Low-Power Low-Error Wireless Sensor Network for Health and Usage Monitoring Systems

by

Farzad Asgarian

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical and Computer Engineering)
in the University of Michigan
2022

Doctoral Committee:

Professor Khalil Najafi, Chair
Professor Yogesh Gianchandani
Assistant Professor Hun-Seok Kim
Associate Professor Alanson Sample
Pascal Sautier, Hutchinson Research Center

Farzad Asgarian

asgarian@umich.edu

ORCID iD: 0000-0003-1440-6303

Dedication

To my Lovely Wife, *Mahshid,*

Our Sweet Little Angel, Gandom, who made me father, &

Our next Gift from the Angels, who gave us something to fight for.

Table of Contents

List of Tables

List of Figures

Abstract

Condition-based maintenance (CBM) is an information-based just-in-time maintenance approach that is designed to replace traditional periodically-scheduled inspections to reduce cost without sacrificing safety/reliability. CBM relies on data collected under actual operating conditions by a network of sensors in a Health and Usage Monitoring System (HUMS). HUMS is used in a variety of applications, including aircrafts, built-structure monitoring, or systems whose maintenance cost is high, or their failure can have life-threatening consequences. The focus of this research is on a HUMS to wirelessly monitor the condition of laminated bearings linking rotor blades to the rotor axis in a helicopter. A minimum of two HUMS nodes per blade is required to collect inertial data synchronously from different locations with timing error <30 μs. This is needed to accurately extract blades' angles and predict the bearings' health. Wireless connectivity poses challenges in terms of low-error time synchronization and extended operational lifetime.

Battery-powered and BLE (Bluetooth Low Energy)-enabled sensor nodes are custom-designed for the above application. Low-error and low-power synchronization protocols are proposed and tested on the developed nodes, including *BlueSync*, *Discrete Adjustments*, and *AdaptSync*. *BlueSync* is compatible with the BLE standard and reduces synchronization error to <1 μs per 60 s of measurement without requiring any wireless resynchronization between transmitter and receiver. This is the lowest reported synchronization error for BLE, which is achieved by placing the timestamping step at the end of packet transmissions and reducing the uncertainty in transmitter timestamping. *Discrete Adjustments* increases a node's operational

lifetime by reducing the overhead of synchronization calculations, which is the time needed for computing frequency adjustments and is indicative of energy efficiency of calculations. Calculations and adjustments are applied only when needed. Up to 15x reduction can be achieved in calculation times. *AdpatSync* enables both low-power consumption and low synchronization error for long sessions that could run for as long as one hour by using previous timing information as training data to estimate the error and apply the appropriate correction on each node, thus eliminating the need for resynchronizations. It. In a 10-minute recording session, *AdaptSync* reduces the error by 7x compared with standard synchronization methods.

To further increase the node lifetime, a 915 MHz, -61 dBm, 2.8 nW wake-up radio (WUR) is designed in TSMC 65 nm CMOS technology, to listen for wake-up commands when the node is in the *sleep* state. With a passive front-end and only two <1.5 mm^2 off-chip components for matching network, it achieves an estimated range of ~150 m, which is enough for many wireless sensor applications. It has a two-stage wake-up architecture that reduces the probability of false wake-ups, and consequently the power consumption of the node.

A complete system is developed for in-flight tests and has been evaluated on a rotor test bench. Nodes with their required IP67 housing (4×4×2 cm^3) weigh 29 g and provide a lifetime of ~110 days with 1~2 hours recording per week. Collected accelerometers and gyroscopes data are used to extract the blades' angles. Results show angles errors of < 1°, compared with measurements obtained from an optical reference, which are well within the limit for the fault-detection algorithms of this application, and validate that the developed wireless system can be reliably used to monitor the laminated bearings. Integrating the WUR with the nodes could increase their lifetime to ~6 months.

Chapter 1 Introduction

1.1 Health and Usage Monitoring Systems (HUMS)

Condition-based maintenance (CBM) is an information-based maintenance technique which is mainly designed to replace the traditional costly method of periodic scheduled inspections without sacrificing safety and reliability. CBM can be applied to any mechanical system/components and relies on the data collected by a Health and Usage Monitoring System (HUMS) from various components under actual operating conditions. HUMS is a network of sensors that enables CBM by measuring the health and performance of mission-critical components of any desired system. A general architecture of HUMS [1] is shown in Figure 1.1, which includes the following four modules:

- *Data Acquisition*: This module is composed of different sensors and their associated interface electronics to measure the desired parameters of the critical components.
- *Data Manipulation*: This stage includes signal processing and conditioning such as filtering, averaging, and feature extraction.
- *Condition Monitoring*: This module uses fault-detection algorithms to generate condition indicators, based on different sensor data collected over time and from multiple locations on the system/components, indicating the status of the system/components' health.
- *Health Assessment*: This module compares the generated condition indicators with predetermined thresholds. Results can be used to prompt corrective actions.

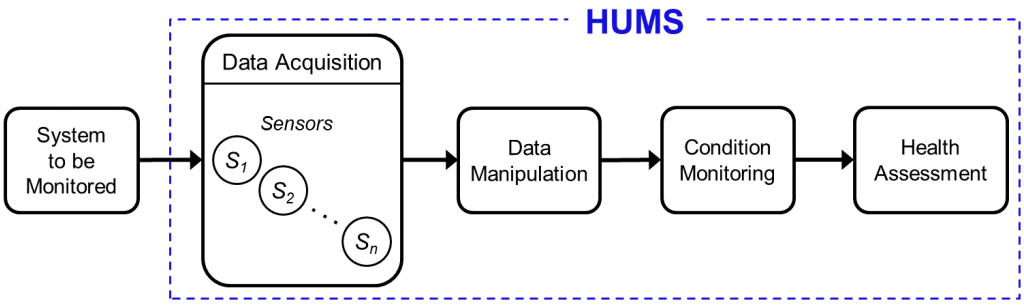


Figure 1.1: Block Diagram of a Health and Usage Monitoring System (HUMS), drawn based on the HUMS architecture presented in [1].

The output of HUMS is then used along with physical models in a prognostic branch to complete the CBM concept and estimate the future health of the system. HUMS and CBM are used in a variety of applications including trains, aircrafts [2], structural monitoring of bridges [3], [4], and mainly systems that their cost of maintenance is high and/or any failure can have life-threatening or catastrophic consequences. The application of this work is part of a helicopter HUMS that will be described in the following sections.

1.2 Helicopter HUMS

Helicopter HUMS automatically monitor the health of mechanical components such as rotor, engines, airframe, and transmission in helicopter systems by measuring dynamic motion and vibration at different locations. In addition to health monitoring, HUMS can also be designed to communicate with the aircraft's data bus to obtain parametric data for usage and event analysis [2]. All of the recorded information during the flight are visualized on the HUMS ground station for evaluation and analysis. This evaluation and analysis is typically not done in real-time and is performed only later to predict any potential future failure and pre-emptively address any issues. The intelligence gained from HUMS helps technicians to pinpoint mechanical faults before they become catastrophic failures, and to make informed maintenance decisions about their aircraft [2].

Benefits of employing HUMS can be summarized as: enhanced safety, increased availability of the aircrafts by reducing unplanned downtimes, and reduced costs of periodic scheduled maintenance and spare parts usage [2], [5]. As an example, a study of HUMS on AH-64 Apaches helicopters found 30% reduction in mission abort, and 20% reduction in maintenance test flights [5]. Similarly, a study on 71 CH-47 Chinook helicopters showed 2957 reduction in maintenance man-hours in 2007-2008 [5].

1.2.1 *Wired vs Wireless HUMS*

In a helicopter HUMS, a variety of sensors, including accelerometers, gyroscopes, tachometers, and blade trackers are used to monitor the dynamic behavior of different components. These sensors are placed at different parts of the helicopter, and their data needs to be transferred to a central processing unit or base station. Traditionally, the communication between the sensors and the HUMS station is provided using wired connections. This wiring significantly adds to the overall weight of the system, and running and securing the wires is a considerable part of system installation [6]. Moreover, the number of possible physical connections to the sensors can be a limiting factor in expanding HUMS [6]. Therefore, a wireless HUMS provides significant potential advantages over a wired one.

In a wireless HUMS architecture, the data acquisition module is typically implemented by an autonomous Wireless Sensor Network (WSN). Wireless connectivity is essential for measuring parameters at different locations of the components that are in hard-to-reach locations, e.g., the rotating blades. The WSN consists of low-power sensor nodes that record different types of data and must transmit them reliably and securely to a base station. For safe and effective integration of the WSN on helicopter parts, there are tight physical constraints such as size and weight. Besides these requirements, one of the crucial challenges of employing a WSN for HUMS is the ability to

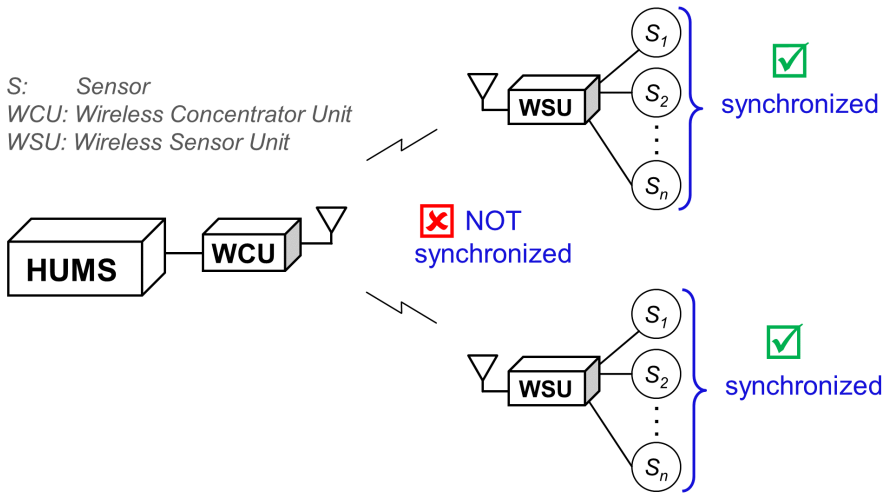


Figure 1.2: Honeywell HUMS wireless architecture [4].

provide synchronization between data points from different sensor nodes. This is because some advanced algorithms used in HUMS, such as synchronous time averaging, for example, rely on accurately synchronizing speed and vibration measurements [6]. In the following section we discuss two approaches that have been reported to deal with the challenge of synchronous measurements in wireless HUMS.

1.2.2 *Time Synchronization in Wireless HUMS*

To address the issue of synchronization when using a WSN for HUMS, an architecture that combines the WSN and the wired system is proposed in [6] by Honeywell Aerospace. In this method, wireless units are placed as close as possible to a group of sensors, and all sensors are connected to their corresponding unit by wires (Figure 1.2). In this case, while there is still some local wiring between the sensors and the nodes, the total wiring is much less than connecting all sensors individually to the HUMS station. In this architecture, synchronization would not be a problem if the installation allows for all parameters of interest to be collected by the same physical unit, i.e., only one wireless node. However, if measurements are conducted by different wireless

nodes, synchronization cannot be achieved with this configuration due to clock offset errors and clock frequency variations between the nodes. The main reason behind developing this architecture was their conclusion that existing commercial wireless technologies were not capable of providing the required microsecond-level synchronization for some advanced HUMS algorithms [6].

Therefore, for applications where physical wiring of the sensors to the same node is not possible, a synchronization mechanism must be designed for the WSN. In [7] a synchronized WSN is used for monitoring aircraft landing gear loads. The time-synchronized network is explained in [8]–[10], and consists of wireless nodes and a base station called wireless sensor data aggregator. Broadcasting a wireless beacon packet in the 2.4 GHz license-free ISM band, combined with precision timekeepers on each node in the system, is employed to synchronize the nodes [7]. In this method, the base station transmits a beacon signal every second. All the wireless nodes then synchronize their sampling times with this beacon signal. A high-precision clock (±3 ~ 5 ppm) is used in the nodes to maintain time between the beacon broadcasts. The maximum timing error of receiving this beacon in different nodes is reported to be ±4 μs [8]. Synchronized sampling within ±30 ~ 50 μs is achieved by using the beacon packet every 10 s, instead of every second.

The process of resetting the error in the nodes by receiving a beacon packet from the base station is called resynchronization. The main problem of this approach is that to reduce the synchronization error the radio has to be used frequently (every few seconds) for resynchronization; no other synchronization mechanism is incorporated in the nodes. Using the radio with this short few second intervals not only increases the average power consumption of the node and decreases its lifetime, but also requires the power supply to provide short duration pulse currents with amplitudes that can be as high as 100 mA for the above system. These high-amplitude pulse currents can degrade the performance and capacity of the limited power source(s)

of the nodes and result in an even shorter lifetime. Shorter lifetime also means more frequent maintenance calls, which we are trying to minimize by implementing a HUMS in the first place.

Therefore, to extend the lifetime of the nodes, we are interested in synchronization methods that can provide the desired timing accuracy with minimum resynchronizations. To achieve this goal, a synchronization protocol must be able to first estimate the difference between the clock of the node and the base station (frequency-drift estimation), and then correct for this error between the resynchronization packets (frequency adjustment). Errors related to the frequency-drift estimation and frequency adjustments are two key factors that contribute to the overall synchronization error and should be minimized. Section 1.4 provides an overview of the intended HUMS application of this thesis, its synchronization requirements, and the synchronization protocols proposed to address the above challenges.

In summary, besides the size and weight constraints, the following attributes should be considered in the design of a WSN for HUMS/CBM applications [11]:

- Energy-efficient electronics for all tasks of the WSN including processing, sensing, harvesting, and communications. The main point here is that all electronics should be designed by considering the available power source(s) and required lifetime for that specific hardware.
- Energy-efficient local processing algorithms to reduce data rate. The key point here is to investigate to what extent performing some processing locally can result in reducing the energy consumption of the nodes. Because, while communication typically requires more power compared with processing, having complex algorithms can also result in more resources and higher power requirements.

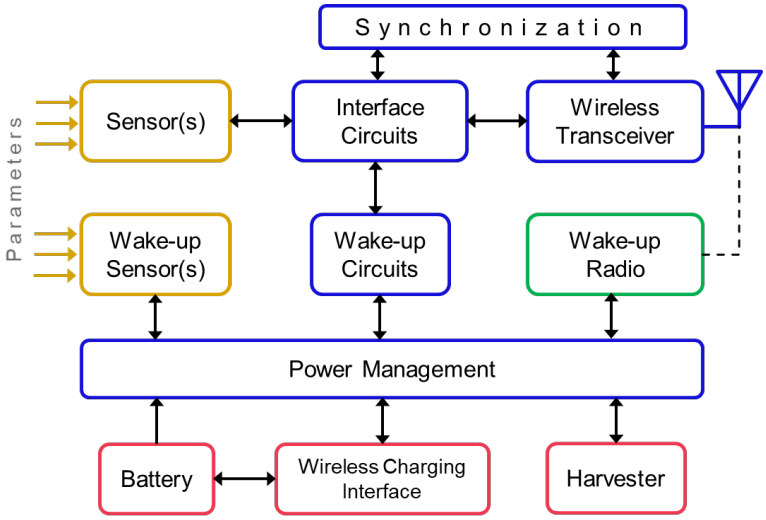


Figure 1.3: Block diagram of a wireless sensor node.

- Communication protocols for both reliable data transfer and synchronized measurements. Synchronization becomes critical when cross-correlation calculations are required for data recorded by different nodes. The accuracy of synchronization depends on the internal clock of the nodes and the implemented synchronization protocol. Both, a low-error and a low-power synchronization protocol is desired to minimize overhead on the power consumption and, consequently, increase the node lifetime.

1.3 Wireless Sensor Nodes

Figure 1.3 shows a general block diagram of a wireless sensor node. A node might have all or some of the shown building blocks. As an example, all nodes need a battery or some type of energy storage device, but they might have a charging interface and/or a harvester module based on a given application.

Nowadays, wireless transceivers are mostly available as system-on-chip (SoC) and blocks like the sensor interface, power management, and synchronization protocols can be implemented on the same platform, i.e., Microcontroller Unit (MCU). In helicopter HUMS, the measurements

are performed during flights or for a short period of time on the ground, and for the rest of the time the nodes can be turned off (*sleep* state) to save power. This scenario is also valid for many other HUMS applications. In this case an additional feature known as wake-up should be added to the nodes so they can be powered down and only activate when they are needed to perform a measurement. We can consider three methods for waking up the nodes:

- *Wake-up Sensor(s)*: These sensors are typically targeting a very specific application and can generate a wake-up signal based on changes in the parameter(s) that they are sensing. An example is an accelerometer programmed to interrupt the MCU whenever the acceleration exceeds a predefined threshold on one axis.
- *Wake-up Circuits*: These are generally low-power internal timers of the SoC or external timing units that can be: (a) programmed to wake up the node at predefined intervals; or (b) used for duty-cycling a higher power module responsible for the wake-up task, e.g., the transceiver or even the wake-up sensor(s). We refer to duty cycling the main transceiver for receiving wake-up command as Wake On Radio (WOR).
- *Wake-up Radio (WUR)*: This is an ultra-low-power receiver designed to be always ON and ready for receiving wake-up commands, sent typically from the base station of the WSN.

Clearly, regardless of the wake-up method, the goal is to achieve a power consumption much lower than the normal power consumption of the node if it were to be ON all the time. Otherwise, the use of this module is not justified as it potentially occupies additional space on the node.

1.4 Research Objective and Contributions

The target HUMS application in this work is monitoring the condition of the laminated bearings on the rotor of helicopters (Figure 1.4). To monitor these bearings, a WSN with a

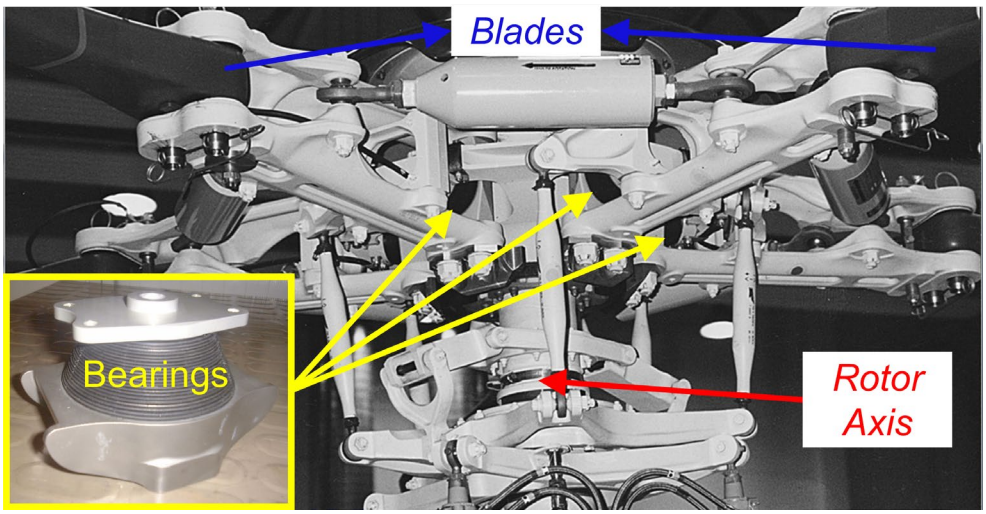


Figure 1.4: Laminated bearings of the helicopter rotor.

minimum of two nodes per blade is required to measure the dynamic motion of rotating blades that are attached to the rotor by these bearings. Each node must have a 6-axis Inertial Measurement Unit (IMU) and consisting of accelerometers and gyroscopes. The preferred full range for the accelerometers is ±32 g, and for gyroscopes is ±4000 °/s. The nodes on the same blade must collect data synchronously, i.e., sample at the same time. The goal is to keep the synchronization error between sampling time by different nodes on the same blade below 30 µs without any resynchronization (i.e., use of the radio) in a 10-minute recording session. The collected data from accelerometers and gyroscopes are reported to a base station. In addition to bench tests on a mock rotor, the complete system including nodes and the base station(s) should have the capability of recording data autonomously during a test flight for approximately two hours. For this purpose, all the components added to the helicopter must have a housing with IP67[1] rating. For each node that will be installed on the blades a total weight of < 100 grams, including the housing and fixtures, is preferred. However, as the base station does not necessarily need to be on the blades and can be

[1] IP code or Ingress Protection Code provides a guideline to the degree of protection provided by mechanical casings and electrical enclosures. The "6" as the first digit indicates "complete protection against dust over extended time", and the "7" as the second digit means "water immersion protection as deep as 1 m for up to 30 minutes".

Accelerometer Range	Synchronization Error	Lifetime	Housing	Weight		Size	
				Nodes	Base Station	Nodes	Base Station
±32 g	< 30 µs	90 days*	IP67	< 100 g	< 200 g	4×4×3 cm^3	12×10×3 cm^3

* assuming 1~2 hours of recroding per week

Table 1.1:Summary of the requirements for the target HUMS application

installed on the rotor, its total weight can be < 200 grams. Similarly, the base station size can be larger up to 12×10×3 cm^3, whereas for the nodes a size of 4×4×3 cm^3 (including housing) is acceptable. With the above weight and size constraints and considering 1~2 hours of recordings per week, continuous operation for 90 days without changing the battery is set as an initial target. To achieve the target lifetime, as discussed in the previous section, a wake-up mechanism is needed in order to be able to put the nodes in the *sleep* state and save power between measurements. From the wake-up methods explained in Section 1.3, the WOR method and a custom-designed WUR are employed in this research. While the WOR is implemented for the target application on the commercial SoC of the nodes, the WUR is a separate ultra-low-power receiver that can be used and integrated in any wireless sensor node. A summary of the requirements for the discussed HUMS application are listed in Table 1.1.

The following contributions by this work enabled us to achieve the above-mentioned goals:

- *BlueSync*: A novel low-power synchronization protocol for significantly reducing synchronization error between nodes and compatible with the Bluetooth Low Energy (BLE) standard and commercial SoCs is proposed. Reducing the timestamping error in both transmitter (base station) and receiver (nodes) is the key factor in reducing the synchronization error. One of the main contributors to the synchronization error in standard communication protocols is the uncertainty in transmitter timestamps, which is reduced in this work by placing the timestamping step at the end of packet transmissions (versus

timestamping just before starting the transmission). Although this method is applied to BLE as a standard protocol, it is also compatible with any other wireless protocol. *BlueSync* can achieve average synchronization errors < 1 μs per 60 s without any resynchronization. This is the lowest reported error for BLE, improving timestamping error by 18x and synchronization error by 100x for equal resynchronization periods.

- *Discrete Adjustments*: Two new techniques are proposed to increase node lifetime by reducing the overhead of synchronization calculations. This overhead is the time needed for computing and applying frequency adjustments during a synchronous task and is indicative of energy efficiency of the calculations, which has not been studied before. The main concept here is to calculate and apply corrections only when they are needed. Up to 15x reduction can be achieved in calculation times. Results show that if error and energy efficiency are equally important, *Discreet Adjustments* provide a better tradeoff by 3~6x.

- *AdaptSync*: This novel approach enables low synchronization errors without using resynchronization packets from the base station in long recording sessions (i.e., > 1 min and as long as one hour). *AdpatSync* uses previous timing information as training data to estimate the error and apply the appropriate correction on each node. By eliminating the need for receiving wireless packets, this method enables both low power consumption and low synchronization error at the same time for long sessions that could run for as long as one hour. In a 10-minute recording session and without using resynchronization, *AdaptSync* reduces the error by 7x compared with standard synchronization methods.

- Hardware, software, and housing for a complete system consisting of BLE-enabled sensor nodes and base stations have been custom-designed and fabricated for the helicopter HUMS application. The system is used to validate the proposed synchronization

algorithms, and meets all the requirements of an in-flight test with the nodes attached to the bearings and blades. The wake-up feature for these tests is implemented by duty cycling the BLE receiver with the low-power timer of the SoC. To the extent possible, the different parts of the system were tested successfully with a mock rotor in conditions similar to those of an in-flight test.

- A new 915 MHz -61 dBm 2.8 nW wake-up radio (WUR) has been designed and fabricated in TSMC 65 nm CMOS technology to increase the lifetime of the nodes during their *sleep* state. The WUR has a passive front-end architecture and a 2-stage address detection block. It requires only two small (<1.5 mm^2) off-chip components for its matching network and achieves a range of ~ 150 m with commercially available antennas. This range is enough for helicopter HUMS and many other WSN applications. The 2-stage wake-up architecture helps reduce the power consumption of the node by reducing the probability of false wake-ups. False wake-ups increase the power consumption of the node in the *sleep* state, as they falsely turn on the main radio of the node with orders of magnitude higher power consumption.

This thesis is organized as follows. Chapter 2 describes the developed BLE-enabled sensor nodes, and detailed implementation of the *BlueSync* synchronization protocol and the *Discreet Adjustments* technique. Measurement results of the proposed protocols using the developed sensor nodes are also presented in this chapter. In Chapter 3 two low-power approaches, including *AdaptSync,* are explained for synchronization protocols. Different wake-up methods, and the design of the WUR along with its measurement results are covered in Chapter 4. Chapter 5 presents the results of testing the developed system consisting of the sensor nodes and a base station on a

rotor test bench. The details of a similar system prepared for in-flight tests are also provided in this chapter. Finally, conclusion and future works are presented in Chapter 6.

Chapter 2 BlueSync: Time Synchronization in Bluetooth Low Energy with Energy-Efficient Calculations

2.1 Introduction

In wireless sensor networks (WSN) providing a common time reference is one of the basic services on which many other services depend [12]. These WSN are widely used in many applications such as structural health monitoring systems [13], and body-area sensor networks (BSN) [14] for rehabilitation and sport medicine. Different techniques have been reported for time synchronization in wireless networks [12], [15]–[22] that are generally based on message passing. One of the first synchronization protocols developed for WSN is the Reference Broadcast Synchronization (RBS) [15]. In this method after timestamping (i.e., recording the timer value) a reference broadcast from the master, the nodes exchange their recorded times (i.e., timer values) with each other. The goal in RBS is to synchronize receivers (i.e., nodes) and one of its main features is that it eliminates the non-deterministic delays associated with the transmitter (i.e., the master). In the Timing-Sync Protocol for Sensor Networks (TPSN) the authors argue that instead of synchronizing a set of receivers, it is better to use the classical approach of doing a handshake between two nodes [16]. By taking advantage of two-way message exchange and performing timestamping at the Medium Access Control (MAC) layer[1], TPSN roughly achieves 2x better performance than RBS [16]. To reduce communication overhead of RBS and TPSN, the Flooding Time Synchronization Protocol (FTSP) was proposed based on one-way message exchange [12].

[1] MAC layer is the layer that controls the hardware responsible for interaction with the wireless transmission medium. In other words, it controls the transmission of data packets over the wireless link.

In FTSP, synchronization between the master and multiple receivers is realized by timestamping a single radio message at both the sender and receiver sides. Combining MAC-layer timestamping and frequency-drift estimation in FTSP results in lower synchronization error (1.5 μs), compared to RBS and TPSN. Frequency-drift estimation means that each receiver or node uses the values of received master timestamp and its own timestamp for a number of radio messages to estimate a relation between the master time and its own time. FTSP has become the de-facto standard for time synchronization [17], and many of its concepts and techniques are utilized in other works.

Nowadays, with the emergence of Internet of Things (IoT), wireless nodes are becoming more resource constrained, which makes implementation of a common time service more challenging. Wireless protocols are also moving towards adding features suitable for sensor-based systems. Bluetooth Low Energy (BLE) is one of these protocols that has been modified for low-power WSN and IoT applications. Moreover, availability in many consumer electronics has drawn a lot of attention to BLE recently. Compared to the classic Bluetooth, different broadcast advertising modes, known as beacons, have been added to the BLE standard. The simplest and most energy efficient advertising mode is the non-connectable undirected option (*ADV_NONCONN_IND*), in which after advertising the desired data, the BLE device does not scan for any packets from other devices. In this mode data can be transferred between the nodes without needing to enter a bonded connection (i.e., pairing). However, there is no time synchronization service in the BLE protocol, and limited works have addressed this issue.

In [22] BLE is used for synchronization of a collaborative brain-computer-interface. The approach is similar to RBS, and as they timestamp in the application layer, their synchronization error of 37.78 ms is relatively large. Ref. [23] discusses how random transmission delays of BLE affect timestamping in synchronization algorithms. In their setup, when two nodes are connected

by BLE, the delay between the transmitter and receiver timestamps has a standard deviation of 2.32 ms. They propose two principles to reduce this time. First, they consider the time difference between receiving and acknowledging a message, while two nodes are connected [23]. This time difference is not affected by the BLE random delays, and has a standard deviation of 0.41 ms. In the second principle, the *connected* event is used and the nodes need to connect and then disconnect for each timestamp. This method decreases the standard deviation to 14.91 µs. However, both approaches require the nodes to make a BLE connection, which increases the communication overhead. This will also affect the scalability of the network as all the nodes in the network must make a connection with a reference node for timestamping. When the number of nodes increases, the time needed for BLE connections will make the approach impractical.

Ref. [24] proposes generating timestamps by adding a few components to the nodes and monitoring the power consumption of a BLE System-on-Chip (SoC). The idea is that during BLE transmission and scanning, the power consumption of the SoC is dominated by the radio. Therefore, by utilizing the internal comparator of the chip, whenever the monitored power passes a configurable threshold, an interrupt is generated to capture the timer value. They implemented this technique on the same BLE SoC that we use in our study, and achieved a standard deviation of 900 ns between timestamps in two nodes. The same concept has been implemented in [25]–[27] with similar results. In [20], [21] a common time service named *Cheepsync* is reported to synchronize BLE advertisers and smartphones as receivers. In *Cheepsync* a general purpose 2.4 GHz transceiver has been modified to work with the BLE standard and serve as advertisers. As also mentioned in [24], modifying a custom radio to work with BLE has helped them remove some of the non-deterministic delays available in BLE chips. With resynchronizing every 100 ms, the timing error of this method is 10 µs, which is relatively high compared to synchronizations on

other wireless protocols [12], [15]–[19]. However, it should be noted that the main goal of *Cheepsync* is synchronizing smartphones that use the nodes; therefore, its applications and constraints are different from other synchronization protocols.

In Section 2.3, we present *BlueSync* [28], a time synchronization service using BLE beacons that is implemented on two commercial SoCs and is compatible with BLE software stacks accompanying the chips. We explain our techniques to overcome the challenges of BLE synchronization including single-channel scan, random BLE delays, and low-error timestamping. Results indicate that by combining our techniques with the concept of flooding proposed in FTSP (i.e., one-way message exchange), *BlueSync* can provide a sub-microsecond timing error (per 60 seconds). To the best of our knowledge this is the lowest synchronization error reported for BLE. Comparing only the timestamping error with the lowest one reported in [24] shows more than 18x improvement on the same hardware.

In previous synchronization protocols, besides frequency-drift estimation, typically frequent resynchronization (or updates) is used to keep the synchronization error low. However, as the reported update intervals are small (from hundreds of milliseconds to few seconds), this approach results in excessive use of radio and consequently high power consumption, which are not practical for emerging low-power applications. Also, it would be more difficult to investigate the effect of drift-estimation techniques on the synchronization error when the update intervals are small. This is because at each resynchronization point: a) the accumulated error resets to the timestamping error; and b) one extra pair of master/slave timestamps is added for estimation.

The main goal in this work is to keep the synchronization error low with minimum radio activity possible in order to consume the least power while achieving low timing error. Target applications are one-hop systems where nodes either wake-up or are turned *ON* to do a

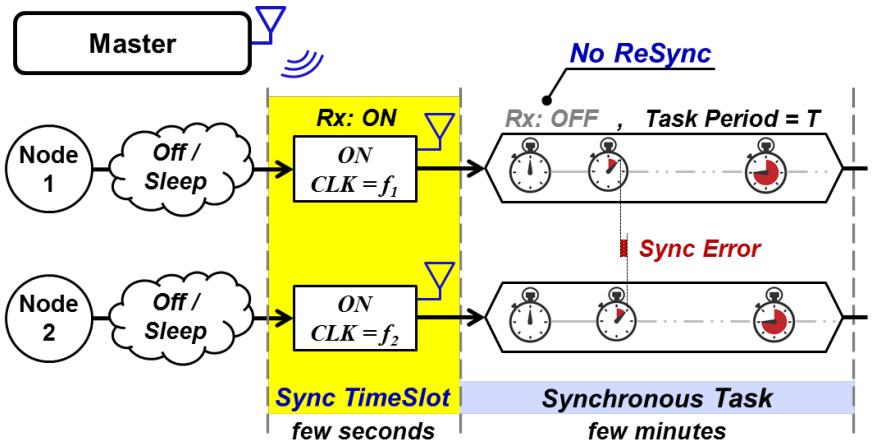


Figure 2.1: Different stages of the system to perform a task at the same time in all nodes. Following startup/wakeup, a short timeslot is dedicated to synchronization packets. After this period, receivers of the nodes are turned off and time synchronization is performed without any resynchronization during the synchronous task.

measurement synchronously, and then go back to sleep or are turned *OFF* to reduce power dissipation and extend battery lifetime. In these applications synchronization speed becomes important too. Therefore, we consider two stages for the proposed system: synchronization timeslot and synchronous task (Figure 2.1). In the synchronization timeslot which is only a few seconds, the nodes receive timing information from the master. During the synchronous task, no resynchronization is performed, and the error is monitored for 10 minutes. We test *BlueSync* with different synchronization packet configurations in the synchronization timeslot. The resulting datasets are used to investigate the tradeoff between synchronization speed and error. We also show how different implementations of frequency-drift management methods can affect both error and energy efficiency. By optimizing calculations, they can be executed up to 5x faster with almost the same error. This translates to energy savings and is especially useful for slower hardware architectures and applications where the lowest error is not necessarily important.

In Section 2.4, a technique called *Discrete Adjustments* is proposed to further reduce the overhead of synchronization calculations, and consequently the energy used for these calculations.

SoC	Processor	Processor Clock	Timer Width	Timer Clock	Tx Current	Rx Current
nRF51	Cortex-M0	16 MHz	16 bit	16 MHz	10 mA	13 mA
nRF52	Cortex-M4	64 MHz	32 bit	16 MHz	6 mA	6 mA

Table 2.1: Comparison of BLE SoCs

Two implementations of *Discrete Adjustments* are discussed and tested with *BlueSync*, which reduce energy consumption up to 15x. This method is not specific to BLE and can be combined with other wireless protocols.

