NTNU
Norwegian University of
Science and Technology

Time synchronization across multiple devices in network nodes

Raghav Khosla

Master of Science in Telematics - Communication Networks and Networked
Submission date: September 2017
Supervisor: Danilo Gligoroski, IIK
Co-supervisor: David Edwin, Nordic Semiconductor

Norwegian University of Science and Technology
Department of Information Security and Communication Technology

Title: Time synchronization across multiple devices in network nodes

Student: Raghav Khosla

Problem description:

With the execution of complex networks in different areas of technology comes the responsibility to ensure effective communication within them so that data is securely transferred and received at the right destination as expected. One major hurdle in achieving this target is synchronizing time across multiple devices in different networks. Effective time synchronization is key especially where multiple events are taking place in the network. For instance, if we have a client-server system, where one server is interacting with many clients, forming a huge network, many tasks need to be performed. This involves timestamping to mark the execution of events which may not be accurate enough owing to the constantly varying drift between the computer clocks. So the goal here is to obtain as much accurate values as one can with respect to real time signals. It is important to pre-schedule the events and sort them in sequence to meet the required targets. Time needs to be synchronized, whether a wired or wireless network, in order to attain optimal stability.

Thesis Work:

Knowing that the local clocks at the network nodes would be continuously drifting apart from each other with respect to time, the goal here is to analyze the drift and the offset variations, frequency stability (temperature drift) as well as frequency tolerance in the networks. This drift between the clocks can be minimized if not eliminated with the help of some mathematical and graphical interpretations and by testing them experimentally with the desired tools.

Also I would question the interpretations from the Tiny-Sync and Mini-Sync algorithms in synchronizing time across Wireless Sensor Networks. I doubt that the uncertainty bound on drift (precision) actually increases as more samples of data points are collected. Another questionable point is the assumption that the drift between any two neighboring nodes be considered to be constant even in a small time interval as that is something impractical. I would consider certain broken points in these two known algorithms which I would try to cover up with some logical reasoning as a part of my thesis.

Moreover, the drift and offset values will be compensated for via an efficient

algorithm. The related upper and lower bounds on relative drift and relative offset need to be theoretically and graphically interpreted. The overall objective would be to obtain optimal solutions, precise enough to show good order of synchronization throughout the network.

The project work would go as follows: In depth research on the existing study done in this regard. This would involve a go through of sampling synchronization with Gigabit Ethernet as well as time synchronization in sensor networks with main focus on the latter. The synchronization problem analysis in each case. Algorithms and protocols proposed. Experimental, theoretical, and graphical conclusions drawn. Comparing the results obtained in each case.

Following this would be the practical implementation in synchronizing time across neighboring nodes (single hop) as well as between multiple/distant nodes (multi-hop) cases in broadcast mesh networks using Nordic Semiconductor tool suite.

All the implementation part would be aided by the nRF51 series Nordic Semiconductor devices using nRF Radio over Bluetooth Low Energy (BLE) stack along with the Programmable Peripheral Interconnect (PPI) functionality.The standard crystal oscillators to be used as reference sources will comply to that on the nRF51 Development Kit (nRF51 Dev Kit or nRF51 DK), being 16 MHz/32.768 kHz Crystal Oscillator (XOSC) (16M XOSC and 32k XOSC) with a cheap quartz crystal for optimal performance efficiency. The RADIO would catalyze the sending and receiving of BLE packets/probe messages among the different nodes by using well-defined and specialized registers of the Radio peripheral directly. Sniffer node periodically broadcasts probe messages. Corresponding timestamps would be generated with the local time and estimated reference time at each sending and receipt of the messages from the source node to the destination node. Events would be logged to the external flash memory. Furthermore, Logic software will be used for capturing data in real time with the Saleae Logic Analyzer to see how the packets from the nRF51 boards are sent and figure out if they are working like they should. Some of the key software and hardware tools to be used from the Nordic Semiconductor tool suite as well as from other sources are highlighted:

– nRFgo studio, nRF51 Software Development Kit (SDK), Master Control Panel, nRF Master Control Panel, ANTware-SW, nRF sniffer, nRFx Command Line Tools, nRF51-BLE-Driver, ARM Cortex-M0 processors, J-Link ARM, nRF51 DK, nRF51 Dongle, Segger J-Link Real Time Terminal (RTT) Viewer, Saleae Logic Analyzer, System on Chip (SOCs) like nRF51422 and nRF51822, Integrated development environments (IDEs) like Embedded Segger Studio and Keil5.

– IoT wireless technologies (ANT, ANT+, Bluetooth and Bluetooth Low Energy (BLE))

Responsible professor:	Danilo Gligoroski, NTNU
Supervisor:	David Devasahayam Edwin, Nordic Semiconductor ASA

Abstract

Effective time synchronization has become a critical issue nowadays, especially where loads of confidential data needs to be transferred via a medium from one point to another. Rapid advancements in technology have paved the way for the design of low-power networks/wireless solutions, for instance, Wireless Sensor Networks, and more.

Sensor networks, being a collection of a large number of interconnected nodes communicating with each other need to prioritize effective synchronization of data as well as time between message exchanges. A commonly stated fact being that the local clocks at the different nodes within the network are continually drifting apart in real time. In an effort to counterattack the drift and offset, precise timestamping needs to be done. This won't eliminate the drift but one expects that more accuracy can be attained while sampling multiple events taking place in the network. Considering the fact that if there is lack of coherence in the events taking place in such networks, such that there is no account of an event being generated and/or terminated, then it is difficult to assure the receipt of the data at the desired location at a particular instant of time.

The report highlights two earlier proposed algorithms for synchronizing time across sensor networks, namely Tiny-Sync and Mini-Sync. Certain loopholes are investigated in the proposed hypothesis and assumptions as well as conclusions drawn from these algorithms.

The entire thesis work focuses on timing synchronization across multiple devices in broadcast mesh type networks. Two separate cases have been brought about. One is synchronizing time between neighboring nodes (single hop) and the other being time synchronization among distant nodes (multi-hop). nRF Radio functionality as in the standard nRF51 specification will be used over BLE stack as the medium for sending/receipt of data packets between the different nodes. All this would be done using 16 MHz/32.768 kHz XOSC with a cheap quartz crystal as the reference sources. The local clocks at the corresponding nodes need to be frequency and phase locked (using a Phase Locked Loop (PLL)) with each other in order to be precisely matched with respect to each other. For this certain protocols/algorithms will be demonstrated for modeling the drift, offset, and other factors crucial for stability. Effective timestamping at each of the transmit/receive points is key to obtain precise estimates on the desired variations. Once the timestamps are generated, these will be used to derive certain mathematical, and graphical results taking

the relative drift between the node clocks due to temperature and other factors, as well the relative offset into account.

The Saleae Logic Analyzer tool will also be used to interpret the time durations (delays) between the defined Radio events as the packet is being transmitted/received.

Further, Segger J-Link RTT Viewer will be used for real time analysis of the captured timestamp values within the transmitted packets.

In addition to this, relevant conclusions will be drawn on the accuracy of synchronization achieved via practical analysis using the embedded software solution designed.

The project work would also address the weak points observed in the stated algorithms from the practical analysis done in an attempt to validate the core idea behind time synchronization in similar networks.

Preface

Since the past few years rapid technological advancements have led to a steep increase in the design of low-power and cost-effective communication devices and systems. With this comes the incessant need to synchronize time in such devices/networks. Not only this, but effective synchronization is extremely crucial especially in multi-hop ad hoc networks where loads of data needs to be transferred between a number of devices.

As it is known, it is impossible to control the clocks since they are continuously drifting apart in time. This may be due to voltage fluctuations, temperature and other environmental factors. Adding on, the drift may be positive or negative. Similarly clocks always have some offset too with respect to real time. All these factors, in effect limit the possibility of secure and reliable communication within a network.

Taking these factors into account, synchronizing time especially in complex networks is a matter of major concern. It requires precise modeling of the system behavior which may vary with the type of application being considered. One way of synchronizing in advanced multi-hop ad hoc networks or even Wireless sensor Networks is timestamping of the events taking place in real time within the network following a proper sequence. This is because these timestamps give the most accurate estimation with regard to the behavior of the internal clocks. If these are not reliable enough then the results wouldn't be as expected and henceforth effective synchronization will not be achieved.

As expected, there were certain challenges faced at different stages especially during the practical work on the thesis, for instance, in the Software Development Life Cycle while designing the *C* code for the embedded software solution. But then with an organized and a personable attitude, it was manageable for me to surpass these obstacles henceforth being able to achieve the expected results.

This thesis has been submitted for the fulfillment of my Master of Science (MSc) degree in Telematics-Communication Networks and Networked Services at Norwegian University of Science and Technology (NTNU). The thesis has been successfully completed in a period 7 months approximately. The thesis project was associated with Nordic Semiconductor ASA, one of the world's leading semiconductor company.

In the end, I would like to acknowledge the efforts of all those who have contributed towards the successful completion of this thesis. I express my deep gratitude to my supervisor at Nordic semiconductor ASA, Mr. David Devasahayam Edwin (Sr. Software Architect) as well as my professor, Danilo Gligoroski at Norwegian University of Science and Technology (NTNU), Norway for being a perennial source of guidance and motivation as well as for playing a significant role towards ensuring the academic perspective of this thesis. Their contribution has been significant throughout the period of my Master's thesis. Right from the initial research part to the practical implementation in order to support the study done in this regard and then even in the final stages of the thesis I have been constantly mentored by my superiors. I am extremely grateful to them for all their endeavors to help me successfully complete this Master's thesis. My extended thanks to Laurent Paquereau and Mona Nordaune at the Telematics department for their valuable assistance during the entire thesis period and not to forget, for giving me the opportunity to pursue my thesis at this esteemed organization.

Contents

List of Figures

List of Tables

List of Acronyms

ATM Asynchronous Transfer Mode.

BLE Bluetooth Low Energy.

CRC Cyclic Redundancy Check.

FTSP Flooding Time Synchronization Protocol.

GPIO General-purpose input/output.

GPIOTE GPIO Tasks and Events.

GPS Global Positioning System.

ICs Integrated circuits.

IDE Integrated Development Environment.

IDEs Integrated development environments.

IoT Internet of Things.

LED Light-emitting diode.

M2M Machine-to-machine.

MAC Media Access Control.

nRF51 DK nRF51 Development Kit.

NTP Network Time Protocol.

P2P Point-to-Point.

PPI Programmable Peripheral Interconnect.

PTP Precision Time Protocol.

RAM Random Access Memory.

RBS Receiver Broadcast Synchronization.

RTT Real Time Terminal.

SDK Software Development Kit.

SDLC Software Development Life Cycle.

SES SEGGER Embedded Studio.

SoC System on Chip or System on a Chip.

SONET Synchronous Optical Network.

TPSN Timing-sync Protocol for Sensor Networks.

WSNs Wireless Sensor Networks.

XOSC Crystal Oscillator.

Chapter 1
Introduction

1.1 Motivation

With time, there has been an evergrowing demand for effective communication in as well as between systems. Especially when it comes to loads of data being transferred to-and-fro within a system or between number of systems, as in wired or wireless networks, the main concerns are that how precisely the communication takes place, the degree of authenticity as well as confidentiality of the data, etc. This calls for the need to adequately synchronize the data as well as time to avoid unwanted delays and other errors in transmission.

Computers being the most commonly used machines today store a lot of private as well as other information having excellent memory capacity. Right from simple data transfers like email to most of the cash transactions, all these are being done online whether via computers or through other machines like ATM. How these systems are reliably optimized to perform such crucial tasks. How is it ensured that the data from a source is arriving at the correct destination and that there is no manhandling of data which may incur heavy losses. Here is where the idea of time synchronization comes into play.

The essential part of synchronizing time in networks is the use of clocks. When multiple events are taking place in a system, it is necessary to sort them in order of their priority or ignore them if unwanted. Clocks need to be used henceforth so as to schedule the tasks in sequence at appropriate time intervals to avoid confusion in the network. It is quite known that the clocks are continuously drifting apart in real time with respect to themselves as well as their neighbors. Thus, there will be always be some offset in them which may be positive or negative. The main goal behind synchronization is to cater for these drift and offset variations as well as other changes in networks with respect to frequency, temperature, voltage, etc. This is achieved via efficient algorithms/protocols to model the system in a manner so as to yield the optimal performance.

More convoluted the network is, more difficult it is the decision-making process to achieve proper synchronization. This involves intelligent decision making in setting priorities within a system to ensure the stability as a whole.

A handful of Nordic's standard software/hardware tools are used for generating timestamps, recording events, matching frequency and phase, etc. Relevant hypothesis and assumptions have been made while taking into account the existing theory proposed and conclusions drawn.

1.2 Significance of the thesis

The oddity of this thesis lies in the transparency of the results obtained while synchronizing across multiple devices within a mesh network. While much of the existing work done in this regard does attempt to convince one as per how and why to synchronize time in networks, yet their reasoning lacks ground on practical basis.

This work will basically aim at eliminating the flaws in the existing algorithms for synchronization like tiny sync and bringing out clear logic behind what is being actually done in this process and why it is being done as well as wherein the network. The focus is mainly on a broadcast mesh network wherein synchronization is done between the neighbors as well as distant nodes pertaining to single hop and multihop cases. Corresponding timestamps are generated and recorded at each of the nodes involved in communication with each other, mathematical interpretations for the drift (including temperature drift) as well as offset variations and other factors like frequency tolerance and stability, data point samples obtained hence giving the constraints, graphical plots to estimate the relative nature of the drift and offset between the clocks, statistical results and the corresponding conclusions drawn.

Moreover, the entire work is done using Nordic's nRF51 dev kit over nRF radio along with a 2.4 GHz transreceiver. Two crystals with a cheap quartz crystal are being used as the reference sources for all the nodes in the mesh instead of one and that too on the same board. One being 16M XOSC and other being 32.768 KHz XOSC. The local clocks at each of the nodes record the events taking place by timestamping, hereby paving the way for the subsequent calculations and interpretations as mentioned in the previous paragraph.

1.3 Report structure

The structure of the rest of the report is as follows: Chapter 2 presents the background and theory which is essential for understanding the following chapters. Chapter 3 describes the work done previously in synchronizing time across multiple devices within the nodes in different networks, with main focus on Wireless Sensor Networks

(WSN's) and a brief discussion about synchronization in Gigabit ethernet. Chapter 4 lists all the tools used along with a detailed description of their key functionalities in the software development process in context with the thesis work. Chapter 5 describes the Methodology with focus on the Training and the Project work. Chapter 6 presents the Experimental analysis for the work being entirely dedicated to the practical implementation involving the synthesis and verification of the parametric variations via an experiment on a nRF51-ble-bcast-mesh type network and analysis of the generated results. Chapter 7 gives the Conclusion on the thesis.Last but not the least, Chapter 8 highlights upon the future possibilities in synchronizing time across multiple devices.

Chapter 2
Background and Theory

This chapter presents the necessary background, theory and all the other information which is fundamental to understand the rest of the report. It describes the key facts that form the basis for analyzing the underlying concept behind synchronizing time within the network(s).

Time synchronization in communication (wired/wireless) networks has been under scrutiny for quite some time over the years. This being due to the incessant need for high speed and reliable data transfers to take place between several entities in a system. Henceforth, the time demands devising of cost-effective strategies for precise modeling of the system in a manner ensuring high degree of confidentiality and authenticity of private data as well as proper scheduling of the events taking place so as to avoid wastage of resources.

Being able to effectively synchronize time throughout a network proves to be favorable to several factors like motion, position as well as proximity within while interaction takes place between different devices/network nodes. [6]

Time synchronization is extremely critical in computer networks for all aspects of management, security, planning and debugging over the network so as to correlate the events actually happening. Moreover, considering time to be the sole frame of reference for different devices within a system, it is impractical to interpret the log file system along with other periodic events taking place between those devices without being able to accurately synchronize time. For instance, if a transaction is being made from one node to another on a shared file system, then the corresponding timestamps to be logged on at each of the send and receive events need to be accurate enough in order to match the sender and the receiver involved in the corresponding events. These timestamps are then bound to precisely estimate the degree of coherence between the neighboring devices (nodes) in a single hop transition as well as between distant devices (nodes) in case of a multi-hop system (network) hereby ensuring secure and reliable communication.[7]

2.1 Important terminologies concerning time synchronization

- **Accuracy:** The proximity of the absolute value of a clock to the 'null' value of offset. [8]
- **Accurate:** Any clock is considered to be accurate when it shows zero offset at a particular time instant. [8]
- **Drift:** The clock drift is defined to be a measure of the skew variation. Drift is also defined as the second derivative of the offset of a clock with regard to time. [8]
- **Joint resolution:** Considering two clocks $C1$ and $C2$ for comparison, the joint resolution in this case equals the sum of the individual resolutions of each of these two clocks. [8] This factor is an indicator of the lower bound on the accuracy with respect to the time intervals generated by the substraction of the corresponding timestamps recorded by one clock from the those generated by the second clock.[8]
- **Node:** A node in general refers to a device. So when doing synchronization between two nodes, it practically implies two individual devices that are being synchronized. However, in case of traditional synchronization methods like Network time protocol (NTP), this indicates an instantiation of this protocol over a local processor.[8]
- **Offset:** The net difference of value between the reported time from the clock and the true value of that time as per the Coordinated Universal Time (UTC). For instance, if the reported time and true time are found to be T_r and T_t respectively, then the offset is measured as T_r-T_t. [8]
- **Peer:** This is an indicator of an instantiation for Network Time Protocol (NTP) over a remote processor which has been connected from the local node via a defined network path. [8]
- **Relative offset:** When comparing two clocks, $C1$ and $C2$ with respect to each other, the time reported by $C1$ replaces the defined true time. So if $C1$ and $C2$ show the time values $Tc1$ and $Tc2$ respectively at a particular moment, then the offset of clock $C2$ relative to clock $C1$ is given as $Tc2 - Tc1$. [8]
- **Resolution:** Resolution is defined as the smallest unit of time used for updating the time on a clock. It is usually expressed in seconds.[8]

  It is defined in comparison to the clock's reported time and not the defined true time.[8] For instance, a resolution of 20 milliseconds implies that the notion of time on that clock has been updated as 0.02 second increments. But this does

not indicate anyhow for this time to be the true time defined in between the updates. [8]

Note: Clocks can be defined to have fine resolutions yet being inaccurate. [8]

– **Skew:** Clock drift and Clock skew are often confused when working with time measuring devices.

These are two different terminologies. Skew is indicative of the first derivative of the clock offset with regard to time. While drift, as defined earlier, corresponds to the second derivative of the clock offset with respect to time. [8]

– **Synchronize:** Two clocks are said to be mutually synchronized if and only if they are accurate with respect to each other, that is, the clocks are having zero relative offset.

Note: Two clocks can work in synchronization with respect to each other yet being inaccurate. This is determined by the preciseness of the true time defined in each of the two cases.[8]

There are two major areas in time synchronization, namely, Wired Network Synchronization and Wireless Network Synchronization.

2.2 Wired Network Synchronization

Two most common time synchronization protocols used for wired communications are Network Time Protocol (NTP) and Global Positioning System (GPS). [1]

2.2.1 Network Time Protocol (NTP)

NTP is quite known as the traditional method of time synchronization since it has been in use for quite some time now. It is known to transmit timing information over the internet with utmost accuracy as well as flexibility.

NTP basically operates on a client-server or a master-slave mechanism. Here the client requests the timing information from the system server to which the server responds. There may be multiple masters (primary servers) yet no election protocol comes into play. The time source for a network using NTP is usually a radio or an atomic clock being attached to a timing server. This source must be accurate enough to be relied upon. Using this time source, the timing information is distributed throughout the network. [8] The client computer (that too with its own local clock) to be synchronized to the server system sends a data packet asking for the corresponding timing information on the server side. Upon receipt of this packet, the server sends back the timing information to the requesting client as an

acknowledgement. These corresponding packet transmissions and receptions in the system allow for synchronization between the computers. [1]

Generally, there is always some sort of delay from when the original message is actually transmitted from the source and then when the acknowledgement is sent back from the destination upon receipt of that message so as to reconstruct the original message version at the source. With this scheme, although the delay is not totally nullified but it is compensated for to a great extent.

NTP finds its application mostly in Local Area Network topologies and is quite capable of delivering results of the order of few milliseconds. It is perhaps more difficult to manage the delays on the World Wide Web due to the excessive network traffic and other hurdles. [8]

In many cases, stratums are used to estimate the distance of the machine from the related time source in terms of the number of hops. One time server having its radio or atomic clock attached sends the time value in the packet that is transmitted to the second time server via NTP, and this process goes on. [8]

2.2.1.1 Advantages

To ensure effective time synchronization, NTP doesn't interfere with a machine with inaccurate time. Therefore, it is very critical for a machine to be self-synchronized, that is, synchronized to itself, for NTP to synchronize with it. In case there are multiple machines to choose from, then NTP doesn't choose the machine with significantly differing time with respect to others even if that machine has a lower stratum. [8]

Moreover, NTP offers flexible security features for avoiding malicious or accidental adjustment of the time parameter which is critical for any machine working on such a system. [8]

2.2.1.2 Limitations

Most wireless devices do not support the use of servers with atomic clocks due to being powered by batteries. Moreover, NTP design does not cater to the energy and computational constraints associated with the sensor nodes in Wireless Sensor Networks. Henceforth NTP is only supported for wired and not wireless communications. [1] [9]

Note: The working of NTP protocol as explained is shown with the help of Figure 2.2 when there is a direct comparison of this traditional method of synchronization with another synchronization protocol, namely, Receiver Broadcast Synchronization (RBS). [1] [9]

2.2.2 Global Positioning System (GPS)

Global Positioning System (GPS) uses satellite communication as a medium for synchronizing the entire network. There is a Global Positioning System (GPS) receiver installed to communicate with the satellites for the timing information. The maximum number of satellites with which the receiver is able to communicate becomes the deciding factor for the timing accuracy to be obtained. It is evident that this number is bound to vary with time, henceforth the timing accuracy also varies. [1]

2.2.2.1 Limitations

Again GPS doesn't work for wireless systems because of the power constraints. Also in wireless systems like Wireless Sensor Networks wherein a large number of nodes/devices are deployed in the system to communicate amongst each other, having a GPS receiver at each node turns out to be too expensive and yet not feasible. Furthermore, GPS relies totally on Line of Sight (LOS) communication which may not always be the case while deploying wireless networks.[1]

Although these synchronization methods are still in use yet sensor networks are totally ignorant of them due to the energy and complexity issues as well as cost and size factors. [9] Therefore, the traditional synchronization methods/protocols are suitable only for wired communication. [1] [9]

2.3 Wireless Network Synchronization

Over the years many advanced low-power wireless solutions have evolved with increased efficiency. Wireless technologies have given the communication industry a whole new shape. As far as the synchronization problem is concerned, wireless field has emerged as a more significant as well as demanding area than the traditional Wired communication.

Synchronizing time in a network may/may not precisely get all the local clocks at the nodes matched to each other as well as the reference but if accurately done, then it may generate results with proximity to the desirable results.

Just like the NTP protocol and GPS are relevant for synchronization in wired networks, similarly there are certain synchronization protocols specifically designed for wireless networks too.

There are three different types of methods for synchronizing time in wireless entities.

First being the simplest relying on the relativity of time. Here the sole purpose is to order the events taking place in the system. For instance, if there are two events known, event 1 and event 2, it is required to determine the relative order of occurrence of these events as to whether event 1 happened before event 2 or viceversa. This timing information can be extracted from the corresponding local clocks at the network nodes. Similarly, this case further extends to sorting of multiple events taking place in the system in a similar manner. [1]

The second method is also based on timing relativity but the network clocks have no dependency on each other. The nodes are well defined to track the drift and the offset variations (due to system changes as well as other external factors) with respect to their neighbors These also have the inbuilt functionality to synchronize their local time with that of the neighboring nodes at any time instant. This is the most commonly used method for synchronization purposes.[1]

The last method is that of global synchronization based on maintaining a universal timescale within the entire network. This is comparatively the most complex of all other previously described procedures henceforth being quite difficult to implement. Moreover, this method is very rarely seen as a requirement while synchronizing time in wireless systems. [1]

Figure 2.1 demonstrates the four major concerns bounding the synchronization schemes applicable for wireless networks.These being send time, access time, propagation time and receive time.[1]

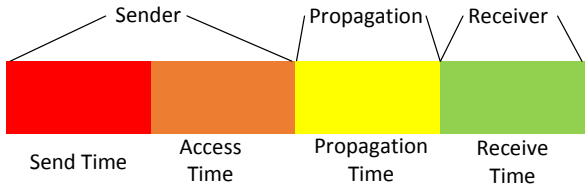

Figure 2.1: *Packet delay components*
Based on a figure in [1]

In Figure 2.1 Send time being the time taken by the sender to construct the message to be transmitted on the network. This would even take into account the involved operating system overhead as well as the time taken for the message to be transferred to the interface within the network for transmission. Next, the delay evolved in accessing the network over the Media Access Control (MAC) layer corresponds to access time. This time delay may be associated with the related MAC scheme used or waiting for an idle channel if the channel is busy with some other transmission upon arrival of a new packet or even due to waiting for an idle slot for

transmission using Time Division Multiple Access (TDMA) scheme. Furthermore, the time taken for the message to be transmitted over the physical medium corresponds to propagation time. Finally, the time it takes for the message to be received at the right destination, processed and transferred back to the originating host is the receive time. [1] [9]

It is a well known fact that during the transmission and the final reception, there will always be some sort of delay incurred due to the varying drift and offset of the clocks as well as other external factors. The above defined time parameters are the main sources for the delays, henceforth minimizing/eliminating any of them would definitely help in improving the overall synchronizing efficiency in the network.[2]

Several algorithms like Receiver Broadcast Synchronization (RBS), Timing-sync Protocol for Sensor Networks (TPSN), Flooding Time Synchronization protocol (FTSP), etc have already been proposed in this regard. Throwing some light upon these will further pave the path to visualize the practical considerations behind attaining synchronization throughout a network.

2.3.1 Reference Broadcast Synchronization (RBS)

The most distinguishing feature of this form of synchronization is that it involves receiver to receiver synchronization than the commonly known sender to receiver synchronization in case of NTP. Here a third party is used instead to broadcast information (beacon) to multiple receivers. The transmitted information doesn't involve any timestamping information to be given to the receiver(s). Instead, the local clocks at each of the nodes are compared to each other so as to precisely estimate the relative phase offsets. The time events basically record the receipt of the reference beacon at each of the desired nodes. [1]

Figure 2.2 on the next page explains how Receiver Broadcast Synchronization (RBS) works in contrast to the traditional method(s) for synchronization. RBS in its simplest form involves one broadcast beacon and two receivers. A packet will be broadcasted to each of the two receivers following which these receivers record the receipt of these packets with the help of their local clocks. This recorded information is then exchanged between the receiving parties to calculate their individual as well as relative offsets with respect to each other. This, in turn, helps to attain a local timescale. Further extension of this protocol may involve synchronization between 'n' receivers (n>2). Due to more number of receivers, more broadcast messages need to be sent which further contribute to alleviate the precision of the synchronization. [1]

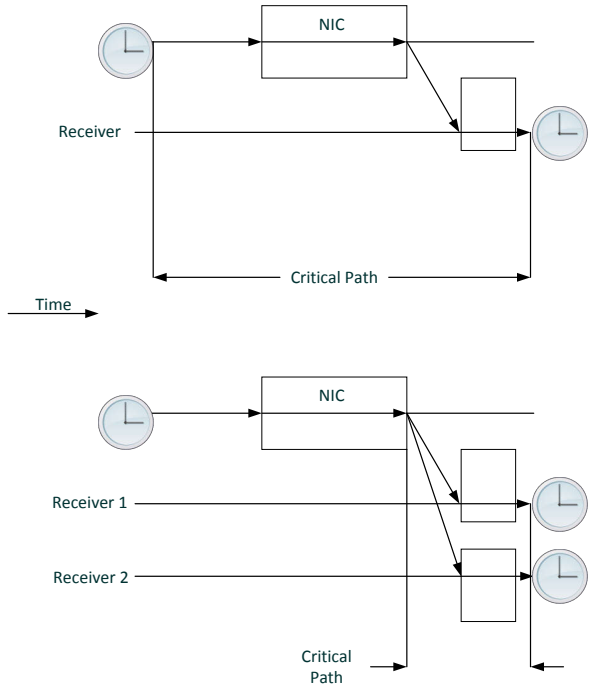

Figure 2.2: *Traditional time synchronization versus Receiver Broadcast Synchronization*
Based on a figure in [1]

2.3.1.1 Advantages

Thus, RBS being based on receiver to receiver synchronization rather than traditional sender to receiver synchronization involves less delay within the network due to elimination of the sender uncertainty (send time) as shown in Figure 2.1. Furthermore, with relatively narrower ranges, it has been considered that the broadcast messages arrive instantaneously henceforth nullifying propagation delay (propagation time). So the only source of uncertainty here lies in the receive time as in Figure 2.1. [1]

2.3.2 Timing Syn Protocol for Sensor Networks

This protocol involves the conventional sender to receiver synchronization. Two phases are involved in the synchronization. The first being level discovery phase wherein a hierarchy is formed with each node in the network being assigned a level. Only one node is assigned a level 0, being the root node. The second phase is the synchronization phase, where an ith node synchronizes to i-1th node and this

continues till each node is synchronized to the root and overall synchronization is achieved throughout the network. [1]

2.3.2.1 Level discovery phase

The root node initiates this process by broadcasting a level discovery packet containing the identity and level of the sender node. All the neighbors, upon receiving this packet assign themselves level 1. Once the nodes have received the packet, these discard the further incoming packets. This process goes on till all the nodes within the network are assigned a level as well as a level discovery packet. [1]

2.3.2.2 Synchronization phase

Figure 2.3 shows a pair of nodes, node A and node B to be synchronized as explained using the two-way message exchange mechanism. Node A sends a synchronization packet to node B containing A's level and the time T_1 when the packet was sent. This packet is received by B at T_2 time instant and at T_3, an acknowledgement is sent back to A containing B's level number as well as the values T_1, T_2 and T_3. This information is finally received at Node A at T_4. Once the drift is known, then A can correct its clock and synchronize itself to B. [1]

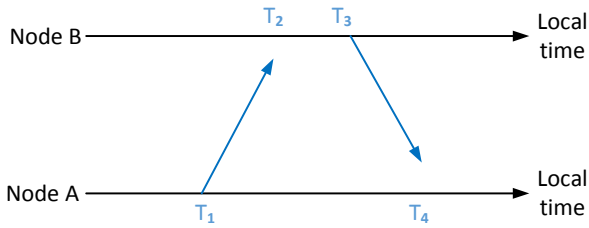

Figure 2.3: *Two-way communication between nodes*
Based on a figure in [1]

2.3.2.3 Advantages

The main advantage of Timing-sync Protocol for Sensor Networks (TPSN) is that it is able to minimize the uncertainty of the sender if not nullify it. Moreover, TPSN is designed to be a multi-hop protocol so transmission range is not an issue. Further, TPSN tends to minimize the sender uncertainty by generating a series of timestamps. All this is low-level timestamping done at the MAC layer. This enables a 2 to 1 better precision in TPSN than RBS henceforth claiming the superiority of the sender to receiver synchronization as compared to the receiver to receiver synchronization.Furthermore, transmission range is a limitation in case of RBS. Since RBS ignores propagation delay for relatively small ranges only, in case of multi-hop

networks, RBS lacks the flexibility to accurately synchronize while TPSN being designed for multi-hop networks would be efficient in such a situation. Additionally, the tree based topology used here allows the timing information to accurately propagate through the network. [1]

2.3.3 Flooding Time Synchronization Protocol (FTSP)

Like TPSN, Flooding Time Synchronization Protocol (FTSP) is also based on the traditional sender to receiver synchronization. Even in FTSP, we have one node assigned as the root node and rest all other local nodes need to be synchronized to the root. [1]

The synchronization process goes as follows. The root node sends out a single radio packet containing the timing information (sender's timestamp on a global timescale) to all the concerned receiver nodes. Each receiver having its local clock is able to record the time of the receipt of this packet from the root. In this manner, each node is aware of the transmission time from the root node as well as the receive time using which these nodes can easily determine their offset with respect to their neighbors. The message being timestamped at both the sender as well as the receiver interfaces happens at the MAC layer itself. Furthermore, due to the local clocks continually drifting apart with respect to real time, precise modeling of the clock drift is inevitable for effective synchronization. [1]

Although wide range of techniques have evolved for estimating the drift and offset variations in system clocks yet FTSP specifically uses Linear Regression for attaining higher precision via drift compensation. [1]

FTSP finds its application in large muti-hop networks. [1] The network structure follows a mesh type topology unlike the tree topology in case of TPSN. The root node is dynamically elected and can be replaced periodically to maintain the global timing information for the entire network. The receiving nodes communicate in an ad hoc fashion to synchronize themselves to the root and to convey the timing information amongst the peers. [1]

2.3.3.1 Advantages of FTSP

FTSP offers certain improvements over the earlier discussed TPSN protocol. Firstly, while TPSN provides a protocol for the management of network in an ad hoc fashion, yet it is totally irresistible to dynamic changes within the network. In other words, if the root node or the current working topology is suddenly changed, then the entire level discovery phase needs to be reinitiated which might lead to unwanted delays in the network. On the other hand FTSP is totally resistant to dynamic changes. Since the root node is dynamically elected and then reelected, this requires FTSP to

be robust so as to meet these requirements. Moreover, a number of synchronization messages are often taken together to fight the critical failures in the system. This further catalyses the ability for FTSP to withstand topology changes.Very much like TPSN, all the timestamping is done at the MAC layer for better precision and minimized jitter. This eliminates the limiting factors for effective synchronization leaving the propagation error delay. [1]

As earlier stated, frequent timestamping of events and linear regression allow for precise estimation as well as compensation for the clock drift and offset parameters.

Figure 2.4 showing the packet construction in FTSP is explained. We have the preamble followed by sync bytes. Next is the data used in transmission and finally there is the Cyclic Redundancy Check (CRC). The actual bytes as in the packet as well as the bytes in the buffer are demarcated by dashed lines and solid lines respectively. At first, the preamble bytes are transmitted by the sender while the receiver adjusts as per the specific carrier frequency. Once the sync bits arrive at the receiver, the receiver evaluates the bit offset to reconstruct the original message. All the required timestamps are positioned at the boundaries pertaining to the sync bytes.[1]

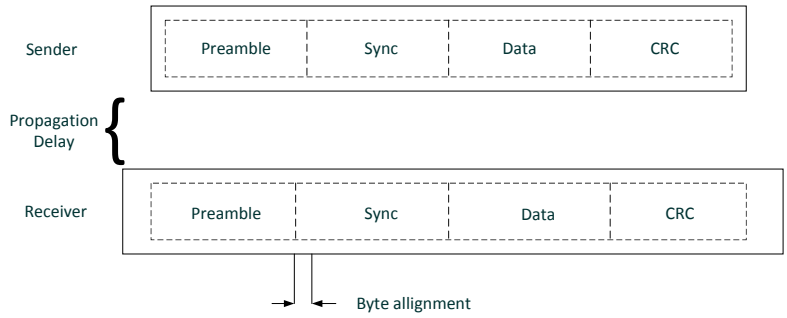

Figure 2.4: *Transmitted data packets using FTSP*
Based on a figure in [1]

All in all, allowance for dynamic changes in topology, robust nature to cope with link and node failures, as well as timestamping at the MAC layer for precision are the key advantages of this synchronization scheme. Furthermore, network wide synchronization is achieved with the flooding of synchronization messages so that all nodes are synchronized to the root node. [1]

Chapter 3
Previous Work

This chapter will focus mainly on time synchronization in Wireless Sensor Networks along with an overview of sampling synchronization with Gigabit Ethernet with reference to the work done till date in this regard.

Over the years fast-paced technological advancements have paved the way for the discovery of low-power wireless solutions being capable of multitasking, such as performing sensing and communication tasks, etc. Sensor technologies have received great attention in this regard.

The main significance behind research in this regard is that very less study has been done solely catering to this area.

Wireless Sensor Networks (WSNs) are being extensively used today for a wide variety of applications as in military, medicine, etc. WSNs being a collection of a large number of nodes interconnected with each other require effective utilization of the limited energy and resources available. These networks are even considered as special sort of multi-hop ad hoc networks. The entire communication takes place by means of data transmission from the source to the destination among several nodes in the network due to lack of support structure within. Henceforth accurate and reliable communication is key to serve the desired purpose.[9]

Although many cryptographic techniques including specialized key management schemes have evolved over the years for secured data transmissions taking place, yet another major concern here is the need for effective time synchronization. Achieving effective synchronization practically is not a trivial task at all. This is evident from the fact that it is infeasible to continuously monitor the clocks/timers being used in the systems for correlating the timing information. Moreover, synchronization is not a single step process but it involves multiple agendas to be simultaneously catered to, which is practically very difficult to implement and achieve. [10]

Effectively synchronizing time in networks may seem to be an easy to do task yet that is not the reality as per the actual case of study. Synchronization at its root level involves two main things. Firstly, to achieve and maintain a uniqueness in the time scale among all the network nodes. Secondly, the necessity to define only one single event as the time zero event.[10]

Synchronizing time in WSNs commonly requires accurate timestamping of the events taking place. This helps in precise ordering of the occurring events. Although these timestamps are not exact values that can be totally relied upon especially where loads of confidential data needs to be transmitted yet these are in effect able to render a precise estimation for the initiating node to have an idea of the time event(s) known to have occurred amongst its peers. Furthermore, considering the local clocks at each node to be measuring the time of the occurring events, synchronization involves matching all the clocks with respect to each other to set them on a unique time scale. In some cases, if a source or a reference for all devices/nodes is considered then it may also be important for all the local clocks to be in synchronization with this reference too.

Usually a standard crystal oscillator with defined frequency characteristics is utilized as the source. Varied crystal oscillators are being manufactured nowadays with their unique drift, frequency tolerance, operating temperature range and other parametric variations. Since the clocks are continuously drifting apart in time and it being impossible to monitor them continuously, henceforth the crystal oscillator to be used is to be chosen keeping in mind the desired considerations as per the network to be synchronized.

The following part of this chapter focuses on the clock synchronization problem along with highlighting the challenges and requirements for time synchronization in Wireless Sensor Networks as well as analysis of the synchronization methods. Then two well-known data collection algorithms, namely, Tiny-Sync and Mini-Sync will be discussed for studying the variations in the clock drift as well as the offset via certain bounded estimates henceforth presenting a solution for the synchronization problem with some mathematical interpretations and graphical depictions. In addition to this, a direct comparison via appropriate results and conclusions is obtained in terms of the applications, limitations and efficiency of these algorithms in synchronizing time in Wireless Sensor Networks. Moreover, apart from synchronizing time between two neighboring nodes using the aforementioned algorithms, an extended unique approach for synchronizing multiple nodes in a multi-hop scenario (distant nodes) has also been explained.

3.1 Clock Synchronization Problem

Computer clocks with hardware oscillators are used in the computing devices. The standard denotion for the clock as a function of real time being expressed as $C(t)$. The rate at which the clock operates is incumbent upon the angular frequency associated with the hardware oscillator of that clock. The rate denoted by (dC/dt) equals unity for a perfectly synchronized clock which is not practically feasible owing to the continuous drifting of the clocks in real time. Even though the clock frequency varies with regard to time yet it can be accurately approximated via an oscillator having fixed frequency. [9].

Equation 3.1 [9] derived from this assumption to represent the local clock at node i, say node 1, becomes

$$C_i(t) = a_i t + b_i, \tag{3.1}$$

where Equation 3.1 denotes $a_i(t)$ as the clock drift defined as the rate (frequency) of clock at node i and b_i being the offset of the clock at node i which is defined as the difference in value corresponding to real time, t.

Furthermore, Equation 3.2 [9] relates the local clocks, $C_1(t)$ and $C_2(t)$ respectively at nodes 1 and 2 as

$$C_1(t) = a_{12} C_2(t) + b_{12}, \tag{3.2}$$

where $C_1(t)$ and $C_2(t)$ denote the clocks at nodes 1 and 2 respectively as a function of time. a_{12} and b_{12} denote the relative drift and the relative offset respectively between the two clocks.

Here drift that is defined as the rate (clock frequency) is otherwise usually expressed in Parts per million (ppm), while offset is usually defined as the difference in the value with respect to real time. [9]

In case of these two clocks being perfectly synchronized in time then the relative drift equals unity (1) considering the clocks to be operating at the same rate. But their relative offset equals zero since these have the same value at that instant. [9]

3.2 Challenges for time synchronization in WSNs

The four main challenges for the synchronization schemes designed for Wireless Sensor Networks (WSNs) remain the same as have been discussed earlier. These

being send time, access time, propagation time and receive time. [6] Being able to eliminate or reduce the effect of any of these major constraints would boost the performance of the applied synchronization scheme in such networks.

3.3 Importance of synchronization in sensor networks

As previously emphasized on the necessity of time synchronization in wireless networks, synchronization in sensor networks has become a matter of major concern due to certain key considerations.

