A Protocol for Loosely Time-Triggered Architectures *

Albert Benveniste[1], Paul Caspi[2]
Paul Le Guernic[1], Hervé Marchand[1], Jean-Pierre Talpin[1], Stavros Tripakis[2]

[1] Irisa/Inria, Campus de Beaulieu, 35042 Rennes cedex, France, Email: firstname.lastname@irisa.fr, Web: http://www.irisa.fr/sigma2/benveniste/
[2] Verimag, Centre Equation, 2, rue de Vignate, F-38610 Gieres, Email: firstname.lastname@imag.fr, Web: http://www.imag.fr/VERIMAG/PEOPLE/Paul.Caspi

Abstract A distributed real-time control system has a time-triggered nature, just because the physical system for control is bound to physics. Loosely Time-Triggered Architectures (LTTA) are a weaker form of the strictly synchronous Time-Triggered Architecture proposed by Kopetz, in which the different periodic clocks are not synchronized, and thus may suffer from relative offset or jitter.
We propose a protocol that ensures a coherent system of logical clocks on the top of LTTA, and we provide several proofs for it, both manual and automatic, based on synchronous languages and associated model checkers. We briefly discuss how this can be used for correct deployment of synchronous designs on an LTTA.

1 Loosely Time-Triggered Architectures (LTTA)

A distributed real-time control system has a time-triggered nature, just because the physical system for control is bound to physics. Loosely Time-Triggered Architectures are a weaker (and cheaper) form of the strictly synchronous Time-Triggered Architecture (TTA) proposed by Kopetz [11].

An LTTA is an architecture in which: 1/ access to the bus occurs quasi-periodically, in a non-blocking way, 2/ writings and readings are performed independently at each extremity of the bus in synchrony with each associated local clock, and 3/ the bus behaves like a shared memory, i.e., values are sustained by the bus and are periodically refreshed, based on a local clock. The term "quasi-periodically" indicates that the different clocks involved, for writing in, reading from, and updating the bus, are not synchronized. Still, this architecture is time-triggered in the sense that these clocks are bound to *physical* time, and deviate from each other in a certain "limited" way. LTTA are in use in several major industries, they have been the subject of the CRISYS project [6], where they are called *quasi-synchronous,* and of several investigations, by Caspi and co-workers, of the fundamental issues raised when deploying control applications on such architectures.

* This work is or has been supported in part by the following projects : Esprit LTR-SYRF (Esprit EP 22703), and Esprit R&D CRISYS EP 25514.

Our main result is presented in section 2. It consists in showing that, by adding some layer of protocol on the top of an LTTA, one can offer a platform ensuring a coherent distribution of *logical* clocks. We formally specify the protocol and prove that it satisfies the desired requirement. For the protocol to be correct, the clocks must be quasi-periodic (periods can vary within certain specified bounds), and must relate to each other within some specified bounds.

That one can offer, on the top of an LTTA, a platform ensuring a coherent distribution of *logical* clocks, has actual interests. Distributed sensor-to-actuators low level feedback control loops are in any case faced with the unavoidable uncertainty of sampling, whatever the actual considered bus architecture is. Hence, in any case, the robustness of the deployed application with respect to this type of "asynchrony" must be considered. This is the subject of [7] and is not considered here. However, complex distributed control applications also involve complex finite state machines, e.g., to govern modes of operation, and reconfigurate the application against degraded modes. The possibility to offer a coherent system of logical clocks is extremely useful to simplify the development and debugging of such finite state machines. Section 3 sketches a methodology for deploying, on an LTTA equiped with the proposed protocol, a finite state machine developed with any synchronous language.

Section 4 addresses the same problem as section 2, but using (nearly) automatic proof techniques. To this end, the protocol is specified using *synchronous languages*—we show the exercise with the languages Lustre and Signal [9]. It is interesting to note that synchronous languages can be used to model asynchronous systems—recall the three devices writer/bus/reader have independent and non synchronized clocks. Then the desired property for the protocol is expressed as an invariant, and proved.

Now, the assumptions ensuring correctness of the protocol, as stated in section 2, are quantitative in nature (tolerance bounds for the relative periods, and time variations, of the different clocks). Handling such type of quantitative assumption is beyond the scope of model checkers such as Lesar [10] or Sigali [14], the model checkers associated with Lustre and Signal, respectively. Hence the assumption is reformulated, by performing some abstraction and expressing it using booleans. While the two assumptions, strictly speaking, are not equivalent, they are good approximations to each other—but this claim cannot be proved by model checking!

Our protocol is presented and analysed in the case of two users: one writer and one reader. Section 5 discusses the case of multiple users. Then, a brief comparison with Kopetz'TTA is provided.