2.2 Designed Sensor Nodes

Among different commercially available BLE SoCs, we selected the nRF5 series from Nordic Semiconductor, which includes nRF51 and nRF52 families. The nRF51 family features a 16 MHz ARM Cortex-M0 processor [29], whereas nRF52 is based on a 64 MHz ARM Cortex-M4 processor [30] and includes versions that are fully compatible with BLE 5. Table 2.1 shows a comparison of two microcontrollers used in this project. One of the benefits of nRF5 series is that without major changes in software/hardware we are able to simulate and test different conditions such as processor speeds and timer widths that are close to some other SoCs. For example, DA14585 from Dialog Semiconductor [31], which is the smallest and lowest power in their BLE family, still runs on Cortex-M0 and has only 16-bit timers (similar to nRF51). Examples of microcontrollers with higher clock speeds and 32-bit timers are Texas Instrument's C2650 (Cortex-M3, 48 MHz) [32] and Silicon Labs EFR32BG13 (Cortex-M4, 40 MHz) [33], which are closer to nRF52 specifications.

The two wireless nodes used in the experiments are custom-designed sensor nodes consisting of a 6-axis inertial sensor (MPU-6050 [34] or ICM-20601 [35]), and either nRF1 or nRF52 as the Bluetooth controller (Figure 2.2). Based on the selected nRF5 family for the nodes,

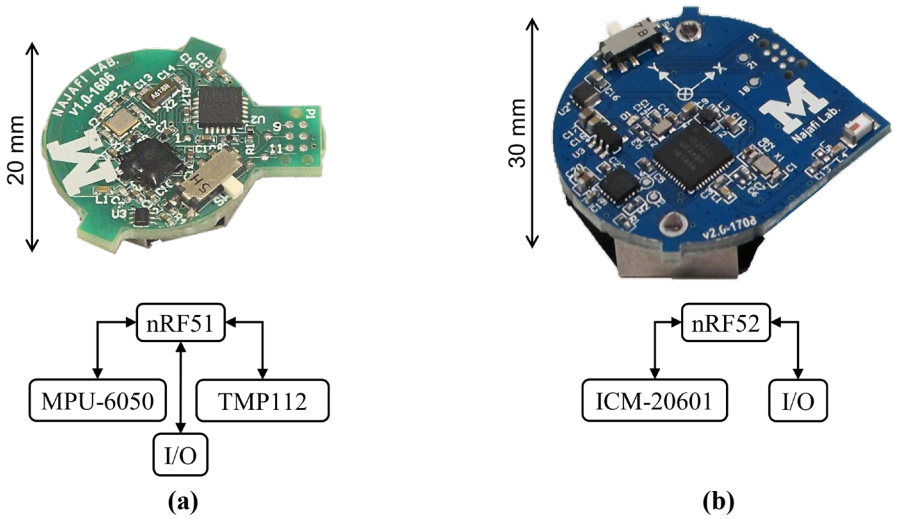


Figure 2.2: Two designed sensor nodes : **(a)** *nRF51 BLE SoC with MPU-6050 (±2000 °/sec gyroscope and ±16g accelerometer), temperature sensor (TMP112), and 3V 140mAh non-rechargeable CR1632 battery;* **(b)** *nRF52 BLE SoC with ICM-20601 (±4000 °/sec gyroscope and ±32g accelerometer), and 3.7V 200mAh rechargeable Li-Ion battery (RJD2450).*

the master role is assigned to the corresponding development kit (DK). We should emphasize that the master/slave terms used throughout this thesis should be viewed from a synchronization standpoint and do not mean that two devices need to be paired in BLE to form a master/slave relation.

For Bluetooth communication, an accurate high frequency crystal is required in all boards. The nRF51 DK contains a 16 MHz ±15 ppm crystal, whereas the nRF52 DK incorporates a 32 MHz ±10 ppm crystal. To be consistent with the masters, crystals with the same frequency tolerance are employed in the nodes. Although, nRF52 operates with a 32 MHz crystal, for both SoCs the maximum the clock frequency available for timers and timestamping is 16 MHz.

2.3 *BlueSync* Implementation

This section is divided into three parts: 1) Timestamping, 2) Frequency-Drift Estimation, and 3) Ticks Adjustment. In each part, both challenges and the proposed solutions are explained.

In Part 1, three main factors that affect timestamping accuracy in BLE are discussed. These are: single-channel scan, hardware events vs software interrupts, and BLE random delay (0-10 ms) for advertising packets.

In Part 2, two methods for frequency-drift estimation are compared: Linear Regression and Average Error. We discuss how different computation requirements for each method such as 64-bit operation and floating-point precision affects calculation time, which eventually translates to energy. By implementing both methods on nRF51 and nRF52, we also show the effect of processor type and clock frequency on these calculation times (Table 2.2). Synchronization error of these different configurations are later compared in the results section (Section 2.6).

The computed drift-estimation parameters are used in scheduling a periodic synchronous task like reading from a sensor. In other words, we need to adjust the timer ticks based on the estimated drift. In Part 3, we explain the techniques to avoid conversion loss during ticks adjustment. Similar to Part 2, the required time for ticks adjustment are compared for different hardware resources and both drift-estimation methods (Table 2.3).

2.3.1 *Timestamping*

Non-connectable BLE beacons are generally broadcast in three advertising channels that are hundreds of microseconds apart from each other. Similarly, the receivers are scanning in all three channels and might receive the data in any of these channels. This uncertainty in channel number results in a large timestamping error and can degrade the performance of any BLE synchronization method compared to previously reported protocols. Thus, the first step is to transmit and more importantly scan in a single channel. In nRF5 series, advertising channels can be selected through a register. But, single-channel scanning can only be achieved by a feature called timeslot API (Application Programming Interface), which allows to directly control the

radio during scan periods (see [36] for sample source code). This method is compatible with the BLE software stack provided for nRF5 series and can be modified to scan in any of the desired three channels.

All the experiments in this work are based on BLE v4.2. However, it is worth mentioning that in the recently released BLE 5, scanning in one channel may be performed by taking advantage of the extended advertising mode. In this newly added feature, after advertising in the three traditional channels (named as primary advertising channels in BLE 5), transmitters may continue advertising in any of the 37 data channels (also known as secondary advertising channels in BLE 5). The first advertisement packet contains information about timing and the channel number of future packets. To make this feature work, receivers must also be able to switch to that specific channel for scanning. Therefore, we can say that all BLE 5 compatible devices should be capable of single-channel scanning as part of the extended advertising mode.

Another important factor in timestamping accuracy is the method by which timer values are recorded. The simplest way is to capture times in the radio interrupt handler, which suffers from software delays and is dependent on the processor speed. A more accurate approach (in nRF5 series) is to employ a feature called Programmable Peripheral Interconnect (PPI), in which a desired *task* can be triggered by an *event* without using the processor. In case of timestamping, timer capture and reset are the *tasks*, whereas the *event* can be any radio event such as *Address* or *End Event*. Figure 2.3 shows the timestamping errors, when two nodes toggle one of their output pins whenever the timer value is captured using the above explained methods. The average timestamping error with the interrupt handler approach is around 4x larger in nRF51 compared to nRF52. This is in line with the fact that the processor clock in nRF52 is 4x faster than that in nRF51. On the other hand, independent of the processor speed, both PPI techniques (i.e, with

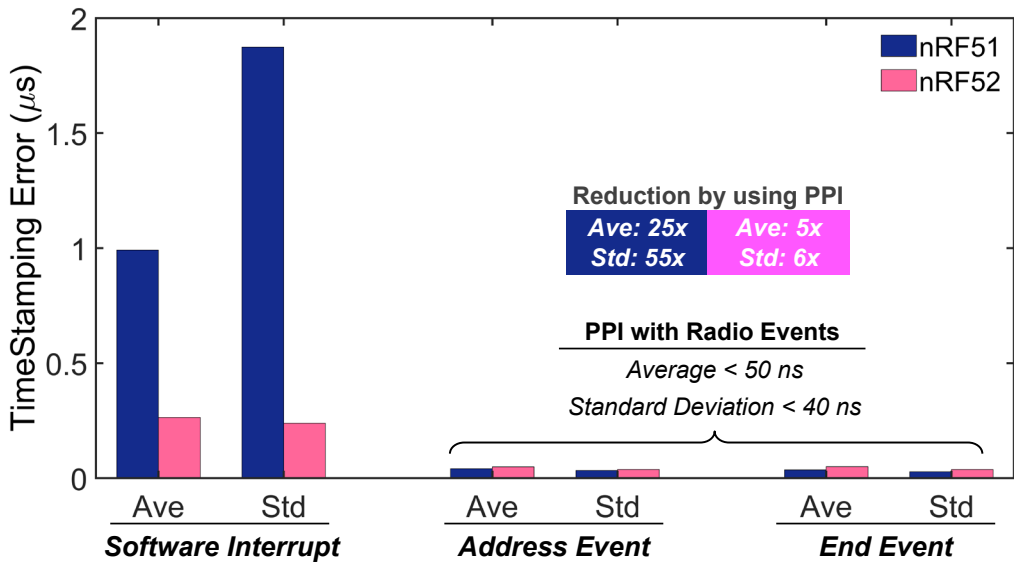


Figure 2.3: Average and standard deviation of timestamping error over 1200 packets on nRF5 SoCs with three methods: using the interrupt handler; PPI with Address Event; and PPI with End Event.

Address or *End Event*) achieve average timestamping errors and standard deviations of less than 50 ns and 40 ns, respectively. Moreover, these errors are 25x (nRF51) and 5x (nRF52) smaller than the software interrupt method. In *BlueSync*, PPI is used along with the *End Event* due to its slightly smaller average error and standard deviation (in comparison with the *Address Event*).

Finally, with the BLE software stack in the transmitter side, i.e., master node, there is always a pseudo-random delay of 0-10 ms at the start of every advertising packet. Moreover, the content of advertising packets cannot be changed exactly before a broadcast. This means that the most recent timer value cannot be sent in the current packet. In the nRF5 series, a radio notification interrupt may be set for a minimum distance of 800 μs from the start of radio activity to update the advertising data. However, as it is not possible to use this interrupt with the PPI feature, it would suffer from software delays in capturing the timer value. To explain how this uncertainty in master timestamps is addressed in *BlueSync*, we use Figure 2.4. In this figure, horizontal lines represent time, and three timelines are shown for a master, and two nodes. The first timeline is for the radio,

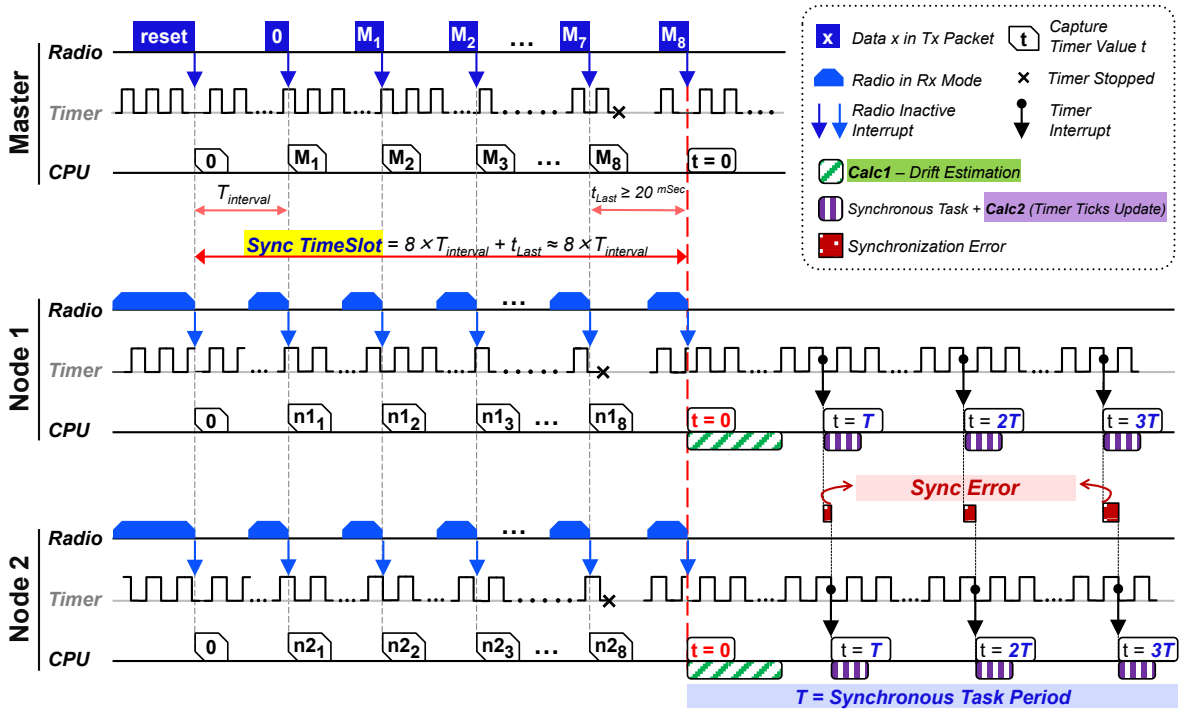


Figure 2.4: BlueSync Protocol with 8 synchronization packets.

and it is showing the contents of the wireless packet for the master, and the status of the radio for the two nodes. The second timeline is for the timer and is showing the LSB of the timer that is toggling with the timer clock frequency. The third timeline is for the CPU and shows the value of the timer that is read by the CPU. The dashed vertical red line divides these timelines to two different parts. On the left side of this dashed line, we have the synchronization timeslot, and on the right side we have the synchronous task. To explain our proposed method for reducing uncertainty in master timestamps, we now only focus on the radio and CPU timelines of the master. As shown in Figure 2.4, after startup or wakeup, the first packet rests the timers, i.e., the CPU timeline is showing a 0. Then either 8 or 16 packets are sent with intervals of $T_{Interval}$. The master timestamps collected after the radio event (shown on the CPU timeline) are transmitted in the next advertisement packet (shown in the radio timeline). With this delayed transmission, there is no need to update the advertisement packet close to the broadcast, and the timestamps are also

collected when the packet has been actually sent, i.e., the random BLE delay is included in timestamping. The last packet in the synchronization timeslot is used to reset timers in master and all the nodes. As this packet is not part of the timestamping for frequency error estimation, it can be sent within 20 ms of the previous broadcast (t_{Last} in Figure 2.4). Considering that packet intervals ($T_{Interval}$) studied here are in the range of 0.1-1 seconds, the last packet has a very low impact on duration of synchronization timeslot and consequently the synchronization speed.

2.3.2 *Frequency-Drift Estimation*

Resetting the nodes' timers with the first synchronization packet is only enough to remove the offset between the nodes. In this case, the frequency drift between the crystals in nodes results in a fast-increasing synchronization error. To reduce this effect, typically more synchronization packets are sent from the master in a known interval. These timestamped packets are helpful in two ways. First, a number of them at the beginning are used to estimate the frequency error between the nodes and the master. This estimation may also be updated upon reception of more timestamps. Second, they can be used for what is called resynchronization. In resynchronization, timers of all nodes are updated with a unique master timestamp or simply the received packet may trigger a timer reset. In both scenarios, the synchronization error at that point goes down to a value close to the timestamping error between the nodes.

In this research, as one of the goals is to reduce radio activity, only a limited number of packets are transmitted in the synchronization timeslot and no resynchronization is performed afterwards. As explained before and illustrated in Figure 2.4, after startup or wakeup, the first packet resets the timers. Then master timestamps are sent with packet intervals of $T_{Interval}$, and the last packet initiates the execution of the desired synchronous task. In Figure 2.4, the start of the synchronous task is exactly on the right-hand side of the dashed vertical red line, where $t=0$ is

shown in the CPU timeline of both nodes. At this point, each node must estimate the frequency error relative to the master based on the captured and received timestamps. This calculation is called *drift estimation* and is shown with green lines inside a rectangle in the CPU timeline of the nodes (Figure 2.4). After this estimation, each node schedules its timer to generate interrupts (vertical black arrows) with the synchronous task period *T*. The required clock ticks for this period *T* are different for each node, as they have different errors relative to the master. At each interrupt, besides performing the desired task, nodes also need to update the ticks required for the next interrupt considering the estimated drift parameters. This is called *timer ticks update* and is shown with purple lines inside a rectangle in the CPU timeline of the nodes (Figure 2.4). Ticks adjustments is explained in the next section 2.3.3. Here, we compare two methods for drift estimation: Linear Regression (LR) and Average Error (AE).

Linear regression has been widely used in previous works and outputs a *slope* and an *offset* computed as

$$slope = \frac{n\sum_{i=1}^{n} M_i - \sum_i M_i \sum_i S_i}{n\sum_i {M_i}^2 - (\sum_i M_i)^2} \quad , \quad offset = \frac{\sum_i S_i \sum_i {M_i}^2 - \sum_i M_i \sum_i M_i S_i}{n\sum_i {M_i}^2 - (\sum_i M_i)^2}, \qquad \textit{Equation 1}$$

where n is the number of timestamps, and M_i/S_i are master/slave timestamps captured at the time of packet i. In AE an error coefficient is calculated by averaging the relative error during each packet interval:

$$C_{AE} = \frac{1}{n}\sum_{i=1}^{n} \frac{(S_i - S_{i-1}) - (M_i - M_{i-1})}{M_i - M_{i-1}} \ , \ S_0 = M_0 = 0. \qquad \textit{Equation 2}$$

With a 16 MHz clock, 16-bit timers only count to around 4.095 ms. Thus, in devices with only 16-bit timers like nRF51, an additional 16-bit counter should be used to form a 32-bit value. Incrementing the counter in nRF5 series can be done through PPI to avoid software delays.

	Method	Timer Width	Time (μs)	
			Single Precision	Double Precision
nRF51	AE	16	317.832	987.506
	LR	16	1513.993	1486.18
	AE	32	310.623	975.229
	LR	32	1506.868	1481.904
nRF52	AE	16	51.375	154.146
	LR	16	41.922	36.562
	AE	32	50.008	153.267
	LR	32	41.484	36.219

Table 2.2: Required time for calculating drift-estimation parameters (with 8 timestamps)

Furthermore, as each second is equal to 16 million ticks, 64-bit integer computations are needed for terms like $\sum M^2$ and $\sum M \times \sum S$ in LR method. Similarly, for the floating-point division in *slope*, both nominator and denominator values exceed 24 bits, which is the maximum number that can be stored without loss in single-precision (SP) floating point format. Any loss in the conversion causes inaccuracy in the results and ultimately increases the synchronization error. Therefore, both *slope* and *offset* must be calculated in double-precision (DP) floating point format. Unlike LR, in AE approach and with maximum packet intervals of 1 second, all terms in the equation are within the range of SP floating point and 32-bit integer formats.

Table 2.2 summarizes the required time for each of these methods, when implemented on nRF5 SoCs for eight timestamps. The impact of the different configurations on synchronization error is discussed in the results sections. For LR, SP results include computing in DP (to avoid large error) and then converting to SP for future ticks adjustments. Hence, SP versions are slightly longer due to this conversion at the end. In nRF51, AE is around 5x and 1.5x faster than LR with SP and DP calculations, respectively. In nRF52, on the contrary, LR runs 1.25x faster with SP and 4x faster with DP operations.

One way to explain these results is to consider that both floating point and 64-bit calculations are executed slower than other integer arithmetic. Floating point operations can slow

down AE, whereas in LR 64-bit operations are dominant. It turns out higher CPU clock and the advanced architecture of nRF52 improves 64-bit calculations more than floating point. It is also clear in Table 2.2 that by switching to nRF52, regardless of floating-point precision, AE times improve 6x while LR benefits from more than 30x improvement. These results may be especially helpful in applications where drift estimation parameters need to be updated as part of the periodic resynchronizations. Also, capturing 32-bit timestamps compared to storing two 16-bit timers can save a few extra shifts in calculations. But, as shown in Table 2.2, it has a negligible effect on timings. Note that nRF51 does not have a 32-bit timer and the values in the table are just for comparison.

2.3.3 *Ticks Adjustments*

Generally, two concepts can be considered for time synchronization. In one approach, nodes store the estimation parameters and whenever an event of interest happens, they mark it with their local clock. Then the captured value is converted to the global clock and might be shared within the network or used for further processing. In another approach studied in this paper, similar to Figure 2.4, nodes need to periodically perform a desired task at the same time, e.g., read from a sensor. In this case, each time the task is carried out, the time of the next execution must be calculated with the frequency drift taken into account. This is achieved by applying AE or LR methods to the total ticks required for the next point in time. The floating-point outcome is then converted to integer and is set as the timer interrupt.

The timer may either increment continuously (without being cleared) or start from zero at the beginning of each task interval. Resetting the timer can be especially useful in devices with 16-bit timers, where another counter/register has to be used for intervals more than 4.095 ms. The reason is that in this setting when the lower 16-bits of the interrupt value is very small, its interrupt

	Method	Timer Width	Reset Timer	Time (µs)	
				Single Precision	Double Precision
nRF51	AE	16	✓	16.92	71.461
	LR	16	✓	21.892	82.325
	AE	16	×	18.776	72.523
	LR	16	×	23.483	81.031
nRF52	AE	16	✓	3.208	7.295
	LR	16	✓	3.92	9.094
	AE	16	×	3.892	8.01
	LR	16	×	4.63	9.099
	AE	32	×	2.934	7.121
	LR	32	×	3.683	8.606

Table 2.3: Required time for calculating ticks adjustments

will happen very close to the counter interrupt responsible for the higher 16-bits. This situation may delay execution of the task and lead to larger synchronization errors. However, when the timer is restarted each time, the total ticks for the task interval can be divided in a way that the interrupt does not happen at very small numbers. For example, a 10 ms interval (160000 ticks) used in the experiments here, may be divided to two 64000 and one 32000 ticks for the last interrupt/reset.

Furthermore, similar to *slope*/*offset* calculations, adjustments with SP floating points may suffer from conversion loss for ticks larger than 24 bits. To avoid this, instead of directly computing total adjusted ticks, the ticks difference from the last point should be estimated. In AE, the error coefficient provides this and in LR, *slope* must be replaced by *slope-1*. By using this technique, even with 100 ppm variation between the clocks, the ticks difference would be less than 24 bits for more than two hours.

Table 2.3 shows duration of ticks adjustment operations for both AE and LR methods with a task interval of 10 ms. Generally, when using 16-bit timers, there are some additional instructions to program/handle the extra resets (see nRF51 results in Table 2.3). Moreover, SP computations are more than 3x and 2x faster in nRF51 and nRF52, respectively. Adjustments are also between 10-25% faster with AE compared to LR (depending on the settings). Considering the repetition of

these adjustments, reducing the CPU time dedicated to them can be beneficial to energy efficiency of the nodes. In the next section, we describe two methods to further reduce the total time spent on ticks adjustments.

2.4 *Discrete Adjustments*

Typically, adjustments are recalculated each time the task is carried out. But we should note that the updated timer interrupt is a conversion of the adjusted ticks in floating point to integer. This implies that depending on the value of the first decimal fraction, it might take up to ten additional task periods for that digit to affect the timer value. Furthermore, as the clock period is only 62.5 ns, the error caused by not immediately including decimal fractions will not be significant, especially for tasks that are planned to run more than a minute. Therefore, in contrast to continuously recalculating the adjustments, we propose *Discrete Adjustments* with the following two implementations.

2.4.1 *One Update per Hundred Runs*

In this method, updates are performed after every one hundred task executions, thus reducing the number of calculations by one hundred times. In between the updates, an adjusted value of the task period is used as the timer interrupt. To avoid adding this period to the timer value every time, the timer is periodically restarted (with the same interrupt value). When computing new adjustments, we must exclude the ticks differences already included in the previous updates and the 99 adjusted periods from the past update.

Figure 2.5 shows the pseudo code of this method, and Table 2.4 includes average times needed to run this method after each timer interrupt. These values are composed of a) a fixed time for checking the count register and updating the interrupt registers; b) adjustment calculation time,

After the counter is reset with last synchronization packet:

- Compute deviation (**Δ**) from desired frequency (equals to C_{AE} in AE and *slope-1* in LR)
- Compute ***ticks_error_T*** for one task period (=*Δ×160000* for T=10ms)
- Compute ***ticks_T*** for one task period (=*160000+ticks_error_T*)
- Update counter interrupt value with *ticks_T*
- Initialize ***count*** to 0; Initialize ***ticks_passed*** to 160000; Initialize ***ticks_added*** to 0;

In the timer interrupt handler:

- ***count*** += 1; ***ticks_passed*** += 160000
- if ***count*** equals to 99
  - Compute ***ticks_error*** for ***ticks_passed***
  - Update counter interrupt value with (*160000 + ticks_error - 99×ticks_error_T - ticks_added)*
  - Set ***ticks_added*** to ***ticks_error***

  else if ***count*** equals to 100
  - Update counter interrupt value with ***ticks_T***
  - Set ***count*** to 0

Figure 2.5: Pseudo code of one update per hundred runs (Discrete Adjustments) with tasks period of 10 ms (i.e., 160000 ticks with 16 MHz clock).

	Method	Timer Width	Time (µs)	
			Single Precision	Double Precision
nRF51	AE	16	3.971	4.519
	LR	16	4.022	4.618
	AE	32	3.033	3.580
	LR	32	3.082	3.679
nRF52	AE	16	1.092	1.133
	LR	16	1.100	1.150
	AE	32	0.764	0.805
	LR	32	0.772	0.822

Table 2.4: Average time required for ticks adjustment with one update per hundred runs (/100 method)

which is divided by 100 for comparison with Table 2.2 and Table 2.3. It is clear that with this approach, SP cases for both AE and LR are 3-5x faster than the typical implementations. This improvement increases to more than 15x (nRF51) and 8x (nRF52) for DP ones. Furthermore, in this method and in both nRF5 SoCs, SP and DP times are a lot closer to each other. For simplicity, we refer to this technique as /100, and its combination with AE and LR is indicated by /100 at the end.

	Method	Timer Width	Time (μs)
			Single & Double Precision
nRF51	AE & LR	16	2.095
		32	2.078
nRF52	AE & LR	16	0.816
		32	0.490

Table 2.5: Average time required for ticks adjustment with pre-calculated values (4L method)

2.4.2 *Adjustments with Pre-Calculated Values*

Similar to the previous implementation, the timer is periodically restarted with an adjusted value of the task period. The decimal fraction of this adjusted period is then multiplied by 100, 1000, 5000, and 10000 to get integer values for ticks adjustments at these times or levels. Again, similar to the /100 method, final timer values at these four levels are modified by deducting the ticks differences already included in the previous updates. All calculations are done only once at the beginning of the synchronous task. The outcomes are an adjusted task period and four values to adjust the timer at four levels. After reaching any of these levels, the corresponding value is just loaded to the timer for one period to correct for the errors. Therefore, no extra calculation is required in the time interrupt handler, and we only need to keep track of the task executions in a count register. This register is reset after 10000 interrupts, and the adjustments restart from level one.

Table 2.5 shows the average time that is needed to run this code in the timer interrupt handler. Besides updating the counter at each level, the code includes a few compare and increment commands. Thus, the times are independent of the drift estimation method and floating-point precision. By employing this approach, ticks adjustments can be another 25-40% faster, compared to the /100 technique. For simplicity, we refer to this method as 4L.

2.5 Duty Cycling Clock Sources

Ref. [17] proposes an approach called Virtual High-Resolution Time (VHT) for low-power time synchronization, which is based on duty cycling the high frequency clock (*HfClk*). In this method, in addition to the *HfClk*, a low-frequency clock (*LfClk*) is considered for the system (typically 32 KHz). An always-on counter is clocked by the *LfClk*, whereas the *HfClk* is only used for timestamping. A ratio between the *HfClk and LfClk* is estimated to convert the time of any synchronization event to a high-resolution value. The duty cycle of the *HfClk* plays an important role in achieving low-power operation. With a resynchronization period of 10 s, the system reported in [17] can have power saving only when the duty cycle is below 10%.

While the goal in [17] is to reduce power consumption between resynchronization packets, the same concept can be applied to scheduling a synchronous task. However, to evaluate the effectiveness of this approach, a few points should be taken into account in estimating the duty cycle value. The *HfClk* is used by the processor, and as a result during task executions (e.g., sensor readings) and computing ticks adjustments (Table 2.3) it cannot be turned off. Similarly, the *HfClk* is needed for BLE operation, and must be active for any potential data exchange in the intervals. Considering that the intervals here are in the range of few milliseconds or even hundreds of microseconds, all of these times can increase the duty cycle. Table 2.3 is helpful for the times needed for ticks adjustments, but task executions and radio communications highly depend on the application. Another important factor in this method is the ratio of the power consumption of the *HfClk* and of the *LfClk*, which is dictated by the hardware platform.

Here, we briefly explain the implementation of *BlueSync* with duty cycling the *HfClk*. In nRF5 series, the *LfClk* is 32 KHz, and is only connected to the real-time counter (RTC). As we do not have any resynchronization, the *HfClk* is always active in the short synchronization timeslot.

This means that all the timestamping and drift estimations are only based on the *HfClk*. The clocks ratio estimation is also performed during the synchronization timeslot. The idea is that after the last synchronization packet, RTC starts and runs for most of the task period, and the timer with *HfClk* only turns on close to the end to provide high accuracy. This is achieved by dividing the adjusted ticks (based on *HfClk*) by the clocks ratio and using the integer part of the quotient as the RTC interrupt value. The RTC triggers the timer, whose interrupt value is the remainder of the adjusted ticks.

One problem with using the RTC in nRF5 series (and potentially other chips) is that its start/restart task has a delay between 15-45 µs. This uncertainty can cause large errors between the nodes. Therefore, synchronization is implemented without resetting the RTC, and the delay of starting it with the last packet should be compensated by one of the following techniques: a) If we start the RTC and timer together, by having the clocks ratio and the timer value after a few *LfClk* cycles, we can estimate the delay; b) Before the last packet, the RTC is started, and at R_0 ticks it triggers the timer. Then, the timer value T_0 is captured upon receiving the synchronization packet. Finally, T_0 is added to the adjusted ticks (similar to an offset in LR), and R_0 is added to the RTC interrupt value. Results show that the estimation in the first method is not as accurate as adding offsets to the calculations. Thus, the second approach is employed in *BlueSync*. It is important to note that the *Discrete Adjustments* method can help in decreasing the *HfClk* duty cycle by reducing the calculation times. However, as it requires to restart the RTC periodically, it would be subject to high errors. Clearly, if a hardware platform provides RTC operation with negligible delays, combining *Discrete Adjustments* and *HfClk* duty cycling can increase the likelihood of power saving in a larger number of applications. The combination of this method with AE and LR is indicated by 32k at the end.

2.6 Results

In all experiments, one DK serves as the master and two of the wireless nodes designed for nRF5 series are used as slaves. Again, we should note that master/slave terms should be viewed from a synchronization standpoint and do not mean that two devices need to be paired in BLE. Generally, the *BlueSync* protocol with AE is programmed on nRF51 nodes, whereas nRF52 nodes run the LR version. To investigate the effect of synchronization speed on average error, different configurations are tested for the synchronization timeslot. For simplicity we name these configurations as x(y), where x is either 8 or 16 representing the number of timestamps/packets, and y is the packet interval in seconds, in which a decimal point must be considered after any zero. As an example, 16(01) means 16 packets with 0.1 s interval. If not stated all error reports are for the 8(1) configuration. As mentioned before, after synchronization timeslot there is no further radio activity (for synchronization purposes) and the nodes start to adjust their interrupts for a 10-minute-long task with a period of 10 ms. To measure the error, the nodes toggle one of their output pins every 10 ms. The distance between transitions in two nodes is treated as the synchronization error. To collect more datasets, the nodes automatically restart after 10 minutes and the same experiment is repeated for at least one hundred times.

Results are presented in five parts. Part 1 shows synchronization error for both nRF51 and nRF52 nodes in two conditions: when only offset is removed between the nodes; and when software interrupts are used for timestamping (Figure 2.6). These results can be treated as starting points before we apply our proposed techniques and *BlueSync* to the system.

In Part 2, maximum synchronization error is plotted in Figure 2.7 for AE, LR, and *Discrete Adjustments* methods (for both SP and DP calculations) against 7 different packet configurations, which represent synchronization speed as explained above. Figure 2.7 shows the error for 91

possible settings in a heat map style and is a good reference for comparing different setups. Parts 3 and 4 provide a more detailed comparison for the two *Discrete Adjustments* methods (explained in Section 2.4) and duty cycling clock sources (covered in Section 2.5), respectively. It is worth mentioning that similar to previous works, average absolute error is used to compare different synchronization protocols, and due to the large number of different settings tested (>100), it is not practical to plot the histogram and cumulative distribution function (CDF) of the error for all of them. Therefore, these two plots are only depicted for the two proposed *Discrete Adjustments* methods. The main reason of choosing *Discrete Adjustments* is that by using them the error fluctuates, and the histogram and CDF are provided to give a better understanding of the error distribution.

Finally, instead of comparing synchronization methods just based on synchronization error, a figure of merit (FOM) is defined in Part 5 to add synchronization speed and calculation times (energy efficiency) to the equation. This enables us to have a better comparison of different methods. In Figure 2.12, drift estimation method is fixed and FOM is plotted to show the trade-off between synchronization speed and accuracy. In Figure 2.13, synchronization timeslot/speed is fixed and FOM is plotted to compare methods with different calculation approaches.

