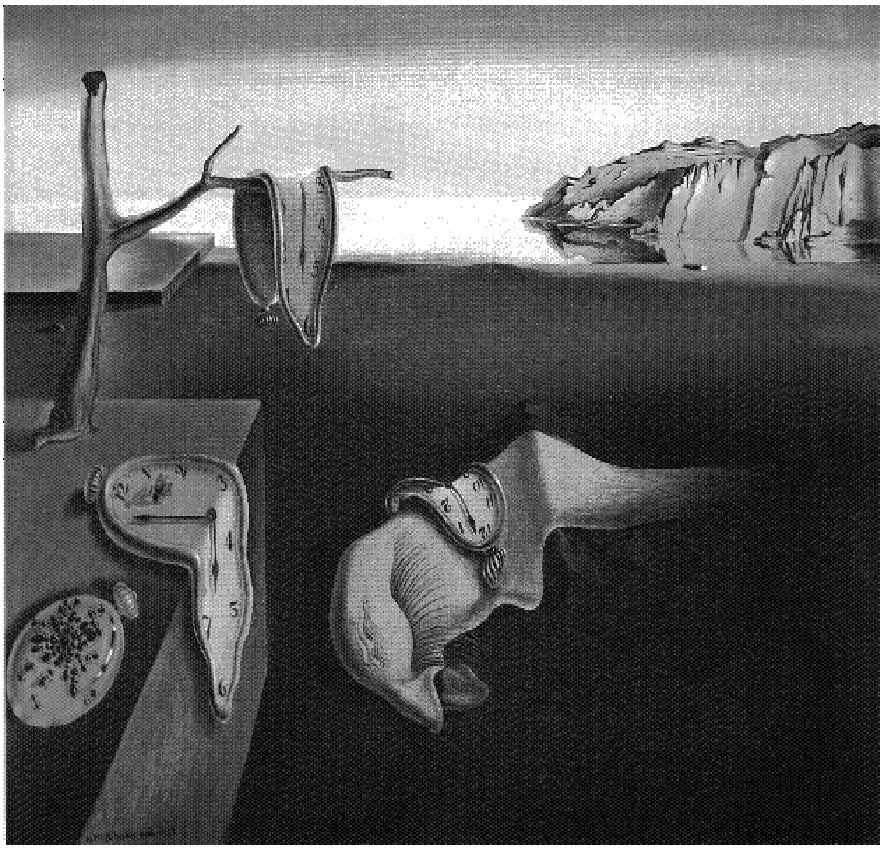
Institut für Automation
Abt. für Automatisierungssysteme

Technische Universität Wien

Projektbericht Nr. 183/1-56
July 1995

UTCSU Functional Specification

Klaus Schossmaier, Ulrich Schmid



Salvador Dali, "Die Beständigkeit der Erinnerung"

UTCSU Functional Specification

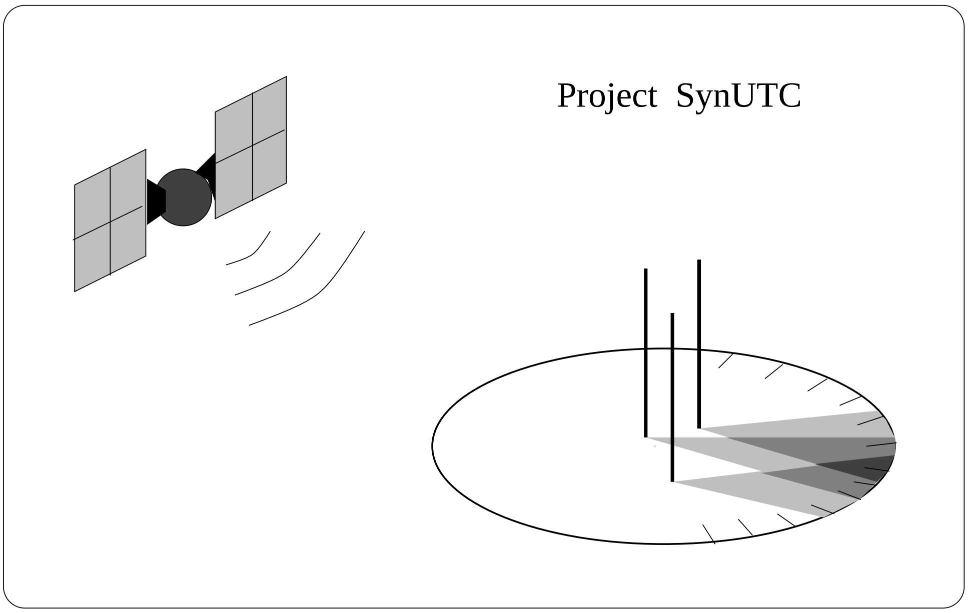
Klaus Schossmaier, Ulrich Schmid
Technical University of Vienna
Department of Automation 183/1
Treitlstraße 3, A-1040 Vienna, Austria
Email: {kmschoss, s}@auto.tuwien.ac.at

July 30, 1995

Abstract

This report contains the final functional specification of the *Universal Time Coordinated (Clock) Synchronization Unit* (UTCSU) currently being developed as an ASIC at the Department of Computer Technology. The UTCSU implements the bottom layers of our novel *Interval-Based Clock Validation* (ICV) technique that will furnish fault-tolerant distributed real-time systems with local clocks synchronized to UTC with very high accuracy. It is based on a number of inventive and unique features devised by the authors, in particular an adder-based clock rate and state correction and a similar accuracy interval maintenance. Developed in the framework of our joint project SynUTC, this document —actually several revisions emanated from discussions with Dietmar Loy and Martin Horauer— forms one of the major links between the research work of our groups.

Keywords: Universal Time Coordinated (UTC), Clock Synchronization Unit, ASIC, Interval-Based Clock Validation (ICV), GPS.



Contents

1 Introduction

This section contains a very brief overview of the research setting of the SynUTC[1] project, a joint project between our Department of Automation and the Department of Computer Technology at TU Vienna. It aims developing software and hardware for providing large distributed fault-tolerant real-time systems with local clocks synchronized to *Universal Time Coordinated* (UTC) with very high accuracy, see [Sch95] for an introduction.

1.1 Components of a Synchronization Subnet

The entire distributed system under consideration is made up from a set of interconnected *synchronization subnets* (SSNs). Each such SSN consists of a collection of nodes connected by a network of point-to-point or, preferably, broadcast/multicast type. Nodes can be classified as follows:

- A **C-node (client)** participates only in a passive way, making use of existing synchronization activities to establish synchronized local time in a cheap manner.
- An **S-node (secondary)** participates actively in synchronization, forming the core of our approach.
- A **P-node (primary)** has access to UTC time (e.g., via a GPS-receiver) and donors this usually most accurate time information to the corresponding SSN.
- A **G-node (gateway)** connects two or more SSNs. Special functionalities are attached to such a node for time dissemination.

Figure 1 shows a sample system with 3 SSNs.

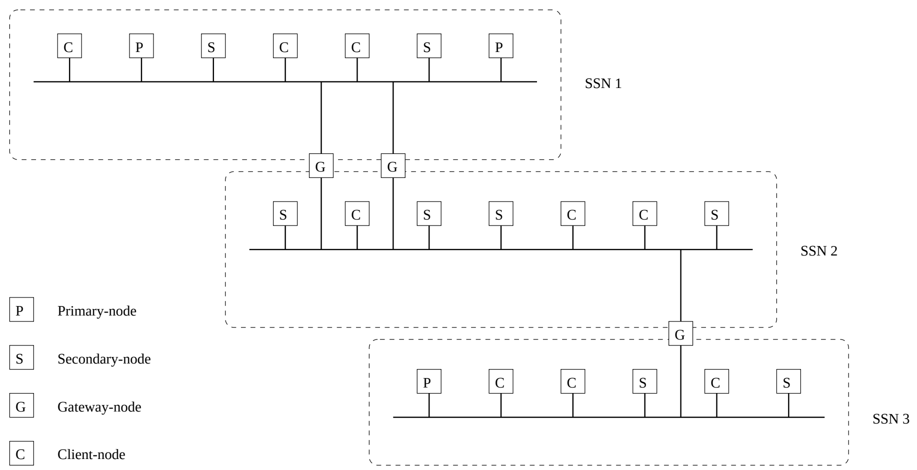


Figure 1: System Components

[1]Supported by the Austrian Science Foundation (FWF) under contract no. P10244-ÖMA.

1.2 Interior of a Node

Structured in three layers, Figure 2 depicts the entities required for clock synchronization at a single node.

