Cape Peninsula
University of Technology

Design and implementation of a device to optimise SCADA Using LTE-M

by

ROBIN MAHARAJ

Thesis submitted in fulfilment of the requirements for the
Degree

Doctor: Electrical Engineering

in the Faculty of Engineering and the Built Environment

at the Cape Peninsula University of Technology

Supervisor: Dr Vipin Balyan
Co-supervisor: Prof Mohamed Tariq Khan

Bellville
September 2021

DECLARATION

I, Robin Maharaj, declare that the contents of this thesis represent my unaided work and that the thesis has not previously been submitted for academic examination towards any qualification. Furthermore, it represents my own opinions and not necessarily those of the Cape Peninsula University of Technology.

14 Dec 21

Signed **Date**

ABSTRACT

With process control systems driving many ERP systems and decision-making applications, there is an inherent need to deliver real-time data to MES systems and ERP at low cost and reduced data latency. Also, with some assets distributed over a large geographical area, there is an intrinsic need for a remote data visualisation system at low operating costs.

There is also an overwhelming need to add low-cost data points to existing process control systems to augment the current data to improve data analytics or provide critical performance data not initially envisioned in the design. In order to achieve this data analysis there is a need to provide a convergence of data to a common gateway from different datasources which includes legacy systems. This research presents solutions to process data from these different data sources for use or interigation by MES systems.

This research presents software and hardware design to add to the data value chain and visualisation of process control systems.

This research also investigates and implements wireless technologies and protocol wrappers and interfaces to enhance the data value chain in a legacy process control system. The technologies investigated are physical layer LoRa nodes, nrf2401l(a 2.4GHz radio module), MQTT, 4-20mAConvertors.

ACKNOWLEDGEMENTS

First, all praise God for allowing me to complete my research/design work successfully and for acquiring new knowledge in the process that I know He has plans for.

My sincere gratitude and appreciation to my supervisor, DR Vipin Balyan, for his invaluable guidance, enthusiastic help, and consistent encouragement throughout the entire research project.

Also would like to extend my thanks to my co-supervisor, Prof MTE KHAN, as well as Chad Venables of RIOTware for developing the Frontend Web API to visualize the data received from the device gateway.

I want to dedicate this thesis to my late parents, my wife Gail, my three kids Aston, Aidon, and Sashni, for all the unconditional love and support, and patience, for all the encouragement. I want to extend a special thanks to my kids for all the sacrifices made. I know they felt it most; thanks, love you guys.

Robin Maharaj

TABLE OF CONTENTS

CHAPTER 1: Proposal

CHAPTER 2: Literature Review

CHAPTER 3: Hardware Design and Implementation

CHAPTER 4 : Design of IIOT Device to parse data directly to SCADA system using LoRa Physical Layer

Chapter 5: Optimising Data Visualisation in the Process Control and IIOT Environments

Chapter 6: Test Bed and Results

CHAPTER 7: Conclusion and Future Work

List of Tables

List of Figures

Appendices

GLOSSARY

Terms/Acronyms/Abbreviations	Definition/Explanation
4IR	4th Industrial Revolution
ADC	Analogue to Digital Converter
ADR	Adaptive Data Rate
AI	Artificial Intelligence
APN	Access Point Name
ATD	Asset Tracking Device
COAP	Constrained Application Protocol
CPU	Central Processing Unit
CSS	Chirp Spread Spectrum
DMA	Direct Memory Access
DNS	Domain Name service
ERP	Enterprise Resource Planning
FSM	Finite State Machine
FTP	File transfer Protocol
GPIO	General Purpose Input Output
GPRS	General Packet Radio Service
GPS	Global Positioning System
GTP	GPRS Tunnelling Protocol
HAL	Hardware abstraction Layer
HTTP	Hyper Text Transfer Protocol
I/O	Input/output
IIOT	Industrial Internet of Things
IoT	Internet of Things
IP	Internet Protocol
ISR	Interrupt Service Routine
JSON	Java Script Object Notation
LAN	Local Area Network
LoRa	Long-range low energy RF technology.

LPWAN	Low Power wide area network
MES	Manufacturing Execution Systems
MQTT	Message Queuing Telemetry Transport
SCI	Serial communications interface
SPI	Serial Peripheral Interface
TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access
UDP	User Datagram Protocol
WAN	Wide area Network
XML	Extensible Markup Language

CHAPTER 1

PROPOSAL AND STRUCTURE OF THESIS

This chapter comprises two parts, the proposal document before embarking on the implementation of the deliverables of this research. It is the guideline document used to carry out this research; as the investigation progressed, the research trajectory evolved. The second part of this chapter includes other preliminaries not included in the proposal document.

1.1 Proposal

Design and implement a device and framework to optimise SCADA using LTE-m incorporating a LoRa communication interface, radio frequency (rf) module with SPI-interface, 4-20mAinterface, and an Ethernet and embedded OPC UA wrapper.

Keywords: 32-bit micro-controller, UART communication interface, RF module, OPC UA, Ethernet wrapper.

1.1.1 Introduction

The race in the 4IR space is reaching epic proportions, especially in the industrial automation space. Process control engineering is by far not excluded from the innovations and drivers of this revolution. Much research and innovation are bringing expensive propriety systems in line with what the 4IR space is achieving. There is also an inherent need for SCADA systems to integrate into Maintenance management systems, ERP systems, and MES systems to present compelling, low-cost data models, improve MTTR, reduce the production cost, etc. This means that many legacy systems need some form of integration capability to bring data that the legacy systems do not have the capacity or mechanisms to integrate into the higher-level systems. The IIOT options also create opportunities for systems to display process control systems to 3rd party users and consumers to give them some visual perspective regarding services provided to them or that they are directly affected by.

The 4th IR is all about data, data consumers, data drivers, data cost, data latency, data integration, data decision making, and the list goes on. The process engineering data is not new; it is the source of data and thus one of the critical data drivers in the 4IR, but this data comes at a price.

To achieve interoperability from field devices to Erp and MES systems, different hardware and software implementations and protocols are researched on a case study basis. Hardware interfacing of 4-20mAinterfaces to wireless node, comparing other technologies to achieves this. LoRa Phy, nrf24l01 radio module, LoRa to IP bridge. A low-level and barebones approach will achieve all these implementations.

1.1.2 Existing and Proposed architecture for Modern Process Control Systems

There are essentially two disparate technology systems that seem to be converging in the 4IR automation world. The one is the legacy process control systems comprising PLCs, HMI, and SCADA. The other is a relatively new role player in the game, i.e., IIOT systems, namely microcontroller-based process systems. The architecture of these systems is different, yet they relate to the same on the higher-level data consumer level. The older legacy systems lend themselves to expensive propriety-based solutions. In contrast, the new microcontroller systems offer incredibly cheaper open architecture and more widely used protocol systems, resulting in substantial cost savings. However, one system does not negate the other, and both systems sometimes need to be implemented in the new data consumer-driven world.

There is a growing need for the systems to augment each other.

There is a growing need for research to get low-level and field-level systems to parse data seamlessly to high-level systems while offering greater interoperability and good latency. This research looks at different field-level devices and interfaces and a communication medium to enhance the seamless interoperability of data between all the levels of the data value chain.

This research provides a solution at an existing Waste Water treatment plant with a legacy process control system, an existing SCADA and HMI system, an SQL database, and geographically dispersed data points. This design will get additional data back into a data warehouse and a High-level system like ERP/ MES system or web service.

A Typical modern Day Process control system consists of a PLC system with an OPC server, a SCADA system, HMI, and sometimes a SQL database server, PLC or PAC with field devices, Actuators, instruments connected to it. The typical modern-day process control system has its local SCADA system together with its HMI. All the time series data acquired from the time series database residing in the SCADA historian can be further stored in SQL format in its Local SQL database. This data can be made available to the high-level application or a web application.

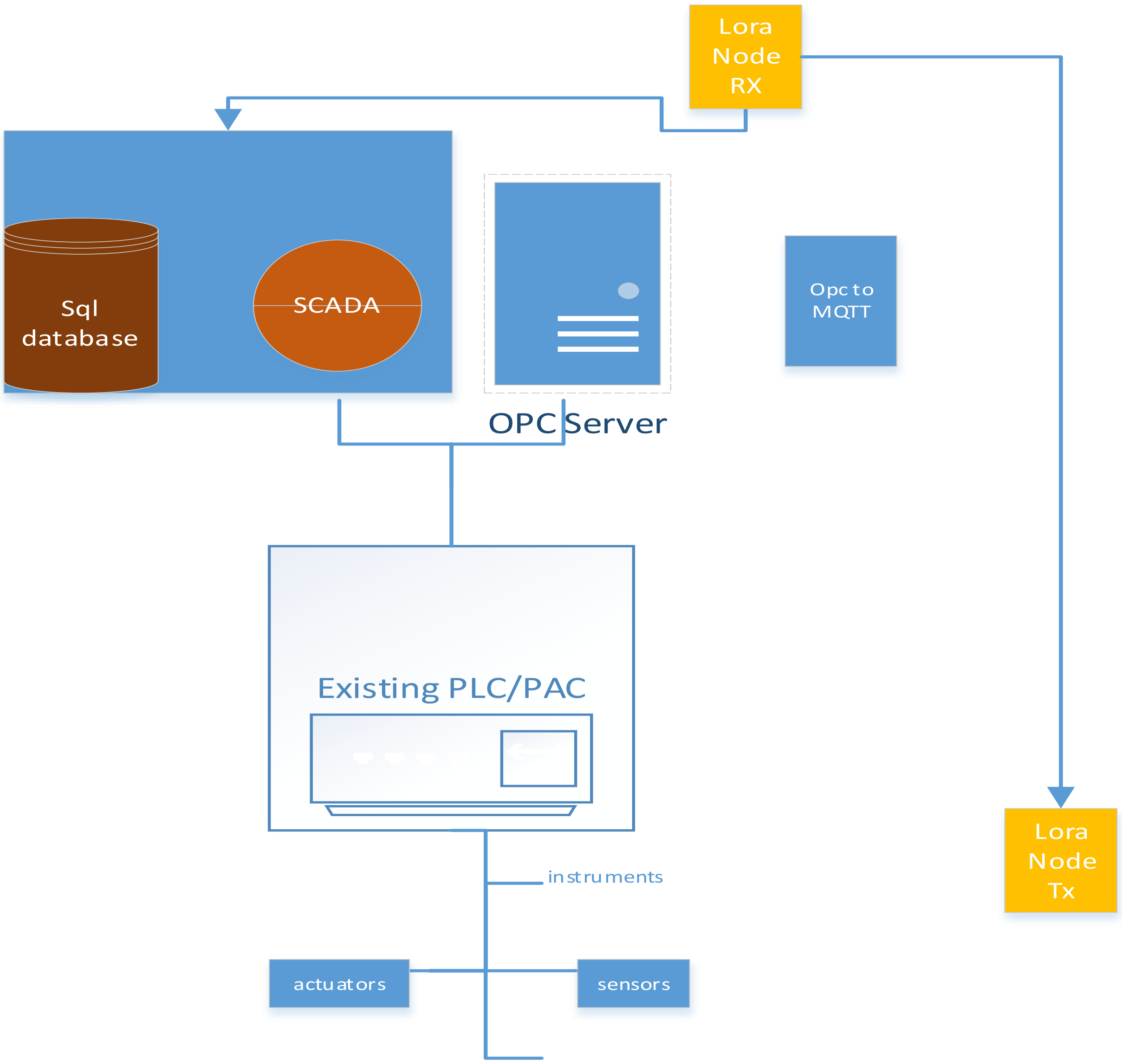


Figure 1.1. Typical modern-day Process Control System with added IOT LoRa-device.

Figure 1.1 represents an architecture where a measuring device was not installed as part of the initial process control system because there was no capacity. The cost of physically adding the additional section to the process control system is too high because there is no more capacity in the physical infrastructure (current process control system/PLC).

With the increasing need to integrate these systems into MES and ERP systems, it is sometimes found that additional field instruments and devices need to be added to the existing process control systems. To augment the data analytics that was not previously part of the existing Process Control System.

This research adds a measuring device that was not installed as part of the original process control system. There could be one of two primary reasons why this scenario exists. Scenario one, it is not practical/viable to add this extra instrument or sensor; either there was no capacity in the existing process control system because of lack of i/o's on the input card or the cost of physically adding the section to the process control system is too high because there is no more capacity in the physical infrastructure in place to add it while the distance between the Process Control System and the physical instrument is quite a considerable distance. On many occasions, these devices are situated hundreds of meters away from the process controller or SCADA Server like in a wastewater treatment plant.

After all the infrastructure is in place, a required device some years later can be quite an expensive and labour-intensive task to add to the process control system, especially if fibre optic or some other wired point-to-point network. If a device was added to the existing process control system, there is no inexpensive off-the-shelf solution to get this device's data into the SCADA or process control system. An affordable wireless solution needs to bring this field device data to the SCADA system or a remote MES/ERP system, and this research investigates two possible solutions.

To add this component to the system. There is an option to use one of two wireless IOTT options: the nrf24L01 2.4 GHz radio module or LoRa to LoRa node implementing the physical layer. (Koon, 2020) Both these options will need an IP bridge or some protocol wrapper to get this data back into the SCADA system, which can be further parsed to the OPC UA server for further transportation. At the source of the instrument, there would also be a need to convert the 4-20mAsignal or a 0-10 V to data variable to be parsed to a "LoRa or NRF2401l" format.

With the present trends in the modern digital era in terms of the Internet of things, there is an ever-growing need for devices to be connected to the Internet or a remote control centre. The industry has undergone substantial technological advancements in the last twenty years, both from a mechanical perspective and from an electronic and power electronic perspective. Low-cost microcontrollers have advanced quite tremendously, giving us 32-bit microprocessor capability and numerous peripherals with numerous communications interfaces. In addition, the cost of this technology has decreased exponentially, making asset tracking a viable option for technology solutions. (Muller, Wings and Bergmann, 2017)

This project will focus on research methods for acquiring data, processing this data converting it to a data algorithm for transmission. The project will also look at efficient buffering and storage techniques as not all the collated data will be transmitted. This research project endeavours to solve these issues using a 32-bit micro-controller as an asset tracking and data acquisition device. It also looked at the cost of transmitting data using different protocols, namely UDP and TCP, in the form of HTTP. All of these translate into a perimeter-based asset-tracking device

1.1.3 Statement of Research Problem

With the 4th Industrial revolution on our doorsteps and the need to improve productivity levels, there is an increasing need for data analytics and Artificial Intelligence. With process control systems driving many ERP systems and decision-making, there is an inherent need to deliver real-time data to MES systems and ERP at low cost and reduced data latency. Also, with some assets distributed over a large geographical area, there is an intrinsic need for a remote SCADA system at meagre costs. (Katti *et al.*, 2018)There is also an inherent need for SCADA systems to integrate into Maintenance management systems, ERP systems, and MES systems to present compelling, low-cost data models, improve MTTR, reduce the production cost, etc.

With the convergence of systems like SCADA, IOT, remote process control systems as well data analytics, there is a need for research to be conducted with the view to optimise and reduce the cost of the different designs and to look at the possibility of introducing industry-standard and to regulate and streamline research in this area.

The number of data sources is increasing exponentially due to the fourth industrial revolution, introducing the need for a framework with a central point to process and aggregate these various data sources.

With so many propriety systems out in the industry and the vendors competing for industrial control systems footprint, many vendors seem to have made it difficult and expensive to integrate into high-level systems. IoT has put a spanner in the works, as embedded systems can now easily interface to low-level systems to present data to different platforms using the available transport protocols. Data that can be used and transported to separate layers for use by other systems or platforms. (Power and Kominek, 2013)

Many asset managers, operational managers, finance managers, and maintenance managers are seeing an increasing need for real-time asset management devices in the process environment to drive decision making of their enterprise resource planning and maintenance management systems to improve asset life cycle costs and Indicators. Present SCADA systems meet most of these needs at very high prices and via propriety protocols; this research will present this data at minimal costs and low latency. This study also intends to highlight the different technologies from a low-level perspective to give integrators and developers insight into process control systems from a data portability perspective.

Most remote assets have varying challenges in terms of price, data transmission, asset management, asset life cycle management, and the list goes on. Also, asset owners cannot rely on the operators to stop the asset when it is outside safe parameters, so the asset owners/managers need asset monitoring and control. In terms of process auditing and trend analysis, there is also a need to monitor asset parameters to improve asset performance and process engineering. The industry requires an asset information interface device capable of delivering these needs at a reasonable data cost.(Calderón Godoy and Pérez, 2018)
Artificial intelligence is making exponential strides in the automation arena, and there is a dire need for data to be transmitted to high-level systems for processing and analytics.

Data, like equipment running hours, electrical motor parameters, etc., are not measured with present systems; this is a critical factor from the frequency-domain data analysis approach. The high cost of these infrastructure assets and the consequence of the failure of these assets creates a crucial need for an asset management device.

A secondary problem that might exist in getting intrinsic data to MES/ERP systems or even to local SCADA systems or applications; is that, if some years later there is a need to add I/Os to existing process control system only to find there is no more capacity on the process control system or in the existing communication infrastructure. Adding these needed I/Os could push up costs, making it unfeasible because of high infrastructure cost and physical hardware costs, or the existing hardware could be obsolete, but implementing this design or variation of the design of this thesis could make it feasible again.

1.1.4 Project Constraints

This project/research is developing a complex microcontroller interface and integrating it into other communication technologies. This project for this dissertation will be implemented on a modular basis on a development board/boards as it is for proof of concept and analysis. Levels of implementation range from an embedded design to relaying data via the lte-m/gsm interface.

This design has an implementation on different levels of the value chain, from hardware implementation to low-level programming to data transmission to high-level programming. Because of the different levels of implementation, it will be challenging to include a detailed design of the different levels of execution as the study will primarily look at achieving proof of concept which will implicitly imply the objectives of this research have been accomplished.

The software implementation of this project requires the system to run over an extended length of time to give an accurate evaluation of the systems. For the full performance of the software to be analysed, an existing process control system would have needed to be implemented; fortunately, this was possible in the form of the Potsdam Wastewater Treatment Plant, but for a lot of the other data points, we had to resort to dummy data.

1.1.5 Background and relevance of proposed Project

The industry has undergone substantial technological advancements in the last twenty years, both from a mechanical perspective and an electronic and power electronic perspective. Low cost, low-powered microcontrollers have advanced quite tremendously, giving us 32-bit microprocessor capability and numerous peripherals with numerous communications interfaces augmented by subsequent progress in the software and protocols arena. It has also allowed the introduction of low-powered devices to serve as data interfaces and relays, which reduces/eliminates the need for mains power.

Also, much of the past research done in this area has been done using rapid development platforms. This research will implement most development using a barebones approach to add value and understanding in this research area to enhance the fundamentals of embedded technologies implementation in third-world countries. However, modules will be implemented using rapid development platforms to decrease the time to implement the project.

This research also aims to make SCADA more affordable for high-level systems to use to provide more data to analyse, thus enhancing the data value chain. This research will also investigate making seamless data integration for high-level systems. All this will translate to: Minimise cost along the lifecycle of remote assets. Minimise the risk of managing and operating remotely located assets. Optimise remotely located equipment operation and production.

In the public sector, government departments like municipalities have thousands of process control systems that are geographically dispersed functioning locally without remote functionality. There are direct costs as well as response time implications concerning these systems. In many instances, these are critical systems affecting the public regarding basic needs, which translates to increased response times to failures. The inherent cost implications can also be reduced in terms of travelling time to some remote sites. There is currently no embedded device capable of bringing the different management systems together into one system for proper automated audit and asset management process. In this dissertation, we will embark on capturing data from numerous nearby devices and process control systems through different communication interfaces or techniques and relay integrate-able data to a remote control centre and or high-level applications.

This project will focus on research methods in acquiring data, processing this data converting it into a data algorithm for transmission. The project will also look at efficient buffering and storage techniques as not all the collated data will be transmitted. We will also look at the cost of sending data using different protocols, namely UDP and HTTP. This research will embark on a journey to solve these issues using a 32-bit micro-controller as an asset tracking and data acquisition device.

This research will employ GPS, LTE-m and Digital signal processing modules using the 32-bit arm microprocessor. (‘LTM-Research’, no date)This research will tap into 4-20 mA signal convert it into a data sentence.

This research will look at the wireless interface to get nearby data that may be excluded from a process control system or a stand-alone instrument in a remote environment that needs to get to the remote data warehouse. This research will explore LORA and/or 2.4GHz RF module to capture the data from the field instrument and relay it to the local SCADA/PAC or the asset management device.

1. 1.6 The Aim and Objectives of this Research

The purpose of this research project is to design and implement a control unit to serve as a communication processor and an embedded device that would interface into a process control system or a SCADA system and present intrinsic data to a higher-level system for real-time decision making and optimisation of the complete business value chain. (L., B. and J., 2015)The device will be implemented on an embedded platform with an LTE-M interface. LTE-M is the simplified industry term for the eMTC LPWA technology; LTE-M is a low power wide area (LPWA) technology, providing minimal device complexity and extended coverage while allowing the reuse of existing LTE base stations.

With the progress of cellular data transmission and improvement in the embedded platforms, some avenues have been introduced to converge the different technologies to add value to the business data value chain. Data analytics is also snowballing as a business tool, and real-time data would multiply the benefits of the data presented with low latency and intelligence at reasonable costs. Another contributing factor for the convergence of data would be to have process data or low-level data available at level 0 and level 1 systems that are directly integrate-able into high-level systems like MES/ERP systems.

Cost is also a significant driver when implementing systems. With the cost of SCADA systems so high, it is found that many asset managers are steering away from remote SCADA systems. If data was more accessible and portable from the low-level systems, this could be a driving force to drive higher-level systems.

The purpose of this device is to serve as a 4IR interface to high-level systems to enhance the data value chain to high-level business processes and data warehouses. This research will also investigate implementing a finite state machine to control data transmission rates and sentences to optimise cost and latency. The device will serve as a hybrid IoT device and an RTU for the transmission of SCADA data.

This research will also investigate the different protocols like OPC UA, MQTT, COAP, and the likes specifically on an embedded platform to give a low-level implication or perspective of using these different protocols. (Silveira Rocha *et al.*, 2018)It is also the purpose of this research to gain founding knowledge to interface different protocols with each other and look at ways to enhance the security aspects of the protocols used in the embedded platforms in the control and automation environment. The research will also look at the data types portability of level 1 systems to a high-level application, specifically in the .net platforms. It will look at the embedded implementation of some of these protocols.

OPC is the interoperability standard for the secure and reliable exchange of data in the industrial automation space and other industries. It is platform-independent and ensures the seamless flow of information among devices from multiple vendors. The OPC Foundation is responsible for the development and maintenance of this standard.

The OPC (Uwe Steinkrauss, 2010)standard is a series of specifications developed by industry vendors, end-users, and software developers. These specifications define the interface between Clients and Servers and Servers and Servers, including access to real-time data, monitoring alarms and events, access to historical data, and other applications.

Research and implement AT commands set for LTE-M, GPS, and other modem interfaces

To build these interfaces, it will be required to research and implement the AT commands to access these modules' various workings and data. The AT commands are generally specific to a specific modem and would have to be implemented differently and separately as each of the modules has its subset of AT commands. (Em and Module, 2012)
In this project, we will also implement a GPS module. The module we will use is the U-blox GPS module. We will transact with this module via one of the UART ports of the Microcontroller. To implement this module successfully, we will be required to research the NMEA commands structure. This module is an integral part of this project to implement asset tracking.

Research the OPC UA, COAP, MQTT protocol
This requirement will be acquired by researching the above topic to understand how the various protocol operates so that the developer can translate this operation into C-code routines for the Microcontroller to parse relevant or requested data to the higher-level application. This research will research the OPC UA protocol in-depth as it has many benefits in the 4IR space(Veichtlbauer, Ortmayer and Heistracher, 2017). This research will investigate the possibility of the embedded device acting as component/client/server in the OPC chain and or implement as an OPC UA wrapper. It is also valuable to research similar protocols to OPC UA to give context to the research; for this purpose, we will explore COAP and MQTT.

1.1.7 Research to implement software Modules

This research aims to achieve a barebones approach to designing asset tracking devices using different communication devices to interface into microcontrollers, capture data, buffer data and transmit it with minimal data cost.

a. the research will embark on design to capturing data via different interfaces which will handling data coming in at different rates from different communications interface

a. we will look at arbitration and priorities of the different interfaces and the microcontrollers handling direct io. For this, the research will investigate using and implementing the Nested Vector Interrupt controller. (Yongde *et al.*, 2014)

c. in terms of data transmission, we will experiment with different ways of sending data and examine the cost and reliability of transmitting data via the chosen methods. With the purpose of the research to add to the data value change, this research will also look at getting data from field devices to the microcontroller device via wireless technologies like using nrf2401 device or using LORA to fetch data from field device on the WAN for use by the high-level systems and API.

d. This project will focus on research methods in acquiring data, processing this data converting it to a data algorithm for transmission. The project will also look at efficient buffering and storage techniques as not all the collated data will be transmitted. I will embark on a journey to solve these issues using a 32-bit micro-controller as an asset tracking and data acquisition device.

This research will employ an embedded finite state machine to control the different transmission modes based on predetermined states. In terms of data transmission, we will experiment with different ways of sending data to examine the cost and reliability of sending data via the chosen methods. We will also look at the cost of transmitting data using different protocols, namely UDP and HTTP. We also look closely at the interface to an LTE-M module or alternate systems to relay the data. The research will also look at optimising power consumption for devices implemented with a battery source.

In terms of coding the different modules, this research will use two coding languages, namely C and C#. the two Integrated Development Environments(IDE) used in this research will be

Keil Uvision for the C Language and Visual studio for the C# implementation. This project will use SQL extensively for implementing database interfaces.

This research will also investigate the different wireless technologies available for data transmission suitable for this research's objectives and implement one of these research solutions to test the implementation of that technology onto an embedded platform for data transmission. This research will present models of the different protocols or simulation results or analysis

e. This research will also include rf interface incorporated into the device to receive additional data, data that the process control system does not cater for but is required by the high-level process or system. This will also include remote data like monitoring outflows of a Waste Water Treatment Plant rate. The two platforms investigated for this functionality will be a local LoRa node with transmitter and receiver configurations. The second platform to get the data will be a 2.4GHz rf module or a nrf24e interface to the Microcontroller via an SPI interface.

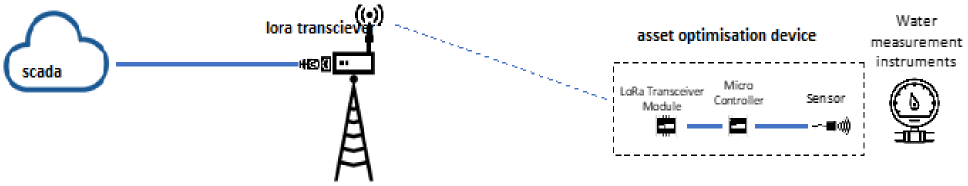


Figure 1.2: Diagram of parsing sensor data using LoRa end node and Gateway

1.1.8 Research Questions and Hypothesis

Test the mix of using different technologies to achieve low-cost SCADA and PAC systems without compromising security and controllability.

Can we achieve a reduction of data cost by employing effective techniques in terms of buffering as well as developing a data algorithm? Explore Different transmission protocols to understand their application and scalability.

Can data be parsed from different data sources to remote central platform or Gateway for visualising? In addition, is there a need for a generic packet format structure?

Which is the most cost-effective data transmission protocol to use for industrial sensor monitoring?

How relevant are the OPC UA protocol in the constrained network environment and other associated protocols?

How does OPC UA compare to other transport layer protocols?

1.1.9 Investigation of Real-time IOT Devices

The concept of real-time asset tracking has been around for years now and has been employed to different assets for different purposes. In South Africa, however, very few researchers, developers, or students have embarked on this field, and if they did, they have adopted the rapid development platform to implement these solutions. In this research proposal, we investigated the different types of asset tracking and monitoring. This research will review these types and combine various asset tracking and monitoring techniques for asset optimisation.

Because this design touches on several different technologies ranging from SCADA to embedded systems to communication processors to protocols and high-level application, this research has and will research work that has been carried out on all of the streams. This research is a multidisciplinary stream. The study also includes a proof of concept of the implementation.

The SCADA servers drive process automation communications and, in most cases, use protocols suitable for direct connection to process equipment, including Modbus, Modbus/TCP, and Ethernet/IP. These protocols are efficient in the process environment and operate best on high bandwidth reliable channels. (Gutierrez-Guerrero and Holgado-Terriza, 2019)

Process automation solutions use PAC or PLC devices to automate process control through connection with field devices. PLCs' essential characteristics in the process environment include determinism of control algorithms, I/O performance, and interconnectivity with field bus technologies.

The communication aspect of process automation is essential, with the solutions building robust local area communication systems as a service for the PLC devices. A typical process automation communication solution distributes PLC devices in a local area, providing sophisticated control at each field location.

This research embarks on both a low-level implementation and a high-level implementation data strategy to encompass a holistic approach in the Remote SCADA environment. What is meant by the above statement is that in terms of the low-level data strategy, this research embarks on the design and implementation of an embedded device that will engage with field devices via serial interface, engage with a process control system via communications processor, store the acquired data in local buffers then concatenate these buffers through serialisation process and transmit this data using LTE-m. (Uwe Steinkrauss, 2010)This embedded device will also attempt to serve as an embedded OPC client for high-level OPC servers. Low-level implementation will also include a finite state machine implemented on the device to manage data transmission based on different asset states.

This research adopts a barebones approach to implementing solutions to develop software libraries and better understand the technologies present. It has been a trend in "third world economies "to implement solutions using rapid development platforms, this has resulted in not understanding the technologies and creating a lag in terms of development of new products because we do not have our libraries to develop new products, this will also promote the entrepreneurial strategy of south Africa.

Some of the research investigated may not at first glance seem relevant to this investigation because of the multidisciplinary pronged approach of this study. Still, all literature reviewed contributes to the final product of this research.

This research, as mentioned, already implements an embedded solution to the SCADA environment that, on many levels, relates to hardware implementation of the intended outcomes. Still, a lot of the research entails investigating different protocols and software integration. The `study lays the foundation for a platform for a myriad of applications that will only require an adaption or replacing some of the modules with other technologies.

One research research implemented a vehicle tracking system for a school bus incorporating Biometrics, GPS, and GPRS also implemented on an Stm32f discovery board. (Bangali, Dr. S. K. Shah, 2015)The approach is fundamentally different and the objectives, but a lot of the work covered is very similar to what we want to implement. The biometric module implemented in this project is used for student identification via fingerprint identification and is verified by EEPROM stored data. The GPS and GPRS Modules are used for vehicle location and internet connectivity. Of particular interest in this research was implementing the communication interfaces, how the buffering was implemented and the data made available for transmission.

This research mainly looked at this research because of the number of communication interfaces implemented in the design. Also of fundamental interest was the data handling capability of the device, as it incorporated a biometric fingerprint module that processed a considerable amount of DSP data. The other area of interest was the Nested vector interrupt controller of the device to hand so many communication interfaces simultaneously yet still functioning to carry its core processes.

The transmission protocol used for the transmission of data is HTTP. This device is developed on a higher application layer than the device we proposed. We also foresee high transmission costs with HTTP as whether you send 3 bytes of data with HTTP or whether you send 80 bytes of data, you are billed for the same with each HTTP session. We are proposing encapsulating data as well as using plain UDP for transmission, thus reducing. We will also provide an analysis of this in this dissertation.

The microcontrollers over the last decade have evolved quite substantially and had capabilities of much more communication interfaces with Advanced DMA capabilities; this allows for interrupt or DMA-based communication interfaces. This thesis will explore and analyse the implementation of a Serial communications interface using DMA and an interrupt-driven controller. (Ekiz *et al.*, no date) During the literature review of this research, the GPRS interface was of particular interest as our research delves into LTE-m, which has significant similarities in terms of the implementation/interface of the module into the Microcontroller.

(Yongde *et al.*, 2014)The design of the Controller for multi-Layer parking controller based on the STM32 was researched based on the successful implementation of the CAN module and the use of the Stm32 arm microcontroller used. This design was based on the predecessor microchip, the STM32f103, instead of the chip we plan to use, but the baseline architecture is very similar. Thus beneficial information was derived from researching this design.

The review of this design lead to understanding the power of using the DMA when implementing the CAN Module as well as the architecture of the STM32 Microcontroller in terms of using the CAN mailbox and the DMA without engaging the microprocessor, thus being able to write macros to realise some of the control functionalities for the CAN Module.

The second part of the literature review investigated data portability from the process control system to the higher levels systems ranging from the PLC/PAC to the SCADA systems to the database management systems residing in a remote data warehouse.

(Kim and Sung, 2018)(L., B. and J., 2015)Standalone OPC UA wrapper for industrial monitoring and control systems. In this research, the OPC UA, an open communication architecture for the manufacturing industry, enables data exchange and management among different industrial automation entities. Of particular interest in this research was that it was designed as a stand-alone wrapper. The wrapper consists of two main components, i.e., UA server and classic client. A critical feature is employing a distributed component object model runtime library implemented in Java for platform independence. The research focuses on non-windows based solutions using low-cost microprocessors as our research will focus on the .net platform tied down to windows applications. The focus of our research is to enhance the data value chain. Making the data portability seamless from lower level system to be integrated into high-level systems with significant cost savings and eliminating the need for expensive legacy SCADA systems.

Another essential component of the research was investigating different or similar protocols or engines to provide seamless data portability. For this the research Evaluation
Performance Evaluation of M2M Protocols Over Cellular Networks in a Lab Environment. It is necessary to investigate the transmission behaviour of standard protocols for machine-to-machine (M2M) communication concerning the peculiarities of cellular networks. In this research, three M2M protocols – CoAP, MQTT, and OPC UA – are compared to each other about their transport mechanisms to evaluate the transmission times and analysing potentials for optimisation. It has been shown that OPC UA achieved the best test results – although OPC UA has the most considerable protocol overhead of all evaluated candidates. (L., B. and J., 2015)This is because OPC UA has the most suitable protocol design for cyclic data transfer. Especially in the case of LTE, the transmission time depends not only on the total amount of data but also on the exact sequencing of data transfer. This has been observed in the evaluation of CoAP. Its implementation of reliable data exchange is not suitable for the transmission of large payloads over cellular networks. Protocols based on TCP achieve better performance due to TCP features like windowing.

1.1.9 Research Design Methodology

The research methodology used in the project is Applied Research Methodology. The research will address problems the industry and parastatals face concerning asset monitoring and optimisation; it will look at how technology can bridge the asset life cycle optimisation gap from an embedded systems perspective. This research can be broken down into three areas: Research software techniques, researching the hardware, and the communication interface. And the third phase is once a working prototype is produced to monitor data transmission using different data transmission techniques and protocols.

This research aims to achieve the necessary knowledge and a proof of concept to provide a baseline to develop or adapt to achieve a device capable of acquiring data and making process control data portable, affordable, and necessary in the data analytics domain. This research will explore all solutions and architecture of making data available for use by the high-level system for data analytics and augment existing support systems to add to the data value chain and improve the business process.

One of the industry's perceptions is that data is too expensive to benefit data analytics. This research aims to clear that perception by implementing a solution that can control data cost and scale the data according to the environment.

As the world embarks on the 4th Industrial revolution, there is an increasing need for experts in the integration environment. This research embarks on gaining a knowledge base to integrate process control systems to high-level systems directly from automation systems or RTU's. This research will look at the different drivers and the likes to see how seamless integration is explicitly achieved in the .NET environment. (Architecture, 2018) The study is essentially an investigation into software, hardware, and existing systems integration to provide a cost-effective hybrid solution to reduce data cost and consumption.

We will use different IDE platforms and software interfaces like Kei Uvision, Visual studio, STMCubemax, MS SQL, etc.

This design essentially will have three major components:

Interfacing the unit to an existing process control system that is at level 1. Acquire level 0 data that might not be included in the process control data. Acquire any other data that might be available nearby.

The device itself will then buffer the acquired data in appropriate data buffers. Implement a finite state machine to control data transmission rates based on the states of the assets.

Serialise the data. Provide security Interface into Lte-m module for data transmission to the remote site. Transmit the data to a remote location and or data-warehouse. De- serialise the data into data tables to provide for integration into high-level systems. Figure 1 below provides a high-level overview of the research proposal.

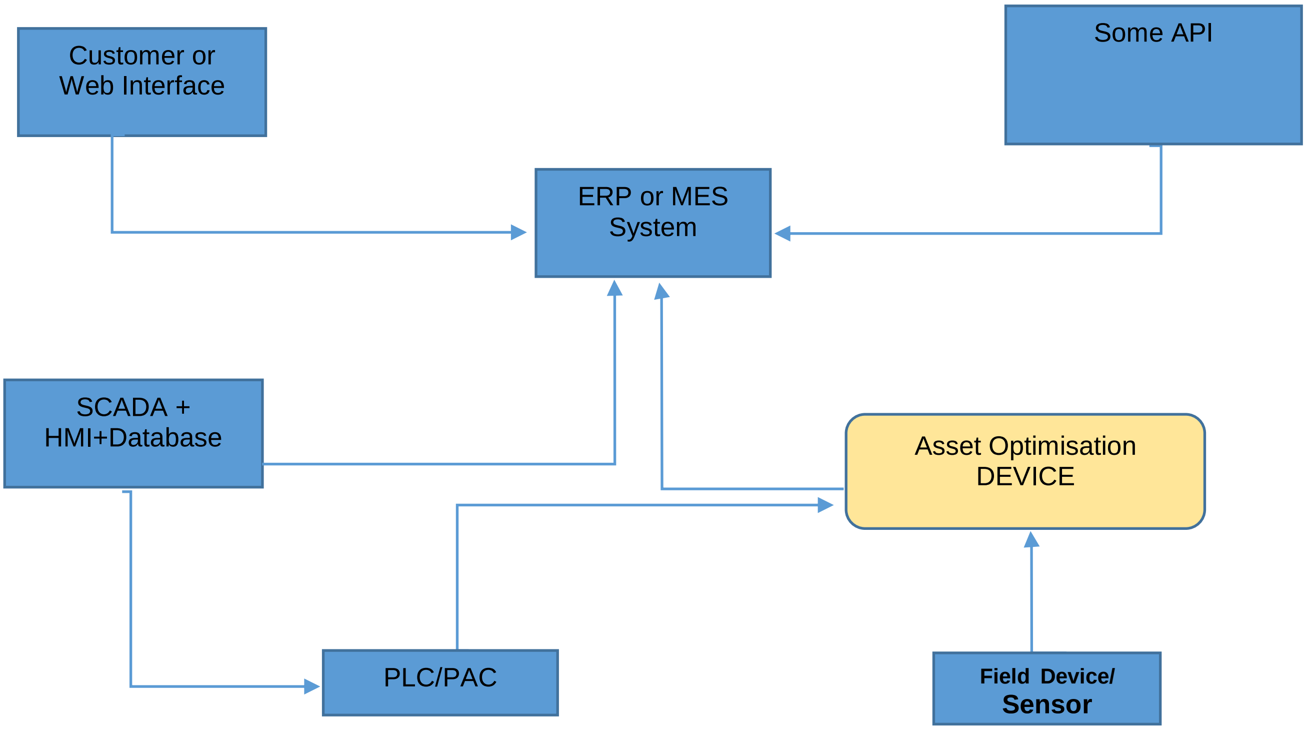


Figure 1.3: High-Level overview of Research Proposal

1.1.10 Literature review and protocols Investigation

A literature review was conducted to understand the present and future available technologies and apply and improve on techniques and technologies researched.

The project requires expert knowledge of embedded design and to be able to program a microprocessor in the C-language. C is a very efficient language that allows a programmer to access the inner workings of microcontrollers. The project employs expert knowledge of the implementation of buffers and numerous modules competing for CPU time. We will also employ advanced c programming techniques and use Finite state machines to implement our code.

Apart from embedded systems programming, it was also essential to programme in various high and mid-level programming languages. Because of the steep learning curve of this research and the different layers of the data value chain, it was essential to structure a plan to carry out the investigation and can be summarised below:

i. Investigate different protocols used in the SCADA environment

Research SCADA protocols like DNP3, Modbus TCP, etc., and determine how they can be translated into a format that will allow data portability through the data value change.

ii. Research Other transport protocols.

Researching the different protocols and implement an embedded variation of the protocol is probably the most defining factor of this project. This research will predominantly investigate the OPC UA, COAP, and MQTT, compare them from an embedded perspective, and implement one or more of them on the device. The study might also delve into the technical breakdown of all of these protocols but will undoubtedly cover the OPC UA and MQTT protocol.

iii. Hardware Implementation

The stm32f discovery board uses a 32-bit arm microprocessor. The datasheet of this development board is 1800 pages long. An in-depth understanding of the datasheet of this development board is required to implement the functions of the board. There is also fantastic support available for this board and similar architectures on the St Microelectronics Website. The 32-bit microprocessor also has numerous peripheral interfaces, all of which will be used to implement this design. The greatest challenge in the hardware will be processing all of the peripherals simultaneously with the arm microprocessor board.

This module is implemented to pass and receive data and commands to the control centre. This communications interface will be implemented via an interrupt-based UART.

This research will also investigate different communication bridges, especially Ethernet or wireless protocols, because of the need to efficiently transport the data acquired to remote locations and with decreased data latency.

Different communication devices have different supply voltages and power demands; we will also investigate this compact working device requirement.

Research the HART protocol as well as the 4-20 mA signal available at field devices to send instrumentation data. This research will explore converting the 4-20 mA signal into data that can be relayed directly or converted to Ethernet without compromising the resilience of the 4-20mAsignal. Also, have to give energy to developing a wrapper or bridge to translate data into Ethernet while passing the data in the required format that allows integration into the high-level systems. The research of the HART protocol in terms of converting this data to Ethernet/OPC UA or any other useable protocol.

iv. Research data acquisition techniques and data algorithms

In many instances, data is a function of two or more signals which can also be referred to as data aggregation. The research will look at various techniques like lookup tables to generate data for parsing to different modules in our microprocessor and transmission to a remote location/centre. There are numerous ways of buffering and storing data but will not venture into these techniques but employ techniques that I am already familiar with through tacit knowledge gained over the years. We will, however, explain the technicalities we used to employ these techniques. In acquiring the transducer/field device directly from the source, we will need to look at methods to translate the 4-20mAsignal into a voltage signal that the Microcontroller can process. The challenge here is to keep the robustness of the field device intact. This could also mean that this research could explore ways of offering diagnostic data of the field device to enhance the data capability of the device to be on par with the Hart devices. In essence, the research will make an affordable, intelligent field device that can directly interface to the Microcontroller, which will pass on the data value from a SCADA system or RTU device.

v. Develop communication interfaces for the Microcontroller.

This research phase will investigate the different communication interfaces required to implement a universal remote asset-tracking device. This could essentially translate to three types of communications interface, varying from Uart driven interfaces to SPI/I2C serial interfaces. The main areas that research will investigate can be broken down as viz. Serial communication interface that will secure data from a process control system via a communications processor, serially acquire any other local data or signals that may not be available via the process controller (like PLC). The third phase of the research will be to serially interface the data available in the buffers and relay it to the LTE-m module for transmission to a remote database.

The infrastructure in terms of wireless communication is certainly not lacking concerning the number of options to use. There are more than three types of major role players, and the competition is heating up. This research will analyse the different transmission technologies to understand their differences, shortcomings, and strengths, implement the GSM/LTE-m interface in this project, and analyse this technology in detail. The other role players that will research to give some perspective are NB IoT, LORA, RPMA, and LTE-M. These are all LPWAN technologies. (L., B. and J., 2015)

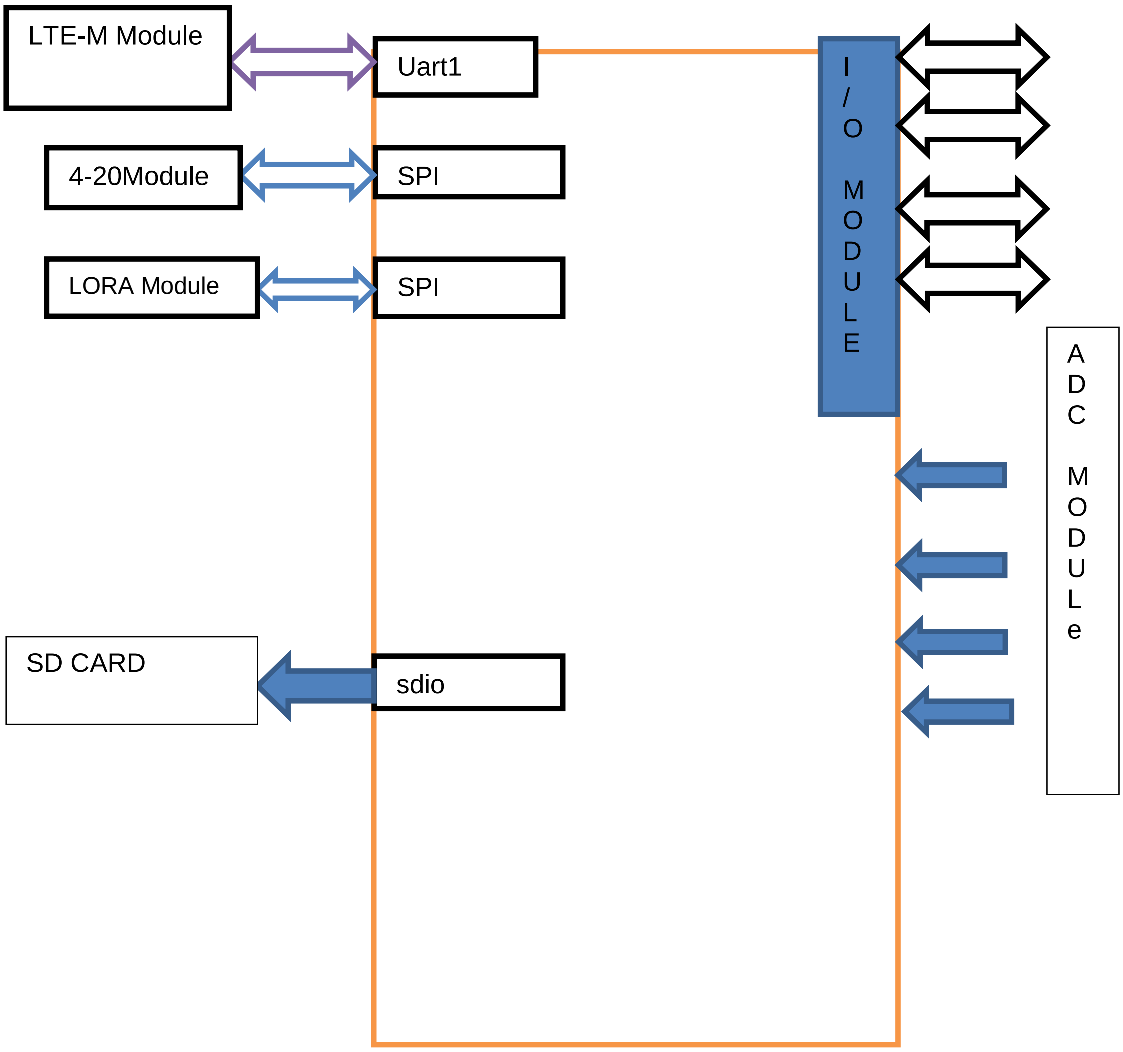


Figure 1.4: Hardware structure of the proposed logger with LoRa Radio Receiver Unit with LTE-m Interface.

