Pulse Arrival Time estimation in a synchronised Body Sensor Network

Project in Advanced Network Technologies

Kai Geißdörfer
Guillermo von Húnefeld

09/10/2016
Abstract

The Pulse Arrival Time is defined as the time it takes a pulse wave to propagate from the aortic valve to some arterial site, like a finger or a toe. It serves as a valuable indicator for a number of physiological parameters like arterial blood pressure [20] or cardiac output [22].

The proposed project introduces a system to measure the time between two events occurring on different sensor nodes that are wirelessly connected using Bluetooth Low Energy. The target scenario is measuring the Pulse Arrival Time as the time between an R-peak in an Electrocardiogram recording and the corresponding pulse wave peak in a Photoplethysmogram recording using a Body Sensor Network.

An algorithm based on the Lightweight Time Synchronisation protocol is implemented to keep the clocks on both nodes synchronised. Events are timestamped on both sensor nodes and the event timestamps of one node are sent to the other where the time interval between a pair of corresponding events is estimated.

The system is implemented on Nordic Semiconductors nRF51 that combines a state of the art ARM Cortex-M0 based microcontroller with a Bluetooth Low Energy compatible radio in one System on a Chip.

The system is evaluated by calculating the error in time between measured values obtained from the system and a number of reference values extracted from clinical recordings. The resulting mean error of $15.42 \pm 2.59\mu s$ suggests the effectiveness of the proposed solution.
**Acronyms**

**AES** Advanced Encryption Standard  
**AFH** Adaptive Frequency Hopping  
**API** Application Programming Interface  
**ASCII** American Standard Code for Information Interchange  
**BLE** Bluetooth Low Energy  
**CC** Compare/Capture  
**CPU** Central Processing Unit  
**CRC** Cyclic Redundancy Check  
**CSMA/CA** Carrier Sense Multiple Access/Collision Avoidance  
**DMA** Direct Memory Access  
**ECG** Electrocardiogram  
**FIFO** First in First out  
**GCC** GNU Compiler Collection  
**GFSK** Gaussian Frequency-Shift Keying  
**GPIO** General-purpose input/output  
**GPIOTE** General-purpose input/output for Taks and Events  
**HAL** Hardware Abstraction Layer  
**HCI** Host Controller Interface  
**HF** High Frequency  
**IEEE** Institute of Electrical and Electronics Engineers
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Full Form</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRQ</td>
<td>Interrupt Request</td>
<td></td>
</tr>
<tr>
<td>L2CAP</td>
<td>Logical Link Control and Adaptation Layer Protocol</td>
<td></td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network</td>
<td></td>
</tr>
<tr>
<td>LED</td>
<td>Light Emitting Diode</td>
<td></td>
</tr>
<tr>
<td>LTS</td>
<td>Lightweight Time Synchronisation</td>
<td></td>
</tr>
<tr>
<td>MCU</td>
<td>Microcontroller Unit</td>
<td></td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
<td></td>
</tr>
<tr>
<td>PAT</td>
<td>Pulse Arrival Time</td>
<td></td>
</tr>
<tr>
<td>PDU</td>
<td>Protocol Data Unit</td>
<td></td>
</tr>
<tr>
<td>PFS</td>
<td>Post-Facto Synchronisation</td>
<td></td>
</tr>
<tr>
<td>PPG</td>
<td>Photoplethysmogram</td>
<td></td>
</tr>
<tr>
<td>PPI</td>
<td>Programmable Peripheral Interconnect</td>
<td></td>
</tr>
<tr>
<td>PTP</td>
<td>Precision Time Protocol</td>
<td></td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
<td></td>
</tr>
<tr>
<td>RTOS</td>
<td>Real Time Operating System</td>
<td></td>
</tr>
<tr>
<td>RX</td>
<td>Reception</td>
<td></td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
<td></td>
</tr>
<tr>
<td>SoC</td>
<td>System on a Chip</td>
<td></td>
</tr>
<tr>
<td>TX</td>
<td>Transmission</td>
<td></td>
</tr>
<tr>
<td>UART</td>
<td>Universal Asynchronous Receiver Transmitter</td>
<td></td>
</tr>
</tbody>
</table>
1 Introduction

Today’s health care systems are centered around high-technology hospitals and not accessible for a large number of patients in many countries. Even patients who benefit from today’s state of the art equipment impose high costs on the public budgets for health treatment [12].

Body sensor networks for personal health monitoring and automated diagnosis tackle those issues by avoiding extended hospitalisation and utilizing the time of expert physicians more efficiently. The Pulse Arrival Time (PAT) can be measured non-invasively using well established, cheap sensory equipment that can be mounted and operated by non-professionals. The PAT yields a number of interesting physiological parameters that can be used to monitor the state of the cardiovascular system of a patient. For these reasons PAT measurement has been target of recent research efforts and is the motivating application for the proposed project.

For health care applications a body sensor network should affect the patients mobility and agility as little as possible. Connecting the sensor nodes wirelessly and without a mains power supply helps to achieve this goal but imposes severe constraints on the energy budget of the sensor nodes. Especially in the case of continuous monitoring, this limits the choice of technologies and demands careful system design in terms of energy awareness.

The proposed system assumes to be provided with two events in the form of rising/falling edges on a General-purpose input/output (GPIO) pin marking the beginning/end of the PAT. By attaching an analog frontend for recording the underlying signals and implementing some well-known processing algorithms, the detection of those events could easily be added to the proposed solution.

To verify whether the system meets the requirements that are to be defined in a subsequent section an evaluation testbench is designed and implemented that can be used to evaluate the accuracy in terms of error in time.

