

STARI: A Case Study in Compositional and Hierarchical Timing Verification

Serdar Taşiran* Robert K. Brayton

Department of Electrical Engineering and Computer Sciences,
University of California at Berkeley.

Abstract. In [TAKB96], we investigated techniques for checking if one real-time system correctly implements another and developed theory for hierarchical proofs and assume-guarantee style reasoning. In this study, using the techniques of [TAKB96], we verify the correctness of the timing of the communication chip STARI.

1 Introduction

We describe the application of the techniques and tools described in [TAKB96] to the verification of the high-bandwidth communication chip, STARI [G93].

STARI (by Greenstreet, [G93, G96]) is a self-timed FIFO that interfaces a transmitter and a receiver that operate at the same clock frequency but may have some skew between their clock signals (Figure 1). STARI can compensate for large, time varying skews and makes high bandwidth synchronous operation possible by eliminating the need for handshakes between the transmitter and the receiver. However, because there are no handshakes, certain timing properties need to be verified to show that the interface functions correctly. In particular, it needs to be shown that no data is duplicated or dropped by the interface.

The FIFO in STARI consists of a cascade of identical stages and thus, the complexity of automatically verifying a monolithic model of an n stage STARI circuit is roughly $O(k^n)$, where k is the size of the model for a single stage. If the circuit is modeled at the gate level, k is rather large, and this limits automatic verification methods to very small n . As long as the circuit is modeled at this level of detail, improvements to verification algorithms are not likely to have a significant effect, since adding a single stage multiplies the resource requirements by k . Hence, one needs to perform verification on a more abstract representation to be able to handle larger n .

The initial proof of correctness for STARI was performed using a theorem-prover ([G93, LG95]). An automatic proof was also published ([HBAB93]). Neither of these studies verified the actual circuit. They operated on simplified, abstract models for STARI which were not proven to be correct and which ignored certain aspects of the circuit implementation. Such simplifications were necessary for these approaches, otherwise the techniques would have become inapplicable or unmanageably complicated.

Our approach provides a formal guarantee of correctness for the circuit itself, and models the implementation more faithfully. We proceed as follows:

- (i) We construct an abstract model for one stage of STARI, which we prove to be correct in the environment that it operates in.
- (ii) By composing n of these abstract models, we obtain an abstract model for STARI on which we prove that the timing properties are satisfied.

(i) implies that the properties are also satisfied by the circuit itself. To achieve (i), we made use of the following, which we had developed in [TAKB96]:

* Supported by SRC under contract DC-324-026.

- An algorithm for checking if a real-time system is a correct abstraction for another: In [TAKB96] we provided a sufficient condition under which a given untimed mapping preserves timed behavior, and gave an algorithm for checking if this condition is satisfied. This algorithm was implemented as part of the verification tool `COSPAN`.
- Assume-guarantee style reasoning for real-time systems: While proving that the abstract model for the FIFO stage was correct, we needed to make assumptions about the environment that it operates in. To discharge these assumptions in a sound way, assume-guarantee reasoning needs to be employed.

The use of multiple levels of abstraction and an inductive argument together with assume-guarantee style reasoning for (i) makes this case study an interesting combination of model checking and theorem proving. The assume-guarantee argument, the abstract model, and the mappings that relate abstract models with the gate level descriptions need to be constructed manually, whereas the abstraction check and the verification of the timing properties on the abstract model are performed automatically by `COSPAN`. The automation afforded by `COSPAN` eliminates the need for having oversimplified abstract models.

In Section 2 we describe the `STARI` circuit. Section 3 presents the verification of the timing properties and contrasts our method with previous ones. Section 4 summarizes the experience from this study and suggests further research.

2 STARI

2.1 Operation of the Interface

Most digital electronic circuits are synchronous, i.e., they make use of a clock signal to define the time step. A high frequency clock signal can be safely distributed over relatively large distances, however, it is hard to control the exact phase of the clock at different points in the distribution network. The difference between the phases of the clock signals at two such points is referred to as *skew*. For systems that are not built on a single chip, such as board level designs, ATM networks, etc., skew can be large and time-varying, which makes it a limiting factor on the performance of purely synchronous systems. Self-timed systems avoid this problem by using handshake protocols. For self-timed systems, if no assumptions are made about the delays of circuits and wires, for each data item that is communicated between two parts of a circuit, an acknowledgment needs to be sent back to the transmitter before another data item can be sent. This can limit the communication bandwidth severely, since only one data item can be in transit at a given time, i.e., two pieces of data need to be separated by the round-trip time between the transmitter and receiver in addition to the response time of the receiver.