2 The proposed protocol and its robustness

2.1 Description of the protocol

See figure 1 for an illustration of this protocol (the three watches shown indicate a different time, *they are not synchronized*). We consider three devices, the

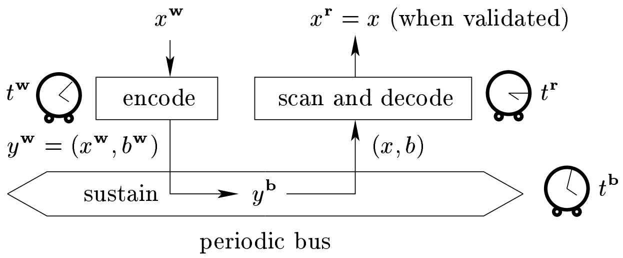

Figure1. *The protocol.* The map $y \mapsto y^{\mathbf{r}}$ is called a *virtual channel.*

writer, the *bus,* and the *reader,* indicated by the superscripts $(.)^{\mathbf{w}}, (.)^{\mathbf{b}}$, and $(.)^{\mathbf{r}}$, respectively. Each device is activated by its own, approximately periodic, clock. The different clocks are *not* synchronized. In the following specification, the different sequences written, fetched, or read, are indexed by the set $\mathbf{N} = \{1, 2, 3, \ldots, n, \ldots\}$ of natural integers, and we reserve the index 0 for the initial conditions, whenever needed.

The writer: At the time $t^{\mathbf{w}}(n)$ of the nth tick of his clock, the writer generates a new value $x^{\mathbf{w}}(n)$ it wants to communicate and a new alternating flag $b^{\mathbf{w}}(n)$ with:

$$b^{\mathbf{w}}(n) = \begin{cases} \mathit{false} & \text{if } n = 0 \\ \mathit{not}\ b^{\mathbf{w}}(n-1) & \text{otherwise} \end{cases}$$

and stores both in its private output buffer.

Thus at any time t, the writer's output buffer content $y^{\mathbf{w}}$ is the last value that was written into it, that is the one with the largest index whose tick occurred before t:

$$y^{\mathbf{w}}(t) = (x^{\mathbf{w}}(n), b^{\mathbf{w}}(n)), \text{ where } n = \sup\{n' \mid t^{\mathbf{w}}(n') < t\} \tag{1}$$

The bus: At the time $t^{\mathbf{b}}(n)$ of its nth clock tick, it fetches the value in the writer's output buffer and stores it, immediately after, in the reader's input buffer. Thus, at any time t, the reader's input buffer content offered by the bus, denote it by $y^{\mathbf{b}}$, is the last value that was written into it, i.e., the one written at the latest bus clock tick preceding t:

$$y^{\mathbf{b}}(t) = y^{\mathbf{w}}(t^{\mathbf{b}}(n)), \text{ where } n = \sup\{n' \mid t^{\mathbf{b}}(n') < t\} \tag{2}$$

The reader: At the time $t^{\mathbf{r}}(n)$ of its nth clock tick, it copies the value of its input buffer into auxiliary variables $x(n)$ and $b(n)$:

$$(x(n), b(n)) = y^{\mathbf{b}}(t^{\mathbf{r}}(n))$$

Then the reader extracts from the x sequence only the values corresponding to the indices for which b has alternated[1]. This can be modeled thanks to the counter m, which counts the number of alternations that have taken place up to the current cycle. Then the value of the extracted sequence at index k is the value read at the earliest cycle when the counter m exceeded k:

$$\begin{aligned} m(0) = 0\,,\, m(n) = \inf\{k > m(n-1) \mid b(k) \neq b(k-1)\} \\ x^{\mathbf{r}}(k) = x(l)\,,\text{ where } l = \inf\{n' \mid m(n') > k\} \end{aligned} \tag{3}$$

Problem statement: The protocol is correct if the two sequences $x^{\mathbf{w}}, x^{\mathbf{r}}$ coincide, i.e.,

$$\forall n : x^{\mathbf{r}}(n) = x^{\mathbf{w}}(n) \tag{4}$$

2.2 Main theorem about robustness

Theorem 1 (sampling theorem). *Let the writing/bus/reading be systems with physically periodic clocks of respective periods $w/b/r$. Then, the protocol of subsection 2.1 satisfies the desired property:*

$$\forall n : x_n^{\mathbf{r}} = x_n^{\mathbf{w}}\,, \tag{5}$$

whatever the written input sequence is, iff the following conditions hold:

$$w \geq b\,,\text{ and } \left\lfloor \frac{w}{b} \right\rfloor \;\geq\; \frac{r}{b}\,, \tag{6}$$

where, for x a real, $\lfloor x \rfloor$ denotes the largest integer $\leq x$.

