LOW-POWER LTE-M BASED REMOTE MONITORING OF PROPANE TANK FILL LEVELS USING ASYNCHRONOUS EMBEDDED RUST FIRMWARE

by

Daniel Kyle Hayes

A thesis submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Master of Science in
Applied Energy and Electromechanical Engineering

Charlotte

2022

Approved by:

Dr. Maciej Noras

Dr. Michael Smith

Dr. James Conrad

ABSTRACT

DANIEL KYLE HAYES. Low-power lte-m based remote monitoring of propane tank fill levels using asynchronous embedded rust firmware. (Under the direction of DR. MACIEJ NORAS)

Remote monitoring of propane tank fuel levels can have many convenience and cost benefits for both suppliers and consumers. For suppliers, physical tank checks can be eliminated while reducing the maintenance and fuel costs for fill trucks. As a result, customer satisfaction can be increased according to better visibility and quicker service. For consumers, monitoring eliminates physical gauge checking, offers a better understanding of usage habits, and eliminates running tanks dry for remote or part-time residences. The current monitoring devices on the market are expensive and are not designed for a battery life of greater than 5 years for wide-area coverage. The development of an open-source cellular device with secure communications and self-sustaining power is investigated. The Rust programming language is reviewed for its possible use in the prototype for its memory safety and speed. During the development of the monitoring device different communication protocols, microcontroller hardware, sensor hardware, and power sources are compared. The resulting prototype uses a Conexio Stratus development kit based on the nRF9160 system-in-package (SiP). The CoAP protocol is utilized with DTLS over LTE-M networks which offers efficient and secure communications to the cloud with pre-shared keys (PSK). A lithium-ion capacitor (LIC) paired with solar-charging eliminates the need for a battery, which can supply power for the entire life of the device. Power profiling demonstrated the LIC's ability to store enough energy to power the device up to 28 days in the absence of any solar charging and a minimum of 90% transmission success rate. Sensor testing showed that the hall-effect sensor setup can accurately measure between 10% and 95% fuel levels with $\pm$ 1% accuracy compared to the gauge position.

ACKNOWLEDGEMENTS

I would like to take the time to extend my gratitude to Dr. Maciej Noras for his guidance and patience. I extend my appreciation to Dr. Michael Smith and Dr. James Conrad for their participation and the donation of their valuable time. I would also like to acknowledge and extend a special thanks to Conexio Technologies for donating a Status cellular development kit in support of my research. Your hardware donation and support are greatly appreciated. Also, I appreciate Nordic Semiconductors for donating a power profiler kit version 2 in support of my efforts. Special thanks to the fantastic, embedded Rust community for their dedication to growing Rust support for embedded devices and providing knowledge for new-comers and veterans alike.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ADC	Analog to Digital Converter.
CoAP	Constrained Application Protocol.
DDoS	Distributed Denial of Service.
DoD	Depth of Discharge.
DTLS	Datagram Transport Layer Security.
EDLC	Electric Double Layer Capacitor.
eDRX	Extended Discontinuous Reception.
ESR	Equivalent Series Resistance.
FFI	Foreign Function Interface.
FSM	Finite State Machine.
GNSS	Global Navigation Satellite System.
IDE	Integrated Development Environment.
IoT	Internet of Things
IP	Intellectual Property.
Kbps	Kilobits per second.
LIB	Lithium-Ion Battery.
LIC	Lithium-Ion Capacitor.
LiPo	Lithium-ion Polymer.
LPWAN	Low-Power Wide Area Network.

LTE	Long-Term Evolution, standard for wireless broadband communication.
LTE-M	Long-Term Evolution Machine Type Communication.
M2M	Machine to Machine.
Mbps	Megabits per second.
MQTT	Message Queuing Telemetry Transport.
NB-IoT	NarrowBand-Internet of Things.
NREL	National Renewable Energy Laborator.
OEM	Original Equipment Manufacturer.
OSI	Open Systems Interconnection.
PPK2	Power Profiler Kit Version 2.
PSK	Pre-Shared Key.
PSM	Power Save Mode.
PWM	Pulse Width Modulation.
QoS	Quality of Service.
RAM	Random Access Memory.
RF	Radio Frequency.
RTOS	Real-Time Operating System.
RTT	Real-Time Transfers.
SIM	Subscriber Identity Module.
SiP	System in Package.

SoC	System on Chip.
SPM	Secure Partition Manager.
SSID	Service Set Identifier.
TAU	Tracking Area Updates.
TCP	Transmission Control Protocol.
TLS	Transport Layer Security.
TOML	Tom's Obvious, Minimal Language.
UART	Universal Asynchronous Receive/Transmit.
UDP	User Datagram Protocol.

CHAPTER 1: INTRODUCTION

Many homes and businesses consume propane for heating purposes and have tank-based storage due to a lack of pipeline access. This means that the tanks must be filled regularly without any standard means for monitoring the fill level. The consumer must keep track of the fill level and order refill services when needed or pay for the supplier to perform physical checks periodically if available. Most propane tank gauges are driven by a rotating magnet positioned by a float mechanism. Currently, there are remote-ready tank gauges on the market, which accepts a hall-effect sensor module to determine fill level based on magnetic flux density as shown in Figure 1.1.

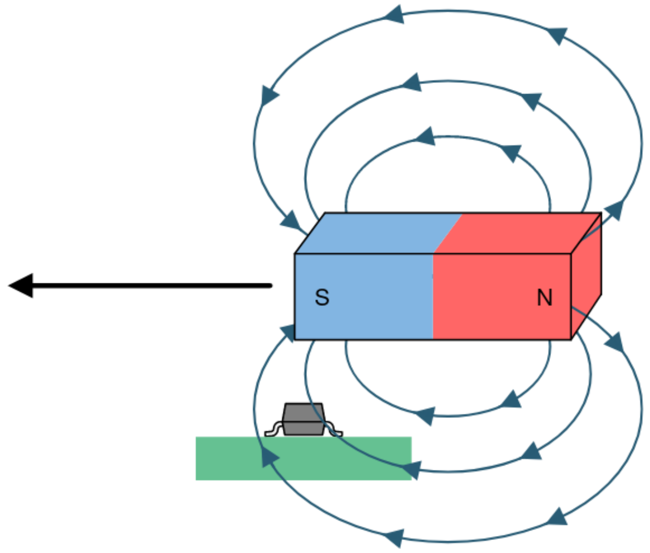


Figure 1.1: Common magnet orientation for hall-effect sensors (Image Courtesy: Texas Instruments).

There are many benefits, both from a provider and consumer perspective, when setting up remote monitoring including:

1. eliminating physical checks;

2. greater insight into usage habits;
3. remote areas or part-time residents do not need to worry about running dry;
4. suppliers can streamline their filling operations;
5. less fuel consumption by fill trucks;
6. better customer satisfaction.

1.1 Motivation and Problem Statement

The benefits for actively monitoring propane tank fuel levels can be significant, but implementing a solution is not as obvious. First, is to determine what options are currently available on the market, specifically devices that available to purchase by end-users. The Generac model 7005 monitor utilizes a Wi-Fi connection for monitoring services which requires an internet connection and comes with a $237 price tag. The Generac model 7009, shown in Figure 1.2, is the cellular version of the 7005 and costs $250 with a required $50 yearly subscription [1].

Figure 1.2: Generac 7009 cellular propane tank monitoring device (Image Courtesy: Generac).

The Generac monitor advertises a 5-6 year battery life with a transmission rate of once per 24 hours.

Monnit offers a propane monitoring device for more commercial-type settings, shown in Figure 1.3 and operates on a 900 MHz frequency with a proprietary protocol. The sensor costs $391, requires a 900 MHz gateway to manage data transmission with common communications, such as Ethernet or a cellular network, and advertises a battery life up to 12 years. The gateway can manage several sensor nodes and adds $242 to $385 to the total cost [2], and requires mains electricity for operation.

Figure 1.3: Monnit wireless propane tank level monitor (Image Courtesy: Monnit).

When considering these two devices, a few things stand out: they do not offer any type of energy harvesting, require battery replacements, and are expensive. Despite the Monnit gateway being able to handle multiple sensors, the sensor's 900 MHz protocol range is only 1200 feet, which is not applicable for a monitoring system with distributed tanks. It seems that there is room for improvement in the propane tank monitoring sector, but what features are required to offer such improvements?

The monitoring sensor is considered an Internet-of-Things (IoT) device because it is a distributed system that relays information over a network. IoT devices almost always consider battery life as a primary concern since devices are dispersed making battery changes very costly when considering both material and labor costs [3]. In some cases, the devices are placed in environments that are not easily or safely accessible compounding the cost of changing batteries. An ideal IoT device would have a battery or power source that can last for the lifetime of the sensor without relying on mains electricity. Secondly, an IoT device needs a reliable communication method to send data with, otherwise the deployment costs would far outweigh any added benefits.

In close relationship with reliable communications, is secure communications. Security is very important to protect data integrity and prevent exploits such as distributed denial-of-service attacks (DDoS) using insecure IoT devices. Security vulnerabilities often arise from memory bugs in firmware, and most embedded device firmware is heavily skewed towards C and C++ programming [4]. These languages are not memory safe which increases the possibility of memory misuse such as double-frees, use-after-free, and out-of-bounds accesses. Rust is a modern, systems programming language that provides high-level features with low-level control while providing memory safety. The application of Rust to the IoT sector may provide an safer alternative to C-based firmware and should be considered for any new deployments.

The security of communications can be improved by eliminating memory vulnerabilities, but the communication method is also important for efficiency. Many previous IoT deployments have been implemented with Wi-Fi networks. While Wi-Fi provides plenty of data bandwidth, security is not a default feature and needs to be managed and maintained by the operator. Cellular IoT deployments can be a safer network option with *batteries included* security since it is licensed as a part of 3GPP Release 13 [5]. Since the release of LTE-M and NB-IoT for wide-area IoT networks, cellular devices have steadily grown and is projected to support over 5 billion devices by 2028 as shown in Figure 1.4 [6].

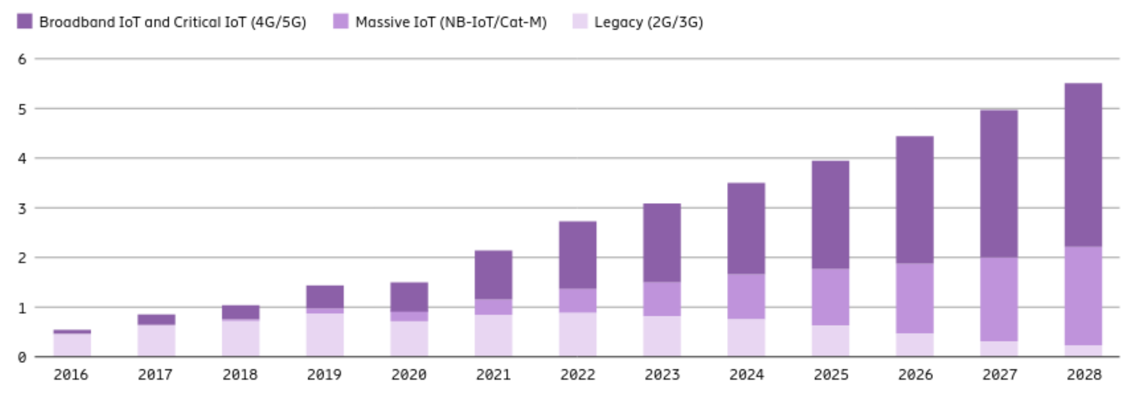


Figure 1.4: Cellular IoT connections by segment and technology in billions (Image Courtesy: Ericsson).

Cellular IoT devices seem like a great option for wide-area deployments which also support mobility; however, energy efficiency could be a concern. Lithium-ion batteries (LIB) have typically been the de facto standard for small, portable devices because of their energy density, but they are not the most environmentally friendly products [7]. Lithium-ion capacitors (LIC) are a novel alternative to LIBs being a hybrid of both LIBs and super-capacitors resulting in a device with greater cycler life than LIBs and greater energy density than super-capacitors [8]; therefore, LICs should be considered as a replacement power source for IoT devices.

Based on these considerations, the research goals are to develop an open-source monitoring device that: uses a reliable and secure communication method, is developed in the Rust programming language for memory safety and reliability, and a power source to eliminate or reduce battery replacements.

1.2 Literature Review

Noseda *et al.* provide insight into the benefits of using the Rust programming language for secure IoT applications [9]. Memory safety bugs are arguably the most common and the most costly when security is considered. Even when security is not a priority, memory misuse can have tremendous impacts on device functionality. Although, C and C++ have static and dynamic code analysis tools, they still can miss memory bugs as detailed in Table 1.1.

Table 1.1: Static and dynamic code analysis tool testing for memory unsafe languages.

Bug	Cppcheck	Splint	GNU C/C++ sanitize
0	✓	✓	✓
1	✓	✓	✓
2	✓	✓	✓
3	missed	✓	missed
4	missed	✓	compile error
5	✓	✓	missed
6	✓	✓	missed
7	missed	✓	missed
8	✓	✓	partly
9	partly	✓	partly
10	✓	missed	missed
11	missed	✓	missed
12	✓	✓	missed
13	missed	✓	partly
14	✓	incompatible	✓
15	missed	incompatible	missed

On the contrary, Rust's compiler is built with memory safety guarantees built-in at the sacrifice of longer compile times. Table 1.2 lists the memory bug type and when Rust handles them.

Table 1.2: Rust's handling of memory bugs.

Bug type	Rust (release build)
Out-of-bounds R/W	Run time
Null dereference	Run time
Type confusion	Run time
Integer Overflow	Run time
Use-after-free	Compile time
Double free	Compile time
Invalid stack R/W	Compile time
Uninitialized memory	Compile time
Data race	Compile time

Overall, Noseda *et al.* concluded that static and dynamic analysis tools provide inadequate protection where Rust excels with memory safety without loss of performance. Their final recommendation is to consider switching to Rust or adding it to existing C code bases for its ergonomics, safety, and performance. Uzlu and Saykol explore utilizing Rust programming concepts for the Internet of Things and compare them to other languages such as C, Haskell, Python, and more [10]. Their findings formed their opinion that Rust is capable of being the main language for IoT systems programming using memory safety and compile time abstractions for maximum efficiency on constrained devices.

