Term project Fall 2018

Embedded system for current measurement from solar panels

Øystein Molvik

Department of Engineering Cybernetics
Norwegian University of Science and Technology

Summary

All across the world, the usage of sensorsystems, or smart sensors, have skyrocketed. As the increasingly changing climate becomes more of an issue, the use of solar panels have increased. This project aims to develop an embedded systems with support for multiple communication protocols, that measures the power from a solar panel, and transmit the data to a back-end application for logging.

The embedded system will utilize LoRaWAN, and Cellular 3G (to some extent). The design and implementation of most of the hardware is covered by another project [1], while the software design and implementation, as well as integration with a back-end application, is covered here.

Table of Contents

List of Tables

List of Figures

Abbreviations

OSI	=	Open Systems Interconnection
GUI	=	Graphical User Interface
ADC	=	Analog to Digital Converter
EUI	=	Extended Unique Identifier
LDO	=	Low-dropout regulator
TTN	=	The Things Network

Chapter 1

Introduction

1.1 Background for the task

During the last years, the terms IoT and smart grids have increased extensively in popularity and became so-called ”buzz words”. The use of sensors, if it’s in one’s own home, company or even cities, has skyrocketed and is becoming more relevant and important by the day. To this very day, there are more than 25 billion connected devices around the world. IoT devices are increasingly being used for aspects like ”smart home”, ”smart city” or even ”smart industry”. The idea of ”smart” is to indicate what you do with the information the sensor gathers, which opened an entirely new field of usage for the sensor data. The climate problem has become an increasingly more relevant topic in today’s media, more and more companies, even individual households, are reverting to renewable energy sources such as solar panels. Due to this, a smart power measurement device, able to use different communication protocols, could become quite attractive.

1.2 Motivation

The main purpose of this device is to gather information about one’s power consumption. To keep it general, it should be able for anyone to adapt the device with the establishment. The device will, therefore, be supporting two different communication protocols. An establishment that doesn’t have coverage for one protocol, could use a different one. This would create better flexibility and make it more attractive. Also, a futuristic possibility with multiple single devices could be to create a micro-grid of devices. This grid could then attempt to predict the flow of skies to anticipate a decrease in production, to off-load to save power in case of small batteries.

1.3 Limitations

For this project, there are certain limitations. For starters, to measure the current and voltage from a solar panel, a specific sensor will be used. Namely the HO 10-P current transducer [5] for current and plain cables for voltage.
Another limitation is to which communication protocols this project will focus on. As the main protocol has not yet been decided, the other one will be Cellular 3G. The Xbee Cellular 3G modem ([6] and [7]), will get a place on the embedded system being created.

Chapter 2

Literature Review

This chapter highlights the findings that are relevant to the task. The considerations being that the task is an embedded system, capable of deployment anywhere in Trøndelag.

2.1 Coverage

For this project coverage is the most essential aspect to consider when choosing a good protocol.
With regards to LoRa, the public coverage around the country is not that widespread. For a LoRa node to function, it needs a LoRaWAN gateway that receives its messages. So far there are only a small amount of them placed around the country. Considering the county of Trøndelag, only Trondheim has any coverage at all. It's shown in figure 2.1, taken from TTN's website [8]. A particularly good property of LoRa is the adaptability and mobility of the nodes by just setting up a new gateway where coverage is required [2]. Comparing the coverage of cellular bands and LoRa, two companies in Norway have

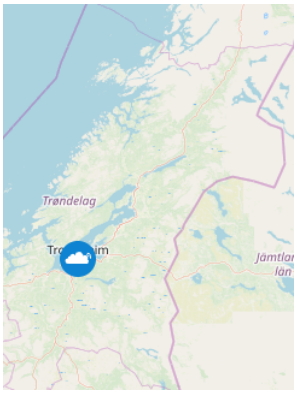
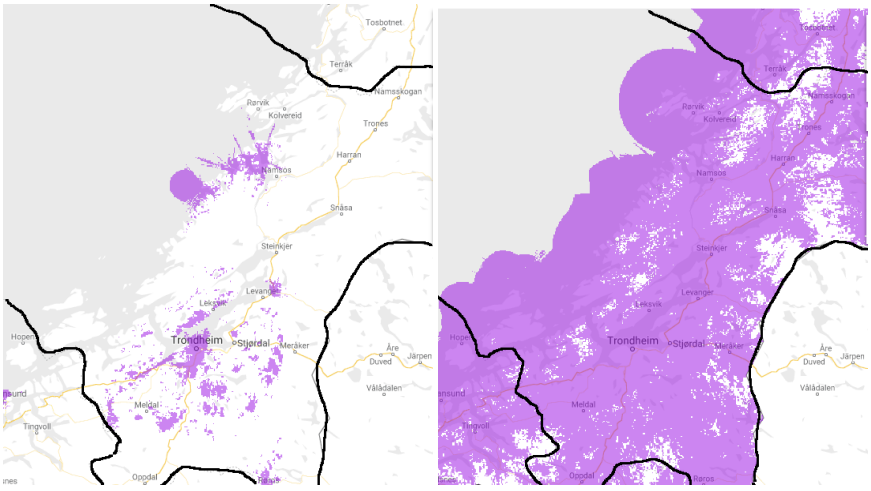


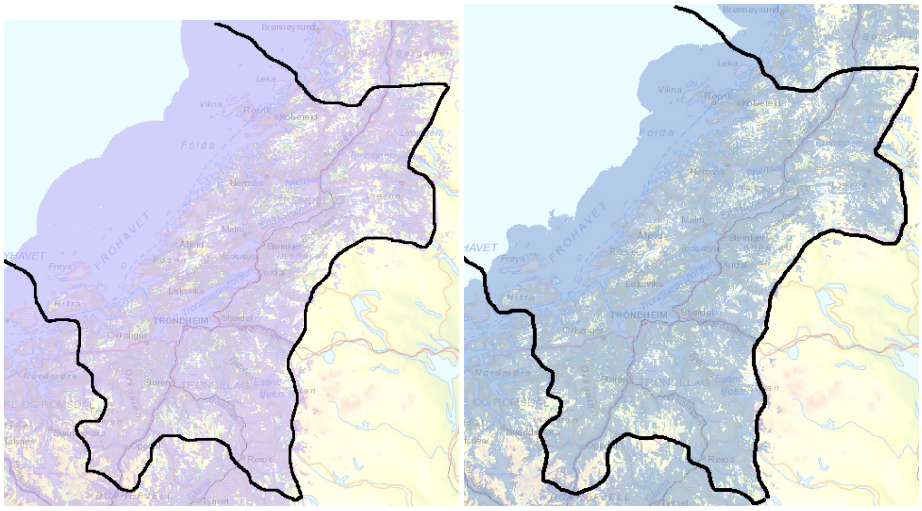
Figure 2.1: Map of LoRaWAN gateways in Trønderlag, illustrated by the blue and white icon.

approximate the same coverage and provide the best option of LTE-M and Nb-IoT, Telia and Telenor. Their coverage of the relevant technologies are shown in figure 2.2 and 2.3. In the battle of best coverage, cellular bands in Trøndelag is a clear winner over LoRa. Nevertheless, the disadvantage of cellular communication is increasing their coverage as they use built cell towers. Only the respective company are allowed to build new ones to expand their coverage. The cost to build a new tower compared to installing a new gateway is significantly higher, and shown in table 2.2. From the figures about coverage, the areas lacking are mainly rural ones. A significant advantage of the LoRaWAN is its flexibility. The only requirement to expand its coverage is installing a new gateway around the desired area. From table 3 in R. Balani's "Energy Consumption Analysis for Bluetooth, WiFi and Cellular Networks" [2] paper, shown in figure 2.5, the operating range for LoRa is between 5-15 kilometers. Therefore not many gateways are required to cover a very large area. For the use of WiFi, the sole requirement is that there is already a WiFi installed at the establishment, that the sensor can connect to.



(a) LTE-M (b) NB-IoT

Figure 2.2: Telia's coverage of IoT protocols in Trøndelag [4].



(a) LTE-M (b) NB-IoT

Figure 2.3: Telenor's coverage of IoT protocols in Trøndelag [4].

2.2 Low power communication protocol

In the search for the most efficient protocol to use in communicating with the device, multiple protocols were considered as explained in the introduction. Those were LoRaWAN, Wifi, LTE-M, and NB-IoT. From M. S. Mahmoud and A. H. Mohamad's paper [2], research has been done to compare different long and short-range connectivity technologies. For this task, "long-range" is an important keyword for optimal availability. Nevertheless, WiFi is still a viable option as most people have WiFi at home, and even at their cabins.
Looking at figure 2.4, multiple low power Wifi modules researched [2], are illustrated with their power consumption. They all have a very high maximum bit rate, varying from 10-54 Mb/S, which is very good for applications with high required throughput. For most of the modules, the Tx and Rx current consumption ranges from 19-250 mA. LoRaWAN modules on the other hand, vary from 20-40 mA, shown in figure 2.5.
LTE-M and NB-IoT differ in several ways, where the most relevant one is that LTE-M is designed for applications where the QoS is highly important, as well as the mobility of being at different locations. This is realized through LTE-M using most of the LTE band, while NB-IoT only uses a subset, making it more power-efficient. This makes NB-IoT a very good choice for static devices, like sensors for simple measurements. From research [9], they compared these two technologies with regards to technical features as well as the more important IoT factors:

1. Quality of Service
2. Battery life & latency
3. Network coverage & range

4. Deployment Model
5. Cost

For this project, the most important factors to consider are battery life & latency and network coverage & range. QoS is not considered too important because the fact of losing a packet on occasion does not harm the entire solution. It [2] states that both NB-IoT and LoRa have their advantages and disadvantages, as LoRaWAN used less power due to it being an asynchronous, ALOHA-based protocol. NB-IoT uses a subset of the LTE band, and therefore requires a regular, but infrequent, synchronization which consumes extra energy. NB-IoT has, in general, better coverage, but the cost of building new stations is immense compared to setting up a new LoRa gateway. This is illustrated in table 2.1 and 2.2.

Table 2.1: Current consumption and latency between LoRa and NB-IoT [2].

	Peak current	Sleep current	Latency
LoRa	32 mA	$1\mu A$	Insensitive to latency
NB-IoT	120/130 mA	$5\mu A$	$< 10s$

Table 2.2: Different cost of LoRa and NB-IoT [2].

	Spectrum cost	Network & Deployment cost
LoRa	Free	\$100-\$/gateway
NB-IoT	>\$500 million/MHz	\$15000/base station

Company	Module	IEEE Protocol	V_{DD} (Volt)	I_{TX} (mA)	I_{Rx} (mA)	I_{sleep} (µA)	Max. Bit Rate (Mb/S)
Microchip [9]	RN171	802.11 b/g	3.3	190	40	4	54
Qual Comm [13]	QCA4004	802.11 n	3.3	250	75	130	10
Gain Span [14]	GS1011M	802.11 b	3.3	150	40	150	11
G2 Microsystem [15]	G2M5477	802.11 b/g	3.3	212	37.8	4	11
Redpine [16]	RS9110-N-11-02	802.11 b/g/n	3.3	19	17	520	11
RTX [17]	RTX41x Series	802.11 b/g/n	3.3	0.760	0.760	3	10

Figure 2.4: Table illustrating power consumption with different low power Wifi modules[2].

