Proceedings

of 2013 IEEE 18th International Conference

on Emerging Technologies & Factory Automation

ETFA 2013

September 10-13, 2013

Cagliari, Italy.

IEEE Catalog Number: CFP13ETF-ART

ISBN: 978-1-4799-0864-6

ISSN: 1946-0740

Servo Design for Improved Performance in Software Timestamping-assisted WLAN Synchronization using IEEE 1588

Aneeq Mahmood
Institute of Computer Technology
Vienna University of Technology
Gußhausstraße 27-29/384, A-1040,
Vienna, Austria.
Aneeq.Mahmood@tuwien.ac.at

Reinhard Exel
Center for Integrated Sensor Systems
Danube University Krems
Viktor Kaplan Straße 2, A-2700,
Wiener Neustadt, Austria.
Reinhard.Exel@donau-uni.ac.at

Abstract

The clock servo is an essential constituent of distributed clock synchronization systems and a determining factor for the synchronization performance. This paper focuses on the clock servo design for better accuracy and precision in software-based clock synchronization over WLAN. The behaviour of the clock servo is analysed in this work and errors arising from software timestamps and the oscillator are evaluated. These errors are then used to investigate the operating conditions of the servo loop which results in the least synchronization offset and jitter. Results are obtained through simulations and then verified experimentally by a prototypical implementation. The results indicate that the synchronization performance is optimal when the servo relies more on the oscillator and less on the software timestamps.

1. Introduction

The use of wireless technology on the factory floor either as stand-alone communication or as part of a wired-wireless hybrid infrastructure can be seen as the next evolutionary step in industries for real-time communication [10]. IEEE 802.11 wireless local area network (WLAN) is one such technology which provides coverage over a large area, and offers flexibility and ease of communication without major changes in the infrastructure. Although WLAN is not inherently designed for deterministic communication, solutions such as IsoMAC [13] exist which proposes a software-based media access control (MAC) layer inside device drivers. Through this software MAC, a time-based scheduling mechanism can be arranged which can assist in real-time communication. The advantage of such a scheme is that it proposes only software modifications inside the device driver and does not require any changes in the hardware. This enables the use of commercial off-the-shelf (COTS) devices for industrial communications; these devices are significantly cheaper than devices whose hardware have been customised to support real-time communication.

A major pre-requisite for any time division multiple access (TDMA)-based mechanism is the provisioning of synchronized clocks in the system. Clock synchronization also enables logging of information with correct timestamps which can assist in monitoring and systems diagnostics. The IEEE 1588 precision time protocol (PTP) is a commonly used master-slave based clock synchronization protocol in industries; it offers synchronization support for different media, such as Ethernet or PROFINET, by using several message transport mechanisms. Although the use of PTP over WLAN is not covered by the standard yet, it is already an active research topic [9].

Whereas PTP overlooks different aspects of synchronization like message transport, master selection, and security and redundancy, it has left the design of the control loop and slave clock adjustment open for implementation. Nevertheless, a proportional and integral (PI) controller is commonly used for steering the slave clock because of its simplistic design and existing widespread usage in industries. The goal in this work is to design the clock servo for PTP using a PI controller for WLAN synchronization. As only COTS WLAN devices are being considered in this work, the synchronization will be established using software timestamps drawn by the operating system (OS). This implies that the clock servo has to be optimised in order to deal with various uncertainties associated with software timestamps. Additionally, the noise coming from the oscillator driving the local clock will be discussed in this work. The combined effect of the oscillator noise and the software timestamping errors will be analysed for optimal clock servo design.

The rest of the paper is structured as follows: Section 2 provides the necessary synchronization nomenclature and the state of the art for the software timestamp-driven clock servo; Section 3 analyses the various error sources contributing to the clock servo; Section 4 provides the simulation analysis of the improved clock servo; Section 5 highlights the performance of the servo with a test-bed im-

978-1-4799-0864-6/13/$31.00 ©2013 IEEE

plementation; Section 6 provides the concluding remarks.

2. Principles of Clock Synchronization and Servo Design

2.1. Clock Synchronization

Using the simple clock model of [6], the value of a clock C at time t can be described as

$$C(t) = at + b, \tag{1}$$

where a is the frequency skew of the clock and b is the offset. For synchronizing this clock to an external reference clock, a highly stable clock source is used whose skew can be considered unity. Such a reference clock C_{ref} can be written as

$$C_{\mathrm{ref}}(t) = t. \tag{2}$$

The goal of clock synchronization is to make the clock C follow the reference clock such that

$$|C(t) - C_{\mathrm{ref}}(t)| \leq \alpha, \tag{3}$$

where α denotes the maximum acceptable offset between the clocks which is defined by the application using synchronized clocks. Though there are other parameters for describing synchronization performance [2], the two commonly used ones are synchronization accuracy and precision. Synchronization accuracy is normally determined by calculating the mean value of the difference between clock C and the reference clock. Synchronization precision is also referred to as synchronization jitter and is measured as the standard deviation from the mean clock error.

