Formal proof of a polychronous protocol for loosely time-triggered architectures

Mickaël Kerbœuf[1], David Nowak[2], and Jean-Pierre Talpin[1]

[1] IRISA & INRIA Rennes ({kerboeuf,talpin}@irisa.fr)

[2] LSV, CNRS & ENS Cachan (nowak@lsv.ens-cachan.fr)

Abstract. The verification of safety-critical systems has become an area of increasing importance in computer science. The notion of reactive system has emerged to concentrate on problems related to the control of interaction and response-time in mission-critical systems. Synchronous languages have proved to be well-adapted to the verification of reactive systems. It is nonetheless commonly argued that real-life systems often do not satisfy the strong hypotheses assumed by the synchronous approach: they are not synchronous. Protocols have however been proposed (e.g. in [1]) to provide an abstract synchronous specification on top of real-time architectures (e.g. loosely time-triggered architectures or LTTA). This abstract model is designed so as to satisfy the synchronous hypotheses and meet the implementation architecture constraints. It makes it possible to design, specify and verify reactive systems in the context of the synchronous approach. In this aim, the present article formalizes the LTTA protocol in the theorem prover Coq and proves its correctness.

1 Introduction

The synchronous approach. The verification of safety-critical systems has become an area of increasing importance in computer science because of the constant progression of software developments in sensitive fields like medicine, communication, transportation and (nuclear) energy. The notion of reactive system has emerged to concentrate on problems related to the control of interaction and response-time in mission-critical systems. These strong requirements lead to the development of specific programming languages and related verification tools for reactive systems. The verification of a reactive system can be done by elaborating a discrete model of the system (i.e. as a finite-state machine) specified in a dedicated language (e.g. a synchronous programming language) and then by checking a property against the model (i.e. model checking). Model checking has been used at an industrial scale.

The Coq proof-assistant. When a property involves parameters or non-linear numerical terms, its verification by model checking is not straightforward and can sometimes be tedious. Another possibility to verify a reactive system is the use of a theorem prover such as Coq [9]. For instance, the semantics of the synchronous language Signal [8] has been formalized in Coq and the correctness of a steam-boiler implemented in Signal has been proved [6]. Coq [9] is a proof-assistant for higher-order logic. It allows the development of computer programs that are consistent with their formal specification. The logical language used in Coq is a variety of type theory, the *Calculus of Inductive Constructions* [10]. Due to the high expressive capability of this logic, proofs in Coq requires human-interaction to *direct* the strategy. The prover can nonetheless automate its most tedious and mechanical parts. Indeed, decisions procedures are implemented.

Synchronous languages (e.g. Esterel [3], Lustre [5], Signal [2]) have proved to be well adapted to the verification of reactive systems. Unfortunately, real systems often do not satisfy the strong hypotheses assumed by the synchronous approach: they are not synchronous.

Loosely Time-Triggered Architectures. A distributed real-time control system has a time-triggered nature just because the physical system for control is bound to physics. A loosely time-triggered architecture (LTTA) is one in which:

- Bus access is quasi-periodic and non-blocking
- Read and write operations are independent
- Values are sustained by the bus and periodically refreshed.

The clock rates at which data are, written to, updated by, read from the bus are not synchronous: a LTTA is a multi-clocked control system in which clocks are moreover bound to *physical* time and deviate one from each others. Here, the term *polychronous* refers to this multi-clocked feature. The LTTA has been extensively investigated in [4] and used in several major industries.

Logical clocks on top of LTTAs. That is why a protocol is proposed in [1] which provides an abstract level on top of an LTTA. This abstract level is such that the the synchronous hypotheses are satisfied. It is then possible to design, specify and verify reactive systems in the context of the synchronous approach.

Outline. In Section 2, we describe the protocol. Section 3 is devoted to previous work, especially partial proofs of the protocol by model checking. In Section 4, we explain our formalization in Coq, and we show in section 5 how this approach can be used as a generic formal framework to prove other implementations. Finally, we conclude in Section 6.

2 Description of the protocol

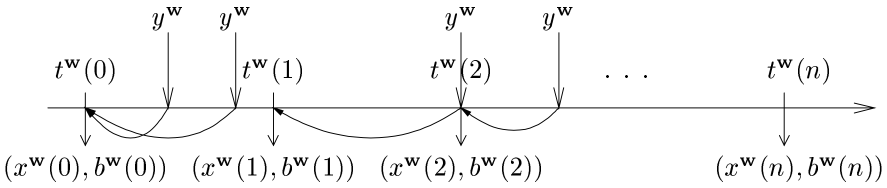
The LTTA is composed of three devices, a *writer,* a *bus,* and a *reader*. Each device $\mathbf{d}$ is activated by its own, approximately periodic, clock (denoted by a function $t^{\mathbf{d}}$).

Writer. At the nth clock tick (time $t^{\mathbf{w}}(n)$), the *writer* generates the value $x^{\mathbf{w}}(n)$ and an alternating flag $b^{\mathbf{w}}(n)$ s.t.:

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

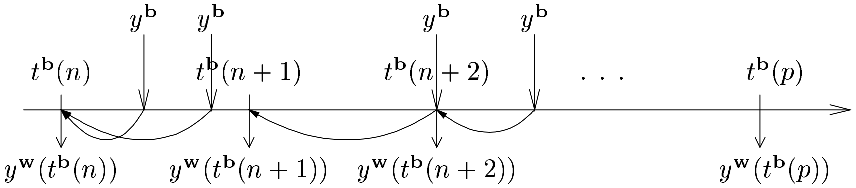
Both values are stored in its output buffer, denoted by $y^{\mathbf{w}}$. At any time t, the writer's output buffer $y^{\mathbf{w}}$ contains the last value that was written into it:

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



Bus. At $t^{\mathbf{b}}(n)$, the *bus* bus fetches $y^{\mathbf{w}}$ to store in the input buffer of the reader, denoted by $y^{\mathbf{b}}$. Thus, at any time t, the reader input buffer is defined by:

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

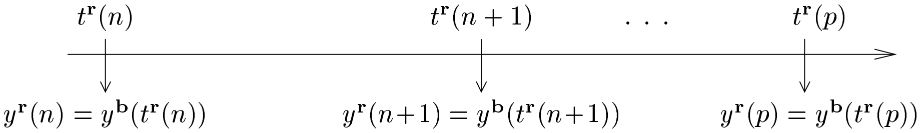


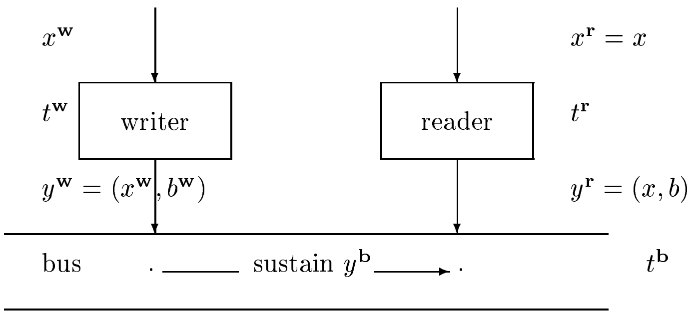
Reader. At $t^{\mathbf{r}}(n)$, the *reader* loads the input buffer $y^{\mathbf{b}}$ into the variables $x(n)$ and $b(n)$:

$$y^{\mathbf{r}} = (x(n), b(n)) = y^{\mathbf{b}}(t^{\mathbf{r}}(n)) \tag{3}$$

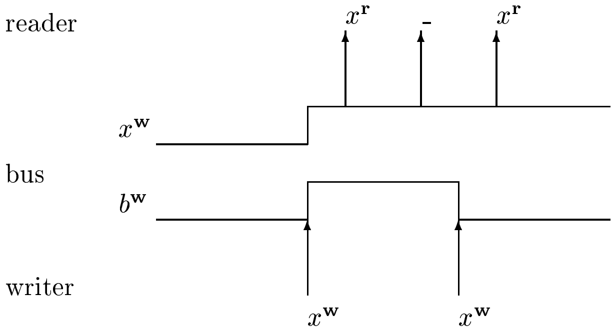
Then, in a similar manner as for an alternating bit protocol, the reader extracts $x(n)$ iff $b(n)$ has changed. This is by the sequence m of ticks where b changes:

$$\begin{aligned} m(0) = 0 \text{ , } m(n) &= \inf\{k > m(n-1) \mid b(k) \neq b(k-1)\} \\ x^{\mathbf{r}}(k) &= x(m(k)) \end{aligned} \tag{4}$$





Example. We illustrate the protocol by the following picture. Notice the role of the flag b: if the writer sends the same value along $x^{\mathbf{w}}$ twice, the boolean flag switch ensures that this value will be read twice on $x^{\mathbf{r}}$. On the opposite, if the value is sent once along $x^{\mathbf{w}}$ and read twice along $x^{\mathbf{r}}$, the boolean flag samples the excess of reading.



Flag switches are detected by the reader by a non predictable but bounded delay according to *physical* time: perfect physical synchrony is lost.

Correctness of the protocol. We define here the expected behavior. In any execution of the protocol, the sequences $x^{\mathbf{w}}$ and $x^{\mathbf{r}}$ must coincide, i.e.,

$$\forall n \cdot x^{\mathbf{r}}(n) = x^{\mathbf{w}}(n) \tag{5}$$

In order to prove the correctness of the protocol, we need to prove that, under some hypotheses on the clocks, the property (5) is true.

3 Previous work

In [1], the following theorem is proved by hand.

Theorem 1 (sampling theorem). *The LTTA protocol satisfies the property (5) iff the following conditions hold:*

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

where w, b and r are the respective periods of the clocks of the writer, the bus and the reader, and where, for $x \in \mathbb{R}$, $\lfloor x \rfloor$ denotes the largest integer less or equal to x.

Since $w \geq b$ then $w/2b < \lfloor w/b \rfloor$. Also note that, if w/b is large then $\lfloor w/b \rfloor \leq w/b$ and $\lfloor w/b \rfloor \sim w/b$. Hence, if $b \sim 0$ (i.e. the bus is fast), then the conditions of theorem 1 reduce to:

$$w \gg b \, , w > r.$$

In [1], it was shown using symbolic model-checking that a discrete SIGNAL model of the LTTA protocol (i.e. a finite-state approximation of the actual protocol) satisfied the desirable requirement of ensuring a coherent distribution of clocks. However, the assumptions ensuring correctness of the actual LTTA protocol are quantitative in nature (tolerance bounds for the relative periods, and time variations, of the different clocks). 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.

