A Time Synchronization Circuit with Sub-microsecond Skew for Multi-hop Wired Wearable Networks

Fardin Derogarian[1a], João Canas Ferreira[a], Vítor M. Grade Tavares[a]

[a]*INESC TEC and Faculty of Engineering, University of Porto (FEUP)*

Abstract

This paper describes and evaluates a fully digital circuit for one-way master-to-slave, highly precise time synchronization in a low-power wearable system equipped with a set of sensor nodes. These sensors are connected to each other in a mesh topology, with conductive yarns used as one-wire bidirectional communication links. The circuit is designed to perform synchronization in the MAC layer, so that the deterministic part of the clock skew between nodes is kept constant and compensated with a single message exchange. In each sensor node, the synchronization circuit provides a programmable clock signal and a real-time counter for time stamping. Experimental results from a fabricated ASIC (in a CMOS 0.35 µm technology) show that the circuit keeps the one-hop average clock skew below 4.6 ns and that the skew grows linearly as the hop distance to the reference node increases. The sub-microsecond average clock skew achieved by the proposed solution satisfies the requirements of many wearable sensor network applications.

Keywords:
Body area networks, Wearable sensor network, Time synchronization protocol, CMOS ASIC

1. Introduction

Time synchronization, with appropriate accuracy, is required in many distributed data acquisition systems, because information about the relative time occurrence of events is generally of importance. Environment

[1]Corresponding author: Fardin Derogarian, Rua Dr. Roberto Frias, 4200 - 465, Porto, Portugal. Email: mpt09020@fe.up.pt, Tel: +351 968 726 403

monitoring, localization, coordinated node sleep/wake-up scheduling mechanisms, , and data fusion [1, 2] are just a few of such applications that are critically dependent on precise synchronization. As an example, the Body Area Network (BAN) described in [2], which is used in gait analysis for evaluation and diagnosis of mobility impairments, requires that inertial and electromyography (EMG) measurements collected by different nodes be time stamped with an error below 0.5 ms.

Since sensor networks, either wired or wireless, are distributed data acquisition systems with limited resources, the design challenge is to preserve performance while maintaining a precise synchronization between the sensor nodes with minimum complexity. The synchronization method then ensures that time keeping at all nodes of the network proceeds with a small and bounded difference relatively to a reference clock.

Usually each Sensor Node (SN) is equipped with an independent clock generator based on a quartz crystal. The frequency tolerance of this kind of oscillators fall in the range of a few parts per million (ppm). However, the oscillators will operate independently at each node, with different frequency and phase. The shifts and skews, even if slight, will accumulate in time, preventing synchronization. A synchronization method, such as a network protocol or dedicated hardware, is then necessary to establish synchronization with an acceptable accuracy [3, 4, 5, 6]. In fact, time synchronization problems have been studied in all areas of networking [3, 1, 7], and typically the main challenge has centered on overcoming the non-deterministic delay effects with the limited resources available. In general, the delay associated with data transfer between two nodes has four components:

- Send time: time for assembling a message and handing it over to the Media Access Control (MAC) layer;
- Access time: time for accessing the communication channel;
- Propagation time: delay due to signal propagation between nodes;
- Receive time: time for receiving and processing the message at the receiver.

Except for propagation time, these delay sources are generally non-deterministic.

Among the many synchronization methods, two-way message exchanges between pairs of nodes are widely used to estimate the time delay and time offset between two nodes [8, 9, 10]. In this approach, the delay and offset are calculated by sending and receiving messages with timing information and by measuring the round trip delay. A second approach uses only one-way communication [11, 12]. In this case, the node that acts as a time reference sends timing information to client nodes, which must then estimate the delay and offset.

Reducing the clock skew, due to delay and offset, depends on the synchronization and estimation method. Eliminating any non-deterministic (systematic) components of delay will increase the estimation accuracy and decrease clock skew. This work, which is intended for wearable wired networks, uses a timing message method with fixed length. The message carries the state of the sender's time counter at the exact moment of transmission. The receiver node processes the message immediately, without buffering. In this way, the delay between the sender and receiver will be mostly deterministic. The only non-deterministic component that remains is related to the local clock frequency differences between the sender and the receiver, because clock oscillators in each sensor are independent and their frequency exhibits small but variable drifts.

This paper[2] presents a fully digital circuit for one-way synchronization based on this approach implemented in the MAC layer. The application context is a wearable mesh network for clinical applications, with node connections made with conductive yarns embedded in textile fabrics [2]. Under these conditions, the ASIC implementation of the circuit has shown to be capable of keeping the clock skew between neighboring nodes below 4.6 ns, on average.

The rest of the paper is organized as follows: Section 2 provides the background and describes related work. Section 3 justifies the proposed synchronization method and details the protocol. The organization and operation of the synchronization circuit is described in Section 4. Section 5 presents experimental results obtained with a small network of sensors. The main conclusions are summed up in Section 6.

2. Background and Related Work

There are many synchronization methods described in the literature which present different kinds of solutions, frequently employing application-dependent

[2]This paper is an extension of [13].

features [7]. One of the most used techniques is the so called master-slave approach. In this method, one of the nodes (the master) serves as the time reference, while the others (slaves), which are one hop away, synchronize their clock with the master. By contrast, in peer-to-peer protocols each node can directly get the timing information from many other nodes in the network (which are also one hop away). Although peer-to-peer protocols are very effective in failure elimination and do not demand the setting of a master node—therefore being more flexible in this sense—they necessitate a more complex control flow, which may impose overheads that cannot be met by the available resources. However, for cases where the number of sensors is large, the option for communication often relies on multi-hop connections instead. In such situations, the synchronization must be performed by sender-to-receiver or receiver-to-receiver approaches. In the former, the receiver node synchronizes its clock with the sender and re-sends timing data to the next node after synchronization; while in the latter, the sender broadcasts timing messages and the receivers in its coverage area exchange messages among themselves instead of interacting with the sender.

An alternative procedure that avoids exchange of information for synchronization is to use a global reference that can be sensed simultaneously by all nodes, such as ambient magnetic fields [14] or electrical fields [15] emitted by power lines. The first approach is able to achieve an average synchronization between all nodes in a multi-hop network of less than 1 ms; the second approach is able to keep clock drifts to within 0.864 µs after 24 hours. The main limitation of these approaches is the need to have additional modules for sensing the power line fields; in addition, these fields are not stable nor available everywhere.

The network time protocol (NTP) is another well-known time synchronization protocol based on two-way message exchange [9]. NTP is one the most used protocols, specially on the Internet, and it is known as an effective, secure and robust protocol. NTP clients synchronize their local clocks to the NTP time servers by statistical analysis of the round-trip time. To achieve highly precise time synchronization, the time servers are equipped with or synchronized to atomic clocks or GPS signals. The significant complexity of NTP makes its implementation in sensor networks difficult. In addition, the non-determinism in transmission time of WSNs can introduce large delays. Therefore, NTP is suitable only for synchronization in Wireless Sensor Networks (WSNs) with low precision demands.