This paper gives a detailed description of the solution, beginning with an introduction to PAT measurement, definition of requirements, walking through the steps of selection of communication technology and a suitable synchronisation algorithm, assessment of previous work and a detailed description of the implementation in terms of hardware and software. An evaluation system is described and results based on reference data extracted from clinical recordings are presented.
2 PAT estimation

The so-called QRS complex with the most noticeable R-peak in an Electrocardiogram (ECG) recording represents the ventricular depolarisation that leads to a contraction of the ventricle and ejection of the blood through the aortic valve [10]. This R-peak is taken as the beginning point for the PAT measurement. Photoplethysmogram (PPG) serves as an indicator for the blood volume changes in the microvascular tissues. A very basic PPG circuit consists of a light source, typically a Light Emitting Diode (LED) with a wavelength in the red or infrared part of the spectrum and a photodiode to measure the part of light, that is not absorbed within the tissue [8]. The systolic peak on the PPG signal indicates the point where the highest blood volume has been reached and it is used as the end point of the PAT measurement [3].

The European Norm for non-invasive blood pressure measurement EN 1060 demands a mean for systematic error of $\pm 5\text{mmHg}$ and a standard deviation of $\pm 8\text{mmHg}$ [2]. For the proposed PAT measuring method, this translates to a maximum estimation error of $1\text{ms}$ [5].

![Figure 2.1: Illustration of PAT in synchronised ECG/PPG recordings](image)
3 Communication Technology

Several wireless technologies commonly found in Body-Sensor-Networks were considered as possible candidates for implementation of PAT measurements. The most prominent ones which are extensively used are ANT [14], IEEE 802.15.4 [18] and the different Bluetooth standards [15].

ANT was designed for low bit-rate and low power sensor networks, in contrast to legacy Bluetooth which was designed for high throughput, less energy-aware applications. ANT uses adaptive isochronous transmission to allow many ANT devices to communicate concurrently without interfering with each other [7].

IEEE 802.15.4 intends to offer low-cost, low-speed ubiquitous communication between devices. Important features include real-time suitability by reservation of guaranteed time slots, collision avoidance through Carrier Sense Multiple Access/Collision Avoidance (CSMA/CA) and integrated support for secure communications. Devices also implement power management functions such as link quality and energy detection [1].

Bluetooth is a standard wire-replacement communications protocol primarily designed for low-power consumption and for simplified discovery and setup of services between devices [6].

3.1 BLE

Bluetooth Low Energy (BLE) (marketed as Bluetooth Smart) is designed to provide considerable reduced power consumption and cost, while still maintaining a similar communication range like preceding Bluetooth standards. Amongst others, BLE is aimed at novel applications in the healthcare and fitness industries. Despite its missing real-time capabilities BLE is generally well suited for small scale, delay tolerant sensor networks. It allows for various topologies including upcoming mesh networking support.

The distinctive property of BLE as opposed to IEEE 802.15.4 or ANT for the application in medical Body Sensor Networks is the wide availability in consumer devices, like Smartphones and Tablets. This allows for easy integration of internet connectivity, display, storage and computing power with the sensor network without having to provide a second communication technology. Imagine a BSN monitoring a patient in his home, detecting ill conditions and making an emer-
Emergency call via the patient’s smartphone. Or a doctor taking a look at the ECG recording of a patient that is wirelessly streamed to his tablet.

For simplicity and to keep the power consumption as low as possible the decision was made to use the connectionless advertising modes offered by the specification. By avoiding the overhead of using higher layer services the efficiency in terms of transmitted bits is maximised.

**Scannable Undirected Advertising**

For the implemented synchronisation algorithm bidirectional packet exchange is necessary as explained in more detail in section 5. The only connectionless advertising mode offering this type of communication is the *Scannable Undirected Advertising* which is explained in the following paragraph.

BLE uses three advertising channels distributed across the available band. Every *T_AdvEvt* an Advertising event (sending the advertisement packet on all three channels) is triggered. *T_AdvEvt* is comprised of a fixed *advInterval* component between 20 ms and 10.24 s and a random *advDelay* between 0 ms and 10 ms to avoid repeated collisions of two incidentally synchronous advertisers. The time between two consecutive advertisement packets within one event is less than 10 ms.

Right after sending an advertisement the radio listens on the channel expecting to receive a Scan Request packet after the interframe spacing time *T_IFS* = 150 μs. If such a packet is received and addressed to the advertiser it responds by sending a Scan Response packet *T_IFS* after having completely received the Scan Request (see figure 3.1).

The Link Layer packet format includes four fields: the preamble, the Access Address, the Protocol Data Unit (PDU), and the Cyclic Redundancy Check (CRC). The *Scannable Undirected Advertisement* packet as well as the *Scan Response* packet include the advertisers advertising address and each offer 31 octets of arbitrary payload. The only packet directed from the Scanner to the Advertiser (*Scan Request*) contains the advertising addresses of both partners and does not allow payload to be transferred in this direction.
3 Communication Technology

Active Scanning

In Scanning mode a device consecutively listens on the three advertising channels for incoming advertisement packets. In Active Scanning mode the device is able to respond to Scannable Undirected Advertisement with a Scan Request. The active scanner ”shall run a backoff procedure to minimize collisions of Scan Requests [...] from multiple scanners [...]. The backoff procedure uses two parameters, backoffCount and upperLimit to restrict the number of Scan Requests messages sent when collisions occur on Scan Responses messages [...]. Upon entering the active scanning mode, the upperLimit shall be set to one and the backoffCount shall be set to one [...]. [Each time a advertising message is received] the backoffCount shall be decremented by one until it reaches the value of zero [...]. After sending a Scan Request [...] the device shall listen for a Scan Response [...] from that advertiser. If this [...] is not received it is considered a failure otherwise it is considered a success. On every two consecutive failures, the upperLimit shall be doubled until it reaches the value of 256. On every two consecutive successes, the upperLimit shall be halved until it reaches the value of one. After success or failure of receiving the Scan Response the device shall set backoffCount to a new pseudo-random integer between one and upperLimit inclusive [...]. During initiating, [the scanner] [...] listens on an advertising channel index for the duration of the scan window, scanWindow. The scan interval, scanInterval, is defined as the interval between the start of two consecutive scanWindows [...]. The scanWindow and scanInterval parameters shall be less than or equal to 10.24s” [16].
4 Previous Work

With BLE being a comparably new technology targeted at end-consumer devices there hasn’t been much effort on providing a synchronisation service to applications using BLE. One of the few approaches CheepSync is discussed in the following paragraph.