In order to allow for standard model checking techniques to be used, two kinds of abstractions of the protocol are necessary:

- It is clear that this protocol and the property to be verified are data-independent w.r.t. the type X of data which is transmitted. Therefore, it is sufficient to verify this protocol with a finite set of finite instantiations of the type X. It is then possible to deduce the correctness of the protocol for any instantiation of the type X, by applying theorems proved in [7]. However,in [1], only the instantiation of X by the type of booleans is considered. It is not proved and not evident that the correctness of the protocol for this instantiation is sufficient to prove the correctness of the protocol for any instantiation of X.
- Conditions (6) are abstraction by conditions on ordering between events. The first condition, $w \geq b$, is abstracted by the predicate:

  $$w \geq b \leftrightarrow \text{never two } t^{\mathbf{w}} \text{ between two } t^{\mathbf{b}}. \tag{7}$$

  The abstraction of the second condition, $\lfloor w/b \rfloor \geq r/b$ requires the following definition of the first instant (of the bus) $\tau^{\mathbf{b}}(n)$ where the bus can fetch the nth writing:

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

  The second condition is then restated as the requirement (8) that no two successive $\tau^{\mathbf{b}}$ can occur between two successive $t^{\mathbf{r}}$:

  $$\left\lfloor \frac{w}{b} \right\rfloor \geq \frac{r}{b} \leftrightarrow \text{never two } \tau^{\mathbf{b}} \text{ between two successive } t^{\mathbf{r}}. \tag{8}$$

This verification has been done twice: with Lustre and its model checker Lesar; and with Signal and its model checker Sigali.

4 Abstraction and formalization in Coq

We investigate the use of the theorem prover Coq as a general formal framework for any implementation of the protocol for LTTAs. In this section we describe our formalization in Coq. The translation of the specification is quite straightforward. We introduce some syntactical elements of Coq to illustrate this point.

4.1 Data, Time and Clocks

Data The type of data is seen as an abstract data type $\mathcal{D}$ (`Data` in Coq). We do not need any relation or hypothesis on this type. It was not the case in the proof by model checking [1] where $\mathcal{D}$ was supposed to be the type of booleans.

```
Parameter Data : Set.
```

Physical time. Physical time is also seen as an abstract data type i.e., a type $\mathcal{T}$ (`Time` in Coq), a binary predicate $\leq$ (`time_le` in Coq) and the assumption that $\leq$ is reflexive, transitive and total. We do *not* assume time is discrete. These are the only hypotheses on physical time we need for our proof.

$$\forall t \in \mathcal{T}, t \leq t$$
$$\forall t_1, t_2, t_3 \in \mathcal{T}, t_1 \leq t_2 \leq t_3 \;\Rightarrow\; t_1 \leq t_3$$
$$\forall t_1, t_2 \in \mathcal{T}, t_1 \leq t_2 \;\vee\; t_2 \leq t_1$$

In Coq, it is written

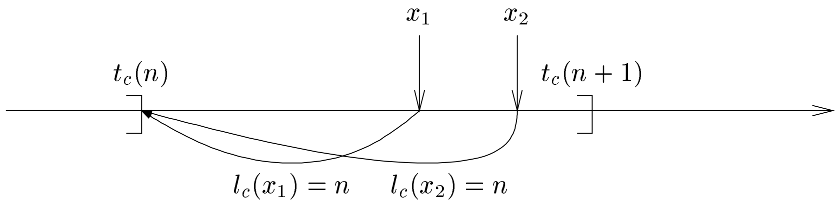
```
Variable Time : Type.
Variable time_le : Time->Time->Prop.
Hypothesis time_le_reflexive :
 (t:Time)(time_le t t).
Hypothesis time_le_transitive :
 (t1,t2,t3:Time)(time_le t1 t2)->(time_le t2 t3)->(time_le t1 t3).
Hypothesis time_le_total :
 (t1,t2:Time)(time_le t1 t2)(time_le t2 t1).
```

The keywords `Variable` and `Hypothesis` mean that it will be possible to instantiate those type (for example, by the type of reals available in Coq), relation and hypotheses in order to obtain specializations of the proved theorems. We will then be able to prove stronger theorems depending on particular instantiations.

Clocks. A clock c is modeled by two (possibly partial) functions. The first one, t_c (`time` in Coq), maps any natural number n in its domain to the instant $t \in \mathcal{T}$ when nth sampling tick occurs. The only assumption on this function is that it is strictly monotonic (`monotonicity` in Coq). The second function, l_c (`lTick` in Coq), maps any time $t \in \mathcal{T}$ to the number of the occurrence of the tick which immediately precedes the instant t. It is defined relationally by its characteristic property (`current_tick` in Coq):

$$\forall n \in \mathbb{N} \;\cdot\; \forall x \in \mathcal{T} \;\cdot\; t_c(n) < x \leq t_c(n+1) \;\Leftrightarrow\; n = l_c(x)$$

This function enables to access the value carried by the writer or by the bus at the *last tick* of its clock.



For instance, if t_c stands for $t^{\mathbf{b}}$, then $l_c(x)$ (noted $l_{\mathbf{b}}(x)$ in this case) correspond to $\sup\{n' \mid t^{\mathbf{b}}(n') < x\}$. Thus, we have:

$$\begin{aligned} y^{\mathbf{b}}(x) &= y^{\mathbf{w}}(t^{\mathbf{b}}(n))\,, \text{ where } n = \sup\{n' \mid t^{\mathbf{b}}(n') < x\} \\ &= y^{\mathbf{w}}(l_{\mathbf{b}}(x)) \end{aligned}$$

A clock c is defined by the functions t_c and l_c. In Coq the type of clock is defined by a structure which embeds `time` (t_c), `lTick` (l_c) and two characteristic properties of t_c and l_c, namely `monotonicity` and `current_tick`.

```
Record Clock : Type := {
  time :> nat->Time;
  monotonicity : (n,n':nat)(lt n n')->(time_lt (time n) (time n'));
  lTick : Time->nat;
  current_tick :
   (n:nat; t:Time)
   (time_lt (time n) t)/\(time_le t (time (S n))) <-> n=(lTick t)
}.
```

The character “>” is used for the convenient mechanism of implicit coercion provided by Coq. Suppose that `c` is of type `Clock`. `c` is a record and not a function. Anyway we can apply it and Coq will instead apply the field `time` f the record `c`. It means we can simply write `c n` instead of `time c n`. It improves the readability.