1.1.11 The Research and Project Plan

i. Research Different protocols and Getting acquainted with the development boards

The development board used for this project is the stm32f discovery board. The Stm32f4 is the primary board that the project will be constructed on. Familiarity with this board is essential, and therefore it was chosen for this research and will be the foundation of all research conducted. The datasheet of this board is approximately 1800 pages long and requires an in-depth understanding. It essentially translates to lead times in getting acquainted with a particular microprocessor. On average, the datasheet of a 32-bit microcontroller is more than 1500 pages, excluding the technical notes. The stm32 development board is a very inexpensive board with a very good support base. Understanding of the architecture of the board promotes optimised usage of the board. It uses an ARM M4 microprocessor with a Jtag interface for easy flash programming. The support base from STMicroelectronics is excellent and has numerous application notes for this platform.

ii. Literature Review

Review similar designs and researchs similar to the design one is implementing is fundamental in any design. This could save a lot of time and point one in the right direction. It also lends itself to giving one idea in terms of design and implementation and gives one perspective in applying the method in the industry. This research will review similar techniques as well as designs using similar hardware and architecture. A good portion of the literature review will cover research researchs covering the different protocols in this research.

iii. Writing and Testing Code.

Writing code will take up a significant portion of this research mainly because most of the code implemented in this research is self-written and includes libraries and interfaces written over years of software development and embedded design. Numerous modules will need to be adapted and would thus require testing as well.

This process will be implemented on a modular basis and will be ongoing throughout the project; the modules will be as follows:

- Coding of the input signals using the ADC module of the Microcontroller. We will employ a multichannel ADC system reading multiple input signals with one read instruction. It is coded in C on the Stm32f4 device.
- Translating the 4-20mAsignal into a variable for data transmission. Coded in C using SPI interface with off-the-shelf 4-20mAmodule.
- Output/actuator signals like info, PWM signal to Power electronics module for any control that we required the device to implement
- Implementing SCI module. There will be various interrupt-based uart modules
- Display information and fault condition coding.
- There is a requirement for buffering as well as storage, and not all info will be transmitted.
- Implement the GSM-2G/3G, LTE, and OPC interpreter modules.
- Re-writing of the firmware of legacy GSM loggers in new data format. Written in C.
- Capturing the different data sentences and developing a data algorithm combines all these buffers into one data sentence.
- Writing of various wrappers to enable data portability between devices and interfaces.
- Visualisation of data, both at the local device level as well as Remote MES Level. Written in C# using Devexpress as well as Node-Red For backend server of LoRa.
- Writing framework of device gateway to process data from different data sources

iv. Hardware development

Existing hardware of the GSM loggers that were previously in commission will be reverse-engineered and will write the new firmware to log data into a new data framework and format. The only hardware development that will be done will be the power supply module and the level shifters interface. All the other hardware will be off-the-shelf modules; this includes the stm32f discovery board, LoRa module, and the LTE-m module. However, the challenge will be to integrate all these modules on one system with different power requirements regarding the voltage supply, primarily because of the number of radiofrequency interfaces employed. The other challenge in terms of the development of the hardware will be EMI. The complete hardware development will be done on a proof of concept basis with different modules connected.

Prove the concept by implementing DATA acquisition and transmitting selected/filtered data to the central control centre.

The process of bench testing will be applied to prove the concept of the above project. It will require to display of output waveforms that would typically drive the actuators. We will require a Graphic USER Interface to remotely display Dash Board type data displayed a remote data centre relayed via the GSM/LTE network. This graphic user interface and visualisation will be implemented in this project, but data will be visualised remotely via a web application.

Prepare Final Presentation. Compile Research into dissertation Document.

This process should take about 3 to 4 months. The document preparation will be ongoing and will be finalised once the hardware is developed and tested.

1.2.1 Structure of the Thesis

1.2.1 Research deliverables

This research will deliver a barebones approach to designing and implementing complete architecture to improve data integration into MES systems. The study embarks on both software and hardware design to converge IT and OT environments into seamless data integration.

i. this thesis will implement different ways and methods of interfacing various industrial sensors into a microcontroller that can be translated to data variables for wireless transmission. This research project will focus on methods for acquiring data, processing it, converting it to a data algorithm for transmission, and interfacing a 4-20mAsensor to a microcontroller as other variations of sensor interfaces.

ii. Connecting devices like Loggers requires a wireless access network as an extension of the Metro Area Network. Connecting many edge devices with low bandwidth requirements using optic fibre or point-to-point wireless is neither time nor cost-effective. The best available solutions utilise narrowband wireless technologies, which enable various flavours of LPWAN.

This research will deliver how best to provide wireless connectivity for a large number of devices that cannot (or need not) be connected using cables to enable their connection for service.

iii. Develop a UDP based framework to receive data from different data sources to be visualised by an MES/ WEB API

iv. This research will look at options for transmitting data using different protocols in the IIOT space

1.2.2 Delineation of the thesis

The interfacing of the various modules poses numerous design challenges, including hardware, software, and electromagnetic interference.

This thesis does not cover the vivid visualisation of the data at the web interface/ but will have a good visualisation merely for proof of concept. It must be realised that only specific modules will be covered in detail because this thesis purview is from the sensor to dashboard in terms of proof of concept. In contrast, some phases of this research will be covered conceptually.

Most of this research presented in the thesis is either hardware or software, which presents a challenge in explaining all the work accomplished to finalise the design of the modules. Only one of the hardware modules will be covered in detail which lay the precedent for all the other modules covered in this research. It must also be noted that some modules will transmit dummy data to prove the concept in some instances.

Considering that this project carried substantial development costs that were self-funded and proved debilitating, sending dummy data instead of actual sensor data like implementing the LoRa end nodes and the GSM loggers.

The design was implemented on a proof of concept basis and as a development prototype. Most of the context and the deliverables of this research lie in the coding of the different hardware modules and the frameworks and Visualisation software written. Only snippets of code are captured in the appendices of the thesis. The various other data sources and technologies are challenging to explain to a third party because of their geographic dispersion, but a real-time map and the API in chapter six will help contextualise this.

This design includes various modules and architectures to bring about a device and framework that serves multiple functions and benefits from asset life cycle optimisation to MES applications.

1.2.3 Contributions of the research

The significant contribution of this research is to present process control data at different levels of the data value chain at reasonable data cost. It allows for contributing to an organisation's data warehouse to provide source data for data analytics and other high-level systems. It also will enhance any Artificial intelligence systems providing intrinsic time series data for trending and self-learning systems.

This asset optimisation device will provide data that was not previously available from the process control systems or the PAC that cannot offer due to lacking hardware capabilities. The device will provide an affordable means of providing this data to the required applications.

This research also provides a framework and common platform for multiple data sources to communicate using a standard packet format.

Portability of data is also a pivotal contributor to this research; with many open-source applications and data analytics gaining momentum in the 4IR space, there is a need for seamless integration of data between systems and applications. This research will also provide the foundation to provide automated process control systems to the point of un-manned systems. Integration into other systems like maintenance management systems can also improve lead times to respond and repair and provide plant-assisted diagnostics where the process control system can report on its state of different components and processes. The data era is here; how one uses, it can be a difference in staying in business.

1.1.4 The Significance of the research project

Significant research has been undertaken concerning SCADA systems as well as different ways of acquiring data remotely, but not much research has been conducted in terms of having a device to receive data from multiple platforms and transmit it to remote data collections agents or device gateway for high interoperability. A device that will serve as an asset tracking device will also serve as an automation data interface to assist data analytics and life cycle/asset management optimisation in South Africa.

This project aims to build the foundation to generate and automate data from expensive and critical assets that can optimise the lifecycle and operational effectiveness. Many assets have high maintenance costs in terms of the Internet of Things, and acquiring data from these devices and offering some controllability in many instances will improve maintenance costs and optimise the asset life cycle.

In this particular project, there are also numerous safety and operational spinoffs. The safety spinoff, especially in unsafe plants either operational or have some form of hazardous implications, can reduce the requirement of personnel operating in those zones as the data can be acquired and controlled remotely if needed. Operational maintenance staff can interact with plant personnel more informed, thus resolving plant issues speedily, especially plants geographically dispersed from the maintenance depots. The turnaround time can be significantly reduced, thus improving maintenance KPI as well as an asset life cycle.

Another significant contribution of this research is to make process control data available to 3rd party users for Business operations optimisation and Dashboard Visualisation to consumers and users of the respective services provided by these process control systems.

This research intends to significantly change the mindset regarding present development platforms in the South African context. It also wants to demystify the role of IoT devices concerning asset life cycle management to implement an asset tracking device to collate data to intelligently use it for the remote data centre and data analysis for asset life cycle management.

This research intends to lay the foundation of getting data to drive business, making use of process control data and plant performance indicators to augment KPI's. Also, making OEE data more measurable, especially Municipalities and State-owned enterprises, and other large organisations to have an accurate, objective data model.

Very few research projects have been undertaken to implement an asset tracking device that will serve as an asset tracking device that will also serve as an automated data interface to assist with lifecycle asset management optimisation in South Africa.

The significance of this project is to re-align the context automation data from devices that impact the asset life cycle, which can also assist in improving the business process by getting additional data to MES and ERP systems.

Getting this data will also assist with keeping third parties, interested parties accurate trended metadata, especially when coming to service delivery standards.

1.2.4 Perspective of this project

It is crucial to put the design approach employed in this design project into perspective. The growing trends in developing new products utilise rapid development platforms like Arduino, MBED, and the likes. These platforms certainly have their place in the industry. Still, the underlying issues in developing countries like South Africa take away the opportunity to understand the building blocks of microcontrollers and embedded system engineering. In these countries, we find developers merely adapting first-world country's design to suit their environments. They simply use libraries written by their 1st world counterparts in their applications without an in-depth understanding of the hardware's architecture. For these counties to acquire this deep understanding of the architecture, they need to code from the datasheet and write their libraries using low-level coding languages to become pioneers/experts in developing embedded products.

This will also give them an in-depth understanding of persistent problems in these countries regarding embedded systems without the need to pull in foreign help to solve some of their design issues. During this design, one of the key objectives was to write "our own" libraries to implement this design, which was accomplished quite successfully.

1.2.5 Research Methodology

*****

The research methodology used in the project is Applied Research Methodology. This research looked at problems the industry, particularly parastatals, face concerning asset lifecycle optimisation. The study also looked at how technology from an embedded systems perspective or Internet of Things perspective can bridge the gap or enhance asset life cycle optimisation and enhance the end-user experience. Much of the research methodology comes from an electronic engineer's tacit knowledge through a long career in asset maintenance.

Bridging the gap between a process control system and IoT. PLC, SCADA, and web services.

This research aims to bridge and marry the worlds of software development, embedded systems design, and hardware implementation. This research implements progressive phases of action to achieve proof of concept. It also incorporated low-level programming using the barebones approach and rapid development platforms like Mbed, Cubemax for STM microcontroller devices, and high-level programming in C# in the .net environment and python to implement system integration to enable web visualisation of the devices. Many existing and emerging protocols were investigated to achieve maximum data interoperability without compromising security.

Presenting the data collected, aggregating the data, and presentation and analytics. This research methodology was based on the premise and needed to integrate engineering process control systems, the operational network, and the IT network. The greatest challenge in this research was wearing different caps at the different levels of the data value chain. Technology is evolving rapidly; the challenge is how emerging technology experts keep abreast of all the technologies at the different levels of the data value chain while maintaining a high level of expertise. There are numerous solutions in the modern era, but the challenge is to put together reliable, accurate frameworks and reliable frameworks and will last the test of time and price.

This research can be broken down into three areas viz: Research software techniques. Researching the hardware as well as the communication interface.

1.2.6 Organisation of the Thesis

This thesis is organised into different chapters for ease of reading and separating the various technologies and modules. The breakdown of the chapters is as per the table of contents.

Chapter 1 is the proposal for this research and some preliminaries of this thesis.

Chapter 2 covers the theoretical research and literature review that was acquired during this project, This chapter

Chapter 3 is the Hardware implementation of the design and supports the technical implementation of the work carried in Chapters 4 and 5.

Chapter 4 is a research published using LoRa Physical layer to parse data to a SCADA system.

Chapter 5 is a research covering visualising data in central gateway and web API from different data sources.

Chapter 6 is the Testing and results of some of the technologies implemented, mainly LoRa Wan Architecture.

Chapter 7 is the conclusion and future work that could emanate from the research.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

The inherent need to get data to High-level systems has created an opportunity to investigate available technologies to produce viable and feasible solutions to add to the data value chain. There are presently many vendors in this space capable of supplying appropriate solutions using present IoT technologies but in the form of propriety solutions. In the open-source world, where the key motive is to make data on all levels more interoperable, there is an exponential increase in knowledge and innovations that, if packaged correctly, can lead to innovative solutions to achieve the objectives of this research.

Part of the literature review process was to understand the transition from legacy systems to new modern technology architectures. Data interoperability also required a complete understanding of the sensor-level data value chain to high-level application software. Each transition level of the data value chain has different implementation and technologies that differ in cost and complexity. Sometimes it requires a hybrid solution to provide the correct fix or solution. Much research is conducted in this space, but most of the research undertaken seems to cover off-the-shelf implementations and rapid development platforms. Even though this seems to be the growing trend, this is also very debilitating as a lot of the building blocks are lost if too much of the research is focused on of-the-self and rapid development platforms.

The literature review of the thesis can be broken up into different modules, just as the thesis is divided into different modules. The literature review, on many occasions, will often present other or similar technologies but may not be implemented in the final proof of concept but was necessary to understand the technology and implement the most appropriate solution/technology in terms of achieving the objectives of this research.

It was necessary to explore hardware, software, and communication protocol options to achieve the outcomes and knowledge to achieve the proof of concept of this research. The literature review purview of the thesis was first to compare the different wireless technologies, namely LoRa physical layer and 2.4Ghz Nrf24l01 radio modules options for transmitting sensor or process controller data in a plant environment for a range greater than 200m, followed by understanding the chosen wireless technology in this case LoRa. Then further investigating the different LoRa options, namely the LoRa node to node Transmission and the LoRa WAN option.

To get the full benefit of the LoRa technology, it was necessary to understand the founding and operational principles of this LPWan technology and its applications and its closest competitors. LPWan technologies also target the IoT world, which implies a level of integration to web-based systems, which also leans the research towards understanding and applying different transport layer protocols to enhance the data value chain from sensor to web visualisation.

Choosing different hardware platforms was also a great challenge from an embedded perspective as there are many options to choose from. In many instances, platforms that researchers and developers are already familiar with are customarily chosen. Familiarity with architecture is usually the key driver in selecting a specific microcontroller type, and this was certainly the case in this research. The platform selected for this research was mainly the STM32 and Arduino(ATMega) microcontroller family.

A literature review was conducted in the various low-level hardware interfaces used in the design and can be categorised as follows:

i. Understanding SPI

ii. 4-20 mA interface into a microcontroller

iii. Reading analogue value using a 0-10V sensor.

iv. Interfacing the LoRa radio Module.

v. Interfacing the nrf24l01 radio module.

vi. Interfacing the enc28j60 Ethernet controller.

vii. Interfacing cell phone modem.

viii. Working with and deploying LoRa wan Gateway R3000 Robustel.

A literature review was also conducted for various software modules and communication protocols and frameworks to realise the objectives of this research and can be categorised as follows:

i. Researching communication protocols relating to process control systems like OPC UA, MQTT, COAP

ii. Researching the LoRa Physical layer and LoRa WAN architectures

iii. Exploring frameworks for web visualisation.

iv. Researching SQL frameworks to optimise data visualisation.

The next phase of this research looked at getting data in a modern format like OPC UA and MQTT. The study was also conducted to understand different technologies in this space. Part of this literature review touches on the OPC UA, MQTT and COAP, and AOQP and how they fit into the IoT space in terms of the data value chain. Some of the low-level interfaces like SPI and DMA were also reviewed.

2.2 Serial Peripheral Interface Protocol (SPI)

Microcontrollers have various interfaces and peripherals that enable interconnection to the internal microcontroller buses.

The Microcontroller has different serial interfaces that can process/parse this data. The most common interfaces used in the modern Microcontroller are SPI, I2C, UART peripheral, and CAN. In this design, the SPI interface is used to interface into the LoRa node and nrf24l01l modules to achieve the wireless transmission required.(Nordic, 2008)

The Serial peripheral interface is an internal interface to the Microcontroller that allows it to communicate with external devices.

This interface was essential for the implementation of interfacing several modules to the Microcontroller. Some of the different interfaces that used the SPI interface in this thesis is:

4-20mA interface to Microcontroller
Microcontroller to LoRa radio Module(Transmitter)
LoRa module to Microcontroller(receiver)\
Microcontroller to ENC28J60 Ethernet controller
Nrf24l01 to Microcontroller (both transmitter and receiver)

The grasp of the SPI concept is essential for any embedded systems designer. The SPI communication stands for the serial peripheral interface communication protocol developed by Motorola in 1972. SPI is a hierarchical synchronous communication full-duplex protocol amongst electronic devices that carries data signals in both directions. It also means that it uses separate lines for the clock and different lines for data to keep both sides in synchrony.

Four key pins make up the SPI interface, namely

Clock, Miso, Mosi, and chip selector manage to transfer data between two or more devices. A master (usually a microcontroller) and one or more slaves are part of an SPI setup.

The Clock pin serves as a coordination mechanism; it tells the devices when to read/write data.

Chip Selector informs the slave device that we want to exchange data; this is done by merely pulling the CS pin down. The devices share the same clock, MISO, and MOSI lines but have their independent chip selector in a multi-slave setup.

Mosi- As the Name suggests, is Master in Slave Out (output pin)
Miso- Is Master In Slave Out (Input pin)

SPI protocol uses four wires named MISO, MOSI, CLK, SS used for master/slave communication. The master is predominantly a microcontroller, and usually, the slaves are other peripherals like sensors, GSM modem, and GPS modem or, like in this design, a module like the nRf2401l module, etc. Multiple slaves are interfaced with the master through an SPI serial bus. The SPI protocol does not support multi-master communication, and it is used for a short distance within a circuit board (Marwedel, 2010). There are four i/o signals associated with the SPI peripheral

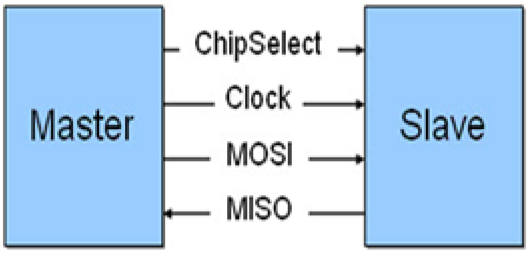


Figure 2.1 Serial Peripheral Interface

MISO (Master in Slave out): The MISO line is configured as the input in a master device and output in a slave device used to receive data from a slave device.

MOSI (Master out Slave in): The MOSI is a line configured as the output in a master device and as an input in a slave device wherein it is used to synchronize the data movement. This terminal is associated with receiving data from the master device.

SCK (Serial Clock): This signal is always driven by the master for synchronous data transfer between the master and the slave. It is used to synchronize the data movement both in and out through the MOSI and MISO lines.

SS (Slave Select) and CS (Chip Select): This signal is driven by the master to select individual slaves/Peripheral devices. It is an input line used to select the slave devices—Master-Slave Communication with SPI Serial Bus.

The SPI communication interface between the two devices is enabled when this line is active, and data exchange can begin.

2.3 4-20mAinput Interface

The most popular ways of reading analogue signals in the industrial process control environment are 0-10v and 4-20ma/0-20mA.Most industrial type process control systems like PLC’s or PAC connects field devices like sensors and actuators via 4-20 ma current loop directly into the process control system.

The 4-20 ma is a very rugged and robust way of processing signals in the industrial environment and has been found to stand against the harshest of industrial conditions, which is why it is so prevalent in the industry today. The 4-20 mA current loop is one of the most common sensor-signalling standards.

Current loops are ideal for signal transmission because of their inherent insensitivity to electrical noise. In a 4-20 mA current loop, all the signalling current flows through all components; the same current flows even if the wire terminations are less than perfect. All components in the circuit experience a voltage drop due to the signalling current flowing through them; this implies that the integrity of the signal from the instrument is not compromised even with severe voltage drops. The signalling current is not affected by these voltage drops as long as the power supply voltage is greater than the sum of the voltage drops around the loop at the maximum signalling current of 20 mA.

The 4-20 ma standard was adapted for process control systems, especially for signals with long wire runs and their robustness. Most standalone instruments are standard with the 4-20mAinput/output or HART communications standard. The other typical industrial process input /output instrumentation standard is the 0-10V.

The operation of the 4-20mAloop is straightforward: a sensor's output voltage is first converted to a proportional current, with 4 mA representing the sensor's zero-level output and 20 mA representing the sensor's full-scale output.

The prime objective of design is to transmit data from the sensor level up to the high-level system. On many occasions, there are high-end instruments at the source where data needs to be transmitted from; these stand-alone instruments have either a 0-10V analogue signal or 4-20mAsignal representing the sensor data. In industrial environments, the 4-20 ma current loop is more common for many reasons, not excluding the robustness of this signal in harsh conditions as well as the accuracy of data in long wire runs.

The 4-20mAsensor interface was designed for Industrial process control systems. Still, the challenge comes to transmitting these values using IOT devices as most microcontrollers in the IoT space are not equipped for a 4-20mAsignal as they process small voltages. (Azhari and Kaabi, 2001) The greatest challenge lies in converting this power-hungry signal to a variable for transmission using a wireless medium. The other challenge, especially in the IoT environment, is the power consumption of these modules, specifically where there is no permanent power source and where batteries are required. Choosing a suitable microcontroller technology depending on the environment and range required is a discussion for another research research.

The 4-20mAcurrent loop also boasts incredible immunity to electrical noise as the same current flows through all components, including poor electrical connections; the power source must maintain the current flowing through the circuit even in severely degraded wire connections. In the 4-20mAsensor environment, there are essentially two types of current loops, namely active(supplying power) and passive relying on loop power.

The operating principle of connecting a 4-20 mA signal to a scalable voltage signal is relatively straightforward. It is known that the same current will flow through the series circuit. The current output of the 4-20mAsensor circuit will always be 0 conditions as 4mA and maximum level as the 20mA where 0mA will be open circuit condition.

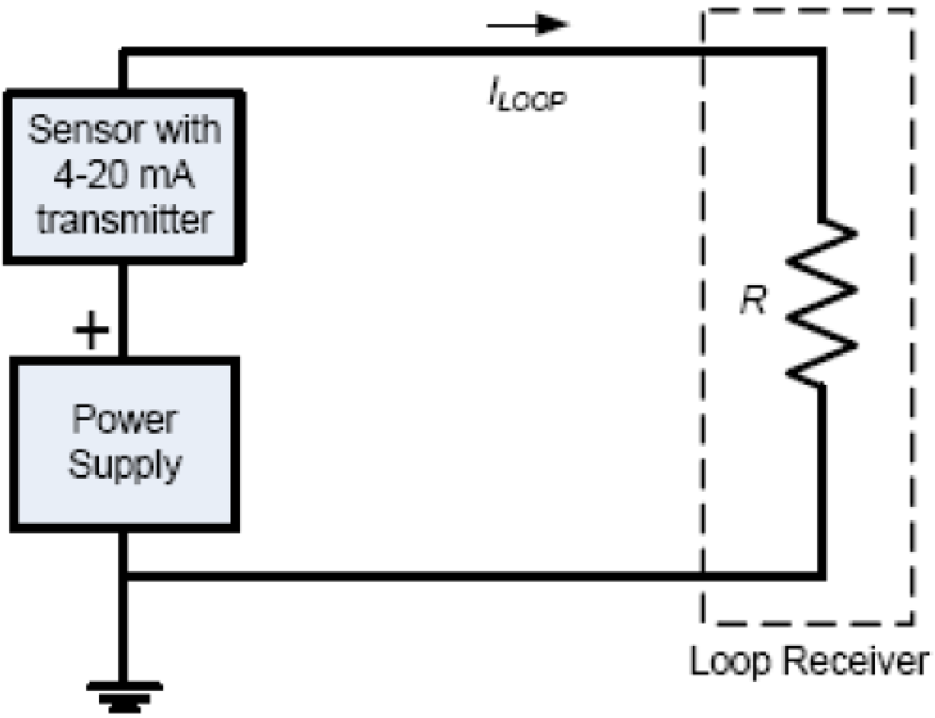


Figure 2.2 Diagram on the principle of reading the 4-20mAas the Voltage difference

The above figure 2.2 demonstrates the principle of reading a 4-20mA signal as a voltage difference for a microcontroller to process as a variable. In principle microcontroller would take ADC samples across the resistor; if a 250 resistor(R) was placed in series with the measuring circuit, using ohms law V=I*R, we could relate 1V to 4mA condition and 5V to 20mA(250*20mA). This would essentially be the volt drop across the resistor. Using the 2 GPIO outputs of the microcontroller and setting them up as ADC, we could easily read volt across the resistor and scale it accordingly and parse this scaled value as a variable in the buffer for the LoRa module.

The other option of parsing the data to the microcontroller via its SPI interface using an off-the-shelf 4-20mAdata interface. Converting/translating the 4-20mAcurrent loop to voltage signal is necessary. In the IoT space, most universal IoT controllers are microcontroller-based systems either capable of processing a 0-5v or 0-3.3V by its ADC controller peripheral. The key benefit of this SPI-interface approach is that a submodule would do all the heavy lifting giving the primary MCU time to do other essential tasks; all the MCU needs to do is process the SPI data whenever it is available or proactively query the interface from time to time. Of course, instruments and field devices have other field device protocols that can be parsed to the microcontroller, like the Hart protocol, which is extremely popular in the instrumentation environment. But this research implemented translating the 4-20 mA signal into data that any modern microcontroller can process. There is, of course, the option to translate the 4-20mAcurrent signal to voltage signal and have the microcontroller 12-bit ADC controller to process this signal. Still, this option will engage the microcontroller processor unnecessarily where a separate interface (as seen in Figure 3.3 below) with its secondary microcontroller can do this mundane task as per the module described below.

The other submodules that form part of this 4-20mAinterface(Figure 3.3) are INA196 current shunt monitor, MCP3201 12-bit ADC, and TPS61041 DC/DC boost converter. It receives output current (4-20mA) from the transmitter and converts it into a voltage (0.4-2V). Then through the AD converter sends a signal to the built-in microcontroller. This built-in micro then presents this data via an SPI interface to other devices that require this data. Accessing this data or values is done using the prescribed instruction codes and registers as per the data sheets of that interface.(Surachman *et al.*, 2011)

In the low power wan, implementing the industrial sensor using a 4-20 mA input signal can be challenging because of its power demands in the standard execution. The other problem in implementing this type of input is that a low-power Wan technology like LoRa is often employed with a battery source. The 4-20mApower supply requirements would need an additional battery to power up the sensor separately or some alternate solution to source the power to this sensor.

In the low power wan environment, the end nodes usually are power by a 4.5 V battery setup. In contrast, the 4-20mAsensor would require a voltage of at least 12v-24 V to operate adequately.

With wireless technologies available to parse process control data, there is a need to convert industrial sensor data to a voltage or data for this data to be transmitted, like the flow rate of the outflow of a wastewater treatment plant.

The principle in this research is to pass industrial data via wireless device to a SCADA system. The 4-20 ma interface used in this research is a single-channel current to voltage converter. As mentioned in other research researchs, there are options for parsing multi-channels 4-20mAinterfaces if the application requires. In it is explained in detail how to accomplish a multi-channel variation of the conversion.

The other method is converting the 4-20 ma current signal to a 0-3.3 Vage signal that the modern microcontroller can efficiently process by its ADC peripheral. When using the ADC peripheral, the signal processed by the arm stm32f microcontroller has a 12-bit resolution, meaning that 4096 samples were processed. This resolution plays a significant factor when scaling the values of the measured signal. Depending on what instrument or device is measured, this signal can be processed and scaled by the microprocessor and stored as a volatile variable to be buffered by the microprocessor and **parsed** to any function() or transferred to serial interface peripheral, which can be processed further for transmission. Now that the signal is translated/processed, the result is a variable stored in buffer/array in the microcontroller ready for transmission.

2.4 Reading the 0-10V signal for transmission

Even though IoT devices have interfaces for processing voltage signal sources, another popular signal interface in the industrial process environment is the 0-10V signal interface. It is not common for IoT devices to process 0-10v signals but rather 0-5V/ 0-3.3V, which is the standard concerning modern microcontrollers ADC inputs. To Process a 0-10v signal, there needs to be some galvanic isolation and level shifting between the microcontroller processing the signal and the sensor itself. There are numerous isolation and level shifting techniques employed in the real world to produce a robust interface, but it is outside the scope of this research. This research also needs to cover processing and to scale this signal for sensors encountered of this type.

An ADC is a peripheral that allows measuring the voltage (between 0 and V_{ref}) on a particular configured input of the microcontroller. The basic principle of operation is that peripheral samples the digital input or value of the signal received at specific intervals, and the peripheral stores an array in the order it was received for further processing with regard to the voltage reference signal. The resolution of the analogue signals is the number of samples taken per cycle. ADC resolution is one of the critical factors to determine how precise the conversion can achieve. If a chip has a resolution of 8-bit (0-255), it can detect 256 different input analogue signal levels. The stm32f4 microcontroller is being used has a 12-bit ADC mode, which increases the resolution of analogue conversion to 4096 steps (from 0 to 4095). This resolution is configurable to 12-bit, 10-bit, 8-bit, or 6-bit, where faster conversion times can be obtained by lowering the resolution.

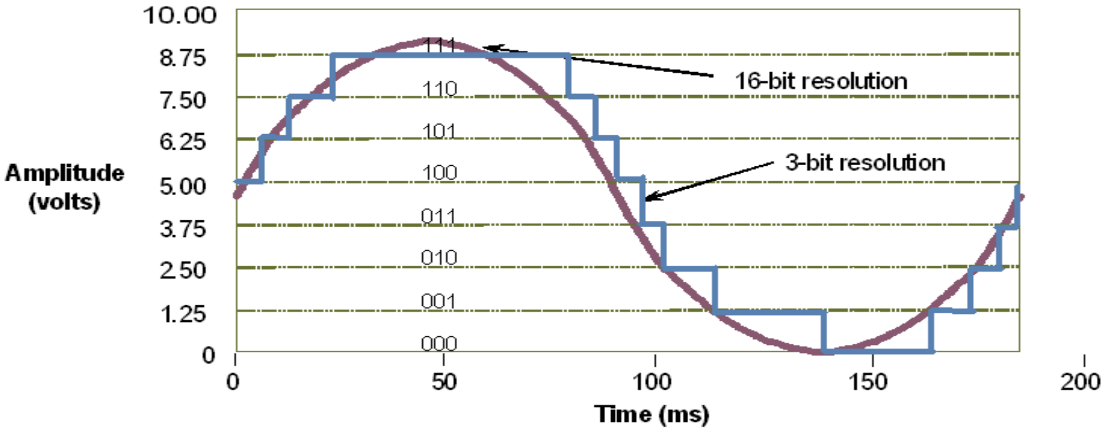


Figure 2.4 **ADC showing 3-Bit resolution and 16-bit resolution Signal [30]**

The STM32F4 has a built-in Analogue to Digital converter.

- Seeing that device will be using DMA to transfer the data from the ADC peripheral to the ADC Buffer. This will also need to be set up. In this design, the peripheral_base_address of the ADC is the Data register of the ADC the device is using. The memory address will be the array created to store the respective converted ADC values ConvertedAdcValue[]. This array will be volatile as the values can change at any given time and outside this code.
- So each index of the array will store one sensor value, the value of the sensor that the GPIO is reading. Each index of the ConvertedAdcValue will be copied to a static array, and each index of that array will be stored as global variables for use in the code.
- E.g., int V1=ConvertedAdcValue[0]; The static buffer created is used in the serialization routine for transmission to the remote control centre. The algorithm will use the variables created from this array in the source code to calculate the 4-20mAsignal.
- The ADC module can be used for determining/reading the voltages on either side of the resistor; the difference of these 2 voltages will represent the 4-20mAsignal from the sensor running hours necessary for the asset management.

2.5 Researching the wireless technologies

Internet of things is a collection of connected objects, sensors, embedded electronics software, and wireless connectivity protocols that collect and exchange information and data with wireless networks connected to the internet. The internet of things has produced opportunities and technologies that can be adapted for the process and automation industry to enhance the data value chain. Another key objective in optimising sensor data is extending the range of the sensor data to a central data concentrator or communication hub to optimize the data cost of the sensors. To realise this opportunity, the research investigated the low-level implementation of wireless technologies to adapt some of these technologies to achieve data optimisation and transfer.

The literature review was conducted to understand two available wireless technologies: LoRa, which is essentially Chirp Spread Spectrum(CSS) technology, and 2.4 GHz radio wireless link using the nrf24L01 module. These two technologies can provide the functional requirement of this research: to extend the range of these industrial sensors from the communication hub or data concentrator. It was necessary to understand the capability of these wireless technologies and their shortcomings to implement them appropriately.

2.5.1 Nrf24L01

The 2.4 GHz ISM radio wireless interface is a familiar radio interface used for transferring data of high data rates and bandwidth and relatively reasonable distances. There are many hardware options and footprints of this wireless technology. The nrf2401 wireless module is one of the cheapest on the market. If a designer is looking for a bit extra range, versions of this module come with an external antenna for a minimal additional cost. For a successful wireless link, this design will need a minimum of two modules and a microcontroller, each with an SPI interface.

The ideal module for this design would have been to use the nrf24E RF module as this module has a built-in 8-bit microcontroller. Still, unfortunately, during the building of the prototype, there weren't any of these devices available, the design implemented two nRF2401L modules with two microcontrollers instead.

Operating frequency of the nrf24L01 module.

The nRF2401l operates at the 2,4Ghz band. When looking at the transmission distance or range, these modules using the proper setup can transmit over several kilometres. While 2.4GHz is technically not a microwave, its wavelength is close. Frequencies over 3GHz are classified as a microwave. Being such a small wavelength allows very high gain antennas to be used, such as parabolic dish antennas. However, because the application was an IoT interface, it is limited to the type of antennae and physical size constraints of the device itself. So a basic lower power module was necessary for this design, but this impacted the range that this option can transmit.

This module aims to connect wireless data between two devices to collate, buffered, and transmitted between the two microcontrollers. To test the data rates, bandwidth, and range, it was necessary to interface these radio modules with a microcontroller using a lower power setup to interface it to an IoT device. To achieve this interface, the device will be using an nrf2401L module, and as mentioned early, for this implementation, two nRF2401l modules will be used, one stm32f4 board and one stm32 Nucleo board.

2.5.2 LoRa

LoRa is an LPWAN technology developed by *Semtech*, an American electronics manufacturer. It uses spread spectrum modulation on sub-GHz ISM bands, offering data rates from 0.3 kbps to 50 kbps. LoRa uses unlicensed wavebands, i.e., 868MHz and 433MHz. The LoRa Alliance, a non-profit association of more than 500 member companies, drives the LoRa protocol as an open standard.(Bäumker, Miguel Garcia and Woias, 2019)

Bidirectional communication is provided by “chirp spread spectrum” (CSS) modulation that spreads a narrow-band signal over a wider channel bandwidth. The resulting signal has low noise levels, enabling high interference resilience, and is difficult to detect or jam.

LoRa offers functionality that is very similar to *Sigfox*, making it ideal for sensor devices. LoRa has an open technological ecosystem, meaning there are several competing open software and vendors available. The LoRa standard (in as much as it is a standard driven by its vendors) lacks some key features; including roaming, packetization and retry, disconnected operations, quality of service, firmware upgrades over the air, and the use of repeaters. Only vendor-customised LoRa solutions address most of these shortcomings.

A network created by LoRa gateway hardware installed as base stations is known as a ‘LoRaWAN,’ LoRa capable devices can connect. LoRaWANs can be open, allowing any LoRa capable device to join through any gateway (though the data is visible only to the device's owner). Alternatively, private LoRaWANs can be established, which only carry traffic from devices authenticated by the gateway.

A large number of LoRa capable devices are available, produced by competing manufactures. Examples included water, gas, and electricity meters, sensors for temperature, CO_2 and other gases, smoke, humidity, visible light, and infrared light, barometric pressure, and soil moisture; motion sensors, altimeters, and accelerometers; door switches; distance measurement; tank level sensors; and detectors for water leaks, parcel trackers, vehicle parking bay available, manhole lid tampering, trash bin status (full/empty), and even rodent traps that send a signal when they are triggered.

LoRa capable actuators are available, including light switches, motor switches, relay switches, servomotors, and valve switches. Many come with a geolocation capability to easily be located; this also helps to identify/verify the source of the data that originates from each deployed device.

LoRa PHY and LoRa WAN

LoRa is a technology that falls within the LPWAN technologies. This technology is essential in realising IOT for two key factors, i.e., significantly lower power consumption with battery life extending to more than two years and relatively long-range data exchange at the expense of meager data rates in the case of LoRa 300bps to 5Kpbs(125Khz). LoRa and LoRaWAN have numerous critical use cases and implementations.

2.5.1 LoRa PHY

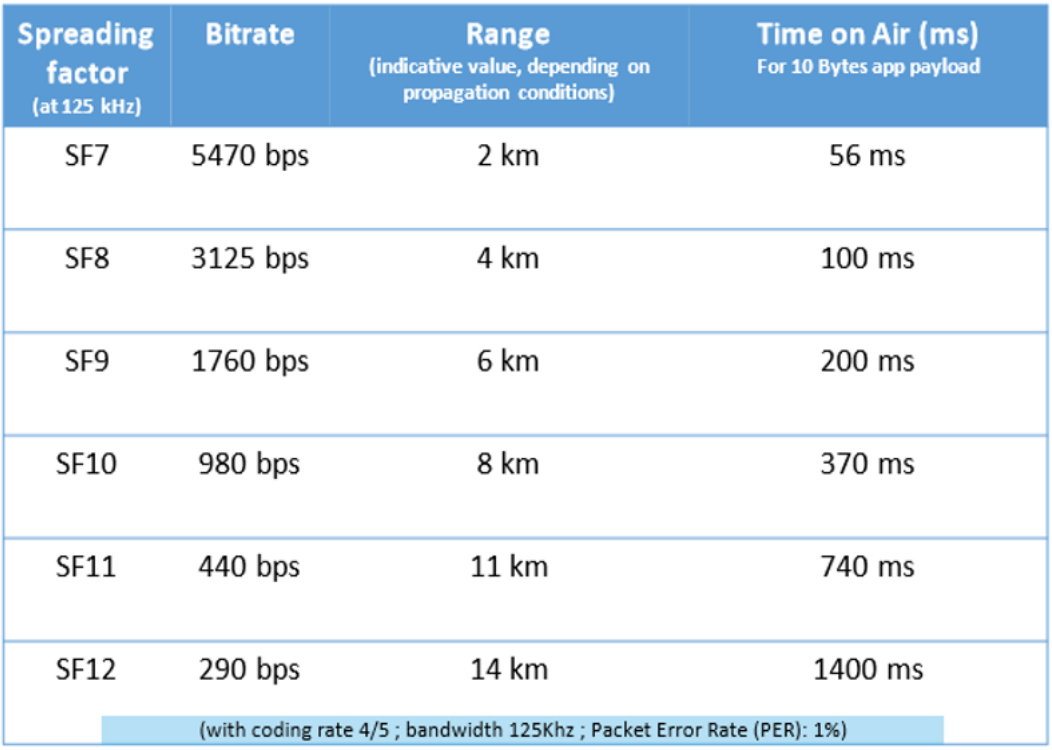
The LoRa®-protocol defines the link-layer, describing how the data is modulated into the transmitted signal. The protocol is proprietary; not all information is freely available. Nevertheless, it is enough to know that due to the used Chirp Spread Spectrum (CSS), the power consumption does not depend on the content of the data. Therefore, the energy required for transmission only depends on the so-called time on-air (ToA) and the output power. Time on air refers to the time it takes a signal to travel from the transmitter to the receiver.

LoRa uses orthogonal spreading factors. This allows the network to preserve the battery life of connected end nodes by making adaptive optimizations of an individual end node's power levels and data rates.

An end device located close to a gateway should transmit data at a low spreading factor since a minimal link budget is needed. However, an end device located several miles from a gateway will need to transmit with a much higher spreading factor.

LoRa is a long-range radio at relatively low bit rates.

Table 2.1 Typical LoRa Range in Urban Environment using ADR.



Spreading factor (at 125 kHz)	Bitrate	Range (indicative value, depending on propagation conditions)	Time on Air (ms) For 10 Bytes app payload
SF7	5470 bps	2 km	56 ms
SF8	3125 bps	4 km	100 ms
SF9	1760 bps	6 km	200 ms
SF10	980 bps	8 km	370 ms
SF11	440 bps	11 km	740 ms
SF12	290 bps	14 km	1400 ms

(with coding rate 4/5 ; bandwidth 125Khz ; Packet Error Rate (PER): 1%)

LoRa is very low bandwidth and consumes very little power, which makes it ideal for battery-powered operations.

LoRa operates in the license-free band. Either 433, 868, or 915 MHz. LoRa has a huge link budget.

LoRa PHY is the Best-effort transmission and, analogous to UDP, doesn't care if it gets there. LoRa Wan Is a layer on top of LORA PHY knows that data gets there like TCP. LoRa WAN, done by the LoRa alliance, makes sure data is arriving and if there are no duplicates. LoRa wan also adds encryption, whereas LoRa node/ LoRa Physical has no encryption.

2.5.2 LoRa Packet Format

The basic LoRa packet comprises three elements, namely the preamble, an optional header, and payload. There are essentially two types of LoRa packets transmission viz. an explicit header mode and implicit header mode.

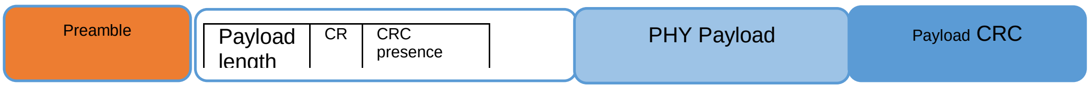


Figure 2.5: Frame format Implicit header mode

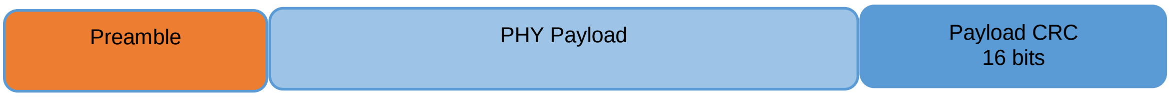


Figure 2.6: Frame format Explicit header Mode

The functional difference between these two modes is the presence or absence of the header, which contains information regarding the payload length, the coding rate, and the presence of 16 bits' cyclic redundancy check. In the implicit mode where the header is absent, the header information must be manually configured on both sides of the radio link.

The robustness of the physical layer can be attributed to one significant parameter of the LoRa Physical Layer, i.e., Forward error correction.

Forward error correction is where redundant bits are added to transmitted data. These redundant bits help restore the data when/if it gets corrupted by interference. The higher the number of error correction bits added, the easier it is to correct the data. However, the payload inherently increases, increasing power consumption in the LPWan transmission, adversely affecting battery life.

The coding rate (CR) parameter (ranges from 1-4) defines the additional bits added to each 4-bit package that together form a code-word.

The coding rate refers to the proportion of the transmitted bits that carries information. The Coding Rate affects the Time on Air (TOA) negatively but makes communication more robust.

These parameters define the equivalent data rate that can be achieved with LoRa®. It can vary from 300 bit/s up to 5.5 kbit/s. Like many other link-layer protocols, LoRa® adds additional constant factors to each transmitted packet to increase the ToA.

(i) The preamble, needed for the synchronization of receiver and transmitter. The minimum length is 6+4 symbols.

(ii) (ii) The optional header. It provides information about payload length, FEC-rate, and whether there is a field for a Cyclic Redundancy Check (CRC) included in the package.

(iii) The optional CRC at the end of each packet adds roughly five symbols to the transmission.

It is a low bandwidth with significantly lower battery or power consumption. The link budget is 14 dBm. Receiver sensitivity for LoRa is excellent.

2.5.3 LoRa Transmitter module

This research implemented both LoRa node to node and LoRa Wan, and therefore two variations of the LoRa Transmitter modules were implemented. The implemented two types will be referred to as LoRa Phy Transmitter and LoRa Wan end node transmitter modules, respectively, for ease of distinguishing between the two. The two implementations covered are implemented separately and covered in two different researchs in this thesis. In real terms of the physical hardware of the two variations, there are essentially very similar. In both instances, the microcontrollers are interfaced with the LoRa radio modules using SPI and the sensors. Both variations of the LoRa transmitter modules are essentially the same, depends on what sensor type is interfaced to the microcontroller. The firmware/software on the modules is different, calling different routines/functions to implement the different options of LoRa.

The design of the LoRa Transmitter module unit relies on the proper use of the microcontrollers peripheral interfaces to get data from different modules and store it in buffers for transmission using the LoRa link layer. To achieve an optimised IoT end device, it was necessary to optimise DMA and Interrupts via the nested Vector interrupt controller of the STM32 microcontroller. This thesis also implemented an in-house buffer as this was key to seamlessly parsing data to the LoRa radio module.

The LoRa Phy transmitter module was relatively straightforward and quick to implement as the was no encryption/decryption of any sort with no device id etc. It is purely one-to-one transmission to the LoRa Phy receiver module. The purpose of this option was just to get data from a data source from a range of about 400 -800m into a SCADA system in an IP format.

Often the literature review was the foundation on which some of the implementations started to progress to a different platform or technology like nrf24l01, LoRa PHY, and LoRa wan. Initially, the research began researching and implementing the nfr24L01 option of getting data to a high-level system only to use similar architecture to accomplish the same objective as with LoRa PHY and subsequently advance to a LoRa WAN. Each of these options has a place in the data value chain and application. For this research, the LoRa Phy and LoRa WAN were implemented but for totally different applications. Even though the Nrf24l01 radio module is not suitable for ranges longer than 150m for data transfer, it certainly outperforms LoRa in terms of bandwidth.

2.6 Other Wireless technologies

In the IoT space, numerous other competing technologies offer similar traits/characteristics to LoRaWan and have been looked at briefly to understand LoRawan fully. The other candidate technologies that have been identified are:

i. Random Phase Multiple Access

ii. Sigfox.

iii. NB-IoT.

2.6.1 RPMA – Random Phase Multiple Access (also known as Ingenu)

RPMA is a patented technology developed by *Ingenu*, the name by which the RPMA technology is popularly known. RPMA operates in the 2.4GHz (ISM) band, resulting in relatively high bandwidth capacity. In dense urban areas, the technology suffers from the possibility of high interference from competing Wi-Fi deployments using the same frequency band. RPMA may also require the edge devices to have the increased processing power, causing them to be poorly suited to battery-based applications. RPMA offers mutual authentication, message integrity, and replay protection, message confidentiality, device anonymity, authentic firmware upgrades, and secure multicasts from a security perspective. The *Ingenu* business model is that of a service provider of OT networks.