`STARI` (Self-Timed At Receiver's Input) is a hybrid-scheme: it is a self-timed first-in first-out queue (FIFO) that connects a transmitter and receiver. The two communicate as though they were part of an ideal synchronous system (Fig. 1 from [G96]) despite a possibly time-varying skew. The FIFO is initialized to be roughly half-full, and during each period of the clock, one value is inserted to

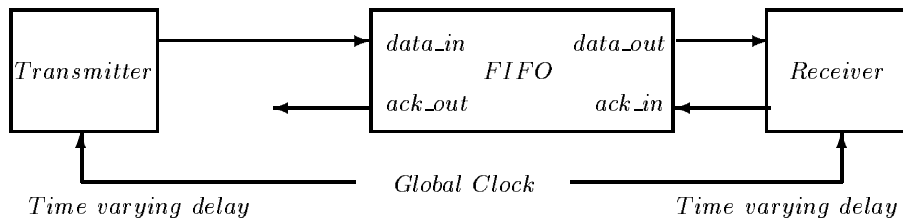


Fig. 1. The STARI interface

the FIFO by the transmitter and one value is removed by the receiver. Because data is inserted and removed at the same rate, no control signals are required to prevent underflow and overflow. However, because of variations in clock skew, there can be short term fluctuations in the clock rate at the receiver or transmitter and it can appear that one of them is working faster than the other. STARI responds to these fluctuations by building up more data in the FIFO when the transmitter is working faster and by supplying data from the FIFO when the receiver is working faster.

For correct operation of the STARI interface, the following two properties need to be proven²:

- (i) Each data value output by the transmitter must be inserted into the FIFO before the next one is output.
- (ii) A new value must be output by the FIFO before each acknowledgment from the receiver.

Intuitively, the longer and faster the FIFO, the more skew it can tolerate. The correctness of the properties above depend on the length of the FIFO, the clock speed, the magnitude of the skew and the speed of operation of FIFO stages. [G93, LG95] verify that if a certain relationship holds between these parameters, then properties (i) and (ii) hold.

In the rest of Section 2 we present the implementation of the STARI circuit and the operation of the interface.

2.2 Dual-rail Coding

In the STARI circuit, each Boolean signal x is represented by the dual-rail code depicted in Fig. 2. The “empty” value is needed to distinguish between two consecutive data items of the same value and one data value asserted for a long time.

$x.t$	$x.f$	x
0	0	E (empty)
0	1	F (false)
1	0	T (true)
1	1	illegal

Fig. 2. Dual rail encoding

² In this study, we focus on timing related properties only and do not consider other properties which need to be proved to ensure correct operation.

2.3 A high-level view of STARI

According to the STARI scheme, the transmitter outputs a data stream, updating the value of $data_in$ in Fig. 1 at each rising edge of its clock input. After each time the transmitter outputs a T or F, it outputs an E to separate the current data item from the next one. The receiver samples its input ($data_out$ in Fig. 1) each time its clock signal goes high, and updates its acknowledge signal after some delay (ack_in in Fig. 1).

The FIFO consists of n identical stages, each of which holds a single data value (See Fig. 3 for an example). Suppose that stage k of the FIFO holds data value d_{old} and that a new data value d_{new} is applied at its inputs by stage $k - 1$. Stage k is “enabled” to read and hold the value d_{new} if stage $k + 1$ has copied the value d_{old} and has notified stage k . After some delay, stage k takes on the value d_{new} and sends an acknowledgment of this fact to stage $k - 1$. Stage k holds this value until a different value is applied to its input by stage $k - 1$ and stage $k + 1$ has acknowledged reading d_{new} . The same data value in the input sequence can be held by many adjacent stages simultaneously, which is what enables the FIFO to hold fewer than n data values (Fig. 3).

