SRI International

CSL Technical Report • March 2001

Formal Verification of Transmission Window Timing for the Time-Triggered Architecture

John Rushby
Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA

This research was partially funded under a subcontract to Honeywell Tucson under Cooperative Agreement NCC-1-377 with NASA Langley entitled *Design, Implementation, and Verification of Fault-Tolerant Modular Aerospace Controls*.

Computer Science Laboratory • 333 Ravenswood Ave. • Menlo Park, CA 94025 • (650) 326-6200 • Facsimile: (650) 859-2844

Abstract

We formally verify the parameters on the timing of message windows in transmitters, receivers, and bus guardians for the Time-Triggered Architecture.

Contents

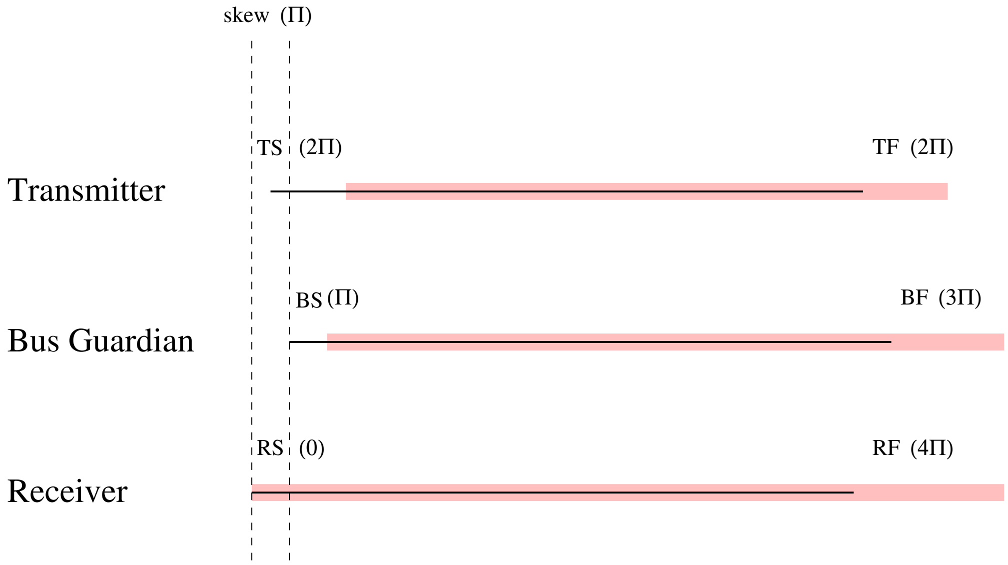
Chapter 1

Transmission Window Timing in the Time-Triggered Architecture

Message transmissions in TTP are governed by a global schedule which records the identity of the node that is to transmit in each slot, and also the start time and duration of the slot. The transmitter starts to send its message some time after the start of the slot, and finishes some time before the allowed duration has elapsed. Dually, receiving nodes start to listen at the beginning of the slot and cease when the duration has expired. The timings of events at transmitters and receivers are driven by their local clocks; these clocks are synchronized to within some threshold, but will not be exactly the same as each other. Consequently, it is possible that clock skew may allow a transmitter to start to send its message before some receivers are ready to listen, or to finish after others have ceased to listen. This could cause some nodes to reject messages that others accept—which violates the fault hypothesis of the TTP group membership algorithm, and could lead to failure.[1] The parameters that govern timing of the transmit and receive windows must be chosen so that this circumstance cannot arise. In addition to transmitters and receivers, *bus guardians* are used to ensure that faulty transmitters cannot broadcast outside their allotted windows. Parameter selection must therefore also consider window timing for the bus guardians.

The TTP protocol specification provides some discussion of bus guardian window timing [TTT99, pp. 114–115], but this has recently been supplanted by a new analysis [Bau01]. The new analysis assumes that all clocks (of TTP controllers and bus guardians) are synchronized within some parameter Π and proposes the following design rules (i.e., parameter selections), which are illustrated in Figure 1.1.

[1]The TTP membership algorithm has a clique-avoidance component that ensures well-defined behavior in this circumstance, but that behavior is not desirable and is intended only to ensure recovery from multiple faults—whereas no faults are present in this scenario.



For each component, the solid line indicates its nominal slot, and the shaded line its window.