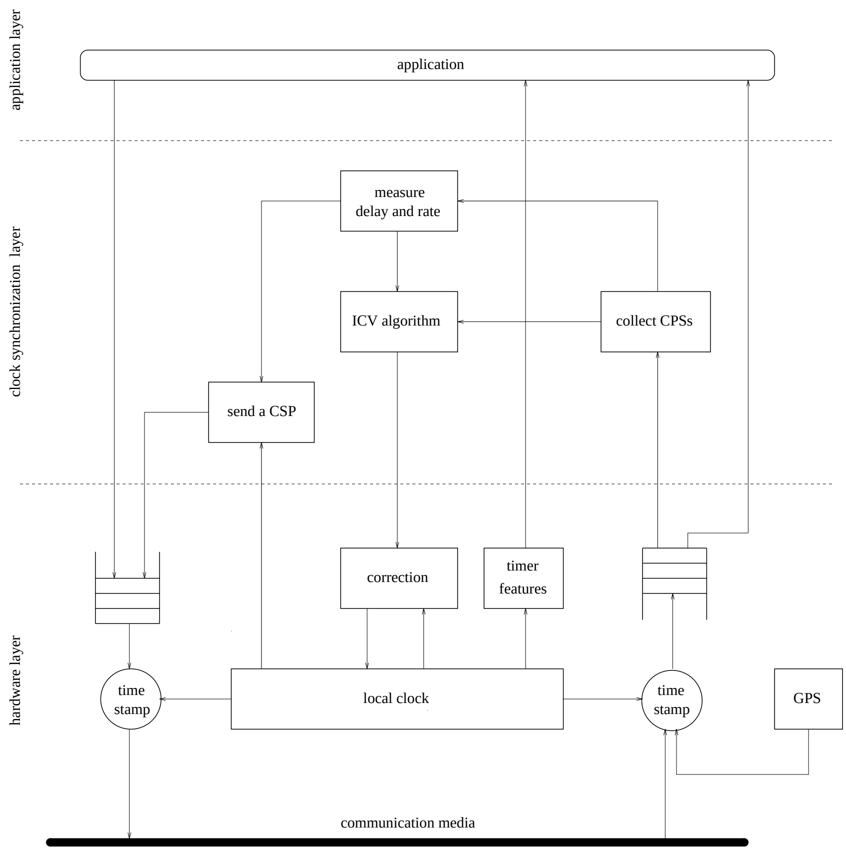


Figure 2: Entities of a Node

The *application layer* runs the user software, which sends/receives application related data packets and utilizes timing features provided by the lower layers.

The software part capable of maintaining global time resides in the *clock synchronization layer*. To simplify the picture, there are entities to send/collect *clock synchronization packets* (CSP's) and entities to measure transmission delays and local clock rates. The intelligence to keep the distributed system in synchrony is encapsulated in the entity labeled ICV algorithm. Basically, it consists of an algorithm to handle *clock states* and another one to deal with *clock rates*.

To achieve a synchronization quality in the range of μs, proper support from the *hardware layer* is decisive. All hardware components related to clock synchronization reside on a *Network Clock Synchronization Coprocessor* (NCSC), which hosts network controller(s), coprocessors, memory, interface logic and our novel *Universal Time Coordinated (Clock) Synchronization Unit* (UTCSU) realized as an ASIC. A prototype NCSC has been developed, for details consult [Hor94].

Our UTCSU, the primary object of this functional specification, is responsible for

- maintaining a state and rate adjustable local clock,
- assisting the protocol to exchange CSPs (in particular timestamping them),
- providing links to GPS-receivers,
- supporting application timer features.

One salient challenge of the UTCSU is to handle several interfaces —i.e., application layer, clock synchronization layer, network controller(s), GPS-receiver(s)— simultaneously, which rises complex timing issues. Especially low access delays for reading/writing of UTCSU registers are mandatory for high synchronization quality.

We should mention evident similarities between our UTCSU and the CSU of [KO87]. However, apart from the (re-used but considerably refined) idea of DMA-based message timestamping, our UTCSU relies on totally different paradigms for design and implementation of seemingly resembling features, e.g., for rate and state correction of the local clock, see Section 2.

1.3 Clock State Algorithm

The clock state algorithm is responsible for ironing out (*a posteriori*) non-systematic, short-term clock deviations inevitably arising during system operation.

The key of the state algorithm is to capture standard time (UTC) with an appropriate *state interval* relative to the local clock. Obviously, a smaller state interval refers to a better knowledge of standard time. To put it simple, every node broadcasts periodically its current state interval and determines from all incoming state intervals a new one with enhanced quality. To avoid non-monotonic clock correction, an ensuing amortization period adjusts the local clock state accordingly. More specifically, a node performs the following operations:

- **Initiation** In a periodic fashion (called *S-duty*) a P/S/G-node initiates a broadcast of its current state interval.
- **Sending** Due to computational overhead and contention on the communication media, a node is able to start sending the state interval only after some delay. A CSP carrying this information is called a *S-packet* and gets assigned a timestamp on actual departure.
- **Collection** A peer node receives S-packets from other nodes in a sequential manner. Upon arrival, packets get timestamped and stored into a suitable data structure. Moreover, a *delay compensation* is applied to the intervals in order to account for uncertainties in transmission delays.
- **Updating** Since CSPs got send and received at different points in time, it is necessary to make state intervals compatible with each other. Two basic operations

must be distinguished: *Dragging* means to move an interval along the time axis without changings its length, which is necessary to follow the progress of time. *Deteriorating* means to expand the state interval in order to account for local clock drifts.

- **Termination** Some fixed time after initiation (called *T-duty*) chosen appropriately so that all CSP's from other nodes have been received, final resynchronization activities are started.
- **Marzullo** Applying a fault-tolerant form of intersection on compatible state intervals from (low-accurate) S-nodes yields a (correct) validation interval.
- **Validation** The essence of the ICV-algorithm comes into play, when the state intervals disseminated by highly accurate nodes are validated by means of the validation interval. If validation succeeds, a correction value for the local clock is computed based on the highly accurate intervals; otherwise, the validation interval is used for that purpose.
- **Amortization** The local clock gets adjusted according to the correction value in a smooth way.

1.4 Measuring Transmission Delays

To make delay compensation working properly, figures about transmission delays of S-packets need to be known. A round trip approach was chosen to measure transmission delays, whereby with some periodicity (called *D-duty*, actually a multiple of S-duty) the following actions take place:

- **Request** A *D-packet* —effectively piggybacked to an S-packet— gets broadcasted by a node to request peer nodes to respond.
- **Response** Some fixed time after request (called *A-duty*) chosen appropriately (so that all D-packets of other nodes may be received), any participating node assembles and broadcasts an *A-packet* that contains transmit and receive timestamps of each received D-packet; the A-packet is timestamped on departure as well.
- **Reception** When an A-packet arrives, the receiving node extracts its appropriate information (3 timestamps) for each participating node and calculates average transmission delay and maximum and minimum values.

1.5 Time Distribution Protocol

A node has to recognize the beginning and end of a synchronization round. If a node's clock is correctly synchronized, this may be achieved just by waiting for the local clock reaching the time of the S-duty and T-duty = S-duty + Λ. Parameter Λ must be chosen appropriately to guarantee that all CSP's from correct nodes have been received. Note that the exact protocol to distribute CSPs among all relevant nodes is of course network dependent. Although each node initiates a synchronization round (S-duty) nearly at the same time, there is an inevitable serialization of all broadcast CSPs induced by the communication channel(s) and node processing.

Since we want to resynchronize a node's clock even in the case when its state is faulty, it is not sound to rely on the local clock only when detecting synchronization rounds. Therefore, the beginning of a synchronization round is also recognized when sufficient many CSP's from other nodes have been received.

G-nodes deserve special attention w.r.t. the time distribution protocol, since they are supposed to be geared towards the SSN with the best time, and should serve as a provider of that time for the remaining SSNs without dropping too much quality. Observe that a G-node might have to run the time distribution protocol differently in each attached SSN, since they can differ substantially in size and speed. Thus, a simple round oriented protocol based on simultaneous S-duties for each SSN might be to weak to solve those requirements.

1.6 Clock Rate Algorithm

The second major algorithm deals with clock rates. It tries to compensate systematic, long-term clock deviations in advance (*a priori*).

Similar to the clock state algorithm, each node attempts to capture standard rate (i.e., 1 s/s) with a *rate interval* that is meant relative to the rate of its local clock. Again, a smaller interval means a better quality. In principle, the clock rate algorithm runs like the previously presented clock state algorithm, with the difference that rate intervals are exchanged instead of state intervals (disseminated by means of special *R-packets*).

The rate algorithm turns out to be harder, since a node is unable to observe its rate directly. In order to attack this deficiency, a suitable rate measurement algorithm has to be employed at each node. It measures the rates of all the other nodes relative to the rate of the own clock. Note that clock values without state-correction are required for that purpose, and that the occurence of clock state errors affect rate measurement. Furthermore, in order to achieve 1 μs accuracy with reasonably large resynchronization intervals (S-duty), the clock rate must be controlled with very fine granularity.

1.7 Miscellaneous Difficulties

All of the subsequently listed problem areas call for much effort. However, appropriate solutions are primarily embedded in the clock synchronization layer; there seems to be no need for special hardware support.

- *Flywheeling* deals with the problem of keeping precise (and as accurate as possible) time in case of total UTC loss, and handling the transition back to normal operation.

- *Startup* addresses the question of putting the entire system into operation.

- *Rejoining* refers to the problem of how to integrate a (re)booting node (remember 4 types) into ongoing synchronization activities.

2 Maintaining Local Time

The most important quantity maintained by the UTCSU is certainly local time.