If a clock source broadcasts its time, adjacent nodes will receive the same timing message at approximately the same time. In this way, receiver nodes can obtain timing information that is subject to very little delay variability. This one-way method has been employed in the RBS protocol [11]. In this approach, which can be used in both wired and wireless networks, a node is selected as a time reference for all other nodes. Each node records its local time immediately after receiving the message and compares it with the received time. This protocol uses a sequence of synchronization messages from the clock source node to estimate both offset and skew of the local clocks relatively to each other. Implementation of the RBS protocol on IEEE 802.11 networks produces one-hop clock synchronization with 1.85 µs±1.28 µs accuracy.

The precision time protocol (PTP), defined by IEEE standard 1588 [16], is a hierarchical master-slave architecture for clock distribution and time synchronization with highly precise synchronization [8]. In this protocol, the clock server periodically sends synchronization messages. A client replies to the server with a message including the reception time of the first message from the server and the replying time. The server then estimates the delay and offset and sends back this information to the client in another message. The client in its turn uses the new information for local clock adjustments. To avoid or minimize the effects of clock drift on time stamping, client nodes periodically set their time to the server time. PTP is a highly precise synchronization protocol, which can ensure clock skew values in the sub-microsecond range.

An FPGA-based implementation of IEEE-1588 over Ethernet is proposed in [17]. The FPGA is used to implement a compensation circuit for local oscillator drift and to estimate time offset and delay with a Kalman filter. In a 10 Mbps Ethernet, the time difference between reference and client nodes is kept below 100 ns. The number and type of FPGA resources needed by this implementation are not reported.

Another hardware-based implementation of IEEE-1588 is briefly described in [18]. This design was synthesized for TSMC 65 nm technology, resulting in an implementation that uses 15115 cells and runs at 550 MHz. The reported synchronization accuracy is in the sub-microsecond range.

Although the aforementioned protocols are widely used to provide synchronization in different kinds of networks, they are not hardware-aware protocols. In other words, their performance depends on the implementation characteristics of different systems. The main challenge in synchronization is the estimation of the non-deterministic delay of data transmission due to the properties of the hardware. If it is possible to keep variations of the communication and processing delay, at hardware level, to a minimum, then it is possible to achieve synchronization with a simpler and more resource-efficient method, as described next.

3. Synchronization Method

This section presents the proposed synchronization method, which is suitable for a hadrware implementation in wearable sensor networks. The idea is based on eliminating the sources of non-deterministic delays during the time information exchange, and by compensating the remaining deterministic delays at the sender. Periodic exchange of time information is used to compensate for the non-deterministic local clock drifts. With an appropriately designed one-way method, it is possible to achieve, in this application context, the synchronization accuracy of two-way methods with smaller hardware cost. It is assumed from here on that all SNs belong to a wired network and hold similar hardware. In particular, they have the same communication circuits and their local system clocks have the same nominal frequency $f_c = 1/\tau$.

3.1. Principle of Operation

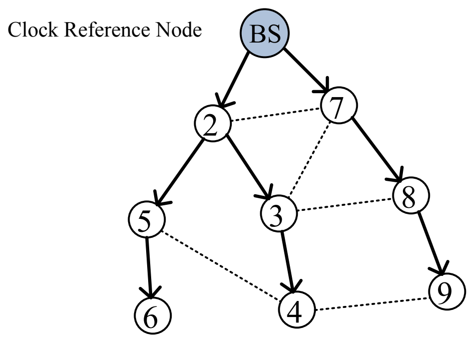

Figure 1: A mesh network of SNs with BS as a base node

The limited availability of resources in sensor networks imposes the need to reduce communication and processing as much as possible. To evaluate a mechanism for synchronization of a wired wearable network, consider the mesh network of SNs shown in Fig. 1. The BS is a server node for collecting data and is also the clock reference. A routing protocol is used to setup the network and route the packets from the sensors to the BS over paths with minimum cost [19]. During network setup, each sensor determines which of its neighboring nodes on the path to the BS has minimum cost and registers it as its *near-node*. The setup defines a tree of paths from BS to all SNs as shown by the bold lines in Fig. 1. Each SN keeps the real-time clock in synchronization with its *near-node* by sending a request message and receiving time information (e.g., SN3 synchronizes with SN2 and SN7 with BS).

To minimize the effects of non-deterministic delays due to the buffering of messages, the synchronization protocol ought to be implemented at a protocol layer below the network layer. In the approach discussed here, the timing information is then processed directly in the MAC layer without any buffering. By avoiding buffering, the delay between SNs will be only caused by the signal propagation and the data sampling process at the receiver node. In this case, the need for a two-way message exchange can be eliminated. To better understand why this simplification can be made, consider the time diagram depicted in Fig. 2(a), which represents a two-way message exchange in a PTP [16] synchronization procedure. Equation (1) and Eq. (2) show the delay and offset estimation for this case.

$$\text{delay} = \frac{(T_2 - T_1) + (T_4 - T_3)}{2} \tag{1}$$

$$\text{offset} = \frac{(T_2 - T_1) - (T_4 - T_3)}{2} \tag{2}$$

In a BAN, the nodes are so close that it can be assumed that all propagation delays are equal (and very small). Assuming then that the non-deterministic delays can be neglected, for a synchronized network it can be shown that Eqs. (1) and (2) simplify to:

$$\text{delay} = s \times \tau \tag{3}$$

$$\text{offset} = 0, \tag{4}$$

where s is the number of clock cycles required to send a message and τ is the system clock period. Since the propagation delay is small and successive clock skew increases are much smaller than τ, Eqs. (3) and (4) will hold for a long period of time, showing that Eqs. (1) and (2) will not provide extra information. Thus, having in mind that the non-deterministic components have been minimized, so the two-way message exchange is not necessary.

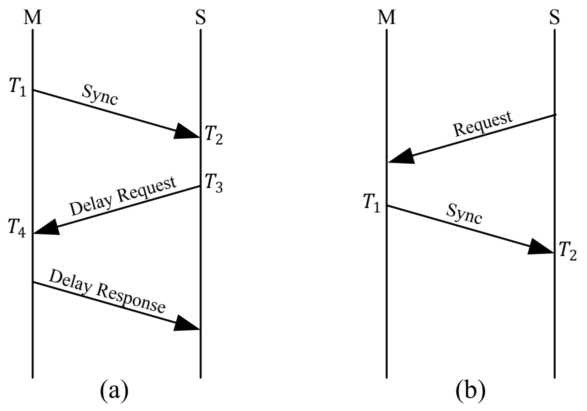

Figure 2: Timing diagram: a) precision time protocol, b) One-way protocol

Now, for the sake of demonstration that a one-way – master-to-slave – message exchange is sufficient for synchronization, let us take the timing digram shown in Fig. 2.b. The master node sends the *Sync* message at time T_1 and the receiver gets it at a time T_2 given by