1. Firstly, sensor networks comprising of a huge density of nodes require adequate collaboration and coordination for the tasks to be performed depending upon the complexity level. So this requires resourceful time co-ordination to achieve optimized performance. One such example of similar tasks to be performed could be data fusion in such networks which combines the data cumulated at different nodes to achieve an aggregate result. Another example could be of a vehicle tracking system relying on all the sensor nodes reporting the location and time of the vehicle to the sink node so as to be synchronized with respect to each other at all times without which the estimates could turn out to be inaccurate. [9]

2. Secondly, optimum synchronization helps saving the battery life for such networks by means of specialized power saving schemes. This proves to be extremely significant for sensor networks owing to the limited energy resources. Such power saving schemes allow for only a node or a set of nodes to be active at any instant, which are to be involved in the task. Apart from these, the rest of the idle nodes may go to sleep for a while until they are commanded to be active. It is also important to ensure that the corresponding receiving end of the node is not inactive when the data is bound to be received there. This, in turn, is more or less dependent upon how the timing is set in the network which further contributes to determining the behavior and performance especially in sensor networks.[9]

The diversified synchronization schemes deployed in WSNs need to cater to the aforementioned considerations. These would also determine the expected performance efficiency of the deployed scheme but with some bounded trade-offs between the requirements for each in terms of precision and energy efficiency. [9]

3.4 Key requirements for deploying varied synchronization schemes in Wireless Sensor Networks

Accurate deployment of efficient synchronization schemes in sensor networks being of primary concern has certain defined requirements that need to be met. These are explained. [9]

Energy efficiency: It is extremely crucial for any synchronization scheme deployed for sensor networks to cater to the energy and resource constraints at all stages of the synchronization process. [9]

Scalability: Another major concern is that owing to the deployment of a huge density of sensor nodes in most of the applications, the scheme used needs to scale well with each of the participating nodes in the network. [9]

Precision: As far as precision is concerned, this requirement is bound to vary as per the specified application in use as well as the goal of synchronization, that is, whether local or global synchronization. While in some cases, a simplified ordering of the messages/events would suffice, however in other cases there may be strict requirements for precise synchronization on the order of few microseconds.[9]

Robustness: In case of the occurrence of a minor or a critical failure effecting the nodes in the network, the deployed synchronization scheme should be flexible enough to resist the change without effecting the normal operation within the network. [9]

Lifetime: Another matter of concern while synchronizing time in such networks is that the universal/global timescale achieved with the help of an algorithm or a protocol must be retained at all timing instants of the operation of the network. [9]

Cost and size: On one side where synchronization is to be achieved, it is also desired to have a cost-effective performance, that is, maximum efficiency at affordable price. Therefore, due to the huge density of nodes deployed in sensor networks, installing a GPS receiver at each node is not practically feasible as it turns out to be very costly. Henceforth, the method to be used must be investigated beforehand keeping the limited cost and size considerations in mind. [9]

Immediacy: It is all the more critical to cater to the emergency situations arising within such networks. Here the sink node is desired to receive the necessary information from any of the other nodes without any incurring delay. Therefore, the protocol designer must ensure to avoid extensive processing if such an immediacy occurs which, in turn, requires pre-synchronization of the network nodes at all times. [9]

3.5 Synchronization methods for sensor networks

Several methods have been proposed in this regard each being unique in terms of efficiency, resource utilization, etc.

As an efficient energy saving technique, the sensor nodes are to be maintained in a low-power state if not switched off completely. This also varies as per the synchronization scheme ought to be deployed in the network. For instance, in some cases, tight synchronization with high precision is desired requiring all nodes to be actively synchronized at all times. While for others this requirement can be relaxed by keeping the nodes inactive for some time when they aren't desired to be used for the task under consideration. It is due to this reason probably that the hardware part of the sensor networks including the processors are comprised of many sleep modes as well as the ability of lowering down high-energy peripheral devices which are not required to do the specific task. [11] [9]

While most of the traditional methods of synchronization like NTP require the clocks to be running at all times so as to acquire meaningful timestamps, one method, namely, Post facto synchronization [11], as a pioneering work by Elson and Estrin is discussed.

In post facto synchronization, all clocks normally run unsynchronized. Upon the arrival of a stimulus, the nodes record the timing with their individual clocks. Moving on, when a third party node behaving as a beacon broadcasts a synchronizing pulse with its radio to all nodes within the area, all receiving nodes take it as an instantaneous time reference to which they normalize their initial timestamps. However, this type of synchronization usually has a limited scope with respect to the transmit range associated with the beacon and that it is able to achieve synchronization at certain specific instants. So it is unsuitable for timing communication over long intervals or distances. Post facto synchronization typically find its use in beam-forming applications, and areas focusing on the relative signal arrival times on a set of spatially separated local detectors. [11]

This further led to the evolvement of the earlier discussed synchronization scheme, that is, RBS. Besides post facto synchronization and RBS, other schemes like TPSN have already been discussed.

3.6 Tiny-Sync and Mini-Sync algorithms

Tiny-Sync and Mini-Sync algorithms have been proposed by Sichitiu and Vcerarittiphan mainly for time synchronization in sensor networks. [9]

The earlier defined relationship as in Equation 3.2 between the local clocks of two neighboring nodes in the network holds.

Figure 3.1 shows the basic working of the algorithm in synchronizing two nodes. Both the Tiny-Sync and Mini-Sync algorithms utilize the traditional synchronization method involving two-way messaging between a pair of nodes for the interpretation of the relative drift and relative offset between the clocks. This scheme is shown and the process is described further. First, node 1 transmits a message probe containing the timestamp t_o, the time just before the transmission as recorded by 1's clock. Furthermore, upon the receipt of this message, node 2 timestamps it as t_b and immediately acknowledges the receipt to node 1. On receiving back the acknowledgement, node 1 timestamps it as t_r. In this manner, the three timestamps generated result in a data point (t_o, t_b, t_r) [2]

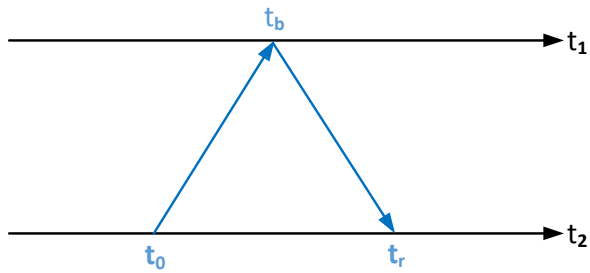

Figure 3.1: *Two-way message exchange between nodes 1 and 2 resulting in the data point (t_o, t_b, t_r)*
Based on a figure in [2]

This generated data point as in Figure 3.1 imposes significant limitations on the possible values for the relative drift and offset parameters being denoted by a_{12} and b_{12} respectively. [9]

The proper sequencing of the timestamps as shown in Figure 3.1 generates the desired inequalities. Since t_o was the first and the foremost one followed by t_b and then finally t_r, the inequality shown in Equation 3.3 [2] being defined for t_o occurring before t_b is given as

$$t_o(t) < a_{12} t_b(t) + b_{12} \tag{3.3}$$

and the inequality shown in Equation 3.4 [2] being defined for t_b occurring before t_r is given as

$$t_r(t) > a_{12} t_b(t) + b_{12} \tag{3.4}$$

In this manner, the measurements are performed repeatedly with each returning probe message determining a set of three possible timestamps. This, in turn, generates a set of data point samples, each imposing its restrictions on the drift and offset values with respect to the two clocks. With increase in the number of data point samples recorded, the overall precision of this algorithm is bound to increase. [9]

Moving on, each data point contributes two constraints on the relative drift and the relative offset with respect to the concerned clocks. These being (t_b, t_o) and (t_b, t_r). [2]

Figure 3.2 on the next page shows the linearity relationship as well as the corresponding constraints imposed on a_{12} and b_{12} as illustrated graphically. Since the two clocks under consideration are assumed to be linearly related, henceforth the inequalities shown in Equation 3.3 and Equation 3.4 hold only until the line represented by Equation 3.2 lies in between the constraints imposed by each of the two data points. [2]

Inspite of all this, the algorithm fails to determine the exact values corresponding to the relative drift and relative offset. This being due to the fact that while dealing with such timers/clocks to record the events, there could be many unwanted sources of delays possibly incurred in the process which are quite tedious to account for in some cases.

From Figure 3.2, it can be seen that the Tiny-Sync algorithm gives certain bounds on a_{12} and b_{12} parameters which just serve as estimates to get an idea of how well the internal system clocks are behaving. The main focus here should be to obtain as much tighter bounds as possible on the drift and the offset so as to achieve greater precision, where $\overline{a_{12}}$ and $\underline{a_{12}}$ respectively are the maximum and minimum of the slopes corresponding to the lines satisfying the constraints. $\overline{b_{12}}$ and $\underline{b_{12}}$ respectively correspond to the greatest and smallest intercept values on the y-axis at the intersection with the lines corresponding to $\overline{a_{12}}$ and $\underline{a_{12}}$. [3] [2]

It can be inferred from Figure 3.2 that a_{12} and b_{12} can be bounded within certain defined limits.

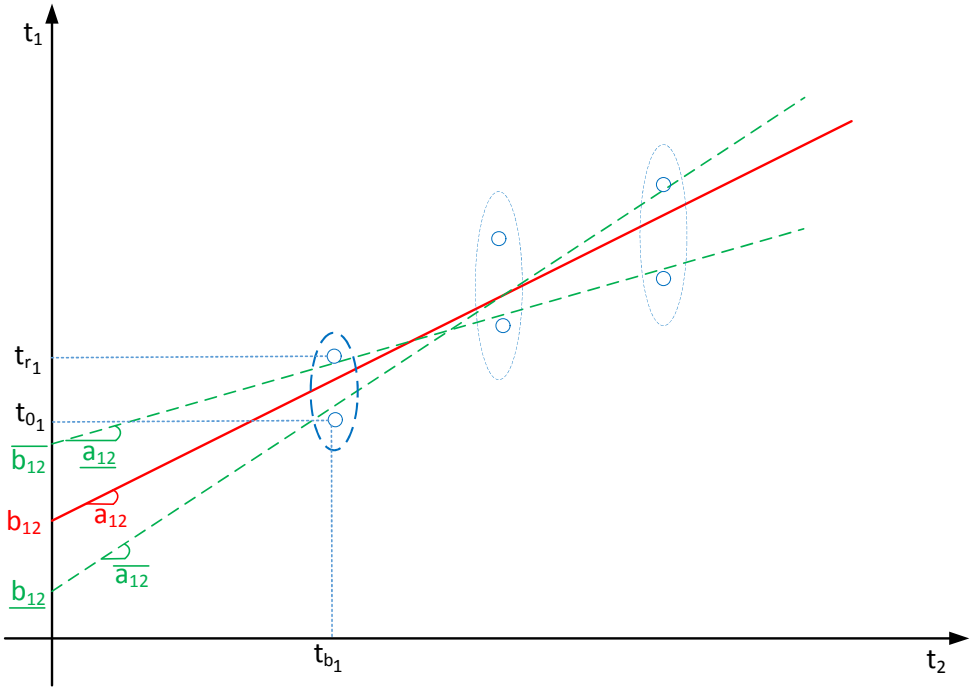

Figure 3.2: *The linear dependence and the constraints imposed on a_{12} and b_{12} by the generated data points*
Based on a figure in [2]

Equation 3.5 [2] [3] denotes $\underline{a_{12}}$ and $\overline{a_{12}}$ as the lower bound and upper bound respectively on the relative drift.

$$\underline{a_{12}} \leq a_{12} \leq \overline{a_{12}} \tag{3.5}$$

Similarly, Equation 3.6 [2] [3] denotes $\underline{b_{12}}$ and $\overline{b_{12}}$ as the lower bound and upper bound respectively on the relative offset.

$$\underline{b_{12}} \leq b_{12} \leq \overline{b_{12}} \tag{3.6}$$

Using the Figure 3.2, more equations have been derived which are self explanatory from the graphical interpretation. [3]

As of a_{12} and b_{12}, Equation 3.7 [3] gives the real value for relative drift estimated to be the midpoint of the range of possible values $\widehat{a_{12}}$.

$$a_{12} = \widehat{a_{12}} \pm \frac{\Delta a_{12}}{2} \tag{3.7}$$

and Equation 3.8 [3] gives real value for relative offset as estimated to be the midpoint of the range of possible values $\widehat{b_{12}}$.

$$b_{12} = \widehat{b_{12}} \pm \frac{\Delta b_{12}}{2} \tag{3.8}$$

In Equation 3.7 and Equation 3.8, Δa_{12} and Δb_{12} represent the uncertainty bounds on the relative drift and relative offset respectively.

Further on, Equation 3.9 [3] takes the the average of the upper bound and lower bound on the relative drift expressed as

$$\widehat{a_{12}} = \frac{\overline{a_{12}} + \underline{a_{12}}}{2} \tag{3.9}$$

as used in Equation 3.7.

Similarly, Equation 3.10 [3] takes the the average of the upper bound and lower bound on the relative offset expressed as

$$\widehat{b_{12}} = \frac{\overline{b_{12}} + \underline{b_{12}}}{2} \tag{3.10}$$

as used in Equation 3.8

Also, Equation 3.11 [3] expresses the uncertainty bound on the relative drift as

$$\Delta a_{12} = \overline{a_{12}} - \underline{a_{12}} \tag{3.11}$$

Likely, Equation 3.12 [3] expresses the uncertainty bound on the relative offset as

$$\Delta b_{12} = \overline{b_{12}} - \underline{b_{12}} \tag{3.12}$$

Therefore, when analyzing varied set of values for a_{12} and b_{12}, there will only be selected valid combinations and only for those valid combinations will the inequalities shown in Equation 3.3 and Equation 3.4 hold true. Moreover, having attained the

estimated values for the relative drift and the relative offset for the concerned clocks, it is possible for node 1 to correct its clock reading accordingly to have it matched with the reading recorded by the clock at the node 2. [2]

3.6.1 Practical considerations while generating data points using Tiny-Sync algorithm

While the initial case did not take into account the possible delays in the two-way message transfer considering node 2 to immediately reply back to node 1 having received the message, yet this is not the case as seen from a practical point of view. There could be delays due to the processing time, etc. [2]

Also that even after considering the intermediate delays encountered, the overall approach is similar to the one explained earlier for generating the data points using Tiny-Sync algorithm. The relations represented in Equation 3.3 and Equation 3.4 and the other analysis part remains almost the same. [2]

It is very much likely for node 2 to delay the response back to node 1 by as much amount of time desirable. But one noticeable fact here is that the preciseness of the estimates attained deteriorates with increase in the overall lag between t_o and t_r. [2]

Figure 3.3 on the following page clearly shows that practically there will be some delay at node 2 between the corresponding message reception followed by transmission henceforth taking two timestamps into account at node 2 rather than only one as considered in the initial case shown in Figure 3.1 on page 23. Now node 2 timestamps one event as t_{br} upon receipt of the message from node 1 and another timestamp as t_{bt} when it resends back to node 1. As a result, two triplets (t_o, t_{br}, t_r) and (t_o, t_{bt}, t_r) are obtained, each representing a data point satisfying the inequalities attained earlier as shown in Equation 3.3 and Equation 3.4. [2] Henceforth, just by using one message probe, two independent data points are attained in contrast to a single one as in the earlier case. [2] [3]

3.6.2 Increasing the accuracy by considering the minimum delay

In case of delay encountered by the probe messages during the transmission, there needs to be a source which caters to this delay. Even if the minimum delay encountered between the nodes is known, it is permissible to boost the overall performance by appropriate adjustment of the obtained data point samples. If not, it is possible to determine the minimum delay via the minimum length of the probe message under consideration as well as the amount of time taken to transmit the message and for other tasks to be performed prior to the transmission and on the message reception as happens in encryption/decryption or Cyclic Redundancy Check (CRC) check. [2]

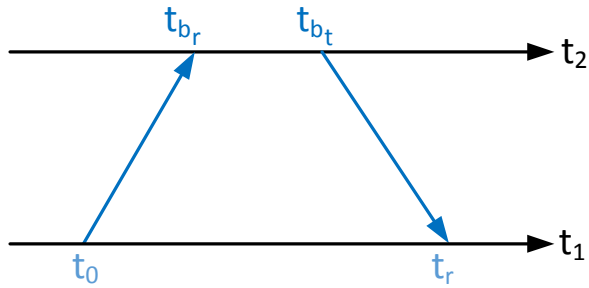

Figure 3.3: *A probe message from node 1 may be returned by node 2 and timestamped at both the send and receive points resulting in two data-points, (t_o, t_{br}, t_r) and (t_o, t_{bt}, t_r)*
Based on a figure in [2]

[2] [3] For instance, x is the delay encountered between just before the message being sent is timestamped as t_0 at node 1 and timestamp t_{br} is obtained on the receipt of the message at node 2. Similarly, y is the delay between the message being resent from node 2, timestamped as t_{bt} and finally being timestamped as t_r upon its arrival back at node 1. Then the data point recorded as $(t_o + x,\ t_b,\ t_r - y)$ is considered to be more accurate than $(t_o,\ t_b,\ t_r)$. If anyhow it is achievable to reduce these delays to a minimum value, then the two nodes are said to be perfectly synchronized to each other with

$$\Delta a_{12} = \Delta b_{12} = 0$$

3.6.3 Data processing with Tiny-Sync and Mini-Sync

To begin with, having two data points, it is possible to obtain estimates on the relative drift and offset with the inequalities defined earlier in Equation 3.3 and Equation 3.4. Optimal bounds on these values are only achievable via two linear programming problems having twice the number of inequalities as the available data points. [2] [3]

One major limitation of this approach is that an increase in the number of data point samples collected leads to higher computational complexity as well as potentially unbounded increase in storage requirements. Moreover, it is equally likely that a limited number of data samples may not render precise drift and offset estimates owing to their rapid fluctuations with time. So it is difficult to model such systems with this approach. [2] [3]

Tiny-Sync algorithm imposes certain restrictions on the actual usability of all the collected data point samples. It compares them with the degree of tightness of the

bounded estimates on a_{12} and b_{12} parameters.

Figure 3.4 shows the constraints obtained from four different data points. Among the first three data points, the first and the last collected data points are the best suitable for the initial estimates on a_{12} and b_{12}. This leads to the removal of the second data point instantly without any further consideration. However, it is known that there may be a probability that with the fourth data point sample collected, the earlier discarded second data point could have had a better match in comparison to the first and third data points. [2] [3] Tiny-Sync basically considers four constraints to be enough at a point of time wherein each of the two data points taken into consideration gives two constraints. Henceforth, on the arrival of a new data point, the two new constraints are compared with the initial four, and then two out of the six available constraints are discarded leaving only four suitable ones.[3] The four constraints recorded each time correspond to a simplified computation with 8 additions, 4 divisions and 4 comparisons. [2] Henceforth, only 8 timestamps (each of the four constraints giving 2 timestamps) are to be stored at a single point of time. It is equally likely that these 4 constraints might be generated using two, three or even four varied data points. [2] [3]

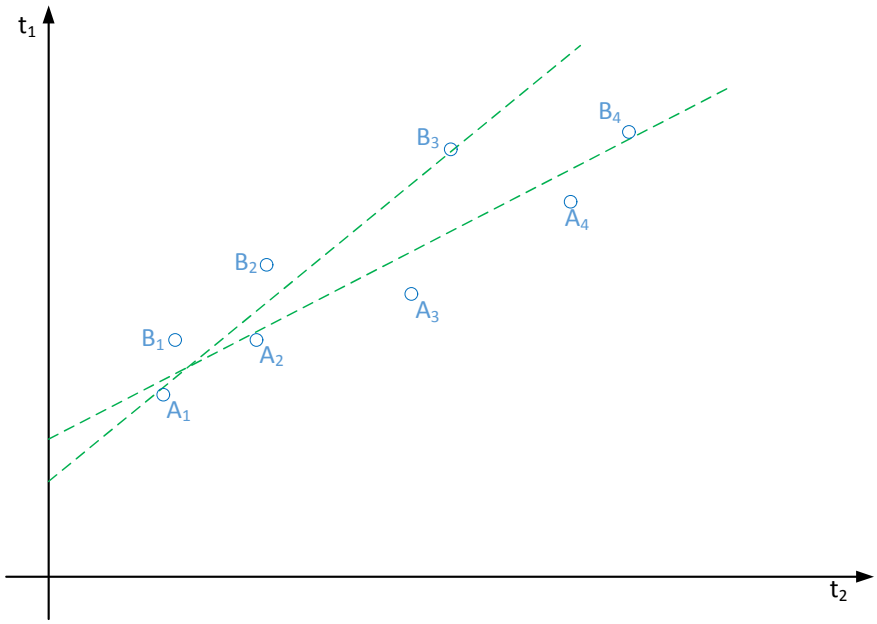

Figure 3.4: *Failure of Tiny-Sync algorithm to give the optimal solution in this case due to the constraint A_2 being discarded upon receipt of the data point $(A_3 - B_3)$ Based on a figure in [2]*

This can be further detailed as explained. Considering a set of four data points $(A_1 - B_1)$, $(A_2 - B_2)$, $(A_3 - B_3)$, and $(A_4 - B_4)$ as in Figure 3.4, where A_i constraint given as (t_{b_i}, t_{o_i}) and B_i constraint given as (t_{b_i}, t_{r_i}). After getting the first two data points $(A_1 - B_1)$, and $(A_2 - B_2)$, the initial estimates on the drift and the offset can be computed. Following this, on receipt of the third data point $(A_3 - B_3)$, somehow there is an improvement in the bounds on the desired estimates (with uncertainty in the bounded estimates Δa_{12} and Δb_{12} being smaller). So the data points $(A_1 - B_1)$ and $(A_3 - B_3)$ giving relatively tighter bounds are stored while $(A_2 - B_2)$ is discarded. Irrespective of the fact that the subsequent data point $(A_4 - B_4)$ could have rendered more precise estimates with A_2. This further gives an inappropriate estimate on b_{12} imposed by A_1 and A_4. [2] [3]

Furthermore, the fact that the Tiny-Sync algorithm cannot be totally relied upon to give an optimal solution for the bounds on the relative drift and the relative offset is evident from Figure 3.4. Henceforth, tiny sync algorithm while aiming to deliver precise estimates might miss the optimal solutions for the same. Thus, disregarding estimates just for being inappropriate in the short run but not considering their future possibilities to be useful enough to render better estimates in combination with others makes tiny sync inefficient in effectively synchronizing time especially in Wireless Sensor Networks. [2] [3]

Also, among a set of possible constraints available at a particular time instant, it is not necessary that all of them will be potentially useful for future considerations too. Taking this into account, the algorithm further imposes a condition for determining whether a constraint is potentially useful or not. [2] [3]

A constraint A_j (for instance, A_2) shown in Figure 3.4 is considered to be potentially useful in the future if and only if it satisfies the condition as given by Equation 3.13 [2] [3] which relates the slopes of two lines passing through the points (i,j) and (j,k) respectively :

$$m\left(A_i, A_j\right) > m\left(A_j, A_k\right) \tag{3.13}$$

where the Equation 3.13 denotes the integers i, j, k to be bounded as $1 \leq i < j < k$; and $m(X, Y)$, denotes the slope of the line passing through the points X and Y. [2][3]

Considering the aforementioned criteria as in Equation 3.13 pertaining to the usefulness of a constraint imposed by a data point, a theorem has been defined.

Equation 3.14 on the facing page [2] [3] relating the slopes of two lines passing through the points (i, j) and (j, k) respectively shows how a constraint A_j (for

instance, A_3) shown in Figure 3.4 can be successfully discarded if it is not considered as potentially useful.

$$m\left(A_i, A_j\right) \leq m\left(A_j, A_k\right) \tag{3.14}$$

wherein Equation 3.14 causes a constraint denoted by a minimum of a set of integers $1 \leq i < j < k$ to not be considered potentially useful since it will never give as much tighter bounds on $\underline{a_{12}}$, $\overline{a_{12}}$, $\underline{b_{12}}$ and $\overline{b_{12}}$ as compared to the other useful ones. [3]

This theorem as in Equation 3.14 gives a meaningful definition to another algorithm known as Mini-Sync which is proposed to be an extension to the Tiny-Sync algorithm. Upon receipt of each new data point, the constraints imposed by the received data point will be compared with respect to the already available constraints to decide whether these are more useful than the others or not and accordingly will replace an existing constraint or will be discarded. It is also possible that multiple existing constraints are discarded on arrival of a new data point. Only the relevant data points that are stored will be used to obtain the desired estimates on the relative drift and offset. The main idea behind Mini-Sync is to obtain an optimal solution which the tiny sync fails to achieve. [2][3]

Moreover, while Tiny-Sync is limited to storing only four data points at a time due to the energy and resource limitations of sensor networks, yet it failed to obtain an optimal solution for effectively synchronizing time between the two nodes. Mini sync, on the other hand proceeds to deliver an optimal solution by storing only relevant (potentially useful) data points and discarding the inappropriate constraints (conditions). Thus, we still obtain an accurate solution with only a limited number of data points. [2] [3]

Note: For more details on the consistency of data points to be potentially useful or not, refer to [12]

In case it is required to determine the number of data points permissible to be stored at a particular instant so as to obtain precise estimates (optimal solution), then this should be a potentially large number theoretically. Moreover, if it is assumed that the intermediate delays between the nodes 1 and 2 are increasing monotonically then Equation 3.13 may be valid for all of the available constraints A_j. But practically this is not feasible since the delays are never constant and are not always monotonically increasing. The clocks are continuously drifting apart from each other (whether positively or negatively). Therefore, only a limited number of the several constraints need to stored so as to obtain optimal bounds. [2][3]

3.6.4 Performance analysis of Tiny-Sync and Mini-Sync in synchronizing an entire network

Several factors such as Round Trip Time (RTT), probing duration, etc effect the functioning of the Tiny-Sync algorithm. [3]

Equation 3.15 [3] assuming Round Trip Time to be defined as the constant difference between t_o and t_r is given as

$$t_{r_i} - t_{o_i} = RTT \qquad \forall i \geq 1 \tag{3.15}$$

All unknown delays within the system from the instant the probe is sent to when it is finally received are constant and accounted for by the Round Trip Time as in Equation 3.15 . [3]

Moving on, the assumption as in Equation 3.15 is justified by means of the following considerations:

1. The basis for the aforementioned assumption shown in Equation 3.15 being that Tiny-Sync only considers the two most appropriate data points at a time for defining constraints on the uncertainties pertaining to the relative drift and offset. Smaller round trip times equivalent to the minimum Round Trip Time of the connection are usually preferable with regard to the data points considered so as to result in as much tighter bounds as possible.

2. Also noticeable is that only slight variations (in microseconds) occur in the round-trip delays in contrast to the sampling intervals (upto tens of seconds).

3. Moreover, these theoretical interpretations/results have already been proven valid with the help of well-defined experimental procedures in coherence with the desired results. [3]

As per the assumption in Equation 3.15, only the first and last data points collected are usually taken into consideration while the others are discarded.

Figure 3.5 on the next page shows that Tiny-Sync performs as efficiently as Mini-Sync. Results have been interpreted using the first and the last data points collected being (t_{o_1}, t_{b_1}, t_{r_1}) and (t_{o_2} , t_{b_2} , t_{r_2}) respectively.[3]

From Figure 3.5, certain equations have been derived which are self-explanatory.

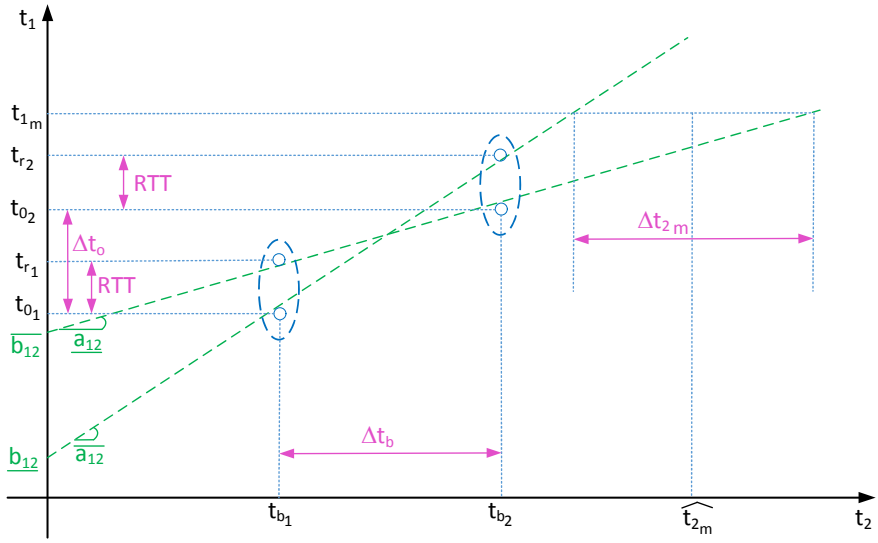

Figure 3.5: *Simplified analysis of Tiny-Sync algorithm*
Based on a figure in [3]

Equation 3.16 [3] considers the difference between the first coordinates of each of the two data points considered as in Figure 3.5, these being t_{o_1} and t_{o_2}, and is expressed as

$$\Delta t_o = t_{o_2} - t_{o_1} \tag{3.16}$$

Similarly, Equation 3.17 [3] considers the difference in between the second coordinates of each of the two data points considered as in Figure 3.5, these being t_{b_1} and t_{b_2}, and is expressed as

$$\Delta t_b = t_{b_2} - t_{b_1} \tag{3.17}$$

The Equation 3.18 [3] gives the lower bound on the relative drift as

$$\underline{a_{12}} = \frac{\Delta t_o - RTT}{\Delta t_b} \tag{3.18}$$

and

The Equation 3.19 [3] gives the upper bound on the relative drift as

$$\overline{a_{12}} = \frac{\Delta t_o + RTT}{\Delta t_b} \tag{3.19}$$

Similarly, Equation 3.20 [3] gives the lower bound on relative offset as

$$\underline{b_{12}} = t_{o_1} - t_{b_1}\overline{a_{12}} \tag{3.20}$$

and

Equation 3.21 [3] gives the upper bound on relative offset as

$$\overline{b_{12}} = t_{o_1} + RTT - t_{b_1}\underline{a_{12}} \tag{3.21}$$

The key factors determining the performance of the two algorithms explained are the uncertainty bounds on the relative drift and offset.

The Figure 3.6 on the facing page illustrates the change in the uncertainty bound as more data samples are collected. [3] It is seen that as the distance between the first and the last data points collected increases, the uncertainty bound on the drift decreases, thus resulting in higher precision. [3]

This is also shown in the form of Equation 3.22 [3] as

$$\Delta a_{12} = \frac{2RTT}{\Delta t_b} \tag{3.22}$$

Henceforth, Equation 3.23 [3] relates the precision on the relative drift via the uncertainty bound Δa_{12} as illustrated in Figure 3.6 on the next page as

$$\lim_{\Delta t_b \to \infty} \Delta a_{12} = 0 \tag{3.23}$$

It is evident from Figure 3.6 that the uncertainty bound remains constant within a set of data points and improves upon the collection of a new data point sample. [3] Practically, precision of the order of $2RTT/T_0$ is feasible to be achieved since the assumption in Equation 3.1 remains valid upto a limited time T_0. [3]

Further, Equation 3.24 [3] shows that the uncertainty bound on the relative offset is also affected by the Round Trip Time as

$$\Delta b_{12} = RTT + t_{b_1}\Delta a_{12} \tag{3.24}$$

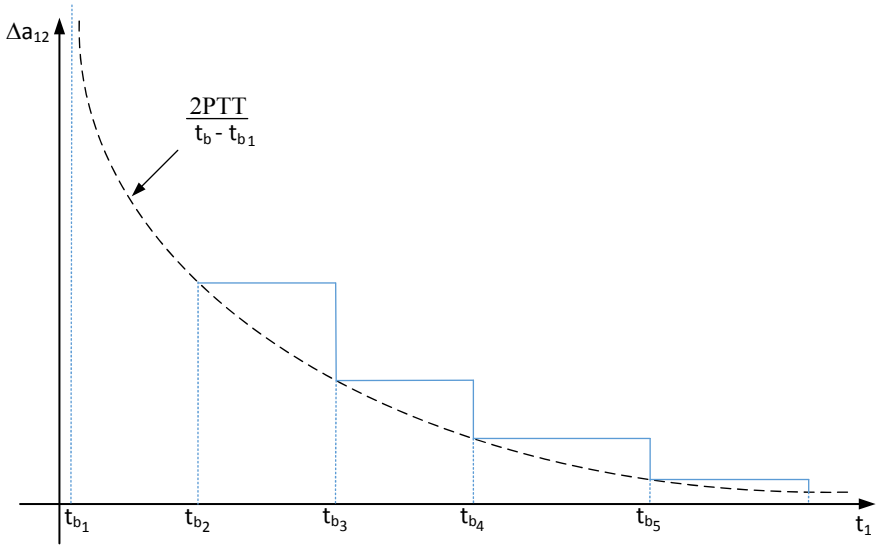

Figure 3.6: *Effect on the uncertainty bound on the relative clock drifts Δa_{12} as new data-points are received*
Based on a figure in [3]

The Equation 3.24 clearly shows that the uncertainty in the relative offset (Δb_{12}) exceeds the minimum value of Round Trip Time by a small amount. For the reduction of the Round Trip Time factor so as to account for the random delay component only, either all the timestamping needs to be done at the MAC layer or an estimate of the lower bound on the Round Trip Time delay factor be known. Furthermore, this leads to improvement in the order of precision attained using this algorithm. [3]

Next, Equation 3.25 [3] relates the precision on the relative offset via the uncertainty bound Δb_{12} as

$$\lim_{\Delta t_b \to \infty} \Delta b_{12} = RTT \tag{3.25}$$

3.6.5 Enforcement of the assumption pertaining to the linearity of the clock drifts

[3] As explained earlier about the continuous drift and offset variations in the clocks owing to the rapid fluctuations in clock frequency over time, the assumption on the linear nature of the clock drifts as in Equation 3.1 on page 19 seems to be impractical. This is because running clocks are continually effected by external factors like applied voltage, temperature, humidity, etc. So even if it is taken for granted that the relation

as in Equation 3.1 on page 19 holds true, yet considering these clocks to linearly drift apart with respect to each other will logically hold only for a limited time frame. Once the desired operating conditions are violated, Tiny- Sync fails to achieve synchronization between the nodes.

The linearity relation also relates to the decrease in the uncertainty bound for the relative drift with the order of $2RTT/\Delta t_b$ as shown in Equation 3.22 on page 34. Furthermore, the moment the clocks start diverging to a non-linear behavior in their drift with respect to time, Δa_{12} starts decreasing at a faster rate than the $2RTT/\Delta t_b$ factor. Also in some cases it may reach a negative value if the underlying factors behind the non-linear behavior are not accounted for. This phenomenon has been explained. [3]

Considering the possible constraints imposed by the data points as a cylindrical pipe with two lines defining the upper and the lower bounds on the relative drift and the relative offset as thin rods through the diagonals, the pipe upon being bent gets the two rods to a parallel position henceforth changing the relative drift in a way that the Δa_{12} factor is reduced in relation to the Δa_{12} of a perfectly straight pipe. [3]

This requires Δa_{12} to be computed periodically after each message probe is sent and the acknowledgement has been received back at the initiating node and is subsequently compared to $2RTT/\Delta t_b$ as in Equation 3.22 where Δt_b as expressed in Equation 3.17 is the difference between the timestamps attained at the send and receive points at node 2. [3]

Equation 3.26 [3] shows that a new constraint will be added as a replacement to the oldest one if and only if

$$\Delta a_{12} < \frac{2RTT}{\Delta t_b} \tag{3.26}$$

Therefore, it is quite evident that the functionality of Tiny-Sync is totally incumbent upon the linearity assumption and if this no longer holds then the algorithm fails at that point and the only possible solution is to replay the algorithm. [3]

3.7 Achieving synchronization between a set of network nodes

Tiny-Sync and Mini-Sync were used to achieve synchronization between two neighboring sensor nodes, node 1 and node 2. There have been attempts to extend this possibility to synchronize the entire network and not just neighboring nodes. This

would involve transmitting data in multiple hops as well as simultaneous coherence between 'n' nodes in a network. For this, the pair-wise synchronization [13] scheme was deployed. [3]

As stated earlier, Wireless Sensor Networks comprise of a huge density of nodes interconnected to each other for exchange of information. While synchronizing two neighboring nodes, there were a limited set of data points that were taken into account so as to give an optimal solution. But when it comes to achieving synchronization between neighboring as well as distant nodes in a network, then a large number of data samples are to be obtained.

Figure 3.7 explains how the transitivity relation applies while synchronizing between three nodes s, u and v in the network. This implies that if s synchronizes with u and u synchronizes to v, then s and v are also mutually synchronized. [2][3]

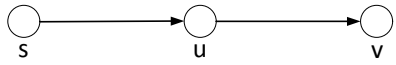

Figure 3.7: *Synchronization transitivity: if s is synchronized with u and u is synchronized with v, then s is synchronized with v*
Based on a figure in [2]

This form of synchronization as in Figure 3.7 takes place in a number of hops in between the nodes in the network and is henceforth termed as multi-hop synchronization. [2] [3] The efficiency of the entire process again relies on the generation of optimally bounded estimates on the relative drift and the relative offset. This process is explained further.

With s being synchronized to u, s is capable of determining the desired bounds on drift and offset relative to u.

Equation 3.27 [2] [3] shows that s determines the bounds on the drift relative to u as

$$\underline{a_{su}} \leq a_{su} \leq \overline{a_{su}} \tag{3.27}$$

and

Equation 3.28 [2] [3] shows that s determines its bounds on the offset with respect to u as

$$\underline{b_{su}} \leq b_{su} \leq \overline{b_{su}} \tag{3.28}$$

Similarly, with u being synchronized to v, u also determines the bounds relative to v.

Equation 3.29 [2] [3] shows that u determines the bounds on the drift relative to v as

$$\underline{a_{uv}} \leq a_{uv} \leq \overline{a_{uv}} \tag{3.29}$$

and

Equation 3.30 [2] [3] shows that u determines the bounds on the offset relative to v as

$$\underline{b_{uv}} \leq b_{uv} \leq \overline{b_{uv}} \tag{3.30}$$

Since s and v are not immediate neighbors and are separated by more than one hop as shown in Figure 3.7, therefore it is desired from u to convey its bounds $\underline{a_{uv}}$, $\overline{a_{uv}}$, $\underline{b_{uv}}$, and $\overline{b_{uv}}$ to s so that s synchronizes with v with the desired bounds on a_{sv} and b_{sv}. [2] [3]

Equation 3.31 [2] [3] gives the bounds on the relative drift with respect to s and v as

$$\underline{a_{sv}} \leq a_{sv} \leq \overline{a_{sv}} \tag{3.31}$$

and

Equation 3.32 [2] [3] gives the bounds on the relative offset with respect to s and v as

$$\underline{b_{sv}} \leq b_{sv} \leq \overline{b_{sv}} \tag{3.32}$$

Equation 3.33 [2] [3] gives the lower bound on the relative drift with respect to s and v as used in Equation 3.31.

$$\underline{a_{sv}} = \underline{a_{su}a_{uv}} \tag{3.33}$$

and

Equation 3.34 [2] [3] gives the upper bound on the relative drift with respect to s and v as used in Equation 3.31.

$$\overline{a_{sv}} = \overline{a_{su}a_{uv}} \tag{3.34}$$

Similarly, Equation 3.35 [2] [3] gives the lower bound on the relative offset with respect to s and v as used in Equation 3.32 on the facing page.

$$\underline{b_{sv}} = \min\{\underline{a_{su}b_{uv}} + \underline{b_{su}}, \overline{a_{su}}\underline{b_{uv}} + \underline{b_{su}}\} \tag{3.35}$$

and

Equation 3.36 [2] [3] gives the upper bound on the relative offset with respect to s and v as used in Equation 3.32 on the facing page.

$$\overline{b_{sv}} = \max\{\overline{a_{su}b_{uv}} + \overline{b_{su}}, \underline{a_{su}}\overline{b_{uv}} + \overline{b_{su}}\} \tag{3.36}$$

Additionally, some generalized expressions have been derived for k nodes in a chain. [3]

Equation 3.37 [3] gives a generalized expression for relating the clocks at nodes 1 and k as

$$t_1 = \prod_{i=1}^{k-1} a_{i(i+1)} t_k + \sum_{i=1}^{k-1} \left\{ \prod_{j=1}^{i-1} \left(a_{i(i+1)}\right) b_{i(i+1)} \right\} \tag{3.37}$$

The corresponding lower and upper bounds on the a_{1k} and b_{1k} are also derived.