Table 3. Particular comparison for different LoRaWAN modules.

Company	Module	IEEE Protocol	Designed for Network Protocols	V_{DD} (Volt)	I_{TX} (mA)	I_{Rx} (mA)	I_{sleep} (µA)	Bit Rate (Kb/S)	Operation Range Km
Microchip [25]	RN2483	Close Alignment with IEEE 802.15.4	LoRaWAN™ Protocol Stack	3.3	38.9* 32.9**	14.2	9.9	5.468 - 300	5 - 15 Km
Multiteh [26]	MTDOT-868-X1-SMA	Close Alignment with IEEE 802.15.4	LoRaWAN™ Protocol Stack	3.3	26 - 41	12	30.9	5.47 - 21.9	Up to 8 km
Nemeus [27]	Nemeus-MM002	Close Alignment with IEEE 802.15.4	LoRaWAN™ Protocol Stack	3.3	20 - 39.5	11.7	<2	0.3 - 40	12 Km line of sight

*Maximum transmitted power and 868MHz band. **This value for Maximum transmitted power and 433MHz band.

Figure 2.5: Table illustrating power consumption with different LoRaWAN modules[2].

Chapter 3

Theory

This chapter identifies and explains the theory required for understanding the basic concepts required for this project, mainly in regards to LoRaWAN.

3.1 LoRaWAN

LoRaWAN is abbreviated from ”Long Range Wide Area Network” and is often mistaken for LoRa. The difference is that LoRa, compared to the OSI model, is only the physical layer, while LoRaWAN adds the data-link layer and network layer. LoRaWAN is a cloud-based MAC-layer but acts as a network layer to manage communication between the LoRaWAN gateways and end-nodes.

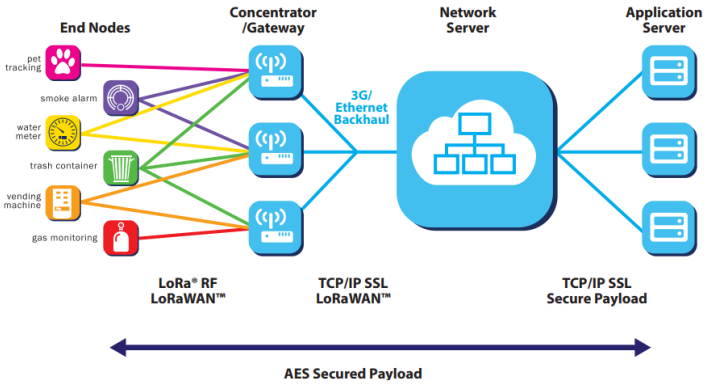


Figure 3.1: LoRaWAN architecture

3.1.1 Classes

End-devices all serve different purposes and tasks. To optimize a variety of application profiles, LoRaWAN utilizes different device classes, namely classes A, B, and C.

Class A

This class is meant for all devices where battery life is of utmost importance, while not needing a full-duplex communication. This class only allows downlink messages when an uplink message has been sent, meaning the sensor can only receive a message after itself has sent one. There are specific receive windows for this, RX1 and RX2, shown in figure 3.2. The windows are static and determined by the network. With TTN the first window is 1 second after the transmission, TX+1s, while the other on is TX+2s. It's important to note that the device cannot be reached by the back-end if a message has not been sent. Therefore the messages will be queued.

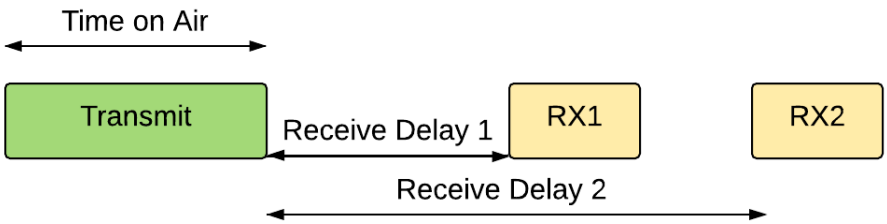


Figure 3.2: Simple illustration of LoRaWAN receive windows.

Class B

Class B acts as a beacon with the same functionality as with class A but adding the possibility of scheduled downlink windows. This makes the device reachable at specific scheduled times as well as after each transmission. This class is suited more for less battery hungry devices which needs some more communication with the back-end.

Class C

Class C is continuous and is suited for devices that require the least amount of latency for received messages. This class has an always-open downlink window, meaning the device is always listening for downlink transmissions.

3.1.2 Physical layer

Bandwidth

LoRaWAN utilizes a few variations of bandwidth. This is because of different regulations around the world. The most common bandwidths and their locations are illustrated in table 3.1.

Bandwidth	Region
779-787MHz and 470-510MHz	China
863-870MHz and 433.05-434.79MHz	Europe
902-928MHz	United States

Table 3.1: Standard bandwidths with location.

DataRate	Configuration	Indicative physical bit rate [bit/s]
0	LoRa: SF12 / 125 kHz	250
1	LoRa: SF11 / 125 kHz	440
2	LoRa: SF10 / 125 kHz	980
3	LoRa: SF9 / 125 kHz	1760
4	LoRa: SF8 / 125 kHz	3125
5	LoRa: SF7 / 125 kHz	5470
6	LoRa: SF7 / 125 kHz	11 000
7	LoRa: FSK	50 000

Table 3.2: Available data rates for LoRa end-devices, stated in LoRaWAN Regional Parameters [3], p. 16.

Data-Rate

The data rate is a combination of two factors, namely the bandwidth and the spreading factor. When speaking about transmitting data, there are multiple data rates available to be used. They depend on the channel in which the data is transmitted. The channels have a bandwidth of either 125 kHz, 250 kHz or 500 kHz. This depends on what type of Class the device supports, but for all classes, there exist 6 different data rates. To save power, it's recommended to utilize the highest data rate. This setting has, on the other hand, a considerably lower range than the lowest data range, so there is a trade-off between power and range. Table 3.2 shows the different data rates available, which are taken from the LoRa documentation on regional parameters [3], p. 16.

3.1.3 Messages

There are two types of messages used in LoRa communication, downlinks and uplinks. Uplinks are messages from the end-device, while downlinks are messages to the end-device. Each message either contains a payload or a join request/response. The payload size is based on which data rate used for the current device. From table 3.2, data rate 0-6 has a payload maximum size of 255 bytes, while data rate 7 has a maximum payload of 64 bytes (see RN2483 Command Reference [**?**]).

3.1.4 Security

Encryption

As the security part of LoRaWAN is not too relevant for the project, it will not be explained in this project. It is worth mentioning that all messages used in LoRa transfers are encrypted with AES 128-bit encryption.

Join procedures

To secure radio transmissions, the LoRaWAN protocol relies on symmetric cryptography using two different session keys. These keys are:

- Unique 128-bit Network Session Key (nwkSKey) share between the network server and end-device.
- Unique 128-bit application session key (appSKey) shared end-to-end at the application level.

When a device is requesting to join a network server via LoRaWAN, there are two methods to achieve this, either ”Activation by personalization (ABP)” or ”Over the air activation (OTAA)”.

To join the network using ABP, the keys nwkSKey and appSKey are generated in advance and hard-coded into the devices. This is a less secure way to connect to a network compared to OTAA. Using OTAA, two 8 byte EUI's and an application key (appKey) are generated in advance and hard-coded into the device. The procedure for joining over OTAA is shown in figure 3.3. The important part is that the device and server negotiate the session keys which may result in a device being denied. Opposed to ABP where the device is automatically joined and no procedure is necessary.

3.2 The Things network

The Things Network (TTN) is a public global network for IoT devices and gateways, using LoRa only. Due to this being a public open network, anyone anywhere in the world can contribute to expanding the network.

3.2.1 TTN Console

The TTN-console is a tool for developers or anyone working with LoRa IoT devices can manage their devices and applications. Essentially the console is an online GUI. As mentioned earlier in section 3.1.4, the keys require in OTAA joining are auto-generated when adding a device to an application in this console.

3.2.2 Devices

Devices in this context involve and end-node/device which is connected to the network and publishing information to it. In IoT, this can be anything and everything. If the device

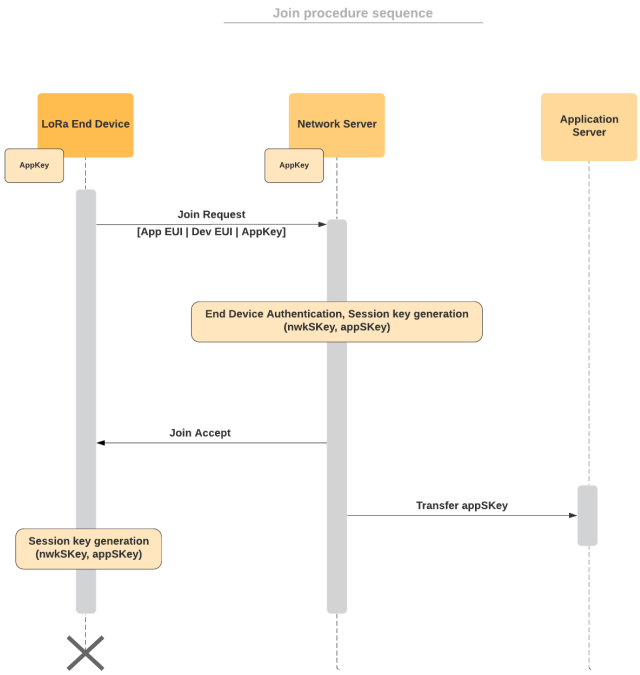


Figure 3.3: Join procedure sequence diagram

contains information that can be shared and used, and connected to a LoRa network, it's an end-device.

3.2.3 Gateways

A gateway is considered a "bridge" between the network and end-node. Nevertheless, the gateway and devices don't "know each other", in a way that the device just broadcasts a message to any gateway that can receive it (based on distance). The gateway will receive the message and route it to the correct network. The gateway will also listen to the TTN-handler in case it has enqueued a downlink message for a specific device. This queue is very important considering some end-devices cannot be reached at any time (see section [3.1.1]).

3.2.4 Application

Application is the back-end software that can be utilized to receive the information the end-node sends. TTN refers to this as an application server and provides two interfaces the application can use to interact with TTN, either HTTP or MQTT.

SDK's

To merge one's back-end application to TTN to receive uplinks, TTN has provided multiple high-level programming languages:

- Go
- Java
- Node-Red
- Node js
- Python

These libraries allow easy setup of remote MQTT-clients for developers who desire to integrate their application with TTN.

Chapter 4

Specification and Design

The main idea of the task at hand, based on findings from the literature, is to create an embedded system that communicates with a back-end application for data storage.
To realize this task, the current sensor at hand [5], had to be integrated with the rest of the hardware chosen for this project. In addition, it's required to have an application/server to store all messages sent from the end node. Even so, it's important to note that all the hardware used in this project, except the sensor [5], was outsourced to a specialization topic [1]. Consequently, mainly the software aspect of the project will be explained furthest into detail, as the hardware aspect can be read about in the aforementioned report [1].

4.1 Functional Specifications

These are the functional specifications that are relevant for a finished device to function properly:

1. Wireless unit using a battery.
2. Powered by micro-USB cable.
3. Be charged using power from the solar panel.
4. Be able to send AND receive messages wirelessly.
5. Unit must be programmable and be debugged.
6. Unit must be able to be reset locally (button), and wirelessly.
7. Low power consumption.
8. Take input from a sensor.
9. Be able to switch communication protocol.

