An ASIC Supporting External Clock Synchronization for Distributed Real-Time Systems *

Klaus Schossmaier

Department of Automation
Vienna University of Technology
A-1040 Vienna, Austria
kmschoss@auto.tuwien.ac.at

Dietmar Loy

Department of Computer Technology
Vienna University of Technology
A-1040 Vienna, Austria
loy@ict.tuwien.ac.at

Abstract

Designing clock synchronization algorithms for distributed real-time systems based on packet networks requires to decide on the amount and functionality of proper hardware support. Targeting a $1\mu s$ precision and similar accuracy measures in case of external UTC supply from GPS-receivers, this paper describes pertinent features of a peripheral device called UTCSU manufactured as an ASIC. The most salient one is the introduction of an adder-based clock, that allows a fine grained rate adjustment, continuous amortization, and maintenance of local accuracy intervals.

Keywords: External clock synchronization, Universal Time Coordinated (UTC), Application Specific Integrated Circuit (ASIC), Very high speed integrated circuit Hardware Description Language (VHDL), Global Positioning System (GPS), Adder-based Clock.

1 Hardware Requirements

A time service of a distributed real-time system comprises a set of nodes each hosting a physical clock, and some communication subsystem to exchange packets between them. Applications require a time service that guarantees a bounded deviation between any pair of (non-faulty) clocks within the system (aka. *precision*), and further a worst case deviation towards UTC (aka. *accuracy*). The challenge is to devise a distributed algorithm satisfying the above two requirements, but coping with clock drifts, packet delivery uncertainties, and various faults. A vast body of research has been developed in this area (see [8] for an overview), nevertheless most solutions have a rather simple choreography in common, i.e., periodic broadcast of packets containing synchronization information, usage of a suitable convergence function, and adjustment of the local clock.

However, in the course of implementing such an algorithm many design decisions have to be made depending on application needs. We focus on a time service that should establish a worst case precision and accuracy (providing that an external reference is available) of $1\mu s$ even in case of faulty system components. An introductionary description of our system model, algorithmic approach taken, and a worst case analysis can be found in [9].

Crucial for the implementation is to identify which parts of the algorithm are being executed in hard- or software. For our purpose, the following features need to be supported by dedicated hardware:

- **Providing a state and rate adjustable local clock.** Essential is the provision of a physical clock, which state resp. rate can be controlled in an absolute resp. relative fashion. Notice that any additional computation is undesirable when obtaining local time.

- **Maintaining local accuracy.** Dealing with the accuracy requirement is done by maintaining an accuracy interval relative to the local clock, that is guaranteed to contain UTC. Hardware support is necessary to represent this dynamic quantity.

- **Timestamping packets.** The uncertainty of packet delivery time has a major impact on the synchronization quality. Hence exact timestamping of both packet transmission and arrival calls for adequate hardware support.

- **Interfacing GPS-receivers.** We arrived at the decision to use GPS technology as UTC supply, due to its world-wide accessibility, high accuracy, and meanwhile affordable receivers.

- **Supporting application timers and event timestamping.** To serve applications, there should be means to trigger actions at programmable points in time, and capabilities to label events with timestamps.

*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

Having unfold hardware requirements of clock synchronization algorithms for distributed real-time systems, arguments concerning modularity, flexibility, and performance, influenced us to realize the demanded features as an ASIC, running under the acronym *Universal Time Coordinated Synchronization Unit* (UTCSU). Notice that the actual algorithm has to be executed at some general purpose CPU, which works in close ties with our ASIC.

2 Functional Units

This section addresses design questions and engineering aspects of the UTCSU, resulting in a description of our peripheral device. A complete functional specification can be found in [11] and details about the chip design in [4]. Figure 1 depicts a simplified block diagram, which lays out all functional units along with their interconnections.

