Proven correct monitors from PSL specifications

Katell Morin-Allory, Dominique Borrione
Tima Laboratory, 46 avenue Félix Viallet 38031 Grenoble Cedex, France
katell.morin@imag.fr, dominique.borrione@imag.fr

Abstract

We developed an original method to synthesize monitors from declarative specifications written in the PSL standard. Monitors observe sequences of values on their input signals, and check their conformance to a specified temporal expression. Our method implements both the weak and strong versions of PSL FL operators, and has been proven correct using the PVS theorem prover. This paper discusses the salient aspects of the proof of our prototype implementation for on-line design verification

1. Introduction

The design of VLSI Systems on a Chip increasingly involves the use and interconnection of large pre-existing components that are provided and validated by separate groups of engineers. In this context, assertion-based design [10] can be viewed as a unifying methodology across design teams and design description levels [12].

Assertions state functional and temporal properties about a component's interface elements and/or state variables. Written in declarative form, they can be adapted to the syntax of various design description languages without any impact on their semantics (this is the approch taken by Accelera with PSL) [3]. The advantage is that asserted properties can be carried along the design steps, and serve a wide range of usages: specify the constaints for correctly using an IP, specify the result delivered by an IP, specify the expected correct behavior from the design at hand, generate and control test data, etc.

As a design property that is declared to be true, an assertion can be evaluated by one or more techniques among simulation, emulation or formal verification. An assertion can also be seen as a high level functional specification for a circuit primarily intended for snooping on events over time. Whether intended for design validation or for online embedded testing, a common technique can be applied: synthesize from the assertion a *property monitor* under the form of a RTL sequential circuit, and interconnect the design and the monitor via the monitored variables. This paper describes our method for providing efficient and proven correct property monitors for a formally defined yet user-understandable assertion language.

Our mechanism applies to a subset of PSL very close to what is referred to as its "simple subset" [3]. Intuitively, we only accept properties that can be evaluated "on the fly" by fixed size monitors during simulation or execution. A formal characterization of the largest PSL subset that satisfies this constraint still remains to be done. We initially selected this language for its clean and formal trace semantics. The method is based on a library of primitive digital components, and a technique to interconnect them. The resulting digital module can be properly connected to the signals of interest in the system under scrutiny, it runs concurrently with it, and notifies its environment when the property checking is terminated with a value true or false, or whether the property is still being evaluated. Both the weak and strong version of the PSL operators are covered. This paper discusses how we proved the correctness of monitors. To the knowledge of the authors, the implementation and the proof of a solution for the strong version of the PSL operators has not been published before.

Related works PSL originates from the Sugar input language to the RuleBase [6] symbolic model checker from IBM. M. Gordon played an essential role in the validation of the semantic definition. He performed a "deep embedding" of PSL in the HOL proof assistant, and used this mechanized system to demonstrate theorems about the semantics, and derive correct-by-construction mathematical observers for PSL properties [11]. Building on this seminal work, Türk translated a small unclocked subset of PSL FL expressions to LTL, still using HOL [15]; the interest of this work is only theoretical, as the resulting automata suffer from combinational explosion. Claessen and Martensson proposed an operational semantic definition guided by the structure of the PSL formula [8]; they exhibited some inconsistencies in the interpretation of a special class of regular expressions, and their work was useful for understanding. The published work only covered weak operators.

In automata-theoretic approaches, the transformation of PSL assertions to automata is exponential in the number of operators. Hardware design approachs are more usable in practice, as they construct a sequential machine whose memories encode rather than enumerate its reachable states. Using IBM's FoCs [2] triggered our interest in the generation of monitors. From a PSL assertion, FoCs produces a source HDL process that acts as an internal monitor to the design under verification (DUV); the presence of error reporting statements makes FoCs output better fit for verification than for emulation and synthesis. Safelogic Verifier (now part of JasperGold) was another formal verifi-

3-9810801-0-6/DATE06 © 2006 EDAA

cation tool taking a large subset of PSL as input. In both cases, weak operators only are recognized, and the formal method underlying the model extraction from PSL was not published.

