CHALMERS

Firmware for synchronizing Chip-Scale Atomic Clock to GPS

Enabling precise and accurate synchronization, and timekeeping, in distributed underwater sensor networks

Master of Science Thesis in Embedded Electronic System Design

PREBEN THORØD

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, June 2015

Firmware for synchronizing Chip-Scale Atomic Clock to GPS
Enabling precise and accurate synchronization, and timekeeping, in distributed underwater sensor networks
Preben Thorød

Examiner: Per Larsson-Edefors

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden June 2015

Summary

This master thesis is conducted as the final research project in the master program Embedded Electronic System Design at Chalmers University of Technology, Gothenburg. The research is done for the Norwegian Defence Research Establishment (FFI) which has formulated the problem definition. The thesis is a case study of firmware development for synchronization of a Chip-Scale Atomic Clock (CSAC) to global positioning system (GPS). The work is intended to solve the major firmware related research and development, necessary to integrate a GPS synchronized, CSAC driven timekeeping system, to FFI's Networked Intelligent Underwater Sensors (NILUS) demonstrator system.

Symmetricom introduced in 2011 the world's first commercial available Chip-Scale Atomic Clock, which improved size, performance and power consumption by orders of magnitude compared to available technology. In 2015, the CSAC is still the leading edge for precise and accurate timekeeping in mobile applications. By combining the performance and features of CSAC and GPS technology, FFI is hoping to drastically improve synchronization in the NILUS network, which would increase the system performance, and possible open up for new applications.

The overall goal of the thesis is to design, implement and evaluate firmware for a newly developed prototype, the "CSAC board". The focus is on providing a reliable solution that has as good time accuracy and precision as the hardware allows.

A bare-metal firmware solution with the important core features has been implemented and tested. The implemented design provides a solution that is capable of synchronizing independent nodes within $\pm$ 200 ns, and with an frequency accuracy as small as 3.6×10^{-10}. It can capture external asynchronous signals and generate absolute timestamps with a resolution of 100 ns. New challenges and possibilities have been discovered through research and experimentation, valuable for the main developers to start final development and system integration. Both the firmware design and underlying hardware design have been shown to be well suited for the task, and a final solution based on this design is recommended.

Contents

Acronyms

1PPS	One pulse per second (*Electrical signal*)
AIS	Automatic Identification System
ASF	Atmel Software Framework (*Driver and application library for Atmel MCUs*)
CSAC	Chip-Scale Atomic Clock
DARPA	The Defense Advanced Research Projects Agency
FFI	Norwegian Defence Research Establishment
FPGA	Field-programmable gate array
GLONASS	Global Navigation Satellite System
GNSS	Global navigation satellite system
GPS	Global positioning system
I2C	Inter-Integrated Circuit (*Bus specification*)
IDE	Integrated development environment
ISR	Interrupt service routine (*Interrupt Handler*)
MCU	Microcontroller
MCXO	Microcomputer controlled crystal oscillator
MEMS	Microelectromechanical systems
NILUS	Networked Intelligent Underwater Sensors
NIST	The National Institute of Standards and Technology
NTP	Network Time Protocol
NVIC	Nested vectored interrupt controller
OCXO	Oven controlled crystal oscillator
PCB	Printed circuit board
PLL	Phase-locked loop
ppb	Parts per billion
ppm	Parts per million
QZSS	Quasi-zenith satellite system
RbXO	Rubidium crystal oscillator
RF	Radio frequency
RTC	Real-time clock
SI	International system of units
TAI	International atomic time *(Temps atomique international)*
TCXO	Temperature compensated crystal oscillator
TE	Ephemeris time
UART	Universal asynchronous serial receiver/transmitter
UT	Universal Time
UT1	Universal Time major version
UTC	Coordinated Universal Time

1 Introduction

Time synchronization, or clock synchronization, is an important topic that often appears in the design of computer systems and distributed networks. It can be described as the ability to adjust an internal clock to tick synchronously to an external clock reference. Practical limits in the performance of the clocks themselves, or in the clock distribution, lead to non-ideal clock accuracy and precision (stability). This creates challenges with keeping time in different parts of the system synchronous. With accuracy one means the offset between the actually generated clock frequency and the nominal frequency. The precision, or the stability of the clock, is a measure of the clock's ability to tick at a constant rate.

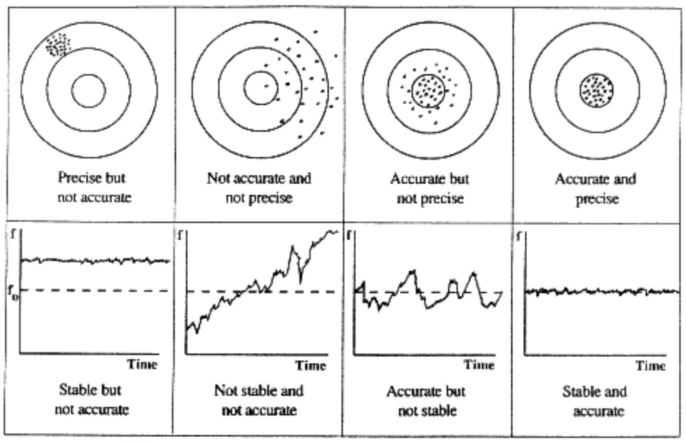

Figure 1.1 Clock accuracy and stability (precision) compared with the marksman analogy. One can see that bad clock accuracy, where the actual frequency has an offset to the nominal frequency, corresponds to bullet hits which deviate away from the center of the target. Stability compares to a small spread but does not say anything about the offset to the nominal value.

The uncertainty and instability in clocks lead to problems with keeping two or more clocks to tick in sync since they are ticking at different rates. Even if two clocks were synchronized with a common starting point, the difference in frequency would cause the two clocks to drift apart and the time error in the system would accumulate. One solution to this can be to periodically resynchronize the clocks, and maybe also estimate and compensate for constant drift error. Different applications have different requirements with respect to accuracy and precision in time synchronization. Many different solutions are available and design decisions have typically been driven by cost and the availability of time reference, in addition to accuracy and precision constraints.

Many of the applications in distributed data processing and communication networks rely on time synchronization between different nodes. An example of such an application is time synchronization in banking and trading systems, where both the time of transaction and the valid balance have to be transparent through the whole system. Another example is the processing of combined sensor data from different sensor nodes, where samples from each node must be mapped to a common time scale.

Network Time Protocol (NTP) is utilized for time synchronization and time distribution on internet and well proven synchronization algorithms are already used in land based wireless radio frequency (RF) communication [3]. Unfortunately, for acoustic underwater communication networks the environment creates challenges that make similar synchronization methods unsuitable [3, 4]. Some of the properties that differentiate underwater acoustic communication from land based communication are the much longer propagation delay, and reflection and diffraction in the water medium. Equally important is the dynamic and changing underwater environment due to weather, climate, seasonal changes, and maritime traffic.

Using acoustic communication through the water medium has been shown to be inadequate for high accuracy and precision time synchronization, mainly due to long and uncertain propagation delay [4]. Without using acoustic or wired communication for time synchronization, independently deployed underwater nodes have limited or no means to maintain synchronization within the network or with other time references. They need to be synchronized before deployment and depend on the accuracy and precision of the internal clocks.

1.1 NILUS

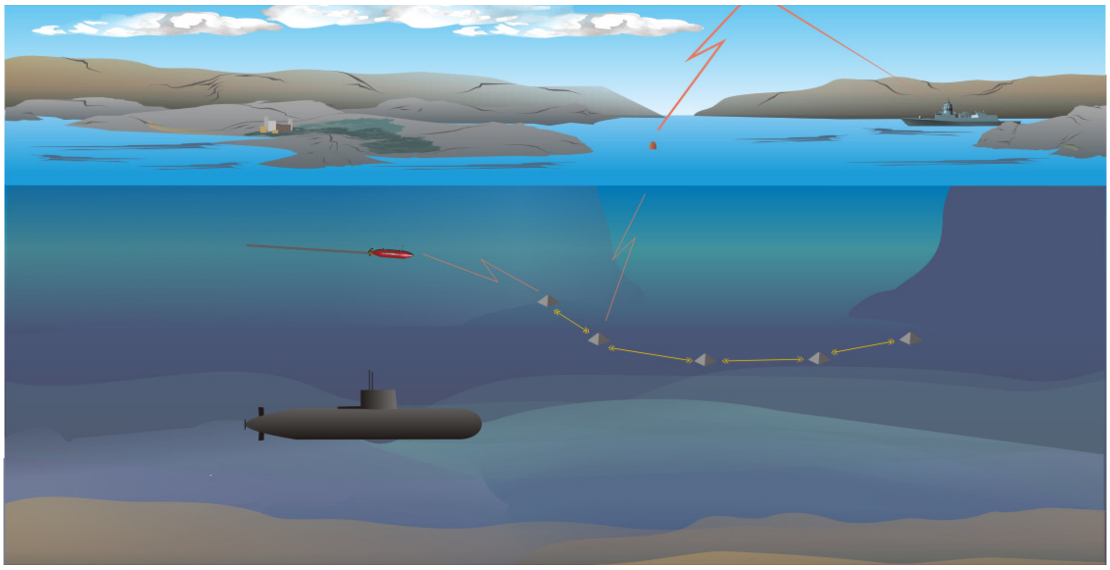

Figure 1.2 Conceptual sketch of underwater communication network with deployed communication and sensor nodes. (Source: FFI)

Norwegian Defence Research Establishment (FFI) has developed Networked Intelligent Underwater Sensors (NILUS), a demonstrator system for easily deployable autonomous underwater sensor networks, where the mentioned difficulties with synchronization and the need for precise and accurate timing are highly relevant [5]. A conceptual sketch of a deployed NILUS network is illustrated in Figure 1.2, where several NILUS sensor nodes communicate acoustically between each other, and with surface nodes through a satellite gateway buoy in the water surface. Each NILUS sensor node is deployed autonomously on the seabed and is powered from an internal battery. They are equipped with passive acoustic and magnetic sensors and have capability of internal data processing. This make them useful for target detection and tracking, among other applications.

The local clocks in the NILUS node have until recently only been regular crystal oscillators, which due to their temperature instability, manufacturing inconsistencies, ageing and more, causes the clocks to drift relative to other NILUS nodes. Even if all the nodes had been perfectly synchronized to absolute time before deployment, the poor accuracy in the crystal oscillators (order of several microseconds per second [6]), would over time ruin the time synchronization in the network. Some of the applications that would benefit from more accurate and precise time synchronization are underwater communication, position tracking, and post-processing of combined data from different nodes. Improved time synchronization in the network could also open for new applications, like using a node as navigation beacon or scheduling of distributed algorithms.

1.2 Clock performance evolution and Chip-Scale Atomic Clock

Achieving the best possible oscillator performance has always been dictated by the practical limits of the available technology and components. Regular crystal oscillators are often troubled by being sensitive to temperature changes, although solutions like temperature compensated crystal oscillators (TCXOs) and oven controlled crystal oscillators (OCXOs) are available for better performance. In the upper scale of performance, time and frequency standards utilizes caesium or rubidium atomic clocks. High accuracy atomic clocks have traditionally not been suited for smaller mobile electronic systems, where weight, size and power consumption are critical. This has now changed since Symmetricom (now Microsemi) in 2011 launched the world's first commercially available Chip-Scale Atomic Clock (CSAC), as an enabler for mobile systems to achieve better timing performance [7][8]. The CSAC is further described in Section 3.2.

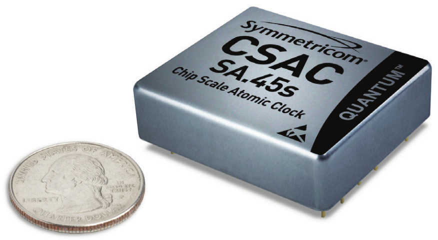

Figure 1.3 Chip-Scale Atomic Clock component. (Source: Microsemi Corporation)

1.3 CSAC board

Taking advantage of the evolution in clock performance, FFI has developed a printed circuit board (PCB), the *CSAC board*, for integrating a CSAC module and a global positioning system (GPS) receiver in the NILUS system. The short-term stability and long-term drift performance of the CSAC and the ability for all NILUS nodes to synchronize to the common GPS time reference before deployment. Time synchronization in the network is solved by using GPS as a common absolute time reference for initial synchronization. After initial synchronization, the synchronization within the network relies on the long-term time drift performance of the CSAC. FFI is hoping that the new solution makes it possible for the network to stay synchronized for the whole duration of the deployment. This would remove the need for more complex and less performing synchronization methods dependent on acoustic communication, and indeed increase the potential performance of the applications.

1.4 Document structure

The document is structured in the following way; The initial definition of work is described in Chapter 2. Relevant theory and background are covered in Chapter 3, followed by the description and discussion about the implementation in Chapter 4. Chapter 5 covers methods and results from measurements and verification of the implementation. A proposal for future work is given in Chapter 6, before final conclusions are drawn in Chapter 7.

2 Initial definition of work

The following sections describe the scope and goals of the work, as defined at the project start.

2.1 Problem statement

The following problem statement defines the main goal and scope for the master thesis:

Design, implement and evaluate firmware for CSAC board with focus on achieving a reliable solution that meets the system specification using concurrent programming methods. With reliable one means both a robust design that will not be prone to faults, and a design that performs as good as possible in terms of time accuracy and precision with the available hardware components. The CSAC board features, both hardware and firmware, should be tested and verified.

2.2 Design options and limitations

The firmware can be designed and implemented in the way that seems most suited for achieving the requirements, with the limitations of using C programming language and the Atmel Studio integrated development environment (IDE). For convenience it is recommended to exploit the Atmel Software Framework (ASF) driver library. The hardware modules have been picked out because they seemed to provide the necessary functionalities, for example the microcontroller (MCU) has been chosen because of the flexible timer modules. It should be possible to use the CSAC board prototype PCB as is, but if design choices or bugs in the hardware prove to be a major issue, the PCB can be modified for the wanted functionality. If either the CSAC board PCB itself or one of the main components prove not to be adequate, suggestions for alternative hardware solutions have to be made.

2.3 Deliverables

The implementation part of the thesis is further divided into the main deliverables defined in the following subsections and they are given the priorities, high-medium-low. High and medium deliverables should ideally be completed, anyhow, the priorities are gives as an indication of which of them that are most important and how the master thesis could be reduced if it is unachievable to complete all of them. High priorities forms the minimal level of acceptance. Low priorities should be completed if there is time.

2.3.1 Real-time clock

Priority: High (RTC)
The CSAC board should provide a RTC to the host processor, formatted as a UNIX timestamp format seconds counter. The RTC should be synchronous with the CSAC and synced with absolute UTC time from GPS.

2.3.2 GPS synchronization

Priority: High

On command from the host processor the CSAC should phase synchronize with the GPS's 1PPS output and the RTC should be updated with absolute time from the GPS module.

2.3.3 Triggered timer-capture and timestamps

Priority: High

On command from the host processor, the MCU should prepare for triggered timer-capture, and after capture provide a timestamp for the event or first clock edge, with the resolution up to 100 ns.

2.3.4 CSAC disciplining against GPS

Priority: High

On command from the host processor the MCU should control and monitor the process of disciplining the CSAC with GPS as the reference.

2.3.5 CSAC control and monitoring

Priority: High

Relevant CSAC control options and status information should be available to the host processor. This could be setting the disciplining time constant and reading CSAC mode and lock condition.

2.3.6 CSAC drift analysis procedure

Priority: Medium

If possible, an automated logging mechanism should be developed to log relative drift performance against GPS time reference.

2.3.7 GPS position and other status information

Priority: Medium

The MCU should provide GPS position coordinates in Automatic Identification System (AIS) 27 bit and 28 bit format when the GPS module has GPS fix. Additionally, other relevant GPS status information should be transparent to the host processor, this could be the number of satellites and such.

2.3.8 I2C host interface

Priority: Medium

An appropriate communication protocol and driver interfaces on both the host processor and the CSAC board must be developed. The host driver must be developed to work under Linux 3.4 kernel with Texas Instruments vendor patches, as currently in use for the host processor. The priority is set to medium since it is more important to implement and test the CSAC and GPS deliverables, which could temporarily be controlled via the serial debug connection.

2.3.9 Long-term drift measurements

Priority: Low

Extended measurements of the CSAC long-term drift. These measurements are very interesting and could support the case study, but since the measurements must be conducted over a long time period and therefore be very time consuming, the priority is set to low.

2.3.10 Suggestions for circuit improvements

Priority: Medium

The CSAC board needs revision, so thoughts on circuit and component changes that can improve the performance or features should be reported.

2.3.11 Suggestions for power consumption improvements

Priority: Low

Other system modules consume comparably more power than the CSAC board and power optimization is therefore not a priority in this work. If time, reports on methods to improve power consumption would still be of interest.

3 Background

This chapter aims to provides the necessary background information so the reader can follow the technical implementation discussion in Chapter 4 and understand the tests and measurements in Chapter 5. It is assumed that the reader is familiar with general concepts of embedded systems and firmware development. An introduction to oscillator specifications and timekeeping is given. GPS is discussed with focus on time synchronization, and important details about the CSAC, the NEO7N GPS module and the SAM3X MCU are described. Ethical concerns are discussed at the end.

3.1 Precision oscillator performance and specifications

As mentioned in the introduction, clocks and oscillators are non-ideal and have different types of instabilities. To be able to select the optimal clock for a design, one needs to know how to interpret reported measurements and specifications. Here is a short walk through of the most common specifications for oscillators.

