NTNU – Trondheim
Norwegian University of
Science and Technology

Energy Harvesting for Sensor Nodes in the Internet of Things

Erick Castillo

Embedded Computing Systems
Submission date: July 2015
Supervisor: Kjetil Svarstad, IET
Co-supervisor: Marius Grannæs, Silicon Labs

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

NTNU – Trondheim
Norwegian University of Science and Technology

Energy Harvesting for Sensor Nodes in the Internet of Things

Erick Alejandro Castillo García

July 2015

MASTER THESIS
Department of Electronics and Telecommunications
Norwegian University of Science and Technology

Supervisor 1: Kjetil Svarstad

Supervisor 2: Marius Grannæs

Abstract

Wireless sensor networks have an extensive range of applications in the real world. From military uses saving lives, to environmental applications monitoring the fauna and weather conditions, but also by checking the health of patients and even by automating our homes. This work presents a solution to implement an energy harvesting sensor network. By using solar energy to power a sensor node we can extend its lifetime beyond the one powered only by batteries. Moreover, this solution attempts to be energy efficient and to achieve a communication scheme in order to create a sensor network where nodes read environmental data and transmit back to a sink node. The communication scheme was successful to synchronize two nodes and transmit packets between them without collisions and avoiding loss of data due to lack of energy. Furthermore, the duty cycling algorithm allowed the node to operate at its maximum performance level, making the best use of its energy available without depleting it.

Contents

List of Figures

List of Tables

List of Acronyms

AEM	Advanced Energy Monitor
API	Application Programming Interface
APO	Application Object
BMAC	Berkeley Medium Access Control
CRC	Cyclic Redundancy Check
DC	Duty Cycle
EEHF	Environmental Energy Harvesting Framework
ENO	Energy Neutral Operation
EWMA	Exponentially Weighted Moving Average
FFD	Full Function Device
GPIO	General Purpose Input/Output
IDE	Integrated Development Environment
IoT	Internet of Things
IP	Internet Protocol
I2C	Inter-Integrated Circuit
Li-ion	Lithium-ion
LQ	Linear-Quadratic

MAC Medium Access Control

MCU Microcontroller Unit

NiMH Nickel Metal Hydride

PAN Personal Area Network

RF Radio Frequency

RFD Reduced Function Device

RTC Real-Time Counter

RX Reception

SMAC Sensor Medium Access Control

SPI Serial Peripheral Interface

TDMA Time Division Multiple Access

TMAC Timeout Medium Access Control

TX Transmission

USB Universal Serial Bus

WSN Wireless Sensor Network

ZDO Zigbee Device Object

Chapter 1

Introduction

The increasing advances in semiconductor materials have contributed to the development of smaller and less expensive wireless sensor networks (WSNs) over the past few years. WSNs have stepped from military applications, to environmental, health and home applications, entering into our daily life to facilitate everyday tasks. Moreover, the proclivity towards the Internet of Things (IoT) is growing due to faster network connections and the increase of smart devices that allow users to connect different electronic appliances and make their use more efficient. What is more, the use of environmental energy is gathering more strength as it can offer free "unlimited" energy and at the same time being environmentally-safe.

Problem Formulation

Green energy and sustainability are terms that have become increasingly important in the development of new technologies. Nowadays, a wide range of industries have adopted these eco-friendly practices not only for the environmental protection, but also for the economic and social benefits they provide. By using energy harvesting to power small devices such as sensor networks, besides the benefits formerly mentioned, these devices can be deployed in areas where power is not easily obtained and their lifespan can be much longer than battery-powered devices. However, traditional energy saving techniques are not suited for energy harvesting sensor networks since the nodes depend on the variability of the energy source and therefore, an efficient synchronization method enabling the communication between the nodes is needed in order to exploit the benefits of environmental energy.

1.1 Objectives

The main objectives of this Master's project are:

1. Find the necessary power requirements for a single node to operate and compare the energy sources to find the most suitable one for this application.
2. Having a single node powered by energy harvesting, it has to send data as soon as it has enough energy to a receiver which is always on and listens on any data that comes along.
3. Having both nodes powered by energy harvesting, find a way to synchronize both devices to communicate between them.
4. Expand the network to multiple nodes.

1.2 Limitations

Taking into consideration that the energy harvesting kit used in this work only contains a simple array of capacitors as an energy buffer and it is not enough to retain enough energy for the node's operation, a simulation of the battery level which takes into account the energy harvested and power consumption of the node is used instead. Furthermore, the communication between the nodes is tested at a close range, therefore any packet losses due to large separations between the nodes are disregarded.

1.3 Methodology

In order to fulfill the objectives in this work, an experimental approach is followed. First, an extensive literature research regarding the background theory, existing applications and techniques has to be done. Then, after having obtained relevant information, a basic solution has to be proposed and experiments made to test the theory and further enhance the solution. Once the implementation is completed, then additional testing will be done to ensure the solution is correct. Finally, when the implementation is fully tested, simulations and experiments will be performed to analyze the different configurations of parameters and determine which results work best to propose a final solution.

1.4 Contributions

- Simulated the node's battery level with respect to the node's power consumption and energy harvested.
- Implemented the Linear Quadratic Tracking algorithm for duty cycling to reduce energy consumption.
- Developed a wireless communication scheme that allows nodes with different duty cycles to communicate between each other, while reducing packet collisions and loss of data.
- Accomplished a strategy to synchronize the clocks of the nodes after a certain period.
- Programmed and tested a low-energy application for sensor nodes encompassing all of the above.

1.5 Structure of the Report

The rest of the report is organized as follows.

Chapter 2 presents a background research on the applications and design considerations of wireless sensor networks. Several traditional techniques for energy conservation are described. Moreover, different energy harvesting sources and storage technologies are compared and a few duty cycling and routing techniques which are envisioned for energy harvesting systems are introduced. In the last part of the chapter, two protocols for implementing the network communication are discussed. Finally, software tools to facilitate the development of the project are presented.

Chapter 3 presents the hardware and software implementation of the sensor node. First, the hardware architecture is introduced, then the energy storage required is discussed and ultimately the software solution is explained.

Chapter 4 discuss the repercussion of the data rate in the transmission range and transmission delay. Furthermore, the results of the energy measurements from the application by using different configurations are presented. Additionally, the behavior of the duty cycling algorithm implementation is analyzed. And finally, the tests of the network communication are examined.

Chapter 5 presents a brief summary of the work achieved and a discussion about the findings. In addition, some recommendations for future work are proposed.

Chapter 2

Theoretical Background

In this chapter, we present basic concepts and design considerations for wireless sensor networks, as well as a classification of techniques for energy conservation in battery operated sensor networks. Furthermore, we outline the factors to consider when implementing an energy harvesting sensor network and a few different wireless protocols that can be used to adapt it to the Internet of Things.

2.1 Wireless sensor networks

Over the past decade, there has been a great amount of research regarding WSNs due to their broad field of applications. These networks, as seen in Figure 2.1, consist of small sensor nodes deployed over some region, whose main functionality is to gather data from the environment and report back to a *sink* through some wireless network protocol, which then in turn is connected to the Internet.

In the following subsections we will present some of the most common applications for WSNs and additionally, some considerations which need to be taken into account when designing one.

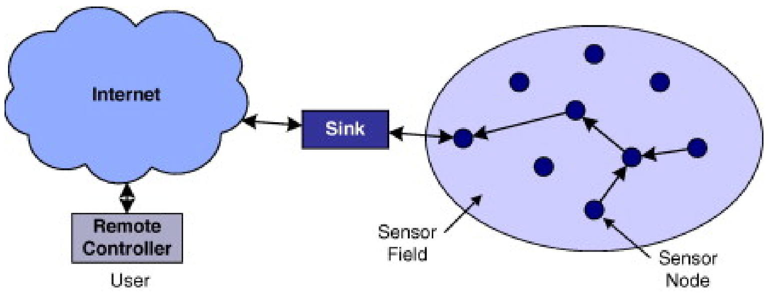


Figure 2.1: Wireless Sensor Network.[1]

2.1.1 Commercial applications

Due to their low production costs, fault tolerance and rapid deployment, WSNs have different applications in the military, some of them being monitoring friendly forces, equipment and ammunition; battlefield surveillance; reconnaissance of opposing forces and terrain; targeting; battle damage assessment; and nuclear, biological and chemical attack detection and reconnaissance [9].

Environmental applications also take advantage of the scalability and low power consumption of the sensor nodes, making it feasible to monitor the migration of wild animals for instance. Another example is sensing a region of the environment and detecting changes in temperature or atmospheric pressure in order to prevent natural catastrophes.

Because of the small size of the nodes, some of the health applications include: integrated patient monitoring, drug administration in hospitals, telemonitoring of human physiological data and tracking of doctors and patients inside a hospital [9].

As for the home applications, wireless sensor nodes can be integrated into almost every electronic device, allowing them to communicate with each other and create smart homes that adapt to the user's requirements. These smart homes can also join the Internet of Things, giving the user the possibility to control every device in the network from a smart phone.

2.1.2 Design considerations

There are several factors to consider when designing a wireless sensor network, however they depend mostly on the specific application it is intended for. The following are mentioned in [9]:

1. Fault tolerance: the level of tolerance to failure that nodes should have is dependent on the type of environment they will be exposed to. According to [9], the reliability of a node not having a failure between the time interval (0,t) can be modelled as a Poisson distribution.
2. Scalability: a sensor network can vary from hundreds to thousands of nodes, therefore the communication protocols should be able to support this increase in the number of devices.
3. Production costs: the cost of a sensor node must be less than the cost of a traditional sensor in order to justify the overall cost of the network.
4. Hardware constraints: every node is composed of four basic components: a sensing unit, a processing unit, a transceiver unit and a power unit. Moreover, some nodes can have a location finding system, a mobilizer and an energy harvesting unit. The main requisites of these nodes are that they should be small, consume very low power and be able to work autonomously.
5. Network topology: sensor nodes can be either mass deployed in the field or positioned individually, hence the network protocol should adapt to different topologies. In addition, nodes can fail or new nodes can be re-deployed, changing the initial topology, and the network should then be able to re-organize itself.
6. Environment: since the nodes can be deployed in different environments, they should be able to work under different conditions depending on each application, for instance high pressure, extreme heat or cold, noisy environments, among others.
7. Transmission media: different wireless communications can be used such as radio, infrared or optical. Being radio the most adequate for WSNs, the industrial, scientific and medical (ISM) bands can be used due to their free and global availability, although for this same reason, it should be taken into consideration that there could be interference from existing applications.

8. Power consumption: there are three domains where the nodes spend their energy: sensing, communication and processing. The sensor energy consumption depends entirely on the type of sensor used and the application, while the processing energy consumption varies depending on energy reduction techniques and low energy modes, as for the radio, it consumes much more energy than processing data, in such a way that transmitting a single bit has the same energy cost as processing a thousand operations [10]. In addition, considering short-range communications, radios consume the same power whether transmitting or receiving [10][11]. Hence, the techniques for minimizing power consumption in WSNs mainly focus on reducing the number of transmissions or the data to be transmitted.

2.2 Techniques for energy conservation in wireless sensor networks

As mentioned in the previous section, the main goal in the design of WSNs is the reduction of power consumption. There are three methods discussed in [1] to achieve this, namely, duty cycling, data-driven approaches and mobility-based approaches. In the following subsections we will present relevant duty cycling techniques and mention some basic information about the other approaches.

2.2.1 Duty cycling

Duty cycling techniques consist on switching the radio off or to sleep mode whenever it is not being used. The fraction of time the nodes are active is called the *duty cycle*, this is usually predefined, although it could be dynamic depending on the application. Moreover, the nodes must have some sleep/wake up scheduling algorithm to coordinate when the communication should happen.

We can classify the following techniques depending on the layer of the network architecture they are implemented at. They can either be sleep/wakeup protocols implemented at the network or application layer or Medium Access Control (MAC) protocols at the data link layer.

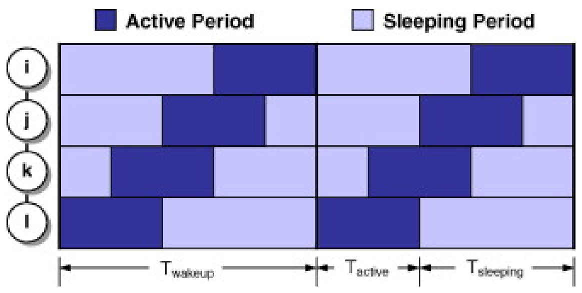


Figure 2.2: Staggered wakeup pattern.[1]

2.2.1.1 Sleep/wakeup protocols

These techniques allow a great flexibility as they can be tailored to any application and used on any MAC protocol without relying on topology aspects. They can be further classified into two categories depending on their level of synchronization.

Fully Synchronized Pattern. This is a simple scheme where all the nodes wake up periodically every T_{wakeup} and remain active for a fixed T_{active}. Due to the large size of its active and sleeping periods, it does not require a very precise clock synchronization. Its main drawbacks are, however, that since all nodes are awake at the same time and may try to transmit, a large number of collisions might occur; it is also not very flexible since the time periods are fixed and it does not adapt to variations in traffic or topology.

Staggered Wakeup Pattern. In this scheme, the network topology is viewed as a tree, with the sink being the root and the sensor nodes the leaves. Nodes located at different levels of the tree, wake up at different times, taking into consideration that there should be some overlap between adjacent levels in order for parents to communicate with their children. In Figure 2.2, we can see an example of the active times for the different levels of nodes and how the neighbors' times overlap

The main advantages are that less collisions occur since only some of the nodes transmit at the same time and hence the T_{active} can be shorter than in the previous scheme. In addition, this scheme is suitable for data aggregation. However, it has some drawbacks such as collisions between the nodes in the same level and fixed time periods.

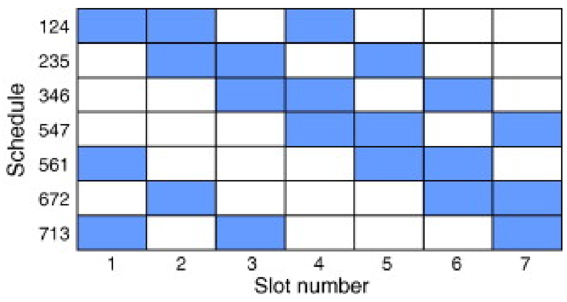


Figure 2.3: Asynchronous Wakeup Protocol.[1]

Asynchronous Wakeup Protocol. In this protocol, every node wakes up independently from the others with the guarantee that neighbors will always have overlapped active periods within a specified number of cycles. Each node is associated with a Wakeup Schedule Function to generate a wakeup schedule. For neighboring nodes, their wakeup schedules have to overlap in order to be able to communicate. In Figure 2.3, an example based on a symmetric (7,3,1)-design is illustrated. This means that all the nodes have the same duty cycle, each schedule repeats every seven slots, each schedule has three active slots out of seven and any two schedules overlap for at most one slot.

This protocol is resilient to packet collisions and variations in the network topology, moreover it does not require a tight synchronization among the nodes. Nonetheless, it consumes more energy due to the nodes having to wake up more often, in addition the packet latency is large and it is not possible to broadcast messages to all the neighbors.

2.2.1.2 MAC protocols with low duty-cycle

MAC protocols optimize the medium access functions based on the specific sleep/wake up patterns and there are many different implementations. They can be classified into two types of protocols, the first one based on Time Division Multiple Access (TDMA) and the second one which is contention-based.

TDMA-based MAC protocols

In this type of protocols, the time is divided into periodic frames consisting of slots, then each node is assigned to one or more slots depending on the algorithm and they turn on their radio only during their own slots. TDMA protocols are energy efficient, but have limited flexibility and scalability due to slot allocation, they also need a tight clock synchronization and are sensitive to interference. Due to these reasons they are not widely used in WSNs.

Contention-based MAC protocols

In contention-based MAC protocols, being the most popular amongst the MAC protocols, the sleep/wakeup scheme is tightly coupled with the MAC protocol. They are robust and scalable, can adapt to traffic variations, but have high energy consumption due to contention and collisions.

Berkeley Medium Access Control (BMAC). The main goals of this protocol are: achieving low power operation, effective collision avoidance, efficient channel utilization, reconfiguration, tolerance to changing RF and networking conditions and scalability [12].

BMAC uses Clear Channel Assessment (CCA) and packet backoffs for channel arbitration. By using CCA it distinguishes between the noise and the ongoing transmissions to determine if the channel is clear. Furthermore, it uses link layer acknowledgments for reliability. Whenever a node receives a unicast packet it immediately sends an acknowledgment packet back and the node receiving the acknowledgment records this in its message buffer. Finally, BMAC implements a technique called Low Power Listening (LPL). Whenever the node wakes up it turns on the radio and listens for activity, if it detects it, then it powers up entirely to receive the packet and returns to sleep after doing so, however if no packet is received it uses a timeout to go back to sleep. This protocol also provides a set of interfaces to configure its operation, however network services like organization, synchronization and routing are not provided by BMAC and have to be built above its implementation.

Sensor Medium Access Control (SMAC). This protocol uses a low duty cycle operation to reduce idle listening, which according to [13] accounts for the 50%-100% of the energy required for receiving radio transmissions. Moreover, to reduce control overhead and latency, it implements a coordinated sleeping among neighboring nodes by exchanging special packets to synchronize their sleep/wakeup periods. Nodes can establish their own schedule or follow the one of a neighbor, forming a virtual cluster. In order to allow communication between different clusters, nodes can follow different schedules as long as they do not overlap. And to support the transmission of different packets, the channel access time is split into two parts, one for synchronization packets and the other for data transfer.

In addition, to overcome the problem of high latencies in multi-hop networks this protocol uses an adaptive listening scheme. This technique allows a node overhearing a transmission to wake up for a short period of time at the end of the transmission, so if the node is the next hop, it can receive the data immediately. One last main feature of SMAC is message passing, this helps to transmit long messages by dividing them into smaller fragments and sending them in a burst.

It is worth mentioning that the parameters of the protocol such as the listen and sleep periods are constant. However, [14] introduces a modified version of this protocol called Timeout Medium Access Control (TMAC) which dynamically ends the active part of the duty cycle based on traffic load thus reducing energy wasted on idle listening.

DMAC. This protocol is optimized for data gathering trees in WSNs and avoids the sleep latency that occurs in SMAC and TMAC by using a staggered sleep/wakeup schedule according to the nodes' position in the tree network topology [15].

Each node has one slot where they can transmit a single packet, although they can request for additional slots if needed, this way the network can adapt to traffic variations. A data prediction scheme is used to allow multiple children of a node to transmit their packets in the same interval. While a More-to-Send (MTS) packet is used when nodes of the same level in the hierarchy tree with different parents compete for channel access.

2.2.2 Data-driven approaches

The main objective of these approaches is to avoid transmitting unnecessary data, thus reducing the communication costs. There are three categories in which they can be classified.

The first one, in-network processing, consists of using a technique called *data aggregation* in the intermediate nodes between the source nodes and the sink. Data aggregation combines the data either by compressing the information of two packets into one or by merging the data, only the first option allows the individual packets to be reconstructed at the sink. Furthermore, this approach is relatively complex and usually application-specific.

The second category, data compression, reduces the amount of information by encoding it at the source nodes and decoding it at the sink.

The last one, data prediction, uses a model of a sensed phenomenon which can predict the values sensed by the nodes and it resides both at the source nodes and the sink. The source node compares the sensed information to the one provided by the model and if it falls out of a certain tolerance it updates the model at the sink, whereas the sink uses the information from the model instead of the one from the source nodes.

Additionally, since these approaches are based on reducing the data transmitted, they can be combined together with duty cycling techniques to benefit from the energy reduction characteristics of both.

2.2.3 Mobility-based approaches

Taking into consideration that the nodes which are closer to the sink have to relay more packets than the rest of the nodes in the network, they are more likely to run out of energy sooner. In order to diminish this problem, mobility-based approaches consist on making some of the nodes mobile to alter the traffic flow and additionally reducing the path length of the communication between nodes and the sink.

2.3 Energy harvesting in wireless sensor networks

Energy reduction techniques help sensor networks work for extensive periods of time, however, this time is still limited by the capacity of the battery and the behavior of the nodes. Another approach to help extend the lifetime of these devices is to use an energy harvesting unit to power the sensor nodes.

An energy harvesting node can be defined as any system which draws part or all of its energy from the environment. A key distinction of this energy is that it is potentially infinite, though there might be a limit on the rate at which it can be used [16]. In the next subsections we present some of the different energy sources which can be used for energy harvesting, also a few energy storage technologies suited for sensor networks and some energy prediction methods. Furthermore, the concept of energy neutral operation is introduced and a few energy harvesting aware techniques for sensor networks are discussed.

2.3.1 Comparison of energy harvesting sources

There are many different energy sources which can be obtained from the environment around us, only suitable transducers are needed to efficiently harvest them. In [17] the following classification is adopted to present the different energy sources:

Electromagnetic radiation

- Solar – this is a very accessible and predictable energy source [Fig. 2.4a]. Even though the energy available depends on the time of the day, the latitude, and the atmospheric conditions; commercially available solar cells provide a typical efficiency of about 15%-20% [17]. As a result, it is a widely used source from power plants to small devices.
- Radio frequency signals – these signals [Fig. 2.4b] are used to power passive electronic devices such as RFID (Radio-frequency identification) tags. However, these devices only respond to a certain frequency and they have to be close to the radio source, making it difficult to implement on wireless sensor networks.

Thermal

In order to harvest this type of energy, a thermal gradient is required. According to the Carnot efficiency, the greater the temperature difference, the greater the conversion efficiency is. A commercial device can provide $100\mu W$ from a $10K$ temperature difference in a $9.3mm$ diameter device $1.4mm$ thick [17].

Mechanical

- Wind/water flow – these sources are also widely available and used mostly at big scales such as in wind turbines [Fig. 2.4c] and hydroelectric plants [Fig. 2.4d]. However due to their size, their adoption into smaller devices is not that common.
- Vibrations – these can be found in most environments and the energy extracted depends on their amplitude and frequency. There are several types of transducers to harvest this type of energy, for instance, piezoelectric, electrostatic and electromagnetic.

(a) Solar energy (b) Radio signals (c) Wind flow (d) Water flow

Figure 2.4: Energy Harvesting Sources.

2.3.2 Storage technologies

Different energy storage technologies exist, each one having their advantages and disadvantages. The three most adequate for wireless sensor networks according to their characteristics presented in [18] are: Lithium-Ion (Li-ion), Nickel Metal Hydride (NiMH) and super-capacitors.

Lithium batteries yield a high output voltage, high energy density, high efficiency, low self-discharge rate and they do not suffer from memory effect-loss of energy capacity due to repeated shallow recharge. Nonetheless, they require a high pulsating charging current and usually an additional charging circuit is used for this.

NiMH batteries have reasonably high energy density, high number of recharge cycles and can be trickle charged, meaning that they do not need an additional charging circuit. On the other hand, they do suffer from the memory effect-loss and their charge-discharge efficiency is lower than Li-ion batteries.

Super-capacitors provide a high charge-discharge efficiency, no memory effect-loss, can be trickle charged and theoretically have an infinite number of recharge cycles. Despite of this, they have a high self-discharge rate and low weight-to-energy density.

In view of the different characteristics of each technology, a possible implementation for wireless sensor networks could be to use an array of super-capacitors as primary storage and a Li-ion battery for secondary storage which is charged whenever the super-capacitor exceeds its capacity. This could be managed either by software as in Prometheus or by hardware as in AmbiMax, both are energy harvesting sensor nodes presented in [18].

2.3.3 Energy prediction methods

The difference between battery-based and energy harvesting sensor networks is that for the later one it is possible to increase the performance of the nodes knowing beforehand that there is enough energy to spend until the next recharge cycle. An important factor to achieve this is an energy prediction method.

According to [18] *effective energy*, which can help to attain energy neutrality, is a function of the expected energy from recharge in a subsequent duration, the energy consumption by non-optional tasks and the current battery level. The following three energy prediction methods are presented in [18]:

Environmental Energy Harvesting Framework (EEHF). This framework uses the concept of an epoch which is based on a single day. In order to predict the energy in future epochs, it uses an autoregressive filter on energy consumption and energy availability over a finite number of previous epochs.

Enhanced-EEHF. This method increases the precision of the EEHF algorithm by dividing each epoch into periods and obtaining estimates for each period instead. Furthermore, it takes into account not only the history of previous epochs, but also the trend between periods in the current cycle.