10. A back-end application for storing and logging the data from the device.
11. USB connection for print feedback in the field.
12. Easily connected to a computer for live debugging.

4.2 Technical Specifications

These are the relevant technical specifications required to realize the functional specifications. A complete list of all technical specifications for this entire project, hardware included, with an association to a functional specification have been derived in the following list:

1. A battery slot for 18650 lithium rechargeable batteries. (see list 4.1 pt. 1).
2. A charging circuit for battery (see list 4.1 pt. 1).
3. A stable power supply of 5V/12V/24V (see list 4.1 pt. 2 and 3).
4. When using LoRa, a chip which is compatible with the European standards (863-870MHz) (see list 4.1 pt. 4).
5. When using LoRa, an antenna compatible with the selected LoRa Chip (see list 4.1 pt. 4).
6. When using LoRa, able to transmit an encoded payload and react to a potential downlink. (see list 4.1 pt. 4 and 6).
7. A header for connecting sensor (see list 4.1 pt. 8).
8. A MCU with an ADC, multiple UARTs and SPI interfaces (see list 4.1 pt. 4, 9 and 11).
9. A MCU able to enter low power consumption modes (see list 4.1 pt. 7).
10. A PCB with the possibility of switching off power for sensor and communication chip (see list 4.1 pt. 7).
11. A header for JTAG interface to update software and debug device (see list 4.1 pt. 5).
12. An UART header for ”printf” debugging (see list 4.1 pt. 12).
13. A reset button connected to all required components (see list 4.1 pt. 6).
14. A general header fit for multiple communication chips (see list 4.1 pt. 9).
15. A back-end application receiving messages from the end-device (see list 4.1 pt. 10).

4.3 Acceptence Criteria

The list for the minimal amount of specifications the device has to pass:

1. Able to be powered by a 18650 battery (see tech. spec. pt. 1).
2. Possibility of fully charging said 18650 battery within 12 hours. (see tech. spec. pt. 1 and 2).
3. Must be able to send an encoded version of a current measurement from HO 10-P over two communication protocols to a back-end application and log the decoded version (see tech. spec. pt. 4, 5, 6, 7, 8, 13 and 14).
4. The option to use and charge the 18650 battery using the power from the solar panel (see tech. spec. pt. 1, 2 and 3).
5. Device must be able to sleep when not active (see tech. spec. pt. 9).
6. Be able to shut off power to the communication chip and sensor before entering sleep (see tech. spec. pt. 10).
7. The entire system must reset when pushing the reset button (see tech. spec. pt. 12).
8. Must be able to receive commands (Ex. from The Things Network), to change data rate, synchronize and reset (see tech. spec. pt. 4, 5 and 6).
9. JTAG interface to update the software and debug the device (see tech. spec. pt. 11).
10. Back-end application to serve the end-node seamlessly as long as it's running.
11. Must be able to connect a computer to the device for live print debugging (see tech. spec. pt. 12).

4.4 Design

This section will be devoted to explaining how the entire system has been designed, explaining everything from the sensor measurement to the log file via the application used. Due to the hardware part being outsourced to the specialization topic [1], that part will only be a short summary.

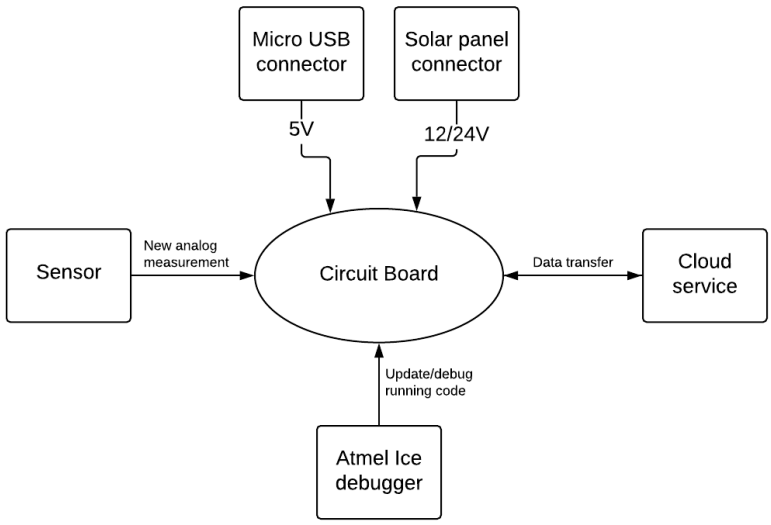


Figure 4.1: Hardware context diagram [1].

4.4.1 Hardware

The hardware communicates with the environment in the ways described in figure 4.1. The device uses a built-in ADC to convert the measurement from the sensor do a digital value which can be processed further. The device has a micro USB connector, from where a 5V when not in the field, or a 12/24V from the solar panel, can be supplied to both power the system and recharge the battery. To ensure a long-lasting battery life without superfluous removal from its installed location for recharging, it's important that it's capable of drawing power from the solar panel.
To view the different modules as well as a more in-depth view of the hardware design, the inner level analysis in figure 4.2 and 4.3 illustrate the idea.

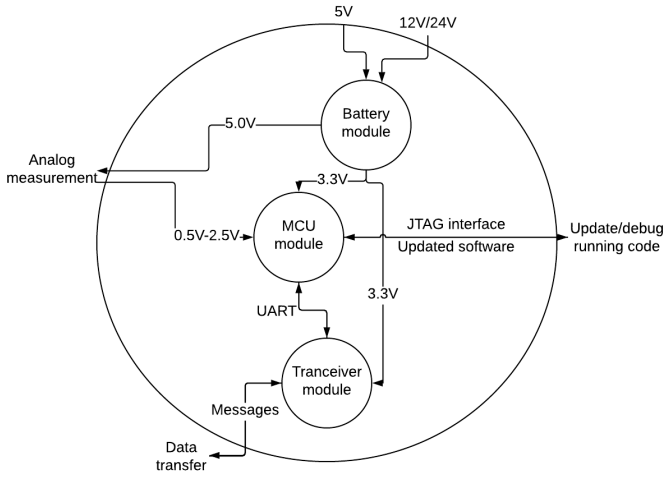


Figure 4.2: Hardware inner analysis level 1 [1].

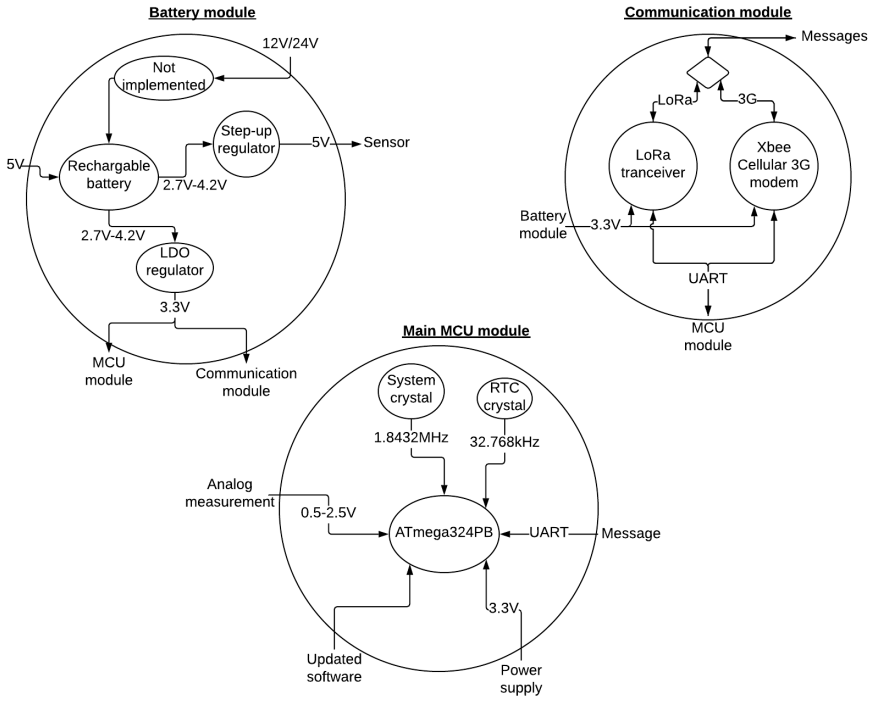


Figure 4.3: Hardware inner analysis level 2 [1].

4.4.2 Software

Comparing the context diagram from the hardware section, the diagram for software, shown in figure 4.5, there are several similarities. The main objective of the software, in this case, is to process the received analog data and transfer it to the back-end application. As mentioned earlier, the LoRa technology is the most optimal when only sending data as bytes, where each byte is a value between 0x00 to 0xFF. Therefore the desired payload needs to be encoded. The current measurement is a floating-point value, and must, therefore, be converted. All the encoding is further explained below.

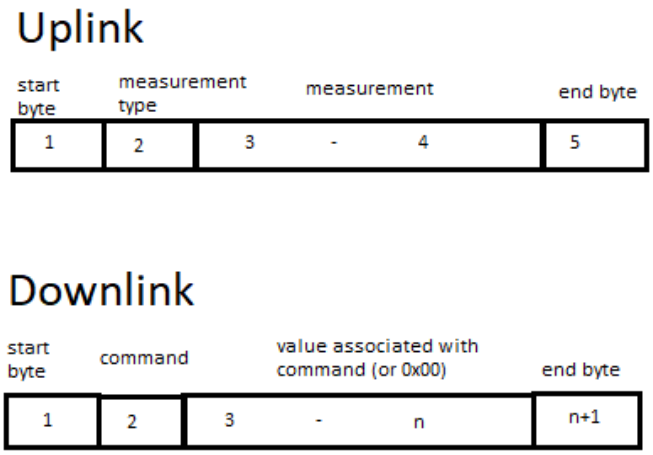


Figure 4.4: Frame used to encode messages to and from device.

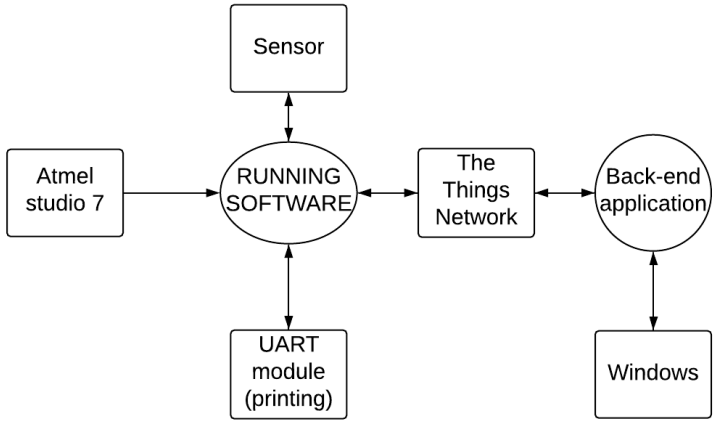


Figure 4.5: Software context diagram [1].

Architecture

The architecture of this software system will be timer-driven. In addition, the main program will be implemented as a state machine and alter between states based on different parameters. The timer-triggered events will be a vital part of changing states in the system and will be further explained in chapter 5.