2.1 NTP Standard

Local time is represented using (modified) NTP time format, see [Mil91]. Full NTP time consumes 64 bit, where the upper 32 bit are interpreted as standard seconds relative to UTC and the lower 32 bit give the corresponding fractional part. The following table defines bit numbers and their time equivalents.

integer part				fractional part			
bit#	time equiv.	bit#	time equiv.	bit#	time equiv.	bit#	time equiv.
+31	68 years	+15	9.10 hours	-01	500 ms	-17	7.62 μs
+30	34 years	+14	4.55 hours	-02	250 ms	-18	3.81 μs
+29	17 years	+13	2.27 hours	-03	125 ms	-19	1.90 μs
+28	8.5 years	+12	1.13 hours	-04	62.5 ms	-20	953.67 ns
+27	4.25 years	+11	34.13 min	-05	31.25 ms	-21	476.83 ns
+26	2.13 years	+10	17.07 min	-06	15.62 ms	-22	238.41 ns
+25	1.08 years	+09	8.53 min	-07	7.81 ms	-23	119.21 ns
+24	194.18 days	+08	4.27 min	-08	3.81 ms	-24	59.60 ns
+23	97.09 days	+07	2.13 min	-09	1.90 ms	-25	29.80 ns
+22	48.55 days	+06	1.07 min	-10	976.56 μs	-26	14.90 ns
+21	24.27 days	+05	32 s	-11	488.28 μs	-27	7.45 ns
+20	12.17 days	+04	16 s	-12	244.14 μs	-28	3.72 ns
+19	6.07 days	+03	8 s	-13	122.07 μs	-29	1.86 ns
+18	3.03 days	+02	4 s	-14	61.03 μs	-30	931.32 ps
+17	1.52 days	+01	2 s	-15	30.51 μs	-31	465.66 ps
+16	18.02 hours	00	1 s	-16	15.25 μs	-32	232.83 ps

ultrafractional part			
bit#	time equiv.	bit#	time equiv.
-33	116.42 ps	-49	1.77 fs
-34	58.21 ps	-50	888.18 as
-35	29.10 ps	-51	444.09 as
-36	14.56 ps	-52	222.04 as
-37	7.28 ps	-53	111.02 as
-38	3.64 ps	-54	55.51 as
-39	1.82 ps	-55	27.76 as
-40	909.50 fs	-56	13.88 as
-41	454.75 fs	-57	6.94 as
-42	227.38 fs	-58	3.47 as
-43	113.69 fs	-59	1.73 as
-44	56.84 fs	-60	867.36 zs
-45	28.42 fs	-61	433.68 zs
-46	14.21 fs	-62	216.84 zs
-47	7.11 fs	-63	108.42 zs
-48	3.55 fs	-64	54.21 zs

The original NTP format follows leap second insertion/deletion of UTC and is therefore not chronoscopic. This is of course undesirable for real-time applications. As a consequence the UTCSU is normally employed with a modified NTP format that does not keep track of leap seconds for such applications.

2.2 Principle of Rate and State Adjustment of Clocks

The design of a local clock being correctable in both state and rate is the focus of interest of these subsections. We first discuss traditional approaches and introduce our new alternative.

2.2.1 Approach 1: Single Counter

A usual digital clock consists of an oscillator and a counter. Note that the NTP format forces the driving frequency to relate to one of the reciprocals of the NTP bit time equivalents. For example, to comply to bit# -23, a frequency of $119.21\text{ns}^{-1} = 8.388608$ MHz has to be chosen. To deliberately speed up or slow down the clock, pulses generated from the oscillator must be manipulated. Two possibilities come into play:

- To vary the frequency of the oscillator itself. However, only a voltage controlled oscillator (VCO) allows to do that, cf. [Mil91].
- Given a quartz oscillator running at a multiple of the nominal frequency, pulses are suppressed or inserted as required, cf. [KO87]. The drawback of this technique is not only a higher bandwidth, but also an unsatisfactory correction granularity (a small rate change means a long delay until one pulse gets modified).

2.2.2 Approach 2: Adder

We propose an alternative digital clock consisting of an oscillator driving an adder instead of a simple counter. Employing an adder gives the freedom to add a particular amount (clock step) to the clock register at every pulse. A rate change can be achieved by varying this amount, which goes in effect almost instantly and holds up linearity. Figure 3 illustrates this simple technique for compensating a 10% slowed down clock.

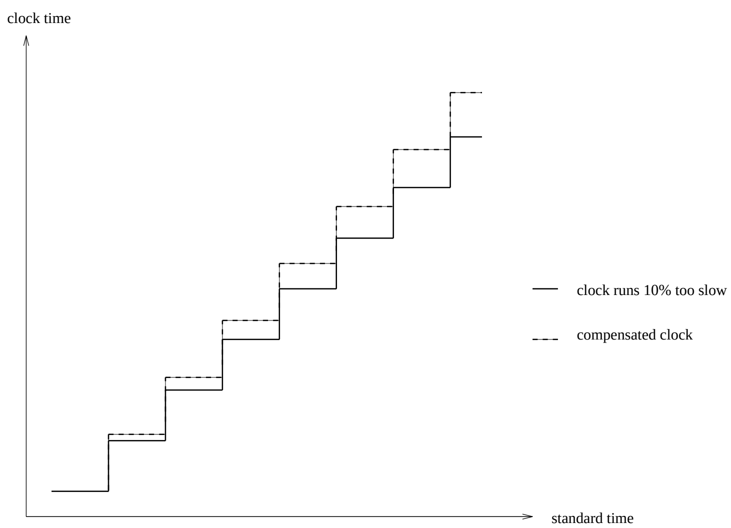


Figure 3: Using variable clock steps

For better understanding, the notion of granularity of a clock should be expanded in the following way:

- **State granularity** G_s refers to the smallest time unit representable by the register for local time.
- **Time granularity** G_t refers to the time period between two updates of the local time register.

Usually, the values for G_s and G_t are equal and get lumped into granularity. There is, however, nothing magic about having a much better state granularity than time granularity. The cost to make this technique working is twofold: First, a full adder is needed, which is more complex and consumes more chip space as a simple counter. Second, the state granularity must be enhanced to account for small rate shifts. Therefore, the local time register must be (internally) extended with an ultrafractional part. Emphasizing again, this does not mean that we can measure absolute time with this quality, it merely allows accumulating minute time portions precisely.

It is worth mentioning that all update operations (and hence the whole system) are fully synchronous w.r.t. the oscillator clock cycle, irrespectively of the correction value, sharply contrasting counter approaches. Moreover, it is even possible to break the 2^k-frequency constraint enforced by the NTP format, since any clock step (in NTP format) can be used. For example, in order to use a 10 MHz source, a step of 100 ns (binary) has to be selected. However, to avoid that truncation errors (which add up at each tick) mess up the intrinsic rate behavior of the underlying physical clock, one usually has to add more ultrafractional bits.

2.2.3 Approach 3: Interleaving Counter and Adder

The abovementioned addition must of course be carried out at each oscillator tick. Lack of performance or chip space, however, might prohibit this implementation. In such a case it is possible to interleave the process of addition and counting: Several oscillator ticks are granted to carry out the addition, during which a simultaneously running counter delivers local time. On completion of the addition, the newly computed local time is put into effect and the counter carries it along. One has to be careful, however, not to violate monotonicity across switching between counter and adder. Note further that the frequency of the oscillator must comply to the NTP standard here.

2.3 Specification of Registers maintaining Local Time

The heart of the UTCSU is a 91 bit register called CLOCK that contains local time and gets updated by an adder as advocated above (cf. approach 2). Bear in mind that CLOCK is a register in conjunction with an adder and has no counting features at all. Although most UTCSU-applications will probably need chronoscopic time, it might be advantageous to provide an optional mechanism for dealing with leap seconds. Note that leap second insertion/deletion must take place instantaneously at well-defined times.

Finally, when CLOCK wraps around (somewhere in year 2036), an interrupt should be generated.

A 40 bit register STEP is needed to hold the amount added to CLOCK at each oscillator tick. STEP has an associated 32 bit preload register STEP/PURE to support atomic update, see Section 3.5. Another register STEPLOW is provided to (pre)load the 8 lower bits of STEP, which should be reinitialize periodically.

Externally accessible is only a portion of register CLOCK, via a 56+8 bit *NTP-bus* that includes an 8 bit checksum (CS) as well. The remaining 35 bit ultrafractional part of CLOCK (called MICROSTAMP) is solely required for internal fine-grained rate adjustment. The 56 bit NTP time is further split into a 24 bit MACROSTAMP and a 32 bit TIMESTAMP. Note that we can adhere to byte-orientation (56 bit) here, even with oscillator frequencies corresponding to NTP-bit greater than $[-24]$, since this means to include the appropriate number of ultrafractional bits only. For example, for our nominal oscillator frequency of 2^{23} Hz, one ultrafractional bit ([-24]) appears in the NTP time.