2.6.1 *Offset Removal and Software Interrupts*

Initially, for studying the impact of frequency-drift estimation and timestamping with PPI, two tests are conducted as follows. First, only the offset is removed by sending a single packet without any timestamp information. Nodes restart their timers upon reception of this packet with a PPI task (to reduce software delays) and generate interrupts without any adjustments. The result is then a representation of the frequency error between crystals employed in the nodes. In the second setup, AE and LR are used, but timestamping is performed by software interrupts. Figure

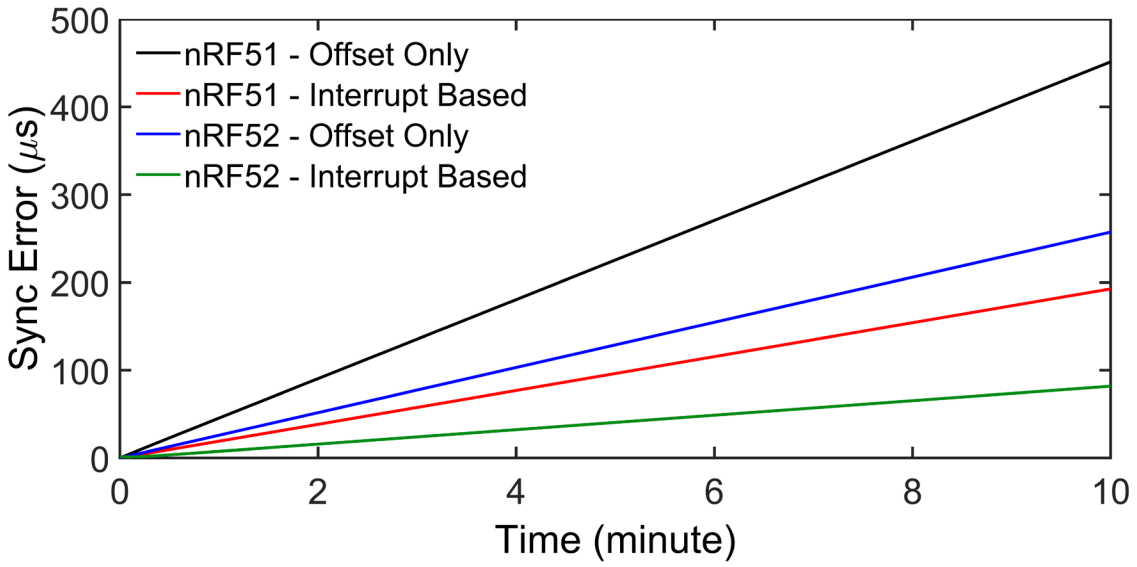
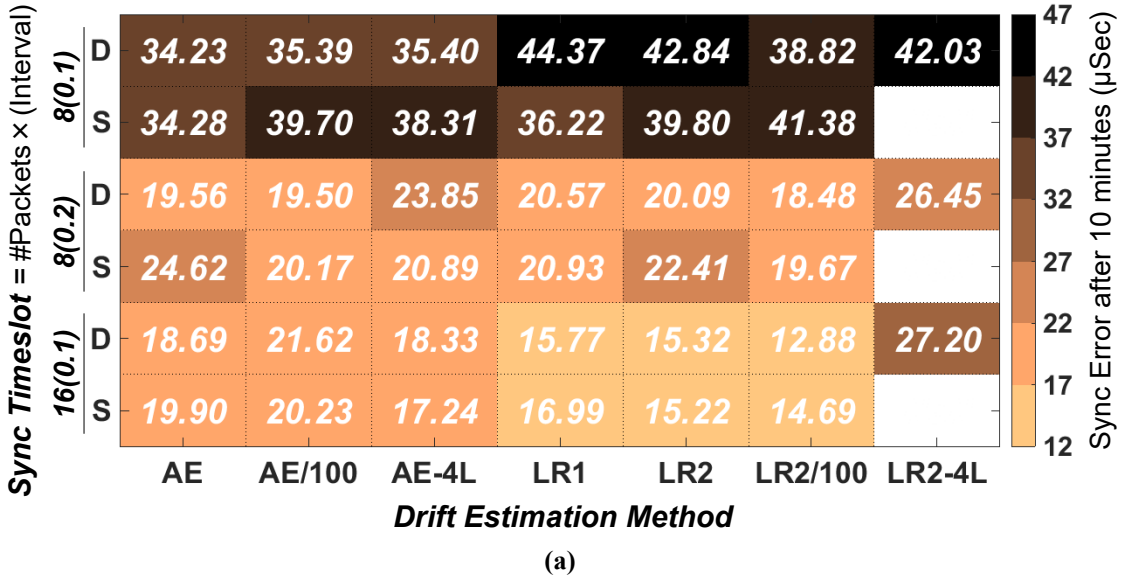


Figure 2.6: Average measured absolute synchronization error across 10-minute sessions when a) only offset is removed between the nodes; and b) software interrupts are used for timestamping.

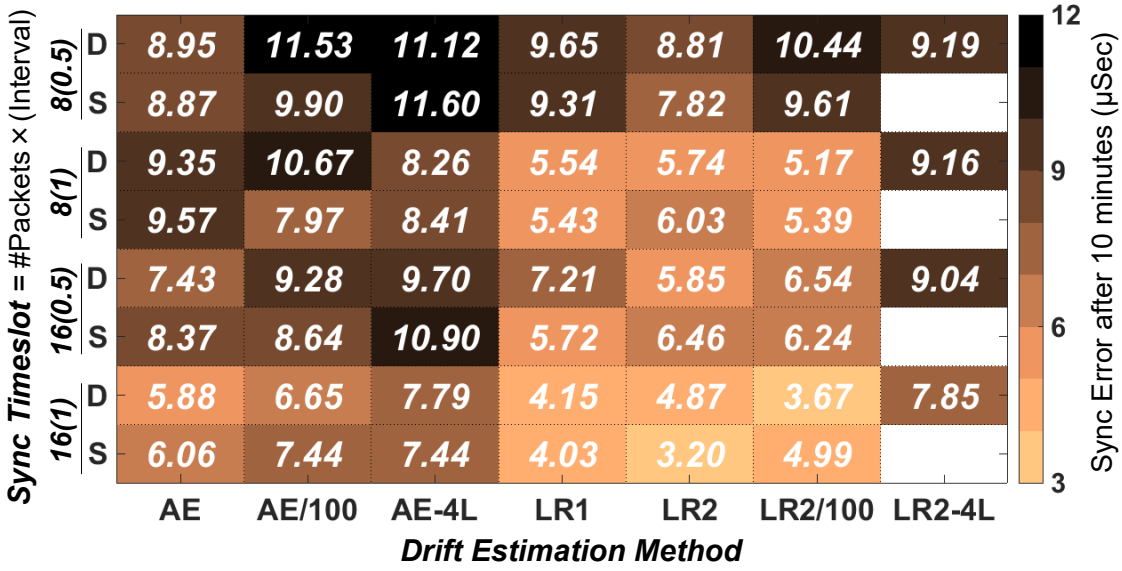
2.6 illustrates the absolute synchronization error measured every 10 ms and averaged over all recorded sessions. In the first test, the average maximum error is 451 µs and 257 µs for nRF51 and nRF52, respectively. The second test shows that even by using less accurate interrupt timestamping, these errors can be reduced by 2.3x (193 µs) in nRF51 and 3.1x (82 µs) in nRF52. Lower error reduction in nRF51 can be justified by its higher timestamping error with software interrupts (Figure 2.3).

2.6.2 *BlueSync*

For *BlueSync* experiments, at first we considered two slightly different versions for LR. In LR1, nodes' timers do not restart with the last packet, and the first interrupt is calculated by adding a pre-defined constant value to the last master timestamp. On the other hand, LR2 is implemented similar to AE and starts from zero at the beginning of the task (Figure 2.4). The idea behind LR1 is that the timestamps used in computing slope/offset were captured by a timer that was restarted with the first packet. As each reset may have a different delay, the concern was that applying the



(a)



(b)

Figure 2.7: Average measured maximum absolute synchronization error after 10 minutes without any resynchronization. Single and double precision floating point calculations are shown by S and D, respectively. **(a)** *Synchronization timeslot smaller than 4 seconds, and* **(b)** *synchronization timeslot between 4-16 seconds.*

same estimation results after the second restart might cause a small error that gets accumulated over time.

Average maximum synchronization errors (during 10-minute sessions) for *BlueSync* and *Discrete Adjustments* method are summarized in Figure 2.7. The results are divided into two

groups based on the synchronization speed. Fast synchronizations (synchronization timeslots smaller than 2 seconds) are tested with 8(01), 8(02), and 16(01) configurations. In the second group, four configurations are considered as 8(05), 8(1), 16(05) and 16(1) for synchronization timeslots between 4-16 seconds. Single and double precision calculations are identified as S and D, respectively. By analyzing the results, there is no significant difference between LR1 and LR2 methods. Therefore, LR2 is chosen to be combined with the *Discrete Adjustments* concept.

As discussed before, in LR both *slope* and *offset* need to be computed in DP, and we can only do ticks adjustments in SP. In the 4L method, there are only four calculations for ticks adjustments. Hence, we do all of them in DP, and in Figure 2.7 there is no data reported for LR2-4L in SP. Comparing the errors with PPI timestamping (Figure 2.7) and their interrupt-based counterparts (Figure 2.6) shows around 24x (nRF51) and 10x (nRF52) improvements for 8(1) configurations. Also, by using the PPI feature, errors are in the same range for both nRF5 SoCs.

2.6.3 *Discrete Adjustments*

To better evaluate the proposed *Discrete Adjustments* approach, the average absolute errors of AE-4L and LR2/100 are plotted against AE and LR in Figure 2.8. It is clear that in both AE-4L and LR2/100 results, synchronization error fluctuates more. As it can be seen in the zoomed insets, these fluctuations can cause some spikes in the error within the first minute. However, when the average error increases above one minute, the impact of fluctuations would become less significant, and as shown in Figure 2.7, in 10-minute sessions average maximum errors are still comparable with AE and LR2. Similar trend is present in AE/100 and LR2-4L results. Furthermore, histogram and cumulative distribution function (CDF) are plotted in Figure 2.9 for AE-4L-16(1) and LR2/100-16(1) methods with DP operations.

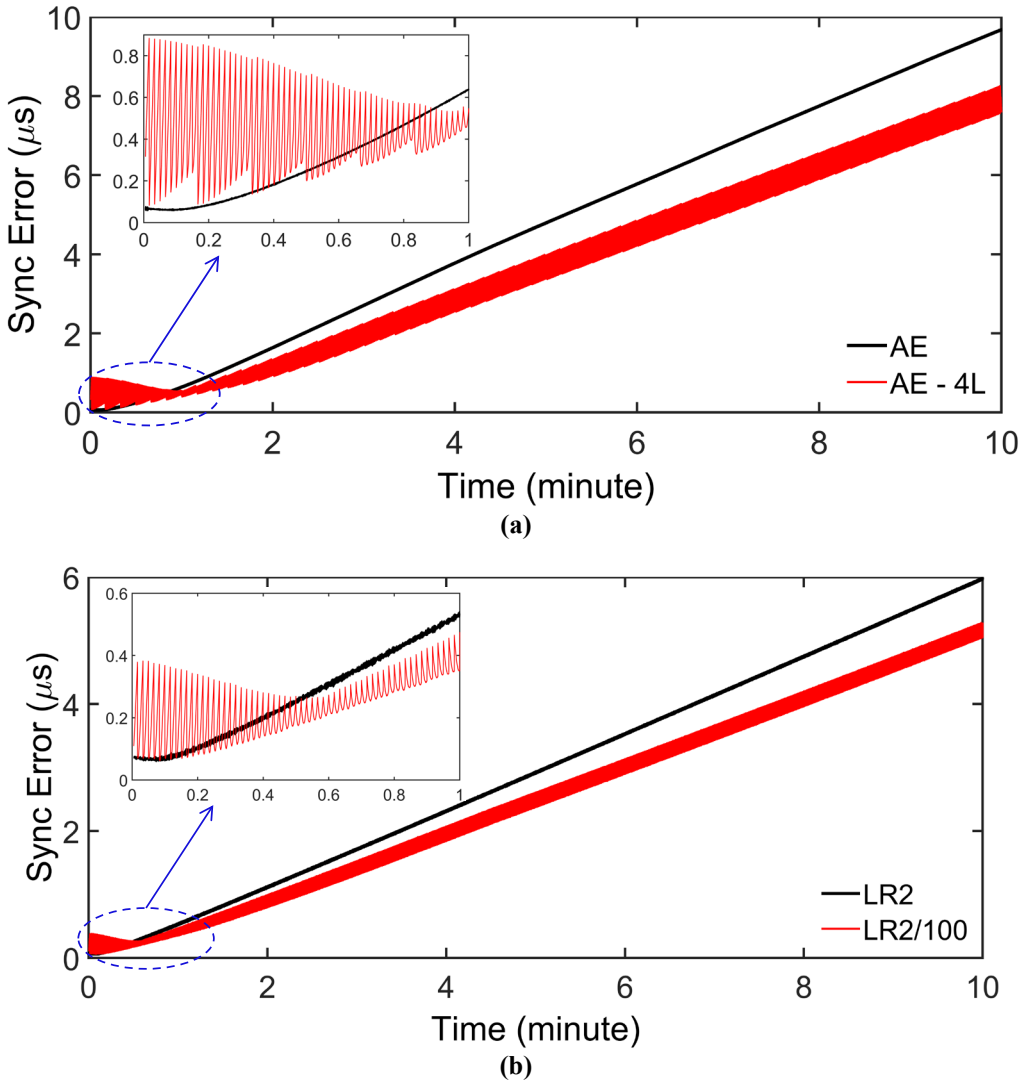


Figure 2.8: Average measured absolute synchronization error across 10-minute sessions with 8(1) configuration with SP operations for **(a)** *AE and AE-4L, and* **(b)** *LR2 and LR2/100.*

The spikes in error for the 4L method (Figure 2.8(a)) are around 2x larger than the spikes for the /100 method (Figure 2.8(b)). This is more clear in the zoomed insets, and immediately after starting the task in the 0~0.2 minute time window. If we use one update per *two*-hundred runs compared to the /100 technique, the expectation is that the spikes amplitude would double and would be around the same value of the 4L method. However, it should be noted that the time needed to run the 4L method is the minimum time possible, as it does not include any calculations and is composed of updating the time, and a few compare and increment commands (Section

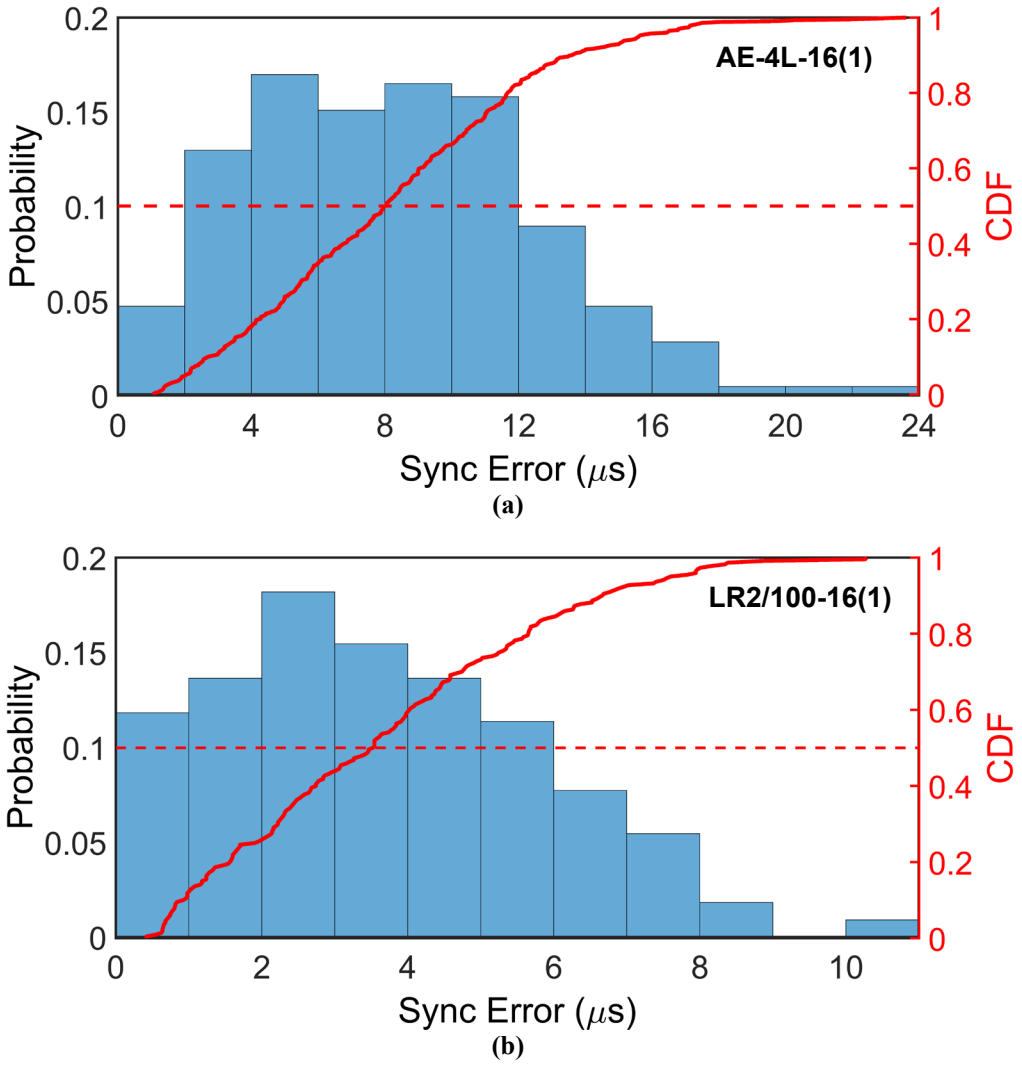


Figure 2.9: Histogram and CDF of maximum absolute measured synchronization error after 10 minutes with DP operations for **(a)** *AE-4L-16(1), Average=7.79 μs, σ=4.23 μs, and CDF(90%)=13.64 μs;* **(b)** *LR2/100-16(1), Average=3.67 μs, σ=2.18 μs, and CDF(90%)=6.66 μs.*

2.4.2). In other words, we can use the /100 method if we are interested in lower spikes in the error. But, if having ~2x higher spikes is acceptable, then the 4L method provides a better tradeoff between the error and ticks adjustment time.

2.6.4 *Duty Cycling Clock Sources*

Average maximum synchronization errors of *BlueSync* (after 10 minutes) with and without using two clock sources are reported in Figure 2.10. The results validate that without degrading

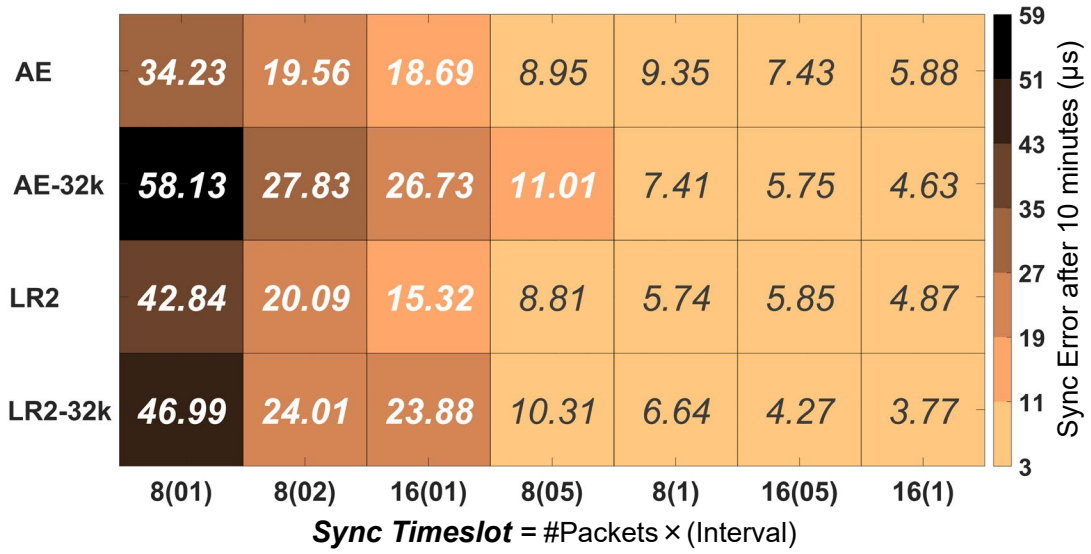


Figure 2.10: Average measured maximum absolute synchronization error after 10 minutes with and without duty cycling the HfClk. All results are for DP calculations.

performance, we can utilize the idea of duty cycling the *HfClk* in scheduling synchronous tasks. One important point in this case is that besides drift compensation, estimation of the ratio of the clocks affects the timing accuracy. This ratio is calculated by running both a timer (clocked by the *HfClk*) and a RTC (clocked by the *LfClk*) for a fixed number of RTC ticks (R_f), and capturing the number of timer ticks at R_f. Depending on the time considered for the estimation, this test can be repeated a few times to calculate an average ratio. To show its effect on the synchronization error, we tested LR2-32k-8(1) with three different estimation lengths of 1, 2, and 8 seconds. As shown in Figure 2.11, 1- and 2- second durations have, respectively, 6x and 2x larger errors, compared to the estimation length of 8 s. In all experiments (Figure 2.10) duration of clock ratio estimation is equal to the synchronization timeslot. Consequently, this estimation affects the accuracy of fast synchronizations more.

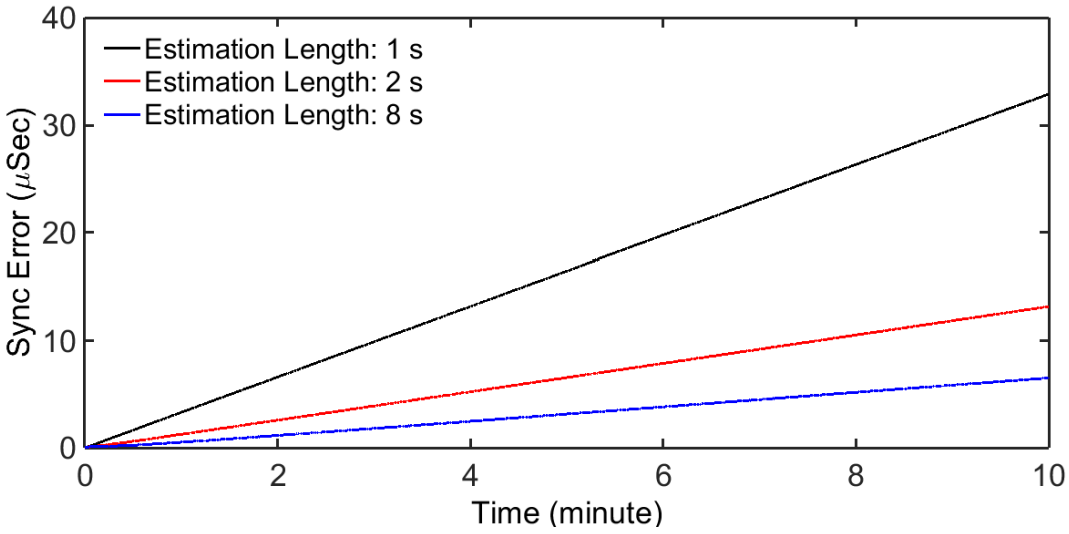


Figure 2.11: Measured synchronization error when three different lengths are considered for estimating the ratio between the HfClk and LfClk in LR2-32k-16(1).

2.6.5 *FOM*

Comparing methods based on the average error is only useful when the goal is achieving the best possible accuracy. In this comparison factors like speed and synchronization overhead (or energy efficiency) are left out. Therefore, we define a FOM as:

$$FOM = \frac{t}{error} \times \frac{1}{nT_P} \times \frac{1}{T_1 + N \times T_2}, N = \frac{t}{T_{Task}}, \qquad \textit{Equation 3}$$

where *error* is average maximum synchronization error after t seconds, n represents number of synchronization packets with interval T_P, T_{Task} is the synchronous-task period, N is number of interrupts during t, and T_1/T_2 are times required for error estimation / ticks adjustments. In this FOM, we are basically removing the effect of resynchronization on error, and adding synchronization speed and calculations overhead to the cost function. Consequently, the FOM has the following three main parts:

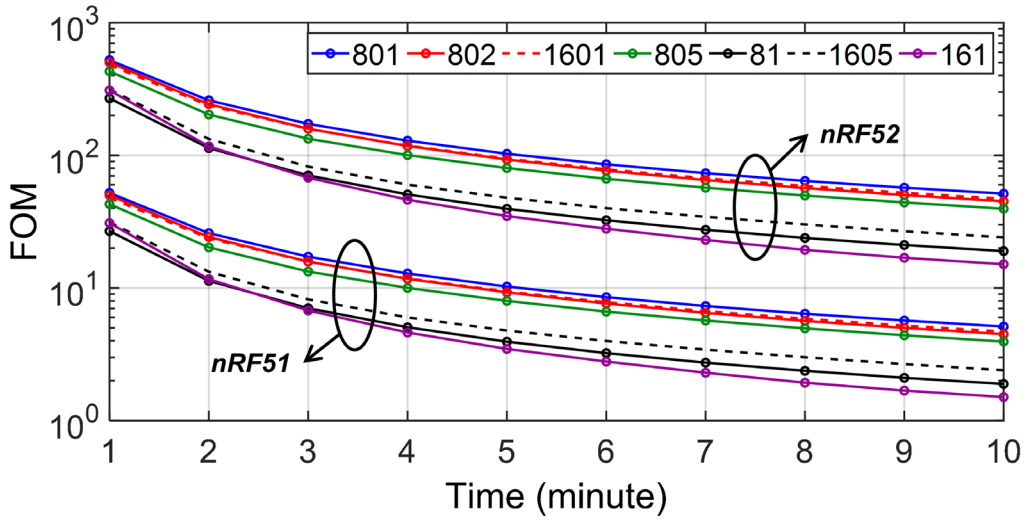


Figure 2.12: FOM for AE with DP calculation for 7 different synchronization packet configurations, plotted for both nRF51 and nRF52.

1. Synchronization Error: error is normalized to the duration of the test, in which no resynchronization is performed (*error*/*t*).

2. Synchronization Speed: shows how much time is required before the nodes can start the task. We use n packets with T_P intervals for synchronization. So, the synchronization timeslot is $n \times T_P$.

3. Calculation Overhead: we have calculations for drift estimation at the beginning, which is T_1 (Table 2.2). Then, after each interrupt, we need to adjust the timer ticks, which the required time is T_2 (Table 2.3-Table 2.5). So, for N interrupts during the whole test session the overhead is $T_1 + N \times T_2$.

Obviously, we are interested in a larger FOM, which means lower error during a longer period with faster synchronization speed and lower calculation times. First, we study the impact of synchronization speed on synchronization accuracy. For this purpose, we can plot the FOM of each method for the seven different packet configurations. Although Figure 2.12 is for AE with

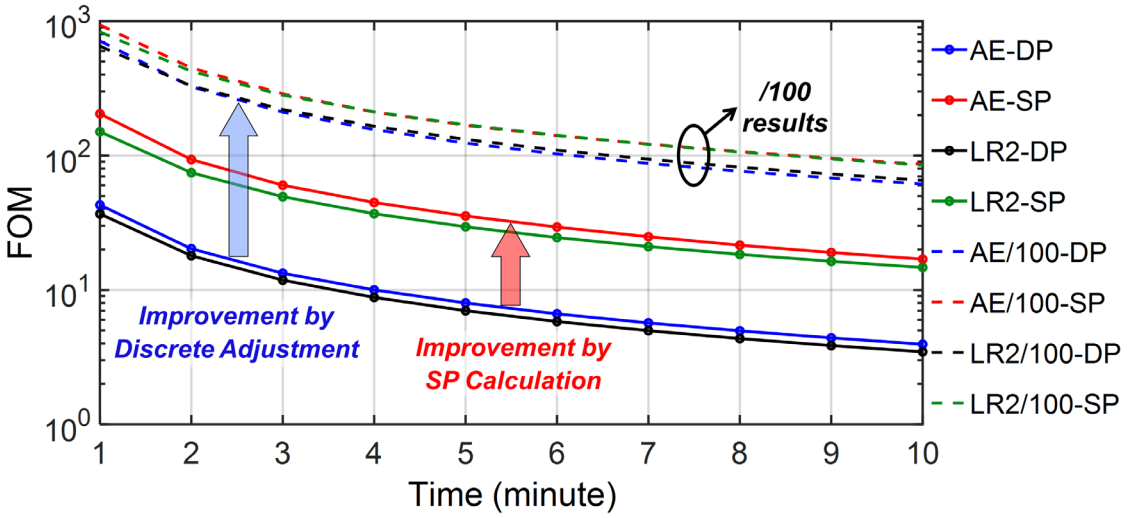


Figure 2.13: Comparison of FOM for AE, AE/100, LR2, and LR2/100 with 8(05) configuration, and both SP and DP calculations on nRF51.

DP operations, we observed a similar trend in all other methods. The results indicate that if in a system speed and accuracy are equally important, fast synchronizations like 8(01) and 16(01) can provide a better performance. Moreover, 8(05) is somewhere in between in terms of both speed and accuracy. The FOM values of nRF52 are higher than nRF51 because of its smaller T_1 and T_2 times (Table 2.2 and Table 2.3), which are due to a faster processor and more advanced architecture.

Second, we compare the FOM of different methods for one specific packet configuration. Again, Figure 2.13 is plotted for 8(05), but other configurations follow the same trend. In both AE and LR2, SP computations have around 2.5x larger FOM. With the FOM scale in Figure 2.13, traces for the /100 and 4L implementations would be close to each other. Therefore, only the points of the /100 approach are illustrated to make the plot readable. The *Discrete Adjustments* method increases the FOM by 3.3x and 6.9x for SP and DP operations, respectively. Its results are also less sensitive to the floating type used. Additionally, it is clear from Figure 2.12 and Figure 2.13 that the FOM values of AE/100 on nRF51 are even slightly higher, compared to AE

implementation on nRF52. Considering that nRF51 has a slower processor, this comparison is another way to demonstrate that the *Discrete Adjustments* approach can improve the performance of synchronization algorithms.

2.7 Comparison with Previous Works

In this section, first, we compare *BlueSync* with other BLE implementations, and then with some prominent synchronization methods reported for other wireless protocols. In [22], due to low accuracy of timestamping in application layer, the synchronization error is 37.78 ms. In *BlueSync*, even by using software interrupts for timestamping (Figure 2.6), the error is three orders of magnitude lower than [22]. In [23] and [24], three techniques are proposed in total to improve the precision of timestamping in BLE. Among them, [24] has the lowest standard deviation of 900 ns, which is based on the implementation of their approach on nRF51. As shown in Figure 2.3, by utilizing the PPI feature in both nRF51 and nRF52, we can get 18x lower standard deviation, i.e., under 50 ns. BLE beacons were also used in *Cheepsync* [20]-[21], where they achieved a synchronization error of 10 μs with 100 ms resynchronization period. In *BlueSync* even with 8(05), the error after 100 ms is around 75 ns (100x smaller) and after 10 minutes is still under 10 μs. Larger error in *Cheepsync* can be justified by the fact that is not designed for traditional WSN applications, and cell phones are part of their system as receivers.

Regarding other wireless protocols, the way synchronization errors are measured and reported in previous studies makes it hard to have a fair comparison. Typically, errors are reported for a specific and small period, which varies between papers. Data are rarely provided for long sessions (i.e., few minutes) without any resynchronization. Therefore, we try to measure *BlueSync* errors in conditions as close as possible to each of the following works. Furthermore, to compensate for different clock frequencies in each study, we divide (normalize) all errors by the

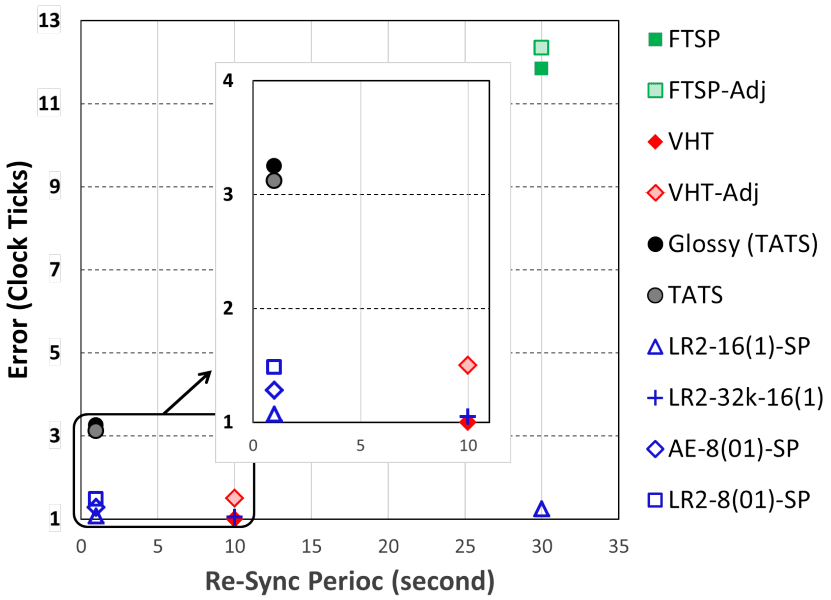


Figure 2.14: BlueSync comparison with other wireless protocols.

clock period. Also, some of the following methods use timestamping to calculate the synchronization errors. Actual error when using timestamps can be higher by up to one clock period, compared to toggling a pin in each node and measuring the distance between the edges (used in *BlueSync*). Hence, we adjust for this additional error by considering an extra 0.5 clock period (on average) for all reported values.

Figure 2.14 summarizes the comparison with other wireless protocols explained below. It is worth mentioning that in all the data reported here and in other previous works, no temperature variation is considered between the nodes, and different synchronization protocols are compared without implementing any compensation for the temperature. In Chapter 6, we will discuss potential solutions for applications where large temperature difference across nodes is expected.

In FTSP [12], an average error of 1.48 µs was measured for 30-second resynchronization period and data polling every 18 seconds. With this polling period, after five measurements the polling time will be the same as a resynchronization point (which can be treated as time zero). As

there is no resynchronization in *BlueSync*, we use the value of these five points relative to 30 seconds and average our results on these times (i.e., 18, 6, 24, 12, and 30 seconds). In FTSP an 8 MHz clock was used. As a result, FTSP error would be 11.84+0.5=12.34 clock periods, whereas LR2-16(1) (with SP operations) in *BlueSync* can achieve 1.24 clock periods. Moreover, even if we consider LR2/100-16(1), this error will be 2.72 (SP) and 1.34 (DP), which are still 4x and 8x smaller than FTPS, respectively.

