Special Issue on the Challenge of Global Time in Large-Scale Distributed Real-Time Systems, , 1–35 ()

Specification and Implementation of the Universal Time Coordinated Synchronization Unit (UTCSU) *

KLAUS SCHOSSMAIER kmschoss@auto.tuwien.ac.at
ULRICH SCHMID s@auto.tuwien.ac.at
Department of Automation, Vienna University of Technology, A-1040 Vienna, Austria

MARTIN HORAUER horauer@jupiter.ict.tuwien.ac.at
DIETMAR LOY loy@ict.tuwien.ac.at
Department of Computer Technology, Vienna University of Technology, A-1040 Vienna, Austria

Editor: W.A. Halang

Abstract. High-accuracy external clock synchronization can only be achieved with adequate hardware support. We analyze the requirements and present the specification and implementation of an ASIC running under the acronym UTCSU dedicated to that purpose. It is built around an elaborated local clock, which is based on an adder driven by a fixed-frequency oscillator. This novel clock design allows a fine grained rate adjustability apt for maintaining both local time with linear continuous amortization and accuracy information as needed in interval-based clock synchronization. Additional features incorporated in our UTCSU are facilities to timestamp clock synchronization data packets, interfaces to couple GPS receivers, some application support as well as sophisticated self-test machinery. Apart from addressing design and engineering issues of the chip, we also provide a basic programming model.

Keywords: Real-time systems, external clock synchronization, Universal Time Coordinated (UTC), Adder-Based Clock (ABC), linear continuous amortization, accuracy intervals, Application Specific Integrated Circuit (ASIC), Very high speed integrated circuit Hardware Description Language (VHDL), Global Positioning System (GPS).

1. Introduction

Dealing with time is inherent to the real-time computing domain, since applications need to interact both correctly and timely with the environment. In order to meet specified time constraints, activities like timestamping external events, scheduling resources, and initiating actions require an advanced time service. A clock is the physical basis of such a service, usually composed of a quartz oscillator that drives a hardware counter. Although small-scale systems are content with a central clock, modern large-scale real-time systems become necessarily distributed due to their spatial outspread and fault-tolerance requirements (Stankovic, 88). Exem-

* This work is part of project SynUTC, supported by the Austrian Science Foundation (FWF) under contract no. P10244-ÖMA. For further project information take a look at http://www.auto.tuwien.ac.at/~kmschoss/synutc.html

plary applications that exhibit such characteristics include transportation (e.g. avionics), manufacturing (e.g. rolling-mill), energy-providing (e.g. nuclear power plant) or scientific (e.g. radio-astronomy) systems. The distributed nature aggravates the installation of a time service considerably, because autonomous running clocks have a tendency to drift apart or might fail grossly in their rate or state. Hence clocks need to be synchronized by virtue of a suitable algorithm executed on each node. Usually, both performance and correctness of the system is vitally affected by the degree of synchrony, cf. (Liskov, 93).

Synchronizing an ensemble of distributed clocks comes in two flavors: *Internal clock synchronization* aims to keep the deviation between clocks bounded, i.e., if $C_i(t)$ and $C_j(t)$ denote two fault-free clocks within a system, the worst case *precision* π satisfies $|C_i(t) - C_j(t)| \leq \pi \ \forall t \geq t_0$. *External clock synchronization* relates to the problem that clocks are required to follow an external reference like UTC, the only legal standard of time. The maximum deviation towards UTC is called *accuracy* α, formally $|C_i(t) - t| \leq \alpha \ \forall t \geq t_0$. Reaching both goals jointly turns out to be a non-trivial research problem, as already pointed out in (Lamport, 87), since a certain tradeoff seems to be involved, cf. (Fetzer and Cristian, 96) in this special issue.

Internal clock synchronization is a well-established field in the distributed computing domain, see (Simons, Lundelius-Welch and Lynch, 90) for an overview and (Yang and Marsland, 93) for a bibliography. Most internal synchronization algorithms are purely software-based, i.e., they run on off-the-shelf processing and networking hardware, providing a precision in the 10 ms-range only. A considerably better precision can be achieved with dedicated hardware support. In the fieldbus area, for instance, there are some more recent research activities, like ESPRIT OLCHFA (Feldmann and Solvie, 95) or the CAN-Bus project by (Gergeleit and Streich, 94), that target a few ms. A time service with precision in the 10 μs-range can be built on top of the pioneering *Clock Synchronization Unit* (CSU) described in (Kopetz, 89). Similar ideas are exploited in the hardware assisted clock synchronization scheme of (Ramanathan, Kandlur and Shin, 90). Even smaller precisions can be attained by means of clock voting with phase locked loops, cf. (Ramanathan, Shin and Butler, 90), but we do not consider such solutions because of their extra clocking network.

The most widely used external clock synchronization scheme is undoubtly the *Network Time Protocol* (NTP) designed for disseminating UTC among workstations throughout the Internet, see (Mills, 91) for an overview. Under realistic conditions, worst case accuracies of approximately 20 ms were observed by (Troxel, 94). There are also some recent solutions of the external synchronization problem in the LAN domain, cf. (Fetzer and Cristian, 96), (Schmid and Schossmaier, 96) or (Verissimo, Rodrigues and Casimiro, 96) all in this special issue. The latter describes a software-based approach that "sprays" external time obtained from GPS satellites into broadcast-type LANs with accuracies in the 10 μs-range.

In our SynUTC project, outlined in (Schmid, 95), we focus on external clock synchronization for large-scale distributed real-time systems and aim 1 μs as both precision and accuracy. Of course, such ambitious goals can only be achieved with proper hardware support. We packed most of it into an ASIC, termed *Universal Time Coordinated Synchronization Unit* (UTCSU) [1], which is the main concern of this paper.

The outline is as follows: After a brief introduction of the overall system architecture in Section 2, we present a functional specification of the required hardware support in Section 3. It encompasses features of in the UTCSU, like our novel adder-based clock, and the embedding hardware. The UTCSU is decomposed into physical units in Section 4, along with an explanation of implementation details. A programming model of the chip is given in Section 5, and Section 6 rounds off by discussing design methodology concerns. Finally, we conclude by reviewing UTCSU centerpieces and by pointing out future developments.

2. System Architecture

From an abstract point of view, modern real-time systems are physically distributed networks of nodes hosting hard- and software resources for providing application-specific services that exhibit predictable behavior. Our focus rests on the time service, a basic subsystem of any real-time system, that offers its pertinent timing features to a number of higher-level services.

2.1. Synchronization Subnets

In order to cope with the complexity of a typical real-time system, we decompose it into so called *Synchronization Subnets* (SSN), similar to (Mills, 91). Each SSN comprises a collection of nodes interconnected by a packet-oriented data network. Regarding the contribution to the time service and considering cost-performance tradeoffs, we can distinguish between four types of nodes:

- *Client-nodes* execute the synchronization algorithm in a passive way. More precisely, they merely glean synchronization data from the attached SSN and adjust their local clocks accordingly.
- *Secondary-nodes* execute the synchronization algorithm in an active way by periodically exchanging synchronization data among other non-Client nodes within the SSN. Their purpose is to provide fault-tolerance, most importantly, guaranteed precision even in case of total loss of external time (aka. *flywheeling*).
- *Primary-nodes* behave like Secondary-nodes but provide access to external time, e.g. GPS receivers, as well. Multiple Primary-nodes are useful for increasing the fault-tolerance level w.r.t. faults of external time sources.
- *Gateway-nodes* act as Secondary-nodes in two or more SSNs, making time dissemination between SSNs feasible. Such nodes have additional functionalities to control the system-wide flow of synchronization data.

An example of a simple system architecture is given in Figure 1, showing three SSNs linked by a single and a double Gateway-node connection. Note that, irregardless of this example, a tree-like hierarchy is not mandatory for our approach.

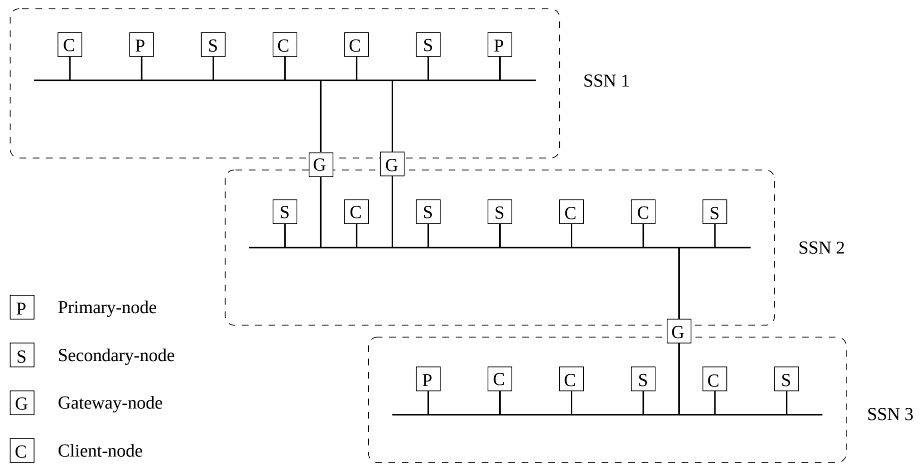


Figure 1. System Architecture

2.2. Node Hardware

Although nodes may have different functionalities, their hardware architecture remains to be uniform. Figure 2 shows a node's components appertaining to the time service.

We restricted ourselves to use a customary packet-oriented data communication subsystem as the only means to exchange synchronization data, hence excluding hardware clocking systems relying on dedicated channels, cf. (Ramanathan, Shin and Butler, 90). A DMA-type *communication coprocessor* (COMCO) provides access to the network, exchanging *clock synchronization packets* (CSPs) and communicating with the CPU via local shared memory. The CPU, which can either be the node's central processor or, preferably, a dedicated microprocessor or microcontroller, is responsible for running the clock synchronization algorithm. The last component in Figure 2, the UTCSU, contains the bulk of hardware support required for clock synchronization. We defer its description to subsequent sections.

At this point, it only remains to explain why we impose a DMA-type communication coprocessor. A major concern in highly accurate/precise clock synchronization is exact timestamping of CSPs at both sending and receiving side. In fact, the well

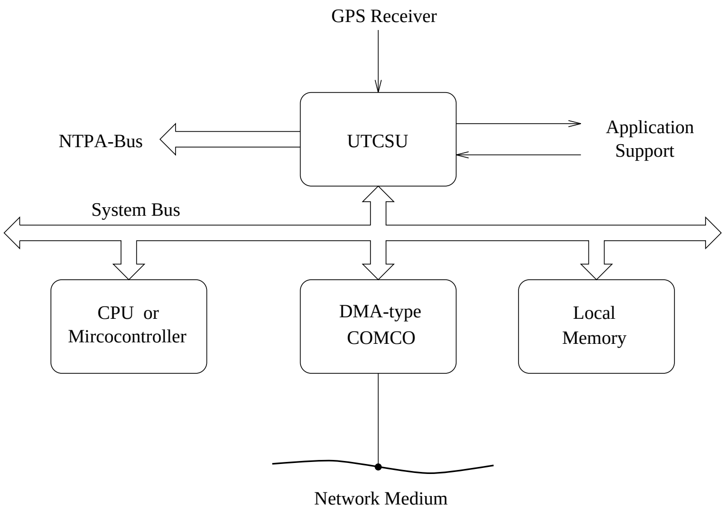


Figure 2. Components of a node

known result of (Lundelius and Lynch, 84), saying that even n ideal clocks cannot be synchronized closer than $\epsilon(1-1/n)$ in case of packet delivery time uncertainty ϵ, justifies this requirement. Hence, the challenge is to insert a timestamp on-the-fly into a CSP that minimizes the variance of transmission/reception latencies. To accomplish this, clock synchronization hardware must be placed as close to the network as possible. Ideally, a CSP should be timestamped at the sender resp. receiver exactly when, say, its first byte is pushed on resp. pulled from the medium. However, this would require support from the interior of the network controller, which is usually not available.