The software timestamps are drawn by the OS by reading a clock tick counter from a register. As compared to hardware timestamping which is carried out inside or just above the physical layer of a device, software timestamps can be drawn anywhere between the data link layer and application [1]. As the timestamp jitter increases with the number of network layers, timestamps should be taken as close as possible to the physical layer. For software timestamping, this is the interrupt handling routine of the device driver. It is invoked by the CPU after it has been informed about the packet transmission or reception, using the hardware interrupt signal.

2.2. Related Work

The implementation of PTP over WLAN has been investigated in [9] where it has been shown that submicrosecond accuracy over WLAN is possible with software timestamps. The authors have used open source WLAN drivers and timestamping is done inside the interrupt handling routine in the driver. However, this study does not discuss the slave clock servo for synchronization. The design of the clock servo in the presence of oscillator noise and hardware timestamps has been discussed in [8]. The authors have shown that a PI controller is the simplest controller which can be used for designing the clock servo. The stability analysis of a clock servo using a PI controller has been carried out in [2]. The author has highlighted the stable area for the operation of the control loop and has also shown the impact of quantization error on the servo performance. The use of a PI controller is also advocated in [1] where the authors have discussed the architecture of software-based wired synchronization. In this study, the authors have discussed the design of filters for timestamps and delay filtering but have not analysed the impact of the errors going into the control loop.

The simulation-based study of [6] has modelled the software timestamping errors as a sum of Bernoulli and Gaussian random variables. The Bernoulli variable simulates large deviations in the interrupt handling and scheduling delays which occur due to the non-deterministic nature of OS such as in Linux. It has been shown that the jitter can be minimised if the disturbances of the Bernoulli variables can be controlled. However, the study does not focus explicitly on the design of a PI controller, but takes packet delay variations into account which are associated with wired communication. This reduces the significance of the results for wireless synchronization.

2.3 Servo Design

Figure 1 shows the basic layout of the clock servo for master-slave synchronization in PTP. The master comprises of a highly stable oscillator and keeps a monotonically increasing time denoted by $t_{\mathrm{m}}[n]$. The timestamps, which are used to determine the offset in PTP, are drawn inside the WLAN device driver after the WLAN device has notified the host CPU through a hardware interrupt, as is the case in [9]. This implies that the timestamp is affected by the time it takes the CPU to react on the interrupt from the WLAN device and also by the time it takes the host CPU to draw the software timestamp by reading a software counter. As a result, the timestamps at the master and the slave side will have additional jitter which are denoted by $e_{\mathrm{m}}[n]$ and $e_{\mathrm{s}}[n]$, respectively. The delay $d[n]$ is retrieved from the propagation delay $d_{\mathrm{p}}[n]$ after passing the latter through a filter as discussed in [1].

The reference signal which steers the control loop is denoted by $r[n]$. The output of the controller G is denoted by $u[n]$ and is fed into the slave clock S which is modelled as an integrator. The noise coming from the oscillator $\epsilon_\tau[n]$ is added to the output of the slave clock to get the signal $t_s[n]$ which is fed back into the controller. This signal is also quantized to obtain $w[n]$ which is equivalent to a 1 pulse per second (PPS) signal. The 1 PPS signals from the master and the slave clock are compared against each other to obtain the overall synchronization accuracy and jitter.

To discuss the performance of the control loop and to analyse the impact of various error sources, a z-domain analysis will be carried out. Hence for the following dis-

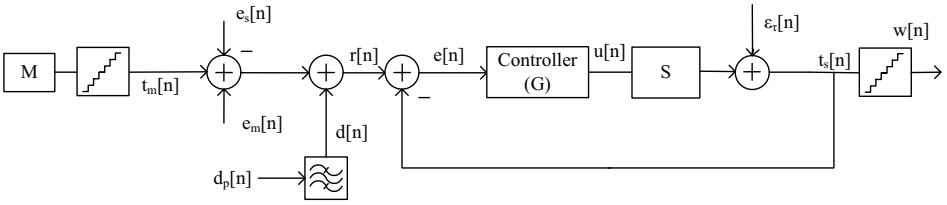

Figure 1. Basic layout for the clock servo

cussion, the z-transform equivalent of all the time-domain signals will be represented by using their respective capital letters e. g.,, $Z\{e_{\mathrm{s}}[n]\} \rightarrow E_{\mathrm{s}}(z)$. In z-domain, the PI controller G can be written as

$$G(z) = k_{\mathrm{p}} + k_{\mathrm{i}} T \frac{z}{z-1}, \tag{4}$$

where T is the synchronization period, and k_{p} and k_{i} are the proportional and the integral coefficient, respectively. From [2], the slave clock is modelled as a sample and hold circuit and an integrator, and is written as