In VHT [17], nodes are resynchronized every 10 s, and a beacon is sent every 2 s, which the nodes timestamp. These timestamps are collected through a wire to calculate the synchronization error. As also pointed out in [19], VHT accuracies are reported as mean signed deviation, which typically results in lower values, compared to mean absolute error. With these conditions, VHT achieves an accuracy of one clock period, which is adjusted to 1.5 clock period for comparison. By averaging the results of LR2-32k-16(1) method at 2, 4, 6, and 8 seconds, we get an error of 65.37 ms or 1.05 clock period. This value is 30% less than the adjusted error of VHT, and basically shows that duty cycling the *HfClk* can be used with high precision in both *BlueSync* and scheduling a synchronous task.

Another well-known synchronization protocol is *Glossy* [18]. *Glossy* takes advantage of concurrent transmission to implement fast network flooding and time synchronization with a re-synchronization interval of 1 s. In [18], the radio clock frequency is 8 MHz, and error is measured as the time difference between activation of output pins. From 3-8 hops, error is almost constant around 400 ns (3.2 clock period) based on mean signed deviation and without drift compensation. In another work [19] from the same group, *Glossy* is tested again with six nodes that can be up to six hops away from each other in the network. The average absolute error by timestamping the signal from a GPS receiver attached to the nodes is 720 ns with a 13 MHz clock, i.e., 9.35 clock

period. Considering that there is an error of 20 ns between the GPS receivers, we do not add the 0.5 clock period (i.e., 38.5 ns) to the reported values in [19]. From a graph for *Glossy* results [19], the node in the first hop has an accuracy of around 250 ns that translates to 3.25 clock periods. We use this value for comparison, as its measurement conditions are closer to *BlueSync*. Ref. [19] also presents another method called Time-of-flight Aware Time Synchronization (TATS), which uses both fast flooding and propagation delay compensation. With 13 MHz clock frequency, re-synchronization period of 1 s, drift compensation, and using the same six nodes in a six-hop network, TATS achieves an error of 240 ns or 3.12 clock periods. Note that this error is for a multi-hop setup, and no data was provided for single hop performance. In *BlueSync*, after 1 s even with 8(01) configuration (fast synchronization) and SP operations, the errors of AE and LR are 1.28 and 1.48 clock periods, respectively. Although these results show that *Glossy* and TATS have lower accuracies, it should be noted that the main idea behind these two methods is to reduce the error in large hops, and they do an excellent job to keep the error low in these networks. This can also justify the fast resynchronization periods of 1 s. Therefore, even with the one-hop error, we still need to consider the applications that *Glossy* and TATS are designed for. Improving error in multi-hop networks is not the topic of this study. The reason that we included this comparison is that these designs are two of the notable works in the field that have sub-microsecond accuracies.

2.8 Summary

The presented *BlueSync* is a time synchronization protocol for BLE sensor networks and is compatible with commercial SoCs. By improving timestamping in BLE and using frequency-drift compensation, *BlueSync* achieves average timing accuracies of <1µs per 60 s. To the best of our knowledge, this is the lowest synchronization error reported for BLE. The challenges for synchronization over BLE, along with features in BLE chips and the new BLE 5 standards that

help address them, are explained in detail. The discussion can be used as a guideline for implementation in other BLE hardware platforms.

Besides BLE specific techniques, with the goal of reducing the synchronization overhead, we investigated two methods for drift estimation and ticks adjustments (AE and LR) for scheduling synchronous tasks. Results show that just by choosing the appropriate technique based on the processor specifications (including clock frequency, timer width, and processor architecture), up to 5x improvement can be achieved in time/energy spent on calculations (independent of the wireless protocol).

In contrast with many of the previous studies that the re-synchronization interval is only a few seconds, the goal here is not to perform any re-synchronization for many minutes (after a short synchronization timeslot at the beginning). The *Discrete Adjustment*s approach with two implementations is proposed to further reduce the computation overhead in these lengthy synchronous tasks. It may be used with both AE and LR, and can reduce the calculation time for ticks adjustments up to 15x. This method is especially effective in processors with lower clock frequencies. We also demonstrated that the concept of duty cycling the *HfClk* can be applied to scheduling synchronous tasks. Practical implementation in *BlueSync* and discussion on how potential power savings are dependent to the target application were provided.

All of the presented methods are experimentally tested in 10-minute long sessions with error measurements every 10 ms, and by considering 7 different configurations for synchronization packets. The synchronization timeslot affects frequency estimation, and consequently the synchronization error. However, the tradeoff between these two has not been well studied in previous works. Results show that when speed and accuracy are both equally important, fast synchronizations (<2 s) perform better. The dataset reported here can give insight to researches in

selecting the appropriate synchronization speed and method, based on their own specific application and constraints on error.

Chapter 3 Low-Power Techniques for Synchronization Protocols

Two methods for reducing the power consumption of running a synchronous task are discussed in this chapter. First, frequency scaling is used to reduce the power consumption during scheduling a synchronous task, without significant increase in the synchronization error [37]. Second, a protocol called *AdaptSync* [38] is proposed to replace frequent resynchronization packets, which are sent to reset the error in the nodes. Both methods are especially effective in reducing power consumption in long synchronous tasks (>1 min). While the implementation here is based on *BlueSync* as the main synchronization protocol, these concepts can also be applied to any other synchronization protocol.

3.1 Frequency Scaling in Time Synchronization

The idea here is to apply the concept of frequency scaling (FS) to the two different stages of operation shown in Figure 2.1: synchronization timeslot and synchronous task. In FS the goal is to dynamically change the clock frequency to lower the power consumption of the processor and its peripherals. Basically, as the synchronization timeslot is short and includes timestamping for drift estimation, it should run on a high frequency clock, whereas the synchronous task may be scheduled with lower frequency clocks.

3.1.1 *Implementation*

The nRF51 SoC used in the designed sensor nodes requires a ±20 ppm 16 MHz crystal for BLE operation. Two 16-bit timers run on clock signals generated from this crystal. The system

supplies a 1 MHz and a 16 MHz clock source for peripherals. The timer frequency is set by a 4-bit prescaler as 16 MHz/$2^{PRESCALER}$, and whenever it is equal to or smaller than 1 MHz, the clock controller selects the 1 MHz peripheral clock for reduced power consumption. Using the 1 MHz clock almost halves the run current of the crystal oscillator (from 470 μA to 250 μA) and lowers timer current by 7.5x (from 30 μA to 4 μA). Therefore, in our sensor nodes we need to reduce the timer clock frequency to 1 MHz or lower to benefit from any power savings.

In synchronization timeslot, the radio needs the 16 MHz clock, and the timer should also run on this frequency to timestamp the synchronization packets with high precision. Then, both AE and LR methods (Section 2.3.2) can be used for frequency-drift estimation. Before the last synchronization packet from the master that triggers the start of the desired task, the nodes stop their timers and change their clock frequency to 1 MHz (or lower). Finally, each node employs its own estimated drift values to schedule a task with 10 ms period for ten minutes. To measure the synchronization error, after each interrupt the nodes toggle one of their output pins, which are recorded by a logic analyzer.

It should be noted that in commercial processors, the conditions to achieve reduced power consumption, and the value of this power saving depend on the features provided to the user. As an example, in our case with nRF51, any clock frequency above 1 MHz does not save power, and also with all clock signals below 1 MHz the reduction in power consumption is the same as the 1 MHz clock. On the other hand, custom designed processors are more flexible in this case, and potentially future ultra-low-power SoCs would include more modes of operation based on clock frequencies.

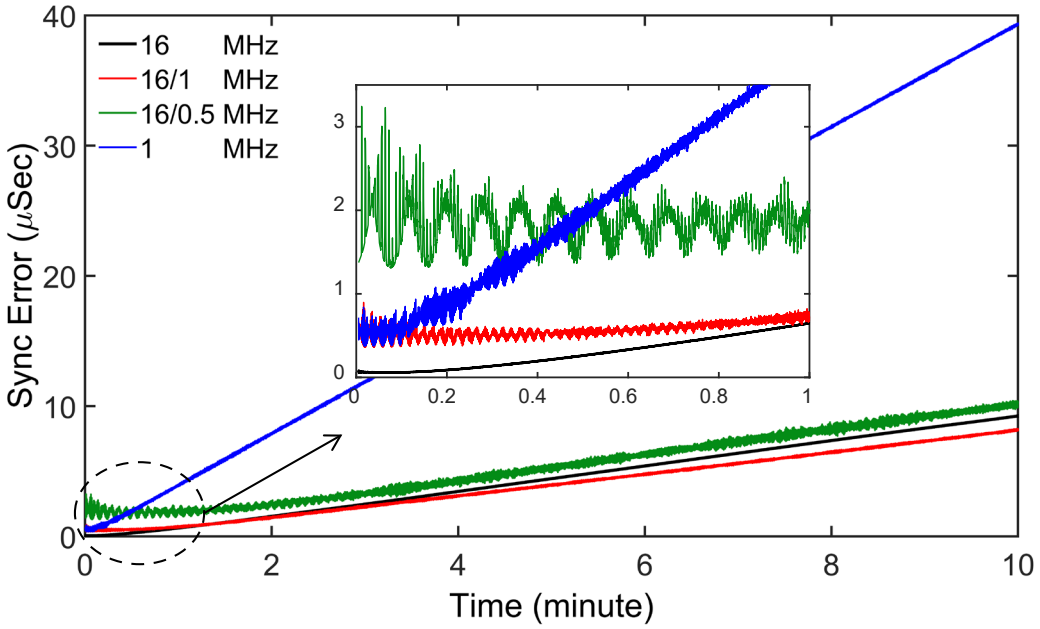


Figure 3.1: Comparing absolute measured synchronization error during ten minutes with and without scaling down the timer clock frequency during the task ("16/x MHz" means using the 16 MHz clock for the synchronization timeslot and using the x MHz clock during the synchronous task).

3.1.2 *Results and Discussion*

To demonstrate the impact of FS on synchronization error, first we consider the case where the 16 MHz clock is used for both synchronization timeslot and scheduling the task. We repeat the same test with the 1 MHz clock. Then we compare the results with two implementations of FS approach, in which the nodes switch the clock frequency to 1 MHz, and 0.5 MHz after running from the 16 MHz clock in the synchronization timeslot. All tests are for 8(1) packet configuration with AE (Section 2.6), and are repeated at least one hundred times for 10-minute long sessions. As shown in Figure 3.1 after ten minutes the 1 MHz FS technique has more than 5x lower synchronization error compared to the 1 MHz results, and also above one minute its error is comparable with the 16 MHz method. This implies that using a high frequency clock in the short synchronization timeslot has a much higher impact on synchronization error, compared to using it

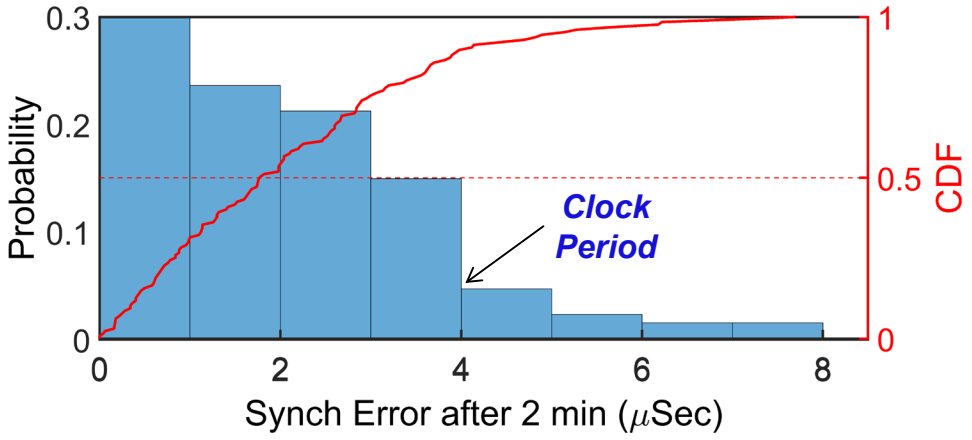


Figure 3.2: Histogram and CDF of absolute measured synchronization error after two minutes with 0.25 MHz timer frequency during the task. Average error is still smaller than clock period.

during the synchronous task. Moreover, even the 0.5 MHz FS results are more than 4x better than 1 MHz ones and are still within the same range of 16 MHz errors after ten minutes.

When scheduling the task with 0.5 MHz and 1 MHz clocks, it is expected to have higher errors at the beginning due to lower adjustment resolution. However, the important point about FS approach is that after starting with a higher error, its error oscillates around the same value for tens of seconds and does not increase immediately in contrast to the 1 MHz results. This point is clear in the zoomed part of Figure 3.1, and is also verified by plotting the histogram and cumulative distribution function (CDF) of the absolute synchronization error after two minutes for 0.25 MHz FS technique (Figure 3.2). These plots show that even after two minutes, the average synchronization error (CDF of 50%) is below one clock tick. Figure 3.3 summarizes the results of the 1 MHz FS approach with different packet configurations for both AE and LR methods, and compares them with the normal *BlueSync* error values.

It is worth mentioning that the presented FS technique has the following differences with the VHT service [17]. First, in VHT the system needs to switch between two different crystals: one low frequency (around 32 KHz) and one high frequency (few MHz). Moreover, each node

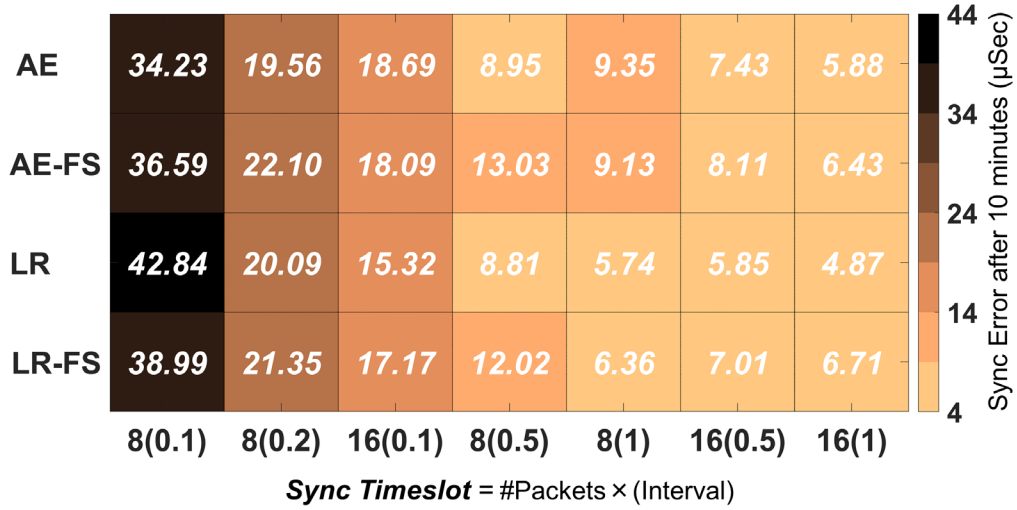


Figure 3.3: Average measured maximum absolute synchronization error after 10 minutes without any resynchronization. FS results are reported for 1 MHz timer frequency during the task.

must estimate a ratio of its two different crystals. However, in the discussed FS method, all clocks are generated from a single crystal, and only the frequency is changed between different stages of operation. Second, in VHT the assumption is that the processor is idle most of the time and can go to sleep mode with its high frequency clock turned off. To achieve power savings, the duty cycle of this clock must be smaller than a limit, which depends on the processor. In contrast, the idea behind FS is that depending on the workload, the processor itself can also run on a much lower clock frequency. Results here show that if the hardware platform provides such a feature, FS does not cause significant decrease in synchronization performance when frequency drift is estimated with high frequency clocks. Clearly, best practice in each network depends on the application, target hardware, and task requirements.

3.2 *AdaptSync*

As discussed in Section 2.1, synchronization protocols are mostly based on the flooding technique [12]. In this method, the master node sends a number of packets containing its timing information at a fixed rate. Other nodes timestamp these packets and estimate the frequency drift

of their clock during the synchronization timeslot. Mostly, linear regression is used for estimation, which results in a *slope* and *offset*. The nodes then use these values to schedule a synchronous task, e.g., measuring acceleration every 10 ms. If the nodes reset their timers with the first synchronization packet, *offset* would be very small. *Slope* values are dependent on crystal oscillators used in nodes as clock sources. Due to this hardware difference even with frequency-drift estimation, the timing error between the nodes starts increasing after a few seconds. In previous works generally the master sends resynchronization packets periodically (e.g., every 30 s) to reset the error. This method requires frequent use of the radio, which typically consumers the most power in a wireless sensor node. Therefore, reducing the error with minimum radio usage is desirable, especially when the duration of the synchronous task is more than few minutes. In *AdaptSync* we show that by using the estimated *slope* values in nodes and relative timing error after a short period of time, we can reduce the error during the rest of the task with minimum resynchronization packets.

The basic idea is that if after drift management, the synchronization error between two nodes increases in one direction (either positive or negative slope), by reversing that direction, the error can be reduced. To implement this idea, we need to find: 1) the sign and value of the error, and 2) a way to temporarily force the error to the opposite direction untill the accumulated error is lowered.

3.2.1 *Error Sign and Value*

First, we investigate the effect of synchronization speed on the error direction. Two nodes are synchronized using *BlueSync* with three different synchronization timeslots: 1.6 s (16 packets, 0.1 s interval), 8 s (8 packets, 1 s interval), and 16 s (16 packets, 1 s interval). After synchronization, nodes toggle one of their I/O pins every 10 ms for 5 minutes based on their

adjusted clock. This test is repeated more than one hundred times for each timeslot value. Error is calculated as the time difference between edges of the two I/O signals. Then, at each point of time we count the number of test sessions with positive error. For 16 s timeslot after 30 s, 100% of sessions have positive error, whereas in the 8 s and 1.6 s timeslots this value is around 85% and 50% during the whole 5 minutes, respectively. This indicates that with lower synchronization speeds, the probability of increasing the error in one direction is higher. Lower synchronization speed also results in lower synchronization error as shown in Figure 2.7. The problem is that different pairs of nodes do not necessarily have a 0% and 100% distribution for the error sign. In a real application, we want to have low error in every run of the system. So, even if 80% of sessions show positive error for two nodes, we cannot risk changing the error direction to negative values, as it will generate large negative errors if the other 20% happens. Therefore, we propose to use a short time at the beginning of the synchronous task to find the error between the nodes wirelessly.

In this setup, nodes are synchronized normally during the synchronization timeslot. After 60 s, the master sends a beacon, and all nodes timestamp this packet. The 60 s time can be changed based on the hardware and maximum acceptable error. Each node then uses its *slope* and *offset* values from linear regression to estimate the time at master:

$$t_{Master} = \frac{t_{Node} - offset}{slope}. \qquad \textit{Equation 4}$$

All nodes then transmit their *slope* and the estimated t_{Master} back to the master. Synchronization error between any two nodes is equal to the difference of their estimations of the master time. This approach was tested with three pairs of boards. To verify its accuracy, nodes were also toggling a pin every 10 ms based on their adjusted ticks. Comparing the time difference between pin toggles and the synchronization error calculated in the master shows up to 10% error, which is acceptable for our purpose.

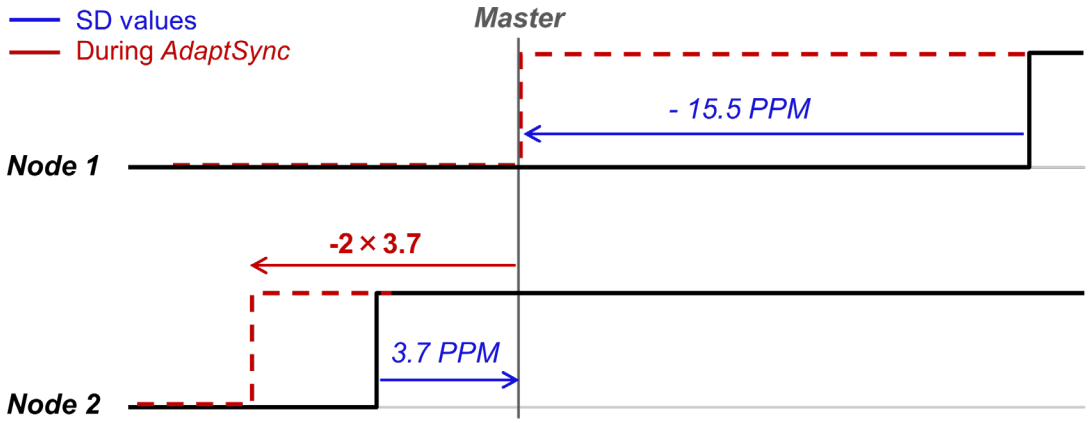


Figure 3.4: Relation between clocks of the two nodes used in tests.

3.2.2 *Reversing the Error*

Now that the master has the sign/value of the error between each pair of nodes, the final step is implementing a mechanism to reverse the error direction and send the parameters values to the nodes. To explain this method, we define Slope Difference (SD) which is equal to (*slope* -1) and shows how much faster (SD>0) or slower (SD<0) a node is (compared to the master). For simplicity, we multiply SD by 10^6 and express it in PPM (should not be mistaken with crystal tolerance). The amount of error variation between any two nodes can be computed as

$$Error\ Variation\ (\mu s) = (\beta_1 SD_1 - \beta_2 SD_2)_{PPM} \times {}^{n}/_{f_{Adj}}, \qquad \textit{Equation 5}$$

where f_{Adj} is the frequency of adjusting the nodes clock, n is number of adjustments, and β_i is a coefficient for node i that besides contributing to the error value allows moving the error to any point of time during adjustments.

To clarify this approach, consider the two nodes in Figure 3.4 where SD_1 = -15.5 PPM and SD_2 = +3.7 PPM. Having SDs with opposite signs is one of the situations that resulted in 100% error increase in one direction in our tests. Using normal synchronization (Figure 3.5), the average

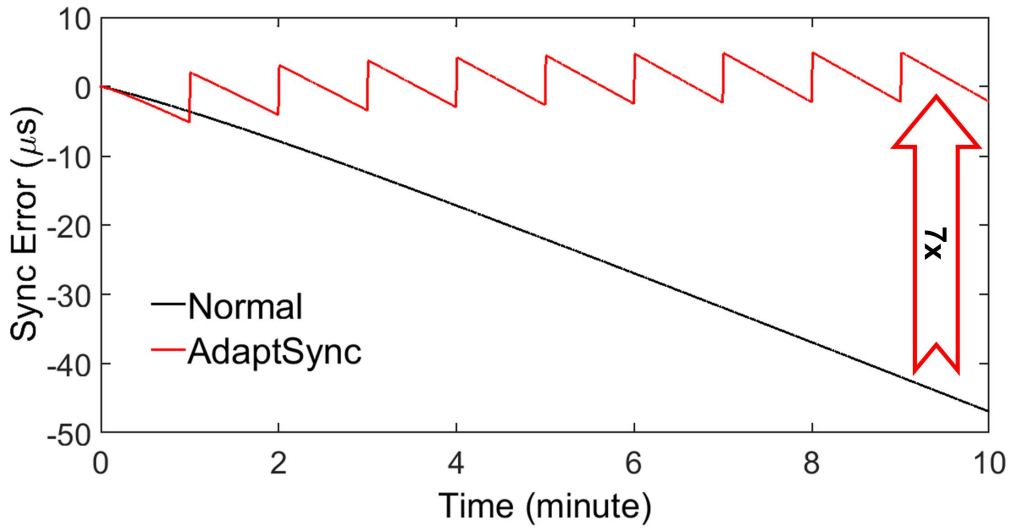


Figure 3.5: Measured synchronization error between two nodes by using normal synchronization and AdaptSync.

maximum error is -5.15 µs after 60 s. As we adjust the clock every 10 ms, f_{Adj} is 100 Hz. To reverse the error +5.15 µs, we assume $\beta_1 = 0$ and $\beta_2 = -2$ (Figure 3.4), which gives $n \approx 69$ based on Equation 5. Zero value of β_1 means that node 1 continues doing normal synchronization, whereas node 2 injects 2×3.7 PPM error for (69×10 ms) every 60 s. By injecting error we mean that it is moving the clock away from node 1, and this is because the nodes were over-corrected with the drift management in normal synchronization. After the 69 adjustments are finished, node 2 goes back to normal synchronization for the remainder of the 60 s. Considering the absolute values of errors for *AdaptSync* in Figure 3.5, there is 7x improvement in average error for a 10-minute recording session. The improvement can be more for longer sessions, as in normal synchronization, error is continuously increasing, while in *AdaptSync* it is oscillating and increasing relatively slowly.

Chapter 4 **Wake-up Radio**

In Chapter 2 and Chapter 3, both low-error and low-power synchronization protocols that are related to the *active* state of the nodes, i.e., when they are ON and are recording data from the sensors, were presented. Consequently, the power savings only apply to the *active* state. As discussed in Section 1.3, in many HUMS applications, the active time of the nodes is very small compared to the time they are idle. Clearly, to save power between measurement sessions, nodes must be put in a low-power mode, which is typically known as the *sleep* state. When a node is put in the *sleep* state, a wake-up mechanism should be utilized to bring back the node to the *active* state. The power consumption of this wake-up mechanism then dominates the power consumption of the node during the *sleep* state, and it should be orders of magnitude smaller than the active power consumption. In most cases, it is not possible to plan for the wake-up in advance, i.e., time of the wake-up cannot be programmed, and wake-up is triggered by an external event or a wake-up command.

In this chapter, first, we discuss two methods called wake on radio (WOR) and wake-up radio (WUR), that can be used to externally wake-up the nodes, i.e., by sending a command from another node or a base station. In Section 4.1 the focus is on power requirements of these methods, and how they affect the overall lifetime of the nodes. Then, as the sensitivity of a WUR sets its maximum range, in Section 4.2 the effect of using different frequency bands on the sensitivity and the size of the nodes is investigated. Section 4.3 reviews previous works and explains the architecture of the proposed WUR. Measurement results are provided in Section 4.4.

4.1 Wake-up Methods

To trigger the wake-up externally by sending a command from another node or a base station, we consider the following two methods:

- *Wake On Radio (WOR)*: In this method, the main transceiver (e.g., the BLE used here) is duty-cycled by a low-power timer. The nRF5 family have a Real-Time Counter (RTC) that runs from the low-frequency clock source of the system (32 KHz). When used to wake-up the node, the SoC consumes 1.9 µA (excluding the BLE receiver). One of the advantages of this method is that it does not add any additional component to the system. However, as the current consumption of the BLE receiver is around 13 mA for nRF51 and 6 mA for nRF52, the duty cycle value should be very low to take advantage of any power savings. This means putting the node in the *sleep* state for a longer time, which translates to a larger wake-up latency (i.e., larger duty cycle period).

- *Wake-up Radio (WUR)*: In contrast to WOR, a WUR is always ON and ready to receive wake-up commands. Therefore, it must have orders of magnitude lower power consumption than the main transceiver, and it should be custom designed as an ASIC to achieve better performance. Wake-up commands are mainly a simple address to be able to wake-up different nodes in a network selectively. One of the design challenges in WURs is the trade-off between power consumption and sensitivity that sets the maximum range of the WUR.

When deciding to use a of WOR or a WUR, we should also take into account the measurement conditions of the system. When the nodes are in *active* state and making measurements using on-board sensors, the inertial sensor dominates the power consumption of the

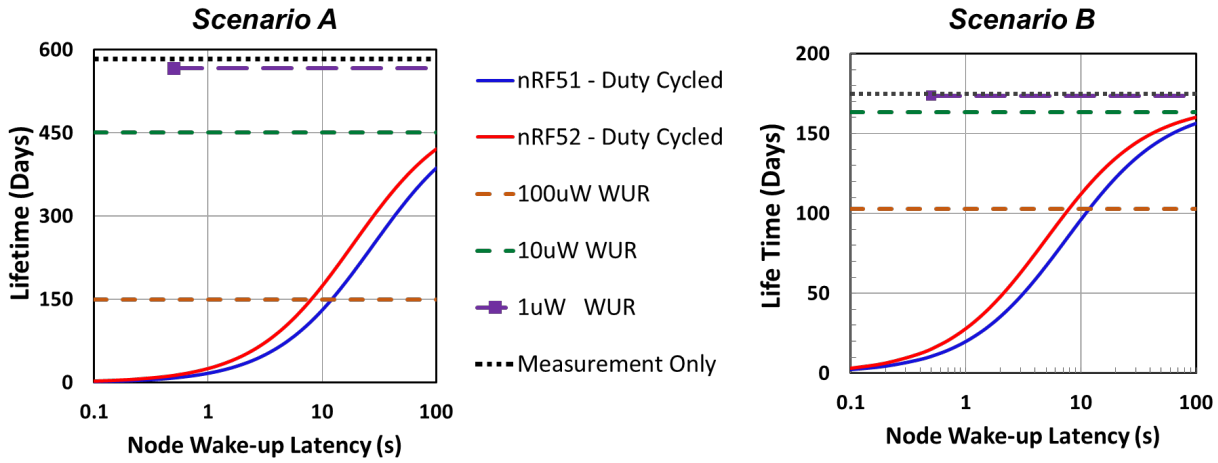


Figure 4.1: Lifetime of the nodes with a 200 mAh battery for Scenario A) 4×1-minute long measurements per day, and Scenario B) 2×1-hour long measurements per week.

nodes by drawing more than 3 mA continuously (compared to < 0.5 mA for the rest of the modules on the nodes). Therefore, the number and duration of each required measurement cycle play an important role in determining the total power usage, and thus the lifetime, of the nodes. We consider two measurement scenarios:

- *Scenario A*: Four measurements, each 1-minute long during a day. This is similar to the condition for HUMS used in trains.
- *Scenario B*: Two measurements, each 1-hour long during a week. This is similar to the condition of HUMS for helicopter rotor.

In Figure 4.1, the lifetime of a node with a 200 mAh battery is plotted against the node wake-up latency for the above two scenarios, when the nRF5 BLE receivers are duty cycled with an ON time of 25 ms. The lifetime is also plotted for three different power consumptions of 1, 10, and 100 µW for the WUR. The node wake-up latency is the duty cycle period in the WOR approach. For the WUR, the node wake-up latency is the time that the node needs to process the received address, which depends on the address length and data rate. The black dotted line in the

two plots represents the maximum achievable lifetime of the node for each scenario, which is calculated by using only the power consumption when in the *active* state and considering a *sleep* state power of zero. All calculations include the required power for synchronization at the beginning of measurements with 8 packets and 0.5 s intervals.

With 10 s latency, lifetime of the duty-cycled nRF5x and the 100-µW WUR are almost the same in both scenarios. When looking at the 1-µW WUR, this lifetime is increased by 3.8x (extra 418 days) and 1.7x (extra 71 days) for scenario A and B, respectively. At 1 s latency for the duty-cycled nRF5x, which is easy to achieve in WURs, the increase in lifetime by having a 1-µW WUR jumps to 22x (extra 541 days) and 6x (extra 145 days) for scenario A and B, respectively. As expected, the increase in lifetime by moving to a lower-power WUR is more significant in scenario A compared to scenario B due to the lower total measurement time, i.e., *active* time. Moreover, the 1-µW WUR achieves 97% and 98% of maximum possible lifetime in scenario A and B, respectively. Therefore, we can consider sub-µw power consumption as one of the design parameters for the WUR in the explained applications.

As discussed in Section 1.4 for the laminated bearing of the rotor (similar to scenario B), continuous operation for 90 days with 1~2 hours recording per week is required. With a 1-µW WUR and the 200 mAh battery of the nodes, a lifetime of more than 170 days is expected. Moreover, with the WOR method and a wake-up latency of 10 s, the lifetime is more than 110 days by using the nRF52. So, both methods satisfy the lifetime requirement of the target application. At the time of preparing for the in-flight test, the WUR chip was not completed. Therefore, for the in-flight test discussed in Chapter 5 the WOR method is used with nRF52 and latencies of more than 10 s.

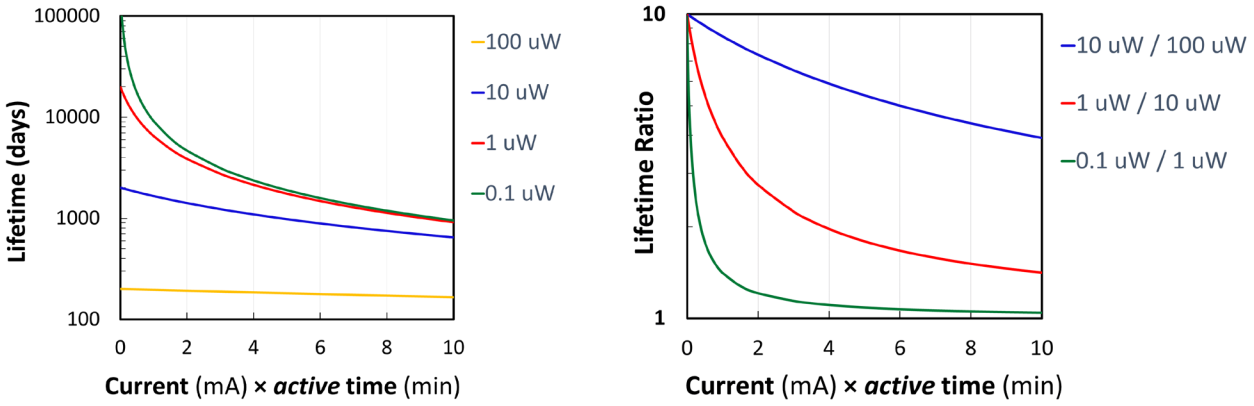


Figure 4.2: Lifetime of the nodes with a 200 mAh battery for four WUR power consumptions plotted against different combinations of current and duration of active state per day (Left figure). Lifetime ratio between WURs with 10x difference in power consumption (Right figure).

To better observe the effect of the measurement cycle on the node lifetime, we consider the product of average required current during *active* time (in mA) and time in *active* state (in minutes) as $I \times t$. Assuming a fixed supply voltage for all modules of the node, this $I \times t$ represents the energy required during the measurements. In Figure 4.2, first, the lifetime of the node is plotted against $I \times t$ for four different WUR power consumptions of 100, 10, 1, and 0.1 µW. Then, the ratio between lifetime of two WUR power consumptions is calculated when the power of the WUR is decreased by 10x starting from 100 µw. It can be seen that with $I \times t$ = 10 mA×min, by going from a 100-µW to a 10-µW WUR, we can achieve a lifetime gain of 3.9x. Decreasing the WUR power from 10 µW to 1 µW results in 1.4x increase in lifetime. However, a node with a 0.1-µW WUR increases the lifetime by only 4% compared with a 1-µW WUR. When considering a lower value of 2 mA×min for $I \times t$, the three above lifetime ratios increase to 7.3 (100/10 µW), 2.7 (10/1 µW), and 1.2 (1/0.1 µW). These plots demonstrate that not necessarily all applications will benefit significantly from having a sub-100 nW WUR, and the lifetime gain also depends on the total energy of the *active* state. As discussed above, for the target application of monitoring the bearings of the helicopter rotor, the nodes can achieve 98% of their maximum lifetime with a sub-µW WUR.