Exponentially Weighted Moving-Average (EWMA) filter. In this approach, the day is divided into forty-eight slots and the available energy in each slot is predicted using the weighted average of the energy availability of previous days for that same slot and the energy estimate of the previous slot. In order to determine the average energy availability of a slot i, the following equation is used:

$$\bar{x}(i) = \alpha \bar{x}(i-1) + (1-\alpha)x(i) \tag{2.1}$$

where $x(i)$ is the actual generated energy, and α the weighting factor. According to experiments in [16] a weighting factor of 0.5 is an optimal value for minimum prediction error. Moreover, this method can adapt to seasonal variations.

2.3.4 Techniques for energy harvesting aware WSNs

While in typical wireless sensor networks the objective is to minimize energy consumption to prolong the battery's life, in energy harvesting WSNs there is a different approach called *energy neutral operation* (ENO). The objective is to consume less energy than the harvested energy in order to achieve a perpetual functioning of the sensor node.

Considering the advantage that energy harvesting sensor nodes have to tune their performance based on the energy availability, we present two techniques to dynamically adapt the duty cycle of the nodes to achieve the best performance possible. Furthermore, we present two networking techniques which take advantage of the energy harvesting as well and whose implementation is tightly coupled with the nodes' individual behavior.

2.3.4.1 Node-level adaptations

Adaptive duty cycling based on a predicted energy model

A technique to adapt the duty cycle of the sensor nodes based on the predicted energy model is presented in [16]. The goal is to dynamically choose the highest duty cycle possible allowed, while maintaining energy neutral operation. This analysis is intended for predictable energy sources, and in this case solar energy is used.

It basically consists of three steps: first, it uses a EWMA filter to predict the energy availability; then, it solves an optimization problem to obtain the optimal duty cycle based on some mathematical analysis; and finally, considering the predicted and actual energy levels it adapts the duty cycle to account for excess or lack of energy.

Adaptive control of duty cycling based on linear quadratic tracking

Another approach is presented in [8] which aims to achieve an ENO-max condition, this consists of obtaining the maximum performance while maintaining energy neutral operation. However, the main difference with the previous technique is that this one does not make any assumptions about the energy model and furthermore, it provides a way to reduce the variations in the duty cycle.

First, an objective function is defined in terms of the node's battery level, which if minimized, it should achieve maximum performance while maintaining energy neutral operation, this condition is referred as ENO-max condition. Considering a node's initial battery level as $B_0 \in [0,1]$ and its battery level at any discrete time t as $B_t \in [0,1]$, equation 2.2 satisfies the ENO-max condition due to the fact that the excess energy is being used while preserving a certain battery level.

$$B_t = B_0 \quad \forall t > 0 \tag{2.2}$$

However, it is not possible for a node to adjust its duty cycle to maintain this condition for all t > 0. Therefore, a cost function (2.3) is defined to obtain an optimal duty cycle by minimizing the average squared deviation of the battery level from its initial level. In case of energy leakage from the battery, this can be viewed as an increase in power consumption, which will be compensated when minimizing the function.

$$\lim_{N\to\infty} \frac{1}{N} \sum_{t=1}^{N} (B_t - B_0)^2 \tag{2.3}$$

Next, a solution to minimize this cost function is presented based on a problem in adaptive control theory. The linear-quadratic tracking problem attempts to apply external control to a dynamic system in order to keep the output at a desired value or trajectory over time by minimizing a cost function. Moreover, the dynamics of the system are linear, while the cost function is quadratic. In addition, for this case the desired trajectory of the output is constant. Taking this into consideration, a first order, discrete-time, linear dynamic system with colored noised is assumed (2.4) where y is the output of the system, u is the control, w is the mean zero input noise and $a, b, c \in \Re$ are real-valued coefficients.

$$y_{t+1} = ay_t + bu_t + cw_t + w_{t+1} \tag{2.4}$$

Now considering that y^* is the constant output value desired, which for this case is B_0, equation 2.3 can be rewritten as:

$$\lim_{N\to\infty} \frac{1}{N} \sum_{t=1}^{N} (y_t - y^*)^2 \tag{2.5}$$

And by minimizing the cost function (2.5), the optimal control can be obtained using the following equation:

$$u_t = \frac{y^* - (a+c)y_t + cy^*}{b} \tag{2.6}$$

which does not depend on the noise w, but depends on the noise coefficient c of previous values.

Additionally, taking into consideration that in this case the coefficients a, b and c are not known a priori, they can be estimated online using gradient descent techniques [8]. This is done by introducing a parameter vector $\theta = (a+c, b, c)^T$ to represent the true coefficients and a feature vector $\phi_t = (y_t, u_t, -y^*)^T$, where the parameter vector is estimated as:

$$\hat{\theta}_{t+1} = \hat{\theta}_t + \frac{\mu}{\phi_t^T \phi_t} \phi_t (y_{t+1} - \phi_t^T \hat{\theta}_t) \tag{2.7}$$

and μ is a positive constant step-size parameter.

To adapt this linear quadratic solution to the sensor nodes some considerations have to be made. First, y_t is the battery level B_t at time t, u_t is the node's duty cycle at time t, and w_t models the moving average of battery level increments. Secondly, the battery inefficiencies are not modeled directly given that the algorithm only observes the actual harvested energy. And finally, since most batteries have approximate linear discharge and recharge rates in the middle region of their voltage discharge curves, the linearity of the system holds as long as the voltage remains in this range.

Finally, this approach provides the possibility to decrease the variance of the control outputs by using the following simple exponential weighting scheme:

$$\bar{u}_t = \bar{u}_{t-1} + \alpha(u_t - \bar{u}_{t-1}) \tag{2.8}$$

where $\bar{u_t}$ is the smoothed control signal and α is the smoothing parameter $\in (0,1]$. For larger values of α, the smoothing occurs only over the last data points, while for lower values of α, it occurs over a longer history of values.

However this smoothed control signal cannot be applied directly to the duty cycle, since it would prevent in some cases to quickly adapt to large variations, which could result in a violation of the ENO-max condition. Therefore a tradeoff parameterized by $\beta \in [0,1]$ which determines the relative contributions of each control signal is used to obtain the duty cycle as follows:

$$\rho_t = \beta u_t + (1-\beta)\bar{u}_t \tag{2.9}$$

where smaller values of β will reduce the variance of the duty cycle by giving more weight to the smoothed control signal, while higher values of beta are used when the variance is irrelevant.

2.3.4.2 Network-level design

Harvesting-aware routing

An efficient way to implement routing protocols in battery powered networks is to use the battery level as a cost metric. However, in energy harvesting networks this is not enough, the harvesting opportunity also needs to be taken into consideration. This technique, presented in [19], first computes the energy potential for each node with the following equation:

$$E_i = w * \rho_i + (1 - w) * B_i \tag{2.10}$$

where w is a weight parameter $(0 \leq w \leq 1)$, ρ_i is the expected rate of energy harvesting at a node i and B_i is the residual battery level at the same node. To predict the energy harvested at each node the EEHF method is used.

Subsequently, it uses the inverse of the energy potential at a node i as the communication cost for all links into that node, this is modelled as follows:

$$c_{ki}(e_{ki}) = 1/E_i \quad \forall k \in \{k | e_{ki} \in E_{comm}\} \tag{2.11}$$

where e_{ki} is the possible wireless hop between the pair of nodes k and i (also called edge), and E_{comm} is the set of edges across which radio communication is feasible for the deployed network topology and radio hardware used.

Finally having obtained the cost metric for each node, a Bellman-Ford method, which is a distributed route discovery algorithm, is used to find the lowest cost routes.

Low-latency routing

Considering a random graph topology, where nodes can have multiple parents, [20] evaluates three approaches to reduce the routing latency in the sensor network.

The first one, only considers the next-hop latency therefore the node transmits the message to the parent with the lowest latency. The second approach chooses the parent which allows the message to reach a grandparent with the lowest latency, this requires additional information however, increasing traffic and computational overhead. The last approach uses the next-hop latency and also considers the estimated latency over the whole path from the parent to the sink. If node x is sending a message to a parent y at a time t, then the latency is expressed as $L(x, y, t)$, the parent set is P_x, and the estimated latency from the parent y to the sink is E_y. The total latency to the sink node can be obtained with the following equation:

$$S(x,t) = \min_{y \in P_x}\{L(x,y,t) + E_y\} \tag{2.12}$$

where E_y is calculated from the sink to the leaf nodes, and if T is the least common multiple of the duty cycles of the parents of node y, then E_y can be determined as follows:

$$E_y = \frac{1}{T}\int_T S(y,t)\,dt$$
$$E_y = \frac{1}{T}\int_T \min_{z \in P_z}\{L(y,z,t) + E_z\}\,dt \tag{2.13}$$

Moreover, this last approach does not add extra data transfer since E_y can be transmitted with the INIT messages, and therefore it is the most suitable for energy harvesting sensor networks.

2.4 Wireless sensor networks for the Internet of Things

WSNs were designed to collect data from the environment and report back to a gateway where the information would then be transmitted to a user interface, being only a one-way communication. Nevertheless, this is not the case anymore with the Internet of Things trending in the past few years, WSNs should be able to send and also receive data in order for the user to control them from a mobile device and this requires a two-way communication from end-to-end devices.

There are two options we will consider for implementing the network communication. The first one being Zigbee, which is a low cost, low power, low data rate, widely used wireless mesh topology; and the second one is Thread, a simple, secure and low power network designed for the home environment.

2.4.1 Zigbee

2.4.1.1 Architecture

The Zigbee standard consists of 4 layers, namely, physical, MAC, network and application layer as seen in Fig. 1.4. The first two layers take full advantage of the physical radio specified by IEEE 802.15.4, whereas the network, security and application software, which are implemented in the firmware stack, are specified by the Zigbee Alliance [21].

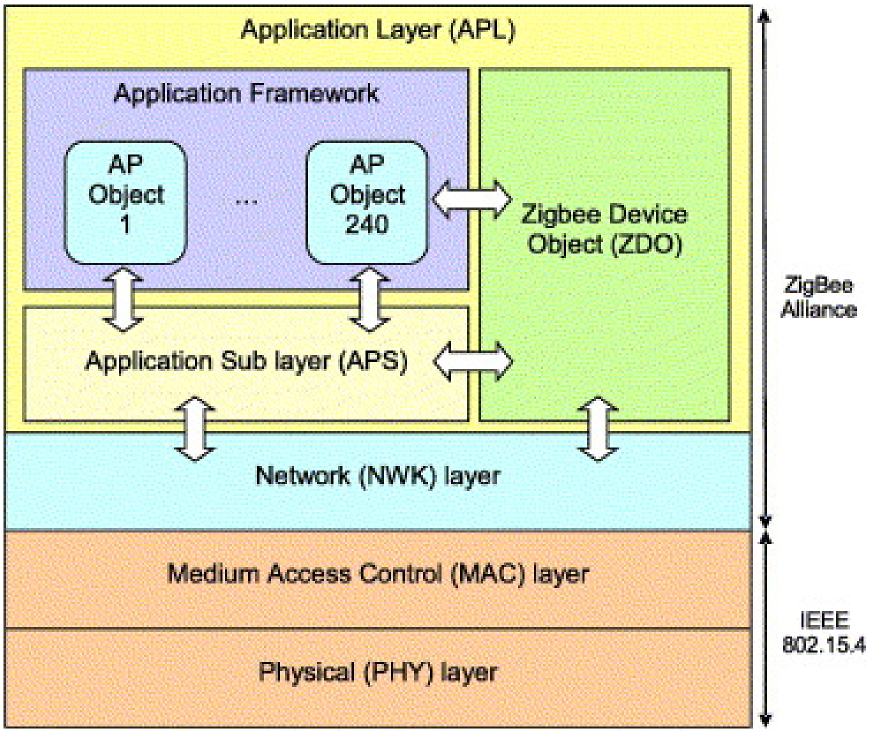


Figure 2.5: Zigbee layer architecture.[2]

Physical layer

The physical layer is in charge of setting on/off the radio, the channel selection, link quality estimation, energy detection measurement and clear channel assessment [2]. It supports three frequency bands: the 2450 MHz band with 16 channels for use worldwide, the 915 MHz band with 10 channels in the USA and the 868 MHz band with 1 channel in Europe, all of them using the Direct Sequence Spread Spectrum (DSSS) access mode.

MAC layer

In the MAC layer two types of nodes are considered:

- Reduced Function Devices (RFD) which can act only as end devices.
- Full Function Devices (FFD) that have a full set of MAC layer functions and can either be coordinators or end devices.

There is also two types of network topologies possible:

- Star topology, where a FFD is considered as the Personal Area Network (PAN) coordinator and other FFDs or RFDs are communicating with it.
- Peer-to-peer topology, where a FFD can talk to other FFDs either directly or using multi-hop, and a PAN coordinator manages the network.

In addition, for both topologies a PAN coordinator may operate with or without a superframe. Using a superframe enables the nodes to communicate in time slots and also allows the coordinator to sleep for a portion of the frame, on the other hand, when not using a superframe the coordinator must always be on and ready to receive data.

The MAC layer also supports channel scan, to locate existing PANs and coordinators, and association/disassociation, to join or leave a PAN [2].

Network layer

The network layer provides a multi-hop network built on top of the IEEE 802.15.4 standard. It considers three types of devices:

- Coordinator: a FFD that organizes the network and maintains the routing tables.
- Router: a FDD with routing capabilities which talks to the coordinator, other routers and end-devices.
- End-device: a RFD or FFD acting as a simple device which can only talk to routers or a coordinator.

In this layer three network topologies are identified: star topology, tree topology and mesh topology, being the mesh the more complex, but most robust and resilient to faults [2]. Furthermore, some of the functionalities supported are multi-hop routing, route discovery and maintenance, security and joining/leaving a network.

When using a mesh topology, routers need to maintain a routing table to forward packets. Basically the behavior they follow when receiving a message is to check if it is intended for them or their children; if not, then consult the routing table for the next hop; if no entry is available, then it starts a route discovery procedure; and if no resources are available to do so, then it uses the tree-based routing. In the tree topology, routers only maintain their addresses and their parent and children addresses, but destination is easily determined due to how addresses are assigned to each node.

The route discovery procedure is based on the Ad hoc On Demand Distance Vector (AODV) routing algorithm. When a node needs a route to a certain destination, it broadcasts a route request message that propagates through the network (also called flooding) until it reaches its destination, while accumulating the cost of the links traversed. This cost can either be a constant value or dynamically calculated based on a link quality estimation [2].

Application layer

The application layer is made up of three parts:

- Application Objects (APO): a portion of software that controls a hardware unit on a device and can interact with other APOs.
- Zigbee Device Object (ZDO): a special object which offers services to the APOs, such as discovery services.
- Application Sublayer: it enables the interaction between APOs and the ZDO by providing data transfer services.

An important part of the application layer are the application profiles, which define message formats and protocols for interactions between APOs that collectively form a distributed application [2].

2.4.1.2 Network specification

In order to implement Zigbee, a network specification needs to be chosen, for wireless sensor networks we will consider Zigbee PRO and Zigbee IP.

Zigbee PRO

This is the most popular choice of Zigbee specification and the most widely adopted for application standards. It offers the feature Green Power, which allows connecting energy-harvesting or self-powered devices into Zigbee PRO networks.

To be able to connect a WSN using Zigbee PRO to Ethernet or Wi-Fi based devices it is necessary to include a gateway in order to translate or map between the different protocols used in the communication.

Another option to achieve this, is to use an end-to-end-based IPv6 architecture. Since it provides 2^{128} unique addresses, it is possible to give a unique IP (Internet Protocol) address to every single device on the Earth. Moreover, memory-efficient implementations of the IP stack show that IP can successfully work in as little as a few kilobytes of RAM (Random Access Memory), and require less than 10 kilobytes of ROM (Read Only Memory) [22]. In [23] a system architecture is proposed where Zigbee is used as the communication medium and IPv6 in the network layer, whereas the communication between the gateway server, the middle-ware and the mobile client is based on IPv4 over Wi-Fi. Thus, enabling any device communicate with each other regarding the communication medium or network protocol used.

Zigbee IP

It offers an architecture with end-to-end IPv6 networking. Moreover, it uses 6LoWPAN (IPv6 over Low power Wireless Personal Area Networks) header compression to reduce the 48 bytes of IPv6 down to 6 bytes for the common case. It also uses PANA (Protocol for Carrying Authentication for Network Access) for authentication, RPL (Ripple routing protocol) for routing, TLS (Transport Layer Security) and EAP-TLS (Extensible Authentication Protocol-TLS) for security and the User Datagram Protocol (UDP) and Transmission Control Protocol (TCP). The downside being that it only supports the Zigbee 2030.5 (Smart Energy 2) application standard.

One last thing to consider is that Zigbee PRO and Zigbee IP devices can only communicate between each other through a special gateway equipped with both specifications.

2.4.2 Thread

This is a robust self-healing mesh network capable of supporting over 250 devices on a single network. It is IP-based, using IPv6 and 6LoWPAN compression, allowing devices to communicate with the cloud. It also provides security at the network and application layers. And only a software enhancement is needed to implement it on IEEE 802.15.4 compliant products.

2.5 Tools

The main objective of this work is to develop a software solution to operate a sensor node. To facilitate and expedite this process some tools were used such as an Integrated Development Environment (IDE) and an energy profiler.

2.5.1 Simplicity Studio

This is a software suite by Silicon Labs which provides an IDE and several other tools to ease the complexity of developing an application [Fig. 2.6]. Among the main tools are the energy profiler, explained in detail in the next subsection; a network analyzer, to scan and capture radio packets using a ZigBee protocol; a Serial Wire Output (SWO) terminal, to observe in real-time the output of the serial trace from the development board; and application examples demonstrating the use of peripherals and other main features.

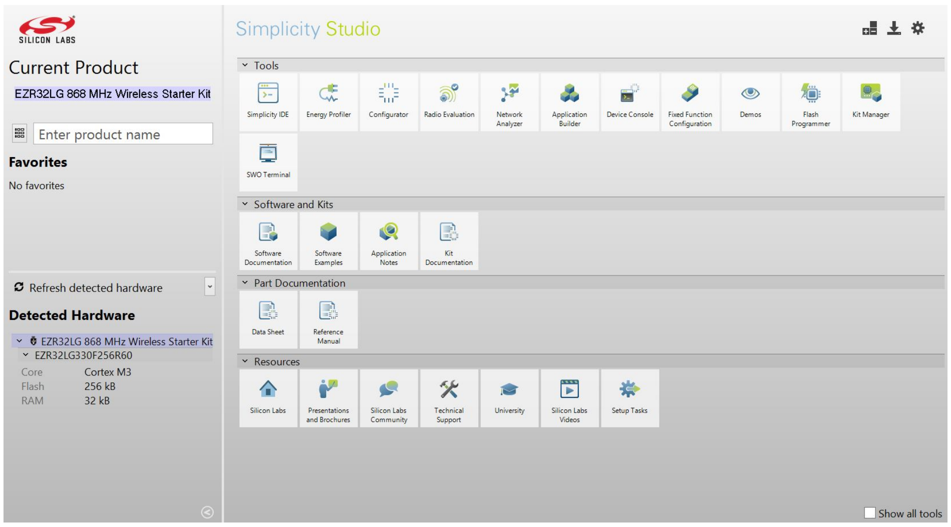


Figure 2.6: Simplicity Studio.

2.5.2 Energy Aware Profiler

By using this tool, the energy consumption of the application running in the development board can be examined. The current is plotted in real-time and several counters sum the average current, the time elapsed and the energy spent, either in a selected range of time or from the total time of operation. In addition, it allows to select a specific point in the graph and see to which section in the code it relates to, this permits the user to find out where the application is spending most of its energy and optimize it.

In order to measure the current, the Advanced Energy Monitor (AEM) consisting of a current sense amplifier, multiple gain stages and signal processing is used. The AEM can measure current signals in the range of $0.1\mu A$ to $150mA$ [3].

2.6 Related work

Wireless sensor networks have been in the market for over the past decade and there are many commercial applications that have been implemented using different platforms such as IRIS, Mica2, MicaZ, TelosB, Tmote Sky and EPIC [24]. These WSN platforms are either based on the MSP430 or ATmega128 microcontrollers in combination with a Zigbee network architecture. However in [24] a different platform using the EFM32G230 microcontroller together with Bluetooth Low Energy (BLE) for network communication is presented.

By using these different platforms combined with other hardware, several implementations of sensor nodes have been developed such as Prometheus, HydroWatch, Heliomote, Ambimax, Everlast and Sunflower [18]. For the energy source all these sensor nodes harvest solar energy either by using a solar panel or photo diodes. However for the energy storage, some use NiMH or Lithium batteries, while some others use supercapacitors and even some of them use a combination of both. When using batteries and supercapacitors an additional charging circuit is needed to manage the energy. In general, most of these applications have similar characteristics essentially differing in the energy requirements.

Chapter 3

Design and Implementation

In this chapter we present the hardware components chosen for the sensor node: processor, radio, sensor and power unit. We also examine some of their main characteristics which make them suitable for this application. Additionally, we discuss what kind of energy storage is needed to support the operation of the node. Finally, we explain the software solution used in the sensor node.

3.1 Hardware architecture

The hardware architecture of the sensor node is composed of four main components as seen in Figure 3.1. The processing, radio and sensing units are integrated into the Silicon Labs' EZR32 Leopard Gecko Development Kit. This kit provides two development boards [Figure 3.2] with different sensors and peripherals in conjunction with a low energy microcontroller and a radio. Additionally, it features the AEM for real-time current and voltage monitoring, which is useful to determine where the application is spending more energy and hence being able to optimize it. While the power unit is an external energy harvester with a small energy buffer which is connected to the development board.

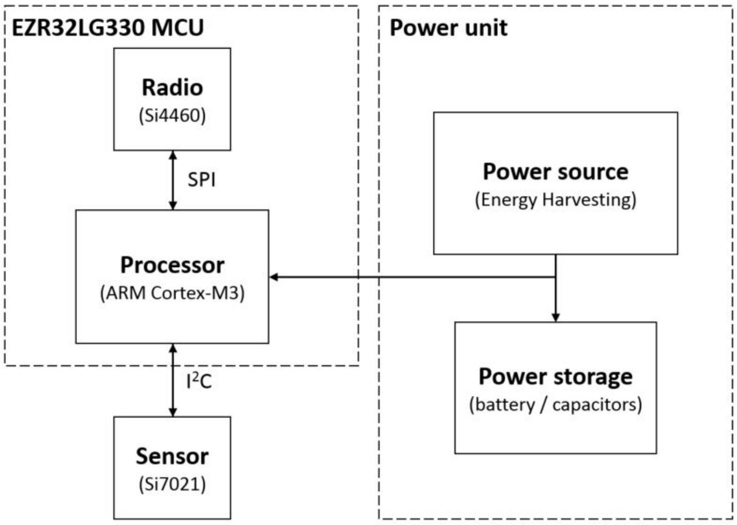


Figure 3.1: Sensor node.

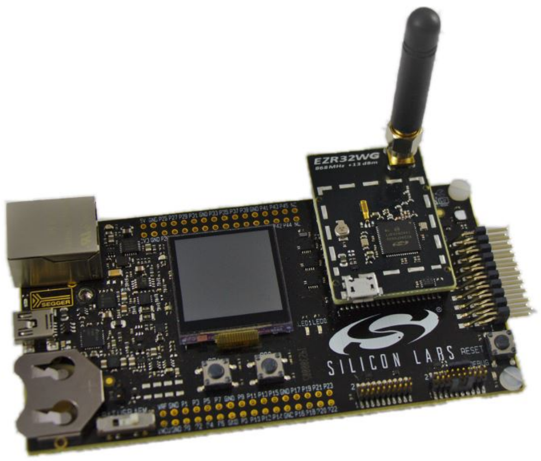


Figure 3.2: EZR32LG development board.[3]

3.1.1 Processing unit

The main unit of the sensor node is the ultra-low power EZR32LG330 wireless micro-controller which consists on a 32-bit ARM Cortex-M3 processor clocked at 48 MHz, with 256 KB of Flash memory and 32 KB of RAM, this makes it suitable to run complex algorithms and a radio protocol stack. It also includes many different peripherals such as a Real-Time Counter (RTC), a low energy timer, different communication interfaces, amongst others. Moreover, each peripheral is individually clocked, allowing the user to disable those which are not being used and thus reducing power consumption.

A main feature of this MCU (Microcontroller Unit) is the different energy modes it offers, these are presented in Table 3.1. The EM0 mode is the normal execution mode where all the peripherals can be enabled and the code is running from Flash. In the EM1 the CPU (Central Processing Unit) is sleeping, but every peripheral can still be active. The EM2 mode reduces energy by turning off the high-frequency oscillator. In the EM3 mode, further energy is reduced by turning the low-frequency oscillator off, but most of the peripherals can still be used with the ultra-low frequency oscillator. Finally, in the EM4 mode almost all functionality is disabled, except for the pin reset, GPIO (General Purpose Input/Output) pin wake-up, GPIO pin retention, backup RTC with RAM retention and the Power-On Reset.