CheepSync

"The CheepSync framework utilizes low-level time-stamping and comprehensive error compensation mechanisms for overcoming uncertainties in message transmission, clock drift and other system specific constraints” [17]. Its synchronisation mechanism is based on undirected, connectionless communication just like the solution proposed in this paper. It uses an Android device as "control unit” and its designers claim an accuracy in the range of 10µs. One of the shortcomings of CheepSync is its implementation on custom host-controller hardware, which is not BLE-compatible. The solution proposed in this paper aims to be completely integrated with the sensor nodes while CheepSync relies on an additional, comparably expensive piece of hardware.
5 Synchronisation Algorithms

A number of synchronisation algorithms were reviewed as possible candidates for the proposed system. The most relevant algorithms taken into account are based on the Precision Time Protocol (PTP), Post-Facto Synchronisation (PFS) and Lightweight Time Synchronisation (LTS). The PTP protocol (IEEE 1588 standard) is used to synchronise clocks in Local Area Networks (LANs). Its synchronisation mechanism is based on the Networks Hierarchy and it provides synchronisation accuracy in millisecond range. The protocol is not specifically targeted at wireless, low power communications but for wired networks of a large number of capable devices [11]. The idea of PFS is not to keep the actual clocks synchronised, but to relate events on a particular device’s clock to the time-line of another device. Usually this algorithm assumes the transmission delay and the drift during the synchronisation to be negligible [4]. In the LTS algorithm the clocks are synchronised, i.e. the offset between the devices is being corrected as often as required. As in PFS, the drift during synchronisation is assumed to be negligible. The design of this algorithm allows for an easy integration with the BLE advertising mechanisms [19].

After comparing these different approaches with each other, LTS was selected as the best option with regards to the requirements. The details of the algorithm are explained in more detail in the following section.

Lightweight Time Synchronisation

The basic mechanism with which LTS attempts to compensate the clock offset between the devices is based on the exchange and timestamping of two particular messages.

One of the nodes sends a packet and timestamps the moment the packet leaves the radio. Then a second device receives this message and timestamps the moment of reception. It will then send a response packet that is again timestamped on both devices.

The original paper assumes that timestamps can be injected into the timestamped packets. This way the sender of the first packet knows all four timestamps after receiving the second message. To inform the other node about the offset a third message has to be sent, containing at least the timestamp of reception of the
second packet. From the four timestamps the offset between the two clocks can be estimated using the following formula:

\[
\hat{O} = \frac{T_3(t_a) + T_0(t_a) - T_2(t_b) - T_1(t_b)}{2}
\]

After calculating, one device can add this offset to its own clock in order to compensate the offset between the devices. This procedure can be repeated periodically according to the synchronisation requirements or triggered by any event.

Figure 5.1: LTS timestamping in BLE Scannable Undirected Advertising

**LTS and BLE Scannable Undirected Advertising**

The **Scannable Undirected Advertising** mode used for the project is well suited to host the LTS functionality. For a more platform-agnostic solution the timestamps are not injected into the packets that they are timestamping. In the given scenario of two equal nodes without any external time reference it is not relevant which node is selected as reference and only one of the nodes needs to estimate the offset. For the proposed implementation the advertiser is selected as reference and the scanner is responsible for compensating the offset. This way the third packet (Scan Response) can be used to transmit the two timestamps from the advertiser to the scanner. A message passing chart illustrating the timestamping and information exchange is found in figure 5.1.
6 Implementation

6.1 Hardware

The one crucial factor limiting the choice of the hardware is direct access to the radio. Most available BLE System on a Chips (SoCs) only offer an Host Controller Interface (HCI) to a proprietary BLE controller stack on a dedicated Microcontroller Unit (MCU) (TI CC2540) or a dual Central Processing Unit (CPU) architecture (TI CC2640), where the user application interfaces with the stack running on a second CPU. In contrast to that Nordic Semiconductors well established nRF51 SoCs offers direct access to the radio peripheral via ARMs Advanced High-performance Bus and only optionally provides a ”SoftDevice” that locks the radio peripheral and acts like a virtual BLE Controller for an application programmer. Essentially this architecture allows for the development of custom BLE stacks or proprietary protocols. Together with a comparably simple hardware interface and potent peripherals like 32bit hardware timers and the Programmable Peripheral Interconnect the SoC was chosen as most appropriate for the proposed project. The following peripherals were used for the implementation and are shortly described here

2.4 GHZ Radio Peripheral

- Gaussian Frequency-Shift Keying (GFSK) modulation
- up to 2Mbps
- up to 4dBm Transmission (TX) power
- Data whitening
- On the fly Advanced Encryption Standard (AES) encryption and CRC
- BLE address whitelisting
- EasyDMA: Write and read from/to Random Access Memory (RAM)
- Events: Ready (ramped up), End (packet received/sent), Address sent/received, whitelist match
6 Implementation

The radio is tailored for BLE protocol needs. It offloads essential parts of the protocol from the CPU by having dedicated hardware for ensuring interframe spacing, device whitelisting, CRC checks etc. The proposed application will use the radio for sending and receiving BLE compliant packets and generating events to trigger the timestamping. For being able to understand the implementation described in section 6.3 it is necessary to describe the functionality and hardware interface of the radio in more detail here: After setting the base parameters like CRC polynom, bitrate, transmit power and Interframe spacing the radio is ready to be activated. The radio has to go through a number of states in order to send or receive a packet (see 6.1). The most efficient way to enable or turnaround the radio from Reception (RX) to TX or vice versa is using shorts. This way the radio can instantly transit to the next state without interaction of the CPU. E.g. after ramping up to RX mode the radio can be set to generate a "READY" event and instantly start listening on the configured channel. When the beginning of a packet is detected the radio will dewhiten the data, check if the address is allowed by the whitelisting policy and write the data into the user-specified memory location. After decoding the address and after reception of the complete packet an event can be triggered and if enabled, the corresponding interrupt routine can be called. The user can then check in a register, whether the CRC of the packet was valid.