However, in this work the goal is to design a WUR with <100 nW power consumption, which can be used in a broader range of ultra-low-power applications.

4.2 WUR Sensitivity

One of the important parameters of a wireless receiver is its sensitivity, which sets the maximum range of communication. To have an estimate of the required sensitivity for the WUR we use the Friis transmission formula

$$P_R = \frac{P_T\, G_T\, G_R\, c^2}{(4\pi R f)^2} \qquad \textit{Equation 6}$$

where P_R is the received power, P_T is transmitted power, G_R is receiver antenna gain, G_T is transmitter antenna gain, R is the distance between transmitter and receiver, f is the frequency of operation, and c is the speed of light. It should be noted that the Friis transmission formula gives the received power under idealized conditions, including correct alignment of the antennas, and an unobstructed free space between antennas with no multipath propagation. Therefore, the maximum range calculated for a specific sensitivity by this formula and the actual achievable range in practice can be different and dependent to the test setup. As an example, the nRF52 has a sensitivity of -97 dBm and by considering a transmitted power of 0 dBm and antenna gains of 0 dBi, its maximum range by using the Friis formula is around 710 m. In outdoor space and when there is a direct line of sight between the nRF52 transmitter and the receiver, ranges up to 650 m have been reported for this chip [39]. However, in an indoor space the measured range goes down to ~ 50 m [40]. This is mainly due to the obstacles and interference and other factors that affect communication between the transmitter and receiver in indoors, such as people, walls, and furniture.

#	Frequency	Maximum Gain	Dimensions (mm)	Series
1	78-108 MHz	-25 dBi	32 × 11 × 1.6	FracFM
2	902-928 MHz	2.1 dBi	12 × 3.0 × 2.4	RUN mXTEND
3	902-928 MHz	1.7 dBi	18 × 7.3 × 0.8	EZConnect
4	824-960 MHz	1.5 dBi	10 × 3.2 × 3.2	BAR mXTEND
5	698-960 MHz	2.3 dBi	24 × 12 × 2.0	ALL mXTEND
6	698-960 MHz	1.1 dBi	30 × 3.0 × 1.0	TRIO mXTEND
7	824-894 MHz	1.9 dBi	7.0 × 3.0 × 1.0	ONE mXTEND
8	880-960 MHz	1.3 dBi	7.0 × 3.0 × 1.0	ONE mXTEND
9	2.4-2.5 GHz	4.2 dBi	12 × 3.0 × 2.4	RUN mXTEND
10	2.4-2.5 GHz	1.7 dBi	7.0 × 3.0 × 2.0	COMPACT Xtend
11	2.4-2.5 GHz	1.3 dBi	6.7 × 6.7 × 0.8	REACH Xtend
12	2.4-2.5 GHz	1.1 dBi	7.0 × 3.0 × 0.9	SLIM Xtend
13	2.4-2.5 GHz	0.2 dBi	4.1 × 2.0 × 1.0	MICRO Xtend
14	2.4-2.5 GHz	2.4 dBi	3.0 × 2.0 × 0.8	NANO mXTEND
15	2.4-2.5 GHz	1.8 dBi	7.0 × 3.0 × 2.0	DUO mXTEND
16	1.71-2.69 GHz	3.0 dBi	10 × 3.2 × 3.2	BAR mXTEND
17	1.71-2.69 GHz	3.1 dBi	24 × 12 × 2.0	ALL mXTEND
18	1.71-2.69 GHz	2.4 dBi	30 × 3.0 × 1.0	TRIO mXTEND
19	1.71-2.17 GHz	1.7 dBi	7.0 × 3.0 × 1.0	ONE mXTEND
20	1.85-2.17 GHz	1.8 dBi	7.0 × 3.0 × 1.0	ONE mXTEND

Table 4.1: Specifications of antennas used in the Friss equation for different frequencies

For fixed transmitter power and antenna gains, the received power is lower for higher frequencies at a fixed distance. However, when the size of the nodes is important, we need to consider the effect of lower frequencies on the antenna size. In Table 4.1, the gain and size of different commercial antennas in four frequencies of 100 MHz, 915 MHz, and 2.4 GHz are listed. These are the frequencies that have been used in most of the recent published WUR works. By assuming a fixed receiver sensitivity of -50 dBm and transmitted power of 10 dBm, the gain of these antennas is used in Equation 6 to calculate the theoretical range, i.e., maximum distance between the transmitter and receiver. These calculated ranges are plotted against the volume of their corresponding antenna in Figure 4.3. Then the range is normalized by the volume of the antenna, i.e., range/volume, and the results are shown in Figure 4.4. It is clear that the 100 MHz antenna has the largest size, the lowest gain of -25 dBi, and consequently the lowest normalized

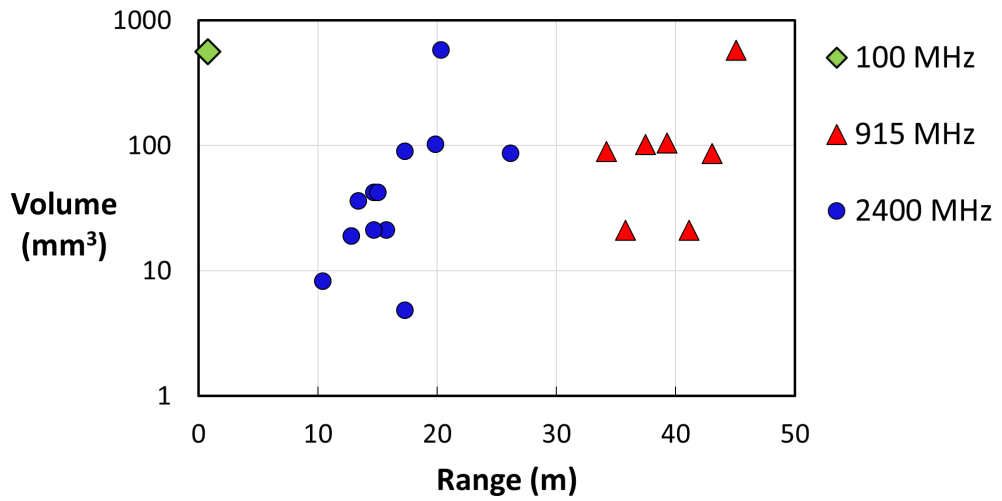


Figure 4.3: The relation between size and calculated range for the antennas listed in Table 4.1 by assuming a transmitted power of 10 dBm and receiver sensitivity of -50 dBm.

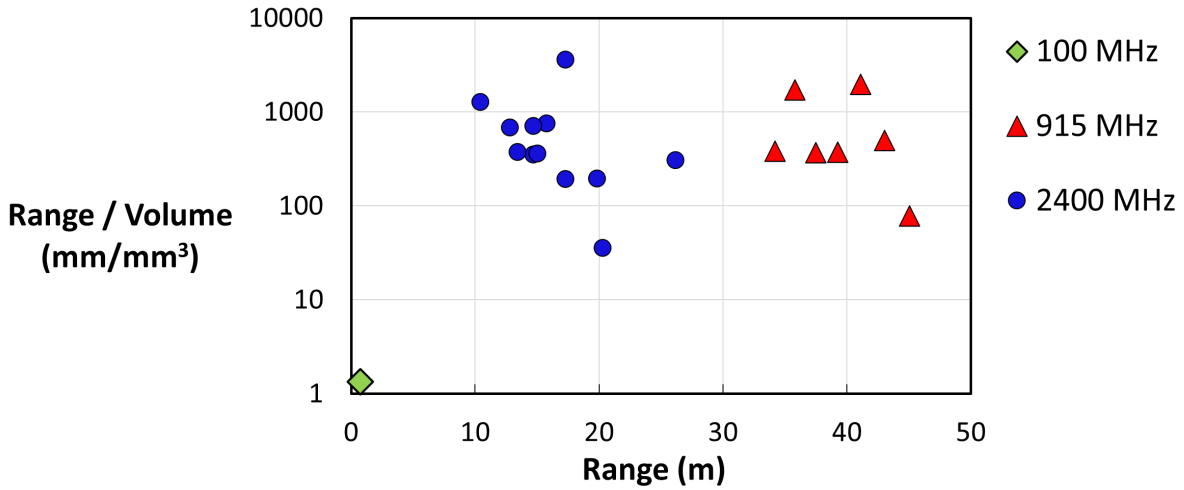


Figure 4.4: Range of the antennas listed in Table 4.1 normalized by their volume. Range is calculated by assuming a transmitted power of 10 dBm and receiver sensitivity of -50 dBm

range. One of the 2.4 GHz antennas (#14 in Table 4.1) has the highest normalized range, mainly because of its lowest volume. However, all the 2.4 GHz antennas achieve shorter ranges compared with the 915 MHz ones. The 915 MHz antennas, in addition to having the longest ranges, are around the same size as of the 2.4 GHz antennas and have similar normalized ranges.

While Figure 4.3 and Figure 4.4 are plotted for a fixed receiver sensitivity, another way to compare different frequency bands is to calculate the range for different receiver sensitivities.

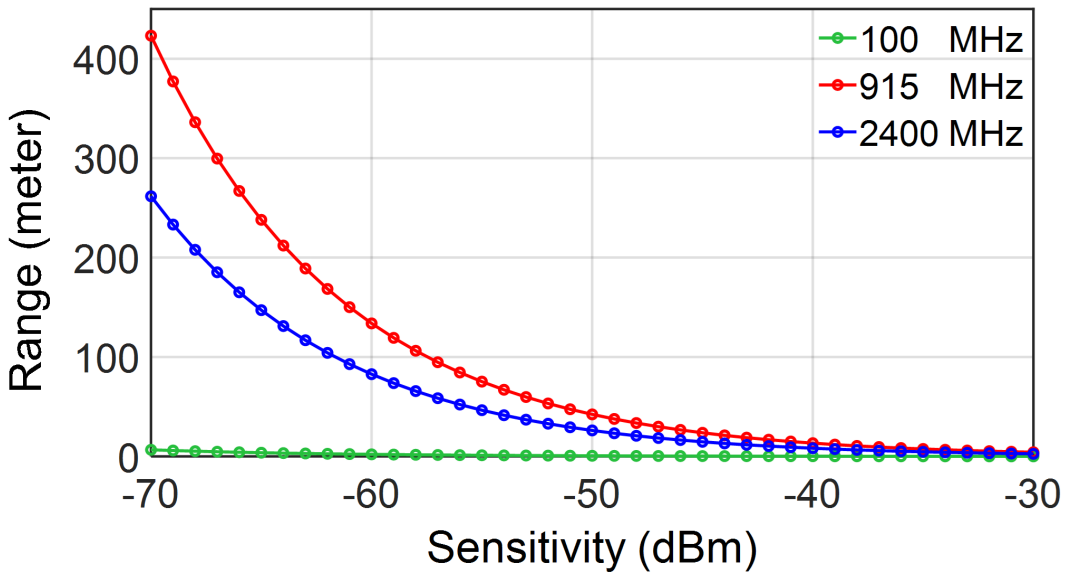


Figure 4.5: Achievable range based on different receiver sensitivities at three frequencies.

Therefore, the gain of the three antennas listed in Table 4.1 (#1, 2, and 9) are used to plot the theoretical range against the receiver sensitivity at 100 MHz, 915 MHz, and 2.4 GHz (Figure 4.5). The assumption is that the same antenna is used in both transmitter and receiver, and P_T = 10 dBm (10 mW). It is clear that for a specific sensitivity, the 915 MHz receiver achieves a longer range. In other words, to cover a specific range, the 915 MHz receiver can have a lower sensitivity (requiring a lower sensitivity for the same distance means the received power is higher, and it is a positive point).

For the HUMS applications discussed here and many other WSN applications, a 50 to 100 m range is adequate. It should be noted that we are using this range to estimate the required sensitivity for the WUR based on the Friis transmission formula. As discussed earlier in this section, the Friis formula gives the sensitivity under idealized conditions and with an unobstructed free space between antennas. Therefore, considering a 50~100 m range does not mean that we want exactly such a range for the HUMS applications, e.g., master and nodes installed on a

helicopter rotor. Obviously, the maximum distance between the master and nodes in a helicopter is only a few meters and is much smaller than 50 m. But, considering the simplifications assumed in the Friis transmission formula, we need to over design for the range and consequently sensitivity to guarantee achieving the desired performance in the real setup.

The 50~100 m range translates to -50 to -55 dBm sensitivity at 915 MHz and -55 to -60 dBm sensitivity at 2.4 GHz. For the 100 MHz frequency, due to the low antenna gain of -25 dBi, the range is below 10 m even with a -70 dBm sensitivity. Therefore, both 915 MHz and 2.4 GHz receivers can provide the required range. The only difference is that the 2.4 GHz receivers must have a few dBm higher sensitivity to achieve the same range as the 915 MHz receivers. The difference of ~5 dBm obtained from Figure 4.5 between the 915 MHz and 2.4 GHz receivers (for a specific range) is affected by the specific antennas used for calculation (#2 and #9 in Table 4.1 with the same size) and can be different for another set of antennas. On the other hand, as shown in Figure 4.3, the smallest size antennas are for the 2.4 GHz band and wherever size is the limiting factor, they can have advantages over the 915 MHz antennas. In this work, we choose the 915 MHz band because of longer range and lower receiver sensitivity requirements.

4.3 WUR Architecture

4.3.1 *Previous Works*

To achieve sub-μW and even nanowatt power consumptions, typically the WUR is designed based on a passive front-end architecture. In this architecture, the first module in the path of the received signal is a passive RF rectifier, and no active RF amplification is implemented to save power. Generally, On-Off Keying (OOK) modulation is used, and if the output voltage of the rectifier is above a certain threshold, it will be detected as a data bit “1”. A comparator is used to perform this comparison. Depending on the design, there might be a baseband (BB) amplifier

between the output of the rectifier and the comparator. A wake-up code or address is assigned to each node to be able to wake-up nodes selectively. Therefore, a digital block is required after the comparator to search for a match between the received data and the node's address. This block is typically implemented by digital correlators and generates the final wake-up signal.

The initial sub-10nW WURs had bulky off-chip transformers (9 cm^2) [41] and matching networks (2.3 cm^2) [42] to achieve more than 25 dB passive gain in the 100-433 MHz frequency range. While in [43] designing the transformer at 9 GHz helped reduce its size, but its -64 dBm sensitivity results in a range equal to a WUR with 20 dBm lower sensitivity at 915 MHz. Moreover, its designed patch antenna for an extra -5 dBm sensitivity was still 4 cm^2. As explained in Section 4.2, the sensitivity of the WUR is an important factor to calculate its range. However, the range is also affected by the frequency band and antenna size, and comparing only the sensitivity of different works, does not give us a clear picture of their relative ranges. Therefore, instead of sensitivity we use range for comparison, and include the dimension of antenna in the overall system size.

In Figure 4.6 the relation between power, range, and size is plotted for prominent WUR designs with the passive front-end architecture (note that the plots are color coded for the frequency of operation). These designs are again used at the end of this chapter (Section 4.4) for comparison with our work, and their detailed specifications are listed in Table 4.3. From the sub-10 nW works, [41] and [42] (green bars) have, respectively, 16x and 8x larger sizes compared to the 2.4 GHz works (blue bars), mainly due to their off-chip components and antenna size at 100-200 MHz. Their ranges are also <13 m, which is 5-25x lower than the 2.4 GHz WURs. In the 9 GHz design [43], size is ~ 20% lower than the 2.4 GHz works, but its range is also 2-5x lower. The WUR in [44] is for a standard communication protocol, i.e., BLE, with a power consumption of 236 nW

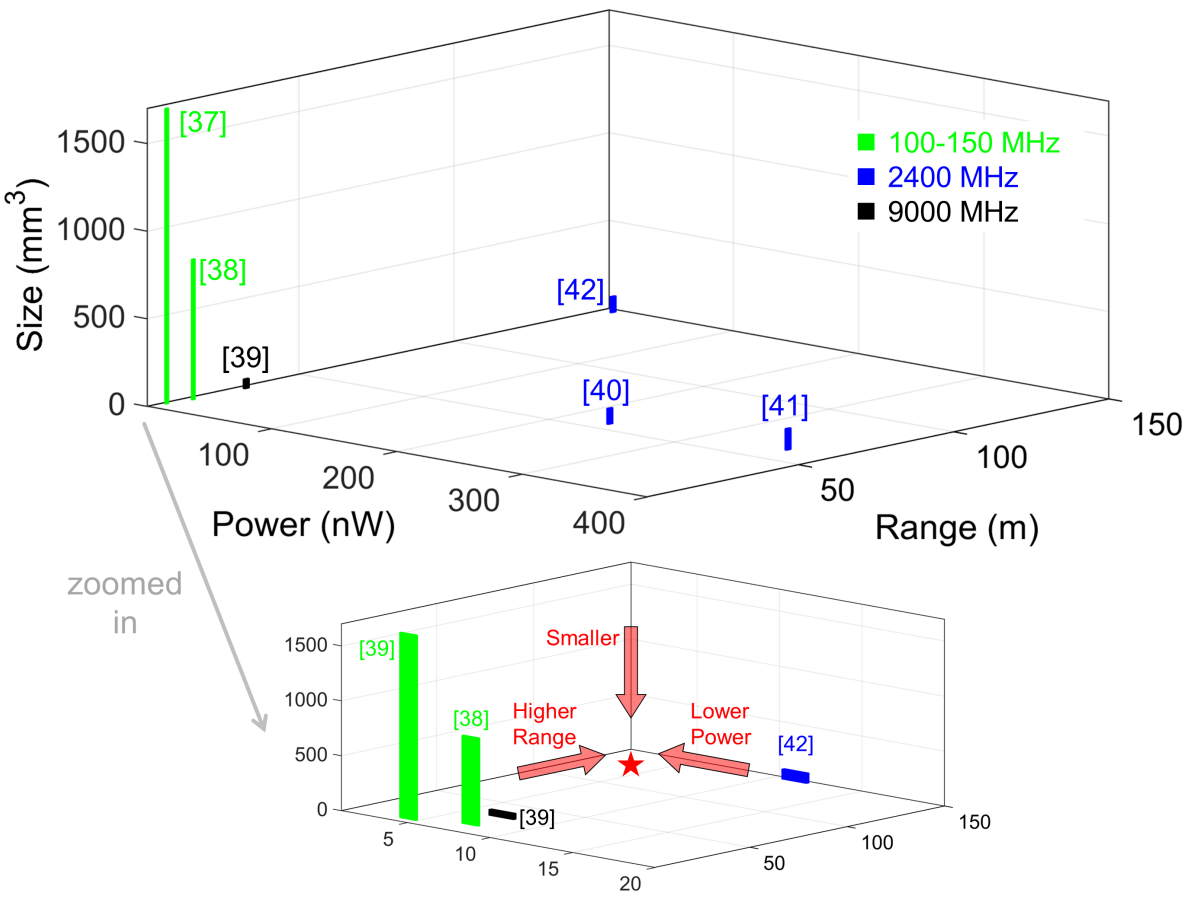


Figure 4.6: Relation between power, range, and size of previous works with the passive front-end architecture.

for 8.192 kbps data rate, and requires an external 32 KHz crystal oscillator. The WUR in [45] is based on an antenna-rectifier co-design for passive gain, and the reported performance is for using the specific designed antenna with an area of 1.23 cm^2. In the reported WUR designs, employing a lower data rate is one the techniques to lower the power consumption. Compared to [44] with 8.192 kbps and [45] with 2.5 kbps, all other four designs [41]–[43], [46] have data rates <300 bps, with 33.3 bps being the lowest in [43]. The data rate and number of bits in the node's address set the wake-up latency. For many applications, including HUMS, a latency of <1 s is acceptable, which justifies using 100-300 bps data rates for 8-32 bit addresses.

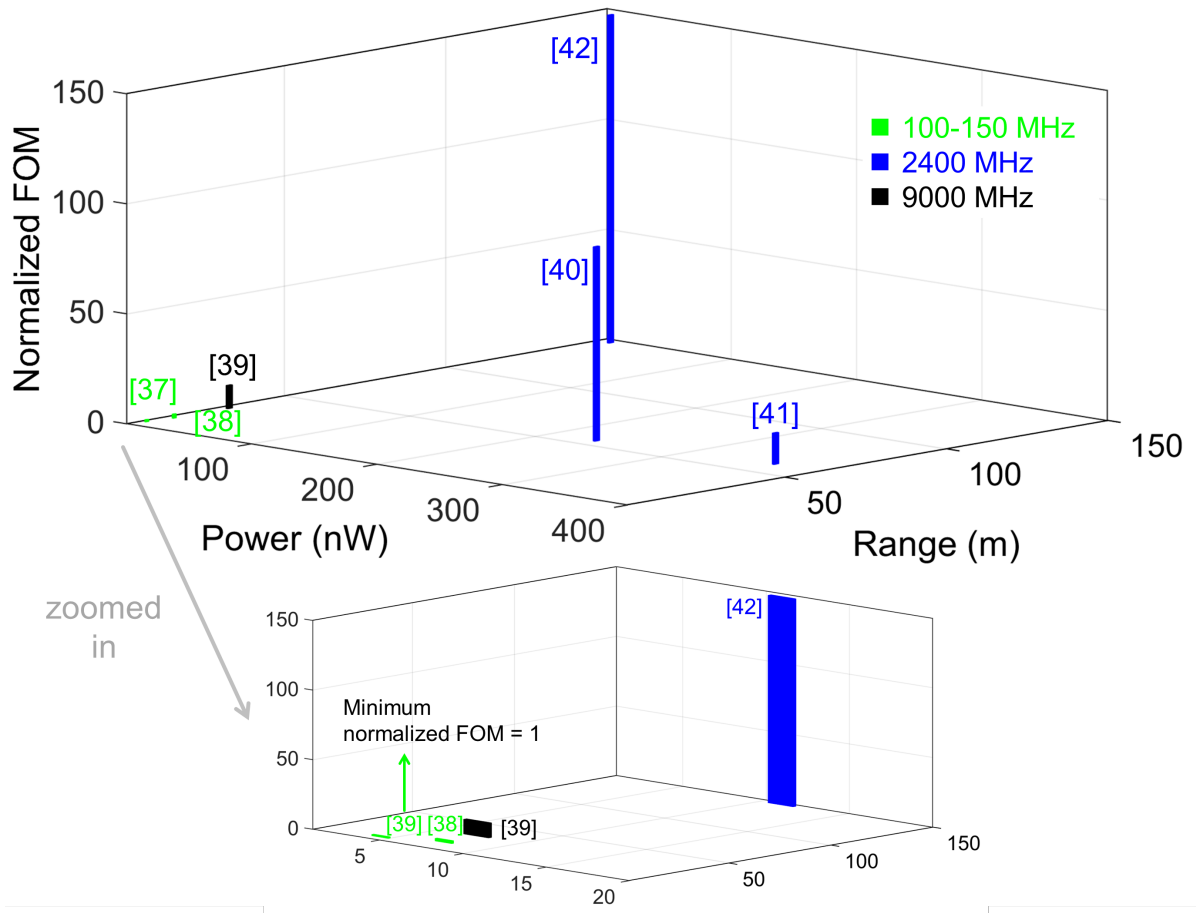


Figure 4.7: FOM of previous works with the passive front-end architecture.

To include all important WUR parameters in comparing different designs, we can define a $FOM = (Range \times DataRate)/(Power \times Size)$. This FOM is calculated and then normalized to the lowest value for the discussed WURs above (Figure 4.7). As expected, [41] and [42] have the two lowest FOMs of 1 and 1.9, respectively, which is due to their bulky off-chip components. Although [44] consumes 236 nW power at 2.4 GHz, it has the second highest FOM of 88, because of its high data rate and relatively small size. The highest FOM is 146 and for the WUR reported in [46], which operates at 2.2 GHz with a reduce data rate of 250 bps. While the 9 GHz design [43] has the lowest size, its FOM is only 10 due its low data rate (33.3 bps) and low range (30 m).

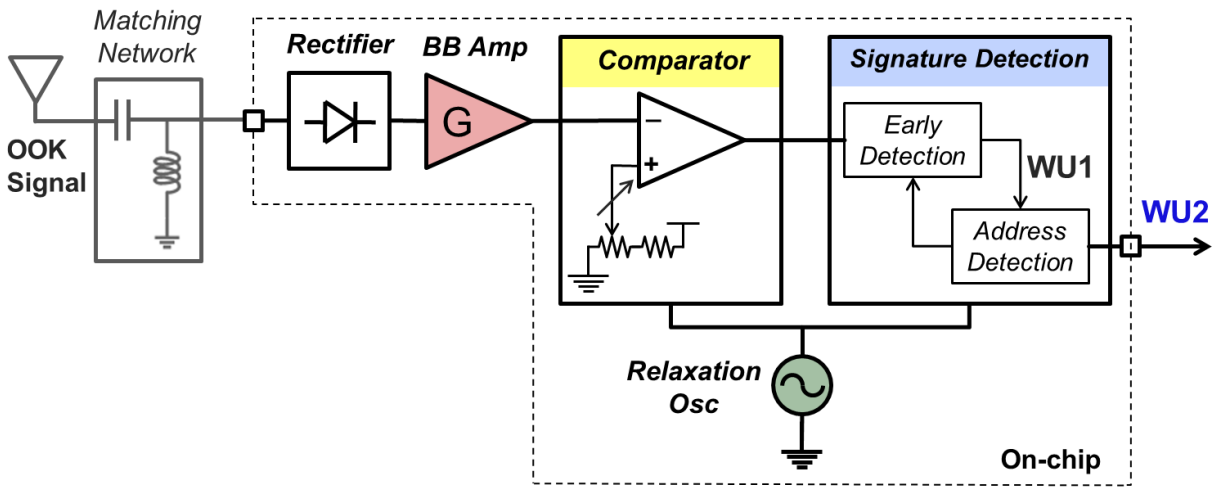


Figure 4.8: Block diagram of the WUR.

Figure 4.8 shows the block diagram of the passive front-end architecture that is used in our design. An off-chip matching network provides a passive voltage boost to the RF signal with OOK modulation, and it requires only two small SMD components (0.0128 cm^2 each). The goal is to lower the power consumption and achieve the desired range and without significantly increasing the system size.

The first on-chip block is the passive RF rectifier, which is implemented with diode-connected transistors and is followed by a gain stage in baseband (BB) to increase sensitivity. The amplified OOK signal is then compared with a reference voltage to generate the digital bit stream. A dynamic comparator with 2x oversampling is used, and its clock is generated from an on-chip oscillator. The detected data is then passed to the signature detection block for comparison with a pre-defined pattern, i.e., node address. The signature detector is composed of an early detection and an address detection block, forming a 2-stage wake-up mechanism. In the following subsections, we first explain the proposed 2-stage wake-up architecture and discuss its advantages. Then, the design and circuitry of other building blocks are presented. The chip is designed in the

TSMC 65 nm CMOS-LP technology with a 0.4 V power supply and target data rate of 100 bps at 915 MHz.

4.3.2 *Two-Stage Wake-up Method*

The concept of the 2-stage wake-up is mainly based on adding patterns that are easy to detect before the transmitted address, and continuously check for them in the first stage of the signature detection block, while the second stage is in standby. Whenever the pattern is detected, a wake-up 1 (WU1) signal is generated by the early detection block that turns on the address detection block, i.e., the second stage, which is consisted of shift registers and correlators. After WU1 is activated, the oversampled received data are clocked into the second stage shift registers, and the number of total bits matched with a programmable address is calculated after every shift. Whenever this number is higher than a programmable threshold, wake-up 2 (WU2) signal is generated. Previous architectures did not have the first stage used here, and the shift registers and correlators were continuously active, calculating the correlation with the coded address with every clock. One way to explain the advantage of a 2-stage wake-up is by considering the false wake-up rate.

False wake-ups can be triggered by circuit noise in steady state and interference from other nodes. They should be minimized as they add to the average power consumption of the node in the *sleep* state. In Figure 4.9, the average power of a node is plotted against false wake-up rate for a WUR power of 20 nW, active time of 15 ms, and active power of 12 mW. The active time is the time that the main transceiver of the node is ON after wake-up and searches for a confirmation of the wake-up. If no confirmation is received, which is the case for false wake-ups, the node goes back to *sleep* with only the WUR being active. Based on Figure 4.9, a false wake-up rate of <1/h is a reasonable assumption.

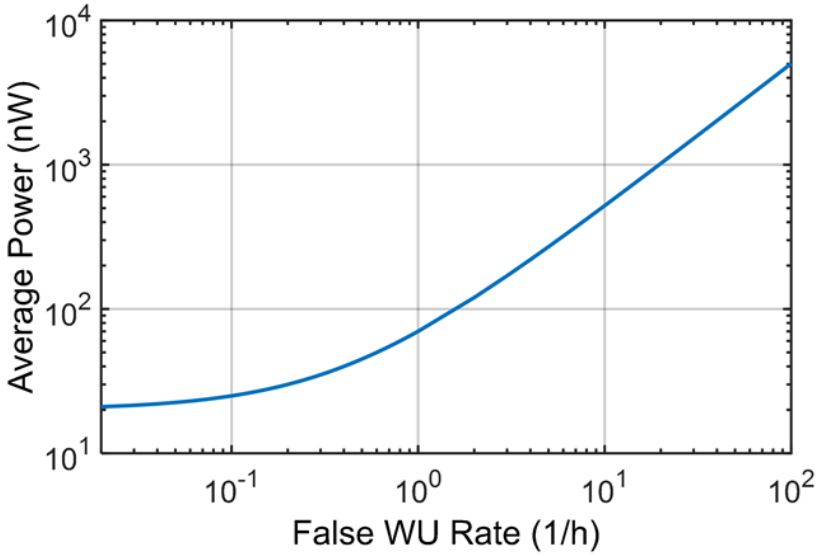


Figure 4.9: Calculated average system power in sleep mode based on false wake-up rate, for a WUR power of 20 nW, active time of 15 ms, and active power of 12 mW.

The false wake-up rate can be simulated by considering the false positive bit rate (FPBR), which represents the cases when a "1" is detected instead of "0" in the received data. One way to lower false wake-ups is to increase the length of address codes. Figure 4.10 shows the probability of having <1/h false wake-up for 2x oversampled 8-bit and 16-bit codes with 3 (out of 16) and 6 (out of 32) bits error tolerance, respectively (bit error tolerance is the maximum number of bits that can be detected falsely, but still result in an address match by the correlators in the address detection block). The maximum possible FPBR is 1.6% for the 8-bit code and 5.4% for the 16-bit code, showing around 3x improvement for the longer code. However, in longer codes, we should consider that the maximum frequency error between the actual sampling rate (i.e., oscillator frequency) and the nominal sampling rate is equal to ±1/*Nc*, where *Nc* is number of correlator bits [43] and is equal to the code length multiplied by the oversampling rate. This means that long addresses require a more accurate clock, which becomes harder to achieve in nanowatt power levels.

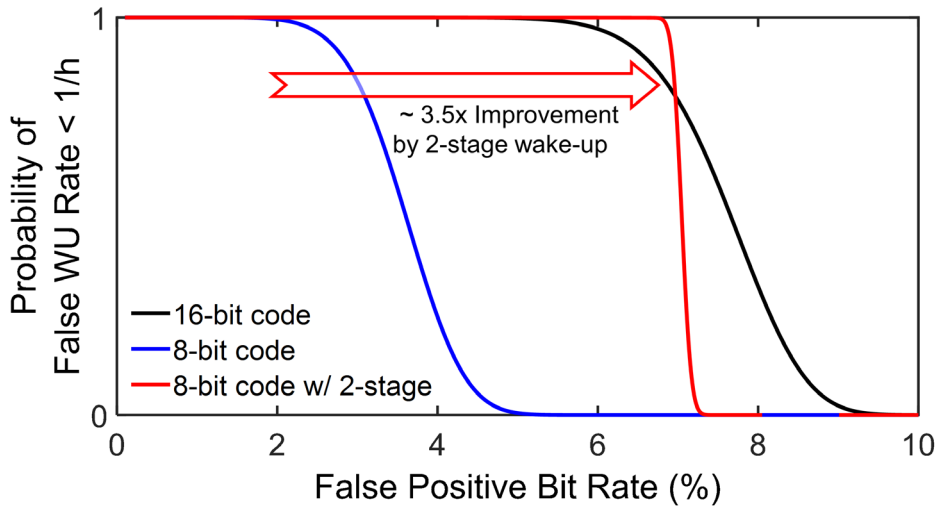


Figure 4.10: Probability of <1/h false wake-ups based on the false positive bit rate (FPBR), plotted for three cases to show the effect of increasing the code length and using the 2-stage wake-up method.