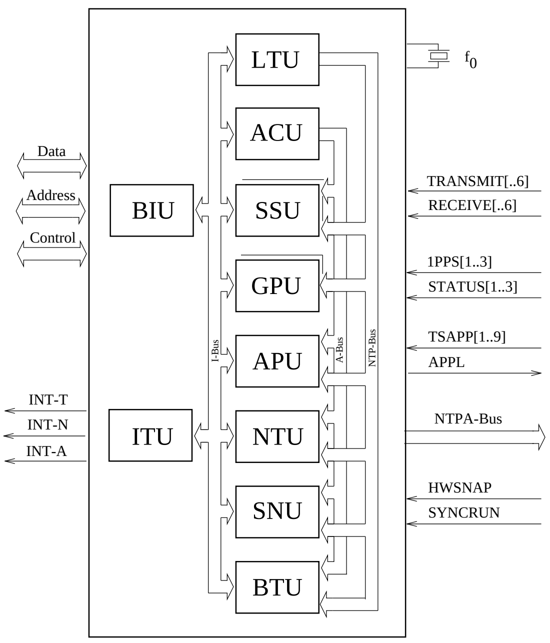

Figure 1: UTCSU Block Diagram

2.1 Bus Interface Unit (BIU)

The ASIC should be applicable to a wide range of existing architectures, in particular data bus widths of 8, 16, or 32 bits, big/little endian byte ordering, different bus access times and interrupt schemes need to be considered. The BIU is responsible to interface the embedding architecture to the internal 32 bit broad I-Bus that interconnects all other UTCSU units.

2.2 Local Time Unit (LTU)

Local time is going to be maintained by a physical clock with sufficient small granularity to track UTC. An ordinary digital clock consists of a counter driven by an oscillator, whose specified frequency f_0 determines the granularity of the clock. For example, encoding UTC with the 64 bit NTP-time format (see [6]), where the upper 32 bits are interpreted as standard seconds relative to UTC and the lower 32 bits give the corresponding fractional part, enforces a binary nominal frequency $f_0 = 2^g$. Deliberate compensation of frequency drifts requires to manipulate ticks from the oscillator, which turns out to be a hard problem. One way is to vary the frequency of the oscillator itself, but only a Voltage Controlled Oscillator (VCO) allows to do that, cf. [6]. Another way is to run the oscillator at a multiple of the nominal frequency, suppressing or inserting pulses as required, cf. [3]. The drawback of this technique is not only a higher bandwidth, but also insufficient correction dynamics.

We propose an alternative digital clock consisting of an oscillator driving an *adder*, which adds a particular amount (clock-step) at each oscillator tick to a register holding local time. A relative rate change can be easily achieved by varying this amount, which goes in effect almost instantly and retains linearity. In such an arrangement the notion of granularity needs to be refined in the following way: *Clock granularity* G_c refers to the smallest time unit representable by the register for local time, whereas *time granularity* $G_t = 1/f_0$ indicates the time period between updates of it. In case of a simple counter-based clock $G_s = G_t$, and gets lumped into granularity. The key to our approach is to use a much better clock granularity than time granularity ($G_c < G_t$), which allows to accumulate minute time portions in order to account for frequency drifts.

An architectural outline of the *adder-based clock* inside the LTU can be found in Figure 2. Extending the NTP-format on the fractional side, the 91 bit register CLOCK holds local time with $G_c = 2^{-59}$ s. Adhering to byte-orientation, register CLOCK gets decomposed in three portions: The 24 bit MACROSTAMP portion together with the adjacent 32 bit TIMESTAMP portion constitute the internal NTP-Bus (see Figure 1), which will be the source for all timestamps. The remaining 35 bit MICROSTAMP portion is solely used for correction purposes administered by register STEP. Supporting a minimal frequency of $2^{20} \approx 1$ MHz requires to make register STEP 40 bits wide, where the upper 32 bits are supposed to be set at each synchronization instant appropriately (see Section 2.4). The lower 8 bits (STEPLOW) are set to a fixed value, in order to reduce the influence of truncation errors for the rate adjustability in case of a non-binary oscillator frequency $f_0 \neq 2^g$. Therefore the effective clock granularity G'_c becomes 2^{-51} s, which defines the smallest unit of interference at each tick as $R_c = G'_c/G_t$, called *correction granularity*. At a nominal frequency of $2^{23} \approx 8.4$ MHz, $R_c = 0.37 \cdot 10^{-8}$, which is equivalent to $1\mu s$ within 270.4 s.

The expense of this clock design is twofold: More register space is necessary to accommodate the enhanced clock granularity, and a 91 bit addition needs to take place at each oscillator tick. To achieve the necessary performance, our UTCSU embodies a *spanning tree carry look ahead adder*, cf. [2].