PPI

- Connect peripheral events and tasks
- Trigger actions without CPU interaction
6 Implementation

- Additionally raise Interrupt Request (IRQ)
- 12 fixed 16 free programmable channels

The main use of the Programmable Peripheral Interconnect (PPI) in the context of this project is the generation of timestamps with a low delay. The idea is to trigger copying of the current value of a running timer into a register upon reception/sending of a packet. After the packet is processed and the CPU is available, an interrupt handler can be called that reads the timestamp and process it as part of the synchronisation algorithm.

Timers

- 32(1)/16(2) Bit Hardware Timers
- 4 Compare Match Channels per Timer
- Power of two prescalers
- Events: compare match
- Tasks: Start, Stop, Clear ...

The timers are used for two essential parts of the system. One is to ensure the correct timings for the BLE stack. E.g. a device has to repeat some packets with a fixed interval. This sending can be triggered by a compare match event of a timer. The second use of the timers is for the applications main clock that will be kept synchronous between the two nodes. For a detailed description of this clock refer to section 6.3.1. Throughout this report timer refers to the hardware peripheral offering timely incrementing of a register and some additional function whereas clock is used for the more abstract service of providing a value for the current time with reference to some zero-point.

HF oscillator

The system is clocked by a High Frequency (HF) quartz oscillator circuit operating at 16 MHz. For the proposed application the accuracy in terms of frequency stability of this oscillator is relevant and will determine the drift of the clocks on the nodes. As different evaluation boards are used and the manufacturers do not provide detailed information on every component of the circuit it is difficult to find reliable values for the stability of the quartz oscillator in use. However for cheap, unstabilised quartz oscillators under normal operating conditions a maximum error of 50\,ppm was measured by various hobbyists.
6 Implementation

**GPIOTE**

- Interface between GPIO and PPI
- Trigger Events on pin change
- Change pin state on task input

Lets the user assign channels from a GPIO pin change (e.g. rising edge) to a General-purpose input/output for Tasks and Events (GPIOTE) Event that can then be routed to a task in a second peripheral via PPI. For the project this will be used to timestamp the reference events generated by the testbench emulator (see 7.1) and for timestamping the reception of packets.

6.2 Choice of Software

This section describes and justifies the choice of programming language, compiler, drivers, BLE stack and Operating System. C was chosen as an efficient, very popular language for programming embedded systems. Apart from that, Nordic Semiconductors Software Development Kit (SDK), the BLE stacks taken into consideration as well as the possible Operating Systems are all implemented in C. The open source GNU ARM embedded toolchain offers good support for the Cortex-M architecture, is free to use and has a large supporting community.

Nordic Semiconductor offers a Hardware Abstraction Layer and drivers for most peripherals as part of their SDK. It is published under a proprietary license that doesn’t allow to redistribute or modify the code. While the Hardware Abstraction Layer (HAL) solely consists of headerfiles that can easily be exchanged by an own implementation, the drivers confine the implementation of higher software layers, add significant overhead and make the solution dependent on the non-permissively licensed code by Nordic Semiconductor. For these reasons the choice was made to only use the header files defining the HAL and to implement all necessary drivers from scratch to meet the specific requirements.

6.2.1 BLE stack

Regarding the BLE stack there were three options targeting the nRF51 that were taken into consideration.

**Nordic Semiconductor SoftDevice**

The SoftDevice offers the most comprehensive functionality, guaranteed BLE compliance and simple inclusion with a GNU Compiler Collection (GCC) and makefile
Implementation

based project. To guarantee reliability and to impede reverse engineering of its proprietary BLE stack Nordic Semiconductor locks the memory used by the SoftDevice as well as access to the Radio Peripheral and some other peripherals. Most of the functionality is only exposed as system calls that can be interpreted as a virtual HCI. Realization of low level timestamping and immediate packet injection as needed for this project requires disassembling, reverse engineering and manipulating the SoftDevice binary which is out of scope of this project.

NimBLE

Another option is the NIMBLE stack that is part of Apache MyNewtOS. The developers claim to offer the first BLE 4.2 compliant OpenSource BLE stack. While its an interesting option to use an open source fullstack solution, the risk of settling for an only recently published operating system that is poorly documented and comes without experience reports outweighs the advantages for the proposed application.

BLESSED

The BLESSED stack, although offering only a reduced set of the link layer functionality, seemed to be a good tradeoff between ease of use and low complexity on the one hand and available functionality on the other. However after experimenting with the stack and taking a closer look at the source code, it turned out that relevant parts of the specification were not implemented. Specifically there was no random delay in the advertising implementation and no backoff procedure for sending scan requests. Additionally the implementation occupied the only available 32bit timer that was planned to be used as basis for the applications main clock. As even the comparably managable implementation did not work reliably it was decided to discard the BLESSED stack in favor of an own implementation of the corresponding parts of the BLE link layer. A detailed description of this implementation can be found in section 6.3.3.

6.2.2 Operating System

Another decision that needed to be made was whether to use an Operating system. While the inclusion of an Operating System (OS) adds additional complexity and can introduce sources of errors it simplifies the implementation of concurrent parts of the application a lot, especially if those parts have different requirements in terms of delay and jitter. To enable concurrent execution of the different parts of the BLE stack, the synchronisation algorithm, the PAT measurement application and debug facilities FreeRTOS was chosen as a lightweight, realtime Oper-
6 Implementation

Figure 6.2: Overview of the system components and their topology

ating system. The FreeRTOS core implementation consists of Lists, Tasks and the Scheduler and implements a priority based, pre-emptive scheduling algorithm with First in First out (FIFO) lists on each priority level. Additional features of FreeRTOS used for this project are SoftwareTimers, Queues, Semaphores and Mutexes.

6.3 Software implementation

This section describes the implementation of the whole system. All of the described functionality was implemented from scratch as part of the project using only the HAL provided with Nordic Semiconductors Software Development Kit.