In the 2-stage approach here, a "1010101010" sequence is transmitted before the desired address, and in the first stage we are looking for 4 to 6 falling edges in 10 bit-time (T_b=1/data rate). This is achieved by enabling a 5-bit timer to keep track of T_b counts, and a 4-bit counter sensitive to falling edges of data (Figure 4.11). If WU1is not activated in $10T_b$, the timer and counter are disabled until the next bit "1" is detected. Counting more than 4 falling edges in $4T_b$ resets the detection, filtering out toggling bits with frequencies higher than 2x the data rate. Based on the data rate of other transmitters in the channel, similar filters can be implemented by using different combinations of timer and counter values. Considering the above configuration for the first stage of an 8-bit code, the probability of <1/h false wake-up is also plotted in Figure 4.10 resulting in a maximum FPBR of 6.7%. This value is more than 4x larger than the FPBR of the normal (1-stage) 8-bit code and slightly (~ 20%) larger than the FPBR of the normal 16-bit code. Therefore, the 2-stage architecture can help reduce false wake-ups without increasing the address bits that impose lower frequency error requirements for the oscillator frequency. This is due to the fact that the configuration used in the first stage does not affect the accuracy of the oscillator frequency, but it

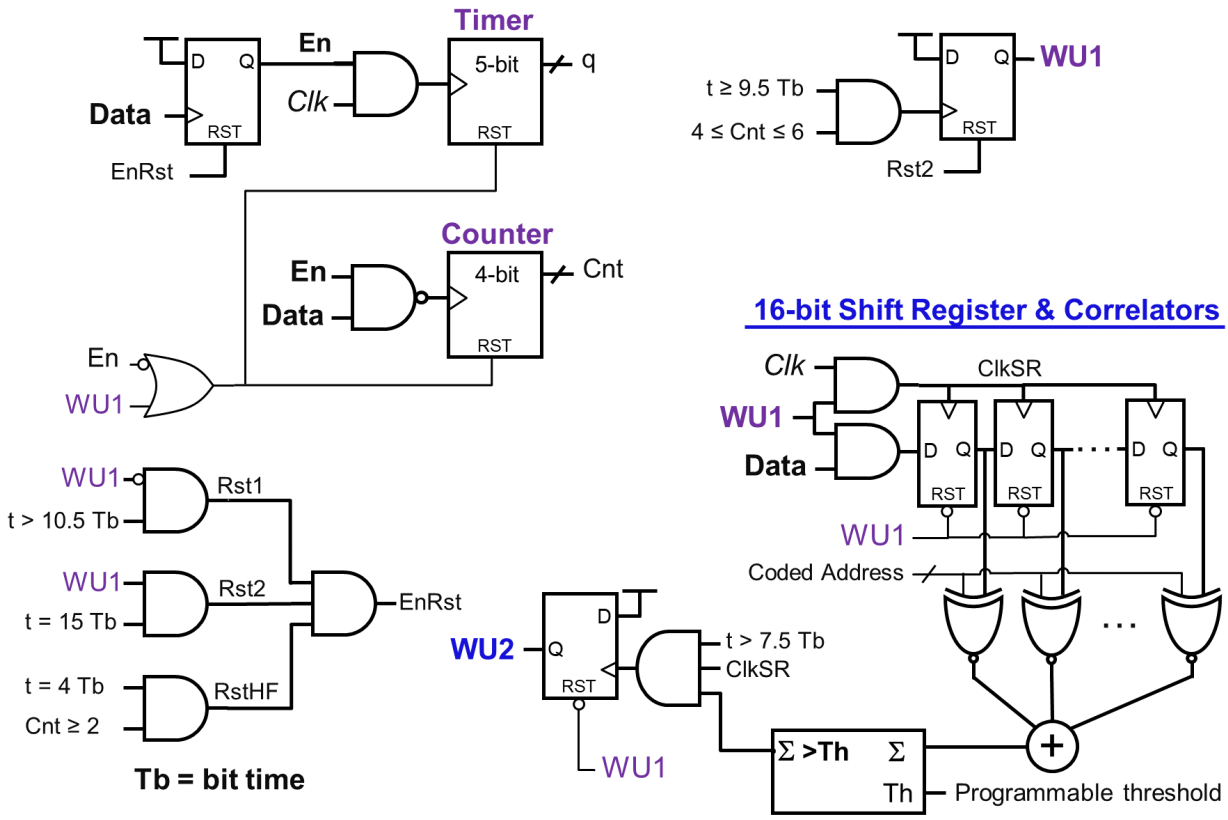


Figure 4.11: The proposed two-stage wake-up detection.

contributes to lowering the probability of false wake-ups. Note that by frequency error requirement, we mean the maximum error between the oscillator frequency and the nominal sampling rate. This error as discussed before is inversely proportional to number of the address bits.

Another advantage of the 2-stage approach is that in previous architectures, before the received address is completely clocked into the shift register, the window of calculating the correlation includes previous random detected bits in the channel, and out of position address bits. Both can increase the probability of false wake-ups. Therefore, the address codes and the threshold should be chosen carefully to have a balance between false and missed wake-ups. As a result, some codes might be excluded from potential addresses, due to their high chance of causing false wake-

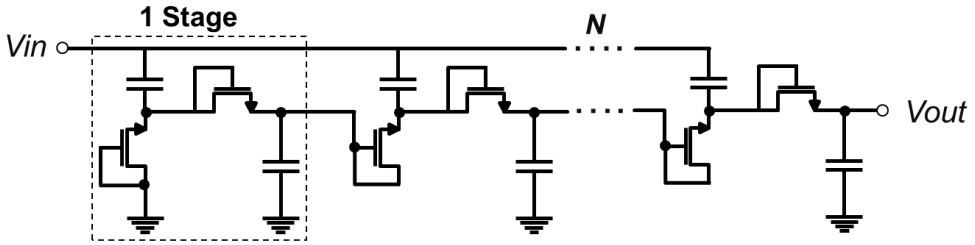


Figure 4.12: N stage CMOS rectifier based on the Dickson Multiplier.

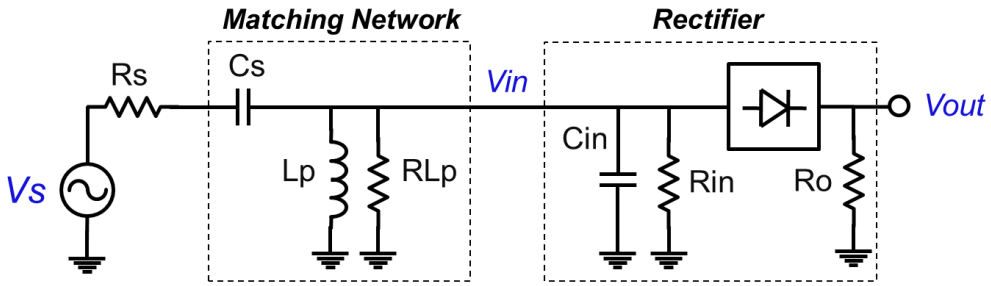


Figure 4.13: Simplified circuit of the matching network and rectifier for calculating passive gain.

ups. All of these become more challenging in short length addresses, as the range of change for the threshold and number of possible address codes are smaller. With the 2-stage method, after WU1 is detected: a) WU2 can only change state after the full address is clocked in, preventing any high number of matched bits during the shift to cause a wake-up; and b) all register bits can be initialized to predefined values, eliminating any uncertainty.

4.3.3 *Passive Rectifier*

The passive CMOS rectifier is implemented using the Dickson multiplier ([47], [48]), which operates in the sub-threshold region (Figure 4.12). The number of stages, transistor type, and transistor size are the design variables for the rectifier [48]. In energy detector receivers, similar to the architecture used here, the noise from the energy detector (i.e., passive rectifier) is dominant for setting the receivers sensitivity [49]. Therefore, we consider the simplified circuit in Figure 4.13 to calculate the SNR at the output of the rectifier (SNR_{out}). In this circuit V_s and R_s are

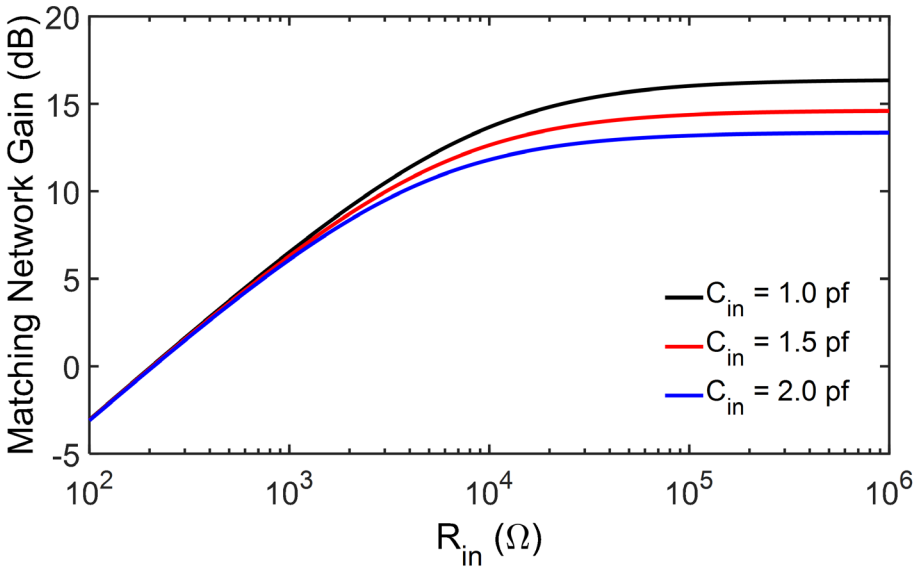


Figure 4.14: Matching network gain with Q_{ind} =50 at 915 MHz for three different C_{in} values.

the antenna voltage and its impedance; C_s and L_p form the matching network with R_{Lp} showing the parasitic resistance of the inductor; C_{in} and R_{in} are the capacitance and resistance seen at the input of the chip. The input capacitance C_{in} is a combination of the rectifier input capacitance, and parasitic capacitances from the inductor, package, bond-wire and pad. The rectifier input capacitance is a function of the number of stages and capacitance of each diode (C_D), which is in the range of 1 fF. All other parasitic capacitances are represented by C_P, which can be around 1-1.5 pF and dominates the C_{in} value.

The passive gain of the matching network depends on the loading condition of the matching network (R_{in} and C_{in}) and is calculated by

$$\frac{V_{in}}{V_s} = \frac{1}{2}\sqrt{\frac{R_{in} \parallel R_{Lp}}{R_s}} \qquad \textit{Equation 7}$$

where $R_{Lp} = \omega L_p Q_{ind}$ and assuming that $L_p \approx 1/\omega^2 C_{in}$, R_{Lp} can be related to C_{in} by $R_{Lp} = Q/\omega C_{in}$. Figure 4.14 shows the matching network gain for different R_{in} and three C_{in} values with Q_{ind} = 50 at 915 MHz. As expected from Equation 7 and R_{Lp} formula, larger C_{in} values reduce gain and

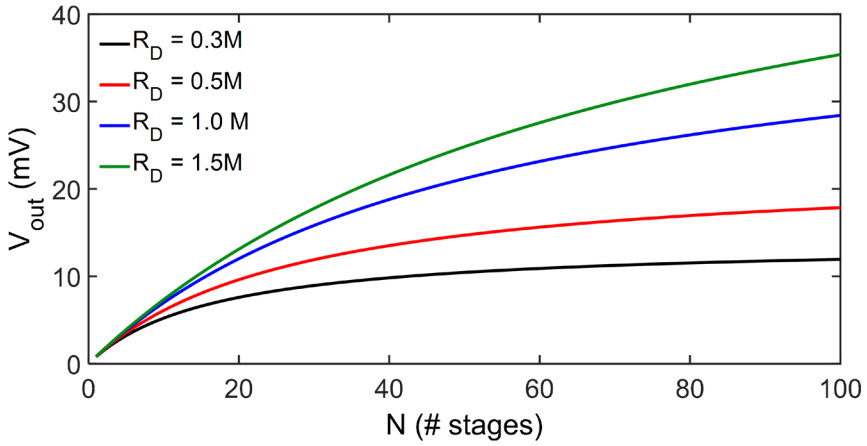


Figure 4.15: Output voltage of the rectifier including the matching network with C_{in}=1 pF, C_D=1 fF, Q_{ind} =50, and input power of Pin= -50 dBm at 915 MHz for four different diode resistance (R_D) values.

should be minimized. The output voltage of the rectifier (V_{out}) in the sub-threshold region is given by [48]

$$V_{out} = 2N \cdot V_T \cdot ln\left(I_0\left(V_{in}/V_T\right)\right) \qquad \textit{Equation 8}$$

where N is number of stages, V_T is the thermal voltage ≈ 26 mV, and I_0 is the zero-th order modified Bessel f unction of the first kind. By replacing V_{in} from Equation 7, V_{out} is plotted in Figure 4.15 for four different diode resistance (R_D) values with C_p = 1 pF, C_D = 1 fF, Q_{ind} = 50, and input power of P_{in} = -50 dBm. Similarly, and with the same above parameters, the power spectral density (PSD) of the rectifier output noise can be estimated by $PSD_{out} = 4K_B\ T\ Rout$, where K_B is Boltzmann's constant, T is absolute temperature, and R_{out} is $2N \times R_D$ (Figure 4.16).

As both V_{out} and PSD_{out} increase at different rates with N, it is expected that an optimum point can be found for the SNR_{out}. Figure 4.17 shows the changes in SNR_{out} (left y-axis) and R_{in} (right y-axis) for varying number of stages N. When increasing N up to the optimum N, the rate of change for the gain is higher than the output noise, but after that while the gain is still increasing, the rate of increase for the output noise becomes higher and the SNR_{out} drops. The slight difference

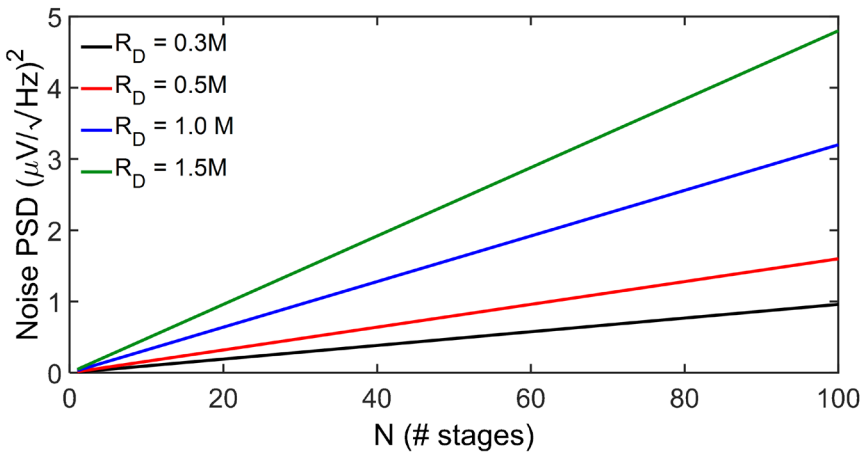


Figure 4.16: PSD of the output noise of the rectifier.

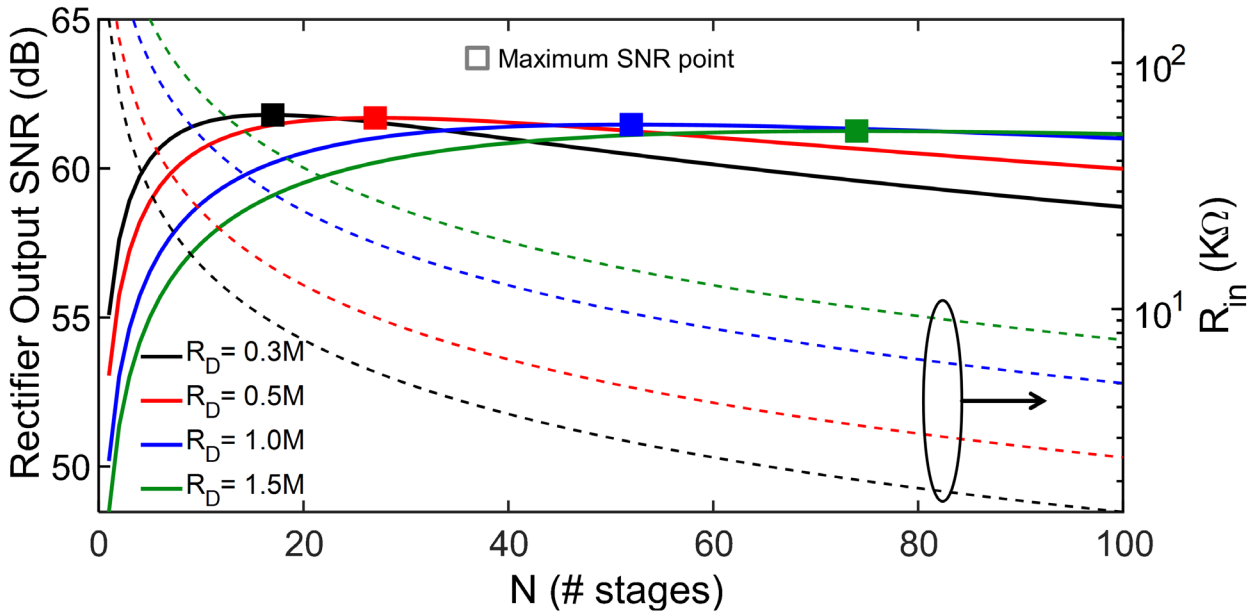


Figure 4.17: SNR of rectifier output with C_p=1 pF, C_D=1 fF, Q_{ind} =50, and P_{in}= -50 dBm at 915 MHz.

in maximum SNR_{out} for different R_D values is due to the small difference in C_{in} as a result of different N values. Remember that C_{in} is dominated by the parasitic capacitance of 1-1.5 pF and C_D has small effect on the C_{in} value. By ignoring the effect of C_D, SNR_{out} and N are plotted this time against varying R_{in} in the left and right y-axis of Figure 4.18, respectively. It can be seen that the SNR_{out} plot is the same for all four R_D values, and they have only different N values to achieve

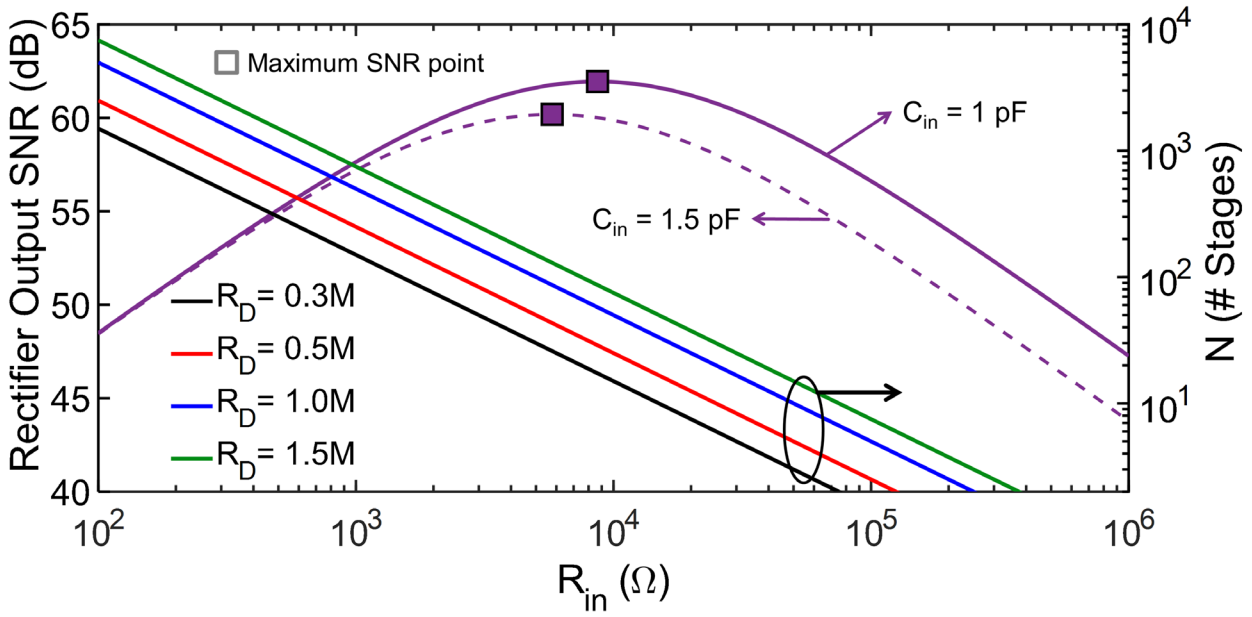


Figure 4.18: SNR of rectifier output with Q_{ind} =50, and P_{in}= -50 dBm at 915 MHz; ignoring C_D and considering two fixed input capacitances of 1 & 1.5 pF.

the optimum R_{in}. On the other hand, SNR_{out} decreases when C_{in} is increased from 1 pF to 1.5 pF, which is expected considering the effect of C_{in} on the matching network gain.

In this work, by considering the SNR_{out} and V_{out} plots, a 50-stage Dickson multiplier is implemented with zero-voltage-threshold (ZVT) transistors that are sized to provide the optimum required R_{in}. Moreover, the coupling capacitors are 62 fF MOM (metal-oxide-metal) capacitors, which is the lowest capacitance available in the design kit.

4.3.4 *Base-band Amplifier and Comparator*

The rectifier output is amplified with a baseband two-stage amplifier working in the subthreshold region and with an active mirror as the first stage (Figure 4.19). The amplifier is designed to provide more than 40 dB gain with a cut-off frequency of 100 Hz. The Large gain of the amplifier help relax the requirements for setting the trip point of the comparator. A dynamic comparator based on a double-tail latch [50] and with 2x oversampling is used to save static power.

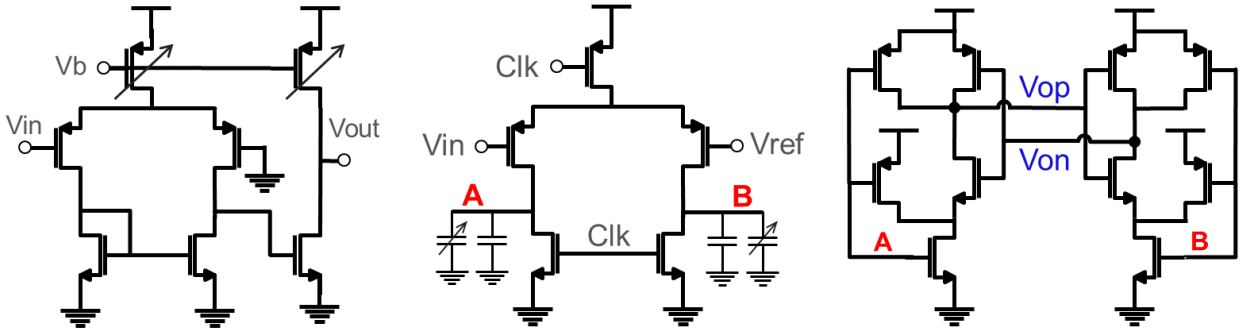


Figure 4.19: Two-stage baseband amplifier and double-tail latch dynamic comparator.

As shown in Figure 4.19, nodes A/B are connected to a fixed 250 fF MOM capacitor to reduce the comparator noise. Three programmable and binary weighted capacitors (LSB=30 fF) are also included in nodes A/B to compensate for the input offset and fine tuning the trip point. The reference voltage of the comparator is provided by a resistive ladder implemented with 32 diode-connected transistors and 5-bit multiplexer to achieve a 12.5 mV step size.

4.3.5 *Clock and Biasing*

A dual-phase relaxation oscillator architecture [51] is used with some modifications to generate the required system clock of f_s = 2 × (data rate) = 200 Hz. First, its reference branch is merged with the beta-multiplier of the chip (Figure 4.20). This helps to save an extra branch of copied current and save the area of an additional on-chip reference resistor, which occupies 160 µm x 94 µm for a programmable 15-29 MΩ resistance, implemented by a combination of P+ diffusion and poly resistors. Moreover, even with currents smaller than 1 nA for the beta-multiplier, generating a few hundred Hz clocks require large capacitors. To reduce the size/area of the capacitors needed, the top current sources of the oscillator are sized 6x smaller than the beta-multiplier PMOS transistors. This results in around 100 pA current for the oscillator, and as the leakage of the switch in the off branch can affect the charging current, two separate current sources

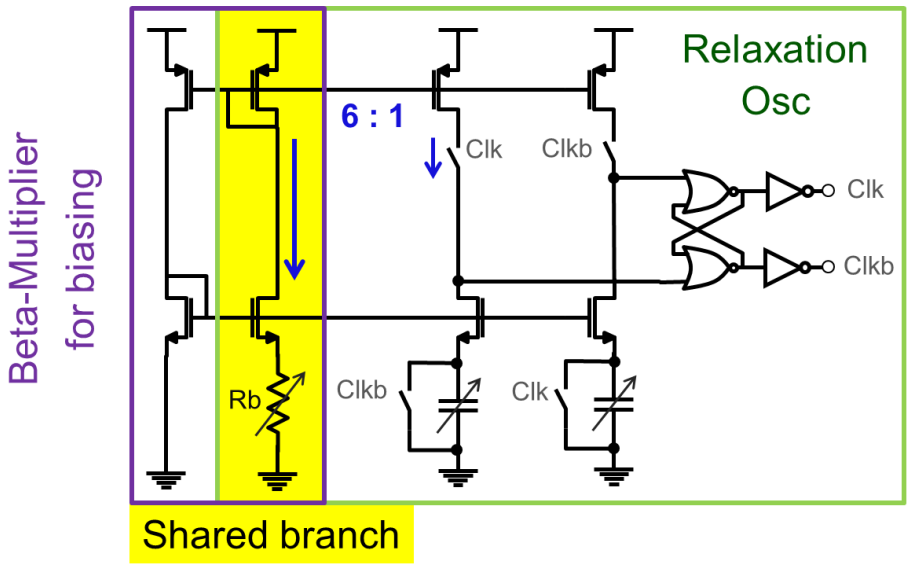


Figure 4.20: Dual-phase relaxation oscillator merged with the beta-multiplier, and two separate current sources with 6:1 current ratio to reduce capacitor sizes for low-frequency clocks.

are considered for each branch. The bias resistor of the beta-multiplier has a fixed 15.5 MΩ (8 units) resistance and a programmable 3-bit part with LSB=1.94 MΩ. The priority of adjustment for this resistor is to set the biasing point of the amplifier, and the oscillation frequency is mainly adjusted with 6 binary-weighted MIM (metal-insulator-metal) capacitors on each branch with LSB=125 fF. Simulations performed in different corner cases confirm that after adjusting the bias resistors to a fixed valued for the amplifier bias, a combination of the capacitors of the oscillator exists that can provide the required 200 Hz clock for the system.

4.4 Measurement Results

The chip was fabricated in the TSMC 65nm CMOS-LP technology and the die photo is shown in Figure 4.21. The WUR blocks occupy an area of 0.604 mm × 0.764 mm = 0.462 mm^2 excluding the pads. The die is packaged in a QFN32-5×5 package to test the WUR with the designed test PCB (Figure 4.21).

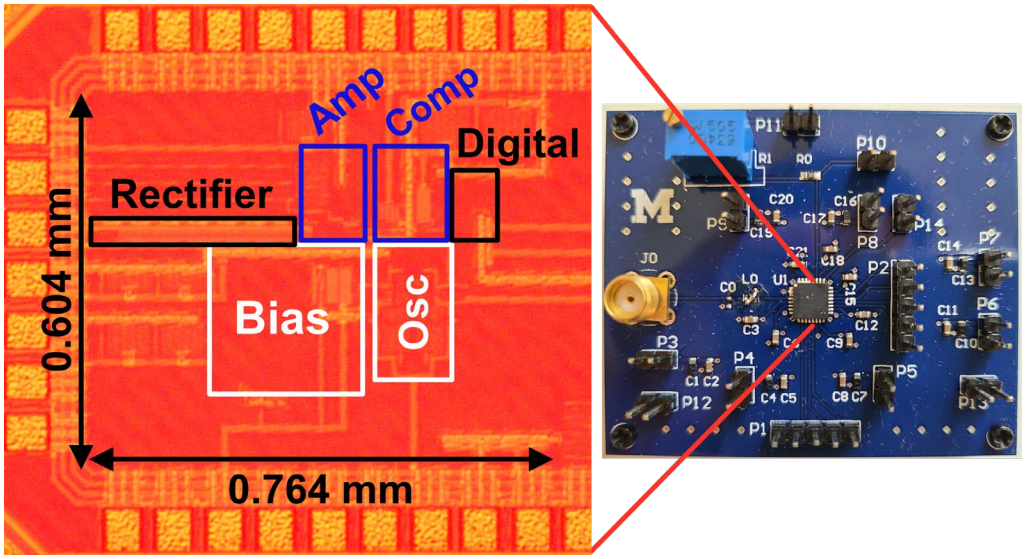


Figure 4.21: Die photo and the packaged chip on the test PCB.

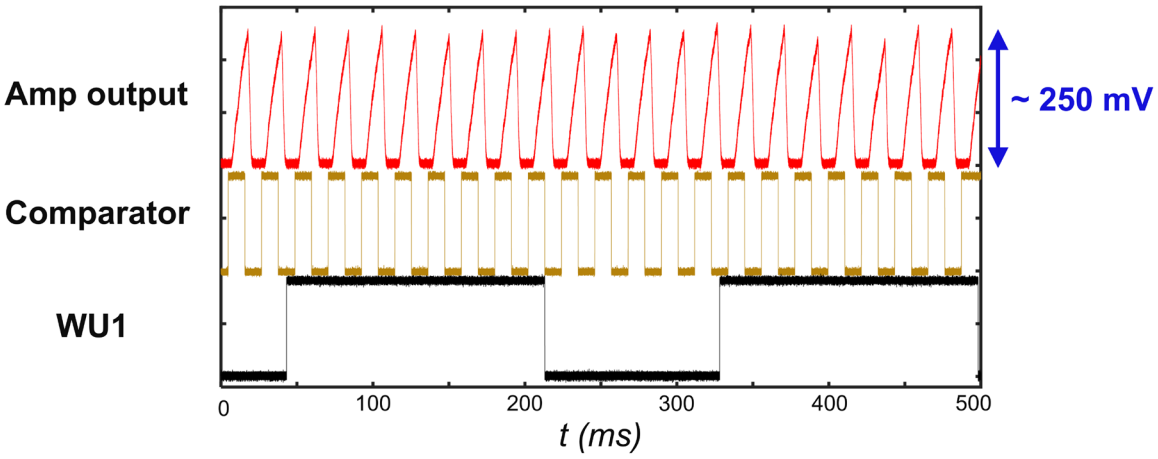


Figure 4.22: Measured buffered amplifier output with -50 dBm (1 mVp) "01" sequence input (w/o the matching network); gain of ~ 50 dB.

First, the WUR is tested without the matching network, and a sensitivity of -52 dBm at 915 MHz and 100 bps data rate is measured at 10^{-3} BER. As the output of the amplifier is pinned out for monitoring, this setup is useful to see the combined gain of the rectifier and the amplifier. When an RF input of -50 dBm (1 mVp) is applied to the chip, the buffered amplifier output changes ~250 mV, indicating a gain of ~50 dB (Figure 4.22). The clock frequency can also be adjusted between 70-500 Hz using the 8 control bits of the oscillator capacitor bank. By adding a 30 nH (Coilcraft

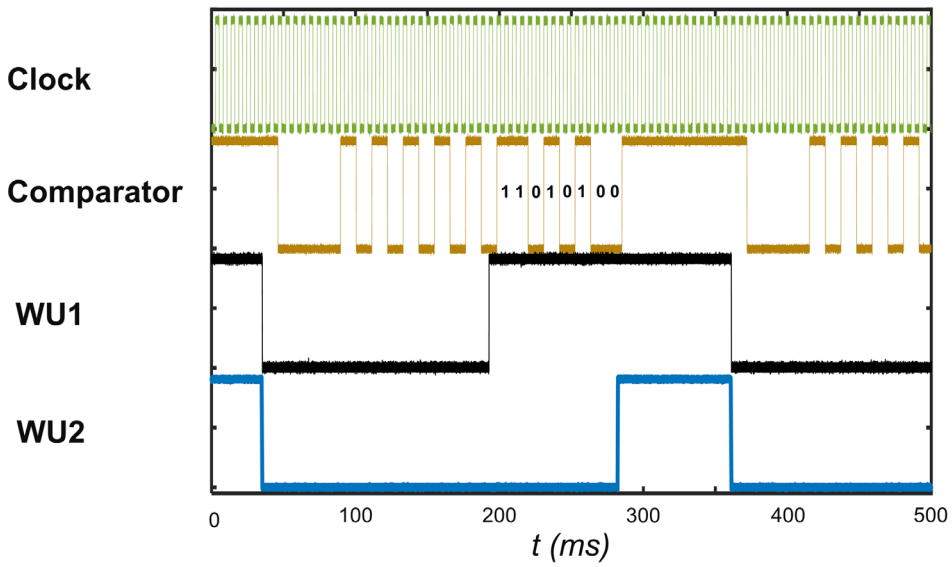


Figure 4.23: Wake-up (WU1 & WU2) detected for "11010100" address with matching network at -61 dBm.

0603DC) inductor and a 6.8 pF capacitor as the matching network, receiver sensitivity goes to -61 dBm. Both of these external components have a size of only 1.6 mm × 0.8 mm. Figure 4.23 shows successful WU1 activation with the "1010101010" sequence, followed by a successful WU2 detection with the "11010100" address that is programmed as the node address.

As discussed in Section 4.3.2, if the early detection block detects more than four falling edges when only half of the expected WU1 sequence is received, it resets its timer and counter, and does not generate a WU1 signal. This feature basically filters transitions with frequencies higher than 2 × (data rate) and prevents them from activating WU1. This is verified in Figure 4.22 by sending the "10" sequence at the frequency of 2 × (data rate) to have 12 back-to-back falling edges. It is clear that while the output of the comparator shows the sequence correctly, no WU1 is generated.

Table 4.2 shows the power breakdown of the WUR, which is measured with continuous data toggling to capture both active and standby mode of the second stage. Comparison with previous passive front-end WURs is summarized in Table 4.3. Considering the discussion in

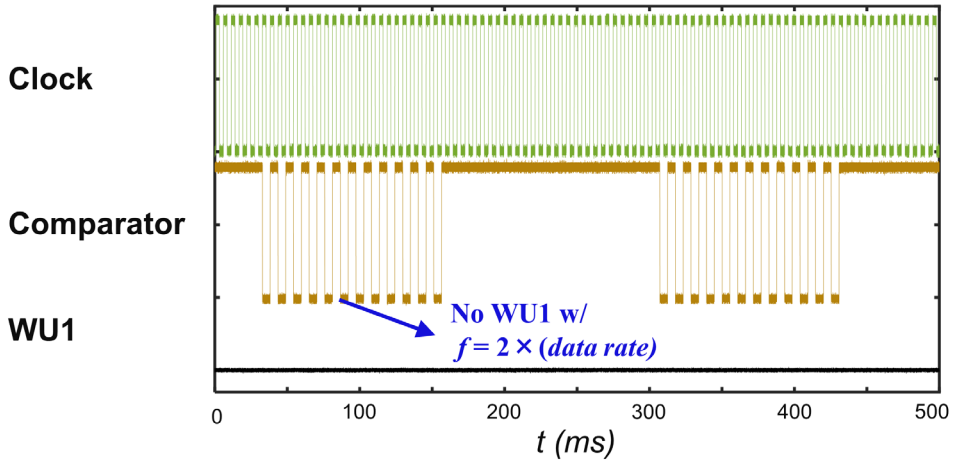


Figure 4.24: By using the timer and counter values, falling edges with frequencies higher than 2× (data rate) are dismissed with no WU1.