2.6.2 Sigfox

Sigfox is named after the French company* that developed it. It was the first LPWAN technology proposed in the IoT market in 2009. *Sigfox* uses ultra narrow-band modulation that supports data rates of 100 bps in the unlicensed frequencies; In South Africa, *Sigfox* uses the 433 MHz and the 868 MHz unlicensed wavebands. *Sigfox* transceivers send minimal amounts of data (12 bytes) very slowly (300 baud), which can usefully connect low-power objects such as electricity meters and smartwatches, which may need to be continuously on and emitting small amounts of data. These very long and very slow messages give *Sigfox* networks a very long range. *Sigfox* claims that each gateway can handle up to a million connected objects, with 30-50 km coverage in rural areas and 3-10 km in urban areas. The small data packet size and baud rate have downsides for download traffic and limit its use as a control technology. Consequently, it is best suited to intermittent monitoring.

Therefore, this technology is a good fit for any application that needs to send small, infrequent bursts of data, such as basic alarm systems, location monitoring, and simple metering. The signal is typically sent several times to “ensure” the message gets through. While this improves reliability, it also introduces limitations, such as shorter battery life for battery-powered applications.

The *Sigfox* business model is an operator for IoT services; its technology and security protocols are proprietary and closed. Sigfox focuses on the network security itself as a general approach, leaving the payload security mechanisms to the end-users at both the transmitting (node) and receiving side (application).

A *Sigfox* network is operated in South Africa, under license, by Dark Fibre Africa. This network is called “SqwidNet”; it provides connectivity only, plus interconnects with other networks. Node chipsets installed with each device must conform to the *Sigfox* specifications.

2.6.3 NB-IoT

Mobile Broadband (LTE, HSPA, 3G) is trending towards higher carrying capacity (10-40mbps) across radio access links. Earlier versions of GSM networks (2G and GPRS) did carry narrowband traffic, but these are being phased out as demand for high bandwidth by end-users increases and the mobile operators move towards LTE. High bandwidth networks are overkill when dealing with devices that intermittently transmit small amounts of data (10-100kbps). Mobile operators recognize this differential and so are now offering LTE-type narrowband services.

Narrowband IoT (NB-IoT or NB-LTE) is an LPWAN radio technology standard using a subset of the LTE standard. It enables a wide range of devices and services to be connected using International Mobile Telecommunications (IMT) bands. It is one of a range of mobile IoT technologies championed by the 3rd Generation Partnership Project (3GPP), a collaboration between groups of telecommunications standards associations.

NB-IoT has a specific focus on enabling indoor coverage (which other LPWAN technologies may struggle with) and enabling a large number of connected devices. The NB-IoT technology is generally deployed "in-band" in the spectrum allocated for Long Term Evolution (LTE).

Several South African mobile network operators offer NB-IoT but practically is not yet available, but this is fast changing.

Table 2.2: Comparison of LPWAN Technologies.

Technology Characteristic	**RPMA (*Ingenu*)**	***Sigfox***	**LoRa**	**NB-IOT**
Coverage range (line of sight)	15 km (Urban)	10 km (Urban), 50 km (Rural)	5 km (Urban), 15 km (Rural)	1 km (Urban), 10 km (Rural)
Radio Frequency Band	ISM 2.4 GHz	ISM 868 MHz / 902 MHz	ISM 868 MHz / 902 MHz / 433 MHz	Licensed LTE
Bidirectional Link	Yes (half duplex)	Yes (half duplex)	Yes (half duplex)	Yes (half duplex)
Over-The-Air Firmware Upgrade	Yes	No	No	Yes
Topology	Star, Tree	Star	Star of stars	Star
Maximum Data rate	8 kb/s	0.1 kb/s	50 kb/s	200 kb/s
Payload length	10 kB	12 B (up-link), 8 B (down link)	Up to 250 B	1600 B
Business model	Only available as a service	Only available as a service	Components available to build-own-operate, and also available as a service	Only available as a service

CHAPTER 3
HARDWARE DESIGN AND IMPLEMENTATION

3.1 Introduction

This chapter implements a lot of hardware interfaces to accomplish the objectives of the research. To give context to some of the modules implemented in Chapters 4 and 5, it was necessary to provide a more detailed technical explanation of the various modules' hardware/software design and implementation.

The hardware design can be broken down into different segments. This thesis focuses on the total data value in acquiring data and visualising field data from existing process control systems and IoT systems. One of the core objectives is to augment the data value chain by adding sensor data values using existing industrial sensors, which means the deployed IoT devices would need some sort of interfacing into the types of the different inputs to the devices.

Development boards were used for all the hardware implemented in this design, which gave this research pace to implement the proof of concept quickly. Many rapid development platforms make development and proof of concept relatively seamless but at the cost of not fully understanding the technologies. It is for this reason that some of the development was also done on a barebones approach. The microcontroller chosen to implement most hardware proof of concept was the STM32 series and the ATMEGA128. So for working with these development boards, STMcubemax, Arduino, and Mbed IDE’s were used to implement some of the hardware devices. Most of the interfaces used in the design were also development-type boards with pre-written libraries available to use with these boards.

To achieve the desired outcomes in this design, it was essential to research the different hardware platforms; Chirp spread spectrum LoRa radio modules, 2.4ghz radio modules, and other microcontrollers ranging from stm32 Nucleo boards, stm32f4 discovery boards, and Arduino.

There was also an immediate requirement to read and understand the datasheets for all the implemented hardware. The stm32f4 data sheet is on its own has more than 1800 pages as well as numerous application notes that complement the datasheet. With some research and forums, this datasheet puts into context the capabilities and issues other developers and designers have had with the stm32f4 development board. Interfacing the microcontrollers into the different hardware interfaces to realise the various outcomes of this design required researching these interfaces and understanding the technology and the libraries associated with the multiple interfaces. Most of the interfaces in this research are either UART, I2C, or SPI.

All the other modules implemented in this design required a detailed familiarisation with their datasheets to understand the interfacing of these modules to the mother module, especially concerning the voltages and electromagnetic interference. Electromagnetic Interference(EMI) is significant because some modules implemented have radio frequency implications.

Electromagnetic Interference plays a vital role in the boards' hardware layout, primarily because of the proximity of these modules to each other. This thesis has not covered any documentation in terms of electromagnetic interference. Still, it expects that cognisance should be taken of this influential factor concerning understanding the design and layout of components and modules, explicitly knowing that many modules of this design operate in the radio frequency domain.

Multiple level shifting circuits and interfaces are employed when interfacing the various modules to the modern microcontroller as it operates mainly with 0-5V and, more commonly, 0-3.3V. Most of the sensors and actuators in the process control environment operate between 12-24V and introduce many transient currents. These level shifters play an essential role in shifting the voltages to the required level/voltage of the microcontroller. The level shifters also provide the microcontroller protection and isolation from these transients. From an output perspective, the level shifters also provide the switching power that the microcontroller on its own does not have.

In this research, various technologies were implemented, all of which required some form of investigation. All of these have been documented in depth in some cases and brief in others.

3.2 Implementing the Nrf24l01 Radio modules

To give some context regarding the processes involved in programming and interfacing these Radio modules to the microcontroller, a more detailed overview will be covered for this particular module where the principle and methods implemented here are consistent in implementing the other hardware modules in this research. It is particularly constant with the interfaces using SPI or similar microcontroller interfaces like I2C; what is essential to understand and is that each module has its instruction codes that need to be parsed to the modules to access the functions of that specific module. These instruction codes are parsed to initialise the modules and use the functions() associated with that particular interface, as seen in the explanation below.

Not forgetting the primary objective of these radio modules is to extend the sensor range of a telemetry device connected to a local process control system or SCADA system.

The concept design of the interface is to translate a signal, specifically an analogue signal of a measuring device that is between 60m and 1000m meters from the process control system or SCADA. This essentially translated two microcontrollers reading an analogue sensor value interfaced with these radio modules, as seen in figure 3.6 below.

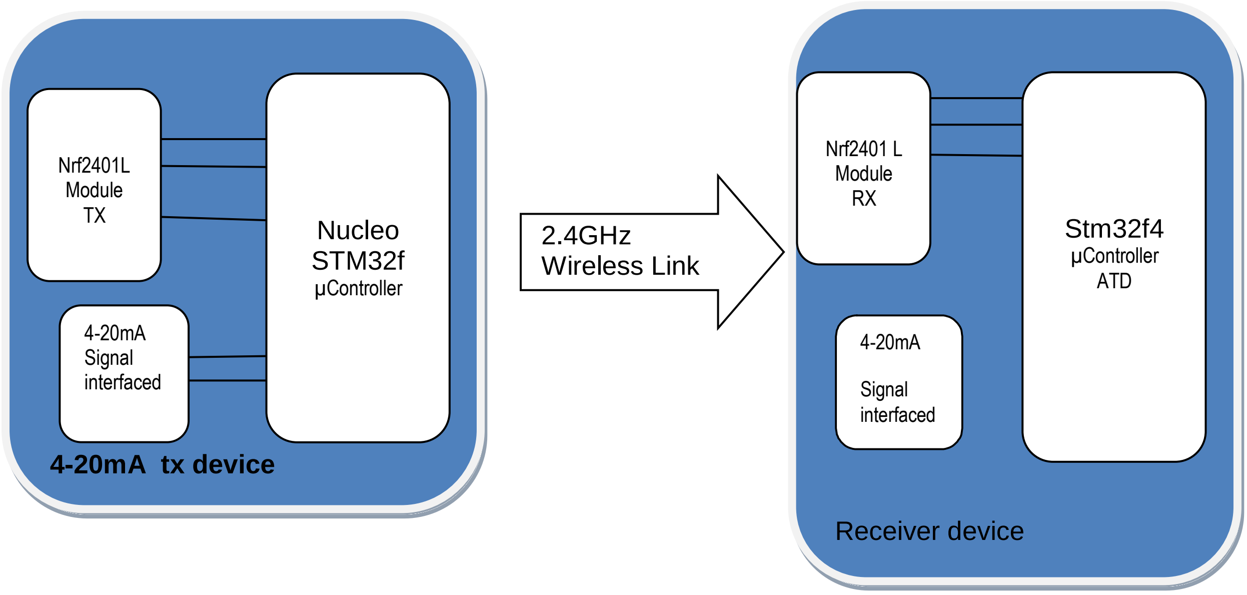


Figure 3.1 **Diagram illustrating rf24l01 wireless technology**

3.2.1 Nrf24l01 Receiver (Rx) module

This module is comprised of an Stm32 Nucleo board with an Nrf24l01 radio module. This board receives the data and sends an acknowledgement of the received data. These modules were implemented to test the data capability of this wireless interface against the LoRa Phy modules.

3.2.2 Nrf24l01 Transmitter (TX Module

For proof of concept, dummy data was transmitted via this device. This hardware implementation was nrf24l01 module interfaced to a Nuceo board via SPI. This board would transmit this Dummy data.

Nucleo nrf2401l Transmitter Module

Setting up this module is relatively straightforward as numerous libraries are written for the nRf2401l module using the Nucleo Board. So all that is required is to import these libraries associated with this board and start using the functions of this board to achieve the needed functionality.

Nucleo Transmitter Software:

The software module in **Appendix C** simulates the dummy data transmitted via the nrf2401l wireless module. (Nordic, 2008)The device simulates the dummy data represented by reading an analogue pot value. The Nucleo board, together with the nRf2401L module, will be set up as a transmitter. Before any coding of the Nucleo, it is necessary to import the Nrf2401l Library to the IDE so that it can be used in the code:

Operating modes

If the device is in receiving mode:

- If CE is high.Allows you to monitor the airwaves to receive packets
- If CE goes low, it puts the device in Standby mode and no longer monitors the airwaves.

If Device in Transmitting mode:

- CE is held low except when we want to transmit a packet
- CSN is the enable pin for the SPI bus. This pin is active low.
- SCK is the serial clock for the SPI bus
- On the microcontroller, SPI configuration CPOL and CPHA must be configured.
- SPI must be set up for 8bits
- SPI of the microcontroller to be set as master and set up as a slave on nRf2401l

To receive data on the nRF2401l, the following must happen:

- CSN must be high, to begin with
- The assert CSN low to alert nrf2401l that it will start receiving SPI data.
- This pin will stay low throughout the transaction.

The hardware is straightforward to set up, and the pins are self-explanatory. We will, however, mention and explain that this module is interfaced with the stm32f mother module via the SPI interface.

The SPI interface allows the device to read/write registers, transmit data and receive data

To send data to or receive data from the SPI port on the 24L01, the device had to be configured to do a few things.

i. SPI Instruction set for the nRF2401

- The CSN pin on the 24L01 must be high to start with,
- Assert the CSN pin low to alert the 24L01 that it is about to receive SPI data. Every time the device wants to send SPI commands, CSN Pin needs to go low.
- Then transmit the command byte of the instruction you wish to send.
- When receiving data bytes for this instruction, send one byte to the nRF24L01 for every byte that the device wishes to get out of the nRF24L01.
- If sending the nRF24L01 data, simply send data bytes and generally don't worry about what gets sent back to you.
- (Nordic 2016)When receiving data from the nRF24L01, it makes absolutely no difference what is contained in the data bytes that have been sent after the command byte, just so long as the correct number of them has been sent.

ii. nRF2401l Data Pipes

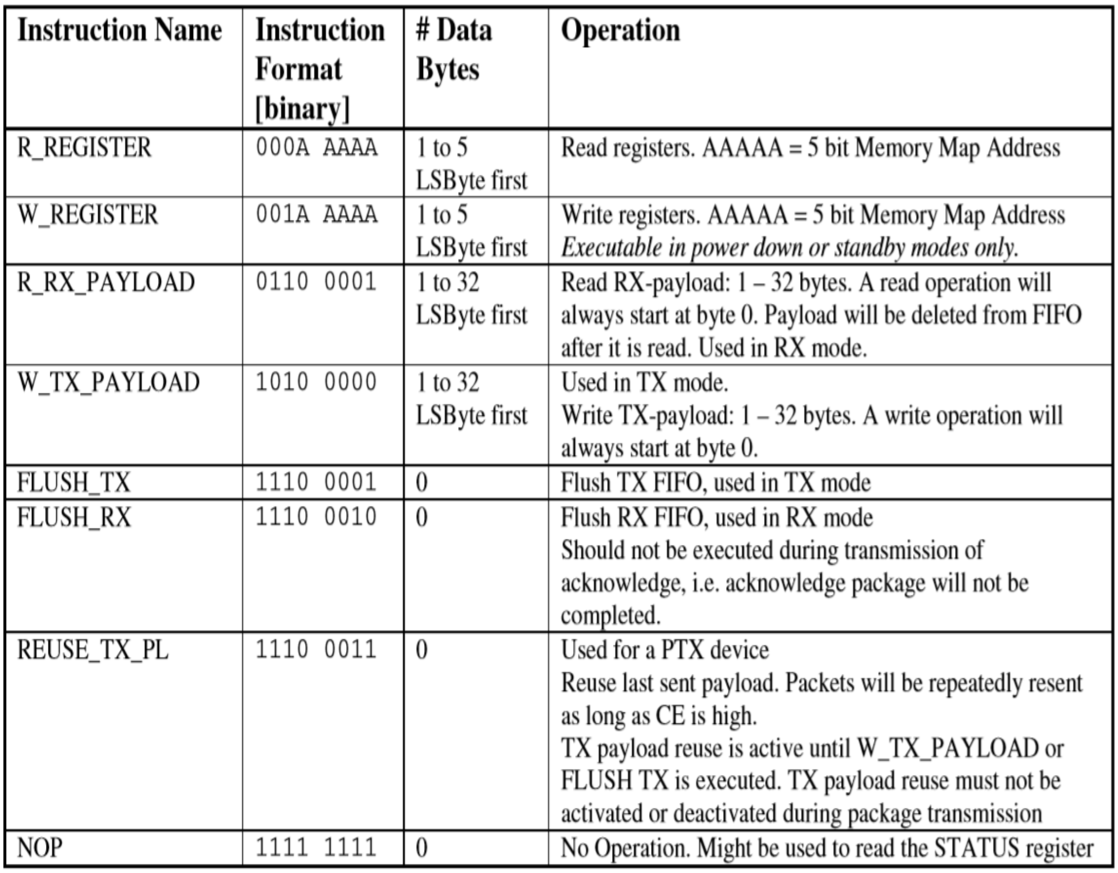
When setting up the wireless module for communication, a few parameters have to be configured to get that from one device to the other. The key parameters:

- Channel: which is the specific frequency channel that communication will take place on. Integers from 0 to 125(7 bits of the particular register) represent frequencies that are mapped
- Reading pipe: the reading pipe is a unique 24, 32, or 40-bit address from which the module reads data(reading and writing pipes addresses are swapped between master and slaves when setting up a single channel wireless link)
- Writing pipe: the writing pipe is a unique address to which the module writes data

- Power Amplifier (PA) level: the PA level sets the chip's power draw and transmission power. For this research (use with the Arduino), we will use the minimum power setting.

The writing and reading pipe addresses are swapped between the two radios communicating with each other, as the writing pipe for each radio is the reading pipe for the other. If this is not done, we will not have a successful connection.

Table 3.1 Instruction Set for nRf2401L



Instruction Name	Instruction Format [binary]	# Data Bytes	Operation
R_REGISTER	000A AAAA	1 to 5 LSByte first	Read registers. AAAAA = 5 bit Memory Map Address
W_REGISTER	001A AAAA	1 to 5 LSByte first	Write registers. AAAAA = 5 bit Memory Map Address *Executable in power down or standby modes only.*
R_RX_PAYLOAD	0110 0001	1 to 32 LSByte first	Read RX-payload: 1 – 32 bytes. A read operation will always start at byte 0. Payload will be deleted from FIFO after it is read. Used in RX mode.
W_TX_PAYLOAD	1010 0000	1 to 32 LSByte first	Used in TX mode. Write TX-payload: 1 – 32 bytes. A write operation will always start at byte 0.
FLUSH_TX	1110 0001	0	Flush TX FIFO, used in TX mode
FLUSH_RX	1110 0010	0	Flush RX FIFO, used in RX mode Should not be executed during transmission of acknowledge, i.e. acknowledge package will not be completed.
REUSE_TX_PL	1110 0011	0	Used for a PTX device Reuse last sent payload. Packets will be repeatedly resent as long as CE is high. TX payload reuse is active until W_TX_PAYLOAD or FLUSH TX is executed. TX payload reuse must not be activated or deactivated during package transmission
NOP	1111 1111	0	No Operation. Might be used to read the STATUS register

Scenario:

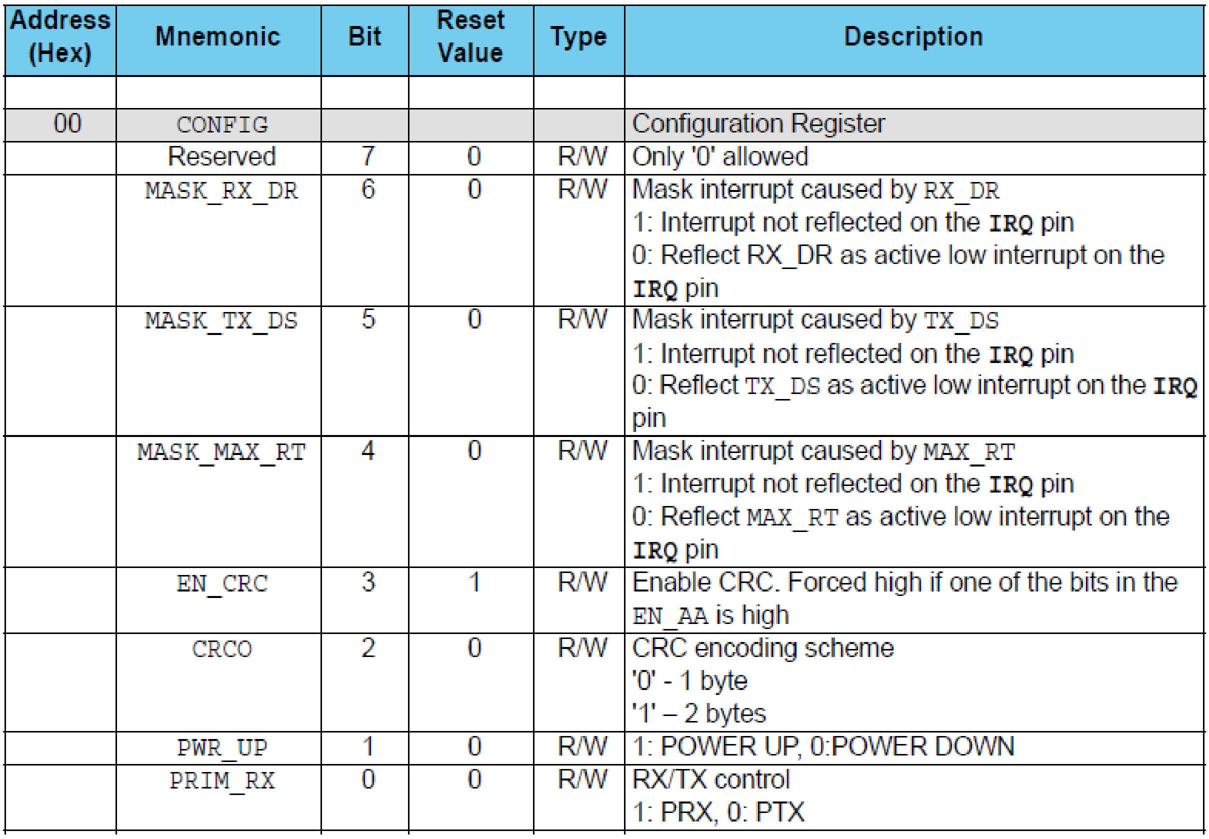
- The device wants to read the contents at a specific register at a known memory address 0x10.
- First, bring CSN low and then send the command byte ‘**00010000**’ to the 24L01. This instructs the 24L01 that the device wants to read register 0x10, which is the TX_ADDR register.
- From the table above from command “00010000.”

- The first three bits **000** translates to read
- The last five bits in the **10000** in binary **is 0x10** in hex.
- To sum up the instruction set bluntly, the first 3 bits are “**what to do,”** and the next 5 bits are “**where to do.”**

Then send five dummy data bytes, and the 24L01 will send back the contents of the TX_ADDR register.

Finally, bring the CSN pin back high. All totalled, the device will receive six bytes. When any command byte is sent, the nRF24L01 always returns the STATUS register. After that, we will have received the five bytes that are contained in the TX_ADDR register (Nordic Semiconductor Inc. 2009).

Table 3.2 Configuration Register of nRF2401L (Nordic, 2008)



Address (Hex)	Mnemonic	Bit	Reset Value	Type	Description
00	CONFIG				Configuration Register
	Reserved	7	0	R/W	Only '0' allowed
	MASK_RX_DR	6	0	R/W	Mask interrupt caused by RX_DR 1: Interrupt not reflected on the **IRQ** pin 0: Reflect RX_DR as active low interrupt on the **IRQ** pin
	MASK_TX_DS	5	0	R/W	Mask interrupt caused by TX_DS 1: Interrupt not reflected on the **IRQ** pin 0: Reflect TX_DS as active low interrupt on the **IRQ** pin
	MASK_MAX_RT	4	0	R/W	Mask interrupt caused by MAX_RT 1: Interrupt not reflected on the **IRQ** pin 0: Reflect MAX_RT as active low interrupt on the **IRQ** pin
	EN_CRC	3	1	R/W	Enable CRC. Forced high if one of the bits in the EN_AA is high
	CRCO	2	0	R/W	CRC encoding scheme '0' - 1 byte '1' – 2 bytes
	PWR_UP	1	0	R/W	1: POWER UP, 0:POWER DOWN
	PRIM_RX	0	0	R/W	RX/TX control 1: PRX, 0: PTX

Understanding the datasheets and the technology itself is essential to writing the code/firmware to get these modules to operate. It also implements some form of debugging to visualize the successful setting of the radio interface before the modules can start sending and receiving data. See appendix for the code to set up the transmitter module.

iii. STM32f4 Receiver Software:

The libraries used to initialise and set up the Nucleo board are the same as those used to set up the STM32f4 Discovery boards because they have the same core processors. Similarly, for the stm32f4, there are also libraries already written for the nRf2401lL device. All that has to be done is download the library, include it in the project, and use the functions. The apparent difference between the Nucleo board and the STM32f4 discovery board is that the Nucleo board is Mbed enabled and the Discovery board is not, but fortunately, both boards are linked to STMCubemax so that the peripherals are accessible via Hal libraries associated with the boards. In other words, the Hardware abstraction layer libraries are available to implement code for initialising the peripherals of the devices.

Not forgetting to interchange the address pipes between the receive and transmit modules. The libraries and functions associated with the NRf24l01 radio module for the Nucleo board and the STM 32f4 discovery board are interchangeable.
Rf Module is set data rate of 2Mbps and output Power of 0dbm.

iv. Setting up the Stmf4 discovery board as Receiver module.

As mentioned above, there are libraries written and are freely available for use with the STM32f4 microcontroller. The following pseudo-code is necessary to get the asset tracking device to operate as the receiver.

- Import the necessary libraries into the project folder.
- Include the necessary header files to be used with the radio module.
- Set the respective Tx address and Rx Address in the receiver module.
- Initialise the buffer in the code i.e. char Bio_ dataOut[32] and char Bio_dataIn[32]

- The datasheet uses an appropriate pin for SPI transfer and DMA mapping to choose available pins for the Nrf24l01 module.
- Some radio modules have a pins pack function; this function can be “called” to set up the pins for the w module.
- Set data rate and power level of the receiver in the module; it is necessary to set data rate to 2mbps and power level to -18dBm.
- The while loop for the data module will look something like the code **Appendix D.**

3.3. Implementing the LoRa Physical Layer devices

Because of the low-level implementation of the LoRa physical layer, it was decided that the proof of concept get carried out on boards already familiar with and the fact we already had these boards from previous projects. The two boards that were used were the Nucleo F401E development board and the StM32f4 discovery board. The Nucleo board was used as the Transmitter module, and the STM32f4 Discovery board was set up as a receiver module. The two IDE used to download the code was Keil Microvision for the STM32f4 and Mbed online compiler for the Nucleo board. Dummy data was placed in the transmit buffer buffer_tx[], and the Nucleo was interfaced via SPI to the LoRa radio module.

3.2.1 LoRa Phy Tx Module

This module was implemented on the sfm32f4 discovery board with two interfaces, namely the 4-20mAinterface and the sx1278 LoRa radio module.

This interface translated to interfacing the 4-20mAboard using the SPI channel of the microcontroller. Analogue signal acquired via this interface was stored in the buffer of a microcontroller(txBufferLoRa); this buffer was the buffer that was accessed via the second SPI of the microcontroller, which was interfaced to the sx1278 LoRa radio module. This buffer would make up the payload of the LoRa packet transmitted via LoRa Phy.

This module used SPI to interface both the modules to the STM32 Nucleo microcontroller. The Nrf24l01 radio modules, specific instruction codes, and registers associated data parsed with the per SPI command. These instruction codes are in the application notes documented with each of the interfaces used in this design. The routines are part of the sourced code of each of the modules implemented.

3.3.2 LoRa Rx end Node for LoRa Phy(LoRa Phy Rx Module)

The LoRa PHY receiver module consists of a LoRa radio module(sx178), a microcontroller, and an Ethernet controller.

The Ethernet controller component of the research was essential in parsing the data received from the LoRa radio module using the LoRa physical layer to some application to be used or visualised. There are many chipsets available to do this wrapping, but the most popular in the embedded world are the enc28j60 and the ws5500. These low-level Ethernet controllers are interfaced to a microcontroller with sufficient RAM using an SPI interface.

This Ethernet controller has a TCP/IP stack built into its SOC. The preferred format of wrapping data is either TCP or UDP; this is true because there are many device drivers for these two formats. Most modern-day SCADA systems and MES systems have device drivers that directly translate TCP/IP data format, there might be a need to write a script to de-serialise the payload, but there is a seamless door to the application. With protocols like MQTT, COAP, and OPC UA, this data exchange is becoming less and less complicated. Also, with the arrival of JSON, even the de-serialising of data from different data sources has become unidimensional.

In essence, with the start of this research, it was assumed that parsing of the data from LoRa physical to a UDP sentence would be cumbersome but researching the different options showed that not only are there different types of Ethernet controllers capable of this translation, but this translation can be done on separate layers. The design implementation of this research was implemented at an embedded level. The chip chosen for the LoRa to Ethernet wrapper was enc28j60 by Microchip.

These chips are powered by 3.3V powered by SPI but are also 5v tolerant

Understanding a bit of the theory of LORA radio transmission to implement the coding of the end devices to achieve the successful LoRa implementation. On the LoRa receiver side, it was also necessary to research a LoRa to IP bridge in an enc28j60 Ethernet controller to accomplish this bridge for proof of concept. Once the LoRa data was translated into IP format, the process was relatively seamless.

This module was implemented on the sfm32f4 discovery board with the enc28j60 Ethernet controller and the sx1278 LoRa radio module.(Semtech Corporation, 2013) This interface entails interfacing the enc28j60 Ethernet controller board using the SPI channel of the microcontroller. The second SPI of the microcontroller, which was interfaced to the sx1278 LoRa radio module, serves as the receiver module for the LoRa message transmitted by the LoRa Tx module.

These LoRa modules interface to the stm32 boards are capable of both receiving and transmitting LoRa. The payload LoRa packet received by the radio module after being demodulated is then written into a buffer of the microcontroller to be accessed by the enc28j60 Ethernet controller. The necessary header and payload are then added in the required format for transmission to the SCADA system, as covered in Chapter 4 of this research.

The STM microcontroller has a power Nested vector interrupt controller that makes transferring data from the respective buffers relatively seamless; another feature or resource of the stm32f4 microcontroller that was also used for data transfer with the microcontroller was the internal Direct memory Access controller of the Microcontroller. This DMA controller was handy as it minimised the engagement of the processor for transferring this data between the different buffers.

Regarding the LoRa physical layer implementation, the design uses two different STM boards: the Nucleo f401e for the tx module and the stm32f4 for the receiver module once proof of concept was achieved on these boards successfully implemented on these boards.

3.4 Visualising the LoRa Physical Layer

This implementation has few challenges concerning visualising the data received from the sensor using the LoRa physical layer. The reason is that the implementation is very low-level, and there are no frameworks that visualise this data directly. This research investigated the capability of this wireless technology without the LORA gateway component, thus eliminating the high-level protocols and frameworks to do this visualisation. (Net and Ua, 2014)

There is, however, the need to process or visualise the data received from this implementation to get benefit from this proof of concept, without which this implementation is pointless. Once the data is transmitted by the LoRa PHY transmitter and received and decoded by the LoRa PHY receiver, the data is useless to the microcontroller unless there is an action event associated with this process value/variable. But if this data could be wrapped into a UDP/TCP packet, it can be parsed using an IP protocol to be parsed to an MES/API.

Once the data is received by the LoRa receiver module and translated to IP format by the LoRa to IP wrapper, the next progression is to visualize this data by the local network. Each of the LoRa receiver modules has its hard-coded Mac address, which means these modules can parse data to the local network using their unique address, which also means data from the different sources and locations can also be differentiated.

Each of the end nodes has its own static IP Address, and port number associated and uses a UDP connection to parse data from the LoRa Phy end node to the c# application for visualisation of the sensor data acquired, which in this case was the flow rates of the treat affluent of the waste-water treatment plant.

To visualise the data from these modules, we implemented a C# application to de-serialise the data received from these LoRa end nodes and visualise it by the local machine.
It is essential to mention that initially, there was one LoRa Phy receiver for each LoRa transmitter i.e. but eventually progressed to having only one LoRa Phy Receiver for all of the four LoRa Phy transmitters.

Appendix B demonstrates the visualising of this data using a C# Application running on the local machine using Devexpress to represent flow rates of a wastewater treatment plant. The flow rates are represented/visualised using Devexpress gauge objects. Visualisation, as implemented in Appendix B, can be summarised below.

- Importing the relevant namespaces for the application
- Enumerate the respective flowmeters(4-20mAanalogue signals) so that they can be selected individually

- Set a template for the gauge object (Devexpress) that can be used to visualise the flow meter.
- Use a state machine (switch statement) to enable the user to select the specific flow reading.
- Read serial data
- Verify serial data.
- De-serialise data using standard c# routine using defined delimiters.
- Handle exceptions and errors.
- Map data to objects for visualisation.

3.5 GSM Logger

This research is centred on moving data from a local point to an MES or web application. Some of these data sources are in a remote location far away from plant infrastructure. These are data points of interest like water pressures in the pipeline, reservoir levels, etc., required by the MES or WEB service. With data cost decreasing, GSM loggers have become viable for conveying intrinsic data from these remote data sources. GSM loggers are not new to the data logging environment. In this research, we employ two variations of GSM Loggers i.e.

i. Type 1- Existing legacy 2g/GSM loggers that was reverse engineered and upgraded with new data format and transmission
ii. A new type of data logger developed to parse data in a new format.

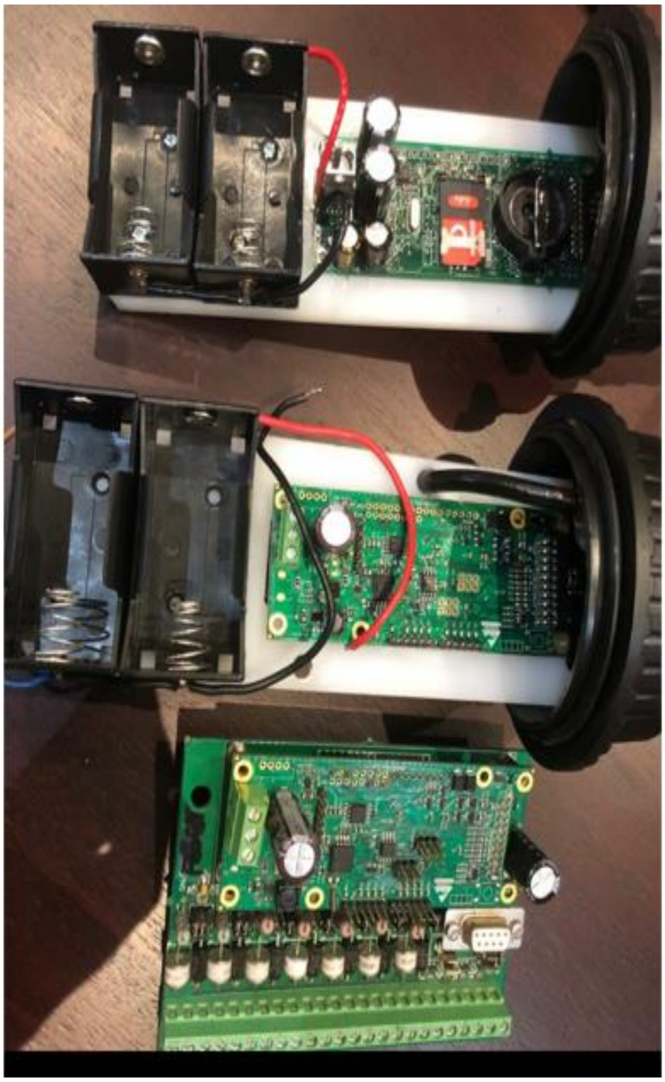


Figure 3.2 Picture of legacy type 2G logger that was reversed engineered.

Presently many legacy data loggers parse data in an outdated format where the overhead of the packet is heavy using legacy frameworks. These legacy loggers have very high latency in terms of data transfer. After researching the datasheet and Application notes of the Microcontroller(ATmega 128)used in the design of the legacy logger, it realised that writing new firmware was very plausible. (Atmel, 2011) These legacy loggers were upgraded to parse data in a modern format. There are existing data loggers present in the water and sanitation environment that can transmit data to a new device gateway for visualising.

Because more than a thousand of these devices are still in commission, discarding these loggers would be fruitless, so there was a business case to re-use the existing loggers. The legacy loggers have a very robust design and a very resilient 4-20mAinterface. They are already equipped with circuitry to power the 4-20mAcurrent loop from the battery source, which only adds to re-engineering these loggers.

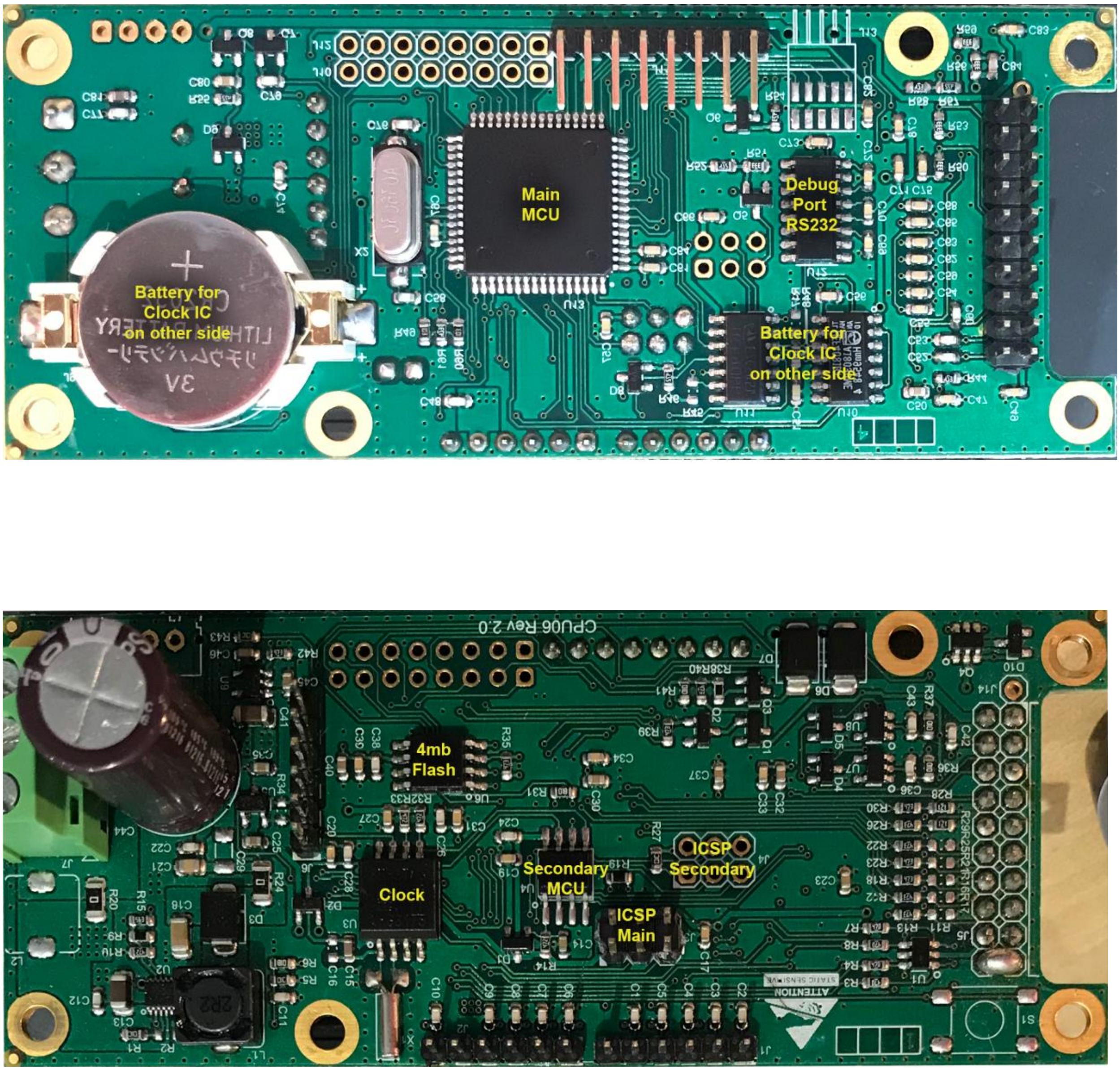


Figure 3.3 Top and Bottom view of Mother Unit of Legacy GSM Logger

The above Figure3.3 shows a close-up look of the middle board, which is the motherboard of the GSM Logger. The board was made with ease reprogramming in mind; as one can see, the ICSP pins on the boards are easily accessible. ICSP is an AVRtiny programming header for the Arduino consisting of MOSI, MISO, SCK, RESET, VCC, GND. It is often referred to as an SPI (Serial Peripheral Interface), which could be considered an "expansion" of the output, but really, you are slaving the output device to the master of the SPI bus.

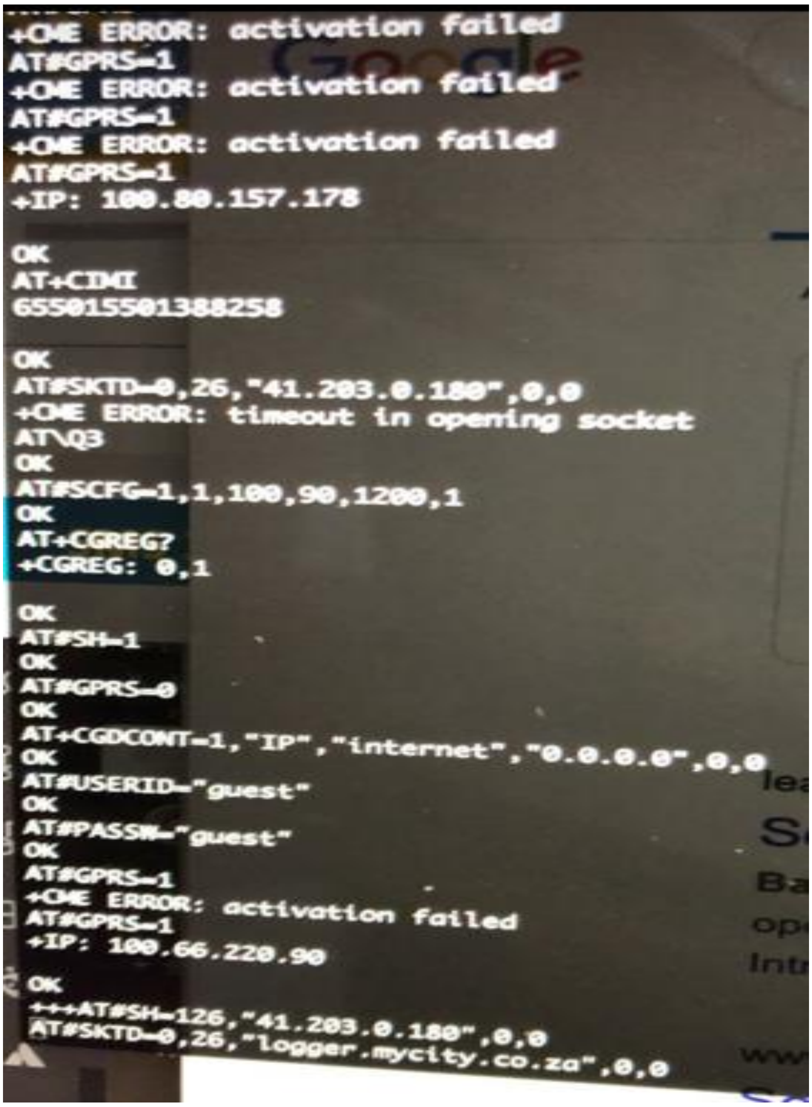
Concerning the AVR chips themselves, they are the way you program the chips in-system. You connect the programmer to these six pins somehow - usually over a ribbon cable with two six-pin IDC headers, but you can just use some jumpers too. The programmer can then

send the production file (elf file) to the chip. You can get the production file from the Arduino software or Atmel Studio.
The secondary MCU see in the figure above is ATtiny13A which is an 8-bit Atmel CMOS microcontroller.('tiny13A - secondary mcu.pdf', no date)

It is important to note that much of the modern architecture is designed to be firmware upgradeable by remote means, making it possible to re-use the mother module if needed. Another incredible feature that this board has, which was only discovered in our fourth software version, was the 4mB onboard flash. Part of the reverse engineering was scraping some hardened epoxy, which is typical for vendors if they do not want third parties to figure out what IC's/chips they are using.

Some of the earlier versions of the new software did not include this chip, which meant trying to optimise caching, which was unnecessary only to discover later that this 4mB chip was included in the design. This enabled about 2048 pages(the standard page size in this chip is 256Bytes) of data to be buffered. See appendix C for some level code for this process.

The old way of passing data can be seen in Figure 3.3 below. It used SNTP protocol to send messages to a host site, de-serialising the messages to extract data.



```
+CME ERROR: activation failed
AT#GPRS=1
+CME ERROR: activation failed
AT#GPRS=1
+CME ERROR: activation failed
AT#GPRS=1
+IP: 100.80.157.178

OK
AT+CIMI
655015501388258

OK
AT#SKTD=0,26,"41.203.0.180",0,0
+CME ERROR: timeout in opening socket
AT\Q3
OK
AT#SCFG=1,1,100,90,1200,1
OK
AT+CGREG?
+CGREG: 0,1

OK
AT#SH=1
OK
AT#GPRS=0
OK
AT+CGDCONT=1,"IP","internet","0.0.0.0",0,0
OK
AT#USERID="guest"
OK
AT#PASSW="guest"
OK
AT#GPRS=1
+CME ERROR: activation failed
AT#GPRS=1
+IP: 100.66.220.90

OK
+++AT#SH=126,"41.203.0.180",0,0
AT#SKTD=0,26,"logger.mycity.co.za",0,0
```

Figure 3.4: Screen capture of the serial port of legacy datalogger.

These legacy devices operate with a 2G modem interface and have costly robust sensor interfaces, but the firmware runs on a legacy framework. It was possible to use the existing loggers to function with new software in a new framework with reverse engineering. This reverse engineering was considered in the context where these devices are still very relevant as a data source and that there would be no need for new hardware if some code could be written to parse the data in a new data format. This essentially would translate to serious cost savings if this legacy hardware were re-used.

The cost of these data loggers, when commissioned more than ten years ago, was approximately R10 000 per logger. Having more than 1200 devices would realize a considerable saving if the same hardware was used. Successful re-engineering would also demonstrate how electronic engineering has progressed. Electronic engineering leans and depends more on the software than previous generation hardware where logic functionality was hardcoded in the devices' hardware and were essentially one-time programmable(OTP) type architectures. Table 1 represents the key differences and benefits between the two different frameworks while using the existing hardware.

Table 3.3: Comparison of logger running legacy framework with the same logger using the new framework.

<table>
<tr><th>Legacy</th><th>New</th></tr>
<tr><td colspan="2">Hardware sample rate</td></tr>
<tr><td>The sensor is read periodically.
CONS:
Values between readings are lost, and potential site anomalies are not recorded.</td><td>Real-time sensing. Any deviation of more than 1% of the current value is recorded.
PROS:
Accurate sensor trend mapping.
All potential site anomalies were recorded.</td></tr>
<tr><td colspan="2">Hardware data upload frequency</td></tr>
<tr><td>Hourly
CONS:
Proactive monitoring is impossible.
Potential losses accumulate while waiting for data.</td><td>Real-time.
Propagation latency effectively 2G network latency.
Average latency 0.8 seconds.
PROS:
Instant access to system status and data.</td></tr>
<tr><td colspan="2">Data protocol used to upload data</td></tr>
<tr><td>Email
CONS:
Slow, expensive</td><td>Propriety IP protocol with 16bit CRC checksum
PROS:
Resilient, fast, and more performant in poor 2G network coverage.</td></tr>
</table>

3.6.1 System Design of new reversed engineered GSM Logger design

The gsm logger has three functional components, namely :

i. The input interface comprising of analogue inputs.
ii. The motherboard
iii. GSM modem interface
iv. Power supply

This module provides isolation between the motherboard, i.e. the microprocessors, and the sensor world. Because this is an industrial logger provision in the design was made to interface 4-20 mA signal, different input voltage signals, and pulse inputs.