Based on research from Ballal *et al.* [11], most current IoT devices function over Wi-Fi or a Local Area Network (LAN) which have been considered superior choices for stationary devices. LPWAN technologies such as LoRaWAN and Sigfox have better coverage and lower power consumption than Wi-Fi; however, these technologies operate on unlicensed frequency bands. Cellular IoT technologies have lower latency and better coverage then LoRaWAN and Sigfox. They also allow for devices to roam, adding mobility. In comparing the cellular technologies, Ballal *et al.* collected and reviewed end-to-end delays, bit-rates, and packet drop metrics for both NB-IoT and LTE-M networks in three different countries. The results are shown in Tables 1.3 and Table 1.4.

Table 1.3: Cellular IoT throughput and latency field results.

Technology	Median Bitrate (Kbps)			Median end-to-end Latency (ms)		
	Denmark	Norway	Sweden	Denmark	Norway	Sweden
NB-IoT	2.102	8.9805	8.7515	1905.004	314.77	373.571
LTE-M	8.5665	12.8055	12.8115	220.326	220.623	219.795

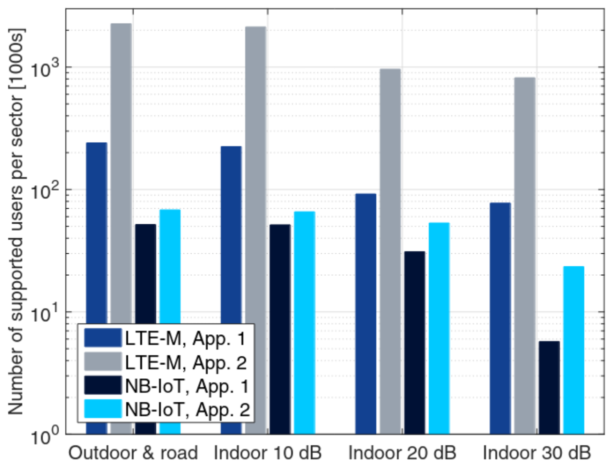
Table 1.4: Number of dropped packets.

Technology	Number of Packet Drops			Number of Packets Sent
	Denmark (Home)	Norway (Roaming)	Sweden (Roaming)	
NB-IoT	180	31	64	1000
LTE-M	36	40	16	1000

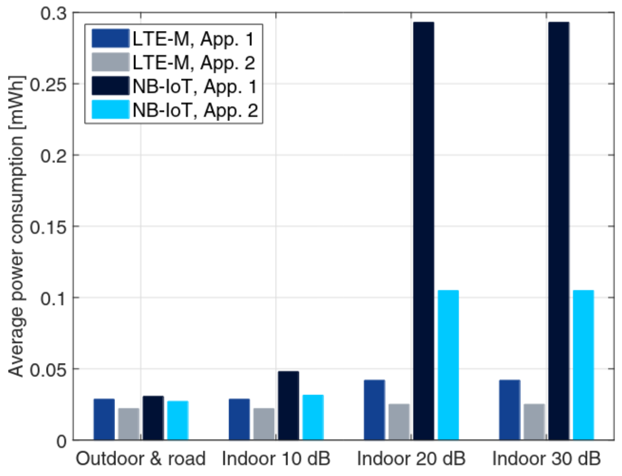
According to Jubin *et al.* [12], many IoT applications do not need high speeds or throughput, rather they require long range and low power with low data rates and bandwidth. In efforts to meet these requirements, unlicensed-spectrum low-power wide area networks (LPWANs) such as LoRa and SigFox were created. These led the way for licensed technologies such as LTE-M and NB-IoT. LoRa, an amalgamation of the words *long* and *range*, is a proprietary LPWAN technology based on chirp spread spectrum (CSS)[13] radio modulation. LoRa describes the physical layer, or the chip itself, LoRaWAN is the media access control layer (MAC) protocol. Although LoRaWAN is an open standard, the LoRa chips themselves are only available from Semtech [14]. LoRaWAN supports both public and private network structures and is widely available with commercially available end devices [15]. SigFox is an LPWAN technology which uses Ultra Narrow Band (UNB) modulation, which uses a slow modulation rate to achieve longer range. SigFox is best for applications that send small payloads infrequently. SigFox has limited downlink capabilities and is susceptible to signal interference. SigFox uses a one-hop star topology which needs a mobile operator to carry the data traffic [16].

Jubin *et al.*'s findings revealed that SigFox had the best penetration but was limited to 12-byte payloads and a maximum of 140 messages per day. LoRaWAN was able to support up to 256-byte payloads but showed less tolerance for interference and coexistence issues.

According to Lauridsen *et al.* [17], their studies on coverage and capacity of LTE-M and NB-IoT in rural areas revealed that LTE-M can provide coverage to 99.9% of outdoor devices and indoor devices with less than 10 dB of additional signal loss. For deep indoor users NB-IoT can provide coverage for 95% of users. LTE-M, by definition, has a Maximum Coupling Loss (MCL) of 156 dB at 1 Mbps in a 1.4 MHz bandwidth. NB-IoT supports a MCL of up to 164 dB at 100 kbps in a 200 kHz bandwidth. When experiencing 30 dB of additional losses in signal strength due to deep-indoor or basement positioning, LTE-M can only provide coverage for 80% of devices. Figure 1.5 revealed NB-IoT as supporting less users per sector and consumed much more power than LTE-M.



(a) Number of supported users per sector in rural area.



(b) Average device power consumption per day with MCL above 150 dB.

Figure 1.5: Comparison of NB-IoT and LTE-M device support and power consumption in rural areas [17].

Extending battery life is essential for the IoT as many devices will be deployed in hard-to-access locations. Some IoT applications require massive deployment, as in more than 100,000 per cell, and replacing the batteries on that scale is costly, cumbersome, and impractical in many cases. Research by Sorensen *et al.* validates battery lifetime estimation models for both NB-IoT and LTE-M technologies [18]. Their findings in Figure 1.6 illustrates that the U-Blox EVK-N211 had the least amount of power consumption over different transmission power levels while using the NB-IoT network compared to the U-Blox EVK-R410M device.

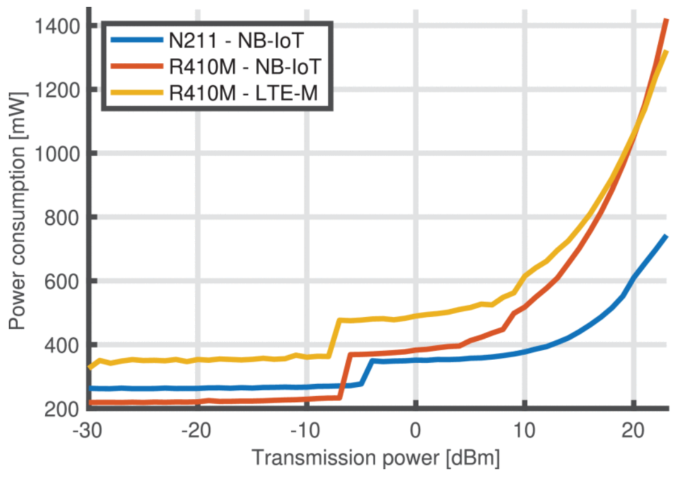


Figure 1.6: Measured power consumption of N211 and R410M as a function of transmission power [18].

El Sousssi *et al.* determined that both NB-IoT and LTE-M can achieve an 8-year battery life in the context of a smart city with poor coverage and a reporting interval of one day. Overall, power consumption of an IoT device can vary greatly based on deployment area and hardware used.

A lot of attention has been given in the past to battery life management for IoT applications; however, considerations for newer technologies, such as lithium-ion capacitors (LIC), should also be explored. Soltani and Beheshti [19] provides a comprehensive review of LICs according to their development, modeling, thermal management, and applications. It is stated that the LIC has a longer cycle life than lithium batteries and high power density of approximately 10kW/kg. LICs are considered to have a lifespan of more than 10 years, but this can be affected by time the cell is not in use (calendar aging) and depth of discharge (DoD), as well as other factors like temperature as shown in Figure 1.7.

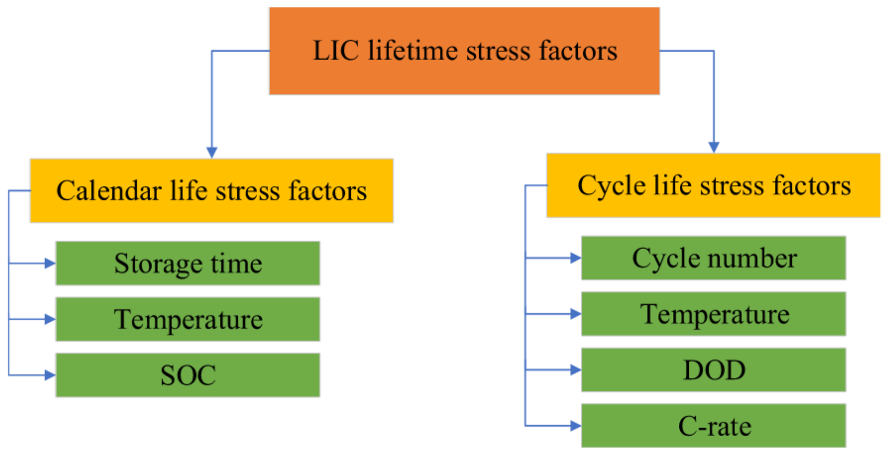

LIC lifetime stress factors
Calendar life stress factors
Storage time
Temperature
SOC
Cycle life stress factors
Cycle number
Temperature
DOD
C-rate

Figure 1.7: LIC lifetime stress factors [19].

LIC stress factors result in the reduction in capacitance and an increase in equivalent series resistance (ESR). These are typically used to indicate the end-of-life for an LIC, which commonly include a 100% increase in ESR or a 20% reduction in capacity. Overall, LICs have good specific power and greater energy density than super-capacitors, and with a longer cycle life in a wide temperature range compared to LIBs.

Work by Damslora [20], explored the use of the nRF9160 cellular modem for a smart meter sensor network with requirements of sending data 24 times per minute using the MQTT protocol and LTE-M. Although, Damslora discusses the importance of power optimization, it is not addressed in their research. The following results, in Tables 1.5 and 1.6, were produced in their experiments using the Nordic Semiconductor nRF9160 development kit while being supplied 4.92 Volts. The development kit also utilized the Zephyr real-time operating system (RTOS) for programming [21].

Table 1.5: Power consumption versus payload size test results.

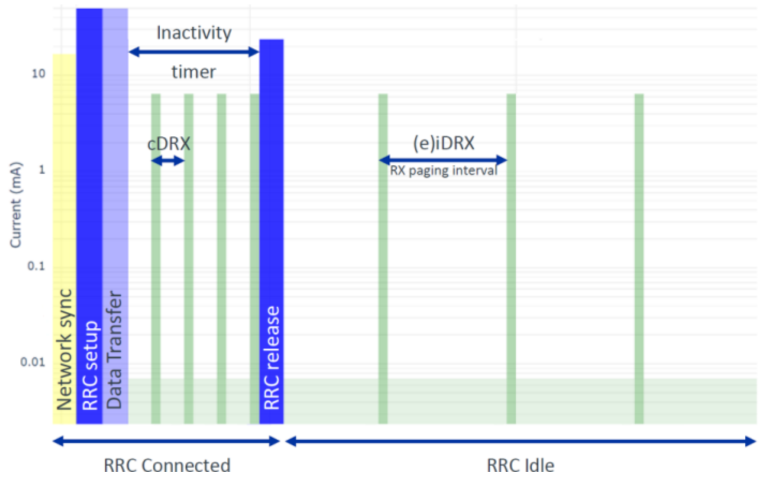
Payload size (bytes)	Current consumption (mA)	Transmission success
0	13.96	Yes
4	15.99	Yes
16	16.76	Yes
64	16.55	Yes
512	16.90	Yes
4096	18.62	Yes

Table 1.6: Signal strength variation by location and network.

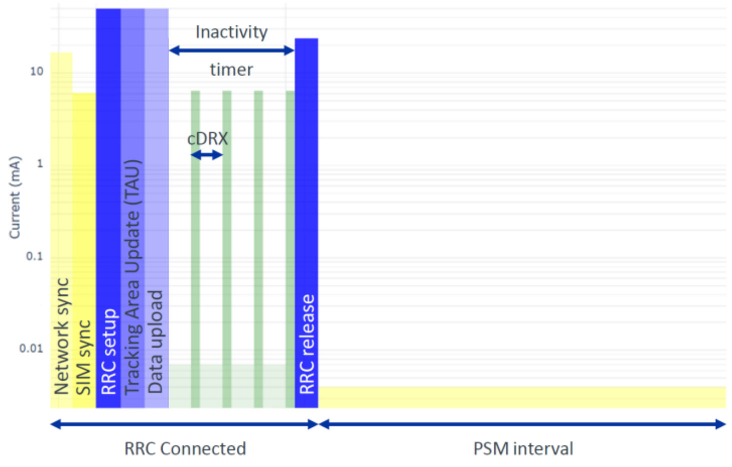
Location	Signal Strength, LTE-M (dBm)	Signal Strength, NB-IoT (dBm)
Third floor	-86	-80
Ground floor	-93	-85
Sub-basement	-114	-105

Research by Vishnubhatla [22], investigates the performance proposition of the nRF9160's coverage, throughput, latency, capacity, and power efficiency. Vishnubhatla proposes a smart-city IoT solution using MQTT, LTE-M, the Zephyr RTOS, and Google Cloud with the nRF9160 to monitor air quality, rainfall, traffic, and more. The conclusion is that the nRF9160 has been extensively tested with leading customers and is a complete solution for integration of cellular IoT devices.

Gabelle provides a comprehensive study of NB-IoT power save modes, which also can be used for LTE-M, using the Zephyr RTOS and nRF9160 cellular modem [23]. The study analyzed energy performance based on payload size, extended discontinuous reception (eDRX), tracking area updates (TAU), power saving modes (PSM), and more. All results were generated using the nRF9160 development kit and an Otii Arc power analyzer from Qoitech [24]. Figure 1.8 compares the estimated current consumption of the nRF9160 modem when utilizing eDRX and PSM modes during connection and idle states.