	Bias & Oscillator	Amp & Comparator	Digital	Reference Voltage
I (nA)	1.8	3	1.8	0.4
P (nW)	0.72	1.2	0.72	0.16

Table 4.2: System power break down with VDD = 0.4 V

Section 4.3.1, the power, size, and range of the designed WUR is added to the 3D plot for better comparison (Figure 4.25). The work presented here provides a balance between important parameters of a WUR. It has the lowest power consumption of 2.8 nW, while achieving a sensitivity of -61 dBm at 915 MHz. This sensitivity is measured with only two external components <0.015 cm^2 as the matching network, which makes it possible to keep the node size small. Its size including the antenna and the matching network is the second lowest after the 9 GHz WUR [43], while it has 5x more range compared with [43]. With the 100 bps data rate and the 18-bit wake-up signature (10 bit for the first stage sequence and 8-bit address) its latency is <200 ms, which is still lower than the assumed limit of 1 s.

The calculated normalized FOMs of all these works are plotted in Figure 4.26. As explained before, normalized FOM is FOM divided by the minimum FOM of all works, and

	This Work	JSSC18 [41]	JSSC19 [42]	JSSC20 [43]	TMTT20 [46]	RFIC17 [45]	ISSCC16 [44]
Frequency (MHz)	915	113	151/433	9000	2200	2400	2400
Power (nW)	2.8	4.5	7.4	7.3 ⁂	11.3	365	236
Data Rate (bps)	100	300	200	33.3	250	2500	8192
Energy per bit (pJ)	28	15	37	219.2	45.2	146	28.8
External Component	2 ($1.6^{mm} \times 0.8^{mm}$) MN	$2^{cm} \times 4.5^{cm}$ transformer	$1.7^{cm} \times 1.4^{cm}$ MN	0.02 cm^2 transformer	MN *	-	XTAL + MN
Sensitivity (dBm)	-61	-69	-76/-71	-64	-65	-61.5	-56.5
Range (m) * **	150	6	12	30	147	64	55
Bulky Component Gain (dB)	-	25	27/33	-	-	-	-
Special Antenna	-	-	-	4.5 cm^2 ○	-	1.23 cm^2 ◇	-
Osc Freq (Hz)	200	600	200	66.6	1 K	20 K	32 K
Osc Requirements	-	External Resistor	External Tuning Voltage	-	Tuning Supply Voltage	-	External XTAL
Address bits	8	16	8	18	64	32	31
VDD (V)	0.4	0.4	0.6/1	0.4	0.5/1	0.1/0.5/0.8	0.5/1
CMOS Technology	65 nm	180 nm	130 nm	65/180 nm	65 nm	65 nm	65 nm
Die Area (mm^2)	0.462**	6	1.95	14	3.9	1.1**	2.25**
Normalized FOM✦	262	1	1.9	10	149	14	88

* MN: Matching Network ** active area

*** Assuming P_{TX} = 10 dBm, and antenna gain of G_{100M}= -25 dB, G_{900M}= 2.1 dB, $G_{2.4G}$= 4.2 dB, and G_{9G}= 3.6 dB to have almost the same antenna size in all frequency bands

⁂ Excluding temperatue compensation blocks

○ Patch antenna to get 5.5 dB more gain

◇ Rectifier-antenna co-design for passive gain; antenna diameter is D = 1.25 cm

✦ Normalized FOM = FOM / minimum (FOM), where FOM = (Range × DataRate) / (Power × Size)

Table 4.3: Comparison with previous passive front-end WURs

$FOM = (Range \times DataRate)/(Power \times Size)$. The lowest FOM is for [41] due to its 9 cm^2 off-chip transformer and despite its power consumption of 4.5 nW. This work has the highest normalized FOM of 262, which is 1.75x better than the second highest one for the WUR in [46]. Comparing with [46], both designs have similar ranges and sizes, however, in this work power

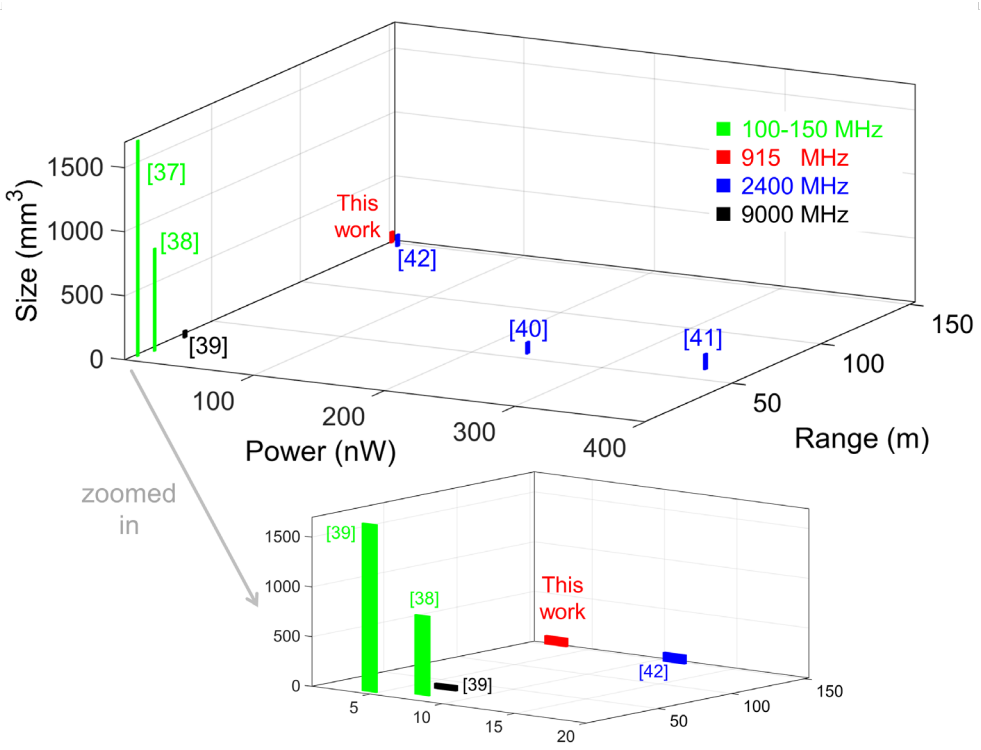


Figure 4.25: Comparison with WURs listed in Table 4.3 in terms of size, power, and range.

consumption is ~ 4x lower, which with the data rate of 100 bps results in 1.6x lower energy per bit. Moreover, setting the desired clock frequency in [46] requires adjusting an external supply voltage from 0.3~0.8 V. The designed chip here, however, operates from a single supply voltage (0.4 V), and adjusting its clock frequency is through programming the capacitor bank of the oscillator, without needing any external voltages or external components.

As a summary, in this chapter, we first compared lifetime of the nodes by using WURs with different power consumptions and by implementing the WOR method on the designed nodes. Results indicate that for 2 hours inertial data recording (acceleration and rotation rates) per week, the nodes' lifetime with the WOR method and a wake-up latency of 10 s is ~110 days. This lifetime is equal to using a 100-µW WUR. A 1-µW WUR increases lifetime to more than 170 days, which is 98% of the maximum possible lifetime with the 200 mAh battery of the nodes. Considering the

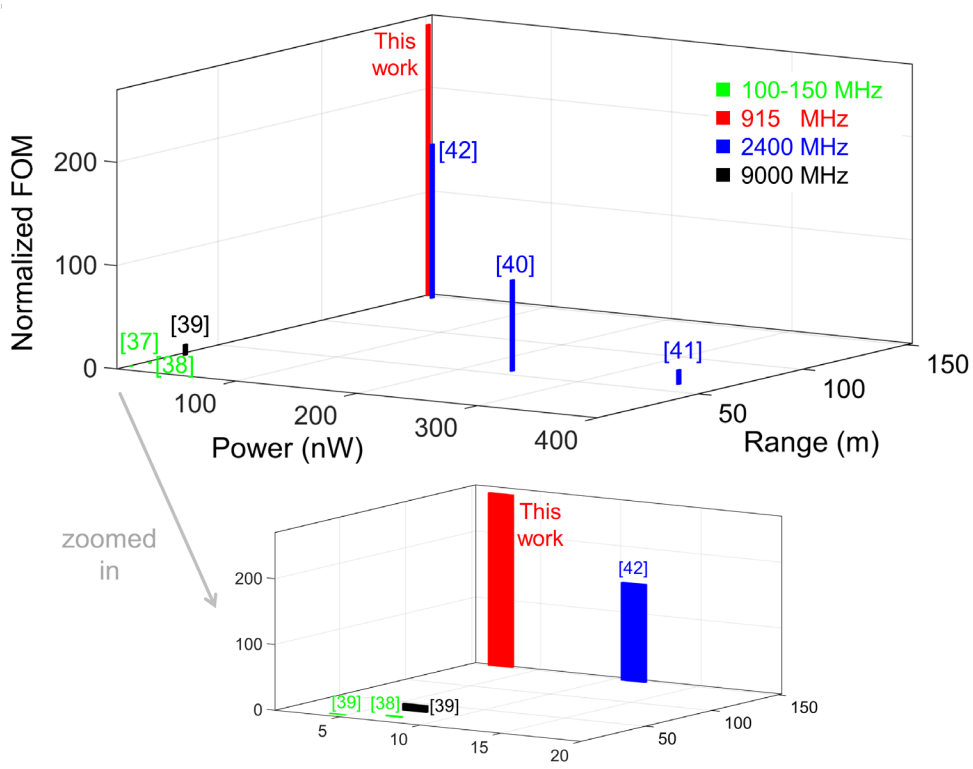


Figure 4.26: Comparison with WURs listed in Table 4.3 in terms of size, power, and normalized FOM.

90 days continuous operation required for monitoring the laminated bearing of the helicopter rotor (Section 1.4), both a WOR with 10 s latency and a sub-µW WUR can achieve a lifetime more than the desired requirements.

For the WUR, our target is not only the HUMS applications discussed, and we are targeting ultra-low-power applications with very low energy consumption during their *active* state. Lower energy consumption means that the nodes either have a lower power sensor or are measuring for shorter period of times (or a combination of both). As a result, our goal for the WUR was power consumptions <100 nW to enable longer lifetime in a broader range of ultra-low-power applications. A WUR with passive front-end architecture and a 2-stage address detection block was designed and fabricated in TSMC 65 nm LP CMOS technology. It consumes 2.8 nW from a single supply voltage of 0.4 V and has a sensitivity of -61 dBm at 915 MHz based on measurements

results. This sensitivity is achieved by using only two small (<1.5 mm^2) off-chip components as the matching network and translates to a range of ~150 m (estimated by using the Friis transmission formula) with commercially available antennas. The 2-stage wake-up architecture also helps reduce power consumption of the node by reducing probability of false wake-ups.

In the next chapter, rotor bench tests and an in-flight test of the developed system including nodes and the master are discussed. As the WUR was not completed at the time of preparation for the in-flight test, the WOR method is used with latencies of more than 10 s.

Chapter 5 **Rotor Tests**

In Chapter 2, we presented the design of a wireless sensing system consisting of wireless sensor nodes, and a Master (baste station) for data acquisition and synchronizing the nodes. Different synchronization algorithms were proposed in Chapters Chapter 2 and Chapter 3, which were implemented and tested on the designed wireless system. Those tests were mainly designed to characterize the synchronization error of the proposed algorithms and were conducted in a laboratory with I/O pins of the nodes connected to an oscilloscope to monitor the error between them. In this chapter we will be using a rotor test bench (Section 5.1) on which our system is installed, and will collect data. The test bench has wired and optical sensors that allows us to compare the acquired data from different systems. Only the nodes are attached to the blades of the test bench, and the Master is connected to a laptop. The result of this test is reported in Section 5.2. Then, with an in-flight test being the target, the hardware and software requirements of an in-flight test are discussed in Section 5.3. Since the actual date of the in-flight test is unknown and is dependent on arrangements between our project sponsor and an aviation company, we prepared all the components, including housings, software and hardware modifications, and tested the system on the ground on the rotor bench test. In this test, a Master board is added for user interface, and another Master board is installed for data acquisition on the rotor. The user interface board is portable and is only used by the operator on ground and is not required for in-flight recordings. Details and results of this test are discussed in Section 5.4.

5.1 Rotor Test Bench

Figure 5.1 shows the laminated thrust bearing, and its position relative to the rotor and blades in a graphic illustration and a mock rotor setup. The bearing is linked to the rotor axis from one end and to the blade from the other end. The blades are helicopter are heavy and they create a lot of centrifugal loads on the rotor and blade attachments [52]. Even for a small two to four passenger helicopter the centrifugal loads can be from 6 to 12 tons and can go as high as 40 tons for large helicopters [52]. The thrust bearings must support all that centrifugal force, and also allow the blades to have the desired angles for moving the helicopter. To monitor the health of this part, HUMS algorithms rely on dynamic inertial data acquired from different parts of the blade and used

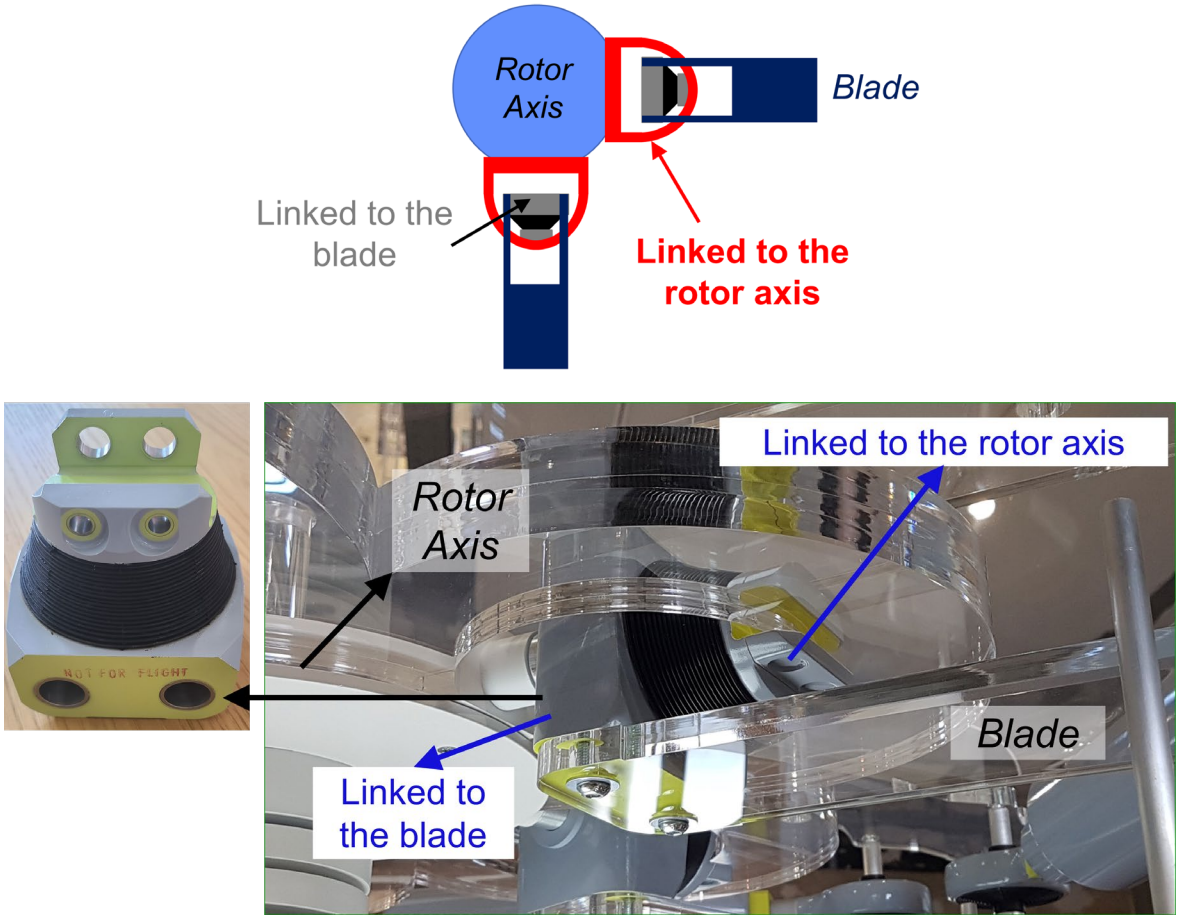


Figure 5.1: Laminated thrust bearing in a mock rotor, and graphic illustration of its connections.

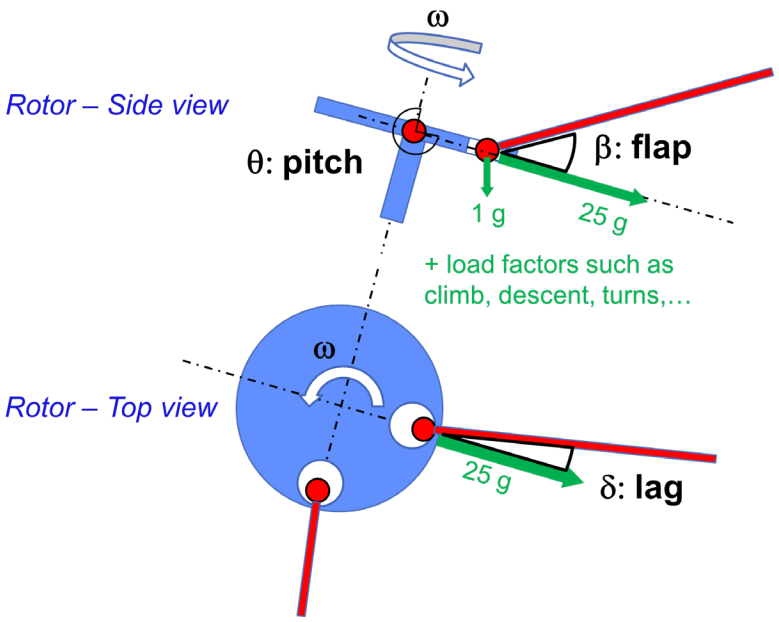


Figure 5.2: Three angles used to characterize movement of rotor blades: pitch, flap, and lag.

to extract three angles called pitch, flap, and lag, which are shown in Figure 5.2. The time duration that these angles stay over a certain threshold can be used to estimate the lifetime of the part. As discussed before, in order to extract these angles, inertial data must be collected synchronously. The maximum acceptable time synchronization error depends on the angle reconstruction methods and fault-detection algorithms, which are implemented by our sponsor. For these tests, the target is a synchronization error of 30 µs. To test the system developed in this work, a smaller rotor and blades fixed in a metallic structure are used in our sponsor's laboratory in France. As shown in Figure 5.3, this setup is also equipped with wired IMUs (MTi-1 IMU from Xsens [53]) and optical sensors that act as a reference for measuring the angles. Different speeds and patterns of rotation can be set for the rotor through its user interface. We refer to this setup as the rotor test bench.

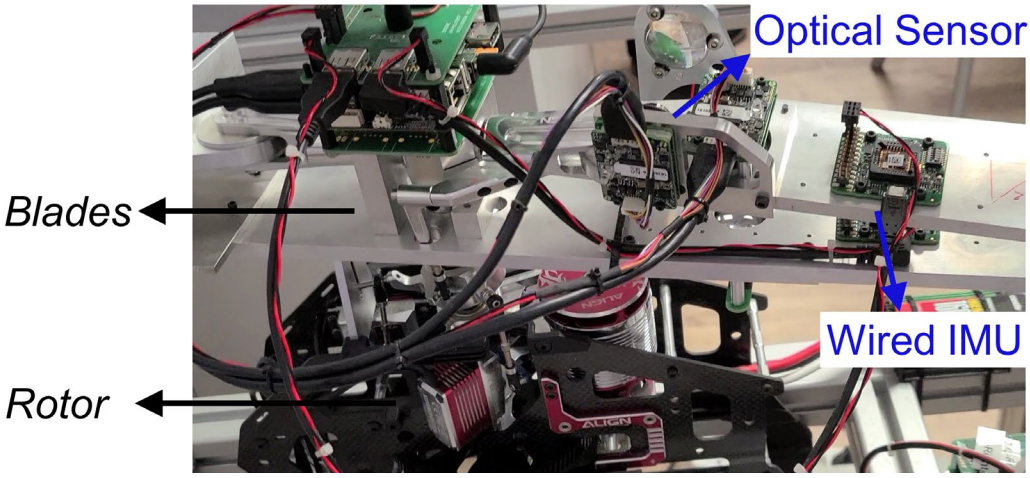


Figure 5.3: Rotor test bench: smaller rotor with wired IMU and optical sensors.

5.2 Bench Tests

In the first set of tests, two wireless sensor nodes and a Master board connected to a laptop are used to validate the feasibility of using wireless nodes for this application. The nodes are the ones with nRF52 SoC and ICM-20601[35] as the IMU (explained in Section 2.2 and shown in Figure 2.2 (b)) and for this test are installed directly on the blade besides the wired IMUs (Figure 5.3). The Master is also the nRF52-DK as discussed in Section 2.2. *BlueSync* with LR2-8(05) method is used as the synchronization protocol. Synchronized data with a sampling rate of 100 Hz were successfully collected by the Master for a test duration of around 200 s. The extracted angles based on the wireless system (i.e., nodes developed in this work), the wired IMUs, and the optical sensors are plotted in Figure 5.4. The extraction of the angles is performed as part of a fault-detection algorithm developed by our sponsor, and an accuracy of 1° compared with the optical reference is considered acceptable. Table 5.1 shows the average absolute error between the optical data as the reference, and the extracted angles of the wired and wireless systems. This average absolute error between A and B with n samples is calculated by $\frac{\sum_{i=1}^{n}|A_i-B_i|}{n}$. For both wired and

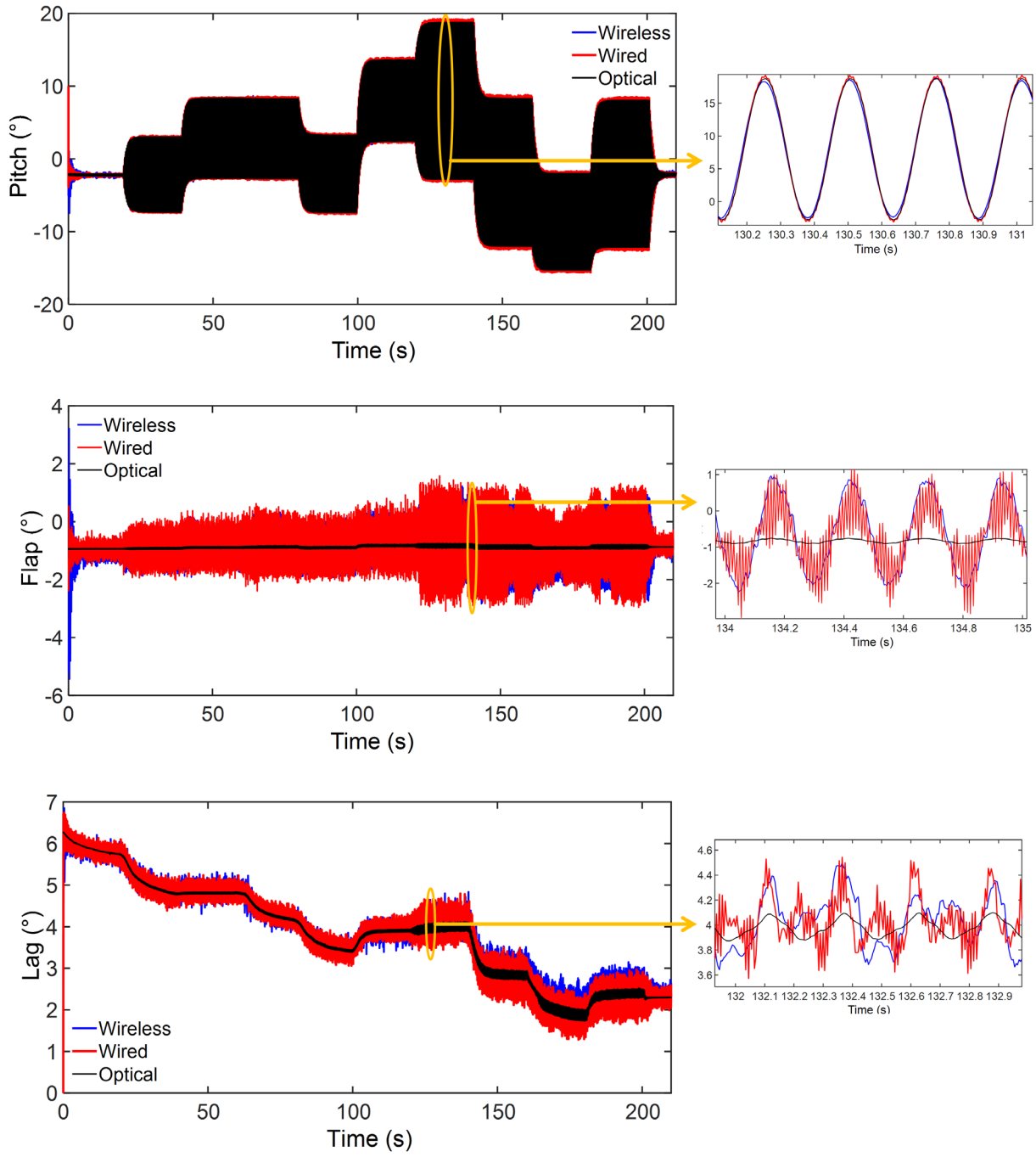


Figure 5.4: Extracted angles from the three systems on the rotor test bench: optical, wired, and wireless.

wireless data, results show errors smaller than 1° for the flap and smaller than 0.4° for pitch and lag angles, which are within the accuracy required by the fault-detection algorithms, and confirms the usefulness of the wireless system.

	Pitch		Flap		Lag	
	Wired	Wireless	Wired	Wireless	Wired	Wireless
Optical	0.11°	0.33°	0.41°	0.54°	0.12°	0.18°

Table 5.1: Average absolute error between the extracted angles (pitch, flap, and lag) of the optical sensors as the reference and the wired /wireless systems (bench tests)

It is worth noting that the current rotor test bench does not have any means of changing the flap angle, and we expect to have *flap* = 0°. The flap angle of around -1° measured by the optical sensor and seen throughout the test is due to the installation of the bench and its metallic frames. As shown in Figure 5.4 and Table 5.1, both wired and wireless systems have also larger errors for the flap angle, which is acceptable considering the small value of the angle itself, and the fact that their errors are still below 1° on average. When comparing the results of the wired and wireless systems for all the three angles, we noticed some differences between them. These differences can be explained by considering two important points. First, the mechanical position, i.e., the point where the sensors are installed on the blade, affects the final value of the extracted angles. Because the wired and the wireless nodes are placed at different positions from the rotor blasé axis, any error in measuring their exact location can show up as error in the extracted angles. Moreover, due to the different positions of the sensors, they will be measuring slightly different acceleration and rotation rates. Second, the IMUs used in the wired and wireless system are not the same and have different specifications. The wired IMU is MTi-1 from Xsense with ±16 g and ±2000 °/s full scale ranges [53], respectively for the accelerometers and gyroscopes, whereas the wireless nodes use ICM-20601 from Invensense with full scale ranges of ±32 g and ±4000 °/s [35]. Table 5.2 provides a comparison of these two IMUs. Considering the above two points and the synchronization error measurements presented in Section 2.6, the differences between the wired and wireless results are sufficiently small and not caused by the wireless nodes and their synchronization algorithms.

	Accelerometer		Gyroscope	
	MT-i 1	ICM-20601	MT-i 1	ICM-20601
Stadard Full Range	±16 g	±32 g	±2000 °/s	±4000 °/s
Nonlinearity	0.5 %	0.5 %	0.1 %	0.3 %
Noise Density	70 µg/√Hz	390 µg/√Hz	0.013 °/s/√Hz	0.003 °/s/√Hz
3 dB Bandwidth	230 Hz	5~218 Hz	230 Hz	5~250 Hz

Table 5.2: Comparison of wired IMU (MT-i 1) and IMU of wireless nodes (ICM-20601)

As mentioned, these wireless nodes are to be eventually tested in an in-flight test with nodes installed on actual helicopter rotor to measure different flight conditions in a two-hour flight. Because the exact date of receiving approval for the flight and possible access to the helicopter before the flight is unknown, this thesis does not have access to this data. In preparation for these in-flight tests, we have prepared a complete system for the in-flight test, and for evaluation on the ground using the rotor test bench. In the following sections we describe the requirements of the in-flight test, how they have been addressed, and the results of the rotor bench tests.

5.3 In-Flight Helicopter Test

5.3.1 *Requirements*

Due to regulations, an IP67 housing must be designed for all the nodes and boards that will be installed on the helicopter. The time between the installation of the system and the actual 2-hour flight is unknown and might be days or weeks. Also, the nodes and the Master board(s) in charge of synchronization and data acquisition are installed in hard-to-reach areas and will have to be turned ON at the time of installation. Therefore, a third User Interface (UI) board and wake-up feature are needed to initiate the test before the flight. Figure 5.5 shows the physical placement of the boards relative to each other. Compared with the bench tests, the following changes are required:

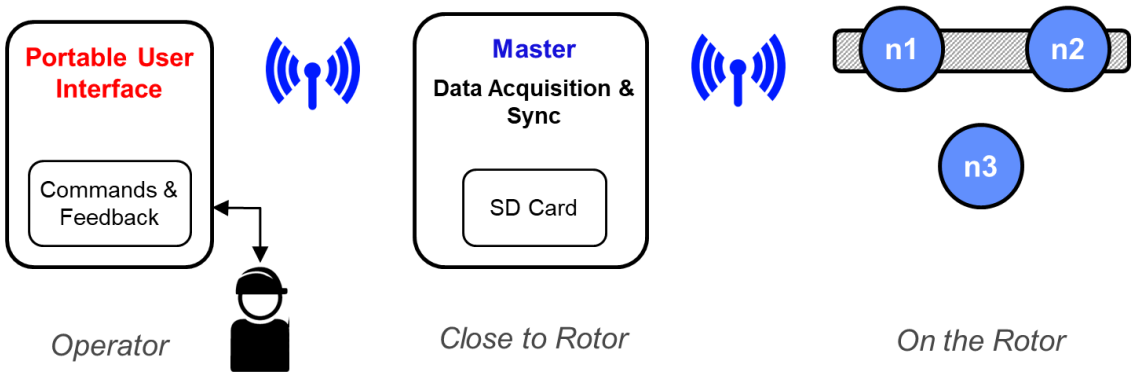


Figure 5.5: Physical placement of the boards for the in-flight test.

- A portable UI board that will be used by an operator on the ground to wake-up the Master(s) and start/stop the test. It will also provide feedback about the status of the system to the operator.
- The Master(s) need to store data on a SD card and act as a bridge between the UI board and the nodes. After receiving the wake-up command from the UI board, they send another wake-up signal to the nodes and wait for a confirmation of wake-up from all the nodes. A similar process is repeated for synchronization, and the status of the system at each stage is reported back to the portable UI board.
- Wake-up is implemented by duty cycling the nRF52 receiver with latencies of >10 s.
- The operator does not have any control over the system other than starting and stopping the recordings. Therefore, a high-level module called test session management must be added to the nodes and Master(s) so that the system can recover from any potential error, especially during the recordings.

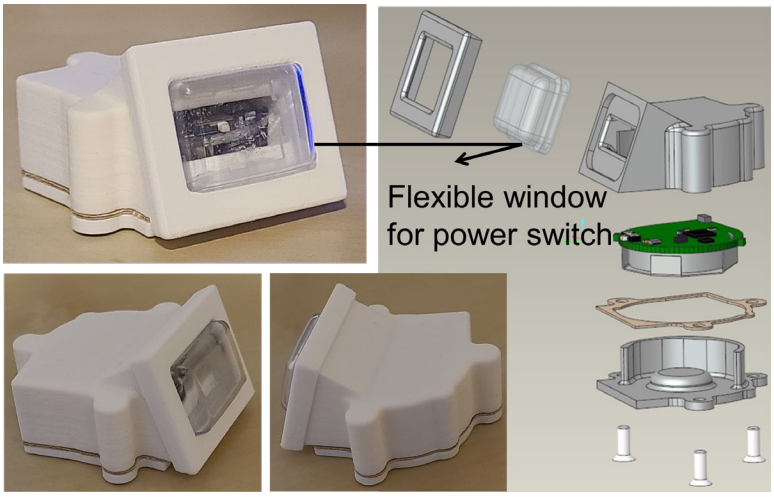


Figure 5.6: 3D printed housing of the nodes with a size of 4×4×2 cm^3, and weight of 29 g including the battery and all the screws.

5.3.2 *Hardware and Housing*

The nodes do not need any additional hardware and just a housing is designed for them. Figure 5.6 shows the 3D printed housing for the nodes, which weighs 29 g with a 200 mAh battery and has a flexible membrane window to enable access to the power switch.

The nRF52-DK is used for both the UI board and the Master(s). The UI board does not need any special housing and power source, and it can be used with a laptop. The main additional hardware is for the Master, which include a rechargeable 1000 mAh Li-ion battery, voltage regulator, an IP67 switch for turning the board ON after installation, and a SD card shield. The battery and voltage regulator, and the wires connecting them to the IP67 switch are placed at the bottom of the housing and below the nRF52-DK (Figure 5.7). The SD card shield is 35 mm × 55 mm and connects to the top side of the PCB. In the current rotor test bench, one of the nodes is placed on top of the Master. Therefore, one node housing is attached to the top cover. The complete Master, including all PCBs and batteries, weighs 156 g.