Given a conventional 32 bit architecture, one cannot access register CLOCK within a single read/write-operation. Hence the UTCSU provides the clock synchronization layer with NTP time primarily by means of a 32 bit TIMESTAMP register in combination with a 24+8 bit MACROSTAMP register (providing CS in its msb), which is latched when TIMESTAMP is read. For initialization, the CLOCK register is (atomically) writable via MACROSTAMP and TIMESTAMP. The following table summarized the registers for local time; a complete register layout of the UTCSU is contained in Section 9.

elements for maintaining local time			
element	NTP bit#	length	specials
CLOCK	[+31,-59]	91	—
oscillator	[-23]	—	nominal osc. frequency 2^{32} Hz $\approx$ 8 MHz
STEP	[-20,-59]	40	preloadable
STEP/PURE	[-20,-51]	32	preload for upper part STEP
STEPLOW	[-52,-59]	8	lower part STEP
NTP-bus	[CS]:[+31,-24]	8+56	NTP time plus checksum
TIMESTAMP	[+7,-24]	32	read/writable
MACROSTAMP	[CS]:[+31,+8]	8+24	latched when TIMESTAMP read

To justify the above configuration, we first note that the time granularity G_t corresponding to the nominal oscillator frequency is 119.21 ns. The full state granularity evaluates to $G_{sf} = 1.73\ as$ (corresponding to NTP bit [-59]). This implies that truncation errors arising from oscillator frequencies $\neq 2^k$ impair the intrinsic rate of the physical clock by at most $R_f = G_{sf}/G_t = 1.73 \cdot 10^{-18}/G_t$. In particular, for a 10 MHz oscillator frequency (as provided by most GPS receivers), R_f evaluates to $1.73 \cdot 10^{-11}$; however, since the binary expansion of the (truncated) remainder reads $.01010111\ldots \approx 0.34$, the actual truncation errors is approximately $0.34 \cdot R_f \approx 0.6 \cdot 10^{-11}$ in this case.

On the other hand, it is obviously not necessary (and practicable on 32 bit architectures) to deal with 40 bit correction values (STEP). Providing the lower 8 bit (which depends solely upon the used oscillator frequency) via a dedicated preload register STEPLOW, only 32 bit correction values (STEP/PURE) must be dealt with. This gives a state granularity G_{sc} of 0.44 fs (corresponding to NTP [-51]), so that for the nominal oscillator frequency, a *rate correction granularity* $R_c = G_{sc}/G_t = 0.37 \cdot 10^{-8}$, which is equivalent to 1 μs per 270.4 s results. R_c can be best interpreted as the smallest non-zero unit of interference to change the clock value at each tick.

Since STEP goes up to NTP [-20], it is possible to use ocsillators with a frequency greater than 1/16-th of the nominal one, so that even 1 MHz is allowed. Changing the nominal frequency impacts the numerical values for R_f and R_c as computed above.

3 Support for Clock State Algorithm

This section describes the UTCSU requirements related to the state algorithm. Starting out with an easy model to describe various timing aspects, the necessary components will be introduced along with an explanation of the interplay between hard- and software.

3.1 Notations and Definitions for Describing Timing Aspects

The progress of time at each node is going to be captured by a sequence of virtual clocks $C_0, C_1, C_2, \ldots$ Each virtual clock C_i has a birth time t_i^0, where $t_i^0 < t_{i+1}^0 \ \forall i \geq 0$. Due to asynchronism, the generation of new virtual clocks won't be staggered precisely with resynchronization period R, instead time $t_{i+1}^0 - t_i^0 = R_i$ will pass. Conceptually, the lifetime of a virtual clock is infinite, although in practice it gets redeemed by the generation of the next one. More specifically, due of computational overheads a virtual clock C_i goes into effect at t_i^1, i.e., switching between two consecutive virtual clocks happens at time points t_i^1, delayed by $D_i = t_i^1 - t_i^0$. State correction will be performed continuously, leading to a particular rate modification during interval $[t_i^1, t_i^2]$. The length of this interval is called amortization period M_i.

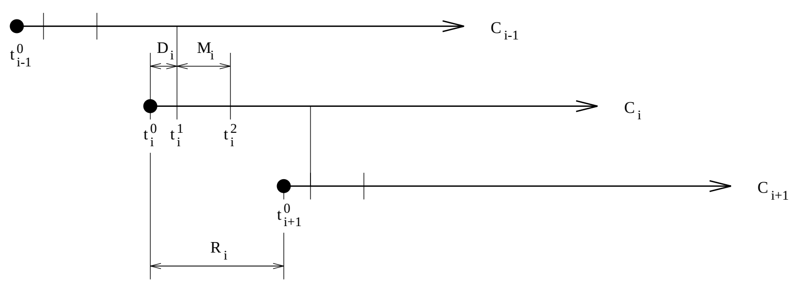


Figure 4: Sequence of virtual clocks

Each virtual clock C_i maintains the following time dependent quantities:

- clock value $c_i(t)$
- upper envelope $E_i(t)$

- lower envelope $e_i(t)$
- upper accuracy $A_i(t) = E_i(t) - c_i(t)$
- lower accuracy $a_i(t) = c_i(t) - e_i(t)$

These quantities are defined $\forall t \geq t_i^0$ and satisfy the sanity condition $e_i(t) \leq c_i(t) \leq E_i(t)$. Former mentioned state intervals are expressed by $[c_i(t) - a_i(t),\, c_i(t) + A_i(t)]$, which are also called unsymmetrical intervals, due to the explicitly set midpoint. For notational convenience all quantities can be packed into a clock vector

$$\mathbf{S}_i(t) = (c_i(t), E_i(t), e_i(t), A_i(t), a_i(t))$$

At birth time t_i^0, an initial clock vector $\mathbf{S}_i(t_i^0)$ can be calculated. In an idealized model the components of such a vector change linearly over time. Without any deliberate or uncontrolled interference clock C_i has a rate ω_i, hence $c_i(t) = c_i(t_i^0) + \omega_i(t - t_i^0)$ for $t \geq t_i^0$. The upper envelope $E_i(t)$ grows with $\omega_i + \Lambda_i$, where Λ_i expresses the deterioration in order to account for clock drifts. The same holds for the lower envelope $e_i(t)$, characterized by λ_i. Equations for accuracies can be obtained by plugging in their definitions. Finally the clock vector becomes to

$$\mathbf{S}_i(t) = \begin{pmatrix} c_i(t_i^0) \\ E_i(t_i^0) \\ e_i(t_i^0) \\ A_i(t_i^0) \\ a_i(t_i^0) \end{pmatrix} + \begin{pmatrix} \omega_i \\ \omega_i + \Lambda_i \\ \omega_i - \lambda_i \\ \Lambda_i \\ \lambda_i \end{pmatrix} (t - t_i^0) \qquad \text{for t} \geq \text{t}_\text{i}^0.$$

The following Figure 5 illustrates the defined quantities.

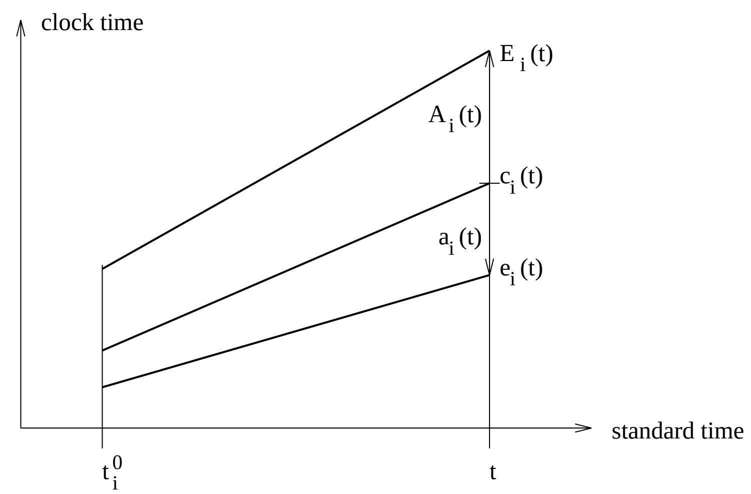


Figure 5: State quantities of a virtual clock

3.2 State Correction by means of Continuous Amortization

The actual transition from one virtual clock to its successor turns out to be a bit intricate. Figure 6 shows an illustrative scenario.