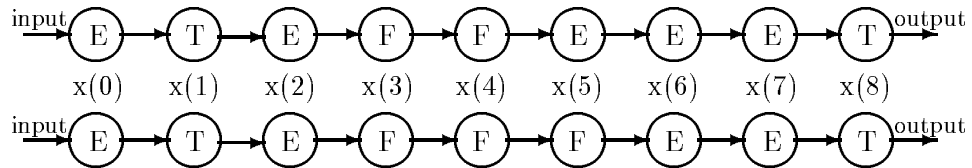


Fig. 3. The FIFO holds the sequence “E,T,E,F,E,T”. Stage 5 is enabled and copies the data value “F”.

2.4 One stage of the FIFO

Each stage of the FIFO consists of two Muller C-elements that hold the value of the $.t$ and $.f$ components of a data item, and a NOR gate that computes the acknowledge output signal of the stage (Fig. 5). A Muller C-element works as follows: when the two inputs are the same, the output takes on this value, when the inputs are different, the output retains its previous value (Fig. 4). To

Input 1	Input 2	Output
0	0	0
0	1	unchanged
1	0	unchanged
1	1	1

Fig. 4. Muller C-element functionality.

understand how data flows down the FIFO, first note that stage k is said to have “acknowledged” the data it holds if its ack_out output is equal to the NOR of $x(k).t$ and $x(k).f$. Thus, the copying of data value E is acknowledged by asserting $ack_out = 1$ and data values T and F are acknowledged by $ack_out = 0$. Let us consider a situation where stages k and $k + 1$ hold the value E and stage

$k - 1$ has the value T. Stage k is enabled to copy the new data from stage $k - 1$. We have $\text{ack_out}(k+1) = \text{ack_in}(k) = 1$ and $\mathbf{x}(k-1).t = 1$ and $\mathbf{x}(k-1).f = 0$. Therefore, both inputs to the C-element that computes $\mathbf{x}(k).t$ are 1, while the output of the C-element, $\mathbf{x}(k).t = 0$. After some delay, $\mathbf{x}(k).t$ becomes 1 while $\mathbf{x}(k).f$ remains unchanged, and, in this way, the value T gets copied to stage k . Transitions from a T or F to E happen similarly.

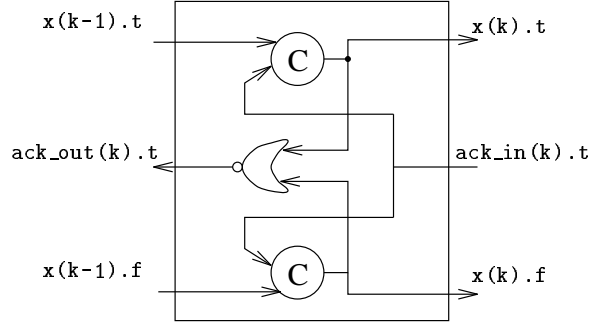


Fig. 5. Stage k of the FIFO.

Given this circuit description for STARI, we now proceed to verify the two timing properties mentioned earlier.

3 Verification of STARI

3.1 Background

The formalism and tools used in the verification of STARI are described in detail in [TAKB96]. In this section, we review the essential facts.

Modeling timed systems: Timed processes. For an example of a *timed process*, refer to Figure 6. A timed process has a set of locations S , where $S_0 \subseteq S$ is designated as initial. The sets of input and output variables are I and O . At each location a unique value is specified for each output variable. *Clocks* are real-valued variables that can be reset when edges are taken, and they increase at the same rate. Edges are conditioned on *clock* and *input predicates*. A *clock predicate* is a positive Boolean combination of inequalities $x \diamond k$ where $k \in \mathbb{N}$ and \diamond is \leq or \geq . An *input predicate* is a condition of the form $i' = i_{new}, i = i_{old}$ which denotes the fact that the input variable i has switched to value i_{new} from i_{old} . At each location there is a clock predicate, called an *invariant*, that needs to be satisfied while the process remains at that location.

The set of input/output waveforms that a process can exhibit constitute the *language* of the process.