Both accuracy and stability are specified in a small unit-less value, often in parts per million (ppm) or parts per billion (ppb). For example, an oscillator could have a specified accuracy of 5×10^{-6}, or 5 ppm, meaning that the actual frequency could be off by a factor of 5 ppm compared to the nominal frequency. Short-term frequency stability is often specified as *Allan deviation* $\sigma^2(\tau)$, and can be described in time domain as shown in Equation (3.1)[9].

$$\sigma^2(\tau) = \frac{1}{(2m-1)} \sum_{i=1}^{m-1} (y_{i+1} - y_i)^2 \tag{3.1}$$

Time deviation y gets sampled at a steady time interval τ and $m-1$ is the total number of intervals. Allan deviation is usually given for different time intervals τ or as a function of τ.

Ageing is used to describe frequency changes over longer periods of time, usually on the order of months or years. In addition to errors originating from limitations in the materials and manufacturing process, oscillators are also subject to variations due to environmental effects. Such instabilities can be temperature, gravity/acceleration, vibration, shock, humidity, etc. Choosing a higher performing oscillator in terms of accuracy and stability, is often a trade-off in terms of increased cost, size, weight and power consumption. *Retrace* is a specification of the oscillators inability to repeatedly reach the same oscillation frequency after it has been powered off.

Table 3.1 shows a comparison between common precision oscillators and their typical specifications [6]. Corresponding CSAC specifications, copied from Table 3.2 further down, are listed in the rightmost column for comparison. The data (except for CSAC) is somewhat dated (2000) but gives a good impression of what kind of trade-offs there are between the different oscillator types. Note that the data is based on oscillators specified for wide temperature ranges. In contrast, laboratory standards that operates over smaller temperature ranges can have better performance when operating in more controlled environments [6]. The temperature compensated crystal oscillator (TCXO), microcomputer controlled crystal oscillator (MCXO) and oven controlled crystal oscillator (OCXO) are three of several solution based around regular crystal oscillators with features for improved temperature stability. As seen in the table, accuracy and stability are traded off by increased size, cost and power consumption. Notice that the OCXO introduces a significant warmup time, due to the need to reach a stable oven temperature. The best performing oscillators are the rubidium, rubidium crystal oscillator (RbXO) and cesium atomic clocks, where rubidium and RbXO oscillators are used as *secondary standards* while cesium atomic clocks are used as *primary standards* [6]. Rubidium clocks generally have better short-term stability than cesium clocks, but worse accuracy and long time stability. RbXOs uses the combination of the long-term stability of the rubidium clock the short-term stability of the OCXO, powering on the rubidium clock and to calibrate the OCXO occasionally, resulting in improved power consumption over a rubidium clock. Note also the reduced temperature range for cesium atomic clocks.

Table 3.1 Typical specifications for wide temperature range precision oscillators compared with CSAC. Accuracy and stability are a trade off with increased size, weight, power consumption and cost. (Source: [6])

	Quartz Oscillators			**Atomic Oscillators**			
	TCXO	**MCXO**	**OCXO**	**Rubidium**	**RbXO**	**Cesium**	**CSAC**
Accuracy (per year)	2×10^{-6}	5×10^{-8}	1×10^{-8}	5×10^{-10}	7×10^{-10}	2×10^{-11}	5×10^{-10}
Aging/Year	5×10^{-7}	2×10^{-8}	5×10^{-9}	2×10^{-10}	2×10^{-10}	0	1×10^{-8}
Temp. Stab. (range, C)	5×10^{-7} $(-55$ to $+85)$	3×10^{-8} $(-55$ to $+85)$	1×10^{-9} $(-55$ to $+85)$	3×10^{-10} $(-55$ to $+68)$	$\times 10^{-10}$ $(-55$ to $+85)$	2×10^{-11} $(-28$ to $+65)$	5×10^{-10} $(-10$ to $+35)$
Stability, $\sigma^2(\tau)$ (τ = 1 s)	1×10^{-9}	3×10^{-10}	1×10^{-12}	3×10^{-12}	5×10^{-12}	5×10^{-11}	2.5×10^{-10}
Size (cm^3)	10	30	$20 - 200$	$200 - 800$	1 000	6 000	17
Warmup Time (min)	0.03 (to 1×10^{-6})	0.03 (to 2×10^{-8})	4 (to 1×10^{-8})	3 (to 5×10^{-10})	3 (to 5×10^{-10})	20 (to 2×10^{-11})	2
Power (W) (at lowest temp.)	0.04	0.04	0.6	20	0.65	30	0.13
Price ($)	10-100	<1 000	200-2 000	2 000-8 000	<10 000	50 000	1 500

3.2 CSAC

The Chip-Scale Atomic Clock was developed by the Defense Advanced Research Projects Agency (DARPA), the National Institute of Standards and Technology (NIST), Symmetricom and others. DARPA's CSAC research program started in 2001, where NIST as a main contributor conducted research to enable the creation of an atomic clock with significant reduced size, weight and power consumption. In 2004 NIST demonstrated the first CSAC, manufactured with standard microfabrication and microelectromechanical systems (MEMS) technology [10]. The core physics package's volume was under 10 mm^3, a factor 700 size reduction compared to the smallest available atomic clock physics packages. One of the main goal of the CSAC program was that the research should result in mass producible commercially available product. In 2011 Symmetricom launched the Quantum SA.45s CSAC with a specified short time stability (Allan deviation) of 2.5×10^{-10} at 1 second integration time, less than 120 mW power consumption, and a package size of 17 cm^3 [11]. A summary of the specifications for the Microsemi SA.45s CSAC option 001 is listed in Table 3.2.

Table 3.2 Summary of Microsemi CSAC SA.45s 001 specifications (Source: Microsemi [11])

Frequency	10 MHz
Operating temperature	-10 to $+35$ °C
Max. frequency change over operating temp. range	5×10^{-10}
Frequency accuracy at shipment	$\pm 5 \times 10^{-11}$
Maximum retrace (48 hours off)	$\pm 5 \times 10^{-10}$
Ageing, monthly (After 30 days of continuous operation)	$< 9 \times 10^{-10}$
Ageing, yearly (After 30 days of continuous operation)	$< 1 \times 10^{-8}$
Stability $\sigma^2(\tau)$ $\tau = 1$ s	2.5×10^{-10}
Stability ($\sigma^2(\tau)$ $\tau = 10$ s	8×10^{-11}
Stability $\sigma^2(\tau)$ $\tau = 1000$ s	8×10^{-12}
1 PPS Synchronization	$\pm$100 ns
Operating power consumption	<120 mW
Weight	< 35 g.
Size	17 cm^3
Price (2011)	$1500

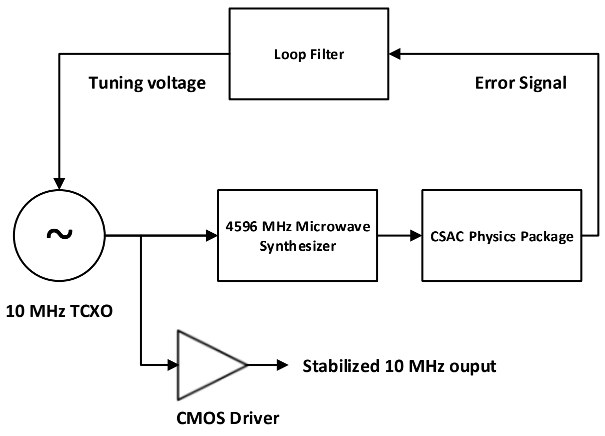

Figure 3.1 CSAC block diagram

The main internal building blocks of the CSAC are illustrated in Figure 3.1. The output signal is generated by a 10 MHz TCXO which is continuously compared and corrected to the ground state hyperfine frequency of the cesium atoms in the physics package. The physics package consists of a cesium vapor cell, heater, and laser optics, among other components. The cesium vapor cell is heated up so the cesium atoms stay in a vapor state. From the system designer's perspective, who implements the CSAC as part of his/her design, the internal operation of the CSAC is not of particular interest, except from knowing that the cesium cell needs to heat up before the TCXO locks to the cesium frequency. The CSAC is specified to reach locked state within 2.5 minutes from power up at 25°C. The CSAC pin configuration is illustrated in Figure 3.2 and reflects some of its features.

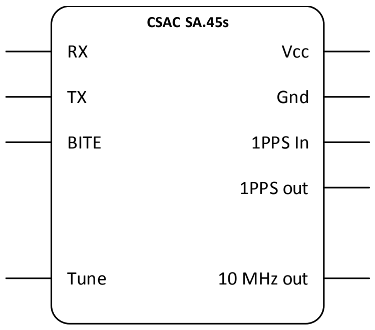

Figure 3.2   CSAC SA.45s pins and signals

The main clock output is a 10 MHz square wave in addition to a one pulse per second (1PPS) output signal. A serial communication port is available to read status information and to control the CSAC. In addition to read the CSAC status information using serial communication, the locked condition can be monitored directly on the Built-In Test Equipment (BITE) pin. A 1PPS reference signal can be connected to the 1PPS input pin. The CSAC can be controlled to synchronize its own 1PPS output signal to this reference 1PPS input signal within $\pm$ 100 ns. To achieve better synchronization than $\pm$ 100 ns and to calibrate the CSAC clock frequency to an external source, the CSAC can be *disciplined* (explained below). The CSAC also has other features such as ultra low power mode, and for support of legacy products, the CSAC allows frequency tuning by applying a steering voltage to the *Tune* input pin.

3.2.1   CSAC disciplining

When disciplining is enabled the internal phase meter measures the relative phase between the CSAC 1PPS signal and the external reference 1PPS signal. Both the phase and frequency is adjusted using a steering algorithm and can ultimately achieve an phase accuracy of $<$ 5 ns and a frequency accuracy of 5×10^{-13}. The time constant of the steering algorithm can be selected by the user in a range 10-10000 s. To achieve successful disciplining one needs to have knowledge of the noise properties of both the CSAC, the reference source, and the phase meter itself, and to set an appropriate time constant. Typical noise performance for different clock sources, for different integration times, is illustrated in Figure 3.3.

For a superior atomic clock reference, with a better short time stability than the CSAC, the time constant can be set low (10-100 s). But for disciplining the CSAC to GPS, which have worse stability than the CSAC up until 3000 s integration time, one needs to select the time constant to where the GPS's instability stops dominating. I.e., the time constant should be set to 3000 s, which is the cross section between the GPS's and CSAC's instability curves.

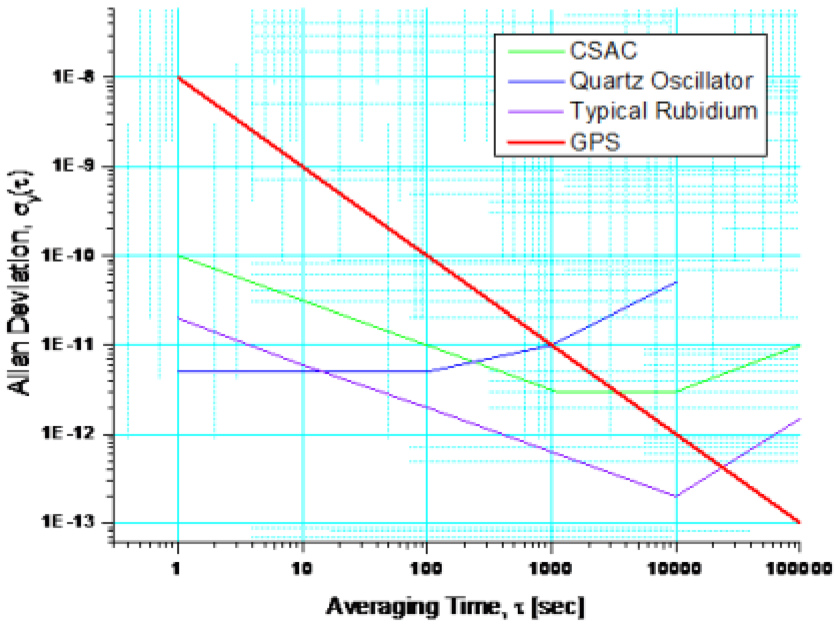

Figure 3.3 Typical instability (Allan Deviation) of CSAC, Quartz oscillator, atomic rubidium clock, and GPS. GPS has bad short time stability, but very good long-term stability. CSAC has better short-term stability than the GPS, but worse long-term stability due to ageing. For disciplining CSAC to GPS, the time constant should be set to 3000 s, where the GPS and CSAC curve intersect. (source: Symmetricom [12])

3.3 NILUS CSAC board

The purpose of the CSAC board is to provide accurate and precise timekeeping, in form of a real-time clock (RTC), and the ability to produce high resolution timestamps to a host Linux processor. The CSAC allows improved long-term clock stability, while the GPS module has the ability to synchronize against absolute time with high accuracy. GPS connection is easily accessible over large geographical areas and several NILUS nodes can be synchronized to the same time reference and deployed independently. Since GPS connection is unavailable under normal operation when the system is deployed underwater, the typical scenario is that the synchronization is done in the short duration when the system is turned on before it is launched from a ship. After deployment, the system depends on the drift performance of the free-running CSAC for the rest of the deployment. Furthermore, the CSAC board utilizes other features in the GPS module, like extracting GPS position.

A prototype of the CSAC board has been manufactured. Figure 3.4 shows a simplified block diagram of the CSAC board. It contains, in addition to the Symmetricom SA.45s CSAC, a GPS receiver and a MCU with interface to a Linux based host platform. The MCU is an off the shelf 32-bit Atmel SAM3X8C ARM Cortex M3, and is connected as an Inter-Integrated Circuit (I2C) slave for the host processor through a standardized system connector [13]. The host will poll for data and send control commands to the CSAC board. The GPS receiver is an u-blox NEO7N-002 global navigation satellite system (GNSS) module and is connected to a PCB mounted GPS antenna [14]. Communication with

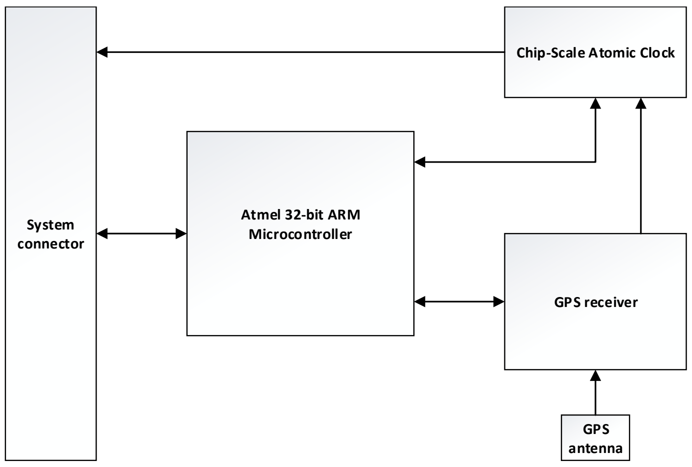

Figure 3.4 Simplified block diagram of CSAC board, showing main components

the GPS module and CSAC is done using universal asynchronous serial receiver/transmitter (UART) serial communication. The CSAC clock signals, a 10 MHz clock and a 1PPS signal, are connected to the MCU's timer modules for supporting the RTC functionality. For timestamp functionality, two buffered timestamp trigger signals are also connected to the timer modules. Typical timestamp trigger signals are event signals and sample frame clocks from data converters. The CSAC has also been wired for using the GPS module's 1PPS timepulse output signal for synchronizing the CSAC to GPS and to discipline the CSAC to GPS to compensate for ageing.

3.4 GPS as time reference

One of the fundamental requirements for achieving the performance we see in today's GPS system is precise and accurate timing [15]. Very small time errors could result in large position errors, and without a reliable and stable clock scheme, GPS would not have been possible. In fact, it was the development of the atomic clock that enabled the realization of the GPS system[16]. A GPS satellite normally has four or more atomic clocks on board, this for redundancy, but also for averaging the clock signals, producing a more stable clock than one single clock source. Information about satellite orbit and location (*ephemeris data*) is exchanged in-between the satellites, as is time information. Two characteristics are especially important with GPS as time reference. The first is availability, GPS coverage is available most places where there is large enough view of open sky. The second is the long-term stability; since many sources contribute to the common time scale, it is one of the most stable long-term time sources available. Initially the best performance was exclusive for military use, but in the 2000s, American politicians opened up the high-end performance of GPS also for civilian usage. This, and the ongoing hardware evolution, have enabled for GPS clock synchronization with uncertainties down to 1 ns [17].

GPS is the best supported GNSS system with better coverage and more satellites than any other GNSS system. But one should notice that other systems, like the Russian Global Navigation Satellite

System (GLONASS), the European Galileo, and the Asian quasi-zenith satellite system (QZSS), are up and coming.

3.5 u-blox NEO7N GPS Recevier

Figure 3.5 u-blox NEO7N GNSS/GPS module. (Source: u-blox AG)

The u-blox NEO7N module, pictured in Figure 3.5, is one of the newest standalone multi GNSS receivers from u-blox. It is marketed as a low power module with high sensitivity, low acquisition times, and a broad feature set. It supports reception of GPS, GLONASS, Galileo, QZSS, and SBAS satellite signals, using either a passive or active antenna [18]. The module has several interfaces for serial communication with a host computer or MCU: UART, USB, SPI, and DDC. The ASCII character serial protocol NMEA 0183, which has historically been used for GPS receivers and other maritime equipment, is supported for sending printable comma separated messages with data such as: position, time, speed, fix and satellite information. Additionally, u-blox's own proprietary binary protocol, *UBX*, is supported for an extended set of commands and messages which are not supported in the somewhat dated NMEA protocol or are specific to u-blox devices [19]. The UBX protocol can also be used for control and configuration of the device. Data messages can either be polled by sending appropriate commands one by one, or specific messages can be configured to be sent periodically to avoid the need for sending query commands.