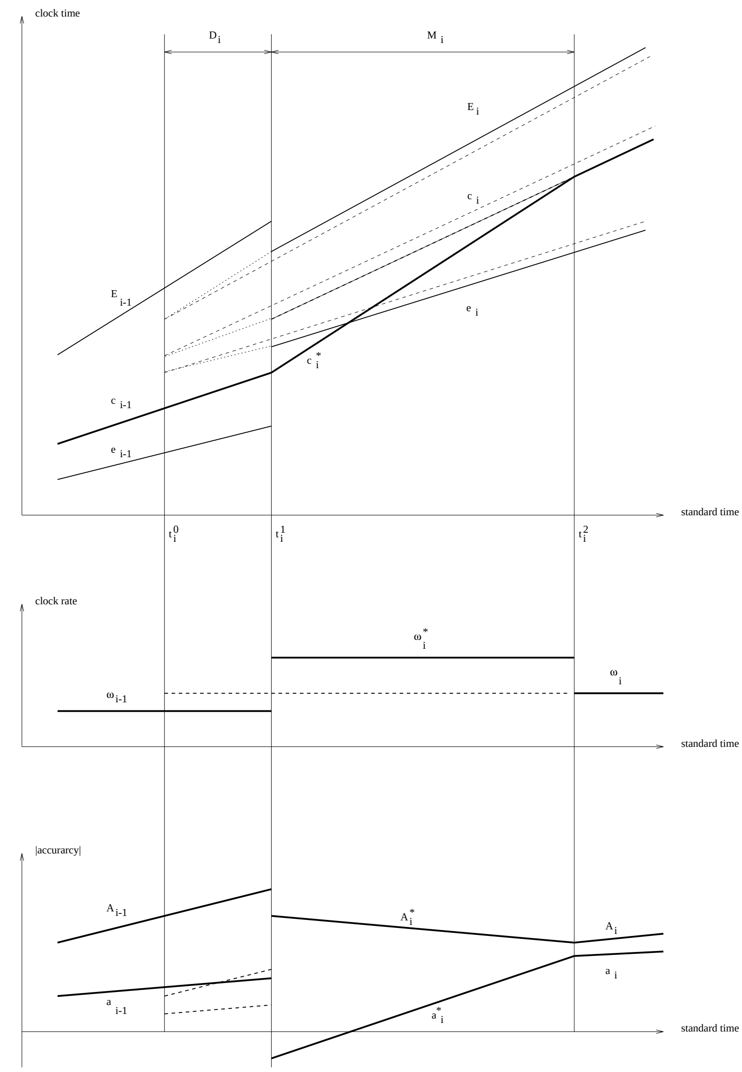


Figure 6: Continuous Amortization

The soon to expiring virtual clock C_{i-1} runs with rate $\omega_{i-1}$and captures standard time with state interval $[E_{i-1}, e_{i-1}]$. At t_i^0 the computation of the next virtual clock commences. All results of that computation take t_i^0 as their reference. Therefore, a consistent new virtual clock must satisfy both sanity conditions $E_i(t_i^0) \leq E_{i-1}(t_i^0)$ and $e_i(t_i^0) \geq e_{i-1}(t_i^0)$. The difference in their clock values is defined as state deviation $\Delta c_i = c_i(t_i^0) - c_{i-1}(t_i^0)$, which has to be corrected during amortization period M_i. The differences of the envelopes $\Delta E_i = E_i(t_i^0) - E_{i-1}(t_i^0)$ and $\Delta e_i = e_i(t_i^0) - e_{i-1}(t_i^0)$ may be corrected instantaneously.

Since computational delay D_i cannot always[2] be determinated, the new virtual clock

[2]In fact, if starting the amortization period is performed by the UTCSU instantaneously at some

C_i can be put into effect not earlier than t_i^1, hence clock C_{i-1} must be used up to that time. However, since the correction values (differences $\Delta c_i, \Delta E_i, \Delta e_i$) are determined w.r.t. t_i^0 but applied at t_i^1, clock C_i and its envelopes deviate slightly from the one obtained in the ideal situation $D_i = 0$. This error, shown by the dashed lines in Figure 6, causes only a slightly larger state interval (remember the sanity conditions above) and is of second or less order. When the new clock takes over at t_i^1, two rules are followed:

- *Clock values* are not allowed to jump and must be adjusted by a temporally rate change during amortization, causing $c_i(t_i^2)$ to display a value that deviates from the non-amortized clock value at t_i^2 by Δc_i. The software determinates the proper rate ω_i^*, such that this clock value is reached after amortization period M_i. At t_i^2 the clock resumes to its normal behavior.

- *Envelopes* may be changed instantaneously. Since it is more advantageous to maintain the envelopes internally via the accuracies (requires much less chip space), the latter jump as well. More specifically, the differences $\Delta A_i = A_i(t_i^0) - A_{i-1}(t_i^0)$ and $\Delta a_i = a_i(t_i^0) - a_{i-1}(t_i^0)$ need to be added to the actual values $A_i(t_i^1)$ and $a_i(t_i^1)$, respectively. During amortization period $[t_i^1, t_i^2]$ accuracies (not the envelopes!) experience a modified deterioration value as well.

As a consequence of these rules, it is possible that —during amortization— a clock value lies outside of the region spanned by the envelopes, leading to negative accuracies. However, this situation can be fixed by regarding a negative accuracy as zero, which translates to a valid state interval enlargement.

3.3 Specification of Registers maintaining Accuracies

Besides local time, the UTCSU provides registers to maintain the upper accuracy $A(t)$ resp. lower accuracy $a(t)$ along with their deterioration $\Lambda(t)$ resp. $\lambda(t)$. The accuracy registers ALPHA+ and ALPHA− span 44 bit each, containing the absolute value of $a(t)$ and $A(t)$ respectively. Therefore, we can handle accuracies up to 7.8 ms. In order to cope with the abovementioned situation (local clock outside envelopes), an internal sign bit causes negative values to be read (via STATE±, see below) as zero. ALPHA± is set to the maximum value by an arbitrary write access.

To deal with state intervals, the full length of ALPHA+ resp. ALPHA− is not required. Therefore, the upper 16 bits of ALPHA+ and ALPHA− constitute the internal 32 bit *A-bus*, which is used to feed various sample registers. In addition, registers STATE+ and STATE− (grouped to form a single 32 bit register for fast read-access) provide the current accuracies (msb of ALPHA±, with zero read when negative), which are latched when the local clock (i.e., TIMESTAMP) is read to enforce atomicity. Updating ALPHA− and ALPHA+ at synchronization instants t_i^1 must be (internally) performed incrementally, by subtracting the (absolute) difference ΔA_i resp. Δa_i written to STATE+ resp. STATE−

predifined time, D_i is known and the new absolute values for clock and envelopes could be put into effect directly. The alternative method using difference values, however, works also when issuing the start amortization command is delayed beyond t_i^0 for some reason.

from the content of ALPHA+ resp. ALPHA−. Note that the sanity conditions (see Section 3.2) ensure that both accuracies are only decremented.

Registers LAMBDA+ (holds Λ) and LAMBDA− (holds λ) are 16 bits long and contain the signed value of deterioration per tick. That is, on each oscillator tick, the content of registers LAMBDA± is added to the corresponding ALPHA±. This implies a maximum deterioration of less than $2^{-36}s$/tick, which is approximately 122 $\mu s/s$ for the nominal oscillator frequency; this should be sufficient even for worst quartz oscillators. Note that ALPHA± do not wrap around when reaching their maximum value. Register LAMBDA+ resp. LAMBDA− is overwritten with preload register LAMBDA/PURE+ resp. LAMBDA/PURE− at t_i^2 to support atomic update, see Section 3.5.

Finally, an accuracy guard should trigger an interrupt when the (upper part of the) accuracy exceeds some threshold given in the 16 bit register BOUND. The following table summarizes all introduced elements; Section 9 presents the complete register layout.

elements maintaining accuracies			
element	NTP bit#	length	special
ALPHA+	±[-8,-51]	44+1	maxvalue when written
ALPHA−	±[-8,-51]	44+1	maxvalue when written
STATE+	+[-8,-23]	16+1	read/writable(decr.)
STATE−	+[-8,-23]	16+1	read/writable(decr.)
BOUND	+[-8,-23]	16+1	—
LAMBDA+	±[-37,-51]	15+1	preloadable
LAMBDA−	±[-37,-51]	15+1	preloadable
LAMBDA/PURE+	±[-37,-51]	15+1	preload for LAMBDA+
LAMBDA/PURE−	±[-37,-51]	15+1	preload for LAMBDA−

3.4 Specification of Elements for Continuous Amortization

Continuous amortization (CA) requires some special attention; Section 3.5 contains a detailed description of all the required steps. It affects the major rate justification register STEP (remember Section 2.3), but also LAMBDA+ and LAMBDA− as well, which may even be negative during CA.

Continuous amortization is started when either (1) NTP-time has reached (or already passed[3]) the value written into AMORTTIMESTART or, alternatively, (2) when the dummy register STARTAMORT is written. AMORTTIMESTART is divided into a AMORTTIMESTARTLOW and AMORTTIMESTARTHIGH part, with the former one actually arming the timer when written. Disarming AMORTTIMESTART (before expiration) must also be possible, e.g., by writing a specific bit pattern to AMORTTIMESTARTLOW. Moreover, there should be a way to recognize whether continuous amortization has already been started, via a certain status bit. Note that this feature

[3] Since AMORTTIMESTART might be initialized slightly after the time they are meant to go off, looking for a full match is not sufficient. Note also that the incrementation of CLOCK by STEP might just skip a specific bit pattern.

is necessary to escape from (almost) never-ending wait in case of NTP time wrapping around in year 2036, by using the wrap-around interrupt provided by the CLOCK register, cf. Section 2.3.

Three registers are controlling amortization: a 32 bit register STEP/AMORT and two 16 bit registers LAMBDA±/AMORT. They act primarily as preload registers and deliver the appropriate values during amortization, which get precomputed by software. The expiration of the interval timer AMORT (counting oscillator ticks) indicates the end of the amortization period and causes a switch back to "pure" values of STEP and LAMBDA±. Note that the 8 bit lower part of STEP (STEPLOW) is fixed and hence not to be affected by this switch.