The delay model. We model the C-elements and the NOR gates used in the STARI circuit as ideal delayless elements followed by inertial delay buffers, in a fashion similar to [MP95]. The output of an inertial delay buffer follows the input with a delay in the range $[d_{min}, d_{max}]$, i.e., input transitions are reflected at the output with a delay in the given range. The timed process modeling an

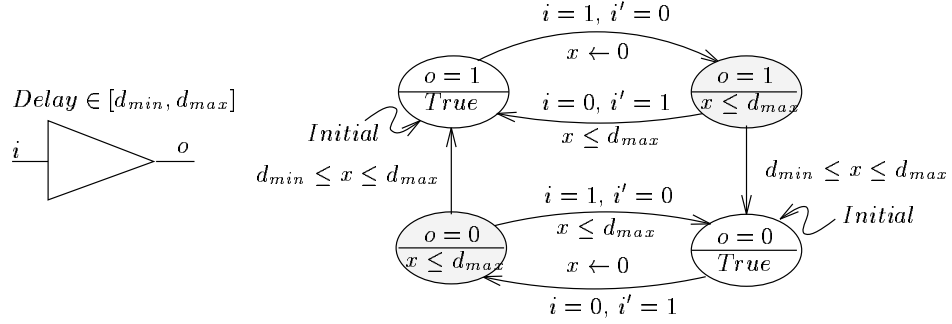


Fig. 6. An inertial delay buffer with delay in the range $[d_{min}, d_{max}]$. The locations where an output change is pending are shaded.

inertial delay buffer is given in Figure 6. If an input pulse lasts less than d_{min} , it is not reflected at the output. If it lasts longer than d_{max} , then it is guaranteed to cause a pulse at the output. Pulses lasting between d_{min} and d_{max} may or may not result in an output pulse: Suppose a transition occurs at the input, and before the corresponding output transition takes place (shaded locations in Fig. 6), the input returns to its original value. Then the output can either remain unchanged (by taking the edges marked $x \leq d_{max}$ from the shaded locations) or reflect both input transitions (by taking the edges marked $d_{min} \leq x \leq d_{max}$).

Abstractions of timed systems. In [TAKB96], we examined three notions for a timed system being a correct abstraction (or implementation) of another: (i) timed language inclusion, (ii) timed simulation, and (iii) the existence of timed behavior preserving mappings. We write $A \preceq B$ to denote the fact that A implements B . For timed processes A and B , a mapping between the locations $h : S^A \rightarrow S^B$ is said to *preserve timed behavior*, iff for each run³ of A , the image of the run under h is a run of B . The existence of such a map implies that B is a correct abstraction of A . We gave an algorithm for checking this condition and implemented it as part of COSPAN. We use this algorithm extensively at several stages of the verification of STARI.

Compositional and assume-guarantee style reasoning. An implementation relation \preceq is *compositional* iff the following holds

$$\text{For all } i, 1 \leq i \leq n, R_i \preceq A_i \text{ implies } (\|_{1 \leq i \leq n} R_i) \preceq (\|_{1 \leq i \leq n} A_i)$$

With the stronger *assume-guarantee style reasoning*, one can prove $(\|_{1 \leq i \leq n} R_i) \preceq (\|_{1 \leq i \leq n} A_i)$ by proving for all $i, 1 \leq i \leq n$, that

$$A_1 \parallel \dots \parallel A_{i-1} \parallel R_i \parallel A_{i+1} \parallel \dots \parallel A_n \preceq A_1 \parallel \dots \parallel A_{i-1} \parallel A_i \parallel A_{i+1} \parallel \dots \parallel A_n$$

This style of reasoning is often more useful, since, while showing $R_i \preceq A_i$ one often needs to make assumptions about the environment that R_i and A_i operate in, and A_1, \dots, A_{i-1} and A_{i+1}, \dots, A_n encapsulate the strongest such set of assumptions.

In [TAKB96] it was shown that timed language inclusion, denoted by \preceq_L is compositional and that assume-guarantee style reasoning can be used in conjunction with it correctly, provided that all timed processes are non-blocking. The

³ A *run* is a sequence of locations, where the time spent at each location is specified.

other two implementation relations mentioned imply timed language inclusion, and thus one can apply assume-guarantee style reasoning using these relations as well.

3.2 Verification Steps

The verification of STARI consisted of the following two steps:

- Constructing an abstraction for a FIFO stage and verifying its correctness within the environment that it operates.
- Verifying properties (i) and (ii) of Section 2 using the abstract model for the entire circuit.

The latter of these steps was performed using the existing timing verification capabilities of COSPAN ([AK96]). The former step is the novel part of our approach and will be detailed below.