Table 3.1: MCU energy modes.

Energy Mode	**Current consumption**
EM0 - Run Mode	211 μA/MHz
EM1 - Sleep Mode	63 μA/MHz
EM2 - Deep Sleep Mode	0.95 μA
EM3 - Stop Mode	0.65 μA
EM4 - Shutoff Mode	20 nA

Energy modes EM1-EM4 are accessed via software calls in the program code, however to return to the EM0 mode there are different wake-up triggers which can be used in the EM1-EM3 modes, while for the EM4 a reset, GPIO wakeup request or backup RTC interrupt are the only options, Figure 3.3 shows the state transitions. All the functionality of energy modes is managed by the Energy Management Unit.

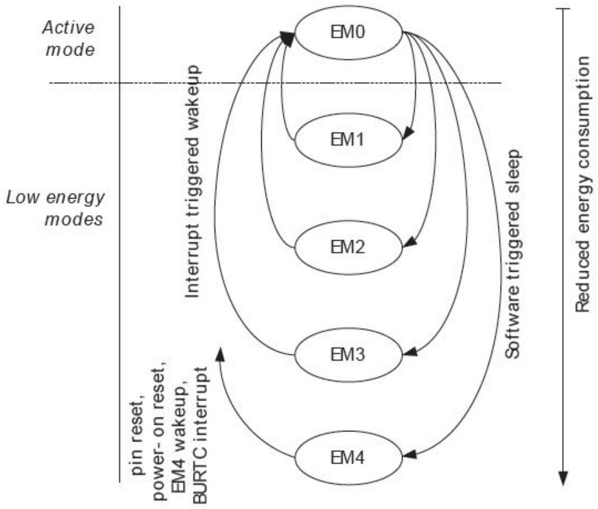

Active mode
Low energy modes
EM0
EM1
EM2
EM3
EM4
Interrupt triggered wakeup
Software triggered sleep
Reduced energy consumption
pin reset, power- on reset, EM4 wakeup, BURTC interrupt

Figure 3.3: MCU energy mode transitions.[4]

3.1.2 Radio unit

The radio unit consists of the low-current Si4460 RF (Radio Frequency) transceiver covering the sub-GHz frequency bands from 119 to 1050 MHz which is included in the physical package of the EZR32LG. It operates as a time division duplexing transceiver where the device alternately transmits and receives data packets. Some of the main characteristics presented in [11] which makes this radio suitable for battery operated applications are shown in Table 3.2.

Table 3.2: Radio characteristics.

Operation frequency	868 MHz
Transmit power	13 dBm
Data rates	100 bps - 1 Mbps
Current consumption	18 mA TX @ +10 dBm 10/13 mA RX
Power amplifier	Up to +20 dBm

The radio offers different operating modes to save energy, Figure 3.4 shows the transitions between them and the current consumption in each one. Each mode disables different components to reduce energy consumption and therefore have different response times when switching to TX (Transmission) or RX (Reception) modes. Moreover, these modes can be accessed by using the radio API commands. The Application Programming Interface (API) is embedded into the device and provides commands to control the chip and retrieve its status and also properties which are general configurations that do not change very frequently, both are communicated from the processor to the radio via a SPI (Serial Peripheral Interface) bus. A description of the API (Application Programming Interface) can be found in [25].

Another feature is the highly configurable packet structure. Some usual communication fields such as preamble, synchronization word, header, packet length and CRC (Cyclic Redundancy Check) can be automatically added by the packet handler, greatly reducing the computational power required by the host MCU to construct or deconstruct a packet [26]. Additionally, the user can configure up to five different fields which can be used to create any data pattern and add different properties to each field such as data whitening and Manchester encoding. Moreover, there are two 64-byte FIFOs (First-In First-Out) integrated in the chip to store packets from TX and RX, but they are only enabled when the radio is in FIFO mode.

Let us consider a scenario where the radio switches from the RX mode to the TX mode to transmit a single packet and afterwards it changes back to the RX mode for $500ms$. We can use equation 3.1 [9] to calculate the radio power consumption P_c, as follows:

$$P_c = N_T[P_T(T_{on} + T_{st}) + P_{out}(T_{on})] + N_R[P_R(R_{on} + R_{st})] \tag{3.1}$$

where $P_{T/R}$ is the power consumed by the transmitter/receiver, P_{out} is the output power of the transmitter, T/R_{on} is the transmitter/receiver on-time, T/R_{st} is the transmitter/receiver start-up time and $N_{T/R}$ is the number of times transmitter/receiver is being switched on per unit time.

Together with the following parameters: N_T=1, N_R=1, $P_T = 5.94mW$, $P_R = 42.9mW$, $T_{on} = 8.57ms$, $R_{on} = 491.43ms$, $T_{st} = 130\mu s$, $R_{st} = 0s$ and $P_{out} = 59.4mW$, we can determine that a total of $21.64mJ$ is consumed by the radio.

Now, using the energy profiler we can observe in Figure 3.5 how the actual measurement compares to this calculation. For this test a single packet of 23 bytes is transmitted, and we can observe that the energy consumption is $22.79mJ$ which is slightly above the expected value probably due to the size of the packet which is not considered in the P_{out} parameter.

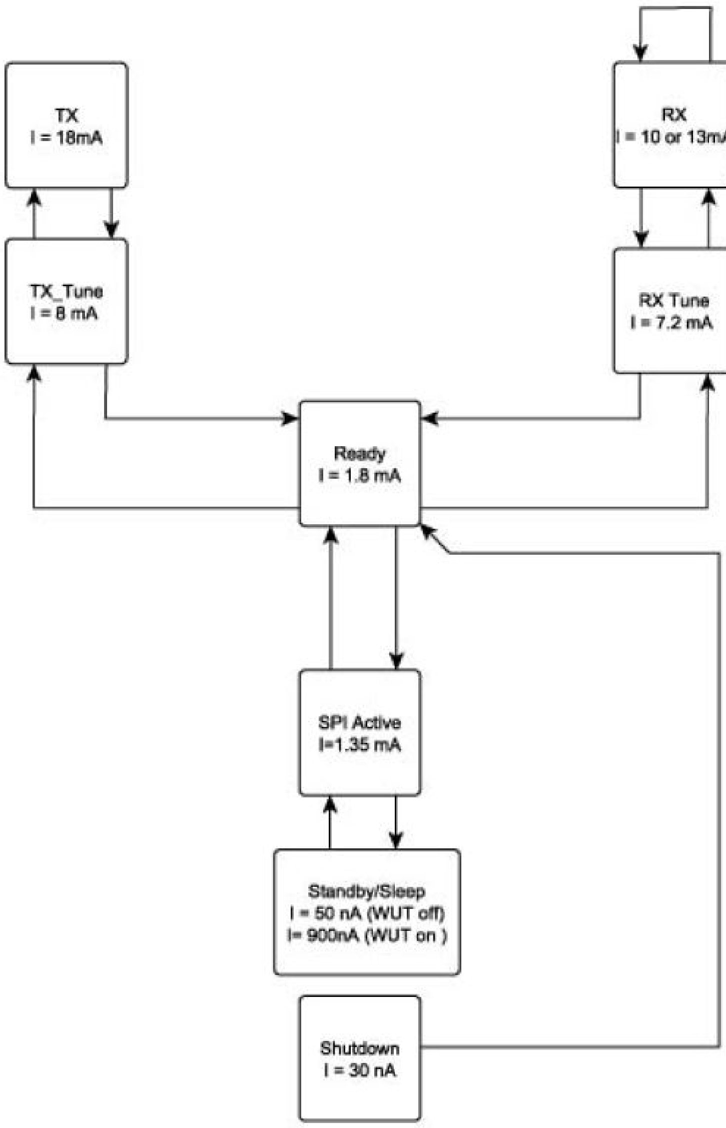
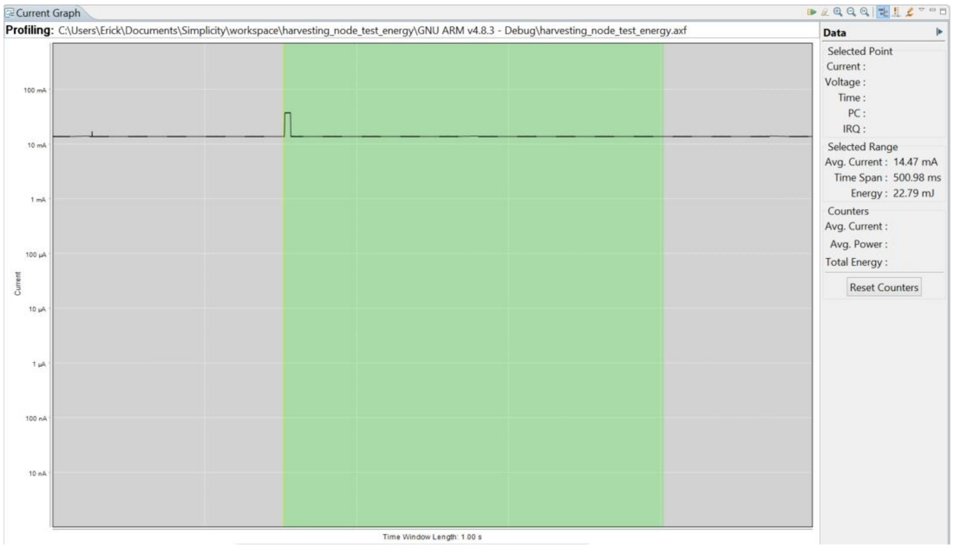


Figure 3.4: Radio operating modes.[5]



Profiling: C:\Users\Erick\Documents\Simplicity\workspace\harvesting_node_test_energy\GNU ARM v4.8.3 - Debug\harvesting_node_test_energy.axf

Data

Selected Point
Current :
Voltage :
Time :
PC :
IRQ :

Selected Range
Avg. Current : 14.47 mA
Time Span : 500.98 ms
Energy : 22.79 mJ

Counters
Avg. Current :
Avg. Power :
Total Energy :

Figure 3.5: Radio power consumption.

3.1.3 Sensing unit

For the sensing unit we use the Si7021 sensor. This is a monolithic CMOS (Complementary Metal-Oxide Semiconductor) Integrated Circuit (IC) included in the EZR32LG, which integrates humidity and temperature sensor elements, an analog-to-digital converter, signal processing, calibration data, and an I2C (Inter-Integrated Circuit) Interface [3]. This facilitates the measurements by handling all the processing in the chip and only communicating the digital readings to the host MCU as seen in Figure 3.6. Additionally, these sensors make environmental applications such as field monitoring suitable for this sensor node.

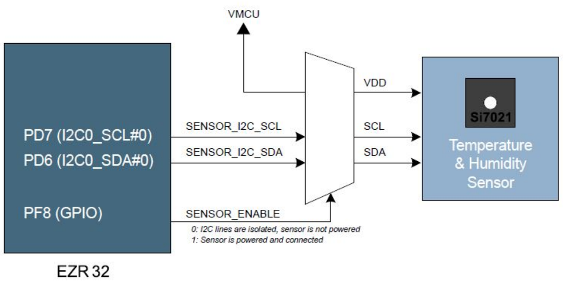
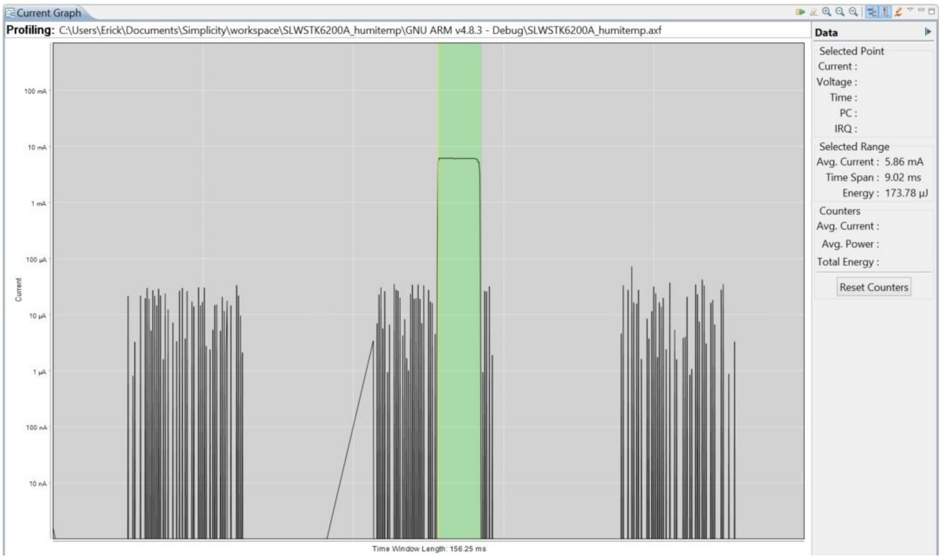

VMCU
PD7 (I2C0_SCL#0)
PD6 (I2C0_SDA#0)
PF8 (GPIO)
EZR 32
SENSOR_I2C_SCL
SENSOR_I2C_SDA
SENSOR_ENABLE
0: I2C lines are isolated, sensor is not powered
1: Sensor is powered and connected
VDD
SCL
SDA
Si7021
Temperature & Humidity Sensor

Figure 3.6: Si7021 connection.[3]

The power consumption of this device is extremely low compared to the radio. We can observe in Figure 3.7 that the measurements are completed in less than $10ms$ and the energy consumed is no more than $174\mu J$, this is why the efforts to reduce energy consumption are focused on delimiting the time the radio is on.


Profiling: C:\Users\Erick\Documents\Simplicity\workspace\SLWSTK6200A_humitemp\GNU ARM v4.8.3 - Debug\SLWSTK6200A_humitemp.axf

Data

Selected Point
Current :
Voltage :
Time :
PC :
IRQ :

Selected Range
Avg. Current : 5.86 mA
Time Span : 9.02 ms
Energy : 173.78 µJ

Counters
Avg. Current :
Avg. Power :
Total Energy :

Figure 3.7: Sensor power consumption.

3.1.4 Power unit

The power unit in this case is composed of a harvesting unit and an energy buffer. As mentioned in the previous chapter, there are different energy sources which can be harvested and for this work the solar energy was elected. This is mainly due to its predictability in order to generate an energy model and the accessibility which requires only that the nodes are placed out in the open, supporting many different applications.

As the solution to harvest solar energy, the "Energy Harvesting Solution to Go Kit" by Würth Elektronik which can be seen in Figure 3.8 was chosen. One of the main reasons is because it provides a connection to power the Silicon Labs' EFM32 Giant Gecko Starter Kit through its expansion header and thus facilitates the development of applications. However given the similarities of EFM (Energy Friendly Microcontroller) starter kits' expansion headers, the compatibility is not limited to the Giant Gecko kit.

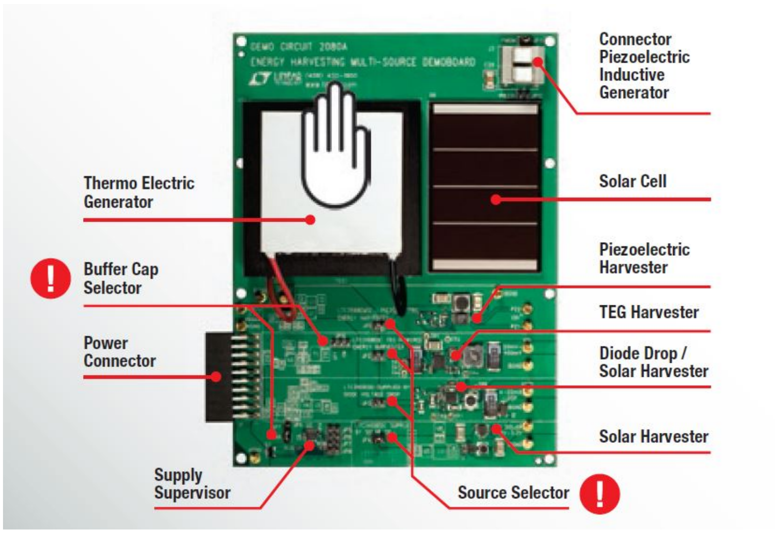


Figure 3.8: Energy Harvesting kit.[6]

The board offers four different harvesting sources namely solar, thermal, piezoelectric and inductive, however only one can be selected at a given time to output the power through the V_{MCU} pin. In addition, it provides a logic signal through the P_{GOOD} pin which goes high the first time the converter reaches the sleep threshold of the programmed V_{OUT}, signaling that the output is in regulation. The P_{GOOD} pin will remain high until V_{OUT} falls to 92% of the desired regulation voltage. Additionally, if P_{GOOD} is high and V_{IN} falls below the $UVLO$ falling threshold, P_{GOOD} will remain high until V_{OUT} falls to 92% of the desired regulation point. This allows output energy to be used even if the input is lost [27].

According to [27] the solar cell input voltage can vary from $1.72V$ to $3.3V$, as for the output it has a switch that enables the V_{MCU} output whenever the voltage reaches $3.15V$ and turns it off when it decreases under $2.25V$, thus allowing a fast voltage rise at start-up and using most of the energy stored in the capacitors.

Furthermore, this kit provides an array of 15 buffer capacitors of $100\mu F$ in parallel which can store up to $8.1675mJ$ at a voltage of $3.3V$. More information about the internal components can be found in the board schematic in the appendix A1.

The harvesting board can be easily customized by using different jumpers and for this case the following configuration was used:

JP1 OPEN	JP7 OPEN
JP2 OPEN	JP8 INSTALLED
JP3 OPEN	JP9 INSTALLED "ON" POSITION
JP4 INSTALLED	JP10 OPEN
JP5 OPEN	JP11 INSTALLED
JP6 OPEN	JP12 OPEN

Where JP4 selects the solar cell energy supply, JP8 routes the LTC3459 P_{GOOD} signal to the P_{GOOD} output, JP9 connects the 15 buffer capacitors to V_{OUT} to store energy and JP11, which is not relevant for this application, configures the AC (Alternating Current) input for use with a $PMDM$ vibration harvester.

3.2 Energy Storage

An important step to implement this solution is to find the appropriate battery type and size to support energy neutral operation in the node. In chapter 2 different battery types were presented along with their advantages and disadvantages, for this solution we chose the NiMH due to its viability of being trickle charged and what is more, it does not require a high charging current as the Li-ion batteries.

This type of battery has a typical charging efficiency of 66% [28], meaning that a fair amount of the energy harvested is lost if it is first stored in the battery and used afterwards. Furthermore, we have to consider the self discharge rate, which for NiMH batteries they will retain 50% to 80% of their capacity after 6 months of storage [29], however since this parameter is greatly dependant on the temperature and recharge cycles, we do not contemplate it in our simulations.

Let us consider a AA sized NiMH battery with a voltage of $1.2V$ and a capacity of $2300mAh$ (at 21°C) [7] which is equivalent to $9{,}936J$. We can observe in Figure 3.9 that it takes approximately 10 hours for the battery to discharge with a constant current of $230mA$, and since our application for the sensor node consumes substantially less current, roughly 14 mA without any duty cycle, we can employ this battery for our sensor node. Additionally, since the sensor node operates on $3.3V$, a step-up DC/DC (Direct Current) converter such as the LTC3105, which is also used in the energy harvesting kit [27], has to be employed to provide the necessary voltage output.

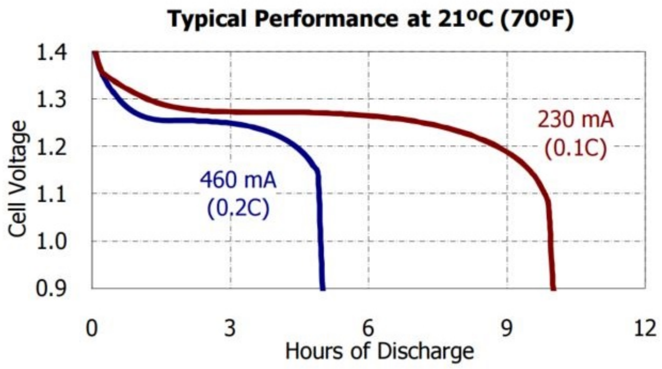


Figure 3.9: AA battery voltage discharge curve.[7]

3.3 Software architecture

The software solution implemented in the C programming language features an energy harvesting sensor network using a duty cycling technique based on adaptive control theory and a communication protocol that synchronizes the transmission times of the nodes with the active times of the neighboring nodes. Furthermore, it is divided into different layers as seen in Figure 3.10.

The solution is based on the following assumptions:

- The sensor nodes contain a rechargeable battery with sufficient capacity such that they can achieve energy neutral operation by adapting their duty cycles.
- The ZigBee routing protocol is considered to be already implemented.

3.3.1 Main Application

Typical techniques for duty cycling such as the ones previously presented in chapter 2 are not completely adaptable to energy harvesting systems, however some of their features can be exploited in order to create a suitable technique for these systems.

The main problem that arises with the duty cycling techniques is the synchronization of the nodes. Some techniques such as TDMA require a tight synchronization be-

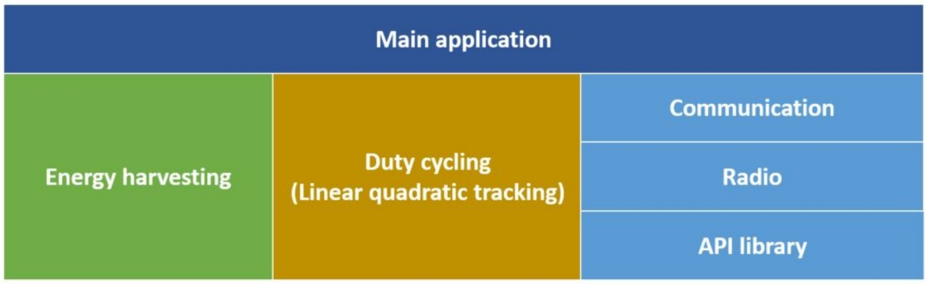


Figure 3.10: Architecture layers.

tween nodes which can be very complex in large networks. This complexity can be eliminated by letting the nodes set up their own wake-up times in a decentralized fashion and go to sleep independently from each other, however it raises concerns about an increase of the latency and the variance of the latency in the network [30]. Nevertheless, for spatial sensing applications such as this, the latency and throughput of the network are not as concerning as the energy consumption.

The solution adopted in this work requires a small synchronization between the nodes, but can also be considered as a decentralized network because the nodes set up their own sleep/wakeup schedules based on their energy parameters. The basic functionality is based on the Fully Synchronized Pattern technique, where the nodes wake up every T_{wakeup}, in this case called DC (Duty Cycle) period, and start transmitting right after that. However in our case, the T_{active} time is not fixed and varies depending on the current duty cycle.

This basic approach has a high probability of collisions, since every node is transmitting at the same time, a possible improvement to reduce collisions is to set an offset on the T_{wakeup} times depending on the level of hierarchy of the nodes in the routing tree, such as in the Staggered Wakeup Pattern technique. However routing information is not available in this case and instead a random delay is used before each transmission, which will be explained in further detail in the next subsections.

An issue related with clock synchronization is the clock drift, according to [13] a typical clock drift does not exceed 0.2ms per second, which is equivalent to 200ppm (parts per million). For instance, the crystal oscillator Si510 has a frequency stability grade A of ±100ppm [31]. But taking into account that the listening periods for the nodes are in the order of seconds, this is not a problem. However, there is still the concern of long-term clock drift, for every 30 minutes the clock could be out of sync by $360ms$, therefore the nodes are synchronized every SYNC period. Some of the initial considerations are presented in Table 3.3.

Table 3.3: Application parameters.

SYNC period	30 minutes
DC period	1 minute
Minimum DC	4%
Minimum DC for TX	5%
Maximum DC	99%
Maximum clock drift	200ppm

In the application layer the node first initializes all the necessary hardware and peripherals. After this is done, the node enters a synchronization state, where it waits for a SYNC packet containing the local time of another node in the network. Upon receiving this packet, the node matches its own clock with it considering the computation delays. However, if the node has not received a SYNC packet after 30 minutes, which is the SYNC period, then it uses a default time to start its own clock. Following the initialization, the node enters into an infinite loop where it follows the state transitions in Figure 3.11. First it wakes up and enters the EM0 mode, setting the proper clock frequency. Then, it obtains the battery level from the harvesting layer and verifies if it has enough energy to operate, otherwise it goes back to sleep.

Subsequently, the node updates its duty cycle using an adaptive algorithm and turns the radio on. Following that, if the SYNC period has expired, it enters the SYNC slot where it listens for a SYNC packet from a parent node, adjusts its clock if needed and retransmits the SYNC packet to its children nodes.