Finally, there is a duty-timer TDUTY for T-duties (termination), cf. Section 1.3, which may trigger an interrupt when NTP-time written to the register has been reached or already passed. Similar to AMORTTIMESTART, it consists of an upper part TDUTYHIGH and a lower one TDUTYLOW that actually arms the duty timer. Disarming a non-expired TDUTY is possible by writing a specific bit pattern to TDUTYLOW; disarming a just expired TDUTY must clear a pending interrupt as well. It should be possible to check for an expired TDUTY via a certain status-bit, to escape from never ending wait as mentioned above.

elements for continuous amortization			
register	NTP bit#	length	specials
STEP/AMORT	[-20,51]	32	preload for upper part STEP
LAMBDA+/AMORT	±[-37,-51]	15+1	preload for LAMDBA+
LAMBDA−/AMORT	±[-37,-51]	15+1	preload for LAMDBA-
AMORTTIMER	[+8,-23]	32	programmable backward counter
STARTAMORT	–	–	start amortization immediately
AMORTTIMESTART	[+31,-16]	48	amortization start time
AMORTTIMESTARTHIGH	[+31,0]	32	preload start time (high part)
AMORTTIMESTARTLOW	[-1,-16]	16	preload and arm/disarm start time
TDUTY	[+31,-16]	48	termination duty timer
TDUTYHIGH	[+31,0]	32	preload TDUTY (higher part)
TDUTYLOW	[-1,-16]	16	preload and arm/disarm TDUTY

3.5 Mechanisms to carry out State Correction

This section reveals the details of state correction in terms of the interface between the synchronization software and the UTCSU. The various steps, schematically shown in Figure 7, are as follows:

1. Upon reception of a CSP, register TIMESTAMP is read at t_i^0 (automatically by the NCSC hardware), thereby latching registers MACROSTAMP in order to acquire the local time of reception. Moreover, some computations to be used in the subsequent resynchronization may be carried out and TDUTY may be (re-)initialized in case of a fault of the own clock.

2. When TDUTY —set to some point in time past the reception of the last correct CSP of a particular round— goes off, some final computations yielding the appropriate correction values for the local clock are started via an interrupt.

3. Following the computation, the preload registers for STEP, LAMBDA±, STATE± and AMORTTIMER are loaded. Moreover, register TDUTY is updated for the next synchronization round. Finally, AMORTTIMESTART is written with the NTP-time where continuous amortization should start.[4]

4. When NTP-time reaches AMORTTIMESTART at t_i^1, continuous amortization commences by simultaneously taking over the previously preloaded values for amortization. More specifically, STEP becomes to STEP/AMORT and LAMBDA± to LAMBDA/AMORT± and ALPHA± get decremented by STATE±. Finally, AMORTTIMER starts decrementing from its preloaded value.

5. Expiration of timer AMORTTIMER at t_i^2 automatically terminates continuous amortization activities, by simultaneously overriding STEP with STEP/PURE and LAMBDA± with LAMBDA±/PURE.

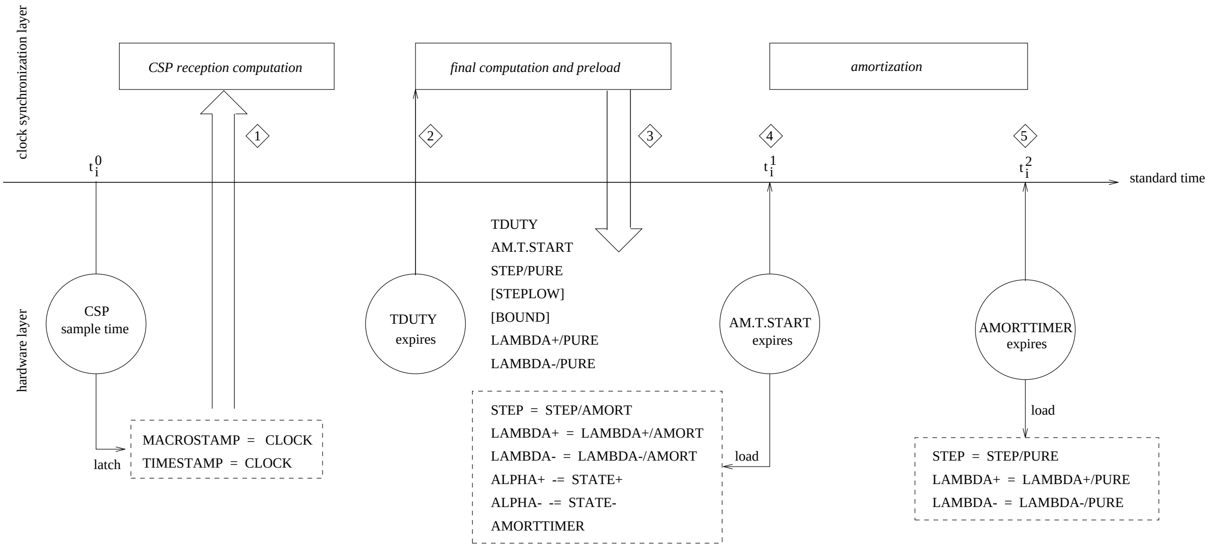


Figure 7: Hard- and Software interplay for State Correction

4 Support for Time Distribution Protocol

The time distribution protocol is responsible for exchanging CSPs among participating nodes within a SSN. This sections specifies the relevant UTCSU requirements to carry out the protocol. It is important to note that those features must be replicated for each attached SSN of a node.

[4]Or STARTAMORT is used directly.

4.1 Specification of Timers

The duty-timer DUTY is responsible to periodically trigger S/R/D/A-duties. DUTY is 48 bit long, hence organized as a 32 bit DUTYHIGH and a 16 bit DUTYLOW part that acutally arms the duty timer on writing. If armed, its contents are compared against the NTP-bus and when there is a transition from smaller (DUTY < NTP-bus) to larger or equal a dedicated interrupt informs the clock synchronization layer. It is possible to disarm DUTY by writing a specific bit pattern to DUTYLOW, and to check for an expired timer via a certain status bit; the latter is necessary to prohibit (almost) never-ending wait in case of NTP time wrapping around in year 2036, by using the wrap-around interrupt of the CLOCK register, cf. Section 2.3.

The quite broad range of DUTY (down to 15.25 μs) allows fine-grained staggering of CSP initiations at different nodes, which could be used to disentangle contention on the communication media and to support multiple SSNs in G-nodes. Software has to take care for an update of register DUTY at each synchronization round. In fact, since R-duties (clock rate algorithm) and D-duties (transmission delay measurement) are usually multiples of S-duties (send CSP for state algorithm), and an A-duty (acknowledgment for D-packets) occurs some fixed time after its corresponding D-duty, it is possible to use a single duty-timer for all of them. Note that R-packets and D-packets are effectively piggybacked to an S-packet when an R-duty and/or D-duty occurs.

specifications of timers			
register	NTP bit#	length	specials
DUTY	[+31,-16]	48	SSN DUTY timer
DUTYHIGH	[+31,0]	32	preload DUTY (higher part)
DUTYLOW	[-1,-16]	16	preload and arm/disarm DUTY (lower part)

4.2 Packet Timestamping

Timestamping of CSPs at both sending and receiving side is a pivotal functionality for high accuracy requirements. Extending the approach of [KO87], packets need to be timestamped just the moment they are actually leaving or arriving a node. This necessitates coordinated support from the UTCSU and the *network clock synchronization coprocessor* (NCSC) built around the former and a suitable network controller chip (like the Intel 82596 Ethernet controller).

Two cases have to be considered:

- **Timestamping outgoing CSPs**

  The basic idea is simple. When the network controller reads some specific byte in the transmit buffer containing an outgoing CSP, the UTCSU samples the current status of the NTP- and A-bus into registers, which get transparently mapped into certain buffer regions. Note that atomic sampling of both NTP-time and accuracies are not absolutely necessary here, since the clock is not read "across" t_i^1 (cf. Section 3.1). Reading STATE± shortly after local time provides only a slightly larger state interval as obtained by an atomic read. Remember that our NTP time is 56 bit, which must be read as a 32 bit TIMESTAMP and a 24 bit (with checksum CS actually 32 bit) MACROSTAMP portion.

- **Timestamping ingoing CSPs**

  Similarly, when the network controller writes a specific address in the receive buffer, the UTCSU samples the current state of the NTP-bus into a register. Note that no accuracy information needs to be sampled on the receiving side. It is important to point out, that any incoming packet may cause such actions, since the network controller might have no means to recognize packets as CSPs immediately on arrival.

  The sampled reception time may be processed further by one of the following methods:

  1. *Processing in an ISR of an interrupt raised upon arrival*: A short FIFO might be required (not implemented in the UTCSU), since a new CSP may arrive before the ISR has been reading the timestamp of the last CSP.
  2. *Transparent mapping into a dedicated field in the receive buffer upon reception*: This method does not need a FIFO, but additional data path transceivers in the NCSC. Most importantly, it requires additional (don't care) bytes to be transmitted in the data packet.
  3. *Collect reception time in dedicated Timestamp-Memory*: This method requires an NCSC with an additional DMA-controller capable of performing the data transfer of reception time from the UTCSU to a small dedicated TS-memory. This must take place concurrently with the packet reception, initiated when the network controller writes the sampling address in the receive buffer as mentioned above.