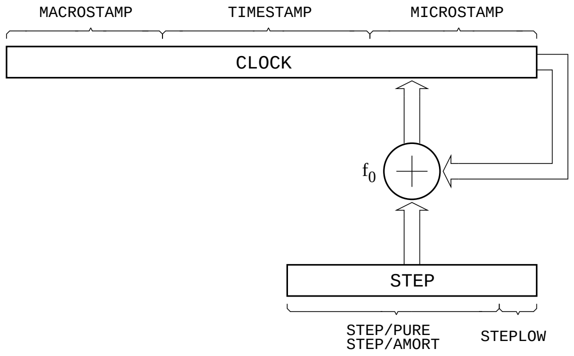

Figure 2: Adder-based Clock

Another item to consider is the non-chronoscopic nature of UTC, because leap seconds need to be inserted or deleted at predefined points in time issued by the Bureau International de l'Heure (BIH). An adder-based clock can easily be modified to deal with leap seconds by supplementing it with a timer, that indicates a pending leap second duty and affects the addition accordingly, i.e., adding or subtracting an extra second. Obviously, in case of a chronoscopic time standard, such as *Temps Atomique International* (TAI) or *GPS-Time*, this feature can be omitted.

2.3 Accuracy Unit (ACU)

Meeting the accuracy requirement means that local time has to follow UTC as close as possible. Unfortunately, UTC is neither directly observable nor permanently accessible, therefore 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 $t \in A(t)\ \forall t \geq t_0$. It should be pointed out, that the OSF/DCE time model is also built upon this notion of time, cf. [7].

Our UTCSU is tailored to the interval-based *clock validation technique* proposed by [9], which relies on accuracy intervals. These are maintained locally at each node meant relative to the local clock $C(t)$, resulting in an *upper accuracy* $\alpha^+(t)$ and a *lower accuracy* $\alpha^-(t)$ such that $A(t) = [C(t) - \alpha^-(t), C(t) + \alpha^+(t)]$. The reason to make accuracies dependable from time is to enlarge them adequately, in order to account for a maximum oscillator drift, hence sustaining UTC inclusion. This process of *deterioration* would go on perpetually, however, periodic resynchronization instants aim to shrink $A(t)$ by virtue of a clock synchronization algorithm and optional UTC supply from GPS-receivers.

The adder-based approach is well suited for maintaining α^+ and α^-. Each accuracy quantity can be tracked by an adder-based clock similar to Figure 2, where ALPHA$\pm$ plays the role of the accumulating register and LAMBDA$\pm$ holds the update for each tick. In accordance with the above arranged effective clock granularity G_c' of 2^{-51} s, registers ALPHA$\pm$ span 45 bits including a sign bit, which can hold accuracies up to 7.8 ms. In analogy to the LTU, the full length of ALPHA$\pm$ is not required for accuracystamping purposes, therefore only the upper 16 bits are in use, forming the internal 32 bit A-Bus (see Figure 1). Registers LAMBDA$\pm$ are 16 bits long plus a sign bit, which allows a maximum deterioration of 2^{-36} s/tick or approximately 122 μs/s at $f_0 = 2^{23}$ Hz. The necessity of a sign bits will be clarified in Section 2.4, and the influence of a non-binary frequency is neglected here.

There are two more features to introduce. The 16 bit registers BOUND$\pm$ guard the values of ALPHA$\pm$, and in case of exceeding them an interrupt will be raised. Updating ALPHA$\pm$ at a resynchronization instant has to be performed in a relative fashion, by affecting them with appropriate values held in the 16 bit registers STATE$\pm$.

2.4 Continuous Amortization

Periodically the clock synchronization algorithm becomes active and computes adjustments for registers CLOCK, STEP, ALPHA$\pm$ and LAMBDA$\pm$ in order to meet both precision and accuracy requirement. Enforcing these adjustments deserves special attention, since applications dictate certain properties. Clock values need to be monotonic and continuous excluding discrete leaps for- or backwards. A technique called *continuous amortization* (see [10]) smears out state adjustments over a specific amount of time T_{amort} by altering the clock rate accordingly. Again, this can be easily done in hardware when an adder-based clock is employed. On the other hand, accuracies are allowed to change instantaneously, however, deteriorations need to be modified during T_{amort} as well due to the clock rate set forth by the state adjustment.