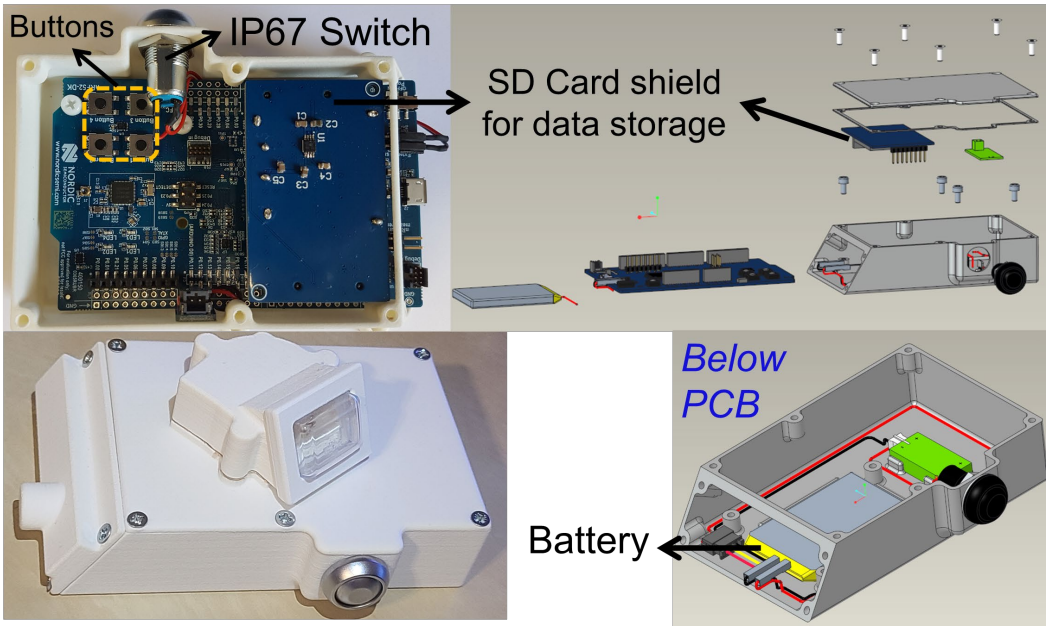


Figure 5.7: 3D printed housing of the Master with a size of 12×8.5×2.5 cm^3, and weight of 156 g including the battery, all PCBs, and screws. Battery and voltage regulator are placed below the PCB.

5.3.3 *Software*

After installation, both Master(s) and the nodes are turned ON. However, they enter a low-power *sleep* mode, where they turn on their receiver for a window of 250 ms every 30 s. This is the WOR method discussed in Section 4.1. On the day of the in-flight test (which can be weeks apart from the installation), the user can send the wake-up command to the Master by pressing Button 1 on the UI board (buttons are shown in Figure 5.7 as the UI board and the Master use the same hardware). The Master then relays this command to the nodes and waits for their confirmation. The Master continues advertising the wake-up command until confirmation is received from all the nodes. Then, it advertises the wake-up status, which can be received by the UI board and displayed to the user.

At this point, the assumption is that the test might be cancelled or might be carried out in a few hours. Therefore, the Master and the nodes are still put in the *sleep* mode, but the frequency

of scan is increased to every 10 s to have lower latencies in response to commands. The user can send a cancel code at any time to reset the Master and the nodes. Whenever it is confirmed that the flight will happen, the user should start the synchronization timeslot by pushing Button 2 (Figure 5.7) on the UI board. From this stage, everything is automated, and the UI board is just used to show the system status to the user.

Whenever the Master receives the synchronization command, it informs the nodes of the time that it will start sending synchronization packets by including an incrementing counter value in the payload. By having a predefined duration for this stage, the nodes can estimate the start of the synchronization timeslot by using the received counter value. After the synchronization packets are sent, the Master scans for packets from the nodes that indicate the result of synchronization for that node. This result is advertised by the Master for both the UI board and the nodes. With successful synchronization, the nodes start recording and sending data and the Master saves the received data on the SD card. If synchronization fails, the Master and the nodes repeat the above process. Note that no user interaction is needed, and the system automatically recovers from any missed wake-up, failed synchronization, and even lost connection during recording.

5.4 Results with In-Flight Test Requirements

Figure 5.8 shows the placement of the Master and the nodes on the rotor test bench. The same sequence that is expected on the day of the in-flight test was used, and the experiment was repeated multiple times. No packet loss was detected for the wireless system in any of the tests. It was verified that when some synchronization packets were not received by one of the nodes the UI board showed the correct feedback, and the Master and the nodes repeated the synchronization routine. The rotor was controlled with a 200-second-long pattern. Three extracted angles for the three used systems (optical, wired, and wireless) are plotted in Figure 5.9. Table 5.3 shows the

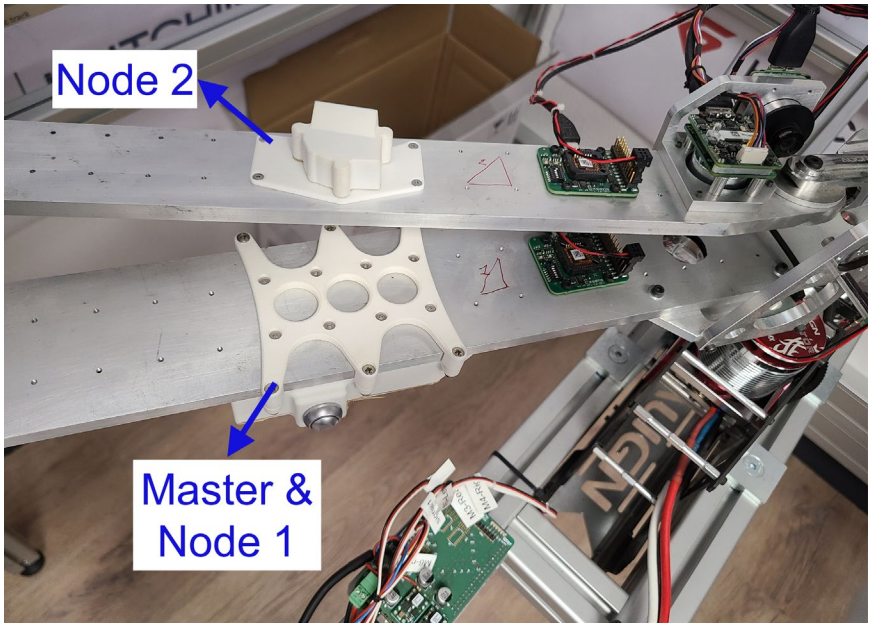


Figure 5.8: Rotor test bench used to verify the in-flight hardware and software.

	Pitch		Flap		Lag	
	Wired	Wireless	Wired	Wireless	Wired	Wireless
Optical	0.20°	0.34°	0.33	0.50	0.22	0.25°

Table 5.3: Average absolute error between the extracted angles (pitch, flap, and lag) of the optical sensors as the reference and the wired /wireless systems (bench tests with in-flight test requirements)

average absolute error between the angles obtained from the optical sensors and the ones extracted from the wired and wireless data.

In both wired and wireless system, the average absolute error for pitch and lag is smaller than 0.4°, whereas the flap error is smaller than 0.6°. All errors are within the acceptable range of the HUMS fault-detection algorithms (< 1°). Figure 5.10 and Figure 5.11 show the moving average of the absolute error with a window size of 20 s for the wired and wireless system, respectively. When calculating the moving average of an array A, each mean is calculated over a sliding window of length k across neighboring elements of A. The window is centered about the element in the current position; result is an array with the same size as A. Here array A is the absolute error with

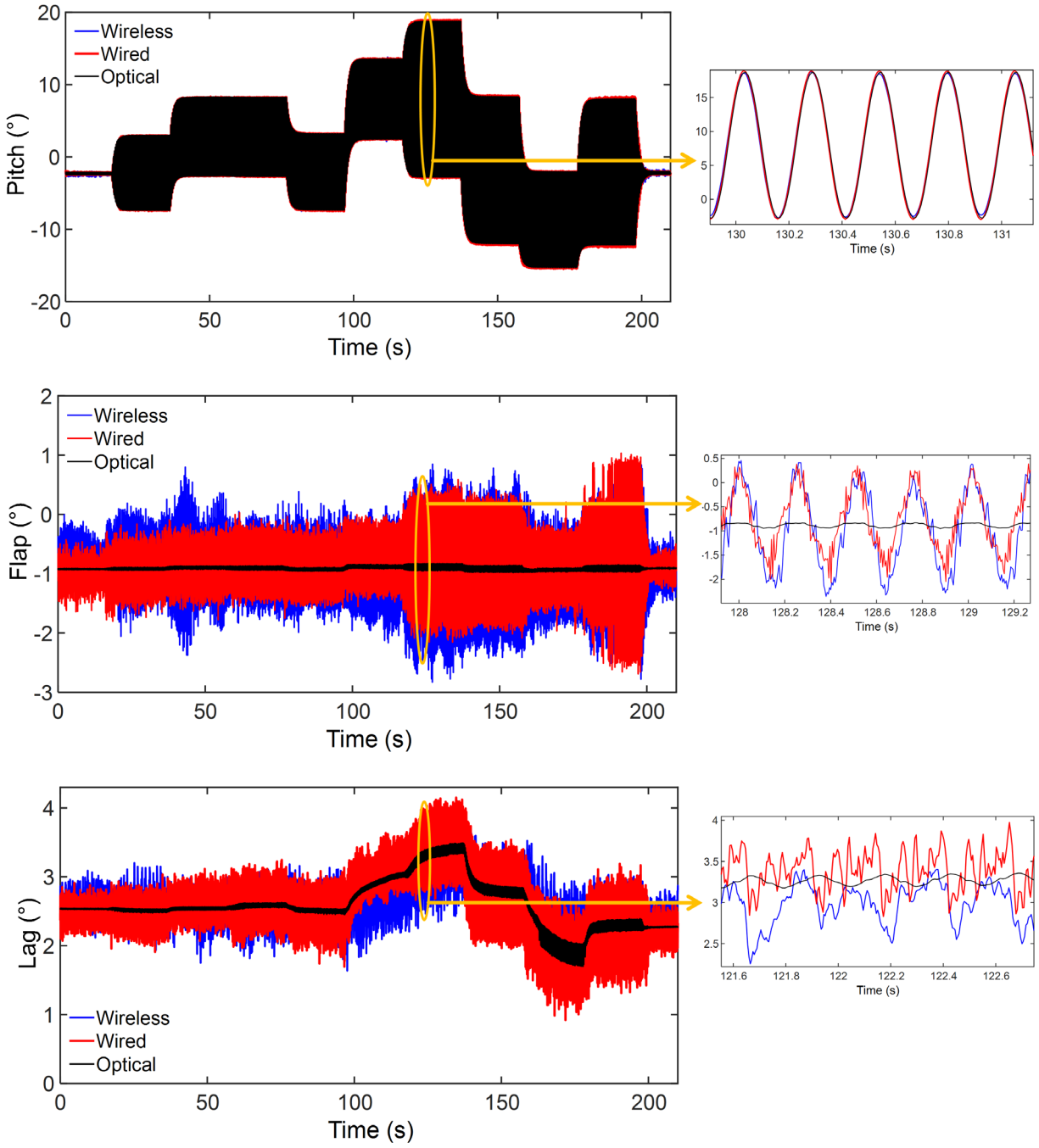


Figure 5.9: Extracted angles from the three systems on the rotor test bench: optical, wired, and wireless (bench tests with in-flight test requirements).

reference to angles from the optical data, window size is 20 s, and the size of A is 200 s (test duration). The difference between the wired and wireless results is explained by the same points

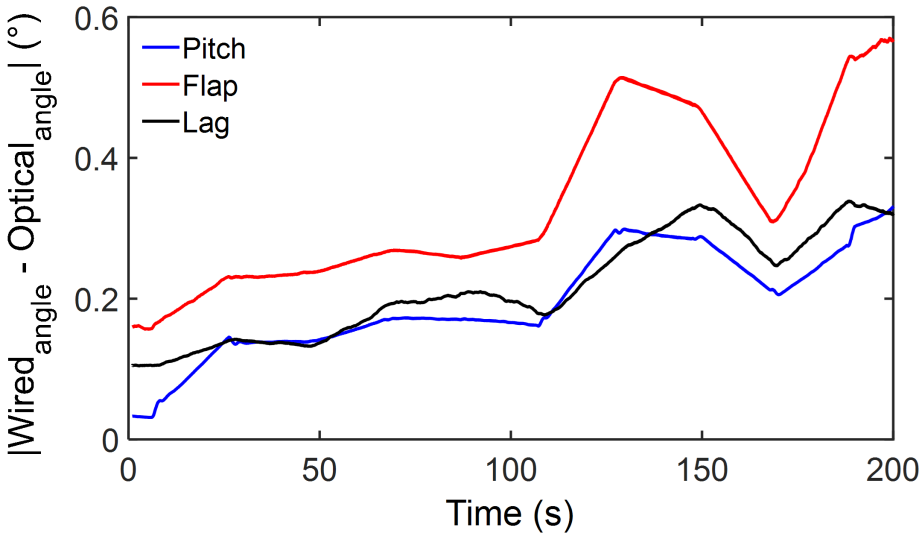


Figure 5.10: Moving average of the absolute error between the extracted angles of the wired system and the optical sensors as reference, with averaging window size of 20 s.

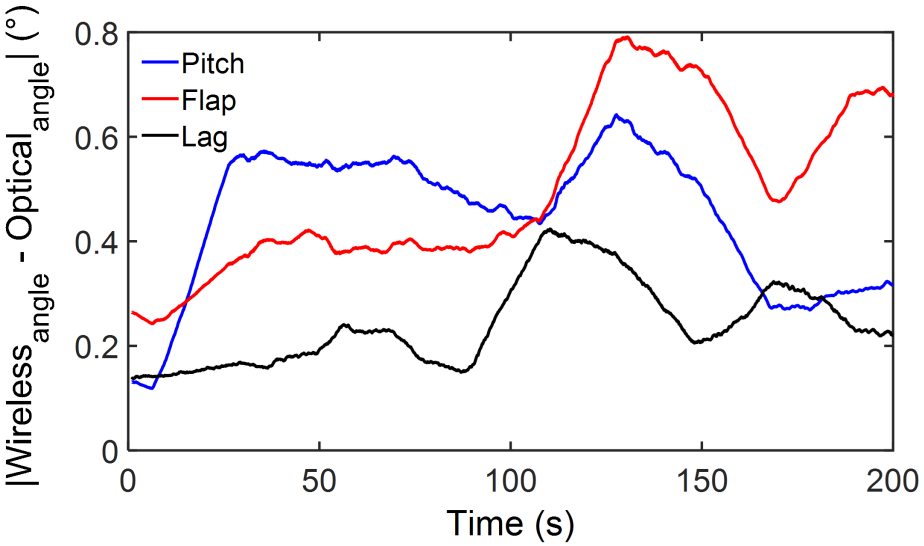


Figure 5.11: Moving average of the absolute error between the extracted angles of the wireless nodes and the optical sensors as reference, with averaging window size of 20 s.

we presented in Section 5.2. Specifically, one of the nodes in this setup is placed on top of the Master box and is elevated from the blade.

As a summary, these results validate that the designed system along with the implemented low-error synchronization protocols enable wirelessly monitoring the bearing of the helicopter

rotor. As discussed, wireless operation for parts like the bearings and hard to reach areas are essential, but it comes with the challenges of time synchronization and lifetime of the system. The proposed system provides both requirements of synchronization error <30 μs and continuous operation for 110 days (target was 90 days, refer to Section 4.1 for more details). Low-error synchronization is achieved without frequent resynchronization (i.e., use of radio) and/or sampling with rates much higher than the required sampling rate, which all of them result in higher power consumption and a shorter lifetime. At the time of preparing for these tests, the WUR chip was not completed, and the WOR method was implemented. By integrating the WUR into the nodes, as explained in Section 4.1, the lifetime of the node is estimated to be increased by an additional 60 days, making it possible to use the system for ~ 6 months without any battery change.

Chapter 6 **Conclusion and Future Works**

In this work, battery powered and BLE-enabled nodes with 6-axis inertial sensors have been custom-designed to monitor the condition of the laminated bearings of rotor blades as part of a helicopter HUMS. Nodes need to be placed on rotating blades to measure their dynamic motion and report the data to a base station. Synchronized measurements is one of the key requirements of fault-detection algorithms that use these data to extract the three angles related to the blades (i.e., pitch, flap, and lag). Therefore, while wireless connectivity is essential for these systems, it comes with the challenges of node lifetime and time synchronization. A synchronization error of <30 µs and continuous operation of 90 days with 1~2 hours recording per week are considered as the target requirements.

Different synchronization protocols were proposed including *BlueSync*, *Discrete Adjustments*, and *AdaptSync*, that can provide the required synchronization in a low-power fashion. *BlueSync* is a BLE compatible synchronization protocol that archives average synchronization error of <1 µs per 60 s without resynchronization and has the lowest reported error for BLE. The two methods presented in *Discrete Adjustments* can improve the energy efficiency of calculations by up to 15x and extend the lifetime. In long recording sessions of >1 min and as long as one hour, *AdaptSync* enables reducing the error by using the previous timing information and without the need to use resynchronization packets for resetting the error. All of these techniques were experimentally tested and verified with the developed sensor nodes.

A system composed of two sensor nodes and a base station was custom-designed, built, and installed on a rotor test bench. Real-time wireless data was recorded during different patterns

of blade rotations. The test bench also has a wired recording system and optical sensors that act as the reference for the extracted angles. The angles extracted with our wireless system were compared with those acquired from the wired system and optical sensors. Results show angles errors of < 1°, compared to the optical reference, which is well within the limit for the fault-detection algorithms of this application, and validate that the developed wireless system can be reliably used to monitor the laminated bearings.

To use the system in an in-flight test, additional hardware, software, and housing for the system were designed. Since the actual date of the flight is unknown, the prepared system was tested on the ground and on the rotor test bench. The WOR method with latencies >10 s was used as the wake-up mechanism to provide ~ 110 days of continuous operation. Different stages of the system, including wake-up, synchronization, lossless data acquisition, and auto recovery from any errors during these stages without user interaction were successfully tested.

To further improve the lifetime of the nodes, a 915 MHz WUR was also designed and fabricated in TSMC 65nm CMOS technology. The WUR is based on a passive front-end architecture, consumes 2.8 nW at 0.4 V, has a sensitivity of -61 dBm and with only two external components as the matching network and commercially available chip antennas can achieve a range of ~ 150 m. It also has a 2-stage wake-up architecture that reduces the power consumption of the node by reducing the probability of false wake-ups during the *sleep* state. By using this WUR in the nodes, their lifetime can be increased by 60 days, which enable monitoring the bearings for ~ 6 months without needing any battery change. As the WUR was not ready at the time of the preparation for the in-flight test, it was not included in that design.

For future works in the synchronization part, we believe that the proposed timestamping method in *BlueSync* for removing uncertainty in the transmitter timestamps is a promising

technique that can be used with other standard wireless protocols. By using this method, the only delay between timestamping in the transmitter and receiver is the propagation delay. As discussed before, with synchronization protocols we are just trying to reduce the speed of timing-error accumulation, and error increase with time is inevitable. Therefore, especially for long recording sessions or when a very low-error synchronization is required, we must have a mechanism to reset the error. With the proposed *AdaptSync* protocol we showed that to increase lifetime of the nodes this error reset can be done without using radio packets. The protocol was tested with up to four nodes and 10 minutes recordings, and error corrections for the nodes were calculated manually. Automating the calculation of the error corrections, and then increasing the number of nodes and recording times can be the next steps.

Moreover, it is worth noting that just like many other synchronization protocols, in all the protocols discussed here we did not compensate for any temperature difference between the nodes. The assumption is that there is no significant temperature change between the nodes, which is valid in many applications where the nodes are placed relatively close to each other (e.g., monitoring the blades of the helicopter). However, for applications where large temperature difference across nodes is expected, one option is to use a temperature-compensated crystal, if power requirements and the size of the system permits. Another option is to use either the temperature sensor embedded in most wireless SoCs or add a temperature sensor to the nodes, and then use it in an additional ticks adjustment block based on the temperature change. For this purpose, the temperature at the time of synchronization timeslot should be recorded, and then whenever this temperature changes above a certain threshold, the additional ticks adjustment operation should start to work. Clearly, as temperature typically does not change very fast, there is no need to read the temperature very frequently. The frequency of reading and the change threshold are application dependent.

As for the WUR, future works can involve making it more robust for real-world applications. This can be done by adding an automatic threshold and offset control loop, and temperature compensation. For the system-level part, testing the system in an actual helicopter and during a flight is the next step. Based on the results of this test, modifications might be needed to the nodes. Integrating the WUR and potentially wireless charging capability or a type of energy harvester into the nodes can help in increasing lifetime of the nodes.

Bibliography

[1] A. A. Hood, “Title of dissertation: Fault Detection on a Full-Scale OH-58 A/C Helicopter Transmission,” 2010.

[2] US Joint Helicopter Safety Implementation Team, “Health and Usage Monitoring Systems Toolkit,” 2013. [Online]. Available: https://ihsf.aero/Toolkits/Toolkit_HUMS.pdf

[3] J. P. Lynch and K. J. Loh, “A summary review of wireless sensors and sensor networks for structural health monitoring,” *Shock Vib. Dig.*, vol. 38, no. 2, pp. 91–130, 2006.

[4] A. Sabato, “Pedestrian bridge vibration monitoring using a wireless MEMS accelerometer board,” in *2015 IEEE 19th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, 2015, pp. 437–442. doi: 10.1109/CSCWD.2015.7230999.

[5] “Honeywell Health and Usage Monitoring Systems Infographic.” https://aerospace.honeywell.com/content/dam/aerobt/en/documents/learn/products/health-and-usage-monitoring/infographics/HUMS-ig.pdf (accessed Jul. 22, 2022).

[6] J. Beran, R. Kalmar, and A. Vechart, “Wireless Sensor Units for HUMS Data Acquisition,” *9th DSTO Int. Conf. Heal. Usage Monit.*, pp. 1–6, 2015, [Online]. Available: http://bit.ly/2tHA9Vj

[7] S. W. Arms *et al.*, “Synchronized wireless sensor network for landing gear loads monitoring,” *Proc. 6th Eur. Work. - Struct. Heal. Monit. 2012, EWSHM 2012*, vol. 1, pp. 439–445, 2012.

[8] S. W. Arms, C. P. Townsend, J. H. Galbreath, D. L. Churchill, and N. Phan, “Synchronized system for wireless sensing, RFID, data aggregation, & remote reporting,” in *Annual Forum Proceedings - AHS International*, 2009, vol. 2, pp. 939–945.

[9] S. DiStasi, C. P. Townsend, J. Galbreath, and S. W. Arms, “Scalable, synchronized, energy harvesting wireless sensor networks,” in *2010 Prognostics and System Health Management Conference*, 2010, pp. 1–5. doi: 10.1109/PHM.2010.5413550.

[10] S. J. Distasi, C. P. Townsend, J. R. Bessette, and P. Richard, “Scalable , Synchronized Network of Lossless Wireless Sensors for Rotorcraft Monitoring,” *15th Aust. Int. Aerosp. Congr.*, 2013.

[11] A. Sanchez Ramirez, K. Das, R. Loendersloot, T. Tinga, and P. Havinga, “Wireless Sensor Network for Helicopter Rotor Blade Vibration Monitoring: Requirements Definition and Technological Aspects,” *Key Eng. Mater.*, vol. 569–570, pp. 775–782, 2013, doi: 10.4028/www.scientific.net/KEM.569-570.775.

[12] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi, “The Flooding Time Synchronization Protocol,” in *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, 2004, pp. 39–49. doi: 10.1145/1031495.1031501.

[13] A. S. Ramirez, R. Loendersloot, T. Tinga, F. L. M. Dos Santos, and B. Peeters, “Helicopter rotor blade monitoring using autonomous Wireless Sensor Network,” in *10th International Conference on Condition Monitoring and Machinery Failure Prevention Technologies 2013, CM 2013 and MFPT 2013*, 2013, vol. 1, pp. 504–514.

[14] S. Bosch, M. Shoaib, S. Geerlings, L. Buit, N. Meratnia, and P. Havinga, “Analysis of Indoor Rowing Motion Using Wearable Inertial Sensors,” in *Proceedings of the 10th EAI International Conference on Body Area Networks*, 2015, pp. 233–239. doi: 10.4108/eai.28-9-2015.2261465.

[15] J. Elson, L. Girod, and D. Estrin, “Fine-Grained Network Time Synchronization Using Reference Broadcasts,” *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 147–163, Dec. 2003, doi: 10.1145/844128.844143.

[16] S. Ganeriwal, R. Kumar, and M. B. Srivastava, “Timing-sync protocol for sensor networks,” in *SenSys’03: Proceedings of the First International Conference on Embedded Networked Sensor Systems*, 2003, pp. 138–149. doi: 10.1145/958507.958508.

[17] T. Schmid, P. Dutta, and M. B. Srivastava, “High-resolution, low-power time synchronization an oxymoron no more,” in *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks, IPSN ’10*, 2010, pp. 151–161. doi: 10.1145/1791212.1791231.

[18] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh, “Efficient network flooding and time synchronization with Glossy,” in *Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks, IPSN’11*, 2011, pp. 73–84.

[19] R. Lim, B. Maag, and L. Thiele, “Time-of-Flight Aware Time Synchronization for Wireless Embedded Systems,” in *Proceedings of the 2016 International Conference on Embedded Wireless Systems and Networks*, 2016, pp. 149–158.

[20] S. Sridhar, P. Misra, and J. Warrior, “CheepSync: A time synchronization service for resource constrained Bluetooth Low Energy advertisers,” in *IPSN 2015 - Proceedings of the 14th International Symposium on Information Processing in Sensor Networks (Part of CPS Week)*, Apr. 2015, pp. 364–365. doi: 10.1145/2737095.2742925.

[21] S. Sridhar, P. Misra, G. S. Gill, and J. Warrior, “Cheepsync: A time synchronization service for resource constrained bluetooth le advertisers,” *IEEE Commun. Mag.*, vol. 54, no. 1, pp. 136–143, Jan. 2016, doi: 10.1109/MCOM.2016.7378439.

[22] U. Ghoshdastider, R. Viga, and M. Kraft, “Wireless time synchronization of a collaborative brain-computer-interface using bluetooth low energy,” in *Proceedings of IEEE Sensors*, Nov. 2014, vol. 2014-Decem, no. December, pp. 2250–2254. doi: 10.1109/ICSENS.2014.6985489.

[23] A. Bideaux, B. Zimmermann, S. Hey, and W. Stork, “Synchronization in wireless biomedical-sensor networks with Bluetooth Low Energy,” *Curr. Dir. Biomed. Eng.*, vol. 1, no. 1, pp. 73–76, Sep. 2015, doi: 10.1515/cdbme-2015-0019.

[24] C. C. Rheinländer and N. Wehn, “Precise synchronization time stamp generation for Bluetooth low energy,” in *2016 IEEE SENSORS*, 2016, pp. 1–3. doi: 10.1109/ICSENS.2016.7808812.

[25] A. Yousefi, K. Somaratne, and F. J. Dian, “Analysis of time synchronization based on current measurement for Bluetooth Low Energy (BLE),” in *2017 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference, IEMCON 2017*, Oct. 2017, pp. 602–607. doi: 10.1109/IEMCON.2017.8117157.

[26] F. John Dian, A. Yousefi, and K. Somaratne, “Performance evaluation of time synchronization using current consumption pattern of BLE devices,” in *2018 IEEE 8th Annual Computing and Communication Workshop and Conference, CCWC 2018*, Jan. 2018, vol. 2018-Janua, pp. 906–910. doi: 10.1109/CCWC.2018.8301666.

[27] K. Somaratne, F. J. Dian, and A. Yousefi, “Accuracy analysis of time synchronization using current consumption pattern of BLE devices,” in *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*, 2018, pp. 841–844. doi: 10.1109/CCWC.2018.8301624.

[28] F. Asgarian and K. Najafi, “Time synchronization in a network of bluetooth low energy beacons,” in *SIGCOMM Posters and Demos 2017 - Proceedings of the 2017 SIGCOMM Posters and Demos, Part of SIGCOMM 2017*, Aug. 2017, pp. 119–120. doi: 10.1145/3123878.3132007.

[29] Nordic Semiconductor, “nRF51822 Product Specification v3.1,” 2015. https://infocenter.nordicsemi.com/pdf/nRF51822_PS_v3.1.pdf (accessed Jul. 23, 2021).

[30] Nordic Semiconductor, “nRF52832 Product Specification v1.1.” https://infocenter.nordicsemi.com/pdf/nRF52832_PS_v1.1.pdf (accessed Aug. 01, 2021).

[31] Dialog Semiconductor, “SmartBond^{TM} DA14585.” https://www.dialog-semiconductor.com/products/bluetooth-low-energy/da14585-and-da14586 (accessed Aug. 01, 2021).

[32] Texas Instruments, “CC2640R2F SimpleLink Bluetooth low energy Wireless MCU.” https://www.ti.com/lit/ds/symlink/cc2640r2f.pdf (accessed Aug. 01, 2021).

[33] Silicon Labs, “EFR32BG13 Blue Gecko Bluetooth Low Energy SoC Family Data Sheet.” https://www.silabs.com/documents/public/data-sheets/efr32bg13-datasheet.pdf (accessed Aug. 01, 2021).

[34] “MPU-6050.” https://invensense.tdk.com/products/motion-tracking/6-axis/mpu-6050/ (accessed Aug. 01, 2021).

[35] “ICM-20601.” https://invensense.tdk.com/download-pdf/icm-20601-datasheet/ (accessed Aug. 01, 2021).

[36] “Timeslot Advertiser-Observer.” https://github.com/NordicPlayground/nRF51-multi-role-conn-observer-advertiser (accessed Aug. 01, 2021).

[37] F. Asgarian and K. Najafi, “Poster: Frequency Scaling in Time Synchronization for Wireless Sensor Networks,” in *2018 14th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, 2018, pp. 113–114. doi: 10.1109/DCOSS.2018.00023.

[38] F. Asgarian and K. Najafi, “Reducing Synchronization Error in Wireless Sensor Nodes by Using Previous Timing Information as Training Data: Poster Abstract,” in *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*, 2019, pp. 432–433. doi: 10.1145/3356250.3361967.

[39] “nRF52 Outdoor Range Test.” https://devzone.nordicsemi.com/nordic/nordic-blog/b/blog/posts/testing-long-range-coded-phy-with-nordic-solution-it-simply-works-922075585 (accessed Nov. 08, 2022).

[40] “nRF52 Indoor Range Test.” https://www.nordicsemi.com/News/2019/04/Bluetooth-5-range-put-to-the-test (accessed Nov. 08, 2022).

[41] P. H. P. Wang *et al.*, “A Near-Zero-Power Wake-Up Receiver Achieving -69-dBm Sensitivity,” *IEEE J. Solid-State Circuits*, vol. 53, no. 6, pp. 1640–1652, 2018, doi: 10.1109/JSSC.2018.2815658.

[42] J. Moody *et al.*, “Interference Robust Detector-First Near-Zero Power Wake-Up Receiver,” *IEEE J. Solid-State Circuits*, vol. 54, no. 8, pp. 2149–2162, 2019, doi: 10.1109/JSSC.2019.2912710.

[43] H. Jiang *et al.*, “A 22.3-nW, 4.55 cm2 Temperature-Robust Wake-Up Receiver Achieving a Sensitivity of -69.5 dBm at 9 GHz,” *IEEE J. Solid-State Circuits*, vol. 55, no. 6, pp. 1530–1541, 2020, doi: 10.1109/JSSC.2019.2948812.

[44] N. E. Roberts *et al.*, “A 236nW -56.5dBm-sensitivity bluetooth low-energy wakeup receiver with energy harvesting in 65nm CMOS,” *Dig. Tech. Pap. - IEEE Int. Solid-State Circuits Conf.*, vol. 59, pp. 450–451, 2016, doi: 10.1109/ISSCC.2016.7418101.

[45] K. R. Sadagopan, J. Kang, S. Jain, Y. Ramadass, and A. Natarajan, “A 365nW −61.5 dBm sensitivity, 1.875 cm 2 2.4 GHz wake-up receiver with rectifier-antenna co-design for passive gain,” in *2017 IEEE Radio Frequency Integrated Circuits Symposium (RFIC)*, Jun. 2017, pp. 180–183. doi: 10.1109/RFIC.2017.7969047.

[46] P. Bassirian *et al.*, “Design of an S-Band Nanowatt-Level Wakeup Receiver with Envelope Detector-First Architecture,” *IEEE Trans. Microw. Theory Tech.*, vol. 68, no. 9, pp. 3920–3929, 2020, doi: 10.1109/TMTT.2020.2987786.

[47] J. Yi, W.-H. Ki, and C.-Y. Tsui, “Analysis and Design Strategy of UHF Micro-Power CMOS Rectifiers for Micro-Sensor and RFID Applications,” *IEEE Trans. Circuits Syst. I Regul. Pap.*, vol. 54, no. 1, pp. 153–166, 2007, doi: 10.1109/TCSI.2006.887974.

[48] S. Oh and D. D. Wentzloff, “A -32dBm sensitivity RF power harvester in 130nm CMOS,” *Digest of Papers - IEEE Radio Frequency Integrated Circuits Symposium*. pp. 483–486, 2012. doi: 10.1109/RFIC.2012.6242327.

[49] V. Mangal and P. R. Kinget, “A wake-up receiver with a multi-stage self-mixer and with enhanced sensitivity when using an interferer as local oscillator,” *IEEE J. Solid-State Circuits*, vol. 54, no. 3, pp. 808–820, 2019, doi: 10.1109/JSSC.2018.2884919.

[50] M. Miyahara, Y. Asada, D. Paik, and A. Matsuzawa, “A low-noise self-calibrating dynamic comparator for high-speed ADCs,” in *2008 IEEE Asian Solid-State Circuits Conference*, 2008, pp. 269–272. doi: 10.1109/ASSCC.2008.4708780.

[51] S. Dai and J. K. Rosenstein, “A 14.4nW 122KHz dual-phase current-mode relaxation

oscillator for near-zero-power sensors," in *2015 IEEE Custom Integrated Circuits Conference (CICC)*, 2015, pp. 1–4. doi: 10.1109/CICC.2015.7338396.

[52] "Centrifugal Force." http://www.copters.com/aero/centrifugal_force.html (accessed Jul. 25, 2022).

[53] "Xsens MTi-1 IMU." https://www.xsens.com/mti-1-imu (accessed Jul. 21, 2022).