A battery backup power supply can be connected to the module for retaining ephemeris data when the module is powered off. With this feature, u-blox NEO7N supports different start up modes: *Cold start*, *Warm start* and *Hot start*. In cold start the module has no information about the last position, velocity, frequency etc, and needs to search through all the satellite channels to find a satellite that can be tracked. Then the ephemeris data needs to be decoded, which is specified to take approximately 18-36 seconds under strong signal conditions. If the module has retained ephemeris data from the last time it was powered on, this data can be used to achieve faster fix acquisition upon start up. Ephemeris data are typically outdated after 4 hours, so a hot start means that the module still has valid ephemeris data upon power up (i.e., module has been powered down for less than 4 hours). A warm start can be conducted if the module has been powered off for more than 4 hours and has retained ephemeris data. The ephemeris data is then outdated, but the receiver has approximate information about time, position and coarse satellite positions (Almanac). For the NEO7N module, the time-to-first-fix for the different start up modes are specified as in Table 3.3 when satellite signal levels are -130 dBm.

Table 3.3 u-blox NEO7N's specified time-to-first-fix with different start up modes.

Mode	**Time-to-first-fix**
Cold Start	29 s
Warm Start	28 s
Hot Start	1 s

Since downloading full ephemeris data from four satellites can either take long time or be unobtainable with bad signal conditions, NEO7N supports more advanced features to boost acquisition performance by extracting ephemeris, almanac, accurate time and satellite information from other sources than the satellites. This is called aided/assisted GPS (A-GPS) and assisting data can be: position, time, frequency, orbit data and more. A network attached system could for example extract position or time information from the network and use this as assisting data. In a precise timing application, a reference frequency signal could be used as assisting frequency data.

Depending on the number of satellites available and the signal quality, different modes of fix can be acquired and reported. *2D fix* can be achieved when only three satellites are available, and full *3D fix*, with both time, position and altitude, can be acquired when more than four satellites are available. Navigational data and a set of accuracy estimates are calculated at a default rate of 1 Hz. A comprehensive set of configurations can be set to tailor the receiver's performance to the application, for example the platform dynamical setting (portable, stationary, automotive, at sea, airborne) and input and output navigational data can be filtered with wanted threshold.

For time synchronization, the module can report GPS time and UTC time (see Section 3.6.4 and 3.6.7). It also provides an electrical time pulse signal for precise timing applications. The time pulse is by default configured as 1PPS and is synchronous to GPS time. The accuracy of the time pulse signal is specified as 30 ns RMS, and 60 ns at the 99-percentile.

3.5.1 Nested vectored interrupt controller

The SAM3X ARM Cortex M3 microcontroller features the ARM nested vectored interrupt controller (NVIC). Preemption of interrupt service routines (ISRs) in the NVIC interrupt system in the SAM3X MCU is dictated by the chosen preemptive interrupt priority level *(group priority level)* [20]. An interrupt with higher group priority level can preempt another ISR but an ISR can not be preempted by an interrupt with equal or lower group priority level. Additionally, one can configure a *subpriority level*. If two or more interrupts with the same group priority level arrive at the same time, the subpriority level determines which of the interrupts that get handled first. The other interrupts are pending and are handled later.

3.6 Timekeeping

3.6.1 Historical timekeeping

Since ancient time, people have urged for a way to keep track of time. The regular passing of the sun over the sky, and astronomical observations like moon and star placement, were used to organize the passing of time into what we know as calendars. The day got further divided by innovations like the sun dial and the pendulum clock [21]. Common to all these methods of timekeeping is that they consist of a steady flow of observable events, or oscillations, and a way to keep count of the number of oscillations. This general method of counting oscillations is still applicable for the most modern and best performing timekeeping systems.

3.6.2 Time scales and the second

Even before it was able to be measured, the second was defined as the *second* 60 division of the hour. In 1884, the Universal Time (UT) was introduced as a time standard based on the Earth's rotation. The second now got defined as a subdivision of the mean-solar time, and UT, with UT1 as the major of several versions, was the time standard in use until 1960 [17]. With increasingly better clocks and observation methods, it was discovered that the speed of the Earth's rotation was not steady. The ephemeris time (TE) was introduced in 1960 as new dynamic time scale which took into account the slowing speed of the Earth's rotation, it held as a time scale until 1967. According to TE, the second was defined as a fraction of a specific tropical year (1900). One of the biggest problem with this definition was that it was not possible to reproduce, which gained motivation for an improved time scale that was both reproducible and independent of astronomical phenomena. The first caesium frequency standard was operational in 1955, and in 1958 the frequency of the caesium was determined in terms of the second. This led to the adoption of the new and current definition of the second in 1967, the international system of units (SI) second. The SI second is defined as following: *The second is the duration of 9 192 631 770 periods of the radiation corresponding to the transition between the two hyperfine levels of the ground state of the cesium 133 atom* [22].

3.6.3 International Atomic Time (TAI)

The international atomic time ((Temps atomique international), TAI) is an atomic time scale that is currently in use worldwide [23]. It is derived from averaging about 400 atomic clocks all over the world, making the time scale more accurate than a single atomic clock and giving better redundancy in case of outfall and faults. TAI is a physical time scale that is independent of the astronomical events and the earth rotation.

3.6.4 Coordinated Universal Time (UTC)

Coordinated Universal Time (UTC), not to be confused with Universal Time (UT1), is the time scale that is currently used for civil timekeeping. UTC ticks with the SI second as interval, at the same rate as TAI, but it is synchronized to approximate mean-solar time (UT1), which makes it correspond to the solar movement and the Earth's rotation. Because the Earth's rotation is irregular and has had the tendency of slowing down over the decades it has been possible to measure, TAI is slowly drifting apart from the UT1. To keep UTC in sync with UT1 it needs to occasionally be adjusted. The Leap Second has been introduced as a measure for this adjustment.

3.6.5 Leap seconds

The leap second is a whole integer offset that is introduced to compensate UTC for the irregularities of the Earth's rotation. This way, TAI and UTC is ticking with the same rate, keeping the TAI as an continuous and linear time scale, perfect for scientific purposes, while making UTC match the solar movement, ideal for daily, navigational and astronomical use. As of April 2015, UTC has been compensated with 35 positive leap seconds, making UTC lag 35 seconds behind TAI. Leap seconds are introduced to ensure that UTC and UT1 never drift apart more than 0.9 seconds.

System developers who utilize UTC face two main problems. Firstly, the duration a day is not always 24 hours (86400 SI seconds) any more. When a positive leap second is inserted, a day last for 86401 seconds, and the clock could read 23:59:60. If a negative leap second is inserted, the day will only have a duration of 86399 seconds, and the clock would jump to 00:00:00 (on the next day) after 23:59:58. Another problem is that the rate of the insertion of leap seconds is irregular and unpredictable. Leap seconds are usually announced six months or more in advance.

3.6.6 UNIX time format

The UNIX time format is used to represent an instant in time in some computer systems. It is realised by a signed 32-bit scalar value that holds the number of seconds since Thursday, 1 January 1970, 00:00:00 UTC, also called *the Epoch* [24]. Positive values represent time after the Epoch. All days are defined to last 86400 seconds, and this lead to a discontinuity in UNIX time when leap seconds are introduced. When a positive leap second is added, the value of the last second of the day get repeated once. The consequence is that a UNIX timestamp that represent the last second of the day a leap second was added, also represents the second before. If a leap second is removed it means that there exist a UNIX timestamps that does not represent a moment in time. Because of the 32-bit range, the UNIX time will overflow in 2038 [25]. Note that the choice of time scale is implementation specific; a system does not necessarily need to use UTC as time scale and some system use UNIX time format with TAI as time scale.

3.6.7 GPS Time

GPS time is the system time in the GPS satellites. It is formatted as the number of weeks and the number of elapsed seconds in that week, since the epoch Sunday, January 6, 1980 00:00:00 UTC. The difference between UTC and GPS time is in whole seconds, and the value of this offset is included as part of the GPS navigation message [15]. GPS time is not compensated with leap seconds and the difference between GPS time and TAI is therefore constant (TAI is always 19 seconds ahead of GPS time).

3.7 Task analysis and design decisions

With the hardware modules and the main wiring already developed, the design space for the present work laid within the firmware design. The biggest initial decisions concerned the scheduling and communication scheme, and whether to use a complete RTOS or implement a custom *bare-metal* solution. Reliability, development effort and maintainability were considered as the most important design requirements. The CSAC board was considered to be a small system, with relatively low complexity, and few real-time demands. No tasks demanded any particularly heavy processing, but most tasks involved communication with the peripheral modules (GPS receiver, CSAC, host processor and debug computer), i.e., a typical control system. Even though parts of the design are highly time critical and the end performance is a result of the system's timing behaviour, the real-time requirements were quite relaxed. Even though the serial communication demands the firmware to handle several communication streams at different rates, the abstract events in the system occur at a rate as low as a few events per second. For comparison, workloads in a media oriented embedded systems could typically include constant data processing and updating of displays and audio streams at rates up to several thousand times per second. There was no need to operate any abstract task at a higher rate than the natural rate of the CSAC and GPS, i.e. 1 Hz. Even though the underlying data rates are in the order of thousands of bytes per second, only a few messages need to be handled within a second. With all important events originating from hardware modules with interrupt capabilities, an interrupt driven bare-metal solution, using buffering, state machines and simple locking mechanisms, seemed most suitable. The control application could also be driven at the rate of one tick per second, making the rate, volume and scheduling of the communication more predictable.

The need for tailoring the low level timer system, in addition to the low number of tasks, supports using a bare-metal solution over for example a full RTOS implementation. A fully event driven system, utilizing event queues and semaphores, was also considered, but the design effort to build such a system seemed unnecessary and would possibly make the code more complex. For a system where scalability was important, implementing either a fully event driven design or using a complete RTOS would make more sense, since it would be much easier to add new tasks later. It seems though unrealistic that this system will be extended much, as the system has most value as a small and simple time module, without any other added features than provided by the timekeeping and GPS functionality. A bare-metal solution with a central control application, scheduled with an internal system tick derived from the CSAC's 1PPS pulse, was therefore chosen.

3.8 Ethics

Engineers and researchers are barely confronted with the same class of ethical dilemmas as doctors, soldiers and politicians face in their daily duty. However, their work plays an increasingly important role for the public's life quality, health and safety. Every day, people entrust their own and others life and safety to technology that engineers have brought to the world, be it safety mechanisms in airplanes, pacemakers, or self-driving cars. Not only does the society, government and managing people trust the engineer's technology, but they also entrust their expertise in the field and base their judgement on their recommendations. This contract of trust makes it utterly important that engineers and researchers deliver quality and transparency in their research, work and communication.

The master thesis is publicly available, and will hopefully be of help for others who research similar topics. It is also a direct contribution to FFI's research. The decision of contributing to defence research can sometimes be controversial, but was not that difficult in this case, much helped by knowing that Chalmers and FFI both encourage openness and ethical discussion, and both have formal frameworks for ethics and clearly stated visions [1] [2]. The thesis work would be a contribution to two organizations that recognized their importance and responsibility in society.

The result of this project can hopefully be used to improve existing sensors, communications or data handling. Although the research can be beneficial for many civilian applications, it is mainly developed in context of improving existing and innovate new defence systems. The task of improving time synchronization and precision by itself can seem distant from operative decisions, where one in worst case choose between life and death. It is important to recognize that the quality of the sub-system can play a significant role in the shaping of abstracted data, that in turn are used for operative decision makers. Therefore, how results and uncertainties are presented is important to enable decision makers to make informed decisions.

4 Implementation

This chapter describes the different parts of the firmware implementation in detail.

4.1 Software modules

The firmware is divided into smaller software modules as illustrated in Figure 4.1. The MCU's built-in *Timer Counter* module is utilized for the timing and RTC features and has its own driver. The hardware UART and I2C modules are utilized for communication with the external devices. Each external device; the CSAC, GPS receiver, host processor and debug port has its own dedicated device driver, handling both the low level communication, like UART transmission and reception, and higher level protocol and data handling. The overall control of the firmware is rooted in a state machine, *the main state machine*, which are described in Section 4.6.

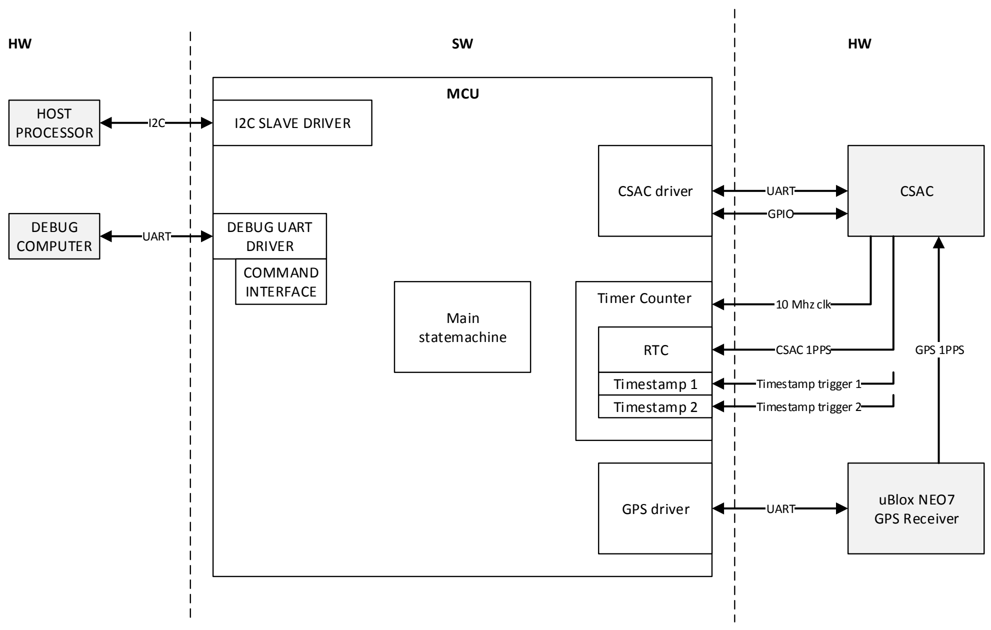

Figure 4.1 Software architecture block diagram showing the different software modules in the firmware

4.2 Control hierarchy and communication

The interaction between the different software modules is illustrated in Figure 4.2 and highlights some important aspects of the chosen architecture. The main state machine is the central unit that monitors and controls the peripheral modules. It is executed every second and progresses through different synchronization states. The state machine is described in detail in Section 4.6. Concurrent execution is achieved by utilizing the different ISRs. This way, time critical tasks like updating the software RTC value and the creation of timestamps are handled with low latency. Incoming serial communication is also handled in ISRs, but the ISR merely buffers the data, which is read out in normal execution context. The newest data is either kept internally in the device drivers or in the state machine. A selection of the status information from the GPS, CSAC and the state machine itself, in addition to the latest timestamps and the current RTC value, has to be available for reading by the host processor. When the host requests to read data, the I2C slave driver has to be ready to respond immediately with the latest data. Therefore, when the different modules get updated data, they also write a copy to a shared memory in the I2C slave driver. To avoid partly updated data due to race conditions, the shared memory in the I2C driver is protected by a mutual exclusive locking mechanism. Further details of this mechanism and the other device drivers are explained in the following sections.

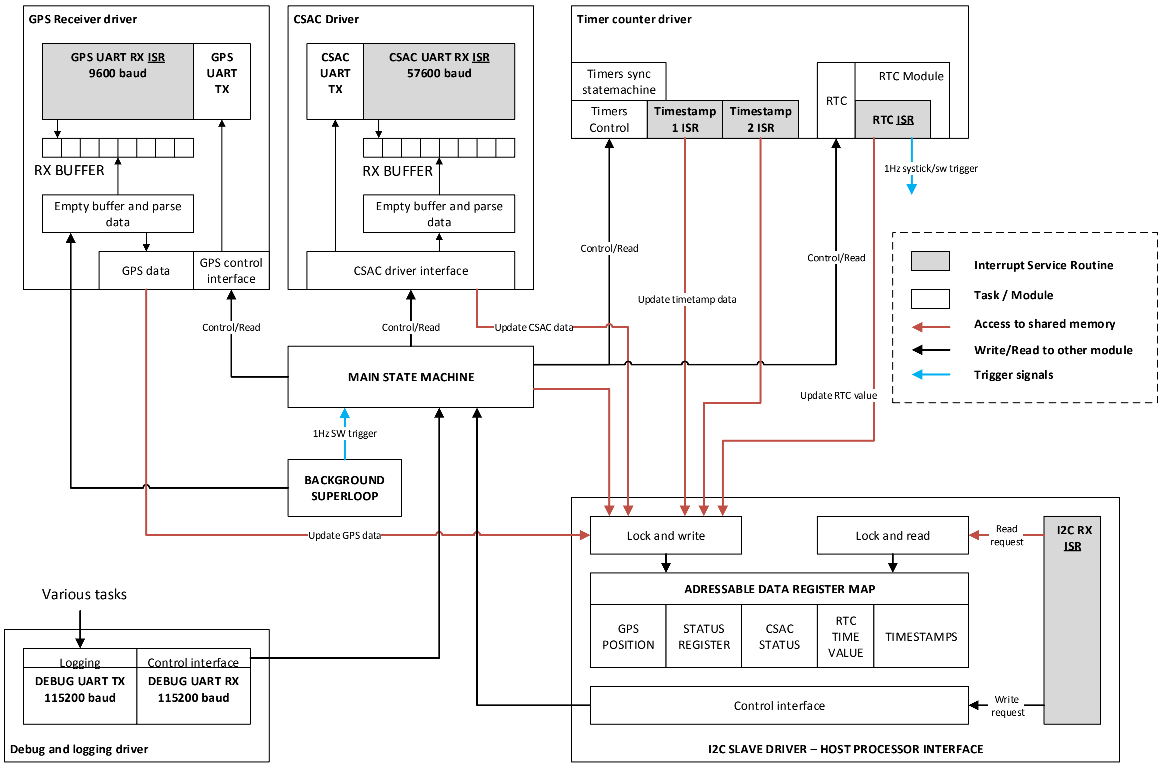

Figure 4.2 Control hierarchy and communication. The main state machine accesses the different modules for control and status information. Each module also writes new status data to the shared memory in the I2C slave driver. Time critical tasks like updating the RTC value, registering timestamps, and buffering incoming serial communication data, are offloaded by ISRs

4.3 CSAC driver

The CSAC driver permits both control and monitoring of the CSAC. From power up, the CSAC cycles through an initialization process before it reaches the locked state. The lock condition, in addition to other status, alarm and mode registers, are available to be read. It is necessary to send commands to the CSAC to start and stop the synchronization and disciplining process. During the disciplining process, the disciplining status and phase error are also monitored.