Having explained the technique of continuous amortization, we proceed describing the necessary machinery in the light of an illustrative example depicted in Figure 3. The period before t_1 shows the clock in the non-amortizing state termed *pure phase*, where CLOCK advances with STEP/PURE (recall Figure 2) and ALPHA$\pm$ deteriorates with LAMDBA$\pm$/PURE. Before switching to continuous amortization, the precomputed values for both *amortization phase* and successive pure phase need to be saved in certain preload registers. Kicking off continuous amortization can happen by software or by arming a dedicated timer. In Figure 3 the amortization phase commences at t_1, which entails a reduction of registers ALPHA$\pm$ according to STATE$\pm$, CLOCK advances from now on with STEP/AMORT and the ALPHA$\pm$ registers deteriorate with LAMDBA$\pm$/AMORT. Once started, another timer is activated to time out T_{amort}, and on expiration at t_2 it causes the transition to the corresponding new pure values.

Observe carefully that in this example ALPHA+ shrinks during the amortization phase, and ALPHA- happens to be negative at the beginning, which justifies the sign bits introduced before. To remedy the passage with the negative accuracy, the outgoing A-Bus will be fed with zero during this time, which translates into a valid enlargement of the afflicted accuracy interval. Furthermore it should become obvious from

Figure 3, that UTC is permanently enclosed by the envelopes induced by CLOCK and ALPHA± as well as that local time represented by CLOCK progresses smoothly.

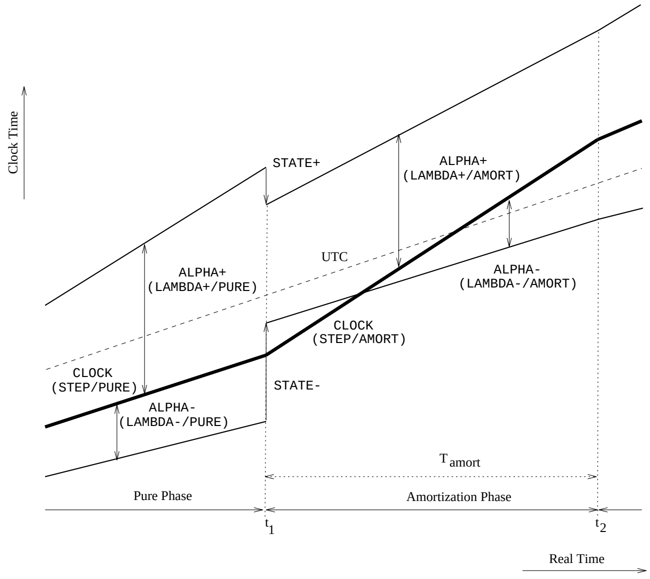

Figure 3: Continuous Amortization

2.5 Synchronization Subnet Unit (SSU)

The SSU is responsible to assist exact timestamping of *Clock Synchronization Packets* (CSP) at both sending and receiving side, which is decisive to achieve tight synchronizations. The well known result of [5], 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. Extending the pioneering work of [3], packets need to be timestamped just the moment they are actually leaving or arriving at a node. This necessitates coordinated support from our UTCSU and from the surrounding hardware, in particular the communication coprocessor [1].

The mechanism to timestamp outgoing CSPs is straightforward. When a DMA-type communication coprocessor reads at a specific address in the transmit buffer containing an outgoing packet, the UTCSU latches the current values of the NTP- and A-bus into registers, which get transparently mapped into the transmit buffer. Figure 4 tries to illustrate such a scenario.

Similarly, timestamping ingoing CSPs happens when the communication coprocessor writes at a specific address in the receive buffer, which causes the UTCSU to sample the current state of the NTP-Bus into registers. Note that no accuracy information needs to be sampled on the receiving side. Furthermore, it is important to point out that any incoming packet may entail such actions, since the communication coprocessor might have no means to recognize packets relevant to clock synchronization immediately on arrival.

Processing the receive timestamp leaves several options: One possibility is to invoke an interrupt service routine, however, a short FIFO would be required, since new packets may arrive before the routine has treated the timestamps of older ones. A more elegant way suggests to map the timestamp transparently into a dedicated region within the receive buffer. No FIFO is needed, still it requires to insert don't care bytes in the packet and additional data path transceivers to funnel the timestamp. Finally, reception times could be collected in a dedicated timestamp memory, which asks for an additional DMA-controller to transfer the sampled reception times from the UTCSU to this memory when packets arrive.