$$T_2 = T_1 + s \times \tau + d_\ell + S_r(t), \tag{5}$$

where d_ℓ is the signal propagation delay from sender to receiver, and $S_r(t)$ is the delay due to the phase difference between sender and receiver clocks. The average value of $S_r(t)$ satisfies the condition

$$\langle S_r(t) \rangle = \frac{1}{2} \times \tau_d = \tau, \tag{6}$$

where τ_d is the data rate period and, for this system, $\tau_d = 2 \times \tau$. The propagation delay d_ℓ is almost constant and usually in the range of a few nanoseconds for wearable systems with conductive yarns, thus negligible in most cases. The clock offset value is given by the term $s \times \tau$. In the approach proposed here, the master node adds the offset value to the *Sync* message, so that $C_S(t)$, the clock skew between sender and receiver, is

$$C_S(t) = T_2 - (T_1 + s \times \tau) = d_\ell + S_r(t). \tag{7}$$

The average value of $C_S(t)$ is then

$$\langle C_S(t) \rangle = d_\ell + \langle S_r(t) \rangle = d_\ell + \frac{1}{2}\tau_d. \tag{8}$$

To reduce the average clock skew, the term $\frac{1}{2}\tau_d$ in Eq. (8) can be removed by using $s' = s + 1$ instead of s. In this case, the last two equations become

$$C_S(t) = T_2 - (T_1 + s' \times \tau - \tau) = d_\ell + S_r(t) - \tau \tag{9}$$

$$\langle C_S(t) \rangle = d_\ell + \langle S_r(t) \rangle - \tau = d_\ell \tag{10}$$

For a multi-hop connection with h hops between any arbitrary SN and the BS, Eqs. (9) and (10) generalize to

$$C_S^h(t) = h\,(d_l + S_r(t) - \tau) \tag{11}$$

$$\langle C_S^h(t) \rangle = h \times d_\ell, \tag{12}$$

because of the accumulation of clock skews at each intermediate node.

The average clock skew given by Eqs. (10) and (12) is the minimum that can be achieved without further data processing, which dramatically simplifies the hardware needed. Because d_ℓ, the delay due to electrical signal propagation, is smaller than the system clock period, it cannot be compensated by this approach. However, this value is much smaller than the clock period needed in typical BAN applications, which means that the one-way method can effectively accomplish a precise synchronization.

The approach proposed in this work then achieves the minimum clock skew using a one-way method based on accounting for the fixed clock offset between nodes, without calculations or data processing. Because the whole process is implemented in hardware at the MAC level, the value of s is fixed: it depends on the transmitter and receiver circuits, and on the length of the timing message. Therefore, the addition of a suitable constant offset $s' \times \tau = (s + 1) \times \tau$ (with the value of s to be determined for each network according to the specifications of the sensor nodes) can be used to achieve high-quality synchronization in wired, wearable sensor networks.

Equations (10) and (12) assume that local clock oscillators of both nodes have the same frequency. However, the actual frequency of each one will be different. Clock frequency drift of each oscillator is unavoidable and may change with external conditions and aging. After k clock cycles, the time difference due to frequency mismatch is

$$\Delta t_{M,S} = k\,(\tau_M - \tau_S)\,, \tag{13}$$

where $\tau_S = 1/f_S$ and $\tau_M = 1/f_M$ are the system clock periods of master and slave nodes, respectively.

In order to keep Eqs. (10) and (12) valid over time, it is necessary to have a procedure for updating and synchronizing the nodes periodically, since otherwise the frequency differences may accumulate to produce a clock offset that cannot be canceled by the method just described.

3.2. General Protocol Operation

The aim of the proposed synchronization protocol is to ensure that a globally synchronized clock signal *Clk-sync* is available at every sensor node. The main purpose of this signal is to be used for synchronized sampling. In addition, each node maintains a counter that can be used to time stamp the acquired signals.

The simplest way to generate *Clk-sync* is to use an auto-reload counter: when the value of a down counter (called TC in the following description) reaches zero, an active transition of *Clk-sync* is generated and the counter is reloaded with a predefined value (called $TCPR$). The value of $TCPR$ in all SNs must be the same.

Having all *Clk-sync* signals synchronized, is equivalent to saying that all TC counters have to be synchronized. Counter TC is driven by the local system clock, which exhibits some clock skew relatively to the system clock of the other nodes. The procedure described in the previous subsection can be used to keep the average value of the skew within bounds. For this, each SN periodically updates its TC value with the TC value received from the timing message.

The current time stamp at each SN is maintained by an up counter TS (Time Stamping counter), which is driven by the synchronized clock signal *Clk-Sync*. The time resolution of TS depends on the frequency of *Clk-Sync*, which must be chosen according to the sensed phenomena: e.g., the EMG signals of [2] have frequencies below 500 Hz, so a sampling rate of at least 1 kHz is necessary.

It should be noted that using *Clk-sync* for the TS counter only ensures that it counts synchronously. It is still necessary to synchronize the contents of TS at the start of the session with a timing message that includes the global current value of TS. Because each SN synchronizes its *Clk-sync* periodically, it is not necessary to do the same for TS: after a one-time TS synchronization, time stamping will stay synchronized while the system is running.

The process is summarized in Fig. 3. After starting, a SN first updates the TC counter for synchronization of the local *Clk-sync* signal; next, the TS

counter is synchronized only once. Counter *TC* must then be synchronized periodically to compensate for the drift of the local system clock.

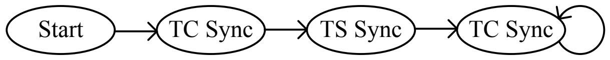

Figure 3: Sequence of MAC messages for clock synchronization generated by a SN.

The protocol defines that synchronization starts when the slave node sends a timing *Request* message to the master node. Note that the same node can act as a slave (to synchronize its own clock signal) and as a master for several other nodes. The synchronization process executed in slave mode is shown in Fig. 4. The node starts by checking the availability of the connection to the master. If it is free, the slave node creates a *Request* in the MAC layer and sends it to the master. The request message specifies whether the request is for *TC* or *TS* synchronization. The slave node will then update its counters with the information received in the *Sync* message from the master node. The status of the received *Sync* message is saved in the *Status* register, so that it can be used in higher level processing by a microcontroller or microprocessor. For example, the value *Status = CRC* indicates that the *Sync* message was received with a CRC error. So, the timing information in the message is not valid and the receiver has to try again.

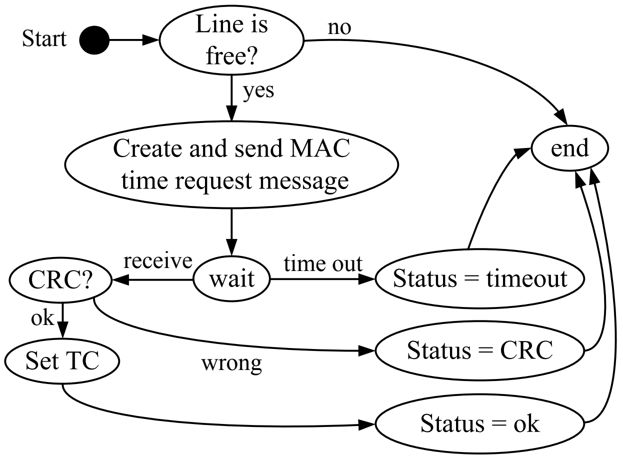

Figure 4: Sending the *Request* and receiving *Sync* messages by slave node.