(a) nRF9160 Extended discontinuous reception.



(b) nRF9160 Power Save Mode.

Figure 1.8: Estimated modem current in connected and idle states.

Gabelle's conclusion showed increasing battery life can be accomplished by minimizing the following parameters:

1. payload size;

2. upload frequency;
3. paging frequency;
4. paging duration;
5. tracking area update frequency;
6. active time duration.

The absolute performance of the nRF9160 is not determined by Gabelle's research, but instead presents a comparative study of NB-IoT power saving techniques.

1.3 Methodology and Contributions

The following steps are used to develop a working prototype that can monitor the fuel level of a single propane tank, or even a fleet of tanks, maintenance free for a targeted 10-year lifespan:

1. Compare and select a low-power wide-area network technology or Wi-Fi;
2. Choose a suitable transport and application protocol with security and power efficiency in mind;
3. Compare available hardware development kits based on communication capabilities and power options;
4. Explore the validity of Rust firmware in IoT applications;
5. Provide an open-source, hall-effect sensor module for R3D type gauges to accurately read fuel levels;
6. Consider LIC energy storage as a sufficient power source for sensor's operational lifetime using solar charging;
7. Present energy consumption field measurements using a power analyzer;

8. Determine the reliability of communications in areas of varying network receptions.

The completion of these objectives will demonstrate the possibility of battery-less IoT devices using a next-generation, embedded programming language called Rust. As a result, the foundation for further cellular IoT research and long-term validation is laid.

CHAPTER 2: NETWORKING STACK FOR IOT

The main feature of an IoT device is its network connectivity. Internet-of-Things is a description that contains many different types of devices. They can be anything from remote sensors to kitchen appliances; however, they all share the commonality of being able to share data using a networking stack. Networks are built as a *stack* of technologies working on top of each other defining the entire communication model. The Open Systems Interconnection (OSI) model consists of 7 layers to describe a full network stack, where the TCP/IP model simplifies it to 4 layers as shown in Figure 2.1.

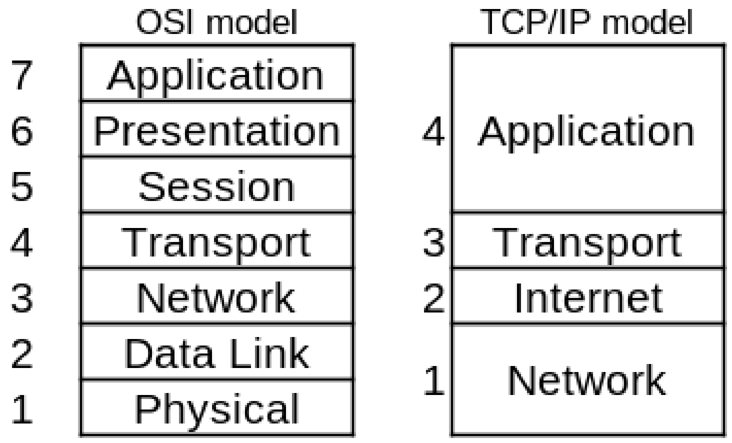


Figure 2.1: OSI and TCP/IP networking layer models.

When choosing protocols for an IoT networking stack, there are several factors that must be considered, such as what protocols are available, are the devices stationary or mobile, what is the power budget, what does the hardware support, and ow much does the hardware cost? With this in mind, the first communication layer to select is the network layer. The two main network protocols for IoT are Wi-Fi and low-power wide-area networks (LPWANs). A LPWAN is not a single technology, but a group of low-power technologies, some of which include SigFox, LoRa [25], and cellular networks such as LTE-M and NB-IoT. Wi-Fi and cellular-based LPWANs are the networks in focus as SigFox and LoRa require proprietary chips and use unlicensed radio frequency (RF) bands making them less flexible and reliable choices.

2.1 Comparing Cellular LPWANs and Wi-Fi Networking

Wi-Fi is a great choice for IoT deployments that require reoccurring, large data transfers as the pricing model is based on data throughput not quantity. For cellular, the 1NCE connectivity platform has a flat rate pricing model set at \$10 for 500 MB of data which is good for 10 years [26]. This flat-rate equates to $\$10/500\ MB = \$0.02/MB$ with a minimum yearly cost of $\$10/10\ years = \$1/year$ minimum if less than 50 MB per year is used. The average Wi-Fi cost in the US is approximately \$36 per month [27]. For a Wi-Fi network that is dedicated to IoT devices, the break-even point for a cellular network would be $\$36\ month/\$0.02\ MB = 1800\ MB/month$ at the lowest network speed. Wi-Fi can offer a lower cost-per-byte with high data throughput and have cheaper hardware, but they can have a higher final cost-per-device since Wi-Fi networks require more maintenance and setup [28]. Wi-Fi IoT fleets may also incur third-party provisioning costs on networks that the fleet operator does not control. If a cellular deployment does end up with a higher cost-per-device, it still can be the best choice as cellular LPWANs come with world-class security and reliability, something that a Wi-Fi IoT fleet would need to provide itself [28]. Part of the extra cost for cellular, is for an expertly managed network since LTE-M and NB-IoT are licensed spectrums [18]. Cellular networks are not typically affected by blackouts and have high reliability, where Wi-Fi can be unavailable during power outages and can suffer from networking issues associated with SSIDs and login credentials. Cellular networks can also handle tower handovers making mobile projects simple. Table 2.1 summarizes the comparisons between Wi-Fi and cellular-based networks for IoT use cases.

Table 2.1: Comparisons between Wi-Fi and cellular-based networks for IoT.

	Cellular	Wi-Fi
Cost per MB	\$0.02 (1NCE SIM)	\$0 (monthly fee)
Modem cost	≈ \$24 (nRF9160)	≈ \$1-\$3 (ESP32)
Reliability and security	High, included	Slightly lower, self-managed
Coverage	Great for mobile and stationary projects	Best for stationary projects, or where cellular does not exist
Data throughput	Kbps range	Mbps range
Deployment	Turn-key	SSID and credentials needed

Overall, the choice between a cellular or Wi-Fi network for IoT is situational. Wi-Fi is a great choice for already in-place Wi-Fi networks and those that focus on stationary devices. Wi-Fi is also a great choice for consumer-level devices where the end-user already has a Wi-Fi network. Cellular networks can work well for both consumer and commercial machine-to-machine (M2M) networks with its turn-key setup, mobility, and built-in security and reliability features. For these reasons, a cellular-based LPWAN is chosen for the networking layer of the communication stack.

2.2 LTE-M and NB-IoT LWANs

Cellular networks typically support two different standards for IoT devices: LTE-M and NB-IoT. LTE-M, also known as Cat-M1, is designed for low-power applications with medium throughput capabilities. Compared to an LTE network, as used by cellular phones, LTE-M has a much narrower bandwidth of 1.4 MHz versus 20 MHz for LTE. The narrower bandwidth limits the amount of throughput but increases the device's range. Secure end-to-end communication, such as TCP/TLS, is suitable over LTE-M networks and mobility is fully supported using the same tower transition protocols as LTE. LTE-M also provides latency in the millisecond range which is important for time-critical applications where real-time communication is needed. NB-IoT, also known as Cat-NB1, operates with a much narrower bandwidth than LTE-M and does not use an LTE physical layer. With a bandwidth of 200 kHz, NB-IoT has less throughput capabilities than LTE-M but supports greater range and signal penetration. NB-IoT is more suited for applications that are static with smaller data payloads and require longer range. A summary of the comparison between LTE-M and NB-Iot networks is shown in Table 2.2.

Table 2.2: Comparisons between cellular supported IoT standards. DL, UL represent downlink and uplink throughput [29].

	LTE-M	NB-IoT
Also known as	eMTC, LTE Cat-M1	LTE Cat-NB1, LTE Cat-NB2
Max throughput (DL/UL)	300/375 kbps	30/60 kbps, 127/169 kbps
Range	Up to 11 km	Up to 15 km
Latency	50-100 ms	1.5-10 s
Mobility/cell re-selection	Yes	Limited
Roaming	Supported	Limited
Deployment density	Up to 50,000 per cell	Up to 50,000 per cell
Battery life	Up to 12 years	Up to 12 years

LTE-M is the networking standard of choice for its mobility capabilities and faster connection speed, which will reduce power consumption. Since cellular LTE-M networking is used, the internet layer will be handled by the subscriber identity module (SIM) card.

2.3 Transport and Application Layers

An LTE-M connection is not the only thing needed to transmit data. Protocols for the transport and application layers are also required and are selected together as they are tightly coupled. The two transport layers most widely available are Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). TCP is connection-oriented, designed to ensure successful delivery of data packets across the network and uses a three-way handshake. If the connection times out before the data confirmation handshake is completed, the transmission is attempted again. In an IoT application, TCP can be sensitive to delays and packet losses during data transmissions [30]. A common application layer using TCP, is Message Queuing Telemetry Transport (MQTT). MQTT offers a Quality of Service (QoS) setting to control to level of confidence provided with data delivery. The QoS settings include:

1. QoS0, the message is sent at most once and does provide a delivery guarantee;
2. QoS1, the message is sent at least once to guarantee delivery;
3. QoS2, the message is sent exactly once using a 4-way handshake [31].

The QoS level setting for MQTT can provide a more lightweight option than a TCP socket alone; however, MQTT still requires an always-on connection making it less power efficient.

UDP is message-oriented, designed for real-time communication and does not verify the delivery of a data packet where packets can also arrive in the wrong order. Since UDP is connection-less, it can be said that UDP has less overhead than TCP. A popular application layer protocol for UDP is the Constrained Application Protocol (CoAP). CoAP is connection-less, so two devices can connect without any prior arrangements making it more power efficient than MQTT. CoAP does not have a controllable QoS, instead the message transfer can be set as confirmable or non-confirmable. A confirmable CoAP transfer will be more reliable as the server will reply with a receive acknowledgement. A non-confirmable transfer maintains the UDP connection-less model [32]. Figure 2.2 illustrates the architecture differences between the two protocols.

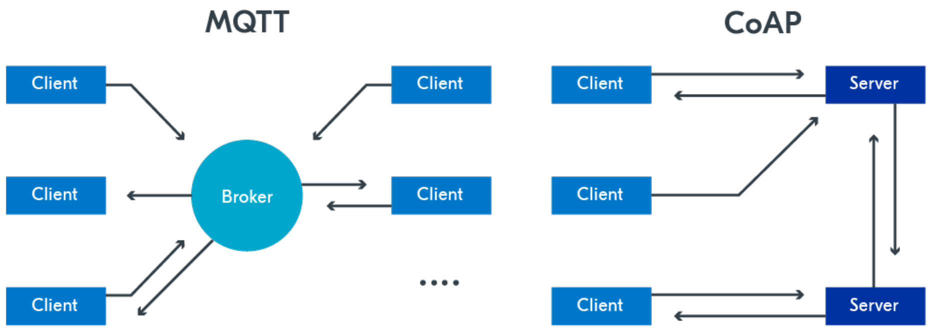


Figure 2.2: MQTT uses a publish/subscribe model, CoAP a request/response model (Image Courtesy: Nordic Semiconductor).

Since CoAP does not need to keep a connection alive, it is typically better for low-power applications, and is used for this reason. CoAP has also been shown to be more efficient with less latency specifically over cellular networks [33].

Communication security must also be considered for all IoT deployments, no matter the size. The connection made to the cloud should be encrypted and authenticated to prevent device impersonation and eavesdropping. A secure connection can be created over the UDP transport with Datagram Transport Layer Security (DTLS). Various credential types and extensions can be used with DTLS. Pre-shared Key (PSK) based authentication is superior to certificate-based authentication from a performance and low-power stand point [34]. A PSK-ID and PSK are assigned to every device that will connect to the server, these act as credentials for validating the connection. The PSK-ID acts as a username where the PSK acts like a password.

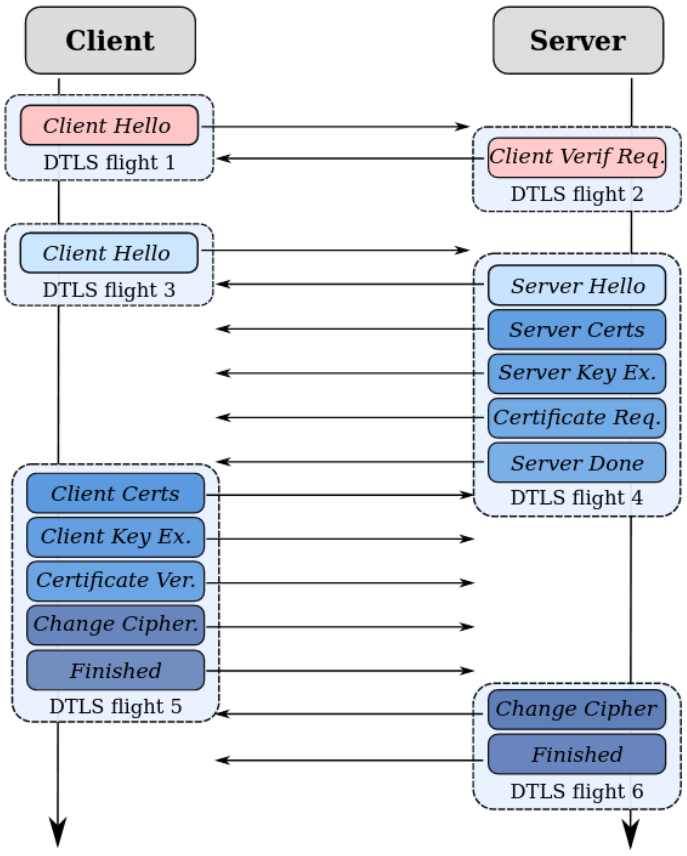


Figure 2.3: DTLS handshake process between a device and server [35].

DTLS will add more overhead than a raw UDP socket and re-introduces a handshake process similar to what TCP had, Figure 2.3, but the added encryption, authentication, and reliability is well worth the cost. Figure 2.4 shows the complete networking stack for communication.