The power supply of this device caters to a battery and/or mains supply. With no mains supply, this device will supply the motherboard with a voltage of 5 V and the sensor loop a 12V supply. The microprocessor controls the power cycles to extend the battery life of the device. The design includes a watchdog to put the device to sleep when not needed and wake up the device to transmit when the prescribed time lapses or the value changes more than a prescribed limit.

The GSM modem is interfaced to a microprocessor via Uart and DMA.

The reverse engineering of this GSM logger demonstrates how technology has evolved over the years to the point where the design architecture is so modular that a 3rd party vendor could simply write some new code. The board is good to go to a point where the new firmware could make it function utterly different from its original design and purpose. When reverse engineering this particular board, there were numerous challenges; the one and most common in propriety type designs is that chip identification is often coated with a coating that inhibits any 3rd party vendor from quickly identifying the chips or IC’s used.

Initially, when we embarked on reverse-engineering the board, we weren't expecting any 100 per cent success. Still, we certainly got more than we bargained for, which proves how the nature of the electronic world today is driven by software more than Hardware architecture or at least how the modern hardware is dependent on the software and firmware component of the devices. The other lesson learned that also speaks to software development is how writing new firmware rejuvenates an ageing hardware device to a new type of device that can communicate in a new framework.

Also, working with an embedded device that is quite legacy(not that legacy just over ten years old) as well as working with some new microcontrollers of today like the stm32f4, we can appreciate the evolution of these microcontrollers like the Nested vector interrupt controller and the DMA of the modern microcontroller certainly adds steep improved functionality. With the new microcontrollers, there are many more options/ programming interfaces available to code these microcontrollers as opposed to the microcontrollers of the legacy-type devices.

New Type GSM Logger

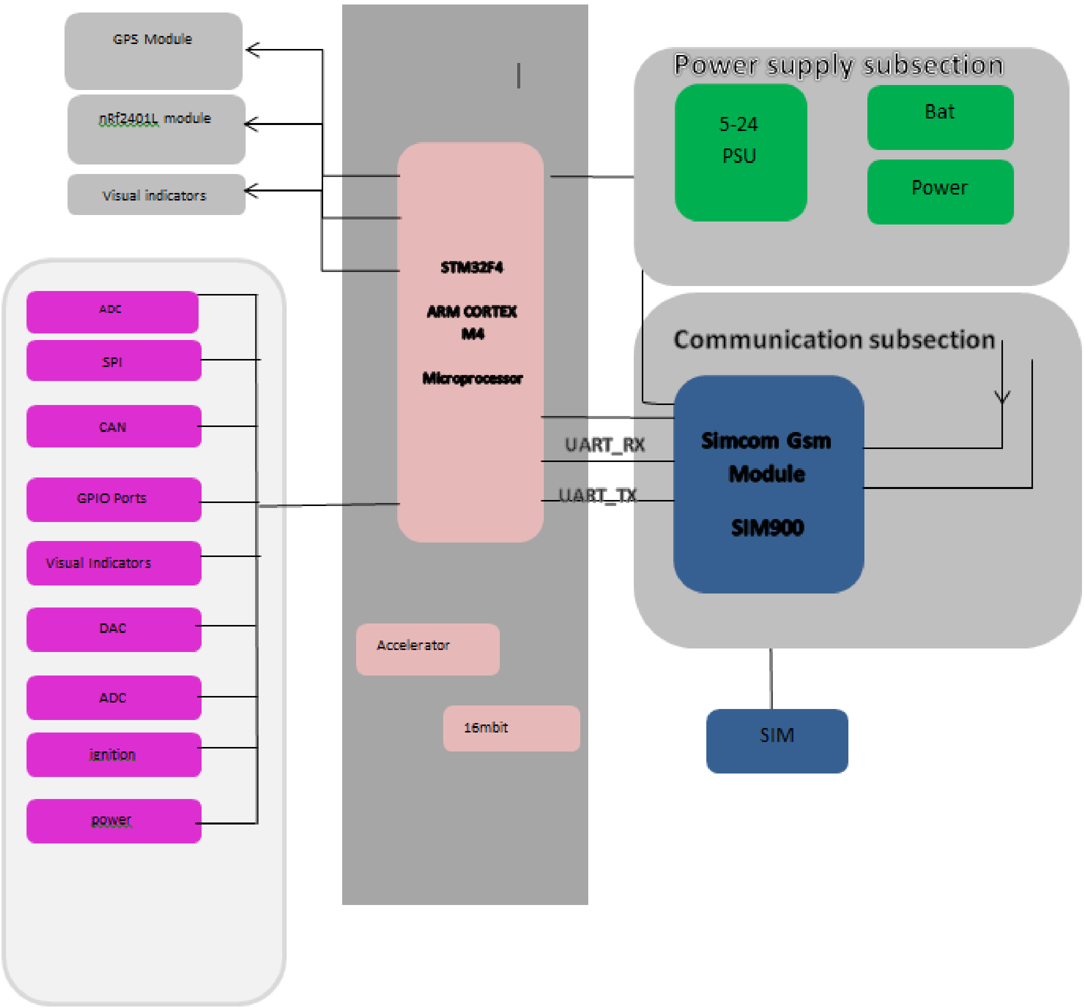


Figure 3.5 Functional diagram of new type GSM Logger

The above diagram is the functional diagram of how design is implemented on the stm32f4 discovery board.

This first phase of the design used an stm32f4, a 32-bit microcontroller. The stm32f4 development board can be seen as the mother module to three other daughter modules interfaced via various communication interfaces in both the Transmitter and receiver modules. These modules are connected to the stm32f4 module via the GPIO, UART, SPI or CAN

There is fundamental software development understanding required to carry out this project. The project requires expert knowledge of embedded design and to be able to program a microprocessor in the C-language or any other low-level language. There are, however, many rapid development platforms that make embedded development more manageable, but with its limitations and it also relies on libraries already developed for that particular microcontroller. C is a very powerful language that allows a programmer to access the inner workings of microcontrollers. The project employs expert knowledge of implementing buffers and numerous modules, all competing for CPU time. The design will also employ advanced c programming techniques and use Finite state machines to execute the code. Apart from the low-level programming requirement, there is a need to have a detailed knowledge of some higher-level languages like C#, python, or the likes.

In this thesis and rapid development, a mikro 4-20mAclick board was used to interface to the microcontroller board. This board employs the same principle of operation mentioned above, using its components to achieve 4-20mAto the data interface. The board incorporates an INA 196 current shunt monitor, its own 12 bit ADC in the form of MCP3201, and a DC/DC boost convertor(TPS61041). The 4-20mAR-click board communicates to the target board the LoRanwan development board using SPI. (Azhari and Kaabi, 2001)This interface can operate with either 5V or 3.3V, making sit very convenient when interfacing to a modern microcontroller.

This board serves as a receiver in the 4-20mAcurrent loop standard. It receives an output current from a transmitter/sensor and translates it into 0.4-2.4

GSM Logger as a data source to the device Gateway

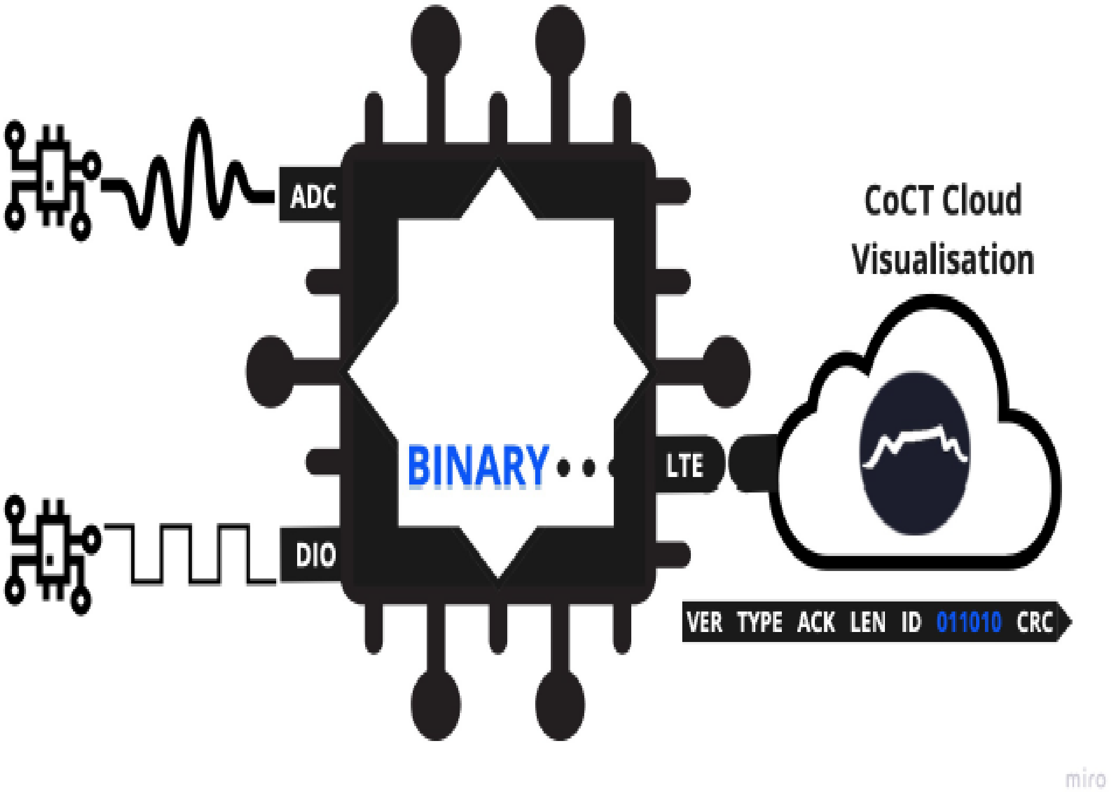


Figure 3.6: Illustration of value chain using a GSM logger as a data source

Figure 3.10 shows how sensor data from a 4-20mAsensor or pulse input can be monitored and uploaded in a binary format and transmitted to the device gateway residing in the cloud.

3.6 Implementing LoRaWAN as a data source

The LoRa wan architecture component comprises multiple LoRa end node devices, one or more LoRawan Gateways a LoRa network server(LNS), and a LoRa application server. Apart from accomplishing their primary function as a data provider, all data sources in this research have a second requirement: publish or parse data to the central device gateway in the prescribed CoCT packet format.

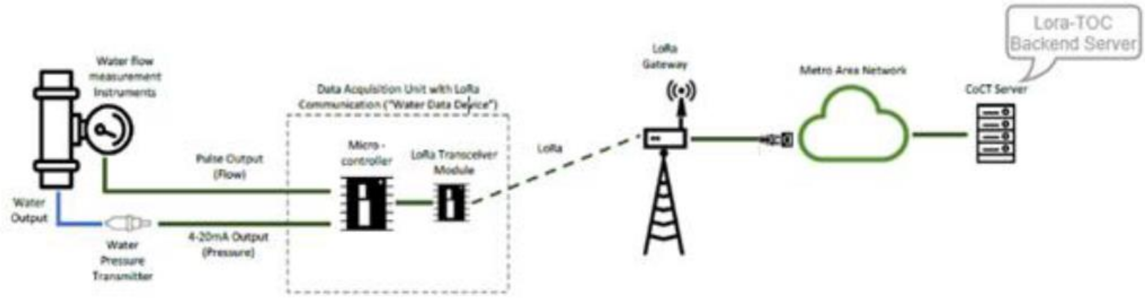


Figure 3.7 Private LoRaWAN Architecture

Payload messages are received from the Robustel R3000 Gateway using the Gateway's GSM interface relayed to the CoCT backend server' in a JSON format. Node-Red gets the message on the server, which decodes the payload into readable metrics and stores the metrics in SQL Database.

3.6.1 LoRa end Node Device.

The LoRa end node device was implemented from a low-level/ embedded solution for this research. This approach lay the platform for the study to better understand the technology for future purposes. Also, LoRa WaN is a technology of interest in South Africa at present. Having an in-depth understanding of this technology will plot a path for using this technology in the south African context and especially the City of Cape Town.

This research did not embark on the off-the-shelf option as this would restrict the learning of the LoRa Technology. Also, one of the study's objectives was to have low power wan technology that could present multiple sensors to one end node for transmission. The off-shelf options available in South Africa only had one sensor, one end node option available at testing.

The development board chosen to develop an end node for this research was the B-L072Z-LOWAN development board powered/driven by STM32 micro. Proof of concept successfully carried out on the LoRa PHY on an STM32f4 discovery development board. Because the B-L072Z board has a built-in LoRa radio module and the fact that it is Mbed-enabled (STM online rapid development IDE) made this an obvious choice for testing and implementing the LoRa Wan end node.

Because the research was testing the technology from a proof of concept perspective and that the LoRa PHY concept implemented in an earlier phase of the design already proved the ability to interface to multiple sensors using a 4-20mAinterface and voltage sensor interface, it was not necessary to interface sensors for this phase but instead sending dummy data seemed acceptable.

3.6.2 LoRa WAN Gateway.

LoRa wan Gateway is a Communication gateway made up of usually a LoRa radio module. LORA CCONCENTATOR and LoRa UDP packet forwarder and cellular interface two Ethernet ports with a WAN interface. Typically a LoRa gateway consists of an SX1301/sx1308 gateway chip and a rudimentary software bridge between the SX1301 registers (low-level interface) and the IP-based world (high-level interface). To further

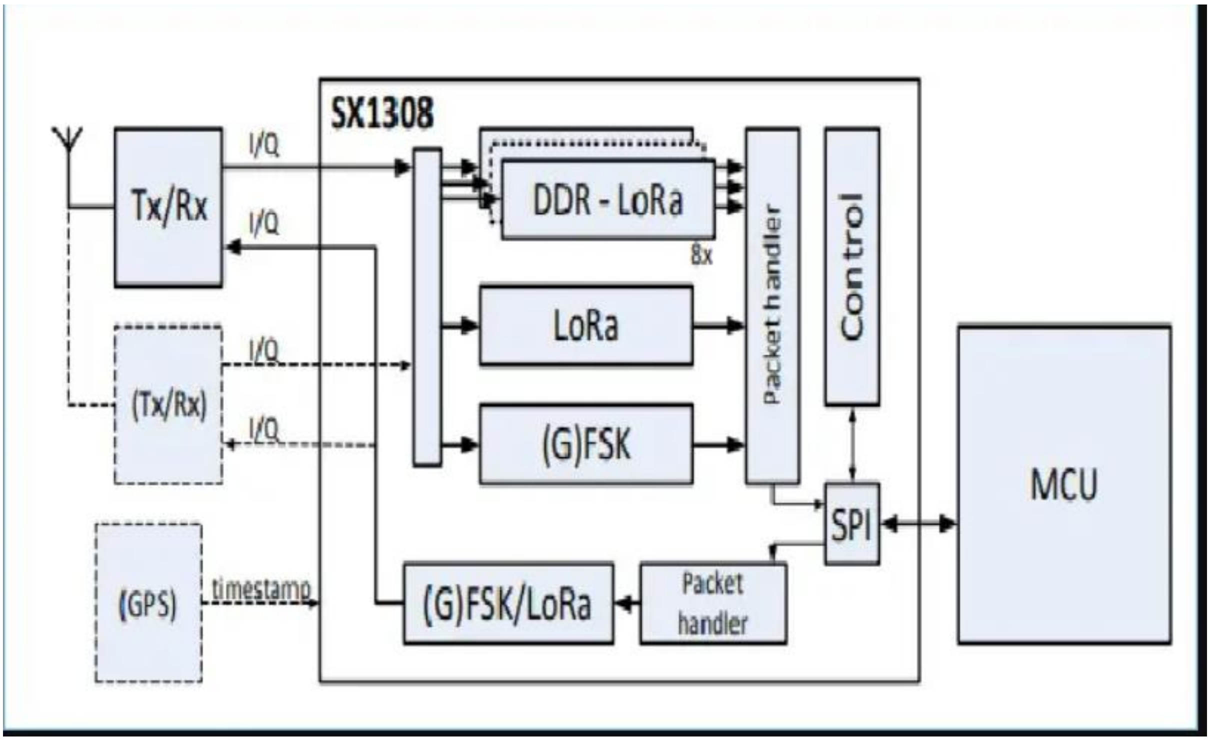


Figure 3.8 Modular layout of the Typical LoRa Wan Gateway

There are few variations of the LoRa wan gateway modules available. The more common ones are Kerlink, Rak LoRa modules, Robustel, etc. For this research, we chose the R3000 LoRa wan gateway. There are options to have the LoRa wan server and application server running on the Gateway itself, or the LNS and application could be running remote in which the Gateway would be configured accordingly.

3.11.1LoRa Gateway Bridge

Translates the packet data received from the UDP Packet forwarder into JSON and parses this data to the LoRa server over MQTT. The LoRa Gateway bridge could be present in the Gateway or Local Network or Remotely.

3.11.2 The LoRa Network Server(LNS)

It is the core component of the LoRa Wan Architecture. It is responsible for de-duplication and handles the received uplink frames from the gateway(UDP Packet forwarder)< Handles the LoRa MAC layer and schedules downlink transmissions.

3.12. The LoRa App Server

LoRa Wan Application Server is responsible for Join requests, encryption of application payloads, and a restful JSON API or MQTT for external services.

The LoRa Wan server used to view the end nodes was Loriot, and the Application server was a cloud-based Loriot Application Server. To get data to the CoCT gateway, we parsed the data from the Loriot Application server using MQTT. The MQTT data was parsed to the COCT back-end server in JSON format.

For visualising the Lora Wan data locally, the Back-end server comprised only of Node-Red and a SQL database. Node-Red is a process flow-based application that is built on top of node.js.

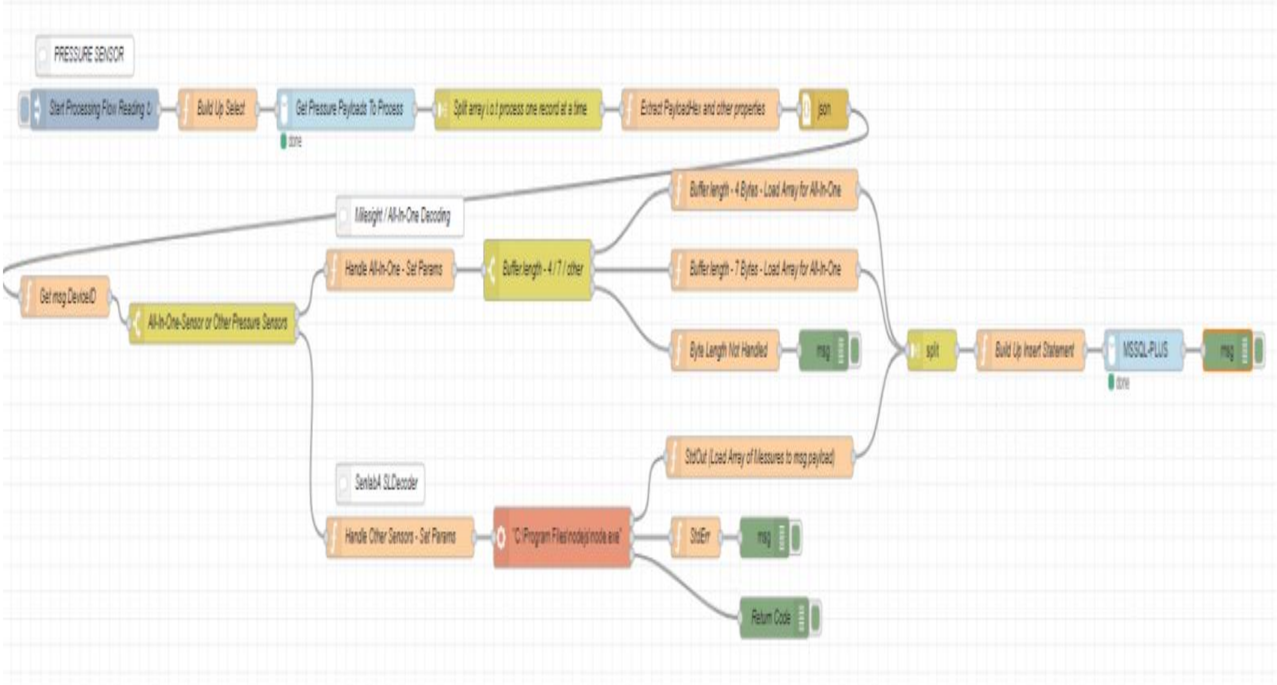

PRESSURE SENSOR
Start Processing Flow Reading
Build Up Select
Get Pressure Payloads To Process
done
Split array i.o.t process one record at a time
Extract PayloadHex and other properties
json
Milesight / All-In-One Decoding
Buffer length - 4 Bytes - Load Array for All-In-One
Get msg DeviceID
All-In-One-Sensor or Other Pressure Sensors
Handle All-In-One - Set Params
Buffer length - 4 / 7 / other
Buffer length - 7 Bytes - Load Array for All-In-One
Byte Length Not Handled
msg
split
Build Up Insert Statement
MSSQL-PLUS
done
msg
StdOut (Load Array of Messures to msg.payload)
Senlab4 SLDecoder
Handle Other Sensors - Set Params
"C:\Program Files\nodejs\node.exe"
StdErr
msg
Return Code

Figure 3.9 Node-red Flow for Lora back-end Server

Flows are essentially a unit of work similar to an application with a specific purpose. The following is a list of Node-Red flows defined and developed to handle the receiving payloads, process the payloads into metrics, and address anomalies. Listener Flow, A single listener, is defined on port http://172.27.0.20:1880/Senlab Figure3.8n - Flow Listener This flow receives all flow payloads and pressure payloads from the primary Kerlink Gateway and stores the payload respectively in tables FlowPayload and PressurePayload. The device must be registered in the Device table to establish whether it is a flow or pressure sensor.

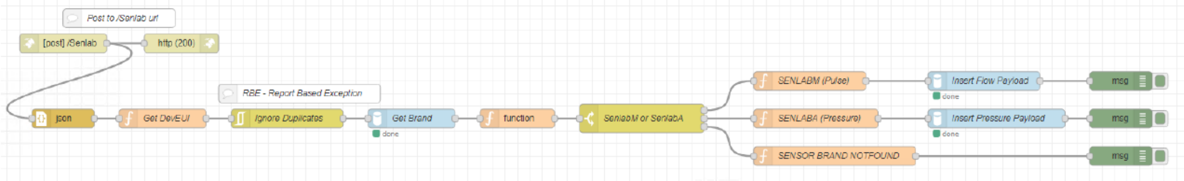


Figure 3.10 Listener installer on COCT Back-end Server.

The listener in Figure 3.6 above receives all Flowrates and pressure payload from the Loriot application server in JSON Format via MQTT. Once the data is received is stored in the respective tables in the database.

3.13 Gateway Data JSON Format

The data format is in JSON, with the app server running on the Gateway with an MQTT bridge.

It comprises information about the LoRa device, including device id, devEUI, port number with metadata including time, frequency, modulation, data rate, coding rate, etc.

```
{

        “app_id”        :        ”LoRaNetworkPotsdam”,
        “dev_id’        :        ”sensor1”,
        “hardware_serial”       : “ffffffffff”,
        “port”  : 222,
        “counter”       : 85,
        “payload_raw” : “AAAAAAAAAAAA=+”,
        “metadata”      : {
                “time”  : ”2021-04-19T09:44:52.82356732”,
                “frequency”     : 868.1,
                “modulation”    : “LORA”,
                “data_rate”     : “SF125BW125” ,
                “coding_rate”   : “4/5”,
                “gateways”      : [
                        {
```

```
                “gtw_id”       : “eui-7076ff005606032c”,
                “timestamp”    : 653587540,
                “time”         :” “,
                “rssi”         : -122,
                “snr”          : -17.8,
                “rf_chain”     : 0’
                “latitude”     : -33.90056,
                “longitude”    : 18.489706,
                “altitude”     ; 19
            }
        ]
    },
```

3.14 SQL Database as a Data source

To enhance the data value chain and parse intrinsic data in a low latency data framework, it was necessary to develop a framework that provided aggregated data to WEB API or an MES or some data analytics platform. In this research, a proof of concept was initiated to extend the data visualisation of plant data to a web visualisation platform. As mentioned earlier, in a typical process plant environment, there is always certainly some database storing key process data and tags from the process control systems of the plant
The research investigated ways of presenting this data in the same format as the other data sources in the standard COCT packet format.

Process plants storing intrinsic data of the plant In the local SQL database. The data stored in this database is the tag historian database of the existing process control system. This data is stored in no particular nomenclature or Naming convention, and its purpose is to provide plant history. This data can be aggregated to supply essential plant statistics and important plant KPI and OEE of the plant. Still, this first-level data needs to be aggregated to provide this functionality. Because the CoCT packet format is prescribed for constrained network environment and IOT in nature, there are limitations in terms of structure; it is, therefore, necessary to parse the data in this prescribed format for the gateway to de-serialise this data accordingly.

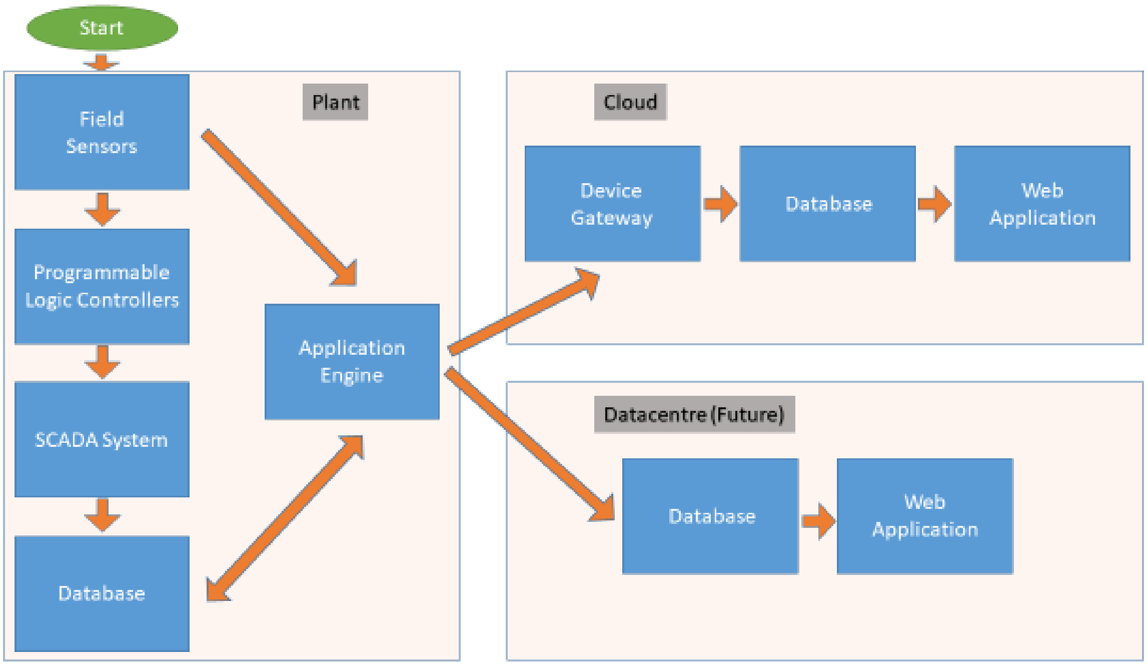
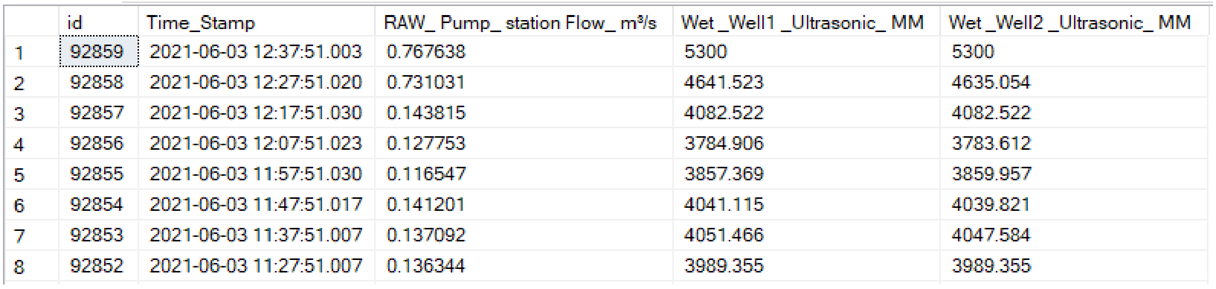


Figure 3.11: Schema of the data value chain of SQL Datasource to WeB API

Figure 3.11 above can be summarised as follows:

- The sensors send the information to either the PLCs or directly to the C# App Engine.
- When sent to the PLCs, it takes the usual route – Sensor to PLC to SCADA to Database and finally to the App Engine where it gets parsed from SQL to JSON objects and ready for further processing.
- When sensor information is sent directly to App Engine, the info gets parsed to JSON objects, transferred to the local plant database, and then to the Device Gateway in the cloud.
- At the Device Gateway, information is stored in the database and processed through the Web server for visualization
- The SQL stored procedures are primarily used for data aggregation.
- The procedures are divided into sections based on the plant's process flow.
- Taking a Waste Water Treatment plant as an example, one of the data aggregation points will be the "Raw Water Pump Station."
- The key metrics will be the Pump availability, based on the run hours and the total pumps running for a specific period.
- Other information such as the flow rate and wet-well height can also be used to determine Overflow conditions due to pumps not running or Overflow conditions due to heavy rain.

Table 3.4 Screenshot of a DB table currently being used.



	id	Time_Stamp	RAW_ Pump_ station Flow_ m^3/s	Wet _Well1 _Ultrasonic_ MM	Wet _Well2 _Ultrasonic_ MM
1	92859	2021-06-03 12:37:51.003	0.767638	5300	5300
2	92858	2021-06-03 12:27:51.020	0.731031	4641.523	4635.054
3	92857	2021-06-03 12:17:51.030	0.143815	4082.522	4082.522
4	92856	2021-06-03 12:07:51.023	0.127753	3784.906	3783.612
5	92855	2021-06-03 11:57:51.030	0.116547	3857.369	3859.957
6	92854	2021-06-03 11:47:51.017	0.141201	4041.115	4039.821
7	92853	2021-06-03 11:37:51.007	0.137092	4051.466	4047.584
8	92852	2021-06-03 11:27:51.007	0.136344	3989.355	3989.355

Above Table 3.4 is a screenshot of a database table currently being used.
Based on the information below, the first line can be identified with heavy rain being present since the pumps are running (flow rate is not “0”), and the wet well is high (5300mm is the max height)

To accomplish this, there is a requirement to aggregate this data and present it in the required JSON format. The Local plant SQL database can provide this data to a third-party application if needs be. However, from an edge perspective, using a constrained network providing SQL queries across a constrained network is not the best solution in terms of latency. However, an option is to provide necessary data using a lightweight framework if this data could be aggregated and presented to the device gateway as a JSON object. This option would translate to designing a C# engine to do this aggregation and parse/translate this data to the LTE modem to transmit this data in the prescribed CoCt packet format.

The working of this C# application or any type of application running on the database server would be accessing several stored procedures of the SQL server and parsing it a Jason objects to the Lte router/device.

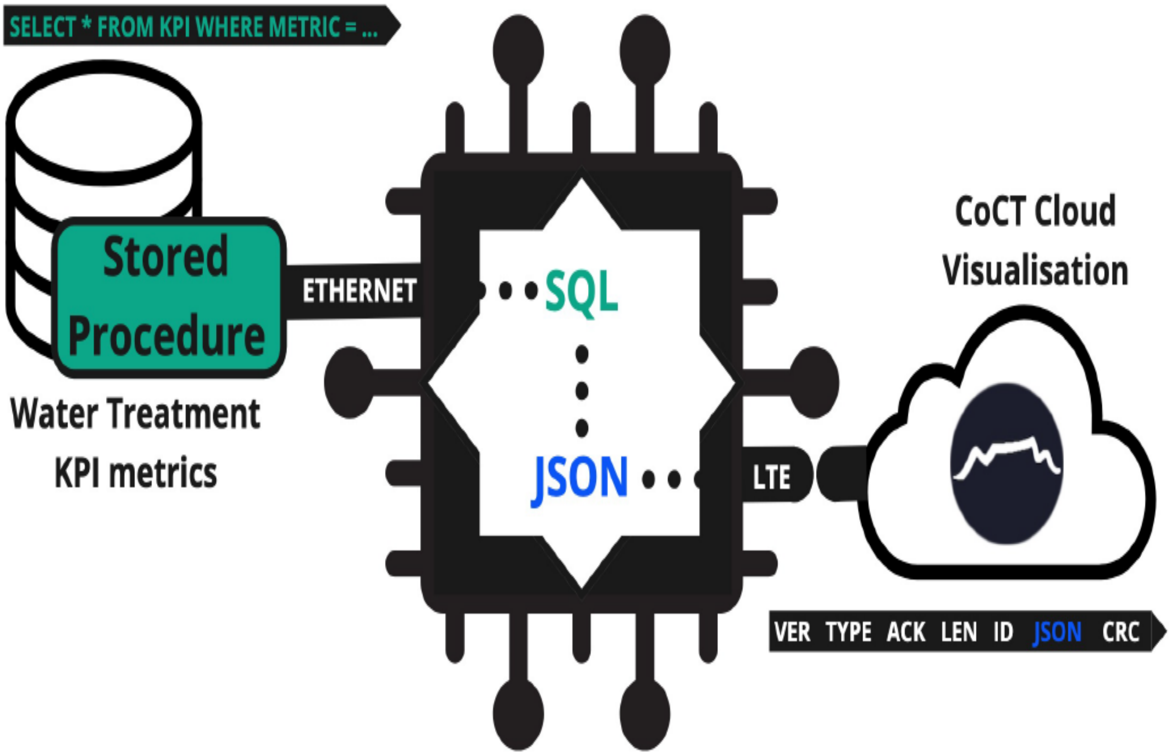

SELECT * FROM KPI WHERE METRIC = ...
Stored Procedure
Water Treatment KPI metrics
ETHERNET
SQL
JSON
LTE
CoCT Cloud Visualisation
VER TYPE ACK LEN ID JSON CRC

Figure 3.12: Graphic of SQL Database as a data source to device CoCT Gateway.

To get from the local plant database to the gateway in the prescribed format for the CoCT gateway, an engine(C# application) had to be installed on the Local SQL server together with a SOC Lte router. This will be responsible for sending the data to the CoCT device Gateway in a CoCt packet format. See Chapter 5

3.14 The device Gateway

The device gateway has been developed to cater to various data sources and to scale and manage all current and future data logging devices.

Even though it's currently for this research developed to log limited data sources, the protocol and abstraction layers allow easy scalability beyond the present data sources. (Astarloa *et al.*, 2016) Careful consideration was taken during the initial design of the gateway, as the existing methodology of the legacy loggers that were re-engineered used email to handle data collection posed numerous limitations and latency issues.

The existing legacy loggers were re-engineered to provide real-time, low-cost updates. In addition, the new gateway, apart from being scalable and device-agnostic, also offer added benefits like a genuine “real-time” telemetry with significant low-cost data implications. Both aspects of these features are demonstrated by combining the new firmware and the device gateway.

We currently leverage the use of Amazon's presence in Cape town to host our gateways and webserver; however, the gateways infrastructure can be hosted anywhere. This migration, including locally hosted, can be done without any hardware updates as we use DNS resolution to route/direct the data.

Cloud-based solutions can be as secure as internally hosted systems. One simply needs to architect the solution around the required level of security. There is a common misconception that cloud-based systems are insecure; however, this is true when incorrect resources set up the infrastructure. Cloud-based systems have many advantages over internally hosted systems, and the most predominant one is your ability to hide core infrastructure from the public realm.

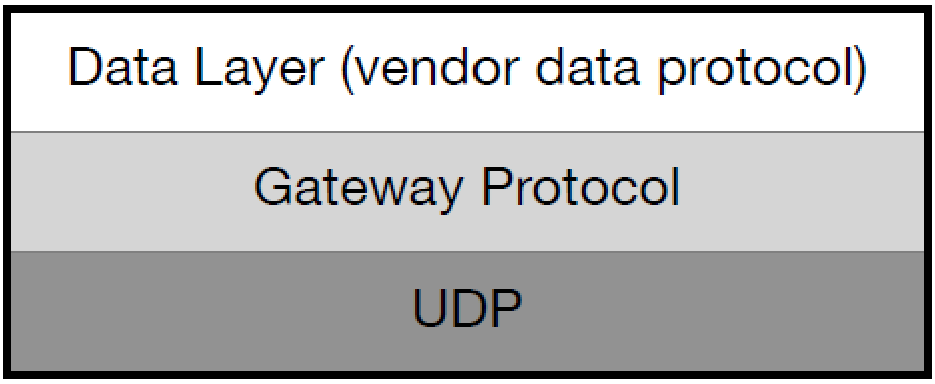
The SOC-Based SQL engine also proves the gateway’s device-agnostic abilities to successfully log data to the same device gateway as the loggers. Both data sources use the same transport layer; however, the data layer handles ‘binary” data for the loggers and ‘JSON” string for the latter.

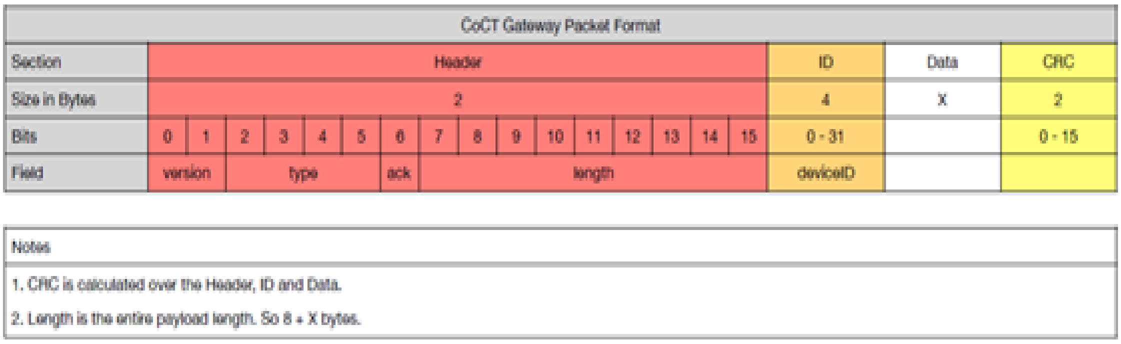
3.15 Gateway Protocol Layers

For this research, we will focus on the protocol implementation of the telemetry layer. However, its worth mentioning that the design of the visualisation engine allows for cloud-based calibration of the incoming data.

IoT telemetry is essentially low bandwidth sensor data. All cloud-based telemetry is essentially some form of IP format data. Most IoT protocols are TCP based as it handles connection and stream management needed to transfer data. For example, MQTT and the email protocol used legacy data loggers is a layer above TCP. When it comes to large scale platforms receiving telemetry from tens/hundreds of thousands of devices, concurrency, arbitration, and data frequency affect scalability and cloud resource requirement (i.e., cost of scalability)

Another consideration for largescale platforms and frameworks is whether any sensor data will be transmitted over GSM. With the move to NB-IoT and other low power Wan solutions, TCP connections are sensitive to delays, connection overhead, and packet loss during data transmissions. This further impacts battery life for a device that is battery operated and also scalability. And so, the natural progression is to move to UDP to mitigate the above factors. For this compelling reason, UDP is used as the Baselayer from which the CoCt gateway protocol evolved. This introduces some complexity to the base protocol of the additional features required to minimise packet loss, but the benefits far outweigh the extra complexity. To have this additional complexity, the protocol is split into two layers. The base “gateway protocol” to handle the complexity mentioned above and other functionality added to optimise the use, and the data layer embeds the actual sensor data being sent.





CoCT Gateway Packet Format

Section	Header																ID	Data	CRC
Size in Bytes	2																4	X	2
Bits	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0 - 31		0 - 15
Field	version		type				ack	length									deviceID		

Notes
1. CRC is calculated over the Header, ID and Data.
2. Length is the entire payload length. So 8 + X bytes.

Figure 3.13: Standard COCT packet format

It is vital to note CoCt packet format is the name given by this research to the packet format we developed and used as a standard to parse data to the device Gateway. The rationale for this was a gateway that could receive data from different data sources and have a serialisation routine associated with the respective data source after a packet verification process. This packet format could also be given to any prospective vendor that wishes to parse data to this gateway where we would produce this format as specification /standard for the vendor to parse data to this gateway where we would have a de-serialization routine written for their data.

The CoCt gateway protocol is a communication layer between the respective device or data source and the gateway itself. This allows the device to register itself onto the network, update its current time/date, send data, and perform some other essential functions that otherwise would have been accomplished by the TCP layer. This protocol is extendible and backward compatible in its current format.

The basic description of the fields of the CoCt packet is seen in the figure above:

Version control(First), Length(forth) and CRC checksum(last) are used to ensure protocol integrity. Once the protocol integrity is validated, The message type(second) is evaluated to understand the device's request. See appendix

The device requesting is identified by the deviceID, which is assigned as part of the registration process. The acks are used to for data handshakes. It is essential to understand that the message types defined at this level have nothing to do with the vendor's actual sensor data and message types. This is part of the “Data Layer” encapsulated in the data (sixth) field. There is underlying complexity in maintaining the device and its data integrity. This is the purpose of this layer.

3.16 Data Layer

The data layer transports the sensor data in a format specific to that device or vendor. This data can be anything from binary to ASCII strings. The decoding of the data layer is done within the Visualisation engine. Why? Reasons are two-fold:

i. Most importantly, to ongoing abstract development from critical gateway functionality and for increased resilience. This will reduce the complexity of the QA and deployment cycles and the potential network-wide downtime caused by developer error. •

ii. To aid rapid development and deployment of new hardware. Infinite complexity can be managed without having to change nor update core protocols. And adds “forward” compatibility required the hardware vendors to ease deployment of new code in parallel to the development of internal

CHAPTER 4

DESIGN OF IIOT DEVICE TO PARSE DATA DIRECTLY TO SCADA SYSTEM

With the advent of the 4th Industrial revolution and the need to improve productivity levels, there is an increasing need for data analytics. With process control systems driving many ERP systems and decision-making, there is an inherent need to deliver real-time data to MES systems and ERP at low cost and reduced data latency. Also, with some assets being distributed over a large geographical area, there is an intrinsic need for a remote SCADA system at low operating costs.

The proof of concept was implemented on an stm32f4 development board to realise the performance of two wireless technologies.

This research implements hardware and software solutions design using two wireless technologies and protocol wrappers and interfaces to enhance the data value chain in a legacy process control system. The technologies investigated are physical layer LoRa nodes, nrf2401l(a 2.4GHz radio module), MQTT, 4-20mAConvertors.

4.1 Introduction

The race in the 4IR space is reaching epic proportions, especially in industrial automation space. Process control engineering is not excluded from the innovations and drivers of this revolution; it is where a lot of research and innovation is taking place to bring expensive propriety systems in line with the 4IR space. There is also an inherent need for SCADA systems to integrate into Maintenance management systems, ERP systems, and MES systems to present an effective, low-cost data model and, in doing so, improve MTTR, reduced production cost, etc. This means that many legacy systems need some form of integration capability or bring data that legacy systems do not have the capacity to integrate into the higher-level solution with minimal costs.

The 4th IR is all about data, data consumers, data drivers, data cost, data latency, data integration, data decision making, and the list goes on. The process engineering data is not new; it is the source of data and thus one of the key data drivers in the 4IR, but this data came at a price.

LoRa Phy, nrf24l01 radio module, LoRa to IP bridge. Different hardware and software implementations and protocols are researched on a case study basis to achieve interoperability from field devices to Erp and MES systems. Hardware interfacing of 4-20mAinterfacing to wireless node, comparing different technologies to achieves this. A low-level and barebones approach will achieve all these implementations.

4.2 Existing and Proposed architecture for Modern Process Control Systems

There are essentially two disparate technology systems that seem to be converging in the 4IR automation world. The legacy process control systems comprising PLCs, HMI, and SCADA, and the other is a relatively new role player in the game, i.e. IIOT systems, namely microcontroller-based process systems. The architecture of these systems is different, yet they relate to the same on the higher-level data consumer level. The older legacy systems lend themselves to expensive propriety-based solutions. In contrast, the new microcontroller systems offer incredibly cheaper open architecture and more widely used protocol systems, resulting in substantial cost savings. However, one system does not negate the other and both systems sometimes need to be implemented in the new data consumer-driven world. There is a growing need for the systems to augment each other. (Pagnon, 2017)

This means that research needs to get low-level and field-level systems to parse data seamlessly to high-level systems offering greater interoperability and good latency when used. This research looks at different field-level devices and interfaces and a communication medium to enhance the seamless interoperability of data between all the levels of the data value chain.

To put some perspective on this, this research provides a solution at an existing Waste Water treatment plant with a legacy process control system with an existing SCADA and HMI system and an SQL database. (Drahos *et al.*, 2018)This design will look at getting additional data back to data in a warehouse and to a High-level system like ERP/ MES system or web service.

A Typical modern Day Process control system comprises a PLC system with an OPC server, a SCADA system, HMI, and sometimes a SQL database server, PLC or PAC with field devices, Actuators, instruments connected to it.(Langmann and Rojas-Pena, 2016) The typical modern-day process control system has its own local SCADA system together with its HMI. All the time series data acquired from the time series database residing in the SCADA historian can be further stored in SQL format in its Local SQL database. This data can be made available to the high-level application or a web application.

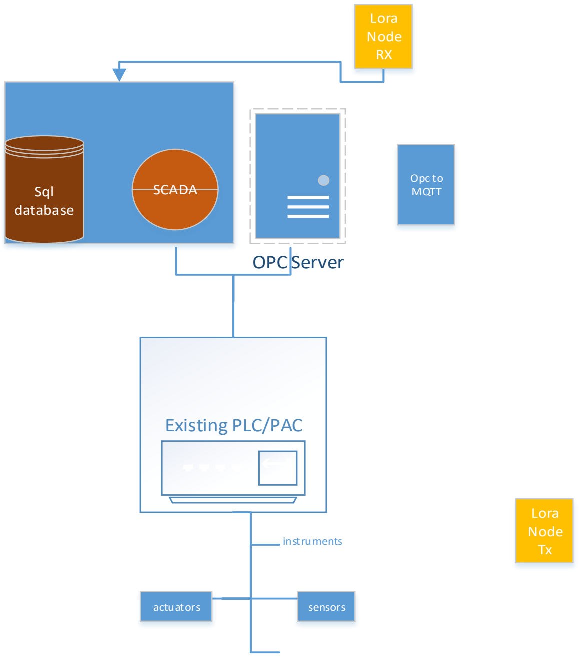


Figure 4.1. Typical modern-day solution with added IoT LoRa-device.