The abstract model for a stage. The abstract model for the FIFO stage describes its behavior at a high level, as in Section 2.3, and expresses bounds on certain response times. The abstract model A (depicted in Figure 7) makes use of only one clock variable. Let us focus on stage k . At location `stable`, the FIFO stage has read and acknowledged its current input. If new data arrives at the inputs of stage k , A moves to location `wait_for_ack`, waiting for stage $k + 1$ to acknowledge having copied the current data. If stage $k + 1$ sends an acknowledgment before the new data arrives at stage k , stage k moves to location `wait_for_data` from location `stable` instead. After new data and an acknowledgment for the old data from stage $k + 1$ has been received, A moves to location `out_pend` from where, after some delay, it moves to `ack_out_pend` and copies the new data to its output. Again after some delay, an acknowledgment is sent to stage $k - 1$ and A moves to location `stable` (see [W3] for COSPAN code describing A).

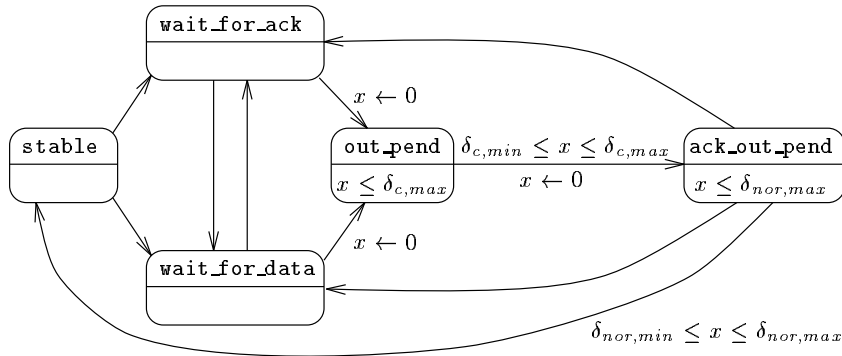


Fig. 7. The timed process A describing the abstract model for a FIFO stage. The minimum and maximum response times of the C-element and the NOR gate are given by $\delta_{c,min}$, $\delta_{c,max}$, $\delta_{nor,min}$ and $\delta_{nor,max}$ respectively. Input predicates labeling the edges and output labels on the locations have been omitted to simplify the figure.

Note that it has been possible to capture the timing information about the stage using one clock only, since only one of the three circuit elements forming

the stage can have a pending output change at any given time. Intuitively, this is guaranteed by the fact that the inputs to a stage will not change unless the stage has acknowledged the previous inputs. This assumption about the environment of a stage is crucial for the correctness of the abstraction, and is taken into account in our verification by the use of assume-guarantee reasoning.

Let F denote the timed process describing one stage of the FIFO at the gate level. F is the composition of processes representing two Muller C-elements and a **NOR**-gate as described by Figure 5 and Section 3.1. Let F_i and A_i denote the detailed and abstract models for the i th FIFO stage. F_i and A_i have structures identical to those of F and I . Also let Tx and Rx be the timed processes describing the transmitter and the receiver. Refer to [W3] for **COSPAN** models of Tx and Rx . We would like to prove that

$$Tx \parallel F_1 \parallel F_2 \parallel \dots \parallel F_n \parallel Rx \preceq_L Tx \parallel A_1 \parallel A_2 \parallel \dots \parallel A_n \parallel Rx \quad (1)$$

which will enable us to prove properties using the abstract description for the circuit given by the right-hand side. We would like to achieve this by showing that for all i , A_i is a correct abstraction for F_i . As noted above, this is true only within the environment that F_i and A_i operate in. For an arbitrary environment $F_i \preceq_L A_i$ does not hold: if the inputs to F are unconstrained, then F_i does not behave like a FIFO element. Therefore, we need to employ assume-guarantee reasoning to carry out the proof. We must prove, for all i , $1 \leq i \leq n$, that

$$\begin{aligned} Tx \parallel A_1 \parallel \dots \parallel A_{i-1} \parallel F_i \parallel A_{i+1} \parallel \dots \parallel A_n \parallel Rx &\preceq \\ Tx \parallel A_1 \parallel \dots \parallel A_{i-1} \parallel A_i \parallel A_{i+1} \parallel \dots \parallel A_n \parallel Rx & \end{aligned}$$