Control and monitoring of the CSAC is done using a simple serial communication protocol, where the MCU sends a command for polling status information or sends a control command and receives an acknowledge message from the CSAC. One of the hardware USART modules in the MCU microcontroller is used for the serial communication. To avoid losing incoming data, an ISR is triggered on reception of a new byte and the new byte is stored to a circular buffer.

The CSAC driver is implemented as a blocking driver for all commands except the synchronization task. All enquiries to the CSAC is done in the main state machine that is executed once per second. One can assume that the CSAC replies to commands right away as long as the CSAC is operational. Because blocking code is usually easier to implement and to interface, a blocking implementation is chosen for the main parts of the CSAC driver. The highly regular and predictable usage of the driver, since it is always used one or two times at the start of each main state machine execution, also supports this decision. A blocking driver would block the background execution in case of communication fall out and could cause the system to hang. Therefore, a timeout functionality is implemented to avoid this. The blocking implementation works as following: A command is sent to the CSAC and the driver immediately start to empty the buffer. The driver does a blocking read of the buffer until a complete message is read before the message is parsed and data is updated. If the driver does not receive a complete message before before a defined time period, it will return with an error and let the calling background task continue its execution.

The CSAC responds to all commands immediately except for the command that initiates the synchronization of the 1PPS signal. For this command, the CSAC starts the synchronization process and replies with a confirmation message when the process is completed. If the CSAC is not able to synchronize, the CSAC itself will time out after three seconds and reply with a synchronization error message. A blocking implementation for this command is not suitable since the driver would in worst case block for three seconds, waiting for the error message. For this command only, a non-blocking implementation is chosen. Here the MCU sends the synchronization command and returns immediately without blocking, and a state machine in the driver keeps track of the communication state. The driver must then be called again periodically in the next executions of the main state machine. For each call to the non-blocking driver, the driver reads the data from the buffer and returns. The driver will close when either a synchronization message is received (success or error) or when the driver times out (exceeds four seconds). Until this, the driver indicates by its return value that communication is still ongoing and it must be called again.

To ensure that the CSAC driver completes and closes before it is called again, the driver as a whole is protected by holding its communication state (READY, BUSY, SYNC_SENT). A call to the driver when the driver is busy will return with an error.

4.4 GPS receiver driver

The GPS driver communicates with the u-blox NEO7N GPS receiver by serial communication using a hardware UART module in the MCU. The reception is interrupt driven and the ISR stores each incoming byte in a circular buffer, similar to the CSAC driver. In contrast to the CSAC where the MCU queries new data by sending a query command, the GPS receiver is configured to periodically send data messages without the need for the MCU to send query commands. As part of the initialization, the GPS driver configures the GPS receiver to disable periodic NMEA messages (enabled by default) and enables UBX messages to be sent periodically once per second. Depending on the state of the main state machine, the GPS driver is used to enable and disable the different UBX messages. The binary UBX protocol is used instead of the string based NMEA protocol to avoid unnecessary converting of string data to binary/numerical data. The UBX messages are also organized better than the NMEA messages, either by gathering data of the same type in one message, or by gathering an essential collection of different types of data in one message. This allows tailoring the communication to the application requirement, reducing unnecessary overhead. The UBX protocol also supports specific u-blox configuration commands that are not supported in the NMEA message standard. Examples of UBX messages that are used are *UBX-NAV-PVT* which is a summary of both position, velocity and time information and the *UBX-TM-TP* message which contains time information about the last GPS time pulse (used for synchronization of the RTC to UTC time, described in Section 4.8.1).

The circular buffer is read and emptied in the firmware's superloop. After a complete message is read, the data and checksum is verified before it is parsed and stored.

4.5 I2C slave driver

The host processor communicates with the CSAC board using I2C. It does that by sending addressed write or read commands. The I2C slave is interrupt driven and will trigger the ISR on transaction changes (Start, read, write, acknowledge, stop etc.). The transaction states are handled in a state machine internal to the ISR.

For reading, the host processor addresses a specific data byte and sends a read command. It is therefore convenient to store all the data that the host has access to in a continuous addressable register map. If the host wants to read a bigger data set, it can send the address of the start position of the data and send successive read commands.

4.5.1 Avoiding race conditions in shared memory

Since the data origins from different modules in the system, this data is a shared resource and needs to be protected to avoid race condition. An example of this could be that the host processor requests a read of the GPS position data (stored over several bytes) while the GPS driver is in the middle of updating the GPS position data. Since the I2C driver is interrupt driven it would reply with a partly new and partly old position if no protection of the shared memory is implemented. Atomic write and read access to the shared memory is needed.

Two types of tasks have write access to the shared memory, either running in interrupt context or background context. The RTC and timestamp ISR runs in interrupt context while the main state machine, the CSAC and GPS driver updates its data in background context and could be interrupted by an I2C request.

To ensure atomic access to the shared memory all the write commands need to be able to complete the writing before the I2C ISR can be allowed to run. For the background tasks this is solved by disabling the I2C interrupt before starting the writing, and enabling I2C interrupt after completion. Atomic write access to the shared memory from the different interrupt sources is ensured by configuring the interrupt priority have the same preemptive priority level (*group priority level*). This means that the RTC, timestamp and I2C ISR can not preempt each other and have to run to completion before another is executed.Since the write access is completed as a whole in interrupt context and none of the interrupts can preempt each other, atomic write access to the shared memory is ensured. If one of the timer interrupts arriving at the same time as the I2C interrupt, the order of execution is determined by the interrupt *subpriority level*. The timer interrupts are given a higher priority than the I2C interrupt to ensure the RTC and timestamps get updated with as little latency as possible.

In contrast to the write operation to the memory, a read access is not atomic to a single execution of the ISR. A typical read transaction consist of I2C state changes and the ISR is triggered several times. This allows the data producers to do write operations in between the I2C ISRs, and race condition is still not avoided. To avoid this, the I2C slave driver copies the whole data register into a local memory on the first read access query. The driver can then output data from this local memory on successive read queries until the host terminates the I2C transaction. A benefit of this implementation is that the read access to the shared memory kept short, avoiding locking the shared resource for a unnecessary long duration.

4.6 Main state machine

The main state machine is the central control application in the firmware and is responsible for controlling the different synchronization and monitoring tasks. In the ultimate operational state of the CSAC board, the CSAC is locked and stable, the RTC is synchronized with absolute UTC time from GPS, and the unit is ready to generate timestamps. Since neither the CSAC nor the GPS is ready directly after power up, a process for monitoring their statuses is needed. Additionally, some intermediate set-up and synchronizing of the MCU's timer modules are needed and the RTC must be updated with UTC time value from GPS. The main state machine is illustrated in Figure 4.3, and tries to reach the state for normal operation, *SYNCED_WITH_GPS_**. Below is a description of the behaviour of the state machine and its transitions; implementation details of each synchronization task is described in more detail in Section 4.7 and 4.8.

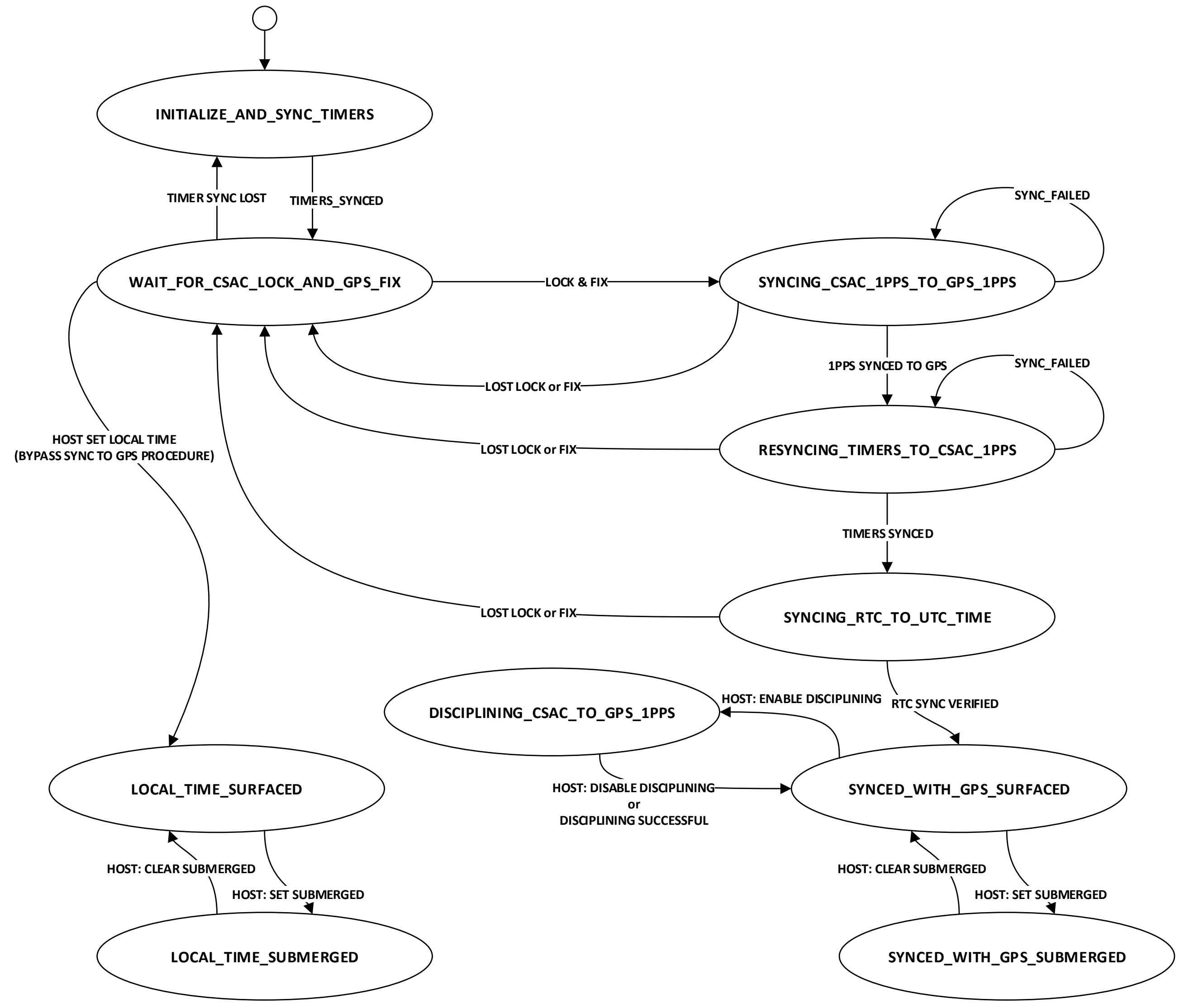

Figure 4.3 Main state machine

From power up, the MCU goes through low level initialization of the MCU modules, drivers and software, before the state machine is started. After this, the state machine is executed once per second, monitoring relevant status data. In the initial state, *INITIALIZE_AND_SYNC_TIMERS*, the internal timer counter modules in the MCU are synced to the CSAC 1PPS signal.

In the *WAIT_FOR_CSAC_LOCK_AND_GPS_FIX* state, the CSAC lock status and GPS fix are monitored, both needed for the following synchronization tasks. The time needed for the CSAC to enter locked mode is under 130 seconds, and is somewhat consistent. Unfortunately, the time the GPS uses to establish fix is not predictable and in worst case, if the GPS coverage is bad, it can not be guaranteed to establish fix at all. With good GPS coverage, GPS fix and sufficient time accuracy estimates can be established in seconds. If GPS fix is unattainable, the host processor can choose to bypass the GPS synchronization process and write its own local time to the CSAC board[1]. The state machine would then enter the state *LOCAL_TIME_SUBMERGED*.

When the CSAC is in locked state and GPS fix is established, the actual synchronization of the RTC to absolute UTC time using GPS starts. It is done by stepping through a sequence of three synchronization tasks, each represented as a state in the state machine. Common for the three synchronization states is that GPS fix must be maintained. If GPS fix is lost or the CSAC loses lock condition, the state machine falls back to the *WAIT_FOR_CSAC_LOCK_AND_GPS_FIX* state.

The first synchronization state is *SYNCING_CSAC_1PPS_TO_GPS_1PPS*, where the CSAC 1PPS pulse is moved, or phase shifted, to correspond to the closest GPS 1PPS pulse. This is done utilizing the CSACs built in functionality for synchronizing to an external 1PPS signal. If the CSAC is unable to synchronize to the GPS 1PPS signal, it times out after 3 seconds. The state machine on the other hand restart the synchronization procedure as long as GPS fix is maintained.

When the synchronization of the CSAC 1PPS to the GPS 1PPS signal has been successfully executed, the state changes to *RESYNCING_TIMERS_TO_CSAC_1PPS*. Since the 1PPS now most probably have been phase shifted compared to the start up condition, the internal MCU timer modules is now out of sync with the CSAC 1PPS signal. The timers are therefore re-synced with the CSAC 1PPS signal, similar to the procedure in the initial state, *INITIALIZE_AND_SYNC_TIMERS*.

From now on, the CSAC clock signals and the internal timers are clocking synchronously with GPS time, but the RTC time value has not been updated to correspond with absolute time. The last step of the synchronization process is done in the state *SYNCING_RTC_TO_UTC_TIME*, where the UTC time value is extracted from the GPS using the UART communication channel and is mapped to the RTC time value.

After this last synchronization step is completed successfully, the state machine enters the state *SYNCED_WITH_GPS_SURFACED*. All synchronization tasks are complete and the NIILUS node can be deployed underwater. From now on, the GPS is not needed more for synchronization purposes, but in this and preceding states, the GPS is also used for tracking position. GPS position is still updated in this state, allowing to track the position until the node is deployed under water. The host processor can send a message, informing that the NILUS node has been deployed under water, and the state machine enters the state *SYNCED_WITH_GPS_SUBMERGED*. This state is identical to the *SURFACED* state, except that the GPS module is not used. On recovery of the NILUS node, the host processor can notify that the node is surfaced, and the state changes back to *SYNCED_WITH_GPS_SURFACED*,

[1]The local time could be the host processor's system time, which could already be obtained from an earlier synchronization or from a battery backed up RTC. In this context, the term *local time* must not be confused with the same-named term used to describe the local time zone.

GPS is enabled, and the GPS position can be reported through the underwater acoustic modem as an aid for locating the node. From the state *SYNCED_WITH_GPS_SURFACED*, the host can also start disciplining of the CSAC to the GPS, where the state machine enters the *DISCIPLINING* state. The host should only start disciplining when the system is in a controlled and stationary environment, since the process needs to complete before returning to normal operation.

To give some insight in what happens to the different clocks and timing signals in each state, Figure 4.4 show a simplified view of the different timing signals corresponding to the states in the state machine earlier described. The three timer channels are used for driving the software RTC and timestamp functionality. Again, the diagram is a simplified view, and each synchronization task is described in more detail in Section 4.7 and 4.8.

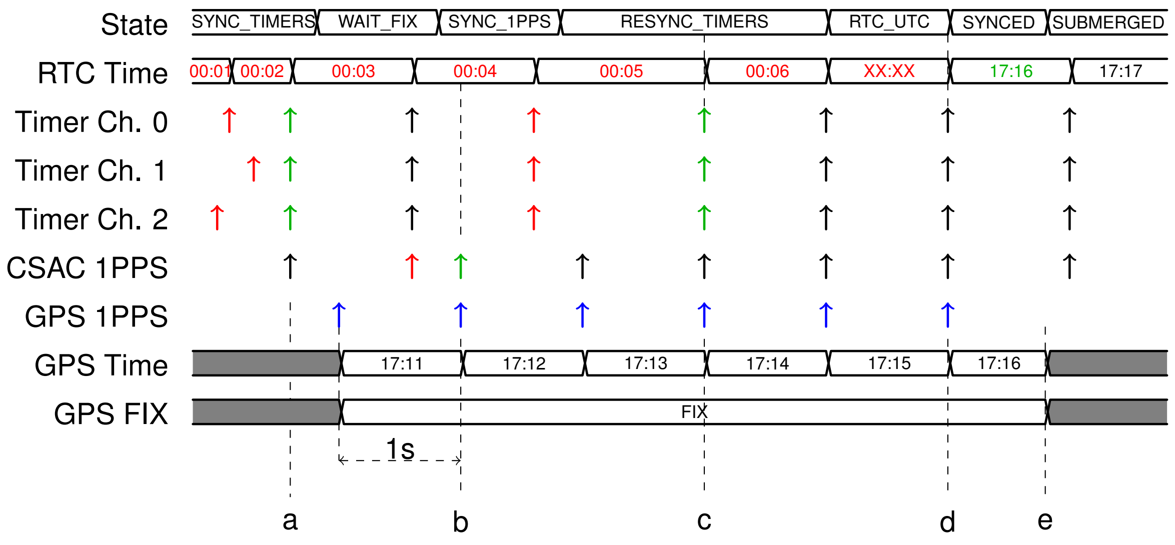

Figure 4.4 *Simplified time diagram showing changes to the various time signals during the different states in the main state machine, during an ideal synchronization. The main goal of the synchronization process is to sync the internal timers with the GPS 1PPS signal (blue arrows), and the RTC with GPS UTC Time. Signals that needs to be synchronized are marked with red arrows. Signals that just have been synchronized are marked in green. Note that the actual behaviour differs from the diagram since some states have longer duration than one second.* ***a)*** *The internal timer channels get synchronized to the CSAC 1PPS signal.* ***b)*** *The CSAC 1PPS signal get synchronized to the GPS 1PPS signal.* ***c)*** *The Timer Counter channels are resynchronized to the CSAC 1PPS signal.* ***d)*** *Whole sync process complete, RTC has been updated with absolute UTC time from GPS.* ***e)*** *Node is deployed underwater*

4.7 Timer Counter

The CSAC outputs timepulse signals, the 10 MHz clock and the 1PPS signal. To be able to generate readable time data for the host processor, an interface to count and convert these timepulse signals is needed. Connecting the built-in *Timer Counter* module in the MCU to the CSAC 10 MHz clock and 1PPS signal makes it possible to keep track of time in seconds and 100 ns resolution. One of the hardware *Timer Counter* modules in the SAM3X MCU is utilized for the RTC and timestamp functionality.