Compared with the CSU of [KO87], our UTCSU provides considerable improvements w.r.t. timestamping uncertainty:

- Timestamping of received CSPs is done when a specific data byte in the receive buffer is written, and not upon the occurrence of the packet reception interrupt.
- We separate the moment of sampling from reading the sampled timestamp by providing (possibly) different buffer addresses triggering this special functionality, both at the transmitting and the receiving side. This gives considerably more freedom avoiding large transmission delay uncertainties caused by network controller peculiarities (e.g. internal FIFOs). Note however, that almost simultaneous sampling and reading must be possible.

The following table summarizes the elements for each SSN block.

elements for packet timestamping			
element	NTP bit#	length	specials
asynchronous input line	—	1	when transmit buffer read
TIMESTAMP/TRANSMIT	[+7,-24]	32	from NTP-bus
MACROSTAMP/TRANSMIT	[CS]:[+31,+8]	24+8	from NTP-bus
STATE+/TRANSMIT	[-8,-23]	16	from A-bus
STATE−/TRANSMIT	[-8,-23]	16	from A-bus
STATELOW±/TRANSMIT	[-16,-23]:[-16,-23]	16(32)	lower bytes of A-bus
asynchronous input line	—	1	when receive buffer written
TIMESTAMP/RECEIVE	[+7,-24]	32	from NTP-bus
MACROSTAMP/RECEIVE	[CS]:[+31,+8]	24+8	from NTP-bus

Note that registers STATE±/TRANSMIT must be accessible via a 32 bit operation simultaneously. For high-accuracy applications, the lower 8 bit of STATE±/TRANSMIT concatenated to 16 bit are provided in both the high and low word of a dedicated 32 bit register STATELOW±/TRANSMIT. This simplifies NCSC design for 32 bit architectures, see the remarks above Figure 9.

To estimate a reasonable number of independent SSN blocks inside the UTCSU, one should consider the following applications:

- Single SSN (most often)
- Single SSN with replicated networks (triple redundant)
- Single gateway (2 or more SSNs)
- Single gateway with replicated networks (triple redundant)

The last application is most demanding, since it requires a total of 6 SSN blocks. If a node connects fewer subnets, the unused timestamping features of an SSN block can be utilized (in conjunction with the NCSC) for application timing features.

4.3 Packet Formats

Considering the amount of the information exchanged within a CSP, we have two different types of networks in mind:

(1) Low throughput networks like field busses

(2) High throughput networks like Ethernet

Whereas "wasting" communication bandwidth by transmitting long timing information poses no problems in (2), one should be very carefully about each additional byte in (1). Our UTCSU/NCSC must support both types, possibly intermixed for different SSNs connected by a gateway node. Consequently, we should consider the following

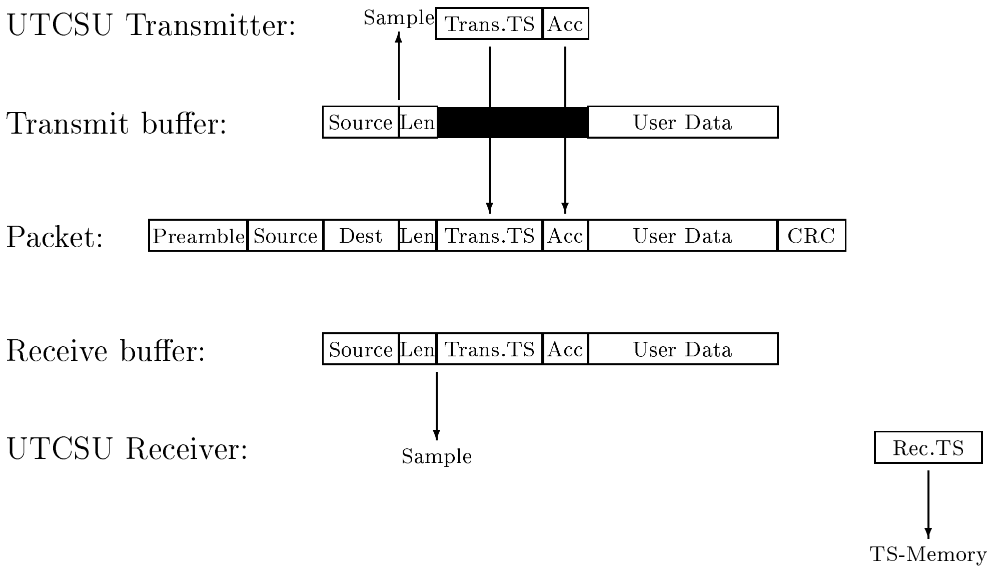


Figure 8: Packet Formats and Timestamping for Fieldbus (16 bit architecture, DMA + TS-memory)

two principal packet formats shown in Figure 8 and 9. However, there are several "hybrid"variants possible. For 32 bit NCSC architectures, it would ease the design if transparent mapping is performed at the granularity of longwords (4 bytes). However, this would mean that both the addresses A and B depicted in Figure 9 must be longword-aligned. This is not the case for high-accuracy applications, where the accuracy provided by STATELOW±/TRANSMIT is only 16 bit long; 2 additional don't care bytes for padding are required in the CSP here.

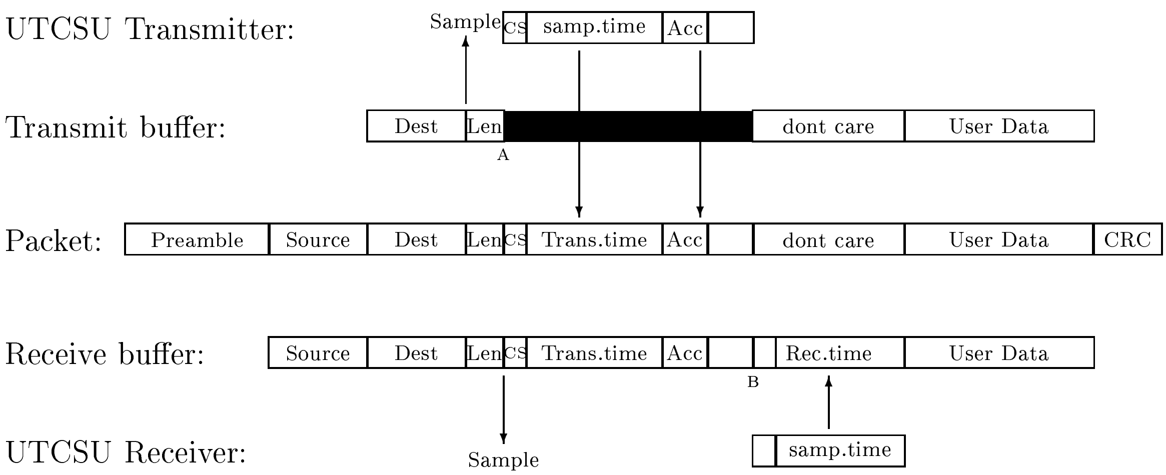


Figure 9: Packet Formats and Timestamping for Ethernet (32 bit architecture, no DMA)

This overhead could be avoided by an NCSC that supports transparent mapping at word level, requiring further data path transceivers. By arranging the position of the accuracy field in the CSP, word swapping may be avoided if STATELOW±/TRANSMIT is provided both in the lower and the upper word of a 32 bit read of the UTCSU with this type of NCSC design.

Of course, it would also be possible to use the DMA + TS-memory alternative mentioned in Section 4.2, as we did in the fieldbus example above. Don't care bytes in the CSP are not required in this case.

5 Support for Clock Rate Algorithm

Since rate intervals deteriorate much slower than state intervals, we will run the rate algorithm purely in software, without any special hardware support. Nevertheless two issues are worth of mentioning:

- To measure clock rates, the required non state-corrected value of a clock can be computed if the sum of all correction values $\sum_k \Delta c_k$ is known. To maintain another 91-bit uncorrected clock register in the UTCSU would certainly be an overkill, although it would improve the robustness of the rate measurement w.r.t. clock state faults.
- Since rate intervals deteriorate with time, it is necessary to increase LAMBDA± periodically. If that period —depending upon the 2nd order deviations of the local quartz clock— is smaller than the period of state resynchronization, it might be necessary to employ the ("continuous") amortization feature of the UTCSU to modify the value of LAMBDA± between normal resynchronizations.

6 GPS Coupling

Clearly, the interface to GPS receivers depends completely on signals and protocols offered by GPS-receivers. Most of them provide a 10 MHz reference frequency disciplined to GPS time and a 1 pps signal synchronized to GPS seconds within a few hundred ns. Higher-order time and additional information like leap-second pending is usually provided some considerably time after the 1 pps signal, usually via an RS232 interface.

The 1 pps pulse allows to view GPS as a sort of "nonstandard" SSN, which generates "CSP arrivals" every second. Consequently, the UTCSU should provide an asynchronous input line 1PPS that triggers sampling the NTP-bus into registers TIMESTAMP/GPS and MACROSTAMP/GPS. It should also cause a dedicated interrupt that is cleared when the sampled time is read off the register in the ISR.