Since the environment of each FIFO stage is different, for each i , the condition to be checked is different. To avoid performing a separate check for each i , we would like to have one set of environment assumptions that is sufficient for showing $F_i \preceq A_i$ and show that this set of assumptions is true for the environment of each module i . Towards this end, we define timed processes E_{left} and E_{right} and show the following:

- (I) For all i , $Tx \parallel A_1 \parallel A_2 \parallel \dots \parallel A_{i-1} \preceq E_{left}$
- (II) For all i , $A_{i+1} \parallel A_{i+2} \parallel \dots \parallel A_n \parallel Rx \preceq E_{right}$

(I) and (II) guarantee that E_{left} and E_{right} are correct abstractions for the left and right sides of the environment of stage i . By the assume-guarantee rule, showing that $F \preceq A$ in the environment defined by E_{left} and E_{right} , i.e.,

$$E_{left} \parallel F \parallel E_{right} \preceq E_{left} \parallel A \parallel E_{right}$$

suffices to prove the containment in Equation 1.

Proving correctness of environment abstractions. E_{left} encapsulates the restrictions on the behavior of the left side of F for it to function correctly and for A to be a correct abstraction. The essential features of E_{left} are that it always outputs a valid data value, and that it does not change this value until the next stage acknowledges having copied this value. We use induction on i to prove that E_{left} is a correct abstraction for $Tx \parallel A_1 \parallel A_2 \parallel \dots \parallel A_{i-1}$. More precisely, we show the following

1. $Tx \preceq E_{left}$ ⁴
2. Assuming $Tx \parallel A_1 \parallel A_2 \parallel \dots \parallel A_{i-1} \preceq_L E_{left}$ we show $Tx \parallel A_1 \parallel \dots \parallel A_i \preceq E_{left}$. By the induction assumption, it suffices to prove $E_{left} \parallel A_i \preceq E_{left}$.

For both steps we specify mappings from the locations of the left-hand side process to those of the right-hand side process, and show that this mapping preserves timed behavior using the COSPAN implementation mentioned in Section 3.1. COSPAN code describing the modules and the mappings is provided at [W3].

The essential feature of E_{right} is that it samples the data at its inputs periodically and after a certain delay, acknowledges having read the data. We proved that E_{right} is a correct abstraction in exactly the same manner as E_{left} . The COSPAN code for E_{right} and the untimed mappings can be found at [W3].

Proving that A is a correct abstraction for F . Given E_{left} and E_{right} , it was rather straightforward to prove that $E_{left} \parallel F \parallel E_{right} \preceq E_{left} \parallel A \parallel E_{right}$. COSPAN code for the untimed mapping is given in [W3].

Time and memory consumption. We report the resource usage for the following checks (Table 1):

- (I) $Tx \preceq E_{left}$
- (II) $E_{left} \parallel A \preceq E_{left}$
- (III) $Rx \preceq E_{right}$
- (IV) $A \parallel E_{right} \preceq E_{right}$
- (V) $E_{left} \parallel F \parallel E_{right} \preceq E_{left} \parallel A \parallel E_{right}$
- (VI) $Tx \parallel A_1 \parallel \dots \parallel A_8 \parallel Rx$ satisfies timing properties (i) and (ii) of Section 2.

For all of these checks the following parameters were used: $\delta_{c,min} = 1$, $\delta_{c,max} = 2$, $\delta_{nor,min} = 1$, $\delta_{nor,max} = 2$, the clock period $\pi = 12$. The delays from the clock to the transmitter and the receiver (see Figure 1) were allowed to be time varying and bounded by 1 time unit.

To serve as a comparison, we tried to verify properties (i) and (ii) of Section 2 using the gate level model for STARI, i.e., using F as the model for a FIFO stage. At this level of detail, we ran out of space using 1 GB of memory for a three stage FIFO. Improvements to our timing verification algorithm could enable us to verify larger FIFOs at the gate level, however, any method working on a monolithic description at the circuit level is bound to run out of resources for a FIFO with a large enough number of stages. Using abstractions as we demonstrated above, one is able to handle larger FIFOs, although not an unbounded number of them, since the number of FIFO elements still enters into the computation (VI).