$$S(z) = \frac{K_{\mathrm{c}}}{z-1} \tag{5}$$

where K_{c} is a scaling constant. Neglecting the quantization errors for a simpler analysis, the output of the closed loop can then be written as

$$T_{\mathrm{s}}(z) = \frac{G(z)S(z)}{1+G(z)S(z)} R(z) - \frac{E_\tau(z)}{1+G(z)S(z)}, \tag{6}$$

where $R(z) = T_m(z) + D(z) - E_S(z) + E_m(z)$ and E_τ is the z-transform for $\epsilon_\tau[n]$. The expansion of (6) results in

$$\begin{aligned} T_{\mathrm{s}}(z) &= \frac{G(z)S(z)}{1+G(z)S(z)} \left[T_{\mathrm{m}}(z) + D(z)\right] \\ &+ \frac{G(z)S(z)}{1+G(z)S(z)} \left[E_m(z) - E_s(z) - \frac{E_\tau(z)}{G(z)S(z)}\right]. \end{aligned} \tag{7}$$

The above equation can be written as $T_{\mathrm{s}}(z) = A(z) + B(z)$ where $A(z)$ represents the deterministic part which reflects monotonically increasing time and propagation delay, and is represented by the first right-hand term in (7). The term $B(z)$ stands for the z-transform of all the major error sources in the synchronization path, namely timestamping jitter from the transmit and the receive side, and jitter from the oscillator. Thus as discussed in [8], if the control loop can compensate for the clock offsets and the skew is assumed constant, the error coming from the slave clock can be obtained by taking the inverse z-transform of $B(z)$. Then, by using suitable control loop parameters, the overall jitter for synchronization can be minimised. In other words, if $r[n]$ shows no monotonically increasing timing input but only random errors from quantization, e. g., oscillator and timestamping errors, then the controller will operate in such a way that the error $e[n]$ is minimised. In that case, the standard deviation of $t_{\mathrm{s}}[n]$ will be the synchronization jitter, and any bias in its value will represent the synchronization offset.

However, it is known that the oscillator noise is not white but is the sum of several power-law noises in the frequency domain as shown in [11] and [5], and the probability density function (PDF) of the timestamping error is unknown and cannot be assumed Gaussian like in [8]. Additionally, the frequency skew is not constant and may vary. Therefore, a closed-form expression for (7) cannot be obtained. As a result, a simulation of the control loop has to be carried out to analyse the synchronization performance of the control loop. The stability of this control loop is defined by the poles of the characteristic equation $1 + G(z)S(z) = 0$, the denominator in (7). From [2], the poles $p_{1,2}$ can be obtained by

$$p_{1,2} = \frac{-(-2 + K_{\mathrm{p}} + K_{\mathrm{i}}) \pm \sqrt{(K_{\mathrm{p}} + K_{\mathrm{i}})^2 - 4K_{\mathrm{i}}}}{2}, \tag{8}$$

where $K_{\mathrm{p}} = K_{\mathrm{c}} k_{\mathrm{p}}$ and $K_{\mathrm{i}} = K_{\mathrm{c}} T k_{\mathrm{i}}$.

3. Modelling of Error Sources

To carry out a simulation of the clock servo shown in figure 1, the oscillator noise needs to be quantified. Similarly, the timestamping errors at the transmit and the receive sides need to be measured. In this section, these two error sources are discussed and analysed in detail.

3.1. Error from the Oscillator

Modern digital clocks comprise of an electronic counter which is driven by a frequency source such as a crystal oscillator. However, a real oscillator shifts away from its natural frequency and gives rise to a skew and random phase noise [4]. For characterising the performance of an oscillator, Allan variance [11] is employed which is a measure of the relative frequency stability of an oscillator as a function of the sampling period τ, which coincides with the synchronization interval T for clock synchronization applications. It can be used for identifying various noise types based upon the sampling period. The Allan variance is also employed for the IEEE 1588 protocol to elect the best master clock using the so-called best master clock algorithm.