Many more products are publicized to support PSL for model checking, simulation waveform generation, emulation and debug; many plug-in products add assertion support to pre-existing verification tools (see [1,5,9] for a list). To our knowledge, in most tools, the coverage of PSL primitives is very restricted, and the majority of companies advertize no formal guarantee that the PSL semantics are correctly supported. In contrast, this paper discloses a proven correct technology.

2. Principles of the FL Monitors Construction

In the PSL terminology, a property satisfaction level, on a finite execution path, may be one of:

- Holds strongly: No bad states have been seen. All future obligations have been met. The property will hold on any extension of the path
- Holds (but does not hold strongly): Same as above except that the property may or may not hold on any given extension of the path
- Pending: No bad states have been seen. Future obligations have not been met. The property may or may not hold on any given extension of the path
- Fail: A bad state has been seen. The property will not hold on any extension of the path.

The main temporal operators of the "foundation language" (FL operators) have a strong and weak version. Intuitively, a property built on weak operators that is still pending at the end of a finite trace is considered satisfied. Conversely a terminating condition should have happened before the end of the trace (the property should at least hold) for strong operators.

The monitors we build reflect these definitions. When implemented in hardware, the monitor outputs display the property satisfaction level, and the indication that the answer is no longer pending may be used as an interrupt to trigger further actions. For technical reasons, the validity result is latched, and thus available on the output one clock cycle after is it known. This small delay is acceptable for all practical purposes. Note that a property satisfaction is not binary, and the negation of a strong operator is weak. Consequently, FL expressions may not be negated or "ored" in our subset (but they may be "anded").

A monitor for a property P is built as a module that takes as inputs the reset, the synchronization signals (clock, handshake, etc.), a signal *Start* that triggers the evaluation, and the signals of the DUV that are operands of the FL operators in P (see Figure 4). The three monitor outputs have the following significance:

- *Checking*: a 1 indicates that output *Valid* is effective at the next synchronization time;
- *Valid*: provides the evaluation result (1 means absence of error, 0 means error);
- *Pending*: a 1 indicates that the monitor has been started and that the satisfaction result is pending; this is significant for strong operators.

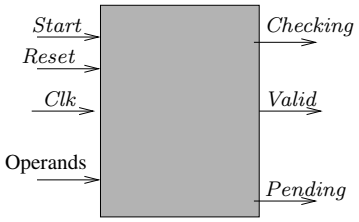

Figure 1. Interface of a monitor

The "weak" version of the PSL operators are a simplified subcase of the "strong" version, so we shall only discuss the construction and the proof of monitors for the strong operators. In the following, we use the VHDL flavor of PSL, but the principles apply to other syntax as well. In the text and the figures, 0 and 1 are used both for bits '0' and '1', and for Booleans False and True.

Example 1 *Assume we are interested in the behavior of signals A, B, C in a design synchronized by the rising edges of its master clock Clk. We want to observe that signals A, B, C satisfy property P, expressed as:*

```
Property P is always (A -> next![2] (B before! C))
        @ rising_edge(clk);
```

P is an invariant, which states that whenever A is 1, starting from two cycles later, it must be the case that B takes the value 1 before C takes the value 1.

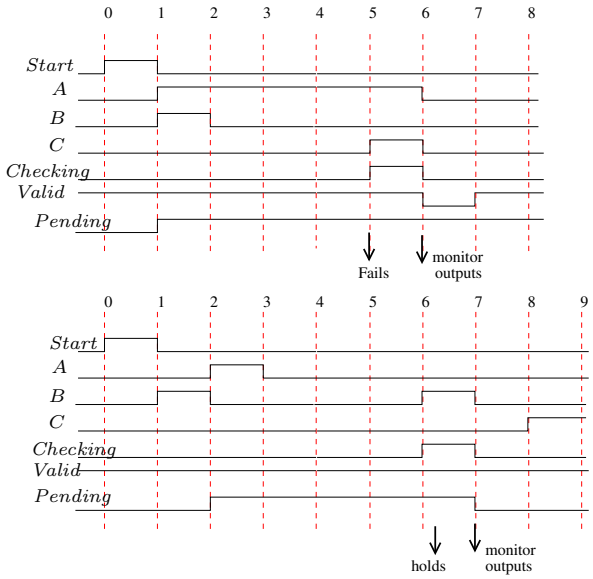

Figure 2. Waveforms for signals A, B, C

Figure 2 gives two possible sequences of values observed on signals A, B, C. The vertical dotted lines represent the successive rising edges of Clk, which have been numbered for the purpose of this explanation. Start, Checking, Valid