Equation 3.38 [3] gives the lower bound on the relative drift with respect to the first node and the kth node as

$$\underline{a_{1k}} = \prod_{i=1}^{k-1} \underline{a_{i(i+1)}} \tag{3.38}$$

and

Equation 3.39 [3] gives the upper bound on the relative drift with respect to the first node and the kth node as

$$\overline{a_{1k}} = \prod_{i=1}^{k-1} \overline{a_{i(i+1)}} \tag{3.39}$$

Similarly, Equation 3.40 [3] gives the lower bound on the relative offset with respect to the first node and the kth node as

$$\underline{b_{1k}} = \begin{array}{c} min \\ a_{i(i+1)} \in \{\underline{a_{i(i+1)}}, \overline{a_{i(i+1)}}\} \\ b_{i(i+1)} \in \{\underline{b_{i(i+1)}}, \overline{b_{i(i+1)}}\} \end{array} \left\{ \sum_{i=1}^{k-1} \left\{ \prod_{j=1}^{i-1} \left(a_{i(i+1)}\right) b_{i(i+1)} \right\} \right\} \tag{3.40}$$

and

Equation 3.41 [3] gives the upper bound on the relative drift with respect to the first node and the kth node as

$$\overline{b_{1k}} = \begin{array}{c} max \\ a_{i(i+1)} \in \{\underline{a_{i(i+1)}}, \overline{a_{i(i+1)}}\} \\ b_{i(i+1)} \in \{\underline{b_{i(i+1)}}, \overline{b_{i(i+1)}}\} \end{array} \left\{ \sum_{i=1}^{k-1} \left\{ \prod_{j=1}^{i-1} \left(a_{i(i+1)}\right) b_{i(i+1)} \right\} \right\} \tag{3.41}$$

[3]

Note: It can be seen that the lower as well as the upper bounds for the relative drift with regard to the ith and $i+1$th nodes tend to approach unity. [3] Moreover, Equation 3.40 and Equation 3.41 show that there is linear degradation in the uncertainty bound for the relative offset as more number of hops are added. [3]

3.8 Sampling Synchronization in Gigabit Ethernet

The evolution of Ethernet can be well interpreted to the time when Asynchronous Transfer Mode (ATM) was the most sort out technology for Sonar and similar synchronous sampling systems. This stemmed from the fact that the entire network infrastructure relied on a synchronous backbone which accounted for all the timing information within the system. Although initially Ethernet lacked a support infrastructure for synchronous operation yet with the rapid advancements in technology

over time, several open standards like Synchronous Ethernet, IEEE 1588, etc have evolved addressing the importance of synchronicity. Moreover, the proposed QNA-TSG scheme allows for using Gigabit Ethernet for synchronous sampling as well as sampling using GPS locking too. [10]

Asynchronous Transfer Mode (ATM) being practically implemented in telecommunication networks uses the switching technique to perform asynchronous time division multiplexing to generate small and fixed sized cells by encoding the data. This is unlike Ethernet or traditional Internet which utilize data frames with variable packet sizes. [14]

Furthermore, ATM is deployed with Synchronous Optical Network (SONET) as the core protocol standard functioning as the backbone of the Integrated Digital Services Network (ISDN). This type of deployment enables ATM to route the switched data packets over the synchronous backbone with the desired infrastructure along with the necessary signaling for synchronization purpose. [10]

ATM allows for easy switching by the hardware due to the fixed cellular structure henceforth avoiding the delays incurred during software switching and frame routing. [14]

Furthermore, ATM technology has a significant contribution to providing a fundamental infrastructure hereby paving the way for diversified network topologies. Although ATM involves asynchronous loading of information into the data packets yet the transport layer functions in a synchronous manner. This, in turn, facilitates both the high rate as well as the low rate sample clock distribution via the synchronous physical layer backbone. Moreover, efficient clock recovery within the network is possible through network switches. This makes it possible for the clock recovered at any port to be used to drive other clock outputs to a different port. While this method for clock recovery is inefficient as it adds to the overall cost since the Phase locked Loops (PLLs) [15] and the clock switches are costly but the good part of this technology is that a single unique clock derived from the master can be distributed throughout the network. [10]

ATM has become a less favorable standard over time being replaced by Ethernet. This is mainly because the underlying framework provided by Ethernet for the transport of data is cheaper as compared to ATM and that Ethernet resists the overhead which, in contrast, constraints the functioning in ATM. This makes Ethernet a more preferable standard for shorter range communication ($<$100 km) as well as for comparatively low data rates ($<$1 Gbps). Therefore, while Ethernet is continually progressing for technological advancements in small scale networking topologies, on the other hand, ATM has been isolated only for applications requiring higher data rates. [10]

Meanwhile, many integration possibilities are being explored to bring the advantages of Synchronous Optical Network (SONET) like clock port mapping merged into the physical layer of Ethernet. Also, two systems connected via an Ethernet switch are now able to mutually share a common clock with the functionality of a Boundary clock making them analogous to signals being transmitted through an independent wire without constraining the overall solution within the network boundaries. [10]

As discussed earlier, when it comes to synchronizing time between the network nodes, it is very important that there is a common clock source as a reference to all the nodes in the system and an event as a synchronization event (starting event) acting as a trigger. The synchronization event is characterized by the moment when a packet carrying data to a destination has been transmitted from the source where the first timestamp is taken while timestamping real time events during the ongoing transmissions and the receptions. [10]

The goal here is achieving a defined synchronization level typically utilized in array processing applications via adequate phase locking via Phase Locked Loop (PLL) [15] at the maximum feasible operating frequency. [10]

For all the nodes to report time on a universal timescale, either a crystal oscillator with high accuracy has to be made accessible to all internal devices (nodes) in the network considering the drift, offset, operating temperature, frequency tolerance, etc or that all nodes need to correlate to one single clock source. [10] For instance, considering 1 ppm accuracy for an oscillator during the entire time interval of the day, there could be upto 86.4 milliseconds offset between the two nodes counting time with regard to each other. Moreover, a time difference of 31.5 seconds may not be very significant for a wrist watch but for beamforming and other related applications, a clock having 1 ppm tolerance may effectively generate a significant bit error rate (out of every million bits, one bit isn't present when expected). [10]

3.9 Constraints on the current technologies

Apart from NTP protocol having been discussed earlier, there is another protocol to be studied in this regard, namely, Precision Time Protocol (PTP).

3.9.1 Precision Time Protocol (PTP)

Just like NTP, Precision Time Protocol (PTP) also fails to account for the fact that the server clock is not accessible to the clients. Henceforth failure in maintaining a unique timescale throughout results in abnormal behavior with respect to the clock drifts, overpowering the goal of effectively synchronizing the network. Although PTP initially provided a network agnostic, appreciable and generic way for attaining

high degree of synchronization yet the current implementations lack the flexibility to generate real world results due to being limited to small bit rates upto 100 Mbps which is unsuitable for a number of applications including real time sensing applications. [10]

In contrast to this, the QNA scheme relying on Gigabit Ethernet supports systems with high data rates exceeding 350 Mb/s, henceforth managing the expected degree of synchronous behavior. [10]

3.9.2 Synchronous Ethernet

Being a new specification, this standard serving as an extension to the earlier discussed PTP protocol attempts to address the possibility of the distribution of a synchronous clock throughout the system with the help of an underlying physical layer. [10]

Chapter 4
Tools

4.1 Development Tools

There is a huge list of development tools, both software as well as hardware, that have been used during the thesis work at Nordic Semiconductor ASA. Due to different tools showing different interactive behaviors, accordingly newer or older versions of these tools had been utilized so as to perform the necessary tasks under consideration.

4.1.1 Hardware Tools

4.1.1.1 nRF51 Development Kit

The main feature supported by the nRF51 Development Kit (nRF51 DK) [16] [17] is its ability to allow for the simultaneous transmissions as well as the receptions of Bluetooth signals made by Nordic Semiconductor, henceforth facilitating the development process of effective software solutions used further for testing purposes. The nRF51 DK, exhibiting a cross-platform efficiency supports several different Integrated development environments (IDEs) with focus here mainly on two, namely, Keil5 [18] and SEGGER Embedded Studio [19]. While Keil5 [18] had been mainly used for testing and debugging of some of the existing applications of Nordic Semiconductor yet the main code for the embedded software solution was designed using SEGGER Embedded Studio.

4.1.1.2 nRF51 Dongle

[20] Yet another interesting product of the Nordic Semiconductor nRF51 tool suite. The nRF51 Dongle finds wide variety of applications including development of varied software solutions for Nordic Semiconductor using Keil, SEGGER Studio, etc. In the thesis work, this product was mainly used in conjunction with the nRF Sniffer [16] software along with Wireshark [21] tool for sniffing Bluetooth Low Energy (BLE) packets on air. This involved capturing and recording the corresponding receptions of the BLE packets on Radio

– Refer to Chapter 17 (2.4 GHz Radio (RADIO)) in [5].

Note: Furthermore, the nRF51 Dongles prove their usefulness for testing purposes via an in depth analysis of the ongoing Bluetooth traffic within the network. In this manner, capturing and recording the data traffic in context of the signal strength indicator (RSSI factor) makes it easy to eliminate the unnecessary Bluetooth traffic in the within the network to avoid congestion as well as to ensure reliable data delivery. [22].

All in all, the nRF51 Dongle proves to be quite handy in interpreting the degree of reliability of the obtained results with respect to the expected results.

4.1.1.3 Logic Analyzer

One of the key tools used for testing purposes. This helped in capturing the data from the nRF51 DK along with the Logic software. The data captured were the signals obtained on the nRF51 board which were analyzed to see whether these were behaving in concordance to the expectations. With the toggling of the system output pins from high to low, and vice versa and connecting the Logic Analyzer hardware tool to any of these pin(s) labelled from 21-24 on the nRF51 board, that is, connecting to a single pin at a time or many at once, the Radio events are precisely generated on a timeline. [22]

Note: All these events occur in the same sequence as defined in the main code shown in Appendix A.1 under test. [22]

[22] In addition to this, a python script was generated for interpreting the data from the Logic Analyzer and yielding a meaningful feedback on the behaviour of the IC. Finally, the Python Saleae Interface helps to examine this interpreted data onto an excel sheet as shown in Figure 5.2 on page 60 and Figure 5.6 on page 64.

4.1.2 Software Tools

4.1.2.1 nRF51 SDK

nRF51 Software Development Kit (SDK) is the standard Software Development Kit for the nRF51 devices. It offers a flexible and well-tested environment for developing the software for the nRF51 series. It serves as a pillar for the designed software solution for the thesis work, being used in multiple tasks and tests in the Software Development Life Cycle (SDLC). Adding on, the nRF51 SDK serves as a storehouse for some already developed Nordic Semiconductor applications, as well as for the essential drivers and libraries responsible for further learning and testing procedures. [23][22]

4.1.2.2 nRF5x-Command-Line-Tool

The nRF5x-Command-Line-Tool [16] [24] is used while working with the nRF51 boards. Being directly implemented from the command line, this tool from Nordic Semiconductor allows for simplified erasing of any current software running on the board as well as for flashing new software and running it along with many other standard functions used for interacting with the selected hardware. [24][22]

4.1.2.3 GNU ARM Embedded Toolchain

"Open source toolchain that allows one to compile and build C, C++ and assembly files."[22] This helped in building executable object files from the given source code. Having cross-platform efficiency and being an open source platform enabled the builds to be far more reproducible than expected. [25] [22]

4.1.2.4 GNU ARM Eclipse Windows Build Tools

The GNU ARM Eclipse Windows Build Tools involved the usage of the make command in the terminal. The make command is necessary for building projects using Makefiles. Furthermore, this set of tools allowed for simplified functionality of the shell commands including make, mkdir, rm, etc that were required for the Makefiles used within the entire project. [26][22]

4.1.2.5 Segger J-Link

This is a debugging tool that has been used for flashing the nRF51 DK. The jlink version used was $5.10n$ so as to ensure synchronous functioning with other tools. [27][22] This even involves a J-Link Real Time Terminal (RTT) Viewer [28] which was used for debugging the C code in real time once the program had been successfully built and loaded in SEGGER Embedded Studio Integrated Development Environment (IDE).

4.1.2.6 Keil μVision

An IDE that is used to compile and load applications onto the nRF51 DK. Keil5 [18] was mainly used for early training in C programming. Later it was replaced with SEGGER Embedded Studio. [18]

Although Keil5 was good for the initial practice sessions yet switching to Embedded Segger Studio was inevitable due to some significant limitations incurred while working with Keil5.

4.1.2.7 Termite

A simplified RS232 terminal using an interface similar to messenger or chat programs, connecting to the COM port that the nRF51 DK is connected to so that it can interact with it, print out text from it and receive text over Bluetooth among other things. [29]

4.1.2.8 Debugger

Just like the Logic Analyzer and the nRF51 Dongle, this software was yet another testing tool provided as an inbuilt functionality within SEGGER Embedded Studio IDE. Since *C* programming is known to be very error prone, henceforth this tool served a significant purpose at different stages in the software development process.[22]

Furthermore, certain sections in the main code as shown in Appendix A.1 required access to specific defined variables or those yet needed to be defined so as to get the desired functionality. This, in turn, required an analysis of the allocated Random Access Memory (RAM) memory as defined in Appendix A.5 and Appendix A.6 or examining the desired call stack while even setting break points to verify whether or not the program was executing showing a normalized behaviour.[22]

4.1.2.9 nRF Toolbox app

A container app developed by Nordic Semiconductor storing other different apps. It has been mainly used in the project to connect to the nRF51 boards and test out how different software is working with these. As a multi-purpose smart phone application, it offers a specialized Universal Asynchronous Receiver/Transmitter (UART) function which served a key purpose in using Bluetooth technology for detecting and displaying the nearby devices using Bluetooth Connectivity. [30] [22]

4.1.2.10 Wireshark

Operating as an open source protocol analyzer for networks, Wireshark tool [21] allows access to the packets being transmitted on air in real time or those being sent through a wire/cable. It has been often used in conjunction with the nRF Sniffer software flashed onto the nRF51 board to generate a list of intercepted packets on air (using Radio as specific to this project)in real time. This further helped to get a deeper insight into the packet formation, the internal structure of these transmitted packets as well as the type of packets being captured. [21][22]

4.1.2.11 SEGGER Embedded Studio

[19] "A Streamlined and Powerful C/C++ IDE". This is another major tool used in the project. It is used for developing embedded applications like the *C* code designed

for the thesis work using this IDE as shown in Appendix A.1. It contains a lot of options for debugging and setup of the software that is developed.

There were certain challenging tasks in the process which required importing of some relevant Nordic Semiconductor projects in the nRF5_SDK_12.2.SES file folder (manually installed in the system) from Keil to SEGGER Studio for which a specialized procedure was defined as illustrated in [31]. This, in turn, helped to get a clear picture of the varied options available as well as the user interface provided in using SEGGER Embedded Studio for software development. Moreover, for achieving this, a Makefile-based build environment served as the source to port to a Segger project." [22][19]

4.1.2.12 Git

Git is a revision control system for handling the project's source code history. Using Git, many people can easily work together on the same project keeping track of the changes in any files and constantly keeping everyone updated on work being done. [32][22]

4.1.2.13 GitHub

GitHub [33] is basically used for hosting purposes for Git repositories so as to administrate the entire project. It allows for easy reviewing of the work that has been done on a specific project, inspection of the revision histories for the project as well as modifications done via a web interface in a user-friendly manner. [22][33]

4.1.2.14 Logic

Software used for capturing of the data along with the Logic Analyzer hardware tool. It enables one to see how the packets from the nRF51 boards are sent and figure out if they are working like they should. Combined with Python Saleae interface this data can be put into Excel sheets for easy access in the future. [34][22]

4.1.2.15 Python

[35] Python has been mainly used for scripting purposes in testing the product functionality. Choosing Python for scripting owes to the ease of use and flexibility offered as compared to other advanced scripting languages. Furthermore, in order to get a better overview of the data that was being transmitted in the packets, the Python Saleae Module turned out to be ideal. With this interface, the data from the Logic Analyzer was converted into excel sheets which were flexibly sorted into orderly and easily readable sheets.[35]

[36] [37] Then there was the nRF5x-pynrfjprog Python interface. This is a simplified interface meant for use by the nrfjprog DLL. It works with Python versions v2.7.x and v3.4.x, and later. This utility serves as an extremely useful tool for testing the nRF5x applications being designed using simple Python interfacing in order to tranfer the data from the nRF51 board onto the external PC. nRF5x-pynrfjprog offers great flexibility for testing all nRF SOCs (System on Chip) as well as other personalized applications.

4.1.3 Other Tools

4.1.3.1 Communication

Effective communication was key to ensure a synchronous flow of the work being during the thesis. There were instant communication channels accessible to me as well as the entire Nordic Semiconductor staff in the software department. Different stages of the project had access to varied communication portals for meeting the desired milestones.

4.1.3.2 Email

Another significant and yet immediate tool available while working on the thesis was the Nordic semiconductor Mail Outlook wherein any sort of support pertaining to working on the desired platform, technical issues faced and other common issues were addressed or scheduled to be addressed via online chat/call or physical meetings.

This turned out to be fruitful in overcoming the series of obstacles incurred during the thesis work henceforth making it possible to successfully achieve the pre-planned goals.

4.1.3.3 Nordic Developer Zone

Alongside Outlook serving as an effective communication channel, another major communication medium was the Nordic Developer Zone accessed via [38].

This was the central platform for all types of queries ranging from guide for using/accessing the different tools as well as any technical issues faced with the tools. All registered users had access to the collaborative support from the entire Nordic Semiconductor Community on day to day hot topics pertaining to the company's already developed ready to use applications as well as help on developing new applications from scratch, etc. All the users were expected to contribute actively on this platform as well as share their knowledge in this regard while working with the Nordic systems. This, in turn, unified the entire Nordic group to progress collectively in strengthening the company's overall reputation.

4.1.3.4 Facebook

The initial ramp-up sessions at Nordic Semiconductor ASA played a significant role in working further independently on this project.The knowledge gained from these training sessions on the company's business as well as their tools/devices under the supervision of David Devasahayam Edwin (Sr. Software Architect at Nordic Semiconductor) as well as with the mutual interaction within the teams through social media like Facebook, etc served as a great source of motivation for me to take up this project for my Master's thesis work at Nordic.

4.1.3.5 Freedcamp

Not to forget, there was Freedcamp that played a crucial role during the ramp-up sessions. Since these sessions required collaborative work in a team of 4-5 members from different backgrounds, henceforth Freedcamp was the unique platform for managing all the tasks planned to be executed during the training. All the tasks to be performed were listed in order along with some description as a guide to perform that work and then there were deadlines set for each task so as to ensure effective time management and to the acquire the desired knowledge. There was also a discussion section where members could share their views to walk through the subsequent milestones step by step.

4.1.3.6 Doodle

Again for the initial training sessions at Nordic Semiconductor ASA prior to the thesis work, the team used Doodle to schedule meetings among themselves and with the Nordic Semiconductor staff. With Doodle there was an accurate overview of each member's availability making it easier to plan a collective meeting.

4.1.3.7 Way of working

The entire thesis work period at Nordic Semiconductor required a deep understanding of the behavior of varied embedded systems. The soul of this thesis work is the embedded software solution as in Appendix A.1 that I have designed in SEGGER Embedded Studio using the SDK version 12.2 and other bunch of tools from the Nordic Semiconductor tool suite. For coping up with the day to day milestones to be achieved, I had to continuously refer to the tool manuals as well as the standard product specifications. This was due to the fact that developing an optimized software solution required a set of functionalities which could only be served via well-defined modules, specialized tools, etc.

4.1.3.8 Documentation

Throughout the training period, the team has exercised a certain way of working with embedded systems which relies mostly on looking up information in manuals and specifications.

The following documentation was used for the training sessions and the subsequent Master's thesis work.

1. nRF51 Reference Manual [5]
2. Bluetooth Core v4.2 specification [39]
3. nRF51422 Product Specification [40]
4. nRF51422 Product Anomaly Notification [41]
5. nRF51422 Product Change Notification [42]

Out of these,nRF51422 Product Anomaly Notification [41] and nRF51422 Product Change Notification [42] helped in investigating any known aberrations/bugs with the standard workarounds known to be aware of. Since the thesis work involved a series of milestones to be achieved, it was very important to refer to these sources so as to keep in constant touch with the desired functionalities such as using specific inbuilt modules and other necessary hardware as well as how to utilize them, etc.

Chapter 5
Methodology

5.1 Training

5.1.1 Introduction

Apart from an in depth research on the theory behind time synchronization across network nodes, the key focus in the thesis has been onto the practical analysis of the proposed algorithms in this regard, providing adequate logical reasoning for the observations, verification of the obtained results and deriving meaningful results/conclusions from them. This mostly relied on the degree of understanding of the software systems, both Nordic specific and General Purpose Software, as well as other hardware specific requirements.

In order to achieve this, learning and getting familiar with a wide range of tools, how to use the desired tools in the Software Development Life Cycle (SDLC) using different IDEs, etc had been significant. This was facilitated by the initial training/ramp-up sessions. The pre-planned and the pre-scheduled training sessions contained a series of interactive preparatory courses. These enabled me to get a better exposure to the skills I would need to master so as to proceed further with my thesis work. Moreover, since I was in a team of 4-5 members in the training, there was better understanding due to the shared knowledge as well as active participation from all. All the tasks in the initial sessions were interlinked so as to establish an organized structure as to what all is brought to application at different stages of the work. This, in turn, required the team to work on the tasks collectively. Once a task had been successfully completed, the team could progress forward moving to the next activity.

While the initial tasks helped in gaining familiarity with the setup of the required tools by following certain tutorials posted on the Nordic Developer Zone, the subsequent tasks discussed further served as a key source to realize the behavior of some of the already developed software applications of Nordic Semiconductor.

5.1.2 Scan response data

The first exercise was based on analyzing the ble_app_uart application in the Nordic Semiconductor nRF51 SDK examples. This task involved a Bluetooth device in the advertising mode sending packets on air with the goal to communicate data to the nearby devices. The nearby devices having detected this advertising device send scan request packets to it so as to attain further information. On receiving a scan response packet from any of those nearby devices, this advertising device further sends a scan response packet with the desired information to be communicated. Although much of the initial code remained the same for this application yet some minor modifications needed to be done and that too in Keil5 [18] IDE. The task was to put the advertising device's name in the scan response packet instead of the advertised packet itself so as to have a minimal size for the advertised packet. [22]

Considering the initial stages, there wasn't much of additional code to be added or changed in this but it still took valuable amount of time to gain familiarity with the key functionalities defined in this application. [22]

5.1.3 Modifying beacon

Taking the things to a higher level, this exercise was somewhat similar to the previous one but not in all respects. This was another Nordic Semiconductor application, namely, ble_app_beacon. Unlike the previously discussed ble_app_uart application where we were advertising as well as sending scan response packets, this example program only advertises BLE packets but doesn't transmit scan response packets. Adding on, in this case, we have non-connectible packets being advertised henceforth acting as a beacon. So the main task at hand was to make these non-connectible packets to be advertised in a connectible manner and even send scan response packets and see the scan response. This was achieved using nRF Sniffer software and Wireshark tools. [22]

5.1.4 PPI

This exercise helped in gaining familiarity with the asynchronous mode of communication between the peripherals and the CPU of the ARM processor. There is a Programmable Peripheral Interconnect (PPI) module associated with the nRF chips which defines a set of tasks and events. The peripheral devices are composed of readable event registers which are modified upon the occurrence of the related events. Furthermore, there is also a flexible functionality to write to the task registers if a corresponding action is needed to be triggered within the peripheral. PPI operates with the support of a set of PPI channels as well as channel groups which also need to be enabled/disabled as per the desired functionality. These channels can be appropriately modeled so as to define an event that would be triggering a particular task.

The task to be performed here was timer initialization and using a periodic event via this initialized timer to be connected to the toggle activity of an Light-emitting diode (LED) (LED being just another General-purpose input/output (GPIO) pin) on the nRF51 DK from high to low states or vice versa.

This was a more advanced and time consuming task in contrast to the earlier performed tasks.

5.1.5 Radio

After having successfully completed the previous tasks, there was the Radio example in the SDK folder. The uniqueness of this task was that no softdevice was used here in contrast to the s130 soft device commonly being used in the earlier performed tasks. This softdevice [16] was a software functioning as a library placed together with the program in the system memory. One major requirement while loading the earlier example programs in Keil environment was to flash this soft device onto the board along with the necessary firmware each time so as to attain the desired results. This was done using the nrfjprog functionality in the Windows Command prompt. [22]

While the use of this softdevice in the previous cases facilitated the interfacing of the radio and the packets being transmitted via Bluetooth protocol, the Radio example utilized the nRF Radio functionality [5] along with the link-layer functionality with reference to the Standard Bluetooth specification (Core Version 5.0 in [43]). [22]

Furthermore, this task also required an in depth understanding of the Product Specification [40] as well as the Bluetooth specification (Core Version 5.0 in [43]) along with the Generic Access Profile (GAP) [44] too for digging deep into some additional details previously unknown, such as byte- and bit- endianness, clock source, etc. [5] [22]

The scan response and the advertisement packets in this case were sent with the nRF radio with access to the Radio Peripheral registers as listed in [5] in the Nordic Semiconductor Infocenter. [45] [22]

5.1.6 Risk analysis

As expected there were a series of challenges encountered at different stages during the thesis work that interfered with the fulfillment of the planned goals.

Following an organized approach, things were accordingly sorted out beforehand and as a backup, certain remedial actions were put forward to cope with the intermediate challenges probably to be encountered so as to keep the project flowing

smoothly and maintain the overall pace towards the successful completion of this Master's thesis. In order to achieve this, an in depth analysis was done with respect to several factors such as the type as well as the extent to which a problem could effect the development work, how frequently could the issue(s) arise, etc. Henceforth, strategic risk analysis/risk management and mitigation served as a boon to overcome such situations without additional stress.

5.1.7 Testing

One of the key phases in the development of the desired embedded software solution was the testing phase. This stage was quite significant to the success or the failure of the resulting solution with respect to the expected results. Most of the coding part was done in Embedded C and then python was used for interfacing along with intelligent decision making policies via algorithms in a mesh type network. Henceforth, designing such a heterogeneous system had not been an easy task at all. Since there were so many files/libraries to be maintained as well as timely updated as per the desired functionality, therefore, it was required to frequently perform efficient tests on the outer layer as well as dig deeper into testing the internal core part of the code.

Adding on, considering a single syntax mistake to be able to give a fatal error, designing the *C* code for this application in SEGGER Studio [19] required an astute observation and thorough understanding of the Computer Science/Information Technology fundamentals.

Tests were differentiated for the firmware and the software on the controllers. Although tools like Wireshark, Saleae Logic Analyzer had being used for verification purposes yet their interpretations were not so accurate to be relied upon in order to attain meaningful conclusions.

So this needed a thorough understanding of the Software Testing Life Cycle (STLC) [46]

5.1.7.1 What to test

The main scope of this project revolves around attaining precise timing calculations. This was achieved by debugging the *C* code using Segger RTT viewer so as to extract the time samples in real time from the program loaded onto the nRF51 board. Not to forget, the Saleae Logic Analyzer tool and the nRF Sniffer software tool alongside Wireshark had been used for verification of the results.

The main code as in Appendix A.1 used the Radio functionality with PPI over BLE stack to achieve synchronization between the network nodes. This required an

extremely accurate testability analysis for the designed solution since the time samples being interpreted between the varied Radio events were of the order of milliseconds and even microseconds. Once the main code had been built and thoroughly tested in Segger environment, it was possible to make some rough estimates on the time interval (delays) between the varied events and that when each event is bound to happen. Subsequently, the Logic was invoked for interpreting these events and verifying the estimated outcomes.

Furthermore, a prior state machine had also been designed which acted as a prototype for the developed *C* code as in Appendix A.1 so as to build the software as per the desired specification. Adding on, in case of any deviations in the obtained results with respect to the expected outcomes, this was a clear indication of some discrepancy in the code functionality. This could have been due to malfunctioning of some specific sections in the code pertaining to clocks, timers, radio, ppi, etc. Thus, the most reliable way for finding the error(s) in such cases with regard to a specific module functionality, or any other deviation from the norm was using the debugging option in SEGGER Studio which could be used either by setting breakpoints in the code or other available means.

The Radio functionality as explained in [5] defines unique events like READY, ADDRESS, PAYLOAD and END for the corresponding packet transmissions as well as receptions.

Figure 5.1 on the following page shows the BLE packet layout. All BLE packets used are bound to follow this standard format. [4]

Here the focus was mainly on advertising packets, and hence the type of layout as selected from Figure 5.1 was the ADV_IND packet layout.

The time measurements recorded on the Saleae Logic Analyzer software were mainly between the consecutive toggling events from high to low states or vice versa between the Radio events READY, ADDRESS, PAYLOAD and END. Having known that one bit is transmitted every second via BLE, if the address field was 40 bits, it could be inferred that there was an interval of 40 microseconds from the READY event when the Radio has been ramped up to the actual start of the transmission demarcated by the ADDRESS event.

Furthermore, with the test mentioned above being done at the initial stages of the testing lifecycle, it was evident that the estimate of 40 microseconds between the READY and ADDRESS events was an invalid prediction since the actual measurements lied within 38 and 42 microseconds. This variation was due to the use of a clock with lower frequency (4 MHz) just at the beginning of the transmission, which, as expected caused some unusual jitter (unstable frequency).

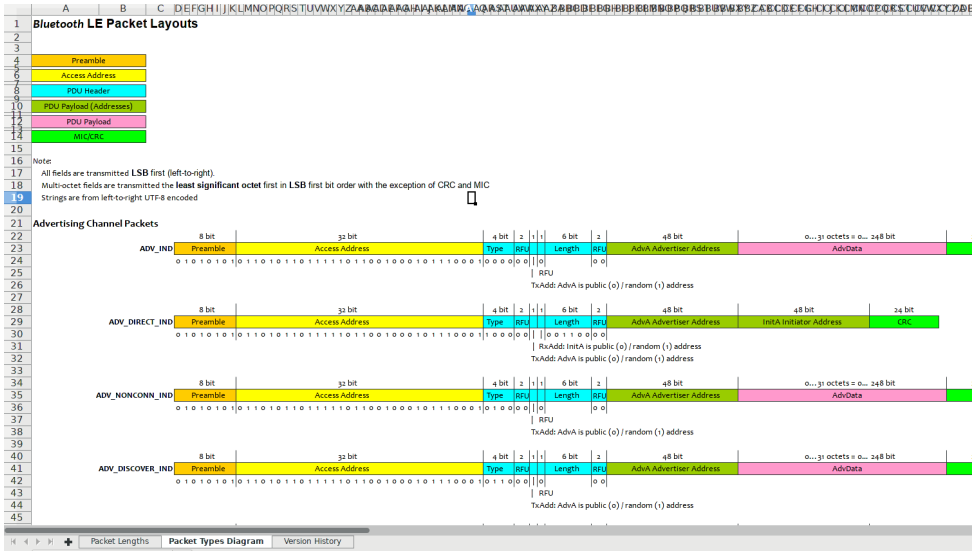

Figure 5.1: *Bluetooth LE Packet Layouts*
Based on a figure in [4]

Performing this test made things more clear for further testing purposes. It indicated the impracticality to perform good automation tests via inaccurate initial assumptions/hypothesis.

5.1.8 How the measurements were done

The main code designed for the thesis work as in Appendix A.1 offers a flexible functionality to connect events to tasks. In this manner, one can possibility connect any of the events among ADDRESS, PAYLOAD, etc to a specific task in the General-purpose input/output (GPIO). Referring to the Nordic Semiconductor nRF51 Reference Manual [5], the main code as in Appendix A.1 allows for each of the READY, ADDRESS, PAYLOAD and END events to be linked to a specific task in the GPIO whether toggle, set or clear. Since the PPI module defined in Appendix A.1 enables this functionality, the main advantage here is that there is less probability of any unnecessary delays to be incurred in the system due to the processing time, response time, etc since there is no CPU intervention in between. Henceforth, the PPI offering CPU independent functionality makes the time taken to transit from an event to a task to be almost negligible.

Furthermore, in order to verify these results on the Logic Analyzer, the grounds of both the Analyzer and the nRF51 board were connected to each other as well as the corresponding lines of the Logic Analyzer representing the four Radio events,

these being READY, ADDRESS, PAYLOAD and END be connected to the pins 21, 22, 23, and 24 respectively on the nRF51 board so as to interpret the desired events on the Logic Software.

This Logic software, pre-installed in the system much before the task, when made to run with the above mentioned setup, captured data from the connected lines but this was only limited to a specific set duration. In this manner, the data captured directly onto the Logic window yields an immediate interpretation of the reliability of the attained results with respect to the degree of closeness to the expected results.

Henceforth, a series of simultaneous tests were conducted following a well-structured approach in order to automate the analyzing procedures.

5.1.9 Analysis of the results

As the initial set of results relied on the analyzer interpretations, therefore the python interface was used. Through this interface the Logic was invoked to capture the relevant data and retain it in a Comma separated Values (CSV) file format. The data was stored in a file by the name 'pycapture.csv' in the system directory.

Additionally, using Python for interfacing here also allowed for simplified processing of the data. For instance, multiple columns were labelled, one for each event so as to record the timing between consecutive events rather than just having the absolute time duration from when the capturing was initiated.

Figure 5.2 on the next page shows multiple sets of data samples interpreting the time durations between the events on Radio that had been recorded while performing this task. As mentioned earlier, there are four unique Radio events under consideration that are bound to occur in an orderly fashion, these being READY, ADDRESS, PAYLOAD and END. Also shown are the expected values for the set of recorded values pertaining to each pair of the events.

In Figure 5.2, the symbols are denoted as

1. R->A denoting the time duration between READY and ADDRESS.
2. A->P denoting the time duration between ADDRESS and PAYLOAD.
3. P->E denoting the time duration between PAYLOAD and END

In this manner, the experiment is repeated many times to get a number of samples. As such Figure 5.2 shows five different sets of samples collected corresponding to the symbols J, B, S, N and E. In this manner,

	R->A (J)	A->P (J)	P->E (J)	R->A (B)	A->P (B)	P->E (B)	R->A (S)	A->P (S)	P->E (S)	R->A (N)	A->P (N)	P->E (N)	R->A (E)	A->P (E)	P->E (E)
1	40.29	113.92	23.15	37.87	131.75	21.31	42	128.3	24.9	42.94	135.32	25.78	41.25	129	24.06
2	40.27	113.91	23.16	37.44	133.19	19.94	42	128.3	24.6	42.88	135.38	25.72	41.25	129.06	24
3	40.31	114.24	22.83	37.69	133.31	19.75	42	128.3	24.8	42.98	135.28	25.76	41.25	129	24.06
4	40.34	113.95	23.12	37.19	132.5	20.56	42	128.3	24.9	43.05	135.21	25.85	41.25	129	24.06
5	40.25	113.97	23.1	36.56	132.81	20.25	42	128.3	24.7	42.97	135.29	25.77	41.25	129	24.06
6	40.28	113.93	23.14	37	133.06	20	42	128.3	24.8	42.94	135.31	25.82	41.25	129	24.06
7	40.3	113.97	23.1	37.81	133.25	19.81	42	128.3	24.8	43.03	135.23	26.37	41.25	129	24.06
8	40.31	113.94	23.13	37.56	132.56	20.5	42	128.3	24.8	43	135.26	25.81	41.25	129	24.06
9	40.31	113.95	23.12	37.88	132.81	20.25	42	128.3	24.7	42.95	135.31	25.78	41.25	129.06	24.06
10	40.3	113.91	23.16	36.25	132.69	20.37	42	128.3	24.7	42.92	135.34	25.71	41.25	129	24.06
11	40.31	113.99	23.09	38.19	131.94	21.13	41.9	128.3	24.7	42.95	135.31	25.78	41.25	129.06	24.06
12	40.31	113.95	23.13	38.12	131.69	21.37	41.9	128.3	24.8	42.93	135.32	25.72	41.25	129	24.06
13	40.31	113.93	23.15	38.37	131.44	21.63	41.9	128.3	24.6	42.93	135.33	25.79	41.25	129	24.06
14	40.27	113.95	23.13	38.69	131.75	21.31	41.9	128.3	24.8	42.96	135.3	25.73	41.25	129	24.06
15	40.29	113.97	23.11	38.87	131.5	21.56	41.9	128.3	24.7	43.09	135.17	25.86	41.25	129	24.06
16	40.3	113.93	23.15	38.44	131.75	21.31	41.9	128.3	24.7	43.12	135.14	25.82	41.25	129	24.06
17	40.35	113.95	23.13	37.25	132.69	20.38	41.9	128.3	24.8	43	135.26	25.79	41.19	129.06	24.06
18	40.31	113.96	23.11	37.56	132.31	20.75	42	128.2	24.8	43.01	135.24	25.76	41.25	129	24.06
19	40.3	114.51	22.56	37.56	132.25	20.81	42	128.2	24.8	42.95	135.31	25.78	41.25	129	24.06
20	40.28	113.93	23.14	37.31	132.94	20.12	42	128.3	24.7	42.94	135.32	25.72	41.25	129	24.06
21	40.3	113.93	23.14	38.69	131.69	21.37	41.9	128.4	24.7	42.91	135.35	25.79	41.19	129.06	24.06
22	40.3	113.95	23.12	37.62	131.5	21.56	42.2	128.1	24.7	42.98	135.28	25.72	41.25	129	24.06
23	40.3	113.98	23.09	38.56	131.63	21.44	42	128.3	24.7	43.09	135.17	25.85	41.25	129	24.06
24	40.3	113.98	23.09	38.62	131.63	21.5	42	128.3	24.8	43.01	135.25	25.84	41.25	129	24.06
25	39.92	113.92	23.15	37.12	131.5	21.56	41.9	128.4	24.7	43.03	135.23	25.84	41.25	129	24.06
26	40.3	113.91	23.16	37.94	132.38	20.69	41.9	128.4	24.6	43.01	135.25	25.82	41.25	129	24.06
27	40.28	113.95	23.12	37.88	132.19	20.88	42	128.3	24.8	43.06	135.2	25.85	41.25	129	24.06
28	40.29	113.95	23.13	37.44	132.44	20.62	41.9	128.4	24.8	42.98	135.28	25.82	41.25	129	24.06
29	40.33	113.97	23.11	37.75	132.06	21.06	42	128.3	24.8	42.98	135.27	25.79	41.25	129	24.06
30	40.32	113.95	23.13	38.75	132.31	20.81	41.9	128.4	24.7	42.96	135.3	25.75	41.25	129.06	24.06
31	40.31	113.94	23.14	38.63	131.44	21.63	42	128.3	24.8	42.96	135.3	25.75	41.25	129	24.06
32	40.33	113.96	23.12	38.81	131.38	21.69	42	128.3	24.7	43.01	135.25	25.8	41.25	129	24.06
33	40.31	113.93	23.15	38.69	131.44	21.63	42	128.3	24.8	42.95	135.31	25.74	41.25	129.06	24
34	40.28	113.93	23.15	37.12	132.25	20.81	41.9	128.3	24.8	42.99	135.27	25.81	41.25	129	24.06
35	40.36	113.94	23.13	38.94	131.75	21.31	41.9	128.3	24.8	43.01	135.25	25.84	41.25	129	24.06
36	40.33	113.92	23.15	38.44	132.44	20.63	41.9	128.3	24.8	43.05	135.21	25.81	41.25	129	24.06
37	40.35	113.97	23.1	37.5	132.5	20.56	41.9	128.3	24.8	42.97	135.29	25.73	41.25	129.06	24.06
38	40.28	113.96	23.11	37	132.5	20.56	41.8	128.4	24.8	42.9	135.36	25.71	41.25	129	24.06
39	40.3	113.96	23.11	37.75	133.06	20	41.9	128.3	24.8	42.96	135.3	25.79	41.25	129	24.06
40	40.27	113.92	23.15	38	132.12	20.94	42	128.2	24.8	42.93	135.33	25.72	41.25	129	24.06
41	40.28	113.94	23.13	37.88	132.62	20.44	41.9	128.3	24.8	42.98	135.28	25.77	41.25	129	24.06
42	40.3	113.91	23.16	38.63	131.62	21.44	42	128.3	24.7	42.89	135.37	25.73	41.25	129.06	24.06
43	40.33	114.3	22.77	38.62	131.56	21.5	41.9	128.4	24.7	43.07	135.19	25.84	41.25	129	24.06
44	40.33	113.93	23.14	38.75	133.25	19.81	41.9	128.4	24.7	42.82	135.44	26.17	41.25	129	24.06
45	40.3	113.92	23.16	38.56	131.62	21.5	42	128.3	24.9	43.03	135.23	25.77	41.25	129	24.06
46	40.33	113.92	23.16	37.81	132.56	20.5	42	128.3	24.7	43.03	135.23	25.86	41.25	129.06	24.06
47	40.27	113.97	23.11	37.12	132.62	20.44	42	128.3	24.7	42.99	135.27	25.73	41.25	129	24.06
48	40.34	113.97	23.11	37.75	132.5	20.56	42	128.3	25	42.92	135.34	25.73	41.25	129	24.06
49	40.33	113.94	23.14	37.94	132.31	20.75	42	128.3	24.7	42.96	135.3	25.78	41.25	129	24.06
Expected:	40	112	24	40	128	24	40	128	24	40	136	24	40	128	24

Figure 5.2: *Recorded transmit event durations between the different Radio events*

1. R->A (J), A->P (J) and P->E (J) represent first set of recorded data samples
2. R->A (B), A->P (B) and P->E (B) represent second set of recorded data samples
3. R->A (S), A->P (S) and P->E (S) represent third set of recorded data samples
4. R->A (N), A->P (N) and P->E (N) represent fourth set of recorded data samples
5. R->A (E), A->P (E) and P->E (E) represent fifth set of recorded data samples

Figure 5.3 shows the variations in the measured time durations as shown in Figure 5.2 between the READY and ADDRESS events for each of the five sets of the data samples.

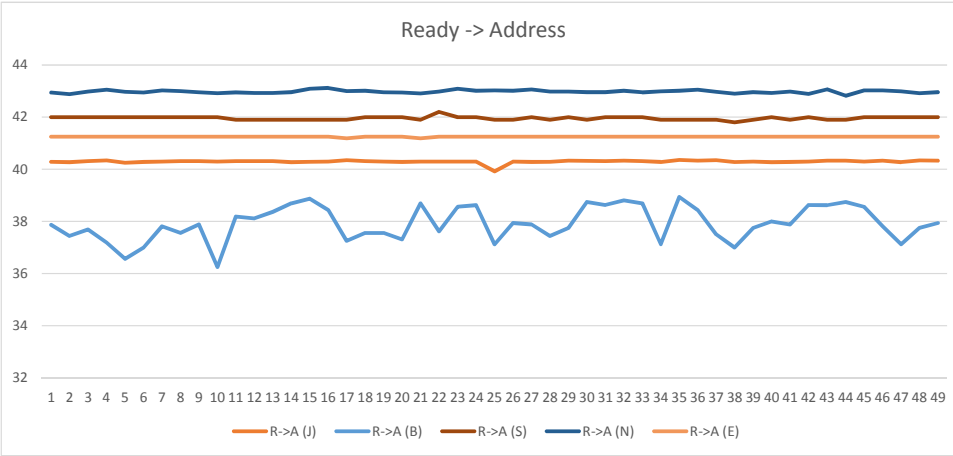

Figure 5.3: *Chart showing the variations in the measured time durations between the READY and ADDRESS events for each of the five sets of the data samples*

Figure 5.4 on the next page shows the variations in the measured time durations as shown in Figure 5.2 between the ADDRESS and PAYLOAD events for each of the five sets of the data samples.

Figure 5.5 on page 63 shows the variations in the measured time durations as shown in Figure 5.2 between the PAYLOAD and END events for each of the five sets of the data samples.