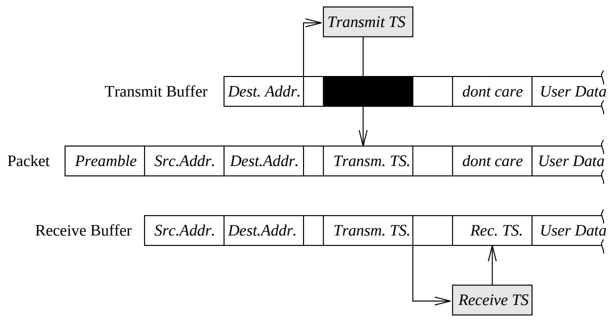

Figure 4: Packet Stamping

In addition, the SSU hosts two 48 bit *duty-timers*, each monitoring the NTP-Bus. A dedicated interrupt will be raised, whenever the NTP-Bus equals or exceeds the timer value. Such programmable timers are useful for clock synchronization algorithms to initiate a CSP transmission or to terminate a CSP reception period.

In a large-scale, hierarchical real-time systems it makes sense to partition the time service into several *Synchronization Subnets* (SSN), hence the name of this unit. Each such SSN consists of a set of nodes connected via a packet network. Endowing the UTCSU with multiple SSUs facilitates gateway functionalities, and allows to handle replicated packet networks (e.g. triple redundant).

2.6 GPS Unit (GPU)

We decided to use GPS technology (consult [1] for an up to date overview) to inject UTC due to 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. It includes a *Standard Positioning Service* (SPS) available for civil users on a continuous, worldwide basis. When *Selective Availability* (SA) is enabled, the horizontal position can be

[1] In our project we are reasonably independent of the actual network; low throughput networks like field-busses (e.g. CAN), medium throughput ones such as Ethernet, or high speed networks (e.g. ATM) are all suitable.

obtained within 100 m (95 percent) and GPS-Time with a maximum error of 340 ns (95 percent). Very briefly, GPS-Time is derived from atomic clocks both at ground stations and inside satellites, which is steered to be within 1 μs of UTC (excluding leap seconds).

There is a great variety of GPS-receivers available, however, most of them output an on-time pulse at 1 pulse per second (aka. *1PPS*), and a few a 10 MHz reference frequency disciplined to the atomic clocks inside the satellites. The associated information identifying the pulse along with other data is provided some considerable time after the 1PPS, usually via a serial interface. A few receivers even tell their status on-line (e.g. locked) with the help of special output lines.

The GPU is responsible for sampling the NTP-Bus into registers and possible status lines of the GPS-receiver when the 1PPS becomes active. The polarity of these input lines are programmable. For redundancy purpose and for the sake of testing several GPS-receivers simultaneously, it is useful to equip the UTCSU with multiple instants of GPUs.

2.7 Application Unit (APU)

For application purposes two conceptually inverse features are supported, namely generating events at specific points in time, and recording the occurrence time of events in question.

The first one is achieved by arming a 48 bit programmable duty-timer. When its value equals or exceeds the one onto the NTP-Bus, it activates a corresponding output line.

The second feature turns out to be more expensive, since time/accuracystamping events requires atomic read access of both NTP- and A-Bus. We provide on chip application support for up to nine different input lines whose polarities are programmable.

2.8 Network Time Unit (NTU)

Exporting the 56 bit NTP-Bus and 32 bit A-Bus allows external devices to capture and process time/accuracystamps independently of the UTCSU operation. In addition, an 8 bit control word is derived from the NTP-Bus to detect/correct bit errors to some extent. Considering the throughput of 96 bits @ 2^{23} Hz (approx. 100 MByte/s) the interfacing NTU becomes a challenging unit to implement. To reduce the pin count, we achieve the necessary performance by pushing out the data via a 48 bit wide multiplexed NTPA-Bus (see Figure 1), driven by both edges of the oscillator pulse.

2.9 Interrupt Unit (ITU)

The ITU handles interrupts, status information and configuration data. All interrupts can be individually enabled/disabled via a configuration register, and the status will be reflected by another compound register. The interrupt sources encountered in the UTCSU are mapped into three dedicated interrupt output lines:

- INT-T becomes active when duty-timers primarily relevant to clock synchronization expire or exceptions (e.g. ALPHA± exceeds BOUND±) arise.
- External events, such as 1PPS signals from GPS-receivers or reception/transmission of CSPs, get announces via INT-N.
- INT-A indicates application-related events originating from the APU, and conditions for chip testing purposes.