One *Timer Counter* module, or block, contains three separate counters that can be configured and clocked independently. A *Timer Counter* block is illustrated in Figure 4.5. The *Timer Counter* block offers flexibility in clock selection and can use external clock sources. Each channel can also be configured for time capturing, based on edge detection of the TIOA input pin, where the current counter value gets stored in a register upon edge detection on the input pin. A compare register is also available to configure the timer channels to act when the current counter value is equal to the value in the compare register. All the mentioned events can be configured to trigger a reset of the counter, which sets the counter value to 0, or trigger the respective ISR handler for the channel. A global synchronization signal (SYNC) can be used to reset all the counters at once.

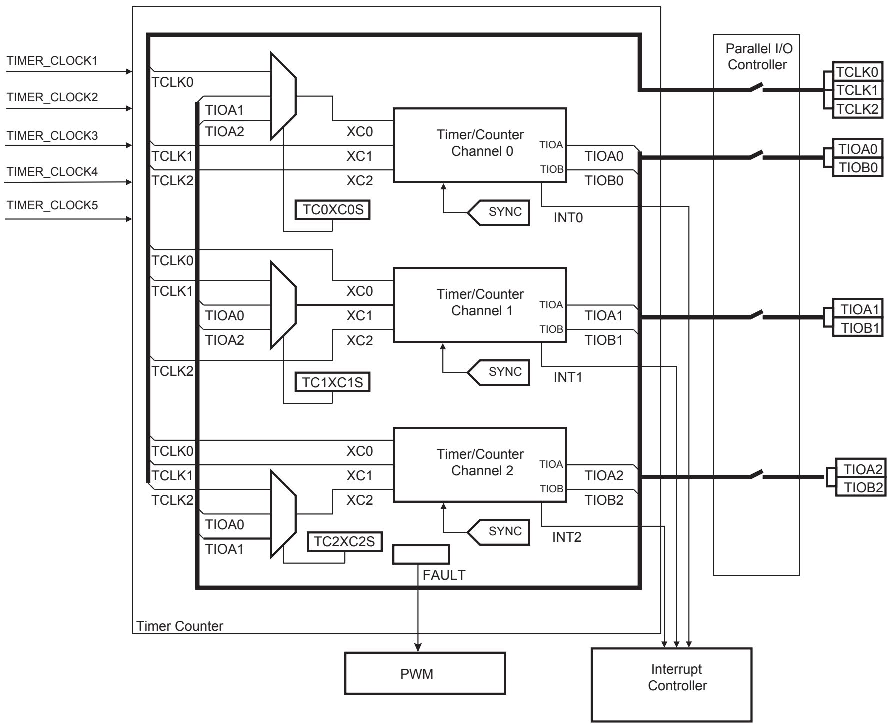

Figure 4.5 SAM3X Timer Counter block with three independent counter channels. (Source: Atmel SAM3X datasheet [26])

4.7.1 Synchronous edge detection and clocking of MCU

The CSAC board prototype was routed so one could choose either a regular crystal oscillator or the CSAC as clock source for the MCU. Initially it seemed beneficial to clock it from a regular crystal oscillator, at least from development perspective, since the core circuit would work without the more expensive CSAC component. Unfortunately, the *Timer Counter* external clock input is not strictly asynchronous and has a synchronous edge detection circuit, consequently, using the crystal oscillator resulted in jitter of $\pm$ one 100 ns pulse in the *Timer Counter* circuit. The CSAC was therefore used as the clock source for the MCU.

4.7.2 Timer Counter configuration

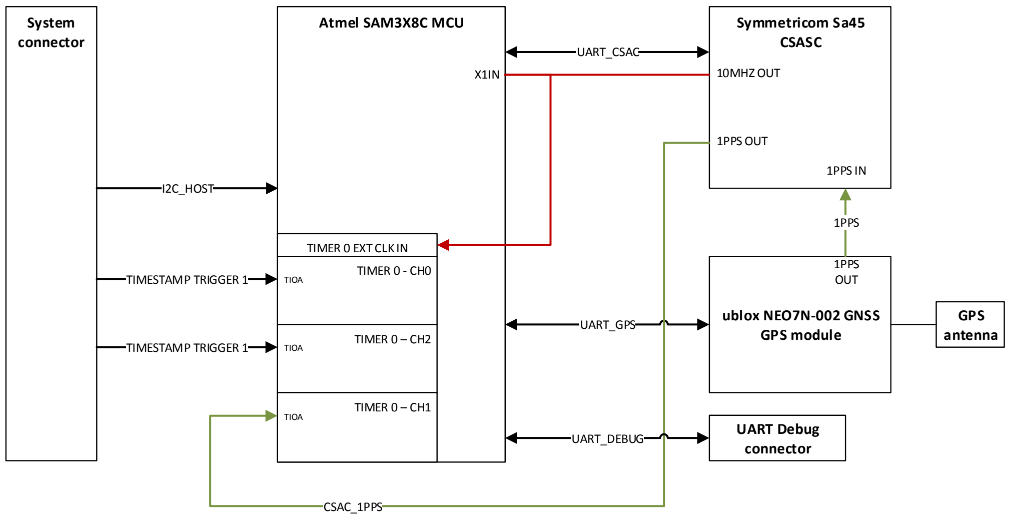

Figure 4.6 Simplified block diagram of CSAC board

Figure 4.6 shows how the different electrical signals are connected to the *Timer Counter* block. All Timer Counter channels are clocked by the external 10 MHz CSAC clock signal. Each channel resets to zero when its counter reaches a compare value of 10 million, resulting in three independent high-resolution counters (100 ns) that reset every second. Channel 1 is used as a 1 Hz source for the software RTC and has the CSAC 1PPS signal connected to its TIOA capture pin. When the the counter reaches the compare value, an interrupt is generated which is used to update the RTC time value and signal other periodic 1 Hz software tasks. The term *1 Hz RTC tick* is used for this interrupt source and *RTC ISR* is used as a name for this specific ISR throughout the rest of the report. Channel 0 and 2 is used for generating high-resolution timestamps, and have external timestamp trigger signals connected to its TIOA input pins, generating interrupts. The RTC and timestamp functionality are described in more detail in section 4.8 and 4.9.

4.7.3 Timer Counter channel synchronization

After the *Timer Counter* block has been configured as part of the initialization of the MCU, the three counters start up running at different moments in time, and run out of sync, both in respect to each other, and in respect to the CSAC 1PPS signal. The synchronization of the counter channels includes two tasks; the first task is to synchronize the three counters so they reset at the same time, the second task is to synchronize them to the external CSAC 1PPS signal. After successful synchronization, all three hardware counters count in sync with the CSAC 1PPS signal.

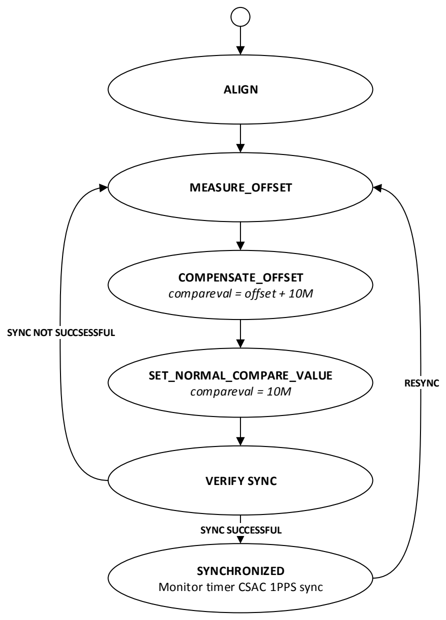

Figure 4.7 States in state machine for synchronization of Timer Counter channels to CSAC 1PPS

A state machine controls the synchronization in several steps, as illustrated in Figure 4.7. The state machine is executed once per timer period (Every second, except for in the *COMPENSATE_OFFSET* state, where the period is extended with 0-1 seconds). As a first step, the *SYNC* signal is triggered from software, resetting the three counters, making them reset at the same time and count in sync. In the second step *MEASURE_OFFSET*, the capture pin on channel 1 is used to measure the time offset to the CSAC 1PPS pulse. For one period, in this step only, the compare value is increased by this offset value. In the following step, the compare value is reset back to 10 million, and the counters are now synchronized with the CSAC 1PPS signal. The synchronization is verified by measuring the offset again in the last step, before entering normal state. In the normal state, as part of the system monitoring, the offset is measured to be able to identify loss of synchronization. Figure 4.8 illustrates the synchronization process.

As an alternative to using the compare value to reset the counters once per second, edge detection on the CSAC 1PPS signal could have been used directly to reset counter channel 1. It was found to be difficult to synchronize the three timers using this approach, since the *SYNC* signal had to be signalled from inside the ISR handler of channel 1, inducing a delay of the synchronization signal. Wiring the CSAC 1PPS signal to each counter channel and using the same approach of resetting was not an option, because the 1PPS signal would occupy the input capture pin needed for timestamp triggering.

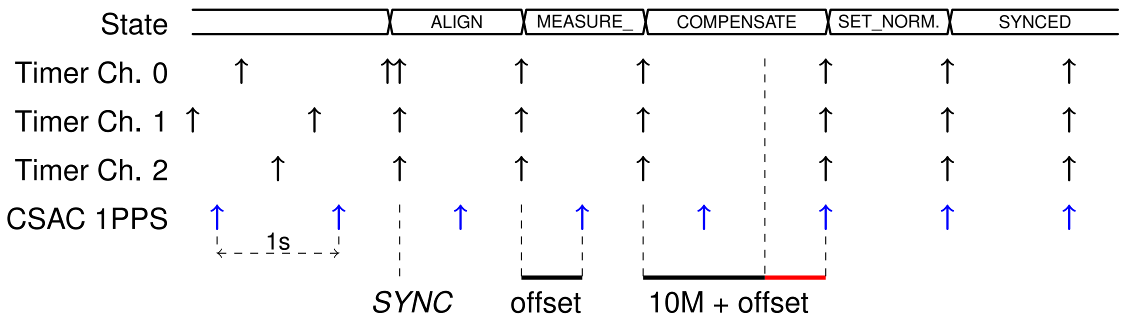

Figure 4.8 Synchronization of Timer Counter channels to CSAC 1PPS pulse. The three counter channels start up out of sync initially, before they are aligned using the SYNC signal in the ALIGN state. After that, the offset between the timers (black arrows) and the CSAC 1PPS signal (blue arrows) is measured in the MEASURE_OFFSET state. In the COMPENSATE_OFFSET state, the normal compare value of 10 M is extended with the offset value, for this period only. In the subsequent state SET_NORMAL_COMPARE_VALUE, the compare value is set back to 10 M. After the VERIFY state (not shown), the counter channels are synchronized as indicated by the SYNCED state.

4.8 Real-Time Clock

The CSAC board provides a one-second resolution RTC synchronous with absolute UTC time, retrieved from GPS. It is implemented as a 32-bit software counter and holds the time in UNIX time format. The counter is incremented in the RTC ISR, triggered by the 1 Hz RTC tick (Timer Counter Channel 1).

4.8.1 Synchronization of RTC to absolute UTC time using GPS

Right after the RTC module has been initialized, it starts counting from a predefined time value, defined at compile time. If it is impossible to establish GPS fix, the host can choose to write a local time to the RTC module, ignoring further synchronization to the GPS. Two main operations are needed to sync the RTC with GPS UTC time. Firstly, the CSAC 1PPS pulse needs to be synchronized with the GPS 1PPS timepulse, secondly the RTC time value must be updated with the absolute UTC time value. These two operations are controlled by the main state machine and correspond to the states *SYNCING_CSAC_1PPS_TO_GPS_1PPS* and *SYNCING_RTC_TO_UTC_TIME*.

Syncing the CSAC 1PPS signal to the GPS 1PPS timepulse is done by utilizing the built-in feature in CSAC which enables to sync the CSAC 1PPS output pulse with an external 1PPS signal. The GPS timepulse output is used as synchronization source and is configured as 1PPS. The GPS timepulse is active when valid GPS fix is established. The GPS fix status is continuously monitored, and when fix is established, the main synchronization state machine issues the synchronization command to the CSAC. The CSAC moves the 1PPS output pulse to the 10 MHz pulse that is closest to the input GPS 1PPS pulse, syncing the CSAC to ± 100 ns of the GPS 1PPS pulse as shown in Figure 4.9 and 4.10. Notice that the 1PPS signal is still synchronous with its 10 MHz clock signal as shown in

Figure 4.10.

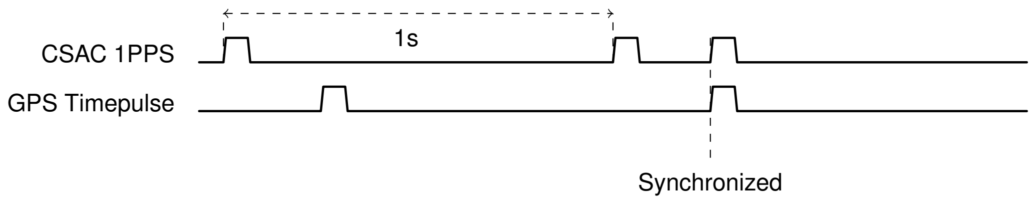

Figure 4.9   CSAC 1PPS signal get synchronized to GPS 1PPS timepulse

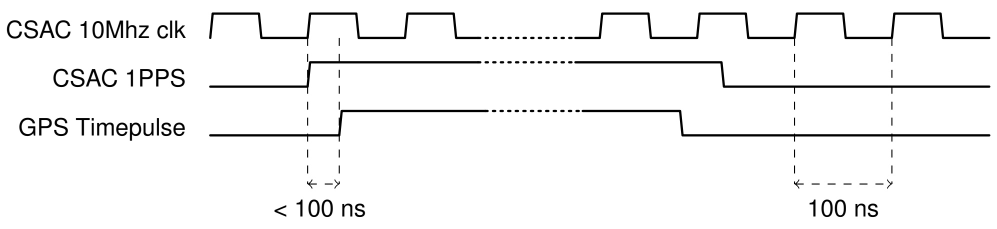

Figure 4.10   CSAC 1PPS synchronized to the GPS 1PPS timepulse within $\pm 100ns$*. Both the 1PPS signals are high for several 10 MHz clock cycles.*

After syncing the CSAC 1PPS signal and re-syncing the counter channels, the counters count seconds in sync with the GPS, but the RTC time value is still not updated with respect to of the absolute UTC time. The GPS module can be configured to send supporting timepulse data on the serial bus. With this option enabled, the GPS module sends a timepulse message periodically, one message per timepulse as illustrated in Figure 4.11. Each message contains the time value for the last timepulse, and this time value is used to update the RTC. In the state *SYNCING_RTC_TO_UTC* in the main state machine, this message is enabled. The timepulse message is formatted in GPS time format and is therefore converted to UTC before it is used to update the RTC. Please refer back to Figure 4.4 to see this operation as the last of all the synchronization steps. Leap seconds handling and the GPS time to UTC conversion is described in Section 4.8.2.

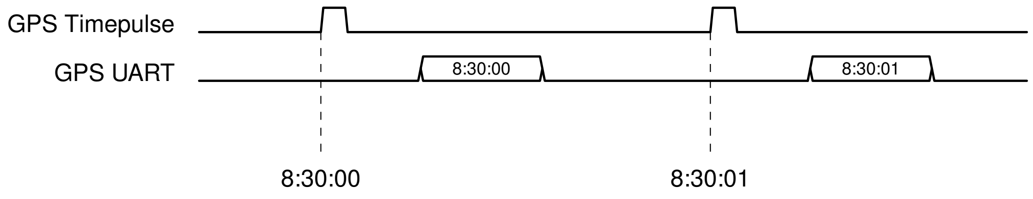

Figure 4.11   GPS timepulse supported by timepulse message containing time value for last timepulse

4.8.2 Leap second handling

Using UTC as a time base for the RTC introduces challenges due to the leap seconds as explained in Section 3.6.5. The conversion from GPS time to UTC is straight forward, as the number of leap seconds can be extracted from the GPS receiver. However, the handling of leap seconds during a leap second update is more challenging. Three aspects are important to look at when choosing support for leap seconds. First of all, the handling of the update of a new leap second requires knowledge of how the host system handles leap seconds. Is it for example okay to halt the time value for one second when a positive leap second is introduced? The second aspect is how the leap second update should be handled if the update should happen after the system is deployed underwater, and what if only some of the nodes have knowledge about the upcoming leap second update. Just getting predictable information about future leap seconds updates is a big challenge. The third aspect is that the host applications that are most depended on correct timekeeping, are data processing applications, and would probably benefit of a linear time scale, over keeping absolute sync with UTC over a leap second update.