In order to make our approach compatible with existing network technology, we employ a refinement of the widely applicable DMA-based coupling method proposed in (Kopetz and Ochsenreiter, 87). It is based on a modified address decoding logic for the local memory, which

- generates timestamp request signals when a certain byte within the transmit/receive buffer of a CSP is read/written, and
- transparently maps certain registers of the UTCSU containing the sampled timestamps into some portions of the transmit/receive buffer.

To illustrate the process of packet stamping, we briefly discuss one possible scenario depicted in Figure 3. In order to send a CSP in the course of executing the algorithm, the CPU puts the packet into local memory and signals the COMCO to take over for transmission. The COMCO in turn fetches CSP data from local memory, performs the required parallel to serial conversion, and pushes the bit stream onto the medium. Having completed those steps, which are performed concurrently with further CPU operations, the CPU is notified about completion of the transmission. What matters is that whenever the COMCO fetches data from the transmit buffer, it has to read across the destination address causing the decoding logic to generate signal TRANSMIT. The UTCSU puts a *transmit timestamp* into a sample register upon occurrence of this signal, which is inserted into a dedicated portion within he packet by transparently mapping the register holding it into the transmit buffer.

By the same token, when the COMCO on the receiving side writes across a predefined address within the receive buffer in local memory, signal RECEIVE is generated by the decoding logic, making the UCTCSU to sample the *receive timestamp*. It can be transferred into the receive buffer or, alternatively, to a dedicated memory partition later on when the received packets are processed.

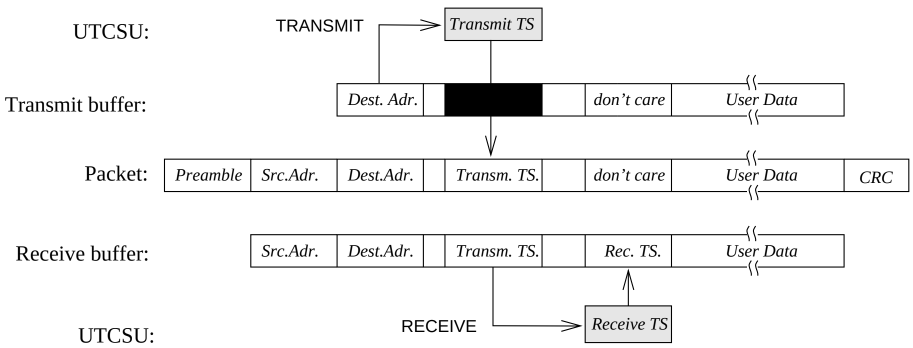


Figure 3. Packet timestamping

DMA-type communication coprocessors are available for a wide variety of networks, ranging from fieldbusses (e.g. CAN or Profibus) over Ethernet based networks up to advanced high-speed FDDI or ATM networks. Unfortunately, most advanced CAN-controllers provide on-chip storage for whole packets, making them unsuitable for our approach. Moreover, we should mention that proving suitability usually means experimental evaluation due to the fact that most controllers utilize internal FIFOs, which introduce uncertainties in the time between fetching a byte from local memory and putting it on the medium. Whereas adjusting the triggering position of transmit/receive timestamp could help in circumventing FIFO impairments, it is nevertheless not easy to find and justify a suitable choice. In general, we should point out that numerous technical hurdles have to be surmounted to make this approach working properly, see (Horauer, 94) for the twisted details of a particular prototype implementation [2].

3. Functional Specification

Modularity, flexibility, performance and other reasons as argued in (Schroetter, 92) influenced us to realize the UTCSU in Figure 2 as an ASIC. The subsequent sections will shed more light on its interior, first by means of a functional specification, see (Schossmaier and Schmid, 95) for a comprehensive version, and later on by documenting the development process, see (Loy, 96) for meticulous details.

3.1. Interfacing

During requirement analysis of the chip we took much care to keep all interfaces as straightforward and general as possible. Our first step to specify the UTCSU is done by giving an interface outline guided by Figure 2.

Coupling the UTCSU to the *System-Bus* enables communication with the other components. A wide range of existing bus architectures, employing data bus widths of 8, 16 or 32 bits, big/little endian byte ordering, different bus access times, interrupt schemes, etc. should be supported without additional glue logic. Note that the connection to the System-Bus is also a prerequisite for transparently mapping UTCSU-registers containing sampled timestamps into transmit/receive buffers in local memory, as required for exact timestamping of CSPs.

Meeting accuracy requirement of 1 μs, we decided to use GPS technology for injection of external time, hence the UTCSU must be able to interface various GPS timing-receivers [3]. For applications with only moderate accuracy requirements or to increase fault tolerance, other sources of external time, like receivers for DCF77 or WWV, cf. (Lichtenecker, 96) in this special issue, can be connected to the UTCSU via the same interface.

To support applications, there should be at least means to trigger actions at programmable points in time and capabilities for event timestamping. Moreover, to ease the implementation of more elaborate timing features, like the proposed timing processor of (Halang and Wannemacher, 96), without changing the UTCSU, we export time/accuracy information via a local unidirectional *NTPA-Bus* for external processing.

Depending on the type of node, these interfaces will have different significance. For instance, only Primary-nodes make use of the GPS interface, whereas Gateway-nodes utilize the interface to several COMCOs extensively. In summary, we give the first rather loose specification rule

SPECS 1 (INTERFACES): *The UTCSU must provide a versatile interface to commercially available system-busses and GPS timing-receivers, and additional interfacing facilities to timestamp and generate external pulses. Furthermore, it should export local time/accuracy information on a dedicated NTPA-Bus.*

3.2. Maintaining Local Time

The maintenance of local time at a node will be the core function of the UTCSU. In the first place, a format has to be established to encode local time. We use a straightforward binary coding scheme, closely following the NTP-time format from (Mills, 91), where the upper 32 bits are interpreted as standard seconds relative to UTC and the lower 32 bits give the associated fractional part. For specification purposes we will use the following notation to denote a specific part of this format: (u, v) covers bits ranging from the most significant bit position u to the least significant one v, thus affecting a quantity given by $\sum_{i=v}^{u} b_i \, 2^i$, where $b_i \in \{0, 1\}$. An optional prefix "$\pm$" denotes a signed value. Table A.1 in the Appendix defines NTP-bit numbers and their time equivalents for the ease of reference. For example, NTP-range $(-1, -8)$ pinpoints values up to slightly less then a second in multiples of 3.81 ms.

The clock synchronization algorithm is responsible for computing adjustments for both state and rate in order to meet accuracy and precision requirements. *State adjustment* means to add/subtract/set instantaneously a particular amount to local time, and *rate adjustment* means slowing down/speeding up its progression. Thus, we need to incorporate a clock in the UTCSU that represents local time in the NTP-based format and is adjustable in both rate and state.

SPECS 2 (LOCAL TIME): *The UTCSU needs to host a digital clock representing local time over the NTP-range* $(+31, -24)$*, which is arbitrarily state adjustable for initialization purposes and rate adjustable not coarser than* 10^{-8} s/s*.*

As mentioned earlier, UTC is going to be our reference timescale, because of its worldwide availability through GPS technology and its legal character. However, UTC should be carefully considered in conjunction with real-time systems due to its non-chronoscopic nature, cf. (Kopetz and Ochsenreiter, 87) The *Bureau International de l'Heure* (BIH) inserts or deletes leap seconds at predefined points in time in order to harmonize it with astronomically derived timescales, e.g. *Universal Time* (UT) versions. Strictly speaking, chronoscopic time standards, such as *Temps Atomique International* (TAI) or *GPS-Time*, should be given preference to establish a time service for real-time systems. Notwithstanding that, we proclaim

SPECS 3 (LEAPS SECONDS): *The clock for local time inside the UTCSU has to be equipped with facilities to handle leap seconds by either inserting or deleting pending ones at programmable points in time.*

3.3. Maintaining Accuracy Intervals

Dealing with the accuracy requirement means that local time has to follow UTC as close as possible. Unfortunately, UTC is neither directly observable nor permanently accessible, so only a range can be determined where UTC currently lies.

More specifically, in our setting we use an *accuracy interval* A(t) capturing UTC in the sense that $t \in A(t)\ \forall t \geq t_0$. This interval-based approach was introduced in (Marzullo, 84), (Marzullo and Owicki, 85) and continued in (Lamport, 87). The OSF/DCE time model (OSF, 92) and newer versions of NTP (Mills, 95) make use of this notion of time as well.

Our UTCSU is tailored to support external clock synchronization using the interval-based *clock validation technique* proposed in (Schmid, 95), see also (Schmid and Schossmaier, 96) in this special issue. The algorithms employed herein rely on accuracy intervals that are maintained locally at a node by an *upper accuracy* $\alpha^+(t)$ and a *lower accuracy* $\alpha^-(t)$ meant relative to the local clock $C(t)$, such that $A(t) = [C(t) - \alpha^-(t), C(t) + \alpha^+(t)]$. Observe that accuracies are always understood as maximum UTC deviations, otherwise we would be clairvoyant. The reason to make accuracies time-dependent is to enlarge them properly in order to account for maximum oscillator drifts, hence sustaining UTC inclusion. This process of linear *deterioration* would go on perpetually, however, periodic resynchronizations executed by a clock synchronization algorithm aim to shrink $A(t)$, by exploiting knowledge of UTC provided by GPS receivers.

SPECS 4 (ACCURACY INTERVAL): *The UTCSU needs to maintain an asymmetrical accuracy interval over the NTP-range* $(-8, -23)$, *which is arbitrarily adjustable and performs deterioration in the range of* $10^{-8} \ldots 10^{-4}$ s/s.

A few supplements to maintain accuracy intervals need to be brought to attention. The accuracy values should be permanently compared against bounding registers, generating an interrupt whenever the former exceeds the latter. Furthermore, a wrap-around during deterioration is undesirable; actually, in such situations we require that accuracy registers stay at their maximum value and trigger an interrupt.

SPECS 5 (ACCURACY INTERVAL OVERRUNS): *The upper and lower accuracy values inside the UTCSU need not wrap-around, and in case of exceeding the corresponding* $(-8, -23)$*-register* BOUND$\pm$ *a dedicated interrupt should be raised.*

3.4. Adder-Based Clocks

Meeting the specified rate adjustability for local time or the deterioration dynamics for accuracies with ordinary clocks consisting of an oscillator pacing a hardware counter turns out to be a challenging task. As a matter of fact, conventional discrete rate adjustment techniques, like tick advancing/delaying employed in the CSU of (Kopetz and Ochsenreiter, 87), would require very high oscillator frequencies (around 100 MHz) for a smooth rate adjustment of 10^{-8} s/s. Moreover, our NTP-based time representation would enforce a binary oscillator frequency. One alternative pursued in (Mills, 92) replaces the fixed-frequency oscillator by a voltage controlled one (VCO) that is put into a carefully engineered phase-locked loop (PLL).