Encoding

To maintain the possibility of scaling, the messages will be encoded in the way illustrated in figure 4.4. This helps keep the transfers at a minimum size to prevent unnecessary energy consumption. The floating-point measurement of the payload is converted to a two-byte hexadecimal number, using the IEEE 754 Half-precision floating-point format. The measurement type is added to keep scalability relevant while giving the option of multiple types of sensors. The rest of the important data, like device ID, is listed in the metadata from TTN and is therefore not added to the frame.

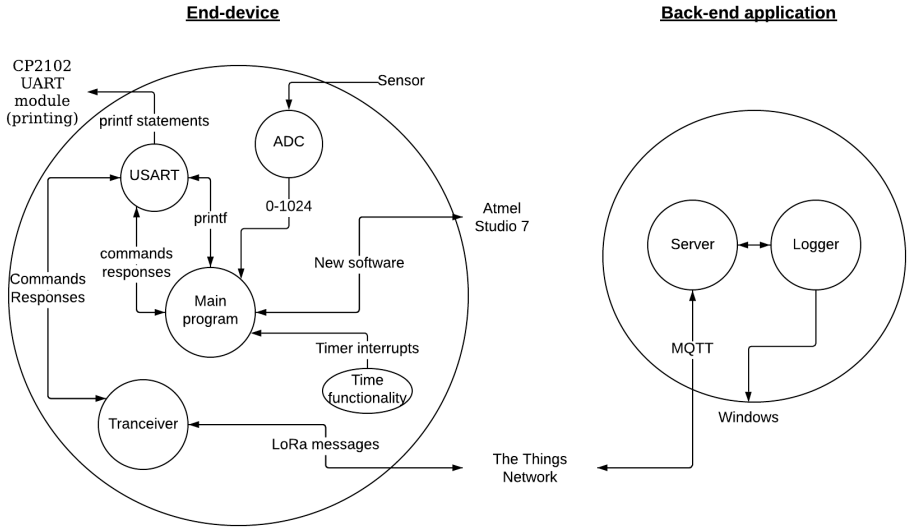


Figure 4.6: Software inner analysis level 1.

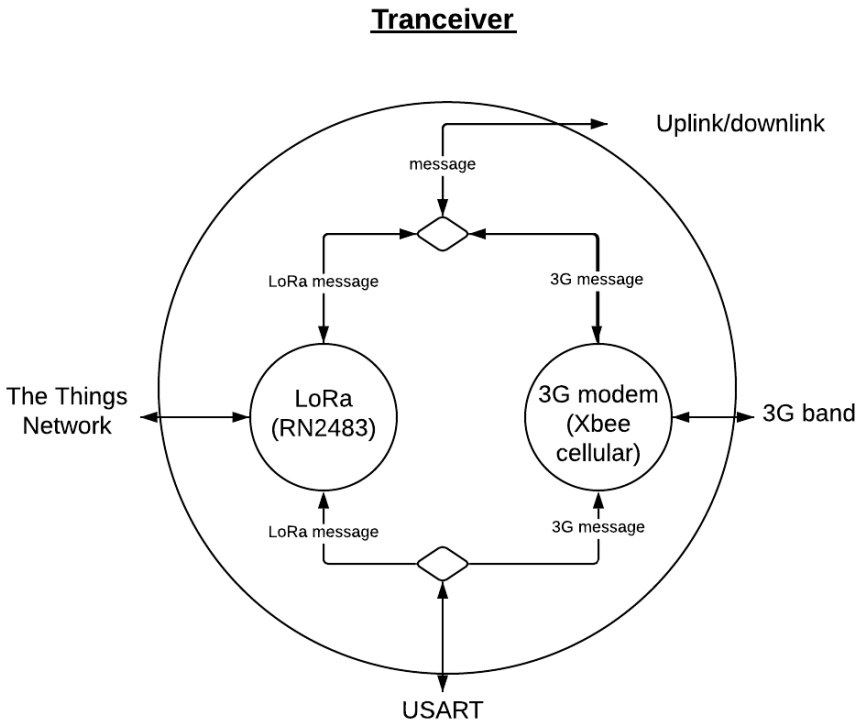


Figure 4.7: Software inner analysis level 2.

Modules main device

The modules that are needed to complete the project are illustrated in figure 4.6 and 4.7. They show all the required drivers needed, including ADC, UART, transceiver and time functionality. The transceiver module, illustrated in figure 4.7, can be further split into RN2483A (LoRa driver), and Xbee Cellular (3G modem driver). As this project mainly focuses on the LoRa aspect of communication based on the discoveries in chapter 2, the transceiver will be thought of as only containing LoRa driver.

ADC This driver was built to be able to read values at chosen times. The ADC does not need to be active continuously, and will therefore only be doing single conversions whenever needed. The ADC should also be built-in hardware on the MCU.

USART For this project, the synchronous version of UART will be used. The module is used between the MCU and a computer (”printf” for debugging), as well as between the MCU and communication chips. Both the RN2483A (LoRa chip) and Xbee cellular 3g

modem use USART to communicate with the MCU. This makes programming easier only needing to create one type of driver.

RN2483A This is the driver for the LoRa chip, in which all functions needed to connect the chip to a gateway as well as joining TTN's network for message transmission. Every command gives a response that is read and asserted, this is important to debugging if the device is offline. In addition to sending messages, the device will also sometimes receive downlinks after a transmission. This is read, analyzed and carried out depending on the command.

Time functionality This module is essential for the purpose of keeping the correct time and having the option to synchronize. This module enables the 8-bit built-in timer which is based on the external crystal of 32.768kHz. With a prescalar of 128, overflow interrupt (256 ticks), equals 1 second $\pm 2\mu s$.

Back-end application

Illustrated in figure 4.5, another vital aspect of the project is the back-end application to log data transferred from the main device. Its specifications are a part of the ones mentioned earlier.

Server The back-end application is split into two scrips, one being the server. The purpose of the server is to receive all payloads from TTN over the MQTT protocol. Furthermore, it separates the relevant data from the unimportant and pipes it to the logger.

Logger The logger receives relevant data-frames from the server and stores them in according to timestamp and device ID. It also keeps tracks of every gateway that receives the message and separates one file for each gateway.

Chapter 5

Implementation

The implementation of the complete system was built in three different processes: Back-end application, end-device software and end-device hardware. The complete hardware solution, using a battery to power it, is illustrated in figure 5.1.

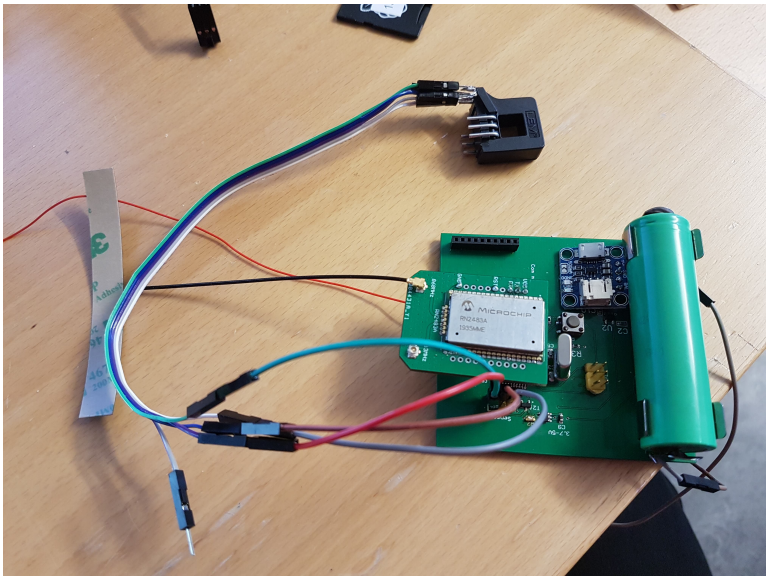


Figure 5.1: LoRaWAN transmission example.

5.1 Hardware

As mentioned earlier in this report, the hardware part of this project was outsourced. A more detailed explanation is therefore provided in that report [1]. For developing the circuit design the tool Altium Designer was used. This software was chosen because it's a familiar software and also very versatile and reliable. The resulting PCB created from Altium is illustrated in appendix A, while the resulting schematics are shown in appendix B.

5.1.1 Components and result

The hardware components were chosen based upon the acceptance criteria explained in the report [1]. A short summary of the main components is provided below, those relevant to the software drivers.

HO 10-P current transducer

This component is mentioned in "Limitations" (1.3), as the chosen sensor for this project to measure current from a solar panel. For details about it, see its data-sheet [5]. The most important data relevant is:

1. Current measurement range: $I_{pn} = [-25, 25]A$.
2. Supply voltage of 5V.
3. Internal reference voltage, $V_{ref} = 2.5V \pm 0.25$.
4. Output voltage range: $V_{out} = V_{ref} \pm 2V$

The sensor also supports external voltage reference, but due to ATmega324PB's option of internal reference of 2.56V, using the internal reference for the sensor gives the best resolution. A very important note when connecting the sensor, is doing it in a way such that a negative current will result in Voltage output range $[V_{ref}, V_{ref}+2V]$, and the positive measurements range $[0, V_{ref}]$.

To transform the measured voltage back to current, a formula was needed. As the datasheet did not directly provide one, a few simple experiments were conducted. This was done by using a DC power supply and a load strong enough to handle 1.5A current. The wire was then looped through the sensor up to 4 times, simulating up to 4*1.5A current. Two measurements for each loop was computed. The result is shown in table 5.1. Afterwards, the data from the table was input to a regression calculator, and the following formula was received:

$$V_{out} = 2.5 - 0.075 I_{pn}, \tag{5.1}$$

where I_{pn} is the current through the sensor. Knowing from previous list that the voltage varies with $\pm 2V$, and that the current range is -25 to 25 A, the theoretical formula for calculating V_{out} was deduced:

$$slope = \frac{(V_{ref} - 0V) - (V_{ref} - 2V)}{I_{pn}^{max} - I_{pn}^{min}} = \frac{2.5V - 0.5V}{0A - 25A} = -0.08 \tag{5.2}$$

which then leads to:

$$V_{out} = 2.5 - 0.08 I_{pn} \tag{5.3}$$

The result from equation 5.3 is very similar to equation 5.1, which supports the decision to use equation 5.3 to convert measured voltage back to current.

N	$I_t[A]$	$V_{out1}[V]$	$V_{out2}[V]$
0	0	2.5	2.5
1	-1.5	4.41	2.38
2	-3.0	2.27	2.27
3	-4.5	2.14	2.16
4	-6.0	2.08	2.03

Table 5.1: Data to verify output voltage to current through it.

Xbee Cellular 3G modem

As mentioned in section 1.3, the Xbee Cellular 3G modem was supposed to be used as a 3G device to be able to implement multiple protocols for the device. During the early testing with this communication chip, it was early discovered that the modem draws too much power for the main board to supply (see datasheet [6]), and is therefore not implemented further. This is further explained in chapter 7.