However for synchronization, the Allan variance as relative measure of frequency stability is not directly applicable; the interesting aspect is the phase error Δt which

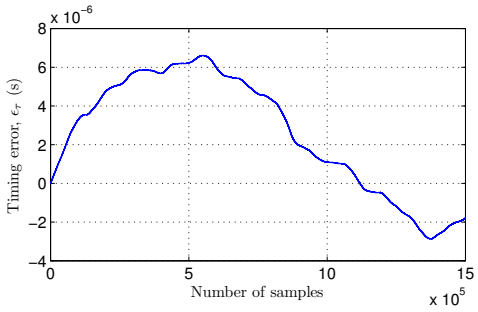

Figure 2. Timing error for an exemplary oscillator

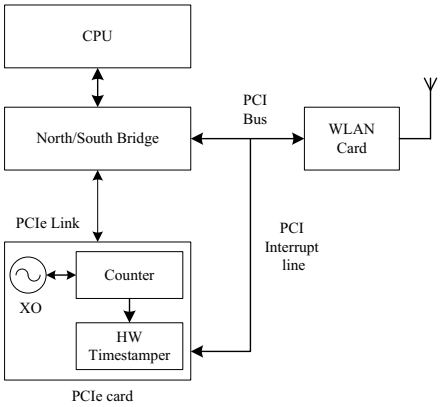

Figure 3. Setup for measuring interrupt handling and timestamping delays

is gained between two synchronization messages spaced τ apart. The timing error $\epsilon_\tau(t)$ from the oscillator is then the accumulated error from the beginning of synchronization till time t as

$$\epsilon_\tau(t) = \int_0^t \Delta t(\tau)\, d\tau. \tag{9}$$

Using the simple clock model of (1), the slave clock at t can then be written as

$$C(t) = at + b + \epsilon_\tau(t), \tag{10}$$

where b is the initial clock offset and a is the clock skew. In reality, the oscillator skew is also not constant. Nevertheless, if the clock adjustment mechanism takes care of the initial clock offset and compensates for the skew, then skew and offset of $\epsilon_\tau(t)$ do not contribute to the clock error. Figure 2 depicts $\epsilon_\tau(t)$ compensated by any constant skew and offset obtained from a 50 ppm oscillator on a FPGA board from [7]. The figure can be interpreted as follows: Even if the slave knew the initial skew and offset precisely and would compensate for these, clock offsets of multiple microseconds would be present without continuous resynchronization because the oscillator drifts away. Using regular synchronization messages, the slave is able to estimate these timing errors and compensate for them, thus minimizing the clock offset with respect to the master.

3.2. Timestamping Jitter

The software timestamping jitter comes from the interrupt handling of the CPU and from drawing the timestamp. These delays are often modelled using commonly known PDFs such as Gaussian, Exponential, Bernoulli, etc. In practice, these PDFs depend on the software implementation, such as the the interrupt system, the capability of the OS, and the host CPU. In this paper, a simple setup is proposed which can accurately determine timestamping and interrupt handling delays and the associated jitter. An illustration of the test setup is shown in figure 3.

The figure depicts a COTS WLAN card which is used for communication and which is connected to the host CPU via the PCI bus. The first indication of a successful packet transmission or reception to the CPU from the card is via the PCI interrupt signal. This interrupt signal is fed into a PCIe Ethernet card. This PCIe card has a physical port which can detect external signals, and a timestamping unit which can draw a hardware timestamp from the clock tick counter upon occurrence of the external signal on its port. The same counter which is being used for hardware timestamping is accessible to the OS through software means as well. Therefore later on, when the CPU acknowledges the interrupt and executes the interrupt handling routine, a software timestamp is drawn from this counter. By calculating the difference between the hardware and software timestamps, the interrupt handling and timestamping delay can be accurately calculated. In this setup, a PCIe card is employed as it does not share any interrupt line with the PCI bus. In general, any peripheral device which does not share the same interrupt line as the WLAN card can be employed. Also, the device should have the capability to timestamp an external event occurring on one of its ports or pins.

To measure the interrupt handling and timestamping delay for actual hardware, the measurement setup is established using appropriate hardware. The WLAN card is the Atheros WLM54G chipset with open source driver support. The Ethernet card is the Syn1588 PCIe card [12] from Oregano Systems. The PCIe card carries a 25 ppm crystal oscillator (XO) which acts as the time source for both hardware and software timestamping. Whenever a packet is transmitted, the resulting interrupt signal is timestamped, and the hardware timestamp is compared against the software timestamp. A similar setup is created for the reception side as well to measure the interrupt handling and timestamping delay at the receiver. The hardware components at the transmit and the receiver sides are housed inside two 2.4 GHz (single core) CPUs using Linux 2.6.32 as operating system.

Figures 4 and 5 provide the histograms for the inter-

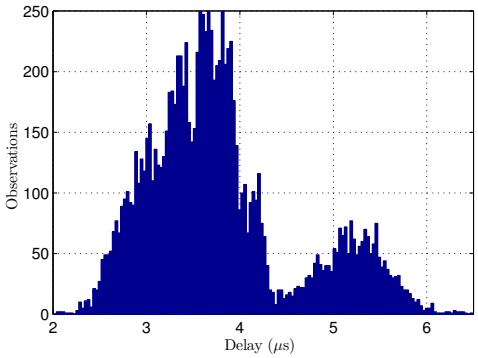

Figure 4. Histogram of the difference between the hardware timestamp, upon detecting the interrupt, and the software timestamp at the transmitter