We propose an *adder-based clock* (ABC) in order to maintain local time, where each oscillator tick entails an addition of a particular amount *clock-step* to a register holding local time. Any rate change can easily be achieved by varying clock-step, which goes into effect instantaneously and retains linearity. To make this approach working properly, however, we have to expand the registers holding local time and clock-step on the less significant side to accumulate minute time amounts. Targeting a nominal frequency of 2^{24} Hz, it becomes necessary to extend the NTP-range of local time to $(+31, -51)$ in order to reach the demanded rate adjustability, since $2^{-51}/2^{-24} \approx 7.45 \cdot 10^{-9}\ s/s$. Of course, timestamps derived for local time are only predicative in the NTP-range $(+31, -24)$, so that the smallest meaningful unit of time becomes to 59.6 ns, called *time granularity.* Bits $(-25, -51)$ are concealed from the outside and solely used for rate correction purposes, whereby the smallest perceivable unit of time becomes to 444.1 as, called *clock granularity.*

Beyond that, the adder-based approach allows us to use a non-binary oscillator frequency, e.g. 10 MHz from GPS timing-receivers. However, the NTP-range of local time has to be further extended to preserve the usually excellent quality of such frequency sources. We found it sufficient to append 8 bits on the less significant side, thus yielding a range of $(+31, -59)$, since truncation errors are limited to 2^{-59} s per oscillator tick. In particular, for a 10 MHz input frequency, the truncation errors impair the frequency by approximately $0.6 \cdot 10^{-11}$.

SPECS 6 (ABC FOR MAINTAINING LOCAL TIME): *The UTCSU needs to implement an adder-based clock, composed of a* $(+31, -59)$*-register* NTPTIME *representing local time, an external oscillator input designed for frequencies in the range of* $2^{20} \ldots 2^{24}$ *Hz and a* $(-20, -59)$*-register* STEP *to hold the increment for each oscillator tick.*

For a better understanding of adder-based clocks, let us briefly examine their operation in more detail. As mentioned above, a particular clock-step is added at each oscillator tick, which can be separated into the reciprocal of the nominal oscillator frequency and a fractional correction value. Since our clock exhibits a much coarser time granularity than clock granularity, an accumulation of correction values may cause discontinuities in the sequence of clock states. More specifically, when the correction values run up to the time granularity at a particular tick, the adder-based clock makes either no advancement or advances by twice the time granularity, depending on the accumulation nature. When correcting an oscillator drift of say 1 ppm with our clock specified in SPECS 6, such effects occur roughly every 60 ms for using a nominal input frequency of 2^{24} Hz. For an in-depth treatment of granularities entagled in clock synchronization consult (Schmid and Schossmaier, 96) in this special issue.

The adder-based approach is also well suited for maintaining the upper accuracy $\alpha^{+}(t)$ and the lower one $\alpha^{-}(t)$. In fact, each accuracy quantity can be tracked by an ABC similar to the one for local time, where one register is playing the accumulating role and another one is holding the deterioration for each oscillator tick. In the sequel we denote ABC components for the lower resp. upper accuracy by adding the suffix "-" resp. "+", or "±" to capture both.

The registers for an ABC representing accuracy span 45 bits internally, including a sign bit, thus ranging over $\pm(-8,-51)$. Note that only the upper 16 bits are considered by the clock synchronization algorithm. Targeting accuracies smaller than 2^{-23} $s \approx 120$ ns appears unrealistic with our approach. Nonetheless, it is facile to scale accuracy values externally as much as desired.

The deterioration registers are 16 bit wide, including a sign bit, covering the NTP-range $\pm(-37,-51)$. This arrangement allows a minimum deterioration of about 10^{-8} s/s, by the same line of justification as for the local clock, and a maximum deterioration of 2^{-36} $s/tick$ or approximately 244 $\mu s/s$ for a nominal oscillator frequency of 2^{24} Hz, which should be sufficient even for worst quartz oscillators. The necessity of sign bits arises from continuous amortization, as explained in Section 3.5.

SPECS 7 (ABCS FOR MAINTAINING AN ACCURACY INTERVAL): *The UTCSU needs to implement two adder-based clocks for the maintenance of an asymmetrical accuracy interval, composed of* $\pm(-8,-51)$*-registers* ALPHA±*, an external oscillator input designed for frequencies in the range of* $2^{20}\ldots2^{24}$ *Hz (the same as for local time), and* $\pm(-37,-51)$*-registers* LAMDBA± *to hold the deterioration for each oscillator tick.*

The UTCSU is designed for a maximum operating frequency of 2^{24} Hz. Trading ASIC production costs against time service qualities and considering downsizing the chip for specific applications, we allow to run the UTSCU with lower frequencies as well. Changing the frequency impacts clock characteristics as shown in Table 1. A lower operating frequency increases the clock rate adjustability (expressed as how long a 1 μs correction takes by applying the smallest non-zero clock-step change), renders the maximum accuracy deterioration smaller, and reduces the meaningful timestamp range.

Table 1. UTCSU characteristics for customary operating frequencies

nominal oscillator frequency [Hz]	rate adjustability [max s for 1 μs]	maximum accuracy deterioration [$\mu s/s$]	meaningful time-stamp range
$2^{20} = 1,048.576$	2148	15	$(+31,-20)$
$2^{21} = 2,097.152$	1073	31	$(+31,-21)$
$2^{22} = 4,194.304$	537	61	$(+31,-22)$
$2^{23} = 8,388.608$	268	122	$(+31,-23)$
$10,000.000$	255	146	$(+31,-23)$
$2^{24} = 16,777.216$	134	244	$(+31,-24)$

3.5. Continuous Amortization

Periodically, the clock synchronization algorithm becomes active and computes adjustments for both rate and state in order to achieve internal/external synchronization. Enforcing these adjustments deserves special attention, since real-time applications dictate specific properties. In particular, timestamps obtained from local clocks need to be monotonic and free of discontinuities of predefined extent. A technique called *linear continuous amortization* is used to carry out state adjustments, cf. (Schmuck and Christian, 90). As a novelty, the UTCSU provides hardware support for this clock setting method.

The basic principle is rather simple: Instead of adjusting the clock state instantaneously, we achieve the same effect by modifying the clock rate for a specific amount of time, called *amortization period* t_{amort}. During this time we say that the local clock is in its *amortization phase*, whereas the remaining period is known as *pure phase*. As an obvious consequence, these two phases alternate perpetually, controlled by the clock synchronization algorithm.

On the other hand, accuracies are allowed to change instantaneously, since they are kept relative to the local clock. However, deteriorations must be modified during the amortization phase due to the clock rate set forth by state adjustment.

Figure 4 gives an example to illustrate both phases. The period before t_1 shows clock $C(t)$ in its pure phase, advancing with pure rate $\dot{C}(t) = \omega$, and accuracies $\alpha^{\pm}(t)$ growing with pure deterioration $\dot{\alpha}^{\pm}(t) = \lambda^{\pm}$. Before kicking off continuous amortization, the parameters for both amortization phase and successive pure phase need to be computed. The amortization phase commences at t_1, which entails a reduction of $\alpha^{\pm}(t)$ by the accuracy adjustments $a^{\pm}$, clock $C(t)$ advances from now on with the amortized rate $\hat{\omega}$, and finally $\alpha^{\pm}(t)$ deteriorates with $\hat{\lambda}^{\pm}$. A counter times out t_{amort} at t_2, causing the transition to the new pure phase.

In our example, $\alpha^{+}(t)$ shrinks during the amortization phase, and $\alpha^{-}(t)$ happens to be negative at the beginning, which justifies the sign bits introduced in Specs 7. To remedy the passage with negative accuracy, the externally accessible accuracy is set to zero during this time, which obviously implies a valid accuracy interval. Recall that we have to maintain the invariant that UTC is permanently enclosed by envelopes $e^{-}(t)$ and $e^{+}(t)$ defined by $C(t) - \alpha^{-}(t)$ and $C(t) + \alpha^{+}(t)$, respectively.

Having explained the mechanism for continuous amortization, we specify additional UTCSU elements separated in issues concerning local time and accuracy.

Specs 8 (Continuous Amortization ref. Local Time): *The adder-based clock for local time has to carry out state adjustments by continuous amortization. To that end, features to switch between pure/amortized clock rates need to be implemented, to commence the amortization phase at a programmable point in time, and a* $(+8, -24)$*-counter* AMORTTIMER *to earmark the end.*

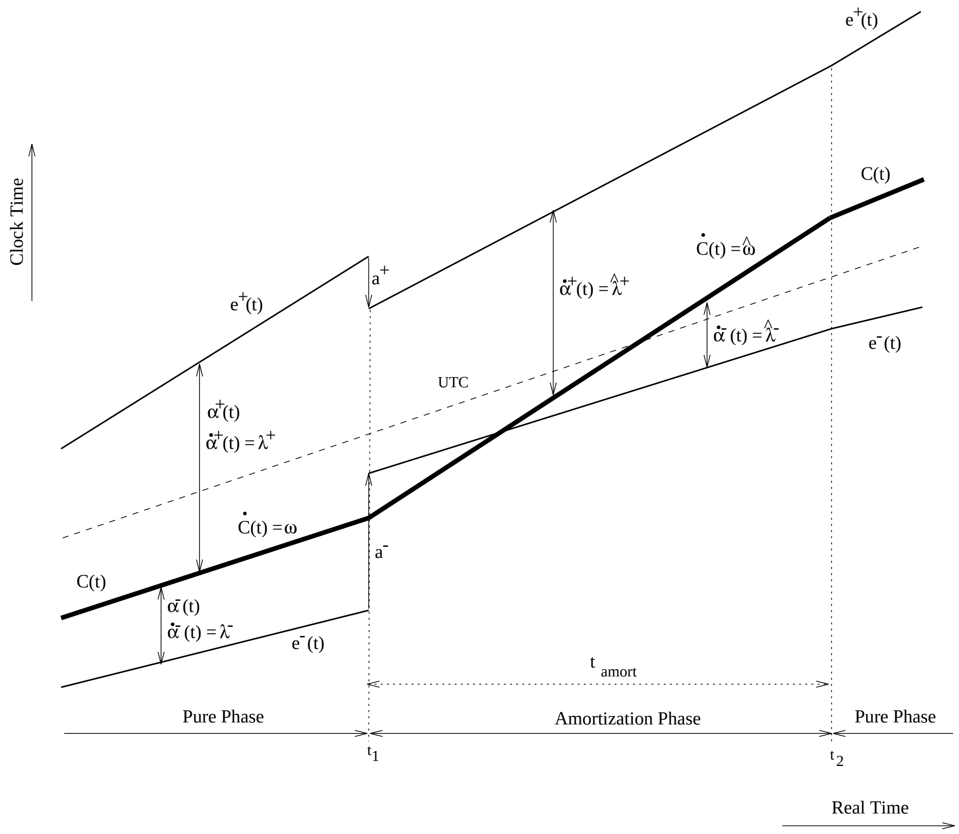


Figure 4. Pure and Amortization Phase

SPECS 9 (CONTINUOUS AMORTIZATION REF. ACCURACY): *The two adder-based clocks for the accuracy interval must switch in lock-step with the clock for local time between pure/amortized deterioration. Additionally, registers* ALPHA± *need to be relative adjustable by* $\pm(-8,-38)$*-registers* STATESET±*, and negative accuracies during the amortization phase have to be suppressed by forcing the externally accessible part* $(-8,-23)$ *to zero.*

If Υ denotes the amount that clock $C(t)$ gains during continuous amortization w.r.t. the non-amortized clock, the previous pure rate ω, the amortized rate $\hat{\omega}$, and the start value T_{amort} of counter AMORTTIMER are related by

$$\frac{\Upsilon}{T_{\mathrm{amort}}} = \hat{\omega} - \omega = \psi.$$

Apparently, there is one degree of freedom involved. Making T_{amort} large entails a smooth transition but might jeopardize the precision/accuracy of the local clock. However, we showed in (Schmid and Schossmaier, 96) that there exists an upper bound on T_{amort} given by $\psi \geq \rho_{\max}$, where $\rho_{\max}$ denotes the sum of the maximum positive and negative drift of any clock in the system, such that results obtained for an instantaneous clock adjustment can be carried over to the continuous amortization case.