MCU: Atmega324PB

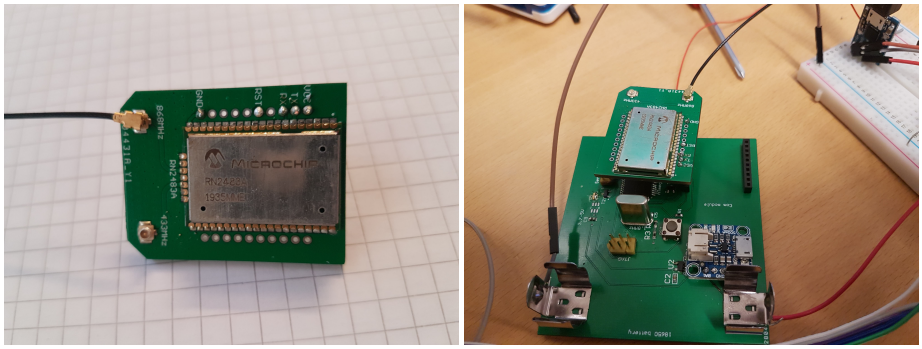
As mentioned in the report [1], the chosen MCU must have the option of at least two UART interfaces, to communicate with the chosen communication chip and the option to print to a computer for debugging. As the most obvious choice was Atmel Studio 7 to use for programming, the JTAG interface was an important requirement for the MCU. Also from the functional specification ”Low power consumption”, from the report [1], the chosen MCU must support different sleep modes. An ADC was also of utmost importance for this project, so an MCU with an embedded one was preferred. Considering all these requirements as well as the availability of the Atmega324PB, this one was chosen.

Tranceiver

To fulfill the specification about receiving and sending messages to a back-end application, a transceiver is an essential component. As mentioned in limitations (1.3), the PCB had to be adaptive to two different communication chips, namely Xbeee Cellular 3G [7] and the RN2483A [10]. The short reason for choosing RN2483A is because it supports all LoRa Class A operations (see section 3.1.1), as well as all data rates DR0-DR5 (from table 3.2). Considering these factors, and also because it is widely popular and easily obtainable to a decent price, RN2483A became the natural choice.

5.1.2 Final result

Due to the mistake of ordering a device without a ground plane, further explained in report [1], p. 21, no testing was able to be completed. Therefore the hardware unit used in this project is a not tested one. Nevertheless, the final product is illustrated in figure 5.2.



(a) Rn2483A circuit board (b) Main circuit board

Figure 5.2: Final result separated.

5.2 The Things Network Console

As explained in section 3.2.1, the console is an online GUI to manage end-devices and applications. For this project, an application with the name ”current-sensor” was created. This was implemented early on, and a device was also added to the application, named ”sensor-otaa”. The application generates its unique identifier, ”app-eui”, as well as different access keys for end-devices and back-end applications. The access keys for end-devices are generated when creating the device. A screenshot of the console is provided in appendix C.

5.3 Embedded Software

This section covers the end-device software and which tools that were utilized, while the back-end software is explained in the next section.

5.3.1 Language and software tool

As a result of choosing ATmega324PB as MCU, the main tool for developing software naturally fell on Atmel Studio 7. Atmel Studio provides compatibility with the avr-gcc toolchain which is used in this project. The chosen programming language resulted in C, which is the most common language. Atmel Studio also provides a possibility for debugging with the Atmel-ICE through JTAG.

5.3.2 USART driver

To communicate with a computer (for debug printing), and also the communication chip (RN2483 for example), USART communication is essential. Therefore, a driver was deemed necessary to provide an interface for both cases. The driver can receive and transmit strings over the UART interface.

5.3.3 Time functionality driver

To save power, the MCU has to be put to sleep whenever it is supposed to be ”Idle”. As mentioned in subsection 4.4.2 under ”Time functionality”, the MCU utilizes an 8-bit timer which is enabled to wake it up every second. This driver keeps track of the time as well as initializes and controls the timer.

5.3.4 ADC driver

As the sensor outputs an analog signal the MCU uses the built-int ADC to convert the signal to a digital value to be further processed.

5.3.5 Data Frame

To keep the data transmission size to a minimum, a specific frame type has been created. It’s illustrated in figure 4.4, and consists of 8 bytes. This driver uses the converted analog voltage and converts it into a frame to be transmitted.

5.3.6 RN2483 driver

Even though the MCU communicates with the chip over USART, the chip requires string-like commands to operate. It Was therefore deemed necessary to implement a driver to make different commands more dynamic and easy to send. Some of the most important routines are explained below.

Joining a network

The device for this project joins a network over the OTAA procedure, which is depicted in figure 3.3. The device EUI, application EUI and application key are all parameters pre-encoded into the network and microcontroller. To join a network, the correct settings must, therefore, be set in the RN2483 chip, and this is done by specific ”mac set” commands, specified in the Command Reference [11]. The join procedure must be completed every time the device powers up.

Transmitting uplink

A standard transmit command with RN2483A looks like this: ”mac tx <port> <payload>” (see RN2483 Command Reference [11]). The payload part of this command is a hexadecimal string where two and two characters represent a byte. An example of a transmit command is shown in figure 5.3, using the created data frame as payload.

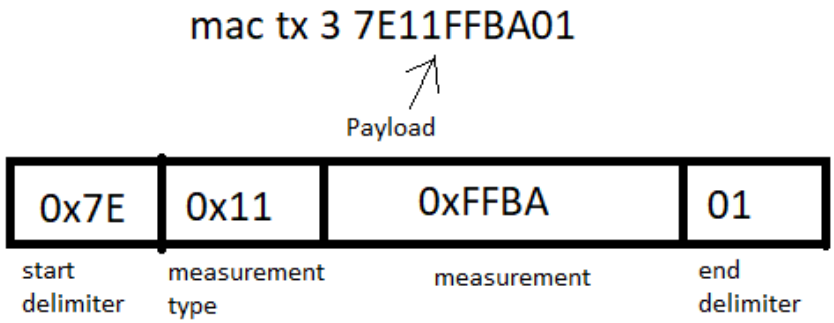


Figure 5.3: LoRaWAN transmission example.

Receiving downlink

When transmitting messages, the chip will receive four possible responses from the gateway according to the Command Reference [**?**]:

1. mac_tx_ok
2. mac_rx <portno> <data>
3. mac_err
4. invalid_data_len

If no downlink is en-queued by the network, pt. 1 is the response. On the other hand, if a message has been en-queued, the second response will be received. In this case, the <portno> is the port number in which sent, and the <data> is the payload in the downlink in hexadecimal pairs, the same as with uplink payload. Because the device is a Class A device (see section 3.1.1), this is the only was the end-device can receive downlinks

5.3.7 Result: LoRa Smart Current Sensor

To realize the LoRa Smart Current Sensor, all the drivers and software mentioned above were merged. The main program utilizes these drivers to realize the state machine mentioned in section 4.4.2. Figure 5.4 illustrates how the different states interact. The four

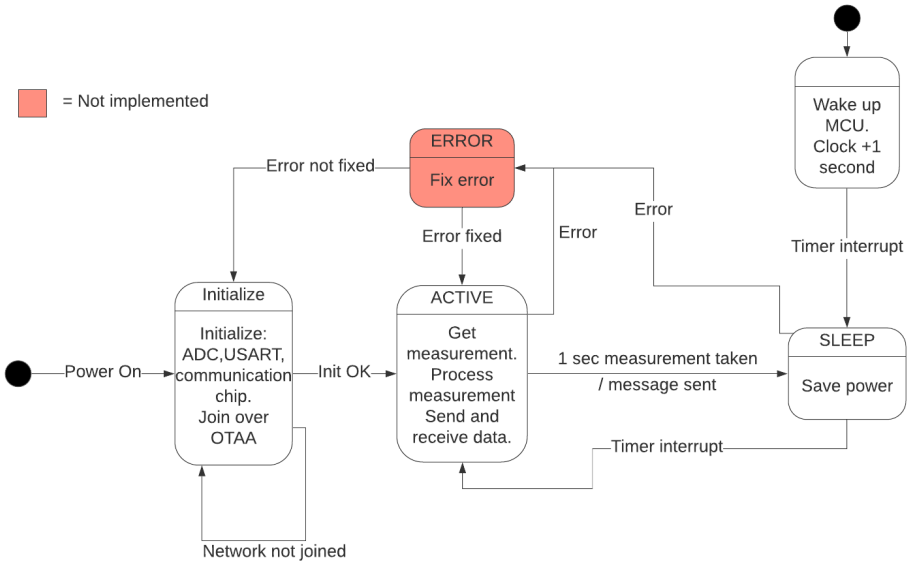


Figure 5.4: Main program state diagram

different states, Initialize, ACTIVE, SLEEP and ERROR will be described in further detail below.

State: Initialize

This is the first state the system enters. In this state, all relevant drivers are initialized and prepared for usage. The following elements are being initialized in this state:

- USART
- ADC
- Timer2
- The LoRa tranceiver

After initializing these drivers, the last requirement for leaving the state is the join procedure, over OTAA. As illustrated in 5.4, the state will loop until it manages to join a network. The initialization of the LoRa transceiver sets the RN2483 chip's application EUI, device EUI and application key which is required to join over OTAA.

State: SLEEP

This represents a power-saving state. The MCU goes into power-save mode, as well as the other modules on the main circuit board are shut off to save power. Every second a timer interrupt will trigger, and the interrupt service routine will change state to ACTIVE.

State: ACTIVE

This state does all of the heavy liftings for the system and is visualized in the activity diagram in figure 5.5.

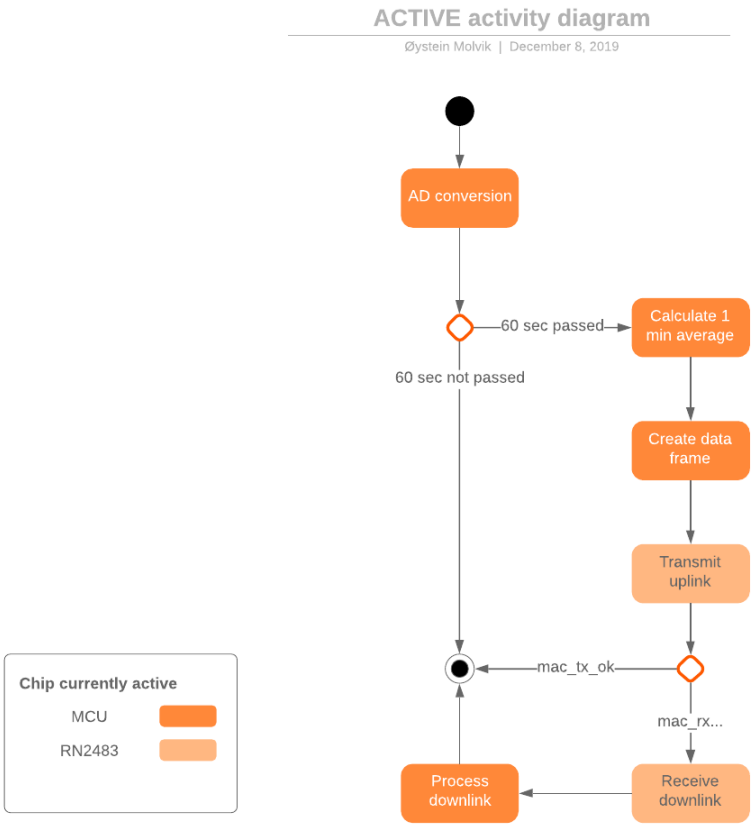


Figure 5.5: ACTIVE state activity diagram

The system will enter ACTIVE state in the beginning of each second. After doing a single AD conversion it checks if whether or not 60 seconds has passed. In case it hasn't,

it simply adds the value and goes to SLEEP state. If 60 seconds have passed, a minute-average measurement will be calculated, and payload converted into a data frame. This data frame is then transmitted over LoRa to the back-end application, and if there are no en-queued downlinks, it goes to SLEEP state, otherwise, the MCU processes the downlink message.