With this in mind, to not support dynamic leap second updates is clearly a justifiable solution. When the RTC is synchronized to GPS, it is still getting absolute UTC time (including leap seconds), but after synchronization it ignores the possibility of leap second updates (this makes sense since the system is to be deployed underwater soon after synchronization anyway). A challenge with this implementations is that different NILUS nodes could be deployed on each side of a leap second update, resulting in a one second time offset between nodes in the network. The operators of the system must take care of not deploying (turn on and synchronize) nodes when a leap second is to be introduced. Another suggested solution was to have the leap second offset as a compile time constant in the firmware, but this solution was scrapped since it would demand a full firmware update of all nodes, to keep the system updated after a leap second introduction.

4.8.3 GPS time to UNIX formatted UTC time conversion

The conversion from GPS time to UTC time, formatted in UNIX time format (seconds since 1. Jan 1970), follows Equation 4.1, all values are in seconds. Since the GPS epoch is defined as 6. Jan 1980, ten years later than UNIX epoch, the difference between between the epochs must be added to the reported GPS time. Positive leap seconds after the GPS time epoch must be subtracted (leap seconds before GPS epoch is per definition included in the GPS time epoch).

$$UTC\_unix = Diff\_epochs - Leap\_seconds\_since\_GPS\_epoch + GPS\_time \tag{4.1}$$

To use the above equation, the current GPS time must first be converted from the GPS time format (Week number and *Time of Week*) to seconds. Note that the u-blox NEO7N is capable of reporting UTC time directly, but using GPS time makes the conversion to UNIX time format easier, and leaves the leap second handling to the MCU firmware. GPS time is also the time format enabled by default for the time pulse message.

4.9 Timestamp generation

After the host processor enables the timestamp function for the specific timestamp channel, and a timestamp trigger signal is received on the CSAC board, the time of the event is stored and made available to the host. The time of the event is stored as two 32-bit values. The first value contains the high-resolution (100 ns) counter value, and the other contains the seconds-resolution RTC time value.

Upon a signal edge detection on the enabled timestamp trigger channel, the current counter value gets immediately stored in a hardware register and an interrupt is generated. Since the timestamp trigger signals are one-shot, buffering of incoming timestamps is not needed.

4.9.1 Mapping of RTC time to timestamps

In the generation of a timestamp, the high-resolution counter value needs to be mapped to the correct seconds value, generating a complete timestamp with both the counter value and the RTC time value. The task of storing the high-resolution counter value is offloaded by the *Timer Counter* module hardware and can be read at a later point in time as long as no other trigger overwrites it. However, it is critical that the RTC value does not get updated in-between a timestamp trigger and the timestamp mapping function. A best case example where the timestamp trigger arrives after the RTC time value has been updated is illustrated in Figure 4.12. Here the timestamp ISR can read the correct RTC time value directly.

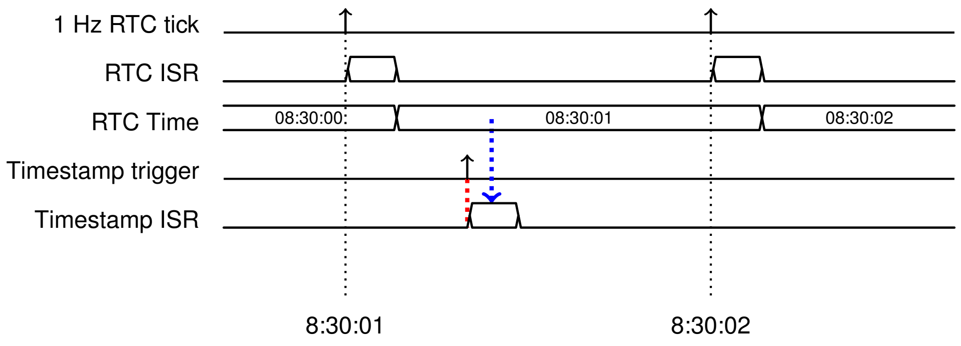

Figure 4.12 Timestamp generation, mapping of RTC time value in a best case situation where the trigger arrives after the RTC time value has been updated and there are several hundred milliseconds until next update.(Note that the execution time of the ISRs are not to scale compared with the 1 Hz RTC tick)

Since the RTC is implemented in software and is handled in the ISR of the 1 Hz RTC tick, the update of the RTC time value is somewhat delayed due to the context switch and execution of the ISR. This means that the timestamp trigger can occur right after a RTC tick but before the available time value in the RTC is actually updated, resulting in reading the old seconds value. This is a special case of race condition that is solved by setting up the NVIC interrupt priority levels in the SAM3X MCU, so that mutual exclusion is provided and the order of execution for the RTC ISR and timestamp ISR is guaranteed. To ensure that the RTC is updated with as little latency as possible, and that no other task in the system can preempt the task, the RTC ISR is given the highest priority in the whole system. For a read of the RTC time value in software, where one is only interested in seconds-resolution, the accuracy is only affected by the short time skew due to the RTC ISR execution, which is acceptable (the timestamp functionality should be used for sub-second resolution).

By setting the preemptive interrupt level, *group priority level*, of the RTC and timestamp trigger equal but higher than all the other interrupt sources in the system, one ensures that the RTC and timestamp task are executed with as low latency as possible. As discussed in Section 4.5, the I2C interrupt also needs to have the same group-priority level as the RTC and timestamp interrupt, but are given the lowest non-preemptive priority (subpriority level), ensuring that it does not interfere with the time critical execution of the RTC and timestamp ISRs. To ensure that the RTC ISR is always executed consistently right after the 1 Hz RTC tick, with as little latency as possible, the RTC interrupt is given a higher *subpriority level* than the timestamp interrupts. If the timestamp trigger arrives at the same instant as the 1 Hz RTC tick, the RTC ISR gets rightfully executed first, updates the RTC time value, and completes before the timestamp ISR is executed. This is illustrated in Figure 4.13.

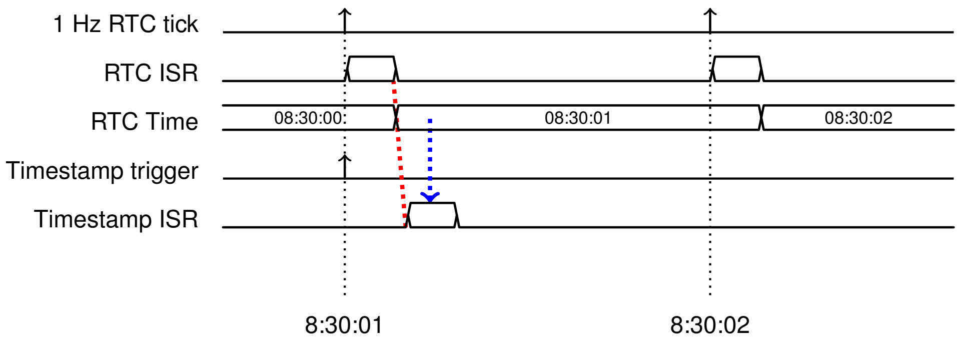

Figure 4.13 Timestamp generation when trigger arrives at the same instant as the 1 Hz RTC tick. Both ISRs have the same group priority level, but the RTC ISR gets handled first because it has higher subpriority level, ensuring the order of execution. The timestamp ISR reads the correct updated RTC time value. (Note that the execution time of the ISRs are not to scale compared with the 1 Hz RTC tick)

The same behaviour applies for scenarios where the timestamp trigger arrives while the RTC ISR is executed and the new RTC time value is not yet updated, as illustrated in Figure 4.14. But now it is the mutual exclusion in form of equal *group priority level* that makes sure the RTC value get updated before the timestamp ISR is executed.

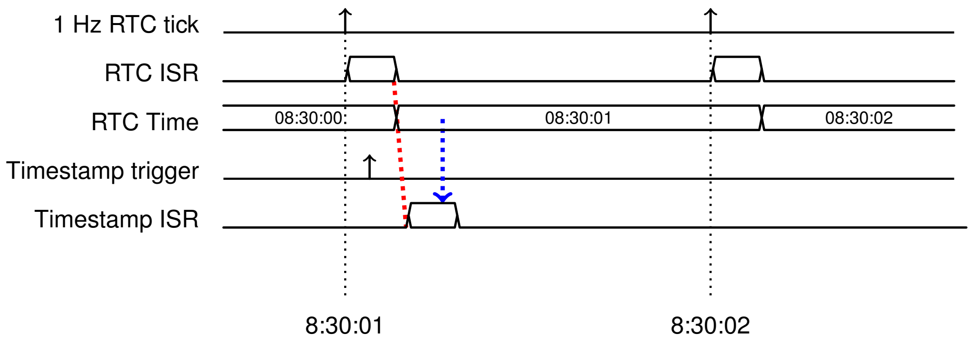

Figure 4.14 Timestamp generation when trigger arrives while RTC ISR is being executed. The equal priotity timestamp ISR cannot preempt the RTC ISR. After the RTC ISR is completed, the timestamp ISR is executed (red line), and reads the updated and correct RTC time value (blue line). (Note that the execution time of the ISRs are not to scale compared with the 1 Hz RTC tick)

Another corner case is when the timestamp trigger arrives just before the 1 Hz RTC tick, or put another way, the 1 Hz RTC tick arrives while the timestamp ISR is being executed. This is illustrated in Figure 4.15, where one can observe that the RTC ISR is forced to wait with updating the new time value until the timestamp ISR has completed. This result in a negligible skew of the update of the RTC time value due to the short execution time of the timestamp ISR.

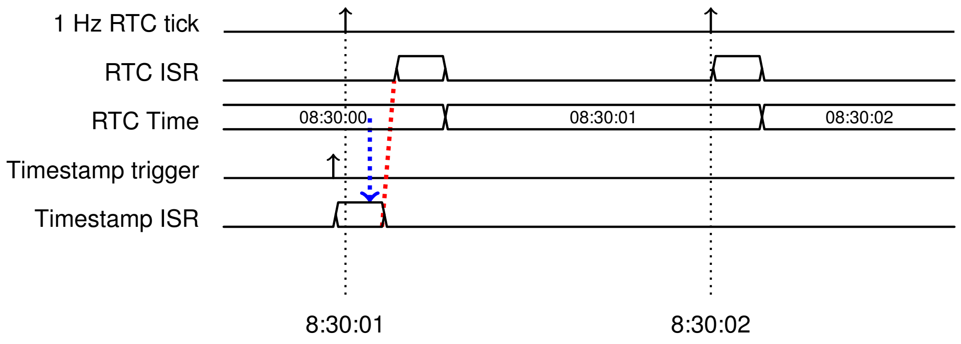

Figure 4.15 Timestamp generation when trigger arrives right before the 1 Hz RTC tick. The timestamp ISR cannot be preempted by the equal priority RTC ISR, and is guaranteed to read out the RTC time value (blue line) before it get incremented. After the timestamp ISR is completed, the RTC ISR is executed (red line), and the RTC time value gets updated. (Note that the execution time of the ISRs are not to scale compared with the 1 Hz RTC tick)

To summarize, being able to both configure preemptive and non-preemptive interrupt levels, makes it possible to both ensure mutual exclusion and guarantee the execution order. Equal preemptive priority level, *group priority level*, ensures mutual exclusion since the two ISRs cannot preempt each other. The order of execution is ensured by the arrival of the interrupts and the second priority level, *subpriority level*. The latency critical constraint of handling the 1 Hz RTC tick is solved by giving the two tasks the highest *group priority level* in the system. With the relative short execution time for both ISRs, there is little danger that they will interfere with other interrupt driven tasks.

5 Measurements and verification

The main goals of the final measurements and verification, are to ensure that the system meets the original requirements, and that the system behaviour corresponds to the initial system design. The following chapter describes various conducted tests and measurements, as well as their results.

5.1 General development, test equipment and methods

Various types of tests and verification were needed for the CSAC board development. Some tests concerned the detailed timing performance at nanosecond resolution, while others were dependent on continuous logging for several days or weeks. A challenge to this was that the long time testing could not interfere with the ongoing development. Firmware development is typically associated with rapid iterations and testing on the actual hardware. Thus, waiting for week long measurements to complete before the latest bug-fix could be tested and verified, was prone to a cumbersome and much delayed development. Fortunately, several CSAC board PCBs and CSACs were provided by FFI. Two fully equipped CSAC boards with CSAC and recovery battery for the GPS receiver were available. A third CSAC board without CSAC, and a CSAC evaluation kit, were also available. It proved to be essential to have this variety in test boards, not only for scheduling different test and development tasks, but also to tailor each test board for different kind of probing. For example, test probes for all the most critical signals were attached semi-permanently to one of the test boards. A 100 MHz four-channels oscilloscope, a 16-channel sampling logic analyser, and digital multimeters, were used regularly during the development and testing.

For long time logging, a development computer that could be left turned on for long periods was used, logging comma separated value (CSV) data to file from the debug terminal. Matlab scripts were written to separate long time logging information from normal event logging and printouts from the debug menu system. This allowed for control and check of the CSAC board, while an independent long time test was being conducted and logged. Because the system's processing headroom was quite high, due to the whole second interval between regular events and short tasks, it was in general no problem to be generous with logging debug information.

5.2 Lab environment and GPS coverage

Early in the project, an unplanned problem regarding the lab environment was identified, which could have had serious impact on the planned execution of the project. All the major development was done from a personal lab at home, and when the initial bring-up of the GPS receiver progressed, it was revealed that the GPS coverage in the lab location was not ideal. The building is surrounded by tall neighbouring blocks, and from the window post where the test boards were placed, the clear line-of-sight of open sky was very limited. At first, the GPS coverage was so poor that it seemed like it would be impossible to establish GPS fix at all in the lab. It did not help that the GPS antenna was a small PCB mounted version. It was necessary to consider the worst scenario and plan for relocation of the lab, or to find a GPS repeater or much better antenna. A short test was conducted on the top-floor balcony, where the view to open sky was much better, and the received GPS signal level was much stronger from more satellites than in the lab. The time-to-first fix was also reduced and the reported position and time uncertainties were lower. After some more experimentation in the lab, by tilting the GPS antennas pointing directly to the sky, and by leaving the CSAC board, with the GPS receivers, always turned on for maintaining fix and satellite orbit information, the situation seemed to improve over time. During a full day, the signal would drop out several times for some hours, to later on reappear. This happened at approximately the same times of day, and was clearly due to the satellites' placement in the viewable sky. Since many of the implementation tasks did not rely on GPS fix, the initial development continued. As time went by, and constant GPS fix was needed, the situation had improved from having a steady GPS fix some small hours a few times per day, to dropping the GPS fix some small hours a few times a day. This could possibly be explained by seasonal changes and improved weather. It was decided that the lab location was sufficient to continue the development and testing without relocating, and tests that relied on better GPS signal could be done from the balcony.

Having varying and somewhat unstable GPS coverage was actually a good thing for the major parts of the development and testing. Since a lot of the effort went into implementing and testing the main state machine and the low level drivers, stimulating state transitions and system behaviour due to lost CSAC lock or lost GPS fix was much easier. On the other hand, having such varying signals was a big problem for the testing of the disciplining process, as discussed in Section 5.8.

5.3 Verification of timestamp generation

To ensure that the timestamp generation works as intended, it had to be verified and tested properly. It was important to show that the stored high resolution counter value corresponded to the actual time difference from the 1PPS reference, and that the high resolution counter value got mapped to the correct RTC seconds value as discussed in Section 4.9.1. It was especially interesting to test the corner cases around where the RTC timer counter channels would overflow and the RTC ISR got executed. This test would also verify that the three timer counter channels were running in sync.

5.3.1 Test setup

A signal generator that could stimulate the timestamp trigger input was needed. To test the corner cases, where the trigger signal edge arrives close to the zero point of the RTC, it was required to place the trigger signal at a known offset relative to the CSAC 1PPS signal. With the known offset, one could verify the timestamp mapping by comparing the reported timestamp with the input stimuli. By probing the 1PPS reference pulse, the input trigger signal, the debug UART communication (where the final timestamp got reported), and the different ISR's, with a logic analyzer, a full overview of the timestamp generation was available.

The easiest way to create a pulse generator that would allow a flexible and predictable trigger pulse was to use a field-programmable gate array (FPGA), and use this to generate all the clock signals that CSAC normally would. An Altera FPGA based Terasic DE0-nano development board was used to replace the CSAC as whole [27]. The FPGA generated a fast clock, replacing the 10 MHz CSAC clock, the CSAC 1PPS signal and the timestamp trigger signal. A block diagram of the test setup is illustrated in Figure 5.1.

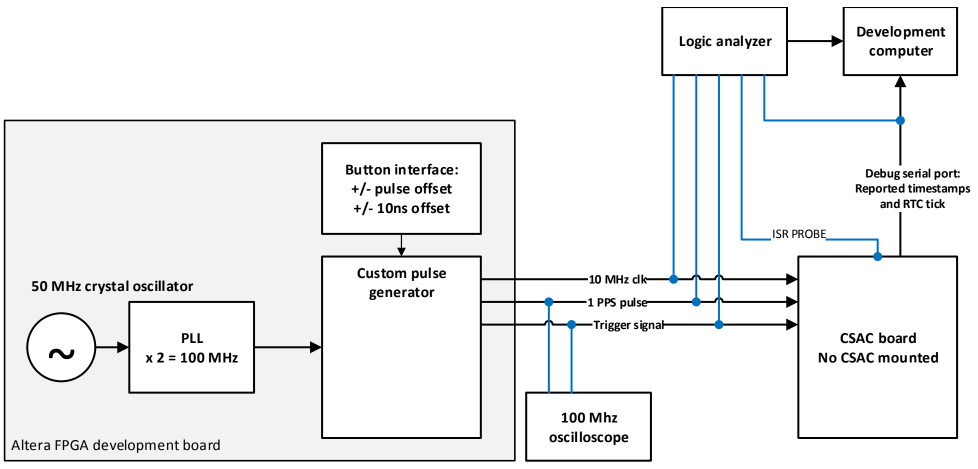
Logic analyzer
Development computer
Button interface: +/- pulse offset +/- 10ns offset
Debug serial port: Reported timestamps and RTC tick
ISR PROBE
Custom pulse generator
50 MHz crystal oscillator
10 MHz clk
1 PPS pulse
Trigger signal
PLL x 2 = 100 MHz
CSAC board No CSAC mounted
100 Mhz oscilloscope
Altera FPGA development board

Figure 5.1 Block diagram of the test setup for verification of timestamp generation. A custom FPGA based pulse generator replaces the CSAC clock signals and generates the trigger signal. A known trigger offset can be set with a resolution of 10 ns and be verified with an oscilloscope. The resulting timestamp can be monitored on the debug terminal, and the internal behaviour and mapping of the RTC seconds value is verified by sampling the various signals with a logic analyser.