Figure 1.1: Illustration of Window Timing Parameters

- The receive window extends from the beginning of the slot to $4\,\Pi$ beyond its allotted duration.
- Transmission begins $2\,\Pi$ units after the beginning of the slot and should last no longer than the allotted duration.
- The bus guardian for a transmitter opens its window Π units after the beginning of the slot and closes it $3\,\Pi$ beyond its allotted duration.

These rules are intended to ensure the following requirements.

Agreement: If any nonfaulty node accepts a transmission, then all nonfaulty nodes do.

Validity: If any nonfaulty node transmits a message, then all nonfaulty nodes will accept the transmission.

A detailed, but informal, analysis has been developed showing that these rules do indeed ensure the requirements [Bau01]. Our goal is to formally verify this analysis. We do this both because it is an interesting and important property that needs strong assurance, and because it is a comparatively simple topic that can serve as a tutorial introduction to formal verification in this domain.

Chapter 2

Verification of Window Timing Parameters

The issue that complicates analysis of window timings is that clock synchronization is imperfect: the clocks of nonfaulty nodes are maintained close together, but they cannot be expected to be in exact agreement. To analyze window timings, we need to formalize the notion of synchronization, which in turn requires that we formalize the notion of clocks. There is a standard way to do this, which was introduced by Lamport and Melliar-Smith [LMS85], and which we have used in formal treatments of problems similar to this [RvH93, Rus99].

Following [LMS85], we distinguish two notions of time: *clocktime*, denoted $\mathcal{C}$ is the local notion of time supplied by each component's clock, while *realtime*, denoted $\mathcal{R}$ is an abstract global quantity. We follow the usual convention and denote clocktime quantities by uppercase Roman or Greek letters, and realtime quantities by lowercase letters.

Formally, processor p's clock is a function $C_p : \mathcal{R} \rightarrow \mathcal{C}$. The intended interpretation is that $C_p(t)$ is the value of p's clock at realtime t.[1] Two clocks are synchronized if their readings at the same realtime instant t are within some small clocktime bound Π of each other.

Definition 1 Clock Synchronization.

$$\mid C_p(t) - C_q(t) \mid \leq \Pi.$$

We assume that the clocks of nonfaulty components are always synchronized within Π; this assumption is discharged by the clock synchronization algorithm of TTP/C, which has been formally verified to do so [PSvH99].

[1] In the terminology of [LMS85], these are actually "inverse" clocks.

2.1 Validity and Agreement

We first consider the Validity requirement on window timing; this decomposes into two subrequirements: one on message timing from a transmitter to its bus guardian, and one on message timing from bus guardians to receivers.

Lemma 1 (Transmitter to Bus Guardian Validity) *If a nonfaulty controller transmits a message, and its bus guardian is also nonfaulty, then its bus guardian will pass the message.*

Proof: Suppose that the slot for message n begins at $Slot\_Start(n)$ and the message is of maximum duration $Slot\_Duration(n)$. Suppose that the transmitter starts to send the message at some offset TS after the start of the slot and finishes TF after its maximum message duration; that is, at $Slot\_Start(n) + Slot\_Duration(n) + TF$. Suppose also that its bus guardian opens its window BS clocktime units after the start of the slot, and closes it TF after its maximum message duration; that is, at $Slot\_Start(n) + Slot\_Duration(n) + BF$.

Let t be the realtime such that

$$C_p(t) = Slot\_Start(n) + TS, \tag{2.1}$$

where p is the transmitter (i.e., t is the realtime at which the transmission begins). The bus guardian window must already be open at this time, so we need

$$C_g(t) \geq Slot\_Start(n) + BS \tag{2.2}$$

where g is the bus guardian for p. Now clock synchronization gives

$$-\Pi \leq C_g(t) - C_p(t) \leq \Pi$$

and so substituting (2.1) gives

$$C_g(t) \geq Slot\_Start(n) + TS - \Pi,$$

and satisfaction of (2.2) then requires

$$TS \geq BS + \Pi. \tag{2.3}$$

As noted in Chapter 1, the parameters chosen for TTP/C are $TS = 2\,\Pi$ and $BS = \Pi$; these clearly satisfy the constraint (2.3).