5.4 Back-end application software

The back-end software is based on the SDK's provided by TTN (see section 3.2.4) and is written in python with TTN's python-API [12]. Python is a familiar high-level programming language and very "up-and-coming" on the back-end front, and therefore became the natural choice. The python-API provided by TTN contains three different objects:

- HandlerClient
- ApplicationClient
- MQTTClient

For this task, only the MQTTClient was used. The client connects to the application created in section 5.2. To connect to the TTN-handler, the client has to be configured with the correct application- id, and key. While connected, if an uplink routine was specified, this function runs whenever an end-device sends an uplink message. The back-end application can be summarized through the illustration in figure 5.6, or the following list:

1. Decode uplink message.
2. Send meta-data and decoded payload to logger.

5.4.1 Logging relevant data

The uplink received by the back-end application contains a lot of unnecessary meta-data. Therefore, before logging, not relevant data must be stripped. The correct data is then logged to a $.csv$ file, one for each day and each gateway. The format of the uplink looks like the following:

Data-rate, time-stamp, RSSI, SNR, Decoded payload

An example looks like this:

5, 1575722812, -115, -3, 2.457

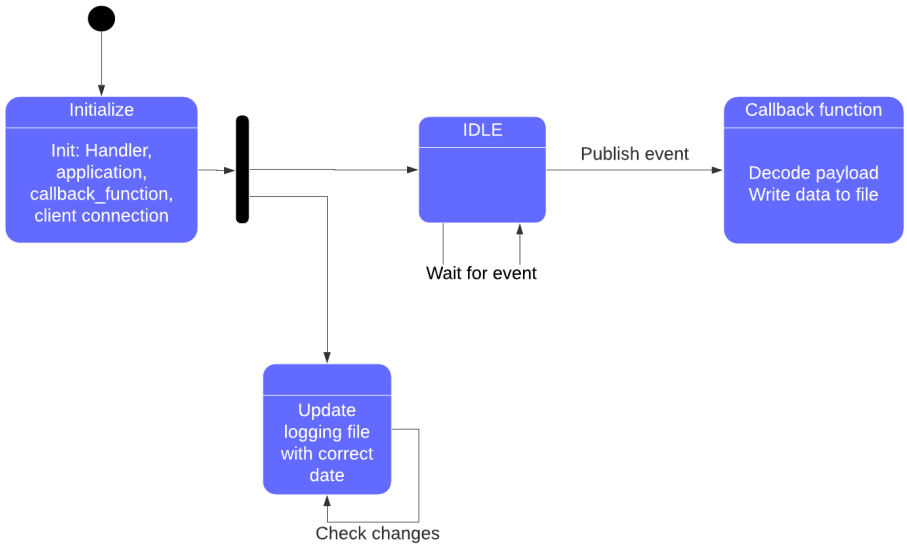


Figure 5.6: State machine diagram for back-end application

Chapter 6

Testing and results

6.1 Testing

This section is dedicated to testing the final product, involving both the end-device (software and hardware) and the back-end application. This is done by doing tests that directly answer to the acceptance criteria specified in section 4.3. If all tests pass, the device passes.

6.1.1 Updating software

After connecting an Atmel-ICE to the device, Atmel Studio was able to detect the device and read its signature. It was also able to change different fuses and verify them, and also flash over new software (see fig. 6.1). When deciding on the design for a JTAG header, a specific variant was chosen, namely the ISP header. This header ended up not supporting the debug mode from Atmel Studio.

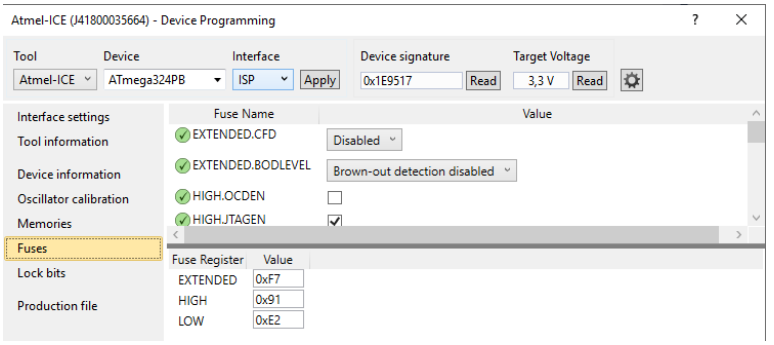
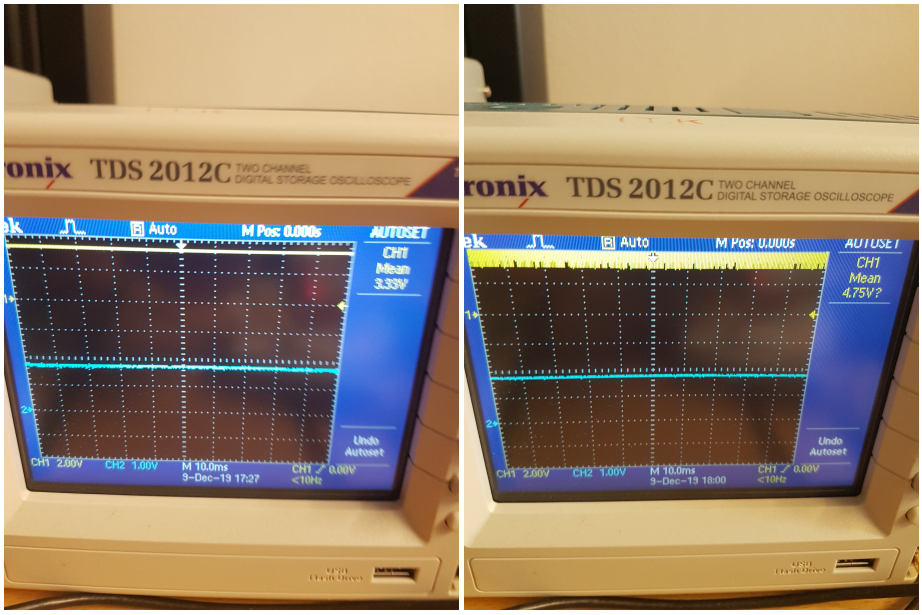


Figure 6.1: MCU connected to Atmel Studio 7.

6.1.2 Power Supply

The device is to be powered by a 18650 battery and provide correct voltage around the circuit. Figure 6.2 illustrates the voltage after the LDO regulator as well as the step-up regulator while using a 18650 battery. Also, the battery is capable of being recharged by plugging 5V (micro USB) into the charge circuit. It was tested on a depleted battery and left overnight. In regards to functional specification 3 (from 4.1), the device is not capable of transforming 12/24V into 5V or be supplied directly with 12/24V.



(a) 3.3V output from LDO voltage regulator. (b) 4.8V output from step-up voltage regulator.

Figure 6.2: Voltage regulator measurements.

6.1.3 Low Power Consumption

This test focuses on the acceptance criteria pt. 5 and 6. To test those, a simple software program was designed to toggle the two GPIO pins used for controlling the two transistors T1 and T2 (see appendix A), every time the device woke up from sleep mode. As mentioned, the timer is set to overflow each second. From figure 6.3, the oscilloscope shows the output from the transistor T1 as on and off with a frequency of 1 second. The same method for T2 yields the same result.

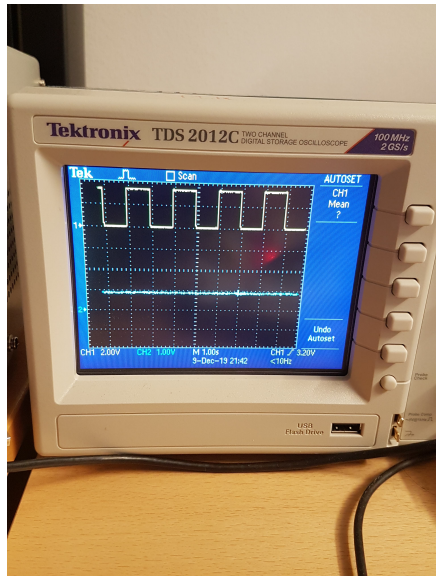


Figure 6.3: Transistor 1 being toggled with a frequency of 1.

6.1.4 USB connection

To pass the acceptance criteria pt. 12, a UART header, with possible a USB connector is required. The hardware chosen for this project does not hold that possibility. UART itself is correctly set up, this can be viewed in figure 6.4. This shows that the UART is properly able to transmit bytes as required. The device has 3 UARTs, so the possibility is technically there, but from a flaw in the design, a header was not implemented. To use the print debugging during this project, a small wire had to be carefully soldered directly on to the TX2 pin on the microcontroller. This worked (as shown in figure 6.4), as a short-term solution, and is not recommended for a long-term solution.

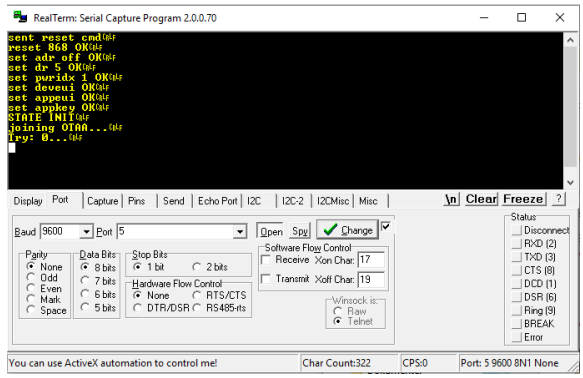


Figure 6.4: UART transmissions.

6.1.5 End-to-end communication

To test device-to-back-end communication, a specific test was designed. Using the same setup as mentioned in section 5.1.1 to verify the measurements, a simulated current of 3A ran through the sensor. Using an oscilloscope, the output voltage from the sensor was measured to 2.31V, as shown in figure 6.5. This information is then encoded into a payload consisting of two bytes and transmitted over LoRa to the back-end application. From figure 6.6, the payload in hexadecimal, decoded voltage measurement and estimated current is shown. To also verify that the logger saves the correct value (at this time, its the voltage), this is illustrated in figure 6.7.

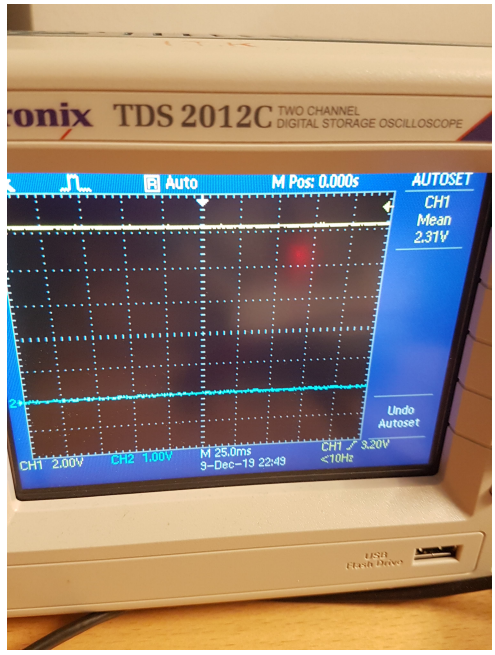


Figure 6.5: Measured voltage from sensor.

```
Received uplink from  sensor-otaa 2019-12-09 16:42:31.194553
Real voltage: 2.308
Real current: 2.400000000000002
Hex payload:  0xfec5
```

Figure 6.6: Uplink received by back-end.