Condition (5) means that the bus provides a coherent system of logical clocks. Note that, since $w \geq b$, then $w/2b < \lfloor w/b \rfloor$ follows. On the other hand, $\lfloor w/b \rfloor \leq w/b$, and $\lfloor w/b \rfloor \sim w/b$ for w/b large. Hence, for a fast bus, i.e. $b \sim 0$, the conditions (6) of theorem 1 reduce to:

$$w \gg b\,,\, w > r. \tag{7}$$

We illustrate the protocol on the figure 2, for two typical cases (shown in black and white arrows on the top of the diagram, respectively), depending on the relative position of the ticks of clocks $t^{\mathbf{b}}, t^{\mathbf{w}}, t^{\mathbf{r}}$. The two thick lines depict the sustained values of the two components, $(x^{\mathbf{w}}, b^{\mathbf{w}})$, of $y^{\mathbf{w}}$. Collect these two sustained values as the tuple $y^{\mathbf{b}}$. The dashed vertical lines depict the ticks of the bus clock $t^{\mathbf{b}}$. The vertical arrows sitting at the bottom depict the ticks of the writer clock. The vertical arrows sitting at the top depict the ticks of the reader clock. Note the role of the boolean flag for validation: even if the value of $x^{\mathbf{w}}$ was unchanged at two successive emissions, these would have been validated,

[1] This is the classical technique used in the Alternating Bit Protocol, to avoid the reader receiving the same message twice.

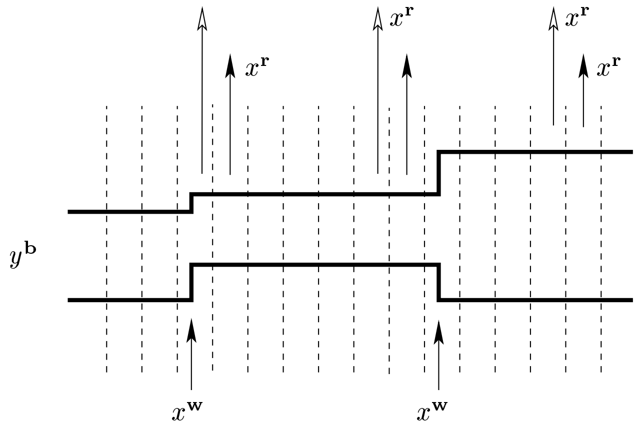

Figure2. *The sampling theorem.*

thanks to the boolean flag. The role of this flag is also enlightened by the two cases for the relative position of the clocks (black and white arrows). The case of the black arrows is the "normal" one. Let us focus on the more tricky case of the white arrows. In this case, when the first instant of clock $t^{\mathbf{r}}$ (first white arrow), then the clock $t^{\mathbf{b}}$ has not yet seen the change in the sustained value of b, therefore the corresponding sustained value for x is not validated by the reader at this point. But a change in the sustained value of b is detected at the second occurrence of $t^{\mathbf{r}}$, and, then, the corresponding sustained value for x is validated. As seen from the diagram, changes in the boolean signals are detected at the receiver with a nondeterministic, but bounded delay. On the other hand, perfect physical synchrony is lost: reading occurs with a nondeterministic, but bounded, delay according to *physical* real time.

Proof. In the following, let $n, m, p \in \{0, 1, 2, \ldots\}$ denote integers. By properly renormalizing the three periods, we can assume, for the bus periodic clock, $b = 1$ and a zero phase. With this convention, let the writing/bus/reading sampling instants be:

$$\begin{aligned} t^{\mathbf{w}}(n) &= nw + \psi w, \ 0 \leq \psi < 1 \\ t^{\mathbf{b}}(p) &= p \\ t^{\mathbf{r}}(m) &= mr + \varphi r, \ 0 \leq \varphi < 1 \end{aligned} \tag{8}$$

where the writing/bus/reading periods are $w/1/r$, and ψ, φ denote the phases of the writer's and reader's clocks.

We first search for conditions ensuring that the bus does not miss any writing. Set:

$$\tau^{\mathbf{b}}(n) = \min\{t^{\mathbf{b}}(p) \mid t^{\mathbf{b}}(p) > t^{\mathbf{w}}(n)\}\,, \tag{9}$$

$\tau^{\mathbf{b}}(n)$ is the first instant where the bus can fetch the nth writing, and we have:

$$\tau^{\mathbf{b}}(n) = \lfloor (n+\psi)w \rfloor + 1,$$

whence:

$$\delta\tau^{\mathbf{b}} \stackrel{\Delta}{=} \min_{n\geq 0\,,\,0\leq\psi<1} \left(\tau^{\mathbf{b}}(n+1) - \tau^{\mathbf{b}}(n)\right) \;=\; \lfloor w \rfloor\,. \tag{10}$$

To ensure that the bus does not miss any writing whatever the phases of the different clocks are, it is necessary and sufficient that the map $n \longmapsto \tau^{\mathbf{b}}(n)$ is strictly increasing. Using (10), this holds iff:

$$\lfloor w \rfloor > 0 \text{ , i.e., } w \geq 1. \tag{11}$$

Next, we investigate conditions ensuring that the reader does not miss a writing, when the latter is fetched by the bus. Define:

$$\tau^{\mathbf{r}}(n) = \min\{t^{\mathbf{r}}(m) \mid t^{\mathbf{r}}(m) > \tau^{\mathbf{b}}(n)\}. \tag{12}$$