Next, Figure 5.6 on page 64 shows the corresponding deviations observed in the measured values pertaining to each of the pair of events from their given expected values as shown in Figure 5.2

Further, Figure 5.7 on page 65 shows the deviations of the measured time durations between the READY and ADDRESS events from their corresponding expected values as shown in Figure 5.2.

Further, Figure 5.8 on page 66 shows the deviations of the measured time durations between the ADDRESS and PAYLOAD events from their corresponding expected values as shown in Figure 5.2.

Further, Figure 5.9 on page 67 shows the deviations of the measured time durations between the PAYLOAD and END events from their corresponding expected values

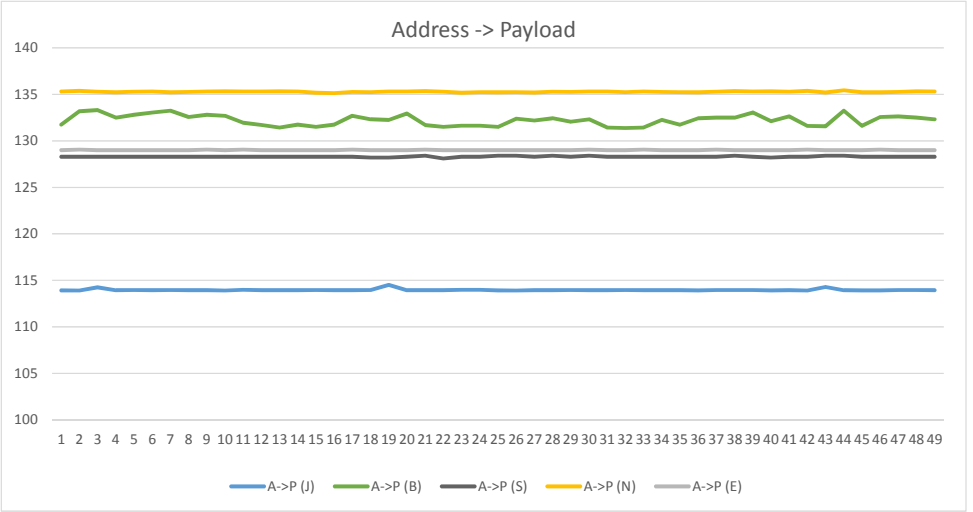

Figure 5.4: *Chart showing the variations in the measured time durations as shown in Figure 5.2 between the ADDRESS and PAYLOAD events for each of the five sets of the data samples*

as shown in Figure 5.2.

5.2 Project

5.2.1 Time is just a shadow (David Edwin)

In a nutshell, all that this project focuses on is the correlation of 'time' as an entity and how it behaves in embedded systems.

Whether the initial ramp-up sessions or the subsequent thesis work, everything was centered on the precise measurements and interpretations with respect to time. Time is a relative quantity yet something that can't be controlled beyond certain admissible limits. Then there are clocks operating at different rates, some at higher while other at lower frequencies. But one thing for sure is that these clocks are continuously drifting apart in real time with respect to themselves as well as in relation to other clocks. Therefore, it is quite evident that for any clock to synchronize with its neighbors or comparable entities, it needs to synchronized to itself too.

So for the case of Radio being used for transmission and reception purposes or any other commonly used systems such as client-server systems, advanced computer networks, etc requiring necessary data to be exchanged, it is very crucial to model the system behavior with time so as to meet the required goals. Also the key areas

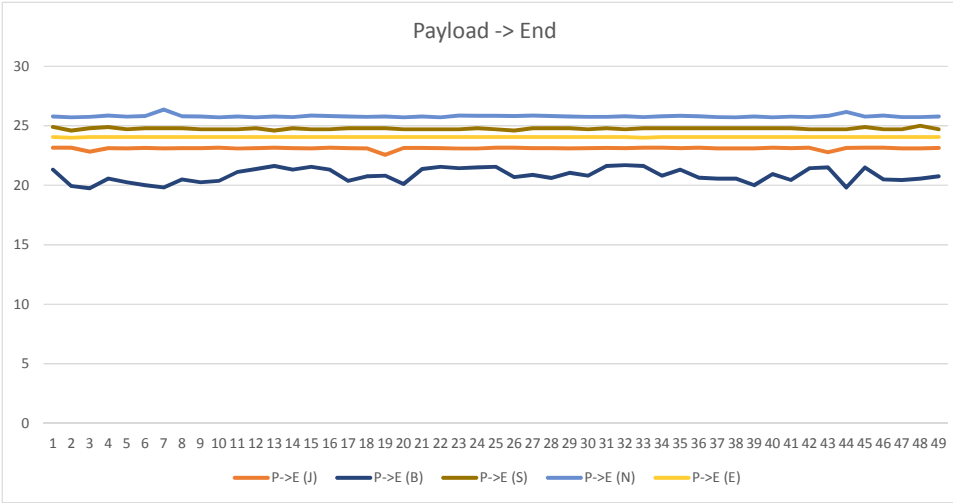

Figure 5.5: *Chart showing the variations in the measured time durations between the PAYLOAD and END events for each of the five sets of the data samples*

of focus for synchronizing time events are drift, offset, jitter, frequency stability, operating temperature (range), frequency tolerance, etc. Many of these have already been discussed in earlier.

5.2.2 Mesh

Data communication between different nodes via Radio helped in conserving energy using BLE as the underlying technology. Bluetooth Low Energy or Bluetooth Smart being an advancement to the standard Bluetooth technology serves as the foundation for the business of Nordic Semiconductor ASA as the major products have been designed relying on this technology.

However, as everything comes with a mix of good and bad, similarly this technology also poses significant constraints like the short range of communication. And this is addressed mainly by the Mesh network solution [47]. The Nordic Semiconductor nRF51 ble-broadcast-mesh used for synchronizing real time events acts as a network of multiple Bluetooth devices interconnected to each other so as to achieve long distance communication among multiple nodes (multi-hop cases) and not only limited to synchronizing neighboring nodes.

	Deviations from expected values														
	R->A (J)	A->P (J)	P->E (J)	R->A (B)	A->P (B)	P->E (B)	R->A (S)	A->P (S)	P->E (S)	R->A (N)	A->P (N)	P->E (N)	R->A (E)	A->P (E)	P->E (E)
1	0.29	1.92	-0.85	-2.13	3.75	-2.69	2	0.3	0.9	2.94	-0.68	1.78	1.25	1	0.06
2	0.27	1.91	-0.84	-2.56	5.19	-4.06	2	0.3	0.6	2.88	-0.62	1.72	1.25	1.06	0
3	0.31	2.24	-1.17	-2.31	5.31	-4.25	2	0.3	0.8	2.98	-0.72	1.76	1.25	1	0.06
4	0.34	1.95	-0.88	-2.81	4.5	-3.44	2	0.3	0.9	3.05	-0.79	1.85	1.25	1	0.06
5	0.25	1.97	-0.9	-3.44	4.81	-3.75	2	0.3	0.7	2.97	-0.71	1.77	1.25	1	0.06
6	0.28	1.93	-0.86	-3	5.06	-4	2	0.3	0.8	2.94	-0.69	1.82	1.25	1	0.06
7	0.3	1.97	-0.9	-2.19	5.25	-4.19	2	0.3	0.8	3.03	-0.77	2.37	1.25	1	0.06
8	0.31	1.94	-0.87	-2.44	4.56	-3.5	2	0.3	0.8	3	-0.74	1.81	1.25	1	0.06
9	0.31	1.95	-0.88	-2.12	4.81	-3.75	2	0.3	0.7	2.95	-0.69	1.78	1.25	1.06	0.06
10	0.3	1.91	-0.84	-3.75	4.69	-3.63	2	0.3	0.7	2.92	-0.66	1.71	1.25	1	0.06
11	0.31	1.99	-0.91	-1.81	3.94	-2.87	1.9	0.3	0.7	2.95	-0.69	1.78	1.25	1.06	0.06
12	0.31	1.95	-0.87	-1.88	3.69	-2.63	1.9	0.3	0.8	2.93	-0.68	1.72	1.25	1	0.06
13	0.31	1.93	-0.85	-1.63	3.44	-2.37	1.9	0.3	0.6	2.93	-0.67	1.79	1.25	1	0.06
14	0.27	1.95	-0.87	-1.31	3.75	-2.69	1.9	0.3	0.8	2.96	-0.7	1.73	1.25	1	0.06
15	0.29	1.97	-0.89	-1.13	3.5	-2.44	1.9	0.3	0.7	3.09	-0.83	1.86	1.25	1	0.06
16	0.3	1.93	-0.85	-1.56	3.75	-2.69	1.9	0.3	0.7	3.12	-0.86	1.82	1.25	1	0.06
17	0.35	1.95	-0.87	-2.75	4.69	-3.62	1.9	0.3	0.8	3	-0.74	1.79	1.19	1.06	0.06
18	0.31	1.96	-0.89	-2.44	4.31	-3.25	2	0.2	0.8	3.01	-0.76	1.76	1.25	1	0.06
19	0.3	2.51	-1.44	-2.44	4.25	-3.19	2	0.2	0.8	2.95	-0.69	1.78	1.25	1	0.06
20	0.28	1.93	-0.86	-2.69	4.94	-3.88	2	0.3	0.7	2.94	-0.68	1.72	1.25	1	0.06
21	0.3	1.93	-0.86	-1.31	3.69	-2.63	1.9	0.4	0.7	2.91	-0.65	1.79	1.19	1.06	0.06
22	0.3	1.95	-0.88	-2.38	3.5	-2.44	2.2	0.1	0.7	2.98	-0.72	1.72	1.25	1	0.06
23	0.3	1.98	-0.91	-1.44	3.63	-2.56	2	0.3	0.7	3.09	-0.83	1.85	1.25	1	0.06
24	0.3	1.98	-0.91	-1.38	3.63	-2.5	2	0.3	0.8	3.01	-0.75	1.84	1.25	1	0.06
25	-0.08	1.92	-0.85	-2.88	3.5	-2.44	1.9	0.4	0.7	3.03	-0.77	1.84	1.25	1	0.06
26	0.3	1.91	-0.84	-2.06	4.38	-3.31	1.9	0.4	0.6	3.01	-0.75	1.82	1.25	1	0.06
27	0.28	1.95	-0.88	-2.12	4.19	-3.12	2	0.3	0.8	3.06	-0.8	1.85	1.25	1	0.06
28	0.29	1.95	-0.87	-2.56	4.44	-3.38	1.9	0.4	0.8	2.98	-0.72	1.82	1.25	1	0.06
29	0.33	1.97	-0.89	-2.25	4.06	-2.94	2	0.3	0.8	2.98	-0.73	1.79	1.25	1	0.06
30	0.32	1.95	-0.87	-1.25	4.31	-3.19	1.9	0.4	0.7	2.96	-0.7	1.75	1.25	1.06	0.06
31	0.31	1.94	-0.86	-1.37	3.44	-2.37	2	0.3	0.8	2.96	-0.7	1.75	1.25	1	0.06
32	0.33	1.96	-0.88	-1.19	3.38	-2.31	2	0.3	0.7	3.01	-0.75	1.8	1.25	1	0.06
33	0.31	1.93	-0.85	-1.31	3.44	-2.37	2	0.3	0.8	2.95	-0.69	1.74	1.25	1.06	0
34	0.28	1.93	-0.85	-2.88	4.25	-3.19	1.9	0.3	0.8	2.99	-0.73	1.81	1.25	1	0.06
35	0.36	1.94	-0.87	-1.06	3.75	-2.69	1.9	0.3	0.8	3.01	-0.75	1.84	1.25	1	0.06
36	0.33	1.92	-0.85	-1.56	4.44	-3.37	1.9	0.3	0.8	3.05	-0.79	1.81	1.25	1	0.06
37	0.35	1.97	-0.9	-2.5	4.5	-3.44	1.9	0.3	0.8	2.97	-0.71	1.73	1.25	1.06	0.06
38	0.28	1.96	-0.89	-3	4.5	-3.44	1.8	0.4	0.8	2.9	-0.64	1.71	1.25	1	0.06
39	0.3	1.96	-0.89	-2.25	5.06	-4	1.9	0.3	0.8	2.96	-0.7	1.79	1.25	1	0.06
40	0.27	1.92	-0.85	-2	4.12	-3.06	2	0.2	0.8	2.93	-0.67	1.72	1.25	1	0.06
41	0.28	1.94	-0.87	-2.12	4.62	-3.56	1.9	0.3	0.8	2.98	-0.72	1.77	1.25	1	0.06
42	0.3	1.91	-0.84	-1.37	3.62	-2.56	2	0.3	0.7	2.89	-0.63	1.73	1.25	1.06	0.06
43	0.33	2.3	-1.23	-1.38	3.56	-2.5	1.9	0.4	0.7	3.07	-0.81	1.84	1.25	1	0.06
44	0.33	1.93	-0.86	-1.25	5.25	-4.19	1.9	0.4	0.7	2.82	-0.56	2.17	1.25	1	0.06
45	0.3	1.92	-0.84	-1.44	3.62	-2.5	2	0.3	0.9	3.03	-0.77	1.77	1.25	1	0.06
46	0.33	1.92	-0.84	-2.19	4.56	-3.5	2	0.3	0.7	3.03	-0.77	1.86	1.25	1.06	0.06
47	0.27	1.97	-0.89	-2.88	4.62	-3.56	2	0.3	0.7	2.99	-0.73	1.73	1.25	1	0.06
48	0.34	1.97	-0.89	-2.25	4.5	-3.44	2	0.3	1	2.92	-0.66	1.73	1.25	1	0.06
49	0.33	1.94	-0.86	-2.06	4.31	-3.25	2	0.3	0.7	2.96	-0.7	1.78	1.25	1	0.06

Figure 5.6: *Calculated deviations of the measured time durations from their expected values*

5.2.3 Tools used

Although Nordic Semiconductor offers a broad set of software/hardware tools, the thesis work mainly utilized some key tools from the nRF51 tool suite along with others. Most of them have already been discussed earlier.

Among the hardware tools used were:

- nRF51 DK [16]
- nRF51 Dongle [20]

The software tools utilised are listed below:

- nRF5-SDK-v12, nRF51 SDK v12.2 SEGGER Embedded Studio (SES) [16]

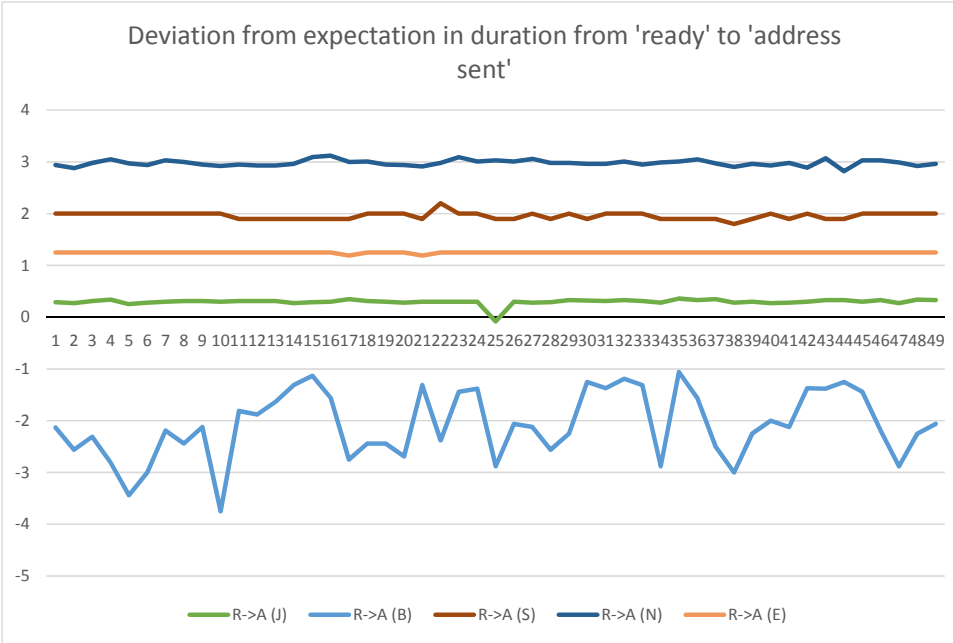

Figure 5.7: *Deviation from expectation in duration from 'ready' to 'address sent'*

- nRF-Sniffer [16]
- nRF5x-Command-Line-Tools-Win32 [16] [24]
- nRF51-Pynrfjprog (nrfjprog Python interface) [36][37]
- nRF-Connect-Windows [16]

Besides this there were other standard tools like the Logic Analyzer hardware, Logic software [34], Wireshark [21], Segger RTT viewer [48].

As mentioned earlier, the entire SDLC right from the design to coding, testing and then finally debugging was carried out using SEGGER Embedded Studio IDE [19].

5.2.4 Understanding of Bluetooth, Bluetooth Low Energy and Bluetooth 5

I have worked on developing Bluetooth Low Energy solutions at Nordic Semiconductor ASA as a part of my Master's thesis project 'Time synchronization across multiple devices in network nodes' in the Software department.

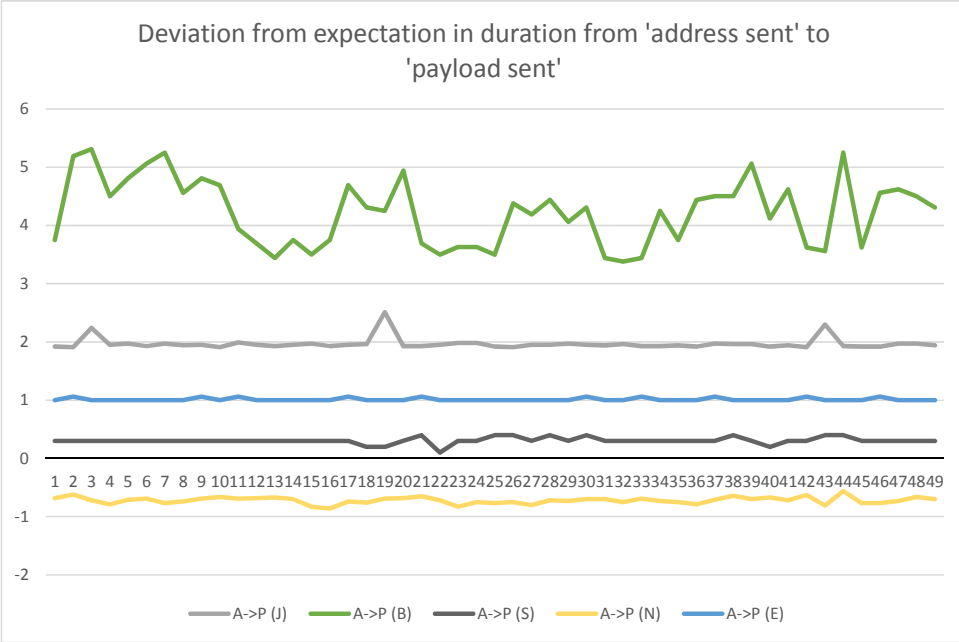

Figure 5.8: *Deviation from expectation in duration from 'address sent' to 'payload sent'*

BLE is an advancement to the standard Bluetooth technology as mentioned earlier.

5.2.4.1 Bluetooth

Bluetooth being a wireless technology allows for data communication between nearby mobile devices without the use of wired connections. [49] Apart from being commonly used in headsets, this technology is also advantageous for many types of wireless file transfers. It could be used between two computers or even between a cellular phone and a computer; or even for interconnecting peripherals like mouse, keyboards, gamepads, printers, etc wirelessly. [50] Moreover, with the available Bluetooth dongles, this technology can be linked to remote desktops lacking the integrated Bluetooth hardware. [50]

5.2.4.2 Bluetooth Low Energy

Moving on, having emerged as a successor to the standard Bluetooth in 2011, there is the Bluetooth Low Energy (BLE) also known as Bluetooth Smart. Although BLE serves the same function as Bluetooth yet the main advantage here is that it operates

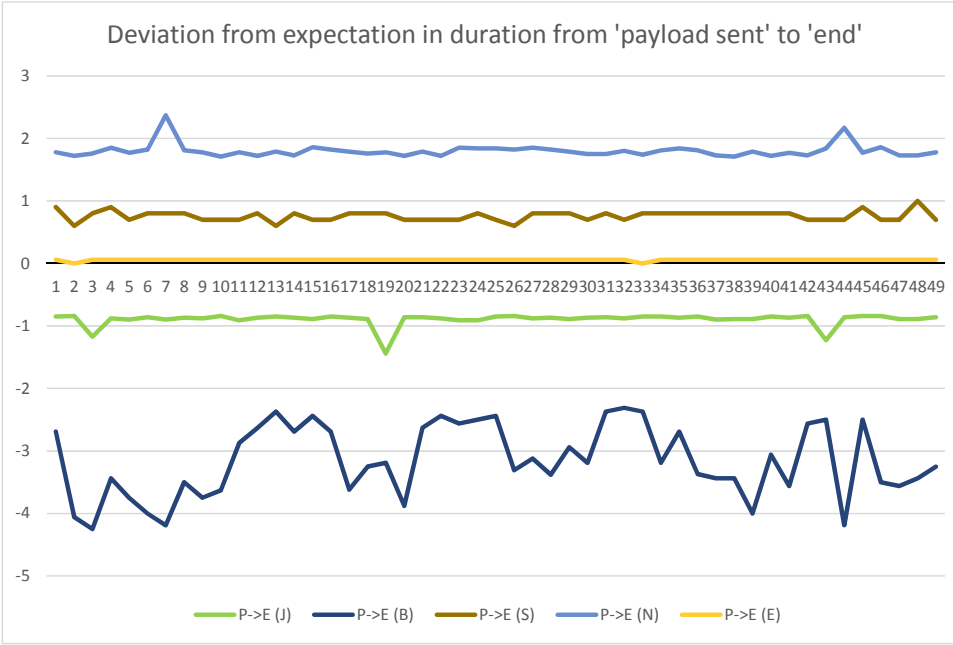

Figure 5.9: *Deviation from expectation in duration from 'payload sent' to 'end'*

on a low-power consumption. This means that BLE allows for running applications for several years and that too with a small battery. Using BLE is very productive especially for Machine-to-machine (M2M) communication purposes. Adding on, this technology may not offer full-proof efficiency to be used for mobile phones, yet it is extremely crucial for applications requiring transfer of limited amount of data at periodic intervals. [49]

Furthermore, BLE acts as a universal standard offering low-power wireless connectivity solutions henceforth improving the feasibility for connecting many different products to tablets or smartphones.[50] [49] "Bluetooth Low Energy enables short-burst wireless connections and uses multiple network topologies, including point-to-point, broadcast and mesh." [51]

1. Point-to-Point topology: This type of network topology facilitates one-to-one communication between devices. This makes it ideal for data transfer purposes along with offering descent flexibility in connected device products like health monitoring devices, fitness trackers, etc. [51]

2. Broadcast: Another type of network topology supported by BLE is the broadcast topology. In contrast to the Point-to-Point (P2P) network topology discussed

previously, the broadcast network topology facilitates communication between one device and several other (connected) devices. One major advantage of using this topology is that it offers optimized domain-specific information exchange henceforth being tailor-made for beacon applications focused on the Point-of-Interest (PoI) as well as item, way-finding services. [51]

3. Mesh: The third type of network topology being Mesh topology has already been explain earlier. This topology helps in functioning large-scale tailor-made networks with a series of devices at the different nodes interconnected to achieve communication throughout the network. Common applications where this type of topology is generally deployed include sensor networks, build automation, asset tracking and any other similar solutions requiring reliable and secure communication between many interconnected devices. [51]

Over the years BLE has been continuously widening its usability so as to deliver cost-effective, secure, scalable and state-of-the-art solutions. It is quite evident from the fact that most of the PCs and even mobile devices/smart phones along with other commonly known applications are having built in functionality of connectivity via BLE. [52]

Apart from the low cost and power consumption offered by this technology, optimum energy efficiency and simplified design characteristics are other advantages of using BLE which, in turn, generate high degree of flexibility as well as ease of integration. [52]

Its ever-growing popularity is also making BLE ideal for use in many smart home applications involving connected door locks and lighting as well as in today's advanced sensor technologies. [52]

In a similar manner to Bluetooth, BLE also uses the 2.4 GHz ISM band for its operation. But in contrast to the standard Bluetooth technology, BLE offers minimal consumption by constantly keeping in sleep mode unless a connection has been initiated. This causes the connection times to be lowered down to a few milliseconds unlike Bluetooth taking upto 100 milliseconds. The reason for reduced connection times being the extremely high data rates at 1 Mb/s. [49]

To sum up the difference between the two technologies discussed, it is evident that both Bluetooth as well as its successor, BLE have quite different applications. The reason being that Bluetooth, inspite of being able to handle loads of data yields a very high rate of power consumption and even adds to the overall cost in operation. On the other hand, BLE/Bluetooth Smart finds its use in short range data transfers henceforth being able to operate on a limited battery power for several years yet delivering a cost-effective performance. [49]

Moreover, BLE offers "Ultra-low peak, average and idle mode power consumption". [52] Apart from this, it also offers some other flexible features like multi-vendor interoperability, simplicity of deployment and usage, enhanced communication range, cross platform efficiency supporting iOS and Android mobile operating systems as well as chief desktop platforms like OSX, Linux, Windows 8, and more. [52]

5.2.4.3 Bluetooth 5

With the rapidly evolving technologies, Bluetooth 5 as a further advancement to BLE has come into the picture and is already making its superior presence felt globally in the market. This technological innovation is extending the scalability of BLE to wider dimensions than just being limited to Personal Area Networks. [52] [53]

Bluetooth 5 being the latest version in the Bluetooth series yields some of the most significant advancements in the Bluetooth specification from the time of Bluetooth 4.0 as well as the evolution of BLE. [53] It offers a great deal of flexibility from the previous versions hereby offering improved range, increased message broadcasting capacity, advertising extensions, improved co-existence, errata as well as higher throughput for Bluetooth based applications. [53]

Errata High throughput Long range Advertising extensions Increased broadcast capacity Improved co-existence

Having been released in December 2016, Bluetooth 5 has been capable enough to quadruple the overall communication range while doubling the speed of low energy connectivity applications and enhancing the capacity of connectionless broadcasts by eight times. Along with offering these significant advantages, Bluetooth 5 is known to provide the industry standard leading power performance. [54].

One of the key features of Bluetooth 5 is its potential to offer "connectionless" Internet of Things (IoT) along with advanced beacon as well as location-based capabilities for domestic, enterprise and industry specific applications. [54]

Chapter 6
Experiments and Results

The experiment performed for the practical analysis of synchronizing time events across multiple devices in the network nodes was based on the Tiny-Sync and Mini-Sync algorithms as have been explained earlier.[2] For this, a well-designed state machine has been proposed based on which the main code for this embedded software solution has been designed. This state machine uses the associated radio events and tasks, and other functionalities as specified in the "nRF51 Reference manual" [5].

Using a 802.11b multi-hop ad-hoc network [2] showing similar delay variations as observed in WSNs, analysis was done with two computers as neighbors to each other and then being at a distance of five hops from each other. With a probe message being sent at a constant frequency of 1 second for about 83 minutes in each of the two cases, 5000 data point samples had been recorded in each case and the corresponding results were generated. [2]

Table 6.1 [2] gives the statistical analysis of the generated data- points in each of the one hop and five hops cases. [2]

Table 6.1: *Statistics of the Data-Points Collected for the One Hop and the Five Hops Experiments*

	Min [ms]	Max [ms]	Avg [ms]	Std Dev [ms]
one hop	3.040	24.217	3.366	1.112
five hops	16.073	177.744	31.540	12.874

Table 6.2 on the following page [2] gives the generated upper bounds for interpreting the relative error between the two studied algorithms, namely, Mini-Sync and Tiny-sync. [2]

Table 6.1 clearly indicates that the time values recorded for the one hop experiment were lower than those recorded when the two computers were five hops away.

Table 6.2: *Upper Bounds of the Relative Error between Mini-Sync and Tiny-Sync*

	Relative errors for			
	$\underline{a_{12}}$	$\overline{a_{12}}$	$\underline{b_{12}}$	$\overline{b_{12}}$
one hop	0.14%	0.19%	0.19%	0.14%
five hops	1.8%	1.7%	1.7%	1.8%

Also, Table 6.2 shows that the five hops experiment incurred a higher degree of relative error than the one hop case. In case of relative drift, this satisfies the relation as in Equation 3.22, that is, the uncertainty bound on the relative drift decreases with increase in the distance between the first and the last collected data points. [3] Similarly, in case of the relative offset the measurements recorded are in concordance with the relation as in Equation 3.24. [3]

In addition to the mathematical interpretations, certain graphical conclusions were also drawn to interpret them with the mathematical results.

Furthermore, Figure 6.1 on the next page shows the evolving bounds on the relative drift as the number of samples collected goes on increasing. [2]

and

Figure 6.2 on page 74 shows the effect on the precision of the relative drift with the change in the uncertainty factor as the number of samples collected goes on increasing. [2]

Similarly, Figure 6.3 on page 75 shows the evolving bounds on the relative offset as the number of samples collected goes on increasing. [2]

and

Figure 6.4 on page 76 shows the effect on the precision of the relative offset with the change in the uncertainty factor as the number of samples collected goes on increasing. [2]

Figure 6.2 again shows that in case of drift, its precision is known to be bounded as 2RTT / (sample_number), where sample_number implies the total number of samples collected. It is evident that the drift precision shows a significant improvement continually as more samples are collected [2]. Also, Figure 6.4 shows that the offset precision is well expected to be on the order of Round Trip Time. The offset estimate shows a significant improvement in the precision for the initial samples, while improving only marginally ahead. [2]

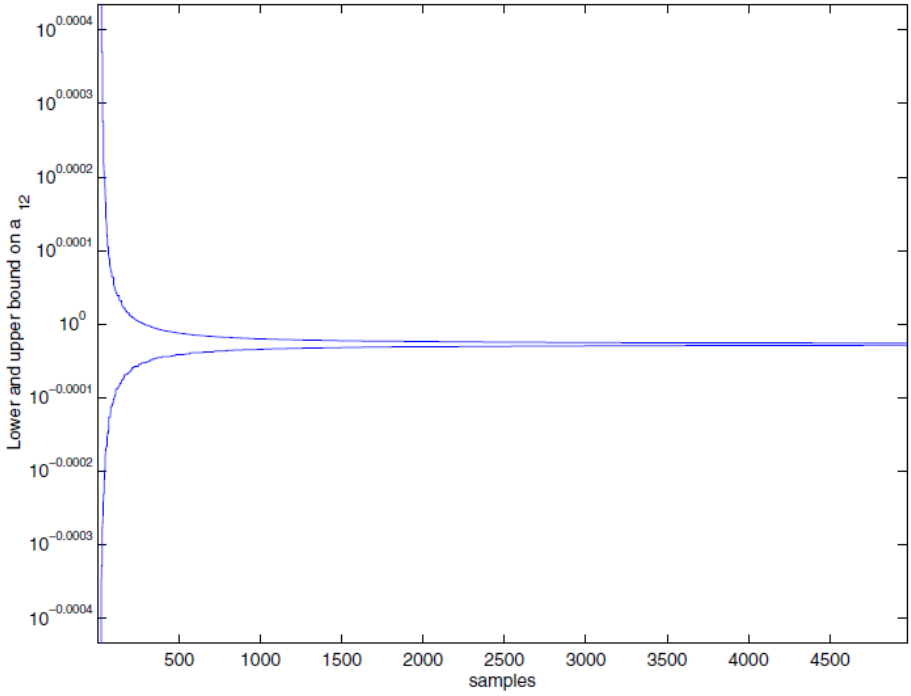

Figure 6.1: *Evolution of the bounds on a_{12} as more samples are collected Based on a figure in [2]*

[2] The Tiny-Sync algorithm is found to be sub-optimal inspite of its simplicity. Moreover, the overall loss in precision accounted for by Tiny-Sync algorithm was realized with the help of measurements pertaining to the upper bounds of the relative error between the Mini-Sync and Tiny-Sync algorithms as shown in Table 6.2. [2] Furthermore, this error was computed after processing each new data sample. [2]. Also it is seen from Table 6.2 that the difference values were smaller than 2 percent in all cases and even smaller than 0.1 percent after processing of the first few samples. [2]

This led to the conclusion that taking into account the necessary practical considerations, results from Tiny-Sync were precise enough in comparison to the results from Mini-Sync. Henceforth the need for using Mini-Sync as an extension to the Tiny-Sync algorithm could be avoided in this scenario. [2]

Note: As stated earlier, from practical point of view, Tiny-Sync prefers to store a limited number of data points (upto four) at one point of time so as to get the

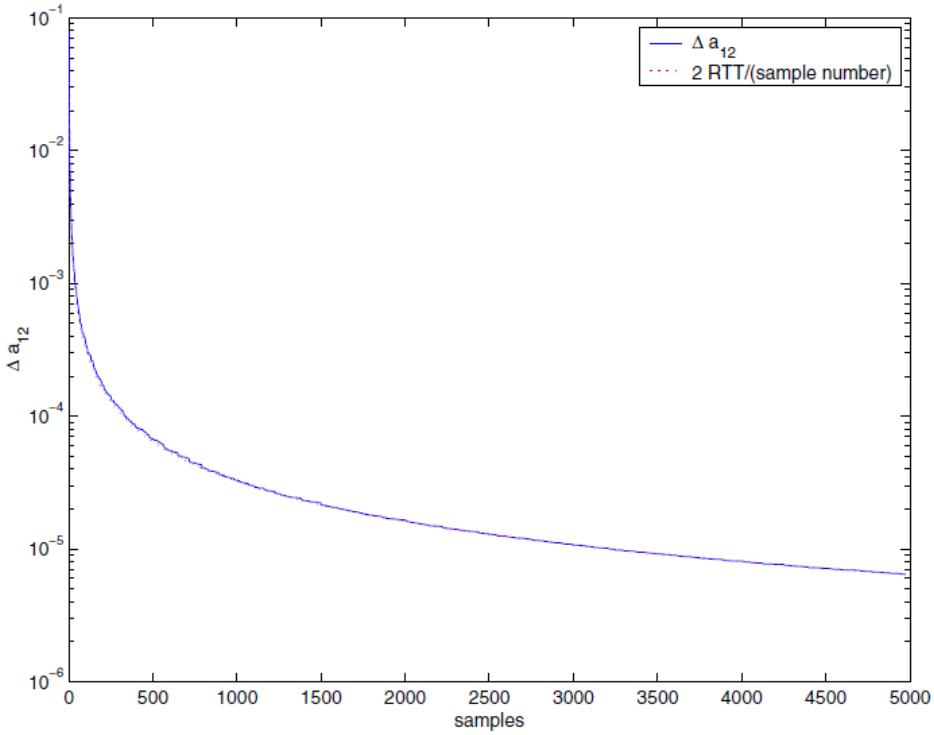

Figure 6.2: *Variation in Δa_{12} as more samples are collected*
Based on a figure in [2]

desired results. [2]

But the experiment as in [2] took 5000 samples of data points in order to distinguish the processing abilities of each of the Tiny-Sync and Mini-Sync algorithms. Finally, for a better analysis of the data processing abilities of Tiny-Sync, certain measurements were done and the results were recorded with and without data processing.

Table 6.3 on the next page [2] relates the uncertainty bounds on the relative drift and the relative offset in terms of $\frac{\Delta a_{12}}{2}$ and $\frac{\Delta b_{12}}{2}(ms)$ for the Raw data as well as the Pre-processed data for each of the one hop and the five hops experiments. [2]

For better understanding of the criteria for a constraint generated from a data point to be considered as potentially useful or not as shown in Equation 3.13 and Equation 3.14, a detailed proof [2] [3] is provided.

Figure 6.5 on page 77 demonstrates that how on attaining the three constraints

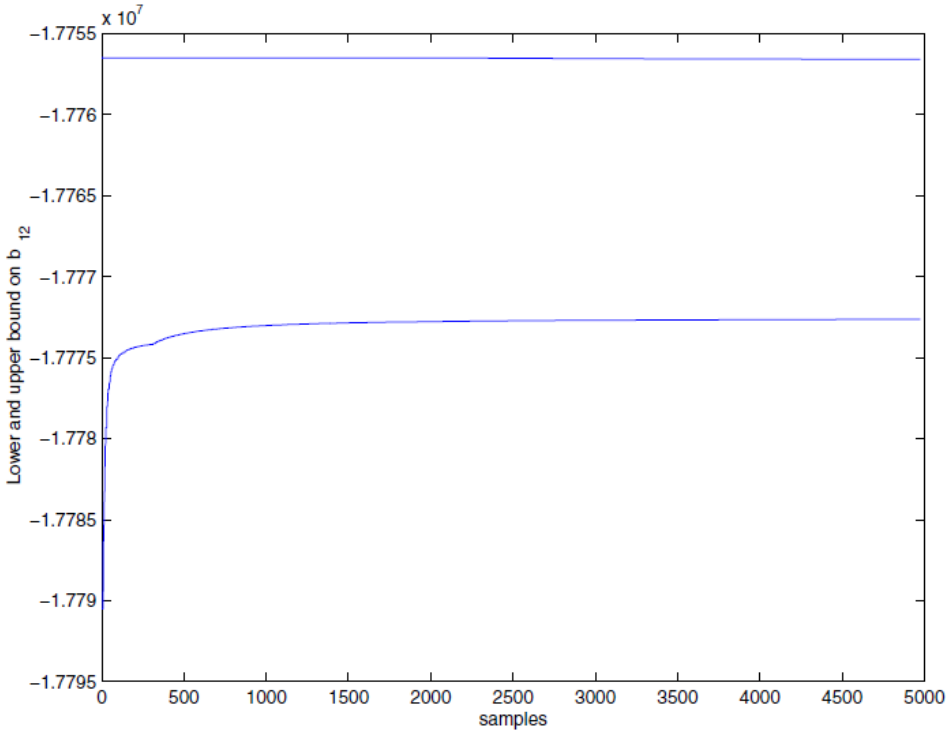

Figure 6.3: *Evolution of the bounds on b_{12} as more samples are collected Based on a figure in [2]*

Table 6.3: *Results for Tiny-Sync with and Without Data Pre-Processing*

	Raw data		Pre-processed data	
	$\frac{\Delta a_{12}}{2}$	$\frac{\Delta b_{12}}{2}(ms)$	$\frac{\Delta a_{12}}{2}$	$\frac{\Delta b_{12}}{2}(ms)$
one hop	$7.133e^{-7}$	2.0941	$2.768e^{-7}$	0.9457
five hops	$5.013e^{-6}$	17.08	$1.167e^{-6}$	3.239

A_i, A_j and A_k, one constraint A_j can be easily discarded as the other two (being A_i and A_k) give better estimates on the relative drift and the relative offset. [2]

The full proof using the Figure 6.5 can be found in [2].

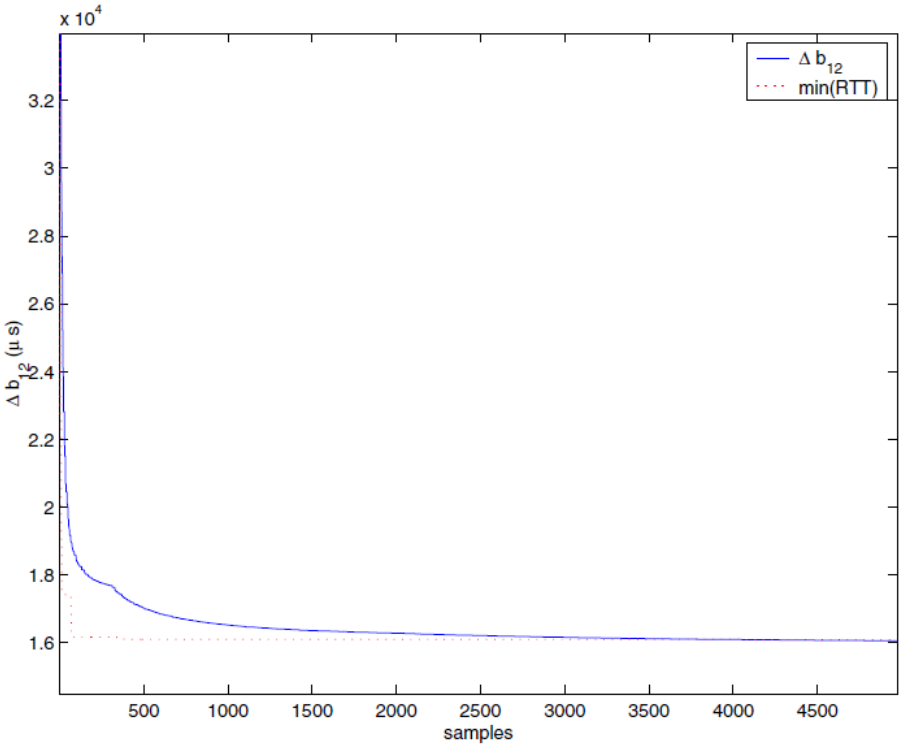

Figure 6.4: *Variation in Δb_{12} as more samples are collected*
Based on a figure in [2]

6.1 Tools used for analysis

1. Two to three nRF51 boards were used (two boards are used as devices for consideration as node 1 and node 2 to be time synchronized with respect to each other).

2. A hub has been used as the network hardware device so as to ensure all the devices are properly connected. While one of the boards acting as the transmitter is connected directly with the PC, the second board acting as the receiver is connected to the hub in a manner that both the devices lie within the span of each other to be accurately synchronized.