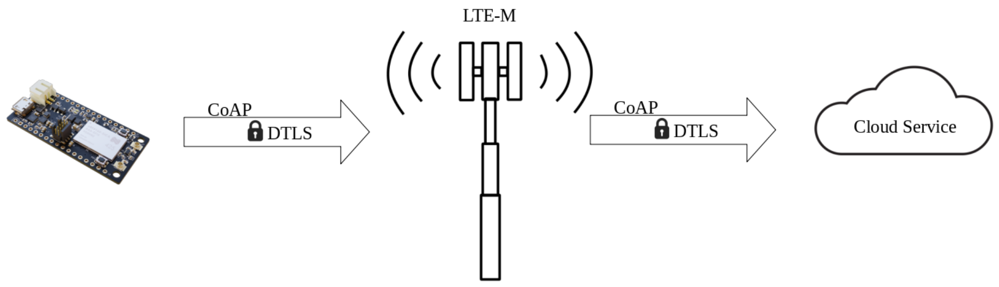


Figure 2.4: Illustration of the networking stack: CoAP, DTLS, and LTE-M layers.

CHAPTER 3: HARDWARE SELECTION

To use the chosen networking stack protocols, hardware must be selected that supports those protocols. The network protocol is the most important factor when choosing hardware since the physical radio must be designed to support a specific network standard, in this case, LTE-M. Most, if not all, micro-controllers can support the transport and application layers since they are software defined, their only limitation being available libraries in the programming language of choice. Of course, libraries are not required, but not having to reinvent the wheel is usually the goal when programming. When selecting the LTE-M hardware, we are specifically looking for a development kit since custom designed hardware is out of scope. The following development kits support LTE-M and are readily available:

1. Particle Boron (U-Blox SARA R510 LTE-M modem) [36];
2. Conexio Stratus (nRF9160 LTE-M/NB-IoT modem) [37];
3. Actinius Icarus (nRF9160 LTE-M/NB-IoT modem) [38];
4. Nordic Thingy:91 (nRF9160 LTE-M/NB-IoT modem) [39].

3.1 Hardware Comparisons

The nRF9160 system-in-package (SiP) from Nordic Semiconductors is one of the most prominent cellular modems on the market. Nordic specializes in chips for ultra-low power wireless technology making them a prime candidate for our hardware choice. All the development kits under review are based on the nRF9160 except for the Boron, which uses the U-Blox R510 modem. The R510 is not a SiP since it has a separate nRF52840 system-on-chip (SoC) dedicated for user applications. The nRF9160 SiP includes a Cortex-M33 processor housed inside the same package as the modem. All the dev kits support the required LTE-M network, but it is good to have flexibility and not the bare minimum when testing. The nRF9160 based kits have support for NB-IoT and Global Navigation Satellite System (GNSS) in addition to LTE-M. Another important aspect for hardware choice, is the amount of prepaid cellular data included. The Boron kit comes with their own branded etherSIM supporting a completely free data plan when using under 100 devices with a 100 MB and 100,000 transaction limit per month. For installations requiring more data, or more devices, requires a subscription plan for which the pricing is not published. The Stratus kit comes with an external SIM card prepaid with 500 MB of data that is guaranteed for 10 years. The Icarus kit has an on-board chip SIM (eSIM) which requires an Actinius account and is only valid for 3 months after activation. It also supports an external SIM card to bypass the account setup step, but it is not included with the kit. Finally, the Thingy:91 kit does not include a SIM card, so it must be supplied by the developer. Based on the amount of data, the time limits, and ease of management, the Stratus seems to be the best choice as 500 MB can last the lifetime of most devices without any extra account hassle.

Another important consideration for an IoT device, is battery and power management. All the kits support a power saving mode to reduce power consumption as cellular modems can be power hungry. Since they all have power saving modes, the ability for charging must be considered. Lithium-ion Polymer (LiPo) batteries are a very popular choice for IoT devices, making a battery charging circuit very important. All the development kits include a charging circuit; however, the Stratus kit has energy harvesting capabilities that the others do not making it a much better candidate for solar charging. The last key feature being considered while choosing a development platform is how much support they have for the Rust programming language. The nRF9160 has official support for the Zephyr RTOS, which is written in the C programming language. Nordic chips are well-supported in the embedded Rust community, also having libraries (crates) specifically for using the nRF9160 modem. The U-Blox modem has official support in C++ and can use most Arduino libraries [40]. U-Blox has very limited Rust support, which is early in the development stages and can only issue AT commands to control the modem.

Table 3.1 summarizes key feature comparisons between all of the mentioned development kits.

Table 3.1: Comparisons between LTE-M supported development kits.

	Boron	Stratus	Icarus	Thingy:91
Manufacturer	Particle	Conexio Tech	Actinius	Nordic Semi
Modem	U-Blox SARA R510	Nordic nRF9160	Nordic nRF9160	Nordic nRF9160
Application processor	Cortex-M4F (nRF52840)	Cortex-M33 (nRF9160)	Cortex-M33 (nRF9160)	Cortex-M33 (nRF9160)
NB-IoT support	No	Yes	Yes	Yes
GNSS support	No	Yes	Yes	Yes
DTLS support	Yes	Yes	Yes	Yes
Rust support	Limited	Yes	Yes	Yes
Flash/RAM	1 MB/256 KB	1 MB/256 KB	1 MB/256 KB	1 MB/256 KB
Total pins	28	33	28	8
I/O pins	20	26	21	8
LiPo charging	Yes	Yes	Yes	No
Energy harvester	No	Yes	No	No
Power save mode	Yes	Yes	Yes	Yes
Prepaid data	100 MB per month total	500 MB	10 MB	10 MB
Data validity	N/A	10 years	3 months	N/A
Accelerometer	No	Yes	Yes	Yes
Environmental Sensors	No	Yes	No	No
Kit price	$60	$89	$130	$125

When considering each development board, the Conexio Stratus is selected as all the hardware support needed for the chosen communication stack, can be used with

Rust programming with plenty of no-hassle data, and has the best charging and energy harvesting capabilities.

3.2 Hardware Features

The Stratus kit is equipped with the nRF9160 SiP, as previously mentioned. A high-level block diagram of the nRF9160's features and peripherals are shown in Figure 3.1.

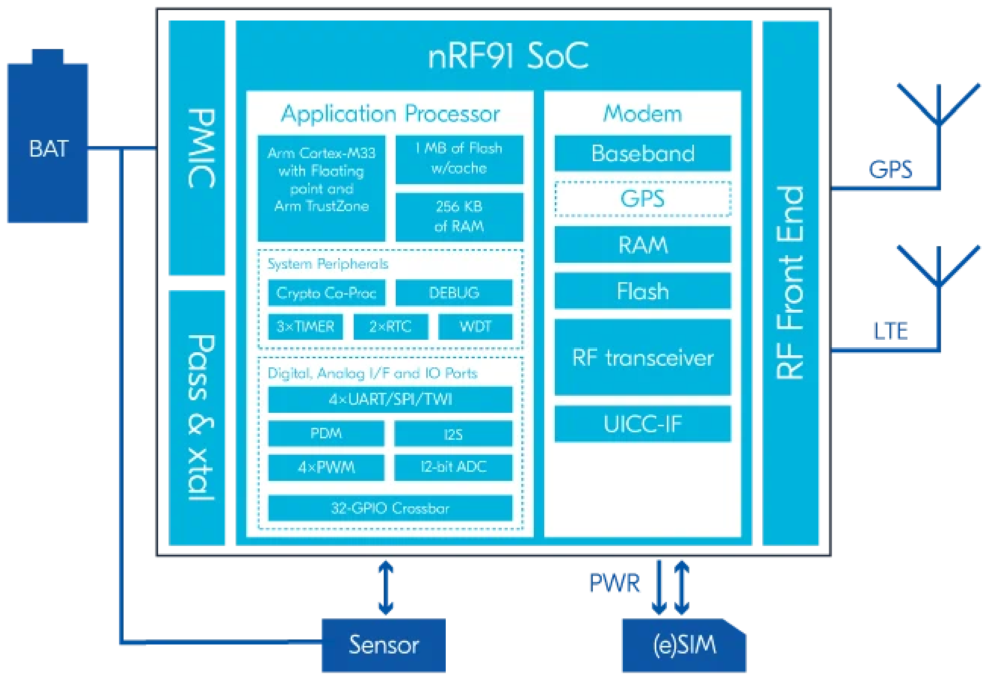


Figure 3.1: nRF9160 SiP block diagram (Image courtesy: Nordic Semiconductors).

The development kit designed around the nRF9160 SiP is shown in Figure 3.2 and illustrates available pins and their functionality.

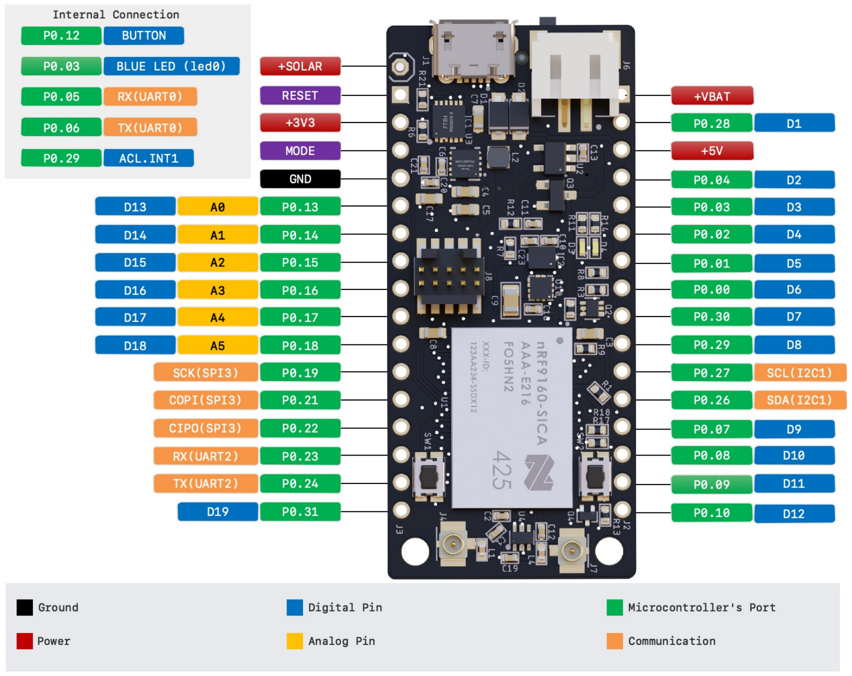


Figure 3.2: Stratus cellular IoT development kit and pinout(Image courtesy: Conexio Technologies).

Although it is not a required feature, the Stratus also has on-board sensors such as a SHT4x temperature and humidity sensor from Sensirion[41], and a ST Microelectronics LIS2DH MEMS digital output, 3-axis accelerometer [42]. These sensors make the kit suitable for weather stations and asset tracking as well. The solar energy harvester is a fantastic addition to the kit and makes it a perfect choice for low-maintenance IoT deployments where changing batteries need to be avoided. The harvesting capability is provided by an e-peas AEM10941 solar energy harvester with maximum power-point tracking [43]. The AEM10941 can charge any type of rechargeable battery or super-capacitor providing excellent power source flexibility. The datasheet is provided in Appendix A. Figure 3.3 displays a high-level operational schematic for the energy harvester.

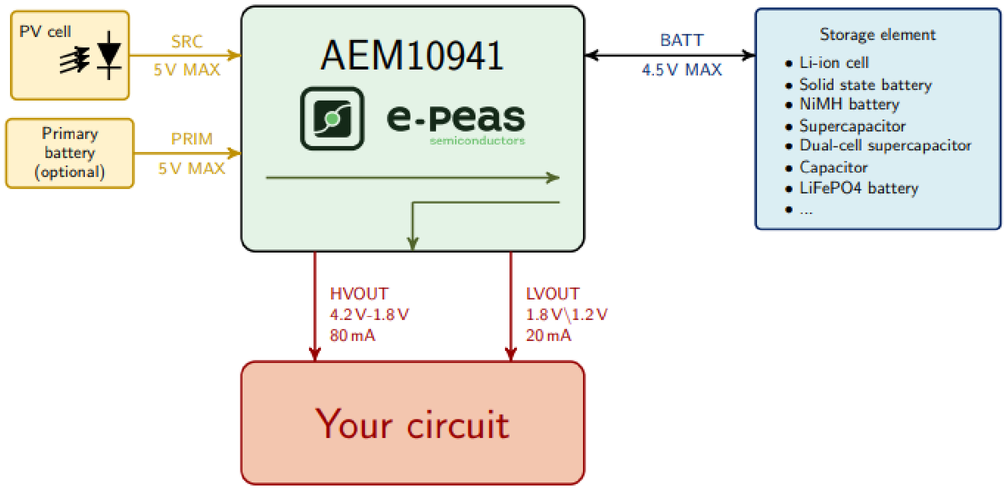


Figure 3.3: Simplified schematic of the AEM10941 energy harvester (Image Courtesy: e-peas).

External antennas are required for both cellular and GNSS (GPS) functionality on the Stratus. Featuring both LTE-M and NB-IoT support, the Laird EFF6989A3S-19MHF1 is used for its small size and wide band-width support range of 698MHz to 6GHz [44]. The antenna is shown in Figure 3.4.

Figure 3.4: Laird external antenna with uFL connector.(Image courtesy: Digikey).

CHAPTER 4: RUST PROGRAMMING

The language is designed to guide you naturally towards reliable code that is efficient in terms of speed and memory usage [45].

The Rust programming language is a systems language built for performance, reliability, and safety. Being a systems level language means that Rust exposes low-level details about memory management, data representation, and concurrency. Rust gives the developer precise control over both the stack and heap memory of an application while being able to provide memory safety guarantees. This is one of the main advantages of Rust over C and C++ frameworks. The C language was developed in the 1970s, well before any concepts of network-connected device security was exploited from memory mishandling [46]. C and C++ were designed for speed and more speed, any notion of safety was added as an afterthought. Rust, on the other hand, was created for speed and safety. Other languages such as Java and Python, are memory safe but they achieve this using garbage collection algorithms to deallocate memory that is no longer needed [47]. This type of memory safety comes at the cost of losing speed and efficiency. Rust has baked memory management into the language using ownership and lifetime rules. These rules are used to determine when memory goes out of scope at which point it is automatically dropped [48]. What makes Rust a good choice for embedded development includes everything previously mentioned, but also because of easy cross-compiling, asynchronous support, and a strong type-system [49]. Rust also has a fantastic package manager, Cargo, which is a cross-compiler by default. Cargo makes it simple to compile an application for an embedded target, control application dependencies, and produce repeatable builds [50]. Rust's *async/await* syntax allows for asynchronous programs to be read as if they were synchronous making them easier to reason about. Asynchronous programming is useful for networking and input/output operations where the CPU can be freed to perform other operations, for example, while it waits for a data packet to arrive [49]. Finally, Rust's strong type-system can be leveraged to prevent hardware misuse, such as trying to configure a digital pin as an analog input. These qualities make Rust a fantastic choice for embedded systems programming and for our device firmware.