⁴ Some technicality is involved here. To prove the basis case, we disallow the transmitter to change its data output if the first FIFO stage has not copied the previous value. Later on, while proving that the interface works correctly, we prove that it is never the case that the transmitter wants to modify the data output and the FIFO is not ready to receive new data.

	Num. of clocks	Memory (MB)	CPU time (seconds)
I	3	0.02	1.3
II	3	0.02	4.9
III	3	0.02	13.4
IV	6	3.18	92.7
V	7	6.33	158.6
VI	12	92.4	6014.1

Table 1.

Comparison with previous approaches. The most important benefit of our approach is the correctness guarantee that it provides for the actual circuit, whereas [LG95] and [HBAB93] based their proofs on oversimplified abstractions of the circuit. Their tools do not provide a mechanism for checking the correctness of abstractions, therefore there is no formal guarantee that the properties they proved at a high level are satisfied by the circuit.

[LG95] and [HBAB93] neglect the time taken by the **NOR** gate (see Figure 5) to compute the acknowledgment output and, furthermore, use a delay model that is less realistic than the inertial delay model that we employ. Since verification of properties on the abstract model is performed automatically in our approach, we did not need to resort to such simplifications.

The proof of [LG95] is for all FIFO lengths n , and makes it easier to see the trade-off between circuit parameters, whereas our approach is still limited by n . However, the proof of [LG95] is rather involved, and one needs to have an in-depth understanding of why the properties are satisfied. The abstraction proofs and environment abstractions that we used were rather straightforward and intuitive.

4 Conclusion

We demonstrated the use of compositionality and hierarchy on the verification of the STARI communication circuit. By using the timed refinement checking algorithm that we had implemented as part of **COSPAN** in a compositional framework, we were able to divide the verification problem into smaller pieces, which enabled us to automatically verify a larger circuit than was previously possible.

This case study demonstrated once more that abstractions are indispensable for verifying large systems, and that compositional and assume-guarantee style reasoning are not only useful techniques for verifying the correctness of abstractions, but are almost always necessary.

The size of our abstract model for STARI, $Tx \parallel A_1 \parallel \dots \parallel A_n \parallel Rx$ still has an exponential dependency on n , although less severe than the gate level model. One problem that remains is the construction of an abstract model for STARI that is parametrized with respect to n . One would then prove the correctness of this abstraction using induction. Parametrized real-time systems have been studied before ([AHV93]) and there is indication that this problem is rather complex.

Acknowledgments

We thank Rajeev Alur and Robert Kurshan for helpful discussions and support with **COSPAN**.

References

- [AHV93] R. Alur, T.A. Henzinger, M.Y. Vardi. Parametric real-time reasoning. In *Proceedings of the 25th ACM Symposium on Theory of Computing*, pp. 592-601, 1993.
- [AK96] R. Alur and R.P. Kurshan. Timing analysis in COSPAN. In *Hybrid Systems III*, Lecture Notes in Computer Science. Springer-Verlag, 1996.
- [G93] M. R. Greenstreet. STARI: A Technique for High-Bandwidth Communication. PhD thesis, Princeton University, 1993.
- [G96] M. R. Greenstreet. STARI: Skew Tolerant Communication. Unpublished manuscript.
- [HBAB93] H. Hulgaard, S. M. Burns, T. Amon, and G. Borriello. Practical Applications of an Efficient Time Separation of Events Algorithm. In *Digest of Technical Papers of the 1993 IEEE Intl. Conf. on Computer-Aided Design*, November 1993.
- [LG95] C. Leung, M. Greenstreet. A Simple Proof Checker for Timing Verification. In *ACM Intl. Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, pages 294-305, November 1995.
- [MP95] O. Maler, A. Pnueli. Timing Analysis of Asynchronous Circuits Using Timed Automata. In *ACM Intl. Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, pages 249-257, November 1995.
- [TAKB96] S. Tasiran, R. Alur, R. P. Kurshan, and R. K. Brayton. Verifying Abstractions of Timed Systems. In *Proc. of the 7th Intl. Conf. on Concurrency Theory, CONCUR '96*, LNCS 1119, pages 546-562, Springer-Verlag, 1996.
- [W3] <http://www-cad.eecs.berkeley.edu/~serdar/stari.html>