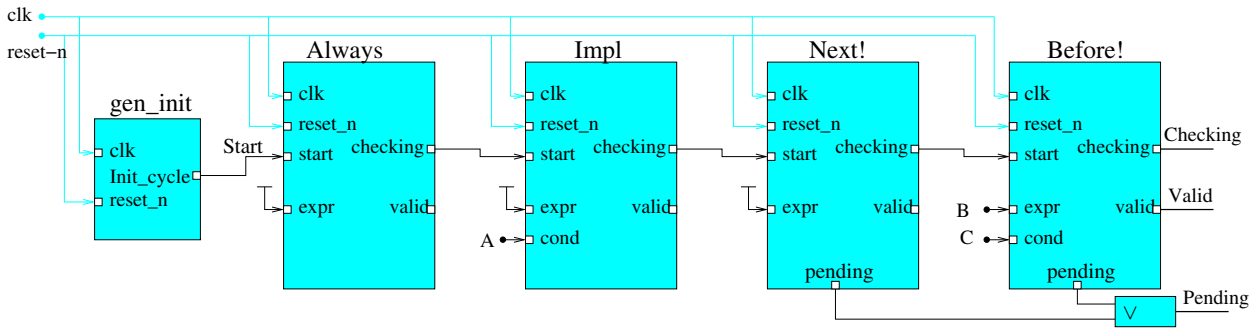

Figure 3. Property monitor for P

and Pending are the input-outputs of the monitor for P, as explained above.

Top waveform: At Clk edge ♯1, A takes value 1, and remains 1 until edge ♯6. Thus, property P holds (on the finite trace ending edge ♯8) if starting from edge ♯3, B eventually holds and takes value 1 before C. P fails at edge ♯5. According to the design of our monitors, output Checking takes value 1 at this same edge, and the value of output Valid is significant one cycle later.

Bottom waveform: At Clk edge ♯2, A takes value 1, and remains 1 for one clock cycle. Thus, P holds if starting at Clk edge ♯4, B eventually holds and takes value 1 before C. On the waveform, P holds at Clk edge ♯6, and the outputs Valid and Pending take the values 1 and 0, meaning that the property holds. Note that if the waveform had stopped at Clk edge ♯5, Pending would still have been 1, and property P would not have hold on the trace, in accordance to the fact that the first operand of before! *would have remained 0.*

Figure 3 shows the monitor for P. The overall monitor takes as inputs the master Clk and $Reset\_n$ signals, and the observed signals A, B, C. It is built as the structural interconnection of primitive monitors, one for each temporal operator in property P: always, impl (for '$\rightarrow$'), next!, before!. An additional primitive module called gen_init starts the global monitoring process.

Structure of a primitive monitor One primitive monitor has been hand crafted for each FL operator of PSL. Operators that take one or two integer parameters, such as next or next_a, have corresponding generic monitors with the same parameters. All primitive monitors share a common basic structure, in which two main blocks can be identified.

- The *Checking Window Block* generates the temporal window for the evaluation of the operands, and sets output $Pending$ and an internal $Check\_en$ signal based on the evaluation requirement ($Start$ input signal) and the semantics of the operator. A shift register is included for the operators that allow an overlap of evaluation windows.
- The *Evaluation Block* checks the operands when $Check\_en$ is 1, and output $Valid$ represents the result. When $Check\_en$ is 0, execution is stopped, and output $Valid$ stays in its default value 1. When reset is active, the monitor stays in its reset state.

Construction of complex monitors by interconnection of primitive components For complex properties, primitive single operator monitors are interconnected to construct a complex monitor. The method is based on the syntax tree of the property, where a node represents a PSL operator, a leaf represents a basic operand (signal or constant value), the edges connect an operator with its operands. Some operators have two operands, some have only one.