The proposed application of measuring time intervals between events occurring on distinct nodes that are wirelessly connected requires a number of modules offering the following functionality:

- Clock - Facility to measure time
- Event detection - Detect events and relate them to points in time
- Communication - Exchange of data between the nodes
- Synchronisation - Common sense of time on different nodes
The modules implementing this functionality are thoroughly described in the next pages. The section is closed by the description of the interval measurement application. An overview of the components and their topology can be found in figure 6.2.

6.3.1 64-Bit software clock

The clock functionality is provided by a custom software clock that is used to timestamp events on each node. As part of the synchronisation procedure this clock is synchronised between both nodes and packets are also timestamped with respect to this clock. The hardware timer on the nRF51 SoC does not allow to arbitrarily change the current value but only to clear the timer at any time. While it is possible to arrange the same reset trigger time on each node taking part in the synchronisation it is more convenient to have one reference clock on one node and correct the clocks on other nodes to match the estimated value of this reference clock. To facilitate this behaviour and to encapsulate handling of overflows, a 64-Bit software clock with a frequency of 1MHz is built around the 32-Bit hardware timer offered by the system. Whenever the software clock is read or modified the value of the underlying hardware timer is added to the software clock value. On every overflow the maximum value of the underlying hardware timer \(2^{32} - 1\) is added to the software clock value. Two functionalities are required for the proposed application.

Timestamping

For the synchronisation algorithm packets must be timestamped with as little delay as possible. The same goes for event timestamping. The timestamping is implemented as a two step procedure. First step is to let the PPI trigger the pushing of the current value of the underlying hardware timer into one of the \textit{Compare/Capture (CC)} registers upon the event to be timestamped. This can be done by connecting the corresponding event of e.g. the radio peripheral (packet address sent/received) to the capture task of the timer via a PPI channel. The second step is to read the timestamp from the application. Upon entering the function the current value of the underlying hardware timer is immediately pushed to a second \textit{CC} register and the hardware timer value is reset to zero. By now there is the old software timer value, the offset of this old software timer value to the event and the offset to the current point in time available to the function. From this information the correct value of the software timer at the current time can be calculated and the time of the event returned to the user.
6 Implementation

Clock offset correction

For correction of the offset of the software clock it is again required to get the real current value of the software clock by adding the value of the underlying hardware timer and then adding the given offset to the result.

6.3.2 Event detection

As described in 7.1 an event is represented as a falling or rising edge on a GPIO pin. The GPIO hardware module can trigger an automated timestamp on the hardware clock on a detected edge. Additionally an interrupt is issued that reads that timestamp as part of the software clock timestamping routine, as described above. The timestamp w.r.t. the synchronised software clock is then send to a queue to be processed by the interval measurement application.

6.3.3 Communication facility

Due to the lack of a suitable open-source BLE stack implementation that allows for low-level timestamping and link-layer packet injection, the respective parts of the specification are implemented to act as a lightweight pseudo-stack to experiment with. The implementation consists of three components. A Timer Module for realizing the timely behaviour as required in the protocol specification. The radio driver to abstract the hardware details and offer a convenient interface for sending and receiving packets. The last component implements the functionality of the BLE Link Layer that is used in this project, namely Scannable Undirected Advertising and Active Scanning.

Hardware Timer module

FreeRTOS comes with an efficient implementation of Software Timers that enable to call registered functions in a timely manner either one time or periodically. While this is a convenient option for implementing timely behaviour on larger timescales, the resolution of one SysTick (1ms as defined for this project) limits the possible use for the time-critical parts of the system.

For the inter-packet timing in the Link Layer a Hardware Timer module was implemented that is heavily inspired by the Software Timer solution found in FreeRTOS. It offers a similar Application Programming Interface (API) and combines the advantages of the headless implementation with the high time-precision of the hardware timer module. As mentioned in section 6.1 each Timer on the nRF51 SoC offers four CC channels that can be used to either trigger an event on a compare match or to read the current counter value by triggering a task that
6 Implementation

pushes the current value of the counter into the corresponding register from which it can be read by the application. To be able to read the timer at any time one channel must not be used for compare match functionality so only three independent compare match events are available for each timer. Another restriction of the implementation is that overflows are not handled which limits the maximum period for a timer to the width of the underlying timer. The proposed implementation uses a 16-Bit timer to continuously run with a prescaled peripheral clock of 1MHz. A user can enable a timer slot by specifying the callback function, a period and the type of the timer (periodical/one-shot). When issuing the start command the current counter value is immediately saved and the corresponding compare channel is set to this value plus the timer period. After enabling the corresponding event and interrupt the timer is active and not using any processing time until the specified point in time. In the interrupt handler a pointer to the corresponding timer is saved and a high priority task woken using FreeRTOS Task Notification facility. The scheduler will immediately load this task if no other task of equal or higher priority is currently running. The now active timer task receives the timer pointer and in case of a periodic timer calculates and sets the next wakeup time. Otherwise the timer is disabled. The registered callback function is then executed from within the tasks context.

This implementation enables execution of the registered functions with low delay and jitter while adding minimum overhead to the application. This was verified by toggling a pin from software and observing delay and jitter using an oscilloscope. However no comprehensive evaluation measurements were recorded. The proposed module has the same limitations as the FreeRTOS software timer implementation, that is callback functions should not block or do excessive processing as the timers are chained in a 'run to completion' manner. Additionally all functions are executed in the same OS priority level.

Radio Driver

The radio driver is responsible for abstracting the details of the hardware and offering services like sending and receiving of packets, setting BLE channels and managing the whitelist entries. It keeps state of the current and the next state of the radio. In many situations it is desired to turnaround the radio after completion of sending/receiving a packet. E.g. after reception of a Scan Request a BLE advertiser usually checks if the Scan Request is addressed to it and then proceeds to send the corresponding Scan Response (see section 3.1). To ensure the correct interframespacing and to offload as much of the processing as possible shorts as described in section 6.1 are used. This way the Radio will already be ramping up to the next state when the Interrupt Routine for the current packet is called.