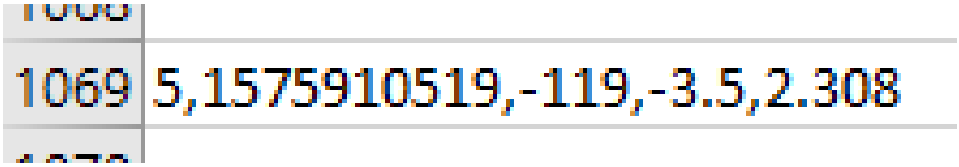


Figure 6.7: Data logged based on uplink.

6.1.6 Reset and receiving commands

From functional specification pt. 6, the unit must be able to be reset locally and wirelessly. The button was implemented and worked as intended. The system resets every time, as well as the other connected components (transceiver and sensor). In regards to resetting from a command from the back-end application, this did not work at all, as the MCU does not have a standard way of resetting it from code. Other commands, like changing the time, data rate or duty cycle, worked properly.

Chapter 7

Disscussion

This chapter focuses on the produced results from testing. These results will be compared with the acceptance criteria, to provide an answer to whether or not the device is good enough.

7.1 Discussion of results

This section compares the end-device and the back-end application (individually) to the relevant acceptance criteria. The relevant acceptance criteria to the end-device are pt. 1, 2, 3, 4, 5, 6, 7, 8, 9 and 11, while pt. 3 and 10 are relevant to the back-end application.

7.1.1 LoRa Smart Current Sensor

The hardware for this device was as mentioned outsourced [1]. Even so, the testing for that project was not done, and become a part of the entire solution. To test the components, the completed tests from chapter 6 were built in such a way that the components were also tested.

Updating software

Section 6.1.1 clearly states that the device is capable of being programmed and identified by the Atmel Studio 7 via the Atmel-ICE debugger. This was the most essential criteria to pass early on, as it gave the basis to do any further work. The chosen ISP header did not support standard debugging via Atmel Studio 7. Hence, acceptance criteria pt. 9 was only partially passed. The criterion is considered ”partially” passed because even though it’s not fully passed, the essential part of the criteria was being able to update the software, which it can.

Power supply

Concerning acceptance criteria pt. 1, 2 and 4 from section 4.3, the test was described in section 6.1.2. The result shows that the device capable of being powered by a 18650 battery, and providing the required voltages around the board. This proves that the two regulators, step-up and LDO is working properly, as shown in figure 6.2. But due to a drawback of the hardware chosen, the device is not capable of being supplied by 12V/24V from the solar panel. This is only possible if a transformer from 12V/24V to 5V is provided. That is not good enough to pass pt. 4. Regarding pt. 2, the device certainly was able to be charged using power from the solar panel, and in under 12 hours. This concludes that pt. 1 and 2 are passed, while pt. 4 failed.

Low power consumption

An important functional specification was to be power efficient. This spec. corresponds to acceptance criteria pt. 5 and 6. Figure 6.3 illustrates that the frequency of when the communication chip would be powered. Same result yield for powering the sensor. As the distance between the vertical stippled lines in the figure represents 1 second, the GPIO pin from the MCU is being toggled every second. Because this toggling happened when the MCU woke up, both pt. 5 and 6 are passed.

End-to-end communication

The main objective of this project was to develop an end-device that can do a sensor measurement, process it, and transmit it to a back-end application for logging. Looking at the test in section [6.1.5], this is working almost properly. The MCU reads 3.308V compared to the measured 3.1 using the oscilloscope. This indicates that the ADC is working correctly. The value was encoded and transmitted to the back-end. Then, the value was decoded and printed to show the same value, shown in figure 6.2. The application then logs the desired value in an .csv file, as shown in figure 6.7. The MCU and back-end application functions properly, hence the acceptance criteria pt. 3 and 10 are passed. The only deviation is the measured current compared to the actual current. The reason for this deviation is unknown and does not compare with the previously mentioned experiment in section 5.1.1. The improvised solution to ”simulate” higher current probably has its drawbacks, which could propose a deviation. For future usage of the sensor, one should use the theoretical formula [5.3] mentioned in section 5.1.1. As the deviation is not a part of the criteria, the device still passes pt. 3 and 10. This test is also able to deduce that the back-end application is serving the end-device seamlessly while running in accordance with acceptance criterion pt. 10.

Reset and receiving commands

The device was supposed to have two different ways of resetting the device, both locally by the use of a button, and wirelessly from commands over LoRa for instance, from functional specification point 6 in section 4.1. The button works as intended, explained in section 6.1.6, and hence passed criteria pt. 7. To pass criteria pt. 8, the entire system had to act accordingly when receiving a specific command from the application. This was not the

case with sending a reset command as explained in section 6.1.6. Hence, the device does not pass pt. 8.

USB connection

Shortly explained in section 6.1.4, the chosen hardware did not support live printing to a computer in the field and fails acceptance criteria pt. 11.

7.2 Areas of Improvement

During the design phase, multiple decisions were made, not all optimal. Here is a list of possible improvements for both hardware and software:

1. The charge circuit is too close to the battery slot and should be moved closer to the communication chip header.
2. As seen in figure 6.2, the communication chip header is mirrored compared to the actual chip. This should be improved.
3. Switch the ISP header with a more common 10-pin JTAG header to provide debug options.
4. Use interrupts from UART to signalize when the LoRa transmission/receive is complete to save more power.
5. Create more dynamic drivers for the different modules.

7.3 Future Work

This is the list of tasks that should be implemented on a later occasion to improve the functionality and usage of the device:

1. To introduce the option to get live prints in the field, an adaptive UART header with a possible USB connector should be implemented.
2. To make error detection easier, a few LEDs should be implemented to provide a visual effect on whether it's working correctly or not.
3. As adaptive data rate ain't possible when using The Things Network, the possibility of receiving the signal strength and signal-noise ratio and using them to choose a suited data rate could beneficial.

Chapter 8

Conclusion

For this project, an embedded system utilizing both LoRaWAN and Cellular 3G (to some extent) was developed. The ambition of the project was to create a proof of concept by developing a smart end-device integrated with a current sensor with the capability of communication over LoRaWAN and cellular 3G. The device is fully capable of doing analog measurements from the sensor and processing the data and transmitting said data to a back-end application also developed for this project. On the other hand, the device is not fully capable of receiving and completing all commands given from the back-end application.

As most of the acceptance criteria were met, the goal is achieved. Nevertheless, some important criteria were not met, so the end-device is in no way optimal. Improvements and further work are illustrated in section 7.2 and 7.3.

Bibliography

[1] Øystein Molvik. Design and implementation of a LoRa End-Device measuring current. Technical report, NTNU, Trondheim, 2019.

[2] Mahmoud Shuker Mahmoud and Auday A. H. Mohamad. A Study of Efficient Power Consumption Wireless Communication Techniques/ Modules for Internet of Things (IoT) Applications. *Adv. Internet Things*, 06(02):19–29, 2016.

[3] LoRa Alliance Technical Committee Regional Parameters Workgroup. LoRaWAN^{TM} 1.1 Regional Parameters. page 16, 2018.

[4] Telia Norge. Dekningskart. `https://www.telia.no/dekning/`, 2019.

[5] LEM. Current Transducer HO-P Series. `https://www.lem.com/sites/default/files/products_datasheets/ho-p_series.pdf`, 2014.

[6] Digi International. Digi XBee Cellular 3G. `https://www.digi.com/pdf/ds_xbee-cellular-3g.pdf`, 2014.

[7] Digi International. DiGi XBee Cellular 3G Global - User Guide. `https://www.digi.com/resources/documentation/digidocs/PDFs/90001541.pdf`, 2019.

[8] The Things Network. Coverage. `https://www.thethingsnetwork.org/country/norway/`, 2019.

[9] Rashmi Sharan Sinha, Yiqiao Wei, and Seung-Hoon Hwang. A survey on LPWA technology: LoRa and NB-IoT. *ICT Express*, 3(1):14–21, 2017.

[10] Microchip Technology Inc. Low-Power Long Range LoRa® Technology Transceiver Module. `http://ww1.microchip.com/downloads/en/devicedoc/50002346c.pdf`, 2017.

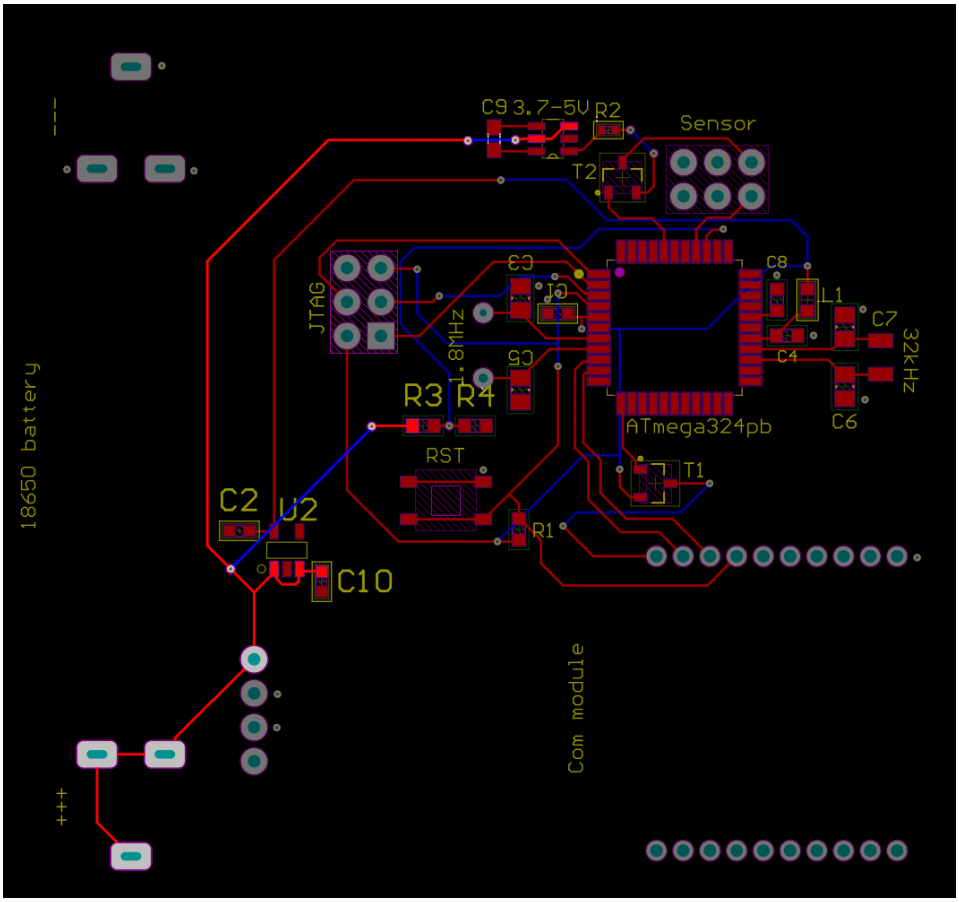
[11] Microchip Technology Inc. RN2483 LoRa Technology Module Command Reference User's Guide. `https://ww1.microchip.com/downloads/en/DeviceDoc/40001784B.pdf`, 2015.