- *Operators:* For each node in the tree structure of a property, a corresponding operator monitor is needed.
- *Operands of Boolean and Temporal operators:* The result of a simple Boolean expression (not a, a or b,...) is directly connected to the corresponding operand input of the operator monitor. If the operand is a FL formula composed of other temporal operators, connect value 1 to the corresponding input of the monitor to disable the evaluation function within the monitor (it is done by other monitors).
- *Connection of consecutive FL operators:* For two operators N1 and N2 such that N2 is an operand of N1, the two monitors are connected in the following way (Fig. 3).
  - Output *Checking* of N1 is fed to input "start" of N2.
  - Output *Valid* of N1 is useless, and unconnected.
  - The clock and reset are shared by N1 and N2.
- *Initial State Generation:* A module "gen_init" is added to generate the initial state signal "init_cycle" one cycle after power up reset. This signal triggers the evaluation of the whole property, and is fed into input *start* of the root monitor.
- *Primary outputs:* *Checking* and *Valid* are those of the rightmost FL operator, *Pending* is the "or" of the *Pending* outputs of all the "strong" monitors.

In terms of post synthesis area, this solution is efficient for complex temporal properties [7], a significant advantage for online monitoring of embedded systems, specially on FPGA.

3. Principles of the proof of Correctness

The proof of correctness of both the library components and the interconnection method, with respect to the formal semantics of the PSL reference manual was done with the

PVS [14] system. PVS provides an integrated environment for the development and analysis of formal verification. It consists of a specification language, a number of predefined theories, a theorem prover. It is based on a higher order typed logic. The choice of PVS was motivated by the fact that the PSL semantics are expressed in second-order logic and thus directly represented by the PVS input formalism. In addition, many proof strategies are automated.

3.1. Monitors modeling

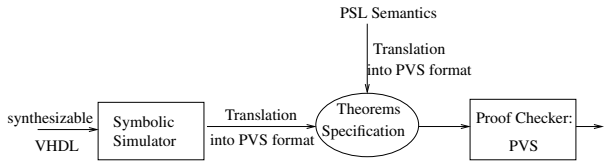

Figure 4. Proof process

The primitive monitors are written in the RTL synthesizable subset of VHDL. To prove each monitor correct, we first extract the finite state machine (FSM) model for the component, and translate it in the PVS input formalism. This process has been automated, with the help of symbolic simulation (see Figure 4).

We use THEOSIM [4] that takes as input a clock synchronized sequential circuit, and computes the state transition functions and output functions in a normalized conditional format. This tool performs a static stabilization of combinational circuits between clock edges (provided there is no combinational loop). This simulator defines the symbolic value of a signal as a function of the previous symbolic value of all signals of its cone of influence. Since bit-vectors are handled globally (*i.e.* represented by only one symbol), we can handle parameterized bit-vectors.

The translation of the symbolic expressions computed by the symbolic simulator into PVS input syntax is fully automated. It is essentially a simple transcription, except that VHDL objects are considered functions in the PVS model, both in their structural and temporal dimensions. More precisely, signals are modelled by time functions: they are represented by a mapping from $\mathbb{N}$ to their type. Since vectors are modelled by functions too, a bit vector is a mapping from $\mathbb{N} \times [i, j]$ (where $i,\ j$ are the first and last index of the bit-vector) to Boolean.

Then, the PSL semantics are represented in PVS, and we generate automatically several theorems for each operator. All theorems are proved with PVS (Figure 4).

Example 2 *The following VHDL text (left of Figure 5) is an excerpt from a primitive monitor RTL description. The output Valid is combinationally connected to an internal signal Valid_t (first line). The rest of the code is a sequential process that computes Valid_t at each rising edge of Clk, as a function of signals Reset_n, Check_en and some expression expr. The evaluate_expr process and the assignment of Valid are concurrent, and require two simulation iterations to stabilize at each rising clock edge. The right part of Figure 5 gives the representation in PVS of signal Valid.*

```
valid <= valid_t;
evaluate_expr: process(clk)
begin
 if clk'event and clk='1'   then
  if reset_n='0'
  then valid_t <= '1';
  else
   if check_en = '1'
   then valid_t <= expr;
   else valid_t <= '1';
   end if;
  end if;
 end if;
end process;
```

```
VALID(t:nat): boolean =
(IF t=0
THEN TRUE
ELSE
  IF NOT RESET_N_(t-1)
  THEN True
  ELSE
   IF CHECK_EN(t-1)
   THEN EXPR_(t-1)
   ELSE True
   ENDIF
  ENDIF
ENDIF)
```