After that, the sensor readings are obtained and the node enters the DATA slot, where it transmits these readings to other nodes and listens for NODE packets. In addition to the sensor readings, these packets also contain the duty cycle of the node, in a similar fashion as SMAC does by periodically broadcasting the listen/sleep schedules to immediate neighbors [13]. At the end of the DATA slot, the node turns off the radio and enters EM2 mode to reduce energy consumption, waiting until the next DC period by using the RTC. The SYNC and DATA slots are further explained in the communication layer.

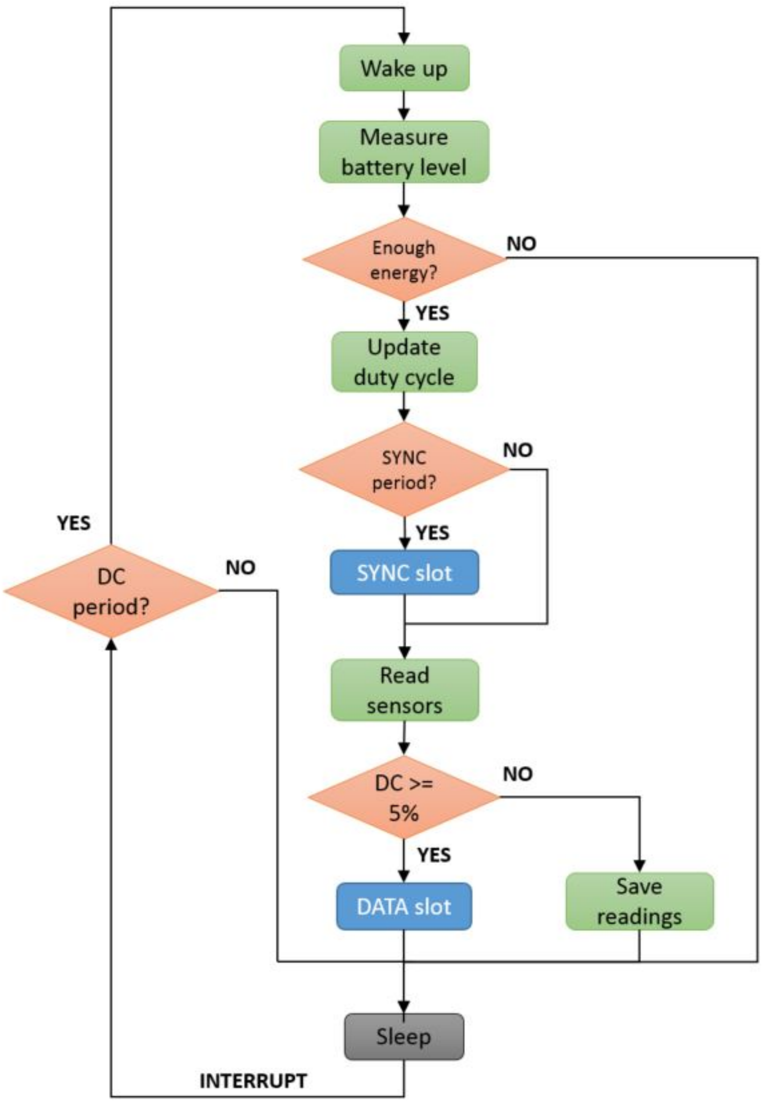


Figure 3.11: Application flowchart.

When the duty cycle is below 5%, to avoid a possible shutdown due to miscalculations on the remaining power, the node adds up the sensor readings instead of transmitting them, this way when the node is back on a higher duty cycle it will be able to transmit an average of these past readings. The source code for the main function can be seen in Appendix B4.

3.3.2 Energy Harvesting

This layer takes care of measuring the voltage of the battery. This is done by using the ADC (Analog-to-Digital Converter) to sample $V_{DD}/3$ using an internal reference of $1.25V$. Furthermore, a 12-bit resolution is used, meaning that the voltage is given by:

$$V_{DD} = \text{sample} * \frac{3.75V}{2047} \tag{3.2}$$

This value is then converted to a percentage level between 0 and 1 using the minimum and maximum voltage values from the linear region of the battery's discharging curve.

3.3.2.1 Testing battery level

Considering that our harvesting unit does not have a battery of enough capacity to support the operation of the application, the battery's level has to be simulated instead using the energy available. In the previous section, we determined the capacity of a AA sized NiMH battery to be $9{,}936J$, however considering that not all of the energy can be used since the voltage will drop below an operational level due to the discharge curve, we consider that 80% of the total energy can be used instead, resulting in a total of $7{,}948.8J$ for the effective energy of the battery.

Subsequently, we need to determine the contribution of the harvester source to the battery level. The solar panel on the energy harvester has a maximum current of $50mA$ with an output of $3.3V$, providing a maximum power of $165mW$. However, due to the absence of a measuring circuit in the sensor node, we use instead the data output presented in [32] from a solar cell with similar characteristics to our solar panel, providing $3.3V$ and $60mA$ as maximum outputs. Based on this data, we plot the harvested power during one day with intervals of 10 minutes in Figure 3.12. The energy harvested is assumed to go through the battery first and then to the sensor node, thus we consider the charging efficiency when calculating this contribution.

Next, we need to calculate the decrease of the battery level due to the node's power consumption. This is done by using the average power during the last DC period with

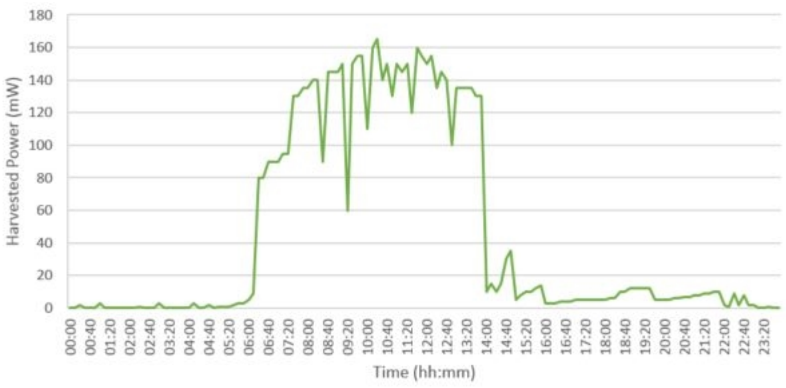


Figure 3.12: Solar harvested power.

the following equation:

$$P_c = \alpha(P_{act}) + (1-\alpha)(P_{slp}) \tag{3.3}$$

where α is the duty cycle, P_{act} is the power consumed during the active state and P_{slp} is the power consumed during the sleeping time.

Finally, we obtain the energy spent or accumulated during the last DC period and add it to the total energy of the battery. Afterwards, a percentage level of the battery is computed. The implementation of this function can be examined in Appendix B3.

3.3.3 Duty Cycling

To calculate the duty cycle two different techniques previously discussed were considered. The predicted model algorithm, first, initializes its parameters at the beginning of the day and afterwards, it updates the model and duty cycle at the beginning of each time slot by using the real energy harvested and modifying its predictions. The linear quadratic (LQ) tracking algorithm on the other hand, is executed every minute and adapts its duty cycle based on the current battery level.

An experimental comparison of both algorithms was made in [8] with three solar and one wind data sets which concluded that the LQ tracking algorithm adapts better to variable weather conditions and avoids dead time. Table 3.4, adapted from [8], presents the results of the experiments using the same solar data set used in [16].

Table 3.4: Performance comparison of duty cycling algorithms

Algorithm	**Predicted model [16]**	**LQ tracking [8]**
Mean Duty Cycle	31.44	33.40
Time Dead (%)	0.55	0.0
Time Full (%)	2.33	0.0

3.3.3.1 Linear Quadratic Tracking implementation

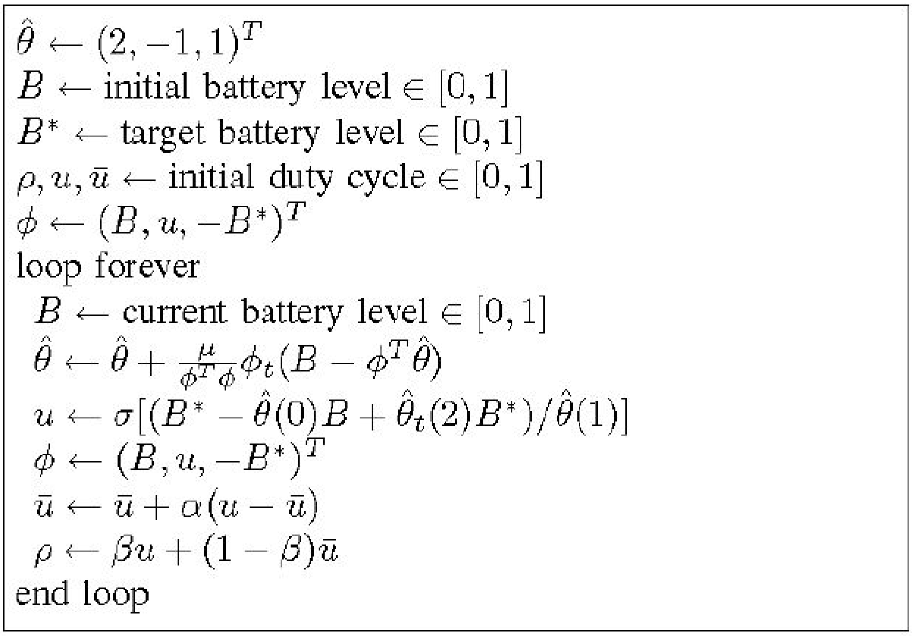
The advantage of this approach is that it makes no assumptions about the nature of the energy source, nor does it require any data or model of the source. It is based on a technique from adaptive control theory where the problem is formulated as a LQ tracking problem and a simple control law is provided to achieve energy neutral operation while maximizing task performance [8]. Furthermore, it implements a tunable mechanism for minimizing the variance of the node's duty cycling profile.

We based our implementation on the algorithm pseudocode in Figure 3.13. First, it initializes the different variables and then using the battery level and equations 2.6, 2.7, 2.8 and 2.9 it calculates the node's duty cycle.

Furthermore, it can be seen in Figure 3.13 that a rectifier function $\sigma(u)$ is introduced to delimit the output value of the duty cycle, and it is defined as follows:

$$\sigma(u) = \begin{cases} 0 & \text{if } u < \rho_{min} \\ 1 & \text{if } u > \rho_{max} \\ u & \text{otherwise} \end{cases} \tag{3.4}$$

Since the parameter estimation algorithm converges much faster if $\hat{\theta}$ is initialized to a reasonable value rather than arbitrarily according to [8], an initial value of $(2, -1, 1)^T$ is proposed. Additionally, a target battery value of 65% was selected to provide a bias towards energy wasting instead of energy deficit. Moreover, an initial battery voltage of $3.3V$ equivalent to a level of 92.85% and duty cycle of 10% were selected in case the node starts operating during the night. The rest of the parameters are as follows: $\alpha = 0.0005$, $\beta = 0.5$ and $\mu = 0.001$. The C code implementation of the algorithm can be seen in Appendix B2.



```
$\hat{\theta} \leftarrow (2, -1, 1)^T$
$B \leftarrow$ initial battery level $\in [0, 1]$
$B^* \leftarrow$ target battery level $\in [0, 1]$
$\rho, u, \bar{u} \leftarrow$ initial duty cycle $\in [0, 1]$
$\phi \leftarrow (B, u, -B^*)^T$
loop forever
  $B \leftarrow$ current battery level $\in [0, 1]$
  $\hat{\theta} \leftarrow \hat{\theta} + \frac{\mu}{\phi^T \phi} \phi_t (B - \phi^T \hat{\theta})$
  $u \leftarrow \sigma[(B^* - \hat{\theta}(0)B + \hat{\theta}_t(2)B^*)/\hat{\theta}(1)]$
  $\phi \leftarrow (B, u, -B^*)^T$
  $\bar{u} \leftarrow \bar{u} + \alpha(u - \bar{u})$
  $\rho \leftarrow \beta u + (1 - \beta)\bar{u}$
end loop
```

Figure 3.13: Algorithm pseudocode.[8]

3.3.4 Communication

The communication between the nodes is performed during the SYNC and DATA slots as mentioned before. These slots together with the sleeping time comprise the DC period as seen in Figure 3.14.

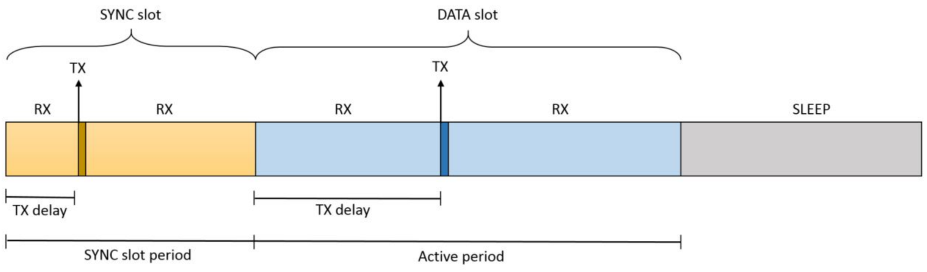


Figure 3.14: DC period.

The SYNC slot is used to transmit a packet with the local clock for synchronization purposes and it is only available after each SYNC period, which should be a sufficient time to see a clock mismatch but not enough to cause problems in the application. The SYNC slot period is fixed at 1.5 seconds, this value has to be less than the active time of the minimum duty cycle allowed in the node which is 4%, equivalent to an active time of 2.4 seconds. Considering that the clock is synchronized during this slot, the packet transmission has to follow a defined propagation order to avoid conflicts. This can be easily achieved by using routing information from the network and transmitting the clock from the root node to the leaf nodes in a tree-like topology. However, since our implementation does not provide routing information, two cases are considered for testing purposes. On the one hand, if the node is the root node, it waits for a delay equal to the maximum clock drift before transmitting the packet. On the other hand, if it is an intermediate node, it listens for a packet coming from a parent node and retransmits to its children nodes. After a node has forwarded a SYNC packet, it uses the clock data to synchronize its own clock considering the transmission and computation delays.

The DATA slot is used to communicate the sensor readings and the node's duty cycle. This slot is reliant on the node's duty cycle and is accessible only when it is 5% or higher, this is in order to provide enough time and energy for transmission. When the duty cycle is 5% the active period will be of 3 seconds, however if there is a SYNC slot before, then the active period will consist of 1.5 seconds. The transmission of the packet is delayed by a random time to avoid collisions in the network, which is common in densely deployed sensor networks. The minimum delay is equal to the maximum clock drift, while the maximum delay depends on the shorter active time between the sender and the receiver, this in order to find a common window where both nodes are active. TX_{max_delay} is defined as:

$$\text{max delay} = \min\{S_t, R_t\} - \text{SYNC}_T - \text{D}_{clk}$$

where S_t is the active time of the sender node, R_t is the active time of the receiver node, SYNC_T is the SYNC period and D_{clk} is the maximum clock drift.

A flowchart describing the operation during this slot can be observed in Figure 3.15. Furthermore, the source code for the initial clock synchronization and the SYNC and DATA slots can be reviewed in Appendix B1.

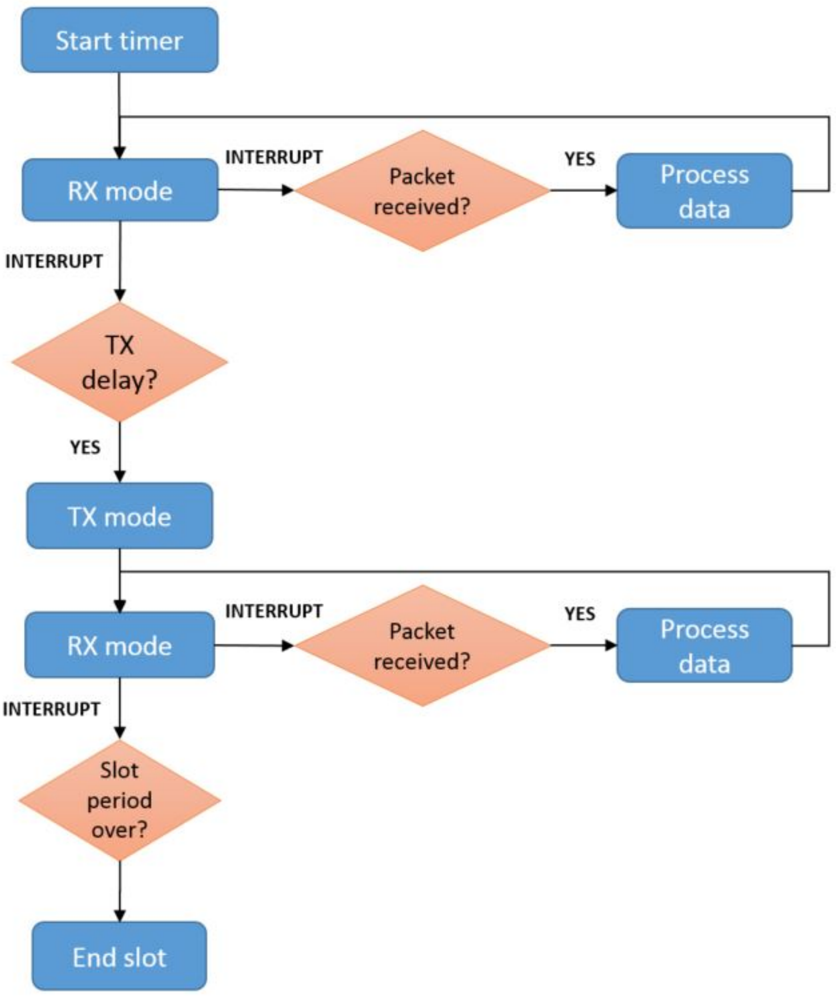


Figure 3.15: DATA slot flowchart.

The random time used to delay the transmission of packets is obtained by using the $rand()$ function and the node's ID is used as the seed to feed this function. However, this method does not provide an uniform distribution when bounded by two values, hence if required, a more complex function could be used instead of $rand()$ to obtain a better distribution. One further improvement to avoid collisions could be to implement a virtual carrier sense as the one used in SMAC [13].

3.3.4.1 Radio

This layer uses the radio configuration stored in the *radio-config-wds-gen.h* file to initialize the radio parameters and registers such as: base RF frequency, power amplifier, modulation type, data rate, packet configuration, packet match, among others. Refer to the appendix A2 to see the full configuration in detail.

Two types of packets are needed for the network communication, NODE DATA and SYNC DATA. The first one is used to transmit sensor readings and the duty cycle and the second one, to share the local clock time. Their structure can be seen in Figure 3.16.

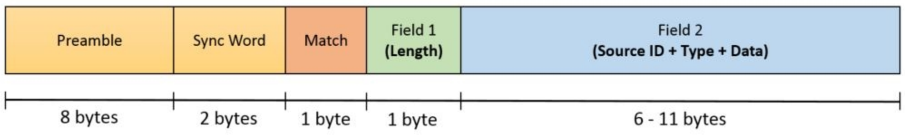


Figure 3.16: Packet structure.

The first 10 bytes of both packets are used for the preamble and synchronization word, these values are predefined in the configuration file and added automatically to the packet via the packet handler. The next byte is used for the packet match, the radio reads this byte and uses a mask to compare it to a predefined value, if they do not match then it drops the packet and starts listening again. This helps to filter packets which are not intended for the node, by using 1 byte and 1 match point we can define up to 256 different node addresses. The following byte is used to store the length of the variable field, which contains the actual data. Finally, the variable field consists of 1 byte for the sender's ID, 1 byte for the packet type and the rest of the bytes for the data.

With support of the API library commands [25], this layer offers functions to initialize, shut down and restart the radio, also to start the transmission and reception of packets, and check if a packet was received or transmitted successfully or if a CRC error occurred. Since the packet handler only takes care of the preamble and synchronization word fields, this layer ensures that the rest of the packet fields are encoded and decoded correctly. The implementation of the transmit and receive functions is found in Appendix B5.

Chapter 4

Results and Analysis

In this chapter the results of the simulations and tests are presented. First, the implications the data rate has on the transmission range and delay are evaluated. Then, energy measurements of the application using different duty cycles and packet sizes are analyzed. Next, the output of the duty cycling algorithm is examined. And finally, the network communication between two nodes and their synchronization is verified.

4.1 Data rate

Most of the radio configuration parameters [Appendix A2] have been chosen with the default values provided by the configurator tool since they are not concerning to this work. However, the data rate has several implications affecting the transmission range and transmission delay which are dependent on the specific application of the sensor node.

The modulation type used for the radio is 2FSK (Frequency Shift Keying). This is the most basic choice of FSK modulation provided and uses two different frequencies to represent binary information, however it can only support data rates up to 500kbps. If a higher data rate is required then the 4FSK modulation, providing up to 1Mbps can be used at the expense of the sensitivity. Moreover, for cases where a narrow spectral band is needed, the 2GFSK/4GFSK (Gaussian Frequency Shift Keying) can be selected even though they spend more energy.

4.1.1 Transmission range

By increasing the data rate the sensitivity of the radio will be reduced, thus lowering the range of transmission. By rearranging Friis transmission equation we can calculate the distance between two antennas for a specific transmission power and sensitivity as follows:

$$R = \sqrt{\frac{P_T G_T G_R \lambda^2}{P_R 16\pi^2}} \tag{4.1}$$

where P_T is the power transmitted, P_R is the power received or sensitivity of the antenna, G_T and G_R are the gain of the transmitting and receiving antennas and λ represents the wavelength. According to [33] typical gain values for antennas used with low-power transmitters are as low as -10 dB to -15 dB.

Considering a free space condition and the following values: P_T = +10 dBm, G_T = -15 dB, G_R = -10 dB and λ = 0.345m, we used equation 4.1 to calculate the distance with different sensitivity values presented in [11] and the results can be seen in Table 4.1

Table 4.1: Tradeoff between data rate and transmission range

Data rate	**Sensitivity (P_R)**	**Range**
500 bps	-126 dBm	9741.1 m
40 kbps	-110 dBm	1543.9 m
100 kbps	-106 dBm	974.11 m
125 kbps	-105 dBm	868.16 m
500 kbps	-97 dBm	345.62 m

4.1.2 Transmission delay

It is clear that by using a higher transmission data rate, packets will be transferred faster and energy consumed will be reduced. We experimented by using different data rates to transmit packets of different sizes in order to observe the time required for transmission, the results can be seen in Table 4.2.

Table 4.2: Tradeoff between data rate and transmission delay

Data rate	23 Bytes	46 Bytes	69 Bytes
500 bps	307.42 ms	698.96 ms	1060 ms
40 kbps	2.57 ms	4.96 ms	7.48 ms
100 kbps	1.42 ms	2.58 ms	3.58 ms
125 kbps	1.36 ms	2.19 ms	3.19 ms
500 kbps	0.81 ms	1.18 ms	1.59 ms

4.2 Energy consumption

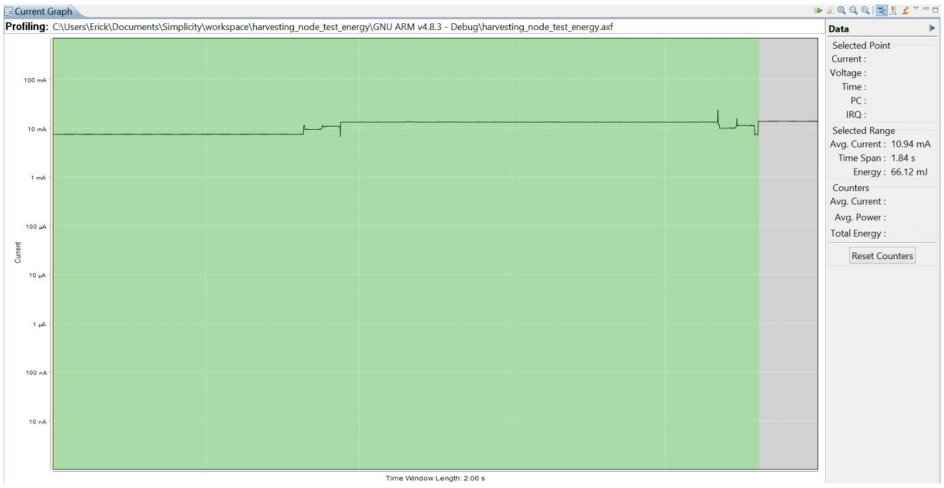
As mentioned before, one of the main considerations in designing a sensor node is to reduce the energy consumption, in this work we applied different commonly used techniques together to achieve this.

The first technique was implementing a duty cycle, this reduced considerably the node's energy consumption by switching between active and sleep states. An algorithm based on a linear quadratic tracking problem was used to dynamically calculate the duty cycle. During the sleep state, the radio is turned off and a low energy mode is entered, this is the most dominant power saving method in the node.