If we now let s be the realtime such that

$$C_p(s) = Slot\_Start(n) + Slot\_Duration(n) + TF \tag{2.4}$$

then s is the latest that p's transmission can continue (clearly $TF \geq TS$). We require that the bus guardian's window is still open at this time.

$$C_g(s) \leq Slot\_Start(n) + Slot\_Duration(n) + BF. \tag{2.5}$$

As before, clock synchronization gives

$$-\Pi \leq C_g(s) - C_p(s) \leq \Pi$$

and substituting (2.4) gives

$$C_g(s) \leq Slot\_Start(n) + TF + Slot\_Duration(n) + \Pi,$$

so that satisfaction of (2.5) requires

$$BF \geq TF + \Pi.$$

alsoIf we select $TS = BS + \Pi$ to satisfy (2.3), and $TF = TS$, this yields

$$BF \geq BS + 2\,\Pi. \tag{2.6}$$

As noted in Chapter 1, the parameters chosen for TTP/C are $BS = \Pi$ and $BF = 3\,\Pi$; these clearly satisfy the constraint (2.6). □

Lemma 2 (Bus Guardian to Receiver Validity) *If a nonfaulty bus guardian passes a message, then it will be accepted by all nonfaulty nodes.*

Proof: We know that the transmitter's bus guardian opens its window BS clocktime units after the start of the slot, and closes it at $Slot\_Start(n) + Slot\_Duration(n) + BF$. Suppose that the receiver is ready to receive RS clocktime units after the start of the slot, and ceases receiving at $Slot\_Start(n) + Slot\_Duration(n) + RF$.

Let t be the realtime such that

$$C_g(t) = Slot\_Start(n) + BS, \tag{2.7}$$

where g is the bus guardian (i.e., t is the realtime at which the bus guardian opens its window). The receiver must already be ready at this time, so we need

$$C_q(t) \geq Slot\_Start(n) + RS \tag{2.8}$$

where q is the receiver. Now clock synchronization gives

$$-\Pi \leq C_q(t) - C_g(t) \leq \Pi$$

and so substituting (2.7) gives

$$C_q(t) \geq Slot\_Start(n) + BS - \Pi,$$

and satisfaction of (2.8) then requires

$$BS \geq RS + \Pi. \tag{2.9}$$

As noted in Chapter 1, the parameters chosen for TTP/C are $BS = \Pi$ and $RS = 0$; these clearly satisfy the constraint (2.9).

If we now let s be the realtime such that

$$C_g(s) = Slot\_Start(n) + Slot\_Duration(n) + BF \tag{2.10}$$

then s is the latest that g's window is open. We require that the receiver's window is still open at this time.

$$C_q(s) \leq Slot\_Start(n) + Slot\_Duration(n) + RF. \tag{2.11}$$

As before, clock synchronization gives

$$-\Pi \leq C_q(s) - C_g(s) \leq \Pi$$

and substituting (2.10) gives

$$C_q(s) \leq Slot\_Start(n) + BF + Slot\_Duration(n) + \Pi,$$

so that satisfaction of (2.11) requires

$$RF \geq BF + \Pi. \tag{2.12}$$

As noted in Chapter 1, the parameters chosen for TTP/C are $BF = 2\,\Pi$ and $RF = 4\,\Pi$; these clearly satisfy the constraint (2.12). □

Theorem 1 (Validity) *If any nonfaulty node transmits a message, then all nonfaulty nodes will accept the transmission.*

Proof: This is a simple consequence of the previous two lemmas: the first ensures that the message from any nonfaulty transmitter will be passed by its nonfaulty bus guardian, and the second ensures that any message passed by a nonfaulty bus guardian will be accepted by a nonfaulty receiver. □

Theorem 2 (Agreement) *If any nonfaulty node accepts a transmission, then all nonfaulty nodes do.*

Proof: The previous theorem ensures that any message sent by a nonfaulty node will be accepted by all nonfaulty nodes. This theorem focuses on the case where the transmitting node is faulty. The fault hypothesis of TTP/C is that at most one fault containment unit (FCU) may fail in any two consecutive rounds. Each node comprises two FCUs that are assumed to fail independently: the controller and the bus guardian.

If a controller fails, it may attempt to transmit a message at an incorrect time. If its message nonetheless falls within its bus guardian's window, then Lemma 2 ensures that it will be accepted (assuming it is correctly formatted) by all nonfaulty receivers. Otherwise, the bus guardian will block the message (if it falls entirely outside its window) or truncate it (if it falls partly outside its window). Blocked messages are not seen by any receivers, and truncated messages will be malformed and rejected by all nonfaulty receivers.