Some useful results about monotonicity follow from the definition of clocks:

$$\begin{aligned} &\forall c, \forall n, \forall n' \ (t_c(n) < t_c(n')) \Rightarrow (n < n') \\ &\forall c, \forall n, \forall n' \ (n \leq n') \Rightarrow (t_c(n) \leq t_c(n')) \end{aligned}$$

They are stated and proved in Coq:

```
Lemma monotonicity_inv :
  (c:Clock; n,n':nat)
  (time_lt (c n) (c n'))->(lt n n').
Lemma monotonicity_le :
  (c:Clock; n,n':nat)
  (le n n')->(time_le (c n) (c n')).
```

The following lemma is a fundamental property of l_c (`lTick` in Coq). It follows from its characteristic property. It guarantees that at any time t, $l_c(t)$ actually occurs after (or at the same time as) any tick n which itself occurs before t: $\forall c, \forall n, \forall t,\ (t_c(n) < t) \Rightarrow (t_c(n) \leq t_c(l_c(t)))$.

```
Lemma following_ticks :
  (c:Clock; n:nat;t:Time)
  (time_lt (c n) t) -> (time_le (c n) (c (lTick c t))).
```

4.2 Writer, bus and reader

Following strictly definitions from [1], the three devices are formalized as follows:

```
Variable tw : Clock.
Variable xw : nat->Data.
Fixpoint  bw [n:nat] : bool :=
  Cases n of
    O => false
  | (S p) => (negb (bw p))
  end.
Definition yw_x [t:Time] : Data := (xw (lTick tw t)).
Definition yw_b [t:Time] : bool := (bw (lTick tw t)).
```

We assume a clock `tw` for the writer and a sequence of values it writes `xw`. `bw` is the sequence of alternating booleans. It is used in order to implement the alternating bit protocol. `yw_x` (respectively, `yw_b`) maps a time $t \in \mathcal{T}$ to the last written value of `xw` (respectively, `bw`) at this time t.

We assume a clock `tb` for the bus. We define `yb_x` (respectively, `yb_b`) which maps a time $t \in \mathcal{T}$ to the last value (respectively, boolean) received by the bus at the time t.

```
Variable tb : Clock.
Definition yb_x [t:Time] : Data := (yw_x (tb (lTick tb t))).
Definition yb_b [t:Time] : bool := (yw_b (tb (lTick tb t))).
```

We assume a clock `tr` for the reader. We define `x` (respectively, `y`) to be the nth received value (respectively, boolean) by the reader.

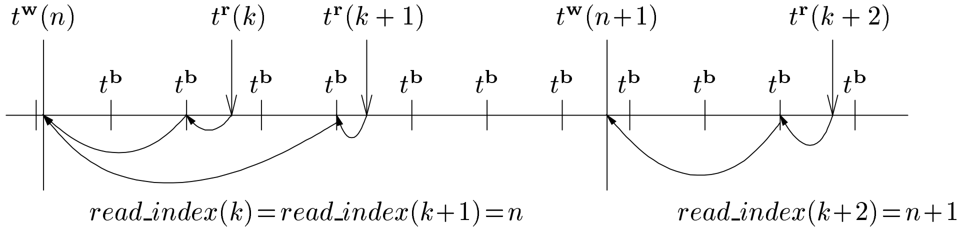
```
Variable tr : Clock.
Definition x [n:nat] : Data := (yb_x (tr n)).
Definition b [n:nat] : bool := (yb_b (tr n)).
```

4.3 Abstraction

Two kinds of abstractions are needed for the automatic proofs of the protocol. The carried data are restricted to finitely enumerated types, and the quantitative assumptions (6) are abstracted into event ordering assumptions (7 and 8). In our approach, the first abstraction is avoided (thanks to the generic type $\mathcal{D}$), but we deliberately keep the second one. It appears to be more general than the initial statement. Indeed, whatever the respective quasi-periods of the writer, the bus and the reader (w, b and r) may be, it ensure all written values are actually fetched by the bus, and then read by the reader. Moreover, we aim at defining in Coq a kind of meta-model for data-flow encodings (like the Lustre and Signal ones proposed in [1]).

In order to be more general, and for more legibility, we did not introduce either the counter of bit alternations detected by the reader, nor the sequence $x^{\mathbf{r}}$ of validated values. Actually, b must be specified only for automatic proofs of the protocol. In this case, the written *values* and the read *values* are related, hence the necessity to implement a mechanism for values discrimination on the reader's side. Here, we aim at validating the protocol whatever the mechanism b for discrimination may be. In Coq, we can relate the *instants* when a value is written with the *instants* when a value is read. We suppose a part of the transmitted values enables the reader to sample correctly the received values. This is the case with b. Then, the correctness property (5) which handles *values* follows.