The FPGA design included an off-chip 50 MHz crystal oscillator, an on-chip phase-locked loop (PLL), a custom pulse generator, and a user interface with buttons and LEDs to control the trigger signal offset. Since it was needed to be able to adjust the trigger signal offset with greater resolution than a whole 10 MHz clock period, the PLL was configured to output 100 MHz as a base clock for the time pulse generator. With the 100 MHz base clock, the timestamp signal offset could be adjusted at both 10 ns and 100 ns resolution. The offset between the timestamp signal and the 1PPS signal was monitored on a 100 MHz oscilloscope, and the generated timestamp could be read on the debug terminal. By capturing all the signals with the logic analyzer, the RTC second mapping could be confirmed.

5.3.2 Results

Already before the test was conducted, the timestamp counters had shown a strange behaviour. When drift measurements were conducted earlier, the GPS 1PPS signal had been connected to the timestamp input. Since the GPS signal was centred around the CSAC zero point after sync, but slowly drifted, the timer counter values could be monitored live on the debug terminal. The confusing part was that the counter never reported a value of zero, but could reach a value of 10 million. (A 10 million step clock value should count from 0 to 9’999’999). This behaviour was examined and verified during this test. A closer look in the Atmel Timer Counter module’s datasheet, showed that this was normal behaviour, and that even if the counter circuit got reset, it would not propagate to the counter value until the following clock cycle. This was corrected in the timestamp ISR by checking for values equal to (and above) 10 million, and replacing them with zero.

The general timestamp generation, including mapping the timestamp to the right RTC second value, was verified as correct and stable, and the ISRs triggered as intended around the corner cases.

5.4 Verification of synchronization sequence

The general test setup for testing the synchronization sequence consisted of a logic analyzer to probe the various time signals, and serial communication from the GPS receiver and logging system.

5.4.1 Timer synchronization and CSAC 1PPS synchronization

The timer counter system and its synchronization to the CSAC 1PPS signal is monitored by its built in state machine. During the development it was shown that the main state machine worked as intended and that the system only reported successful synchronization when it was achieved. Each stage of the timer counter synchronization, and the steps in the main state machine, are logged to the debug terminal under normal operation, and could be correlated with the electrical time signals, by capturing both the serial communication and the signals.

The captured data verified that the synchronization worked, and was executed at the right step in the main state machine. The system has also been reset numerous times during the last months of development, and the state machine behaves as intended. The initial timer counter alignment is executed first, followed by the CSAC heat-up sequence and GPS fix acquisition. The final synchronization to GPS is first executed when both CSAC is locked and GPS fix is acquired, as intended.

5.4.2 RTC synchronization to UTC

The RTC synchronization was tested to verify that the RTC value got synchronized to GPS and updated with the correct value. Here, the firmware could, by bad design or error, introduce a full second error. By looking at the captured data of the reported GPS time pulse message from the GPS, the reported internal RTC value and the corresponding time pulses, the synchronization process was verified to work as intended.

5.4.3 State machine behaviour when no GPS signal

One benefit of having bad GPS coverage at the lab location, was that behaviour under non-ideal GPS conditions could be monitored regularly. This was useful to cover different state transitions in the main state machine. It was also used to simulate submerged operation of the system. It was shown that the state machine did not proceed to the initial synchronization before proper GPS fix was acquired, this was as intended. If the system lost GPS signal after synchronization, similar to what would happen when the system is deployed underwater, the timer system and overall state machine maintained stable.

5.5 Initial time error after synchronization

Even if GPS is one of the best time sources available with respect to long-term stability, the short-time stability is not as great. Since the synchronization process should be completed as soon as possible, there is no available time for averaging out the short-term instabilities. In fact, in this system, averaging the incoming GPS timepulse would involve a full disciplining of the CSAC, since it is the CSAC that essentially needs to be synchronized perfectly. This could take from many hours up to days, depending on the quality of the GPS reception. As a consequence, the synchronization to the GPS time pulse signal is instantaneous, and the short-time uncertainties in the GPS signal, and in the CSAC 1PPS synchronization process, would propagate to the CSAC board's timekeeping system. This would lead to an initial finite time error between the specific NILUS node and absolute time, as well as relatively between the nodes in the NILUS network.

The main sources for the initial time error, are the uncertainty in the GPS time pulse signal (30 ns RMS, 60 ns at the 99-percentile), and the quantization error from the CSAC 1PPS synchronization process (±100 ns). One could also consider the propagation delay through the circuit and timer module, but this contribution should be comparably low, as with the circuit to circuit variance, and would therefore not contribute much to relative time error in the network.

Measuring the absolute time error was not done, since it required access to a reference time standard with good short-term stability. One could have used the CSAC board itself to retrieve an estimate of absolute time by averaging over long time (as shown in Section 5.6), but having the board powered on for a long time contradicted with the need for frequently restarting the board.

On the other hand, tests of relative synchronization between two nodes were conducted sporadically during the last period of development. The test was done by powering up two independent CSAC boards at the same time, and let them complete the synchronization process. Then a common trigger signal was used to trigger timestamps on both units. The results was as expected, the relative time error laid within ±200 ns. To get some trustworthy statistics, it would have been beneficial to redo this test more systematically and thoroughly.

5.6 CSAC time drift and stability

As mentioned, the initial time error from the synchronization process should be very small. It is the long-term clock stability and frequency deviation that would probably cause synchronization problems in the long run. A constant frequency deviation between nodes would result in a linear time drift, and the time error would accumulate. This would lead to both an absolute time error and relative time error in the network. To gain better insight about CSAC's performance and its ability to keep the network synchronized over time, the frequency deviation and clock stability of the CSAC was evaluated by conducting time drift measurements as described below.

Two timestamp channels on a CSAC board were used to measure instant time deviation. The

GPS timepulse signal and a second CSAC (CSAC#1) on another CSAC board were connected to individual timestamp channels. Relative time deviation between the two CSACs, and between GPS and CSAC#2, were sampled each second with a time deviation resolution of 100 ns. Any instabilities in the reference clock (CSAC#2) would influence the measurements. A sketch of the measurement setup is shown in Figure 5.2. None of the CSACs had been disciplined in advance of this measurement, so the measurement illustrates the out-of-the-box performance, with some possible shelf-time and ageing. Notice that even if a CSAC is used as time reference in this specific measurement, the ultimate reference for absolute time and long-term-stability, is the GPS time pulse.

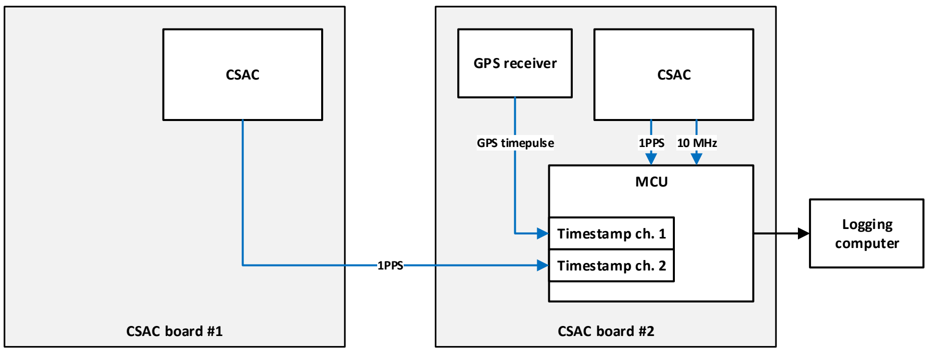

Figure 5.2 Two timestamp channels on a CSAC board (#2) are used to conduct long time measurements of time deviation of GPS timepulse and another CSAC (#1).

The first 37 hours of the measurement is illustrated Figure 5.3. Time deviation between the GPS timepulse and CSAC#2 is shown in blue, while the red line shows the time deviation between CSAC#1 and CSAC#2. There are apparently some time drift, since the time deviation change over time. Both curves approximate a straight line, which can be seen easier by fitting a linear model to both curves. The slope of the time drift curve indicates the frequency deviation, and linear time drift indicates a constant frequency deviation. If we assume that the GPS has an ideal or much better accuracy and long-term stability, this constant frequency deviation represents the accuracy of the CSAC. The slope of the GPS curve is positive, which means that the GPS frequency is slightly lower than the frequency of the CSAC used as time reference.

Not surprisingly, the GPS curve is very linear, indicating good long-term stability. Remember that the curve also includes the instabilities from the CSAC used as time reference, and hence the CSAC's long-term stability also seems relatively good. Comparing the two curves, we see that the CSAC vs. CSAC curve has some slight fluctuations, and it is much smoother than the GPS curve. This corresponds well with the expectations of much better short-term stability in the CSAC than GPS, and the GPS having the best long-term stability.

The two curves can be studied closer and compared by removing the linear time drift. Figure 5.4 and Figure 5.5 show the residual time deviation after the linear time drift has been removed. The non-linear change in time deviation indicates frequency drift, and we can evaluate the clock stability graphically by looking at the noise characteristics in time domain. From comparing the two, we can clearly see that they have different noise characteristics. Common for them both is the slow

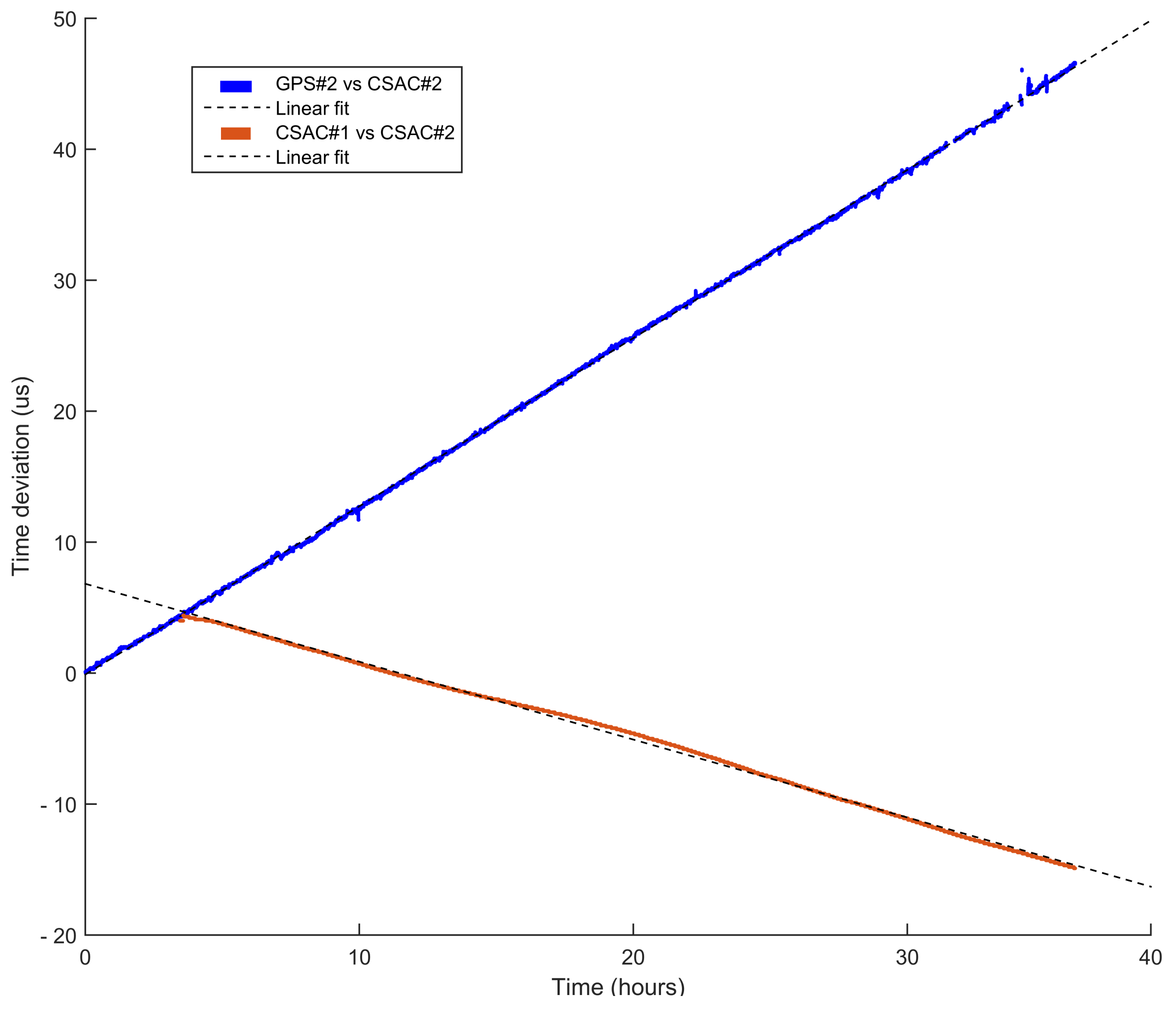

Figure 5.3 Measured time deviation between two CSACs (red) and between GPS timepulse and CSAC (blue). The approximately linear time drift means good long-term stability (constant frequency). The GPS has poorer short-term stability, but has better long-time stability (straighter line). The CSAC has better short-term stability, but has some slow fluctuations. The slope indicates constant frequency deviation. CSAC#1 was powered up and synchronized with GPS four hours after the start of the measurement. Note: The CSAC units were not disciplined prior to these measurements

periodical fluctuation, with its peak after 20 hours. This instability comes likely from the CSAC(s), but the source is unknown. It could be a form of beat frequency between the two CSACs, due to the very small frequency difference between them, but this does not explain why it also appears in the GPS vs CSAC data. The only other recognizable phenomena in the CSAC vs CSAC plot, is the ragged line due to quantization error in the timer counter. The short-term noise appears only in the GPS vs CSAC plot, and originates from the GPS. In this plot there is also some bigger peaks, which correlate to drops in GPS signal quality. A loss of GPS fix over several minutes caused the big peak at the end of the measurement.

Using the slope coefficient from the linear fit for GPS vs. CSAC, results in an average time drift of

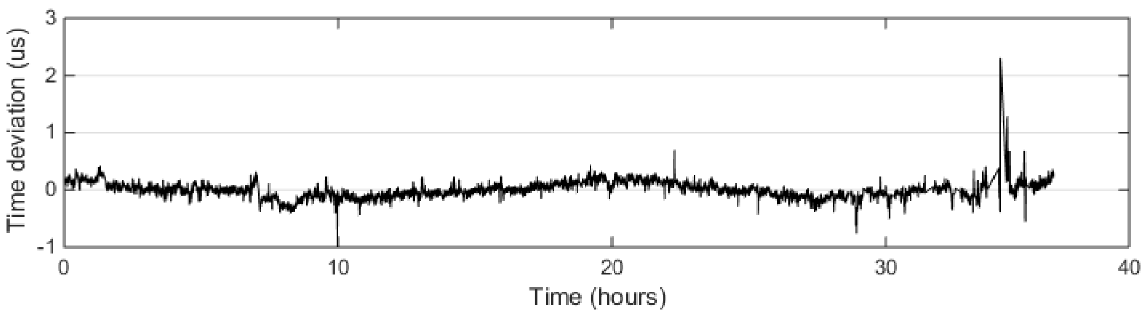

Figure 5.4 GPS vs. CSAC#2 residual time deviation after removing linear time drift. Short-term instability slow fluctuation. The bigger peaks corresponds with poorer GPS signal. The last peak corresponds with loss of GPS fix for several minutes. Note: The CSAC units were not disciplined prior to these measurements

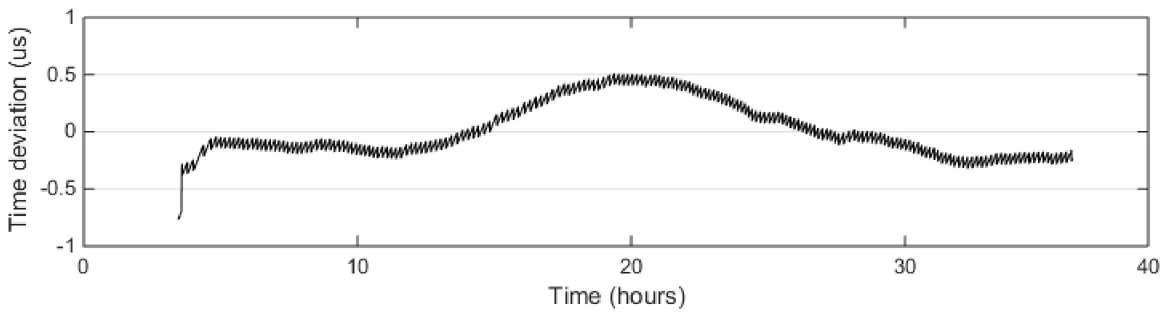

Figure 5.5 CSAC#1 vs. CSAC#2 residual time deviation after removing linear time drift. Slow, almost periodic fluctuations. The ragged line is due to quantization error in the timer. Note: The CSAC units were not disciplined prior to these measurements

3.6×10^{-10}, or 0.36 ppb, If we assume that the linear model is a good representation of absolute time, this value represent the absolute frequency accuracy of the CSAC, and is within the specified retrace accuracy of $\pm 5 \times 10^{-10}$ after being powered off for more than 48 hours [11]. With the same assumption, the absolute frequency of the CSAC would be 9’999’999.9964 MHz and have a frequency deviation of 3.6 mHz (Eq. 5.1 and Eq.5.2).

$$f_{CSAC} = \frac{10M}{1s + 3.6x10^{-10}s} = 9\,999\,999.9964\,\mathrm{MHz} \tag{5.1}$$

$$\Delta f = 3.6\,\mathrm{mHz} \tag{5.2}$$

It is not as simple to extract a numerical value for the clock stability, since it is more difficult to separate the instability sources on lower integration times. One of the most important observations from the measurement, is that the constant frequency deviation (linear slope) dominates the influence on time drift, over the influence from clock stability (fluctuations around slope). In other words, the accuracy is worse than the precision. If time drift must be further improved, it is the constant frequency deviation that should be prioritized. This is good news, because improving the precision is difficult, since it would involve replacing the clock with a more precise model. On the other hand, the accuracy can potentially be improved by disciplining the CSAC against GPS as an absolute reference. Since the system will be powered on and off between successive deployments, its the actual retrace behaviour combined with ageing that sets the limit for how often it is necessary to discipline the CSAC.