A similar argument obtains if the bus guardian fails. If it fails in such a way that it passes its controller's message, then a lemma similar to those given earlier establishes that the timing of the nonfaulty controller's message is such that it is accepted by all nonfaulty receivers. If the guardian blocks or truncates the message, then it will not be accepted by any nonfaulty receiver. □

Notice that the Agreement property can be violated if there are multiple faults: the windows of a controller and its guardian that are both faulty could slide to a point where they transmit messages that arrive within the windows of some nonfaulty receivers, but not others (because of clock skew).

2.2 Nonoverlapping Slots

The properties established in the previous section ensure that the windows for a given slot line up appropriately. However, another concern is that the windows for adjacent slots must not overlap (this issue was raised by Tom Phinney of Honeywell when we used this material in a PVS training session).

We need to ensure that a message sent by a nonfaulty transmitter (or passed by a nonfaulty guardian) cannot reach another nonfaulty component that still has its window open from the previous slot, or that has already opened its window for the next slot. The design rule we propose is the following.

- A slot can start no earlier than 4Π after the end of the previous slot.

This is formalized as follows

$$Slot\_Start(n+1) \geq Slot\_Start(n) + Slot\_Duration(n) + 4\Pi. \tag{2.13}$$

The property we require is that the window of one component must not overlap the window of another component in the following slot (by symmetry, this also takes care of the previous slot). This means that the realtime at which the first component closes its window at the end of one slot should be no earlier than the realtime at which the second opens its window for the start of the next slot. We only need consider cases where one of the components can be sending or passing a message (i.e., a transmitter or bus guardian), and the other can be passing or receiving a message (i.e., a bus guardian or receiver).

Lemma 3 (Transmitter to Bus Guardian Separation) *A nonfaulty controller finishes transmitting its message before its nonfaulty bus guardian opens its window for the next slot.*

Proof: Let t be the realtime at which transmitter p finishes sending its message in slot n. Then

$$C_p(t) = Slot\_Start(n) + Slot\_Duration(n) + TF. \tag{2.14}$$

We require that the window of its bus guardian g opens for the next slot no earlier than t. That is,

$$C_g(t) \leq Slot\_Start(n+1) + BS. \tag{2.15}$$

Now clock synchronization gives

$$-\Pi \leq C_g(t) - C_p(t) \leq \Pi$$

and so substituting (2.14) gives

$$C_g(t) \leq Slot\_Start(n) + Slot\_Duration(n) + TF + \Pi$$

and satisfaction of (2.15) then requires

$$Slot\_Start(n+1) \geq Slot\_Start(n) + Slot\_Duration(n) + TF + \Pi - BS.$$

As noted in Chapter 1, the parameters chosen for TTP/C are $TF = 2\,\Pi$ and $BS = \Pi$; these clearly satisfy the constraint (2.13).□

The dual case is the following.

Lemma 4 (Bus Guardian to Transmitter Separation) *A nonfaulty controller does not start transmitting its message until after its nonfaulty bus guardian closes its window at the end of the previous slot.*

Proof: The proofs are more uniform if we reinterpret the lemma as requiring that the bus guardian closes its window before the transmitter starts transmitting in the next round.

Let t be the realtime at which bus guardian g closes its window in slot n. Then

$$C_g(t) = Slot\_Start(n) + Slot\_Duration(n) + BF. \tag{2.16}$$

We require that the window of its transmitter p opens for the next slot no earlier than t. That is,

$$C_p(t) \leq Slot\_Start(n+1) + TS. \tag{2.17}$$

Now clock synchronization gives

$$-\Pi \leq C_g(t) - C_p(t) \leq \Pi$$

and so substituting (2.16) gives

$$C_p(t) \leq Slot\_Start(n) + Slot\_Duration(n) + BF + \Pi$$

and satisfaction of (2.15) then requires

$$Slot\_Start(n+1) \geq Slot\_Start(n) + Slot\_Duration(n) + BF + \Pi - TS.$$

As noted in Chapter 1, the parameters chosen for TTP/C are $TS = 2\,\Pi$ and $BF = 3\,\Pi$; these clearly satisfy the constraint (2.13).□

Theorem 3 (Separation) *Messages sent or passed by nonfaulty components do not arrive before other compnents have finished the previous slot, nor after they have started the following one.*

Proof: The previous two lemmas are representative of the arguments involved; the other cases that need to be considered are bus guardian to receiver and the reverse (these are the cases where the $4\,\Pi$ in (2.13) is tight), transmitter to receiver and the reverse, and bus guardian to bus guardian.