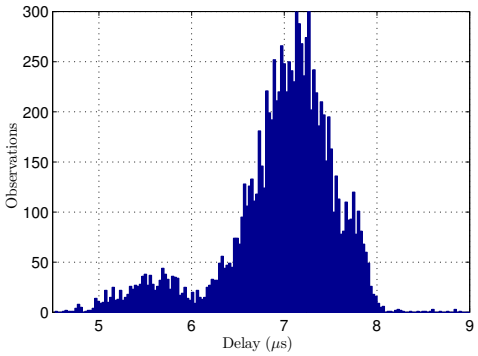

Figure 5. Histogram of the difference between the hardware timestamp, upon detecting the interrupt, and the software timestamp at the receiver

rupt handling delays for the transmitter and the receiver respectively. They show that the distribution for both delays is not Gaussian as often assumed. These delays are affected by the arrival of interrupts of higher priority than for WLAN communication and by Linux task preemption and, hence, can spread over several microseconds. The delay at the receive side is higher than at the transmitter side. One reason for such a behaviour can be that reception involves transfer for the packet from the WLAN card to the host over the North/South bridge as shown in figure 3. The counter for reading the software timestamp is on the PCIe card which is accessed also via the North/South bridge. Because of the occupation of the bridge by the WLAN chipset, the drawing of software timestamp can be delayed which results in higher interrupt handling delay at the receiver.

Apart from different delays at the transmitter and the receiver, the histograms in figure 4 and 5 shows one large and one smaller distribution of the delay. The reason for the smaller distribution at the transmitter is the occurrence

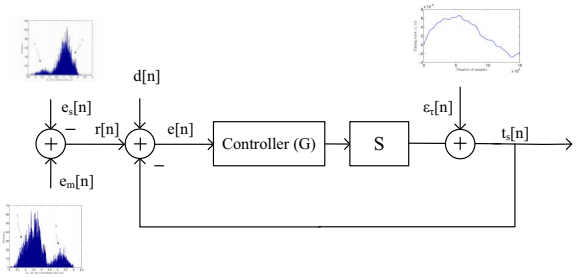

Figure 6. IEEE 1588 clock servo

of higher priority interrupts which increases the interrupt handling delay. At the receiver side, the smaller distribution occurs when the North/South bridge is not being used for other operations. In this case, the interrupt handling delay decreases. However, most of the time, the delay at the receiver remains in the larger distribution.

4. Simulation-based Analysis

Using the error sources from section 3, a simplified version of figure 1 is simulated in Matlab. This simplified servo layout is shown in figure 6 along with the error sources. The goal is to suppress the error sources which are present in the system. As a result, the monotonically increasing master clock is skipped from figure 6. Additionally, the filter which has been inserted in figure 1 for filtering the propagation delay is removed, because it only shifts the clock but has no impact on the servo stability. This scenario applies, for instance, when the propagation delays do not change significantly between the synchronization messages, i. e., when the distance of the wireless link varies only slowly.

The goal for synchronization is to minimise the Root Mean Square (RMS) offset value at the output of the slave clock. The RMS offset contains information about both synchronization offset and jitter, and minimising the RMS offset can provide a single criterion for the optimal control loop parameters. The synchronization interval is set to 1 s; the constant K_c can be set to unity as well so that $K_\mathrm{p} = K_\mathrm{c}k_\mathrm{p} = k_\mathrm{p}$ and $K_\mathrm{i} = K_\mathrm{c}Tk_\mathrm{i} = k_\mathrm{i}$. To find the $[K_\mathrm{p}, K_\mathrm{i}]$ value which results in minimum RMS offset, the simulation is carried out for a wide range of K_p and K_i values.

Figure 7 shows the RMS offset as a function of K_p and K_i as a 3D plot. The flatter region in figure 7 is the region where the control loop is unstable and where the output of the slave increases boundlessly. To focus on the stable region of the control loop and to focus on the RMS offset minimisation capabilities of the control loop, the unstable region is truncated to 5 µs. Figure 7 also shows that the stable region of the control loop looks like a valley; the lowest point in the valley correspond to the $[K_\mathrm{p}, K_\mathrm{i}]$ value which generates the minimum RMS offset. Around this value, increasing either K_p or K_i results in the increase of the RMS offset and therefore a degradation of

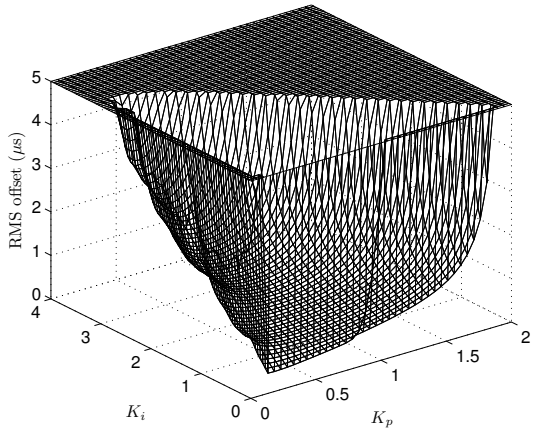

Figure 7. Synchronization accuracy in terms of RMS offset as a function of K_p and K_i