3. Logic analyzer tool has been used along with the Saleae Logic 1.2.12 software.

4. Segger RTT viewer has been used for debugging the *C* code in real time environment.

5. SDK used is the nRF5_SDK_12.2.SES from Nordic Semiconductor.

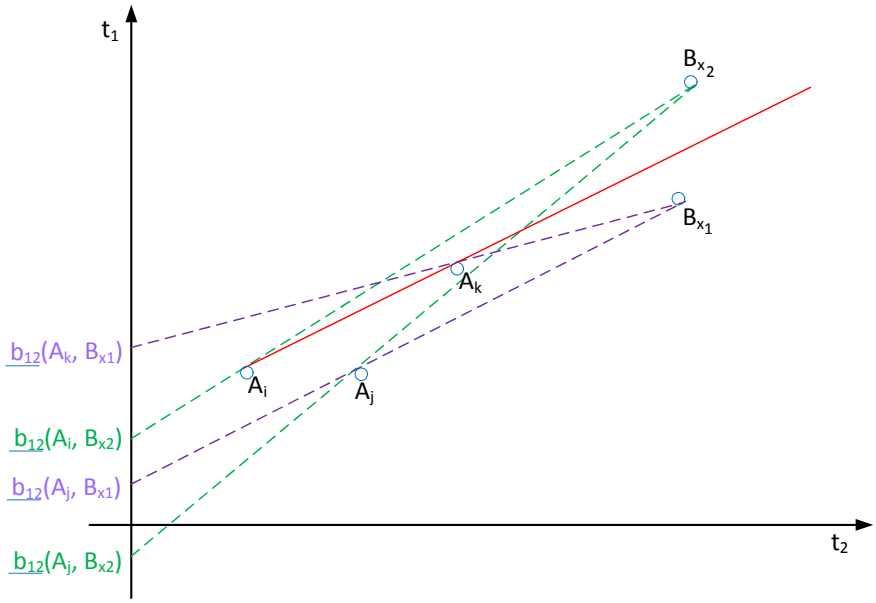

Figure 6.5: *A_j can be safely discarded as A_i or A_k will result in better estimates in all cases*
Based on a figure in [2]

6.2 Theory

A ble-broadcast mesh type network with a series of nodes interconnected to each other has been taken into account. This is similar to the behaviour shown as discussed earlier in the experiment demonstrated in [2].

As per the Tiny-Sync algorithm discussed earlier, two immediate nodes are taken into consideration, say node 1 and node 2. Each of these two nodes are having their separate local clocks keeping the timing information. Since both the nodes are along a different timeline initially, therefore it is quite certain that they are not instantly synchronized.

To synchronize them with respect to each other, two crystal oscillators have been used. One being a high frequency 16M Crystal Oscillator (XOSC) and another being a low frequency 32k XOSC. Each of these oscillators used as reference sources consist of a cheap quartz crystal which helps in giving optimized performance and that too at a limited cost. To know more about crystal oscillators refer to [55].

Note: These specific crystal oscillators have been selected taking into account the essential considerations such as offset, frequency tolerance, drift, frequency

stability,operating temperature, etc. The data sheets for many crystal oscillators have been accessed from the web. But for major considerations such as drift, temperature stability and more with regard to the crystal oscillators used, refer to Sections 8.1.2 and 8.1.5 as in the nRF51422 Product Specification [40].

One of the main tools used for this experiment are the nRF51 boards acting as devices or nodes to be synchronized. (A number of boards have been used to achieve the time synchronization task over the given network). These nRF51 boards are crucial for the experimental analysis. They are incorporated with two XOSC crystals out of which the experimental results rely on the measurements from the crystal with high frequency clock. The nRF51422 System on Chip or System on a Chip (SoC) [56][40] is also inbuilt into the nRF51 DK along with the standard "ARM Cortex-M0 processor" [57] used. Although a brief description of the nRF51 DK has been provided earlier yet to have a deeper understanding, refer to [45] [17].

As stated earlier, for two clocks to be synchronized with respect to each other, two conditions need to be fulfilled.

1. Firstly, both the clocks at node 1 and node 2 must must be counting time in the same manner.
2. Secondly, there needs to be a time zero event (t_o) having occurred and defined within the network.

Two neighboring nodes are taken into consideration. All the transmissions are done using nRF Radio over the BLE stack. There is one single hardware timer as timer0 which is being used in the experiment. It is known that the timer is continuously running as timers/clocks are entities beyond manual control. In this single-hop synchronization process, there are two modes defined for each of the nodes. First, when node 1 sends acting as a transmitter, node 2 acts as the receiver to receive the information from node 1. Upon receipt of the original message from node 1 at node 2, there is some delay due to some information processing taking place along with other uncertainties that may occur within the network at random intervals. Similarly, in the other round, node 2 acts as the transmitter wherein it sends some information to node 1 which is now behaving as the receiver.

Synchronization here is being achieved by timestamping real time events with the help of nRF timer from Nordic Semiconductor.

Note: Hardware timers are preferred in comparison to the computer clocks so as to avoid unnecessary delays that could occur due to CPU intervention in the system as in the processing time which could further delay the response and so on.

As per the Tiny-Sync algorithm, after considering the unknown delays, there are 4 timestamps collected in one complete cycle from node 1 to node 2 and finally back to node 1 from node 2. These being t_o, t_{br}, t_{bt}, and t_r as discussed. First, just before the packet is sent from the node 1, it is timestamped as t_o. Then at some time in between the packet transmission, node 2 is activated in the receiver mode. Upon receipt of this packet carrying the timestamp t_o, this event is timestamped as t_{br} at node 2. Ensuing some processing delay, again the node 2 is activated but this time in the transmitter mode so as to transmit back the information to node 1 acting as the receiver now. Once again, the moment when the packet is sent from node 2 is timestamped as t_{bt} and finally timestamped as t_r upon being received at node 1.

Note: Another important consideration here is that the packet sent back from node 2 will contain all the three previous timestamps that have been recorded, these being t_o, t_{br} and t_{bt} respectively in the order of which they are obtained.

In this manner, two separate data points will be generated with each full cycle of this synchronization process between the two nodes as explained earlier.

Moreover, the experiment is repeated several times so as to collect a set of data points in order to obtain optimally bounded estimates on the relative drift and relative offset as interpreted by the Tiny-Sync algorithm. [2][3]

6.3 BLE Packet specification

The information carriers in this synchronization process are the Bluetooth Low Energy (BLE) packets, each of them having a packet structure in concordance with the standard BLE packet specification as shown in Figure 5.1

We take into account the layout for the ADV_IND packets as in Figure 5.1 since the process involves advertising the packets on air which can be captured in real time with the help of Wireshark tool along with their estimated receive times corresponding to the device being sniffed.

From Figure 5.1, The main divisions for the transmitted BLE packet are the Preamble; Address; S0; Length; S1; Payload and Cyclic Redundancy Check (CRC) respectively. This is also explained to some extent in Section 17.1.2 of the nRF51 Reference Manual [5].

6.4 Designed state machine for the analysis

As discussed earlier, although the main code designed using embedded *C* in SEGGER Embedded Studio serves as the building block for the successful development of the BLE based software solution yet the driving force to have implemented this specific

design for the *C* code at the first place is the pre-analysis of the desired functionality expected from the resulting solution.

Figure 6.6 demonstrates how the design for this embedded software solution was in effect the outcome from the prototyping done via an informational and efficient state machine.

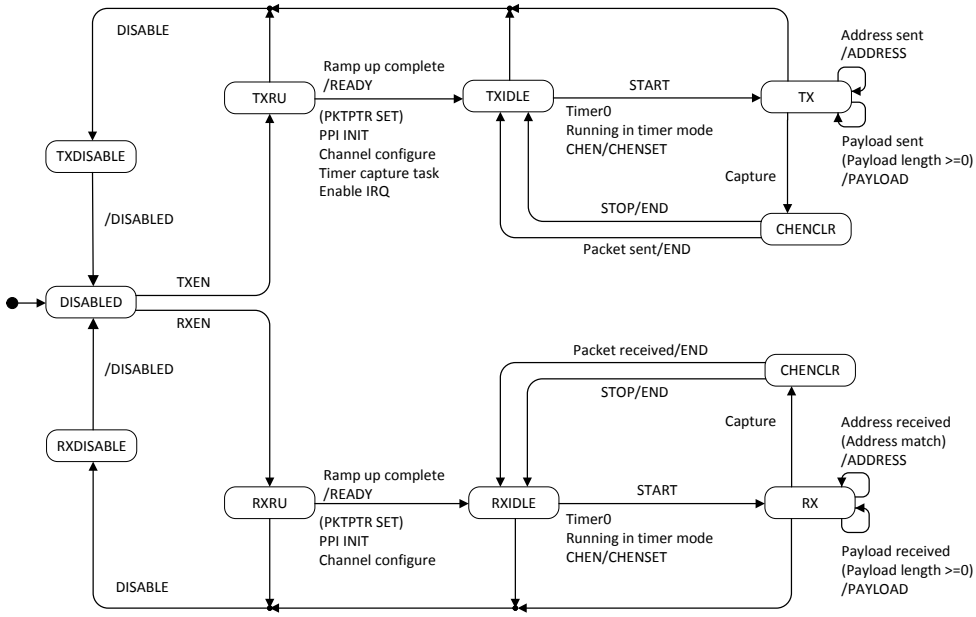

Figure 6.6: *Designed state machine for the analysis*

Note: For further detailed understanding of the radio functionality refer to Chapter 17 (2.4 GHz Radio (RADIO)) in the nRF51 Reference Manual" [5].

The state machine as shown in Figure 6.6 works as follows:

The following TASKS in RADIO [5] have been defined:

– TXEN: defined in transmitter mode

– RXEN: defined in receiver mode

The following EVENTS in RADIO [5] have been defined:

– READY: defined in transmitter and receiver modes

– ADDRESS: defined in transmitter and receiver modes

– PAYLOAD: defined in transmitter and receiver modes

– END: defined in transmitter and receiver modes

– DISABLED: defined in transmitter and receiver modes

Although the entire nRF51 Reference Manual [5] serves as a base for the practical analysis done in the thesis yet after having known the basic stuff in the manual, a stronger emphasis has been laid to the following sections so as to understand the working of the Radio states in the state machine:

– Chapter 10: Peripheral Interface [5]

– Chapter 15: GPIO tasks and events (GPIOTE) [5]

– Chapter 16: Programmable Peripheral Interconnect (PPI) [5]

– Chapter 17: 2.4 GHz Radio (RADIO) [5]

– Chapter 18: Timer/counter (TIMER) [5]

[5] Following the manual and the designed state machine shown in Figure 6.6, things are quite self-explanatory as to how the different tasks, events, registers, etc have been used for the Radio which is known to operate simultaneously in two different modes, namely, Transmitter mode and Receiver mode respectively. Besides this, at certain stages shortcuts have been enabled. A shortcut here establishes a direct relation between a task and an event associated with one unique peripheral. With the help of such a shortcut, a corresponding task is triggered when the event associated to that task is triggered. One such shortcut that could be used here was READY/START with READY as an Event and START being a Task.

Moreover, in the state machine as in Figure 6.6, there is the PPI functionality defined. The main.c file as in Appendix A.1 shows that the code has not been designed solely with the Radio example in SDK 12.2 SES or with PPI functionality only. Instead there is a combined functionality using the Radio with PPI along with timers for synchronizing time in network nodes.

Figure 6.7 on the following page explains the basic functioning of the timer.

The timer being used as shown in Figure 6.7 allows for an independent Capture functionality.There are four main tasks defined for the timer (Refer to Chapter 18 in the nRF51 Reference Manual [5]), these being START; STOP; SHUTDOWN

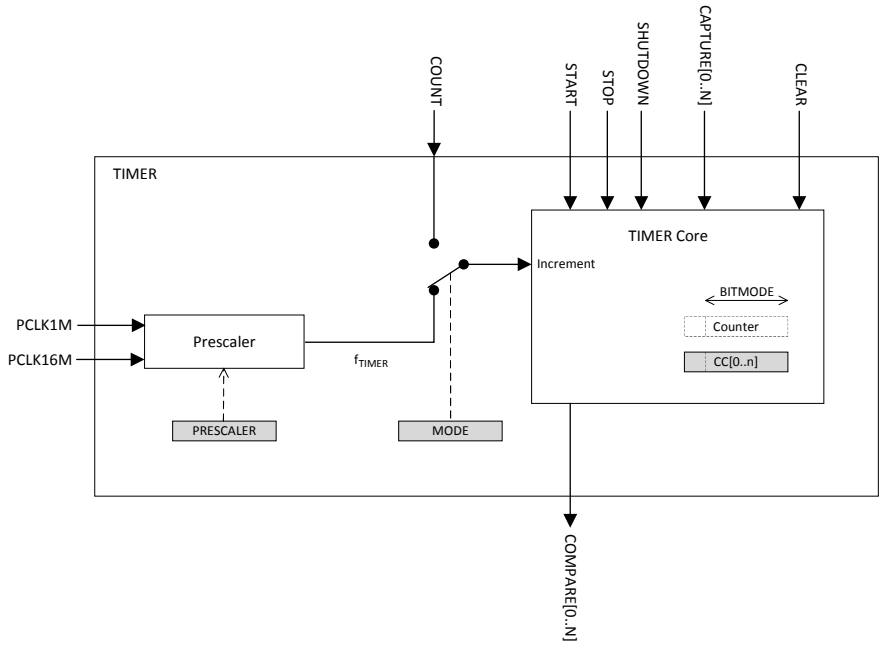

Figure 6.7: *Block schematic for timer/counter*
Based on a figure in [5]

and CAPTURE. Besides this there are four Capture/Compare Registers defined as CC[0], CC[1], CC[2] and CC[3] (Refer to Tables: 141 & 148 in sections 18.2 and 18.3 in the manual [5]). Also, there are two instances defined for this nRF Timer, these being TIMER & COUNTER modes (Refer to section 18.1 in the manual [5]). However, the timer used for this embedded solution design operates solely in the "Timer" mode since the Counter mode doesn't use the actual timer frequency as well as the "PRESCALER register" (Refer to Table: 147 in section 18.3 of the manual [5]) that is defined in the Timer mode.

Using the PPI functionality enables different peripherals to interact autonomously with the defined tasks and events without incurring any kind of CPU intervention. [5] This, in turn, lowers the probability of the unnecessary delays incurred during the transmission or reception due to the CPU processing time yet not eliminating all possible delays.

Note: For more details on the system components, refer to nRF51422 Product Specification [40].

The state machine as shown in Figure 6.6 utilizes a single instance of timer as

timer0, which is continuously running but is set to start counting time at specific tasks/events. It is also known that the timer stops only after one full cycle between the two nodes under consideration is complete. Then the timer may be started again for repetition of the same experiment for synchronizing each new pair of nodes within the network.

Note: The key RADIO states have been defined in section 17.1.8 in the manual [5].

There are two main considerations over here while performing synchronization in a single-hop process:

1. Firstly, considering the initial stages as shown in the state machine in Figure 6.6, once the Radio has been ramped up and is READY for transmission wherein the PACKETPTR Radio register (Refer to Table 110 of Chapter 17 in the manual [5]) has also been set along with intialization of the PPI module functionality using any of the available channels (as defined in section 16.1 in the manual [5]), at this point an initial timestamp is recorded and captured by using the CAPTURE task of the timer to and loaded into the CC[0] register. Subsequently, the captured timestamp is then loaded into the packet which is to be trasnsmitted from node 1 to 2.

   To load the timestamp value to the packet, a radio interrupt, namely, RADIO_IRQHandler is defined as in Appendix A.1. Just like the standard Interrupt functionality in the embedded systems, calling this interrupt at the READY event will spontaneously interfere with the normal execution of the program henceforth causing the desired timestamp (defined as t_o) to be loaded into the packet just before it is sent from node 1.

   Note: At this point, node 1 as one of the nRF51 boards used in the experimental analysis is acting as the transmitter.

   It is considered that this capture task is activated again once the ADDRESS and PAYLOAD events have occurred, which also implies that the packet has been successfully transmitted.

   Following this capture, the RADIO is DISABLED via the END/DISABLE shortcut and remains in this state until it is enabled again but this time for reception.

   Again the same process is repeated as that which occurred when the RADIO was in transmit mode. Now node 2 as the second nRF51 board used is enabled as the receiver so as to receive this packet from node 1.

   Note: In the receive mode, there is no requirement of calling the interrupt to interfere with the code execution since the time to be captured is when the

packet is actually received by node 2 and that will be t_{br} as shown in Figure 3.3. This time is recorded and captured using CC[1] register available.

Following the unusual delay in the processing of this packet information at node 2, now it will be node 2 acting as the transmitter to send back information to node 1.

2. Secondly, again before this packet is transmitted from node 2, the RADIO_IRQHandler as in Appendix A.1 is called in the same way as earlier so as to load all the three timestamps attained upto that point in the process. The most recent timestamp is recorded as t_{bt} as shown in Figure 3.3 when the packet is sent from node 2 and is captured and stored in CC[2] register. Therefore, t_{bt} along with the other two timestamps recorded previously as t_o and t_{br} as shown in Figure 3.3 are loaded into the packet which is sent back from node 2 to 1.

   Then again when the packet has been transmitted, there is the END/DISABLE shortcut that has been invoked to mark the end of the transmission, henceforth disabling the Radio.

   Next, the Radio is activated again but in the receiver mode for node 1 to receive the packet. Upon receipt of this packet at node 1, another timestamp is taken and recorded as t_r.

   Finally, after node 1 has received the required timing information, the Radio is DISABLED.

The entire process is repeated to synchronize time between each pair of network nodes.

6.5 Description of the Code functionality

The interoperable nature of devices designed to support the Bluetooth technology ecosystem has been mainly derived from the key elements defined in the "Bluetooth core specification" (See Core Version 5.0 in [43]) [58] [44]

Advanced Bluetooth Low Energy solutions are finding great utility in many IoT applications independent of continuous connectivity but thriving on longer battery life.

Following an Agile approach in the software development process, there were many factors that were taken into account before bringing out the final design for this BLE embedded software solution.

To achieve the desired goals, some initial sessions were undertaken to analyze, test and debug the already existing Nordic Semiconductor applications in different

Integrated Development Environments like Keil5 [18] and SEGGER Embedded Studio [19]. Some of these Nordic Semiconductor projects I had worked upon in are ble_app_beacon, ble_app_uart, blinky, ble_app_bps, ppi, radio, gpiote, timer, etc. This, in turn, gave a deeper insight into how these applications were functioning internally yet covering both the back-end as well as front-end functionalities. Furthermore, seeing how these applications have been seamlessly integrated with cross-platform efficiency served as the basis for the proposed state machine as in Figure 6.6 followed the implementation of the embedded software design as in Appendix A.1.

Based on the knowledge acquired through the prior ramp-up sessions, I had designed and tested my code for the practical analysis of the thesis work. This, in turn, was used to synchronize time events between the sending and receipt of BLE packets between multiple nodes in a broadcast mesh type network.

6.5.1 Main code functionality

Appendix A.1 involves the packet transmissions and receptions being done using nRF Radio over BLE. All the packets on air being BLE packets have a packet structure in concordance with the standard BLE packet specification as in Figure 5.1

Each packet used in this experiment was 25 bytes in size having divisions as:

1. Preamble: 1 byte (8 bits)
2. Access Address (Base + Prefix): 4 byte (32 bits)
3. S0: 1 byte
4. Length & S1: 1 byte; either Length is 0 and S1 is 1 or viceversa.
5. Payload: 12 bytes (96 bits)
6. 6 byte Address

Note: The PAYLOAD part involves 3 timestamps of 4 bytes each.

The packet contents as per the standard BLE packet layout in Figure 5.1 are shown in Appendix A.1 as

```
uint8_t packet[] = {0b01000000, 15,
0, 0xee, 0x03, 0x21, 0x35, 0x1c, 0xee, 0x07,
0x09,0,0,0,0,0,0,0,0,0,0,0,0};
```

The set of zeroes put at the last are allocated for the corresponding timestamps collected.

As discussed earlier, two crystal oscillator clocks, one being a high frequency 16M XOSC and the other being a low frequency 32k XOSC are used. Both these crystals together with the ARM Cortex-M0 processor are placed on nRF51 board. A hardware timer as timer0 is used.

The code snippet from Appendix A.1 shows the timer0_init function defined for initializing timer0. It defines the frequency, mode of operation, etc for the timer used.

```
/** @brief Function for Timer 0 initialization*/

static void timer0_init(void)
{
    nrf_drv_timer_config_t timer_cfg = NRF_DRV_TIMER_DEFAULT_CONFIG;
     timer_cfg.frequency = NRF_TIMER_FREQ_31250Hz;
     timer_cfg.mode = NRF_TIMER_MODE_COUNTER;
    ret_code_t err_code =
    nrf_drv_timer_init(&timer0, &timer_cfg, timer_event_handler);
    APP_ERROR_CHECK(err_code);
}
```

The section of the code as in Appendix A.1 shows how the PPI functionality [5] is integrated into the Radio [5] to minimize CPU intervention for certain tasks and events. A function named ppi_init is defined for initializing the PPI functionality as demonstrated in Figure 6.6.

```
/** @brief Function for initializing the PPI peripheral.
*/
static void ppi_init(void)
{
    uint32_t err_code = NRF_SUCCESS;
    err_code = nrf_drv_ppi_init();
    APP_ERROR_CHECK(err_code);

// Configure 1st available PPI channel
to stop TIMER0 counter on
TIMER1 COMPARE[0] match, which is every
even number of seconds.
    err_code = nrf_drv_ppi_channel_alloc(&ppi_channel1);
    APP_ERROR_CHECK(err_code);
```

```
    err_code = nrf_drv_ppi_channel_assign(ppi_channel1,
nrf_drv_timer_event_address_get(&timer0, NRF_TIMER_EVENT_COMPARE0),
(uint32_t)&NRF_TIMER0->TASKS_CAPTURE[1]);
    APP_ERROR_CHECK(err_code);
    // Enable PPI channel
    err_code = nrf_drv_ppi_channel_enable(ppi_channel1);
    APP_ERROR_CHECK(err_code);
   }
```

The section of the code as in Appendix A.1 shows how the PPI is set to toggle pins 21-24 (on the nRF51 board) on Radio events READY, ADDRESS, PAYLOAD, and END which can then be observed on the Saleae Logic Analyzer once the program is successfully built and loaded in SEGGER Studio IDE

```
 //PPI Channels Used for debugging
    err_code = nrf_drv_ppi_channel_alloc(&ppi_channel1);
    APP_ERROR_CHECK(err_code);
    err_code = nrf_drv_ppi_channel_assign(ppi_channel1,
    (uint32_t) (&NRF_RADIO->EVENTS_READY),
    nrf_drv_gpiote_out_task_addr_get(21));
    APP_ERROR_CHECK(err_code);

    err_code = nrf_drv_ppi_channel_alloc(&ppi_channel6);
    APP_ERROR_CHECK(err_code);
    err_code = nrf_drv_ppi_channel_assign(ppi_channel6,
    (uint32_t) (&NRF_RADIO->EVENTS_ADDRESS),
    nrf_drv_gpiote_out_task_addr_get(22));
    APP_ERROR_CHECK(err_code);

    err_code = nrf_drv_ppi_channel_alloc(&ppi_channel4);
    APP_ERROR_CHECK(err_code);
    err_code = nrf_drv_ppi_channel_assign(ppi_channel4,
    (uint32_t) (&NRF_RADIO->EVENTS_PAYLOAD),
    nrf_drv_gpiote_out_task_addr_get(23));
    APP_ERROR_CHECK(err_code);

    err_code = nrf_drv_ppi_channel_alloc(&ppi_channel3);
    APP_ERROR_CHECK(err_code);
    err_code = nrf_drv_ppi_channel_assign(ppi_channel3,
    (uint32_t) (&NRF_RADIO->EVENTS_END),
    nrf_drv_gpiote_out_task_addr_get(24));
    APP_ERROR_CHECK(err_code);
```

Then the radio and the crystal clock are initialized.

The code snippet from Appendix A.1 defines the radio_init function for initializing the Radio functionality. It defines the Radio parameters such as frequency to be set, advertising channel set to 37, setting of the Radio registers, radio mode, etc.

```
void radio_init(){
    uint32_t err_code = NRF_SUCCESS;

    //Set frequency, advertising channel 37.
    NRF_RADIO->FREQUENCY = 2UL;  // Frequency bin 2, 2402MHz

    //Set whitening to same as channel
    NRF_RADIO->DATAWHITEIV = 37;
    NRF_RADIO->TXADDRESS = 0; //Base0 + prefix0
    NRF_RADIO->RXADDRESSES = 1;

    //Set TXPOWER
    NRF_RADIO->TXPOWER = RADIO_TXPOWER_TXPOWER_0dBm;

     uint32_t override_val = NRF_FICR->OVERRIDEEN &
                             FICR_OVERRIDEEN_BLE_1MBIT_Msk;

  if (override_val == FICR_OVERRIDEEN_BLE_1MBIT_Override) {
    NRF_RADIO->OVERRIDE0 = NRF_FICR->BLE_1MBIT[0];
    NRF_RADIO->OVERRIDE1 = NRF_FICR->BLE_1MBIT[1];
    NRF_RADIO->OVERRIDE2 = NRF_FICR->BLE_1MBIT[2];
    NRF_RADIO->OVERRIDE3 = NRF_FICR->BLE_1MBIT[3];
    NRF_RADIO->OVERRIDE4 = NRF_FICR->BLE_1MBIT[4];
  }
```

The code snippet from Appendix A.1 sets the MODE register in the Radio.

```
    NRF_RADIO->MODE = RADIO_MODE_MODE_Nrf_1Mbit;
```

The section of the code as in Appendix A.1 sets the initial value to the CRCINIT register as

```
    //Set CRC initial value
    NRF_RADIO->CRCINIT = 0x555555;
```

The section of the code as in Appendix A.1 defines the CRCCNF Radio register.

```
    //Set CRC size and address include
    NRF_RADIO->CRCCNF =
            (3 << RADIO_CRCCNF_LEN_Pos)  |
            (true << RADIO_CRCCNF_SKIPADDR_Pos);
```

The code snippet from Appendix A.1 defines CRCPOLY register as

```
NRF_RADIO->CRCPOLY = (1<<24) | (1<<10) | (1<<9) | (1<<6) | (1<<4) |
(1<<3) | (1<<1) | 1;
```

The code snippet from Appendix A.1 defines PCNF0 and PCNF1 Radio registers as

```
//Set size of length, s0 and s1
    NRF_RADIO->PCNF0 =
            (1 << RADIO_PCNF0_S0LEN_Pos) |
            (6 << RADIO_PCNF0_LFLEN_Pos) |
            (2 << RADIO_PCNF0_S1LEN_Pos);

    NRF_RADIO->PCNF1 =
            (100 << RADIO_PCNF1_MAXLEN_Pos) |
            (0 << RADIO_PCNF1_STATLEN_Pos)  |
            (3 << RADIO_PCNF1_BALEN_Pos)    |
            (RADIO_PCNF1_ENDIAN_Little << RADIO_PCNF1_ENDIAN_Pos)   |
            (true << RADIO_PCNF1_WHITEEN_Pos);
```

The section of the code as in Appendix A.1 sets the Access address shown in Figure 5.1 for Bluetooth as defined in "Core Version 5.0" [43].

```
    //Set Radio address
    set_address0(0x8E89BED6);
```

The code snippet from Appendix A.1 defines another function to initialize the crystal clock as

```
void start_crystal_clock(){
    // Start 16 MHz crystal oscillator
    NRF_CLOCK->EVENTS_HFCLKSTARTED = 0;
     * (int *) 0x40000000 = 1;
```

```
    // Wait for the external oscillator to start up
    while (NRF_CLOCK->EVENTS_HFCLKSTARTED == 0)
    {
        // Do nothing.
    }
     /* Start low frequency crystal oscillator for app_timer(used by bsp)*/
    NRF_CLOCK->LFCLKSRC = (CLOCK_LFCLKSRC_SRC_Xtal << CLOCK_LFCLKSRC_SRC_Pos);
    NRF_CLOCK->EVENTS_LFCLKSTARTED = 0;
    NRF_CLOCK->TASKS_LFCLKSTART    = 1;

    while (NRF_CLOCK->EVENTS_LFCLKSTARTED == 0)
    {
        // Do nothing.
    }

}
```

The section of the code as in Appendix A.1 invokes the shortcut between the END event and the DISABLE task within a common peripheral such that enabling this shortcut triggers the DISABLE task when the END event is generated. The main advantage of invoking this shortcut is to reduce the propagation delay with respect to that encountered with PPI. Henceforth, the END>DISABLE shortcut is enabled as

```
NRF_RADIO->SHORTS = (1 << RADIO_SHORTS_END_DISABLE_Pos);
```

Once the timer0 has started, the section of the code as in Appendix A.1 defines the Radio interrupt, namely, RADIO_IRQHandler that is to be enabled on READY event as

```
void RADIO_IRQHandler(void) {
    NRF_RADIO->EVENTS_READY = 0;

    uint8_t *packet = (uint8_t *) NRF_RADIO->PACKETPTR;

    NRF_TIMER0->TASKS_CAPTURE[0]=1;
    uint32_t transmit_time = NRF_TIMER0->CC[0];
    NRF_GPIO->OUT ^= (1 << 21);
    write_uint32(&packet[10], ack_time);
    write_uint32(&packet[14], recv_time);
```

```
    write_uint32(&packet[18], transmit_time);
    NRF_RADIO->TASKS_START = 1;
}
```

The code snippet from Appendix A.1 shows that with Radio in transmit mode, there are two situations that occur alternatively; these being sending/transmitting packets and waiting for the packets. Here the PACKETPTR Radio register is set individually for the advertised packet as well as the sent packet. During transmission, the packet pointed to via this 32 bit (4 byte) address is to be transmitted. Since the PPI functionality is merged with the Radio, henceforth some pre-programmed channels as defined in [5] are also set. This is done as

```
void send_packet(char* send_packet){
    //NRF_RADIO->PACKETPTR = (uint32_t) adv_packet;

    //Set packet pointer
    NRF_RADIO->PACKETPTR = (uint32_t) send_packet;

    NRF_RADIO->EVENTS_DISABLED = 0U;
    NRF_RADIO->EVENTS_READY = 0U;
    NRF_RADIO->TASKS_TXEN   = 1U;


    while (NRF_RADIO->EVENTS_DISABLED == 0U)
    {
        // wait
    }

}
```

Similarly, with Radio in the receive mode, the code snippet from Appendix A.1 shows that the PACKETPTR register contains the 32 bit (4 byte)address to which the received packet will be written to. This is done as

```
void start_receiver(char* receive_packet){
    //Set packet pointer
    NRF_RADIO->PACKETPTR = (uint32_t) receive_packet;

    NRF_RADIO->EVENTS_READY = 0U;
    // Enable radio and wait for ready
    NRF_RADIO->TASKS_RXEN = 1U;
    while (NRF_RADIO->EVENTS_READY == 0U)
```

```
    {
        // wait
    }

    NRF_RADIO->EVENTS_END = 0U;
    // Start listening and wait for address received event
    NRF_RADIO->TASKS_START = 1U;


    // Wait for end of packet or buttons state changed
    while (NRF_RADIO->EVENTS_END == 0U)
    {
        // wait
    }
    NRF_TIMER0->TASKS_CAPTURE[1]=1;
    recv_time = NRF_TIMER0->CC[1];

    ack_time = read_uint32(&receive_packet[18]);
}
```

Furthermore, the section of the code as in Appendix A.1 defines the dual functionality of the Radio as a transmitter as well as a receiver.

```
while(1){
        // Transmit
        NRF_GPIO->OUTSET = (1 << 24);

        send_packet(packet);

        NRF_GPIO->OUTCLR = (1 << 24);

        // Receive
        start_receiver(my_array);
        // TODO look in my_array, extract the two timestamps, print them out

    }
```

Note: The same code as in Appendix A.1 works for both the transmit and the receive cases. First, when the packet is sent from node 1 keeping radio in transmit mode. Then once the packet is transmitted, the radio switches to the receive mode with node 2 receiving the packet. Following this, the process repeats but now node 2 is to transmit while node 1 receives the packet. This has also been clearly demarcated in the state machine as shown in Figure 6.6.

The code snippet from Appendix A.1 demonstrates that upon the RADIO_IRQHandler being called on the READY event (being similar to a function call in C), the desired timestamp that has been captured and stored in the Capture/Compare register is loaded into the Radio packet to be transmitted. This is done as

```
NVIC_ClearPendingIRQ(RADIO_IRQn);
    NVIC_EnableIRQ(RADIO_IRQn);
    APP_ERROR_CHECK(NRF_LOG_INIT(timestamp));
    NRF_LOG_INFO("NRF_TIMER0->CC[0] = %d", NRF_TIMER0->CC[0]);
```

Note: At this point, the captured timestamp is written to the packet to which the PACKETPTR register points to.

Also, the section of the code as in Appendix A.1 shows that at the third READY event on the transmitter side with node 2 acting as the transmitter, the three desired timestamps that have been captured and stored in the respective Capture/Compare registers are loaded into the Radio packet to be transmitted back to node 1. This is done as

```
 NVIC_ClearPendingIRQ(RADIO_IRQn);
    NVIC_EnableIRQ(RADIO_IRQn);
    APP_ERROR_CHECK(NRF_LOG_INIT(timestamp));
    NRF_LOG_INFO("NRF_TIMER0->CC[0]+CC[1]+CC[2] = %d",
                  NRF_TIMER0->CC[0]
                  +NRF_TIMER0->CC[1]
                  +NRF_TIMER0->CC[2]);
```

Note: This stage with the recorded timestamp(s) being loaded into the packet does not facilitate reception yet.

Although the same code works for both the transmissions as well as the receptions on Radio yet as explained earlier, there are many sources of internal delays in the system that cannot be ignored for practical considerations. Moreover, this fact is even more clear from the designed state machine as shown in Figure 6.6 which demonstrates the transmissions and the receptions individually.

Once the main program as in Appendix A.1 had been successfully built and loaded onto the nRF51 board, then the Saleae Logic Analyzer was used to interpret the time durations (delays) between the Radio events. Here, the first toggle occurring on the READY event was corresponding to the "GPIO Tasks and Events (GPIOTE)" (Refer to Chapters 14 and 15 in the manual [5]) toggle.

6.5.2 Other key code files

nRF51 SDK v12.2 has been used in the SDLC which provides for a wide range of drivers and other libraries for further considerations like in test procedures, etc.

Apart from the main code as in Appendix A.1, there are a series of other important files which serve as a functional backbone in the software development process. These have been listed further.

1. sdk_config.h- This header file as shown in Appendix A.2 serves as a rich source of information about the varied modules available providing a huge set of functionalities to enable and disable these modules (Refer to the manual [5]) as well as configuring them.

2. memorymap.xml & flash_placement.xml- Both of these files as shown in Appendix A.6 and Appendix A.5 respectively play a key role in the RAM memory organization of the system. While memorymap.xml defines the allocated address space for each of the RAM as well as flash memory, the flash_placement.xml file defines where the program is exactly located within the address space allocated for the flash. These files were the key locators for the specific addresses within the RAM memory (SRAM) and the flash memory address space. The flash memory and the Static RAM (SRAM) for this project in SEGGER Studio IDE were allocated 15.8 KB and 2.9 KB respectively of the total available memory.

3. nrf_drv_timer.h- This header file as in Appendix A.4 defined the interface available for the available timer driver.

4. nrf_drv_timer.c- This c file as in Appendix A.3 accounted for the implementation of the provided timer driver.

Note: Both the nrf_drv_timer.h & nrf_drv_timer.c files as shown in Appendix A.4 & Appendix A.3 respectively acted as a source of abstraction over the hardware timers.

6.6 Procedure

The experiment has been performed on the basis of the Tiny-Sync algorithm which relies on synchronizing time between the network nodes via timestamping of real time events as well as collecting and correlating data samples. The theoretical explanations of these algorithms, as discussed earlier, focus on their performance efficiency. But the accuracy attained by modeling the system behavior can be truly realized with the help of practical implementation of the same.

A ble-broadcast-mesh type network is taken under consideration which shows a behavior similar to that of WSNs. The underlying strategy for conducting this experiment is more or less similar to that discussed earlier for one hop and five hops cases. In order to analyze the performance level of the Tiny-Sync and Mini-Sync algorithms, multiple sets of data points have been generated. To have a more clear picture as to how these generated data points correlate to estimate the drift, offset and other external factors of delay in such systems running with timers, effective logical reasoning has been provided to see how time is synchronized throughout the network.

Instead of a message probe being sent in a fixed duration over a random time interval as illustrated in the experiment in [2], here we have BLE packets carrying the timing information maintained by the local clocks at each of the nodes. Considering these packets to be simultaneously transmitted on air over RADIO [5] in a real time phenomenon, the timestamps recorded at each of the corresponding transmit and receive points are loaded into the respective packets just before they are transmitted.

In this manner, by having a time zero event (t_o) defined within one cycle consisting of two transmissions and two receptions as well as by timestamping these real time events, the network nodes are mutually synchronized counting the time in an identical manner henceforth enhancing the reliability of the transmitted data within the network. The single timer instance being used here is known to be constantly running in addition to being set to record the desired time events within the network. The timer should not be stopped before one cycle completes and that all the four desired timestamps have been attained as illustrated by the Tiny-sync algorithm.

6.7 Analysis of the Results

The analysis part was carried out using a set of tools from Nordic semiconductor and others. These include the Saleae Logic Analyzer, Segger RTT Viewer, Sniffer software, Wireshark, etc.

The experiment has been performed for advertising BLE packets on radio. Sniffer was just used once the main code had been properly built and loaded in Segger. nRF51 Dongle used in conjunction with the Sniffer software and Wireshark tools acted as a sniffing device to verify that the packets being sniffed were corresponding to the connected device. For this it was needed to ensure that the sniffer firmware was flashed properly on the nRF51 boards (nRF51 Board specifications: PCA10028 (V 1.1.0 & V 1.2.0)). Wireshark was used alongside sniffer to capture the desired network traffic.

For advertising, we have the access address of Bluetooth as a hex value ($0x8E89BED6$)

as shown in Appendix A.1 from the Bluetooth core Specification (Refer to Core Version 5.0 in [43]). This is defined in the radio_init function as

```
//Set Radio address
set_address0(0x8E89BED6);
```

It is important to note that these tools were used as a source of verification to see whether or not the device being programmed was functioning as expected. There is no accountability for the packet receive times in Wireshark to be interpreted to be precise enough to yield any meaningful conclusions on the relative drift and relative offset between the clocks owing to the uncertainty of the internal/external delay components in the system.

6.7.1 Results from the Saleae Logic Analyzer

The main interpretation of the timing information (delays) between the varied Radio events was brought about with the help of Logic Analyzer hardware tool used in conjunction with the Logic Software. The four Radio events under consideration being the READY, ADDRESS, PAYLOAD, and END events as discussed earlier. For the practical analysis of the Tiny-Sync algorithm, while some samples were generated with node 1 (one of the two nRF51 boards) as the transmitter and node 2 (second nRF51 board) as the receiver, the same procedure was also repeated taking node 2 as the transmitter and node 1 as the receiver as illustrated in the algorithm.

The experiment had been repeated a number of times for an accurate analysis with several data point samples which, in turn, has put forward some logical results and conclusions derived from them.

Considering the standard packet size of 25 bytes for all transmitted packets, only the last 20 bytes of the packet (neglecting the 1 byte corresponding to the Preamble and the 4 bytes allocated for the Access address) are taken into account to see whether the transmitted packets are showing the expected behavior or not.

As seen from the initial set of samples generated, these show the time between the toggle events on READY, time between the toggles on the READY and ADDRESS events as well as time between the toggles on the ADDRESS and PAYLOAD events.

Here, node 1 as one nRF51 device acted as the transmitter and node 2 as another nRF51 device acted as the receiver.

The following key points need to be carefully considered for taking the measurements on the Saleae Logic 1.2.12 software using the Logic Analyzer hardware tool connected to the nRF51 board.

1. Each toggle represents a change of state from high to low or viceversa.

2. The first toggle on the READY event corresponds to the GPIO Tasks and Events (GPIOTE) toggle and is neglected. This owing to the fact that this toggle is due to resetting of the connected nRF51 board. So all the measurements begin from the second toggle on the READY event.

3. Most of the measurements on the analyzer were done at a sampling rate of 50 MS/s for intervals of 5 seconds and 10 seconds. There were some measurements done at 100 MS/s sampling rate for 5 seconds and 10 seconds.

4. There are four channels selected. These being channel 0, channel 1, channel 2, and channel 3 corresponding to the pins 21, 22, 23 and 24 respectively on the nRF51 DK along with the associated Radio events as READY, ADDRESS, PAYLOAD and END respectively.

Figure 6.8 shows the first sample recorded as the time between the toggle events on READY, starting from the second toggle since the first toggle just corresponds to the GPIOTE toggle. This sample is recorded as 1.02747914 seconds. This was done using one channel only, that is, channel 0 for the READY event under consideration.

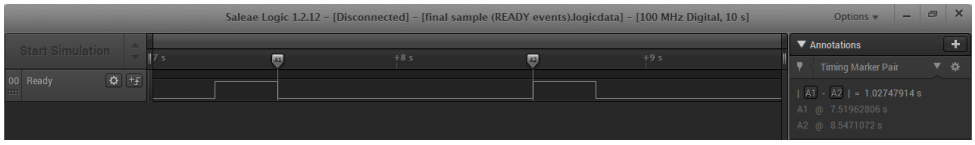

Figure 6.8: *Time between the second and third toggle on READY at sampling rate* 100 *MS/s for* 10 *seconds duration; the first toggle corresponds to the GPIOTE and is not considered for time measurements*

Next, Figure 6.9 on the following page shows the next recorded sample from when the RADIO is ramped up to the actual START of the radio for transmission. This is the time between the READY and the ADDRESS events, that is, $A1 - A2$ as recorded to be of the order of microseconds, being 61.24 microseconds (μs).

and

Figure 6.10 on the next page shows the third sample recorded at a sampling rate of 50 MS/s for a duration of 5 seconds. The time recorded between the ADDRESS and PAYLOAD events is in milliseconds unlike the time between the READY and ADDRESS as recorded previously. This time recorded as $B1 - B2$ equals 0.16106 milliseconds (ms).