The crucial part of the driver is built around the Radio Interrupt handler and
will be explained using the example of receiving a packet that is supposed to be answered immediately: If the radio was in receive mode and a packet was completely received, the Interrupt handler checks for CRC errors or not whitelisted senders and immediately aborts the rampup process if the conditions are not met and triggers the RX rampup again. If the packet is not damaged and allowed by the whitelist, the corresponding user callback function is called where an application or a higher layer can read, process or copy the packet. By registering a new input buffer via an API call to the Radio driver it is possible to implement a zero-copying mechanism for incoming packets.

**Link Layer**

The presented implementation of the link layer implements two roles as required to realize the synchronisation application: *Scannable Undirected Advertising* and *Active Scanning*. The implementation of both roles is thoroughly described in the following section. From an application developer's point of view the services must be initialized with a struct of parameters like intervals and pointer to packets and buffers. After initialization the service can be started and stopped and runs asynchronously using FreeRTOS tasks and timers. The application may optionally initialize report queues for a number of events that can be used to react upon events in the stack.

Like mentioned in section 3.1 the time between two Advertising Events (sending one packet on each of the three channels) is composed of a fixed Interval plus a random delay. I.e. there must be one random number available for every Advertising event. For this purpose a low priority thread is started that keeps a queue of random numbers filled thereby providing a random pool to sample numbers from. The fixed time interval between the events ($T_{advEvt}$) is based on a periodic software timer function, that starts a hardware timer with one of the random delay values taken from the queue. The use of a hardwaredtimer allows for a higher resolution of the interval as opposed to the maximum resolution of one SysTick. This second function then changes the period of the underlying hardware timer to the fixed interval specified by the user. Then the radio is set to the next (first) advertising channel and the advertising packet as pointed to by the user is sent. As described in section 6.3.3 the radio is setup to immediately turnaround and start to listen after sending the advertisement. The radio now waits for an incoming *Scan Request* packet that is expected to arrive after $T_{IFS}$ (150 $\mu$s). The reception of the packet is timed out by another hardware timer that disables the radio after a fixed time (internal parameter of the Link Layer implementation). For every received packet that is allowed by the whitelisting policy a previously registered callback function is called by the radio driver from within the radio interrupts context. At the time of entry of this callback the radio is already ramping up for sending the
scan response as specified by the user. In the callback the type and destination address of the packet is checked and if its not a scan request or not directed at the advertisers address the sending is immediately cancelled and the radio set back to listening state. Otherwise the radio is prepared to be disabled after sending the response. Optionally a report is sent to the application via a queue.

After starting the scanning every \textit{scanInterval} a scan window is opened by setting the radio to listen on the next channel and bringing it to the receive state. As already mentioned shorts are used to setup the radio so that it immediately starts to send the scan request after receiving a scannable advertisement. Additionally, a callback function is invoked by the radio driver for every incoming packet that is allowed by the whitelisting policy. The callback function that is executed from within the radio interrupts context checks if the packet is the expected one, as the scanner is either waiting for any scannable advertisement or an outstanding scan response to a previously sent \textit{Scan Request}. In the case of waiting for a \textit{Scannable Undirected Advertisement} the status of the backoff counter is checked and if the scanner is supposed to be backing off, the radio is immediately stopped and reset to receive mode. If the scanner is waiting for advertisements and allowed to send a request by the backoff policy the radio that is currently ramping up for sending the request is set to go into receive mode after finishing sending the request. All events (received advertisement, sent request, received response) can optionally be reported to the application using queues.

\section*{6.3.4 Synchronisation system}

Synchronisation system in this context refers to the facility that automatically synchronises the software clocks as described in section 6.3.1 by adapting the scanners to the advertisers clock. It is composed of three main components that are described in the following section. All components are present on both the scanner and the advertiser but only the lowest layer part (timestamping) is implemented in the same way on both sides.

\textbf{Synchronisation Application}

Application in this context means the highest layer component of the system that is responsible for triggering the synchronisation (correcting the clock once), collecting all necessary timestamps from the lower layer components, calculating the offset between the clocks on the scanner side and adapting its local main clock. The main challenge this component faces is to arrange the meeting of scanner and advertiser in time. The scanner starts scanning after system startup and waits until the first successful packet exchange is signalled by the link layer. The advertiser also starts advertising right after system startup. It then keeps advertising until
it has sent a fixed number of scan responses (N in figure 6.3) or a maximum number of advertisements (M in 6.3) without receiving corresponding requests. An experiment to determine reasonable values for these parameters was conducted and is described in 7.2. After sending those maximum number of packets the advertiser stops advertising and closes the current ”round” of advertising. The application on the scanner can quit scanning for the current round after the first received scan response. However this might lead to M packets being sent by the Advertiser that expects to receive at least N Scan Requests. Determining good parameters in terms of efficient radio usage is out of scope of this project. The Finite State Machine representation of the procedure for the advertiser as well as for the scanner can be found in figure 6.3.

In every scan response packet the timestamp of the first sent advertisement of the current round is included. This way both scanner and advertiser have an anchor point on the assumed to be synchronised clock that can be used to arrange the next meeting in time. There are various options when to arrange the meeting e.g. at fixed intervals, soon enough to not cross some estimated maximum synchronisation error or dependent on the user application payload that needs to be exchanged (see 6.3.5). In section 6.1 a maximum error of the underlying oscillator of 50ppm was given. Consequently in the worst case the clocks of two sensor nodes drift with 100ppm. The minimum synchronisation interval to meet the requirement of an error smaller 1ms is therefore 10s. The current implementation is programmed to synchronize every 5 seconds in order to meet the requirements and to transfer all payload of the evaluation application without sending additional packets (see 6.3.5 for calculations).
Packet timestamping

As shown in section 5 advertisements and scan requests must be timestamped both when sending and receiving in order to estimate the offset between the clocks of the scanner and the advertiser. The radio peripheral can set an event when an address from an incoming packet was decoded or the address of an outgoing packet was sent. This event is used to trigger the copying of a hardware timer value to a certain register by hardware. However this value must be retrieved with respect to the software timer as described in section 6.3.1. For all packets this is done from within the radio callback functions in the link layer implementation where the values are copied either to the output packet buffer (advertiser) or the scan report (scanner).