Notice that it is possible for two receivers to be in different slots at the same realtime (unless the separation is increased to $5\,\Pi$). □

We also need to be sure that no component is required to start its next slot before it has finished its previous one. This follows because each of TF, BF, and RF is less than the $4\,\Pi$ constraint in (2.13).

Chapter 3

Formal Verification of Window Timing Parameters with PVS

In this chapter, we use the formal verification system PVS [OSRSC98] to check the analyses performed in the previous chapter. Readers are assumed to have some familiarity with PVS. We begin by introducing types and variables corresponding to the various entities introduced in the analysis,

```
windowtiming: THEORY
BEGIN
  realtime: TYPE = real
  t: VAR realtime

  clocktime: TYPE = nat

  slot: TYPE = nat
  n: VAR slot

  Slot_Start: [slot -> clocktime]
  Slot_Duration: [slot -> {c: clocktime | c>0} ]

  proc: TYPE+
  p, q, g: VAR proc
```

Most of this is straightforward. The range type of the function `Slot_Duration` uses a *predicate subtype* (one of the most powerful features of the PVS type system) to ensure that slot durations are always strictly positive. We use the nonempty type `proc` to represent both controllers and bus guardians.

Next, we introduce the function `C(p, t)` that represents clocks (written $C_p(t)$ in the previous chapter), the uninterpreted constant `Pi` and state the axiom `synchronized`, which asserts that clocks (implicitly of nonfaulty processors) are synchronized within `Pi`.

```
C(p, t): clocktime
Pi: clocktime

synchronized: AXIOM abs(C(p,t) - C(q,t)) < Pi
```

Then, we introduce the constants `TS`, `BS`, and `BF` and define `Tstart(n)` as the clocktime at which the transmitter starts to send its message in the `n`'th slot. We similarly define `Tfinish(n)`, `Bstart(n)`, and `Bfinish(n)` as the times when the transmitter finishes its transmission, the bus guardian opens its window, and closes it, respectively.

```
TS, TF, BS, BF: clocktime

Tstart(n): clocktime = Slot_Start(N) + TS
Tfinish(n): clocktime = Slot_Start(n) + Slot_Duration(n) + TF

Bstart(n): clocktime = Slot_Start(n) + BS
Bfinish(n): clocktime = Slot_Start(n) + Slot_Duration(n) + BF
```

Now, we can define a predicate that captures the first of the properties in which we are interested.

```
T_BG_Start_OK(p, g): bool =  FORALL n, t:
  C(p, t) = Tstart(n) IMPLIES C(g, t) >= Bstart(n)
```

This predicate is *true* of two processors `p` and `g` (implicitly a transmitter and its bus guardian) if, whenever `t` is the realtime at which `p` starts its transmission, the corresponding clocktime at `g` is already greater than the time at which it opens its window.

We want to prove that this property is true provided `TS > BS + Pi`. We state this as a lemma.

```
T_BG_Start: SUBLEMMA TS >= BS + Pi IMPLIES T_BG_Start_OK(p, g)
```

We know, from the proof used in the previous chapter, that this result is a consequence of the various definitions employed, plus the clock synchronization assumption. The PVS proof command `grind-with-lemmas` is a very powerful command that expands definitions, instantiates named lemmas, and applies decision procedures. The default version of the command guesses the wrong instantiations in this case, so we need the following more muscular version of the command to discharge the lemma.

```
(GRIND-WITH-LEMMAS :IF-MATCH all :LEMMAS "synchronized")
```

The keyword argument `:IF-MATCH all` tells the PVS theorem prover to search for all ways to instantiate the axiom `synchronized`.