Thanks to the mechanism of the boolean alternating flag b, $\tau^{\mathbf{r}}(n)$ is the first instant where the reader sees the nth writing. Thus, using (10), our last duty is *not to miss a reading during any period of length $\delta\tau^{\mathbf{b}}$*, equivalently:

$$\lfloor w \rfloor \geq r\,. \tag{13}$$

Combining (11) and (13) yields the theorem. $\diamond$

We can generalize theorem 1 to time varying periods. Without loss of generality we can again assume that the bus clock is perfectly periodic, with period 1 and phase 0:

Theorem 2 (sampling theorem with quasi periodic clocks). *Assume that writing/bus/reading are only approximately periodic, i.e., they are related via the following equations—compare with (8):*

$$\begin{array}{ll} t^{\mathbf{w}}(n) = nw(1+\delta^{\mathbf{w}}(n)) + \psi w \text{ , } 0 \leq \psi < 1 \text{ , } |\delta^{\mathbf{w}}(n)| \leq \delta^{\mathbf{w}} \\ t^{\mathbf{b}}(p) \;= p \\ t^{\mathbf{r}}(m) = mr(1+\delta^{\mathbf{r}}(m)) + \varphi r \;\text{ , } 0 \leq \varphi < 1 \text{ , } |\delta^{\mathbf{r}}(m)| \leq \delta^{\mathbf{r}}, \end{array} \tag{14}$$

where $\delta^{\mathbf{w}}, \delta^{\mathbf{r}}$ are some given bounds. Then sufficient conditions, for the protocol formalized in section 2.1 to ensure condition (5), are the following:

$$w(1-2\delta^{\mathbf{w}}) \geq 1 \text{ , and } \lfloor w(1-2\delta^{\mathbf{w}}) \rfloor \geq r(1+2\delta^{\mathbf{r}}), \tag{15}$$

compare with conditions (11) and (13).

The proof of this theorem is a straightforward adaptation of that of theorem 1, and we leave it to the reader as an exercise. Of course, the interest of theorem 2 is that it guarantees a good degree of robustness of the protocol with respect to imperfect sampling clocks, since the relative *periods* of the clocks can vary with respect to each other, up to some maximal bound. Also, the jitter terms $\delta^{\mathbf{w}}(n)$ and $\delta^{\mathbf{r}}(m)$ can incorporate variable propagation delay in the bus.

3 Application to the deployment of synchronous programs

Figure 3-left shows a synchronous design, with three components. In figure 3-

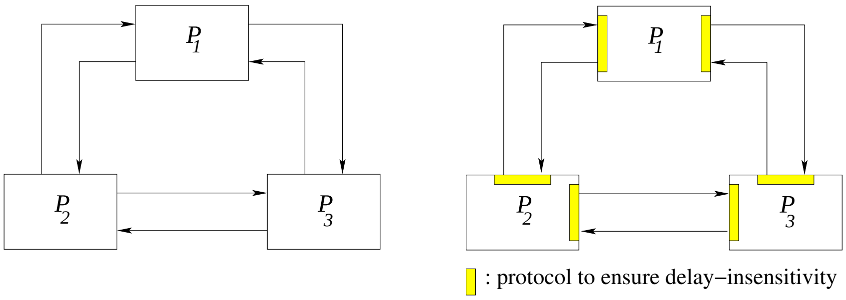

Figure3. *A synchronous design, securing delay-insensitivity.*

right, additional synchronous modules are shown, they are depicted using the light grey rectangles and can be regarded as “protocols”. Based on the theory developed in [3][4][5], these protocols aim at securing delay-insensitivity of the communications.

This means that the semantics of each individual synchronous component in the system is not modified if the perfectly synchronous communication channels are substituted with asynchronous channels of FIFO type, with unknown and unbounded delays, but no loss. This is formalized as the mathematical property (4). Of course the global timing of the program is lost. The automatic synthesis of such protocols is achieved by enforcing the so-called *endo/isochrony* of the synchronous commmunication network, see [3]. At this stage delay-insensitivity is achieved: each flow is individually preserved by the communication (but their global synchronisation is not).

Now we show in figure 4-left how to adapt this design for the case of an LTTA, i.e., with *loosely synchronised* channels. Note that loosely synchronised channels, if used directly, can duplicate data or can lose data. The idea is to include, on each port, the protocol proposed in section 2. These additional protocols are figured in figure 4-left by the dark rectangles. In figure 4-right, we have redrawn the figure 4-left in a different way. We combine the dark rectangles together with the “physical” channels, to obtain so-called “virtual” channels. The analysis performed in section 2 shows that each virtual channel satisfies the property (4). Therefore endo/isochrony guarantees a correct-by-construction deployment.