Master mode operation depends on the type of the *Request* message, as shown in Fig. 5. When a time stamp value is requested, the master simply replies with a message containing its own *TS* value. For a clock counter value request (*TC*), the master replies with the value of the difference *TC-Offset*, where *Offset* is the value used to compensate the deterministic part of delay (the value denoted by s' in Eq. (9)), which is fixed and equal for all sensor nodes. For the prototype described in Section 5, this value is set by software as part of the node's initialization sequence.

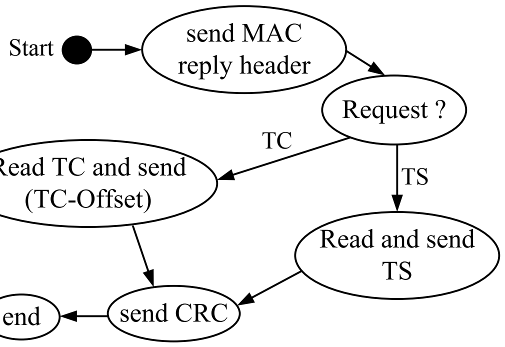

Figure 5: Replying to a *Request* message with a *Sync* message (in master node).

4. The Synchronization Circuit

This section describes a synchronization circuit that is based on the method just described. The circuit is part of the fully-digital sensor node communication system, which has been implemented both in an Actel low-power FPGA and in a CMOS ASIC.

Figure 6 shows the block diagram of the circuit including all necessary modules to send and receive timing information. The three modules in gray (*Signal Detector, TX Line SW* and *Line Driver*) do not belong to the synchronization circuit, but they are involved in the communication process. An 8-bit internal system bus connects the module to the system core, which controls the time synchronization circuit, and reads or writes to the registers (e.g., the real-time time stamp). A 16-bit bus is used to connect the internal modules and to access the two counters *TC* and *TS*. The signal *Clk-sync* is available at one of the output pins. The synchronization circuit also manages a real-time time stamp that is available via the internal bus. The operation of each module is described next.

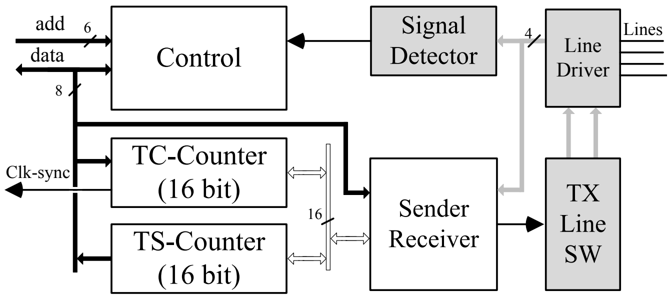

Figure 6: Block diagram of the circuit.

4.1. Control Module

The *Control* module manages all the activities related to time synchronization. It is connected to the system core by the internal 8-bit bus and also to the *Signal Detector*. All the register values can be read or written via this module. The main operation of *Control* is shown in Fig. 7.

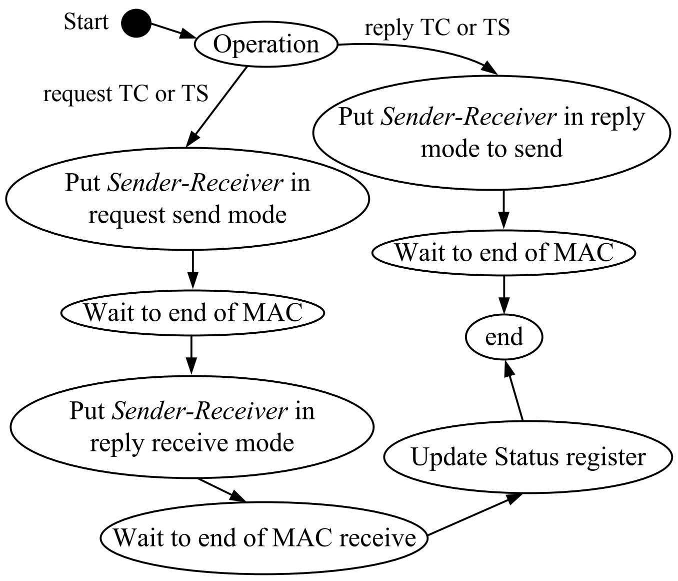
Start
Operation
reply TC or TS
request TC or TS
Put Sender-Receiver in reply mode to send
Put Sender-Receiver in request send mode
Wait to end of MAC
Wait to end of MAC
end
Put Sender-Receiver in reply receive mode
Update Status register
Wait to end of MAC receive

Figure 7: Operation of *Control* module.

Normally, this module is in sleep mode unless one of the following events occurs:

1. *Incoming timing message*: The SNs use several kinds of MAC messages, with timing synchronization being just one of them. The detection of a timing request message from an adjacent SN is carried out by the *Signal Detector* module. This module is responsible for determining the type of the MAC message by decapsulating and processing the MAC header, and forwarding the rest of incoming data to the destination module (which is the synchronization circuit for timing messages). For timing messages, the MAC header also determines whether the request is for updating the *TC* or the *TS* counter. The *Control* module receives the timing message directly from the *Signal Detector* module and creates the appropriate reply message.
2. *Local command*: The request procedure for either *TC* or *TS* can be started by the node's microcontroller with a command sent via the SPI port to the *Control* module. The latter sets up the synchronization module for sending and receiving timing information, and starts the transmission. After sending the message, the *Control* module sets up the *Sender-Receiver* module to directly receive the MAC reply message.

4.2. TC-Counter and TS-Counter Modules

The *TC-Counter* module shown in Fig. 8.a generates the clock signal *Clk-sync* for time stamping and to be used as a synchronized clock by other parts of the SN. Whenever the 16-bit down counter *TC* reaches zero, it is reloaded with the value of the 16-bit *TCPR* register. The periodic *Reload* signal is used to drive the clock generation circuit *Clock_gen*, which produces the reference clock signal *Clk-sync*. As mentioned before, this clock signal is used as input clock for the *TS-Counter* module, and is also available for use as clock reference in other parts of the SN. The output of this module is a pulse signal with duration of 1 or 32 system clock cycles. The wider pulses are necessary for some circuits. For example, when *Clk-sync* is used as an external interrupt of a microcontroller, a short pulse width may not be sufficient to generate a valid interrupt.