Figure 5. VHDL excerpt and translation into PVS

3.2. Modeling the PSL semantics in PVS

The semantics of PSL are defined on the traces of all the observed signals on a time range $[t_0, T]$. Let $\mathcal{FL}$ denote the set of FL expressions. Semantics are modelled by a mapping Sem from $\mathcal{FL} \times \mathbb{N} \times \mathbb{N}$ to Boolean. Let φ be a FL expression, then $\mathrm{Sem}(\varphi, t_0, T)$ is inductively defined on the syntactic tree of φ on the time range $[t_0, T]$: for each operator Ω, we define a function Sem_Ω implementing the semantics of Ω and depending on function Sem. Functions Sem and Sem_Ω are mutually dependent. When φ is only a Boolean, $\mathrm{Sem}(\varphi, t_0, T)$ is defined by the value of φ at t_0.

Example 3 (Semantic function of $\mathrm{next\_e}$**)** *The* $\mathrm{next!}[i]$ *operator is a strong operator. It specifies that there is a next cycle, i.e., that the length of the trace on which the signal are observed is greater than* $i + 1$*, and that the property holds on cycle* $i + 1$.

The modeling of this operator semantics in PVS gives us:

$$\mathrm{Sem}_{\mathrm{next}}(\varphi, k, t_0, T) = \begin{cases} (T - t0) >= 0 \wedge \mathrm{Sem}(\varphi, t0, T) & \text{if } k = 0 \\ \mathrm{Sem}_{\mathrm{next}}(\varphi, k-1, t_0 + 1, T) & \text{if } k >= 1 \end{cases}$$

The $\mathrm{next\_e!}[i, j](\varphi)$ *operator specify that there is at least one cycle within the subrange* $[i, j]$ *of next cycles on which* φ *holds. This operator is based on a rewriting of the* next *operator. Its modeling in PVS is given by:*

$$\mathrm{Sem}_{\mathrm{next\_e}}(\varphi, i, j, t_0, T) = \exists k \in [i, j], \mathrm{Sem}_{\mathrm{next}}(\varphi, k, t_0, T)$$

3.3 Equivalence modeling

Let M be the monitor implementing a FL property φ. The equivalence deals on the one hand with $\mathrm{Sem}(\varphi, t_0, T)$, on the other hand with the output signals of M: $Checking_M$ and $Valid_M$ (relevant one cycle after $Checking_M$ takes value 1) and $Pending_M$. More formally, the equivalence can be modelled by the following expression:

$$\begin{array}{c} \forall \varphi, \forall t_0 \in \mathbb{N}, \forall T \geq t_0, H(\varphi, t_0, T) \implies \\ (S(\varphi, t_0, T) \iff (V(\varphi, t_0, T) \wedge P(\varphi, t_0, T)) \end{array} \quad (1)$$

where $S(\varphi, t_0, T)$ denotes an expression dealing with Sem, $V(\varphi, t_0, T)$ an expression dealing with $Valid_M$

and $Checking_M$, $P(\varphi, t_0, T)$ an expression dealing with $Pending_M$ and $H(\varphi, t_0, T)$, the hypothesis under which the equivalence can be proved.

Let us define formally these four expressions. The value of the semantics of expression φ is relevant only if $Start_M$ was true at time t_0. As a first approximation, assume that $S(\varphi, t_0, T)$ is defined by $Start_M(t_0) \implies \mathrm{Sem}(\varphi, t_0, T)$.

Expression of V Expression V specifies that no bad states have been seen. By definition of monitors, the relevance of $Valid_M$ is given by an active $Checking_M$ at the previous cycle. Since $Checking_M$ just represents the signal computed by the checking window, and since $Valid_M$ is defined by $True$ when this signal is not active, for all cycles t and for all monitors $Checking_M(t) \implies Valid_M(t+1)$ is equivalent to $Valid_M(t+1)$. This property is generated and verified for each monitor in PVS.