3.6. Event Timestamping and Generating

Hitherto we have specified elements to maintain local time and accuracies, which will eventually be used for event time/accuracy-stamping and event generating purposes. These two features play an important role in several domains.

The first one assists in exact CSP timestamping at both sending and receiving side. As argued in Section 2.2 this feature is crucial for tight clock synchronizations. Extending the pioneering work of (Kopetz and Ochsenreiter, 87), CSPs are stamped with local time and accuracy just at the moment when they are actually leaving the node, and with local time when arriving at the peer node. This necessitates coordinated support from the UTCSU and the embedding hardware. The UTCSU obtains the CSP transmission/reception events via dedicated input lines for latching time/accuracy in dedicated registers, and further the embedding hardware is in charge of mapping them transparently into the CSP transmit/receive buffer, as already elaborated in Section 2.2.

Considering Gateway-nodes and the need to cope with fault-tolerant communication architectures (i.e. triple redundant), we have to accommodate multiple units supporting independent CSP stamping.

SPECS 10 (CSP STAMPING): *The UTCSU needs to be equipped with six units each capable of sampling the* $(+31, -24)$*-range of register* NTPTIME *and the* $(-8, -23)$*-range of registers* ALPHA$\pm$ *on a CSP transmission, and the* $(+31, -24)$*-range of register* NTPTIME *on a CSP arrival. Both types of events are announced by special polarity programmable input lines* TRANSMIT[1..6] *and* RECEIVE[1..6].

In a similar way, an external time reference is linked to the UTCSU. We decided to use the GPS system to inject UTC due to our high accuracy and availability requirements. GPS is an earth orbiting satellite based navigation system operated by the US AIR FORCE under the direction of the DEPARTMENT OF DEFENSE, see (Dana, 96) in this special issue for an overview. It includes a *Standard Positioning Service* (SPS) for a worldwide civilian use. When *Selective Availability* (SA) is enabled, the horizontal position can be obtained within 100 m (95 percent) and GPS-Time with a maximum error of 340 ns (95 percent). Note that GPS-Time is steered to be within 1μs of UTC, without taking leap seconds into account. It is derived from atomic clocks both at ground stations and on board the satellites.

There is a large number of different GPS timing-receivers available. Nearly all of them output an on-time pulse at 1 pulse per second (aka. *1PPS*), and more expensive models also provide a 10 MHz reference frequency disciplined to GPS-Time. The associated information identifying the pulse along with other data is provided some considerable time after the 1PPS, usually via a serial interface. We exempt the UTCSU from processing this information and defer it to the CPU. Finally, a few GPS timing-receivers even tell their status on-line (e.g. data valid, satellites out of view) by means of special output lines.

For redundancy purpose and for the sake of testing several receivers against each other, it appears appropriate to equip the UTCSU with multiple units for coupling GPS receivers.

SPECS 11 (GPS COUPLING): *The UTCSU needs to be endowed with three independent units capable of sampling the* $(+31, -24)$*-range of register* NTPTIME *and the status of the connected GPS timing-receiver whenever the corresponding 1PPS becomes active. Polarity programmable input lines* 1PPS[1..3] *resp.* STATUS[1..3] *mediate these events resp. report the receiver status.*

Most clock synchronization algorithms periodically invoke a routine that broadcasts local time/accuracy via CSPs to the other nodes within the SSN. Hence, the UTCSU has to provide one *duty-timer* (DT) for starting a CSP transmission and one for terminating the reception period. A duty-timer consists of a writable register and a comparator to check whether local time is equal or greater, which will be reported by a dedicated interrupt. Special care is required to arm/disarm such DTs to avoid programming pitfalls, cf. Section 5.2.

SPECS 12 (DUTY TIMERS): *The UTCSU needs to provide two freely programmable duty-timers for each SSN attachment to support the protocol for CSP exchange, in total* DUTYA[1..6] *and* DUTYB[1..6] *all ranging over* $(+31, -16)$. *If they equal or exceed the current local time then a dedicated interrupt should be raised.*

Indeed, application requirements on a time service can vary considerably. The conceivable spectrum ranges from basic features, e.g. reading the local clock or providing a programmable frequency output, over more advanced ones, like a time-stamp FIFO as required by the distributed event-based monitoring system VTA of (Schmid, 94), up to elaborate timing support, such as the high-precision timer developed by (Halang and Wannemacher, 96).

We decided to provide only basic on-chip application support. It includes atomic readings of local time and accuracy, time/accuracy-stamping of external application-related events, and generation of interrupts with the help of a duty timer. Future advanced application features can be realized externally by tapping the NTPA-Bus, which exports local time and accuracy as showed in Figure 2. The following specification rule reveals more details about all those UTCSU features.

SPECS 13 (APPLICATION SUPPORT): *First, register* NTPTIME *in the* $(+31, -24)$*-range and registers* ALPHA$\pm$ *in the* $(-8, -23)$*-range must be atomically readable by simple read operations. Second, the UTCSU needs to be furnished with nine independent units dedicated to application-support for stamping events that occur on polarity programmable input lines* APP[1..9]*, with local time/accuracy in the same ranges as above. Third, a dedicated* NTPA-Bus *has to export local time/accuracy, again in the same ranges. Finally, a* $(+31, -16)$*-duty-timer* APPL *is to be provided for generating external pulses and application-specific interrupts.*

3.7. Self-Test and Debugging Features

To ease system development and implementation of applications depending on a time service, most test- and debugging-features should be supported by hardware. A classical method is to guard register NTPTIME with additional control bits, providing a means to detect/correct flipped bits to some extent.

SPECS 14 (CHECKSUMS): *The UTCSU must protect the* $(+31, -24)$*-range of register* NTPTIME *by computing an appropriate 8 bit checksum to enforce a Hamming distance of 4.*

Another well known self-checking mechanism is based on compression functions over a sequence of time/accuracy values. More specifically, the UTCSU should compute two functions over a certain time frame: *blocksums*, which are summations over a specified range, and *signatures*, which employ a generator polynomial akin to CRC-checksums. These calculations can be done in parallel by a redundant UTCSU or by a general purpose CPU, occasionally verified against the values derived inside the UTCSU. Note that one has to trade (low) checking frequency for (high) detection latency.

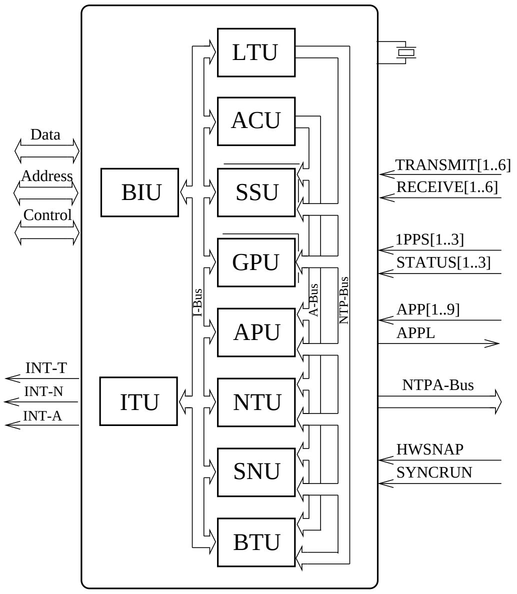
SPECS 15 (BLOCKSUMS AND SIGNATURES): *The UTCSU needs to compute blocksums and signatures of the* $(+31, -24)$*-range of register* NTPTIME *and the* $(+8, -23)$*-range of registers* ALPHA$\pm$ *over a start/stop prearranged period of time.*

A *snapshot* denotes a mechanism that samples relevant internal registers to get a glance of the current UTCSU state. Either triggered by hardware or software, registers NTPTIME and ALPHA$\pm$ need to be saved in their full extent to certain shadow registers for postprocessing. Besides checking for local errors, this mechanism allows experimental evaluation of the time service precision, by triggering simultaneously snapshots on all nodes with a common line. To be complete, another special input line allows the UTCSU to (re)start its operation from a well defined state when an external pulse occurs. This features multiple redundant UTCSUs starting simultaneously at each node and allows to bypass elegantly initial clock synchronization for the sake of testing purposes.

SPECS 16 (SNAPSHOTS): *The UTCSU needs to include a soft- and hardware snapshot mechanism to make atomic full-length shots of registers* NTPTIME *and* ALPHA±. *Furthermore we require a feature to start the UTCSU from a well defined state by activating the special input line* SYNCRUN.

4. Unit Implementation

Functional issues, ensuing from our desire to capture all hardware requirements for a highly accurate/precise time service, guided the process of developing the UTCSU specification. In the course of designing this ASIC, we crafted a layout of physical units fostering a clean top-down implementation. More specifically, starting from the specification (Schossmaier and Schmid, 95) compiled in Section 3, the UTCSU internal units portrayed in Figure 5 were eventually identified and designed in (Loy, 96). A first look at the boundary reveals interfaces for the System-Bus, GPS receivers, CSP stamping, and applications. The interior consists of 10 generic units connected by a 32 bit broad I-Bus, resulting in a homogeneous architecture. The following subsections introduce these units in some detail, depending on the level of interest and originality.



unit	full name
ACU	Accuracy Unit
APU	Application Unit
BIU	Bus Interface Unit
BTU	Built-In Test Unit
GPU	GPS Unit
ITU	Interrupt Unit
LTU	Local Time Unit
NTU	Network Time Interface Unit
SNU	Snapshot Unit
SSU	Synchronization Subnet Unit

Figure 5. UTCSU block diagram

4.1. Bus Interface Unit (BIU)

The BIU contains logic for embedding the UTCSU in a wide variety of system architectures as demanded in SPECS 1. In particular, it makes the ASIC applicable to System-Bus widths of 8, 16 or 32 bits, little or big endian byte ordering, and different access times. A technique called *dynamic bus sizing* is used to solve this common interfacing problem and we recommend (Motorola, 90) for technical details.