Figure 4.1 shows the added components to a system where the initial design did not include or accommodate adding of field devices. This is because either there was no capacity to have it in the process control system or the cost of physically adding the section to the process control system is too high because there is no more capacity in the physical infrastructure (current process control system/PLC) in place to add it.

With the growing need to integrate these systems into MES and ERP systems, it is sometimes found that these field instruments and/or devices need to be added to the existing process control to augment the data analytics that was not previously part of the existing Process Control System.

This research presents adding a measuring device that was not installed as part of the original process control system. There could be one of two primary reasons why this scenario exists. Scenario one, it is not practical/viable to add this extra instrument or sensor; either there was no capacity in the existing process control system because of lack of i/o's on the input card or the cost of physically adding the section to the process control system is too high because there is no more capacity in the physical infrastructure in place to add it while the distance between the Process Control System and the physical instrument is quite a considerable distance. On many occasions, these devices are situated hundreds of meters away from the process controller or SCADA Server like in a wastewater treatment plant.

A device that is required at present, which is essentially some years later after all the infrastructure is in place, can be quite an expensive and labour intensive task to add to the process control system, especially if fibre optic or some other wired point-to-point network.(Tao *et al.*, 2014) If a device was added to the existing process control system, there is no inexpensive off-the-shelf solution to get this device's data into the SCADA or process control system. An affordable wireless solution needs to bring this field device data to the SCADA system or a remote MES/ERP system; this research investigates two possible solutions.

To add this component to the system. There is an option to use one of two wireless IOTT options, namely the nrf24L01 2.4 GHz radio module or LoRa to LoRa node implementing the physical layer. Both these options will need an IP bridge or some protocol wrapper to get this data back into the SCADA system, which can also be further parsed to the OPC UA server for further transportation. At the source of the instrument, there would also be a need to convert the 4-20mAsignal or a 0-10 V to data variable to be parsed to a “LoRa or NRF2401l” format.

On researching the wireless platforms to transmit process data from plants, two technologies were realised as most appropriate. They could provide a viable solution for getting data from standalone industrial instruments back into the process system or SCADA system for local visualisation of data or remote visualization and /or input into MES or ERP systems.

Other technologies were researched to implement an appropriate wireless solution to transmit acquired data from the field device to the SCADA system. Still, two wireless technologies that stood out in terms of range, robustness, and bandwidth were nrf24L01 and LoRa node to node communication (LoRa PHY).

After researching and physically implementing the two technologies from an embedded perspective interfacing both the LoRa and the nrf24L01 modules to an stm24f4 development board, it was discovered:

The nrf24l01has far better data rates and bandwidth than the LoRa node but is significantly poorer in SNR and link budget and the distance it can cover for an end-to-end wireless data transmission. The nrf24l01 module is very inferior concerning range as it could only achieve up to 120 meters without data degradation.

LoRa, on the other hand, was very flexible in sending different data rates depending on the surrounding transmission conditions; it also has a built-in data rate adapter build into to LoRa physical layer. (This relates to the Spreading factor and forward error correction setting). LoRa also shows considerable superiority in terms of the SNR as compared to the NRf2401l radio. The NRF 24L0l module has superiority in data rates, but the NRF module is significantly poorer in range when compared to the LoRa module. This significant superiority

SNR robustness of LoRa can be attributed to the propriety signal transmission Chirp Spread Spectrum(CSS) that LoRa PHY uses.

Once received from the wireless interface, data needs to be converted by the IP bridge to parse data to the SCADA device driver/dll. LoRa node has significant bandwidth restriction but can be overcome or augmented by bit encoding techniques to optimize data transmission. LoRa node/LoRa PHY can be likened to UDP transmission and LoRa Wan to TCP. This means some form of CRC needs to be implemented to ensure proper data integrity if this is a priority, especially when using just LoRa Phy.

4.3 4-20 ma-to-data Interface

Most industrial type process control systems like PLC's or PAC connects field devices like sensors and actuators via 4-20 ma current loop directly into the process control system.
The 4-20 ma is a very rugged and robust way of processing signals in the industrial environment and has been found to stand the test of time in the harshest of industrial conditions, which is why it is so popular in the industry today. The 4-20 mA current loop is one of the most common sensor signalling standards.

Current loops are ideal for signal transmission because of their inherent insensitivity to electrical noise. In a 4-20 mA current loop, all the signalling current flows through all components; the same current flows even if the wire terminations are less than perfect—all the components in the loop drop voltage due to the signalling current flowing through them. The signalling current is not affected by these voltage drops as long as the power supply voltage exceeds the sum of the voltage drops around the loop at the maximum signalling current of 20 mA.

The 4-20 ma standard was adapted for process control systems, especially for signals with long wire runs and their robustness. Most standalone instruments are standard with the 4-20mAoutput or HART communications standard. There is, of course, process instrumentation with the 0-10V input/output.

The operation of the 4-20mAloop is straightforward: a sensor's output voltage is first converted to a proportional current, with 4 mA representing the sensor's zero-level output and 20 mA representing the sensor's full-scale output.

Converting/translating the 4-20mAcurrent loop to voltage signal is necessary. In the IoT space, most universal IoT controllers are microcontroller-based systems either capable of processing a 0-5v or 0-3.3V by its ADC controller peripheral. The other option of parsing the data to the microcontroller via its SPI interface using an off-the-shelf 4-20mAinterface. There are, of course, instruments and field devices that have other field device protocols that can be parsed to the microcontroller, like the Hart protocol, which is extremely popular in the instrumentation environment. But this research implemented translating the 4-20 mA signal into data that any modern microcontroller can process.

As mentioned, the off-the-shelf translators(4-20 ma to data) using an interface via the SPI peripheral of the microcontroller. The other submodules that form part of this 4-20mAinterface are the INA196 current shunt monitor, MCP3201 12-bit ADC, and TPS61041 DC/DC boost converter. It receives output current (4-20mA) from a transmitter and converts it into a voltage (0.4-2V). Then through the ADC converter sends a signal to the built-in microcontroller. This built-in micro then presents this data via an SPI interface to other devices that require this data.

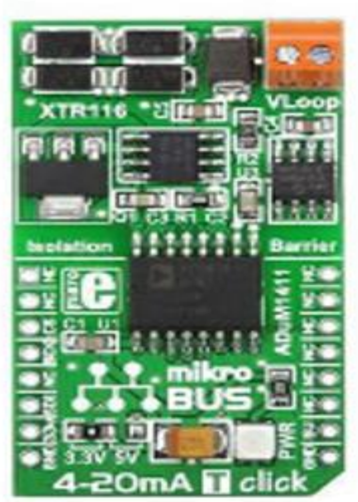


Figure 4.2: Module for measuring sensors 4-20mAwith SPI interface.

The microcontroller has different serial interfaces that can process/parse this data. The most common interfaces used in the modern microcontroller are SPI, I2C, UART peripheral, and CAN. In this design, the SPI interface is used to interface into the LoRa node and nrf24l01l modules to achieve the wireless transmission required.

With wireless technologies available to parse process control data, there is a need to convert industrial sensor data to a voltage or data for this data to be transmitted, like the flow rate of the outflow of a wastewater treatment plant.

The principle in this research is to pass industrial data via wireless device to a SCADA system. The 4-20 ma interface used in this research is a single-channel current to voltage converter. As mentioned in other research papers, there are options for parsing multi-channels 4-20mA interfaces if the application requires. Details on how to accomplish a multi-channel variation explained in these research papers relate mainly to using multiple SPI interfaces of the microcontroller.

The other method is converting the 4-20 ma current signal to a 0-3.3 Vage signal that the modern microcontroller can efficiently process by its ADC peripheral. When using the ADC peripheral, the signal processed by the arm stm32f microcontroller has a 12-bit resolution, meaning that 4096 samples were processed. This resolution plays a significant factor when scaling the values of the measured signal. Depending on what instrument or device is measured, this signal can be processed and scaled by the microprocessor and stored as a volatile variable to be buffered by the microprocessor and **parsed** to any function() or transferred to serial interface peripheral, which can be processed further for transmission.

Now that the signal is translated/processed the result is a variable stored in buffer/array in the microcontroller ready for transmission.

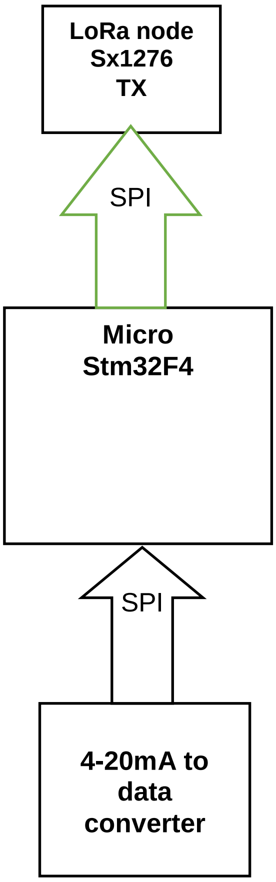


Figure 4.3: LoRa Transmitter node with 4-20mAConverter

4.4 The LoRa Transmitter

The LoRa transmitter, which consists of the LoRa radio module,4-20mAinterface, and STM32f4 development board, essentially converts the 4-20mAsignal into a variable parsed to a buffer in the microcontroller. This data is parsed from this buffer to the payload of the LoRa radio module to be transmitted over the air. The LoRa radio module is interfaced to the micro via SPI

.

4.5 The LoRa Receiver.

The LoRa receiver, which consists of the LoRa radio module, IP bridge(enc28j60), and STM32f4 development board, decodes data payload from the LoRa modulated signal and parses this data to the microcontroller. This data is parsed to a buffer in the microcontroller, an interface to the LoRa radio module via SPI. This data is then translated to an IP packet by the enc28j60 module. This module is connected to a packet manager application to the local network.

The packet manager is then connected to the local SCADA network via a router. It is also possible to have the packet manager and SCADA system residing on the same machine.

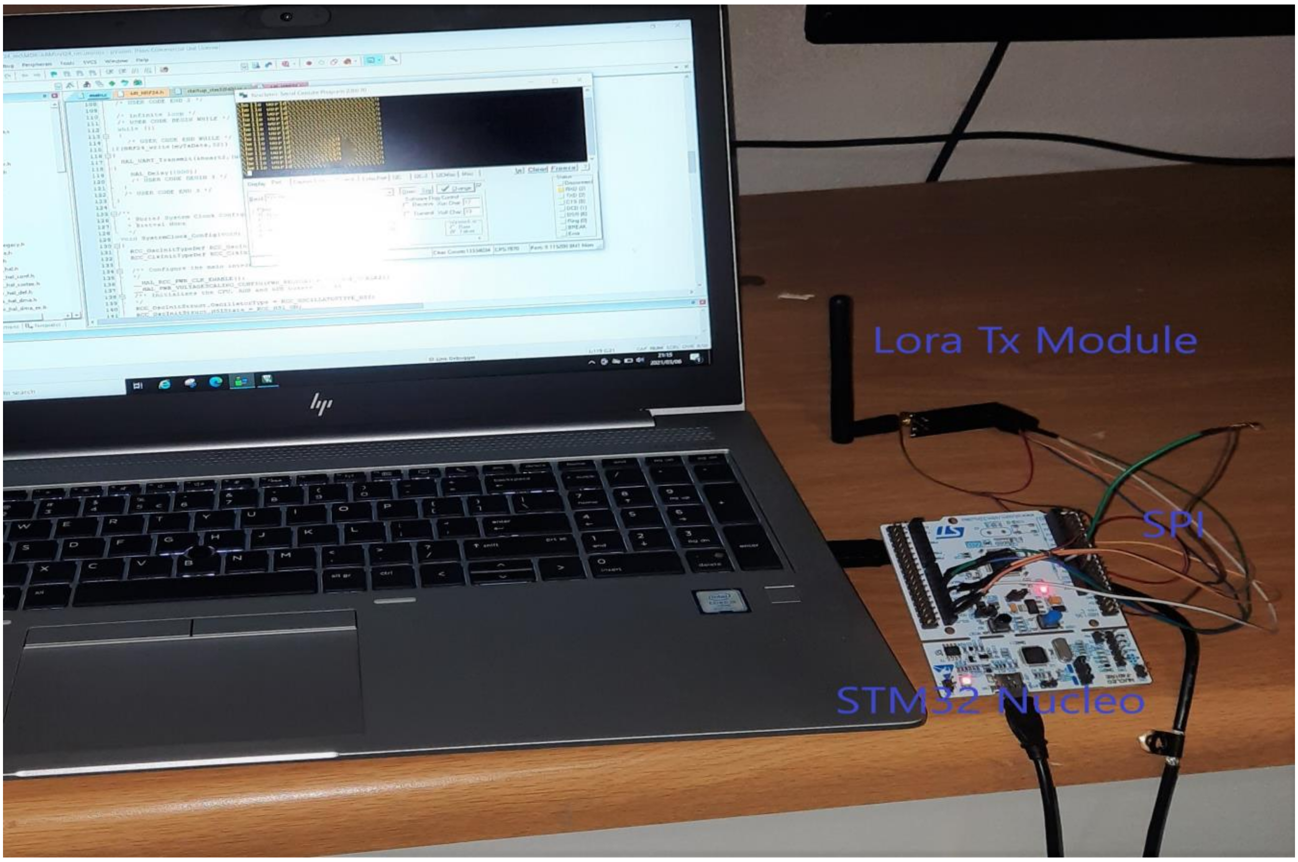


Figure 4.4: LoRa Tx module using Stm Nucleo Board

4.4.1 Phase 1 Development phase using Stm Nucleo board with LoRa Radio(Transmitter Tx) and Stm32f4 with LoRa Radio module(Receiver Rx).

The above figure illustrates the first phase of the design, which implements the proof of concept where the purpose was to test the pure idea of the LoRa physical layer. Dummy data was transferred from the Tx buffer of the microcontroller (the Stm Nucleo board) and parsed to the LoRa radio module for transmission. The Received LoRa CSS signal is received by the LoRa module, parsing the data into the RX buffer in the microcontroller (the stm34f4 discovery board). This data is visualised by Putty software via UART of the microcontroller.

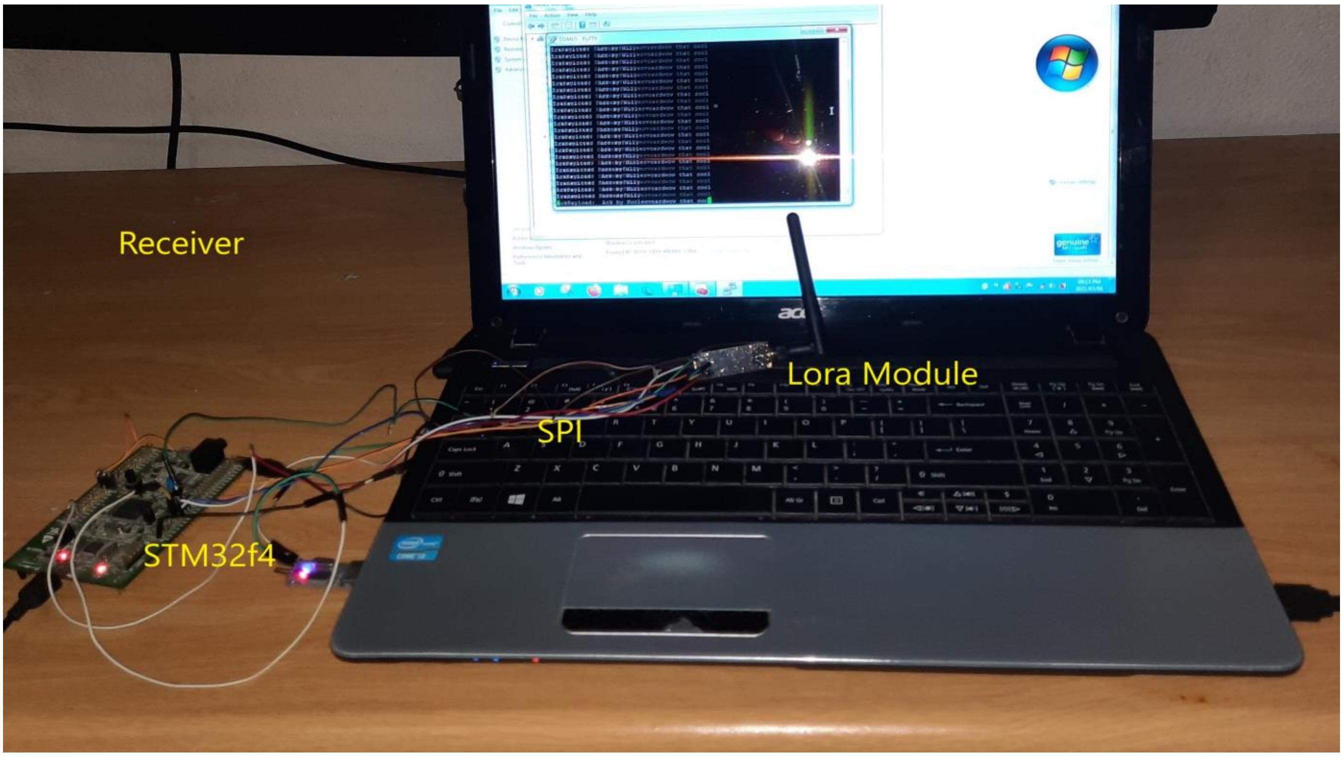


Figure 4.5 LoRa Receiver module implemented on STM32 Discovery Board

4.4.2 Phase 2: Implementation of design using the B-L072Z development board.

Phase 2 of the development implemented the solution on a different development board (B-L072Z running STM32L072CZY6TR MCU), which already included a built-in LoRa radio module(sx1276) interfaced into the microcontroller, which made working with the development board more convenient and less cumbersome. Only the 4-20 ma *mikroe* boards needed to be interfaced to the board in case of the transmitter node, and the ENC28J60 Ethernet module had to be interfaced to the Receiver node. The software coding was no different than using the previous board(phase 1). Still, the general hardware implementation was a lot more tidy and robust to work with, especially for testing the range of this implementation.

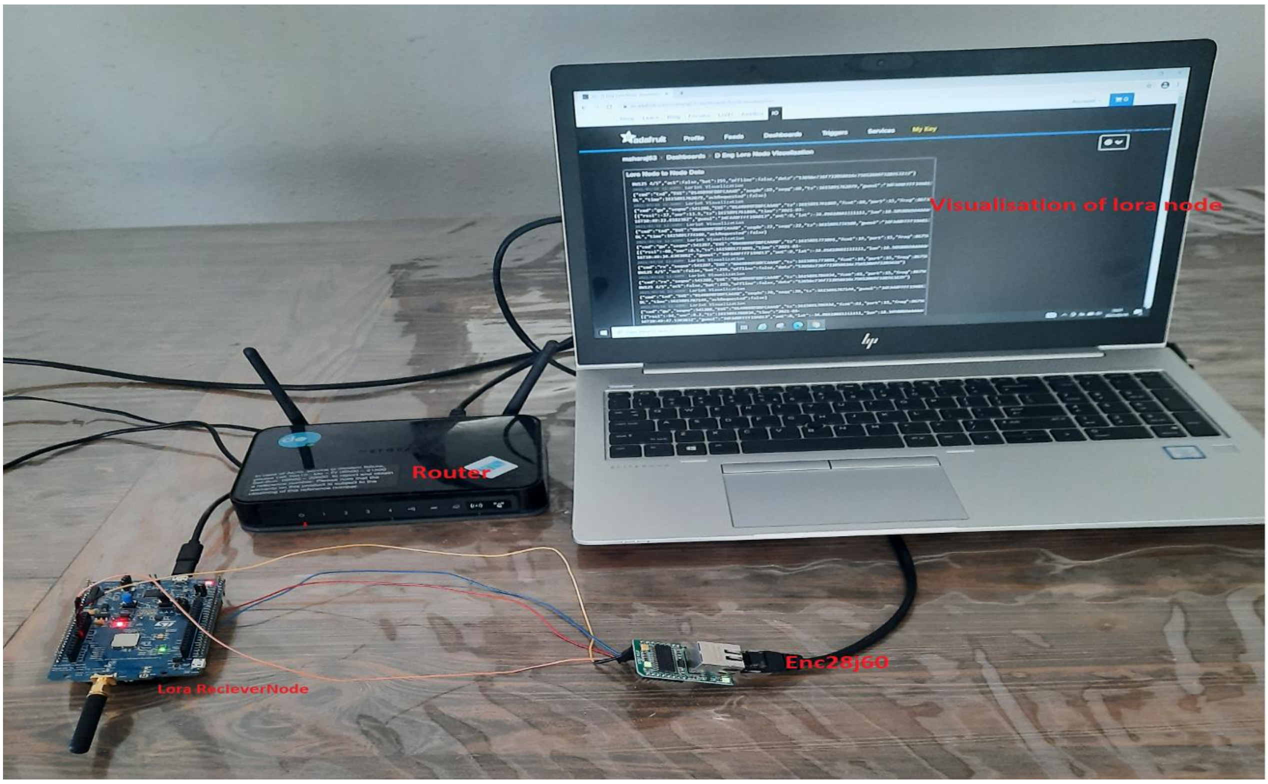


Figure 4.6: Phase 2 Receiver module with enc28j60 module (LoRa to IP wrapper)

4.6 The nrf24l01 Module

The nRF24l01 transceiver module operates in a 2.4 GHz worldwide ISM Band. This module uses GFSK modulation for data transmission with data transfer rates of either 250kbps,1Mps, or 2Mbps. The module supports programmable output power of 0dbm to -18dbm. There are also different variations of these modules available in the market today. The differences of these modules lie in the with/without power amplification option, which can be programmed via the SPI interface; the other variation is also with/without LNA, also the antennae available to attach to the module, which also has a significant effect on the range of the module.

The module is rated to consume 12mA at 0dbm during which makes it ideal for low-power applications. The transceiver also easily interfaces to microcontrollers using an SPI interface at a data rate of up to 10MBps. It should also be noted that varying the data rate also impacts the range of this module. During the implementation of this research, the transmission range was a determining factor and therefore used the 250Kbps.

The nrf240l was easily implemented using stm32f4 microcontroller devices, and the interfacing between the nrf24l01 radio module and the micro was accomplished using the SPI serial interface. There are also numerous open-source libraries available that make interfacing to the nrf24l01radio module less cumbersome. Setting up of this module can be summarised in the following steps:

- Importing open source libraries to use for interfacing the stm32f4 microcontroller to the Nrf24l01 module.
- Setting up and initialising the GPIO port of the microcontroller
- Setting up and initialising the SPI interface of the microcontroller
- For debugging, also setting up and initialising UART.
- Defining a unique address yet standard radio pipe on Both transmitter and receiver of the NRF modules.
- Implementing the various function from the libraries imported to set up the device as either a transmitter or receiver.
- Set Channel ();
- Set Payload size ();
- Calling the appropriate functions where necessary.

4.7 LoRa Wan And LoRa PHY

It is imperative to understand the differences between LoRa Phy over LoRa Wan

LoRa WAN

In LoRaWAN, the end node communicates with a Gateway where the end node is registered using its device EUI, app EUI to differentiate itself from the end nodes and authenticate itself by the gateway and the app server via OOTA or APB methods.

With LoRa Phy, the LoRa end node communicates directly on a one-to-one basis with another end node without any authentication by either node. It is merely a LoRa payload transfer from one node to another using the LoRa radio module like RM95, sx1276, sx1278, etc.

4.7.1 LoRa PHY

The LoRa®-protocol link-layer stanadards defines the device-to-device Lora physical layer parameters [6], describing how the data is modulated into the transmitted signal. The protocol is proprietary, and not all information is freely available. Nevertheless, it is enough to know that due to the used Chirp Spread Spectrum (CSS), the power consumption does not depend on the content of the data. Therefore, the energy required for transmission only

depends on the so-called time-on-air (ToA) and the output power. Time on air refers to the time it takes a signal to travel from the transmitter to the receiver.
LoRa uses orthogonal spreading factors. This allows the network to preserve the battery life of connected end nodes by making adaptive optimizations of an individual end node's power levels and data rates. This enhances LoRa's footprint in the LPWAN technologies when compared to its counterparts.

An end device close to a gateway should transmit data at a low spreading factor since a minimal link budget is needed. However, an end device located several miles from a gateway will need to transmit with a much higher spreading factor. This relates to LoRa's adaptive rate feature. LoRa is a long-range radio at relatively low bit rates. LoRa is also very low bandwidth and consumes very little power, making it ideal for battery-powered operations.

LoRa operates in the license-free band. Either 433, 868, or 915 MHz   LoRa has a huge link budget.
LoRa PHY is the Best-effort transmission and, analogous to UDP, doesn't care if it gets there. LoRa Wan Is a layer on top of LORA PHY knows that data gets there like TCP. LoRa wan also add encryption, whereas LoRa node/ LoRa Phy has no encryption.
The payload of the standardized packet is fixed to 10 bytes, being close to the proposed 12 bytes maximum of TTN. For a package sent via the LPWAN, the payload increases by at least 13 bytes to a total length of 23 bytes.

4.7.2 LoRa Packet Format

The basic LoRa packet comprises three elements, namely the preamble, an optional header, and payload. There are essentially two types of LoRa packets transmission: explicit header mode and implicit header mode.

The functional difference between these two modes is the presence or absence of the header, which contains information regarding the payload length, the coding rate, and the presence of 16 bits' cyclic redundancy check. In the implicit mode where the header is absent, the header information must be manually configured on both sides of the radio link.
The robustness of the physical layer can be attributed to one significant parameter of the LoRa Physical Layer, i.e. Forward error correction.

Forward error correction is where redundant bits are added to transmitted data. These redundant bits help restore the data when it gets corrupted by interference. The higher the number of error correction bits added, the easier it is to correct the data. However, the payload inherently increases, increasing power consumption in the LPWan transmission, adversely affecting battery life.

The coding rate (CR) parameter (ranges from 1-4) defines the additional bits added to each 4-bit package that together form a code-word.

The coding rate refers to the proportion of the transmitted bits that carries information. The Coding Rate affects the ToA negatively but makes communication more robust.

This research will not implement LoRa Wan but LoRa PHY. The LoRa radio module used to implement this design is an SX1276 LoRa radio. This research did not focus on the LoRa Wan technology per se but rather the implementation of the LoRa physical layer using a LoRa radio module interfaced with an stm32f4 32-bit microcontroller.

The device translates an industrial process control signal in the form of a 4-20mAsignal. It converts this signal in the form of data using the LoRa physical layer to another LoRa transceiver module that will receive this data. The data received cannot be consumed by any system in its present state. The data will need to be translated into a format sent/consumed to/by a SCADA server.

This research was not concerned about transmitting the acquired data to a remote site or passing the data to a gateway but using the LoRa PHY as wireless communication to get data from one end of the plant to another approximately anywhere from 200m to another 800m. The primary objective lies in using wireless technology instead of using labour-intensive and expensive infrastructure like fibre optic to get data into a SCADA system.

The physical layer/radio interface provides an excellent conduit for low power signal communications where a low data rate, low power, and long distant range are necessary.

The interfacing of the LoRa radio module to the microcontroller is easily interfaced to the microcontroller with the SPI interface. In the modern era, rapid development tools like the ST CUBEmax makes this interface quite simple to the seasoned low-level programmer. There are also many open source libraries available to adapt and use to interface the LoRa radio to the microcontroller. The LoRa node was also recently incorporated.

The typical LoRa wan architecture includes a LoRa wan gateway, LoRa network server, and LoRa App Server.

The node to node implementation, on the other hand, does not make use of a gateway, and data is transmitted directly to a LoRa node which is essentially a transceiver. There are many off-the-shelf implementations of the LoRa PHY, but this research implements using the SX1276 LoRa radio module, open-source libraries for the sx1278 module is also adaptable to use with sx1276.

When implementing the code for the microcontroller to set up and implement the LoRa node, careful consideration must be given to comply with the guidelines/rules of the LoRa alliance. The interface between the LoRa radio module and the microcontroller is SPI.

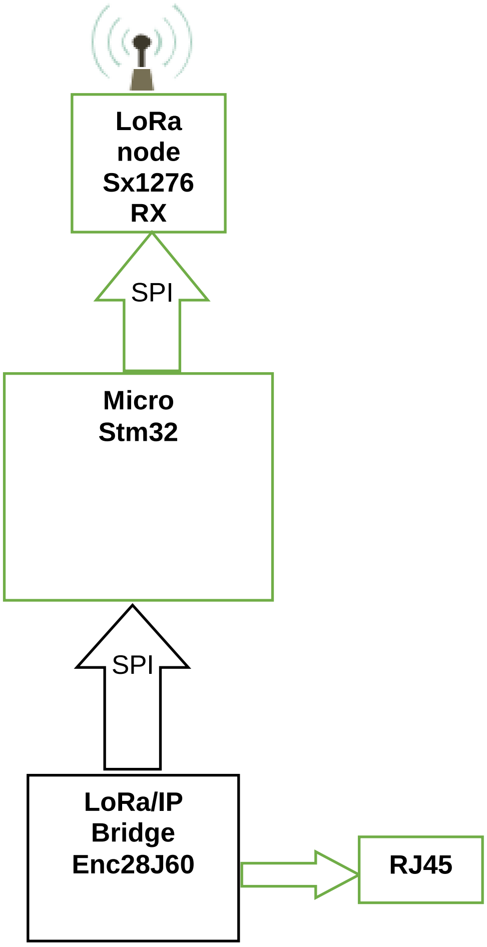


Figure 4.7: LoRa node receiver with IP/LoRa Bridge

4.8 IP Wrapper/Bridge

On the receiving end of the LoRa device, the data would then be parsed to the SCADA via a LoRa to IP wrapper. The LoRa to IP wrapper/bridge is merely a software bridge from LoRa

PHY format to IP, parsing the data in the required format for the SCADA device driver to handle. The serialisation of the data from LoRa to IP would be carried out by the microcontroller, which in this case is the STM 32 device interfaced with an ENC28J60 Ethernet module. The output of this Ethernet module is UDP.

The bridge is simply a high-performance device to connect two dissimilar or similar networks. In this research, our specific need is to connect/ translate the format of the LoRa message to IP format for parsing the acquired data to the SCADA system. Once the data is presented to the SCADA system with the familiar format that the SCADA device driver can recognise, it can be further de-serialised /translated by the SCADA device driver to be processed by the SCADA system. The device driver of the SCADA system (Ignition in this research) will connect to one or more ports on a given IP address and bring in any data as a value of a tag. For this research, we will parse the data as UDP data, and thus the device driver of the Ignition SCADA system will only listen/read the data from the LoRa device via the IP/LoRa bridge.

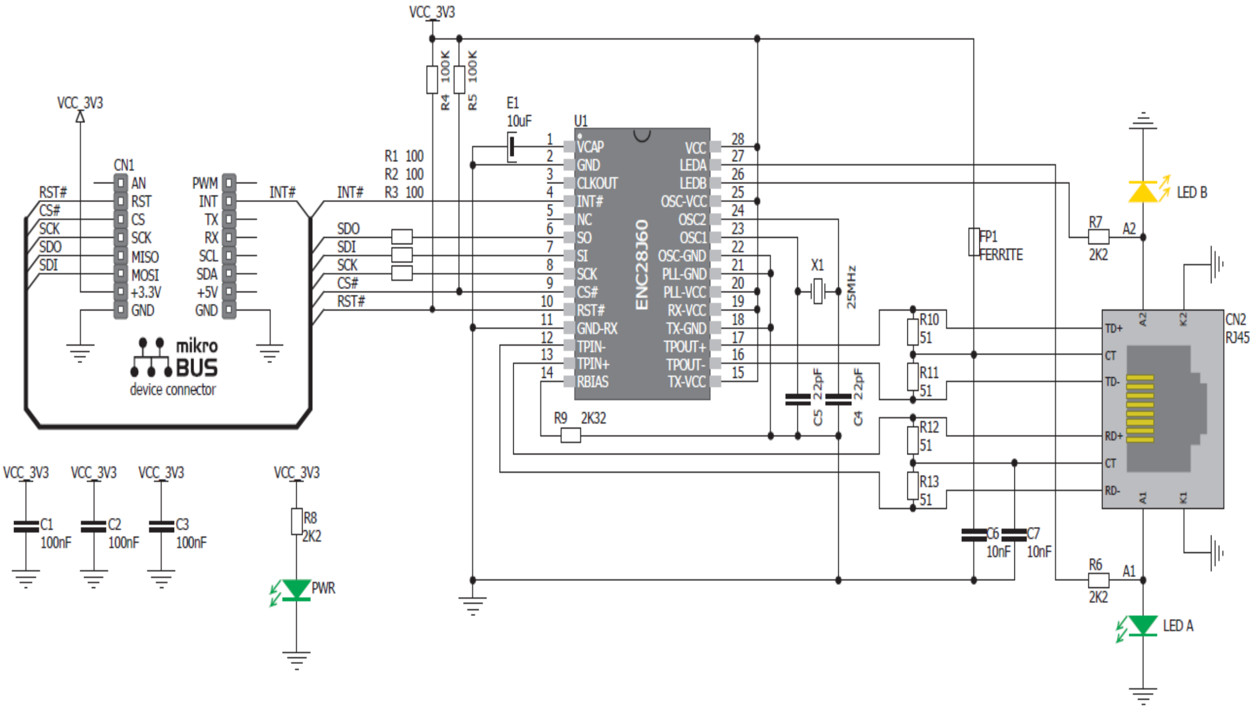


Figure 4.8: ENC28j60 interface used to connect LoRa Receiver to SCADA Server

Both LoRa and Ethernet have different frame formats and practice other MAC mechanisms and routing algorithms. Because we are using the LoRa node to node implementation (LoRa physical layer and not LORAWAN communication), there are no routing algorithms used as opposed to if we had to implement LORAWAN to IP bridge. The embedded code needs to consider CRC issues, arbitration, handling of errors, etc., and compliance with the LoRa specifications.

Special mention should be made to thoroughly understand the datasheet of the LoRa radio module and the SPI operation codes to implement the different configurable functionality.
The LoRa transceiver (receiver end) receives the LoRa format data from the transmitting LoRa device node. Both the LoRa radio modules SX1276 are interfaced to the microcontroller via SPI peripheral or interface. The low-level drivers/libraries move data either from the microcontroller buffer to the radio module in the transmitter or from the radio module to the microcontroller buffer in the receiver radio module. The data parsed to the microcontroller radio buffer is then encapsulated/serialised into an IP/UDP format by the LoRa to IP bridge.

The queued datagrams received from the LoRa receiver are presented as an IP/UDP frame, using the LoRa device as a unique network address and a dedicated SCADA port number. This, however, is not a direct translation from the LoRa data buffer to IP. Still, initially, the data gets serialised from the LoRa module's buffer of the LoRa module by the microcontroller and parses this serialised data to the Tx buffer of the ENC28j60 controller.

The enc28J60 controller is interfaced with the microcontroller via the SPI. When data is available in the buffer of the microcontroller initiates (via an interrupt routine) the transfer of this data via SPI commands to the TX buffer of the enc28j60 controller. The controller then adds the Mac and PHY components to the data to implement an IP data frame.

There are numerous low levels of functions written to carry out this implementation and can be summarised as follows.

- Initialisation the enc28j60.
- Assigning a Mac address to the controller
- Writing to the TX buffer
- Sending the packets

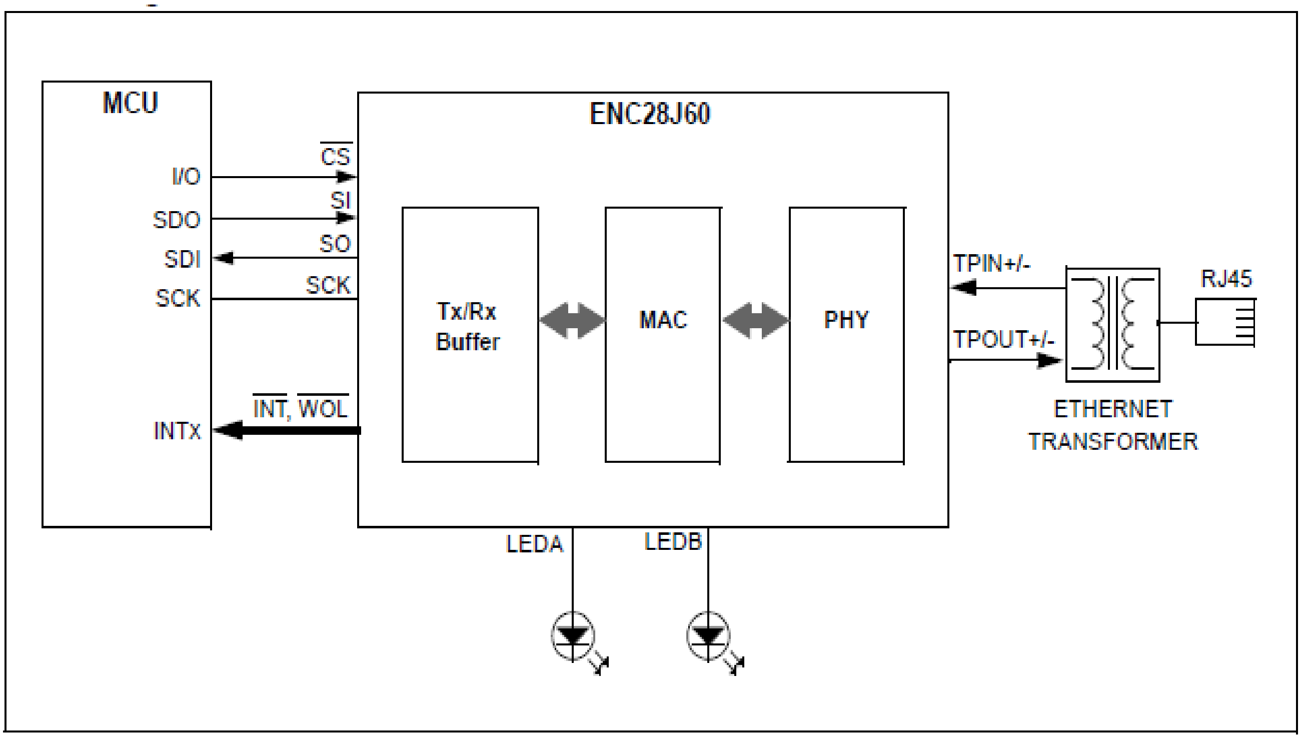


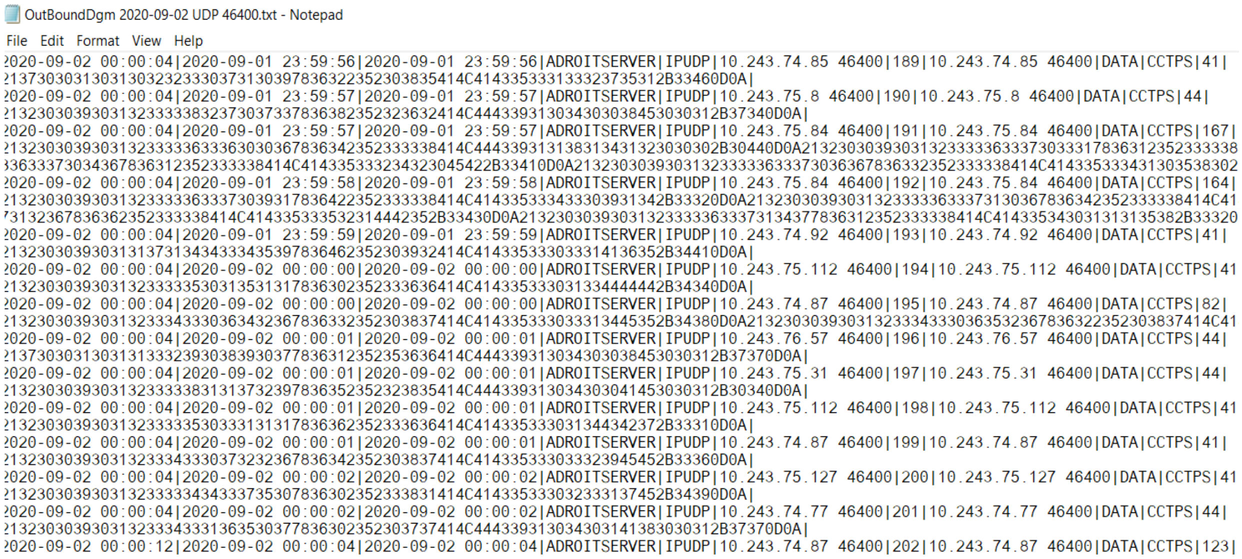
Figure 4.9: Device architecture of the ENC28J60

4.8.1 Testing and implementing The IP Bridge

Before implementing the IP wrapper, part of the designing process was to send some dummy data to the SCADA to understand how the SCADA would resolve the data and write serialisation routines for the IP/LoRa bridge to implement. This also indicated how to serialize the data at the LoRa node. To achieve this, we sent the data via the Sockets application program in the delimited format that the bridge would implement parsing the delimited data and a dedicated port number to configure the SCADA agent server to accept UDP data via this port number.

During the testing of the wrapper, we found a lot of packets with errors arriving at the SCADA system. There was a need to implement a packet manager or buffer to manage the packets arriving at the SCADA system. It was necessary to instate a packet manager and adapt to the LoRa/IP bridge to accomplish this. The packets leaving the bridge were configured to be destined for an assigned port number.

Before passing the UDP packets to the SCADA, it is necessary to have implemented a packet manager/buffer. The purpose of this packet manager/buffer is to parse only packets in a certain format to the SCADA while discarding any non-compliant packets. This packet manager takes care of processes that UDP doesn't handle. It also serves as a serialisation routine that formats the data in the format that SCADA drivers can handle/process.



```
2020-09-02 00:00:04|2020-09-01 23:59:56|2020-09-01 23:59:56|ADROITSERVER|IPUDP|10.243.74.85 46400|189|10.243.74.85 46400|DATA|CCTPS|41|
213730303130313032323330373130397836322352303835414C414335333133323735312B33460D0A|
2020-09-02 00:00:04|2020-09-01 23:59:57|2020-09-01 23:59:57|ADROITSERVER|IPUDP|10.243.75.8 46400|190|10.243.75.8 46400|DATA|CCTPS|44|
2132303039303132333338323730373337836382352323632414C444339313034303038453030312B37340D0A|
2020-09-02 00:00:04|2020-09-01 23:59:57|2020-09-01 23:59:57|ADROITSERVER|IPUDP|10.243.75.84 46400|191|10.243.75.84 46400|DATA|CCTPS|167|
21323030393031323333363333630303678363423523333338414C444339313138313431323030302B30440D0A2132303039303132333336333337303333178363123523333338
33633337303436783631235233333338414C414335333234323045422B33410D0A213230303930313233333633333730363678363323523333338414C4143353334313035383 02
2020-09-02 00:00:04|2020-09-01 23:59:58|2020-09-01 23:59:58|ADROITSERVER|IPUDP|10.243.75.84 46400|192|10.243.75.84 46400|DATA|CCTPS|164|
21323030393031323333363333373039317836422352333333 8414C41433533343330393134 2B33320D0A21323030393031323333363333373130367836342352333333 8414C41
73132367836362352333333 8414C414335333532314442352B33430D0A21323030393031323333363333373134377836312352333333 8414C414335343031313135382B33320
2020-09-02 00:00:04|2020-09-01 23:59:59|2020-09-01 23:59:59|ADROITSERVER|IPUDP|10.243.74.92 46400|193|10.243.74.92 46400|DATA|CCTPS|41|
213230303930313137313434333435397836462352303932414C41433533303333314136352B34410D0A|
2020-09-02 00:00:04|2020-09-02 00:00:00|2020-09-02 00:00:00|ADROITSERVER|IPUDP|10.243.75.112 46400|194|10.243.75.112 46400|DATA|CCTPS|41
21323030393031323333333530313531317836302352333636414C4143353330313334444442B34340D0A|
2020-09-02 00:00:04|2020-09-02 00:00:00|2020-09-02 00:00:00|ADROITSERVER|IPUDP|10.243.74.87 46400|195|10.243.74.87 46400|DATA|CCTPS|82|
21323030393031323333343330363432367836332352303837414C4143353330333331344535 2B34380D0A21323030393031323333343330363532367836322352303837414C41
2020-09-02 00:00:04|2020-09-02 00:00:01|2020-09-02 00:00:01|ADROITSERVER|IPUDP|10.243.76.57 46400|196|10.243.76.57 46400|DATA|CCTPS|44|
21373030313031313332393038393037783631235235363641 4C444339313034303038453030312B37370D0A|
2020-09-02 00:00:04|2020-09-02 00:00:01|2020-09-02 00:00:01|ADROITSERVER|IPUDP|10.243.75.31 46400|197|10.243.75.31 46400|DATA|CCTPS|44|
21323030393031323333333831313732397836352352323835414C444339313034303041453030312B30340D0A|
2020-09-02 00:00:04|2020-09-02 00:00:01|2020-09-02 00:00:01|ADROITSERVER|IPUDP|10.243.75.112 46400|198|10.243.75.112 46400|DATA|CCTPS|41
21323030393031323333333530333131317836362352333636414C414335333031344342372B33310D0A|
2020-09-02 00:00:04|2020-09-02 00:00:01|2020-09-02 00:00:01|ADROITSERVER|IPUDP|10.243.74.87 46400|199|10.243.74.87 46400|DATA|CCTPS|41|
21323030393031323333343330373232367836342352303837414C414335333033323945452B33360D0A|
2020-09-02 00:00:04|2020-09-02 00:00:02|2020-09-02 00:00:02|ADROITSERVER|IPUDP|10.243.75.127 46400|200|10.243.75.127 46400|DATA|CCTPS|41
21323030393031323333333434333735307836302352333831414C4143353330323331374 52B34390D0A|
2020-09-02 00:00:04|2020-09-02 00:00:02|2020-09-02 00:00:02|ADROITSERVER|IPUDP|10.243.74.77 46400|201|10.243.74.77 46400|DATA|CCTPS|44|
21323030393031323333343331363530377836302352303737414C444339313034303141383030312B37370D0A|
2020-09-02 00:00:12|2020-09-02 00:00:04|2020-09-02 00:00:04|ADROITSERVER|IPUDP|10.243.74.87 46400|202|10.243.74.87 46400|DATA|CCTPS|123|
```

Figure 4.10: Snapshot of outbound packets from packet manager to SCADA.

4.9 Comparing the NRF and the LoRa Node

On researching the wireless platforms to transmit process data from plants, two technologies were realised as most appropriate. They could provide a viable solution for getting data from standalone industrial instruments back into the process system or SCADA system for local visualisation of data or remote visualisation and /or input into MES or ERP systems.

There were other technologies researched to decide to implement an appropriate wireless solution to transmit acquired data from the field device to the SCADA system. Still, two wireless technologies that stood out in terms of transmission range, robustness, and bandwidth were nrf24L01 and LoRa node to node communication (LoRa PHY).

After researching and physically implementing the two technologies from an embedded perspective interfacing both the LoRa and the nrf24L01 modules to an stm24f4 development board, it was discovered:

The nrf24l01has far better data rates and bandwidth than the LoRa node but is significantly poorer in SNR and link budget and the distance it can cover for an end-to-end wireless data transmission. The nrf24l01 module is very inferior concerning range as it could only achieve a range of up to 150 meters without degradation of data.