If $S(\varphi, t_0, T)$ is verified, signal $Valid_M$ must be verified during some cycles. But we cannot know *a priori* when $Checking_M$ (and therefore $Valid_M$) is active. For some operators, it depends only on a parameter (*e.g.* next with delay), for others it depends on the activity of an event (*e.g.* next_event). To unify our modeling on all properties, we look at $Valid$ on the whole time range $[t_0, T]$.

$$V(\varphi, t_0, T) = \forall t \in [t_0, T],\ Valid_M(t+1) \tag{2}$$

Expression of P Signal $Pending$ is 1 when some future obligations have not been met. If Property φ should hold (or hold strongly) then at the end of the trace, all the future obligations must have been met. Since the computation of $Pending$ is delayed by one cycle, $P(\varphi, t_0, T)$ is defined by:

$$P(\varphi, t_0, T) = \quad \neg Pending(T+1) \tag{3}$$

Expression of S On $[t_0, T]$, $Valid_M$ can depend on several active $Start_M$. In our first approximation of $S(\varphi, t_0, T)$, the semantics took into account only one active $Start_M$ at t_0. We generalize this definition to the occurrence of several active $Start_M$ on the studied range. We thus give the following definition:

$$\begin{array}{rl} S(\varphi, t_0, T) = & \forall t \in [t_0, T+1], \\ & Start_M(t) \implies \mathrm{Sem}(\varphi, t, T) \end{array} \tag{4}$$

Expression of H The two previous paragraphs give us the expression of the equivalence between the operator semantics and the monitor output valuations. Let us now examine the hypothesis under which this equivalence is verified.

- Signals $Checking$ and $Valid$ are active only when signal $Reset_M$ is not active, *i.e.* $Reset_M$ is 1. We need $Reset_M$ to be not active on $[t_0, T]$.
- The second hypothesis deals with $Start$. On $[t_0, T]$, $Valid_M$ might depend on an active $Start_M$ occurring before t_0; in that case, the equivalence would not be verified. To exclude this irrelevant case, we may either assume that $Start_M$ was never active before t_0 (this is too strong an assumption), or assume that $Start_M$ was never active since the last active $Reset_M$. We use the second assumption.

These two hypotheses give us:

$$\begin{array}{rcl} H(\varphi, t_0, T) & = & \exists t' \in [0, t_0], \neg Reset_M(t') \\ & \wedge & \forall t_1 \in [t', T], Reset_M(t_1) \\ & \wedge & \forall t_2 \in [t'+1, t_0], \neg Start_M(t_2) \end{array} \tag{5}$$

3.4. Proof of Correctness

In this section, we break up the main lines of the proof of equivalence between the PSL formal operator semantics and the results delivered by their associated monitor. A complete proof can be found in [13]. It is is done by induction on the depth of Formula (1), whatever FL operator is considered. Let $\Omega_1, \ldots, \Omega_n$ be n FL operators, and $\varphi_n = \Omega_n \ldots \Omega_1 \mathrm{op}_1 \ldots \mathrm{op}_n$ a FL expression where op_i is a list of operands for Ω_i. Parameter n represents the depth of the formula (denoted $|\varphi_n|$).

Let M be the monitor implementing φ_n. It is composed of n basic monitors $M_1, \ldots M_n$(*cf.* Fig. 3). We denote $Start_i$, $Checking_i$, $Valid_i$ and $Pending_i$ the input and outputs of M_i. Input $Start_M$ is input $Start_n$. Similarly output $Valid_M$ is output $Valid_1$. The $Reset$ input of all monitors are connected to the global $Reset_M$ input. In Formula (1), the terms $H(\varphi_n, t_0, T)$, $S(\varphi_n, t_0, T)$, $V(\varphi_n, t_0, T)$ and $P(\varphi_n, t_0, T)$ can be rewritten into:

$$\begin{array}{rcl} H_n(\varphi_n, t_0, T) = & \exists t' \in [0, t_0], \neg Reset_M(t') \\ & \wedge & \forall t_1 \in [t', T], Reset_M(t_1) \\ & \wedge & \forall t_2 \in [t'+1, t_0 - 1], \neg Start_n(t_2) \\ V_n(\varphi_n, t_0, T) = & \forall t \in [t_0, T], Valid_1(t+1) \\ S_n(\varphi_n, t_0, T) = & \forall t \in [t_0, T+1], \\ & Start_n(t) \implies \mathrm{Sem}(\varphi_n, t, T) \\ P_n(\varphi_n, t_0, T) = & \forall t \in [t_0, T], P_{n-1}(\varphi_{n-1}, t_0, T) \\ & \wedge & \neg Pending_n(T+1) \end{array}$$

Our induction hypothesis is given by:

$$\mathcal{I}(n) = \left\{ \begin{array}{l} \forall |\varphi_n| = n, \forall t_0 \in \mathbb{N}, \forall T \geq t_0, H(\varphi_n, t_0, T) \implies \\ (S(\varphi_n, t_0, T) \iff (V(\varphi_n, t_0, T) \wedge P(\varphi_n, t_0, T))) \end{array} \right.$$

Base Case: The first step is the proof of $\mathcal{I}(1)$. The expression φ_1 has only one operator, the operands of which are Boolean: we deal with a primitive monitor. Since the induction is done whatever φ_n, we must prove $\mathcal{I}(1)$ for all FL operators. We thus generate and prove in PVS a theorem corresponding to expression $\mathcal{I}(1)$ for each operator and its primitive library monitor.

Inductive Case Assuming $\mathcal{I}(n-1)$, we want to prove $\mathcal{I}(n)$. The proof of this implication is based on two facts:

- First, due to the interconnection method, $Checking_n$ is equal to $Start_{n-1}$.
- Second, we can rewrite expression φ_n into:

$$\varphi_n = \Omega_n(\varphi_{n-1}, \mathrm{op}_n)$$

By definition of Sem, we have

$$\mathrm{Sem}(\varphi_n, t_0, T) = \mathrm{Sem}_{\Omega_n}(\varphi_{n-1}, \mathrm{op}_n, t_0, T)$$

In the function defining Sem_{Ω_n}, φ_{n-1} is always used as a parameter of the Sem function. Furthermore, operand φ_{n-1} of Sem_{Ω_n} can be a Boolean expression. It may be substituted by $\lambda t.\mathrm{Sem}(\varphi_{n-1}, t, T)$. This lambda term is a Boolean function, and is generalized to any Boolean function.

These two rewritings are the salient aspects of the inductive proof. For each operator, we simplify the proof in two equivalences, the conjunction of which is stronger than $\mathcal{I}(n-1) \implies \mathcal{I}(n)$.

3.5. Results

To gain experience, we first proved the weak version of the primitive monitor library, then extended the model to the strong version of the FL operators and performed the proof of the strong monitors. No errors were found in the weak library. For one strong operator, $\mathrm{next\_e}$, the base case proof lead to an error in the proof of $\mathcal{I}(1)$. Since the weak version of this monitor was correct, the error had to come from the implementation of the $Pending$ signal: this signal was not significant at cycle $T+1$ the end of the trace, but at $T+j-i$ (where j and i are parameters of this operator). This error was corrected and the new correctness proof succeeded.

For each operator, we defined a theory in which we specify several theorems. It represents approximately 270 lines of code. Then this theory is automatically type checked: this generates roughly 100 typed proof obligations. The length of the proof is quite variable. As an example, for $\mathrm{next\_a}$, we used 10 instructions: from the catch-all strategy "grind" that automatically completes a proof branch by installing rewritings and simplifications, to the "inst" that instantiates a quantified variable by a given expression or the use of an intermediate lemma. For $\mathrm{next\_event}$, we use a thousand instructions. For a family of operators, the intermediate lemmas and proof strategies could largely be reused.