4.2. Interrupt Unit (ITU)

Throughout the UTCSU specification we encountered several dedicated interrupts to announce certain conditions asynchronously. We grouped all possible interrupt sources in three categories, resulting in the following interrupt lines:

- Interrupt INT-T indicates events relevant to the clock synchronization algorithm as specified in SPECS 12 (duty-timers) and SPECS 5 (accuracy overruns).
- Interrupt INT-N indicates external events as specified in SPECS 10 (stamping leaving/arriving CSPs) and SPECS 11 (1PPS pulses from GPS-receivers).
- Interrupt INT-A indicates application-related events as demanded in SPECS 13 as well as events for chip testing purposes.

The ITU contains the logic for generating these interrupts. Any interrupt source can be individually enabled/disabled via configuration registers UTCINTEN1/2, and pending interrupts are reflected by registers UTCSTAT1/2. For further details about handling these registers turn to Section 5.2.

4.3. Local Time Unit (LTU)

According to SPECS 6 and 8 local time has to be maintained by an adder-based clock. This boils down to set up a 91 bit adder capable of carrying out an addition at each oscillator tick. Due to the advanced space and time requirements, much effort has been invested to devise an optimal adder architecture, cf. (Horauer and Loy, 95).

Figure 6 shows schematically the architecture of the LTU with the 91 bit adder NTPADDER right in the center. The addend is realized as a positive feedback of the previous adder output held by the $(+31, -59)$-register NTPTIME, which in turn provides local time. Initialization takes place via multiplexer NTPMUX and preload registers MSSET, TSSET and USSET.

The augend is supplied by the $(-20, -51)$-register STEPPUREACT resp. STEPAMORT during pure resp. amortization phase. Their granularity is extended by register STEPLOW, which holds a fixed value in the $(-52, -59)$-range to account for non-binary oscillator frequencies. Muliplexer AMORTMUX is in charge of switching between pure and amortizing clock-step values as demanded by SPECS 8. As long as the programmable 32 bit counter AMORTTIMER is running, the augend originates from STEPAMORT, otherwise from STEPPUREACT. The counter can be activated by a duty timer or immediately by writing a dedicated register address. Preload register STEPPURE is necessary to hold the clock-step value for the following pure phase, which starts when the counter expires. See Figure 4 for a better understanding of the transitions between the two phases.

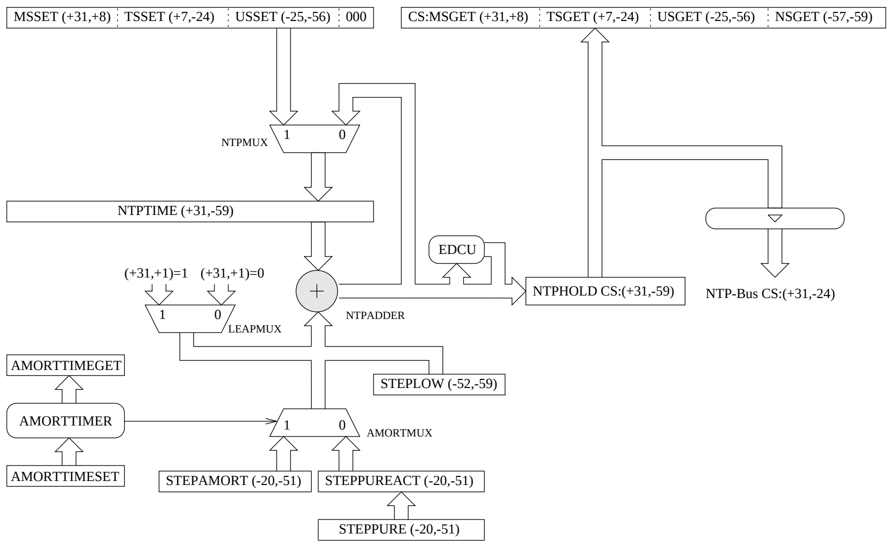


Figure 6. Local Time Unit

According to SPECS 3, the adder-based clock has to cope with leap seconds as well. Basically, the leap second correction hardware consists of multiplexer LEAPMUX, which affects the upper 32 bits of the augend in such a way that either one standard second is added or subtracted just as required. An additional duty-timer can be programmed to initiate these time corrections.

An 8 bit *Checksum* (CS) is computed for the $(+31, -24)$-range of register NTPTIME by the *Error Detection and Correction Unit* (EDCU) as stipulated in SPECS 12, using a modified Hamming Code of distance 4. The CS bits together with the NTPADDER output are buffered on each rising edge of the oscillator pulse in an intermediate register NTPHOLD. This introduces a delay of one oscillator tick between computation and sampling, but the clock synchronization algorithm is not affected, since adjustments are only applied in a relative fashion. Therefore, we can operate the local clock one tick ahead of actual time internally, so that correct time is perceived after buffering.

Adhering to byte-orientation, the output of register NTPHOLD is decomposed into four portions. The 24 bit *Marcostamp*-range $(+31, +8)$ together with the 32 bit *Timestamp*-range $(+7, -24)$ including the 8 CS bits constitute the internal 64 bit NTP-Bus (see Figure 5) from where all timestamps are obtained. The remaining 32 bit *Microstamp*-range $(-25, -56)$ and the 3 bit *Nanostamp*-range $(-57, -59)$ are internally used for rate corrections. Nevertheless, all portions are atomically accessible through holding registers CS:MSGET, TSGET, USGET and NSGET for testing purposes.

4.4. Accuracy Unit (ACU)

The ACU maintains $\alpha^+(t)$ and $\alpha^-(t)$ as demonstrated in Figure 4, forming the local accuracy interval $A(t)$ around local clock $C(t)$. Specs 7 and 9 demand a dedicated ABC for each $\alpha^+(t)$ and $\alpha^-(t)$, but due to symmetry, it suffices to discuss only one of them.

Apart from a few modifications and supplements, the basic structure of this adder-based clock is similar to the one inside the LTU, see Figure 7. In the center we find a 45 bit adder ACCADDER, which is able to deal with negative numbers. Unlike the LTU-adder, however, it stays at its upper limit and triggers an interrupt INT-T instead of wrapping around in case of overrun.

Register ALPHA is situated in the feedback loop of the addend, so that it actually provides the current accuracy value $\alpha^+(t)$ or $\alpha^-(t)$, respectively. Initialization with the preloaded value in the $\pm(-8, -38)$-register ALPHASET happens via multiplexer ALPHAMUX, whereby the $(-39, -51)$-part is fixed to all ones upon initialization.

In analogy to the LTU, multiplexer LAMBDAMUX selects the augend from $\pm(-37, -51)$-register LAMBDAPUREACT resp. LAMBDAAMORT during the pure resp. amortization phase as specified in Specs 9. Both registers hold 16 bit signed deterioration values that are internally sign-extended to 45 bits via multiplexer LAMBDASMUX. Executing continuous amortization for local time controlled by counter AMORTTIMER, switching between deterioration values takes place simultaneously with the clock-step registers inside the LTU. No deterioration takes place right at the transition from pure to amortization phase, but rather register ALPHA is relatively adjusted via multiplexer STATEMUX by an offset value previously written to register STATESET. Again, Figure 4 helps to clarify the subtleties at this pivotal transition point.

During the amortization phase, the content of accuracy register ALPHA can become negative. Following Specs 9, multiplexer AMUX converts such negative values into zero controlled by the associated sign bit. The raw accuracy values are nevertheless accessible via registers ALPHAGET and NALPHAGET, which are sampled simultaneously when LTU register TSGET is read.

Both negative and positive accuracy are exported in the $(-8, -23)$-range, constituting the UTCSU internal A-Bus+ resp. A-Bus-. Together they are referred as the 32 bit broad A-Bus (see Figure 5), from where all accuracystamps are taken. Note

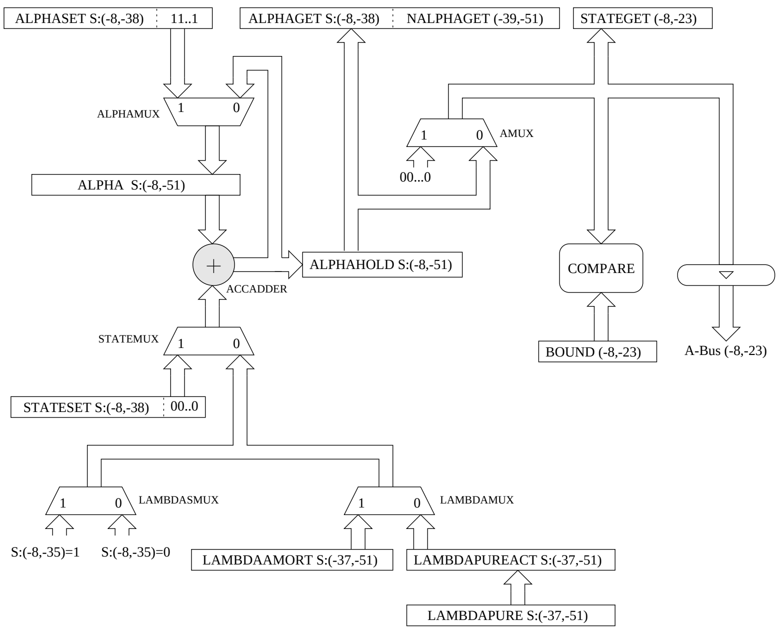


Figure 7. Accuracy Unit (generic)

that the output of adder ACCADDER is buffered at each rising edge of the oscillator pulse into register ALPHAHOLD, thus the accuracies on the A-Bus suffer from the same latency of one tick as local time on the NTP-Bus; recall our comments in Section 4.3.

Implementing Specs 5 results in a 16 bit comparator block COMPARE that permanently checks the A-Bus against a $(-8, -23)$-register BOUND. If the current value on the A-Bus happens to be greater or equal to BOUND, an interrupt INT-T will be raised.

4.5. Synchronization Subnet Unit (SSU)

Rule Specs 10 demands one SSU for each SSN attachment, where each unit comprises a set of registers to sample the NTP- and A-Bus on the occurrence of packet reception/transmission events mediated by the embedding hardware as explained in Section 2.2. The latter is also responsible to map transparently the corresponding SSU registers, see Section 5.3 for an enumeration, into a certain portion of the transmit/receive buffer of the ajoined COMCO.

Moreover, SPECS 12 introduces two duty timers for each SSU, that perform a comparison of the NTP-Bus against a 48 bit preload value. An interrupt INT-T is raised whenever the NTP-Bus becomes greater or equal to the preloaded value, provided that this interrupt source is enabled.

4.6. GPS Unit (GPU)

External clock synchronization requires access to an external time source. As imposed by SPECS 11, we use GPS receivers to inject UTC or GPS-Time. Figure 8 shows an exemplary interface structure to couple one receiver to a Primary-node. For redundancy purposes, up to three receivers can be attached to a single UTCSU.