Some GPS receivers also provide status information that helps assessing the quality of the provided GPS time. Therefore, an additional input line STATUS is provided by the UTCSU, which is sampled with 1PPS into the (otherwise used by the checksum CS) highest byte of the MACROSTAMP portion of the sampled local time. Note that the polarity of 1PPS and STATUS should be programmable. The following table summarizes the requires features.

elements for GPS coupling			
element	NTP bit#	length	specials
1PPS	—	1	polarity programmable
STATUS	—	1	polarity programmable
TIMESTAMP/GPS	[+7,-24]	32	from NTP-bus
MACROSTAMP/GPS	[1 byte]:[+31,+8]	32	from NTP-bus plus STATUS

It makes sense to connect several GPS receivers to a single UTCSU for redundancy. Supporting this architecture would also alleviate systematic evaluation of GPS receivers, since a single UTCSU feeded with a very stable oscillator could be used to observe several GPS receivers. At least 3, preferably 4 such GPS blocks are suggested.

There is even a further possibility of coupling insofar that we can use the (usually very stable) 10 MHz output frequency provided by most GPS receivers as oscillator frequency for the UTCSU.

7 Interfaces

7.1 Application Timing Features

A reasonable way is to leave high-level timing features outside the UTCSU, by making the carefully maintained NTP time available to other dedicated components via an externally supplied (multiplexed) NTP bus. Realize that all timing features (relative/absolute, once/periodic) eventually boil down to absolute timing with attached actions.

There is a feature that would greatly improve fault recovery: We should provide some bits in the UTCSU that tells whether an application referenced local time since the last synchronization. If so, higher level recovery is necessary, if not, restarting clock synchronization is sufficient.

7.2 Interrupts

To keep our UTCSU as flexible as possible, we should avoid vectorized interrupts. Several dedicated interrupt output lines should be provided instead:

- **INT_T** Interrupts to clock synchronization layer stemming from duty-timers and exceptions, e.g. accuracies exceed BOUND.
- **INT_N** Interrupts to clock synchronization layer originating from external events, like a pps signals from GPS receivers and reception or transmission of CSPs. Interrupts form the latter are logically equivalent to the ones from a network controller and can be disabled if desired.
- **INT_A** Interrupts to application layer, which are related to application timing capabilities that should not interfere with the internal ones. Besides interrupts for ASIC testing purposes should be placed here as well.

7.3 External NTP- and A-Bus

Both the full NTP-bus and the A-bus should be externally accessible. Of course, given the total of 96 bits, multiplexing is certainly necessary, and an automatic checksum verification bus protocol should be devised. This issue is difficult, because attached devices usually need the lsb of the NTP time ($\approx 120ns$), which makes multiplexing with a multiple of the clock frequency mandatory!

However, using externally attached devices for SSN-support is really worth while for high-accuracy applications, since a zero latency acquisition of transmission timestamps is possible here.

7.4 Miscellaneous

- In order to obtain high synchronization precision, low access-delays to UTCSU registers are required. However, this calls for synchronizer stages running at a multiple of the nominal oscillator frequency. Hence an external (or in future internal) PLL should be used.
- One should consider to use a nominal oscillator frequency of $2^{24} \approx 16$ MHz (at least at S-nodes) in order to decrease the effects of granularity (adding up due to transmission delay measurement etc.).
- It must be possible to re-initialize registers containing a constant value to correct for flipped bits.

8 Test Features

A tentative list looks like:

- Provisions to recognize flipped bits (error detecting codes)
- Selftests in general
- Some functionality for recognizing a stopped/slow clock
- Software-snapshot for atomically sampling all important (that is, time-dependent) internal UTCSU registers for debugging purposes
- Additional hardware-snapshot feature for atomically sampling the NTP- and A-bus, when a special pin is pulled (useful for experimental evaluations). Note that a conventional boundary scan is by far not that powerful.
- Provisions to bring clocks to a halt (SW-Reset, useful for warm starts)
- A "Go-Signal" for starting all UTCSU activities simultaneously by hardware

9 UTCSU Register Layout

Figure 10 depicts the whole register layout of the UTCSU. It is partitioned into blocks for local time, accuracy, SSN, GPS, snapshot and application. All registers are drawn to be aligned according to the NTP format.

References

[Hor94] M. Horauer. *Entwicklung einer Network Timestamp Unit für einen Versatile Timing Analyzer zum Monitoring von verteilten Echtzeitsystemen*, Diploma Thesis, Dept. of Computer Technology, Vienna University of Technology, 1994. (in German)

[KO87] H. Kopetz, W. Ochsenreiter. *Clock Synchronization in Distributed Real-Time Systems*, IEEE Trans. Comput. C-36(8), 1987, p. 933–939.

[Mil91] D.L. Mills. *Internet Time Synchronization: The Network Time Protocol*, IEEE Transactions on Communications, 39(10), October 1991, p. 1482–1493.

[Sch95] U. Schmid. *Synchronized Universal Time Coordinated for Distributed Real-Time Systems*, Control Engineering Practice, 3(6), 1995, p. 877–884.

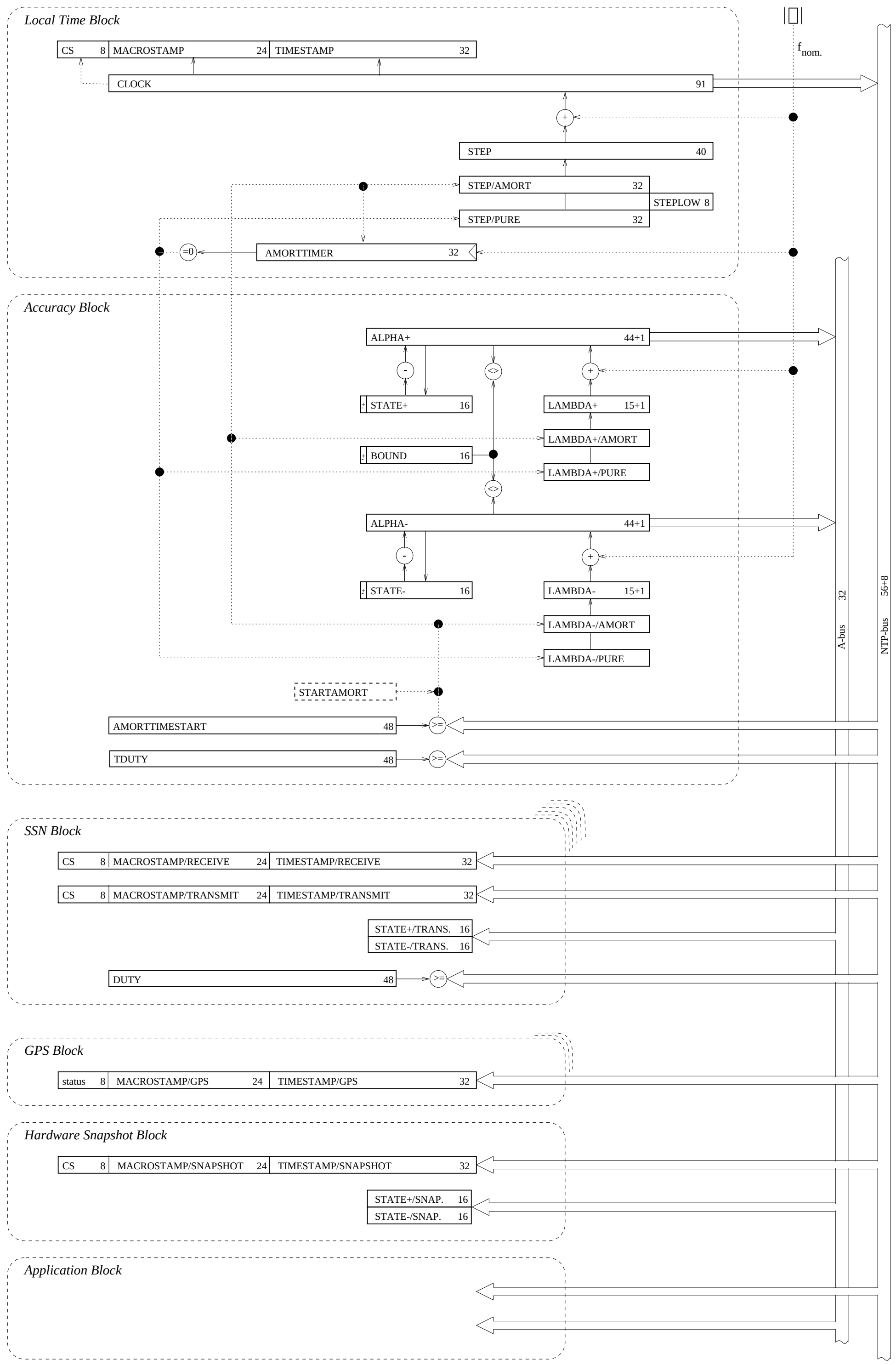


Figure 10: Total Register Layout of UTCSU