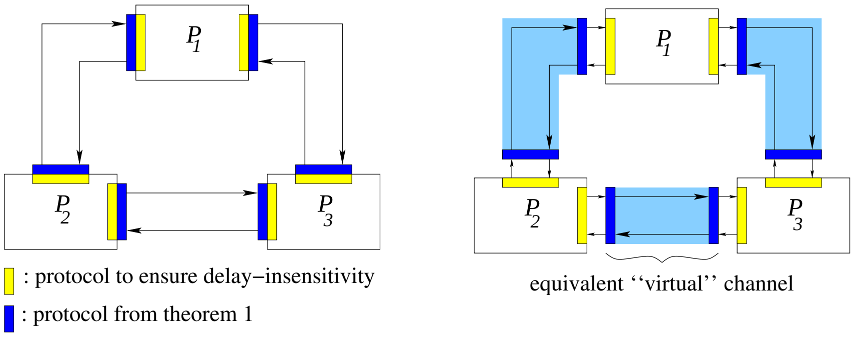

Figure4. *Encoding events as state changes. Introducing "virtual" channels.*

4 Automatic proofs of the protocol

In this section, we redo the section 2 by replacing human specifications and proofs, by computer ones. Synchonous languages with associated proof tools are used to this end. Some comments are in order.

The two theorems shown before involve both timing and logical aspects. The timing aspects can be modeled using parametric timed automata, since we want to prove the theorems using *symbolic* values for the periods and not instantiated ones. Unfortunately, the verification problem for parametric timed automata is undecidable [1], therefore, model checkers such as Kronos [8] or Uppaal [13], cannot be used. Semi-algorithmic methods for parametric timed automata exist (e.g., in Hytech), but they are not guaranteed to terminate. Instead we have decided to use standard model checking, and therefore abstractions are needed.

Two kinds of abstractions will be considered.

1. We want to show identity (5) irrespectively of the actual values for the input $x^{\mathbf{r}}$. Since we do not use theorem provers, we will actually prove (5) for finitely enumerated types. The proof with Lustre/Lesar assumes an input which is of type bit stream with a given concrete width.
2. Then, we need to propose boolean type of assumptions to replace the quantitative assumptions (6) or (15), from section 2. We ask the reader to have the proof of theorem 1 at hand. From (9,10,11), we get that the first condition, $w \geq b$, in (6) is abstracted as the following predicate:

$$(6) : w \geq b \ \leftrightarrow \ \text{never two } t^{\mathbf{w}} \text{ between two successive } t^{\mathbf{b}}. \tag{16}$$

Finding an abstraction for the second condition, $\lfloor w/b \rfloor \geq r/b$, in (6), is slightly less obvious. We just reexpress it by requiring that the sequence $\tau^{\mathbf{r}}(n)$ defined in (12) shall be strictly increasing, i.e., it shall never be the case that two or more $\tau^{\mathbf{b}}(n)$'s occur between two successive $t^{\mathbf{r}}(m)$'s:

$$(6) : \left\lfloor \frac{w}{b} \right\rfloor \geq \frac{r}{b} \ \leftrightarrow \ \text{never two } \tau^{\mathbf{b}}(.) \text{ between two successive } t^{\mathbf{r}}. \tag{17}$$

To complete the requirements, we must characterize $\tau^{\mathbf{b}}(.)$, this is provided by the formula (9).

Note that, since these abstractions have been performed *manually,* this is a (minimally) assisted proof, not a fully automated one.

The Lustre/Lesar proof. With these abstractions, the Lustre/Lesar proof is shown in Fig. 5 and Fig. 6. Note that these abstractions result in finite-state systems, which could be checked in principle with any model checker. As for the different delays involved in the reader/bus/writer (see formulas (1,2,3)), these are emulated by simply making the three clocks $t^{\mathbf{w}}, t^{\mathbf{b}}, t^{\mathbf{r}}$ pairwise exclusive: by doing so, instantaneous transfer can occur, neither from writer to bus, nor from bus to reader—this way is a bit of cheating, but it makes life a lot easier.

The Signal/Sigali proof. Signal has a small number of constructs, recalled in Fig. 9. Using the same abstractions as for Lustre, the proof in Signal/Sigali is shown in Fig. 7 and Fig. 8. There are some differences with the Lustre/Lesar proof. Firstly, unknown but short delays at ingress/egress of the bus are included (the shift processes). Also, the process fifo_2 is a cascade of 1-bounded fifos, so we know that it satisfies the requirements of theorem 1. This fifo_2 has an unspecified delay, hence we can synchronize it with the output of the protocol, xr, and check whether they are both equal; this is performed in process equal. In fact, this proof says that the cascade of two 1-bounded fifos is a correct abstraction of the protocol, so the protocol satisfies the requirements of theorem 1. This style of proof deeply uses the capability, for Signal, of handling nondeterministic systems, and thus of emulating asynchrony.