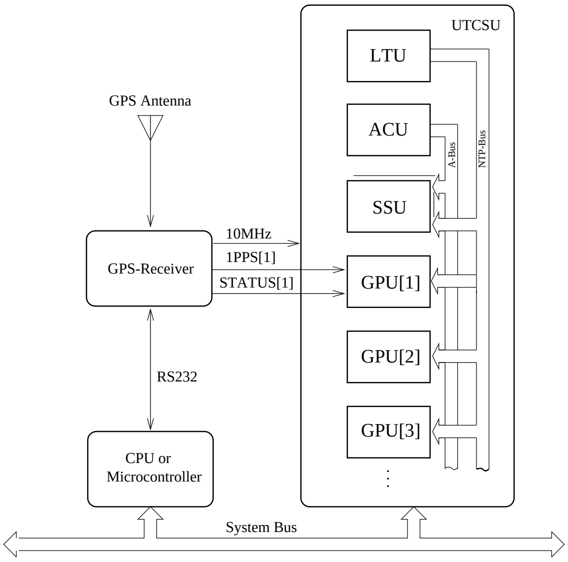


Figure 8. GPS-receiver coupling

Both lines 1PPS[1] and STATUS[1] of the receiver are directly connected to the GPU[1] in order to trigger a timestamp and sample the receiver status on the occurrence of an active 1PPS. The interior of a GPU is made up of our usual sample registers, see Section 5.3 for a listing, whereas the most significant CS bit is replaced by the status bit of the receiver. To adapt the GPU for different GPS receivers, the active 1PPS edge can be programmed to be either the rising or the falling one. Advanced GPS timing-receivers offer an additional frequency output with high stability characteristics, e.g. line 10Mhz in our example, which is most suitable for being used as the oscillator input frequency pacing the ABCs.

Events signalled by the 1PPS are conceptually analogous to the arrival of CSPs. However, information identifying the pulse (full seconds relative to UTC) is delivered after the occurrence of the 1PPS, usually via a RS232 interface. The task of preprocessing this data, sending commands to the receiver for configuration purposes, and interpreting other data concerning status, navigation, etc. are not handled by the UTCSU, but rather by the CPU of the embedding hardware.

4.7. Application Unit (APU)

According to Specs 13, the APU provides features to generate interrupts at specific points in time and to record the occurrence time of application-related events. The former is implemented by a duty-timer APPL to generate an INT-A interrupt, similar to one in the SSU as explained in Section 4.5. The second feature is more expensive, since tracing events requires atomic sampling of both NTP- and A-Bus. The APU accommodates nine register sets (see Section 5.3) to stamp pulses on the polarity programmable input lines APP[1..9] with time and accuracy simultaneously. Moreover, to ease higher-level recovery of applications in case of a clock fault, CPU read accesses of these registers are memorized by a certain reference status bit.

4.8. Network Time Interface Unit (NTU)

Additional application timing support can be provided externally by means of the NTPA-Bus as required by Specs 13. The UTCSU produces 64 bit time information onto the NTP-Bus and 32 bit accuracy information onto the A-Bus at each oscillator tick; given an operating frequency of 2^{24} Hz, this amounts to 192 MByte/s. Indeed, exporting such a stream of data turns out to be a challenging matter. To reduce the pin count of our chip, the NTU achieves this performance by pushing out data via the 48 bit wide multiplexed NTPA-Bus driven by both edges of the oscillator pulse.

4.9. Snapshot Unit (SNU)

The SNU comprises three debugging/test services called software snapshot, hardware snapshot and synchronous operation. As motivated by Specs 16, snapshots are means to sample the current internal state of the UTCSU.

A software snapshot can be triggered by writing a dedicated SNU register address or on expiration of a duty-timer. This entails a simultaneous sampling of the complete LTU register NTPHOLD into the already introduced registers MSGET, TSGET, USGET, and NSGET, as well as sampling ACU registers ALPHAHOLD± into registers ALPHAGET±, NALPHAGET± and STATEGET±. Further reads without latching semantics deliver the before sampled data.

A hardware snapshot is triggered by an external pulse on the polarity programmable line HWSNAP. Due to the asynchronous nature of such an event, it merely samples both NTP- and A-Bus into appropriate SNU registers, enumerated in Section 5.4.

Finally, input line SYNCRUN gives the UTCSU a "GO"-functionality. In other words, all three ABCs can be started simultaneously at a specific point in time triggered by the test environment.

4.10. Built-In Test Unit (BTU)

In principle, the sequence of all time/accuracy values generated inside the UTCSU could be computed externally as well. Unfortunately, a continuous check is prohibited by the tremendous throughput, as already pointed out in Section 4.8.

However, SPECS 15 specifies two compression methods over a certain set of time/accuracy-stamps, that allow less frequent data exchanges between the UTCSU and external verification devices. As a result, the UTCSU is endowed with a BTU that computes signatures and blocksums for both NTP- and A-Bus. Note that these two methods can be used independently of each other.

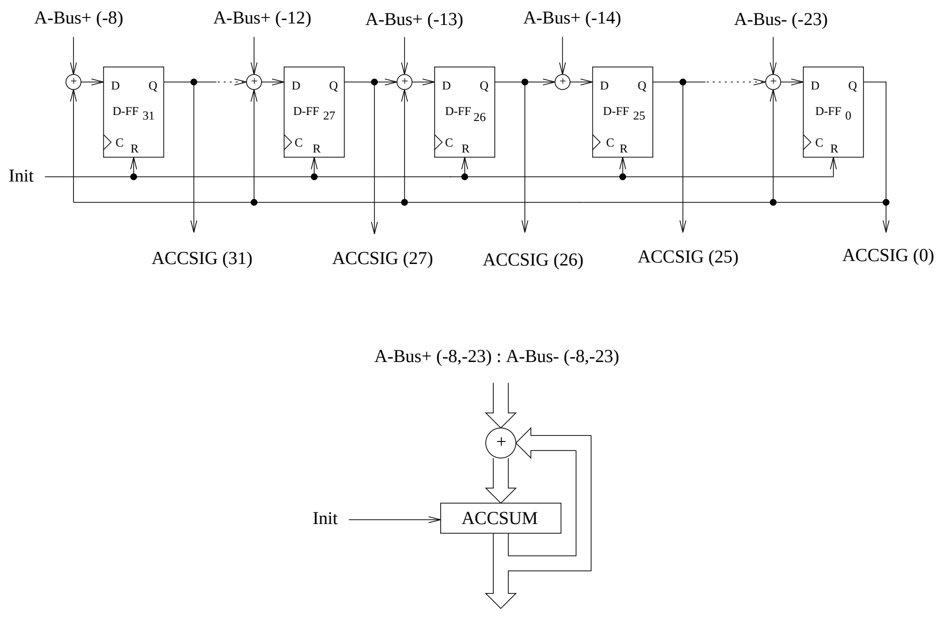


Figure 9. Compressing accuracystamps with signatures and blocksums

The upper half of Figure 9 shows the logic to compute a signature for the A-Bus, where the accuracy values are fed from above into a *linear feedback shift register* (LFSR), cf. (Yarmolik and Kachan, 93). These registers are implementing the evaluation of a polynomial $P(x)$ with coefficients out of $\{0,1\}$. Each power x^k is reflected by a D-Flip-Flop (D-FF) fed from the x^{k-1} stage exored with the corresponding bit from the input data. Powers with non-zero coefficient obtain an additional feedback from the D-FF$_0$ output. In particular, a polynomial of degree 56, namely $P_{\mathrm{NTP-Bus}}(x) = x^{56} + x^{22} + x^{21} + x^1 + 1$, compresses timestamps and a polynomial of degree 32, namely $P_{\mathrm{A-Bus}}(x) = x^{32} + x^{28} + x^{27} + x^1 + 1$, takes care of accuracystamps. The lower half of Figure 9 shows an adder feedbacked by the 32 bit register ACCSUM to compute blocksums over the A-Bus.

Only a complete sequence of time/accuracy-stamps is useful to be processed in the LFSR or adder block, hence specific start/stop-points have to be defined. Transitions between pure and amortization phases are particularly suitable for that purpose. Section 5.4 remarks on programming issues about these features.

5. Programming Model

We wrap up our UTCSU picture by presenting a synoptic view from the programming standpoint. Organized in subjects concerning clocks, interrupts, sampling and testing, we inspect essential operations tagged with programming guidelines. Tables will summarize related UTCSU elements by showing their type and providing a brief description. Three types can be distinguished: An internal register indicated by the corresponding NTP-range or width, a pseudo register to trigger certain actions, or a pin for external events. A forthcoming datasheet will provide a complete programming description including register maps, timing diagrams and electrical characteristics.

5.1. Clock Management

The management of ABCs can be separated in issues concerning initialization and adjustments. Initializing the UTCSU turns out to be a delicate matter, since several stages have to be passed through, namely

- set all registers to a default value via a hardware reset,
- set augend (clock rate and accuracy deterioration) of ABCs, and
- set addend (clock and accuracy state) of ABCs.

All UTCSU registers are set to a default value whenever the chip is powered up or when an external hardware reset occurs. In particular, LTU-register NTPTIME becomes zero, ACU-registers ALPHA± all ones, and the various registers holding the corresponding augend are cleared to prevent a progression of the adder-based clocks.

The second stage targets the active augends of the adder-based clocks belonging to the pure phase, i.e. LTU-register STEPPUREACT and ACU-registers LAMBDAPUREACT± holding the pure clock-step resp. pure accuracy deterioration values. As shown in Figures 6 and 7, these registers cannot be initialized directly. Still, we get a handle on them by first writing their preload registers STEPPURE resp. LAMBDAPURE± and by subsequently commencing a short period of continuous amortization. Only register STEPLOW can be set directly to account for a non-binary oscillator frequency. Table 2 summarizes programming elements relevant at this stage.

Table 2. UTCSU elements for rate/deterioration initialization

element	type	description
STEPPURE	$(-20,-51)$	clock-step for pure phase
STEPLOW	$(-52,-59)$	clock-step for non-binary frequency adaption
LAMBDAPURE+	$\pm(-37,-51)$	upper accuracy deterioration for pure phase
LAMBDAPURE-	$\pm(-37,-51)$	lower accuracy deterioration for pure phase

In the third stage, the addends of the adder-based clocks, i.e. LTU-register NTPTIME and ACU-registers ALPHA±, are initialized. Since they reflect the state of local time/accuracy, their value has to be set appropriately to correspond to the time of setting them. As a consequence, preload registers must be provided to hold the initialization values in advance that are transferred atomically to NTPTIME and ALPHA± upon a suitable event. Local time is initialized with the aid of three 32 bit registers MSSET, TSSET and USSET. When no accuracy information is at hand for initialization, we can stick to the maximum introduced during hardware reset, which serves literally as infinity. Otherwise, preload register ALPHASET supplies both ALPHA+ and ALPHA-, launching a symmetrical accuracy interval. The event of actually transferring preloaded values is generated by writing pseudo-registers NTPSET and ALPHASET, cf. Table 3. To ease simultaneous starting of distributed UTCSUs during the testing phase or in redundant configurations, the transfer event can also be a pulse on the external line SYNCRUN.

Table 3. UTCSU elements for time/accuracy initialization

element	type	description
MSSET	$(+31,+8)$	macrostamp portion of clock state
TSSET	$(+7,-24)$	timestamp portion of clock state
USSET	$(-25,-56)$	mircostamp portion of clock state
ALPHASET	$\pm(-8,-38)$	initial upper/lower accuracy
NTPSET	pseudo	sets clock state and upper/lower accuracy on write
ALPHAPNSET	pseudo	sets only upper/lower accuracy on write
SYNCRUN	pin	sets clock state and upper/lower accuracy on external pulse

Once initialized, clocks need to be adjusted periodically in order to maintain internal/external synchronization. Adjustments values for local time and accuracies are handed over at each pure to amortization phase transition as already explained in Section 3.5. The UTCSU enforces these adjustments autonomously, which renders programming rather simple. In fact, the elements of Table 4 need to be computed before the next amortization phase, encompassing values for clock-step register STEPAMORT, for deterioration registers LAMBDAAMORT±, and for accuracy adjustment registers STATESET±. Additionally, registers STEPPURE and LAMBDAPURE± need to be preloaded for the subsequent pure phase. The duration in oscillator ticks of the amortization phase can be set via counter AMORTTIMESET, whereas the start can be either triggered by writing pseudo register STARTAMORT or by activating duty-timer DUTYB[1]. Managing DTs is covered in Section 5.2.