LoRa, on the other hand, was very flexible in sending different data rates depending on the surrounding transmission conditions; it also has a built-in data rate adapter build into to LoRa physical layer. (This relates to the Spreading factor and forward error correction setting). LoRa also shows considerable superiority in terms of the SNR as compared to the NRf2401l radio. (Nordic, 2008)The NRF 24L0l module has superiority in data rates, but the NRF is significantly poorer in range when compared to the LoRa module. This significant superiority SNR robustness of LoRa can be attributed to the propriety signal transmission Chirp Spread Spectrum(CSS) that LoRa PHY uses.

Once received from the wireless interface, data needs to be converted by the IP bridge to parse data to the SCADA device driver/dll. LoRa node has significant bandwidth restriction but can be overcome or augmented by bit encoding techniques to optimize data transmission.(Calderón Godoy and Pérez, 2018) LoRa node/LoRa PHY can be likened to UDP transmission and LoRa Wan to TCP. This means some form of CRC needs to be implemented to ensure proper data integrity if this is a priority, especially when using just LoRa Phy.

4.10 The complete data value Chain

The SCADA system can also store this times series database(historian) which can then be stored in a SQL database used by a high-level system like an MES/ ERP system. Now that this data is part of the SCADA data architecture, the SCADA system can also parse this data to the OPC server. The OPC UA server makes this data available for remote ERP/MES clients. (Katti *et al.*, 2018)This data can be further converted to a lightweight protocol like MQTT in the publisher/subscriber module for a cloud-based web solution.

OPC UA is a non-agnostic architecture that is employed for seamless data interoperability solutions. The primary objective of an OPC UA server is to expose information that clients can use to manage underlying real-time processes. This OPC UA technology of exposing data to the high-level system is fast gaining ground.

Information or data collected describes the state and behaviour of the process, and the OPC server can transfer this data in both directions.

The two wireless options will need an IP bridge to get this data back into the SCADA system, which can also be passed to the OPC Ua server for further transportation. (Calderón Godoy and Pérez, 2018) At the source of the instrument, there would be a need to convert the 4-20mAsignal to serial data format.

The LoRa to IP bridge can also have more than one level of implementation to achieve data portability. The data acquired by the LoRa device or the NF2401 module needs to be parsed to the SCADA system. For seamless portability in line with 4IR specifications, it is essential to make this process seamless. Depending on the level of integration, either a low-level implementation using the ENC28J60 or the w5500 is another implementation of this convertor/wrapper. Many SCADA systems do have driver libraries to parse data from other vendors using their device drivers. In this research, the Ignition SCADA system integrated the data acquired from the LoRa node.

The principle of operation of this LoRa to IP bridge is pretty straightforward. The ENC280j60 controller has an industry-standard SPI interface that can be easily interfaced with a modern microcontroller. The TX/RX buffer is the data that needs to be parsed from the microcontroller; this is the data that needs to be transformed to an IP frame.

4.10.1 Data Value Chain from LORA Node TX To Ignition SCADA Server

The field device datagrams received by the device driver of the SCADA system are used by the SCADA engine can be visualised by the SCADA system. This translation of the data presented to the SCADA system is translated by the device driver of the SCADA system using the customs UDP device driver

Measurements recorded by the LORA NODE TX are partitioned into small elements – called datagrams.

- Each datagram is forwarded to the LORA NODE RX via a LoRa physical layer wireless link

- The LoRa Node Rx forwards the datagrams to a Packet Manager (Data switch) using the Packet Data format.
- A Data Switch receives the datagrams on the local Data Server network.
- The Data Switch pushes the datagrams to the Ignition Master server, located on the local Data Server network.

The data transfer process through the network is explained by discussing each link below.

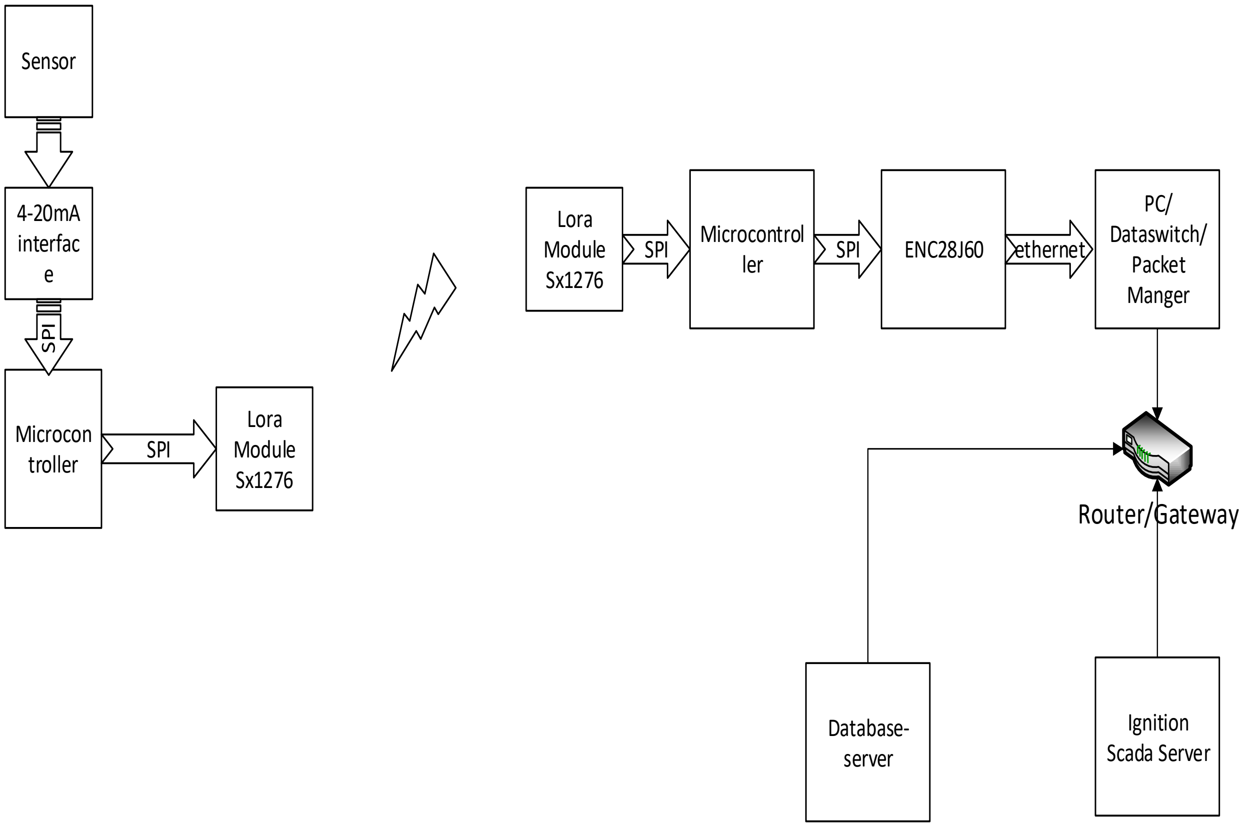


Figure 4.11: Complete Data Value Chain

4.10.1 4-20mAinterface and – LoRa Tx Link

Data transfer between the 4-20mAand the LoRa Tx is over an SPI of the microcontroller. A timeout is used to determine the start and end of each frame (datagram). The LoRa Tx buffer has a queue that can store up to four datagrams (to be forwarded to the LoRa Rx Node). There is no handshaking between the 4-20mAinterface's buffer and the LoRa Tx Node to control the flow of datagrams. If the interface transmits more datagrams than the LoRa Tx can Transmit, the latter will be silently discarded.

4.10.2 LORA NODE TX –LoRa Node RX wireless communication

The data between the LORA NODE TX and LoRa node RX is transmitted using the propriety CSS. Each of the respective nodes is configured to serve either as the LoRa Transmitter module or the LoRa receiver module. The LoRa modules are the interface to the microcontroller via SPI. There are many open-source libraries for the interface between specific microcontrollers and LoRa radio modules. The libraries used in this research with Stm32f4 micro and LoRa SX 1278 radio module.

4.10.3 LoRa Node Rx/Data Switch Link

The queued datagrams received from the LoRa Receiver Module are encapsulated in an IP/UDP frame, using the unique network address and a dedicated (SCADA) port number. The LoRa Node Rx has the interface enc28j60 controller responsible for wrapping the LoRa data format into UDP format. (Resources and Features, 2015)(Calderón Godoy and Pérez, 2018)The encapsulated datagram is forwarded to a Gateway (listener) on the Data Switch(Packet Manger). The Data Switch will strip the encapsulation and check the LORA NODE RX datagram for conformity –

- start of frame
- end of frame
- LORA NODE TX identifier

Any inconsistency will be treated as a malformed datagram and silently discarded. The IP addresses of the LoRa radio Module(receiver) on the SCADA Network is static. The Data Switch Gateway will maintain the relationship between the LoRa Radio Module(Transmitter)identifier and the wireless Network IP address.

4.10.4 Data Switch – Ignition Master Link

The data switch packet manager is an application running on Pc buffering the data acquired from the different LoRa modules. But in this research, the buffered is a straightforward buffer with transmission delay check. The Data Switch(packet manager) resident to the same network thus private IP range as the SCADA Server will identify each LoRa Rx device using the same address convention used by the LoRa device to identify itself. This identifier is, in

Switch Terminology, called the SID (Subscriber Identifier). The parsing of data to the Ignition SCADA system will be made using the UDP/TCP driver.

The TCP Client installed on the Ignition Master Server will transfer the datagrams received to the Ignition server for processing and handling. As the packet manager sends it, the datagram will be delivered to the Ignition Master Server. The packet transport mechanism will be completely transparent to the server. i.e., As if the packet manager had been directly connected to the server. The results of the analysis of the data will be accessible to Ignition slave stations (terminals).

The Ignition SCADA system can also receive MQTT data via the MQTT broker installed on the ignition SCADA in the form of the MQTT transmission module. So that any topics that the Ignitions server is configured to listen to will be translated/'heard.''

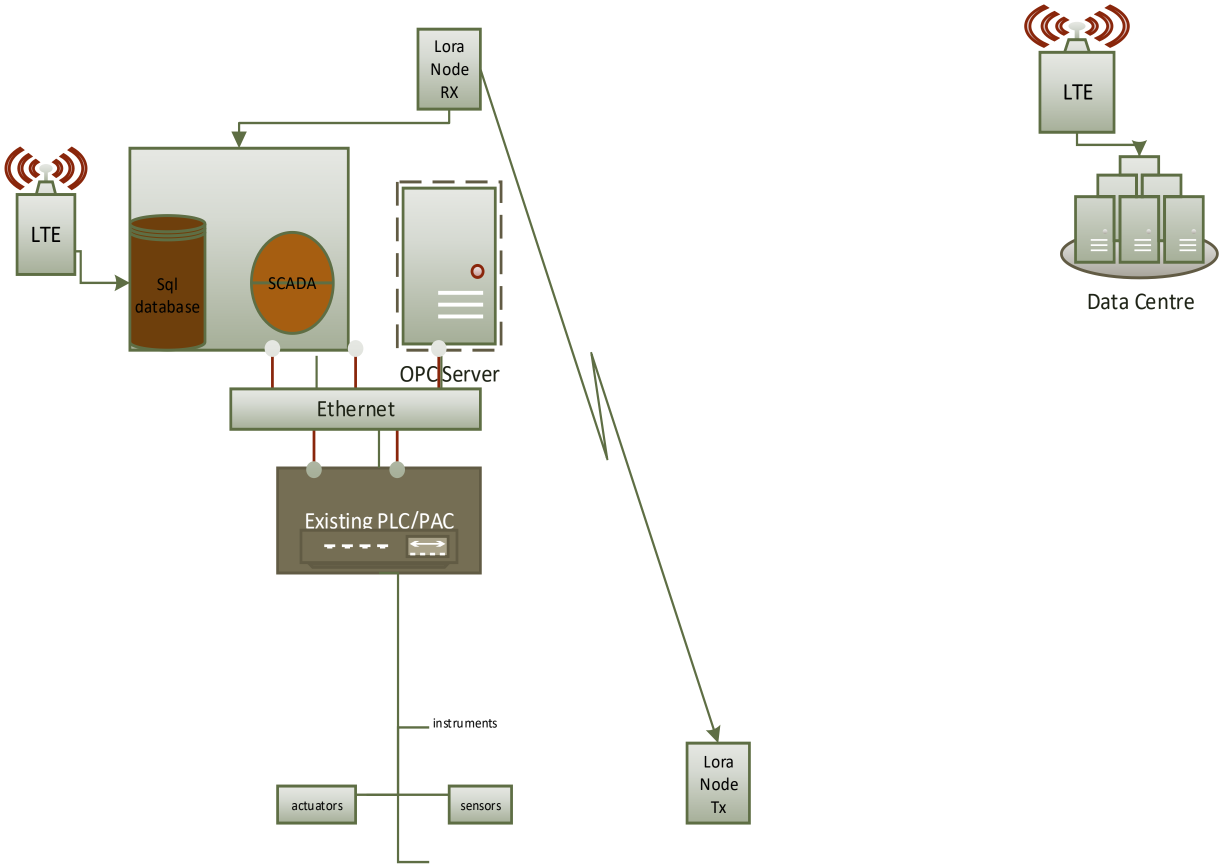


Figure 4.12: Architecture of a system with Added IIOT device using LoRa Phy

4.11 Conclusion

System integrators have many challenges to include new data points into existing industrial process costs with minimal costs. It can be seen from the above research that this is very plausible and achievable if data can be parsed in some IP frame format.

The challenges lie in managing this acquired data without compromising the security of the process control system as well as not compromising the deterministic nature of the protocols used in a process control system which inherently means that the packet manager mentioned in this research, as well as the rules of managing the transmission of the data both by the physical layer as the application layer, take care of this when implemented. The mixing of different systems and protocols to meet the system's specific requirements sometimes needs integration of both automation protocols with IT transport layer protocols are becoming a new norm in the IIOT world. Also, with the advent of MQTT, more options are fast becoming available, especially wrt parsing data to Higher-level applications.

CHAPTER 5

OPTIMISING DATA VISUALISATION IN THE PROCESS CONTROL AND IIOT ENVIRONMENTS

In the present context of the 4th Industrial revolution and Artificial Intelligence, there is a growing need to get data from different data sources in standard data format or Data structure for use in Manufacturing Execution Systems(MES), Enterprise Resource Planning(ERP), or Web API. There are also converging technologies from both the legacy process control systems and the emerging Internet of Things (IoT) systems. This chapter presents a solution to achieve this convergence in using IoT technology, extracting available data, and making it available to high-level systems in a standard, low latency framework. This research presents different protocols like OPC UA, Message Queuing Telemetry Transport (MQTT), and Constrained Application Protocol(COAP) to achieve this data transportation/acquisition.(L., B. and J., 2015) It also presents an emerging Low Power Wide Area Network (LPWAN) technology, LoRa WAN, to augment the data of the process control system, explicitly extending the range of sensors to wireless data points. This research presents a proof of concept to achieve this optimized data visualization.

5.1 INTRODUCTION

Digital transformation creates a need for protocols convergence to ensure cross-system data exchange (Bassi, 2017)and efficient machine–to–machine communication. OPC UA, MQTT, and COAP are some protocols that facilitate cross-system data exchange in the fast-changing data environment. These protocols functionally achieve common goals in data interoperability but have significant differences in their make in both overheads and data latency.

In the IIOT and the 4IR space, there is also a need to increase the data points without implementing expensive process control systems but rather to add to the existing process control system with additional sensors functioning in the IoT domain. If there were a need for an ideal protocol/architecture, it would have been a protocol like OPCUA had it not been for its heavy-weight nature. Protocols like MQQT. COAP and AOQP offer a lightweight solution

to facilitate this digital transformation but do not offer the frills of a protocol/architecture like OPC UA. (Pauker *et al.*, 2016)
Process data is now seeing itself integrated into high-level systems, and this data is also being made available to web interfaces where even the end-user/customer can view process/manufacturing information in real-time or, for the very least minimal latency. There is, of course, the concern of the security of data that is where architectures like OPC UA come to the rescue. But in the IoT scheme of things seems to be too heavyweight. There is also a need to provide this data at a minimal cost.

There are a few debilitating factors in the South African context when researching/testing the options of transferring data across communication networks, especially cellular networks that block traffic on non-standard ports. UDP is restricted and barred from going across cellular networks as a single user, even for research or academic purposes. However, we were not limited to UDP protocols when from a Lan perspective.

5.2 OVERVIEW OF PROTOCOLS

From an IIoT perspective, there is often the question of TCP or UDP.
TCP/IP specifies that the telegram must be acknowledged automatically after transmission. If the transmission is negative or there is no acknowledgement, TCP/IP repeats the message several times automatically. However, if the acknowledgement is impossible for technical reasons, TCP/IP consumes energy unnecessarily.

So what this spells out is that if the network is unreliable and power is a key driver, then TCP/IP is not a viable option as this could be too energy-intensive for a low-power WAN solution. (Silveira Rocha *et al.*, 2018)TCP in its pure implementation would not be suited for IoT or lower power applications in unreliable networks. In IIoT frameworks closely associated with TCP/IP, MQTT addresses some of the drawbacks of a pure TCP connection.

There are many emerging LPWAN technologies, and quite a relatively new radio-based IoT technology is NB-IoT. However, data was already transmitted via radio before NB-IoT. Another example is 802.15.4 with IPv6 over Low-Power Wireless Personal Area Networks(**6LoWPAN)**. 6LoWPAN is a transmission protocol on IP Point-to-Point Protocol(PPP) and was specified for radio and wire. The layer above 6LoWPAN is UDP; if a radio channel is disturbed, it generally makes no sense to start a new telegram transmission.

UDP does not send an acknowledgement and does neither expects an acknowledgement, making it most suitable for constraint low power WAN networks. If there is a need for acknowledgement and elect to use UDP, the acknowledgement must be done in the protocol layer above it. The protocol layer above UDP can be a protocol like COAP in the case of 6LoWPAN. CoAP regulates that a telegram is sent (with or without acknowledgement). This means that the programmer himself can decide whether he needs an acknowledgement or not. Most modern programming libraries have COAP libraries to facilitate the seamless use of this UDP-based framework.

One of the other objectives of this research was to demonstrate how to make low-level or field-level sensor information data available to High-level/application layer MES/ERP systems with non-propriety and interoperable protocols.(Calderón Godoy and Pérez, 2018) The challenge in the modern context is to marry IOT technology with higher-level systems to give customers or end-users the comfort of information, allowing maintenance and operation managers to optimise systems and processes with this added intrinsic data.(Veichtlbauer, Ortmayer and Heistracher, 2017)

In the IIOT world, one of the most significant challenges is seamlessly and effortlessly getting sensor data to a WEB API or MES. More and more processes are being automated in the modern industrial environment, thus providing new possible data sources to high-level systems. But not all of the data at these process control systems data need to be exposed to the MES/high-level system; there needs to be some sort of aggregation and packaging of the data in the format that these high-level systems require.

At the same time, these present and existing process systems do not always provide all the data that the MES system requires, and sometimes there is a need to get additional data from the same environment/system at minimal costs. In some cases, the data points are geographically dispersed, and devices need to provide for this data source. This research has delved into all the different data sources to transmit the data from these different data sources to a standard “device gateway”. This research looked at the various technologies and frameworks to get data to the device gateway. The purview of this research, as seen in figure 4.1, covers the following data sources:

A. LoRa WAN Gateway
B. Data logger using GSM/LTE.
C. SQL database with JSON engine using System on Chip LTE modem.
D. Or any Other data source.

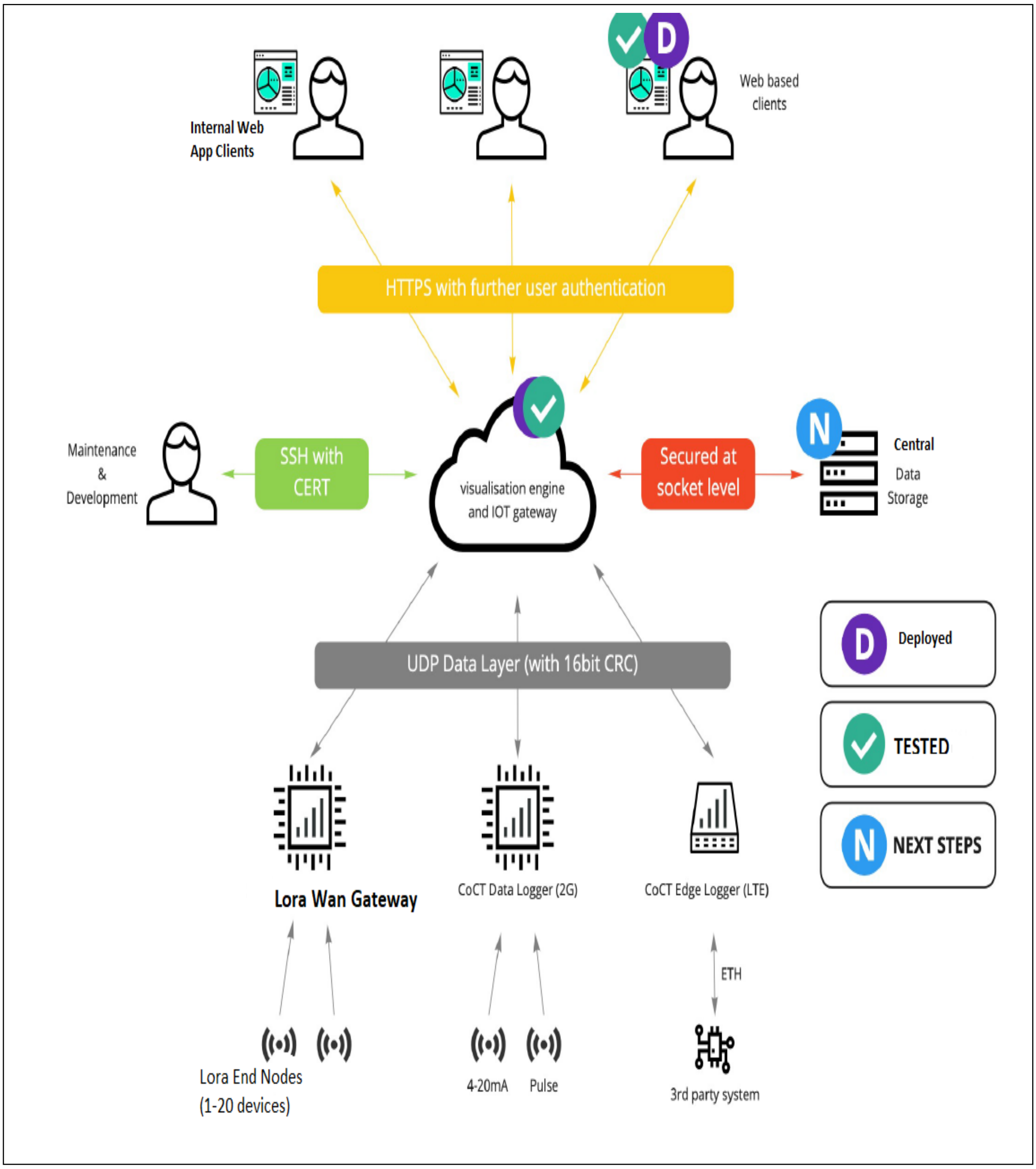


Figure 5.1 Architecture of New Framework

5.3 LORAWAN

LoRa WAN is the LPWAN technology that was used to get the additional data required by the MES that the existing Process Control System did not provide.

LPWAN, LoRa WAN, in particular, are gaining ground in the IIoT space and are finding themselves being implemented in a wide range of applications. But to get the full benefit of these technologies, it is apparent that there needs to be some low-level implementation of the technology and some system integration into MES systems.

Many developers and engineers chose the off-the-self and rapid development route, but those routes fail to deliver the full potential of the technology. One of the problems/debilitating factors encountered during the implementation of the LoRa WAN solution was the fact they're all the off-the-shelf products encountered for a LoRa end node had only one analogue sensor input coupled to the LoRa radio node.

It would be beneficial to have two or more analogue inputs interfaced with one LoRa radio module in an industrial plant. For this reason, this research implemented two analogue-type sensors with the capacity to interface more if required.

The true benefit of these IoT implementations lies in the integrative approach and the appreciation and implementation of the emerging protocols and applications to get true value for money. This research will present a solution encompassing this ideology: appreciating the hardware architectures, writing some low-level code, and adapting existing libraries to get an optimised solution.

Some of the present limiting factors with off-the-shelf solutions are:

i. most LoRa end nodes only allow one sensor signal per end device.
ii. Managing transmission intervals are limited with the shelf devices.
iii. On the LoRa gateway: Limitations on which UDP packet Forwarder can be used.
iv. Hardware Configurability limitations if no embedded development is done.

5.4 Present and Future architectures

There are numerous research researchs published with architectures presented for LoRa public and private networks. Still, few address the system integration of LoRa to MES systems purely because many LoRa implementations have their propriety application server coupled to it. (Bassi, 2017) LoRa is one of the fastest-growing LPWAN technologies that warranted some investigation into a technology option for getting industrial sensor data to a high-level system.

There are presently various Process automation plants controlling different wastewater water treatment plant processes. These processes are controlled by both PLCs and local SCADA systems. All this plant information is present locally at the plant in SCADA, PLC, and sometimes local SQL databases. All this data is used by the process systems as part of the control system and the visualisation of the control system when it comes to the SCADA system.

Leveraging this local data at a remote site or system can be valuable if this data could be aggregated and parsed as data to high-level systems or a web application or even for data analytics. The local SQL data can also be augmented by IoT devices/systems that can be added to the same central API/MES. For example, in the Wastewater plant environment, IoT devices could relay Air quality, Inflow, and Outflow data of the plant, which can be added /aggregated to existing process plant data. The aggregated data would provide intrinsic value to the customer/end-user in the form of performance dashboards. The IIoT component of this is where LoRa WAN seems a viable technology option.

This research implements a wastewater treatment plant dashboard using various IoT devices as well as the local SQL plant database to visualise Important KPIs in a web-based application.

Another challenge in accomplishing the above is to keep cost and labour at a minimum while providing additional data to the MES/web application.
In this research, data sources were investigated to present/parse data to the MES/web application, namely:

i. Existing data from process control systems and SCADA are stored in an SQL database. Here only relevant data needed by the MES will be extracted from this process control database and made available to be parsed to a High-level system like an MES or Web API.
ii. Additional IoT data using LPWAN technologies, specifically LoRa WAN, to enhance the data of these plants; to provide Overall Equipment Effectiveness (OEE) data and Key Performance Indicators (KPI) data. (Tao *et al.*, 2014)
iii. Additional data sources using 2g/3g data loggers parsing data in a prescribed format.

5.4.1 Plant data

PLC/PAC systems have variables and tags already defined locally in these process-controlled systems; plants with local SCADA systems also have these tags mapped to the SCADA systems.

Some Modern process plants may also have a SQL database storing this data, especially in Water and Waste Water Treatment Plants. But this data does not get processed further than the plant, and there is an opportunity to make this data available to MES /high-level API. This data that is local to the plant can be aggregated and sent to the remote application server. In this research, we demonstrate how some of the data from these plants can be a data source for MES/API to be translated to end-user/customer information to improve production KPI's and Customer perspective.

5.4.2 LoRa WAN IoT DEVICE Architecture

To provide additional data to the MES systems, it was necessary to implement a solution to offer the low-cost answer suitable to provide data to the MES without the need for complex hardware infrastructure to achieve this additional data. In Water or Wastewater treatment plants, these data points are sometimes more than 1200m from the nearest communication gateway or process control plant.

Finding low-cost wireless technology was the straightforward plan, but this technology needed to be adapted for the wastewater treatment environment. The wastewater environment is industrial, and as mentioned earlier, many LoRa-off-the-shelf devices offer only one analogue signal per end device. Added to this, the device does not conform to any rugged specifications like IP67 rating. The end device required to capture the data necessary for the wastewater environment needs to conform to these specifications to endure extraordinary environmental challenges experienced in WWTP's.

5.4.3. Cost-Benefit of LoRa WAN option:

Also, with as many as 20 additional standalone data points, GSM/LTE loggers can be a costly option. In South Africa, just the GSM data logger without sensors could retail around R15 000 toR20 000 each. To get this additional data to the MES would cost R400 000 per plant for this option with additional monthly data cost for each of the twenty devices. With LoRa on the other hand, it would require 20 LoRa nodes at R700 each plus one LoRa WAN Gateway of R5000 bringing the total cost of LoRa solution under R20 000. Concerning the monthly data cost, the LoRa solution will on have one device gateway logging data to a remote site putting the operational cost of GSM 20 times the cost of using the LoRa WAN option.

5.4.4. LORA Technology Briefly

LoRa is a spread spectrum modulation(Orange, 2016). One of the most significant challenges in acquiring data in the environment like a wastewater treatment plant or a water treatment plant is the distance of the plants from the central control room, typically where all the SCADA is housed. There are several distributed process control plants, all relaying data to the central control centre.

Adding data points to these distributed plants utilizing existing legacy process control systems can be very expensive, especially when there is no more capacity to adding these additional data points. Also, getting sensor data even when there is capacity can still be very expensive and labour-intensive because of the need to use fibre. In the modern era, wireless options seem to be the preferred choice. Fortunately, this is where options like LoRa can deliver on the needs of the system

This research presents a variation of the standard implementation of LoRa WAN. To get data to the Device gateway, it was necessary to implement a private LoRa WAN network with an MQQT bridge implemented on the LoRa WAN Gateway. The Typical LoRa WAN Implementation consists of multiple end nodes, LoRa gateway, LoRa Network server, LoRa application server. (Koon, 2020)The LoRa gateway either has the legacy Semtech packet forwarder, or another popular UDP packet server is the LorIoT packet forwarder. Most of these packet forwarders have an Application server coupled to them for Visualisation purposes. The application server could be local to the Gateway or a remote location.

This research implements the LoRa end node with multiple sensors using the B-L072Z-LRWAN1 development board. The development process concerning interfacing of the LoRa radio module was done in a two-phase approach. The 4-20 ma was interfaced via SPI by mikroe Click board, and the 0-5V sensor was interface using the onboard ADC peripheral of the development board. All coding for these interfaces was done using the mbed platform. There were fortunately many rapid development platform libraries that made this interfacing relatively seamless.

The first phase was to interface an STM32F4 discovery board to the two sensor variations mentioned above and the interface sx1276 LoRa radio module. During this 1st phase, the sensors and the LoRa radio module were connected with wire jumpers for proof of concept but were too cumbersome for actual testing.

In the 2nd phase, because we worked with radio technology, i.e. LoRa chirp spread spectrum, we wanted to limit the number of jumper connections and any factors that could negatively influence the results.

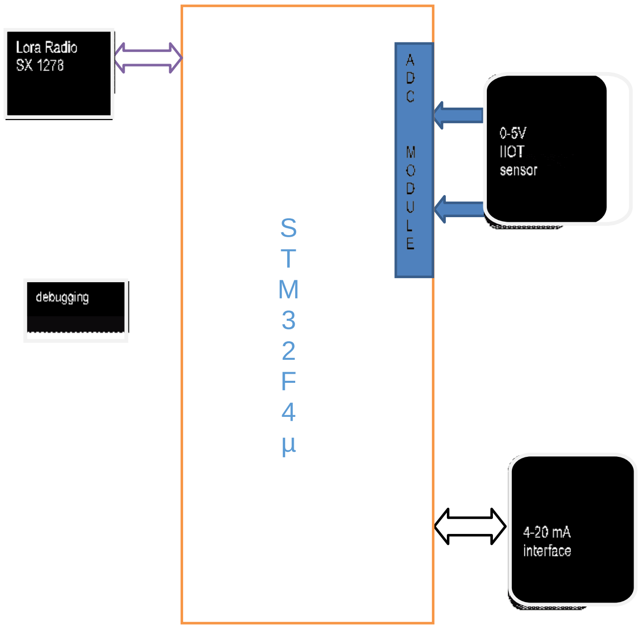


Figure 5.2 Hardware layout LoRa end Node Device

The above diagram Figure 2 illustrates the first phase of the hardware development for the LoRa End Node. The proof of concept and testing was carried out on an STM32F4 discovery board. The coupling to the 4-20mAinterface was implemented using the microcontrollers SPI to read analogue values.   (Semtech Corporation, 2013)The SX1276 LoRa radio module was also interfaced to the discovery board using SPI, whereas the 0-3.3V IOT sensor was interfaced directly to the microcontroller using its 12-bit resolution ADC interface.

All the coding and deploying in the 1st phase was done using the Keil µvision IDE and STM CUBEmax. This proved the concept before rolling out to the more expensive development board used in phase 2.

In the second phase, we used the B-L072Z-LRWAN1 development. This board has a built-in sx1276 LoRa radio module. Instead of interfacing the two sensor types directly to each development, as in the case of phase one, dummy sensor data was added to the variables defined in phase one to test the multiple end nodes more robust and less robust cumbersome. The focus of the testing was to test the capability of the communication interface as well as data rates and latency. The B-L072Z-LORWAN1 board is mbed-enabled, so all coding and deploying was done using the mbed rapid development platform.

5.5 Implementing the different Data sources

To register the respective end nodes on the LoRaWAN Gateway, each end node needs to be programmed with their unique dev EUI, an app EUI. Gateway(Robustel R3000)would parse data to the Gateways GSM modem in the same format. The difference between this option and the GSM data logger is that this option would allow for 20 different end nodes transmitting sensor data to one gateway, where the gateway would then be transmitted to the central device gateway. The benefit of this option is the reduced hardware cost, as well as there is no need for the external permanent power source to get these analogue values.

5.5.1 LoRa WAN gateway as a Data source

There are many implementations of this LPWAN technology which also include public or private network options. A Private LoRa WAN network is essentially a LoRa WAN Architecture where the LoRa WAN server and application server either is resident on the gateway itself or the Local network; in other words, the visualisation and authentication are done local to the network and is not done via a WAN.

The other differentiator separating these two variations is the LoRa WAN Server and LoRa application server implemented. The LoRa WAN server can further be made “more private” by deploying a LoRa UDP packet forwarder that is propriety and excludes the legacy option of the LoRa WAN server commonly referred to as the SEMTECH legacy packet forwarder.(Bäumker, Miguel Garcia and Woias, 2019)

The LoRa WAN gateway used for testing and proof of concept was the Robustel R3000 LoRa WAN Gateway. UDP packet forwarder installed on this gateway was LoRa Server.

LoRa packet forwarder is a program running on a LoRa gateway and interfaces to the LoRa concentrator to pull and push while interacting with the LoRa Network server, which could have a remote address or could be running locally on the gateway or the machine on the local network(Koon, 2020). The packet forwarder and LoRa WAN network server communicates with each other by a defined UDP packet forwarder. The legacy UDP packet forwarder has numerous shortcomings. It is not recommended for deployment on a large-scale network or an unreliable network, for that matter. Still, this does not apply because this implementation is local to a plant and not functioning in the public domain.

Additional to this application, an MQTT bridge was installed to the Robustel gateway to parse the data from the LoRa Gateway to the COCT Device Gateway.

In this research, LoRa WAN implementation is simply 5 LoRa end node devices all 600m to 1200m away from the Robustel gateway. The LoRa WAN Implementation can be seen in **Figure 5.3** below. Each of the devices for testing purposes had two analogue 4-20mAdummy sensors interface to it.

Usually, connecting up to 20 LoRa end nodes with multiple sensors transmitting to a LoRa WAN gateway is possible. The LoRa WAN gateway is configured to have a LoRa network server and application server local to the plant and using MQTT to parse this data to the device gateway.

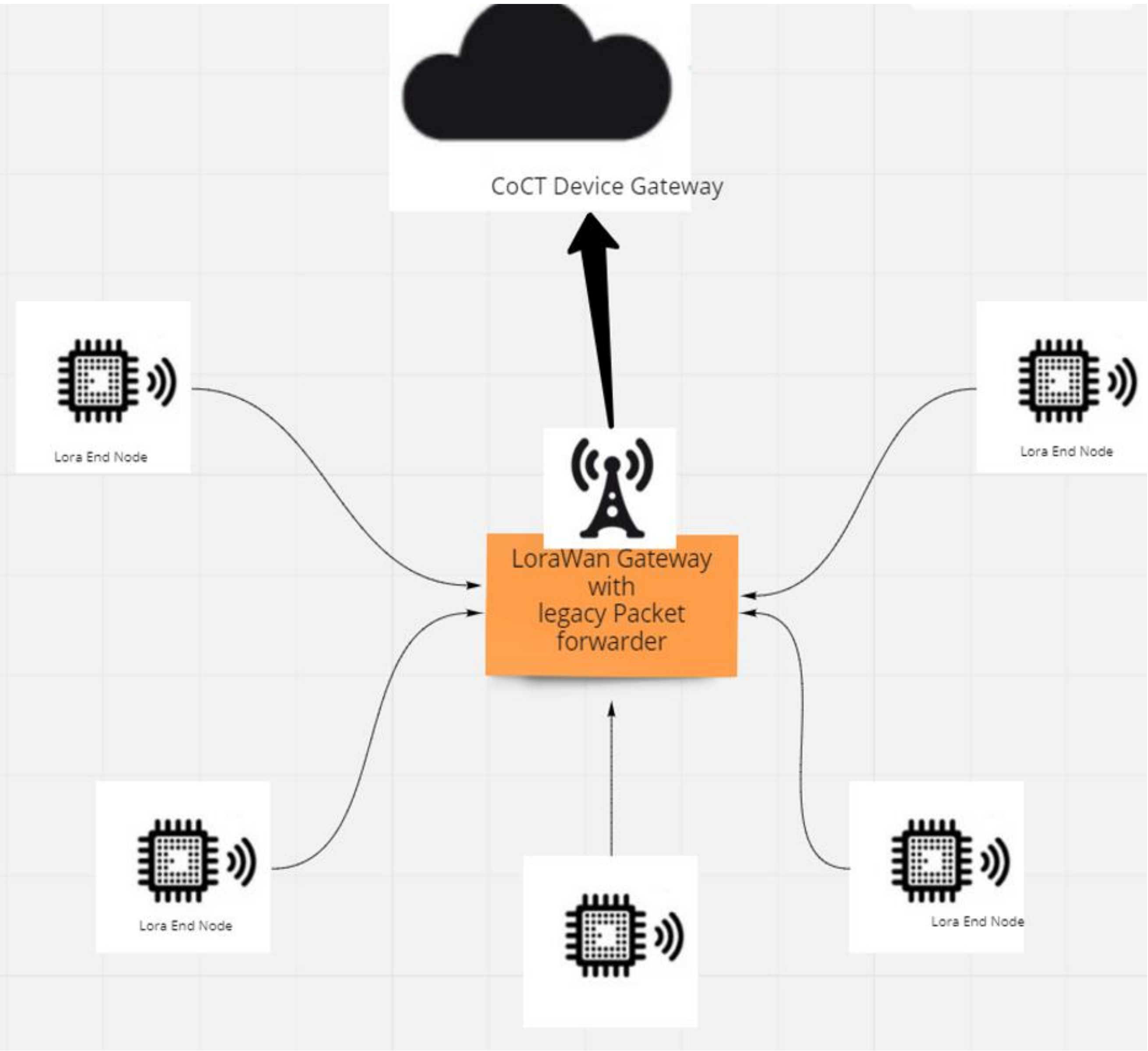


Figure 5.3: LoRa WAN as a Data Source to Device Gateway

To receive the data acquired by the LoRa devices via the LoRa gateway, the device EUI was used as the device ID for the central device gateway for the web service. In line with parsing data to a common gateway, we subscribed to the same format as shown in **table 2.**

5.5.2 LTE/GSM Logger as a data source.

One of the most common types of data loggers out there is GSM/LTE loggers. They are prevalent in remote locations. These loggers are a microcontroller interfaced to a couple of digital and analogue inputs and coupled to a GSM modem via Uart or some other Serial communication interface(SCI).

These are normally some propriety type device parsing data via TCP or UDP using a sort of IoT framework. So for this research and to add more context to this research, one of the data sources was the standard GSM/LTE data logger. We also wrote code very similar LoRa end node except the LTE logger was transmitting data directly to the device Gateway, whereas the LoRa end node was transmitting data first to LoRaWAN Gateway with a built-in UDP packet forwarder with other translations before it could transmit to the device gateway.

The GSM/LTE data logger could also be implemented in very remote sites with minimal I/O requirement; this is also the most cost-effective way of adding these data points from remote sites.

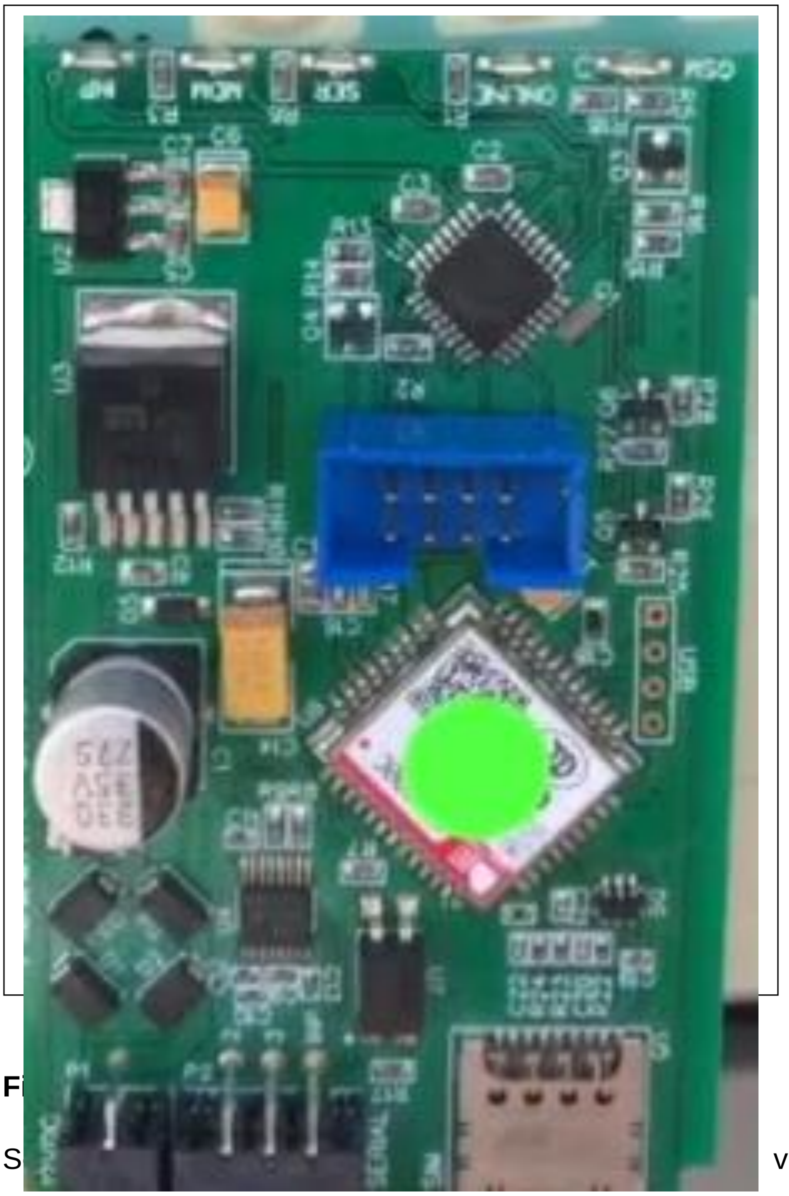
The GSM logger(Figure 5.4) could also interface directly into the existing process control system to expose some data “tags” that the remote MES system might require. The GSM logger used in this research that was developed for this research had two hardware variations, namely:

i. 2 x 4 to 20mA interfaced to GMS modem transmitting data to the remote site in a power-optimized way.
ii. It has a built-in Modbus wrapper to exposed Modbus registers from a plc in this Schneider m340 plc that the MES system requires.(Gutierrez-Guerrero and Holgado-Terriza, 2019)

The GSM logger forms two of the explicit device types that the device gateways receive data from. The data format from this data source also transmits in the prescribed COCT Gateway packet format, as seen in **Table 5.2.**

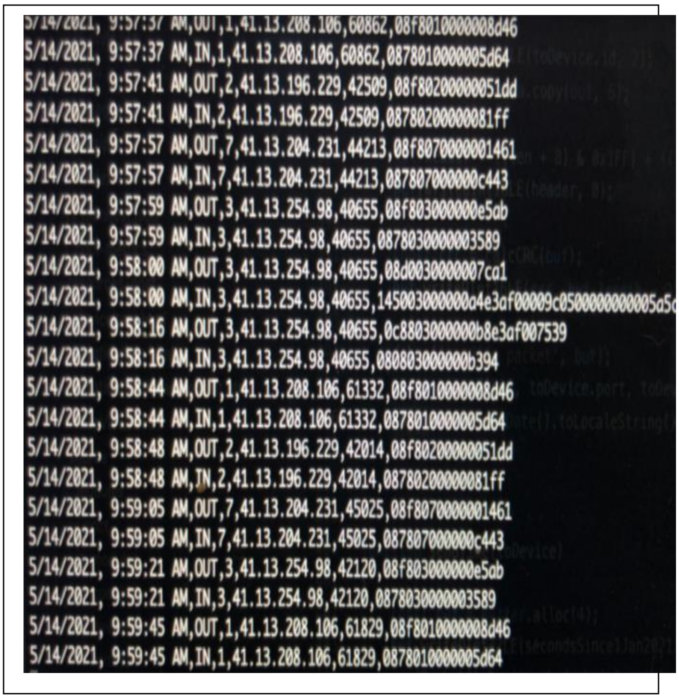
The firmware of this logger was written to parse data in the format as per CoCT packet format. This research also implemented two hardware variations of this data logger. One variation was where there was power available at the location, and the other was battery-powered.

The one that was battery operated, it was necessary to implement a circuit to boost the battery voltage to voltage of greater than 12V to power the current loop for the 4-20mAsensors(Azhari and Kaabi, 2001). The code written for the battery version of the hardware was also sensitive to optimize power consumption to allow the device to log data for long periods without changing batteries.