A practical example can illustrate better how these results could influence the performance of a synchronized system like NILUS. The relative frequency deviation between the two CSACs in Figure 5.3 was measured to be 1.7×10^{-10}, or 0.17 ppb. If the node has a 48 kHz sample clock for data acquisition, derived from the CSAC, the two separate sampling systems could with a linear drift of 0.17 ppb, stay in sync for over 16 hours, before the time drift is more than a whole sample period. If two perfectly accurate CSAC would start up with the opposite worst case frequency deviation due to retrace error (1 ppb relative), the 48 kHz sample clocks would drift a whole sample for every 5.8 hour (Eq. 5.3 to 5.5).

$$drift = 1.7 \times 10^{-10} t \tag{5.3}$$

$$drift = \frac{1}{f} = \frac{1}{48\,\mathrm{kHz}} 2.08 \times 10^{-5} s \tag{5.4}$$

$$t = \frac{1}{2.08 \times 10^{-5} s \times 1.7 \times 10^{-10}} \approx 5.8 hours \tag{5.5}$$

5.7 Crystal oscillator time drift and stability

A time drift measurement of a regular crystal oscillator was also conducted, using the same measurement setup as in Section 5.6. The time drift for the crystal oscillator vs. CSAC is plotted in Figure 5.6. Noice that the time deviation scale is now in milliseconds. Using linear fit, the approximate average time drift for the crystal oscillator was -1.8×10^{-6}, or -1.8 ppm, a factor 5000 worse than the CSAC. Figure 5.7 show the residual time drift after removing the average time drift from the linear fit. The scale is still in milliseconds, and there are clearly large changes in the rate of time drift, indicating frequency drift. Notice here that the frequency drift itself is much larger than the total time drift in the CSAC. These numbers are relative to the CSAC time scale and not the average of the GPS signal, but as seen in Figure 5.6, the difference between the GPS and CSAC is negligible on this scale.

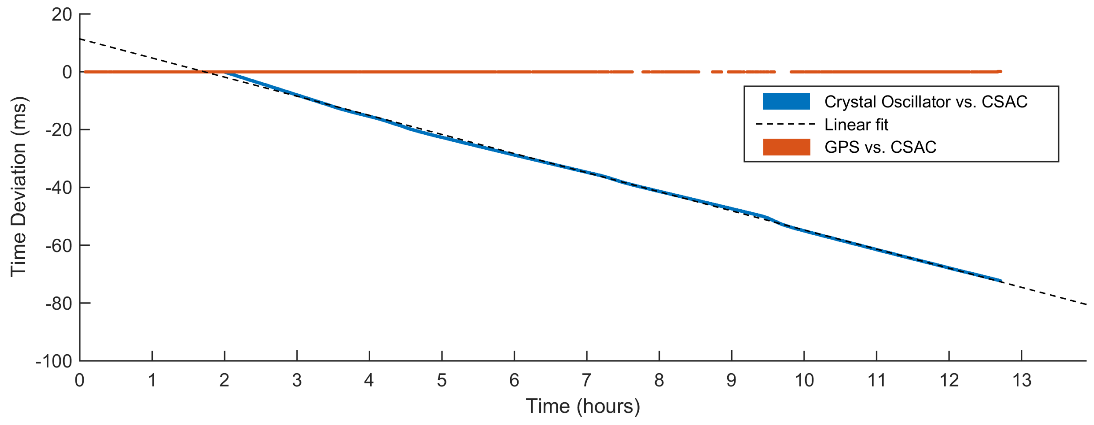

Figure 5.6 *Crystal oscillator vs. CSAC/GPS. The cystal oscillator has an average time drift of −1.8 ppm, a factor 5000 worse than the CSAC frequency accuracy. Time deviation scale is in milliseconds.*

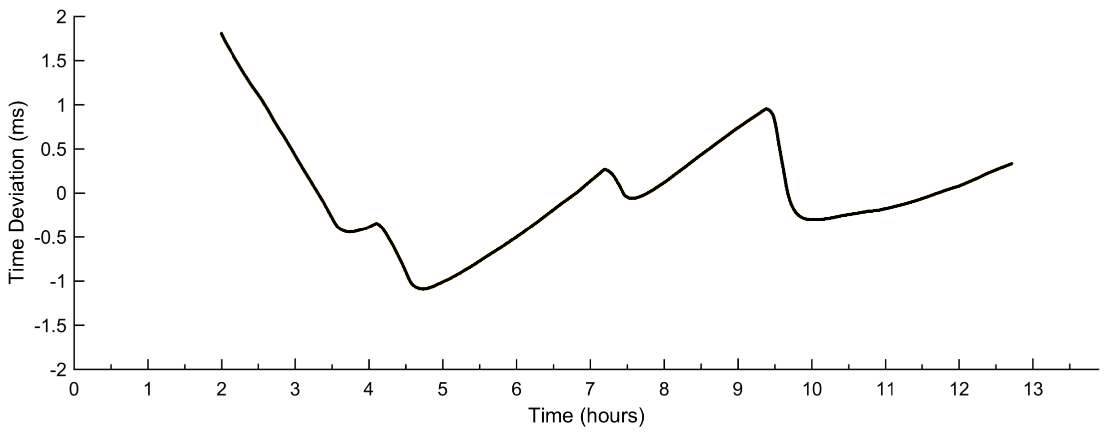

Figure 5.7 *Crystal oscillator vs. CSAC residual time drift after removing average time drift. Changes in time drift rate indicates frequency drift.*

5.8 CSAC disciplining against GPS

Test of disciplining of CSAC against GPS was conducted two times on two CSAC boards, but unfortunately none of the four disciplining attempts managed to achieve lock. The tests were manually stopped after five and seven days, respectively. The testing was conducted at the usual lab location, indoor with the CSAC board located by the window. 3000 s was used as time constant in the first run, and 5500 s in the second. Even if the phase error could stay low for long time, the GPS signal indoor was not stable enough to perform disciplining. This was not totally unexpected, but the initial plan to perform the disciplining test at the top floor balcony had to be abandoned due to weather conditions and lack of time.

5.9 System start up time for full synchronization

Knowledge about the system start up time is valuable in an operational context. Under good GPS conditions, the system should ideally synchronize as fast as possible, making it possible to power up the system right before deployment under water. The main contributors for the overall system start up time, i.e. from power-up to the full synchronization is completed, are the CSAC heat-up process, and the acquisition of GPS fix. Although the initial firmware process steps execute slowly, clocked by the 1 Hz tick, they contribute little to the total start up time, except when the system is rapidly restarted (CSAC is already heated, and the GPS receiver does a hot start).

It was shown that the CSAC heat-up process always finished within the specification (< 120 seconds), and usually within 60 seconds. When hot starting the GPS receiver, as was usually the case during daily development, it was shown that GPS fix acquisition could be completed before the CSAC was heated up. However, the typical scenario during a real deployment, would be that the GPS receiver does a cold start, but the GPS coverage is good (open sea). Unfortunately, even if the test is simple, it was difficult to conduct a realistic test as this repeatedly and thoroughly. The biggest challenges were that it was not practical to power off the system for four hours or more, or to disassemble the backup battery between each test, and the testing was also conflicting with other ongoing long-term testing or crucial debugging. Additionally, the test should have been conducted outdoors.

Even though the start up time is not tested in an environment similar to the operational environment, some conclusions can still be drawn. For a completely cold start of the whole system, the lower limit of the system's start up time will be in the order of tens of seconds to 2 minutes, dictated by the CSAC. The upper limit for the total time for full synchronization is infinite, and depends on the environment and GPS coverage for the specific occasion. For finding the typical start up time, the system needs to be tested more outdoors, ideally after integration with the full system to take into account the influence of the other system modules and the enclosure's potential attenuation of GPS signals.

6 Proposal for future work

6.1 Circuit changes

The initial hardware design opted for use of four individual timestamp trigger inputs, where two inputs where routed to one timer counter channel. The timer counter channel must be configured in capture mode, which only enabled one trigger input. The current hardware design therefore supports only two timestamp trigger inputs, and if more are needed, this must be considered in the next iteration of the CSAC board. Notice that this could require using a different MCU model.

During development, especially when performing long-term measurements, it was often very useful to measure the GPS time pulse signal by connecting it to one of the timestamp trigger inputs. If the next design iteration allows for more timer counter channels, connecting the GPS timepulse to a dedicated timestamp trigger should be considered.

Other modifications and suggestions for FFI are documented separately, outside this thesis.

6.2 Disciplining

The process of disciplining has to be tested further. Achieving phase lock should not be a problem, as long as the GPS coverage is good enough. After successful disciplining, the new frequency deviation should be measured and compared with the old value. There are also a couple of configuration in the GPS receiver that should be tested, which should improve the timing performance of the receiver, and thereby also the timepulse output. Additionally, some effort should be spent to find an optimal time constant for the disciplining process.

6.3 Frequency assisted GPS

It should be tested if there are any signification improvements in GPS fix acquisition or other measures, by using the CSAC clock signal to assist the GPS receiver with a a stable frequency.

6.4 Host side I2C driver

The complete, host side, I2C driver interface has not been developed, and needs to done.

7 Conclusions

To summarize the project as a whole, it has been shown that by using modern off-the-shelf components, like the CSAC and u-blox NEO7N GPS receiver, it is possible to develop a high performance, accurate and precise timekeeping system with relatively small development effort and low implementation complexity. As seen all over the integrated circuit industry, the evolution of integrated circuits and sensor systems leads to packing more and more advanced features into encapsulated hardware modules. The system developer's main task is more oriented around stitching together suitable modules. Although the abstracted features can seem simple and easy to comprehend, it is important to acknowledge the advanced underlying technology and requirements. The CSAC disciplining and proper use of the GPS receiver is a good example of this. Even though it is not difficult to utilize the right commands for controlling the disciplining process, to optimize and exploit the full potential of the system require better understanding through thoroughly testing and experimentation over time, possible outside the scope of this thesis.

The overall execution of the project was successful, with all of the high priority goals completed and verified. Not all medium priority deliveries were fully completed, for example the development of the host-side I2C driver had to be dropped. The system's core features work reliably and as as intended, with a performance that should meet the expectation. The overall time precision and accuracy are limited by the performance of the CSAC and GPS, where the firmware and timekeeping introduces little to zero additional time error. Still, the performance of the synchronization of CSAC to GPS could be further improved by doing more research and testing. Some small features still need to be implemented and tested, and the disciplining process must be researched more. The implemented design provides a solution that is capable of synchronizing independent nodes within $\pm$ 200 ns, and with an frequency accuracy as small as 3.6×10^{-10}. It can capture external asynchronous signals and generate absolute timestamps with a resolution of 100 ns. GPS position, important system information, and diagnostics, are prepared to being accessed by either the host processor or by an operator via the debug interface. The system has features like the terminal menu system and logging interface, so it should be easy to continue with long-term logging and measurements.

For future work, the most valuable outcome of the project is that the initial hardware design and prototype were very suitable, and can be kept with only small changes. The same applies for the firmware design.

References

[1] Chalmers University of Technology, “Ethics Policy for Chalmers,” http://www.chalmers.se/insidan/EN/about-chalmers/organization/professional-ethics/ethics-policy-for, May 2005, accessed: 2015-06-15.

[2] FFI, “FFIs etiske retningslinjer,” http://www.ffi.no/no/Om-ffi/Documents/Etiske_retningslinjer_2007.pdf, October 2007, accessed: 2015-06-15.

[3] A. A. Syed and J. Heidemann, “Time Synchronization for High Latency Acoustic Networks,” in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications*. IEEE, 2006.

[4] R. Otnes, A. Asterjadhi, P. Casari, M. Goetz, T. Husøy, I. Nissen, K. Rimstad, P. van Walree, and M. Zorzi, *Underwater Acoustic Networking Techniques*. Springer Berlin Heidelberg, 2012.

[5] R. Otnes, H. Buen, U. Datta, V. Forsmo, J. Kjoell, T. Nordfjellmark, S. Richardsen, and P. van Walree, “Experiences from using the demonstrator system NILUS (Networked Intelligent Underwater Sensors),” http://promitheas.iacm.forth.gr/UAM_Proceedings/uam2011/UAM2011%20Paper%2028.3.pdf, November 2011, accessed: 2015-01-18.

[6] John R. Vig, “Quartz Crystal Resonators and Oscillators For Frequency Control and Timing Applications - A Tutorial,” http://www.am1.us/Local_Papers/U11625%20VIG-TUTORIAL.pdf, January 2000, accessed: 2015-05-25.

[7] Microsemi Corporation, “Quantum SA.45s Chip Scale Atomic Clock,” http://www.microsemi.com/products/timing-synchronization-systems/embedded-timing-solutions/components/sa-45s-chip-scale-atomic-clock, accessed: 2015-01-18.

[8] Symmetricom, “The SA.45S Chip-Scale Atomic Clock,” http://scpnt.stanford.edu/pnt/PNT11/2011_presentation_files/18_Lutwak-PNT2011.pdf, November 2011, accessed: 2015-01-18.

[9] D. W. Allan, “Clock characterization tutorial,” 1983, pp. 249–264.

[10] C. T.-C. Nguyen and J. Kitching, “Towards Chip-Scale Atomic Clocks,” in *Solid-State Circuits Conference, 2005. Digest of Technical Papers. ISSCC. 2005 IEEE International*, Feb 2005, pp. 84–85 Vol. 1.

[11] Microsemi Corporation, “Quantum SA.45s Chip-Scale Atomic Clock Datasheet,” http://www.microsemi.com/document-portal/doc_download/133305-quantum-sa-45s-csac, accessed: 2015-05-20.

[12] Steve Fossi, Symmetricom, “Product How-To: Disciplining a precision clock to GPS,” http://www.eetimes.com/document.asp?doc_id=1279697, May 2012, accessed: 2015-05-25.

[13] Atmel Corporation, “ATSAM3X8E Cortex-M3 MCU,” http://www.atmel.com/devices/SAM3X8E.aspx, January 2015, accessed: 2015-01-18.

[14] u-blox AG, “NEO-7 Series GPS/GNSS modules,” https://u-blox.com/en/gps-modules/pvt-modules/neo-7.html, January 2015, accessed: 2015-01-18.

[15] ——, *GPS - Essentials of Satellite Navigation*. u-blox AG, 2009.

[16] B. P. Stephen T. Powers, “The origins of GPS,” *GPS World*, 05,06 2010.

[17] E. F. Arias, “The Metrology of Time,” *Philosophical Transactions*, vol. 363, no. 1834, pp. 2289–2305, 2005.

[18] u-blox AG, “NEO-7 Series GPS/GNSS modules Data Sheet,” https://u-blox.com/images/downloads/Product_Docs/NEO-7_DataSheet_%28GPS.G7-HW-11004%29.pdf, November 2014, accessed: 2015-05-22.

[19] ——, “NEO-7 Series GPS/GNSS modules Receiver Description and Including Protocol Specification,” https://u-blox.com/images/downloads/Product_Docs/u-blox7-V14_ReceiverDescriptionProtocolSpec_Public_%28GPS.G7-SW-12001%29.pdf, February 2013, accessed: 2015-05-22.

[20] ARM Ltd., “Cortex-M3 Devices Generic User Guide,” http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0552a/BABHGEAJ.html, March 2015, accessed: 2015-04-10.

[21] A. J. Hebra, *The Physics of Metrology*. Wien: SpringerWienNewYork, 2010.

[22] N. I. of Standards and Technology, “The International System of Units (SI),” *NIST Special Publication 330 2008 Edition*, 2008.

[23] Z. Jiang and E. F. Aria, “Use of the Global Navigation Satellite Systems for the Construction of the International Time Reference UTC,” in *China Satellite Navigation Conference (CSNC 2013*. Springer Berlin Heidelberg, 2013.

[24] The IEEE and The Open Group, “The Open Group Base Specifications Issue 7,” *IEEE Std 1003.1-2008, 2013 Edition*, 2008.

[25] W. LeFebvre, “Time waits for no one,” *UNIX Review’s Performance Computing*, vol. 17, no. 10, pp. 57–58, 09 1999.

[26] Atmel Corporation, “SAM3X-SAM3A Series Complete Datasheet,” http://www.atmel.com/Images/Atmel-11057-32-bit-Cortex-M3-Microcontroller-SAM3X-SAM3A_Datasheet.pdf, March 2015, accessed: 2015-04-10.

[27] Terasic Inc., “DE0-Nano Development and Education Board,” http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=165&No=593&PartNo=1, accessed: 2015-05-25.