The wearable system from [2] is designed to capture EMG signals using a 1 kHz sampling frequency. To generate the *Clk-sync* reference for the EMG signal sensing part from a 20 MHz system clock , the *TC* counter must have at least a 16 bits, which is the length chosen for the current implementation. Therefore, the value of *TCPR* must be in range 1–65535 (33–65535 for the wide pulse mode). The frequency f_{sync} of *Clk-sync* is given by

$$f_{\text{sync}} = \frac{f_s}{1 + \text{TCPR}}, \tag{14}$$

where f_s is the frequency of system clock. For $f_s = 20\,\text{MHz}$, the value of f_{sync} is in the range from 300 Hz to 10 MHz.

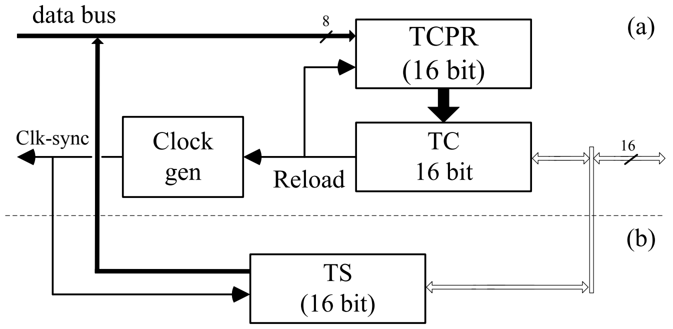

Figure 8: Counters: a) TC-Counter, b) TS-counter

Figure 8(b) shows the *TS-Counter* module. This module includes *TS* and generates a time stamp that can be read by the system, or even written, via a 16-bit internal bus. As for *TC*, other SNs are able to obtain the value of *TS* for updates of their own *TS* counter. The time interval between *TS* overflows is:

$$T_{\text{ov}} = \frac{65536}{f_{\text{sync}}}. \tag{15}$$

In fact, TS does not count the time ticks continuously, instead is periodically reset every time interval T_{ov} (when overflow occurs). However, this will happen simultaneously in all nodes, which means that the information on relative event occurrence, between nodes, is always preserved. The BS can keep the absolute time counting for all events.

4.3. Sender-Receiver Module

In our context, synchronizing two SNs is equivalent to ensuring that their *TC-Counters* have identical values at the same time. This ideal situation cannot be guaranteed at all times, but a small bound on the difference between the counters may be enough for practical purposes. This is achieved by an exchange of special messages between the nodes. All processing of timing messages is performed by the *Sender-Receiver* module. It may operate in one of three different modes:

- *Request for timing information (Fig. 9):* start the synchronization process by sending a request message;
- *Reply to a Request (Fig. 10):* reply to a timing request message;
- *Reception of timing message (Fig. 11):* receive timing information in response to a request message.

4.3.1. Timing Request Message

In order to illustrate the operation of *Sender-Receiver* module, we will consider the situation where node SN3 in Fig. 1 needs to synchronize its *TC* value with its *near-node* SN2.

The process is started by the upper layers of the control software of node SN3 by sending a command (via the internal bus) instructing the *Control* module to activate the request mode. The *Control* modules instructs the *Sender-Receiver* module to start communication with SN2. The configuration of the module in this mode is shown in Fig. 9.

The *Control* module generates both the *EN_req* and *Mac_sel* signals: the first enables the request mode and the second specifies that the request concerns the value of *TC*. Based on the request type (which could be *TC* or *TS*), the *Mac_ Request* module generates the appropriate MAC message, which must then be encoded for transmission. The line encoding scheme used in our implementation is non return to zero inverted (NRZI). Therefore, the output of *mac request* is encoded before driving the line. The module *TX SW* (Fig. 6) forwards the MAC message to the line connected to SN2. At the end of the process, the *Control* module changes the configuration of the *Sender-Receiver* module to reception mode and waits for a reply from SN2.

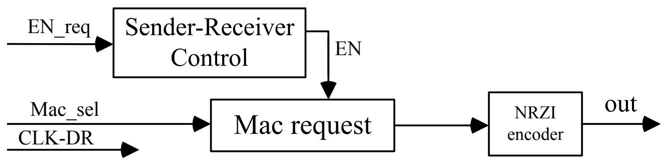

Figure 9: Configuration of *Sender Receiver* to send a request.

4.3.2. Timing Reply Message

When the *Signal Detector* module of SN2 detects the timing message request, it forwards the incoming data to the synchronization circuit. The *Control* module enables the reply mode by asserting the *EN_rep* (Enable-reply) signal. In this mode, node SN2 must send the value of its *TC* counter to SN3, after performing an adjustment to account for the offset between the nodes. The configuration of the *Sender-Receiver* module is the one shown in Fig. 10. In that figure, *B-TCTS* is a 16-bit register for buffering the sampled value of *TC* in node SN2 at the start of reply processing. The 16-bit register *Offset* is used to memorize the offset value, which will be used to compensate for the skew introduced by the communication delay. The value of *Offset* can be managed by the upper software layers through the *Control* module. Module *Sub* is a serial subtracter that is used to subtract *Offset* from the *TC* value buffered in *B-TCTS*. According to Eq. (5), the *Offset* value must be added to the *TC* value that is sent back to SN3. However, *TC* is a down-counter, so the *Offset* value must be subtracted instead. It should be noted that a 1-bit serial subtractor can be used to perform the subtraction of two 16 bit values, since communication is serial and the subtraction can be performed during transmission.

To protect and check the validity of the data at the receiver node, the *CRC5* module generates a 5-bit cyclic redundancy check, which is concatened to the outgoing data. A multiplexor controlled by *Send-Receive Control* selects, in order, the *Mac reply*, *Sub* and *CRC5* modules to build the full reply message at the output, which goes through the NRZI encoder before being sent to node SN3.

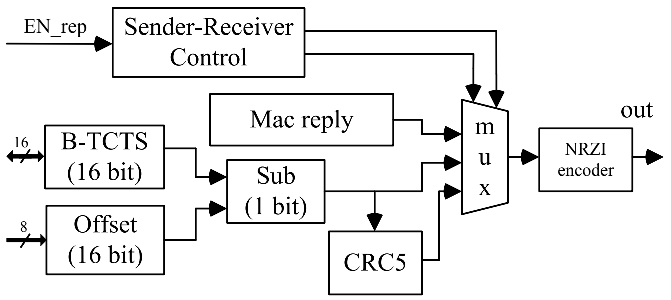

Figure 10: Configuration of *Sender-Receiver* for reply to a timing information request.

4.3.3. Reception of Timing Messages

As mentioned previously, node SN3 changes the configuration of the *Sender-Receiver* module after sending the timing request message. The configuration for reception is shown in Fig. 11. In this case, register *B-TCTS* acts as a Serial-In-Parallel-Out buffer to receive the *TC* value. After the *CRC5* module confirms the validity of the received data, the value buffered in *B-TCTS* is loaded to the *TC*.