Fi

S values, dead banding was also incorporated in the code. In terms of transmission intervals, the design made it possible to update the transmission intervals of data logged, but the default was set to 15 minutes. Data was also transmitted if the change was more significant than a prescribed variation of the value last read.



```
5/14/2021, 9:57:37 AM,OUT,1,41.13.208.106,60862,08f8010000008d46
5/14/2021, 9:57:37 AM,IN,1,41.13.208.106,60862,0878010000005d64
5/14/2021, 9:57:41 AM,OUT,2,41.13.196.229,42509,08f80200000051dd
5/14/2021, 9:57:41 AM,IN,2,41.13.196.229,42509,08780200000081ff
5/14/2021, 9:57:57 AM,OUT,7,41.13.204.231,44213,08f8070000001461
5/14/2021, 9:57:57 AM,IN,7,41.13.204.231,44213,087807000000c443
5/14/2021, 9:57:59 AM,OUT,3,41.13.254.98,40655,08f803000000e5ab
5/14/2021, 9:57:59 AM,IN,3,41.13.254.98,40655,0878030000003589
5/14/2021, 9:58:00 AM,OUT,3,41.13.254.98,40655,08d0030000007ca1
5/14/2021, 9:58:00 AM,IN,3,41.13.254.98,40655,145003000000a4e3af00009c0500000000005a5
5/14/2021, 9:58:16 AM,OUT,3,41.13.254.98,40655,0c8803000000b8e3af007539
5/14/2021, 9:58:16 AM,IN,3,41.13.254.98,40655,080803000000b394
5/14/2021, 9:58:44 AM,OUT,1,41.13.208.106,61332,08f8010000008d46
5/14/2021, 9:58:44 AM,IN,1,41.13.208.106,61332,0878010000005d64
5/14/2021, 9:58:48 AM,OUT,2,41.13.196.229,42014,08f80200000051dd
5/14/2021, 9:58:48 AM,IN,2,41.13.196.229,42014,08780200000081ff
5/14/2021, 9:59:05 AM,OUT,7,41.13.204.231,45025,08f8070000001461
5/14/2021, 9:59:05 AM,IN,7,41.13.204.231,45025,087807000000c443
5/14/2021, 9:59:21 AM,OUT,3,41.13.254.98,42120,08f803000000e5ab
5/14/2021, 9:59:21 AM,IN,3,41.13.254.98,42120,0878030000003589
5/14/2021, 9:59:45 AM,OUT,1,41.13.208.106,61829,08f8010000008d46
5/14/2021, 9:59:45 AM,IN,1,41.13.208.106,61829,0878010000005d64
```

Figure 5.5: Real-time capture of packets arriving at device gateway.

Figure 5.5 is the capture of the real-time raw data being received by the device gateway before it can be processed by Web API.

5.5.3 LOCAL Plant SQL Database as Data source.

In a typical process plant environment where the architecture consists of many process control systems made up of PLCs, SCADA, and HMI’s with a local SQL database, the database stores all the pertinent data points of all the process control systems. These data points/tags are an essential source of data for the MES systems.

This is also another data source that the device gateway can receive data from. This data is aggregated data that the C# application makes available to an LTE modem from the transmission to the device gateway in the prescribed CoCT Gateway packet format. This aggregation will take place locally to the database server and will do all the heavy lifting in data aggregations and queries. This aggregation will be achieved by an engine that essentially will perform all the database queries and aggregation.

The JSON Engine is merely a C# application that resides on the plant database server that is responsible for aggregating data and presenting it in a prescribed JSON format so that the device gateway can understand for Use by the Web server system. The C# application will connect to the Microsoft SQL server and extract data using SQL queries. The data will appear as virtualised hardware device, much like the GSM loggers.

Attached to this plant server is a physical hardware device accepting input data via Ethernet to submit/parse to the existing device gateway. Any updates and/or new data found will be timestamped and buffered for transmission immediately to the Device Gateway. Should connectivity be disrupted, the application will cache the timestamped data for transmission later.

The C# application will package the data according to the gateway packet format seen i**n Table 1.** One of the fields of the header of the gateway packet format is a type that represents the data source type; the other field data the C# application will serialize the data is device ID will represent as specific aggregated KPI value from 0 to 255. So all data coming from the SQL data source will have a device ID of 0 to 255, which is reserved by the device gateway.

5.5.4. *Getting Data using OPC UA and MQTT from embedded OPC UA Data Source*

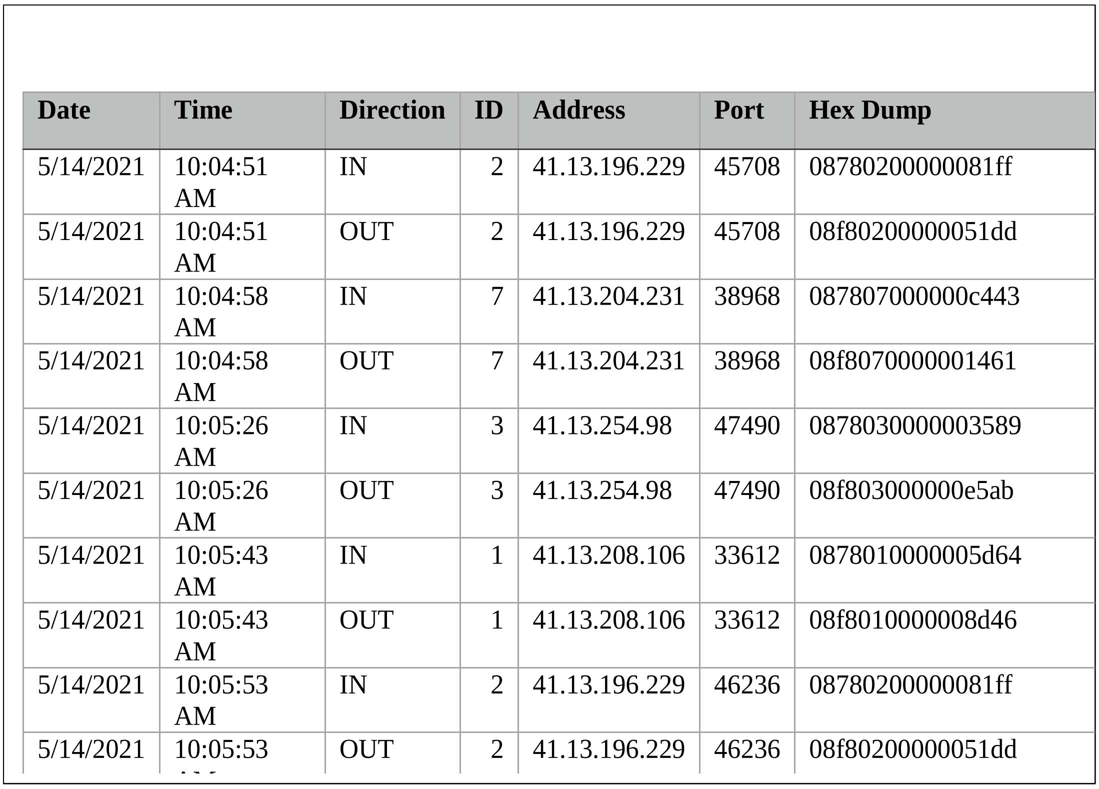
There are numerous ways of getting data from an OPC UA data source to an MES system. Still, for this research to demonstrate how OPC UA data is parsed to the Device Gateway for Visualisation To our WEB API, we will use a WAGO PLC with an EWON communication gateway.

The data accessed from the WAGO PLC, which has an embedded OPC UA server, make the Tags available to the EWON communication interface, which exposes these tags using MQTT(Profanter *et al.*, 2019). Not many PLCs presently has an embedded OPC UA server built into the process control system. Usually, any framework can be a client to this OPC UA data. Still, for this design, the data was parsed to the gateway using the EWON communication gateway where the EWON became the client, and the data was parsed to the CoCT gateway using MQTT in JSON format.

5.2 The Device Gateway

The purpose of the device gate is to have a gate way that could be a software communication interface to different data sources and data formats. The function of the device gateway is to process data from various data sources and make it available to the webserver/MES. This is achieved by having a data structure tied to the format of the UDP data frame. Figure 5.6 shows the database schema of the device gateway.

Table 5.1: Packet capture of Device Gateway



Date	Time	Direction	ID	Address	Port	Hex Dump
5/14/2021	10:04:51 AM	IN	2	41.13.196.229	45708	08780200000081ff
5/14/2021	10:04:51 AM	OUT	2	41.13.196.229	45708	08f80200000051dd
5/14/2021	10:04:58 AM	IN	7	41.13.204.231	38968	087807000000c443
5/14/2021	10:04:58 AM	OUT	7	41.13.204.231	38968	08f8070000001461
5/14/2021	10:05:26 AM	IN	3	41.13.254.98	47490	0878030000003589
5/14/2021	10:05:26 AM	OUT	3	41.13.254.98	47490	08f803000000e5ab
5/14/2021	10:05:43 AM	IN	1	41.13.208.106	33612	0878010000005d64
5/14/2021	10:05:43 AM	OUT	1	41.13.208.106	33612	08f8010000008d46
5/14/2021	10:05:53 AM	IN	2	41.13.196.229	46236	08780200000081ff
5/14/2021	10:05:53	OUT	2	41.13.196.229	46236	08f80200000051dd

At the gateway, there are two stages; the first stage is to de-serialize the data, i.e. function readPacketDetails(buf). The second phase is to interpret the payload.

As it can be seen from the Gateway Packet format demonstrated in **table 5.2,** we can differentiate from 16 different data sources based on the 4-bit type field of the data packet received. The device gateway will de-serialize according to the type field received in the UDP packet. So each type will have its own de-serializing routine/function.

The device ID is another differentiator which the gateway server can use to process data. Apart from using this field

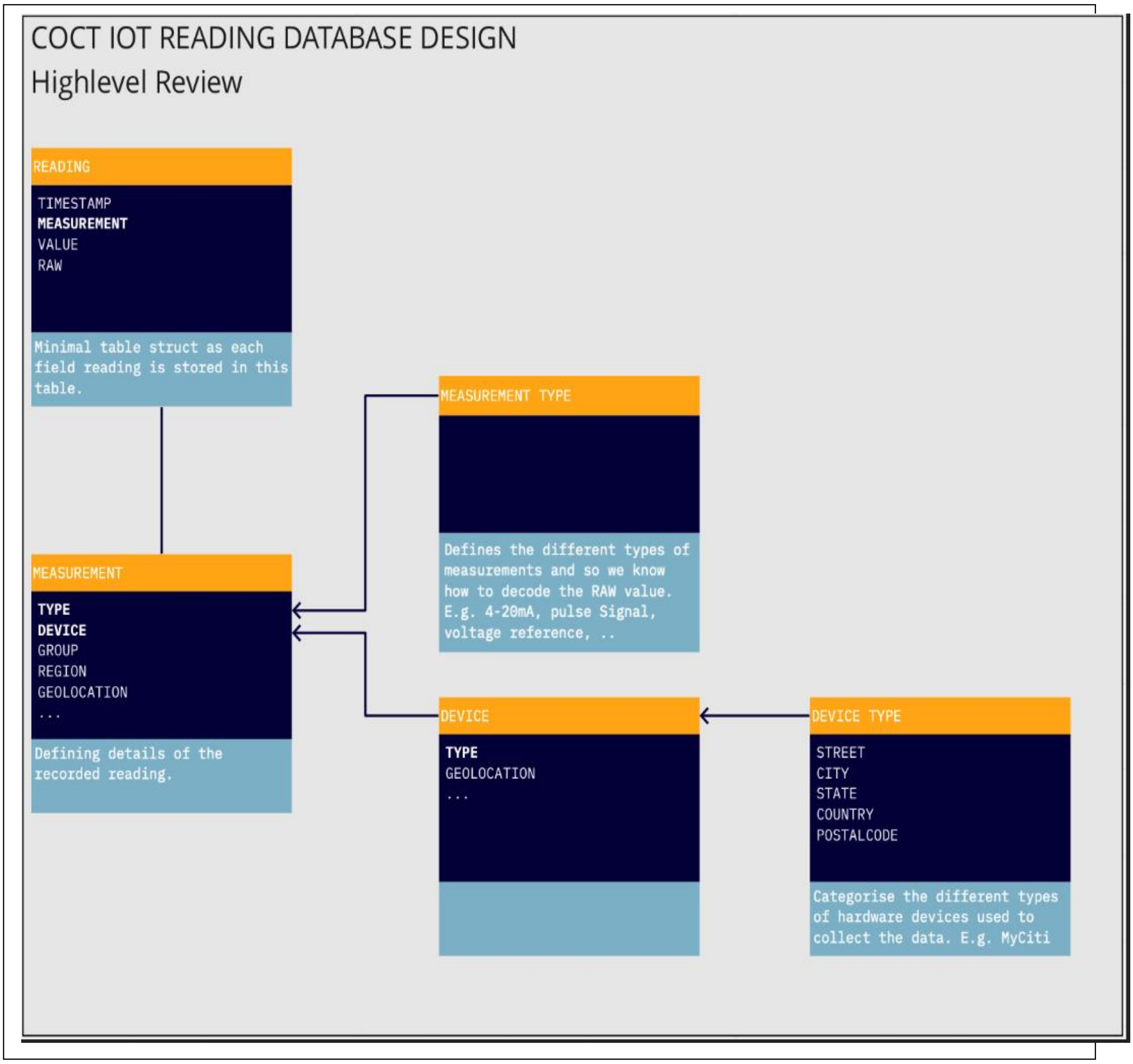

COCT IOT READING DATABASE DESIGN
Highlevel Review
READING
TIMESTAMP
MEASUREMENT
VALUE
RAW
Minimal table struct as each field reading is stored in this table.
MEASUREMENT TYPE
Defines the different types of measurements and so we know how to decode the RAW value. E.g. 4-20mA, pulse Signal, voltage reference, ..
MEASUREMENT
TYPE
DEVICE
GROUP
REGION
GEOLOCATION
...
Defining details of the recorded reading.
DEVICE
TYPE
GEOLOCATION
...
DEVICE TYPE
STREET
CITY
STATE
COUNTRY
POSTALCODE
Categorise the different types of hardware devices used to collect the data. E.g. MyCiti

Figure 5.6: Database mapping of data received from device Gateway

To assign a unique address to the device transmitting the data in the case of hardware type (gsm_logger_io module), we also used it to parse different KPI values from the SQL Data source to the device gateway.

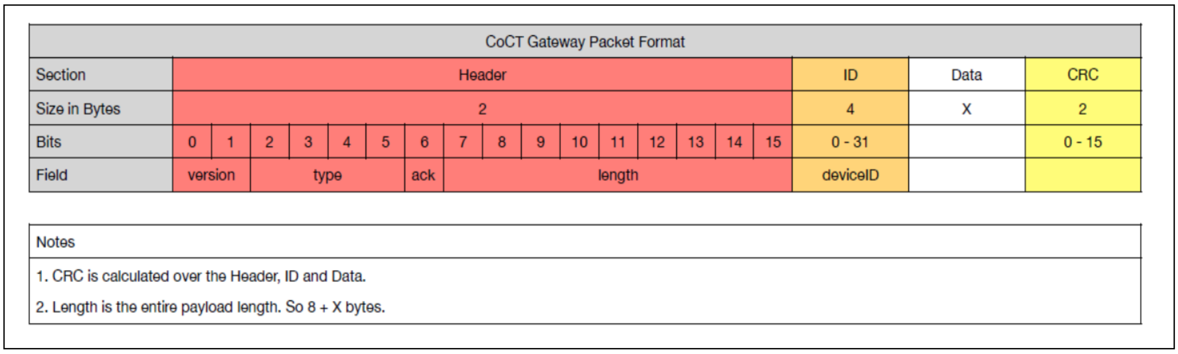
For data where the data source is a SQL data source and where the data is aggregated by data engine: what was essentially done was that we assigned a unique device ID to a specific KPI together with type field so that the device gateway recognizes the need to treat device ID as KPI data and write to the Database accordingly based on type field. A specific serialization routine is associated with the device_ID.

The device gateway will handle the device ID as KPI' populating the relevant table in the webserver database. Bit no 2 to 5 represents type (data source), So if the type bits represent SQL data source, then the device gateway knows that device ID will represent a specific KPI. For this proof of concept device ID, 0 to 255 were reserved for SQL data source.

For example: if Belt press availability is a KPI required. This data is sitting in the local plant database as belt press running status. The engine will aggregate this data by counting the number of belt presses running. The result is made available as key/value pair, where the key is the enumerated data type from 0 to 255 and value is the actual aggregated value representing that specific KPI. The key is then the device ID that the Device Gateway is expecting. The value is the payload(data that the gateway is expecting.)

On the other side, these enums are associated with specific KPIs on the device gateway.

Table 5.2: Gateway Packet Format



CoCT Gateway Packet Format

Section	Header																ID	Data	CRC
Size in Bytes	2																4	X	2
Bits	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0 - 31		0 - 15
Field	version		type				ack	length									deviceID		

Notes
1. CRC is calculated over the Header, ID and Data.
2. Length is the entire payload length. So 8 + X bytes.

Looking at **Table 5.2,** it can be seen that the field named type is what the device gateway uses to differentiate the different data sources and thus will de-serialize the payload data according to this field. The device gateway is a software engine that receives UDP packets, performs a few checks and balances(packet manager and de-serializes the data, and populates the necessary fields in the database.

5.6 Understanding, comparing implementing the protocols

The three protocols were discussed briefly because the parsing data to the remote system are essential to understand the complete data value chain. Solutions, especially in the Industrial Internet of Things(IIoT), require a mix of protocols but using the suitable protocol at the right layer/tier of the data value chain. Some of the critical driving decision-makers in choosing the appropriate protocol often come down to how lightweight the protocol is, the network's reliability, what the purpose of the layer is trying to achieve and the layer of security required, and controllability expected of that layer.(Veichtlbauer, Ortmayer and Heistracher, 2017) We discuss these very relevant protocols relevant in the IIoT space as well the OT environment.

5.6.1 OPC UA.

OPC Unified architecture is a connectivity framework that introduces a platform independence framework/architecture.
OPC UA is an object-oriented framework. OPC UA=IEC 62541. OPC UA is based on a client/server architecture that uses TCP/IP and HTTP/SOAP as the underlying frameworks(Cavalieri *et al.*, 2017). It is important to understand that OPC UA is an architecture and not a Protocol on its own. It uses different protocols to complete its framework/architecture. In layman’s terms, data is packed in a certain way, so the other side recognises this data-packed in this “sequence” identifies this as OPC UA and can de-serialise this packed data according to the framework/architecture rules.

One of the key objectives is to achieve device interoperability independent of propriety protocols and API of device manufacturers. OPC UA was initially intended or had its roots in the factory automation of Large manufacturing plants within a LAN environment(Drahos *et al.*, 2018). This protocol is a quiet heavyweight in its make-up and this is quite understandable due to its origins. The protocol has been implemented in various environments and is gaining considerable ground in its implementation in different environments. End devices will publish or render data to API’s standard architecture that API can unpack without any datacasting. The addition of a” browse-able” built-in information model includes executable services like read, write, method-call, subscribe, etc.

OPC UA essentially converts the hardware protocol of the device into a standardized device model. (Katti *et al.*, 2018) One of the modern evolutions of the OPC UA protocols is embedded OPC UA, which essentially means the OPC UA server is embedded in the SOC, and its tags can be exposed to any high-level client. PLC’s like WAGO are presently using embedded OPC UA servers to Expose tags to high-level systems.

5.7.2. MQTT

In the IoT space, simple and lightweight protocols are gaining ground, so if the requirement is to bypass complexity and move to a small footprint solution that guarantees a secure and reliable data exchange in industrial automation, you are bound to come across MQTT.

MQTT is messaging protocol, not a data communications protocol; it does not specify a particular format for the payload data. The data is determined by each client connecting; in the case of MQTT, the publisher and subscriber need to agree in the format in advance, or they will not have any clue what the payload means. MQTT is explicitly not interoperable but is used in industrial applications for lightweight data transfer.

MQTT is a lightweight protocol based on IP used by mainly IoT platforms. MQTT has PUB/SUB architecture(Architecture, 2018).

Connection is always initiated by the client to the broker using port 1883 or 8883 for secure/encrypted,

The broker is usually publically accessible via the internet and acts as a communication bridge or central node/point between the different clients. There are, of course, options to install the broker locally without public access. Regarding the content of the message, MQTT does not care.

It is up to the clients and broker to agree on the message format. The basic understanding of the MQTT architecture translates to the publisher publishes a topic, and whoever listens to that topic can see the message content.(L., B. and J., 2015). MQTT allows an unlimited number of clients/subscribers to listen to a published topic.

TABLE 5.3. Summary of comparison of protocols

Description	**OPC UA**	**MQTT**	**COAP**
Protocol binding	TCP	TCP	UDP
	Server/ client	Pub/sub	Server/ client
Security	excellent	Good	fair
Reliability over networks	Only good over very stable networks with excellent bandwidths	Good for transferring data/commands over unstable connections	Good for client/server connections. Preferred for stable networks
architecture	Client-server model	Broker is the centre of the network	Node is the centre of the network
Controllability of nodes	Offers secure controllability.	No controllability built into the protocol	Offers controllability of nodes within the protocol

Table 2 gives a snapshot comparison of the three protocols that relate to their applicability in different environments concerning data transfer. (L., B. and J., 2015)

5.7.3. COAP

Constrained Application protocol as defined by the RFC7252 standard is an open IoT standard. The protocol is completely asynchronous, which means it is essentially not connection orientated, making it very efficient for web applications.

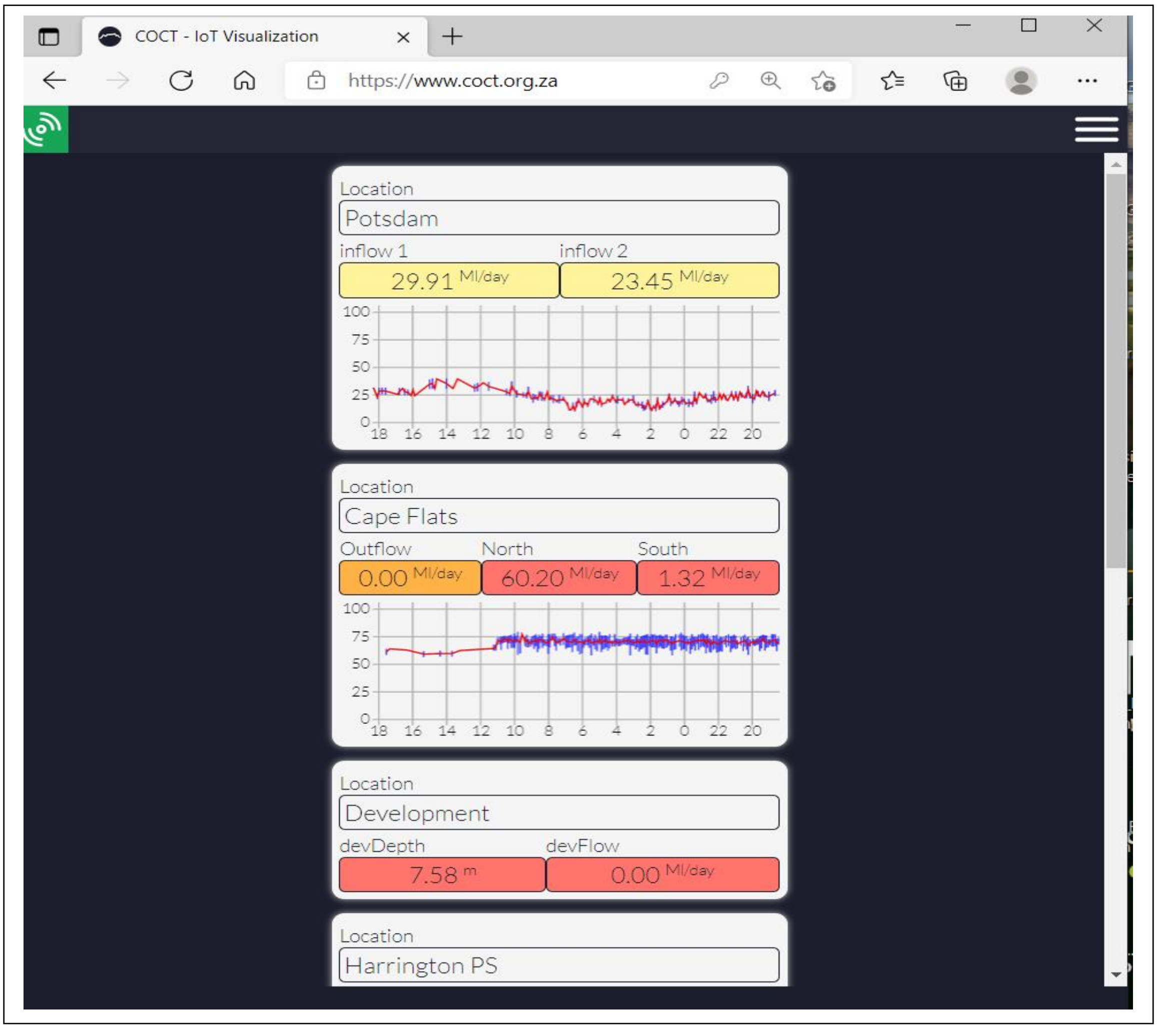
COAP was designed to support the RESTFUL protocol, synonymous with GET, PUT, POST, and DELETE verbs(L., B. and J., 2015).

COAP is typically associated with ports 5683 and 5684. COAP implements UDP binding and supports Uniform Resource Identifier(URI). COAP also 4byte binary header.

5.7 RESULTS

So in principle, the device gateway was reading data from 4 different data sources, namely:

i. Stand-alone LTE data logger
ii. LoRa WAN data logger.
iii. Local Plant SQL data engine.
iv. WAGO PLC using EWON Communication Gateway



Location: Potsdam
inflow 1: 29.91 Ml/day
inflow 2: 23.45 Ml/day

Location: Cape Flats
Outflow: 0.00 Ml/day
North: 60.20 Ml/day
South: 1.32 Ml/day

Location: Development
devDepth: 7.58 m
devFlow: 0.00 Ml/day

Location: Harrington PS

Figure 5.7 : Web API Visualisation

The above Figure 5.7 is a snapshot of a web page rendered showing visualisation of devices from 2 wastewater treatment plants, namely Potsdam and Cape Flats waste Water treatment plants

Figure 5.6 shows how the type is one of the differentiators in the database schema. The LTE logger, LoRa WAN, and SQL data engines represent the device gateway's different types as a differentiator. (Kovar *et al.*, 2016)The MES or web server has no idea what the specific data source is. All it queries is get method () based on message type that gets rendered as de-serialized payload. All the data being parsed to the device gateway is UDP.

5.8 CONCLUSIONS

We successfully parsed all these data sources to a standard device gateway, negating the need for expensive visualization software like a Modern Propriety SCADA system. It also negates the need for expensive propriety end devices in the form of off-the-shelf data loggers that are very expensive.

In the IoT space, where microcontrollers are cheaper by the day, and libraries for implementing these decodes are also freely available from different open-source platforms, solutions of this nature are easily implemented by engineering personnel with minimum coding/software development experience.

The other realisation in this research is how easily we can integrate technologies to Web APIs and MES systems for providing data for end-users and data analytics using JSON and other frameworks.

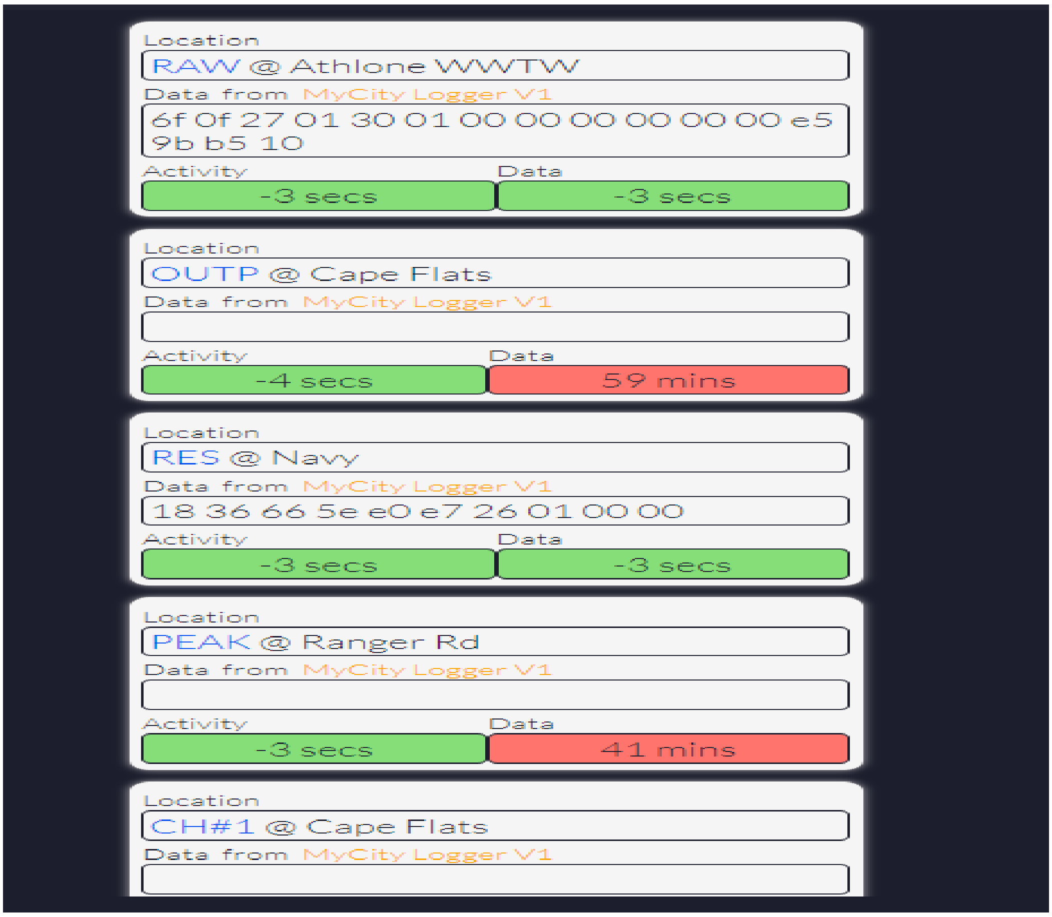
One of the other pertinent realizations of this research is how close the OT and IT worlds have converged in data exchange. With protocols like OPC UA, MQTT, and COAP being used extensively and interchangeably, who knows? There just might be an open-source protocol combining these protocols to let us have one that has all the best traits of each of these protocols.

CHAPTER 6

TESTBED AND RESULTS

This chapter describes the results and findings after successful completion/building of the asset tracking device/data sources. The design was implemented on a proof of concept basis, with each module hard-wired and physically deployed to various Cape Town locations as seen in the map below (Figure 6.2). The devices depicted below with metrics include the remote type devices like the GSM loggers and LoRa WAN end Nodes, all of which transmit data to the Device Gateway(see Figure 6.1).

The Asset tracking device, once initialized, connects to the device Gateway, which for the test was a multi-socket sockets application installed on a server. For the proof of concept, all that was tested for was the successful reception of the serialised data from ATD's as UDP packets, the webserver further visualised this data. Figure 6.1



Location
RAW @ Athlone WWTW
Data from MyCity Logger V1
6f 0f 27 01 30 01 00 00 00 00 00 00 e5 9b b5 10
Activity -3 secs
Data -3 secs

Location
OUTP @ Cape Flats
Data from MyCity Logger V1
Activity -4 secs
Data 59 mins

Location
RES @ Navy
Data from MyCity Logger V1
18 36 66 5e e0 e7 26 01 00 00
Activity -3 secs
Data -3 secs

Location
PEAK @ Ranger Rd
Data from MyCity Logger V1
Activity -3 secs
Data 41 mins

Location
CH#1 @ Cape Flats
Data from MyCity Logger V1

Figure 6.1: Metrics of devices using WEB API.

The specifics that were tested were \;

- Successful serialisation of the data into UDP packets.
- The different data transmission modes based on the different acquired states were simulated via pot connected to the ADC Module.

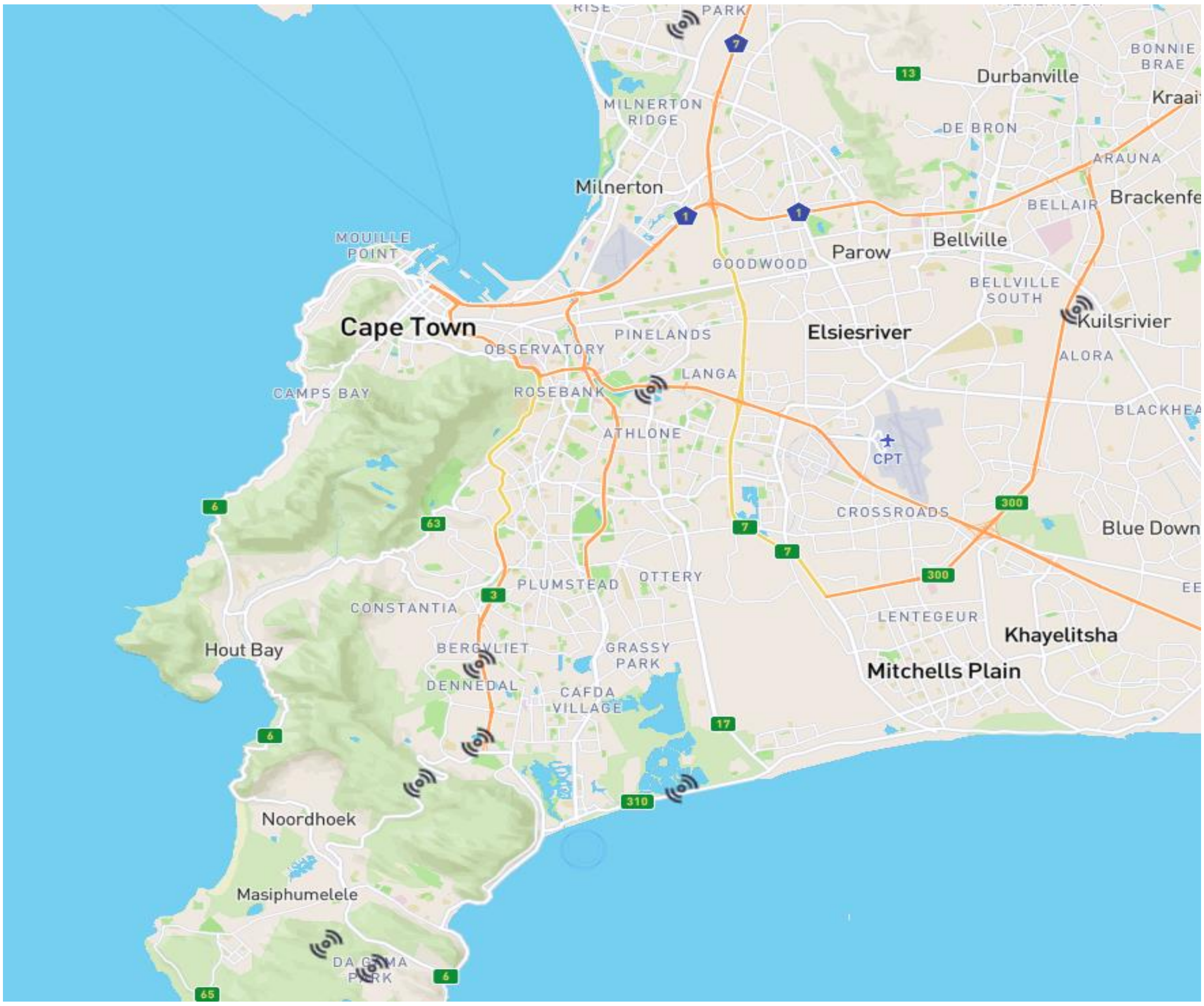


Figure 6.2 Locations of various devices deployed around Cape town.

6.1 LoRa Wan using Robustel R3000 Gateway

Because of the limitations of the LoRa physical layer in terms of interoperability and range, it was necessary to extend the research and implementation to the LoRa Wan. For this, we procured Robustel R3000 LoRaWan Gateway and three Stm32 devices with a built-in LoRa radio module to test the range and data capability of the LoRa wan architecture use in a wastewater treatment plant environment. The environment of the wastewater treatment plant, even though it is situated within a city metropole area, has fewer propagation obstacles than if deployed in the metropole area. Line of sight with obstacles is easy to achieve of the sparse distribution of buildings and obstacles with the confines of the Wastewater treatment plant.

This exercise was conducted to test the technical feasibility of installing a LoRa WAN solution.

The test also needed to consider that these LoRa end nodes will be installed below ground in manholes where some industrial-type sensors are located.

Purpose

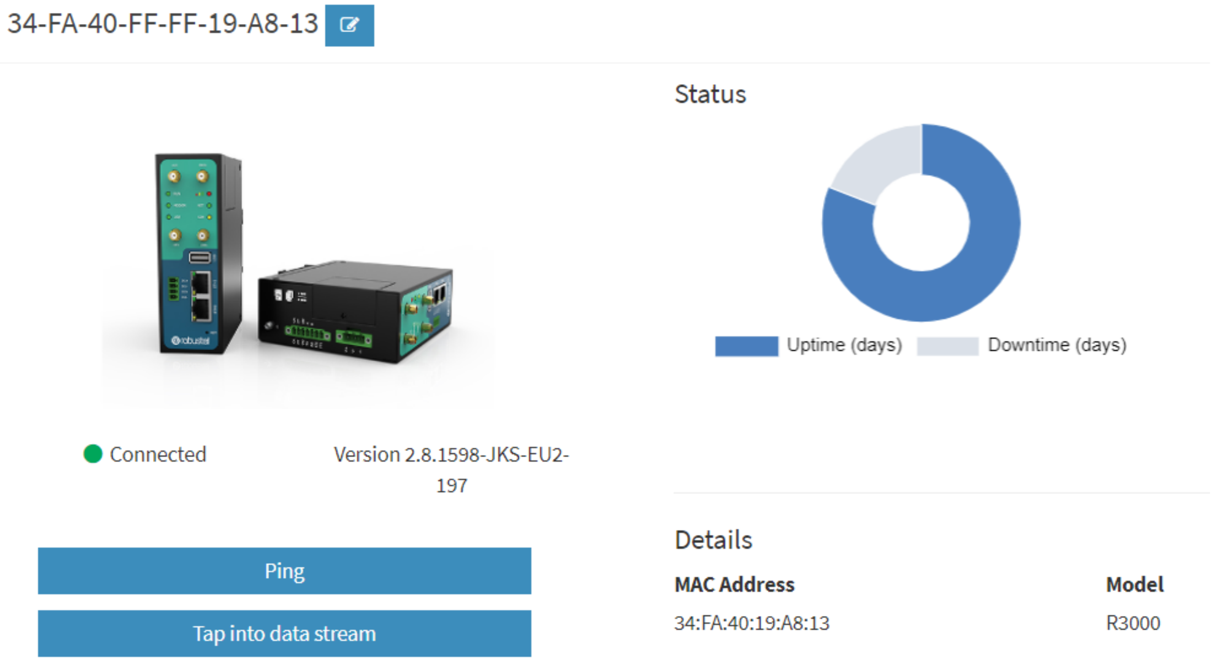
This document provides the Potsdam wastewater treatment plant results for a LoRa signal propagation test carried out at two maintenance holes more than 1200 meters away from the gateway using adaptive data rate functionality.

Dev EUI_1:

Dev EUI_2:

Dev EUI_3:

Gateway EUI: 34FA FF FF 19 18 13



34-FA-40-FF-FF-19-A8-13

Status

● Connected

Version 2.8.1598-JKS-EU2-197

Details

MAC Address	Model
34:FA:40:19:A8:13	R3000

Figure 6.3 Robustel Gateway used to Test incoming data from End Nodes

6.2 Scope

This chapter contains all the test results for the Potsdam Wastewater Treatment Plant signal propagation test and recommendations and next steps.

6.3 Test equipment.

Robustel R3000 LoRaWAN gateway mounted on building seven meters from the ground powered by its local power supply. Sim card was installed in the gateway to provide connectivity to the Loriot Network server.

Three LoRa end nodes :L01device with dev eui configured to

Dev eui1:

Dev eui2;

Dev EUI_3

6.4 Test methodology

Loriot packet forwarder was running on the Robustel Gateway with Sim Card installed on the gateway to backhaul the test data to the Loriot Network and application server. The gateway was installed at the Potsdam site, and LoRa end nodes were used to test. There were essentially three end nodes used for this test. Each of the end nodes was configured with its own unique DevEUI. In the LoRaWAN Stack, the dev Eui is the primary source of the first uplink authentication when the device sends a join request.

Each of the end nodes configured with their own unique DevEUI powered on would send a join request. The gateway would subsequently accept the Join request using OTAA from the end node. This is a standard LoRa Wan handshake when a device joins a network.

Once the device joins the gateway, a sequence of dummy sensor data is sent to the gateway in uplink messages. This data can be visualised on the LoRa application server. This data is also stored on the LORIOT Database.

Interpreting the data:

6.4.1 Receiver Sensitivity

The sensitivity of a radio receiver at room temperature is given by: Eqn. 1 (Orange, 2016)

$$S = -174 + 10\ \log_{10} BW + NF + SNR \qquad \text{Eqn. 1}$$

The first term is due to thermal noise in 1 Hz of bandwidth and can only be influenced by changing the receiver's temperature. (Orange, 2016) The second term, BW, is the receiver bandwidth. NF Is the receiver noise figure and is fixed for given hardware implementation. Finally, SNR represents the signal-to-noise ratio required by the underlying modulation scheme. The signal-to-noise ratio and bandwidth are available as design variables to the LoRa designer.

6.4.2 SNR and Spreading Factor

The basic premise of the spread spectrum is that each bit of information is encoded as multiple chips. (Semtech Corporation, 2013)The relationship between the bit and chip rate for LoRa modulation, and respectively, is given by: Eqn. 2

$$R_c = 2^{SF} R_b \qquad \text{Eqn. 2}$$

The performance of the LoRa modulation itself, forward error correction (FEC) techniques, and the spread spectrum processing gain combine to allow significant SNR improvements. Some example SNRs for both conventional and LoRa modulation formats are shown in the table below. Where SF is the spreading factor, SNR Is the minimum ratio of wanted signal power to noise demodulated. The lower this number, the more sensitive the receiver will be. Negative numbers indicate the ability to receive signal powers below the receiver noise floor:

RSSI(received signal strength indicator) measured in dBm. The absolute minimum is -115dBm. If the RSSI is, for example -30dBm, that would indicate a relatively strong signal. And the converse is true where a signal of -100dBm would indicate a significantly weak signal.

SNR(Signal to noise ratio) for LoRa generally between -20dBm and +10dBm. LoRa, when compared to other technologies, has a significantly high tolerance to noise because of its propriety CSS propagation. The integrity of the data is good, even though SNR is poor.

The number of uplink messages transmitted against the number received is also a good indication of LoRa wan reliability, seeing that LoRa wan is a “connectionless” architecture.

6.4.3 Forward Error Correction

The LoRa modem also employs Forward Error Correction (FEC) that permits the recovery of bits of information due to corruption by interference. This requires a small overhead of additional encoding of the data in the transmitted packet. Depending upon the coding rate selected, the additional robustness attained in the presence of thermal noise alone is shown in the family of curves below

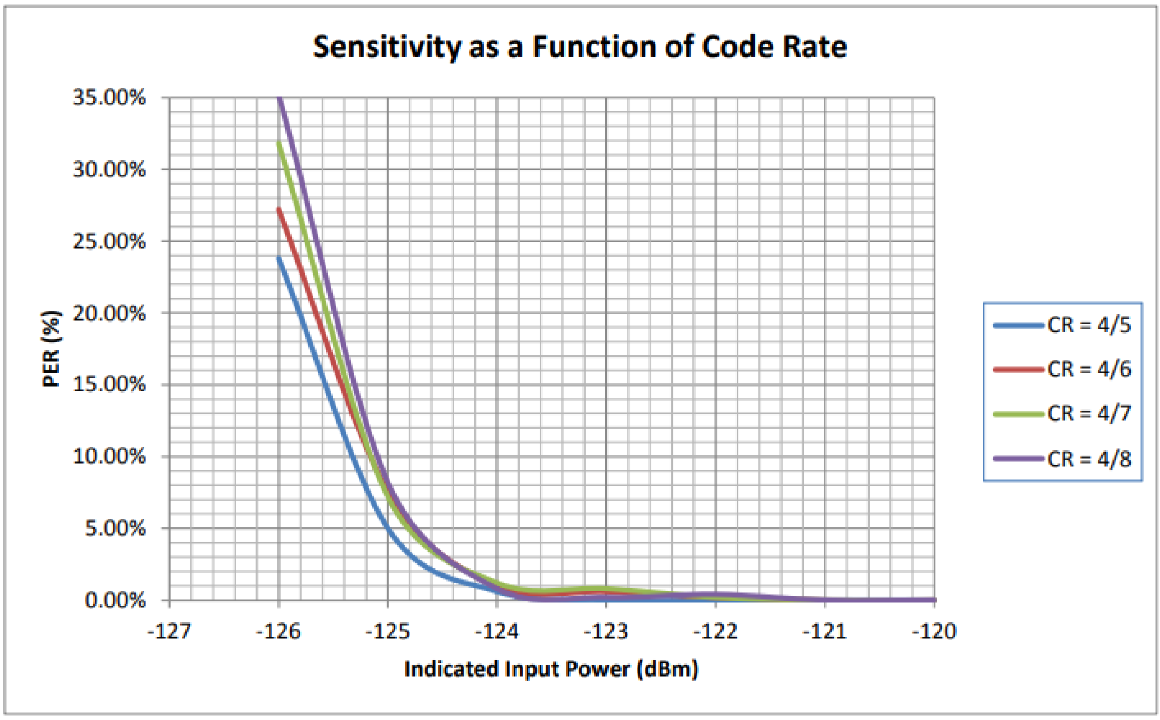


Figure 6.2: Data plot of LoRa performance using different coding Rates

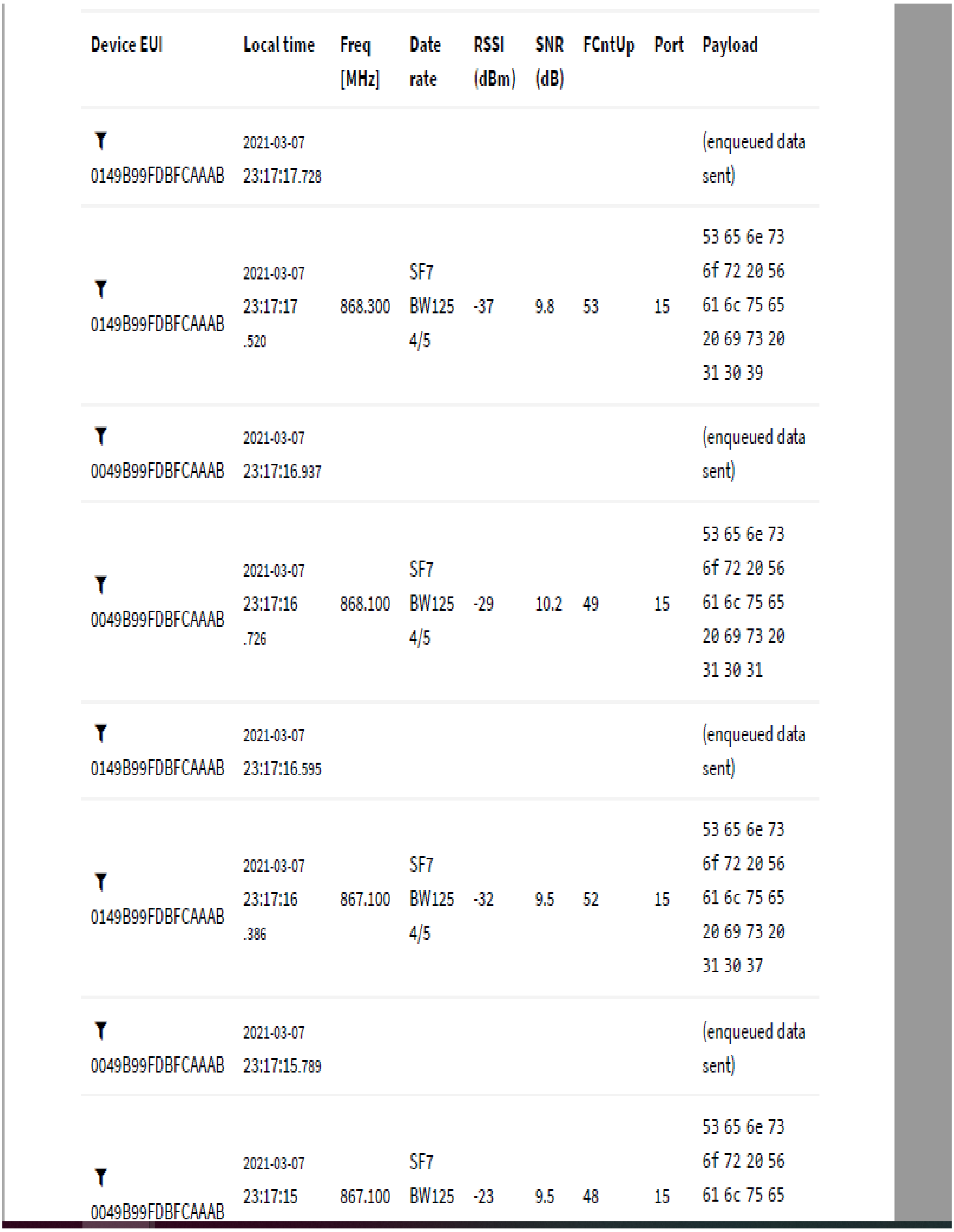
The real performance gain of FEC, however, is in the presence of bursts of interference. If the radio link is likely to be subject to such interference, the use of FEC should be evaluated. The table below then shows how the increase in coding rate influences time on-air for a fixed bandwidth of 250 kHz at SF = 10

6.3.5 Test Results

The data is captured in the spreadsheet below. The data was extracted from the Loriot network server

Dev Eui
Local time: Time and date of data received at LoRa network server
Freq:
Data rate: Spreading factor, Bandwidth, and Forward error correction
RSSI
SNR
FCnt Up
Port
Payload received

Table 6.1 List of different parameters visualised by LoRa APP Server(Loriot)



Device EUI	Local time	Freq [MHz]	Date rate	RSSI (dBm)	SNR (dB)	FCntUp	Port	Payload
0149B99FDBFCAAAB	2021-03-07 23:17:17.728							(enqueued data sent)
0149B99FDBFCAAAB	2021-03-07 23:17:17 .520	868.300	SF7 BW125 4/5	-37	9.8	53	15	53 65 6e 73 6f 72 20 56 61 6c 75 65 20 69 73 20 31 30 39
0049B99FDBFCAAAB	2021-03-07 23:17:16.937							(enqueued data sent)
0049B99FDBFCAAAB	2021-03-07 23:17:16 .726	868.100	SF7 BW125 4/5	-29	10.2	49	15	53 65 6e 73 6f 72 20 56 61 6c 75 65 20 69 73 20 31 30 31
0149B99FDBFCAAAB	2021-03-07 23:17:16.595							(enqueued data sent)
0149B99FDBFCAAAB	2021-03-07 23:17:16 .386	867.100	SF7 BW125 4/5	-32	9.5	52	15	53 65 6e 73 6f 72 20 56 61 6c 75 65 20 69 73 20 31 30 37
0049B99FDBFCAAAB	2021-03-07 23:17:15.789							(enqueued data sent)
0049B99FDBFCAAAB	2021-03-07 23:17:15	867.100	SF7 BW125	-23	9.5	48	15	53 65 6e 73 6f 72 20 56 61 6c 75 65

The results were as expected. The coding rate changes as the distance from the gateway increase when the end node is linearly and proportionally moved away from the gateway. It is important to note that even when the SNR values were relatively poor, the coding rate \changed without compromising the integrity of the data. However, the Loriot server did not account for the messages transmitted over a 30min period, which could mean that the packet forwarder would not transmit those messages to the application server where SNR was severely poor.