An exactly similar specification and proof is used to state and prove the corresponding result for the end of the window,

```
T_BG_end_OK(p, g): bool = FORALL n, t:
   C(p, t) = Tfinish(n) IMPLIES C(g, t) <= Bfinish(n)

T_BG_end: SUBLEMMA TF + Pi <= BF IMPLIES T_BG_end_OK(p, g)
```

Finally, we can state the result (corresponding to Lemma 1 in the previous chapter) that asserts that the TTP/C parameter selections ensure validity of transmitter to bus guardian window timing.

```
T_BG_validity: LEMMA
  TS=2*Pi AND TF=2*Pi AND BS = Pi AND BF = 4* Pi =>
     T_BG_Start_OK(p, g) AND  T_BG_end_OK(p, g)
```

This result is a straightforward consequence of the previous two sublemmas; we again use the `grind-with-lemmas` proof command, but this time we specify the keyword argument `:DEFS NIL` to avoid expanding the defined terms.

```
(GRIND-WITH-LEMMAS :DEFS NIL :LEMMAS ("T_BG_Start" "T_BG_end"))
```

The specifications and proofs given above can be adapted straightforwardly to establish Validity in the bus guardian to receiver case (corresponding to Lemma 2 in the previous chapter), and also for transmitter to receiver validity (corresponding to the case in the argument for Agreement where the bus guardian is faulty). However, all these specifications and proofs have a similar form, so that it is more attractive and economical to state and prove the general result once and for all.

We can define a `window` to be a record comprising a pair of clocktimes where the second is strictly greater than the first (this requires *dependent* predicate subtyping). Then we can define what it means for one window `w1` to be *within* another window `w2`, which we write as `w1 <= w2`. Infix operators such as <= can be overloaded in PVS by defining them in prefix form.

```
window: TYPE = [# start: clocktime,
                  finish: {c: clocktime | c>start} #]

w1, w2: VAR window

<=(w1, w2): bool = FORALL (p1, p2: proc), (s,f: realtime):
    C(p1, s) = w1`start AND C(p1, f) = w1`finish
        IMPLIES C(p2, s) >= w2`start AND C(p2, f) <= w2`finish
```

This definition says that `w1 <= w2` if, whenever `s` and `f` are realtimes at which some processor's clock corresponds to the start and finish of `w1`, then any other processor's clock will already be beyond the start, but before the end, of `w2`, respectively.

Then we can state the theorem that says `w1` will always be within `w2`, provided `w1` always starts at least `Pi` after `w2`, and ends at least `Pi` earlier.

```
  timing: LEMMA w1`start >= w2`start + Pi
            AND w1`finish <= w2`finish - Pi
          IMPLIES w1 <= w2
```

The proof is similar to the more specific cases seen earlier.

```
(SKOSIMP)
(EXPAND "<=" +)
(SKOSIMP)
(GROUND)
(("1" (GRIND-WITH-LEMMAS :IF-MATCH ALL :LEMMAS "synchronized"))
 ("2" (GRIND-WITH-LEMMAS :IF-MATCH ALL :LEMMAS "synchronized")))
```

First we Skolemize to eliminate the universal-strength quantifiers, expand the new definition of <= and Skolemize again, then case-split and use `grind-with-lemmas` as before on each branch.

Then, we can specify the parameters that define the transmission windows for each kind of component.

```
  send_window(n: slot): window =
     (# start := Slot_Start(n)+2*Pi,
        finish := Slot_Start(n)+Slot_Duration(n)+2*Pi #)
  guardian_window(n: slot): window =
     (# start := Slot_Start(n)+Pi,
        finish := Slot_Start(n)+Slot_Duration(n)+3*Pi #)
  rcv_window(n: slot): window =
     (# start := Slot_Start(n),
        finish := Slot_Start(n)+Slot_Duration(n)+4*Pi #)
```

And then we specify the three required relationships between the various windows.

```
% Bus guardian is within receiver window
  bg_rcv: LEMMA guardian_window(n) <= rcv_window(n)

% Transmitter is within guardian window
  send_bg: LEMMA send_window(n) <= guardian_window(n)

% Transmitter is within receiver window (in case bg is faulty)
  send_rcv: LEMMA send_window(n) <= rcv_window(n)
```

Each of these is proved by the following command, which invokes the powerful `grind` strategy, with the information that the `timing` theorem can be used as a rewrite rule.

```
  (GRIND :REWRITES "timing")
```

We state the conditions on nonoverlapping windows in terms of the following function.

```
|>(w1,w2): bool = FORALL (p1, p2: proc), (f: realtime):
    C(p1, f) = w1`finish IMPLIES C(p2, f) <= w2`start
```