We define a function called *read_index* which maps to a given reading tick k the writing tick *read_index*(k) corresponding to the instant (on the writer's clock) when the writer emitted the value that can be read at the instant k (on the reader's clock). The following figure illustrates this function:



This function is defined as follows:

$$read\_index \ : \ \left| \begin{array}{l} \mathbb{N} \to \mathbb{N} \\ k \mapsto l_{\mathbf{w}}(t^{\mathbf{b}}(l_{\mathbf{b}}(t^{\mathbf{r}}(k)))) \end{array} \right.$$

The moment of the kth reading tick is $t^{\mathbf{r}}(k)$. The last tick on the bus at this time $(l_{\mathbf{b}}(t^{\mathbf{r}}(k)))$ occurs at $t^{\mathbf{b}}(l_{\mathbf{b}}(t^{\mathbf{r}}(k)))$. The carried value at that time corresponds to the value sent by the writer at its previous tick: $l_{\mathbf{w}}(t^{\mathbf{b}}(l_{\mathbf{b}}(t^{\mathbf{r}}(k))))$. According to the protocol statement, we actually have the following relation:

$$\begin{aligned} \forall k \in \mathbb{N},\ & (x(k), b(k)) \\ & = y^{\mathbf{b}}(t^{\mathbf{r}}(k)) \\ & = y^{\mathbf{w}}(t^{\mathbf{b}}(l_{\mathbf{b}}(t^{\mathbf{r}}(k)))) \\ & = \big(\ x^{\mathbf{w}}(l_{\mathbf{w}}(t^{\mathbf{b}}(l_{\mathbf{b}}(t^{\mathbf{r}}(k))))) \ ,\ b^{\mathbf{w}}(l_{\mathbf{w}}(t^{\mathbf{b}}(l_{\mathbf{b}}(t^{\mathbf{r}}(k)))))\ \big) \\ & = (x^{\mathbf{w}}(read\_index(k)), b^{\mathbf{w}}(read\_index(k))) \end{aligned}$$

Now, we focus on *read_index*, which relates the *instants* when a value is written with the *instants* when a value is read. To prove the correctness of the protocol, we only have to prove that *read_index* is increasing, and that it covers $\mathbb{N}$ (so that all written values are actually read):

$$\begin{array}{l} \forall k_1, k_2 \in \mathbb{N},\ k_1 < k_2 \Rightarrow read\_index(k_1) \leq read\_index(k_2) \\ \quad \forall n \in \mathbb{N},\ \exists k \in \mathbb{N} \text{ st. } n = read\_index(k) \end{array}$$

Thus, all written values are actually read (and possibly more than once) in a correct order. Whatever the mechanism b for discrimination may be, it is possible to validate $x(0)$ and each $x(k+1)$ such that $b(k+1)$ and $b(k)$ are different.

$$\forall n \in \mathbb{N},\ \exists k \in \mathbb{N},\ \text{st. } x^{\mathbf{w}}(n) = x(k)\ \wedge\ b^{\mathbf{w}}(n) = b(k)$$

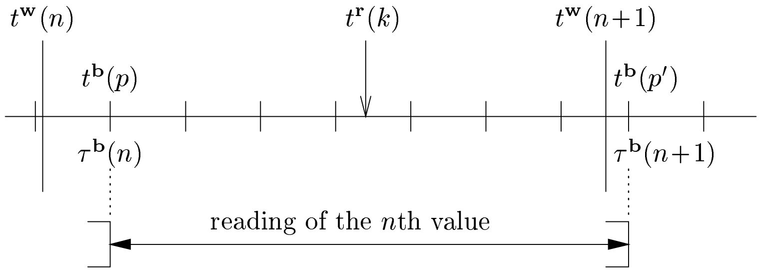
The property (5) follows when $\forall k \in \mathbb{N},\ b^{\mathbf{w}}(k+1) \neq b^{\mathbf{w}}(k)$. It is actually the case with the alternating bit protocol.

4.4 Correctness of the protocol

This result holds under the specific conditions (7) and (8). We state them in Coq with the unique following assumption:

$$\forall n \in \mathbb{N},\ \exists k \in \mathbb{N},\ \text{st. } \tau^{\mathbf{b}}(n)\ <\ t^{\mathbf{r}}(k)\ \leq\ \tau^{\mathbf{b}}(n+1)$$

It guarantees that all written values are actually fetched by the bus ($\tau^{\mathbf{b}}(n)$ always exists, and $\tau^{\mathbf{b}}(n+1) \neq \tau^{\mathbf{b}}(n)$ since there is at least one instant $t^{\mathbf{r}}(k)$ which occurs in between them), and all fetched values are actually read by the reader ($\tau^{\mathbf{b}}(n) < t^{\mathbf{r}}(k) \leq \tau^{\mathbf{b}}(n+1)$). This assumption is illustrated by the following picture:



To state this condition, we formally define $\tau^{\mathbf{b}}$ as follows:

$$\forall n \in \mathbb{N},\ \exists k \in \mathbb{N},\ \text{st.} \begin{cases} \quad \tau^{\mathbf{b}}(n) = t^{\mathbf{b}}(k) \\ \wedge\ t^{\mathbf{w}}(n) < t^{\mathbf{b}}(k) \\ \wedge\ \forall k' \in \mathbb{N},\ k' < k \Rightarrow t^{\mathbf{b}}(k') \leq t^{\mathbf{w}}(n) \end{cases} \tag{9}$$

5 A formal framework for any implementation

Principles The Coq encoding of the protocol for LTTAs we described in the previous section can be seen as a high level abstracted implementation. It is founded on the smallest set of physical requirements (e.g. time is an abstract domain which only comes with a reflexive, transitive and total relation) and logical requirements (e.g. no two successive writing ticks can occur without a bus tick in between them). Thus, any other implementation must provide at least these requirements. Its correctness then follows.

We can refine this approach by adding an intermediate level between Coq and the analyzed implementation. This interface details the expected form of the time domain (variable $\mathcal{T}$ in Coq) and its order (`time_le` in Coq), and the data domain (variable $\mathcal{D}$ in Coq). It must also make explicit the clocks, and the first instant $\tau^{\mathbf{b}}(n)$ where the bus can fetch the nth writing. Then the hypotheses concerning the time domain must be proved, and the assumptions concerning the correctness must be restated. To prove the correctness of any implementation built upon the model denoted by the intermediate level, we only have to prove its specification implies the assumptions of its interface.