In addition, the energy modes provided by the microcontroller and radio presented in chapter 3 alongside with the use of interrupts, further reduce the power used while waiting for a delay or data to be fetched. Finally, the HFRCO which is an internal RC oscillator providing a clock frequency from 4 to 28 MHz was used as the clock source due to the lower power consumption than the alternative which is a crystal oscillator.

In order to measure the energy consumption of the node, the EZR32LG board was connected through the USB debugger and the Energy Profiler was used to trace the energy profiles.

In Figure 4.1 we can observe the energy required for the initialization of the node, in which for testing purposes the initial clock synchronization is limited to 1 second, making a total of 1.84 seconds before entering normal operation. The energy consumed is only of $66.12mJ$, but considering that the maximum initial synchronization period can take up to 30 minutes, the maximum energy spent in this section can ascend up to $81.75J$.



Profiling: C:\Users\Erick\Documents\Simplicity\workspace\harvesting_node_test_energy\GNU ARM v4.8.3 - Debug\harvesting_node_test_energy.axf

Data

Selected Point
Current :
Voltage :
Time :
PC :
IRQ :

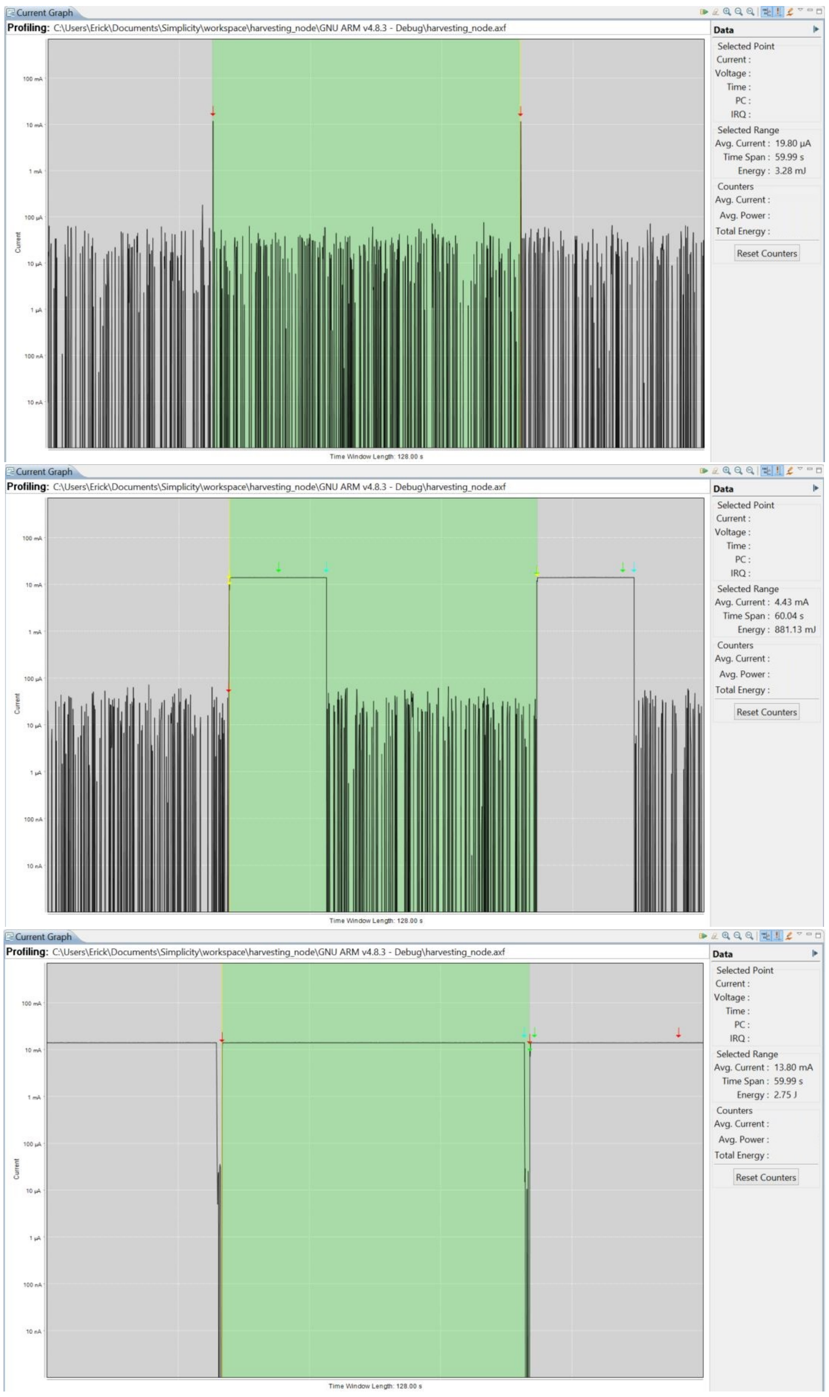
Selected Range
Avg. Current : 10.94 mA
Time Span : 1.84 s
Energy : 66.12 mJ

Counters
Avg. Current :
Avg. Power :
Total Energy :

Figure 4.1: Energy trace of node's initialization.

In Figure 4.2 we present the energy consumption of the node using different duty cycles of 4%, 33% and 99% respectively. The average power consumption during the active mode is $45.7mW$, while in the sleeping mode is only of $7.6\mu W$. When the minimum duty cycle for operation of 4% is used, the node only wakes up to read the sensors and saves the readings before going back to sleep, this consumes a total of $3.28mJ$ for one DC period. Now considering a typical duty cycle of 33%, where the node reads the sensors and transmits one packet, the total energy consumed during the DC period is of $881.13mJ$. Finally, when using a 99% duty cycle, which is nearly impossible because the node has to be active almost the entire time, we can notice that the total energy spent is $2.75J$, which would be the maximum energy spent during a DC period, since the node cannot reach the 100% duty cycle due to application parameters.

Finally, in Figure 4.3 we show the energy required to transmit packets of 23 bytes, 46 bytes and 69 bytes respectively. The normal data packet consists of 23 bytes, consuming a total of $143.32\mu J$, this is a very small portion of the total energy spent during the DC period. We can also observe that the increase in the time required for packet transmission and therefore energy consumed escalate together with the packet size, however it does not happen in a linear fashion.



Profiling: C:\Users\Erick\Documents\Simplicity\workspace\harvesting_node\GNU ARM v4.8.3 - Debug\harvesting_node.axf

Time Window Length: 128.00 s

Data
Selected Point
Current :
Voltage :
Time :
PC :
IRQ :
Selected Range
Avg. Current : 19.80 µA
Time Span : 59.99 s
Energy : 3.28 mJ
Counters
Avg. Current :
Avg. Power :
Total Energy :

Profiling: C:\Users\Erick\Documents\Simplicity\workspace\harvesting_node\GNU ARM v4.8.3 - Debug\harvesting_node.axf

Time Window Length: 128.00 s

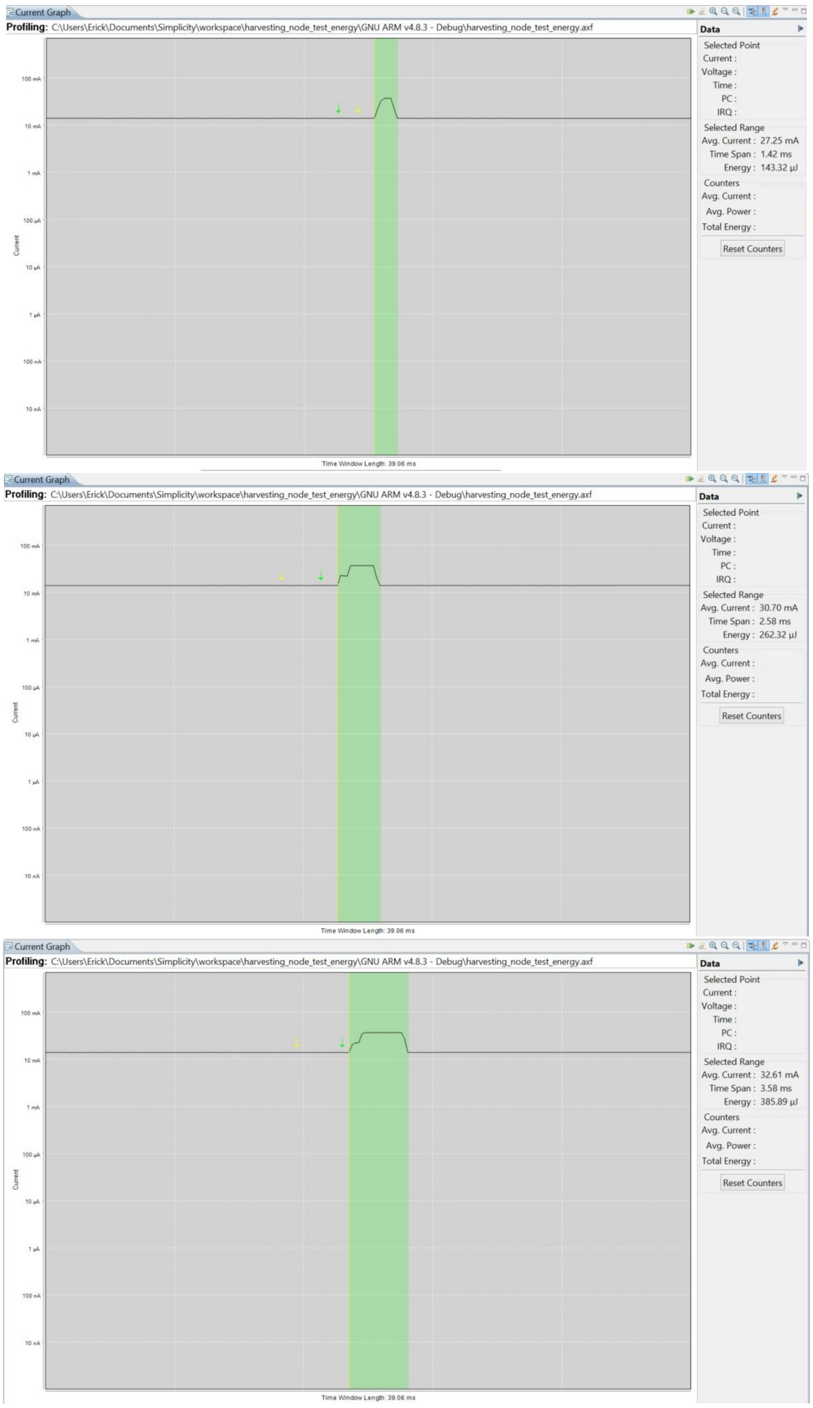
Data
Selected Point
Current :
Voltage :
Time :
PC :
IRQ :
Selected Range
Avg. Current : 4.43 mA
Time Span : 60.04 s
Energy : 881.13 mJ
Counters
Avg. Current :
Avg. Power :
Total Energy :

Profiling: C:\Users\Erick\Documents\Simplicity\workspace\harvesting_node\GNU ARM v4.8.3 - Debug\harvesting_node.axf

Time Window Length: 128.00 s

Data
Selected Point
Current :
Voltage :
Time :
PC :
IRQ :
Selected Range
Avg. Current : 13.80 mA
Time Span : 59.99 s
Energy : 2.75 J
Counters
Avg. Current :
Avg. Power :
Total Energy :

Figure 4.2: Energy trace using different duty cycles.



Profiling: C:\Users\Erick\Documents\Simplicity\workspace\harvesting_node_test_energy\GNU ARM v4.8.3 - Debug\harvesting_node_test_energy.axf

Data
Selected Point
Current :
Voltage :
Time :
PC :
IRQ :
Selected Range
Avg. Current : 27.25 mA
Time Span : 1.42 ms
Energy : 143.32 µJ
Counters
Avg. Current :
Avg. Power :
Total Energy :

Profiling: C:\Users\Erick\Documents\Simplicity\workspace\harvesting_node_test_energy\GNU ARM v4.8.3 - Debug\harvesting_node_test_energy.axf

Data
Selected Point
Current :
Voltage :
Time :
PC :
IRQ :
Selected Range
Avg. Current : 30.70 mA
Time Span : 2.58 ms
Energy : 262.32 µJ
Counters
Avg. Current :
Avg. Power :
Total Energy :

Profiling: C:\Users\Erick\Documents\Simplicity\workspace\harvesting_node_test_energy\GNU ARM v4.8.3 - Debug\harvesting_node_test_energy.axf

Data
Selected Point
Current :
Voltage :
Time :
PC :
IRQ :
Selected Range
Avg. Current : 32.61 mA
Time Span : 3.58 ms
Energy : 385.89 µJ
Counters
Avg. Current :
Avg. Power :
Total Energy :

Figure 4.3: Energy trace of packet transmission with different sizes.

4.3 LQ Tracking simulation

For these tests, one of the boards was connected through the USB (Universal Serial Bus) debugger and printed out the data through a virtual COM port. A terminal was used to capture this data. For the energy harvested data we used the numbers plotted in Figure 3.12.

First, we tested different fixed duty cycles to analyze the variation of the battery level in order to corroborate our model. In Figure 4.4 we can observe three tests with duty cycles of 5%, 50% and 80% respectively. Here we can notice that the extremely low energy consumption of the sensor node allows operating at a duty cycle of 50% under a sunny day while maintaining the same battery level. In addition, in case the weather conditions are not optimal, using the minimum duty cycle for radio communication of 5% would allow to recharge the battery. However if a very high duty cycle is used, we can see that the final battery level at the end of the day would be less than at the beginning and the node would most likely deplete its battery in a few days, depending on weather conditions.

For the second part, we used the duty cycle algorithm with the fixed parameters specified in section 3.3.3.1, with the exception of α which controls the smoothing of the duty cycle output. Figure 4.5 presents three tests with an α of 1, 0.05 and 0.0005 respectively. From these tests we can see that the algorithm can achieve energy neutrality while having a good performance. Moreover, the variation in duty cycles can be reduced by modifying the parameter α if the application requires it, however when the smoothing is too high, then the node fails to adapt quickly to abrupt changes in battery level and there can be some dead times.

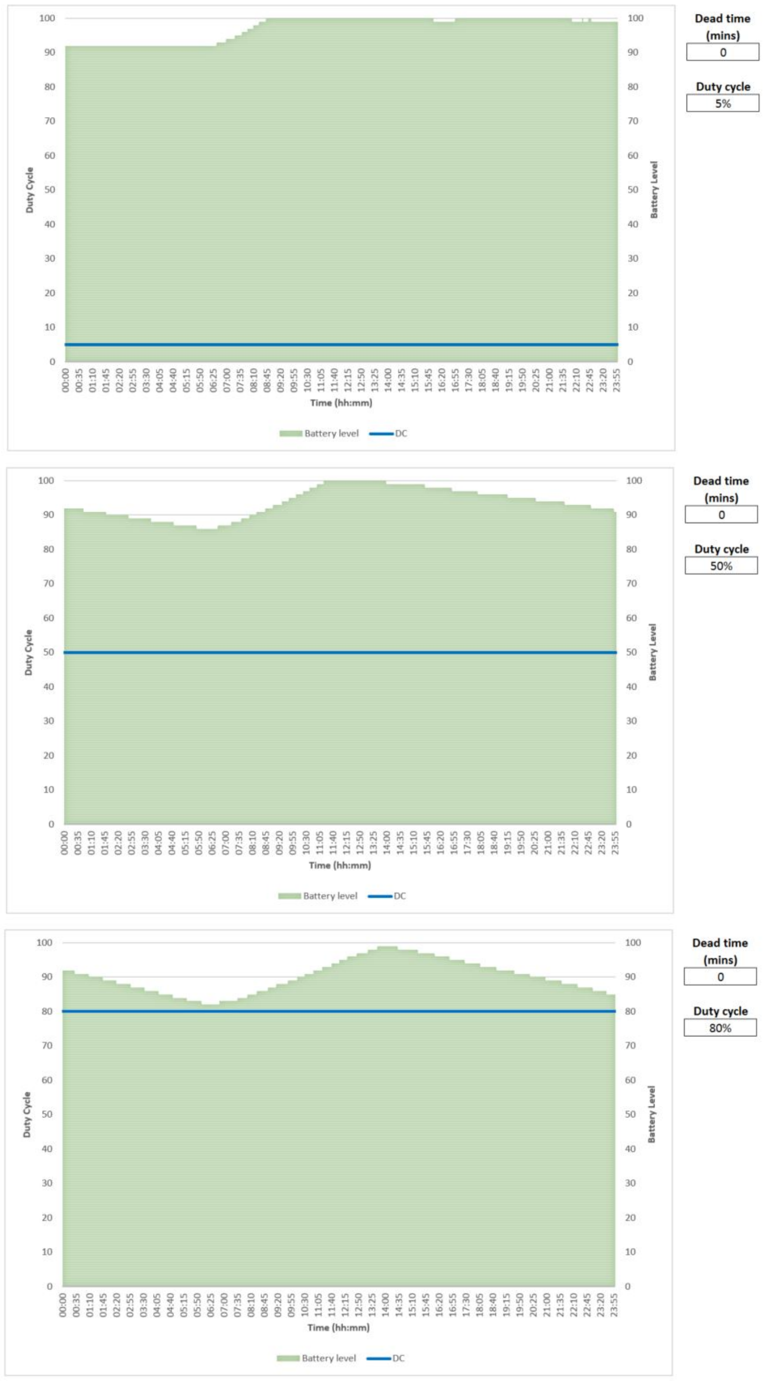
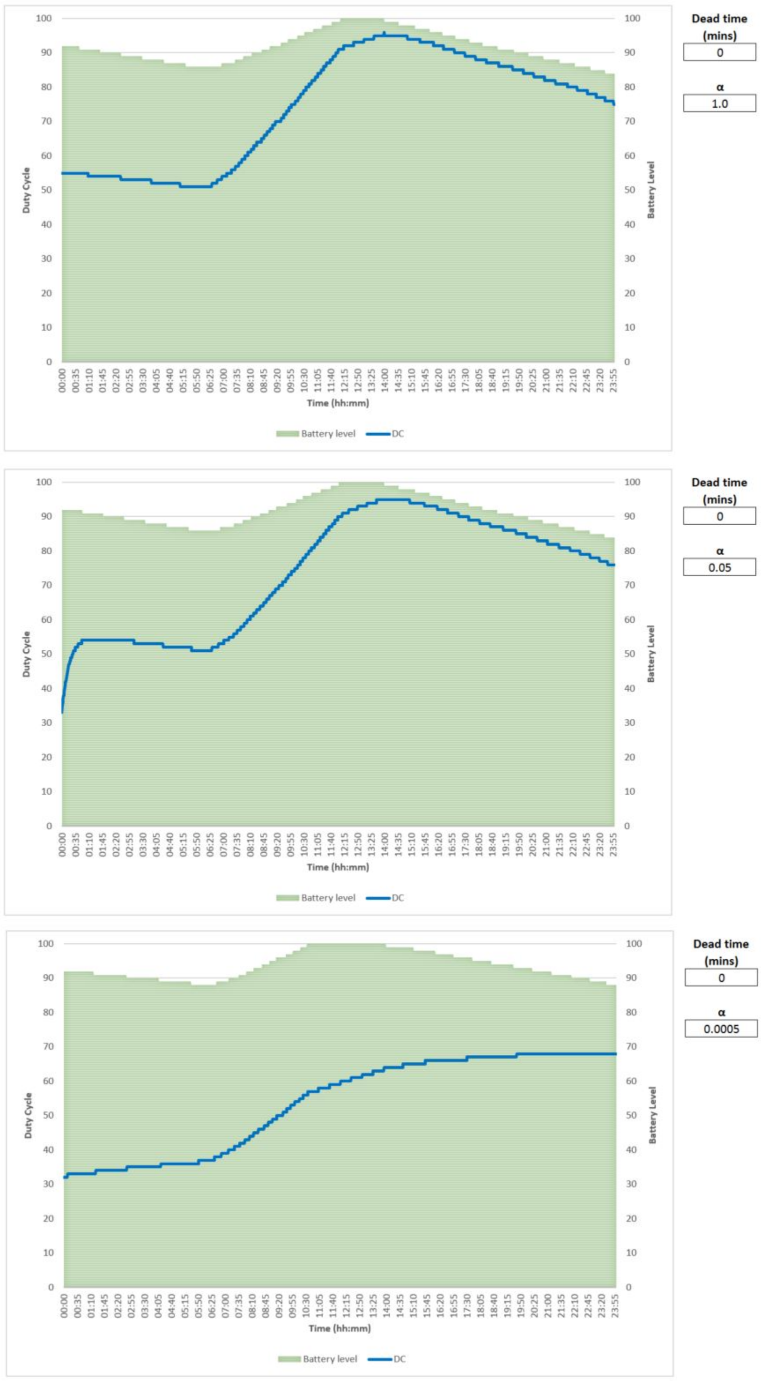


Figure 4.4: Battery variation with fixed duty cycles.



Dead time (mins) 0

α 1.0

Dead time (mins) 0

α 0.05

Dead time (mins) 0

α 0.0005

Figure 4.5: LQ tracking duty cycle.

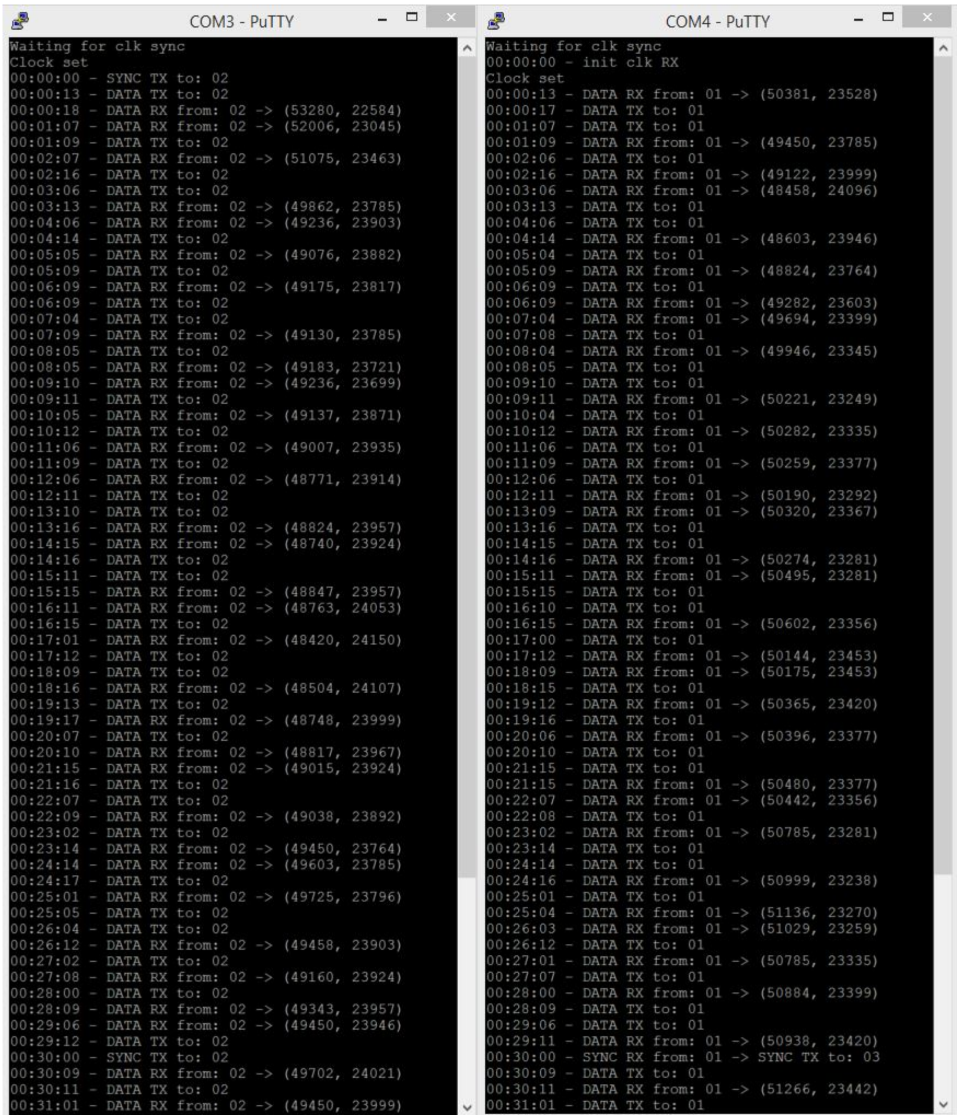
4.4 Network testing

To test the communication between the nodes, the two development boards were powered via the USB socket, providing a constant power since our harvesting unit does not possess the required battery. However, they used the test battery values to modify their duty cycles in real-time. In addition, they were printing data through a virtual COM port.