Timestamp injection

The timestamps of a sent advertisement and a received scan request are collected on the advertisers side and together sent to the scanner as part of the payload in the scan response packet. In order to reduce the payload occupied by the synchronisation data the two timestamps are sent as an offset from the full 64-Bit timestamp of the first sent advertisement of the current synchronisation round as explained above. The maximum delay between this first advertisement and the current one is bound by the known advertising parameters and together with the clock resolution of 1 MHz can be used to reduce the size of each timestamp from 64 to 24 bits. This gain can either be used to increase the efficiency by sending more user application data per packet or by sending shorter packets and saving energy. The total amount of data to be sent from the advertiser to the scanner in one synchronisation round is therefore $4B + 2 \cdot 3B = 10B$.

6.3.5 Interval measurement application

For evaluation of the synchronisation system in terms of the proposed application of PAT estimation a distributed application is implemented that measures an emulated PAT as the time between two events on the scanner and the advertiser. In the evaluation scenario event refers to a rising and a falling edge of one pulse with a known reference width. In a real world application the events would be points in time where the peak of the corresponding signal (ECG/PPG) was detected. Details on the emulation framework can be found in section 7.1. This paragraph describes how the event timestamps are exchanged and how the estimated value is calculated that is then compared to the reference and presented as final error metric in section 7.3.
Piggybacking

As mentioned above, timestamps of events are collected in a queue on both sensor nodes. The timestamps are already related to a common timebase. The last step is to transfer the values from the advertiser to the scanner. The key measure to energy efficiency in this project is to reduce radio active time. The idea is to send usage data (event timestamps) together with synchronisation payload in the same packets. This greatly reduces the energy consumption as additional bits in a packet that is sent anyway come with zero overhead regarding ramping up/down and framing. This approach is referred to as piggybacking throughout this document.

The maximum required communication interval (synchronisation and data exchange) depends on the available payload size that can be exchanged given the proposed BLE communication service (62 Bytes per exchange), the fixed size of synchronisation data (10B) and the usage data rate. Additionally it is upper bound by the maximum synchronisation interval (10s, see 6.3.4). A maximum assumed heart rate of 200 bpm is equivalent to 3.4 events per second. Taking up the idea of timestamp series compression from the synchronisation procedure, only the first timestamp is sent as 64 Bit value. The preceding ones are sent as a 24 Bit offset from this first timestamp. This way timestamps within a $16.7s$ window can be compressed, which is more than the upper bound given by the synchronicity requirements. Adding it all up a maximum of $1+16$ event time stamps can be piggybacked to the synchronisation payload per packet exchange. This leads to a maximum communication interval of $5s$.

Every $5s$ the measurement application on the scanner receives a maximum of 17 event timestamps, that are decompressed with respect to the first event. The difference of those timestamps and the event timestamps in the scanners queue result in the final estimated event intervals.
7 Evaluation

7.1 Testbench

The goal of the testbench is to evaluate the system in terms of error in time. The idea is to emulate a large number of reference PATs with a known duration and to measure this emulated PAT with the proposed system. By comparing the reference value with the measured value a metric for the error can be derived. An overview of the components can be found in figure 7.1.

Reference data

The reference data is extracted from clinical data obtained with a Bluetooth legacy based Body Sensor Network. The data is preprocessed using zero-phase FIR band-pass filters. In a next step the ECG R peak and PPG pulse wave peak are detected using the QRS detection algorithm by Pan and Tompkins [9] and blood pressure peak detection algorithm by Zong et al [21]. By matching each ECG R peak to the resulting pulse wave peak and extracting the time difference between the two peaks 8520 PAT values are stored as the reference database.

Emulator

Emulator refers to the device that outputs the two events marking one PAT interval on a GPIO pin. It is a simple application running on an additional nRF51 board, that polls the Universal Asynchronous Receiver Transmitter (UART) peripheral for incoming characters. After receiving an American Standard Code for Information Interchange (ASCII) encoded value for a PAT it sets up one of the hardware timer peripherals to generate events with the corresponding time interval between the events. The first event is connected to a 'set pin high' task of the GPIOTE peripheral via the PPI and an interrupt is registered to be executed after this first event. In the interrupt routine the task is then changed to set the pin low on the next event. This way the pin outputs a pulse that whose width corresponds to the reference value received via UART. For a small number of pulses the pin output was recorded using an oscilloscope and the time between the rising and the falling edge (defined as the voltage passing \( V_{cc}/2 \)) was measured and compared to the input value. The deviation was smaller than 1\( \mu s \) for all measured values, which
shows that the emulator is suitable for evaluating the system under test in terms of the timely requirements in the ms-range.

Central unit

The central unit has access to the reference database and is connected to the emulator and to the one sensor node that is calculating the estimated time between two events. The functionality is implemented as a multithreaded python application running on a raspberry pi. One thread subsequently reads the reference values from the database, sends them to the emulator via UART and puts them in a queue. The other thread waits to receive the estimated values in \( \mu s \) as sent by the system. Those values arrive in bursts of a number of values every 5 seconds as the data is piggybacked onto the synchronisation packets. As soon as such a burst arrives, the estimated values are extracted from their ASCII representation and the corresponding reference values are received from the queue. By subtracting the values from each other an error for each of the PAT values can be derived and stored for later evaluation. To simplify the merging of events at the scanners side the central node makes sure that no two corresponding events are spread across a synchronisation procedure. I.e. it waits until a synchronisation procedure has finished and then starts to trigger the emulation of pulses as long as the end of those pulses will not happen to be after the next synchronisation event. This way the advertiser always sends N ’start’-events that can be directly matched to N ’stop’-events at the scanners side. This is an unrealistic assumption for a real world PAT measurement application however it does not affect the validity of the results.
### 7 Evaluation

#### 7.2 Packet success rates

For being able to set appropriate parameters for the number of trials of sending a packet, an approximate success rate must be known. An experiment for recording those rates for all three packets is described here.