We can then say `w1 |> w2` to indicate that the end of window `w1` is no later (in realtime) than the start of window `w2`, for all pairs of processors. The following theorem establishes that this relationship holds providing at least `Pi` separates the respective clocktimes.

```
separation: THEOREM w2`start >= w1`finish+Pi => w1 |> w2
```

This is proved by the following command.

```
(GRIND-WITH-LEMMAS :IF-MATCH ALL :LEMMAS "synchronized")
```

The seven lemmas that establish separation between various combinations of components are then stated as follows. In the previous chapter, we asserted that these are all ensured by the bound $4\,\Pi$; here, we state the tight bound for each pair.

```
send_bg_sep: LEMMA
  Slot_Start(n+1) >= Slot_Start(n)+Slot_Duration(n)+2*Pi =>
    send_window(n) |> guardian_window(n+1)

send_rcv_sep: LEMMA
  Slot_Start(n+1) >= Slot_Start(n)+Slot_Duration(n)+3*Pi =>
    send_window(n) |> rcv_window(n+1)

rcv_send_sep: LEMMA
  Slot_Start(n+1) >= Slot_Start(n)+Slot_Duration(n)+3*Pi =>
    rcv_window(n) |> send_window(n+1)

rcv_bg_sep: LEMMA
  Slot_Start(n+1) >= Slot_Start(n)+Slot_Duration(n)+4*Pi =>
    rcv_window(n) |> guardian_window(n+1)

bg_send_sep: LEMMA
  Slot_Start(n+1) >= Slot_Start(n)+Slot_Duration(n)+2*Pi =>
    guardian_window(n) |> send_window(n+1)

bg_bg_sep: LEMMA
  Slot_Start(n+1) >= Slot_Start(n)+Slot_Duration(n)+3*Pi =>
    guardian_window(n) |> guardian_window(n+1)

bg_rcv_sep: LEMMA
  Slot_Start(n+1) >= Slot_Start(n)+Slot_Duration(n)+4*Pi =>
    guardian_window(n) |> rcv_window(n+1)
```

All these lemmas are proved by the following command.

```
(GRIND-WITH-LEMMAS :IF-MATCH ALL :LEMMAS "separation")
```

Chapter 4

Conclusion

We have formally verified that the design rules (i.e., choices of parameters) on the timing of message windows in TTP/C ensure desirable and necessary properties. In particular, messages sent by nonfaulty transmitters will never be cut off by nonfaulty bus guardians and will be accepted by nonfaulty receivers. We have also verified that there is no overlap between the windows of one slot and those of the next.

Our formal proofs are no longer than the informal arguments in [Bau01], while the mechanically checked PVS proofs are actually shorter—because we can use the expressive logic of PVS to state and prove key results in their general form, and then simply instantiate them to obtain the desired cases.

In addition to their intrinsic value, we have found the formulas stated and proved in this report to be useful as introductions to the techniques of mechanized formal methods and to PVS.

Bibliography

[Bau01] Günther Bauer. Bus guardian window timing. Unpublished draft, Technische Universität Wien, Real-Time Systems Group, Vienna, Austria, April 2001.

[LMS85] L. Lamport and P. M. Melliar-Smith. Synchronizing clocks in the presence of faults. *Journal of the ACM*, 32(1):52–78, January 1985.

[OSRSC98] S. Owre, N. Shankar, J. M. Rushby, and D. W. J. Stringer-Calvert. *User Guide for the PVS Specification and Verification System*. Computer Science Laboratory, SRI International, Menlo Park, CA, September 1998. Three volumes: Language, System, and Prover Reference Manuals.

[PSvH99] Holger Pfeifer, Detlef Schwier, and Friedrich W. von Henke. Formal verification for time-triggered clock synchronization. In Charles B. Weinstock and John Rushby, editors, *Dependable Computing for Critical Applications—7*, volume 12 of *Dependable Computing and Fault Tolerant Systems*, pages 207–226, San Jose, CA, January 1999. IEEE Computer Society.

[Rus99] John Rushby. Systematic formal verification for fault-tolerant time-triggered algorithms. *IEEE Transactions on Software Engineering*, 25(5):651–660, September/October 1999.

[RvH93] John Rushby and Friedrich von Henke. Formal verification of algorithms for critical systems. *IEEE Transactions on Software Engineering*, 19(1):13–23, January 1993.

[TTT99] Time-Triggered Technology TTTech Computertechnik AG, Vienna, Austria. *Specification of the TTP/C Protocol*, July 1999.