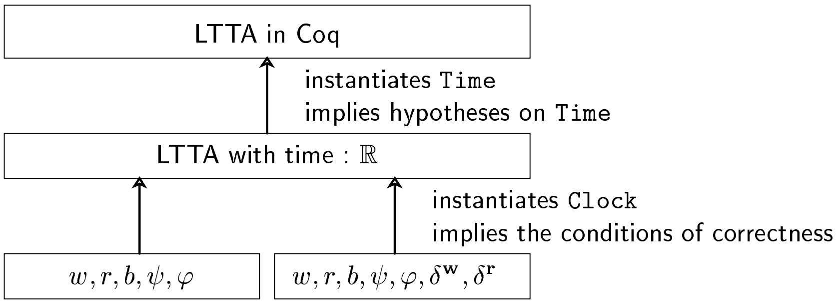
Examples Consider the manual proof of theorem 1 in [1]. It is built upon the explicit periods w, b and r (respectively of the writer, the bus and the reader) and the phases ψ and φ (respectively of the writer and the reader). The time domain (continuous) is denoted by $\mathbb{R}$. The logical statement of $\tau^{\mathbf{b}}$ (9) in Coq is implied by the functional statement $\tau^{\mathbf{b}}(n) = \lfloor (n+\psi)w \rfloor + 1$. In this approach, the correctness of the protocol comes from:

$$w \geq b \text{ , and } \left\lfloor \frac{w}{b} \right\rfloor \geq \frac{r}{b} \quad \Rightarrow \quad \forall n \in \mathbb{N},\ \exists k \in \mathbb{N},\ \text{st. } \tau^{\mathbf{b}}(n) \ < \ t^{\mathbf{r}}(k) \ \leq \ \tau^{\mathbf{b}}(n{+}1)$$

Now, in [1], another theorem is stated in order to take into account *approximately* periodic clocks. $t^{\mathbf{w}}$, $t^{\mathbf{b}}$ and $t^{\mathbf{r}}$ are restated including jitter terms $\delta^{\mathbf{w}}$ and $\delta^{\mathbf{r}}$ which denote the variations within a certain bound of w and b during execution. In this approach, the time domain and its order, the data domain and the clocks have the same nature as in the first approach without jitter. All we have to prove is the following property:

$$\begin{array}{l} w(1-2\delta^{\mathbf{w}}) \geq 1 \text{ , and } \lfloor w(1-2\delta^{\mathbf{w}}) \rfloor \geq r(1+2\delta^{\mathbf{r}}) \\ \Rightarrow \forall n \in \mathbb{N},\ \exists k \in \mathbb{N},\ \text{st. } \tau^{\mathbf{b}}(n) \ < \ t^{\mathbf{r}}(k) \ \leq \ \tau^{\mathbf{b}}(n{+}1) \end{array}$$

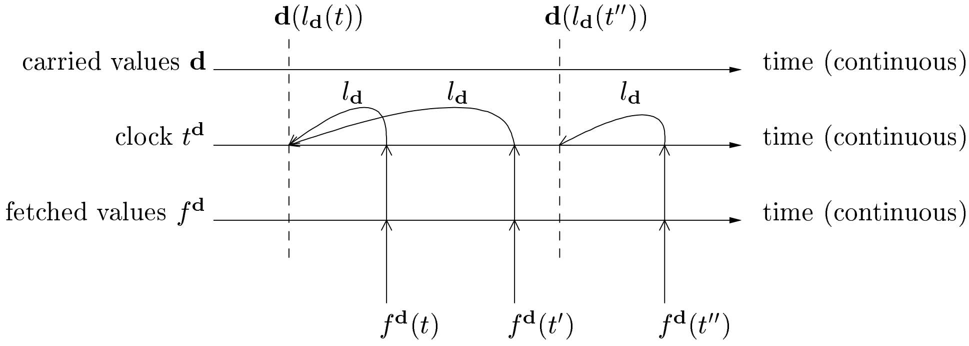
The following picture illustrates the use of the Coq approach as a generic formal framework to prove these two implementations:



5.1 LTTA in Signal

We illustrate here the same principle for the Signal solution suggested in [1]. We first define the intermediate level of any synchronous data-flow approach. Then, we show how proving the Signal implementation matches these requirements.

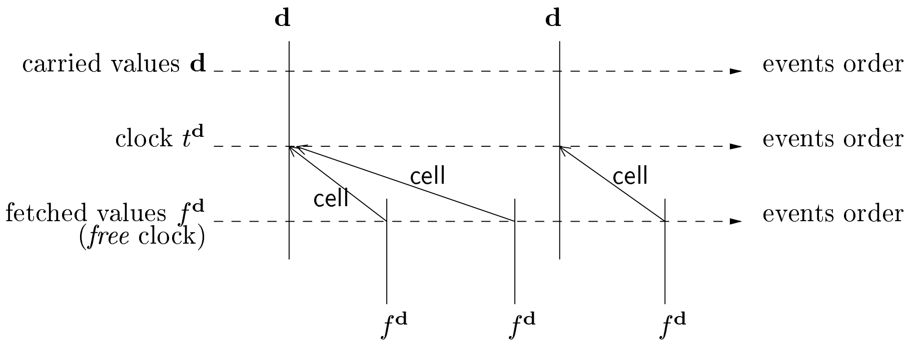
To any device d of a LTTA is associated a clock which provides the sampling instants. It is possible to access a value at any time thanks to the function $l_{\mathbf{d}}$ associated to the device d which enables to access the value carried at the previous tick:



Data-flow synchronous approaches In these approaches, the time continuum is abstracted. Only the notions of precedence and simultaneity are relevant. It is therefore very simple to abstract the time domain using the sampling events. In synchronous data-flow approaches, the clock $t^{\mathbf{d}}$ only defines the ordered set of sampling instants, and the carried values $\mathbf{d}$ are represented by a *signal* synchronized with $t^{\mathbf{d}}$. In order to make it possible to fetch the carried values *at any time*, we introduce a signal $f^{\mathbf{d}}$ whose clock[1] (noted $\hat{f^{\mathbf{d}}}$) is completely free. For that purpose, we use the cell construct of Signal. It enables to memorize the last value carried by a given signal. $f^{\mathbf{d}}$ can be simply defined as follows:

$$f^{\mathbf{d}} \ := \ (\mathbf{d} \ \mathsf{cell} \ \hat{f^{\mathbf{d}}}) \ \mathsf{when} \ \hat{f^{\mathbf{d}}}$$

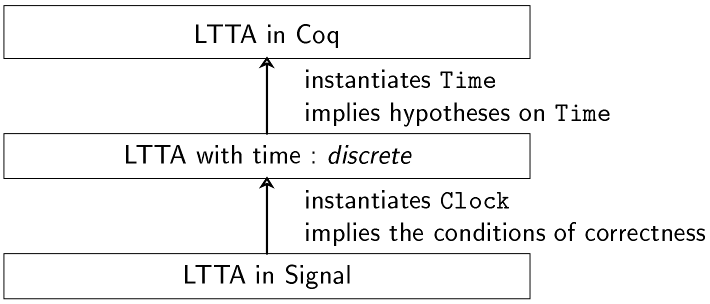
The following picture illustrates this abstraction:



This abstraction can be encoded in Coq using the translation scheme detailed in [8].

[1] in a data-flow synchronous approach, by *clock* we mean the ordered set of instants where a signal is present

LTTA in Signal In the last step, we prove the specification suggested in [1] guarantees that no two successive writing ticks can occur without a bus tick in between them, and that no two successive $\tau^{\mathbf{b}}$ can occur without a reading tick in between them. This implies the condition for correctness, i.e. $\forall n \in \mathbb{N},\ \exists k \in \mathbb{N}$, st. $\tau^{\mathbf{b}}(n) < t^{\mathbf{r}}(k) \leq \tau^{\mathbf{b}}(n+1)$. It can be easily proved using the Propositional Linear Temporal Logic (PLTL) also encoded in Coq [8]. This property comes from the shift_2 process. It introduces an interleaving constraint upon the reader and the writer clocks. The following picture illustrates this approach and underlines the use of Coq as a general formal framework to prove de correctness:



6 Conclusions and future work

We gave a formal proof of the correctness of a protocol for loosely time-triggered architectures using the Coq proof-assistant. Unlike [1], we did not have to restrict the model of the protocol to that of a finite-state system: we introduced a minimal set of assumptions about physical time. Since any other implementation of the LTTA protocol must at least guarantee these minimal requirements, our Coq model can be used as a generic formal proof framework. We illustrated this aspect by considering the Signal implementation of [1].

Directions of further studies comprise the specialization of our theorems by instantiating the abstract data type for time by the type for reals provided in Coq. Using the library of theorems and the decision procedures for reals, we could prove the numerical property from [1]. Another direction is to consider the verification of the synchronous data-flow implementation of the protocol. It could be done using the formalization of Signal in Coq and its library of theorem [8]. Finally, an attractive aspect of the use of Coq is the extraction of a reference implementation of the protocol. The only difficulty is that this protocol involves partial function that are difficult to deal with in Coq[2].

[2] `http://pauillac.inria.fr/pipermail/coq-club/2002/thread.html#569`

References

1. A. Benveniste, P. Caspi, P. L. Guernic, H. Marchand, J.-P. Talpin, and S. Tripakis. A protocol for loosely time-triggered architectures. In *Embedded Software Conference (EMSOFT'2002)*, volume 2491 of *Lecture Notes in Computer Science*, 2002.
2. A. Benveniste and P. Le Guernic. Synchronous Programming with Events and Relations: the SIGNAL Language and its Semantics. *Science of Computer Programming*, 16(2):103–149, 1991.
3. G. Berry and G. Gonthier. The ESTEREL Synchronous Programming Language: Design, Semantics, Implementation. *Science of Computer Programming*, 19:87–152, 1992.
4. P. Caspi, C. Mazuet, R. Salem, and D. Weber. Formal design of distributed control systems with lustre. In *Computer Safety, Reliability and Security, 18th International Conference, SAFECOMP'99, Toulouse, France, September, 1999, Proceedings*, volume 1698 of *Lecture Notes in Computer Science*, 1999.
5. N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The Synchronous Dataflow Programming Language LUSTRE. *Proc. of the IEEE*, 79(9):1305–1320, September 1991.
6. M. Kerbœuf, D. Nowak, and J.-P. Talpin. Specification and verification of a steam-boiler with Signal-Coq. In *Proceedings of the 13th International Conference on Theorem Proving in Higher Order Logics (TPHOLs 2000)*, volume 1869 of *Lecture Notes in Computer Science*, pages 356–371. Springer-Verlag, Aug. 2000.
7. R. Lazić and D. Nowak. A unifying approach to data-independence. In *Proceedings of the 11th International Conference on Concurrency Theory (CONCUR 2000)*, volume 1877 of *Lecture Notes in Computer Science*, pages 581–595. Springer-Verlag, Aug. 2000.
8. D. Nowak, J.-R. Beauvais, and J.-P. Talpin. Co-inductive axiomatization of a synchronous language. In *Proceedings of the 11th International Conference on Theorem Proving in Higher Order Logics (TPHOLs'98)*, volume 1479 of *Lecture Notes in Computer Science*, pages 387–399. Springer-Verlag, Sept. 1998.
9. The Coq development team. The Coq proof assistant reference manual : Version 7.3.1. Technical report, INRIA, 2002.
10. B. Werner. *Une Théorie des Constructions Inductives.* PhD thesis, Université Paris VII, 1994.