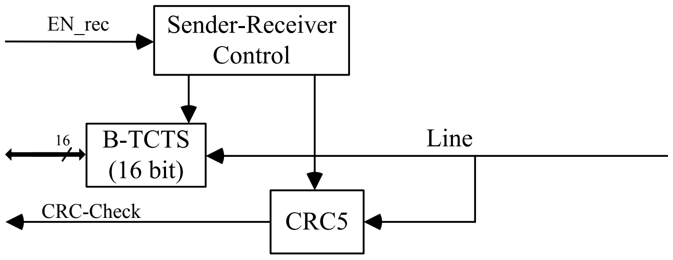

Figure 11: Configuration of *Sender Receiver* to receive timing message.

The process for updating *TS* is the same as the one used for *TC*, with the only difference that, in the reply step, the value of *TS* will be sent without any adjustment. As mentioned in Section 3, *TS* counters inherit synchronization from *Clk-sync* signals; exchanging *TS* values is necessary only for time stamp alignment.

4.4. Implementation characteristics

The ASIC version of the circuit has been fabricated in a 0.35 µm CMOS technology and is shown in Fig. 12. The main parameters of the ASIC are summarized in Table 1. The hardware was described at the RTL level using Verilog HDL. The Actel Libero IDE and the ModelSim simulator were used for functional validation, because the circuit was first prototyped on a Actel FPGA IGLOO nano. Although both ASIC and FPGA versions of the circuit share a single HDL description, there are some differences at the chip level. Nevertheless, the synchronization circuit for both versions is the same. Since the FPGA is equipped with an internal RAM, this block is used in the circuit; the ASIC version includes an embededd 2K SRAM block. Since the RAM block is not used by the time synchronization module both implementations have the same synchronization performance. It should be noted that the FPGA version supports system clock frequencies up to 40 MHz, less than the ASIC's system clock frequencies, which can go up to 70 MHz.

For the ASIC version, which was synthesized with Synopsys Design Compiler from the same HDL source, the standard-cell physical synthesis and post-layout simulation were done with Cadence Encounter and NCSim, respectively. The label *TS* in the figure indicates the approximate area occupied by the synchronization circuit. It uses 971 cells (765 combinational gates and 206 flip-flops), a number which is significantly smaller than the 15115 cells required by the hardware implementation of the more complex IEEE-1588 protocol reported in [18].

Table 1: Main characteristics of the ASIC.

Parameter	**Value**
Technology	0.35 µm
# Logic cells (Core)	5130
# Logic cells (TS)	971 (19 %)
Area (Core)	$0.687\,\text{mm}^2$
Area (TS)	$0.138\,\text{mm}^2$
Embedded RAM	2 KB
Supply voltage	3.3 V
Clock frequency	up to 70 MHz
Data rate	up to 35 Mbps

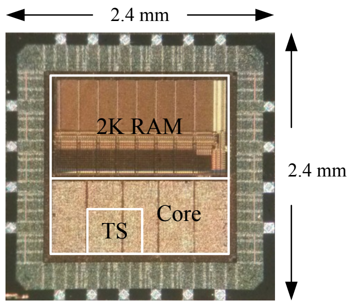

Figure 12: CMOS ASIC (0.35 µm) microphotograph with indication of the synchronization circuit (TS)

5. Experimental Results

This section presents experimental results obtained with the ASIC version of the circuit. The system clock frequency of each node is 20 MHz, which results in a data rate of 10 Mbps. With this system clock frequency, the synchronized clock can be set in the range 305 Hz–10 MHz. The network of SNs used for the measurements is composed by the nodes BS, SN2, SN3, SN4, SN5 and SN6 of Fig. 1.

5.1. Physical Layer Signaling

The physical layer signals generated during message exchange (request and reply) for synchronization of the *TC* are shown in Fig. 13. As mentioned before, the line between the nodes is bidirectional, so the reply message appears on the same line immediately after the request message. The least significant bit (LSB) of the adjusted *TC* value is sent first; the message trailer consists of the CRC check bits.

For the evaluated setup, the entire process, from sending the request message to the end of the reply message, takes 5.2 µs. To keep network nodes synchronized, it is necessary to exchange timing messages periodically, which requires the utilization of system resources, such as channel time, and should be take into account. For an interval between timing messages of 1 ms only 0.52 % of the channel time will be used for synchronization, a very small percentage of the total traffic on the network.

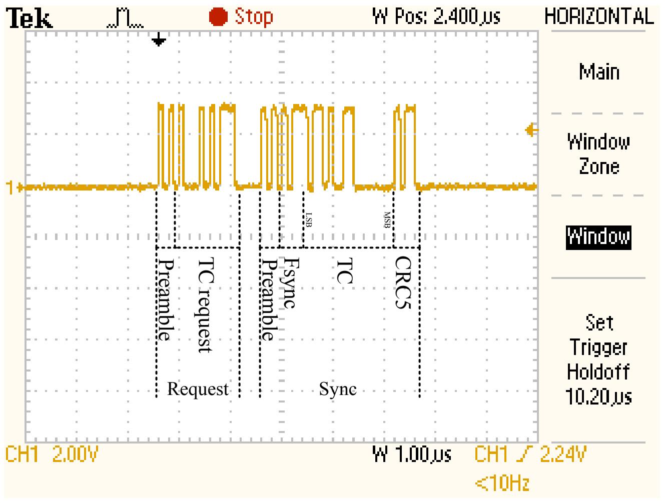

Figure 13: Physical layer signals during timing message exchange between SN2 and BS.

Figure 14 presents part of a logic simulation showing the *TS* synchronization of SN3 and SN4 for a scenario in which *Clk-sync* is configured to operate at 81 kHz. The simulation shows precisely when the changes of the *TS* counter occur. This internal information cannot be obtained by observing the external IC signals. The final *TS* value can be read via the SPI port, but its instantaneous value is not externally accessible. In the figure, *L* indicates the state of the communication line between the two nodes. For each of the nodes, the simulation shows the *Clk-sync* signal and the contents of the *TS* counter.

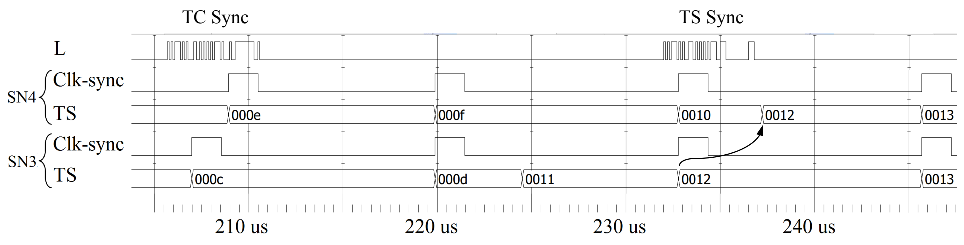

Figure 14: Simulation showing synchronization of *Clk-sync* signal and *TS* counter.

At the beginning of the simulation, both *Clk-sync* (generated by *TC*) and *TS* are out of synchronization. The synchronization of *TC* starts at $t = 205.7\,\mu s$, when node SN4 sends a *Request* message to SN3. The round trip message takes 5.2 µs to complete (ending at $t = 210.7\,\mu s$). The receiver takes 50 ns (one system clock cycle) to check the validity of the message and loads the receiver's *TC* counter at $t = 210.7\,\mu s$, resulting in a synchronized *Clk-sync* at $t = 220\,\mu s$.