4. Conclusion and current works

In this paper, we proved a methodology to generate monitors from PSL assertions. The technology is based on a library of primitive components that implement the weak and strong FL operators of PSL. Monitors for complex expressions are built by structurally interconnecting these basic blocks. Thanks to the use of theorem proving techniques, we obtain a proof that is independent of the valuation of the temporal parameters (number of cycles, or time interval) present in many operators.

A prototype implementation automatically synthesizes a monitor from a PSL property, as a RT-level VHDL component net-list. One or more monitors can be linked to the DUV, for formal verification, simulation, emulation, or online testing. Recent improvements to the prototype have included a user interface to help identify these interconnects, and the instrumentation of the synthesis on a FPGA platform with several software modules. The history of the observed signals can be saved, extracted, and viewed as waveforms, to help debug the DUV when a monitor outputs a property failure. Systematic experiments and measurements, reported in [7], show that the area of our monitors increases gracefully with the number of nested PSL operators, and the upper bound of the observation window of the (next, $\mathrm{next\_event}$) operators.

Future works in this area aim at applying the same principles on designs described at a more abstract level. The difficulty lies in correctly connecting synthesizable monitors with asynchronous communication mechanisms.

Acknowledgements The authors thank Y. Wolfsthal, E. Zarpas and G. Shapir from the IBM Haifa Research Laboratory for their interest and for giving them free access to RuleBase and FoCs. Miao Liu was the main author of the primitive monitors library.

References

[1] http://www.haifa.il.ibm.com/projects/verification/RB_Homepage/publications.html♯sugar.

[2] Y. Abarbanel, I. Beer, L. Glushovsky, S. Keidar, and Y. Wolfsthal. FoCs: Automatic Generation of Simulation Checkers from Formal Specifications. In *Computer Aided Verification*, LNCS. Springer-Verlag, 2000.

[3] Accellera. *Property Specification Language Reference Manual, Version 1.1*, 2004.

[4] G. Al Sammane. *Simulation symbolique des circuits décrits au niveau algorithmique*. PhD thesis, Université Joseph Fourier, July 18, 2005. (in French).

[5] A. K. B. Cohen, S. Venkataramanan. *Using PSL/Sugar for Formal and Dynamic Verification*. VhdlCohen Publishing, 2nd edition, 2004.

[6] S. Ben-David, C. Eisner, D. Geist, and Y. Wolfsthal. Model checking at IBM. *Form. Methods Syst. Des.*, 22(2):101–108, 2003.

[7] D. Borrione, M. Liu, P. Ostier, and L. Fesquet. PSL-based online monitoring of digital systems. In *Forum on specification & Design Languages (FDL'05)*, Sep 2005.

[8] K. Claessen and J. Mårtensson. An Operational Semantics for Weak PSL. In A. J. Hu and A. K. Martin, editors, *Formal Methods in Computer-Aided Design (FMCAD'04)*, volume 3312. LNCS, Springer, 2004.

[9] Formal Methods Group. Guide to Sugar Formal Specification language. IBM Haifa Research Laboratory, Nov. 2000. Version 1.3.1.

[10] H. Foster, A. Krolnik, and D. Lacey. *Assertion-Based Design*. Kluwer Academic Publishers, Jun. 2003.

[11] M. Gordon, J. Hurd, and K. Slind. Executing the formal semantics of the Accellera Property Specification Language by mechanised theorem proving. In D. Geist and E. Tronci, editors, *Correct Hardware Design and Verification Methods (CHARME'03)*, volume 2860. LNCS , Springer, Oct. 2003.

[12] E. Marschner, B. Deadman, and G. Martin. IP reuse hardening via embedded Sugar assertions. In *IP Based SoC Design 2002*, Oct. 2002.

[13] K. Morin-Allory and D. Borrione. A proof of correctness for the construction of property monitors. In *IEEE Intl. High Level Design Validation and Test Workshop*, Dec. 2005.

[14] N. Shankar, S. Owre, J. Rushby, and D. Stringer-Calvert. *PVS Prover Guide*. Computer Science Laboratory, SRI International, Menlo Park, CA, 2001.

[15] T. Türk and K. Schneider. From PSL to LTL: A formal validation in HOL. In J. Hurd and T. Melham, editors, *Theorem Proving in Higher Order Logics*, LNCS, Aug. 2005.