Each node had an ID number, being 0x01 for node 1 and 0x02 for node 2. Node 1 was powered up first since it is considered the root node in the tests, while node 2 was started afterwards. We can observe in Figure 4.6 the data output of both nodes during the first 30 minutes. At minute 0 a SYNC packet is sent from node 1 to node 2, allowing node 2 to set up its clock time and start operating. For the first 30 minutes we can observe both nodes exchanging DATA packets, however we can see that in some cases such as in minute 8, there is a mismatch on the timestamp. This is due to the small difference in the clocks which can be adjusted by modifying the parameter CLOCK_INIT_DELAY in the application source code. At minute 30, the SYNC period is over and node 1 sends a SYNC packet to node 2, which re-transmits to node 3 and so on.

In order to test the SYNC packets, the SYNC period was changed to 3 minutes for a faster testing. In this case, node 1 is considered the root node, whereas node 2 is a node in the next level of hierarchy in the routing tree topology. In Figure 4.7 the outputs of both nodes are printed out. The first SYNC packet of node 1 is used to set the initial clock of node 2. The next two SYNC packets of node 1 are received by node 2 and then re-transmitted to the nodes in the next level of the routing tree, which in this test was node 3. After each node re-transmits their SYNC packet they synchronize their own clock with it.

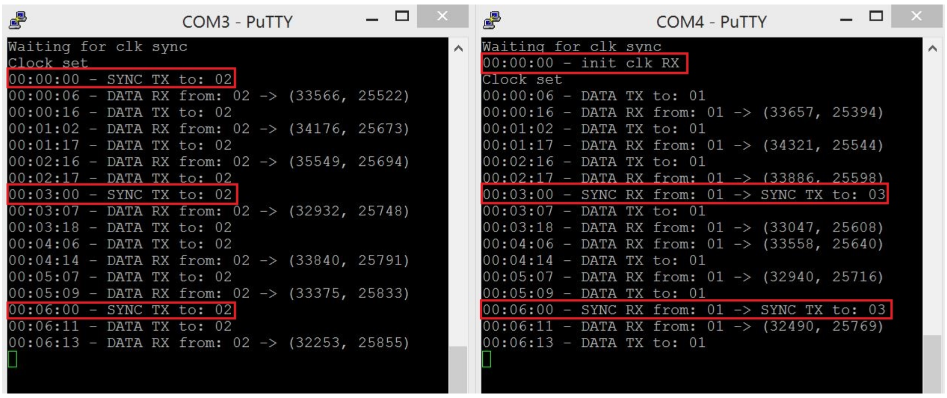
Another feature of the node is the ability to aggregate data when the battery level is not enough for transmitting a packet. This works by saving the values of the sensor readings and whenever the node has enough energy, then it sends an average of the values, this can be seen in Figure 4.8.



```
Waiting for clk sync
Clock set
00:00:00 - SYNC TX to: 02
00:00:13 - DATA TX to: 02
00:00:18 - DATA RX from: 02 -> (53280, 22584)
00:01:07 - DATA RX from: 02 -> (52006, 23045)
00:01:09 - DATA TX to: 02
00:02:07 - DATA RX from: 02 -> (51075, 23463)
00:02:16 - DATA TX to: 02
00:03:06 - DATA TX to: 02
00:03:13 - DATA RX from: 02 -> (49862, 23785)
00:04:06 - DATA RX from: 02 -> (49236, 23903)
00:04:14 - DATA TX to: 02
00:05:05 - DATA RX from: 02 -> (49076, 23882)
00:05:09 - DATA TX to: 02
00:06:09 - DATA RX from: 02 -> (49175, 23817)
00:06:09 - DATA TX to: 02
00:07:04 - DATA TX to: 02
00:07:09 - DATA RX from: 02 -> (49130, 23785)
00:08:05 - DATA TX to: 02
00:08:05 - DATA RX from: 02 -> (49183, 23721)
00:09:10 - DATA RX from: 02 -> (49236, 23699)
00:09:11 - DATA TX to: 02
00:10:05 - DATA RX from: 02 -> (49137, 23871)
00:10:12 - DATA TX to: 02
00:11:06 - DATA RX from: 02 -> (49007, 23935)
00:11:09 - DATA TX to: 02
00:12:06 - DATA RX from: 02 -> (48771, 23914)
00:12:11 - DATA TX to: 02
00:13:10 - DATA TX to: 02
00:13:16 - DATA RX from: 02 -> (48824, 23957)
00:14:15 - DATA RX from: 02 -> (48740, 23924)
00:14:16 - DATA TX to: 02
00:15:11 - DATA TX to: 02
00:15:15 - DATA RX from: 02 -> (48847, 23957)
00:16:11 - DATA RX from: 02 -> (48763, 24053)
00:16:15 - DATA TX to: 02
00:17:01 - DATA RX from: 02 -> (48420, 24150)
00:17:12 - DATA TX to: 02
00:18:09 - DATA TX to: 02
00:18:16 - DATA RX from: 02 -> (48504, 24107)
00:19:13 - DATA TX to: 02
00:19:17 - DATA RX from: 02 -> (48748, 23999)
00:20:07 - DATA TX to: 02
00:20:10 - DATA RX from: 02 -> (48817, 23967)
00:21:15 - DATA RX from: 02 -> (49015, 23924)
00:21:16 - DATA TX to: 02
00:22:07 - DATA TX to: 02
00:22:09 - DATA RX from: 02 -> (49038, 23892)
00:23:02 - DATA TX to: 02
00:23:14 - DATA RX from: 02 -> (49450, 23764)
00:24:14 - DATA RX from: 02 -> (49603, 23785)
00:24:17 - DATA TX to: 02
00:25:01 - DATA RX from: 02 -> (49725, 23796)
00:25:05 - DATA TX to: 02
00:26:04 - DATA TX to: 02
00:26:12 - DATA RX from: 02 -> (49458, 23903)
00:27:02 - DATA TX to: 02
00:27:08 - DATA RX from: 02 -> (49160, 23924)
00:28:00 - DATA TX to: 02
00:28:09 - DATA RX from: 02 -> (49343, 23957)
00:29:06 - DATA RX from: 02 -> (49450, 23946)
00:29:12 - DATA TX to: 02
00:30:00 - SYNC TX to: 02
00:30:09 - DATA RX from: 02 -> (49702, 24021)
00:30:11 - DATA TX to: 02
00:31:01 - DATA RX from: 02 -> (49450, 23999)
```

```
Waiting for clk sync
00:00:00 - init clk RX
Clock set
00:00:13 - DATA RX from: 01 -> (50381, 23528)
00:00:17 - DATA TX to: 01
00:01:07 - DATA TX to: 01
00:01:09 - DATA RX from: 01 -> (49450, 23785)
00:02:06 - DATA TX to: 01
00:02:16 - DATA RX from: 01 -> (49122, 23999)
00:03:06 - DATA RX from: 01 -> (48458, 24096)
00:03:13 - DATA TX to: 01
00:04:06 - DATA TX to: 01
00:04:14 - DATA RX from: 01 -> (48603, 23946)
00:05:04 - DATA TX to: 01
00:05:09 - DATA RX from: 01 -> (48824, 23764)
00:06:09 - DATA TX to: 01
00:06:09 - DATA RX from: 01 -> (49282, 23603)
00:07:04 - DATA RX from: 01 -> (49694, 23399)
00:07:08 - DATA TX to: 01
00:08:04 - DATA RX from: 01 -> (49946, 23345)
00:08:05 - DATA TX to: 01
00:09:10 - DATA TX to: 01
00:09:11 - DATA RX from: 01 -> (50221, 23249)
00:10:04 - DATA TX to: 01
00:10:12 - DATA RX from: 01 -> (50282, 23335)
00:11:06 - DATA TX to: 01
00:11:09 - DATA RX from: 01 -> (50259, 23377)
00:12:06 - DATA TX to: 01
00:12:11 - DATA RX from: 01 -> (50190, 23292)
00:13:09 - DATA RX from: 01 -> (50320, 23367)
00:13:16 - DATA TX to: 01
00:14:15 - DATA TX to: 01
00:14:16 - DATA RX from: 01 -> (50274, 23281)
00:15:11 - DATA RX from: 01 -> (50495, 23281)
00:15:15 - DATA TX to: 01
00:16:10 - DATA TX to: 01
00:16:15 - DATA RX from: 01 -> (50602, 23356)
00:17:00 - DATA TX to: 01
00:17:12 - DATA RX from: 01 -> (50144, 23453)
00:18:09 - DATA RX from: 01 -> (50175, 23453)
00:18:15 - DATA TX to: 01
00:19:12 - DATA RX from: 01 -> (50365, 23420)
00:19:16 - DATA TX to: 01
00:20:06 - DATA RX from: 01 -> (50396, 23377)
00:20:10 - DATA TX to: 01
00:21:15 - DATA TX to: 01
00:21:15 - DATA RX from: 01 -> (50480, 23377)
00:22:07 - DATA RX from: 01 -> (50442, 23356)
00:22:08 - DATA TX to: 01
00:23:02 - DATA RX from: 01 -> (50785, 23281)
00:23:14 - DATA TX to: 01
00:24:14 - DATA TX to: 01
00:24:16 - DATA RX from: 01 -> (50999, 23238)
00:25:01 - DATA TX to: 01
00:25:04 - DATA RX from: 01 -> (51136, 23270)
00:26:03 - DATA RX from: 01 -> (51029, 23259)
00:26:12 - DATA TX to: 01
00:27:01 - DATA RX from: 01 -> (50785, 23335)
00:27:07 - DATA TX to: 01
00:28:00 - DATA RX from: 01 -> (50884, 23399)
00:28:09 - DATA TX to: 01
00:29:06 - DATA TX to: 01
00:29:11 - DATA RX from: 01 -> (50938, 23420)
00:30:00 - SYNC RX from: 01 -> SYNC TX to: 03
00:30:09 - DATA TX to: 01
00:30:11 - DATA RX from: 01 -> (51266, 23442)
00:31:01 - DATA TX to: 01
```

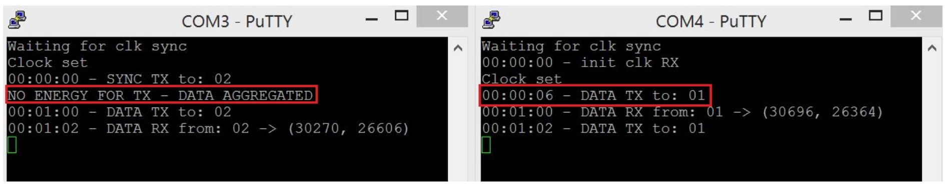
Figure 4.6: Data packet test.



```
Waiting for clk sync
Clock set
00:00:00 - SYNC TX to: 02
00:00:06 - DATA RX from: 02 -> (33566, 25522)
00:00:16 - DATA TX to: 02
00:01:02 - DATA RX from: 02 -> (34176, 25673)
00:01:17 - DATA TX to: 02
00:02:16 - DATA RX from: 02 -> (35549, 25694)
00:02:17 - DATA TX to: 02
00:03:00 - SYNC TX to: 02
00:03:07 - DATA RX from: 02 -> (32932, 25748)
00:03:18 - DATA TX to: 02
00:04:06 - DATA TX to: 02
00:04:14 - DATA RX from: 02 -> (33840, 25791)
00:05:07 - DATA TX to: 02
00:05:09 - DATA RX from: 02 -> (33375, 25833)
00:06:00 - SYNC TX to: 02
00:06:11 - DATA TX to: 02
00:06:13 - DATA RX from: 02 -> (32253, 25855)
```

```
Waiting for clk sync
00:00:00 - init clk RX
Clock set
00:00:06 - DATA TX to: 01
00:00:16 - DATA RX from: 01 -> (33657, 25394)
00:01:02 - DATA TX to: 01
00:01:17 - DATA RX from: 01 -> (34321, 25544)
00:02:16 - DATA TX to: 01
00:02:17 - DATA RX from: 01 -> (33886, 25598)
00:03:00 - SYNC RX from: 01 -> SYNC TX to: 03
00:03:07 - DATA TX to: 01
00:03:18 - DATA RX from: 01 -> (33047, 25608)
00:04:06 - DATA RX from: 01 -> (33558, 25640)
00:04:14 - DATA TX to: 01
00:05:07 - DATA RX from: 01 -> (32940, 25716)
00:05:09 - DATA TX to: 01
00:06:00 - SYNC RX from: 01 -> SYNC TX to: 03
00:06:11 - DATA RX from: 01 -> (32490, 25769)
00:06:13 - DATA TX to: 01
```

Figure 4.7: SYNC packet test.



```
Waiting for clk sync
Clock set
00:00:00 - SYNC TX to: 02
NO ENERGY FOR TX - DATA AGGREGATED
00:01:00 - DATA TX to: 02
00:01:02 - DATA RX from: 02 -> (30270, 26606)
```

```
Waiting for clk sync
00:00:00 - init clk RX
Clock set
00:00:06 - DATA TX to: 01
00:01:00 - DATA RX from: 01 -> (30696, 26364)
00:01:02 - DATA TX to: 01
```

Figure 4.8: SYNC packet test.

Chapter 5

Summary and Recommendations for Further Work

In this final chapter we present a summary of the results obtained in this work and the conclusions derived from them. In addition, a discussion of the strengths and limitations of the work is given and recommendations for future work are proposed.

5.1 Summary and Discussion

The hardware architecture of the sensor node was implemented using the EZR32LG330 microcontroller containing the Si4460 radio transceiver, together with the Si7021 temperature and relative humidity sensor and the Energy Harvesting kit as the solar energy source [Fig. 3.1]. As for the energy storage, it was determined that a AA sized NiMH battery with a typical voltage of 1.2V and a capacity of 2300mAh was a good option to support the operation of the node while harvesting solar energy. However, since the harvesting kit had a different energy storage composed of capacitors, we simulated the battery's energy level by using solar harvesting data [Fig. 3.12] in combination with the energy consumption of the node.

For the software solution, a low-energy application was developed consisting of a duty cycling technique and a wireless communication scheme and it was tested in the hardware mentioned before. The Linear Quadratic Tracking algorithm was used to calculate the duty cycle based on the battery voltage level. The communication scheme was achieved by dividing the transmission slot into two, one for synchronization and another for data [Fig. 3.14]. During the synchronization slot, the root node can transmit the clock time so all other nodes in the network make adjustments in case of clock drifting. While in the data slot, each node takes into consideration the active time of the current DC period from its own and its neighbors to find a common time when both are awake to avoid packet losses and also waits for a random time before transmitting to avoid collisions.

After experimenting with different data rates and comparing its repercussions in the transmission range and transmission delay, we concluded that for this application a data rate of 100 kbps was a suitable option. This is because it provides a range of nearly $1km$ in free space conditions and it only spends $1.42ms$ to send a data packet of 23 bytes, using a total energy of 143.32 μJ.

During the energy consumption tests we showed how by increasing the packet size, the time and energy spent during transmission rises. Moreover, we corroborated the low energy consumption by showing that with the minimum duty cycle of 4% the node's power consumption is only of $54.7\mu W$ using a total energy of $3.28mJ$ in one DC period. Furthermore, if a bigger duty cycle of 33% is used, it still consumes as little as $14.7mW$ which is equivalent to a total energy of $881.13mJ$ during each minute.

Furthermore, the battery level simulation was verified using constant duty cycles and the LQ tracking algorithm was tested using different values of smoothing (α), which reduces the variation of the duty cycles in case the application needs it. An $\alpha = 0.0005$ was considered to be a convenient value to provide enough smoothing without failing to adapt to rapid variations in the battery level.

In overall, this implementation achieves good results in managing the energy available to perform at its maximum level, while providing a reliable communication scheme. However, it does require a battery of enough capacity to sustain the operation and a routing protocol is needed to test the multi-hopping transmission.

5.2 Recommendations for Further Work

Wireless sensor networks is a broad field and there are many possibilities for improvement. We recommend the following to improve this work:

Routing aware algorithm

After implementing a duty cycling technique to control the behavior of each node individually, the next step is to develop a routing algorithm which considers different factors such as energy harvesting potential, battery level and latency to make the network more efficient. In the background chapter two routing techniques were presented, a harvesting-aware routing and a low-latency routing. The first one aims to balance the energy use of the sensor nodes in the network by employing a cost metric which considers the expected rate of harvesting energy and the residual battery. The second one, on the other hand, seeks to reduce the latency in the network by calculating the lowest latency path to the destination.

Acknowledgment packets

A further improvement to make the network communication more reliable is to use acknowledgment packets. Whenever a node sends out a data packet it should wait for a certain period for an acknowledgment from the receiving node, if it does not receive it in due time then it means the packet has been lost and the node should retransmit. However, it might happen that the receiving node is out of battery, therefore the number of retransmissions should be kept low to avoid unnecessary waste of energy and instead the information should be aggregated for the next slot.

Data aggregation

Whenever the sensor node does not have enough energy to transmit data it aggregates it. This is simply done by adding up the samples of the sensor and transmitting an average when it has enough energy. For certain applications this is accurate enough, but in case every reading is essential, every value should be saved in the node and then transmitted when it has enough energy.

Appendix A

Additional Information

A.1 Schematics

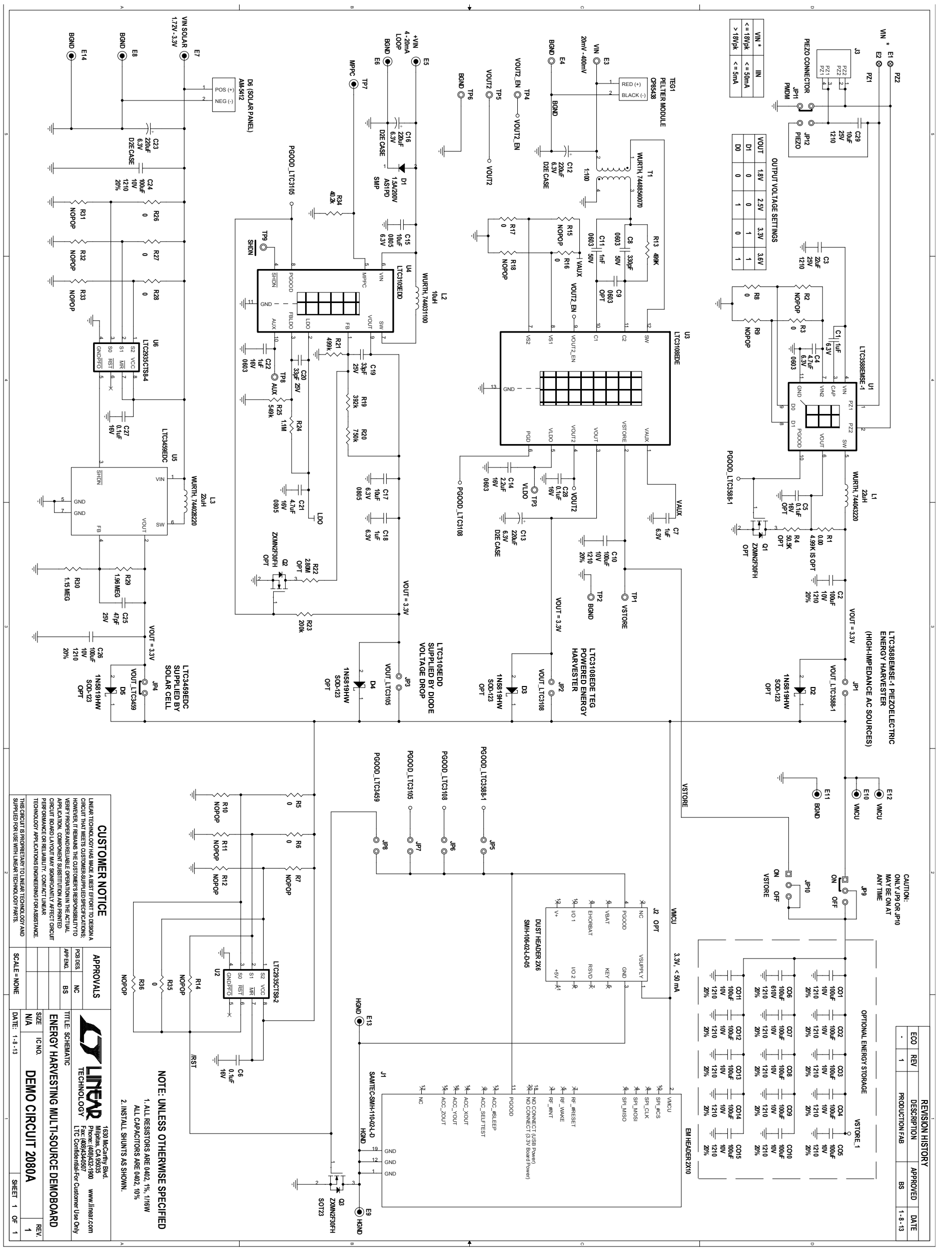
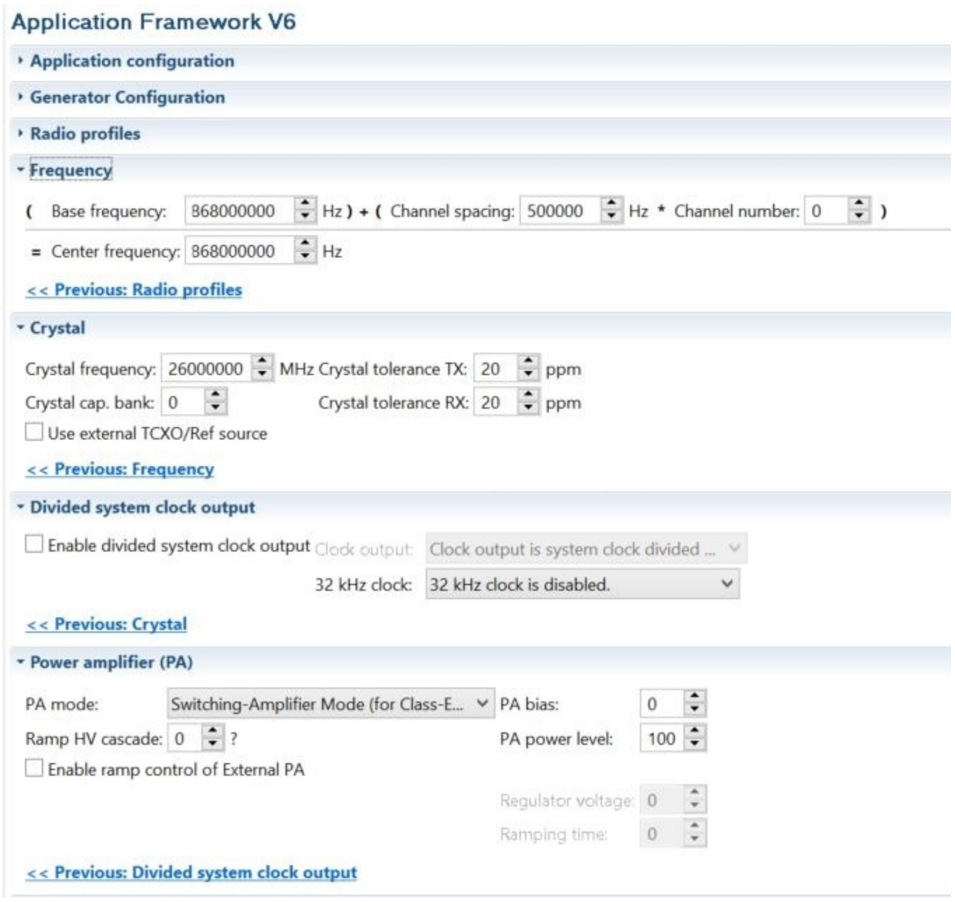

LTC3588EMSE-1 PIEZOELECTRIC ENERGY HARVESTER (HIGH-IMPEDANCE AC SOURCES)
LTC3108EDE TEG POWERED ENERGY HARVESTER
LTC3105EDD SUPPLIED BY DIODE VOLTAGE DROP
LTC3459EDC SUPPLIED BY SOLAR CELL
OPTIONAL ENERGY STORAGE
REVISION HISTORY
CUSTOMER NOTICE
ENERGY HARVESTING MULTI-SOURCE DEMOBOARD
DEMO CIRCUIT 2080A
NOTE: UNLESS OTHERWISE SPECIFIED
1. ALL RESISTORS ARE 0402, 1%, 1/16W
ALL CAPACITORS ARE 0402, 10%
2. INSTALL SHUNTS AS SHOWN.

Figure A.1: Schematic of Energy Harvester.