4.1 Embedded Rust Firmware

The device firmware used for this research is open-source and available on Github at https://github.com/dkhayes117/propane-monitor-embassy/.

Nordic Semiconductors solely supports an RTOS called Zephyr and is written in C. Nordic's cellular modem library, *nrfxlib*, is also written in C and is closed-sourced [51]. To be able to interact with the nRF9160's modem, Rust must be able to inter-operate with the C programming in their library. This interoperability is achieved through Rust's foreign function interfaces (FFI), otherwise, completely reverse engineering the modem library to Rust would be required. The FFIs are type translation definitions and are used create Rust function wrappers around the C functions. With the FFIs in place, the rest of the firmware can be written in pure Rust.

There are two structures defined, *Payload* and *TankLevel*. The TankLevel structure is used to contain all the data associated with a single measurement operation. TankLevel holds three data measurements: tank and battery levels, and an optional timestamp to be used with GPS synchronization, which has not been implemented. To save power, the firmware is not transmitting every time a measurement cycle is completed. Instead, the Payload structure is used to buffer the measurements where they can be sent all at once. The Payload structure contains fields for holding multiple measurements, the connection strength during transmission, a timeout counter, and a private location field used only to label testing data.

Before the main logic can begin, interrupts EGU1 and IPC are enabled for modem communication purposes, and the universal asynchronous receive/transmit (UART) peripherals are explicitly disabled to save power. Next, the heap data section is initialized for the memory allocator, which is required for the CoAP protocol and for serializing the payload data into JavaScript Object Notation (JSON) format. The following steps creates and initializes everything needed for the firmware's state machine:

1. take ownership of all device peripherals;
2. set pin 29 as disconnected to disable on-board accelerometer for power savings;
3. define pins to control the hall effect sensor, an LED, and the battery measurement circuit;
4. configure the analog-to-digital (ADC) converter for two sampling channels used to measure the tank and battery levels;
5. calibrate the ADC;
6. initialize the cellular modem;
7. install the PSK-ID and PSK in the modem for cloud authentication;
8. construct a Payload instance;
9. create a ticker(timer) to control how often the device does work.

When these steps are completed, the device is ready to begin measurements and is in the Ready state. Figure 4.1 is a finite state machine (FSM) diagram that describes how the firmware should operate.

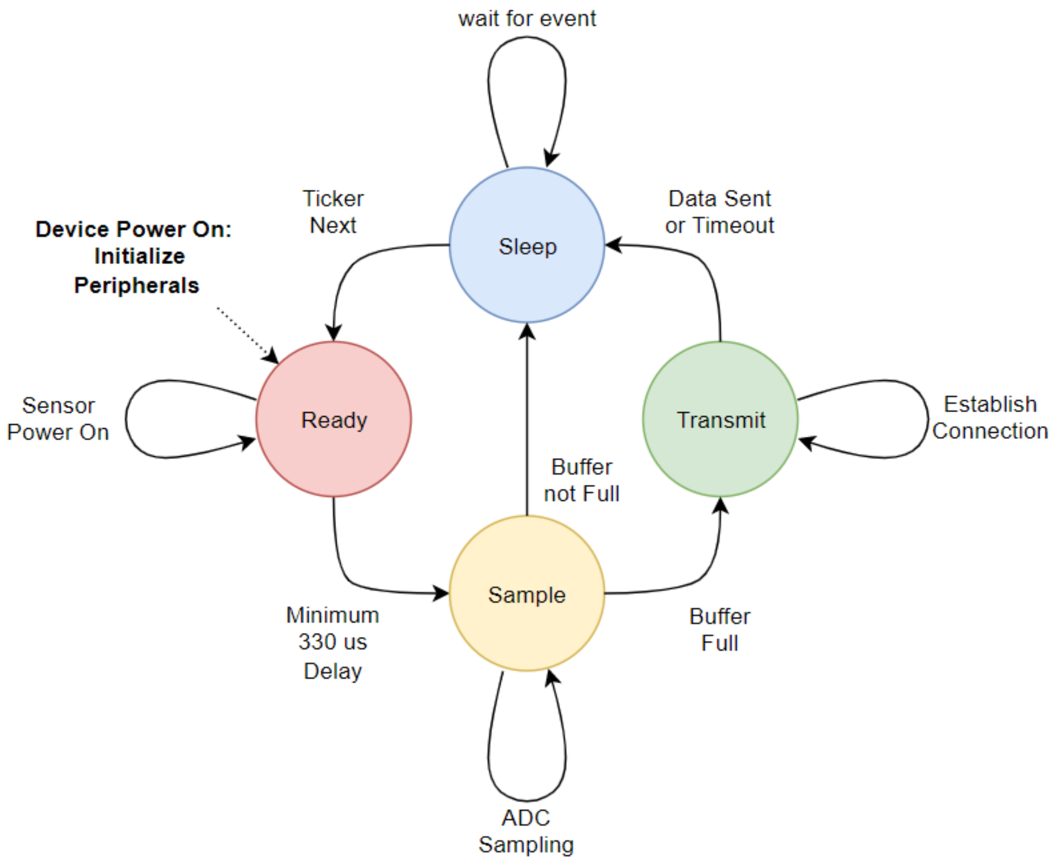


Figure 4.1: State-machine diagram of firmware operational algorithm.

The FSM is inside a loop; therefore, the firmware will not return from this loop unless there is an unrecoverable error. The hall-effect sensor must be powered up and the battery measurement circuit enabled before the ADC can begin to sample. According to the datasheet in Appendix B, the hall-effect sensor can take up to a maximum of 330 µs to be ready for use after power up. A 500 µs delay is used to ensure the sensor has enough to time to stabilize before a measurement is taken. After the delay, we are in the Sample state and the ADC is passed a mutable buffer equal to the number of channels being sampled. The buffer is where the sample data will be stored in the same order that the ADC channels were configured. After completing a sampling cycle, the TankLevel structure containing our data is buffered in the Payload structure. When the Payload buffer is full, the FSM will move to the Transmit state, otherwise it will transition back to the Sleep state and wait for the next ticker event.

The buffer capacity is set to 6 with a 10-minute ticker, which means that the device will wake from sleep and take a measurement every ten minutes and transmit once every hour. If the application sends the measurement data on every cycle, more data will be used and much more power required. The same amount of measurement data is being sent, but the DTLS data packets require header information for the server to authenticate and receive them properly. This means that the header bytes would need to be sent 6 times an hour versus once with the buffered data method. If the payload vector is full, the application transitions into the Transmit state where the cellular modem is used. The transmit state executes the following logic:

1. a DTLS socket is created and connected to the cloud server;
2. the connection signal strength is recorded;
3. a CoAP request is constructed;
4. the payload is serialized in JSON format;
5. the data is sent to the cloud over the socket;
6. the socket is deactivated for low-power sleep.

This data transmission code is wrapped with a 30 second timeout. If signal conditions change or a network issue arises to prevent a socket connection, the timeout will trigger and keep the device from fully discharging the power source. The application is not meant for mission critical purposes; therefore, a timeout event counter is incremented, and the previous data is cleared and the loop continues as if the transmission was successful.

CHAPTER 5: EXPERIMENTAL SETUP

Now that the device firmware is setup to take sensor samples and send data to the cloud over LTE-M, the device needs to be setup for experimental testing. To accurately determine the fuel level, a hall effect sensor is used to detect the gauge's magnetic flux. The sensor needs to be properly positioned on the tank gauge and tested to generate an ADC to percentage conversion equation. Next, a mounting jig is used to attach the gauge mechanism to a servo for easy positioning of the float for data collection. Power source selection and solar considerations are reviewed followed by the details of the power profiling setup. Finally, the method for flashing and debugging the development kit is introduced so the final prototype can be field tested.

5.1 Hall Effect Sensor for Detecting Needle Position

Many commercial, propane tank float mechanisms utilize a magnet to rotate the gauge needle to display the tank's current fill percentage. This allows for the tank to remain sealed eliminating a leak hazard and makes replacing the gauge easy. This design offers the opportunity for detecting the fill-level of the tank by monitoring the magnetic flux induced by the rotating magnet. A fixed-mounted, hall effect sensor can quantify the amount of flux incurred allowing for the conversion into a fill-level percentage to match the shown fill on the physical gauge. An example of a gauge that allows for sensor mounting is the R3D shown in Figure 5.1. The gauge has a black plastic placeholder where a hall effect sensor can be mounted.

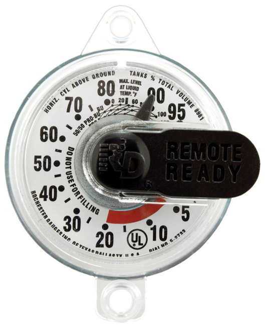


Figure 5.1: Remote ready R3D propane gauge (Image Courtesy: Rochester Sensors).

Distributors of monitoring devices currently on the market, typically do not offer the sale of a mountable hall effect sensor separately from the entire device. The hall effect module that goes with the R3D gauge is only available to original equipment manufacturers (OEM), therefore a custom sensor setup is used [52]. The Texas Instruments DRV5055-Q1 ratiometric, hall effect sensor was the sensor of choice. First, the sensor responds linearly to magnetic flux density allowing for accurate position sensing. The automotive grade sensor also has a temperature range of -40°C to 150°C and implements compensation for magnetic temperature drift, making it perfect for outdoor applications. The DRV5055-Q1 comes in 5 different sensitivity options which vary based on the maximum range of flux density that can be measured in milliteslas (mT), and the millivolt output change per millitesla (mV/mT), or sensitivity. The A4 version of the DRV5055-Q1 was used for initial testing as the float's magnetic flux density is unknown, and the A4 has the greatest sensing range of $\pm$ 169 mT. The first tests using the A4 hall effect sensor is used to determine the best position for the sensor with respect to the gauge. These tests showed the optimal position to be 9 mm from the gauge center as shown in Figure 5.2.

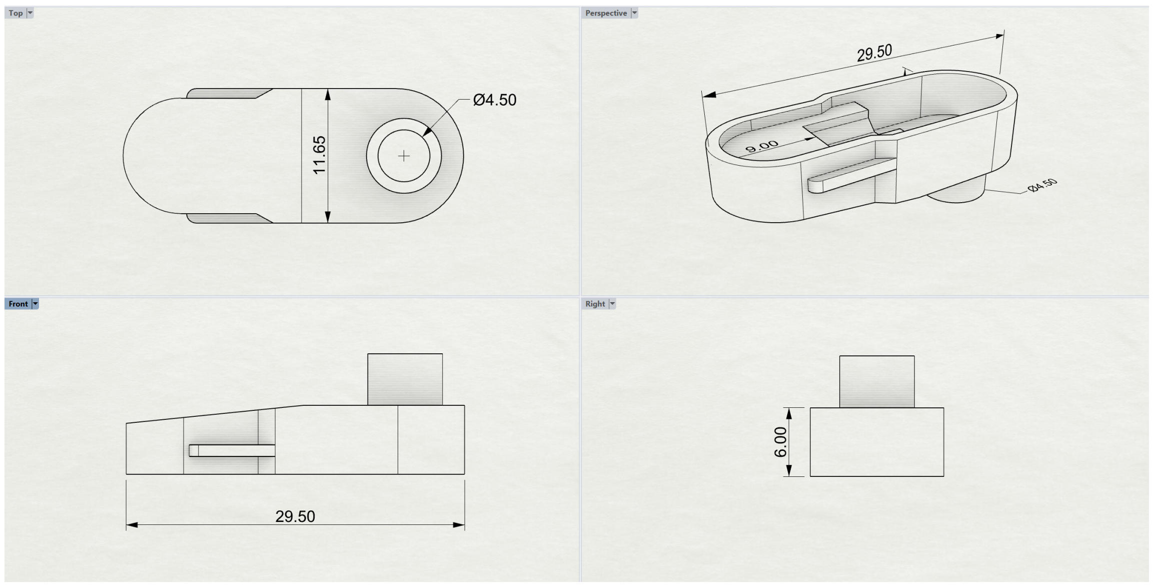


Figure 5.2: Sensor housing 3D model with dimensions.

This placement yields the most voltage output swing by the sensor while keeping a linear relationship between 10% and 95% positions. During these position tests, the magnetic flux extremes was calculated to ensure the proper sensitivity option is selected using the sensor magnetic response equation 5.1.

$$V_{OUT} = V_Q + \beta * (Sensitivity_{(25^{\circ}C)} * (1 + S_{TC} * (T_A - 25^{\circ}C)) \tag{5.1}$$

where V_q is typically half of V_{CC}

Beta is the applied magnetic flux density

$\text{Sensitivity}_{(25^{\circ}\text{C})}$ depends on the device option and V_{CC}

S_{TC} is typically 0.12

T_A is the ambient temperature

Testing at 20°C, 68°F, yielded a maximum and minimum voltage output of 2.839 V and 2.196 V respectively. These voltage output values represent magnetic flux values of 27.133 mT and -24.330 mT; therefore, to maximize voltage swing between these flux values, the A2 sensor version was used with a range of ± 42 mT.

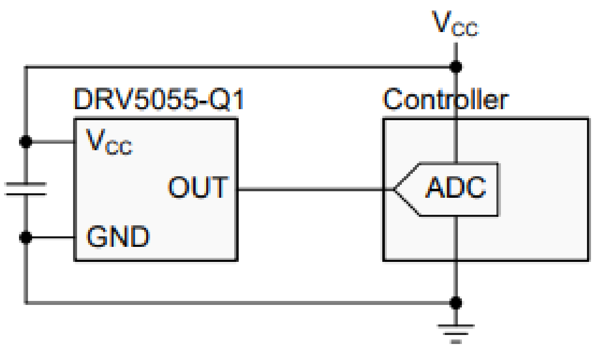


Figure 5.3: Typical sensor schematic.