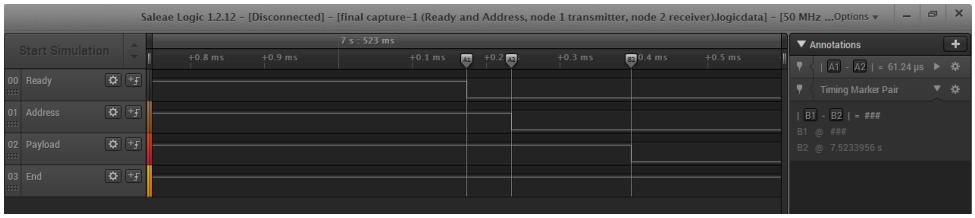

Figure 6.9: *Time between READY and ADDRESS at sampling rate* 50 *MS/s for* 5 *seconds duration*

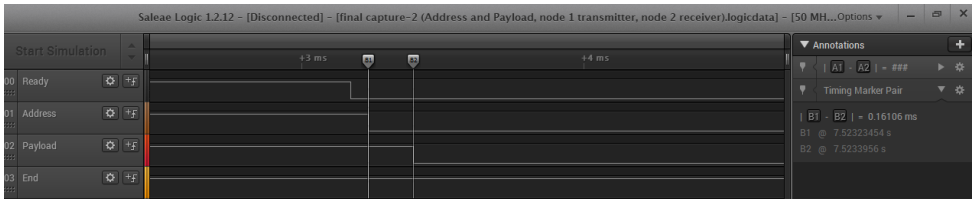

Figure 6.10: *Time between ADDRESS and PAYLOAD at sampling rate* 50 *MS/s for* 5 *seconds duration*

From Figure 6.9, it is evident that there will always be some delay after the Radio has ramped up to the start of the transmission on radio which cannot be ignored considering the practical implications of the same.

Moreover, interpreting the results obtained from the first set of samples as shown in Figure 6.8 on the previous page, Figure 6.9 and Figure 6.9, the following conclusions can be drawn:

1. Firstly, when measuring the time between the consecutive toggles on READY leaving the first toggle, it is usually observed to be of the order of seconds. This is because this time interval corresponds to the time between the READY event facilitating the transmission from one node with Radio being in transmit mode to the READY when the Radio switches to the receive mode for reception on the other node or viceversa. This is also clearly demarcated by the state machine as in Figure 6.6.

2. Secondly, the time between the READY and the actual start of the transmission, that is, time between READY and ADDRESS is mostly observed as a delay in microseconds.

3. Thirdly, the next time recorded between the ADDRESS and the PAYLOAD is mostly observed as delay of the order of milliseconds, henceforth being smaller

than the previously recorded delay between READY and ADDRESS.

Moreover, interpreting this time value of 0.16106 milliseconds with the last 20 bytes of the packet being used, 0.16106 milliseconds equals 160 microseconds approximately. And 20 bytes equals 160 bits. This gives a precise picture as for the transmission of each bit henceforth proving that the BLE packet being transmitted is behaving as it is expected to do so.

In the same manner, multiple samples are taken instead of just one or two so as to acquire a better understanding of the timing behaviour among the varied RADIO events on the Saleae Logic. All the samples are recorded on the Logic in a unique manner where:

1. $$A1 - A2$$

   represents the time between READY and ADDRESS events.

2. $$B1 - B2$$

   represents the time between the ADDRESS and the PAYLOAD events.

3. $$C1 - C2$$

   represents the time between the PAYLOAD and END events.

Adding on, in order to take reliable measurements, one main consideration here was to take into account the temperature readings for sampling the time events on the analyzer. As per the nRF51 reference manual [5], the nRF51 DK has a specialized temperature sensor (Refer to Chapter 22 in the manual [5]) for which there is a START as a triggering task. Furthermore, a register called as TEMP register is there to store the recorded temperature value and when needed, this value can be read from this register. [5]

For the initial temperature readings, the temperature of the PC was noted to be 25.8 degree Celsius. The board acting as the transmitter and being connected to the Logic Analyzer hardware as well as the PC showed a temperature range of 25.4 to 28.2 degrees on the Celsius scale. While the other nRF51 board acting as the receiver and connected nearby to the transmitting device via a hub showed a temperature of the range 22.6 to 25.4 degrees on the same scale.

Considering these conditions, the time events were sampled on the analyzer in the same manner as earlier but within a fixed temperature range.

Figure 6.11 shows the time between the READY and ADDRESS, that is, $A1 - A2$, as 61.26 microseconds when sampling is done on the Saleae Logic at a rate of 50 MS/s for a duration of 10 seconds.

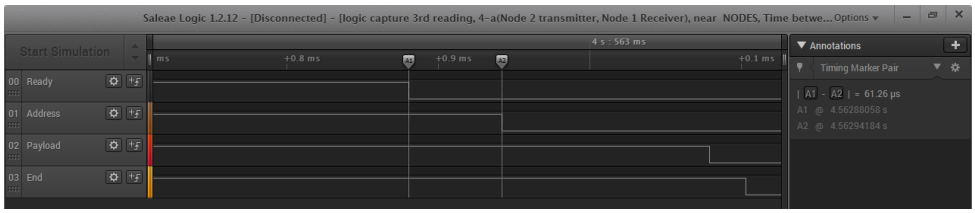

Figure 6.11: *Time between READY and ADDRESS at sampling rate* 50 *MS/s for* 10 *seconds duration*

Next, Figure 6.12 shows the time between the ADDRESS and PAYLOAD, that is, $B1 - B2$, as 0.13702 milliseconds when sampling is done at a rate of 50 MS/s for a duration of 10 seconds as

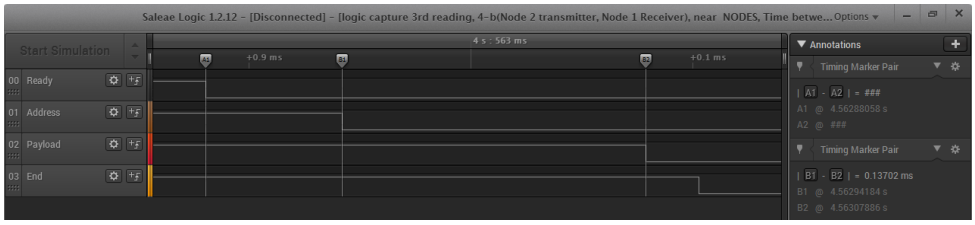

Figure 6.12: *Time between ADDRESS and PAYLOAD at sampling rate* 50 *MS/s for* 10 *seconds duration*

Finally, Figure 6.13 on the facing page shows the time between the PAYLOAD and END, that is, $C1 - C2$, as 24.06 microseconds when sampling is done at a rate of 50 MS/s for a duration of 10 seconds as

As seen from the above three samples as in Figure 6.11, Figure 6.12 and Figure 6.13, while the time measured between the READY and ADDRESS as well as PAYLOAD and END is in microseconds, that between the ADDRESS and PAYLOAD is in milliseconds being the longest time duration among the three time values recorded.

Yet another set of samples were recorded under the same temperature readings mentioned earlier. But this time the sampling was done at 50 MS/s for 5 seconds

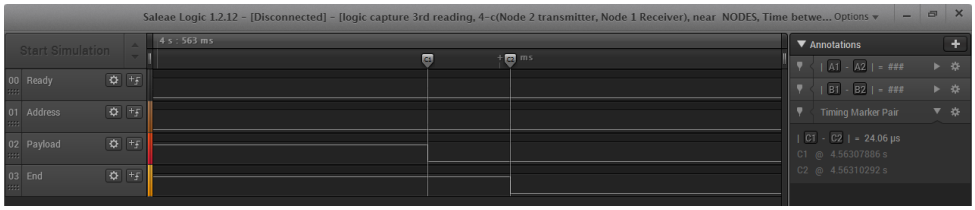

Figure 6.13: *Time between ADDRESS and PAYLOAD at sampling rate* 50 *MS/s for* 10 *seconds duration*

duration.

Figure 6.14 shows the time between the READY and ADDRESS, that is, $A1 - A2$, as 61.24 microseconds when sampling is done on the Saleae Logic at a rate of 50 MS/s for a duration of 5 seconds as

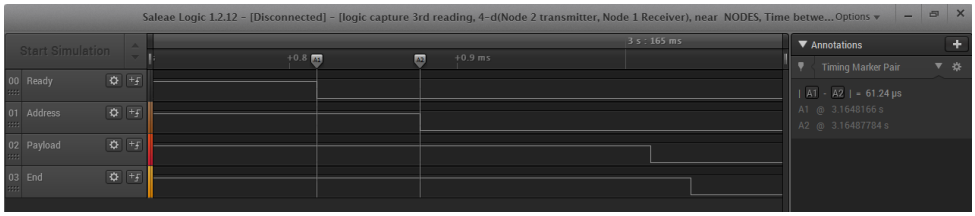

Figure 6.14: *Time between READY and ADDRESS at sampling rate* 50 *MS/s for* 5 *seconds duration*

Next, Figure 6.15 shows the time between the ADDRESS and PAYLOAD, that is, $B1 - B2$, as 0.13702 milliseconds when sampling is done at a rate of 50 MS/s for a duration of 5 seconds as

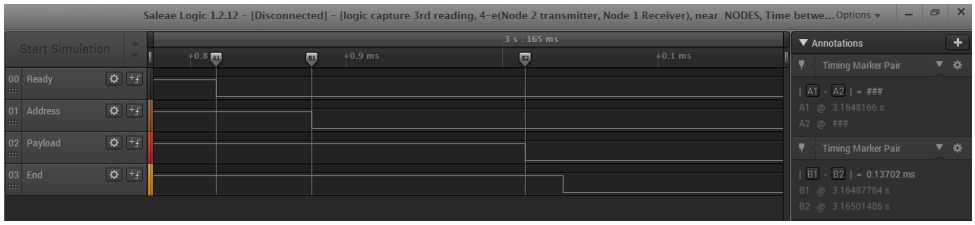

Figure 6.15: *Time between ADDRESS and PAYLOAD at sampling rate* 50 *MS/s for* 5 *seconds duration*

Finally, Figure 6.16 shows the time between the PAYLOAD and END, that is, $C1 - C2$, as 24.06 microseconds when sampling is done at a rate of 50 MS/s for a duration of 5 seconds as

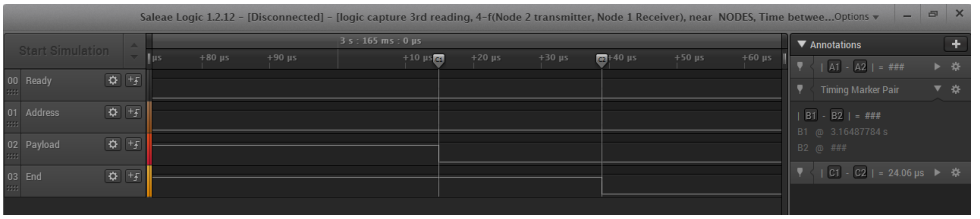

Figure 6.16: *Time between PAYLOAD and END at sampling rate* 50 *MS/s for* 5 *seconds duration*

It is seen from Figure 6.14, Figure 6.15 and Figure 6.16 that the only difference from the previous set of three values recorded as shown in Figure 6.11, Figure 6.12 and Figure 6.13 is seen in the time between the READY and ADDRESS in Figure 6.11 and Figure 6.14 while the other time durations are precisely the same. This difference is observed mainly due to the reduced sampling duration in the latter when recording the readings.

Furthermore, while the main purpose of using the Logic Analyzer tool was to interpret the timing behavior between multiple radio events as in READY, ADDRESS, PAYLOAD and END so as to analyze the expected packet behavior yet there were a series of actions that were taking place just when the RADIO was ramped up and READY for transmission as shown in the state machine in Figure 6.6. Therefore, certain measurements were recorded considering only the READY event on the channel0 of the analyzer. So just pin 21 on the nRF51 board was connected now instead of connecting all the four pins from $21 - 24$ for analyzing multiple RADIO events. A set of readings for taken at varying sampling rates as well as sampling durations.

Figure 6.17 on the facing page shows the time between the consecutive toggles on the READY event as 0.25307266 seconds at sampling rate 100 MS/s for 10 seconds duration.

Figure 6.18 on the next page shows the time between the consecutive toggles on the READY event as 1.13702865 seconds at sampling rate 100 MS/s for 5 seconds duration.

Figure 6.19 on the facing page shows the time between the consecutive toggles on the READY event as 2.68701526 seconds at sampling rate 50 MS/s for 10 seconds duration.

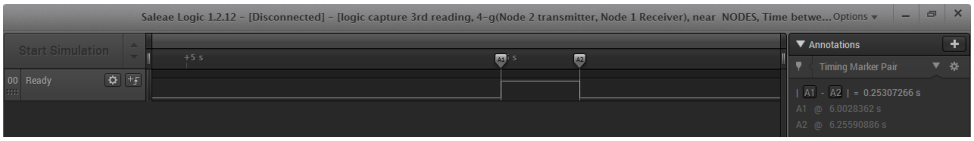

Figure 6.17: *Time between consecutive toggles on the READY event at sampling rate* 100 *MS/s for* 10 *seconds duration*

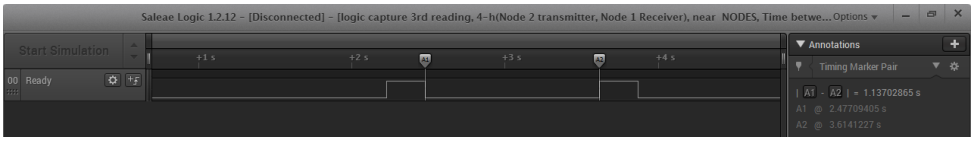

Figure 6.18: *Time between consecutive toggles on the READY event at sampling rate* 100 *MS/s for* 5 *seconds duration*

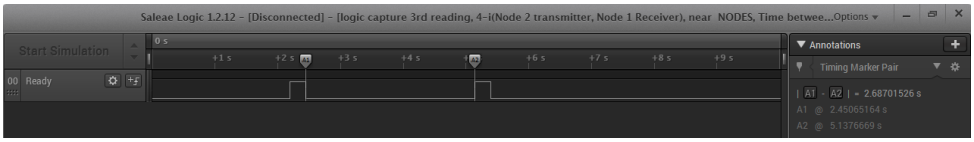

Figure 6.19: *Time between consecutive toggles on the READY event at sampling rate* 50 *MS/s for* 10 *seconds duration*

Figure 6.20 shows the time between the consecutive toggles on the READY event as 1.33695224 seconds at sampling rate 50 MS/s for 5 seconds duration.

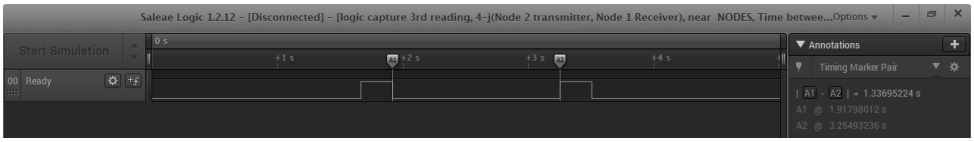

Figure 6.20: *Time between consecutive toggles on the READY event at sampling rate* 50 *MS/s for* 5 *seconds duration*

From Figure 6.17 and Figure 6.18, it is evident that the time between the consecutive toggles on READY while sampling at 100 MS/s rate is quadruple (four times more) for a duration of 5 seconds than for a duration of 10 seconds at the same sampling rate.

Furthermore, from Figure 6.19 and Figure 6.20, it can be inferred that considering

the time between the consecutive toggles on READY, but now at sampling rate being half of that used earlier, that is, 50 MS/s, the recorded time interval for a duration of 10 seconds is approximately twice that of the time recorded for 5 seconds duration at the same sampling rate of 50 MS/s.

The experiment was further repeated for more samples.

Another set of readings were taken at 18.4 degrees on the Celsius scale. Both the communicating devices (nRF51 boards) showed the same temperature.

Figure 6.21 shows the time between the READY and ADDRESS, that is, $A1 - A2$, as 61.26 microseconds when sampling is done on the Saleae Logic at a rate of 50 MS/s for a duration of 5 seconds as

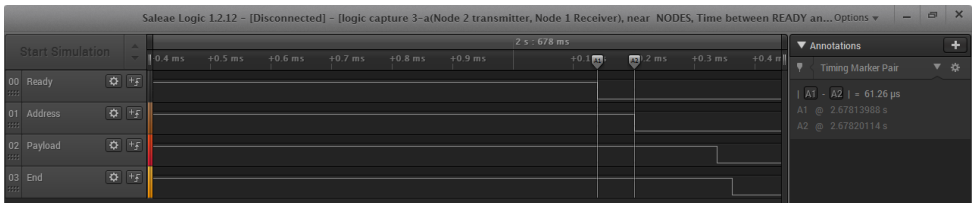

Figure 6.21: *Time between READY and ADDRESS at sampling rate* 50 *MS/s for* 5 *seconds duration*

Figure 6.22 shows the time between the ADDRESS and PAYLOAD, that is, $B1 - B2$, as 0.137 milliseconds when sampling is done on the Saleae Logic at a rate of 50 MS/s for a duration of 5 seconds as

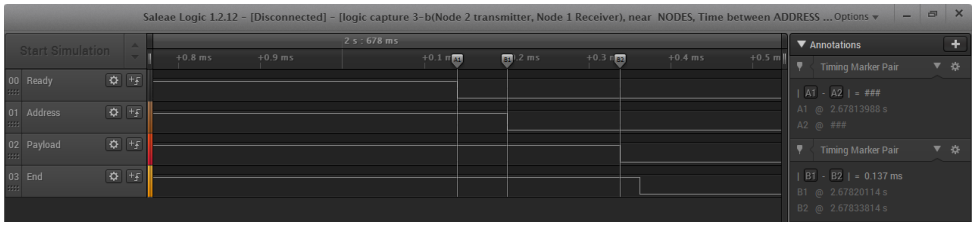

Figure 6.22: *Time between ADDRESS and PAYLOAD at sampling rate* 50 *MS/s for* 5 *seconds duration*

Figure 6.23 on the facing page shows the time between the PAYLOAD and END, that is, $C1 - C2$, as 24.08 microseconds when sampling is done on the Saleae Logic at a rate of 50 MS/s for a duration of 5 seconds as

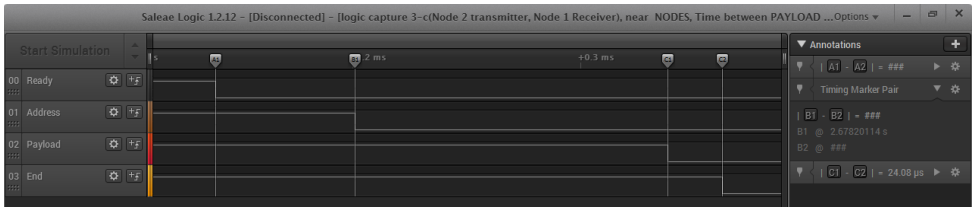

Figure 6.23: *Time between the PAYLOAD and END at sampling rate* 50 *MS/s for* 5 *seconds duration*

In the above three time samples recorded as in Figure 6.21, Figure 6.22 and Figure 6.23 respectively, difference is mainly seen in the time between ADDRESS and PAYLOAD as well as PAYLOAD and END events as shown in Figure 6.22 and Figure 6.23 respectively.

6.7.2 Debugging with Segger RTT viewer

Along with the results from the Logic Analyzer, it was equally important to capture the desired timestamp values via debugging the *C* code in real time. Since SEGGER Embedded Studio has been used as the platform for embedded software development, therefore nothing could have been better for performing this task than a flexible debugger like the Segger RTT viewer/J-Link RTT viewer which has been discussed earlier. The jlink version used was 5.10n with reference to [28]. A detailed tutorial [48] was extremely useful for getting familiar with analyzing the system behavior in real time.

Furthermore, it was required that to enable the logging functionality over the RTT, the logger module had to be enabled. To achieve this, some key modifications were made in the sdk_config.h file as shown in Appendix A.2.

- `#define NRF_LOG_ENABLED 1`
- `#define NRF_LOG_BACKEND_SERIAL_USES_UART 0`
- `#define NRF_LOG_BACKEND_SERIAL_USES_RTT 1`
- `#define NRF_LOG_DEFERRED 0`

Moreover, as shown in Appendix A.1, a function returning a 32 bit timestamp value had been defined as

```
uint32_t timestamp()
{
  return NRF_TIMER0->CC[0];
}
```

Then in the main function in the code shown in Appendix A.1, after calling the radio interrupt handler to enable the interrupt functionality at the start, the defined timestamp function was called.

```
APP_ERROR_CHECK(NRF_LOG_INIT(timestamp));
NRF_LOG_INFO("NRF_TIMER0->CC[0] = %d", NRF_TIMER0->CC[0]);
```

Following these steps, the RTT could successfully log the captured timestamps on the window.

In order to effectively log the timestamps, the Segger RTT Viewer also needs to be operated following an ordered sequence of steps.

1. The main code is compiled and flashed onto the device. This is done by using the $'Build'$ option in SEGGER Studio IDE. Since pca10028 nRF51 boards were used, there is an option under Build as build ppi_pca10028 to compile the C code followed by which Build and Run option under Build is selected for flashing the program onto the device.

2. Next, by configuring the settings on the J-Link RTT Viewer (the version used was $5.10n$) as in [48], the RTT Viewer window pops-up where it is to confirmed that the RTT Viewer was actually connected to the device that has been programmed. This is done by clicking on the Log option upon which some information is displayed confirming the connection.

   ```
   J-Link RTT Viewer V5.10n: Logging started.
   LOG:  Global terminal added.
   LOG:  Terminal 0 added.
   LOG:  Device "NRF51422\_XXAC" selected.
   LOG:  Found SWD-DP with ID 0x0BB11477
   LOG:  Found Cortex-M0 r0p0, Little endian.
   LOG:  FPUnit: 4 code (BP) slots and 0 literal slots
   LOG:  CoreSight components:
   LOG:  ROMTbl 0 @ F0000000
   LOG:  ROMTbl 0 [0]: F00FF000, CID: B105100D, PID: 000BB471 ROM Table
   LOG:  ROMTbl 1 @ E00FF000
   LOG:  ROMTbl 1 [0]: FFF0F000, CID: B105E00D, PID: 000BB008 SCS
   ```

```
LOG:  ROMTbl 1 [1]: FFF02000, CID: B105E00D, PID: 000BB00A DWT
LOG:  ROMTbl 1 [2]: FFF03000, CID: B105E00D, PID: 000BB00B FPB
LOG:  ROMTbl 0 [1]: 00002000, CID: B105900D, PID: 000BB9A3 ???
LOG:  RTT Viewer connected.
```

3. Go to Input → Sending and click Send on Enter.[48]

4. To test and see whether the connected board is being able to properly communicate with the RTT, an $'r'$ is typed in the text field followed by pressing enter. This would show the connected board resetting.

   Note: If multiple boards are connected at the same time, then the appropriate serial number needs to selected corresponding to the device required to communicate with this Real Time Terminal as explained in [48]

5. Once this is done, then the logging is started by clicking Data → Start Logging as shown in the text.

```
J-Link RTT Viewer V5.10n: Logging started.
LOG:  Global terminal added.
LOG:  Terminal 0 added.
LOG:  Device "NRF51422\_XXAC" selected.
LOG:  Found SWD-DP with ID 0x0BB11477
LOG:  Found Cortex-M0 r0p0, Little endian.
LOG:  FPUnit: 4 code (BP) slots and 0 literal slots
LOG:  CoreSight components:
LOG:  ROMTbl 0 @ F0000000
LOG:  ROMTbl 0 [0]: F00FF000, CID: B105100D, PID: 000BB471 ROM Table
LOG:  ROMTbl 1 @ E00FF000
LOG:  ROMTbl 1 [0]: FFF0F000, CID: B105E00D, PID: 000BB008 SCS
LOG:  ROMTbl 1 [1]: FFF02000, CID: B105E00D, PID: 000BB00A DWT
LOG:  ROMTbl 1 [2]: FFF03000, CID: B105E00D, PID: 000BB00B FPB
LOG:  ROMTbl 0 [1]: 00002000, CID: B105900D, PID: 000BB9A3 ???
LOG:  RTT Viewer connected.
LOG:  Logger started.
```

6. Furthermore, on selecting 'Terminal 0' or 'All Terminals', the data displayed pertains to the captured timestamp values stored in the four Capture/Compare registers, namely CC[0], CC[1], CC[2], and CC[3] (see Table 148 in section 18.3 in [5]).

   On selecting Terminal 0 option, the data is displayed.

```
[8d]APP:INFO:NRF\_TIMER0->CC[0] =
0[8d]APP:INFO:NRF\_TIMER0->CC[0]+CC[1]+CC[2] = 5
```

and

On selecting All Terminals, the data is displayed.

```
0>  [8d]APP:INFO:NRF\_TIMER0->CC[0] =
0[8d]APP:INFO:NRF\_TIMER0->CC[0]+CC[1]+CC[2] = 5
```

7. Henceforth, the captured data appears on the J-Link window along with the Log details. But it is equally important to terminate the logging time interval once the desired data has been logged. This is done by clicking Data → Stop Logging which terminates the Logging session.

```
-Link RTT Viewer V5.10n: Logging started.
LOG:  Global terminal added.
LOG:  Terminal 0 added.
LOG:  Device "NRF51422\_XXAC" selected.
LOG:  Found SWD-DP with ID 0x0BB11477
LOG:  Found Cortex-M0 r0p0, Little endian.
LOG:  FPUnit: 4 code (BP) slots and 0 literal slots
LOG:  CoreSight components:
LOG:  ROMTbl 0 @ F0000000
LOG:  ROMTbl 0 [0]: F00FF000, CID: B105100D, PID: 000BB471 ROM Table
LOG:  ROMTbl 1 @ E00FF000
LOG:  ROMTbl 1 [0]: FFF0F000, CID: B105E00D, PID: 000BB008 SCS
LOG:  ROMTbl 1 [1]: FFF02000, CID: B105E00D, PID: 000BB00A DWT
LOG:  ROMTbl 1 [2]: FFF03000, CID: B105E00D, PID: 000BB00B FPB
LOG:  ROMTbl 0 [1]: 00002000, CID: B105900D, PID: 000BB9A3 ???
LOG:  RTT Viewer connected.
LOG:  Logger started.
LOG:  Logging stopped.
```

8. Finally, the Log details are recorded as

```
\# SEGGER J-Link RTT Viewer V5.10n Data Log File
\# Compiled: $18:40:23$ on Feb $19$ $2016$
\# Logging started $@$ $22$ Aug 2017 $08:24:57$
\# Logging stopped $@$ $22$ Aug $2017$ $08:35:10$
```

Chapter 7
Conclusion

The Master's thesis work on 'Time synchronization across multiple devices in network nodes' involved a thorough analysis of the time synchronization problem in networks similar to Wireless Sensor Networks. The initial part of the work focused on summarizing the existing literature study done in this regard with reference to well-known research papers on IEEE website as well as from other known authors. Based on the study done till date in considering the clock synchronization problem, varied synchronization protocols/schemes proposed, need/requirement of synchronization schemes, etc relevant conclusions were drawn. The main area of concern while summarizing the theory proposed in this area was an in depth analysis of the two well-known algorithms, namely, Tiny-Sync and Mini-Sync used to synchronize time in sensor networks.

The thesis work makes a consistent effort to analyze these algorithms in detail, and put forward their degree of efficiency in synchronizing time by interpreting well-defined mathematical equations as well as graphical and statistical results for modeling the relative drift and the relative offset between the clocks. Considering the well- known fact that the clocks are continuously drifting apart in time with respect to each other as well as with respect to a common reference or a source such as crystal oscillators, it has been important for the clocks to self-synchronize too in addition to synchronizing amongst peers.

The core part of the thesis work has been into the development of a unique embedded software solution for Nordic Semiconductor ASA in *C* using Keil5, SEGGER Embedded Studio and a variety of tools from Nordic Semiconductor and others so as to facilitate the accurate realization of the synchronizaton problem. Keeping the basic idea as proposed by the Tiny-Sync algorithm for synchronizing time between two nodes, nRF Radio had been used over Bluetooth Low Energy stack for packet data transmission between the Nordic Semiconductor nRF51 devices. The core task here was timestamping real time events in a nRF51-ble-bcast-mesh type network with hardware timers via transmitting BLE packets over Radio. Two crystal oscillators,

one being a high frequency 16M XOSC crystal and the other being a low frequency 32k XOSC crystal, placed on the nRF51 board itself, were used as reference sources. However, the main measurements were recorded for the high frequency crystal exclusively. The packets carrying the timestamp information were in concordance with the standard BLE packet layout. With a significant amount of prototyping being done via state machine analysis, component diagrams, etc the main code has been designed using Embedded *C* programming.

The entire Software Development process was carried out using Agile methodologies involving rapid testing as well as prototyping through the different stages of the development. Moreover, SEGGER Embedded Studio IDE was used as the core platform for development along with continuous exposure to the Software Testing Life Cycle via applying test automation procedures on the already developed Nordic Semiconductor software applications in Keil5. Besides this the SDK versions 12 and 12.2 were used. Additionally, a series of code files containing the standard libraries and essential drivers for software development as well as other defined core functionalities were there that were required to be maintained as well as periodically upgraded to ensure that the software was being built as per the desired software specification.

Once the main code was properly built and loaded in SEGGER, there was the Logic Analyzer tool which served to be extremely efficient in interpreting the time durations (delays) between the varied Radio events (being READY, ADDRESS, PAYLOAD and END) so as to analyze the packet behavior. Instead of just one or two samples, a number of data point samples have been attained. To achieve high degree of accuracy in the results, it was extremely crucial to utilize the temperature sensor functionality of the connected nRF51 board(s) as defined in the nRF51 Reference Manual. Taking different temperature readings, the time measurements had been done to correlate the degree of coherence attained in the data samples.

Not only this, but since the results from the Logic analyzer tool were a source of verification for the incurred time events, it was also significant to capture the timestamp values in real time. For this the SEGGER J-Link RTT Viewer had been used for debugging the error-prone *C* code in real time and the measurements were recorded. Using the RTT viewer was incumbent on the activation of the Logger module in the sdk_config.h file as well as enabling of the logging functionality in the main code too. This turned out to be a quite innovative piece of work for this thesis since seeing how advanced systems behave in real time proved to be the "soul" for taking a valid stand from what was truly expected out from research in this area.

Adding on, taking into the account the practical considerations in mind, Tiny-Sync algorithm has been known to rely on the linearity relationship between the clock

drifts upon the failure of which the algorithm loses its validity. The research done in this regard as well as the practical analysis of the results clearly indicate that it is not possible for the clocks to drift linearly as intended. This is something logically out of control due to the vast majority of underlying factors for these unpredictable variations like temperature, voltage, and other environmental factors. Although efforts have been made to take the temperature stability into account yet it is quite evident that any such temperature measuring device has some or the other precision error due to which even the recorded temperature may not comply to 100% accuracy criteria.

Moreover, it was even observed that there were some unusual delays within the network during the corresponding packet transmissions and receptions. This was clear from the results obtained on the Analyzer. For instance, taking the time between the READY and ADDRESS events, this was indicative of the time from when Radio had been ramped up to when the actual transmission of the packet from the transmitter node was initiated. It could have been considered that the packet is immediately sent on the READY event itself instead of on the ADDRESS event but this was not practically feasible.

Also, Tiny-Sync imposes a severe limitation on the number of data points to be stored at one time so as to achieve higher precision but the results from the practical analysis done contradict this idea as proposed in the algorithm. In effect, the results clearly indicate that better coherence is attained with a number of data point samples. Since the main idea is of synchronizing time between multiple nodes in a similar manner by timestamping of real time events, having more number of data samples truly emphasizes a significant degree of compliance with the expected results.

Apart from this, the thesis work has enabled me to gain expertise in working with Bluetooth Low Energy technology, tools used in developing Bluetooth Low Energy solutions and developing software programs for the Integrated circuits (ICs) produced by Nordic Semiconductor.

Chapter 8
Futurework

An in depth analysis of the efficiency of the different algorithms proposed for synchronizing time across network nodes in sensor networks, like the Tiny-Sync and Mini-Sync algorithms, urges onto investigating further possibilities to either extend the scope of these algorithms in generating optimally bounded estimates on the relative clock drift and offset or discovering more advanced algorithms with sufficient logical assumptions and reasoning in concordance with the known facts so as to model the system behavior precisely with regard to the relevant parametric variations.

Moreover, considerable efforts may be made in seeing how these algorithms behave with varied existing synchronization schemes and that new methodologies be derived for integrating advanced functionalities so as to ensure more reliable outcomes.

Last but not the least, with the rapid advancements in technology leading to the development of complex systems monitoring control of crucial information transfers from the source to the destination, and even considering the vast future possibilities, effectively synchronizing data and time for reliable and secure communication is inevitable.

References

[1] M. Roche. Time Synchronization in Wireless Networks. [Online]. Available: http://www.cs.wustl.edu/~jain/cse574-06/ftp/time_sync/index.html

[2] M. L. Sichitiu and C. Veerarittiphan, "Simple, accurate time synchronization for wireless sensor networks," in *2003 IEEE Wireless Communications and Networking, 2003. WCNC 2003.*, vol. 2, March 2003, pp. 1266–1273 vol.2. [Online]. Available: http://ieeexplore.ieee.org/document/1200555/

[3] S. Yoon, C. Veerarittiphan, and M. L. Sichitiu, "Tiny-sync: Tight time synchronization for wireless sensor networks," *ACM Trans. Sen. Netw.*, vol. 3, no. 2, Jun. 2007. [Online]. Available: http://www4.ncsu.edu/~mlsichit/Research/Publications/tsACMToWSN.pdf

[4] David Devasahayam Edwin. Bluetooth LE Packet Layouts. [Online]. Available: https://freedcamp.com/Erlends_Projects_tmc/IT2901_Mesh_IzsT/files/versions/2031747y

[5] Nordic Semiconductor, *nRF51 Series Reference Manual*. [Online]. Available: http://infocenter.nordicsemi.com/pdf/nRF51_RM_v3.0.1.pdf

[6] M. Roche. Time Synchronization in Wireless Networks. [Online]. Available: https://www.cse.wustl.edu/~jain/cse574-06/ftp/time_sync/index.html

[7] E. run Technologies. Network time synchronization. [Online]. Available: https://www.endruntechnologies.com/network-time-synchronization.htm

[8] I. Cisco Systems. Network Time Protocol: Best Practices White Paper.

[9] F. Sivrikaya and B. Yener, "Time synchronization in sensor networks: a survey," *IEEE Network*, vol. 18, no. 4, pp. 45–50, July 2004. [Online]. Available: http://ieeexplore.ieee.org/document/1316761/

[10] M. Henderson and T. Shaver, "Sampling synchronization with gigabit ethernet," in *OCEANS 2009*, Oct 2009, pp. 1–7. [Online]. Available: http://ieeexplore.ieee.org/document/5422060

[11] J. Elson and D. Estrin, "Time synchronization for wireless sensor networks," in *Proceedings 15th International Parallel and Distributed Processing Symposium. IPDPS 2001*, April 2001, pp. 1965–1970. [Online]. Available: http://ieeexplore.ieee.org/document/925191/

[12] M. D. Lemmon, J. Ganguly, and L. Xia, "Model-based clock synchronization in networks with drifting clocks," in *Proceedings. 2000 Pacific Rim International Symposium on Dependable Computing*, 2000, pp. 177–184. [Online]. Available: http://ieeexplore.ieee.org/document/897300/

[13] S. Ganeriwal, R. Kumar, and M. B. Srivastava, "Timing-sync protocol for sensor networks," in *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, ser. SenSys '03. New York, NY, USA: ACM, 2003, pp. 138–149. [Online]. Available: https://www.academia.edu/4297325/Timing-sync_protocol_for_sensor_networks

[14] techopedia. Asynchronous Transfer Mode. [Online]. Available: https://www.techopedia.com/definition/5339/asynchronous-transfer-mode-atm

[15] Ian Poole. PLL Phase Locked Loop Tutorial. [Online]. Available: http://www.radio-electronics.com/info/rf-technology-design/pll-synthesizers/phase-locked-loop-tutorial.php

[16] Nordic Semiconductor. nRF51 DK. [Online]. Available: http://www.nordicsemi.com/eng/Products/nRF51-DK

[17] ——. nRF51 Development Kit, User Guide. [Online]. Available: http://infocenter.nordicsemi.com/pdf/nRF51_Development_Kit_User_Guide_v1.2.pdf

[18] ARM, "μVision IDE." [Online]. Available: http://www2.keil.com/mdk5/uvision/

[19] Segger, "Embedded Studio." [Online]. Available: https://www.segger.com/embedded-studio.html

[20] Nordic Semiconductor. Development Tools and Software. [Online]. Available: https://www.nordicsemi.com/eng/Products/Bluetooth-low-energy/nRF51822/Development-Tools-and-Software2/(language)/eng-GB#hardware

[21] Wireshark Foundation, "Wireshark." [Online]. Available: https://www.wireshark.org/

[22] B. Deraas, E. H. Langseth, J. V. Bjørgan, N. I. Kvam, S. O. Wie, "It2901 informatics project ii for nordic semiconductor," 2017. [Online]. Available: https://www.overleaf.com/7863907qjdyggwvxzjd#/27662380/

[23] Nordic Semiconductor. nRF52 DK. [Online]. Available: https://www.nordicsemi.com/eng/Products/Bluetooth-low-energy/nRF52-DK

[24] ——, "nRF5x-Command-Line-Tools-Win32." [Online]. Available: https://www.nordicsemi.com/eng/nordic/Products/nRF51822/nRF5xCommand-Line-Tools-Win32/33444

[25] ARM, "GNU ARM Embedded Toolchain." [Online]. Available: https://developer.arm.com/open-source/gnu-toolchain/gnu-rm

[26] GNU, "The GNU MCU Eclipse Windows Build Tools." [Online]. Available: http://gnuarmeclipse.github.io/windows-build-tools/

[27] Segger, "J-Link / J-Trace Downloads." [Online]. Available: https://www.segger.com/downloads/jlink

[28] ——, "J-Link RTT Viewer." [Online]. Available: https://www.segger.com/products/debug-probes/j-link/technology/real-time-transfer/rtt-viewer/

[29] CompuPhase, "Termite: a simple RS232 terminal." [Online]. Available: https://www.compuphase.com/software_termite.htm

[30] Nordic Semiconductor, "nRF Toolbox App." [Online]. Available: https://www.nordicsemi.com/eng/Products/Nordic-mobile-Apps/nRF-Toolbox-App

[31] Erlend Langseth, "Segger Embedded Studio, Adapted for nRF51422." [Online]. Available: http://folk.ntnu.no/erlendhl/Segger%20Embedded%20Studio.html

[32] Git, "Git." [Online]. Available: https://git-scm.com/

[33] GitHub, "GitHub." [Online]. Available: https://github.com/

[34] Saleae, Inc, "Saleae Logic Analyzer." [Online]. Available: https://www.saleae.com/

[35] Python Software Foundation, "Python." [Online]. Available: https://www.python.org/

[36] Nordic Semiconductor. nRF5x pynrfjprog. [Online]. Available: http://infocenter.nordicsemi.com/pdf/nRF5x_pynrfjprog_v1.0.pdf

[37] ——, *infocenter pynrfjprog*. [Online]. Available: http://infocenter.nordicsemi.com/index.jsp?topic=%2Fcom.nordic.infocenter.tools%2Fdita%2Ftools%2Fpynrfjprog%2Fpynrfjprog_reference.html

[38] ——, "Nordic Developer Zone." [Online]. Available: https://devzone.nordicsemi.com/questions/

[39] Bluetooth SIG, "Bluetooth Core Specification v 4.2." [Online]. Available: https://www.bluetooth.com/log-in?btorgReturnURL=%2fdocman%2fhandlers%2fdownloaddoc.ashx%3fdoc%2520id%3d286439%26%2520ga%3d1.50101350.1102963934.1486640504

[40] Nordic Semiconductor, *nRF51422 Product Specification v3.2*. [Online]. Available: http://infocenter.nordicsemi.com/pdf/nRF51422_PS_v3.2.pdf

[41] ——, "nRF51422 Product Anomaly Notice v3.1." [Online]. Available: http://infocenter.nordicsemi.com/pdf/nRF51422-PAN_v3.1.pdf

[42] ——, "nRF51422 Product Change Notification." [Online]. Available: http://infocenter.nordicsemi.com/pdf/pcn_093_v1.2.pdf

[43] Bluetooth SIG. Core Specifications. [Online]. Available: https://www.bluetooth.com/specifications/bluetooth-core-specification

[44] ——. Generic Access Profile | Bluetooth Technology Website. [Online]. Available: https://www.bluetooth.com/specifications/assigned-numbers/generic-access-profile

[45] Nordic Semiconductor. Nordic Semiconductor Infocenter. [Online]. Available: https://infocenter.nordicsemi.com/index.jsp

[46] Software Testing Fundamentals, "Software Testing Life Cycle." [Online]. Available: http://softwaretestingfundamentals.com/software-testing-life-cycle/

[47] Nordic Semiconductor, "nRF Connect Mesh Beta Release - Making BLE Mesh development easy." [Online]. Available: https://devzone.nordicsemi.com/blogs/998/nrf-connectmesh-beta-release-making-ble-mesh-deve/

[48] ——. Debugging with Real Time Terminal. [Online]. Available: https://devzone.nordicsemi.com/tutorials/6/

[49] LinkLabs. Bluetooth Vs. Bluetooth Low Energy: What's The Difference? [Online]. Available: https://www.link-labs.com/blog/bluetooth-vs-bluetooth-low-energy

[50] How-To-Geek, C. Hoffman. More Than Headsets: 5 Things You Can Do With Bluetooth. [Online]. Available: https://www.howtogeek.com/165845/more-than-headsets-5-things-you-can-do-with-bluetooth/

[51] Bluetooth SIG. How It Works | Bluetooth Technology Website. [Online]. Available: https://www.bluetooth.com/what-is-bluetooth-technology/how-it-works

[52] Nordic Semiconductor. Bluetooth low energy. [Online]. Available: http://www.nordicsemi.com/eng/Products/Bluetooth-low-energy

[53] ——. Bluetooth 5. [Online]. Available: http://www.nordicsemi.com/eng/Products/Bluetooth-5

[54] Bluetooth SIG. Bluetooth 5 | Bluetooth Technology Website. [Online]. Available: https://www.bluetooth.com/what-is-bluetooth-technology/how-it-works/bluetooth5

[55] techopedia. What is a Crystal Oscillator? [Online]. Available: https://www.techopedia.com/definition/2245/crystal-oscillator

[56] Nordic Semiconductor. nRF51422, Multiprotocol Bluetooth low energy and ANT/ANT+ and 2.4GHz proprietary System-on-Chip. [Online]. Available: https://infocenter.nordicsemi.com/index.jsppdf/nRF51422_PB_v2.5.pdf

[57] ARM. Cortex-M – Arm. [Online]. Available: http://www.arm.com/products/processors/cortex-m

[58] Bluetooth SIG, "Bluetooth Core Specification v 5.0," 06.12.2016.