A.2 Radio Configuration



Application Framework V6

- Application configuration
- Generator Configuration
- Radio profiles
- Frequency

(Base frequency: 868000000 Hz) + (Channel spacing: 500000 Hz * Channel number: 0)

= Center frequency: 868000000 Hz

<< Previous: Radio profiles

- Crystal

Crystal frequency: 26000000 MHz Crystal tolerance TX: 20 ppm

Crystal cap. bank: 0 Crystal tolerance RX: 20 ppm

☐ Use external TCXO/Ref source

<< Previous: Frequency

- Divided system clock output

☐ Enable divided system clock output Clock output: Clock output is system clock divided ...

32 kHz clock: 32 kHz clock is disabled.

<< Previous: Crystal

- Power amplifier (PA)

PA mode: Switching-Amplifier Mode (for Class-E... PA bias: 0

Ramp HV cascade: 0 ? PA power level: 100

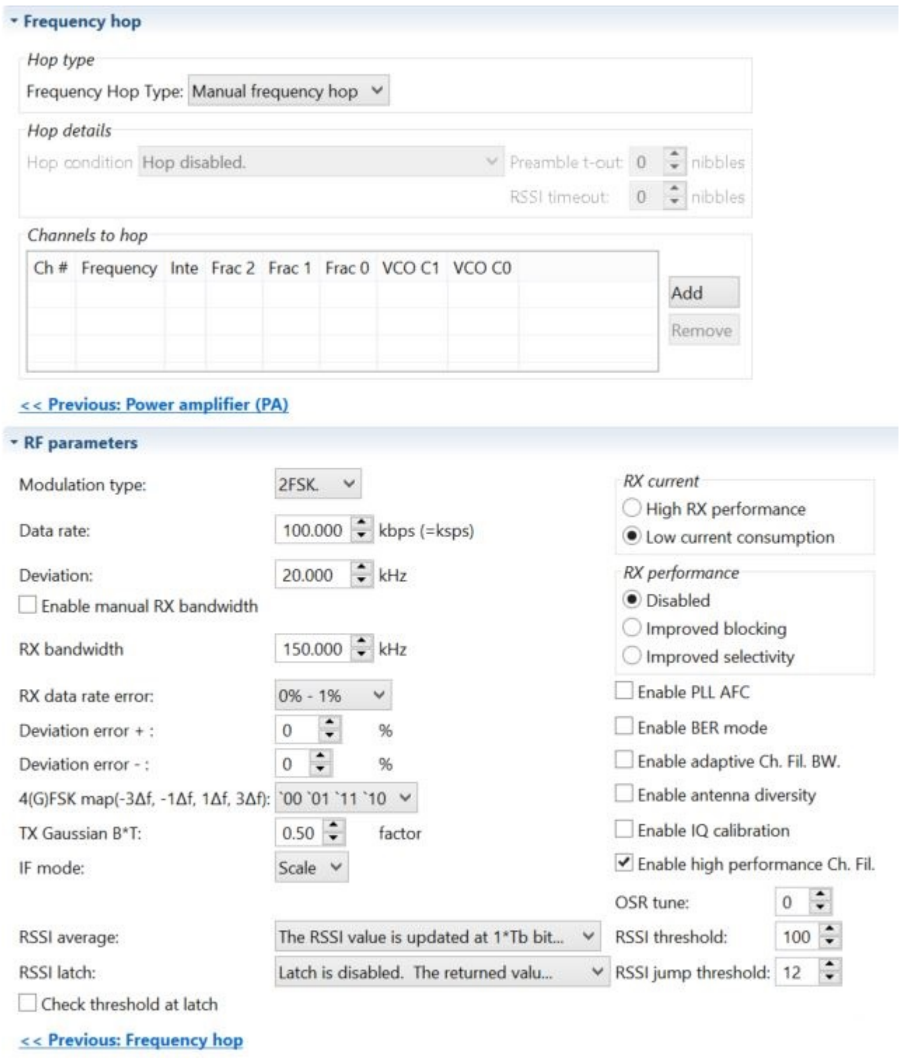
☐ Enable ramp control of External PA

Regulator voltage: 0

Ramping time: 0

<< Previous: Divided system clock output

Figure A.2: Radio configuration 1.



Frequency hop

Hop type

Frequency Hop Type: Manual frequency hop

Hop details

Hop condition: Hop disabled. Preamble t-out: 0 nibbles

RSSI timeout: 0 nibbles

Channels to hop

Ch #	Frequency	Inte	Frac 2	Frac 1	Frac 0	VCO C1	VCO C0

RF parameters

Modulation type:	2FSK.	
Data rate:	100.000	kbps (=ksps)
Deviation:	20.000	kHz
☐ Enable manual RX bandwidth		
RX bandwidth	150.000	kHz
RX data rate error:	0% - 1%	
Deviation error + :	0	%
Deviation error - :	0	%
4(G)FSK map(-3Δf, -1Δf, 1Δf, 3Δf):	`00 `01 `11 `10	
TX Gaussian B*T:	0.50	factor
IF mode:	Scale	
RSSI average:	The RSSI value is updated at 1*Tb bit...	
RSSI latch:	Latch is disabled. The returned valu...	
☐ Check threshold at latch		

RX current

High RX performance [✓ Low current consumption]

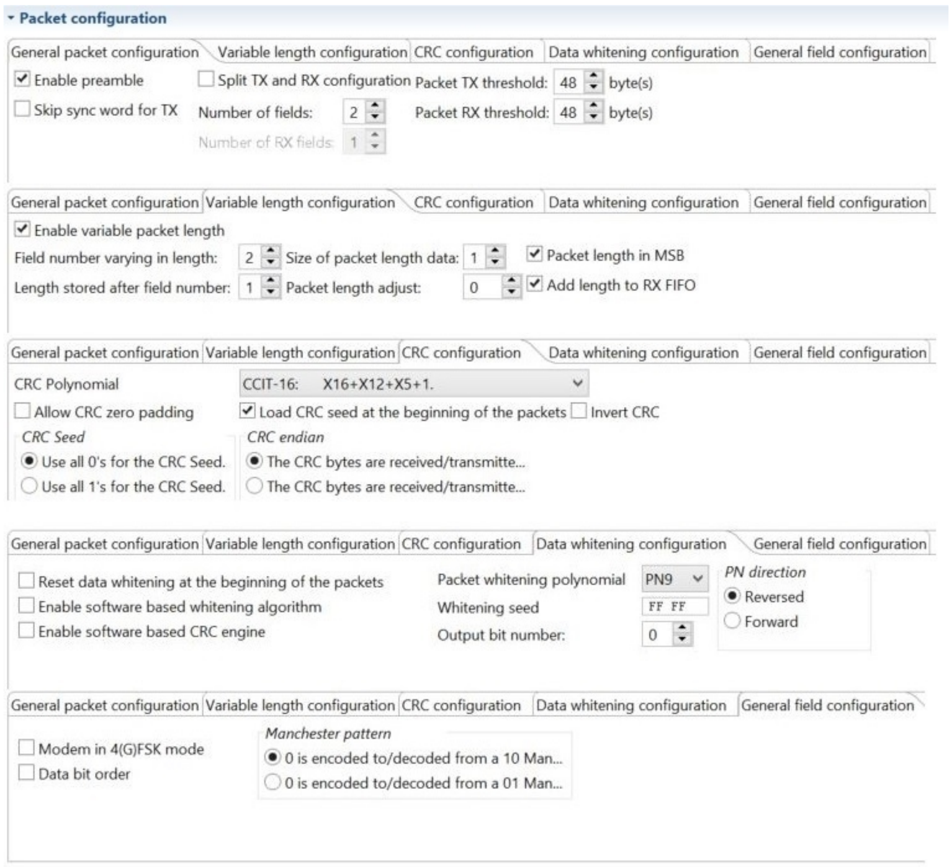
RX performance

[✓ Disabled] Improved blocking Improved selectivity

☐ Enable PLL AFC
☐ Enable BER mode
☐ Enable adaptive Ch. Fil. BW.
☐ Enable antenna diversity
☐ Enable IQ calibration
☑ Enable high performance Ch. Fil.

OSR tune:	0
RSSI threshold:	100
RSSI jump threshold:	12

Figure A.3: Radio configuration 2.



Packet configuration

General packet configuration | Variable length configuration | CRC configuration | Data whitening configuration | General field configuration

[✓ Enable preamble]
Skip sync word for TX
Split TX and RX configuration
Number of fields: 2
Number of RX fields: 1
Packet TX threshold: 48 byte(s)
Packet RX threshold: 48 byte(s)

General packet configuration | Variable length configuration | CRC configuration | Data whitening configuration | General field configuration

[✓ Enable variable packet length]
Field number varying in length: 2
Size of packet length data: 1
[✓ Packet length in MSB]
Length stored after field number: 1
Packet length adjust: 0
[✓ Add length to RX FIFO]

General packet configuration | Variable length configuration | CRC configuration | Data whitening configuration | General field configuration

CRC Polynomial: CCIT-16: X16+X12+X5+1.
Allow CRC zero padding
[✓ Load CRC seed at the beginning of the packets]
Invert CRC

CRC Seed
[✓ Use all 0's for the CRC Seed.]
Use all 1's for the CRC Seed.

CRC endian
[✓ The CRC bytes are received/transmitte...]
The CRC bytes are received/transmitte...

General packet configuration | Variable length configuration | CRC configuration | Data whitening configuration | General field configuration

Reset data whitening at the beginning of the packets
Enable software based whitening algorithm
Enable software based CRC engine
Packet whitening polynomial: PN9
Whitening seed: FF FF
Output bit number: 0

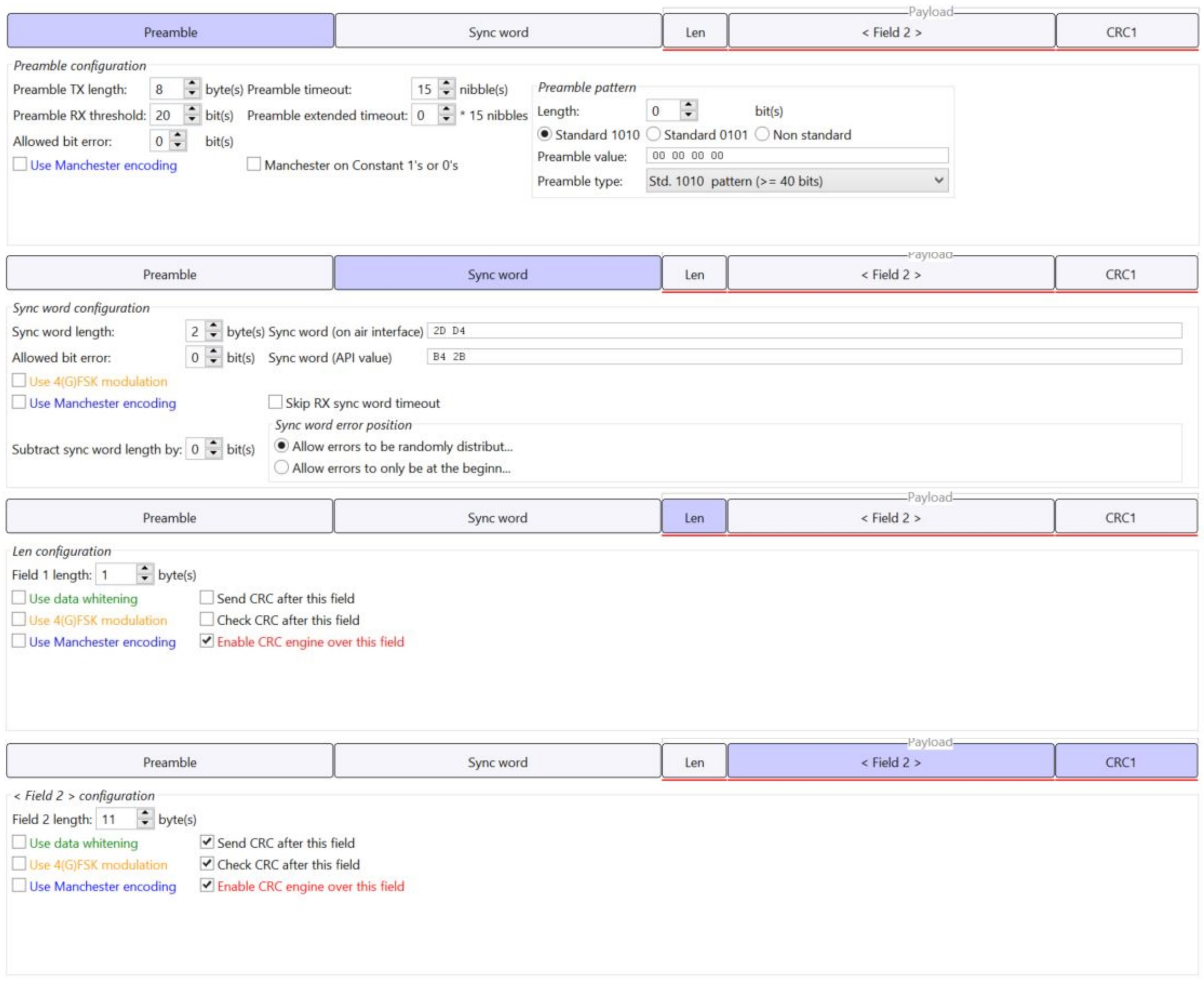
PN direction
[✓ Reversed]
Forward

General packet configuration | Variable length configuration | CRC configuration | Data whitening configuration | General field configuration

Modem in 4(G)FSK mode
Data bit order

Manchester pattern
[✓ 0 is encoded to/decoded from a 10 Man...]
0 is encoded to/decoded from a 01 Man...

Figure A.4: Radio configuration 3.



Preamble configuration

Preamble TX length: 8 byte(s) Preamble timeout: 15 nibble(s)
Preamble RX threshold: 20 bit(s) Preamble extended timeout: 0 * 15 nibbles
Allowed bit error: 0 bit(s)
☐ Use Manchester encoding ☐ Manchester on Constant 1's or 0's

Preamble pattern

Length: 0 bit(s)
● Standard 1010 ○ Standard 0101 ○ Non standard
Preamble value: 00 00 00 00
Preamble type: Std. 1010 pattern (>= 40 bits)

Sync word configuration

Sync word length: 2 byte(s) Sync word (on air interface) 2D D4
Allowed bit error: 0 bit(s) Sync word (API value) B4 2B
☐ Use 4(G)FSK modulation
☐ Use Manchester encoding ☐ Skip RX sync word timeout
Subtract sync word length by: 0 bit(s)

Sync word error position

● Allow errors to be randomly distribut...
○ Allow errors to only be at the beginn...

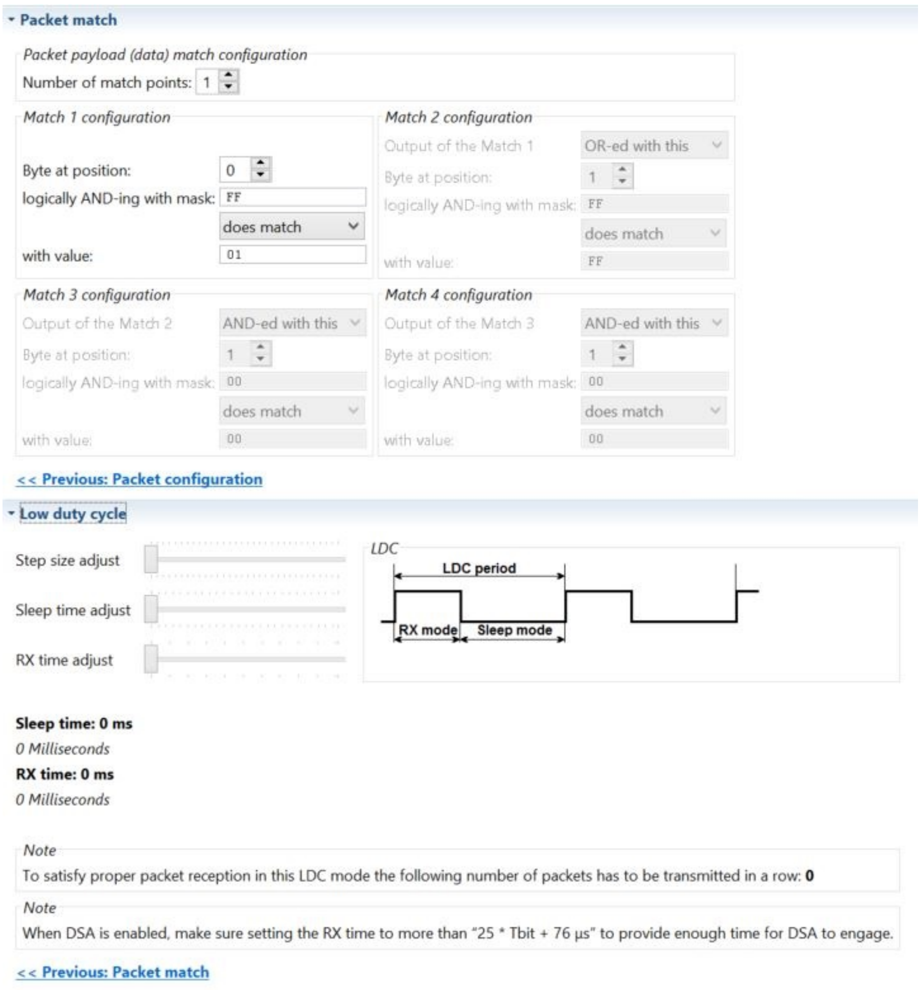
Len configuration

Field 1 length: 1 byte(s)
☐ Use data whitening ☐ Send CRC after this field
☐ Use 4(G)FSK modulation ☐ Check CRC after this field
☐ Use Manchester encoding ✓ Enable CRC engine over this field

< Field 2 > configuration

Field 2 length: 11 byte(s)
☐ Use data whitening ✓ Send CRC after this field
☐ Use 4(G)FSK modulation ✓ Check CRC after this field
☐ Use Manchester encoding ✓ Enable CRC engine over this field

Figure A.5: Radio configuration 4.



Packet match

Packet payload (data) match configuration

Number of match points: 1

Match 1 configuration

Byte at position: 0
logically AND-ing with mask: FF
does match
with value: 01

Match 2 configuration

Output of the Match 1: OR-ed with this
Byte at position: 1
logically AND-ing with mask: FF
does match
with value: FF

Match 3 configuration

Output of the Match 2: AND-ed with this
Byte at position: 1
logically AND-ing with mask: 00
does match
with value: 00

Match 4 configuration

Output of the Match 3: AND-ed with this
Byte at position: 1
logically AND-ing with mask: 00
does match
with value: 00

<< Previous: Packet configuration

Low duty cycle

Step size adjust
Sleep time adjust
RX time adjust

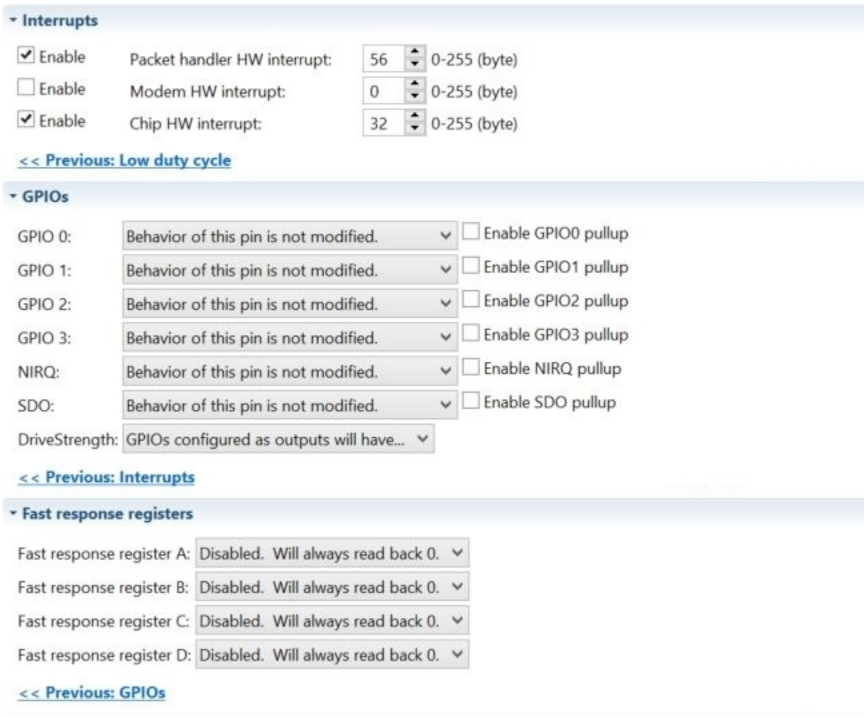
Sleep time: 0 ms
0 Milliseconds
RX time: 0 ms
0 Milliseconds

Note
To satisfy proper packet reception in this LDC mode the following number of packets has to be transmitted in a row: **0**

Note
When DSA is enabled, make sure setting the RX time to more than "25 * Tbit + 76 µs" to provide enough time for DSA to engage.

<< Previous: Packet match

Figure A.6: Radio configuration 5.



Interrupts

[✓ Enable]	Packet handler HW interrupt:	56	0-255 (byte)
Enable	Modem HW interrupt:	0	0-255 (byte)
[✓ Enable]	Chip HW interrupt:	32	0-255 (byte)

<< Previous: Low duty cycle

GPIOs

GPIO 0:	Behavior of this pin is not modified.	Enable GPIO0 pullup
GPIO 1:	Behavior of this pin is not modified.	Enable GPIO1 pullup
GPIO 2:	Behavior of this pin is not modified.	Enable GPIO2 pullup
GPIO 3:	Behavior of this pin is not modified.	Enable GPIO3 pullup
NIRQ:	Behavior of this pin is not modified.	Enable NIRQ pullup
SDO:	Behavior of this pin is not modified.	Enable SDO pullup
DriveStrength:	GPIOs configured as outputs will have...	

<< Previous: Interrupts

Fast response registers

Fast response register A:	Disabled. Will always read back 0.
Fast response register B:	Disabled. Will always read back 0.
Fast response register C:	Disabled. Will always read back 0.
Fast response register D:	Disabled. Will always read back 0.

<< Previous: GPIOs

Figure A.7: Radio configuration 6.

Appendix B

Harvesting node C code

The software solution is built upon a Silicon Labs MCU project using Simplicity Studio. It uses the *emlib* libraries and some drivers to manage the peripherals of the EZR32LG330. The source code which was developed in this work for the harvesting sensor node is organized as follows:

- config
  - app_config.h
- headers
  - communication.h
  - duty_cycling.h
  - energy_harvesting.h
  - radio.h
- src
  - communication.c
  - duty_cycling.c
  - energy_harvesting.c
  - main.c

– radio.c

Several important functions are presented in the rest of this appendix.