[12] Johan Stokking. The Things Network Python SDK. https://github.com/TheThingsNetwork/python-app-sdk, 2019.

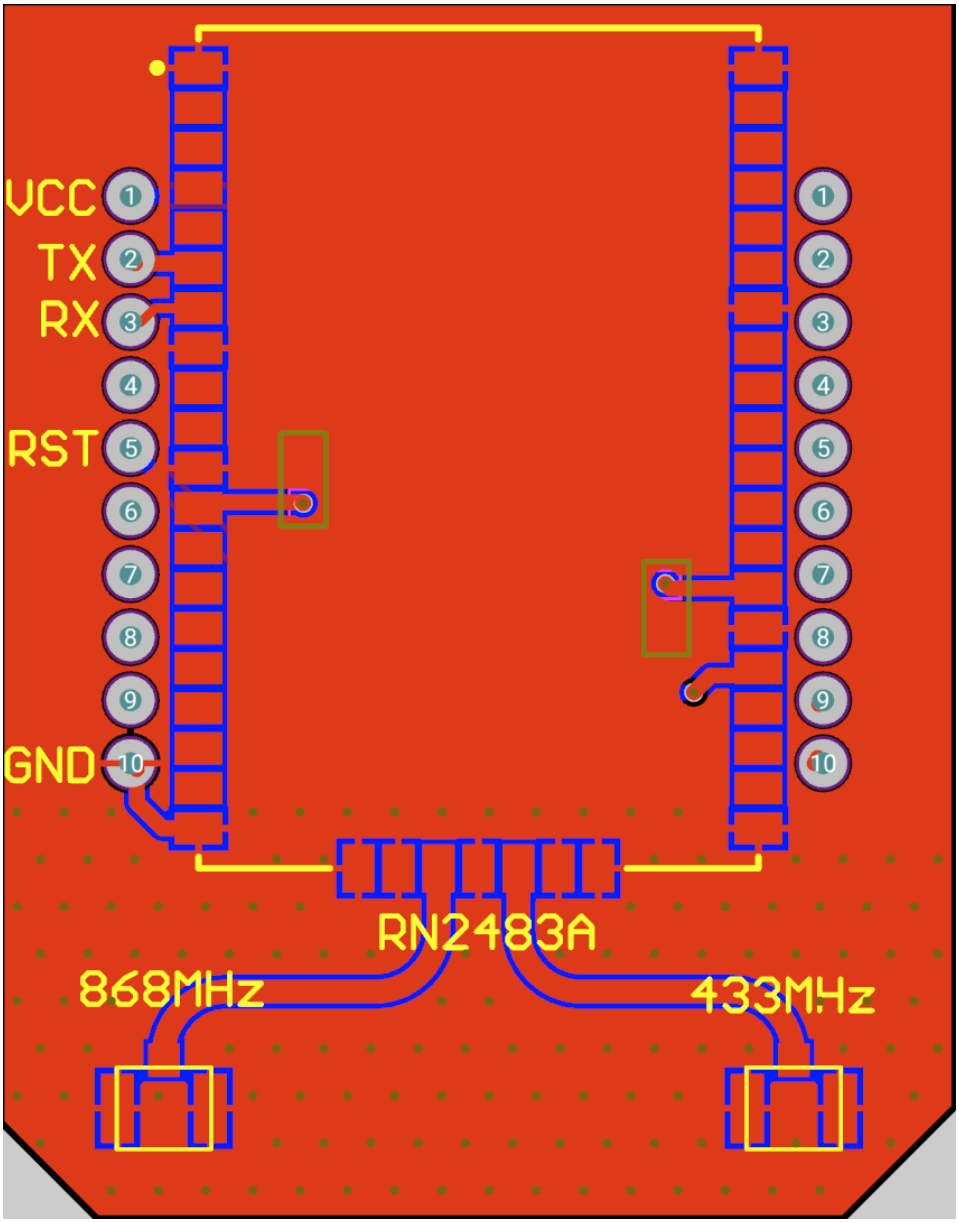
Appendix

Appendix A: PCB's

End-device PCB

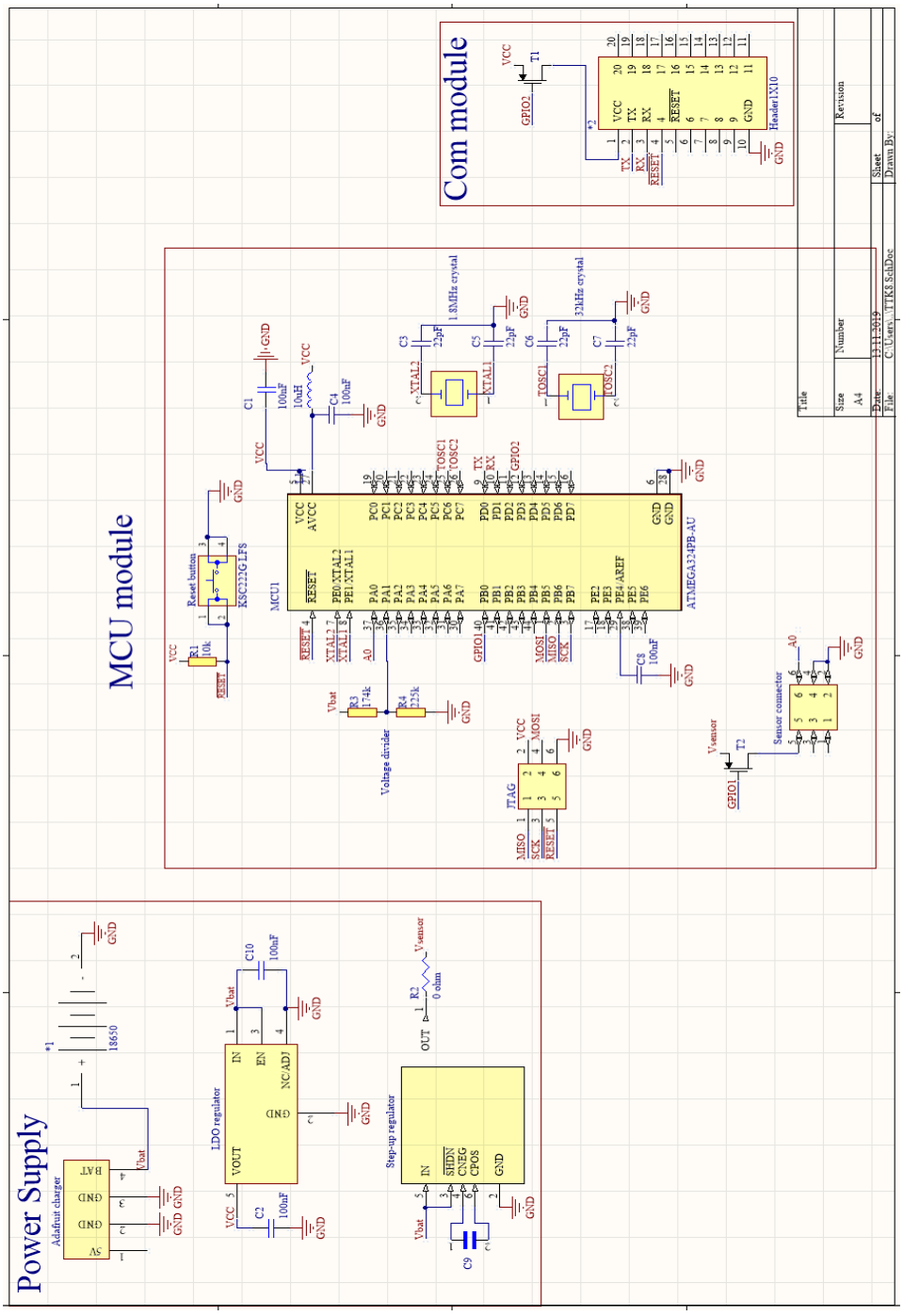


LoRaWAN RN2483A PCB

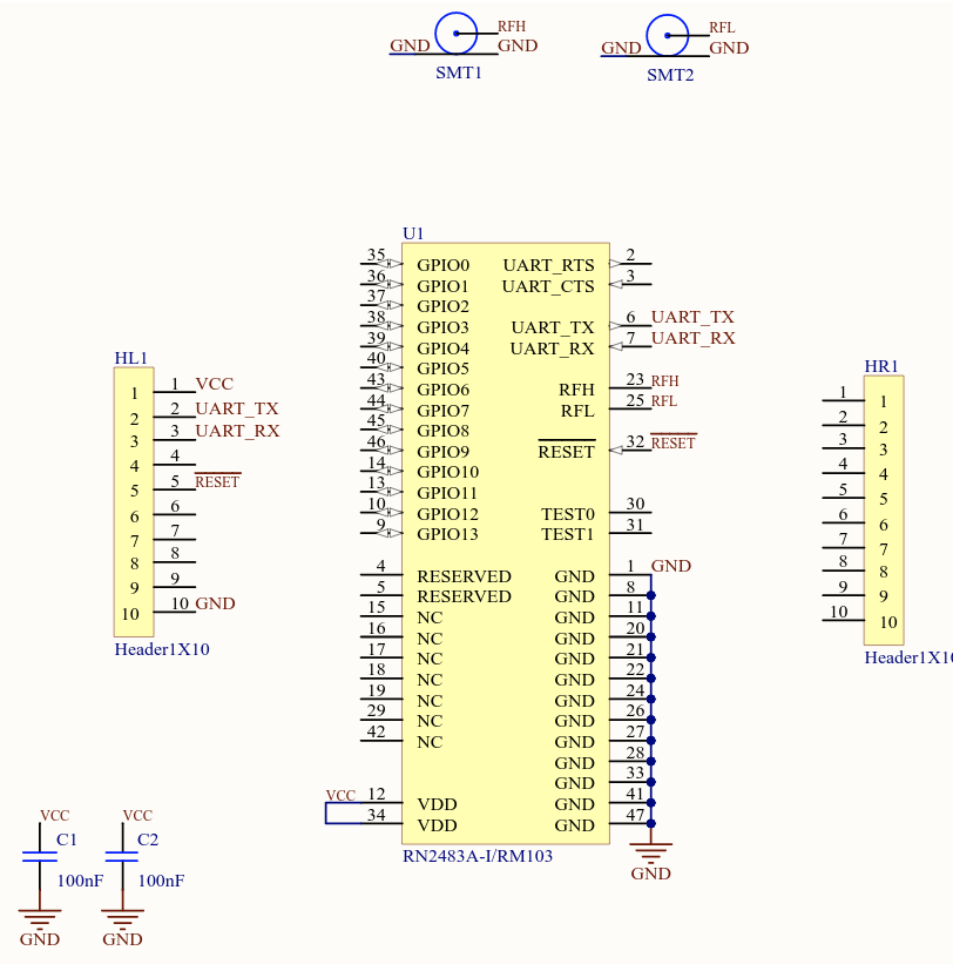


Appendix B: Schematics

End-device schematic



LoRaWAN RN2483A schematic



Appendix C: TTN's Console