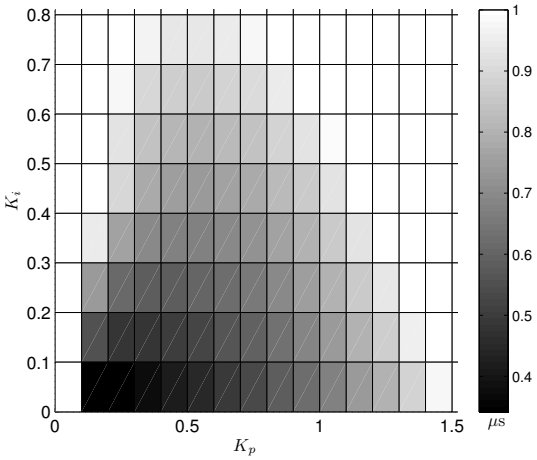

Figure 8. RMS offset regions as a function of K_p and K_i

the synchronization quality.

To further focus on the stable region, a 2-D plot is generated from the 3D plot as shown in figure 8. This figure shows that the minimum RMS offset is obtained when the $[K_\mathrm{p}, K_\mathrm{i}]$ value is small; the minimum RMS offset value is 228 µs when $K_\mathrm{p} = 0.16$ and $K_\mathrm{i} = 0.01$. Thus, the best tracking of the master clock is obtained when the controller is run as an *almost* proportional controller with minimal integral portion. The reason is that a smaller value of K_p will limit the impact of the incoming jitter from the software timestamps. If a large value of K_p is chosen, this will allow a larger error to go through the controller resulting in a larger fluctuation of the slave clock. This is evident in both figure 7 and 8. Therefore for systems with software timestamping-based synchronization, the synchronization offset can be minimised by using a low value of K_p.

Nevertheless, the integral coefficient K_i is required to remove any steady-state error in the control loop. However, as the propagation delay $d[n]$ has already been compensated, the steady-state error is small. Consequently, only a small value of K_i is required to remove residual clock offsets.

To highlight the impact of the oscillator error, another simulation is carried out which focuses only on the oscillator noise and the RMS offset value between the master and the slave clocks. Figure 9 shows the 2-D plot for the RMS offset of the slave clock, when considering zero timestamping errors and only the timing error according to figure 2. For this simulation, the region where the RMS offset is minimum is shaded with dark colour; outside the dark-shaded area is the region with higher RMS offset truncated at 5 ns. A comparison between figure 8 and 9 reveals that a wide range of $[K_\mathrm{p}, K_\mathrm{i}]$ values can be used when only the timing errors from the oscillator need to be compensated.

Whereas $K_\mathrm{p} = 0.16$ and $K_\mathrm{i} = 0.01$ resulted in the lowest RMS offset in figure 8, the clock adjustment in the slave without any timestamping errors requires larger K_p and K_i values for optimal synchronization performance. This case is applicable to hardware timestamping, where RMS offsets in the nanosecond range and below are possible [3]. With the increased jitter of the software timestamps the K_p and K_i values naturally need to be lowered, while still being able to follow any timing errors of the oscillator.

Therefore, by using small values for the $[K_\mathrm{p}, K_\mathrm{i}]$ pair, the clock servo minimises the amount of information which is being delivered through software timestamps by the reduced loop-bandwidth of the control servo. By doing so, the timestamping errors which are fed into the clock servo are also limited; the servo starts to believe more in the information given by the oscillator, and the overall synchronization performance is improved.

5. Test-bed Implementation

A test-bed implementation has been carried out to analyse the servo behaviour for an actual system and to compare its results with the ones obtained from the simulation. For this purpose, the setup from figure 3 is established to constitute a 1588 master-slave hierarchy. Both master and the slave use standard 50 ppm crystal oscillators as their clock sources. Master and the slave devices are stationary and are in line of sight with each other. The proportional and the integral coefficients of the controller in the slave clock are set to 0.1 and 0.01 , respectively. For the test-bed implementation, the synchronization interval is set to 1 s. The synchronization accuracy is verified by comparing the 1 PPS signals from the master and the slave clock. The average difference between the 1 PPS signals provides the mean clock offset while the standard deviation from the