B.1 communication.c

```c
/***************************************************************************
 * @brief  Wait in RX mode for a sync packet to  initialize  local clock
 ***************************************************************************/
time_t COMMUNICATION_Init_Clk_Sync(void)
{
  radioPkt_data dataRX = {0};
   printf ("Waiting for clk sync\n");

  // Allocate timers for transmission delays
  if(ECODE_EMDRV_RTCDRV_OK != RTCDRV_AllocateTimer(&
     syncTxDelayTimerId))
    { printf ("Failed to allocate RTC 2!!"); while(1); }
  if(ECODE_EMDRV_RTCDRV_OK != RTCDRV_AllocateTimer(&syncTxOverTimerId
     ))
    { printf ("Failed to allocate RTC 3!!"); while(1); }
  if(ECODE_EMDRV_RTCDRV_OK != RTCDRV_AllocateTimer(&
     dataTxDelayTimerId))
    { printf ("Failed to allocate RTC 4!!"); while(1); }

  // Set a timeout in case there is no other nodes
  if (ECODE_EMDRV_RTCDRV_OK != RTCDRV_AllocateTimer(&clockInitTimerId))
    { return 0; }
  if (ECODE_EMDRV_RTCDRV_OK != RTCDRV_StartTimer(clockInitTimerId,
     rtcdrvTimerTypeOneshot, CLK_INIT_TIMEOUT, clockInitCallback, NULL))
    { return 0; }

  // Start listening
  dataRX.length = SYNC_PAYLOAD_SIZE;
  EZRADIO_StartRx(0, dataRX.length);

  // Wait for a SYNC packet or until timeout
  do {
    EMU_EnterEM2(true);

    if ( radioIrqReceived )
    {
      radioIrqReceived = false;

      // Check if packet is received and read out
      if (EZRADIO_CheckReceived(&dataRX))
      {
        if (dataRX.type == SYNC_DATA)
        {
          RTCDRV_StopTimer(clockInitTimerId);
          RTCDRV_FreeTimer(clockInitTimerId);

          // Add delay in computation time
          return dataRX.payload.cur_time;
        }
      }
    }
  } while (!clockInitTimeout);

  RTCDRV_StopTimer(clockInitTimerId);
  RTCDRV_FreeTimer(clockInitTimerId);

  return 0;
}

/*****************************************************************************
 * @brief Listen for a SYNC packet from a parent node and retransmit
 *****************************************************************************/
void COMMUNICATION_Sync_TRx(void)
{
  radioPkt_data dataRX = {0};
  radioPkt_data dataTX = {0};
  transmitSync = false;
  slotOver = false;

```

```c
  // Start a timer to signal when SYNC slot is over
  RTCDRV_StartTimer(syncTxOverTimerId, rtcdrvTimerTypeOneshot,
     SYNC_SLOT_PERIOD, syncTxOverCallback, NULL);

  // Since there is no routing information, a static implementation between node 1
     and node 2 is used, node 1 transmits the clock information and node 2 updates its
      clock and retransmits the information
#if (NODE_ID == 0x01)

  // Delay the transmission of the SYNC packet to account for clock drift
  RTCDRV_StartTimer(syncTxDelayTimerId, rtcdrvTimerTypeOneshot,
     MIN_TX_DELAY_TIME, syncTxDelayCallback, NULL);

  // Stay in this loop until SYNC slot is over
  do {
    EMU_EnterEM2(true);

    if ( radioIrqReceived )
    {
      radioIrqReceived = false;

      // Check if packet is sent
      if (EZRADIO_CheckTransmitted())
      {
        clearLED(LED_TRANSMIT);
      }
    }

    // Transmit packet to all level 2 nodes in routing table
    if (transmitSync)
    {
      transmitSync = false;
      dataTX.dest_id = 0x02;
      dataTX.src_id = NODE_ID;
      dataTX.type = SYNC_DATA;
      dataTX.length = SYNC_PAYLOAD_SIZE;
      dataTX.payload.cur_time = getTime();
      EZRADIO_StartTx(0, &dataTX);
```

```
            setLED(LED_TRANSMIT);
        }
    } while (!slotOver);

    // Stop timer
    RTCDRV_StopTimer(syncTxDelayTimerId);

#else

    // Start listening for packets with N bytes
    dataRX.length = SYNC_PAYLOAD_SIZE;
    EZRADIO_StartRx(0, dataRX.length);

    // Stay in this loop until SYNC slot is over
    do {
        EMU_EnterEM2(true);

        // Check if radio interrupt is received
        if (radioIrqReceived)
        {
            radioIrqReceived = false;

            // Check if packet is sent
            if (EZRADIO_CheckTransmitted())
            {
                clearLED(LED_TRANSMIT);
            }

            // Check if packet is received
            if (EZRADIO_CheckReceived(&dataRX))
            {
                // If SYNC packet is received, synchronize clock and retransmit
                if (dataRX.type == SYNC_DATA)
                {
                    // Transmit packet to all level 3 nodes in routing table
                    if (!transmitSync)
                    {
                        dataTX.dest_id = 0x03;
                dataTX.src_id = NODE_ID;
                dataTX.type = SYNC_DATA;
                dataTX.length = SYNC_PAYLOAD_SIZE;
                dataTX.payload.cur_time = dataRX.payload.cur_time;
                EZRADIO_StartTx(0, &dataTX);
                setLED(LED_TRANSMIT);
                transmitSync = true;
              }

              // Synchronize clock
              stopClock();
              setTime(dataRX.payload.cur_time, MIN_TX_DELAY_TIME +
      CLOCK_SYNC_DELAY);
              startClock();
            }
          }
        }
      } while (!slotOver);

#endif

  // Stop timer
  RTCDRV_StopTimer(syncTxOverTimerId);

  // Put the radio in sleep state until next TX/RX slot
  ezradio_change_state(
      EZRADIO_CMD_CHANGE_STATE_ARG_NEXT_STATE1_NEW_STATE_ENUM_SLEEP
      );
}

/**************************************************************
 * @brief Transmit a data packet over RF and listen for data packets
 **************************************************************/
void COMMUNICATION_Data_TRx(sensorData *sensor_data, uint8_t dc)
{
  uint16_t max_tx_data_delay = 0;
  uint16_t tx_data_delay = 0;
  radioPkt_data dataRX = {0};
  radioPkt_data dataTX = {0};

  // Obtain minimum duty cycle of neighboring nodes
  max_tx_data_delay = dc;

  for (int i = 0; i < MAX_NETWORK_NODES; i++)
  {
    if (nodeInfo_table[i].node == 0) {break;}

    if (nodeInfo_table[i].dc < max_tx_data_delay)
        {
          max_tx_data_delay = nodeInfo_table[i].dc;
        }
  }

  // Convert duty cycle to time in ms
  max_tx_data_delay = ((max_tx_data_delay * PARAM_DC_PERIOD) / 100) * 1000
          - SYNC_SLOT_PERIOD - MAX_SLOT_CLK_DRIFT;

  // Obtain a random time for the TX delay
  tx_data_delay = rand_time(MIN_TX_DELAY_TIME, max_tx_data_delay);

  // Start a timer to wait a random time before TX
  RTCDRV_StartTimer(dataTxDelayTimerId, rtcdrvTimerTypeOneshot, tx_data_delay,
      dataTxDelayCallback, NULL);

  // Start listening for packets with N bytes
  dataRX.length = NODE_PAYLOAD_SIZE;
  EZRADIO_StartRx(0, dataRX.length);

  // Stay in this loop until data slot is over
  do {
        EMU_EnterEM2(true);

        // Transmit after a random delay
        if (transmitData)
        {
          transmitData = false;
```

```

      // Stop timer
      RTCDRV_StopTimer(dataTxDelayTimerId);

      // Send DATA packet
      dataTX.dest_id = 0x02;
      dataTX.src_id = NODE_ID;
      dataTX.type = NODE_DATA;
      dataTX.length = NODE_PAYLOAD_SIZE;
      dataTX.payload.node_data.sensor_data.rhData = sensor_data->rhData;
      dataTX.payload.node_data.sensor_data.tempData = sensor_data->tempData;
      dataTX.payload.node_data.dc = dc;
      EZRADIO_StartTx(0, &dataTX);
      setLED(LED_TRANSMIT);
    }

    // Check if radio interrupt is received
    if ( radioIrqReceived )
    {
      radioIrqReceived = false ;

      // Check if packet is sent
      if (EZRADIO_CheckTransmitted())
      {
        clearLED(LED_TRANSMIT);
      }

      // Check if packet is received
      if (EZRADIO_CheckReceived(&dataRX))
      {
        if (dataRX.type == NODE_DATA)
        {
          // If new DC is received from another node, update in table
          uint8_t nodeInfo_next = 0;

          // Save duty cycles of other nodes
          for(nodeInfo_next=0; nodeInfo_next<MAX_NETWORK_NODES;
    nodeInfo_next++)
```

```
          {
            if (nodeInfo_table[nodeInfo_next].node == 0 || nodeInfo_table[
      nodeInfo_next].node == dataRX.src_id)
            {
              break;
            }
          }

          if (nodeInfo_next < MAX_NETWORK_NODES)
          {
            nodeInfo_table[nodeInfo_next].node = dataRX.src_id;
            nodeInfo_table[nodeInfo_next].dc = dataRX.payload.node_data.dc;
          }
        }
      }

      // Check if packet is received with CRC error
      if (EZRADIO_CheckCRCError())
      {
        printf ("Pkt rxd - CRC Error\n");
      }
    }
  } while (activeMode);
}
```

B.2 duty_cycling.c

```
/*****************************************************************
 * @brief Function to update the duty cycle
 *****************************************************************/
uint8_t DUTY_CYCLING_GetDC(float_t bat_level)
{
  float_t temp = 0.0;
  float_t op1 = 0.0;
  float_t op2 = 0.0;

```

```
// Obtain the actual battery level
B_cur = bat_level;

// Compute the parameter vector
op1 = PARAM_STEP_SIZE / vector_mult(feature_vector, feature_vector, 3);
op2 = B_cur − vector_mult(feature_vector, param_vector, 3);
temp = op1 * op2;

param_vector[0] = param_vector[0] + feature_vector[0]*temp;
if (param_vector[0] <= 0) param_vector[0] = INIT_Pvec_0;
param_vector[1] = param_vector[1] + feature_vector[1]*temp;
if (param_vector[1] >= 0) param_vector[1] = INIT_Pvec_1;
param_vector[2] = param_vector[2] + feature_vector[2]*temp;
if (param_vector[2] <= 0) param_vector[2] = INIT_Pvec_2;

// Compute the duty cycle
dc = (B_tgt − param_vector[0]*B_cur + param_vector[2]*B_tgt) / param_vector[1];
if (dc < PARAM_MIN_DC)    dc = PARAM_MIN_DC;
else if (dc > PARAM_MAX_DC) dc = PARAM_MAX_DC;

// Update the feature vector
feature_vector [0] = B_cur;
feature_vector [1] = dc;
feature_vector [2] = −B_tgt;

// Smooth the change of the duty cycle
dc_smooth = dc_smooth + PARAM_SMOOTHING*(dc − dc_smooth);

// Control the variance of the duty cycle
dc_real = PARAM_VARIANCE*dc + (1 − PARAM_VARIANCE)*dc_smooth;

dc_real = dc_real*100;

if (dc_real < APP_MIN_DC || dc_real > APP_MAX_DC)
{
   printf ("error: invalid duty cycle");
   return APP_MIN_DC;
}
```

```

  return dc_real;
}
```

B.3 energy_harvesting.c

```
/*******************************************************************
 * @brief Calculate the battery's energy for testing purposes
 *******************************************************************/
static void getTestBattEnergy(float_t dc)
{
  float_t P_harv = 0.0;
  float_t P_cons = 0.0;
  float_t E_tot = 0.0;

  // Obtain the harvested power from experimental values [mW]
  P_harv = (float_t)power_harvested[ph_index];
  ph_cnt++;
  if (ph_cnt >= 10) {
    ph_cnt = 0;
    ph_index++;
    if (ph_index >= 144) ph_index = 0;
  }

  // Adjust the harvested power with the charge efficiency of the battery
  P_harv *= BATT_CHARGE_EFFICIENCY;

  // Obtain the power consumption according to last duty cycle [mW]
  P_cons = dc*NODE_ACTIVE_PC + (1-dc)*NODE_SLEEP_PC;

  // Total energy gained (+) or consumed (-) during last minute [J]
  E_tot = ((P_harv - P_cons) * 60) / 1000;

  // New battery's energy level
  batt_energy += E_tot;

```

```c
  // Check boundaries of battery's energy level
  if(batt_energy > BATT_EFFECTIVE_ENERGY) batt_energy =
     BATT_EFFECTIVE_ENERGY;
  else if (batt_energy < 0) batt_energy = 0;
}


/****************************************************************
 * @brief Function to measure the current battery level
 ****************************************************************/
float_t ENERGY_HARVESTING_getBatteryLevel(uint8_t duty_cycle)
{
  uint32_t vBatt = 0;
  float_t lBatt = 0.0;

#if (USE_TEST_VALUES)
  // Use experimental values
  getTestBattEnergy((float_t)duty_cycle/100);

  // Convert battery energy to percentage level
  lBatt = batt_energy / BATT_EFFECTIVE_ENERGY;
#else
  // Sample the Vdd/3 to obtain the battery voltage
  adcConversionComplete = false;
  ADC_Start(ADC0, adcStartSingle);
  while (!adcConversionComplete) EMU_EnterEM1();
  vBatt = ADC_DataSingleGet(ADC0);

  // Convert battery voltage to percentage level
  lBatt = (float_t)(vBatt - MIN_BATTERY_LEVEL) / (MAX_BATTERY_LEVEL -
     MIN_BATTERY_LEVEL);
#endif

  return lBatt;
}
```

B.4 main.c

```c
/*********************************************************************
 * @brief  Main function
 *********************************************************************/
int main(void)
{
  I2CSPM_Init_TypeDef i2cInit = I2CSPM_INIT_DEFAULT;
  uint8_t duty_cycle = APP_INIT_DC;
  float_t battery = 0.0;
  sensorData sens_data = {0};
  sensorData avgSens_data = {0};

  // Chip errata
  CHIP_Init();

  // Configure HFRC frequency
  CMU_HFRCOBandSet(cmuHFRCOBand_28MHz);

#if (DEBUG_ENERGY)
  // Trace energy profile
  BSP_TraceSwoSetup();
#endif

  // Enable GPIO for sensor readings and LEDs
  gpioSetup();

  // Enable ADC for battery voltage readings
  adcInit();

#if (DEBUG_DISPLAY)
  // Initialize the display module
  DISPLAY_Init();
  RETARGET_TextDisplayInit();
  printf("\f");
#elif (DEBUG_USART)
  // Enable UART(VCOM) for testing
  RETARGET_SerialInit();
  RETARGET_SerialCrLf(1);
  printf("\f");
#endif

  // Initialize RTC
  RTCDRV_Init();

  // Initialize sensor bus
  I2CIDM_Init(&i2cInit);

  // Initialize radio and reset FIFOs
  EZRADIO_Init();
  EZRADIO_ResetTRxFifo();

  // Set up the seed for random TX times
  srand(NODE_ID);

  // Get initial sensor status
  if (!Si7013_Detect(i2cInit.port, SI7021_ADDR, NULL))
  { printf("Failed to detect sensor!!"); while(1); }

  // Initialize clock
  time_t network_time = COMMUNICATION_Init_Clk_Sync();
  if (network_time > 0)
  {
    setTime(network_time, CLOCK_INIT_DELAY);
  }
  printf("Clock set\n");

  // Set up RTC
  if (ECODE_EMDRV_RTCDRV_OK != RTCDRV_AllocateTimer(&
      clockUpdateTimerId))
  { printf("Failed to allocate RTC!!"); while(1); }
  if (ECODE_EMDRV_RTCDRV_OK != RTCDRV_StartTimer(clockUpdateTimerId,
      rtcdrvTimerTypePeriodic,
        CLOCK_UPDATE_MS, clockUpdateCallback, NULL))
  { printf("Failed to start RTC!!"); while(1); }


  // Main loop
  while (1)
  {
    // Node wakes up
    setLED(LED_ACTIVE);

    // Measure battery level
    battery = ENERGY_HARVESTING_getBatteryLevel(duty_cycle);

    // Check if there is enough battery to operate
    if (battery > 0)
    {
      // Update duty cycle
      duty_cycle = DUTY_CYCLING_GetDC(battery);
      active_period = (duty_cycle * PARAM_DC_PERIOD) / 100;

      // Turn the radio on
      EZRADIO_Restart();

      // Enter SYNC slot after each SYNC period
      if (syncUpdate)
      {
        syncUpdate = false;
        COMMUNICATION_Sync_TRx();
      }

      // Get sensor readings
      Si7013_MeasureRHAndTemp(i2cInit.port, SI7021_ADDR, &sens_data.rhData, &
    sens_data.tempData);

      // If duty cycle is very low aggregate data, only transmit when it's above a
    threshold
      if (duty_cycle < APP_MIN_DC_TX)
      {
        aggregateData(sens_data.rhData, sens_data.tempData, &avgSens_data);
        active_period = 0;
      }
```

```
        else
        {
          if (saved_data)
          {
            aggregateData(sens_data.rhData, sens_data.tempData, &avgSens_data);
            sens_data.rhData = avgSens_data.rhData;
            sens_data.tempData = avgSens_data.tempData;
            saved_data = false;
          }

          // Transmit node's data
          COMMUNICATION_Data_TRx(&sens_data, duty_cycle);
        }

        // Turn the radio off
        EZRADIO_Shutdown();
      }
      else
      {
        duty_cycle = 0;
        active_period = 0;
        EZRADIO_Shutdown();
      }

      // Node goes to sleep
      clearLED(LED_ACTIVE);
      do {
        EMU_EnterEM2(true);
      } while (!activeMode);
    }
}
```

B.5 radio.c

```
/**************************************************************************
 * @brief Set radio to TX mode, switch to RX when done
 *******************************************************************/
bool EZRADIO_StartTx(uint8_t channel, radioPkt_data *data)
{
  ezradio_request_device_state();

   if (ezradioReply.REQUEST_DEVICE_STATE.CURR_STATE ==
      EZRADIO_CMD_REQUEST_DEVICE_STATE_
  REP_CURR_STATE_MAIN_STATE_ENUM_TX) {
     return false;
  }

  // Build packet structure
  *(uint8_t *)(radioPkt + APP_PKT_MATCH_LOC) = data->dest_id;
  *(uint8_t *)(radioPkt + APP_PKT_LEN_LOC) = data->length;
  *(uint8_t *)(radioPkt + APP_PKT_SRCID_LOC) = data->src_id;
  *(uint8_t *)(radioPkt + APP_PKT_TYPE_LOC) = data->type;

  switch (data->type)
  {
  case NODE_DATA:
    *(uint32_t *)(radioPkt + APP_PKT_DATA_LOC) = data->payload.node_data.
     sensor_data.rhData;
    *(uint32_t *)(radioPkt + APP_PKT_DATA_LOC + 4) = data->payload.
     node_data.sensor_data.tempData;
    *(uint8_t *)(radioPkt + APP_PKT_DATA_LOC + 8) = data->payload.node_data
     .dc;
    break;
  case SYNC_DATA:
    *(uint32_t *)(radioPkt + APP_PKT_DATA_LOC) = data->payload.cur_time;
    break;
   default:
    break;
  }

  // Fill the TX FIFO with data
  ezradio_write_tx_fifo(data->length, (uint8_t *)radioPkt);

  // Start sending packet, START immediately, Packet n bytes long, go to RX when
     done ****change when want to retransmit
  ezradio_start_tx(channel, 0x80, data->length);

  return true;
}


/***************************************************************************
 * @brief Check if Packet received IT flag is pending
 ***************************************************************************/
bool EZRADIO_CheckReceived(radioPkt_data *data)
{
  if ( false == ezradio_hal_NirqLevel())
  {
    // Read ITs, clear pending ones
    ezradio_get_int_status(~(
     EZRADIO_CMD_GET_INT_STATUS_REP_PH_PEND_PACKET_RX_PEND_BIT
     ), 0u, 0u);

    // Check the reason for the IT
    if (ezradioReply.GET_INT_STATUS.PH_PEND &
     EZRADIO_CMD_GET_INT_STATUS_REP_PH_PEND_PACKET_RX_PEND_BIT
     )
    {
      // Read the data from RX FIFO
      ezradio_read_rx_fifo(data->length, (uint8_t *) &radioPkt[0u]);

      // Read out individual  fields
      //data->length = *(uint8_t *)(radioPkt + 1u);
      data->src_id = *(uint8_t *)(radioPkt + 2u);
      data->type = *(uint8_t *)(radioPkt + 3u);

      switch (data->type)
        {
        case NODE_DATA:
            data->payload.node_data.sensor_data.rhData = *(uint32_t *)(radioPkt +
     APP_PKT_DATA_LOC);
```

```
                data->payload.node_data.sensor_data.tempData = *(uint32_t *)(radioPkt
    + APP_PKT_DATA_LOC + 4u);
                data->payload.node_data.dc = *(uint8_t *)(radioPkt +
    APP_PKT_DATA_LOC + 8u);
              break;
            case SYNC_DATA:
                data->payload.cur_time = *(uint32_t *)(radioPkt +
    APP_PKT_DATA_LOC);
              break;
            default :
              break;
            }

          return true;
        }

        // Reset FIFO
        ezradio_fifo_info(EZRADIO_CMD_FIFO_INFO_ARG_FIFO_RX_BIT);
      }

    return false;
}
```

Bibliography

[1] G. Anastasi, M. Conti, M. D. Francesco, and A. Passarella, "Energy conservation in wireless sensor networks: A survey," *Ad Hoc Networks*, vol. 7, no. 3, pp. 537 – 568, 2009.

[2] P. Baronti, P. Pillai, V. W. Chook, S. Chessa, A. Gotta, and Y. F. Hu, "Wireless sensor networks: A survey on the state of the art and the 802.15.4 and zigbee standards," *Computer Communications*, vol. 30, no. 7, pp. 1655 – 1695, 2007. Wired/Wireless Internet Communications.

[3] Silicon Labs, *User Manual EZR32LG 868MHz Wireless Starter Kit*, 2015.

[4] Silicon Labs, *EZR32LG Reference Manual*, 2015.

[5] Silicon Labs, *AN633: Programming Guide for EZRadioPRO Si4x6x Devices*, 2014.

[6] "Energy harvesting brochure." [Online]. Available: http://www.we-online.com/harvest.

[7] Energizer, *ENERGIZER NH15-2300 Data Sheet.*

[8] C. Vigorito, D. Ganesan, and A. Barto, "Adaptive control of duty cycling in energy-harvesting wireless sensor networks," in *Sensor, Mesh and Ad Hoc Communications and Networks, 2007. SECON '07. 4th Annual IEEE Communications Society Conference on*, pp. 21–30, June 2007.

[9] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks*, vol. 38, no. 4, pp. 393 – 422, 2002.

[10] G. Pottie and W. Kaiser, "Wireless integrated network sensors (wins): Principles and practice," *CACM'00*, 2000.

[11] Silicon Labs, *Si4464/63/61/60*, 2012.

[12] J. Polastre, J. Hill, and D. Culler, "Versatile low power media access for wireless sensor networks," in *Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems*, SenSys '04, (New York, NY, USA), pp. 95–107, ACM, 2004.

[13] W. Ye, J. Heidemann, and D. Estrin, "Medium access control with coordinated adaptive sleeping for wireless sensor networks," *Networking, IEEE/ACM Transactions on*, vol. 12, no. 3, pp. 493–506, 2004.

[14] T. Van Dam and K. Langendoen, "An adaptive energy-efficient mac protocol for wireless sensor networks," in *Proceedings of the 1st international conference on Embedded networked sensor systems*, pp. 171–180, ACM, 2003.

[15] G. Lu, B. Krishnamachari, and C. S. Raghavendra, "An adaptive energy-efficient and low-latency mac for data gathering in wireless sensor networks," in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, p. 224, IEEE, 2004.

[16] A. Kansal, J. Hsu, S. Zahedi, and M. B. Srivastava, "Power management in energy harvesting sensor networks," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 6, no. 4, p. 32, 2007.

[17] J. M. Gilbert and F. Balouchi, "Comparison of energy harvesting systems for wireless sensor networks," *international journal of automation and computing*, vol. 5, no. 4, pp. 334–347, 2008.

[18] S. Sudevalayam and P. Kulkarni, "Energy harvesting sensor nodes: Survey and implications," *Communications Surveys Tutorials, IEEE*, vol. 13, pp. 443–461, Third 2011.

[19] A. Kansal, J. Hsu, M. Srivastava, and V. Raqhunathan, "Harvesting aware power management for sensor networks," in *Design Automation Conference, 2006 43rd ACM/IEEE*, pp. 651–656, 2006.

[20] H. Kwon, D. Noh, J. Kim, J. Lee, D. Lee, and H. Shin, "Low-latency routing for energy-harvesting sensor networks," in *Ubiquitous Intelligence and Computing*, pp. 422–433, Springer, 2007.

[21] T. Cutler, "Implementing zigbee wireless mesh networking," July 2005.

[22] A. Dunkels and J. Vasseur, "Ip for smart objects," July 2010.

[23] N. Khalil, M. Abid, D. Benhaddou, and M. Gerndt, "Wireless sensors networks for internet of things," in *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014 IEEE Ninth International Conference on*, pp. 1–6, April 2014.

[24] A. Kouche, "Towards a wireless sensor network platform for the internet of things: Sprouts wsn platform," in *Communications (ICC), 2012 IEEE International Conference on*, pp. 632–636, June 2012.

[25] Silicon Labs, *AN625: Si446x API Descriptions*, 2012.

[26] Silicon Labs, *AN626: Packet Handler Operation for Si446x RFICs*, 2013.

[27] Linear Technology Corporation, *Demo Manual DC2080A*, 2004.

[28] "Nimh battery charging basics." [Online], May 2015. Available: http://www.powerstream.com/NiMH.htm.

[29] Energizer, *Nickel Metal Hydride (NiMH) Handbook and Application Manual*.

[30] O. Dousse, P. Mannersalo, and P. Thiran, "Latency of wireless sensor networks with uncoordinated power saving mechanisms," in *Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing*, pp. 109–120, ACM, 2004.

[31] Silicon Labs, *Crystal Oscillator (XO) 100 KHz to 250 MHz*, 2013.

[32] A. Kansal, D. Potter, and M. B. Srivastava, "Performance aware tasking for environmentally powered sensor networks," *ACM SIGMETRICS Performance Evaluation Review*, vol. 32, no. 1, pp. 223–234, 2004.

[33] Atmel, *Range Calculation for 300 MHz to 1000 MHz Communication Systems*, 2010.