Appendix A
C-Code

A.1 main.c

```
/* Copyright (c) 2014 Nordic Semiconductor. All Rights Reserved.
 *
 * The information contained herein is property of Nordic Semiconductor
      ASA.
 * Terms and conditions of usage are described in detail in NORDIC
 * SEMICONDUCTOR STANDARD SOFTWARE LICENSE AGREEMENT.
 *
 * Licensees are granted free, non-transferable use of the information.
      NO
 * WARRANTY of ANY KIND is provided. This heading must NOT be removed
      from
 * the file.
 *
 */

/** @file
*
* @defgroup ppi_example_main main.c
* @{
* @ingroup ppi_example
* @brief PPI Example Application main file.
*
* This file contains the source code for a sample application using PPI
      to communicate between timers.
*
*/
#include <stdbool.h>
#include <stdint.h>
#include <stdio.h>
#include "nrf.h"
#include "nrf_delay.h"
#include "app_error.h"
#include "boards.h"
#include "nrf_drv_ppi.h"
#include "nrf_drv_timer.h"
```

```
#include "nordic_common.h"
#include "nrf_drv_gpiote.h"
 #include "SEGGER_RTT.h"

#define NRF_LOG_MODULE_NAME "APP"
#include "nrf_log.h"
#include "nrf_log_ctrl.h"

#include "bsp.h"
#include "app_button.h"

#ifdef BSP_LED_0
    #define GPIO_OUTPUT_PIN_NUMBER BSP_LED_0  /**< Pin number for
        output. */
#endif
#ifndef GPIO_OUTPUT_PIN_NUMBER
    #error "Please indicate output pin"
#endif

int printf(const char *fmt,...);
uint8_t my_array[100];
uint32_t counter = 0;

const nrf_drv_timer_t timer0 = NRF_DRV_TIMER_INSTANCE(0);


static uint32_t *radio_register = (uint32_t *) 0x40001000;
nrf_ppi_channel_t ppi_channel1, ppi_channel2, ppi_channel3,
    ppi_channel4, ppi_channel5, ppi_channel6, ppi_channel7,
    ppi_channel8, ppi_channel9, ppi_channel10, ppi_channel11,
    ppi_channel11, ppi_channel12, ppi_channel13;



uint8_t packet[] = {0b01000000, 15, 0, 0xee, 0x03, 0x21, 0x35, 0x1c, 0
    xee, 0x07, 0x09,0,0,0,0,0,0,0,0,0,0,0,0};
uint32_t recv_time = 0;
uint32_t ack_time = 0;


void
write_uint32(uint8_t *dest, uint32_t src) {
    uint32_t low_mask = (1<<9) - 1;
    dest[0] = (src      ) & low_mask;
    dest[1] = (src>> 8) & low_mask;
    dest[2] = (src>>16) & low_mask;
    dest[3] = (src>>24) & low_mask;
}
uint32_t
read_uint32(uint8_t *bytes) {
    return (bytes[3]<<24) | (bytes[2]<<16) | (bytes[1]<<8) | (bytes[0])
        ;
```

```
}


// Timer event handler. Not used since timer is used only for PPI.
void timer_event_handler(nrf_timer_event_t event_type, void * p_context
    ){}

/** @brief Function for initializing the PPI peripheral.
*/
static void ppi_init(void)
{
    uint32_t err_code = NRF_SUCCESS;

    err_code = nrf_drv_ppi_init();
    APP_ERROR_CHECK(err_code);

    // Configure 1st available PPI channel to stop TIMER0 counter on
        TIMER1 COMPARE[0] match, which is every even number of seconds.
    err_code = nrf_drv_ppi_channel_alloc(&ppi_channel1);
    APP_ERROR_CHECK(err_code);
    err_code = nrf_drv_ppi_channel_assign(ppi_channel1,
                                          nrf_drv_timer_event_address_get
                                              (&timer0,
                                              NRF_TIMER_EVENT_COMPARE0)
                                              ,(uint32_t)&NRF_TIMER0->
                                              TASKS_CAPTURE[1]);

    APP_ERROR_CHECK(err_code);



    // Enable PPI channel
    err_code = nrf_drv_ppi_channel_enable(ppi_channel1);
    APP_ERROR_CHECK(err_code);
   }


/** @brief Function for Timer 0 initialization*/

static void timer0_init(void)
{
    nrf_drv_timer_config_t timer_cfg = NRF_DRV_TIMER_DEFAULT_CONFIG;
     timer_cfg.frequency = NRF_TIMER_FREQ_31250Hz;
     timer_cfg.mode = NRF_TIMER_MODE_COUNTER;
    ret_code_t err_code = nrf_drv_timer_init(&timer0, &timer_cfg,
        timer_event_handler);
    APP_ERROR_CHECK(err_code);
}
```

```
/**
 * @brief Function for application main entry.
 */




uint32_t low_mask(char n) {
  return (1<<(n+1)) - 1;
}

void set_address0(uint32_t address) {
  uint32_t base = address << 8;
  NRF_RADIO->BASE0 = base;
  uint32_t prefix = (address >> 24) & low_mask(8);
  NRF_RADIO->PREFIX0 = prefix;
}




void radio_init(){
    uint32_t err_code = NRF_SUCCESS;

    //Set frequency, advertising channel 37.
    NRF_RADIO->FREQUENCY = 2UL;  // Frequency bin 2, 2402MHz

    //Set whitening to same as channel
    NRF_RADIO->DATAWHITEIV = 37;
    NRF_RADIO->TXADDRESS = 0; //Base0 + prefix0
    NRF_RADIO->RXADDRESSES = 1;

    //Set TXPOWER
    NRF_RADIO->TXPOWER = RADIO_TXPOWER_TXPOWER_0dBm;

     uint32_t override_val = NRF_FICR->OVERRIDEEN &
         FICR_OVERRIDEEN_BLE_1MBIT_Msk;

  if (override_val == FICR_OVERRIDEEN_BLE_1MBIT_Override) {
    NRF_RADIO->OVERRIDE0 = NRF_FICR->BLE_1MBIT[0];
    NRF_RADIO->OVERRIDE1 = NRF_FICR->BLE_1MBIT[1];
    NRF_RADIO->OVERRIDE2 = NRF_FICR->BLE_1MBIT[2];
    NRF_RADIO->OVERRIDE3 = NRF_FICR->BLE_1MBIT[3];
    NRF_RADIO->OVERRIDE4 = NRF_FICR->BLE_1MBIT[4];
  }

    NRF_RADIO->MODE = RADIO_MODE_MODE_Nrf_1Mbit;

    //Set CRC initial value
    NRF_RADIO->CRCINIT = 0x555555;
```

```
    //Set CRC size and address include
    NRF_RADIO->CRCCNF =
            (3 << RADIO_CRCCNF_LEN_Pos)   |
            (true << RADIO_CRCCNF_SKIPADDR_Pos);

    NRF_RADIO->CRCPOLY = (1<<24) | (1<<10) | (1<<9) | (1<<6) | (1<<4) |
         (1<<3) | (1<<1) | 1;

    //Set size of length, s0 and s1
    NRF_RADIO->PCNF0 =
            (1 << RADIO_PCNF0_S0LEN_Pos) |
            (6 << RADIO_PCNF0_LFLEN_Pos) |
            (2 << RADIO_PCNF0_S1LEN_Pos);

    NRF_RADIO->PCNF1 =
            (100 << RADIO_PCNF1_MAXLEN_Pos) |
            (0 << RADIO_PCNF1_STATLEN_Pos)   |
            (3 << RADIO_PCNF1_BALEN_Pos)     |
            (RADIO_PCNF1_ENDIAN_Little << RADIO_PCNF1_ENDIAN_Pos)    |
            (true << RADIO_PCNF1_WHITEEN_Pos);

    //Set Radio address
    set_address0(0x8E89BED6);


    NRF_RADIO->SHORTS =
            (1 << RADIO_SHORTS_END_DISABLE_Pos);

}

     uint32_t timestamp()
    {
      return NRF_TIMER0->CC[0];
    }

void timer_init(){
    uint32_t err_code;
    //Configure timer and compare for timer0
    nrf_drv_timer_config_t timer_cfg = NRF_DRV_TIMER_DEFAULT_CONFIG;
    timer_cfg.bit_width = NRF_TIMER_BIT_WIDTH_24;
    timer_cfg.frequency = NRF_TIMER_FREQ_1MHz;

    err_code = nrf_drv_timer_init(&timer0, &timer_cfg,
        timer_event_handler);
    APP_ERROR_CHECK(err_code);
    nrf_drv_timer_extended_compare(&timer0, NRF_TIMER_CC_CHANNEL0, 5,
        NRF_TIMER_SHORT_COMPARE0_CLEAR_MASK, false);


    //Configure Timer SHORTS
    NRF_TIMER0->SHORTS = (TIMER_SHORTS_COMPARE0_STOP_Enabled <<
        TIMER_SHORTS_COMPARE0_STOP_Pos);
```

```
    nrf_drv_timer_enable(&timer0);

}

void ppi_init_new(){


    ret_code_t err_code = nrf_drv_gpiote_init();
    APP_ERROR_CHECK(err_code);

    err_code = nrf_drv_ppi_init();
    APP_ERROR_CHECK(err_code);




    //Config event task
    nrf_drv_gpiote_out_config_t task_config4 =
        GPIOTE_CONFIG_OUT_TASK_TOGGLE(true);
    ret_code_t err_code2 = nrf_drv_gpiote_out_init(24, &task_config4);
    APP_ERROR_CHECK(err_code);

    nrf_drv_gpiote_out_config_t task_config =
        GPIOTE_CONFIG_OUT_TASK_TOGGLE(true);
    err_code2 = nrf_drv_gpiote_out_init(23, &task_config);
    APP_ERROR_CHECK(err_code);

    nrf_drv_gpiote_out_config_t task_config2 =
        GPIOTE_CONFIG_OUT_TASK_TOGGLE(true);
    err_code2 = nrf_drv_gpiote_out_init(22, &task_config2);
    APP_ERROR_CHECK(err_code);

    nrf_drv_gpiote_out_config_t task_config3 =
        GPIOTE_CONFIG_OUT_TASK_TOGGLE(true);
    err_code2 = nrf_drv_gpiote_out_init(21, &task_config3);
    APP_ERROR_CHECK(err_code);


    // Configure 1st available PPI channel to stop TIMER0 counter on
        TIMER1 COMPARE[0] match, which is every even number of seconds.

    //PPI Channels Used for debugging
    err_code = nrf_drv_ppi_channel_alloc(&ppi_channel1);
    APP_ERROR_CHECK(err_code);
    err_code = nrf_drv_ppi_channel_assign(ppi_channel1,
                                          (uint32_t) (&NRF_RADIO->
                                              EVENTS_READY),
```

```
                                        nrf_drv_gpiote_out_task_addr_get
                                            (21));
    APP_ERROR_CHECK(err_code);

    err_code = nrf_drv_ppi_channel_alloc(&ppi_channel6);
    APP_ERROR_CHECK(err_code);
    err_code = nrf_drv_ppi_channel_assign(ppi_channel6,
                                        (uint32_t) (&NRF_RADIO->
                                            EVENTS_ADDRESS),
                                        nrf_drv_gpiote_out_task_addr_get
                                            (22));
    APP_ERROR_CHECK(err_code);

    err_code = nrf_drv_ppi_channel_alloc(&ppi_channel4);
    APP_ERROR_CHECK(err_code);
    err_code = nrf_drv_ppi_channel_assign(ppi_channel4,
                                        (uint32_t) (&NRF_RADIO->
                                            EVENTS_PAYLOAD),
                                        nrf_drv_gpiote_out_task_addr_get
                                            (23));
    APP_ERROR_CHECK(err_code);

    err_code = nrf_drv_ppi_channel_alloc(&ppi_channel3);
    APP_ERROR_CHECK(err_code);
    err_code = nrf_drv_ppi_channel_assign(ppi_channel3,
                                        (uint32_t) (&NRF_RADIO->
                                            EVENTS_END),
                                        nrf_drv_gpiote_out_task_addr_get
                                            (24));
    APP_ERROR_CHECK(err_code);

    //Enable ppi
    err_code = nrf_drv_ppi_channel_enable(ppi_channel1);
    err_code = nrf_drv_ppi_channel_enable(ppi_channel6);
    err_code = nrf_drv_ppi_channel_enable(ppi_channel4);
    err_code = nrf_drv_ppi_channel_enable(ppi_channel3);
    APP_ERROR_CHECK(err_code);


    //nrf_drv_gpiote_out_task_enable(23);
    nrf_drv_gpiote_out_task_enable(21);
    nrf_drv_gpiote_out_task_enable(22);
    nrf_drv_gpiote_out_task_enable(23);
    nrf_drv_gpiote_out_task_enable(24);
}


void start_crystal_clock(){
    // Start 16 MHz crystal oscillator
    NRF_CLOCK->EVENTS_HFCLKSTARTED = 0;
     * (int *) 0x40000000 = 1;
```

```
    // Wait for the external oscillator to start up
    while (NRF_CLOCK->EVENTS_HFCLKSTARTED == 0)
    {
        // Do nothing.
    }
     /* Start low frequency crystal oscillator for app_timer(used by
         bsp)*/
    NRF_CLOCK->LFCLKSRC            = (CLOCK_LFCLKSRC_SRC_Xtal <<
        CLOCK_LFCLKSRC_SRC_Pos);
    NRF_CLOCK->EVENTS_LFCLKSTARTED = 0;
    NRF_CLOCK->TASKS_LFCLKSTART    = 1;

    while (NRF_CLOCK->EVENTS_LFCLKSTARTED == 0)
    {
        // Do nothing.
    }

}



void send_packet(char* send_packet){
    //NRF_RADIO->PACKETPTR = (uint32_t) adv_packet;

    //Set packet pointer
    NRF_RADIO->PACKETPTR = (uint32_t) send_packet;

    NRF_RADIO->EVENTS_DISABLED = 0U;
    NRF_RADIO->EVENTS_READY = 0U;
    NRF_RADIO->TASKS_TXEN   = 1U;


    while (NRF_RADIO->EVENTS_DISABLED == 0U)
    {
        // wait
    }

}


void start_receiver(char* receive_packet){
    //Set packet pointer
    NRF_RADIO->PACKETPTR = (uint32_t) receive_packet;

    NRF_RADIO->EVENTS_READY = 0U;
    // Enable radio and wait for ready
    NRF_RADIO->TASKS_RXEN = 1U;
    while (NRF_RADIO->EVENTS_READY == 0U)
    {
        // wait
    }
```

```
    NRF_RADIO->EVENTS_END = 0U;
    // Start listening and wait for address received event
    NRF_RADIO->TASKS_START = 1U;


    // Wait for end of packet or buttons state changed
    while (NRF_RADIO->EVENTS_END == 0U)
    {
        // wait
    }
    NRF_TIMER0->TASKS_CAPTURE[1]=1;
    recv_time = NRF_TIMER0->CC[1];

    ack_time = read_uint32(&receive_packet[18]);
}

void send_scan_res(char* scan_res_packet){


    NRF_RADIO->EVENTS_DISABLED = 0U;
    NRF_RADIO->EVENTS_READY = 0U;
    NRF_RADIO->TASKS_TXEN   = 1U;
    while (NRF_RADIO->EVENTS_READY == 0U)
    {
        // wait
    }

    //NRF_RADIO->TASKS_START = 1U;

    while (NRF_RADIO->EVENTS_DISABLED == 0U)
    {
        // wait
    }
}


#define BUTTON_DETECTION_DELAY          10

static volatile uint16_t button_tick = 0;
static volatile uint16_t button_min_hand = 0;
static volatile bool button_pressed = false;

static void button_handler(uint8_t button, uint8_t action)
{

    switch(action) {
    case APP_BUTTON_PUSH:
        button_pressed = true;
        break;
    case APP_BUTTON_RELEASE:
        button_pressed = false;
```

```
        //SEGGER_RTT_printf(0, "button_ticks are: %d, button_min_hand
            is %d\n", button_tick, button_min_hand);
        button_tick = 0;
        button_min_hand = 0;
        break;
    }
}


void buttons_init(void)
{
    uint32_t err_code = 0;
    static app_button_cfg_t buttons[] = {
        {1, APP_BUTTON_ACTIVE_LOW, NRF_GPIO_PIN_PULLUP, button_handler}
    };
    err_code = app_button_init((app_button_cfg_t *)buttons,
                               sizeof(buttons) / sizeof(buttons[0]),
                               BUTTON_DETECTION_DELAY);
    APP_ERROR_CHECK(err_code);
    err_code = app_button_enable();
    APP_ERROR_CHECK(err_code);

}



void button_tick_increment(void)
{
    if (button_pressed == true) {
        if (button_tick >= 65534) {
            button_min_hand++;
            button_tick = 0;
        }
        button_tick++;
    }
}

void RADIO_IRQHandler(void) {
    NRF_RADIO->EVENTS_READY = 0;

    uint8_t *packet = (uint8_t *) NRF_RADIO->PACKETPTR;

    NRF_TIMER0->TASKS_CAPTURE[0]=1;
    uint32_t transmit_time = NRF_TIMER0->CC[0];
    NRF_GPIO->OUT ^= (1 << 21);
    write_uint32(&packet[10], ack_time);
    write_uint32(&packet[14], recv_time);
    write_uint32(&packet[18], transmit_time);
    NRF_RADIO->TASKS_START = 1;
}

int main(void)
```

```
{
    NVIC_ClearPendingIRQ(RADIO_IRQn);
    NVIC_EnableIRQ(RADIO_IRQn);
    APP_ERROR_CHECK(NRF_LOG_INIT(timestamp));
    NRF_LOG_INFO("NRF_TIMER0->CC[0] = %d", NRF_TIMER0->CC[0]);

/*
                            //0b00000010       //Device address: 0xee, 0
                                x1c, 0x35, 0x21, 0x03, 0xf1, sent LSByte
                                 first
    uint8_t adv_packet[100] = {0b01000000, 15, 0, 0xee, 0x03, 0x21, 0
        x35, 0x1c, 0xee, 8, 0x09, 'N','i','c','o','l','a','i'};
    uint8_t recieve_packet[100] = {0};
    uint8_t scan_res_packet[100] = {0b01000100, 16, 0, 0xf1, 0x03, 0x21
        , 0x35, 0x1c, 0xee, 9, 0x09, 'R','e','s','p','o','n','s', 'e'};
*/

    //power_manage();


    int  button_tick = 0;
    uint32_t err_code;
     //app_button_is_pushed(0);
    ppi_init_new();
    radio_init();
    start_crystal_clock();
    timer_init();
    NRF_RADIO->INTENSET = RADIO_INTENSET_READY_Msk;
    NRF_RADIO->INTENSET = RADIO_INTENSET_ADDRESS_Msk;
    NRF_RADIO->INTENSET = RADIO_INTENSET_PAYLOAD_Msk;
    NRF_RADIO->INTENSET = RADIO_INTENSET_END_Msk;

    NVIC_ClearPendingIRQ(RADIO_IRQn);
    NVIC_EnableIRQ(RADIO_IRQn);
    APP_ERROR_CHECK(NRF_LOG_INIT(timestamp));
    NRF_LOG_INFO("NRF_TIMER0->CC[0]+CC[1]+CC[2] = %d",
                 NRF_TIMER0->CC[0]
                 +NRF_TIMER0->CC[1]
                 +NRF_TIMER0->CC[2]);

    NRF_GPIO->DIRSET = (1 << 21) | (1 << 22) | (1 << 23) | (1 << 24);
    NRF_GPIO->OUTCLR = (1 << 21) | (1 << 22) | (1 << 23) | (1 << 24);
    int tick = 0;

    uint8_t v_nr_1 = 0;
    uint8_t v_nr_2 = 0;

    while(1){
        // Transmit
        NRF_GPIO->OUTSET = (1 << 24);
```

```
        send_packet(packet);

        NRF_GPIO->OUTCLR = (1 << 24);

        // Receive
        start_receiver(my_array);
        // TODO look in my_array, extract the two timestamps, print
            them out

    }


}
```

A.2 sdk_config.h

```
#ifndef SDK_CONFIG_H
#define SDK_CONFIG_H
// <<< Use Configuration Wizard in Context Menu >>>\n
#ifdef USE_APP_CONFIG
#include "app_config.h"
#endif
// <h> nRF_Drivers


// <e> GPIOTE_ENABLED - nrf_drv_gpiote - GPIOTE peripheral driver
//==========================================================
#ifndef GPIOTE_ENABLED
#define GPIOTE_ENABLED 1
#endif
#if  GPIOTE_ENABLED
// <o> GPIOTE_CONFIG_NUM_OF_LOW_POWER_EVENTS - Number of lower power
    input pins
#ifndef GPIOTE_CONFIG_NUM_OF_LOW_POWER_EVENTS
#define GPIOTE_CONFIG_NUM_OF_LOW_POWER_EVENTS 1
#endif

// <o> GPIOTE_CONFIG_IRQ_PRIORITY  - Interrupt priority


// <i> Priorities 0,2 (nRF51) and 0,1,4,5 (nRF52) are reserved for
    SoftDevice
// <0=> 0 (highest)
// <1=> 1
// <2=> 2
// <3=> 3

#ifndef GPIOTE_CONFIG_IRQ_PRIORITY
#define GPIOTE_CONFIG_IRQ_PRIORITY 3
#endif
```

```
// <e> GPIOTE_CONFIG_LOG_ENABLED - Enables logging in the module.
//==========================================================
#ifndef GPIOTE_CONFIG_LOG_ENABLED
#define GPIOTE_CONFIG_LOG_ENABLED 0
#endif
#if  GPIOTE_CONFIG_LOG_ENABLED
// <o> GPIOTE_CONFIG_LOG_LEVEL  - Default Severity level

// <0=> Off
// <1=> Error
// <2=> Warning
// <3=> Info
// <4=> Debug

#ifndef GPIOTE_CONFIG_LOG_LEVEL
#define GPIOTE_CONFIG_LOG_LEVEL 3
#endif

// <o> GPIOTE_CONFIG_INFO_COLOR  - ANSI escape code prefix.

// <0=> Default
// <1=> Black
// <2=> Red
// <3=> Green
// <4=> Yellow
// <5=> Blue
// <6=> Magenta
// <7=> Cyan
// <8=> White

#ifndef GPIOTE_CONFIG_INFO_COLOR
#define GPIOTE_CONFIG_INFO_COLOR 0
#endif

// <o> GPIOTE_CONFIG_DEBUG_COLOR  - ANSI escape code prefix.

// <0=> Default
// <1=> Black
// <2=> Red
// <3=> Green
// <4=> Yellow
// <5=> Blue
// <6=> Magenta
// <7=> Cyan
// <8=> White

#ifndef GPIOTE_CONFIG_DEBUG_COLOR
#define GPIOTE_CONFIG_DEBUG_COLOR 0
#endif

#endif //GPIOTE_CONFIG_LOG_ENABLED
```

```
// </e>

#endif //GPIOTE_ENABLED
// </e>









//==========================================================
// <e> PERIPHERAL_RESOURCE_SHARING_ENABLED - nrf_drv_common -
    Peripheral drivers common module
//==========================================================
#ifndef PERIPHERAL_RESOURCE_SHARING_ENABLED
#define PERIPHERAL_RESOURCE_SHARING_ENABLED 0
#endif
#if  PERIPHERAL_RESOURCE_SHARING_ENABLED
// <e> COMMON_CONFIG_LOG_ENABLED - Enables logging in the module.
//==========================================================
#ifndef COMMON_CONFIG_LOG_ENABLED
#define COMMON_CONFIG_LOG_ENABLED 0
#endif
#if  COMMON_CONFIG_LOG_ENABLED
// <o> COMMON_CONFIG_LOG_LEVEL  - Default Severity level

// <0=> Off
// <1=> Error
// <2=> Warning
// <3=> Info
// <4=> Debug

#ifndef COMMON_CONFIG_LOG_LEVEL
#define COMMON_CONFIG_LOG_LEVEL 3
#endif

// <o> COMMON_CONFIG_INFO_COLOR  - ANSI escape code prefix.

// <0=> Default
// <1=> Black
// <2=> Red
// <3=> Green
// <4=> Yellow
// <5=> Blue
// <6=> Magenta
// <7=> Cyan
// <8=> White

#ifndef COMMON_CONFIG_INFO_COLOR
#define COMMON_CONFIG_INFO_COLOR 0
```

```
#endif

// <o> COMMON_CONFIG_DEBUG_COLOR  - ANSI escape code prefix.

// <0=> Default
// <1=> Black
// <2=> Red
// <3=> Green
// <4=> Yellow
// <5=> Blue
// <6=> Magenta
// <7=> Cyan
// <8=> White

#ifndef COMMON_CONFIG_DEBUG_COLOR
#define COMMON_CONFIG_DEBUG_COLOR 0
#endif

#endif //COMMON_CONFIG_LOG_ENABLED
// </e>

#endif //PERIPHERAL_RESOURCE_SHARING_ENABLED
// </e>

// <e> PPI_ENABLED - nrf_drv_ppi - PPI peripheral driver
//==========================================================
#ifndef PPI_ENABLED
#define PPI_ENABLED 1
#endif
#if  PPI_ENABLED
// <e> PPI_CONFIG_LOG_ENABLED - Enables logging in the module.
//==========================================================
#ifndef PPI_CONFIG_LOG_ENABLED
#define PPI_CONFIG_LOG_ENABLED 0
#endif
#if  PPI_CONFIG_LOG_ENABLED
// <o> PPI_CONFIG_LOG_LEVEL  - Default Severity level

// <0=> Off
// <1=> Error
// <2=> Warning
// <3=> Info
// <4=> Debug

#ifndef PPI_CONFIG_LOG_LEVEL
#define PPI_CONFIG_LOG_LEVEL 3
#endif

// <o> PPI_CONFIG_INFO_COLOR  - ANSI escape code prefix.

// <0=> Default
// <1=> Black
```

```
// <2=> Red
// <3=> Green
// <4=> Yellow
// <5=> Blue
// <6=> Magenta
// <7=> Cyan
// <8=> White

#ifndef PPI_CONFIG_INFO_COLOR
#define PPI_CONFIG_INFO_COLOR 0
#endif

// <o> PPI_CONFIG_DEBUG_COLOR  - ANSI escape code prefix.

// <0=> Default
// <1=> Black
// <2=> Red
// <3=> Green
// <4=> Yellow
// <5=> Blue
// <6=> Magenta
// <7=> Cyan
// <8=> White

#ifndef PPI_CONFIG_DEBUG_COLOR
#define PPI_CONFIG_DEBUG_COLOR 0
#endif

#endif //PPI_CONFIG_LOG_ENABLED
// </e>

#endif //PPI_ENABLED
// </e>

// <e> TIMER_ENABLED - nrf_drv_timer - TIMER peripheral driver
//==========================================================
#ifndef TIMER_ENABLED
#define TIMER_ENABLED 1
#endif
#if  TIMER_ENABLED
// <o> TIMER_DEFAULT_CONFIG_FREQUENCY  - Timer frequency if in Timer
    mode

// <0=> 16 MHz
// <1=> 8 MHz
// <2=> 4 MHz
// <3=> 2 MHz
// <4=> 1 MHz
// <5=> 500 kHz
// <6=> 250 kHz
// <7=> 125 kHz
// <8=> 62.5 kHz
```

```
// <9=> 31.25 kHz

#ifndef TIMER_DEFAULT_CONFIG_FREQUENCY
#define TIMER_DEFAULT_CONFIG_FREQUENCY 0
#endif

// <o> TIMER_DEFAULT_CONFIG_MODE - Timer mode or operation

// <0=> Timer
// <1=> Counter

#ifndef TIMER_DEFAULT_CONFIG_MODE
#define TIMER_DEFAULT_CONFIG_MODE 0
#endif

// <o> TIMER_DEFAULT_CONFIG_BIT_WIDTH - Timer counter bit width

// <0=> 16 bit
// <1=> 8 bit
// <2=> 24 bit
// <3=> 32 bit

#ifndef TIMER_DEFAULT_CONFIG_BIT_WIDTH
#define TIMER_DEFAULT_CONFIG_BIT_WIDTH 0
#endif

// <o> TIMER_DEFAULT_CONFIG_IRQ_PRIORITY - Interrupt priority


// <i> Priorities 0,2 (nRF51) and 0,1,4,5 (nRF52) are reserved for
    SoftDevice
// <0=> 0 (highest)
// <1=> 1
// <2=> 2
// <3=> 3

#ifndef TIMER_DEFAULT_CONFIG_IRQ_PRIORITY
#define TIMER_DEFAULT_CONFIG_IRQ_PRIORITY 3
#endif

// <q> TIMER0_ENABLED - Enable TIMER0 instance


#ifndef TIMER0_ENABLED
#define TIMER0_ENABLED 1
#endif

// <q> TIMER1_ENABLED - Enable TIMER1 instance


#ifndef TIMER1_ENABLED
#define TIMER1_ENABLED 1
```

```
#endif

// <q> TIMER2_ENABLED  - Enable TIMER2 instance


#ifndef TIMER2_ENABLED
#define TIMER2_ENABLED 1
#endif

// <q> TIMER3_ENABLED  - Enable TIMER3 instance


#ifndef TIMER3_ENABLED
#define TIMER3_ENABLED 0
#endif

// <q> TIMER4_ENABLED  - Enable TIMER4 instance


#ifndef TIMER4_ENABLED
#define TIMER4_ENABLED 0
#endif

// <e> TIMER_CONFIG_LOG_ENABLED - Enables logging in the module.
//==========================================================
#ifndef TIMER_CONFIG_LOG_ENABLED
#define TIMER_CONFIG_LOG_ENABLED 0
#endif
#if  TIMER_CONFIG_LOG_ENABLED
// <o> TIMER_CONFIG_LOG_LEVEL  - Default Severity level

// <0=> Off
// <1=> Error
// <2=> Warning
// <3=> Info
// <4=> Debug

#ifndef TIMER_CONFIG_LOG_LEVEL
#define TIMER_CONFIG_LOG_LEVEL 3
#endif

// <o> TIMER_CONFIG_INFO_COLOR  - ANSI escape code prefix.

// <0=> Default
// <1=> Black
// <2=> Red
// <3=> Green
// <4=> Yellow
// <5=> Blue
// <6=> Magenta
// <7=> Cyan
// <8=> White
```

```
#ifndef TIMER_CONFIG_INFO_COLOR
#define TIMER_CONFIG_INFO_COLOR 0
#endif

// <o> TIMER_CONFIG_DEBUG_COLOR  - ANSI escape code prefix.

// <0=> Default
// <1=> Black
// <2=> Red
// <3=> Green
// <4=> Yellow
// <5=> Blue
// <6=> Magenta
// <7=> Cyan
// <8=> White

#ifndef TIMER_CONFIG_DEBUG_COLOR
#define TIMER_CONFIG_DEBUG_COLOR 0
#endif

#endif //TIMER_CONFIG_LOG_ENABLED
// </e>

#endif //TIMER_ENABLED
// </e>

// <e> UART_ENABLED - nrf_drv_uart - UART/UARTE peripheral driver
//==========================================================
#ifndef UART_ENABLED
#define UART_ENABLED 1
#endif
#if  UART_ENABLED
// <o> UART_DEFAULT_CONFIG_HWFC  - Hardware Flow Control

// <0=> Disabled
// <1=> Enabled

#ifndef UART_DEFAULT_CONFIG_HWFC
#define UART_DEFAULT_CONFIG_HWFC 0
#endif

// <o> UART_DEFAULT_CONFIG_PARITY  - Parity

// <0=> Excluded
// <14=> Included

#ifndef UART_DEFAULT_CONFIG_PARITY
#define UART_DEFAULT_CONFIG_PARITY 0
#endif

// <o> UART_DEFAULT_CONFIG_BAUDRATE  - Default Baudrate
```

```
// <323584=> 1200 baud
// <643072=> 2400 baud
// <1290240=> 4800 baud
// <2576384=> 9600 baud
// <3862528=> 14400 baud
// <5152768=> 19200 baud
// <7716864=> 28800 baud
// <10289152=> 38400 baud
// <15400960=> 57600 baud
// <20615168=> 76800 baud
// <30924800=> 115200 baud
// <61865984=> 230400 baud
// <67108864=> 250000 baud
// <121634816=> 460800 baud
// <251658240=> 921600 baud
// <268435456=> 57600 baud

#ifndef UART_DEFAULT_CONFIG_BAUDRATE
#define UART_DEFAULT_CONFIG_BAUDRATE 30924800
#endif

// <o> UART_DEFAULT_CONFIG_IRQ_PRIORITY - Interrupt priority


// <i> Priorities 0,2 (nRF51) and 0,1,4,5 (nRF52) are reserved for
    SoftDevice
// <0=> 0 (highest)
// <1=> 1
// <2=> 2
// <3=> 3

#ifndef UART_DEFAULT_CONFIG_IRQ_PRIORITY
#define UART_DEFAULT_CONFIG_IRQ_PRIORITY 3
#endif

// <q> UART_EASY_DMA_SUPPORT - Driver supporting EasyDMA


#ifndef UART_EASY_DMA_SUPPORT
#define UART_EASY_DMA_SUPPORT 1
#endif

// <q> UART_LEGACY_SUPPORT - Driver supporting Legacy mode


#ifndef UART_LEGACY_SUPPORT
#define UART_LEGACY_SUPPORT 1
#endif

// <e> UART0_ENABLED - Enable UART0 instance
//==========================================================
```

```
#ifndef UART0_ENABLED
#define UART0_ENABLED 1
#endif
#if  UART0_ENABLED
// <q> UART0_CONFIG_USE_EASY_DMA  - Default setting for using EasyDMA


#ifndef UART0_CONFIG_USE_EASY_DMA
#define UART0_CONFIG_USE_EASY_DMA 1
#endif

#endif //UART0_ENABLED
// </e>

// <e> UART_CONFIG_LOG_ENABLED - Enables logging in the module.
//==========================================================
#ifndef UART_CONFIG_LOG_ENABLED
#define UART_CONFIG_LOG_ENABLED 0
#endif
#if  UART_CONFIG_LOG_ENABLED
// <o> UART_CONFIG_LOG_LEVEL  - Default Severity level

// <0=> Off
// <1=> Error
// <2=> Warning
// <3=> Info
// <4=> Debug

#ifndef UART_CONFIG_LOG_LEVEL
#define UART_CONFIG_LOG_LEVEL 3
#endif

// <o> UART_CONFIG_INFO_COLOR  - ANSI escape code prefix.

// <0=> Default
// <1=> Black
// <2=> Red
// <3=> Green
// <4=> Yellow
// <5=> Blue
// <6=> Magenta
// <7=> Cyan
// <8=> White

#ifndef UART_CONFIG_INFO_COLOR
#define UART_CONFIG_INFO_COLOR 0
#endif

// <o> UART_CONFIG_DEBUG_COLOR  - ANSI escape code prefix.

// <0=> Default
// <1=> Black
```

```
// <2=> Red
// <3=> Green
// <4=> Yellow
// <5=> Blue
// <6=> Magenta
// <7=> Cyan
// <8=> White

#ifndef UART_CONFIG_DEBUG_COLOR
#define UART_CONFIG_DEBUG_COLOR 0
#endif

#endif //UART_CONFIG_LOG_ENABLED
// </e>

#endif //UART_ENABLED
// </e>

// </h>
//==========================================================

// <h> nRF_Log

//==========================================================
// <e> NRF_LOG_ENABLED - nrf_log - Logging
//==========================================================
#ifndef NRF_LOG_ENABLED
#define NRF_LOG_ENABLED 1
#endif
#if  NRF_LOG_ENABLED
// <e> NRF_LOG_USES_COLORS - If enabled then ANSI escape code for
    colors is prefixed to every string
//==========================================================
#ifndef NRF_LOG_USES_COLORS
#define NRF_LOG_USES_COLORS 0
#endif
#if  NRF_LOG_USES_COLORS
// <o> NRF_LOG_COLOR_DEFAULT  - ANSI escape code prefix.

// <0=> Default
// <1=> Black
// <2=> Red
// <3=> Green
// <4=> Yellow
// <5=> Blue
// <6=> Magenta
// <7=> Cyan
// <8=> White

#ifndef NRF_LOG_COLOR_DEFAULT
#define NRF_LOG_COLOR_DEFAULT 0
#endif
```

```
// <o> NRF_LOG_ERROR_COLOR - ANSI escape code prefix.

// <0=> Default
// <1=> Black
// <2=> Red
// <3=> Green
// <4=> Yellow
// <5=> Blue
// <6=> Magenta
// <7=> Cyan
// <8=> White

#ifndef NRF_LOG_ERROR_COLOR
#define NRF_LOG_ERROR_COLOR 0
#endif

// <o> NRF_LOG_WARNING_COLOR - ANSI escape code prefix.

// <0=> Default
// <1=> Black
// <2=> Red
// <3=> Green
// <4=> Yellow
// <5=> Blue
// <6=> Magenta
// <7=> Cyan
// <8=> White

#ifndef NRF_LOG_WARNING_COLOR
#define NRF_LOG_WARNING_COLOR 0
#endif

#endif //NRF_LOG_USES_COLORS
// </e>

// <o> NRF_LOG_DEFAULT_LEVEL - Default Severity level

// <0=> Off
// <1=> Error
// <2=> Warning
// <3=> Info
// <4=> Debug

#ifndef NRF_LOG_DEFAULT_LEVEL
#define NRF_LOG_DEFAULT_LEVEL 3
#endif

// <e> NRF_LOG_DEFERRED - Enable deffered logger.

// <i> Log data is buffered and can be processed in idle.
//==========================================================
```

```
#ifndef NRF_LOG_DEFERRED
#define NRF_LOG_DEFERRED 0
#endif
#if  NRF_LOG_DEFERRED
// <o> NRF_LOG_DEFERRED_BUFSIZE - Size of the buffer for logs in words.
// <i> Must be power of 2

#ifndef NRF_LOG_DEFERRED_BUFSIZE
#define NRF_LOG_DEFERRED_BUFSIZE 256
#endif

#endif //NRF_LOG_DEFERRED
// </e>

// <q> NRF_LOG_USES_TIMESTAMP  - Enable timestamping


// <i> Function for getting the timestamp is provided by the user

#ifndef NRF_LOG_USES_TIMESTAMP
#define NRF_LOG_USES_TIMESTAMP 1
#endif

#endif //NRF_LOG_ENABLED
// </e>

// <h> nrf_log_backend - Logging sink

//==========================================================
// <o> NRF_LOG_BACKEND_MAX_STRING_LENGTH - Buffer for storing single
    output string
// <i> Logger backend RAM usage is determined by this value.

#ifndef NRF_LOG_BACKEND_MAX_STRING_LENGTH
#define NRF_LOG_BACKEND_MAX_STRING_LENGTH 256
#endif

// <o> NRF_LOG_TIMESTAMP_DIGITS - Number of digits for timestamp
// <i> If higher resolution timestamp source is used it might be needed
     to increase that

#ifndef NRF_LOG_TIMESTAMP_DIGITS
#define NRF_LOG_TIMESTAMP_DIGITS 8
#endif

// <e> NRF_LOG_BACKEND_SERIAL_USES_UART - If enabled data is printed
    over UART
//==========================================================
#ifndef NRF_LOG_BACKEND_SERIAL_USES_UART
#define NRF_LOG_BACKEND_SERIAL_USES_UART 1
#endif
#if  NRF_LOG_BACKEND_SERIAL_USES_UART
```

```
// <o> NRF_LOG_BACKEND_SERIAL_UART_BAUDRATE  - Default Baudrate

// <323584=> 1200 baud
// <643072=> 2400 baud
// <1290240=> 4800 baud
// <2576384=> 9600 baud
// <3862528=> 14400 baud
// <5152768=> 19200 baud
// <7716864=> 28800 baud
// <10289152=> 38400 baud
// <15400960=> 57600 baud
// <20615168=> 76800 baud
// <30924800=> 115200 baud
// <61865984=> 230400 baud
// <67108864=> 250000 baud
// <121634816=> 460800 baud
// <251658240=> 921600 baud
// <268435456=> 57600 baud

#ifndef NRF_LOG_BACKEND_SERIAL_UART_BAUDRATE
#define NRF_LOG_BACKEND_SERIAL_UART_BAUDRATE 30924800
#endif

// <o> NRF_LOG_BACKEND_SERIAL_UART_TX_PIN - UART TX pin
#ifndef NRF_LOG_BACKEND_SERIAL_UART_TX_PIN
#define NRF_LOG_BACKEND_SERIAL_UART_TX_PIN 9
#endif

// <o> NRF_LOG_BACKEND_SERIAL_UART_RX_PIN - UART RX pin
#ifndef NRF_LOG_BACKEND_SERIAL_UART_RX_PIN
#define NRF_LOG_BACKEND_SERIAL_UART_RX_PIN 11
#endif

// <o> NRF_LOG_BACKEND_SERIAL_UART_RTS_PIN - UART RTS pin
#ifndef NRF_LOG_BACKEND_SERIAL_UART_RTS_PIN
#define NRF_LOG_BACKEND_SERIAL_UART_RTS_PIN 8
#endif

// <o> NRF_LOG_BACKEND_SERIAL_UART_CTS_PIN - UART CTS pin
#ifndef NRF_LOG_BACKEND_SERIAL_UART_CTS_PIN
#define NRF_LOG_BACKEND_SERIAL_UART_CTS_PIN 10
#endif

// <o> NRF_LOG_BACKEND_SERIAL_UART_FLOW_CONTROL  - Hardware Flow
    Control

// <0=> Disabled
// <1=> Enabled

#ifndef NRF_LOG_BACKEND_SERIAL_UART_FLOW_CONTROL
#define NRF_LOG_BACKEND_SERIAL_UART_FLOW_CONTROL 0
#endif
```

```
// <o> NRF_LOG_BACKEND_UART_INSTANCE  - UART instance used

// <0=> 0

#ifndef NRF_LOG_BACKEND_UART_INSTANCE
#define NRF_LOG_BACKEND_UART_INSTANCE 0
#endif

#endif //NRF_LOG_BACKEND_SERIAL_USES_UART
// </e>

// <e> NRF_LOG_BACKEND_SERIAL_USES_RTT - If enabled data is printed
    using RTT
//==========================================================
#ifndef NRF_LOG_BACKEND_SERIAL_USES_RTT
#define NRF_LOG_BACKEND_SERIAL_USES_RTT 0
#endif
#if  NRF_LOG_BACKEND_SERIAL_USES_RTT
// <o> NRF_LOG_BACKEND_RTT_OUTPUT_BUFFER_SIZE - RTT output buffer size.
// <i> Should be equal or bigger than \ref
    NRF_LOG_BACKEND_MAX_STRING_LENGTH.
// <i> This value is used in Segger RTT configuration to set the buffer
     size
// <i> if it is bigger than default RTT buffer size.

#ifndef NRF_LOG_BACKEND_RTT_OUTPUT_BUFFER_SIZE
#define NRF_LOG_BACKEND_RTT_OUTPUT_BUFFER_SIZE 512
#endif

#endif //NRF_LOG_BACKEND_SERIAL_USES_RTT
// </e>

// </h>
//==========================================================

// </h>
//==========================================================

// <h> nRF_Segger_RTT

//==========================================================
// <h> segger_rtt - SEGGER RTT

//==========================================================
// <o> SEGGER_RTT_CONFIG_BUFFER_SIZE_UP - Size of upstream buffer.
#ifndef SEGGER_RTT_CONFIG_BUFFER_SIZE_UP
#define SEGGER_RTT_CONFIG_BUFFER_SIZE_UP 64
#endif

// <o> SEGGER_RTT_CONFIG_MAX_NUM_UP_BUFFERS - Size of upstream buffer.
#ifndef SEGGER_RTT_CONFIG_MAX_NUM_UP_BUFFERS
```

```
#define SEGGER_RTT_CONFIG_MAX_NUM_UP_BUFFERS 2
#endif

// <o> SEGGER_RTT_CONFIG_BUFFER_SIZE_DOWN - Size of upstream buffer.
#ifndef SEGGER_RTT_CONFIG_BUFFER_SIZE_DOWN
#define SEGGER_RTT_CONFIG_BUFFER_SIZE_DOWN 16
#endif

// <o> SEGGER_RTT_CONFIG_MAX_NUM_DOWN_BUFFERS - Size of upstream buffer
    .
#ifndef SEGGER_RTT_CONFIG_MAX_NUM_DOWN_BUFFERS
#define SEGGER_RTT_CONFIG_MAX_NUM_DOWN_BUFFERS 2
#endif

// </h>
//==========================================================

// </h>
//==========================================================

// <<< end of configuration section >>>
#endif //SDK_CONFIG_H
```