5 Discussion

In this section, we discuss how our LTTA protocol can be extended to multiple users. Then we briefly compare LTTA with TTA.

Extension of the protocol to multiple users. Consider the case of several pairs $\{\text{writer}, \text{reader}\}$, transmitting several message sequences over the LTTA bus. Assume, for instance, that the bus scans periodically the output buffers of all writers.

Focus on one particular writer. For the analysis, we cluster together all readers into an overall "all_reader". Call $t^{\mathbf{b}}(n)$ the sequence of instants at which the bus fetches messages from the output buffer of *this* writer, to the input buffer of *some* reader. Equivalently, $t^{\mathbf{b}}(n)$ is the sequence of instants at which the bus fetches messages from the output buffer of *this* writer, to the (virtual) input buffer of *all_reader.*

Just apply Theorems 1 or 2, to the pair $\{\text{this writer}, \text{all\_reader}\}$. If the (quasi) period b driving the $t^{\mathbf{b}}(n)$'s satisfies the assumptions of the theorems, then the protocol is correct for this pair. This implies that the protocol is also correct for $\{\text{this writer}, \text{some\_reader}\}$, where some_reader denotes one particular reader. Assume that there are J users, and each user writes with period w and reads with period r, then it is enough that the actual bus period satisfies the conditions (6) of theorem 1 with b/J substituted for b.

Clearly, the choice of periodic scanning is the default choice. It can be adapted if the writing periods are different, for different users, in order to ensure proper balancing. Such quantitative issues are beyond the scope of this paper. But clearly, the theorem can be accomodated to such adaptation.

A brief comparison with Kopetz'TTA. We warn the reader that this brief discussion is by no means authoritative: closer investigations would be required for firm assessment and comparisons.

This being said, the first remark is that LTTA does not require implementing a clock synchronization algorithm, and our protocol is cheap. On the other hand, when designing the system based on LTTA, the engineer must consider issues of relative speed of writing/bus/reading. This difficulty does not appear with TTA, at a first glance. However, we think that the timing considerations when using LTTA, are quite natural for the designer. The bottom line is that, from the strict point of view of synchronization, LTTA seems an attractive approach.

The very question is that of fault tolerance. As extensively discussed by John Rushby [16], Kopetz'TTA takes advantage of the strict TTA, in order to provide fault tolerance and allow for a strict separation of different functions multiplexed over the same bus. The corresponding issue is certainly carefully considered by the industrial users of the LTTA approach, but we must say that we did not study it in detail.

6 Conclusion

We have presented a weakened form of time-triggered architecture, we called it *loosely* time-triggered. In contrast to Kopetz'TTA, clocks are periodic but are *not* synchronized. Strict synchronization may not be useful for continuous control, since modern control design is anyway robust against phase uncertainties—this justifies considering LTTA for real-time embedded control. We have proposed a protocol that offers, on the top of LTTA, a coherent system of logical clocks, and we have sketched how this can be used for correct deployment of controller designs based on synchronous languages. This protocol is of interest per se, as it accepts clocks with offset and jitter. But the way we have analysed the protocol is also of interest, it illustrates the use of synchronous languages for modeling asynchronous architectures. Still, a complete *automatic* proof of our theorems with their exact assumptions formulated in quantitative terms, is to be done.

This study was a first attempt toward designing distributed embedded systems that are robust against imperfect synchronization of the architecture. One important remaining question is the following: can LTTA offer lower cost fault-tolerance, as compared to TTA? This is not obvious, since Rushby [16], advocates that strict compliance with time is the key to fault-tolerance in TTA.