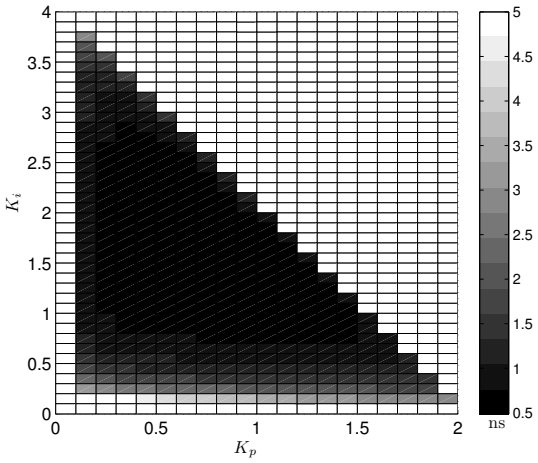

Figure 9. RMS offset regions using only the oscillator noise

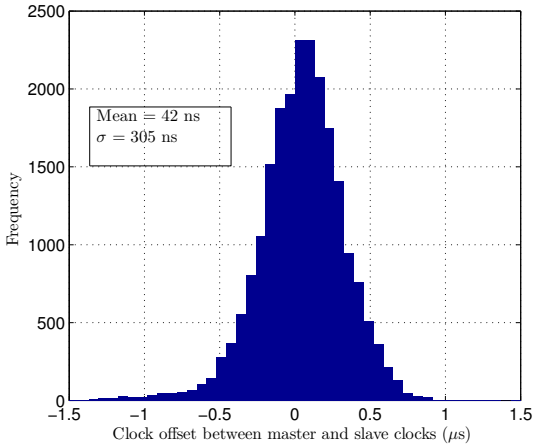

Figure 10. Histogram showing the synchronization accuracy and precision

mean value provides the synchronization jitter.

Figure 10 shows the histogram of clock offset between the master and the slave clocks. The final mean synchronization offset is found out to be 40 ns with a jitter of 305 ns. As the mean offset is consistent between multiple measurements (cf. table 1), it can be removed by calibration and is neglected in the following. Therefore, the jitter corresponds to the RMS offset. This RMS value of 305 ns is slightly higher when compared with the simulation results where the RMS offset has been 228 ns. However, in simulation, the oscillator noise from the master clock has not been considered which is not the case in the test-bed implementation. Moreover, the oscillators are directly exposed to environment which can impact their performance to some extent and result in more overall jitter. Considering the state of the art, the results from this work can be

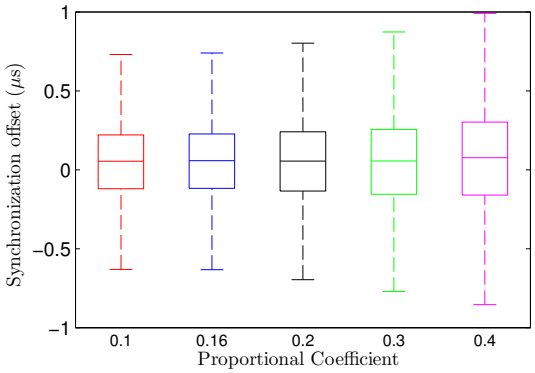

Figure 11. Boxplot for jitter versus increasing values of K_p

K_p	Mean	Jitter	RMS offset
0.10	42 ns	305 ns	308 ns
0.16	46 ns	312 ns	315 ns
0.20	41 ns	325 ns	327 ns
0.30	39 ns	372 ns	374 ns
0.40	47 ns	427 ns	429 ns

Table 1. Synchronization performance with increasing proportional coefficient, K_p

compared against those from [9] which also use interrupt-based software timestamping for WLAN synchronization. The synchronization jitter in [9] is 581 ns which is almost double when compared with the jitter achieved in this work. Hence through proper servo design, the higher synchronization accuracy and precision can be achieved even with software timestamps.

To make sure that the minimum synchronization jitter occurs at low $[K_\mathrm{p}, K_\mathrm{i}]$ value and the results of the implementation are consistent with those from the simulation, a set of measurements have been carried out with increasing values of K_p with fixed $K_\mathrm{i} = 0.01$. The results of these measurements are displayed in figure 11 using a boxplot. In this figure, it can be seen that the mean synchronization offset remains the same regardless of the value of K_p. This indicates that the steady-state error resulting from the clock skew has been sufficiently dealt with by using $K_\mathrm{i} = 0.01$. However, the immediate impact of increasing K_p from 0.1 to 0.16 and then to 0.2 has been that jitter from the timestamps increasingly passes through the control loop. As a result, the overall synchronization jitter increases. By further increasing the value of K_p to 0.3 and 0.4, the jitter continues to increase and the synchronization precision degrades. Therefore, it can be established that the results obtained from the test-bed implementation and the simulation are consistent with each other. The mean clock offset, jitter, and the RMS offset for $K_\mathrm{i} = 0.01$ and varying K_p values are listed in table 1.