Table 4. UTCSU elements for time/accuracy adjustments

element	type	description
STEPAMORT	$(-20, -51)$	clock-step for amortization phase
LAMBDAAMORT+	$\pm(-37, -51)$	upper accuracy deterioration for amortization phase
LAMBDAAMORT-	$\pm(-37, -51)$	lower accuracy deterioration for amortization phase
STATESET+	$\pm(-8, -38)$	upper accuracy adjustment in two's complement
STATESET-	$\pm(-8, -38)$	lower accuracy adjustment in two's complement
AMORTTIMESET	32 bit counter	duration of amortization phase in oscillator ticks
STARTAMORT	pseudo	starts amortization phase on write
DUTYB[1]	$(+31, -16)$	duty-timer to start amortization phase

5.2. Interrupt Management

There are as much as 64 interrupt sources within the UTCSU, which are statically mapped to three dedicated interrupt lines introduced in Section 4.2. Interrupt processing in general works as follows, cf. Table 5: To initialize an interrupt source for further interrupts, we have to clear it by setting the appropriate bit in register UTCINTCLEAR1 or UTCINTCLEAR2, and to enable it by setting the appropriate bit in register UTCINTEN1 or UTCINTEN2. A pending interrupt is mirrored by a status bit in registers UTCSTAT1 or UTCSTAT2, which can be polled by the *interrupt service routine* (ISR) to identify the originating source of the interrupt. Before leaving the ISR, the interrupt source must be cleared as described above.

Many interrupts are associated with external events, like leaving/arriving CSPs, 1PPS pulses from GPS timing-receivers, or occurrence of application-oriented events. Their handling will be described in Section 5.3. Here we proceed with interrupts that are internally caused by the UTCSU, in particular by DTs and accuracy overruns. Of course, many other exceptional UTCSU conditions are also announced by INT-T interrupts, like an overrun of NTPTIME somewhere in year 2036 or clocking failures.

Table 5. UTCSU elements for interrupt handling

element	type	description
UTCINTCLEAR1/2	32 bit each	clears pending interrupts
UTCINTEN1/2	32 bit each	enables/disables interrupts
UTCSTAT1/2	32 bit each	reflects the status of the interrupt condition

Various duty-timers are used for generating interrupts or for triggering certain actions, see Table 6 for a complete list. Each DT spans 48 bits, organized in a 32 bit high-part $(+31, 0)$ and in a 16 bit low-part $(-1, -16)$ including an enable bit E to arm/disarm its operation. If armed, the content of each duty-timer is permanently compared against the current time on the NTP-Bus and corresponding status bits in registers UTCSTAT1 and UTCSTAT2 indicate the condition "less" (duty-timer $<$ NTP-Bus) or "greater-or-equal" (duty-timer $\geq$ NTP-Bus). In case of a transition from "less" to "greater-or-equal", a dedicated interrupt INT-T will be generated. Note that activating a DT with an old value w.r.t. the current time given on the NTP-Bus also entails an interrupt.

Table 6. UTCSU elements for duty-timers

element	type	description
DUTYA[1..6]-HIGH	$(+31, 0)$	SSU duty-timers A[1..6]
DUTYA[1..6]-LOW	E:$(-1, -16)$	– " –
DUTYB[1]-HIGH	$(+31, 0)$	duty-timer to start amortization
DUTYB[1]-LOW	E:$(-1, -16)$	– " –
DUTYB[2]-HIGH	$(+31, 0)$	duty-timer to terminate CSP receptions
DUTYB[2]-LOW	E:$(-1, -16)$	– " –
DUTYB[3]-HIGH	$(+31, 0)$	duty-timer to insert/delete leap seconds
DUTYB[3]-LOW	E:$(-1, -16)$	– " –
DUTYB[4..6]-HIGH	$(+31, 0)$	auxiliary SSU duty-timers B[4..6]
DUTYB[4..6]-LOW	E:$(-1, -16)$	– " –
SW-HIGH	$(+31, 0)$	SNU duty-timer for software snapshot
SW-LOW	E:$(-1, -16)$	– " –
APPL-HIGH	$(+31, 0)$	APU duty-timer for applications
APPL-LOW	E:$(-1, -16)$	– " –

Without external scaling, the capacity to hold accuracies in registers ALPHA$\pm$ is limited to 7.81 ms each, while overflows cause an interrupt INT-T. A more selective supervision of maximum accuracy can be programmed with the help of registers BOUND$\pm$, also displayed in Table 7. Similar to duty-timers, whenever the condition "below" (A-Bus$\pm$ $<$ BOUND$\pm$) tips to "above-or-equal" (A-Bus$\pm$ $\geq$ BOUND$\pm$), an interrupt INT-T will be generated. Setting registers BOUND$\pm$ to the maximum disables this feature.

Table 7. UTCSU elements to bound accuracies

element	type	description
BOUND+	$(-8,-23)$	bound for upper accuracy
BOUND-	$(-8,-23)$	bound for lower accuracy

5.3. Sampling Management

Sampling activities can stem from software primitives or from external events. The first ones are also known as *atomic clock readings*, providing allied portions of local time/accuracies only. Whenever register TSGET is read by software, it entails a simultaneous latching of full-scale local time and accuracy in certain UTCSU-registers listed in Tables 8 and 9. Later on the sampled values can be read externally at register addresses without latching semantics.

Table 8. UTCSU elements for sampling local time

element	type	description
CS:MSGET	CS:$(+31,+8)$	macrostamp portion of clock state together with checksum
TSGET	$(+7,-24)$	timestamp portion of clock state (causes atomic sampling)
USGET	$(-25,-56)$	microstamp portion of clock state
NSGET	$(-57,-59)$	nanostamp portion of clock state

Table 9. UTCSU elements for sampling local accuracies

element	type	description
ALPHAGET+	$\pm(-8,-38)$	high portion of signed upper accuracy
ALPHAGET-	$\pm(-8,-38)$	high portion of signed lower accuracy
NALPHAGET+	$(-39,-51)$	low portion of upper accuracy
NALPHAGET-	$(-39,-51)$	low portion of lower accuracy
STATEGET+	$(-8,-23)$	upper accuracy on A-Bus+
STATEGET-	$(-8,-23)$	lower accuracy on A-Bus-

Crucial for UTCSU operation are features to stamp external events with time/accuracy values. Apart from sampling, a INT-N interrupt is raised upon occurrence if enabled at all. Table 10 recapitulates all elements relevant for this functionality.

Registers UTCCONF1 and UTCCONF2 determine the polarity on which input pulses cause sampling. Besides that, they contain bits to control leap-second insertion/deletion and to configure self-test features, cf. Section 5.4.

In a straightforward way, pulses on pins TRANSMIT[1..6] resp. RECEIVE[1..6] inform the UTCSU about leaving resp. arriving CSPs. The various sample registers can be found in the second and third block of Table 10.

Similarly, pulses on pins 1PPS[1..3] resp. STATUS[1..3] inform the UTCSU about an active on-time pulse resp. status of a connected GPS timing-receiver. The fourth block of Table 10 indicates the corresponding sample registers.

Finally, pins APP[1..9] are available for time/accuracy-stamping of application events. The bottom block of Table 10 show the associated sample registers including a pseudo one called APPCLEAR. Every read access of registers MSAPP[1..9], TSAPP[1..9] or ACCAPP[1..9] sets a status bit in register UTCSTAT1, which can be cleared by writing APPCLEAR.

Table 10. UTCSU elements for stamping external events

element	type	description
UTCCONF1/2	32 bit each	configuration bits, e.g. polarity of input pulses
TRANSMIT[1..6]	pins	indicates a CSP transmission
MSXMT[1..6]	CS:$(+31, +8)$	sample of upper NTP-Bus portion
TSXMT[1..6]	$(+7, -24)$	sample of lower NTP-Bus portion
ACCXMT[1..6]	$(-8, -23)$:$(-8, -23)$	sample of A-Bus
RECEIVE[1..6]	pins	indicates a CSP arrival
MSRCV[1..6]	CS:$(+31, +8)$	sample of upper NTP-Bus portion
TSRCV[1..6]	$(+7, -24)$	sample of lower NTP-Bus portion
1PPS[1..3]	pins	indicates a 1PPS pulse from GPS receivers
STATUS[1..3]	pins	indicates the status of GPS receivers
MSGPS[1..3]	status:CS:$(+31, +8)$	sample of upper NTP-Bus portion with receiver status
TSGPS[1..3]	$(+7, -24)$	sample of lower NTP-Bus portion
APP[1..9]	pins	indicates an external application event
MSAPP[1..9]	CS:$(+31, +8)$	sample of upper NTP-Bus portion
TSAPP[1..9]	$(+7, -24)$	sample of lower NTP-Bus portion
ACCAPP[1..9]	$(-8, -23)$:$(-8, -23)$	sample of A-Bus
APPCLEAR	pseudo	clears the application reference status bit

5.4. Testing Management

Snapshot mechanisms are implemented for test and verification purposes as motivated in Section 3.7. Before using them, certain configuration bits in UTCCONF1 must be set appropriately.

If enabled, a pulse on pin HWSNAP triggers a hardware snapshot, i.e., the NTP-Bus is latched into registers MSSNU and TSSNU, and the A-Bus into ACCSNU. The top block of Table 11 summarizes these elements.

A software snapshot is triggered *directly*, when pseudo register SWSNAP is written, or *programmed* when duty-timer SW fires. All registers listed up in Tables 8 and 9 are affected by activating this mechanism, which provides a comprehensive view of the current state of the whole chip.

Table 11. UTCSU elements for snapshots

element	type	description
HWSNAP	pin	triggers a hardware snapshot
MSSNU	CS:$(+31, +8)$	sample of upper NTP-Bus portion
TSSNU	$(+7, -24)$	sample of lower NTP-Bus portion
ACCSNU	$(-8, -23)$:$(-8, -23)$	sample of A-Bus
SWSNAP	pseudo	triggers a software snapshot
SW	$(+31, -16)$	duty-timer to trigger a software snapshot

The computation of signatures and blocksums for external verification can be started and stopped by a SYNCRUN event, a software snapshot, or at the beginning of an amortization phase. Configuration bits in register UTCCONF1 must be set accordingly to enable the desired operation. Furthermore, these events (except of SYNCRUN) latch the results into appropriate registers as given in Table 12. Usually, the period over which these compression functions are calculated include the pure phase, the length of which is provided by counter PUREPHASE for external verification purposes.

Table 12. UTCSU elements for external verification

element	type	description
MSSIG	32 bit	signature of macrostamp portion of time
TSSIG	32 bit	signature of timestamp portion of time
ACCSIG	32 bit	signature of upper and lower accuracy
MSSUM	32 bit	blocksum of macrostamp portion of time
TSSUM	32 bit	blocksum of timestamp portion of time
ACCSUM	32 bit	blocksum of upper and lower accuracy
PUREPHASE	32 bit counter	duration of pure phase in oscillator ticks

6. Design Methodology

The chip design process evolved in several stages. In the following we document them briefly by pointing out their particular objective and the tools used.