Figure 5.3 shows the sensor wiring diagram where V_{CC} is 3.3 V and the capacitor value is 20 pF (10 pF minimum). The capacitor is soldered directly to the pins of the TO-92 sensor package along with the sensor wire, then potted with electronics epoxy into the 3D printed housing. The final sensor module is shown in Figure 5.4.

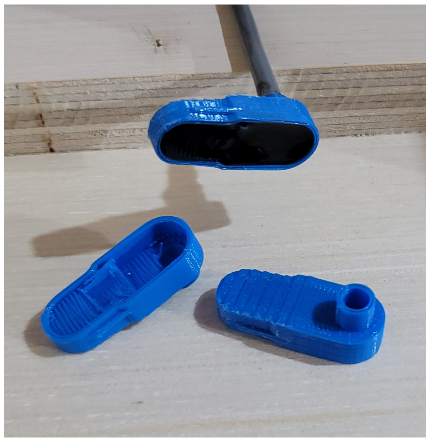


Figure 5.4: Sensor housing with potted sensor electronics.

5.2 Positional Testing Setup

To properly test the sensor measurements, the gauge float assembly is mounted into a fixed position on a plywood substrate as shown in Figure 5.5.

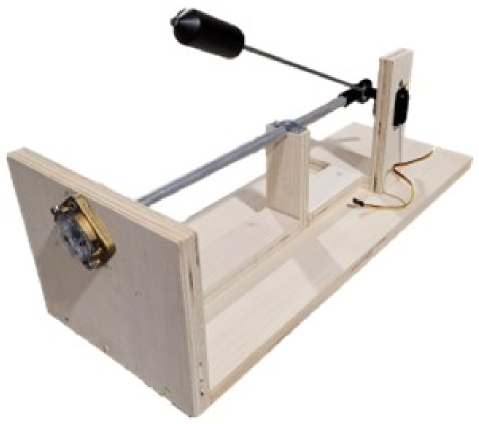
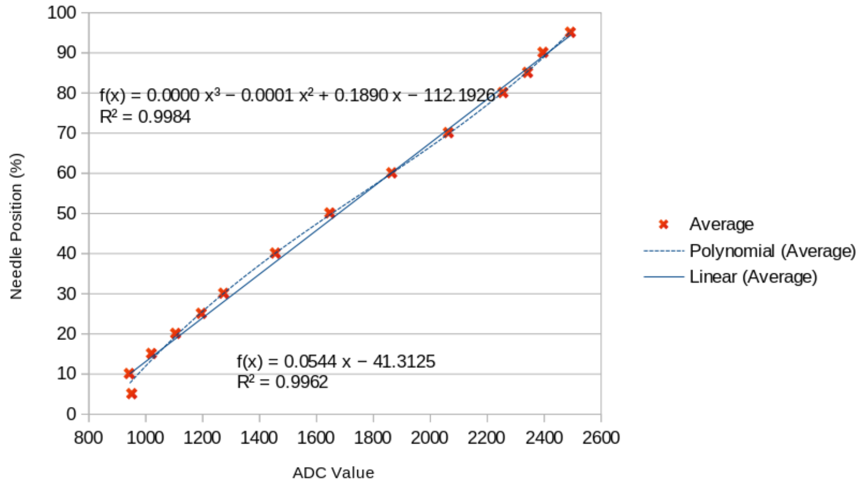
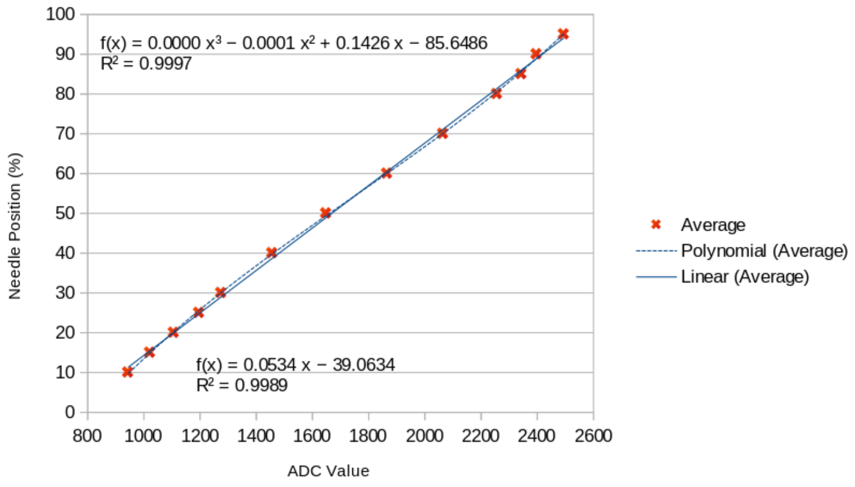


Figure 5.5: Tank level testing jig with servo positioning.

The rotating float is attached to a digital servo, Hitec 645MW, so the position can be accurately and repeatably set. Most servos are controlled with a pulse width modulation (PWM) signal operating at a 50 Hz frequency which has a 20 ms period. The nRF9160's PWM0 peripheral from the Stratus development board is setup to control the float position by controlling the servo. Using the tank level testing jig and servo, 13 fill positions are used between the 5% and 95% points. Each position is measured with an ADC oversampling value of 3, which means that every ADC conversion is taking the average of $2^3(8)$ measurements. Each measurement cycle begins at 5% and ends with 95% and is repeated for 10 cycles. Having the gauge move to each position one at a time instead of taking 10 measurements before moving to the next ensures that the results are repeatable. The results provided as Appendix D, are tabulated and fitted with both polynomial and linear regressions. The data shows that any measurements below 10% are unreliable as the sensor output begins to increase instead of continuing to decrease. If values under 10% are included, then the fit-curve resembles a third-order polynomial more than a straight line. Figure 5.6 illustrates the average ADC value for each needle position recorded, each point represents a ten value average, and show both linear and polynomial regressions. Excluding measurements below 10% yields a much tighter fit and matches the linear equation with less variance.



(a) Sensor data: range 5%-95%; a linear and polynomial fit



(b) Sensor data: range 10%-95%; a linear and polynomial fit

Figure 5.6: Linear and polynomial fitting of ADC measurements to percentage fill level.

The final result is shown in Equation 5.2 and is used in the firmware to convert the ADC measurements into a fill percentage value.

$$Fill_{\%} = 0.0534 * value_{ADC} - 39.0634 \tag{5.2}$$

5.3 Power Source and Solar Charging

Power source selection for a low-power, IoT device is very important for long-term service. Lithium-Ion Batteries (LIB) are very common choices for their high-energy density (Wh/kg) and small footprints; however, they do have disadvantages. For example, LIBs experience performance degradation after a limited amount of discharge/charge cycles, typical battery life is around 500 cycles up to over 4000 cycles with different electrolytes and cathode/anode material combinations [53]. LIBs also have a limited amount of power density and are not the most environmentally friendly product. Supercapacitors, or Electric Double Layer Capacitors (EDLC), have very high-power density (W/kg), but have a much lower energy density compared to LIBs. Lithium-Ion Capacitors (LIC) are a newer technology which has started to emerge to bridge the gap between LIB and EDLC technologies. An LIC is a hybrid electrochemical system combining the functions of LIBs, using the negative graphite electrode, and EDLCs, using the positive carbon electrode, in a single device [54]. Figure 5.7 uses a Ragonne plot to illustrate the relationship of energy density and power density between different energy storage technologies.

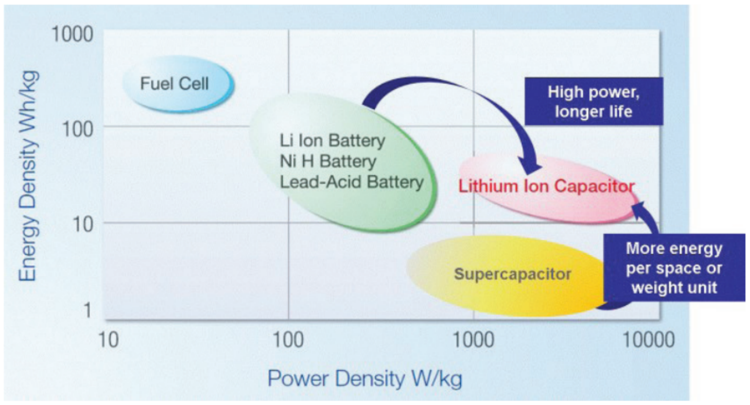


Figure 5.7: Ragonne plot of different energy storage technologies, including LIC characteristics compared to LIB and EDLC [55].

For IoT, the selling point is LIC's long life. In most applications, it will be unnecessary to change the LIC during the life of the device [55]. The Vinatech VEL13353R8257G LIC was selected for its wide temperature range, low self-discharge, and high capacitance rated at 3.8 V and 250 Farads (F). According to the datasheet, Appendix C, the LIC has a rated life of 20,000 cycles and a capacity of 90 milliamp-hours (mAh) which equals $90mAh * 3.8V = 342mWh$ of energy storage. If a device averages 400 µA of current draw at 3.8 V, the LIC would last for approximately $342mWh/(400\mu A * 3.8V) = 225$ hours, or 9.375 days. Even if a full discharge/charge cycle happened once a day, the LIC would only experience 3,650 cycles over a 10-year period, which isn't even 20% of its rated cycle life. LICs are a fantastic choice for IoT devices as they can power the device for its entire lifetime without replacement when combined with a charging system like the solar energy harvester on the Stratus board.

A generic solar panel rated at 700 mW with a 5 V output is connected to the energy harvester. According to the National Renewable Energy Laboratory (NREL), the testing site for the device should get an average of 4.25 to 4.50 kWh/m^2/day as shown in Figure 5.8.

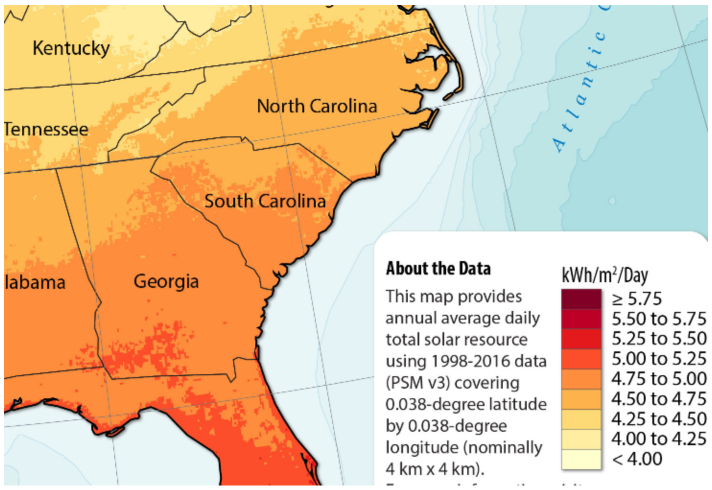


Figure 5.8: Global horizontal solar irradiance (Image Courtesy: National Renewable Energy Laboratory).

The solar panel is approximately 6.25 in^2 or 0.004 m^2, so the panel should be able to generate 17 Wh per day over 4.25 hours for commerical solar panels. This generic solar panel is not commercial grade, so according to the 0.7-watt specification, the panel can only produce 2.975 Wh per day. This means that the solar panel can provide $\frac{2.975Wh}{5V*4.25h} = 140$ mA of charging current. The e-peas solar harvester can charge at a maximum of 80 mA; therefore, the charging system will have an average daily power budget of $80mA * 3.8V * 4.25hours = 1.292$ Wh of total energy. This amount of energy produced is enough to charge the LIC over 3 times its capacity. From this, it can be concluded that the solar charging setup can keep the device fully charged for continual and maintenance free use.

5.4 Power Measurement

The Nordic power profiler kit 2 (PPK2) is used to document the device's power use during testing. The PPK2 device is shown in Figure 5.9.

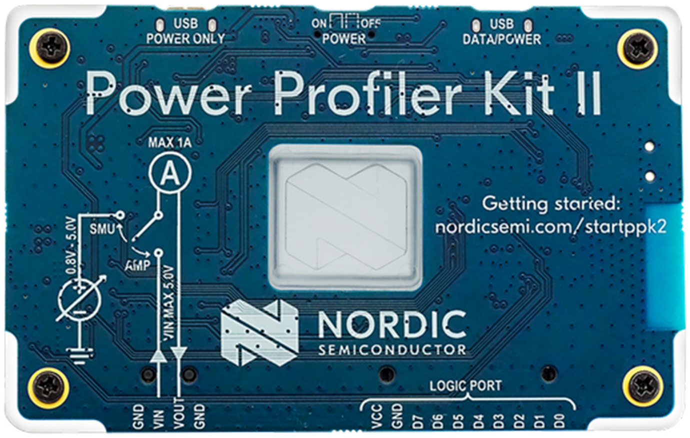


Figure 5.9: Nordic power Profiler Kit Version 2 (Image Courtesy: Nordic Semiconductor) [56].

The PPK2 measures real-time power consumption with up to 100 kilo-samples per second (ksps) and has a measuring range of 500 nA to 1 A. If the device under test requires more than 400 mA the PPK2 needs both usb ports connected to supply the required power. The PPK2 features two measurement modes: source meter mode acts as a power supply and provides the power to the device; ampere meter mode is placed in the current path between the power source and the device. The connection setup for both modes are shown in Figure 5.10.