To synchronize the time stamp *TS*, node SN4 sends a request at $t = 232\,ns$. Node SN4 replaces its *TS* value with the value received from SN3 (which is 0x0012) at time $t = 237.2\,\mu s$. So, after 237.2 µs both nodes are synchronized. A further noteworthy aspect of the simulation is that during the previous process node SN3 has synchronized its *TS* value with node SN2 at $t = 224.5\,\mu s$ (new value 0x0011).

5.2. One-Hop and Multi-Hop Clock Skew

The one-hop clock skew between SN2 and BS for various values of the parameter *Offset* can be seen in Fig. 15. The oscilloscope signals shown in the figure were generated by using the *infinite time persist* display mode. The system clock of the BS is used as reference in all skew measurements.

The offset value for compensating the constant delay value was measured to be $s = 0x0030$. The observed clock skew variation range is 50 ns, that is one period of the system clock, as described by Eqs. (7) and (11). The measured average one-hop clock skew is 54 ns, i.e., the sum of signal propagation delay and $S_r(t)$. The measurements confirm that the synchronization circuit is able to keep the clock skew in the bounds defined by theoretical analysis.

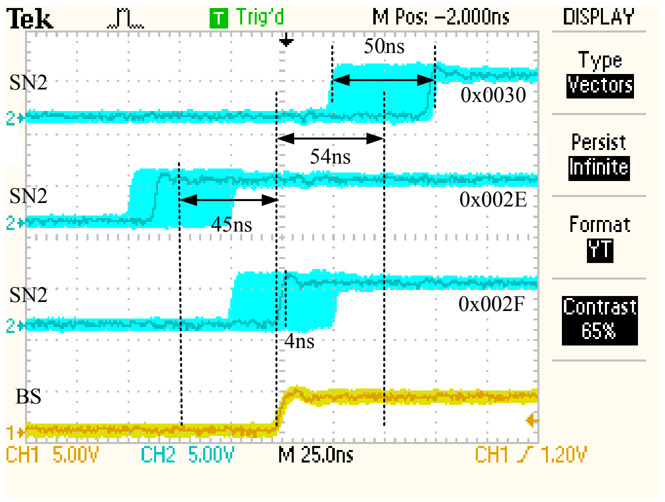

Figure 15: Measured one-hop clock skew for Offset =0x002E, 0x002F and 0x0030.

By setting the *Offset* parameter to $s' = s - 1 = 0x002F$, the *Clk-sync* shifts one clock cycle to the left, causes the skew to decrease. In the setup used for the measurements, the average clock skew is 4.6 ns, which is much smaller than the the total clock skew variation of 54.6 ns. This value is the minimum skew achievable by the proposed method and shows the feasibility of minimizing the average clock skew by selecting appropriate values for the offset. The effect of using the offset value 0x002E is also shown in the Fig. 15, with one more clock period shift as expected. So, by changing the offset value it is possible to shift the *Clock-sync* in both directions by an arbitrary value. Regardless of the offset value and its effect, the peak-to-peak clock jitter is 50 ns.

The clock skew increases with the number of hops. Figure 16 depicts the clock signal at different nodes (the offset value, 0x002F, is the same for all the nodes). As can be seen, and as expected, the average clock skew increases by almost 4.6 ns as the number of hops increases. In agreement with Eq. (11), the clock skew variation range also increases: 50 ns at SN2, 100 ns at SN3 and SN5, and 150 ns at nodes SN4 and SN6. The comparison between the measured and calculated values is shown in Fig. 17, confirming the correct operation of the circuit.

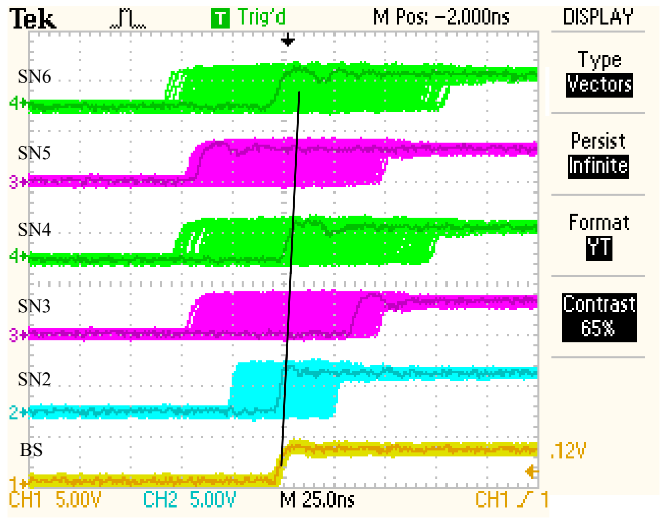

Figure 16: Measured multi-hop clock skew with *Offset* = 0x002F.

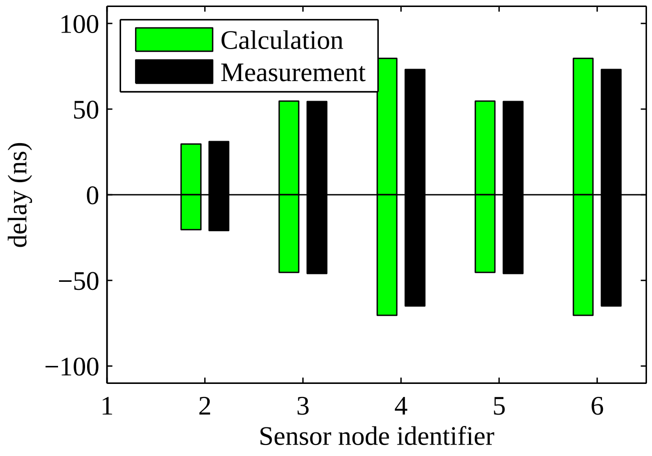

Figure 17: Comparison between the measured and calculated values for multi-hop clock skew with *Offset* = 0x002F.

Figure 18 depicts the measured current consumption of the circuit as a function of the *Clk-sync* frequency from 305 Hz to 5 MHz. The current increase linearly from 0.18 mA to 0.64 mA as the frequency increases. With *Clk-sync* = 1 kHz the total current is almost 0.18 mA. Most of the current is used to drive the *Clk-sync* pin that is available as an output of the ASIC.

