASIMI'07, 507–514, 2007.

[KLS+02] M. Kim, I. Lee, U. Sammapun, J. Shin, and O. Sokolsky. Monitoring, Checking, and Steering of Real-time Systems. In *Proc. RV'02*. ENTCS 70(4), 2002.

[MMP92] O. Maler, Z. Manna and A. Pnueli, From Timed to Hybrid Systems *Real-Time: Theory in Practice*, 447-484, LNCS 600, 1992.

[MN04] O. Maler and D. Nickovic Monitoring Temporal Properties of Continuous Signal *FORMATS/FTRTFT'04*, 152–166, 2004.

[MP95] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer, 1995.

[N08] D. Nickovic, *Checking Timed and Hybrid Properties: Theory and Practice*, PhD. Thesis, 2008.

[ND07a] T. Nahhal and T. Dang, Test Coverage for Continuous and Hybrid Systems *CAV'07*, 449–462, 2007.

[ND07b] T. Nahhal and T. Dang, Guided Randomized Simulation *HSCC'07*, 731–735, 2007.

[NM07] D. Nickovic and O. Maler, AMT: A Property-Based Monitorting Tool for Analog Systems *FORMATS'07*, 304–319, 2007

[SB00] F. Somenzi and R. Bloem, Efficient Büchi automata from LTL formulae, *CAV'00*, 248–263, LNCS 1855, 2000. 1855.

[VW86] M.Y. Vardi and P. Wolper, An Automata-theoretic Approach to Automatic Program Verification, *LICS'86*, 322–331, 1986.