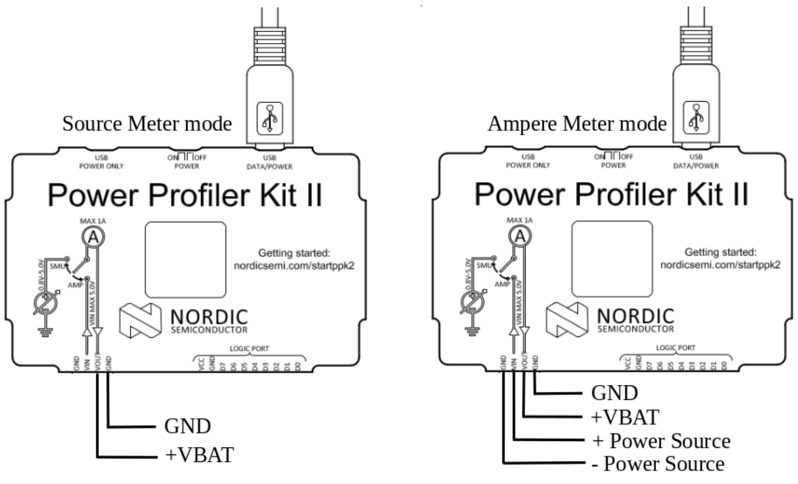
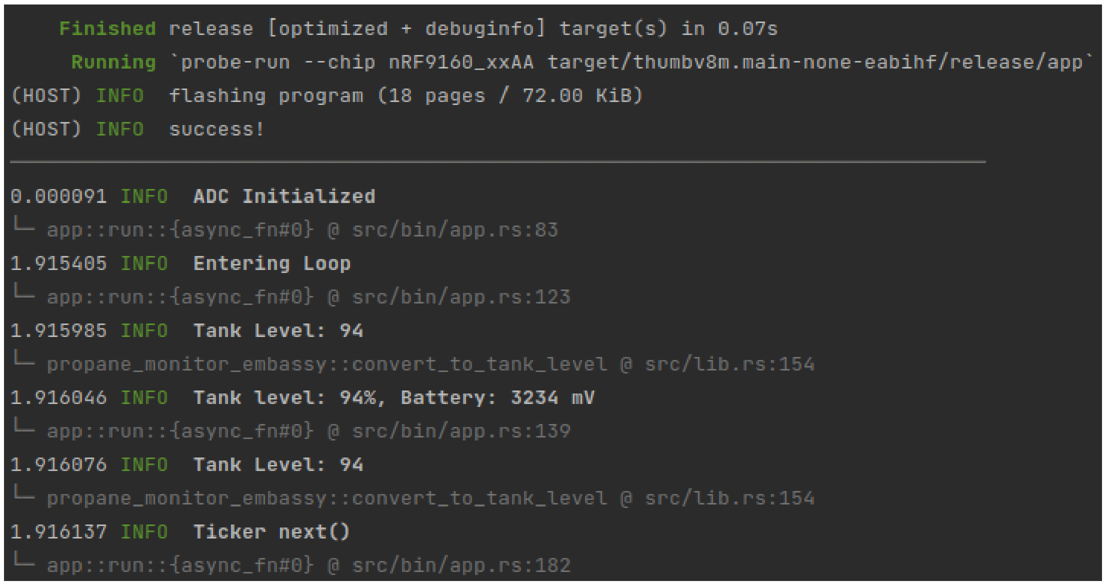


Figure 5.10: Connection diagram for source meter and ampere meter measurement modes.

The PPK2 is used in conjunction with Nordic's Power Profiler software where the mode and voltage output are controlled. The software also charts the power readings while displaying average window current, max window current, window size, and charge.

5.5 Flashing and Debugging

When developing and running firmware a way to flash the executable to the device and debug it is very important. The Probe-Run[57] Rust crate is used as the project runner, which will flash the executable to the device and run it just like a native application would. Probe-Run is built on the Probe-RS library[58] which supports CMSIS-DAP, ST-Link, and Segger J-Link debugger probes. The debugger connects to the device's JTAG pin header where the chip is flashed, and any logs are printed to the console using real-time transfers (RTT). With the development board powered and the debugger connected, running the application with build an executable, flash it to the device, and attach the debugger for RTT logs to be printed to the console. An example of the RTT logs is shown in Figure 5.11.



```
    Finished release [optimized + debuginfo] target(s) in 0.07s
     Running `probe-run --chip nRF9160_xxAA target/thumbv8m.main-none-eabihf/release/app`
(HOST) INFO  flashing program (18 pages / 72.00 KiB)
(HOST) INFO  success!
────────────────────────────────────────────────────────────────────────────────
0.000091 INFO  ADC Initialized
└─ app::run::{async_fn#0} @ src/bin/app.rs:83
1.915405 INFO  Entering Loop
└─ app::run::{async_fn#0} @ src/bin/app.rs:123
1.915985 INFO  Tank Level: 94
└─ propane_monitor_embassy::convert_to_tank_level @ src/lib.rs:154
1.916046 INFO  Tank level: 94%, Battery: 3234 mV
└─ app::run::{async_fn#0} @ src/bin/app.rs:139
1.916076 INFO  Tank Level: 94
└─ propane_monitor_embassy::convert_to_tank_level @ src/lib.rs:154
1.916137 INFO  Ticker next()
└─ app::run::{async_fn#0} @ src/bin/app.rs:182
```

Figure 5.11: Example logs printed to console with Defmt RTT.

CHAPTER 6: RESULTS

All of the hardware components are mounted into an enclosure for testing as shown in Figure 6.1.

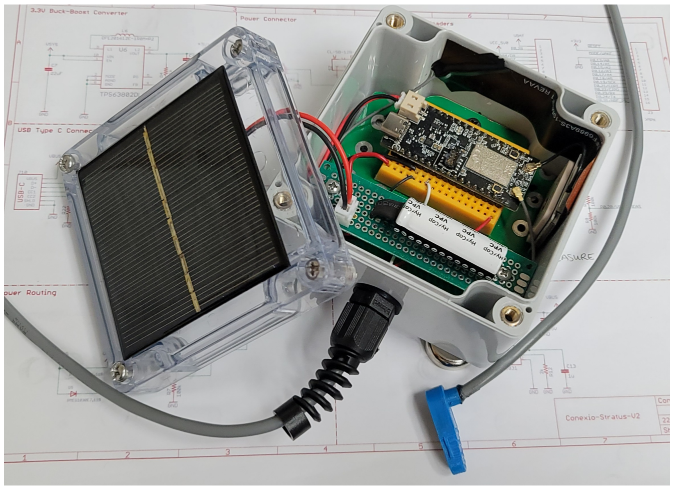
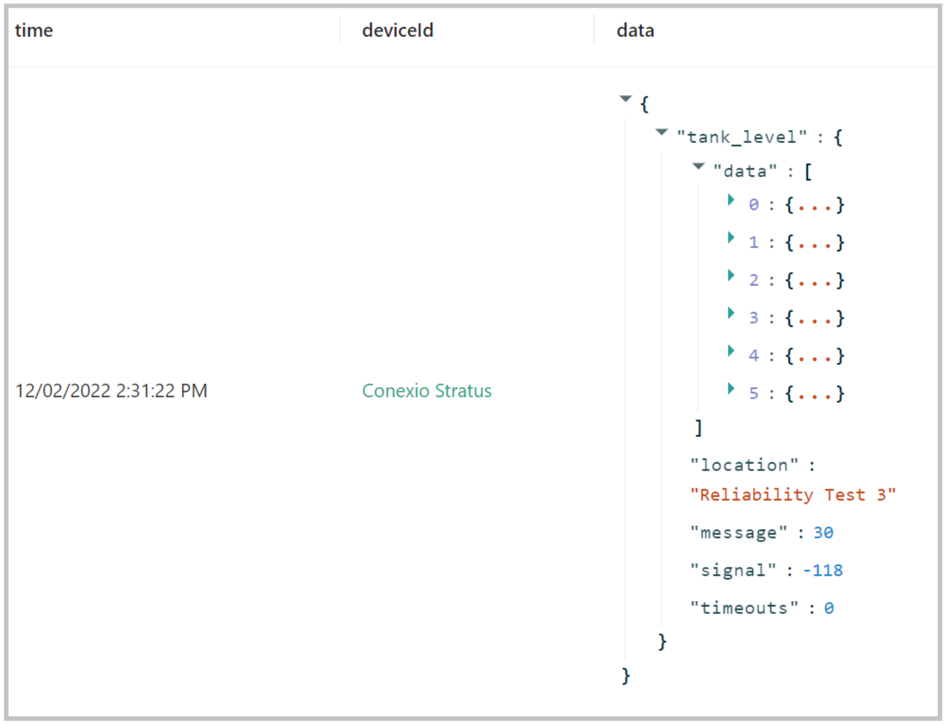


Figure 6.1: Test hardware mounted in enclosure including development board, antennas, LIC, and solar panel.

Any IoT cloud service can be used that supports CoAP and DTLS, in this case, Golioth is used for its simple interfaces [59]. After the device sends a data transmission to the cloud, the data can be viewed on their website. An example is shown in Figure 6.2, displaying a timestamp, the device identification, and the data in JSON format. Seeing the data in the cloud confirms a successful transmission.



time	deviceId	data
12/02/2022 2:31:22 PM	Conexio Stratus	see below

```
{
  "tank_level" : {
    "data" : [
      0 : {...}
      1 : {...}
      2 : {...}
      3 : {...}
      4 : {...}
      5 : {...}
    ]
    "location" :
    "Reliability Test 3"
    "message" : 30
    "signal" : -118
    "timeouts" : 0
  }
}
```

Figure 6.2: Example LightDB log entry after data transfer (Image Courtesy: Golioth IoT).

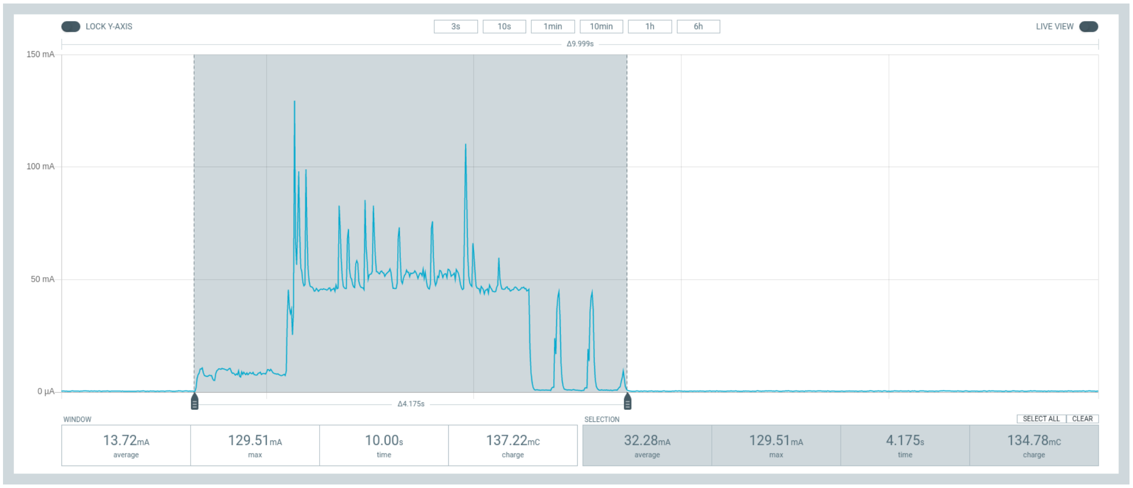
Each transmission with 6 data structures, including the DTLS header data, requires approximately 348 bytes. The 1nce SIM card being used is prepaid with 500 MB of data, therefore $\frac{500MB}{348B} = 1,436,781$ transmissions can be sent. Assuming one transmission per hour, the $10 data plan would last for $\frac{1,436,781}{(24*365)} = 164$ years; however, the data plan is only valid for 10 years. Transmissions could be sent approximately every 3.6 minutes for 10 years before the data plan expires or runs out of data. At that point, the SIM card can be reloaded with another 500 MB of data for $10.

The device uses the most amount of energy during a data transfer since the modem is active. Figure 6.3 shows a power profile during a data transmission with a signal strength of -105 decibel-milliwatts (dBm). The transmission averaged 32.28 mA of current at 3.3 V which is 106.52 mW of power for a total of 4.175 seconds.

$$(32.28mA * 3.30V) = 106.52mW$$

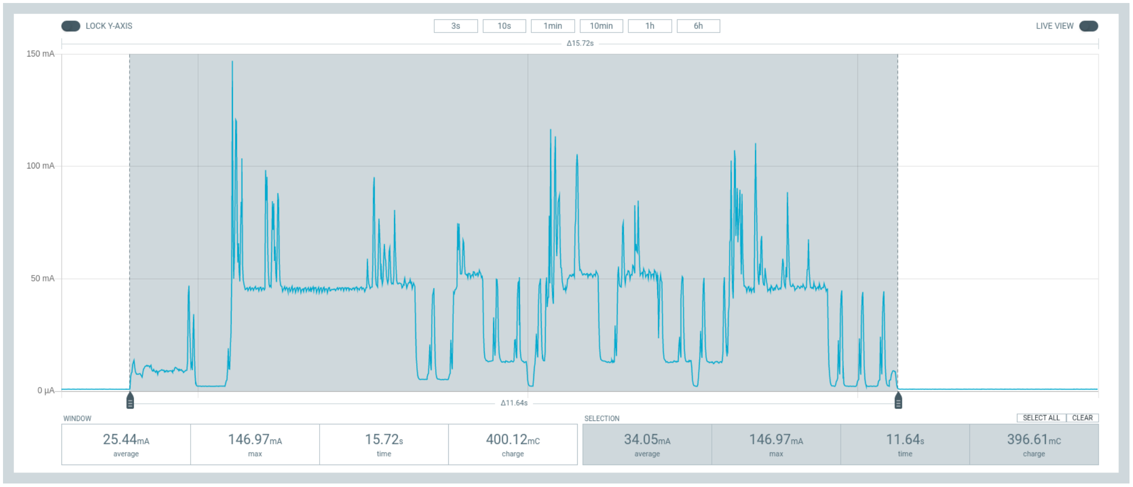
The signal strength affects the total transmission time, the weaker the signal, the longer it takes to send the data which will use more energy. Figure 6.4 shows a transmission, with a signal strength of -126 dBm averaged 34.05 mA at 3.3 V which is 112.37 mW of power for 11.64 seconds.

$$(34.05mA * 3.30V) = 112.37mW$$



WINDOW average	max	time	charge	SELECTION average	max	time	charge
13.72mA	129.51mA	10.00s	137.22mC	32.28mA	129.51mA	4.175s	134.78mC

Figure 6.3: Power profile window during a cellular transmission cycle at -105 dBm.