References

1. R. Alur, T.A. Henzinger, and M.Y. Vardi. Parametric Real-time Reasoning. In *Proc. of the 25th Annual Symposium on Theory of Computing (STOC),* ACM Press, 1993, pp. 592-601.
2. R. Bannatyne. Time Triggered Protocol: TTP/C, *Embedded Systems Programming,* 9/98, pp. 52-54.
3. A. Benveniste, B. Caillaud, and P. Le Guernic. From synchrony to asynchrony. In J.C.M. Baeten and S. Mauw, editors, *CONCUR'99, Concurrency Theory, 10th International Conference*, volume 1664 of *Lecture Notes in Computer Science*, pages 162–177. Springer, August 1999.
4. A. Benveniste, B. Caillaud, and P. Le Guernic. Compositionality in dataflow synchronous languages: specification & distributed code generation. *Information and Computation,* 163, 125-171 (2000).
5. A. Benveniste. Some synchronization issues when designing embedded systems from components. In *Proc. of 1st Int. Workshop on Embedded Software, EMSOFT'01,* T.A. Henzinger and C.M. Kirsch Eds., LNCS 2211, 32–49, Springer Verlag, 2001.
6. P. Caspi, C. Mazuet, R. Salem, and D. Weber. Formal design of distributed control systems with lustre. In *Proc. Safecomp'99*, September 1999.
7. P. Caspi. Embedded control: from asynchrony to synchrony and back. In *Proc. of 1st Int. Workshop on Embedded Software, EMSOFT'01,* T.A. Henzinger and C.M. Kirsch Eds., LNCS 2211, 80–96, Springer Verlag, 2001.
8. C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool Kronos. In *Proceedings of "Hybrid Systems III, Verification and Control",* 1996. Lecture Notes in Computer Science 1066, Springer-Verlag.
9. P. Le Guernic, T. Gautier, M. Le Borgne, C. Le Maire. Programming Real-Time Applications with Signal. *Proceedings of the IEEE,* 79(9):1321-1336, September 1991.
10. N. Halbwachs, F. Lagnier and P. Raymond. Synchronous observers and the verification of reactive systems. In *Third Int. Conf. on Algebraic Methodology and Software Technology, AMAST'93,* Twente, M. Nivat and C. Rattray and T. Rus and G. Scollo, Eds., Workshops in Computing, Springer Verlag. Jun. 1993.
11. H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications.* Kluwer Academic Publishers. 1997. ISBN 0-7923-9894-7.
12. L. Lamport. Time, clocks and the ordering of events in a distributed system. *Communication of the ACM*, 21:558–565, 1978.
13. Kim G. Larsen, P. Pettersson, and Wang Yi. UPPAAL in a Nutshell. In *Springer International Journal of Software Tools for Technology Transfer,* 1(1-2), 134–152, Dec. 1997.
14. H. Marchand, E. Rutten, M. Le Borgne, M. Samaan. Formal Verification of SIGNAL programs: Application to a Power Transformer Station Controller. *Science of Computer Programming,* 41(1):85–104, Aug. 2001.
15. M. Pease, R.E. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–237, 1980.
16. J. Rushby. Bus architectures for safety-critical embedded systems. In *Proc. of 1st Int. Workshop on Embedded Software, EMSOFT'01,* T.A. Henzinger and C.M. Kirsch Eds., LNCS 2211, 306–323, Springer Verlag, 2001.

```
const n = 3;   -- the input is a bit stream of width 3

node writer(x : bool^n) returns (xw: bool^n; bw: bool);
let
    bw = true → pre not bw;
    xw = x;
tel

const init = false^n;

node reader(x: bool^n; b: bool) returns (cro: bool; xr: bool^n);
let
    cro = not (b = (false → pre b));
    xr  = if cro then x
          else (init → pre xr);
tel

node bus(xw: bool^n; bw: bool) returns (xr: bool^n; br: bool);
let
    xr, br = (xw, bw);
tel

node faster(cb, cw: bool) returns (prop: bool);
var w_before_b: bool;
let
    w_before_b = if cw then true
                 else if cb then false
                      else (false → pre w_before_b);
    -- tells that there is an unmatched cw
    prop = not (cw and (false → pre w_before_b));
-- this node implements (17)
tel

node firstafter(cb, cw: bool) returns (cbw: bool);
var waiting: bool;
let
    cbw = cb and (false → pre waiting) ;
    waiting = if cw then true
              else if cbw then false
                   else (false → pre waiting);
-- this node implements (9)
tel
```

Figure5. The Lustre/Lesar proof of theorem 1, part 1.

```
node vecteq(xw: bool^n; xr: bool^n) returns (prop: bool);
var aux: bool^(n+1);
let
    aux[0] = true;
    aux[1..n] = aux[0..n-1] and (xr = xw);
    prop = aux[n];
tel

node compare(cw: bool; xw: bool^n; xr: bool^n) returns (prop: bool);
var equal: bool; last: bool^n; unmatched: bool;
let
    last = if equal then xw else (init → pre last);
    – stores the value to be matched
    equal = vecteq(xr, (init → pre last));
    – tells whether the value to be matched is actually matched
    unmatched = if cw and not (true → pre equal) then true
                     else if equal then false
                           else false → pre unmatched;
    – tells that there are two values waiting for match
    prop = not(cw and (false → pre unmatched));
    – a new value should not arrive while two values are waiting for match
tel

node verif(cw, cb, cr: bool; (x: bool^n) when cw)
returns (prop: bool; xw, xr, xro: bool^n; bw, br: bool; cro: bool );
let
    xw, bw = if cw then current writer(x)
                 else ((init, false) → pre(xw, bw));
    xr, br =   if cb then current bus((xw, bw) when cb)
                 else ((init, false) → pre(xr, br));
    cro, xro =if cr then current reader((xr, br) when cr)
                 else ((false, init) → pre(cro, xro));

    prop = compare(cw, xw, xro);

    assert faster(cb, cw) and faster(cr, firstafter(cb, cw));
    – these assertions implement (16) and (17)
    assert #(cw, cb, cr);
    – so as not to get bored by simultaneous clocks
tel

(*
moucherotte% lesar albert2.lus verif
–Pollux Version 2.0
TRUE PROPERTY
moucherotte%
)
```