6.5 Conclusion

The attenuation of the signal happens due to having to pass through and around several structures like buildings, houses etc. The height of the gateway certainly provides significant improvement of signal and LoRa Wan as a whole. Also, in many LoRa white researchs, the general feeling that the height of the gateway, especially in metropolitan areas, significantly impacts packet success. In rural or sparsely populated structures, packet loss is very minimal. Also, with adaptive data rate being implemented, which relates to coding rate and forward error correction, packet loss is significantly decreased but with a compromise in data rates.

CHAPTER 7
CONCLUSION AND FUTURE WORK

7.1 Conclusion

At the beginning of this research, the objectives set out are the foundation of many research options concerning visualising data from different data sources. But this research certainly challenged the norm introducing a new perspective to system integration as well data portability. So many frameworks expose numerous possibilities in the IoT space with a little bit of exposure. The present technologies in terms of interoperability are making data and the use thereof very affordable. The interoperability of data provides data and technology convergence opportunities to augment present systems in Artificial Intelligence and augmented reality worlds. The platforms out there are so interchangeable that they also make data visualisations seamless. It is time that the end-users start seeing the direct benefit.

This research proved that gap between embedded systems and MES systems seem to be narrowing quite fast. Also, microcontrollers are growing to be capable of much more than the legacy microcontrollers of the past. There is also system on chips options that make interfacing of technologies easier to implement, thus enabling MES to be more effective. Protocols like OPC UA(Ye and Hong, 2018) that directly connect process control systems and embedded systems to MES systems are also crucial to narrowing the gap between the embedded worlds, the MES and WEB API. Protocols Like MQTT and COAP also assist in parsing data with low overhead, thus making level 0 (field-level data) more available, thus adding to the data value chain.

One of the other outcomes of this research is that to make the most of technology, and there needs to be adapting to the technology to get the full benefit because not all technologies meet the end-user’s requirement like a hand in glove. Numerous wrappers and interfaces were employed in this research, which means there is undoubtedly room for standard protocols to do away with these wrappers and interfaces. In addition, many visualisation platforms have built-in scripting functions that allow for more user-defined functionality.

Even though most of the implementations of this research were done using embedded and/or low-level platforms, reduced effort and minimal time taken to accomplish these implementations prove that the convergence of technologies is far more possible and seamless than in previous years.

The questions in terms of the life cycle are always on the table. With technology changing so fast, there is always the million-dollar question of how viable is the investment and how long will one derive benefit. The research highlighted that there is always a possibility of reusing existing hardware and rewriting the firmware in a new frame format, as was the case of the GSM logger device where existing hardware was used and re-written to a new format architecture.

7.2 Future Work

As the research progressed through the years, a few factors and concerns were realised after engagement with industry experts and interested third parties that were intrigued by the research.

The concerns that were realised in order of concern was;

i. Security
ii. End-User Interface and Visualisation.
iii. Controllability (can the framework allow remote control processing.

All the work accomplished in this research will be useless if the end-user can not use this framework or evens parts of this research, so the above concerns were shortcomings, which will be the basis of future work of this research.

The future work of this research will lean towards controllability and security. This future work will explore the options of using a CAN/Ethernet wrapper(Resources and Features, 2015) to introduce the controllability of the end devices. CAN has been a significant role-player in the automotive industry and has become a relatively new player in the industrial IOT space. Many modern microcontrollers standardise CAN as a peripheral on their devices, making CAN a viable option in the industrial IOT space. With JSON format also being used in the embedded platform, there is undoubtedly a case to have JSON/CAN wrapper to control industrial IoT devices.

Bibliography

Architecture, O. P. C. U. (2018) ‘IoT’, (June), pp. 1–48.

Astarloa, A. *et al.* (2016) ‘Intelligent gateway for industry 4.0-compliant production’, in *IECON Proceedings (Industrial Electronics Conference)*. doi: 10.1109/IECON.2016.7793890.

Atmel (2011) ‘8-bit Atmel Microcontroller Programmable ATmega128L’, *Www.Atmel.Com/Atmel/Acrobat/2467S.Pdf*, (8-bit Atmel Microcontroller), p. 384. Available at: http://www.atmel.com/images/doc2467.pdf.

Azhari, S. J. and Kaabi, H. (2001) ‘4-20 Ma Current Loop Transmitters’, 14(1), pp. 35–40.

Bangali, Dr. S. K. Shah, S. A. (2015) ‘Real Time School Bus Tracking System with Biometrics, GPS and GPRS using ARM Controller’, *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*. Ess & Ess Research Publications, 4(8), pp. 7023–7027. doi: 10.15662/ijareeie.2015.0408029.

Bassi, L. (2017) ‘Industry 4.0: Hope, hype or revolution?’, in *RTSI 2017 - IEEE 3rd International Forum on Research and Technologies for Society and Industry, Conference Proceedings*. doi: 10.1109/RTSI.2017.8065927.

Bäumker, E., Miguel Garcia, A. and Woias, P. (2019) ‘Minimizing power consumption of LoRa® and LoRaWAN for low-power wireless sensor nodes’, *Journal of Physics: Conference Series*, 1407(1). doi: 10.1088/1742-6596/1407/1/012092.

Calderón Godoy, A. J. and Pérez, I. G. (2018) ‘Integration of sensor and actuator networks and the SCADA system to promote the migration of the legacy flexible manufacturing system towards the industry 4.0 concept’, *Journal of Sensor and Actuator Networks*, 7(2). doi: 10.3390/jsan7020023.

Cavalieri, S. *et al.* (2017) ‘OPC UA integration into the web’, in *Proceedings IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society*. doi: 10.1109/IECON.2017.8216590.

Drahos, P. *et al.* (2018) ‘Trends in industrial communication and OPC UA’, in *Proceedings of the 29th International Conference on Cybernetics and Informatics, K and I 2018*. doi: 10.1109/CYBERI.2018.8337560.

Ekiz, H. *et al.* (no date) ‘Design and Implementation of a Can / Ethernet Bridge’, (Figure 1), pp. 1–10. Available at: http://www.can-cia.de/fileadmin/cia/files/3icc/proceedings/1996_ekiz_kutlu_baba_powner.pdf.

Em, H. and Module, P. C. E. (2012) ‘AT Command Interface Specification’, pp. 1–391.

Gutierrez-Guerrero, J. M. and Holgado-Terriza, J. A. (2019) ‘Automatic configuration of OPC UA for industrial internet of things environments’, *Electronics (Switzerland)*, 8(6). doi: 10.3390/electronics8060600.

Katti, B. *et al.* (2018) ‘SA-OPC-UA: Introducing Semantics to OPC-UA Application Methods’, in *IEEE International Conference on Automation Science and Engineering*. doi: 10.1109/COASE.2018.8560467.

Kim, W. and Sung, M. (2018) ‘Standalone OPC UA wrapper for industrial monitoring and control systems’, *IEEE Access*, 6(July), pp. 36557–36570. doi: 10.1109/ACCESS.2018.2852792.

Koon, J. (2020) ‘LoRaWAN Empowers Very Low-power, Wireless Applications’, *Tech Idea Research*, 1.0.

Kovar, J. *et al.* (2016) ‘Virtual Reality in Context of Industry 4.0’, *Proceedings of the 2016 17Th International Conference on Mechatronics - Mechatronika (Me) 2016*.

L., D., B., C. and J., J. (2015) ‘Performance evaluation of M2M protocols over cellular networks in a lab environment’, *2015 18th International Conference on Intelligence in Next Generation Networks, ICIN 2015*, pp. 70–75. doi: 10.1109/ICIN.2015.7073809.

Langmann, R. and Rojas-Pena, L. F. (2016) ‘A PLC as an industry 4.0 component’, in *Proceedings of 2016 13th International Conference on Remote Engineering and Virtual Instrumentation, REV 2016*. doi: 10.1109/REV.2016.7444433.

‘LTM-Research’ (no date).

Muller, M., Wings, E. and Bergmann, L. (2017) ‘Developing open source cyber-physical systems for service-oriented architectures using OPC UA’, in *Proceedings - 2017 IEEE 15th International Conference on Industrial Informatics, INDIN 2017*. doi: 10.1109/INDIN.2017.8104751.

Net, S. and Ua, O. P. C. (2014) ‘Programming an OPC UA . NET Client with C # for the SIMATIC NET OPC UA Server’.

Nordic (2008) ‘nRF24L01+ Single Chip 2.4GHz Transceiver Product Specification v1.0’, *Building Research*, (1), p. 78.

Orange (2016) ‘LoRa Device Developer Guide’, p. 42. Available at: https://partner.orange.com/wp-content/uploads/2016/04/LoRa-Device-Developer-Guide-Orange.pdf.

Pagnon, W. (2017) ‘The 4th Industrial Revolution – A Smart Factory Implementation Guide’, *International Journal of Advanced Robotics and Automation*. doi: 10.15226/2473-3032/2/2/00123.

Pauker, F. *et al.* (2016) ‘A Systematic Approach to OPC UA Information Model Design’, in *Procedia CIRP*. doi: 10.1016/j.procir.2016.11.056.

Power, L. and Kominek, D. (2013) ‘Keys To Developing an Embedded UA Server’, *Matrikon Opc*, pp. 1–7.

Profanter, S. *et al.* (2019) ‘OPC UA versus ROS, DDS, and MQTT: Performance evaluation of industry 4.0 protocols’, in *Proceedings of the IEEE International Conference on Industrial Technology*. doi: 10.1109/ICIT.2019.8755050.

Resources, D. and Features, D. (2015) ‘Compact CAN-to-Ethernet Converter Using 32-Bit ARM ® Cortex ™ -M4F MCU TI Designs’, (February).

Semtech Corporation (2013) ‘SX1272/3/6/7/8 LoRa Modem Design Guide, AN1200.13’, (July), p. 9. Available at: https://www.rs-online.com/.

Silveira Rocha, M. *et al.* (2018) ‘Performance Comparison between OPC UA and MQTT for Data Exchange’, in *2018 Workshop on Metrology for Industry 4.0 and IoT, MetroInd 4.0 and IoT 2018 - Proceedings*. doi: 10.1109/METROI4.2018.8428342.

Surachman, A. *et al.* (2011) ‘A Simple Microcontroller-Based 4-20 mA Current Loop Receiver for Sensors with Current Transmitters’, *Jurnal Otomasi Kontrol dan Instrumentasi*, 2(1), p. 15. doi: 10.5614/joki.2010.2.1.2.

Tao, F. *et al.* (2014) ‘IoT-Based intelligent perception and access of manufacturing resource toward cloud manufacturing’, *IEEE Transactions on Industrial Informatics*. doi: 10.1109/TII.2014.2306397.

‘tiny13A - secondary mcu.pdf’ (no date).

Uwe Steinkrauss, ascolab G. (2010) ‘Whitepaper. Overview: OPC Unified Architecture’, pp.

1–12.

Veichtlbauer, A., Ortmayer, M. and Heistracher, T. (2017) ‘OPC UA integration for field devices’, *Proceedings - 2017 IEEE 15th International Conference on Industrial Informatics, INDIN 2017*, pp. 419–424. doi: 10.1109/INDIN.2017.8104808.

Ye, X. and Hong, S. H. (2018) ‘An AutomationML/OPC UA-based Industry 4.0 Solution for a Manufacturing System’, in *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*. doi: 10.1109/ETFA.2018.8502637.

Yongde, Z. *et al.* (2014) ‘The controller development of multi-layer parking equipment based on STM32’, *International Journal of Smart Home*, 8(1), pp. 303–310. doi: 10.14257/ijsh.2014.8.1.31.

APPENDICES

Appendix A: Code Snippet of GSM logger(MCU serialisation of Data)

```
bool sendPacket(const uint8_t type, const bool ack, const uint32_t id, uint8_t* data, const
uint16_t len)
{
   bool result = false;
   uint16_t header;
   uint16_t crc;
   uint16_t packetLen;
   int32_t  tmp;

   memset(&header, 0, sizeof(header));
   memset(&crc, 0, sizeof(crc));

   packetLen = sizeof(header) + sizeof(id) + len + sizeof(crc);
   header = packetLen + (0x00 << 9) + ((type & 0xF) << 11);
   if (ack) header += (0x1 << 15);

   tmp = 0;
   tmp = calcCRC(tmp, (uint8_t *) &header, sizeof(header));
   tmp = calcCRC(tmp, (uint8_t *) &id, sizeof(id));
   if (len > 0) tmp = calcCRC(tmp, (uint8_t *) data, len);
   crc = tmp & 0xFFFF;

   result = gprs.sendOpen(packetLen) &&
          gprs.sendData((uint8_t *) &header, sizeof(header)) &&
          gprs.sendData((uint8_t *) &id, sizeof(id)) &&
          ((len == 0) || gprs.sendData(data, len)) &&
          gprs.sendData((uint8_t *) &crc, sizeof(crc)) &&
          gprs.sendClose();

   if (!result) gprs.closeUdpConnection();
   if (result)
   {
       cntSends++;
       tsActivity = clk.get();
   }
   return result;
}
```

Appendix B :

(At the Device gateway)

Stage 1: de-serialize the Data: This section of the code handles packet verification and data integrity.

```
function readPacketDetails(buf)
{
   const details = {length: 0, version: 0, type: 0, ack: 0, id: 0, crc: 0, crcPassed: false,
lengthPassed: false};

   details.length = buf.readUInt16LE(0) & 0x1FF;
   details.lengthPassed = (details.length == buf.length);
   details.crc = calcCRC(buf);
   details.crcPassed = (details.crc == buf.readUInt16LE(buf.length - 2));

   const header = buf.readUInt8(1) >> 1;
   details.version =  header & 0x3;
   details.type = (header >> 2) & 0xF;
   details.ack = (header >> 6) & 0x1;

   details.id = buf.readUInt32LE(2);

   return details;
}
```

Second Phase: Once packet is verified and CRC Checked,Interpret the payload

Routine Below: How to handle the payload.

```
gateway.on('message', (buf, rinfo) => {
    const pInfo = readPacketDetails(buf);
    pInfo.port = rinfo.port;
    pInfo.address = rinfo.address;

    if (pInfo.crcPassed)
        debug('gateway',  'got ' + buf.length + ' bytes from device ' + pInfo.id + ' on ' +
pInfo.address + ':' + pInfo.port);
    else
        debug('gateway',  'got a ' + buf.length + ' byte corrupt packet from ' +
pInfo.address + ':' + pInfo.port);

    if (pInfo.lengthPassed && pInfo.crcPassed)
    {
        logTraffic(new Date().toLocaleString() + ',IN,' + pInfo.id + ',' + pInfo.address + ',' +
pInfo.port + ',' + buf.toString('hex'));

        broadcastDeviceActivity(pInfo.id, 'device', utils.timestamp(), buf.toString('hex', 6,
buf.byteLength - 2), () => {
            if (pInfo.version == 0)
            {
```

```
            if (pInfo.ack == 0)
            {
                switch (pInfo.type)
                {
                    case 0x1:     // MSG_TYPE_TIME
                        pInfo.ack = 1;
                        sendTime(pInfo, buf);
                        break;
                    case 0xA:     // MSG_TYPE_DATA
                        pInfo.ack = 1;
                        if (savedSensorReading(buf, pInfo)) sendPacket(pInfo);
                        break;
                    case 0xF:     // MSG_TYPE_PING
                        pInfo.ack = 1;
                        sendPacket(pInfo);
                        break;
                    default:
                        debug('packet', 'Unknown packet type ' + pInfo.type + ' received.');
                }

            } else debug('packet', 'Not Valid');

            // update tsActivity in the devices table
            if ((pInfo.ack != 0) || (pInfo.type != 0xA)) updateDeviceActivity(pInfo.id);
        }
    });

  }
});
```

Appendix C:

Main. c program of Nfr24l01 radio module Transmitter module

The is the c code for implementing the nrf24l01 module on an STm32f4 using STMCubemax.

```
/* Some of this code was automatically generated using
```

The code for the transmitter and the receiver module barring a few changes.

```
*/
#include "main.h"


#include "MY_NRF24.h"



SPI_HandleTypeDef hspi1;


UART_HandleTypeDef huart2;



void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_SPI1_Init(void);
static void MX_USART2_UART_Init(void);
uint64_t TxpipeAddrs = 0x11223344AA;
char myTxData[32] = "hello world";
int main(void)
{

  HAL_Init();

  SystemClock_Config();


  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_SPI1_Init();
  MX_USART2_UART_Init();
  NRF24_begin(CEpin_GPIO_Port,CSNpin_Pin,GPIO_PIN_9,hspi1);
nrf24_DebugUART_Init(huart2);
printRadioSettings();
NRF24_stopListening();
NRF24_openWritingPipe(TxpipeAddrs);
NRF24_setAutoAck(false);
NRF24_setChannel(52);
NRF24_setPayloadSize(32);
  while (1)
  {




if(NRF24_write(myTxData,32))
{
	HAL_UART_Transmit(&huart2,(uint8_t *)"Transmitted sucessfully\r\n",strlen("transmitted successfully\r\n"),10);
}
		HAL_Delay(1000);
  }
```

```
}

void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};



  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE2);



  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 16;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
  RCC_OscInitStruct.PLL.PLLQ = 7;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
```

```
 }
}

static void MX_SPI1_Init(void)
{



  hspi1.Instance = SPI1;
  hspi1.Init.Mode = SPI_MODE_MASTER;
  hspi1.Init.Direction = SPI_DIRECTION_2LINES;
  hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi1.Init.NSS = SPI_NSS_SOFT;
  hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_32;
  hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi1.Init.CRCPolynomial = 10;
  if (HAL_SPI_Init(&hspi1) != HAL_OK)
  {
    Error_Handler();
  }



}

static void MX_USART2_UART_Init(void)
{



  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }


}


static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, CSNpin_Pin|CEpin_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : PC13 */
  GPIO_InitStruct.Pin = GPIO_PIN_13;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pins : CSNpin_Pin CEpin_Pin */
  GPIO_InitStruct.Pin = CSNpin_Pin|CEpin_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
```

```
}
```

Appendix D:

c# code to de-serialise flowrates of Wastewater treatment plant received from LoRa End nodes. The code below is used to visualise the data received from LoRa end node(enc28j60) using the C# application running on a local network of a wastewater treatment plant.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System. Data;
using System. Drawing;
using System. Linq;
using System. Text;
using System.Windows.Forms;
using System.Net;
using System.IO;
using System. Timers;
using System.Net.Sockets;
using System.Text.RegularExpressions;
using DevExpress.XtraBars;
using System.Runtime.Serialization.Formatters.Binary;
namespace scale
{
 public partial class Form1 : DevExpress.XtraEditors.XtraForm
 {
 private System.Timers.Timer _timer;
 public enum Flows
 {
 None = 0,
 Inflow_1 = 1,
 Inflow_2 = 2,
 OutFlow_2= 3,
 Qotflow_4 = 4
 }
 public Form1()
 {
 InitializeComponent();
 }
 private void butClose_ItemClick(object sender, ItemClickEventArgs e)
 {
```

```
 try
 {
 stopFlowReader();
 this.Close();
 }
 catch
 {
 }
 }
 private void startFlowReader()
 {
 _timer = new System.Timers.Timer(1000);
 _timer.Elapsed += new ElapsedEventHandler(_timer_Elapsed);
 _timer.Enabled = true; // Enable it
}

 public void stopFlowReader()
{
 _timer.Stop();
 }
 void _timer_Elapsed(object sender, ElapsedEventArgs e)
 {
 decimal _Weight = 0;
 try
 {
 // Inflow_1 = 172.16.1.101;       static IP address assigned to LoRa end node measuring
inflow point 1
 // Inflow_2 = 172.16.1.150                static IP address assigned to LoRa end node measuring
inflow point 2

 // Outflow_1 = 172.16.2.100             static IP address assigned to LoRa end node measuring
outflow point 1

 //Outflow_2 = 172.16.1.151              static IP address assigned to LoRa end node measuring
outflow point 2

 _timer.Stop();
 string _IPAddress = string.Empty;
 var _flow = (Flows)lkpFlows.EditValue;
 switch (_flow)
 {


 }
 if (_flow== Flow.None) { this.dGauge1.Text = string.Format
("{0:f0}", _Weight); return; }
 if (Flow.Inflow_1 == _flow) _IPAddress =
"172.16.1.150";
 if (Flow.Inflow_2 == _flow) _IPAddress = "172.16.1.101";
 if (Flow.Outflow_1 == _flow) _IPAddress =
"172.16.2.100";
 if (Flow.Outflow_2== _flow) _IPAddress =
"172.16.1.151";
 if (_IPAddress.Length == 0) { this.dGauge1.Text =
string.Format("{0:f0}", _Weight); return; }
 int nCount = 0;
 decimal[] Data = new decimal[10];
 using (TcpClient client = new TcpClient())
 {
 client.Connect(new IPEndPoint(IPAddress.Parse(_IPAddress),
 3002));
 //client.Connect(new IPEndPoint(IPAddress.Parse
("172.16.1.150"), 3002));
 while (true)
 {
 NetworkStream strm = client.GetStream();
 if (strm != null)
 {
 string[] flowrate = Regex.Split(ReadData(strm), "\r
\n");
 string[] flowrateFinal = flowrate[0].Split(new char[]
{ 'k' });
 decimal Flowrate = 0;
string resultString = Regex.Match(flowRateFinal
[0].Trim(), @"\d+").Value;
 if (flowRate.Length > 0)
 {
```

```
if (decimal.TryParse(resultString, out Flowrate)
== true)
{
Data[nCount] = Flowrate;
}
}
if (nCount >= 9)
{
_Flowrate = 0;
foreach (decimal nVal in Data)
{
if (nVal > 0) _Weight = nVal;
}
nCount = 0;
break;
}
else
{
nCount = nCount + 1;
}
System.Threading.Thread.Sleep(100);
}
}
}
}
catch (Exception ex)
{
_Flowrate = 0;
}
finally
{
_timer.Start();
}
try
{
this.dGauge1.Text = string.Format("{0:f0}", _Flowrate);
}
catch
```

```
 {
 }
 }
 private string ReadData(NetworkStream network)
 {
 string Output = string.Empty;
 byte[] bReads = new byte[1024];
int ReadAmount = 0;
 while (network.DataAvailable)
 {
 ReadAmount = network.Read(bReads, 0, bReads.Length);
 Output += Encoding.UTF8.GetString(bReads, 0, ReadAmount);
 }
 return Output;
 }
 private void labelControl1_Click(object sender, EventArgs e)
 {
 }
 private void button1_Click(object sender, EventArgs e)
 {
 int nCount = 0;
 decimal[] Data = new decimal[10];
 decimal _Flowrate = 000;
 using (TcpClient client = new TcpClient())
 {
 //client.Connect(new IPEndPoint(IPAddress.Parse(_IPAddress),
3002));
 client.Connect(new IPEndPoint(IPAddress.Parse("172.16.2.100"),
 3002));
 while (true)
 {
 NetworkStream strm = client.GetStream();
 if (strm != null)
 {
 string[] weight = Regex.Split(ReadData(strm), "\r\n");
 string[] weightFinal = weight[0].Split(new char[]
{ 'k' });
 decimal Weight = 0;
 string resultString = Regex.Match(weightFinal[0].Trim
(), @"\d+").Value;
 if (weightFinal.Length > 0)
 {
 if (decimal.TryParse(resultString, out Weight) ==
true)
 {
 Data[nCount] = Weight;
 }
 }
 if (nCount >= 9)
 {
 _Weight = 0;
foreach (decimal nVal in Data)
 {
 if (nVal > 0) _Weight = nVal;
 }
 nCount = 0;
 break;
 }
 else
 {
 nCount = nCount + 1;
 }
 }
 this.dGauge1.Text = string.Format("{0:f0}", _Weight);
 System.Threading.Thread.Sleep(100);
 }
 }
 }
 private void button2_Click(object sender, EventArgs e)
 {
 using (TcpClient client = new TcpClient())
 {
 //client.Connect(new IPEndPoint(IPAddress.Parse(_IPAddress),
3002));
 //client.Connect(new IPEndPoint(IPAddress.Parse
("172.16.2.100"), 3002));
```

```
 client.Connect(new IPEndPoint(IPAddress.Parse("172.16.1.101"),
 3002));
 while (true)
 {
 string str;
 using (NetworkStream stream = client.GetStream())
 {
 byte[] data = new byte[1024];
 using (MemoryStream ms = new MemoryStream())
 {
 int numBytesRead;
 while ((numBytesRead = stream.Read(data, 0,
data.Length)) > 0)
 {
 ms.Write(data, 0, numBytesRead);
 foreach (decimal val in data)
 {
 MessageBox.Show(val.ToString());
 this.dGauge1.Text = string.Format
("{0:f0}", val);
 }
 str = Encoding.ASCII.GetString(ms.ToArray(), 0,
 (int)ms.Length);

MessageBox.Show(data[5].ToString());
 }
 }
 }
 }
 }
 }
 private void button3_Click(object sender, EventArgs e)
 {
 using (TcpClient client = new TcpClient())
 {
 //client.Connect(new IPEndPoint(IPAddress.Parse(_IPAddress),
3002));
 //client.Connect(new IPEndPoint(IPAddress.Parse
("172.16.2.100"), 3002));
 client.Connect(new IPEndPoint(IPAddress.Parse("172.16.1.101"),
 3002));
 //string str;
 using (NetworkStream stream = client.GetStream())
 {
 byte[] data = new byte[1024];
 int numBytesRead = stream.Read(data, 0, data.Length);
 if (numBytesRead >0)
 {
 string str = Encoding.ASCII.GetString(data, 0,
numBytesRead);
 MessageBox.Show(str);
 this.dGauge1.Text = string.Format("{0:f0}",str);
 }
 }
 }
 }
 private void Form1_Load(object sender, EventArgs e)
 {
 Dictionary<int, string> colorEnums = Enum.GetValues(typeof(Scale))
 .Cast<Flow>().ToDictionary(x => (int)x, x => x.ToString());
 lkpFlow.Properties.ValueMember = "Key";
 lkpFlow.Properties.DisplayMember = "Value";
 lkpFlow.Properties.DataSource = colorEnums;
 // startFlowReader();
 }
 private void button4_Click(object sender, EventArgs e)
 {
 startFlowReader();
}
}
}
```

Appendix E:

```
const byte i2cPin       = PIN_PG1;     // - enable i2c communications with the tiny13A
const byte softPowerPin = PIN_PD7;  // HIGH = MCU is powered else LOW MCU switches off.

static const int csPin    = 8;     // Pins for the Data Flash chip
static const int resetPin = 8;
static const int wpPin    = 7;

DataFlash dataflash;
clock     clk;
cache     che(100, 66, 16, 247);
telitGPRS gprs(Serial1);
sensors   inputs;

uint16_t  msPause, msCount;

uint8_t   buffer[16];
uint32_t  msBufferSent;
uint8_t   cntSends;
// uint32_t  tsCheckClock;
uint32_t  tsActivity;
uint32_t  tsSensorActivity;


void debug(const __FlashStringHelper *string)
{
   gprs.unlatchSerial();
   Serial1.print(string);
   gprs.latchSerial();
}


void debug(const char* fmt, ...)
{
   int count;
   va_list args;
```

```
  byte b;

  va_start(args, fmt);
  count = strlen(fmt);

  gprs.unlatchSerial();
  for (int i = 0; i < count; i++)
  {
    if (fmt[i] == '%')
    {
    if (fmt[i+1] == '%')
    {
      Serial1.write(fmt[i++]);
    } else {
      switch(fmt[i+1])
      {
      case 'd': Serial1.print(va_arg(args, int32_t));
            break;
      case 'i': Serial1.print(va_arg(args, int32_t));
            break;
      case 'l': Serial1.print(va_arg(args, uint32_t));
            break;
      case 'w': Serial1.print(va_arg(args, uint16_t));   // use this for byte values, as the print doesn't work with byte.
            break;
      case 'c': Serial1.print((char)va_arg(args, int16_t));
            break;
      case 's': Serial1.print(va_arg(args, char*));
            break;
      case 'h': b = (uint8_t) va_arg(args, uint16_t);
            if (b < 16) Serial1.write(48);
            Serial1.print(b, HEX);
            break;
      default: ;
      }
      i++;
    }
    } else
```

```
        Serial1.write(fmt[i]);
    }
    gprs.latchSerial();

    va_end(args);
}


void bufferSensorValues()
{
    uint8_t  readBuffer[16];
    uint32_t temp = clk.get();
    uint32_t values = 0;
    memset(&readBuffer[0], 0, sizeof(readBuffer));
    memcpy(&readBuffer[0], &temp, 4);
    values = inputs.getValue(0) & 0x3FF;
    temp = inputs.getValue(1) & 0x3FF;
    values += (temp << 10);
    temp = inputs.getValue(2) & 0x3FF;
    values += (temp << 20);
    memcpy(&readBuffer[4], &values, 4);
    values = inputs.getValue(5);
    temp = inputs.getValue(6);
    // values += (temp << 16);
    memcpy(&readBuffer[8], &values, 4);
    memcpy(&readBuffer[12], &temp, 4);
    tsSensorActivity = clk.get();
    if (!che.add(&readBuffer[0], 16)) debug(F("FAILED to cache reading :-("));
}


int calcCRC(int crc, uint8_t * ptr, int count)
{
   char i;
   while (--count >= 0)
   {
      crc = crc ^ (int) *ptr++ << 8;
      i = 8;
      do
      {
         if (crc & 0x8000)
            crc = crc << 1 ^ 0x1021;
         else
            crc = crc << 1;
      } while(--i);
   }
   return crc;
}


bool sendPacket(const uint8_t type, const bool ack, const uint32_t id, uint8_t* data, const
uint16_t len)
{
    bool result = false;

    uint16_t header;
    uint16_t crc;
    uint16_t packetLen;

    int32_t  tmp;

    memset(&header, 0, sizeof(header));
    memset(&crc, 0, sizeof(crc));

    packetLen = sizeof(header) + sizeof(id) + len + sizeof(crc);
    header = packetLen + (0x00 << 9) + ((type & 0xF) << 11);
    if (ack) header += (0x1 << 15);

    tmp = 0;
    tmp = calcCRC(tmp, (uint8_t *) &header, sizeof(header));
    tmp = calcCRC(tmp, (uint8_t *) &id, sizeof(id));
    if (len > 0) tmp = calcCRC(tmp, (uint8_t *) data, len);
    crc = tmp & 0xFFFF;

    result = gprs.sendOpen(packetLen) &&
             gprs.sendData((uint8_t *) &header, sizeof(header)) &&
            gprs.sendData((uint8_t *) &id, sizeof(id)) &&
            ((len == 0) || gprs.sendData(data, len)) &&
            gprs.sendData((uint8_t *) &crc, sizeof(crc)) &&
            gprs.sendClose();

    if (!result) gprs.closeUdpConnection();
    if (result)
    {
        cntSends++;
        tsActivity = clk.get();
    }

    return result;
}

bool sendPacketTime()
{
    uint16_t* milliseconds;
    uint32_t* seconds;
    uint32_t* marker;
    uint8_t   buf[10];

    marker = (uint32_t*) &buf[0];
    seconds = (uint32_t*) &buf[4];
    milliseconds = (uint16_t*) &buf[8];

    *milliseconds = 0;
    *marker = clk.get(seconds);

    return sendPacket(MSG_TYPE_TIME, false, COCT_SITE_ID, buf, 10);
}

bool handleIncomingDataFromServer()
{
    uint8_t  *data;
    uint16_t length;
    uint32_t *temp;
    uint32_t *seconds;
    uint16_t *milliseconds;
    int crc = 0;

    uint8_t  version = 0;
    uint8_t  type    = 0;
    bool     ack     = false;
    uint32_t id      = 0;

    uint32_t test;

    // uint32_t difference = 0;
    if (gprs.recv())
    {
        cntSends = 0;
        gprs.data(&data, &length);

        temp = (uint32_t *) data;
        if ((*temp & 0x1FF) == length)
        {
            crc = calcCRC(crc, data, length - 2);
            temp = (uint32_t *) (data + length - 2);
            if ((crc & 0xFFFF) == (*temp & 0xFFFF))
            {
                // Length and CRC passed !
                memcpy(&id, data+2, sizeof(id));
                temp = (uint32_t *) (data+1);
                version = (*temp >> 1) & 0x3;
                type = (*temp >> 3) & 0xF;
                ack = (((*temp >> 7) & 0x1) == 1);
                data += 6;

                if (version == 0)
                {
                    // Everything should be ready now to simply check response and handle data
                    if (ack)
                    {
                        // Responses to data sent are handled here
        switch (type)
        {
            case MSG_TYPE_TIME:
                seconds = (uint32_t *) data;
                milliseconds = (uint16_t *) (data+4);
                temp = (uint32_t *) (data+6);
                test = clk.set(*seconds, *milliseconds, *temp);

                //temp = (uint32_t *) data;
                //difference = clk.get();
                //difference = (difference > *temp) ? difference - *temp : *temp - difference;
                debug("Remote clock is %l.%w, with marker = %l... %l\r\n", *seconds, *milliseconds, *temp, test);
                //if (!clk.isSet() || (difference > 5)) clk.set(*temp - 1);
                //tsCheckClock = clk.get() + 360;
                break;
            case MSG_TYPE_DATA:
                if (msBufferSent > 0)
                {
                    msBufferSent = 0;
                    // Delete the oldest record from the cache
                    che.shift();
                }
                break;
            case MSG_TYPE_PING:
                break;
            default:
                debug("Received ACK for unknown message type (%h)\r\n", type);
        }

    } else {
        // Requests from the server are handled here
        debug("No logic to handle request (%h) from server.\r\n", type);
    }
}

tsActivity = clk.get();
```

```
            }
        }
    }

    return true;
}


void onReceiveI2C(int numBytes)
{
    uint8_t  id;
    uint32_t pulseInterval = 0;
    uint16_t pulseCount = 0;

    uint32_t  value;
    uint8_t  count = 0;

    while (Wire.available())
    {
        value = (uint8_t) Wire.read();
        if (count == 0)
        {
            id = value;

        }  else if (count < 5) {
            pulseInterval += (value << (8 * (count - 1)));

        } else pulseCount += (value << (8 * (count - 5)));
        count++;
    }

    debug("I2C: id=%w, ms=%l, cnt=%w.\r\n", id, pulseInterval, pulseCount);
    if ((count == 7) && (id < 3)) inputs.setPulse(id, pulseInterval / pulseCount);
    // if (count == 3) inputs.setPulse(id, pulseInterval & 0xFFFF);
}

void setup()
{
    uint8_t status;
    DataFlash::ID id;
    uint16_t counter = 0;

    pinMode(i2cPin, OUTPUT);
    pinMode(softPowerPin, OUTPUT);

    digitalWrite(i2cPin, HIGH);
    digitalWrite(softPowerPin, HIGH);

    Serial1.begin(57600);
    Wire.onReceive(onReceiveI2C);
    Wire.begin(3);
    msPause = 0;
    msCount = 0;
    memset(&buffer[0], 0, sizeof(buffer));
    msBufferSent = 0;
    cntSends = 0;
    tsSensorActivity = 0;
    tsActivity = 0;

    // DataFlash configuration
    SPI.begin();
    dataflash.setup(csPin);
    dataflash.begin();

    status = dataflash.status();
    dataflash.readID(id);
    debug("DataFlash: status(%h) manufacturer(%h) d0(%h) d1(%h) ext(%h)", status,
id.manufacturer, id.device[0], id.device[1], id.extendedInfoLength);

}



void loop()
{
    msPause = 0;
    // get the GSM engine up and running !
    if (!gprs.isOnline())
    {
        if (!gprs.isEnabled() && !gprs.enable()) msPause = 5000;
        if ((msPause == 0) && gprs.isRegistered() && gprs.isGprsRegistered())
        {
            // configure and connect here
            if (gprs.isActive())
            {
                if (!gprs.openUdpConnection())
                {
                    gprs.deactivate();
                    msPause = 5000;
                    debug(F("Gprs active but connection failed"));
                }
            } else {
                if (!(gprs.signalStrength() && gprs.configure() && gprs.activate() &&
gprs.openUdpConnection()))
                {
                    msPause = 5000;
                    // Retry 3 times then disable the modem and start again
                    debug(F("First gprs connection failed"));
                }
            }

        } else msPause = 1000;

    } else {

        msPause = 5000;

        // First process any data back from the server
        handleIncomingDataFromServer();

        // if (!clk.isSet() || ((tsCheckClock > 0) && (clk.get() > tsCheckClock)))
        if (clk.checkClockNow())
        {
```

```
      // First things first, need to set the clock as soon as the device comes online
      if (sendPacketTime()) msPause = 10000;

    } else {

      // Business as normal

      if ((msBufferSent == 0) && (che.size() > 0) && che.next(&buffer[0], sizeof(buffer))) msBufferSent = millis() - 10000;

      if ((msBufferSent != 0) && (millis() - msBufferSent > 9999))
      {
        if (sendPacket(MSG_TYPE_DATA, false, COCT_SITE_ID, buffer, sizeof(buffer)))
        {
          msPause = 10000;
          msBufferSent = millis();
        }

      } else if (clk.get() - tsActivity > 60) {

        if (sendPacket(MSG_TYPE_PING, false, COCT_SITE_ID, NULL, 0)) msPause = 10000;
      }
    }
  }


  if (msPause == 0) msPause = 100;
  msCount = 0;

  if (gprs.hasEventFired_NOCARRIER()) gprs.closeUdpConnection();
  if (gprs.hasEventFired_SRING()) msPause = 0;

  while (msCount < msPause)
  {

    if (gprs.isReadPending()) break;
    if (((msCount % 1000) == 0) && clk.isSet() && inputs.read() &&
(inputs.percentageChangeOf(1) || (clk.get() - tsSensorActivity > 3600)))
    {
      //debug("Got new values for inputs");
      bufferSensorValues();
      inputs.rollValues();
      break;
    }

    msCount += 100;
    delay(100);
  }

}
```

Appendix F;

```
#include <Arduino.h>


// I2C definitions
#define I2C_SDA         PB0                   // serial data pin
#define I2C_SCL         PB2                   // serial clock pin
#define I2C_SDA_HIGH()  DDRB &= ~(1<<I2C_SDA) // release SDA    -> pulled HIGH by resistor
#define I2C_SDA_LOW()   DDRB |=  (1<<I2C_SDA) // SDA as output -> pulled LOW  by MCU
#define I2C_SCL_HIGH()  DDRB &= ~(1<<I2C_SCL) // release SCL    -> pulled HIGH by resistor
#define I2C_SCL_LOW()   DDRB |=  (1<<I2C_SCL) // SCL as output -> pulled LOW  by MCU

#define NOP __asm__ __volatile__ ("nop\n\t")

const uint32_t maxLapse = 105800000;
const uint32_t rampDownInterval = 60000;

uint32_t tsStart[2];
uint32_t tsLapse[2];
uint16_t pulses[2];
bool     state[2];
uint8_t  pin[2] = {PIN_PB3, PIN_PB4};

uint32_t  tsActivity;
bool      temp;


void nop()
{
   for(int i = 0; i < 25; i++)
NOP;
}
```

```
/*
   I2C init function
*/
void I2C_init(void)
{
   DDRB  &= ~((1<<I2C_SDA)|(1<<I2C_SCL));  // pins as input (HIGH-Z) -> lines released
   PORTB &= ~((1<<I2C_SDA)|(1<<I2C_SCL));  // should be LOW when as ouput

   I2C_SCL_HIGH();
   I2C_SDA_HIGH();
   nop();
}

/*
   I2C transmit one data byte to the slave, ignore ACK bit, no clock stretching allowed
*/
void I2C_write(uint8_t data)
{
   // transmit 8 bits, MSB first
   for(uint8_t i = 8; i; i--)
   {
      //  ??? SDA LOW for now (saves some flash this way)
      // SDA HIGH if bit is 1
      if (data & 0x80) I2C_SDA_HIGH(); else I2C_SDA_LOW();
      nop();
      I2C_SCL_HIGH();                              // clock HIGH -> slave reads the bit
      data<<=1;                                   // shift left data byte, acts also as a delay
      nop();
      nop();
      I2C_SCL_LOW();                               // clock LOW again
      nop();
   }

   // release SDA for ACK bit of slave
   I2C_SDA_HIGH();
   nop();
   I2C_SCL_HIGH();                                           // 9th clock pulse is for the ACK bit
```

```
    nop();
    nop();                                  // ACK bit is ignored, just a delay    must
                                            implement this!!!
    I2C_SCL_LOW();                          // clock LOW again
    nop();
}

/*
    I2C start transmission
*/
void I2C_start(uint8_t addr)
{
    nop();
    I2C_SDA_LOW();                     // start condition: SDA goes LOW first
    nop();
    nop();
    I2C_SCL_LOW();                     // start condition: SCL goes LOW second
    nop();
    I2C_write(addr);                   // send slave address
}

/*
    I2C stop transmission
*/
void I2C_stop(void)
{
    I2C_SDA_LOW();                     // prepare SDA for LOW to HIGH transition
    nop();
    I2C_SCL_HIGH();                    // stop condition: SCL goes HIGH first
    nop();
    nop();
    I2C_SDA_HIGH();                    // stop condition: SDA goes HIGH second
    nop();
}


bool sendReading(const uint8_t id, const uint32_t lapse, const uint16_t count)
{
    bool successful = true;

    I2C_start(0x06); // ID is actually 3
    I2C_write(id+1);
    I2C_write(lapse & 0xFF);
    I2C_write((lapse >> 8) & 0xFF);
    I2C_write((lapse >> 16) & 0xFF);
    I2C_write((lapse >> 24) & 0xFF);
    I2C_write(count & 0xFF);
    I2C_write((count >> 8) & 0xFF);
    I2C_stop();

    delay(100);
    return successful;
    // if (successful) pulses[id] = 0;
}


/*
    Timing calculation readings: ~4404329ms = 1 hr, ~52925355ms = 12h1m,
    and therefore 1 day = ~105703899ms
    If a pulse measures longer than 105799999, the rate is assumed 0 units.
*/
void checkForPulse(const uint8_t id)
{
    tsActivity = millis();
    temp = digitalRead(pin[id]);

    if (temp != state[id])
    {
        state[id] = temp;
        if (temp)
        {
            if (tsStart[id] > 0)
            {
                pulses[id]++;
                if (sendReading(id, tsActivity - tsStart[id], pulses[id]))
                {
                    tsLapse[id] = tsActivity - tsStart[id];
                    tsStart[id] = tsActivity;
                    pulses[id] = 0;
                }

            } else tsStart[id] = tsActivity;
        }
        delay(100);

    } else {

        if ((tsLapse[id] > 0) && (tsLapse[id] < maxLapse) && ((tsActivity - tsStart[id]) > (tsLapse[id] + rampDownInterval)))
        {
            tsLapse[id] += rampDownInterval;
            sendReading(id, tsLapse[id], (pulses[id] > 0) ? pulses[id] : 1);
        }
    }
}


void setup()
{
    tsActivity = 0;

    for(int i = 0; i < 2; i++)
    {
        tsStart[i] = 0;
        tsLapse[i] = 0;
        pulses[i] = 0;
        state[i] = false;
        pinMode(pin[i], INPUT);
    }

    I2C_init();
}
```

```
void loop()
{
   uint8_t i;
   for(i = 0; i < 2; i++) checkForPulse(i);
}
```