6. Conclusion and Outlook

In this paper, the performance of the clock servo for IEEE 1588 over WLAN has been discussed using a PI controller and software timestamps. The two main error sources which have been identified in this work are the timestamp errors and the noise coming from the oscillator. The oscillator noise has been obtained through an existing hardware setup. To measure the noise of the software timestamps, a measurement setup has been created which is capable of measuring the interrupt handling and timestamping delay and jitter regardless of where the software timestamps are drawn in the protocol stack. This approach is different from the state of the art which associates commonly-known PDFs with the software timestamping jitter, yet without measuring these for real equipment.

The noise from software timestamps and the oscillator is fed into the simulation model of the clock servo to determine the $[K_\mathrm{p}, K_\mathrm{i}]$ values which give optimum synchronization performance. Simulation results indicate that synchronization performance is best when the $[K_\mathrm{p}, K_\mathrm{i}]$ values are small. The small K_i value helps in dealing with the variable clock skew to remove steady-state clock offset; the low value of K_p limits the impact of the timestamp jitter on the performance of the clock servo. Consequently, the clock servo relies less on the information which is provided in the timestamps and trusts more on the oscillator steering the slave clock. Hence for software timestamping-based synchronization, confiding in the oscillator and limiting the impact of (the noise from) the timestamps gives the best synchronization performance. This claim is also verified in this paper using a test-bed implementation for varying K_p values.

In this paper, the goal has been to improve the synchronization performance in terms of mean offset and jitter. In practice, the transient phase of the clock servo in which it tries to settle to its steady-state value is also of significance. For most applications, it is desirable that the convergence time of control loop is kept small. Additionally, if the control loop diverges somehow from its steady state, it is important that the time to resynchronize to the master clock is small so that the underlying application is not adversely affected. These issues regarding the time it takes to (re)synchronize are not discussed in this paper but are considered a part of future work.

References

[1] K. Correll and N. Barendt. Design Considerations for Software Only Implementations of the IEEE 1588 Precision Time Protocol. In *Conference on IEEE 1588 Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, 2006.

[2] J. C. Eidson. *Measurement, Control, and Communication Using IEEE 1588*. Spinger, 2006. pp. 45–50, 62–67.

[3] R. Exel. Clock Synchronization in IEEE 802.11 Wireless LANs using Physical Layer Timestamps. In *Proc. of the International IEEE Symposium on Precision Clock Synchronization for Measurement, Control and Communication, ISPCS*, San Francisco, USA, Sept. 2012.

[4] G. Gaderer, P. Loschmidt, and T. Sauter. Improving Fault Tolerance in High-Precision Clock Synchronization. *IEEE Trans. Ind. Informat.*, 6(2):206–215, May 2010.

[5] G. Gaderer, A. Nagy, P. Loschmidt, and T. Sauter. Achieving a Realistic Notion of Time in Discrete Event Simulation. *International Journal of Distributed Sensor Networks*, 2011:11, July 2011.

[6] G. Giorgi and C. Narduzzi. Modeling and Simulation Analysis of PTP Clock Servo. In *IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication, 2007*, pages 155–161, Oct. 2007.

[7] P. Loschmidt, R. Exel, A. Nagy, and G. Gaderer. Limits of Synchronization Accuracy Using Hardware Support in IEEE 1588. In *Proc. of the International IEEE Symposium on Precision Clock Synchronization for Measurement, Control and Communication, ISPCS*, pages 12–16, Ann Arbor, MI, USA, Sept. 2008.

[8] D. Macii, D. Fontanelli, and D. Petri. A master-slave synchronization model for enhanced servo clock design. In *International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, pages 1–6, oct. 2009.

[9] A. Mahmood, G. Gaderer, H. Trsek, S. Schwalowsky, and N. Kerö. Towards High Accuracy in IEEE 802.11 Based Clock Synchronization using PTP. In *Proc. of the International IEEE Symposium on Precision Clock Synchronization for Measurement, Control and Communication, ISPCS*, pages 13–18, Munich, Germany, Sept. 2011.

[10] T. Sauter. The Three Generations of Field-Level Networks–Evolution and Compatibility Issues. *IEEE Transactions on Industrial Electronics*, 57(11):3585–3595, 2010.

[11] D. Sullivan, D. Allan, D. Howe, and F. Walls. Characterization of Clocks and Oscillators. Technical Note 1337, NIST, 1990.

[12] O. Systems. Syn1588 PCIe Network Interface Card. http://www.oreganosystems.at, Last accessed: 28/03/2012.

[13] H. Trsek and J. Jasperneite. An isochronous medium access for real-time wireless communications in industrial automation systems - A use case for wireless clock synchronization. In *International IEEE Symposium on Precision Clock Synchronization for Measurement Control and Communication (ISPCS), 2011*, pages 81–86, sept. 2011.