A.3 nrf_drv_timer.c

```
/* Copyright (c) 2015 Nordic Semiconductor. All Rights Reserved.
 *
 * The information contained herein is property of Nordic Semiconductor
    ASA.
 * Terms and conditions of usage are described in detail in NORDIC
 * SEMICONDUCTOR STANDARD SOFTWARE LICENSE AGREEMENT.
 *
 * Licensees are granted free, non-transferable use of the information.
    NO
 * WARRANTY of ANY KIND is provided. This heading must NOT be removed
    from
 * the file.
 *
 */

#include "sdk_common.h"
#if NRF_MODULE_ENABLED(TIMER)
#define ENABLED_TIMER_COUNT (TIMER0_ENABLED)
#if ENABLED_TIMER_COUNT
#include "nrf_drv_timer.h"
#include "nrf_drv_common.h"
#include "app_util_platform.h"

#define NRF_LOG_MODULE_NAME "TIMER"

#if TIMER_CONFIG_LOG_ENABLED
#define NRF_LOG_LEVEL       TIMER_CONFIG_LOG_LEVEL
```

```
#define NRF_LOG_INFO_COLOR  TIMER_CONFIG_INFO_COLOR
#define NRF_LOG_DEBUG_COLOR TIMER_CONFIG_DEBUG_COLOR
#else //TIMER_CONFIG_LOG_ENABLED
#define NRF_LOG_LEVEL       0
#endif //TIMER_CONFIG_LOG_ENABLED
#include "nrf_log.h"
#include "nrf_log_ctrl.h"

/**@brief Timer control block. */
typedef struct
{
    nrf_timer_event_handler_t handler;
    void *                    context;
    nrf_drv_state_t           state;
} timer_control_block_t;

static timer_control_block_t m_cb[ENABLED_TIMER_COUNT];

ret_code_t nrf_drv_timer_init(nrf_drv_timer_t const * const p_instance,
                              nrf_drv_timer_config_t const * p_config,
                              nrf_timer_event_handler_t
                                  timer_event_handler)
{
    timer_control_block_t * p_cb = &m_cb[p_instance->instance_id];
    ASSERT(((p_instance->p_reg == NRF_TIMER0) && TIMER0_ENABLED) || (
        p_instance->p_reg != NRF_TIMER0));


#ifdef SOFTDEVICE_PRESENT
    ASSERT(p_instance->p_reg != NRF_TIMER0);
    ASSERT(p_config);
#endif
    ret_code_t err_code;

    if (p_cb->state != NRF_DRV_STATE_UNINITIALIZED)
    {
        err_code = NRF_ERROR_INVALID_STATE;
        NRF_LOG_WARNING("Function: %s, error code: %s.\r\n", (uint32_t)
            __func__, (uint32_t)ERR_TO_STR(err_code));
        return err_code;
    }

    if (timer_event_handler == NULL)
    {
        err_code = NRF_ERROR_INVALID_PARAM;
        NRF_LOG_WARNING("Function: %s, error code: %s.\r\n", (uint32_t)
            __func__, (uint32_t)ERR_TO_STR(err_code));
        return err_code;
    }
```

```
    /* Warning 685: Relational operator '<=' always evaluates to 'true
        '"
     * Warning in NRF_TIMER_IS_BIT_WIDTH_VALID macro. Macro validate
        timers resolution.
     * Not necessary in nRF52 based systems. Obligatory in nRF51 based
        systems.
     */

    /*lint -save -e685 */

    ASSERT(NRF_TIMER_IS_BIT_WIDTH_VALID(p_instance->p_reg, p_config->
        bit_width));

    //lint -restore

    p_cb->handler = timer_event_handler;
    p_cb->context = p_config->p_context;

    uint8_t i;
    for (i = 0; i < p_instance->cc_channel_count; ++i)
    {
        nrf_timer_event_clear(p_instance->p_reg,
            nrf_timer_compare_event_get(i));
    }


    nrf_timer_mode_set(p_instance->p_reg, p_config->mode);
    nrf_timer_bit_width_set(p_instance->p_reg, p_config->bit_width);
    nrf_timer_frequency_set(p_instance->p_reg, p_config->frequency);

    p_cb->state = NRF_DRV_STATE_INITIALIZED;

    err_code = NRF_SUCCESS;
    NRF_LOG_INFO("Function: %s, error code: %s.\r\n", (uint32_t)
        __func__, (uint32_t)ERR_TO_STR(err_code));
    return err_code;
}

void nrf_drv_timer_uninit(nrf_drv_timer_t const * const p_instance)
{

    #define DISABLE_ALL UINT32_MAX
    nrf_timer_shorts_disable(p_instance->p_reg, DISABLE_ALL);
    nrf_timer_int_disable(p_instance->p_reg, DISABLE_ALL);
    #undef DISABLE_ALL

    if (m_cb[p_instance->instance_id].state == NRF_DRV_STATE_POWERED_ON
        )
    {
        nrf_drv_timer_disable(p_instance);
    }
```

```
    m_cb[p_instance->instance_id].state = NRF_DRV_STATE_UNINITIALIZED;
    NRF_LOG_INFO("Uninitialized instance: %d.\r\n", p_instance->
        instance_id);
}

void nrf_drv_timer_enable(nrf_drv_timer_t const * const p_instance)
{
    ASSERT(m_cb[p_instance->instance_id].state ==
        NRF_DRV_STATE_INITIALIZED);
    nrf_timer_task_trigger(p_instance->p_reg, NRF_TIMER_TASK_START);
    m_cb[p_instance->instance_id].state = NRF_DRV_STATE_POWERED_ON;
    NRF_LOG_INFO("Enabled instance: %d.\r\n", p_instance->instance_id);
}

void nrf_drv_timer_disable(nrf_drv_timer_t const * const p_instance)
{
    ASSERT(m_cb[p_instance->instance_id].state ==
        NRF_DRV_STATE_POWERED_ON);
    nrf_timer_task_trigger(p_instance->p_reg, NRF_TIMER_TASK_SHUTDOWN);
    m_cb[p_instance->instance_id].state = NRF_DRV_STATE_INITIALIZED;
    NRF_LOG_INFO("Disabled instance: %d.\r\n", p_instance->instance_id)
        ;
}

void nrf_drv_timer_resume(nrf_drv_timer_t const * const p_instance)
{
    ASSERT(m_cb[p_instance->instance_id].state ==
        NRF_DRV_STATE_POWERED_ON);
    nrf_timer_task_trigger(p_instance->p_reg, NRF_TIMER_TASK_START);
    NRF_LOG_INFO("Resumed instance: %d.\r\n", p_instance->instance_id);
}

void nrf_drv_timer_pause(nrf_drv_timer_t const * const p_instance)
{
    ASSERT(m_cb[p_instance->instance_id].state ==
        NRF_DRV_STATE_POWERED_ON);
    nrf_timer_task_trigger(p_instance->p_reg, NRF_TIMER_TASK_STOP);
    NRF_LOG_INFO("Paused instance: %d.\r\n", p_instance->instance_id);
}

void nrf_drv_timer_clear(nrf_drv_timer_t const * const p_instance)
{
    ASSERT(m_cb[p_instance->instance_id].state !=
        NRF_DRV_STATE_UNINITIALIZED);
    nrf_timer_task_trigger(p_instance->p_reg, NRF_TIMER_TASK_CLEAR);
}

void nrf_drv_timer_increment(nrf_drv_timer_t const * const p_instance)
{
    ASSERT(m_cb[p_instance->instance_id].state ==
        NRF_DRV_STATE_POWERED_ON);
```

```
    ASSERT(nrf_timer_mode_get(p_instance->p_reg) !=
        NRF_TIMER_MODE_TIMER);

    nrf_timer_task_trigger(p_instance->p_reg, NRF_TIMER_TASK_COUNT);
}

uint32_t nrf_drv_timer_capture(nrf_drv_timer_t const * const p_instance
    ,
                               nrf_timer_cc_channel_t cc_channel)
{
    ASSERT(m_cb[p_instance->instance_id].state ==
        NRF_DRV_STATE_POWERED_ON);
    ASSERT(cc_channel < p_instance->cc_channel_count);

    nrf_timer_task_trigger(p_instance->p_reg,
        nrf_timer_capture_task_get(cc_channel));
    return nrf_timer_cc_read(p_instance->p_reg, cc_channel);
 }


void nrf_drv_timer_compare(nrf_drv_timer_t const * const p_instance,
                           nrf_timer_cc_channel_t cc_channel,
                           uint32_t               cc_value,
                           bool                   enable_int)
{
    nrf_timer_int_mask_t timer_int = nrf_timer_compare_int_get(
        cc_channel);

    if (enable_int)
    {
        nrf_timer_int_enable(p_instance->p_reg, timer_int);
    }
    else
    {
        nrf_timer_int_disable(p_instance->p_reg, timer_int);
    }

    nrf_timer_cc_write(p_instance->p_reg, cc_channel, cc_value);
    NRF_LOG_INFO("Timer id: %d, capture value set: %d, channel: %d.\r\n
        ", p_instance->instance_id, cc_value, cc_channel);
}

void nrf_drv_timer_extended_compare(nrf_drv_timer_t const * const
    p_instance,
                                    nrf_timer_cc_channel_t cc_channel,
                                    uint32_t               cc_value,
                                    nrf_timer_short_mask_t
                                        timer_short_mask,
                                    bool                   enable_int)
{
    nrf_timer_shorts_disable(p_instance->p_reg,
        (TIMER_SHORTS_COMPARE0_STOP_Msk << cc_channel) |
```

```
            (TIMER_SHORTS_COMPARE0_CLEAR_Msk << cc_channel));

        nrf_timer_shorts_enable(p_instance->p_reg, timer_short_mask);

        (void)nrf_drv_timer_compare(p_instance,
                                    cc_channel,
                                    cc_value,
                                    enable_int);
        NRF_LOG_INFO("Timer id: %d, capture value set: %d, channel: %d.\r\n
            ", p_instance->instance_id, cc_value, cc_channel);
}

void nrf_drv_timer_compare_int_enable(nrf_drv_timer_t const * const
    p_instance,
                                      uint32_t channel)
{
    ASSERT(m_cb[p_instance->instance_id].state !=
        NRF_DRV_STATE_UNINITIALIZED);
    ASSERT(channel < p_instance->cc_channel_count);

    nrf_timer_event_clear(p_instance->p_reg,
        nrf_timer_compare_event_get(channel));
    nrf_timer_int_enable(p_instance->p_reg,
        nrf_timer_compare_int_get(channel));
}

void nrf_drv_timer_compare_int_disable(nrf_drv_timer_t const * const
    p_instance,
                                       uint32_t channel)
{
    ASSERT(m_cb[p_instance->instance_id].state !=
        NRF_DRV_STATE_UNINITIALIZED);
    ASSERT(channel < p_instance->cc_channel_count);

    nrf_timer_int_disable(p_instance->p_reg,
        nrf_timer_compare_int_get(channel));
}

#endif // ENABLED_TIMER_COUNT
#endif // NRF_MODULE_ENABLED(TIMER)
```

A.4 nrf_drv_timer.h

```
/* Copyright (c) 2015 Nordic Semiconductor. All Rights Reserved.
 *
 * The information contained herein is property of Nordic Semiconductor
    ASA.
 * Terms and conditions of usage are described in detail in NORDIC
 * SEMICONDUCTOR STANDARD SOFTWARE LICENSE AGREEMENT.
 *
 * Licensees are granted free, non-transferable use of the information.
    NO
```

```
 * WARRANTY of ANY KIND is provided. This heading must NOT be removed
     from
 * the file.
 *
 */

/**@file
 * @addtogroup nrf_timer Timer HAL and driver
 * @ingroup    nrf_drivers
 * @brief      Timer APIs.
 * @details    The timer HAL provides basic APIs for accessing the
     registers
 *             of the timer. The timer driver provides APIs on a higher
      level.
 *
 * @defgroup   nrf_drv_timer Timer driver
 * @{
 * @ingroup    nrf_timer
 * @brief      Multi-instance timer driver.
 */

#ifndef NRF_DRV_TIMER_H__
#define NRF_DRV_TIMER_H__

#include "nordic_common.h"
#include "sdk_config.h"
#include "nrf_timer.h"
#include "sdk_errors.h"
#include "nrf_assert.h"

#ifdef __cplusplus
extern "C" {
#endif

/**
 * @brief Timer driver instance data structure.
 */
typedef struct
{
    NRF_TIMER_Type * p_reg;            ///< Pointer to the structure
        with TIMER peripheral instance registers.
    uint8_t          instance_id;      ///< Driver instance index.
    uint8_t          cc_channel_count; ///< Number of capture/compare
        channels.
} nrf_drv_timer_t;

#define ENABLED_TIMER_COUNT (TIMER0_ENABLED)

#define TIMER0_INSTANCE_INDEX 0

/**
 * @brief Macro for creating a timer driver instance.
```

```
 */
#define NRF_DRV_TIMER_INSTANCE(id) \
{                                                                   \
    .p_reg            = CONCAT_2(NRF_TIMER, id),                    \
    .instance_id      = CONCAT_3(TIMER, id, _INSTANCE_INDEX),       \
    .cc_channel_count = NRF_TIMER_CC_CHANNEL_COUNT(id),             \
}

/**
 * @brief Timer driver instance configuration structure.
 */
typedef struct
{
    nrf_timer_frequency_t frequency;          ///< Frequency.
    nrf_timer_mode_t      mode;               ///< Mode of operation.
    nrf_timer_bit_width_t bit_width;          ///< Bit width.
    uint8_t               interrupt_priority; ///< Interrupt priority.
    void *                p_context;          ///< Context passed to
        interrupt handler.
} nrf_drv_timer_config_t;

/**
 * @brief Timer driver instance default configuration.
 */
#define NRF_DRV_TIMER_DEFAULT_CONFIG \
{

    \
    .frequency          = (nrf_timer_frequency_t)
        TIMER_DEFAULT_CONFIG_FREQUENCY,\
    .mode               = (nrf_timer_mode_t)TIMER_DEFAULT_CONFIG_MODE,
                \
    .bit_width          = (nrf_timer_bit_width_t)
        TIMER_DEFAULT_CONFIG_BIT_WIDTH,\
    .interrupt_priority = TIMER_DEFAULT_CONFIG_IRQ_PRIORITY,
                           \
    .p_context          = NULL
                                                                    \
}

/**
 * @brief Timer driver event handler type.
 *
 * @param[in] event_type Timer event.
 * @param[in] p_context  General purpose parameter set during
     initialization of
 *                       the timer. This parameter can be used to pass
 *                       additional information to the handler function
     , for
 *                       example, the timer ID.
 */
```

```
typedef void (* nrf_timer_event_handler_t)(nrf_timer_event_t event_type
    ,
                                           void * p_context);

/**
 * @brief Function for initializing the timer.
 *
 * @param[in] p_instance          Pointer to the driver instance
     structure.
 * @param[in] p_config            Initial configuration.
 *                                If NULL, the default configuration is
      used.
 * @param[in] timer_event_handler Event handler provided by the user.
 *                                Must not be NULL.
 *
 * @retval NRF_SUCCESS             If initialization was successful.
 * @retval NRF_ERROR_INVALID_STATE If the instance is already
     initialized.
 * @retval NRF_ERROR_INVALID_PARAM If no handler was provided.
 */
ret_code_t nrf_drv_timer_init(nrf_drv_timer_t const * const p_instance,
                              nrf_drv_timer_config_t const * p_config,
                              nrf_timer_event_handler_t
                                  timer_event_handler);

/**
 * @brief Function for uninitializing the timer.
 *
 * @param[in] p_instance Pointer to the driver instance structure.
 */
void nrf_drv_timer_uninit(nrf_drv_timer_t const * const p_instance);

/**
 * @brief Function for turning on the timer.
 *
 * @param[in] p_instance Pointer to the driver instance structure.
 */
void nrf_drv_timer_enable(nrf_drv_timer_t const * const p_instance);

/**
 * @brief Function for turning off the timer.
 *
 * Note that the timer will allow to enter the lowest possible
     SYSTEM_ON state
 * only after this function is called.
 *
 * @param[in] p_instance Pointer to the driver instance structure.
 */
void nrf_drv_timer_disable(nrf_drv_timer_t const * const p_instance);

/**
 * @brief Function for pausing the timer.
```

```
 *
 * @param[in] p_instance Pointer to the driver instance structure.
 */
void nrf_drv_timer_pause(nrf_drv_timer_t const * const p_instance);

/**
 * @brief Function for resuming the timer.
 *
 * @param[in] p_instance Pointer to the driver instance structure.
 */
void nrf_drv_timer_resume(nrf_drv_timer_t const * const p_instance);

/**
 * @brief Function for clearing the timer.
 *
 * @param[in] p_instance Pointer to the driver instance structure.
 */
void nrf_drv_timer_clear(nrf_drv_timer_t const * const p_instance);

/**
 * @brief Function for incrementing the timer.
 *
 * @param[in] p_instance Pointer to the driver instance structure.
 */
void nrf_drv_timer_increment(nrf_drv_timer_t const * const p_instance);

/**
 * @brief Function for returning the address of a specific timer task.
 *
 * @param[in] p_instance Pointer to the driver instance structure.
 * @param[in] timer_task Timer task.
 *
 * @return Task address.
 */
__STATIC_INLINE uint32_t nrf_drv_timer_task_address_get(
                                        nrf_drv_timer_t const * const
                                            p_instance,
                                        nrf_timer_task_t timer_task);

/**
 * @brief Function for returning the address of a specific timer
    capture task.
 *
 * @param[in] p_instance Pointer to the driver instance structure.
 * @param[in] channel    Capture channel number.
 *
 * @return Task address.
 */
__STATIC_INLINE uint32_t nrf_drv_timer_capture_task_address_get(
                                        nrf_drv_timer_t const * const
                                            p_instance,
                                        uint32_t channel);
```

```
/**
 * @brief Function for returning the address of a specific timer event.
 *
 * @param[in] p_instance  Pointer to the driver instance structure.
 * @param[in] timer_event Timer event.
 *
 * @return Event address.
 */
__STATIC_INLINE uint32_t nrf_drv_timer_event_address_get(
                                        nrf_drv_timer_t const * const
                                            p_instance,
                                        nrf_timer_event_t timer_event);

/**
 * @brief Function for returning the address of a specific timer
    compare event.
 *
 * @param[in] p_instance Pointer to the driver instance structure.
 * @param[in] channel    Compare channel number.
 *
 * @return Event address.
 */
__STATIC_INLINE uint32_t nrf_drv_timer_compare_event_address_get(
                                        nrf_drv_timer_t const * const
                                            p_instance,
                                        uint32_t channel);

/**
 * @brief Function for capturing the timer value.
 *
 * @param[in] p_instance Pointer to the driver instance structure.
 * @param[in] cc_channel Capture channel number.
 *
 * @return Captured value.
 */
uint32_t nrf_drv_timer_capture(nrf_drv_timer_t const * const p_instance
    ,
                               nrf_timer_cc_channel_t cc_channel);

/**
 * @brief Function for returning the capture value from a specific
    channel.
 *
 * Use this function to read channel values when PPI is used for
    capturing.
 *
 * @param[in] p_instance Pointer to the driver instance structure.
 * @param[in] cc_channel Capture channel number.
 *
 * @return Captured value.
 */
```

```
__STATIC_INLINE uint32_t nrf_drv_timer_capture_get(
                                        nrf_drv_timer_t const * const
                                            p_instance,
                                        nrf_timer_cc_channel_t
                                            cc_channel);

/**
 * @brief Function for setting the timer channel in compare mode.
 *
 * @param[in] p_instance Pointer to the driver instance structure.
 * @param[in] cc_channel Compare channel number.
 * @param[in] cc_value   Compare value.
 * @param[in] enable_int Enable or disable the interrupt for the
     compare channel.
 */
void nrf_drv_timer_compare(nrf_drv_timer_t const * const p_instance,
                           nrf_timer_cc_channel_t cc_channel,
                           uint32_t               cc_value,
                           bool                   enable_int);

/**
 * @brief Function for setting the timer channel in extended compare
     mode.
 *
 * @param[in] p_instance       Pointer to the driver instance structure
     .
 * @param[in] cc_channel       Compare channel number.
 * @param[in] cc_value         Compare value.
 * @param[in] timer_short_mask Shortcut between the compare event on
     the channel
 *                             and the timer task (STOP or CLEAR).
 * @param[in] enable_int       Enable or disable the interrupt for the
     compare
 *                             channel.
 */
void nrf_drv_timer_extended_compare(nrf_drv_timer_t const * const
    p_instance,
                                    nrf_timer_cc_channel_t cc_channel,
                                    uint32_t               cc_value,
                                    nrf_timer_short_mask_t
                                        timer_short_mask,
                                    bool                   enable_int);

/**
 * @brief Function for converting time in microseconds to timer ticks.
 *
 * @param[in] p_instance Pointer to the driver instance structure.
 * @param[in] time_us    Time in microseconds.
 *
 * @return Number of ticks.
 */
__STATIC_INLINE uint32_t nrf_drv_timer_us_to_ticks(
```

```
                                        nrf_drv_timer_t const * const
                                            p_instance,
                                        uint32_t time_us);

/**
 * @brief Function for converting time in milliseconds to timer ticks.
 *
 * @param[in] p_instance Pointer to the driver instance structure.
 * @param[in] time_ms    Time in milliseconds.
 *
 * @return Number of ticks.
 */
__STATIC_INLINE uint32_t nrf_drv_timer_ms_to_ticks(
                                        nrf_drv_timer_t const * const
                                            p_instance,
                                        uint32_t time_ms);

/**
 * @brief Function for enabling timer compare interrupt.
 *
 * @param[in] p_instance Pointer to the driver instance structure.
 * @param[in] channel    Compare channel.
 */
void nrf_drv_timer_compare_int_enable(nrf_drv_timer_t const * const
    p_instance,
                                      uint32_t channel);

/**
 * @brief Function for disabling timer compare interrupt.
 *
 * @param[in] p_instance Pointer to the driver instance structure.
 * @param[in] channel    Compare channel.
 */
void nrf_drv_timer_compare_int_disable(nrf_drv_timer_t const * const
    p_instance,
                                        uint32_t channel);


#ifndef SUPPRESS_INLINE_IMPLEMENTATION

__STATIC_INLINE uint32_t nrf_drv_timer_task_address_get(
                                        nrf_drv_timer_t const * const
                                            p_instance,
                                        nrf_timer_task_t timer_task)
{
    return (uint32_t)nrf_timer_task_address_get(p_instance->p_reg,
        timer_task);
}

__STATIC_INLINE uint32_t nrf_drv_timer_capture_task_address_get(
                                        nrf_drv_timer_t const * const
                                            p_instance,
```

```
                                        uint32_t channel)
{
    ASSERT(channel < p_instance->cc_channel_count);
    return (uint32_t)nrf_timer_task_address_get(p_instance->p_reg,
                        nrf_timer_capture_task_get(channel));
}

__STATIC_INLINE uint32_t nrf_drv_timer_event_address_get(
                                        nrf_drv_timer_t const * const
                                            p_instance,
                                        nrf_timer_event_t timer_event)
{
    return (uint32_t)nrf_timer_event_address_get(p_instance->p_reg,
        timer_event);
}

__STATIC_INLINE uint32_t nrf_drv_timer_compare_event_address_get(
                                        nrf_drv_timer_t const * const
                                            p_instance,
                                        uint32_t channel)
{
    ASSERT(channel < p_instance->cc_channel_count);
    return (uint32_t)nrf_timer_event_address_get(p_instance->p_reg,
                        nrf_timer_compare_event_get(channel));
}

__STATIC_INLINE uint32_t nrf_drv_timer_capture_get(
                                        nrf_drv_timer_t const * const
                                            p_instance,
                                        nrf_timer_cc_channel_t
                                            cc_channel)
{
    return nrf_timer_cc_read(p_instance->p_reg, cc_channel);
}

__STATIC_INLINE uint32_t nrf_drv_timer_us_to_ticks(
                                        nrf_drv_timer_t const * const
                                            p_instance,
                                        uint32_t timer_us)
{
    return nrf_timer_us_to_ticks(timer_us,
        nrf_timer_frequency_get(p_instance->p_reg));
}

__STATIC_INLINE uint32_t nrf_drv_timer_ms_to_ticks(
                                        nrf_drv_timer_t const * const
                                            p_instance,
                                        uint32_t timer_ms)
{
    return nrf_timer_ms_to_ticks(timer_ms,
        nrf_timer_frequency_get(p_instance->p_reg));
}
```

```
#endif // SUPPRESS_INLINE_IMPLEMENTATION


#ifdef __cplusplus
}
#endif

#endif // NRF_DRV_TIMER_H__

/** @} */
```

A.5 flash_placement.xml

```
<!DOCTYPE Linker_Placement_File>
<Root name="Flash Section Placement">
  <MemorySegment name="$(FLASH_NAME:FLASH)">
    <ProgramSection alignment="0x100" load="Yes" name=".vectors" start=
        "$(FLASH_START:)" />
    <ProgramSection alignment="4" load="Yes" name=".init" />
    <ProgramSection alignment="4" load="Yes" name=".init_rodata" />
    <ProgramSection alignment="4" load="Yes" name=".text" />
    <ProgramSection alignment="4" load="Yes" name=".dtors" />
    <ProgramSection alignment="4" load="Yes" name=".ctors" />
    <ProgramSection alignment="4" load="Yes" name=".rodata" />
    <ProgramSection alignment="4" load="Yes" name=".ARM.exidx"
        address_symbol="__exidx_start" end_symbol="__exidx_end" />
    <ProgramSection alignment="4" load="Yes" runin=".fast_run" name=".
        fast" />
    <ProgramSection alignment="4" load="Yes" runin=".data_run" name=".
        data" />
    <ProgramSection alignment="4" load="Yes" runin=".tdata_run" name=".
        tdata" />
    <ProgramSection alignment="4" keep="Yes" load="Yes" runin=".
        fs_data_run" name=".fs_data" />
  </MemorySegment>
  <MemorySegment name="$(RAM_NAME:RAM);SRAM">
    <ProgramSection alignment="0x100" load="No" name=".vectors_ram"
        start="$(RAM_START:$(SRAM_START:))" />
    <ProgramSection alignment="4" load="No" name=".fast_run" />
    <ProgramSection alignment="4" load="No" name=".data_run" />
    <ProgramSection alignment="4" load="No" name=".tdata_run" />
    <ProgramSection alignment="4" load="No" keep="Yes" name=".
        fs_data_run" address_symbol="__start_fs_data" end_symbol="
        __stop_fs_data" />
    <ProgramSection alignment="4" load="No" name=".bss" />
    <ProgramSection alignment="4" load="No" name=".tbss" />
    <ProgramSection alignment="4" load="No" name=".non_init" />
    <ProgramSection alignment="4" size="__HEAPSIZE__" load="No" name=".
        heap" />
    <ProgramSection alignment="8" size="__STACKSIZE__" load="No"
        place_from_segment_end="Yes" name=".stack" />
```

```
    <ProgramSection alignment="8" size="__STACKSIZE_PROCESS__" load="No
        " name=".stack_process" />
  </MemorySegment>
  <MemorySegment name="$(FLASH2_NAME:FLASH2)">
    <ProgramSection alignment="4" load="Yes" name=".text2" />
    <ProgramSection alignment="4" load="Yes" name=".rodata2" />
    <ProgramSection alignment="4" load="Yes" runin=".data2_run" name=".
        data2" />
  </MemorySegment>
  <MemorySegment name="$(RAM2_NAME:RAM2)">
    <ProgramSection alignment="4" load="No" name=".data2_run" />
    <ProgramSection alignment="4" load="No" name=".bss2" />
  </MemorySegment>
</Root>
```

A.6 memorymap.xml

```
<!DOCTYPE Board_Memory_Definition_File>
<root name="nRF51422_xxAC">
  <MemorySegment name="FLASH" start="0x00000000" size="0x00040000"
      access="ReadOnly" />
  <MemorySegment name="SRAM" start="0x20000000" size="0x00008000"
      access="Read/Write" />
</root>
```

A.7 thumb_crt0.s

```
// SEGGER Embedded Studio, runtime support.
//
// Copyright (c) 2014-2016 SEGGER Microcontroller GmbH & Co KG
// Copyright (c) 2001-2016 Rowley Associates Limited.
//
// This file may be distributed under the terms of the License Agreement
// provided with this software.
//
// THIS FILE IS PROVIDED AS IS WITH NO WARRANTY OF ANY KIND, INCLUDING
    THE
// WARRANTY OF DESIGN, MERCHANTABILITY AND FITNESS FOR A PARTICULAR
    PURPOSE.
//
//
//                          Preprocessor Definitions
//                          ------------------------
// APP_ENTRY_POINT
//
//   Defines the application entry point function, if undefined this setting
//   defaults to "main".
//
// INITIALIZE_STACK
//
//   If defined, the contents of the stack will be initialized to a the
```

```
//    value 0xCC.
//
// INITIALIZE_SECONDARY_SECTIONS
//
//    If defined, the .data2, .text2, .rodata2 and .bss2 sections will be initialized.
//
// INITIALIZE_TCM_SECTIONS
//
//    If defined, the .data_tcm, .text_tcm, .rodata_tcm and .bss_tcm sections
//    will be initialized.
//
// FULL_LIBRARY
//
//  If defined then
//    - argc, argv are setup by the debug_getargs.
//    - the exit symbol is defined and executes on return from main.
//    - the exit symbol calls destructors, atexit functions and then debug_exit.
//
//  If not defined then
//    - argc and argv are zero.
//    - the exit symbol is defined, executes on return from main and loops
//

#ifndef APP_ENTRY_POINT
#define APP_ENTRY_POINT main
#endif

#ifndef ARGSSPACE
#define ARGSSPACE 128
#endif
  .syntax unified

  .global _start
  .extern APP_ENTRY_POINT
  .global exit
  .weak exit

  .section .init, "ax"
  .code 16
  .align 2
  .thumb_func

_start:
  /* Set up main stack if size > 0 */
  ldr r1, =__stack_end__
  ldr r0, =__stack_start__
  subs r2, r1, r0
  beq 1f
#ifdef __ARM_EABI__
  movs r2, #0x7
  bics r1, r2
#endif
```

```
  mov sp, r1
#ifdef INITIALIZE_STACK
  movs r2, #0xCC
  ldr r0, =__stack_start__
  bl memory_set
#endif
1:

  /* Set up process stack if size > 0 */
  ldr r1, =__stack_process_end__
  ldr r0, =__stack_process_start__
  subs r2, r1, r0
  beq 1f
#ifdef __ARM_EABI__
  movs r2, #0x7
  bics r1, r2
#endif
  msr psp, r1
  movs r2, #2
  msr control, r2
#ifdef INITIALIZE_STACK
  movs r2, #0xCC
  bl memory_set
#endif
1:

  /* Copy initialised memory sections into RAM (if necessary). */
  ldr r0, =__data_load_start__
  ldr r1, =__data_start__
  ldr r2, =__data_end__
  bl memory_copy
  ldr r0, =__text_load_start__
  ldr r1, =__text_start__
  ldr r2, =__text_end__
  bl memory_copy
  ldr r0, =__fast_load_start__
  ldr r1, =__fast_start__
  ldr r2, =__fast_end__
  bl memory_copy
  ldr r0, =__ctors_load_start__
  ldr r1, =__ctors_start__
  ldr r2, =__ctors_end__
  bl memory_copy
  ldr r0, =__dtors_load_start__
  ldr r1, =__dtors_start__
  ldr r2, =__dtors_end__
  bl memory_copy
  ldr r0, =__rodata_load_start__
  ldr r1, =__rodata_start__
  ldr r2, =__rodata_end__
  bl memory_copy
  ldr r0, =__tdata_load_start__
```

```
  ldr r1, =___tdata_start___
  ldr r2, =___tdata_end___
  bl memory_copy
#ifdef INITIALIZE_SECONDARY_SECTIONS
  ldr r0, =___data2_load_start___
  ldr r1, =___data2_start___
  ldr r2, =___data2_end___
  bl memory_copy
  ldr r0, =___text2_load_start___
  ldr r1, =___text2_start___
  ldr r2, =___text2_end___
  bl memory_copy
  ldr r0, =___rodata2_load_start___
  ldr r1, =___rodata2_start___
  ldr r2, =___rodata2_end___
  bl memory_copy
#endif /* #ifdef INITIALIZE_SECONDARY_SECTIONS */
#ifdef INITIALIZE_TCM_SECTIONS
  ldr r0, =___data_tcm_load_start___
  ldr r1, =___data_tcm_start___
  ldr r2, =___data_tcm_end___
  bl memory_copy
  ldr r0, =___text_tcm_load_start___
  ldr r1, =___text_tcm_start___
  ldr r2, =___text_tcm_end___
  bl memory_copy
  ldr r0, =___rodata_tcm_load_start___
  ldr r1, =___rodata_tcm_start___
  ldr r2, =___rodata_tcm_end___
  bl memory_copy
#endif /* #ifdef INITIALIZE_TCM_SECTIONS */

  /* Zero the bss. */
  ldr r0, =___bss_start___
  ldr r1, =___bss_end___
  movs r2, #0
  bl memory_set
  ldr r0, =___tbss_start___
  ldr r1, =___tbss_end___
  movs r2, #0
  bl memory_set
#ifdef INITIALIZE_SECONDARY_SECTIONS
  ldr r0, =___bss2_start___
  ldr r1, =___bss2_end___
  mov r2, #0
  bl memory_set
#endif /* #ifdef INITIALIZE_SECONDARY_SECTIONS */
#ifdef INITIALIZE_TCM_SECTIONS
  ldr r0, =___bss_tcm_start___
  ldr r1, =___bss_tcm_end___
  mov r2, #0
  bl memory_set
```

```
#endif /* #ifdef INITIALIZE_TCM_SECTIONS */

  /* Initialise the heap */
  ldr r0, = __heap_start__
  ldr r1, = __heap_end__
  subs r1, r1, r0
  cmp r1, #8
  blt 1f
  movs r2, #0
  str r2, [r0]
  adds r0, r0, #4
  str r1, [r0]
1:

  /* Call constructors */
  ldr r0, =__ctors_start__
  ldr r1, =__ctors_end__
ctor_loop:
  cmp r0, r1
  beq ctor_end
  ldr r2, [r0]
  adds r0, #4
  push {r0-r1}
  blx r2
  pop {r0-r1}
  b ctor_loop
ctor_end:

  /* Setup initial call frame */
  movs r0, #0
  mov lr, r0
  mov r12, sp

  .type start, function
start:
  /* Jump to application entry point */
#ifdef FULL_LIBRARY
  movs r0, #ARGSSPACE
  ldr r1, =args
  ldr r2, =debug_getargs
  blx r2
  ldr r1, =args
#else
  movs r0, #0
  movs r1, #0
#endif
  ldr r2, =APP_ENTRY_POINT
  blx r2

  .thumb_func
exit:
#ifdef FULL_LIBRARY
```

```
  mov r5, r0 // save the exit parameter/return result

  /* Call destructors */
  ldr r0, =___dtors_start___
  ldr r1, =___dtors_end___
dtor_loop:
  cmp r0, r1
  beq dtor_end
  ldr r2, [r0]
  add r0, #4
  push {r0-r1}
  blx r2
  pop {r0-r1}
  b dtor_loop
dtor_end:

  /* Call atexit functions */
  ldr r2, =_execute_at_exit_fns
  blx r2

  /* Call debug_exit with return result/exit parameter */
  mov r0, r5
  ldr r2, =debug_exit
  blx r2
#endif

  /* Returned from application entry point, loop forever. */
exit_loop:
  b exit_loop

  .thumb_func
memory_copy:
  cmp r0, r1
  beq 2f
  subs r2, r2, r1
  beq 2f
1:
  ldrb r3, [r0]
  adds r0, r0, #1
  strb r3, [r1]
  adds r1, r1, #1
  subs r2, r2, #1
  bne 1b
2:
  bx lr

  .thumb_func
memory_set:
  cmp r0, r1
  beq 1f
  strb r2, [r0]
  adds r0, r0, #1
```

```
  b memory_set
1:
  bx lr

  // default C/C++ library helpers

.macro HELPER helper_name
  .section .text.\helper_name, "ax", %progbits
  .global \helper_name
  .weak \helper_name
\helper_name:
  .thumb_func
.endm

HELPER ___aeabi_read_tp
  ldr r0, =___tbss_start___-8
  bx lr
HELPER ___heap_lock
  bx lr
HELPER ___heap_unlock
  bx lr
HELPER ___printf_lock
  bx lr
HELPER ___printf_unlock
  bx lr
HELPER ___scanf_lock
  bx lr
HELPER ___scanf_unlock
  bx lr
HELPER ___debug_io_lock
  bx lr
HELPER ___debug_io_unlock
  bx lr
HELPER abort
  b .
HELPER ___assert
  b .
HELPER ___aeabi_assert
  b .
HELPER ___cxa_pure_virtual
  b .
HELPER ___cxa_guard_acquire
  ldr r3, [r0]
#if defined(___thumb___) && !defined(___thumb2___)
  movs r0, #1
  tst r3, r0
#else
  tst r3, #1
#endif
  beq 1f
  movs r0, #0
  bx lr
```

```
1:
  movs r0, #1
  bx lr
HELPER ___cxa_guard_release
  movs r3, #1
  str r3, [r0]
  bx lr
HELPER ___cxa_guard_abort
  bx lr
HELPER ___getchar
  ldr r0, =-1 // EOF
  bx lr
HELPER ___putchar
  ldr r0, =-1 // EOF
  bx lr
  // char ___user_locale_name_buffer[];
  .section .bss.___user_locale_name_buffer, "aw", %nobits
  .global ___user_locale_name_buffer
  .weak ___user_locale_name_buffer
  ___user_locale_name_buffer:
  .word 0x0

#ifdef FULL_LIBRARY
  .bss
args:
  .space ARGSSPACE
#endif

  /* Setup attibutes of stack and heap sections so they don't take up room in the elf file */
  .section .stack, "wa", %nobits
  .section .stack_process, "wa", %nobits
  .section .heap, "wa", %nobits
```