As mentioned in 3.1 for each advertising event the same advertisement packet is sent on all three advertising channels subsequently until a request is received. If a scanner receives one of these three advertisements and it passes the CRC check it is counted as a successful advertisement otherwise as a failure. In the case of a successful advertisement the scanner can either send a scan request or backoff. Sending the scan request is counted as a success while backoff is denoted as failure. The correct reception of the scan request is checked and also denoted as success or failure. The same applies to the scan response packet. For being able to count the success/failure for each packet type a sequence number has to be introduced and tracked together with the channel on which each packet was sent. Using packet type, sequence number and advertising channel each packet can be unambiguously identified and the error rates as explained above can be calculated on one central unit that gathers the information from both advertiser and scanner for each advertising event.

It has to be noted that the outcome of the experiment heavily depends on the link layer parameters especially the length of the scan window and the RF channel properties including distance between the nodes, obstacles, noise and interference at the moment of each packet transmission. The described procedure is therefore only to be understood as a general way to obtain values for the error rates rather than as to produce valid results. However some example results recorded in an urban residential area are presented in table 7.1.

An experiment with two possible outcomes (success/failure) and an invariant probability, like successful reception of a packet is called a Bernoulli trial. A sequence of statistically independent Bernoulli trials is described by the binomial distribution and its cumulative distribution function can be used to calculate the number of trials necessary for having at least one success with a probability greater than a specified value.

<table>
<thead>
<tr>
<th>Packet type</th>
<th>Percentage of successful packets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advertisement</td>
<td>55.12</td>
</tr>
<tr>
<td>Scan Request sent</td>
<td>99.99</td>
</tr>
<tr>
<td>Scan Request received</td>
<td>99.42</td>
</tr>
<tr>
<td>Scan Response</td>
<td>99.39</td>
</tr>
</tbody>
</table>

Table 7.1: Packet success rates
7 Evaluation

While it is necessary to know the maximum number of times the advertising and the scan request packets have to be sent in order to be received with a probability greater than a specified value, they are acknowledged so they can be repeated until received successfully at least once. However in a real world application the calculation can be used to define an upper limit at which the system can be assumed to be in a broken state with a defined probability.

In the case of the unacknowledged scan response it is necessary to send the packet N times in order to be received at least once with a probability greater than the specified value in any case. Given an arbitrary required probability of at least one success of $99.99\%$ for a correctly received scan response given a received scan request it turns out that four scan responses have to be sent before stopping to advertise.

7.3 Results

This section presents and discusses three fundamental results of the project. The first is the approximate packet loss rates for the three types of BLE packets used for every data exchange. Another is the estimated offset between the clocks on both nodes calculated every 5s as part of the synchronisation procedure. The last result is the timely deviation between the reference values and corresponding measurements as described in section 7.1.

Offset estimation

To verify the assumption about maximum clock drift the synchronisation application as described in 6.3.4 can be used to estimate the offset every $syncInterval(5s)$. This is done in two independent runs: In the first run (blue line in figure 7.2) the scanners clock is corrected after estimating the offset to the advertisers clock whereas in the second run (blue line in figure 7.2) the the clock is corrected only once after startup. With every $syncInterval$ the current estimated offset is output via UART and presented as a graph over time in figure 7.2.

As can be seen from both lines the drift between the two nodes is approximately linear with around $2.8ppm$ slope and way below the worst case value of $100ppm$ given in section 6.3.4. With the specified $syncInterval$ of 5s the maximum estimated offset between the clocks is less than $14\mu s$ for all measurements. This result proves that the implemented synchronisation procedure is generally capable of providing a basis for measuring the event interval with an accuracy within the required $1ms$. 
Emulated PAT error

The results for the 8520 emulated PAT values recorded with the testbench as described in 7.1 are presented as a histogram in figure 7.3. The mean error is 15.42\(\mu s\) and has a standard deviation of 2.59\(\mu s\).

The distribution of errors is not centered around zero which can be explained by a lack of drift compensation. As shown in figure 7.2, the offset between the clocks increases as time passes within a syncInterval so the later within the syncInterval the events occur the larger the error will be. However if the offset could be completely compensated, the error distribution could be expected to be located very close to zero. Apparently there’s a systematic error, that can be either introduced by the testbench or be a result of one of the assumptions of the synchronisation algorithm, inaccurate timestamping or limited timer resolution. However all errors are below a twentieth of the specified requirement of a maximum error of 1\(ms\) in time.

8 Discussion

The results presented in section 7.3 show that the system design and implementation are well suited to solve the proposed problem of online PAT estimation and that the overall project goal of designing and implementing a wireless synchroni-
Discussion

Figure 7.3: Histogram of errors recorded with the testbench

sation algorithm for estimating PATs could be achieved. The results obtained by emulating PAT values from clinical recordings are very promising and well above the defined requirements.

The objective of developing an energy-aware system was also addressed. While the real energy consumption of the system was neither optimised nor evaluated, the core components of the system are designed with energy-awareness in mind and with a minimum overhead regarding communication. This is achieved by using connectionless communication, piggybacking synchronisation and application data and using compression to avoid sending redundant data.

The hardware used for the system is cheap and commercially available and apart from the very thin HAL only free, Open Source software was used.

While the implementation involved a relatively large amount of work for a student project, the careful initial considerations and system design allowed for a straightforward programming and testing of the system.

The biggest challenge was the lack of a mature, open-source BLE stack implementation that allows to add functionality on lower levels of the communication stack. It was very challenging to read through the BLE core specification and implement the necessary parts within the given timeframe. While the implementation works well for the purpose of the project and is able to communicate with commercial BLE devices, it offers only little functionality and the compliance with the BLE specification was not proved.

There’s one obvious starting point for further improvement of the system. As
shown in chapter 7.3 the drift is approximately linear and therefore easy to compensate for. This could be used for improving the accuracy of the event interval estimation and to predict a more reasonable point in time to trigger the next synchronisation procedure without violating the limits imposed by the error requirements. Apart from that, it would be interesting to add analog frontends for ECG and PPG recordings and to implement the peak detection algorithms to evaluate the system in a real world scenario.
Bibliography