A Model of the LTTA protocol in SIGNAL

An overview of SIGNAL In SIGNAL, a process P consists of simultaneous equations over signals. A signal x describes a possibly infinite flow of discretely-timed values v. An equation $\boldsymbol{x} = f\boldsymbol{y}$ denotes a relation between a sequence of operands $\boldsymbol{y}$ and a sequence of results $\boldsymbol{x}$ by an operator f. Synchronous composition $P \,|\, Q$ consists of the simultaneous solution of the equations P and Q in time. SIGNAL requires three primitive operators: pre references the previous value of a signal in time (the equation $x = \mathsf{pre}\, y$ or $x = y\$1\,\mathsf{init}\, v$ initially defines x by v and then by the previous value of y in time), when samples a signal (the equation $x = y\,\mathsf{when}\, z$ defines x by y when z is true) and default merges two signals (the equation $x = y\,\mathsf{default}\, z$ defines x by y when y is present and by z otherwise).

$$P ::= \boldsymbol{x} := f\,\boldsymbol{y} \mid P \,|\, Q \mid P/x \qquad f \in F \supseteq \{\mathsf{pre}\,v \mid v \in V\} \cup \{\mathsf{when}, \mathsf{default}, \ldots\}$$

As an example, we consider the definition of a counter: Count. It accepts an input event rst and delivers the integer output val. A local variable cnt, initialized to 0, stores the previous value of val (equation cnt := val$1 init 0). When the event rst occurs, val is reset to 0 (i.e. 0 when rst). Otherwise, cnt is incremented (i.e. (cnt + 1)). The activity of Count is governed by the clock of its output val which differs from that of its input rst.

```
process Count = (? event rst ! integer val)
  (| cnt := val$1 init 0
   | val := (0 when rst) default (cnt + 1)
   |) where integer cnt end;
```

time	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}
rst		*tt*					*tt*				*tt*	*tt*
val	1	0	1	2	3	4	0	1	2	3	0	0
cnt	0	1	0	1	2	3	4	0	1	2	3	0

SIGNAL implementation of the LTTA The methodology used in SIGNAL to implement the LTTA consists of the progressive and compositional refinement of the requirement expressed by theorem 1: $x^r(n) = x^w(n), \forall n \geq 0$ that preserves the property of flow equivalence: xr and xw hold the same successive values. This yields the process ltta.

```
process ltta = (? boolean xw; event cw, cb, cr ! boolean xr, i, zi)
  (| (xb, bb, sbw) := bus (xw, writer(xw, cw), cb)
   | (xr, br, sbb) := reader (xb, bb, cr)
   | (i, zi) := prove (sbb, br, cr)
   | objective (sbw, sbb, cb, cr)
   |) where boolean bw, xb, bb, sbw, sbb, br;
```

The process ltta is decomposed into its three components reader, bus and writer connected by one-place buffers. The writer accepts an input x^{w} and defines the boolean flag b^{w} that will be carried along with it over the bus. The bus forward its inputs x^{w} and b^{w} to the reader as the result x^{b} and b^{b} of a one-place buffer. The reader loads its inputs x^{b} and b^{b} from the bus and samples x^{r} from x^{b} upon a switch of b^{b}. Each of the processes reader, bus and writer operate at independent (input) clocks c^{w}, c^{b} and c^{r}.

```
process writer = (? boolean xw; event cw ! boolean bw)
  (| bw ^= xw ^= cw | bw := not (bw$1 init true) |);
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 switch (br) |)
  where boolean yr;
end;
```

The key switch process emits an output signal c iff two successive occurrences zb and b of the boolean flag differ (notice the importance of the initial condition: zb must be initialized to true).

```
process switch = (? boolean b ! event c)
 (| zb := b$1 init true | c  := (when b when not zb) default (when not b when zb) |)
 where boolean zb;
end;
```

We now detail the definition of the desynchronizing one-place buffer which simulates asynchrony. The process buffer alternates between the receipt of an input (x, b) and the emission of an output (bx, bb). The alternate process makes these operations exclusive by using a boolean flip-flop signal b (notice, again, the importance of the initial condition: zb must be initialized to false for receive to precede send). The process current sustains its input signals (wx, wb) and allows to retrieve them at a given clock c.

```
process buffer = (? boolean x, b ; event c ! boolean bx, bb, sb)
  (| (sx, sb) := shift (x, b) | (bx, bb) := current (sx, sb, c) ||)
  where boolean sx;
    process alternate = (? boolean x, sx ! )
      (| x  ^= when b | sx ^= when not b | b  := not (b$1 init false) |)
        where boolean b; end;
    process shift = (? boolean x, b ! boolean sx, sb)
      (| (sx, sb) := current (x, b, ^sb) | alternate (x, sx) |);
end;
process current = (? boolean wx, wb; event c ! boolean rx, rb)
  (| rx := (wx cell c init false) when c | rb := (wb cell c init true)  when c |);
```

The process buffer introduces an unspecified delay (materialized by the input clock c), hence we can synchronize it with the output of the protocol xr without affecting the bus or the writer, and check whether they are equal.