2.10 Snapshot Unit (SNU)

This unit supports debugging features for the UTCSU during system integration phase. Debugging requires atomic read access of several registers, whereas the triggering events can originate from two kinds of sources.

A *software snapshot* is triggered either by writing a dedicated SNU register address or by expiration of the snapshot duty-timer. This entails a simultaneous sampling of register CLOCK and registers ALPHA±. Further reads without latching semantics deliver the sampled data.

A *hardware snapshot* is intended for an experimental evaluation of the time service precision. It takes place when the input line HWSNAP becomes active, causing a sample of the current NTP- and A-Bus into dedicated hardware snapshot registers.

To be complete, there is another special input line SYNCRUN that (re)starts the UTCSU from a well defined resynchronization state. This allows to start multiple redundant UTCSUs simultaneously at each node, bypassing initial synchronization efforts.

2.11 Built-In Test Unit (BTU)

To guarantee and to verify proper system behavior, test and self-test machinery is assembled in this unit. The SSU, GPU, APU and SNU are tested off-line at unit level, whereby the BTU enables the self-test option and generates test stimuli.

The LTU and ACU will be tested on-line at chip level. As explained earlier, they permanently generate values for local time resp. accuracy by pushing them onto the NTP- resp. A-Bus. The BTU is in charge of continuously computing *signatures* and *blocksums* over all appearing time and accuracy values. Occasionally, these measures are compared against others, produced by a redundant UTCSU or a general purpose CPU.

3 Design Methodology

We wrap up our exposition by enumerating the stages of the chip design process. Starting out from a written functional specification of the UTCSU, we created a behavioral VHDL model to get familiar with the specification. This model was embedded into a simulation environment including a very basic VHDL model for both CPU and communication coprocessor. In this stage we executed simple algorithms to ensure that all intended functionalities for the clock synchronization algorithm are onhand and work properly. It also served as a feedback to the written specification. A refining process lead us to the still foundry independent *register transfer level* VHDL description ready for logic synthesis.

SYNOPSYS Design Compiler was used to synthesize the *netlist* for AMS in a 0.8 μm standard cell CMOS process and for ES2 in a 0.7 μm standard cell CMOS process. SYNOPSYS Test Compiler inserted *full scan path logic* and *boundary scan logic* according IEEE 1149.1. CADENCE back-end tools finalize the design with *place&route* for ES2. The chip size can be estimated by 100 mm^2.

Conclusion

In this paper we gave a description of a custom VLSI chip designed to support clock synchronization for distributed real-time systems. We motivated the features packed into our chip and revealed details about their implementation up to some reasonable level. Readers interested in the nuts and bolts of our UTCSU are again referred to the technical report [11] or the dissertation [4]. A forthcoming datasheet will provide a complete programming description including register maps, timing diagrams and electrical characteristics.

Besides all architectural strength, we would like to reinforce advantages of the adder-based clock approach. First of all it avoids the use of a Phase-Locked-Loop (PLL) with all its peculiarities, and pulse insertion/suppression techniques with its asynchronity. In general, any time dependent quantity can be tracked with nearly arbitrary granularity by proper increment/decrement operations decoupled from a possible non-binary operating frequency. As pointed out, it eases a fine grained clock rate adjustment apt for both pure and amortization phase. Also leap seconds can be treated. All these benefits are at the expense of an adder instead of a counter and some additional register space.

Further research will be devoted to study the adder-based clock design, and to customize the chip for specific applications. Joining asynchronous running units, in our case the UTSCU with surrounding devices, represents a particular vexing problem and calls for more exploration.

Acknowledgments

We would like to thank the head of project SynUTC Ulrich Schmid for his fabulous coordination and invaluable contributions to specify the UTCSU. Furthermore we appreciate Martin Horauer's persistence to carry out the logic synthesis and routing of this ASIC, providing us with plenty of helpful insights.

References

[1] P.H. Dana, "Global Positioning System (GPS) Time Dissemination for Real-Time Applications", to appear in *Journal of Real-Time System*, 1996.

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

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

[4] D. Loy, *GPS-Linked High Accuracy NTP Time Processor for Distributed Fault-Tolerant Real-Time Systems,* Dissertation, Department of Computer Technology, Vienna University of Technology, to appear 1996.

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

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

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

[8] B. Simons, L. Lundelius-Welch, N. Lynch, "An Overview of Clock Synchronization," *Lecture Notes on Computer Science*, No. 448, pp. 84–96, 1990.

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

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

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