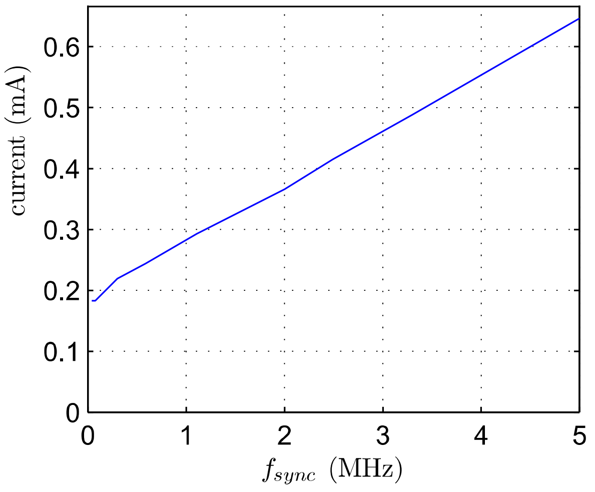

Figure 18: The circuit current consumption for f_{sync} up to 5 MHz

The measured value of 4.6 ns is due to the signal propagation from sender output node to the input port of receiver node, an unavoidable inherent characteristic of the communication path (I/O pads and connecting yarn). In resource-limited systems, such as sensor networks, achieving a smaller clock skew is difficult in practice, even if more sophisticated two-way methods are used, as it would require more processing and a more complex implementation, Therefore, the low-overhead approach described here makes it practical for many wearable systems to achieve a very-low skew in range of sub-microsecond (4.6 ns for the wired, yarn-based application used for evaluation) with a one-way method and without any further processing.

6. Conclusion

The circuit described in this paper was designed for establishing time synchronization between sensor nodes of a wearable system. The synchronization is based on one-way master-to-slave message exchange implemented in the MAC layer, in order to avoid the non-deterministic delays caused by data processing and buffering in the higher levels of the protocol stack. By directly sending and processing the timing information without buffering, the proposed approach leads to an average clock skew of a few nanoseconds and total skew in range sub-microsecond. The circuit generates two synchronized values: a programmable clock signal and a real-time counter for time stamping purposes.

Experimental evaluation with an ASIC implementation obtained an average one-hop clock skew of 4.6 ns, which is the time required for signal propagation from sender output to the receiver input. Based on theoretical calculations, in a multi-hop network, the global average time skew grows linearly with hop count; this is supported by the experimental results. The low skew values provided by this approach satisfy the requirements of many BAN applications. Even for networks whose nodes are 10 hops away from the time reference node, the average global skew will typically be under 50 ns with worst-case skew under 500 ns peak-to-peak. A value of 10 hops exceeds the largest inter-node distance of many, if not all, existing wearable systems. The proposed circuit achieves the best synchronization performance that could be achieved by PTP, but with fewer timing messages and calculations, less complexity, smaller size, and therefore leads to better energy efficiency.

Acknowledgments

This work was financed by the ERDF – European Regional Development Fund through the COMPETE Programme (Operational Programme for Competitiveness) and by National Funds through the FCT — Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project PROLIMB PTDC/EEA-ELC/103683/2008 and through Ph.D. grant SFRH/BD/75324/201.

References

[1] P. Ranganathan, K. Nygard, Time synchronization in wireless sensor networks: A survey, Intl. J. UbiComp 1 (2) (2010) 92–102. 2

[2] A. Zambrano, F. Derogarian, R. Dias, M. Abreu, A. Catarino, A. Rocha, J. da Silva, J. Ferreira, V. Tavares, M. Correia, A wearable sensor network for human locomotion data capture, in: 9th Intl. Conf. on Wearable micro and nano technologies for personalized health; pHealth, 2012, pp. 216–223. 2, 3, 10, 14

[3] F. Sivrikaya, B. Yener, Time synchronization in sensor networks: a survey, IEEE J. Network 18 (4) (2004) 45–50. 2

[4] S. Lasassmeh, J. Conrad, Time synchronization in wireless sensor networks: A survey, in: Proc. IEEE SoutheastCon, 2010, pp. 242–245. 2

[5] Y.-C. Wu, Q. Chaudhari, E. Serpedin, Clock synchronization of wireless sensor networks, IEEE J. Signal Processing 28 (1) (2011) 124–138. 2

[6] I.-K. Rhee, J. Lee, J. Kim, E. Serpedin, Y.-C. Wu, Clock synchronization in wireless sensor networks: An overview, J. Sensors 9 (1) (2009) 56–85. 2

[7] B. Sundararaman, U. Buy, A. D. Kshemkalyani, Clock synchronization for wireless sensor networks: A survey, Elsevier J. Ad Hoc Networks 3 (2005) 281–323. 2, 4

[8] K. Lee, J. Eidson, IEEE-1588 standard for a precision clock synchronization protocol for networked measurement and control systems, in: 34th Annual Precise Time and Time Interval (PTTI) Meeting, 2002, pp. 98–105. 3, 5

[9] D. Mills, Internet time synchronization: the network time protocol, IEEE J. Communications 39 (10) (1991) 1482–1493. 3, 4

[10] S. Ganeriwal, R. Kumar, M. B. Srivastava, Timing-sync protocol for sensor networks, in: Proc. 1st Intl. Conf. on Embedded Networked Sensor Systems, 2003, pp. 138–149. 3

[11] J. Elson, L. Girod, D. Estrin, Fine-grained network time synchronization using reference broadcasts, in: Proc. 5th Symp. on Operating systems design and implementation, SIGOPS, 2002, pp. 147–163. 3, 5

[12] C. Lenzen, P. Sommer, R. Wattenhofer, Optimal clock synchronization in networks, in: Proc. 7th ACM Conf. on Embedded Networked Sensor Systems, SenSys '09, 2009, pp. 225–238. 3

[13] F. Derogarian, J. C. Ferreira, V. M. G. Tavares, A time synchronization circuit with an average 4.6 ns one-hop skew for wired wearable networks, in: 17th Euromicro Conf. on Digital System Design DSD, 2014, pp. 146–153. 3

[14] A. Rowe, V. Gupta, R. R. Rajkumar, Low-power clock synchronization using electromagnetic energy radiating from AC power lines, in: 7th ACM Conf. on Embedded Networked Sensor Systems, ACM, 2009, pp. 211–224. 4

[15] M. Buevich, N. Rajagopal, A. Rowe, Hardware assisted clock synchronization for real-time sensor networks, in: 34th IEEE Symp. on Real-Time Systems RTSS,, 2013, pp. 268–277. 4

[16] IEEE 1588-2008 Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems (2008). 5, 7

[17] W. Huang, Y. Jin, W. Wang, Y. Shi, Y. Zhou, Hardware-based solution of precise time synchronization for networked control system, in: Intl. Conf. on Electronics, Communications and Control ICECC,, 2011, pp. 4324–4328. 5

[18] J. W. Park, J. H. Hwang, W. Y. Chung, S. W. Lee, Y. S. Lee, Design time stamp hardware unit supporting IEEE 1588 standard, in: Intl. Conf. on SoC Design ISOCC, 2011, pp. 345–348. 5, 19

[19] F. Derogarian, J. C. Ferreira, V. M. G. Tavares, A routing protocol for WSN based on the implemention of source routing for minumum cost forwarding method, in: Proc. 5th Intl. Conf. on Sensor Tech. Appl. SENSORCOMM, 2011, pp. 85–90. 7