Starting out from the functional specification of (Schossmaier and Schmid, 95), an elaborate process of successive refinement was conducted that eventually ended up in coding and synthesizing all required units, cf. (Loy, 96). Based on this knowledge, we worked out a complete behavioral VHDL description of the UTCSU. At the same time, we coded a simulation model of the node consisting of a basic VHDL model of both CPU and embedding hardware including the COMCO. At the next stage, algorithmic operations were verified to ensure that all required functionalities are present and behave as required. Again, this model was refined and recoded to meet our specification and to make the code ready for synthesis.

To obtain a *gate-level netlist*, we used SYNOPSYS DESIGN COMPILER [4] for synthesis into two foundry libraries, viz AMS [4] 0.8 μm and ES2 [4] 0.7 μm standard cell CMOS process. A postprocessing tool was used to verify the resulting code against the behavioral simulation. Table 13 summarizes some technical chip data derived at this stage. Afterwards, SYNOPSYS TEST COMPILER [4] inserted *full scan path logic* with a multiplexed flip-flop style and *boundary scan logic* according to IEEE 1149.1. CADENCE DFWII [4] back-end tools finalized the design with *place&route* for ES2. Timing requirements and technology rules were cross-checked by parasitic extraction.

Table 13. UTCSU chip data for ES2 0.7μm

area (with pre-estimated routing) [mm^2]	100
max. operating frequency [MHz]	25
equivalent gates	66.500

7. Conclusions

The obstacles on the road to implement a highly accurate/precise time service in the real-time systems domain emerge from two difficulties. First, a clock circuitry is necessary for maintaining local time/accuracy with sufficient state graininess and rate dynamics. Second, recording and generating events in the proximity of the clock has to be done with minimal uncertainty. Only profound and well designed hardware allows to meet these requirements. Hence, a major result of our research is the development of a custom VLSI chip that incorporates adder-based clocks and sophisticated event handling facilities. In this paper we gave the specification of the UTCSU in terms of 16 rules focusing on the key implementation parts, and presented a basic programming model.

Future efforts are devoted to redesign of this ASIC to advance it beyond the prototype version. Especially, a higher operating frequency, a reduced chip size, and more application-specific features are desireable. It is planned to bring out a whole familiy of UTCSUs varying in performance, pin-count, package size, power consumption and of course functionality. We also have in mind to tailor it — in conjunction with the embedding hardware — towards emerging applications in computer science relying on a time service, like multimedia or mobil computing.

Appendix

Table A.1 gives the time equivalence of NTP-bit numbers divided in columns for the integer part $(+31, 0)$, the fractional part $(-1, -32)$ and the ultrafractional part $(-33, -64)$. Note that "n" stands for nano (10^{-9}), "p" pico (10^{-12}), "f" for femto (10^{-15}), "a" for atto (10^{-18}) and "z" for zepto (10^{-21}).

Table A.1. NTP-bit numbers and their time equivalence

integer part		fractional part		ultrafractional part	
bit#	time equivalence	bit#	time equivalence	bit#	time equivalence
+31	68 years	-01	500 ms	-33	116.42 ps
+30	34 years	-02	250 ms	-34	58.21 ps
+29	16 years	-03	125 ms	-35	29.10 ps
+28	8.5 years	-04	62.5 ms	-36	14.56 ps
+27	4.25 years	-05	31.25 ms	-37	7.28 ps
+26	2.13 years	-06	15.62 ms	-38	3.64 ps
+25	1.08 years	-07	7.81 ms	-39	1.82 ps
+24	194.18 days	-08	3.81 ms	-40	909.50 fs
+23	97.09 days	-09	1.90 ms	-41	454.75 fs
+22	48.55 days	-10	976.56 μs	-42	227.38 fs
+21	24.27 days	-11	488.28 μs	-43	113.69 fs
+20	12.17 days	-12	244.14 μs	-44	56.84 fs
+19	6.07 days	-13	122.07 μs	-45	28.42 fs
+18	3.03 days	-14	61.03 μs	-46	14.21 fs
+17	1.52 days	-15	30.51 μs	-47	7.11 fs
+16	18.02 hours	-16	15.25 μs	-48	3.55 fs
+15	9.10 hours	-17	7.62 μs	-49	1.77 fs
+14	4.55 hours	-18	3.81 μs	-50	888.18 as
+13	2.27 hours	-19	1.90 μs	-51	444.09 as
+12	1.13 hours	-20	953.67 ns	-52	222.04 as
+11	34.13 min	-21	476.83 ns	-53	111.02 as
+10	17.07 min	-22	238.41 ns	-54	55.51 as
+09	8.53 min	-23	119.21 ns	-55	27.76 as
+08	4.27 min	-24	59.60 ns	-56	13.88 as
+07	2.13 min	-25	29.80 ns	-57	6.94 as
+06	1.07 min	-26	14.90 ns	-58	3.47 as
+05	32 s	-27	7.45 ns	-59	1.73 as
+04	16 s	-28	3.72 ns	-60	867.36 zs
+03	8 s	-29	1.86 ns	-61	433.68 zs
+02	4 s	-30	931.32 ps	-62	216.84 zs
+01	2 s	-31	465.66 ps	-63	108.42 zs
+00	1 s	-32	232.83 ps	-64	54.21 zs

Notes

1. Although our UTCSU owes much to the CSU of (Kopetz, 89) in general, it will become apparent that almost all features are entirely different in both functionality and implementation.
2. We adapted a FORCE CPU-30 board by constructing a piggy back equipped with the LANCE 82596CA from INTEL. A second generation is under development build on top of standard IP-Modules.
3. Currently we are performing a longterm evaluation experiment to tail out accuracy and availability of several GPS timing-receivers.
4. Synopsys Design Compiler and Synopsys Test Compiler are registrated trademarks of Synopsys, Inc. AMS is a trademark of Austria Micro Systeme Int. ES2 is a tradmark of European Silicon Structures. Cadence DFWII is a trademark of Cadence, Inc.

References

Dana P.H., "Global Positioning System (GPS) Time Dissemination for Real-Time Applications", (this issue), 1996.

Feldmann K. and Solvie M., "Die Uhr im Feldbus", *Elektronik,* Vol. 6, pp. 112–122, 1995.

Fetzer C. and Cristian F., "Integrating External and Internal Clock Synchronization", (this issue), 1996.

Gergeleit M. and Streich H., "Implementing a Distributed High-Resolution Real-Time Clock using the CAN-Bus", Proceedings of the 1^{st} *international CAN-Conference,* September 1994.

Halang W.A. and Wannemacher M., "High Accuracy Concurrent Event Processing in Hard Real-Time Systems", (this issue), 1996.

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

Horauer M. and Loy D., "Adder Synthesis", *Austrochip '95,* Vienna University of Technology, Department of Computer Technology, 1995.

Kopetz H., *Clock Synchronization Unit (CSU) Datasheet,* Research Report 22/89, Vienna University of Technology, November 1989.

Kopetz H. and Ochsenreiter W., "Clock Synchronization in Distributed Real-Time Systems", *IEEE Transactions on Computers,* C-36(8), pp. 933–939, August 1987.

Lamport L., *Synchronizing Time Servers,* Technical Report, Digital Research Center 18, June 1987.

Lichtenecker R., "Terrestrial Time Signal Dissemination", (this issue), 1996.

Liskov B., "Practical uses of synchronized clocks in distributed systems", *Distributed Computation* 6, pp. 211–219, 1993.

Loy D., *GPS-Linked High Accuracy NTP Time Processor for Distributed Fault-Tolerant Real-Time Systems,* PhD thesis, Vienna University of Technology, Department of Computer Technology, April 1996.

Lundelius J. and Lynch N., "An Upper an Lower Bound for Clock Synchronization", *Information and Control,* vol. 62, pp. 190–204, 1984.

Marzullo K., *Maintianing the time in a distributed system. An example of a loosly-coupled distributed service,* PhD thesis, Department of Electrical Engineering, Stanford University, 1984.

Marzullo K. and Owicki S., "Maintaining the Time in a Distributed System", *ACM Operating Systems Review,* vol. 19, number 3, pp. 44–54, July 1985.

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

Mills D.L., *Modelling and Analysis of Computer Network Clocks*, Technical Report 92-5-2, University of Delaware, Electrical Engineering Department, May 1992.

Mills D.L., "Improved Algorithms for Synchronizing Computer Network Clocks", *IEEE Trans. Networks*, pp. 245-254, June 1995.

Motorola, *MC68030 Enhanced 32-Bit Microprocessor User's Manual*, Prentice Hall, 1990.

OSF: *Introduction to OSF DCE,* Englewood Cliffs, NJ: Prentice Hall, 1992.

Ramanathan P., Kandlur D.D. and Shin K.G., "Hardware-Assisted Software Clock Synchronization for Homogeneous Distributed Systems", *IEEE Transactions on Computers*, 39(4), pp. 514–524, April 1990.

Ramanathan P., Shin K.G. and Butler R.W., "Fault-Tolerant Clock Synchronization in Distributed Systems", *IEEE Computer* 23(10), pp. 33–42, 1990.

Schmid U., "Monitoring Distributed Real-Time Systems", *Journal of Real-Time Systems* 7, pp. 33–56, 1994.

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

Schmid U. and Schossmaier K., "Interval-based Clock Synchronization", (this issue), 1996.

Schmuck F. and Christian F., "Continuous Amortization need not affect the Precision of a Clock Synchronization Algorithm", Proceedings of the 9th ACM Symposium on *Principles of Distributed Computing,* pp. 133–143, 1990.

Schossmaier K. and Schmid U., *UTCSU Functional Specification,* Technical Report 183/1-56, Vienna University of Technology, Department of Automation, July 1995.

Schroetter J., *Surviving the ASIC experience,* Prentice Hall, 1992.

Shin K.G. and Ramanathan P., *Transmission Delays in Hardware Clock Synchronization*, IEEE Transactions on Computers 37(11), pp. 1465–1467, November 1988.

Simons B., Lundelius-Welch J. and Lynch N., "An Overview of Clock Synchronization", in Simons B. and Spector A., (eds.): *Fault-Tolerant Distributed Computing*, Springer Lecture Notes on Computer Science 448, pp. 84–96, 1990.

Stankovic J.A., "Misconceptions about Real-Time Computing", *IEEE Computer,* 21(10), pp. 10–19, 1988.

Troxel G.D., *Time Surveying: Clock Synchronization over Packet Networks,* PhD thesis, Massachusetts Institut of Technology, Department of Electrical Engineering and Computer Science, 1994.

Verissimo P., Rodrigues L. and Casimiro A., "CESIUMSPRAY: a Precise and Accurate Global Clock Service for Large-scale Systems", (this issue), 1996.

Yarmolik V.N. and Kachan I.V., *Self-Testing VLSI Design,* Elsevier Science Publisher, 1993.

Yang Z. and Marsland T.A., "Annotated Bibliography on Global States and Times in Distributed Systems", *ACM SIGOPS Operating Systems Review*, pp. 55–72, June 1993.