Figure6. The Lustre/Lesar proof of theorem 1, part 2

```
process protocol = (? boolean xw; event cw, cb, cr ! boolean xr , inv)
    (| (xb, bb, sbw) := bus (xw, writer(xw,cw), cb)          % writer + bus %
     | (xr, br, sbb)  := reader (xb, bb, cr)                 % reader %
     | cb ^= sbw default cb                                  % condition (16) %
     | cr ^= (when switched(sbb)) default cr                 % condition (17) %
     | xok := fifo_2 (xw)                                    % fifo_2 satisfies (4) %
     | inv  := equal (xok, xr)                               % tests if xok=xr %
     |) where boolean bw, xb, bb, sbw, sbb, br, xok;

    process writer = (? boolean xw; event cw ! boolean bw)
        (| bw ^= xw ^= cw
         | bw := not (bw$1 init true)
         |);                                                 % bw: boolean flag %
   process bus = (? boolean xw, bw; event cb ! boolean xb, bb, sbw)
        (| (xb, bb, sbw) := buffer (xw, bw, cb) |);
   process reader = (? boolean xb, bb; event cr ! boolean xr, br, sbb)
        (| (yr, br, sbb) := buffer (xb, bb, cr) | xr := yr when switched (br) |)
        where boolean yr; end;                               % switched(br) validates xr %

   process switched = (? boolean b ! boolean c)
        (| zb := b$1 init true | c := (b and not zb) or (not b and zb) |)
        where boolean zb; end;                               % c=true when b alternates %

   process buffer = (? boolean x, b ; event c ! boolean bx, bb, sb)
        (| (sx, sb) := shift_2 (x, b) | (bx, bb) := current_2 (sx, sb, c) |)
        where boolean sx; end;                               % delays, sustains, filters %

   process shift_2 = (? boolean x, b ! boolean sx, sb)       % see shift_1 %
        (| (sx, sb) := current_2 (x, b, ^sb) | interleave (x, sx) |);
      process current_2 = (? boolean wx, wb; event c ! boolean rx, rb)
           (| rx := (wx cell c init false) when c
            | rb := (wb cell c init true) when c |);         % see current_1 %
      process interleave = (? boolean x, sx ! )
           (| x ^= when b | sx ^= when not b | b := not (b$1 init false) |)
           where boolean b; end;                             % x and sx interleave %

   process equal = (? boolean y, z ! boolean inv)
        (| i := (y and z) or (not y and not z) default inv
         | inv := i $1 init true
         |); where boolean i; end;                           % tests if y=z %

   process fifo_2 = (? boolean x ! boolean xok )
        (| xok := shift_1(shift_1(x)) |);
      process shift_1 = (? boolean x ! boolean sx)           % x,sx satisfy (4) %
        (| sx := current_1 (x, ^sx) | interleave (x, sx) |);
         process current_1 = (? boolean wx; event c ! boolean rx)
           (| rx := (wx cell c init false) when c |);        % current triggered by c %

end;
```

Figure7. The Signal/Sigali proof

Sigali:

——————————————

```
set_reorder(1);
read("protocol.z3z");
read("Creat_SDP.z3z");
read("Verif_Determ.z3z");
POSSIBLE(B_False(S,inv)); → resultat False
Always(B_True(S,inv)); → resultat True
```

——————————————

Figure8. The Sigali script.

$z := x$ op y	$z_\tau \neq \perp \Leftrightarrow x_\tau \neq \perp \Leftrightarrow y_\tau \neq \perp$, $\forall k : z_k = \mathrm{op}(x_k, y_k)$
y ^$=x$	$y_\tau \neq \perp \Leftrightarrow x_\tau \neq \perp$ (x and y possess identical clocks)
^x	the clock of x : ^$x \in \{\mathrm{T}, \perp\}$, ^$x_\tau \neq \perp \Leftrightarrow x_\tau \neq \perp$
$y := x\$1$ init x_0	$x_\tau \neq \perp \Leftrightarrow y_\tau \neq \perp$, $\forall k > 1 : y_k = x_{k-1}, y_1 = x_0$ (delay)
$x := u$ when b	$x_\tau = u_\tau$ when $b_\tau = \mathrm{T}$, otherwise $x_\tau = \perp$
$x := u$ default v	$x_\tau = u_\tau$ when $u_\tau \neq \perp$, otherwise $x_\tau = v$
(| P | Q |)	parallel composition
$y := x$ cell h init x_0	(| $y := x$ default ($y\$1$ init x_0) | ^$y :=$ ^x default h |)

Figure9. Signal operators (left), and their meaning (right). In this table, x_τ denotes the status (absence, or actual value) of signal x in an arbitrary reaction τ, $\{\mathrm{T}, \mathrm{F}\}$ is the boolean domain, and the special value $\perp$ denotes absence in the considered reaction.