WINDOW average	max	time	charge	SELECTION average	max	time	charge
25.44mA	146.97mA	15.72s	400.12mC	34.05mA	146.97mA	11.64s	396.61mC

Figure 6.4: Power profile window during a cellular transmission cycle at -126 dBm.

Between the data transmissions, the device is in sleep mode where the lowest achievable current draw is approximately 115µA at 3.3 V with the firmware's current setup. Figure 6.5 displays a 114µA current draw which then spikes to 484µA for 65 ms which represents a sensor sampling operation. During the sample measurement, the device will require 1597.2 µW of power to use the hall effect sensor and 379.5 µW of power while sleeping.

$$(484\mu A * 3.30V) = 1597.2\mu W$$

$$(115\mu A * 3.30V) = 379.5\mu W$$

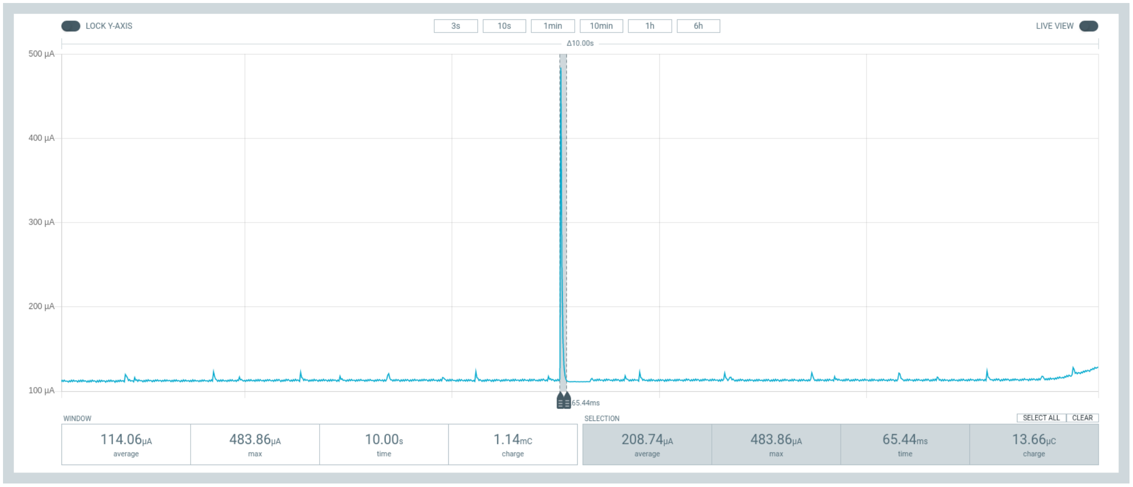


Figure 6.5: Power profile window during a transition from sleeping to sensor sampling.

Throughout testing with different signal strengths between -85 dBm and -130 dBm, the typical transmission time is between 4 seconds and 15 seconds. MOst areas have a strength of -105 dBm or better, so the energy consumption calculations are figured assuming this level of signal. Also, since the modem and sensor are not always drawing current, they must be proportioned according to the time they are used.

$$106.52mW(\frac{4.175s}{3600s}) = 123.53\mu W$$

$$1597.5\mu W(\frac{65ms*6}{3600s}) = 173.1pW$$

$$379.5\mu W(\frac{3600s-(390ms+4.175s)}{3600s} = 379\mu W$$

$$123.53\mu W + 173.1pW + 379\mu W = 502.7\mu W$$

$$\frac{342mWh}{502.7\mu W} = 680.3\ hours$$

The total average instantaneous power the device uses is 502.7 µW. The Vinatech LIC power source has a total energy capacity of 342 mWh, so based on actual measurements, the device can operate for approximately 680.3 hours, or 28.3 days, without any solar charging. The energy consumption is dominated by data transfers. For example, if the signal strength was at a worst case scenario of around -130 dBm, the device would use 3 times as much energy, cutting the operation time to approximately 9 days without any charging. These calculations are assuming a transmission every hour; however, this rate could always be extended if greater periods without charging are required.

The prototype has been theoretically shown to be able to sustain the device for at least 10 years with the solar charged LIC, but reliable communications over the cellular network should be demonstrated. The prototype is relocated to 3 different areas of varying signal strength to provide a field comparison of networking reliability. Figure 6.6 shows the prototype deployed in the field attached to an in-ground, 500 gallon tank.

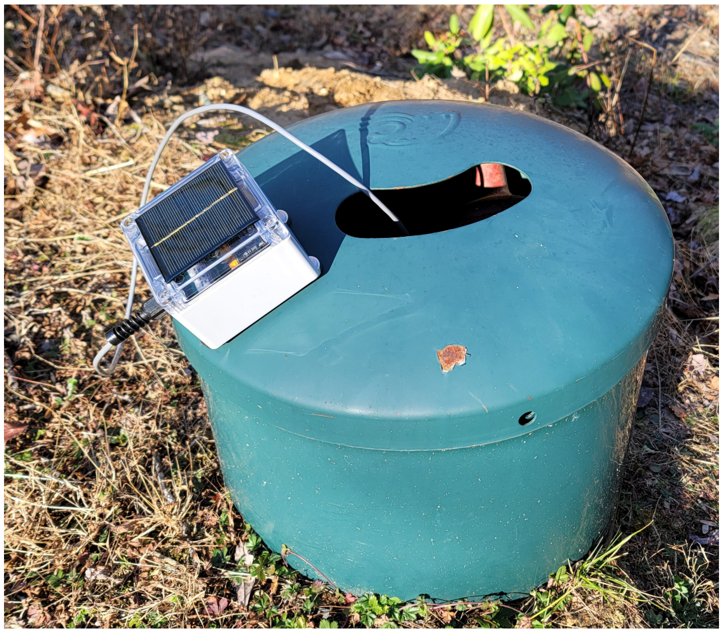


Figure 6.6: Working prototype attached with magnetic feet to an in-ground, 500 gallon tank.

The field reliability results are shown in Table 6.1 demonstrating that LTE-M was able to maintain a success rate of at least 95% in areas of typical signal strengths. Areas with weaker signals, approximately -120 dBm or worse, had more issues with connecting to the network. Note that the firmware implements a 30-second timeout and the modem is completely turned off between transmissions. Keeping the device connected to the cell tower and using eDRX or PSM to conserve power between transmissions, instead of turning the modem off, could increase reliability at weaker signal locations.

Table 6.1: Communication reliability based on signal strength.

	Average signal	Successful transmissions	Timeouts / failures	Success rate
Test 1	-88 dBm	98	2	98 %
Test 2	-104 dBm	95	5	95 %
Test 3	-120 dBm	91	9	91%

CHAPTER 7: CONCLUSIONS AND FUTURE WORK

The purpose of this thesis was to research and validate the use of cellular-based communications with embedded Rust for IoT use-cases, such as a propane fuel level monitoring system. Although, cellular modems are much more expensive than Wi-Fi modems, the turn-key security, reliability, and lack of configuration management makes cellular a viable choice for many IoT projects. Cellular fits well with stationary, mobile, and commercial type applications as demonstrated by the propane tank monitoring sensor application. The system developed could allow propane suppliers to offer tracking and filling services based on tank fill levels for their customer's visibility and satisfaction. Not only were the cellular transmissions reliable with failure rates of less than 5% with typical signal strength, but was also cost effective at only \$0.02 per MB of transferred data. Security was provided with PSK-authenticated, DTLS sockets using Nordic's modem library. It was also demonstrated that powering the device with a solar-charged LIC provided ample power and energy storage and is sustainable for at least 10-years. The device can operate for a minimum of 9 days without any solar charging assistance. The embedded, asynchronous Rust firmware provide to be a viable and flexible framework that offers memory safety guarantees without sacrificing efficiency or speed.

The research presented opens the door for future cellular-based IoT research using a memory-safe programming language in Rust, and the possibility of eliminating batteries in devices. Areas that can benefit from this research includes agriculture, wildlife monitoring cameras, hazardous waste areas, and anywhere maintenance-free operation is a key factor. Specific features that would aid in these areas includes adding over-the-air firmware updates, GPS functionality, and lowering power consumption. Research on the long-term use of lithium-ion capacitors and solar charging for power sources would also be valuable in further validating battery-less IoT devices.

REFERENCES

[1] "Lte fuel level monitor." `https://www.generac.com/specialpages/lte-lp-fuel-level-monitor`. Last accessed 2022-11-14.

[2] "Wireless propane tank level monitor." `https://www.monnit.com/products/sensors/propane-tank-monitoring/lpg/MNS2-9-IN-HE-MG/`. Last accessed 2022-11-14.

[3] Y.-K. Chen, "Challenges and opportunities of internet of things," in *17th Asia and South Pacific Design Automation Conference*, pp. 383–388, 2012.

[4] S. Cass, "The 2015 top ten programming languages," *IEEE Spectrum, July*, vol. 20, 2015.

[5] Y.-P. E. Wang, X. Lin, A. Adhikary, A. Grovlen, Y. Sui, Y. Blankenship, J. Bergman, and H. S. Razaghi, "A primer on 3gpp narrowband internet of things," *IEEE Communications Magazine*, vol. 55, no. 3, pp. 117–123, 2017.

[6] "Ericsson mobility report." `https://www.ericsson.com/en/reports-and-papers/mobility-report/reports`. Last accessed 2022-12-01.

[7] J. F. Peters, M. Baumann, B. Zimmermann, J. Braun, and M. Weil, "The environmental impact of li-ion batteries and the role of key parameters–a review," *Renewable and Sustainable Energy Reviews*, vol. 67, pp. 491–506, 2017.

[8] "Lithium-capacitor." `https://www.vinatech.com/eng/product/lithium-capacitor.phps`. Last accessed 2022-12-01.

[9] M. Noseda, F. Frei, A. Rüst, and S. Künzli, "Rust for secure iot applications: why c is getting rusty," in *Embedded World Conference 2022, Nuremberg, 21-23 June 2022*, WEKA, 2022.

[10] T. Uzlu and E. Şaykol, "On utilizing rust programming language for internet of things," in *2017 9th International Conference on Computational Intelligence and Communication Networks (CICN)*, pp. 93–96, IEEE, 2017.

[11] K. D. Ballal, R. Singh, L. Dittmann, and S. Ruepp, "Experimental evaluation of roaming performance of cellular iot networks," in *2022 Thirteenth International Conference on Ubiquitous and Future Networks (ICUFN)*, pp. 386–391, IEEE, 2022.

[12] J. S. E, A. Sikora, M. Schappacher, and Z. Amjad, "Test and measurement of lpwan and cellular iot networks in a unified testbed," in *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*, vol. 1, pp. 1521–1527, 2019.

[13] B. Reynders and S. Pollin, "Chirp spread spectrum as a modulation technique for long range communication," in *2016 Symposium on Communications and Vehicular Technologies (SCVT)*, pp. 1–5, 2016.

[14] "Semtech." `https://www.semtech.com/lora`. Last accessed 2022-11-18.

[15] "Lora alliance." `https://lora-alliance.org/`. Last accessed 2022-11-18.

[16] "Sigfox." `https://www.sigfox.com/en`. Last accessed 2022-11-19.

[17] M. Lauridsen, I. Z. Kovacs, P. Mogensen, M. Sorensen, and S. Holst, "Coverage and capacity analysis of lte-m and nb-iot in a rural area," in *2016 IEEE 84th Vehicular Technology Conference (VTC-Fall)*, pp. 1–5, 2016.

[18] A. Sorensen, H. Wang, M. J. Remy, N. Kjettrup, R. B. Sorensen, J. J. Nielsen, P. Popovski, and G. C. Madueno, "Modeling and experimental validation for battery lifetime estimation in nb-iot and lte-m," *IEEE Internet of Things Journal*, vol. 9, no. 12, pp. 9804–9819, 2022.

[19] M. Soltani and S. H. Beheshti, "A comprehensive review of lithium ion capacitor: development, modelling, thermal management and applications," *Journal of Energy Storage*, vol. 34, p. 102019, 2021.

[20] B. J. Damslora, "Data collection in a cellular sensor network with nrf9160," Master's thesis, NTNU, 2019.

[21] "The zephyr project." `https://www.zephyrproject.org/`. Last accessed 2022-11-13.

[22] A. Vishnubhatla, "Cellular iot using nrf9160kit," 2020.

[23] F. Gabelle, "Narrowband-iot power saving modes–a comprehensive study," 2021.

[24] "Qoitech." `https://www.qoitech.com/`. Last accessed 2022-12-01.

[25] N. I. Osman and E. B. Abbas, "Simulation and modelling of lora and sigfox low power wide area network technologies," in *2018 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE)*, pp. 1–5, 2018.

[26] "IoT data plan; pricing - 10 dollars per 10 years: 1nce - IoT SIM." `https://1nce.com/en-us/pricing/`, Mar 2022.

[27] "How much does internet cost per month?." `https://www.forbes.com/home-improvement/internet/internet-cost-per-month/`. Last accessed 2022-11-14.

[28] "Cellular vs. wifi for IoT: How to choose the right one." `https://www.particle.io/iot-guides-and-resources/cellular-vs-wifi-for-iot/`. Last accessed 2022-11-14.

[29] "Cellular IoT - what is cellular IoT." https://www.nordicsemi.com/Products/Low-power-cellular-IoT/What-is-cellular-IoT?lang=en#infotabs.

[30] J. Wirges and U. Dettmar, "Performance of TCP and UDP over narrowband internet of things (NB-IoT)," in *2019 IEEE International Conference on Internet of Things and Intelligence System (IoTaIS)*, pp. 5–11, 2019.

[31] A. S. Sadeq, R. Hassan, S. S. Al-rawi, A. M. Jubair, and A. H. Aman, "A qos approach for internet of things (IoT) environment using mqtt protocol," *2019 International Conference on Cybersecurity (ICoCSec)*, 2019.

[32] A. Parmigiani and U. Dettmar, "Comparison and evaluation of lwm2m and mqtt in low-power wide-area networks," in *2021 IEEE International Conference on Internet of Things and Intelligence Systems (IoTaIS)*, pp. 8–14, 2021.

[33] S. Mijovic, E. Shehu, and C. Buratti, "Comparing application layer protocols for the internet of things via experimentation," in *2016 IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI)*, pp. 1–5, 2016.

[34] G. Restuccia, H. Tschofenig, and E. Baccelli, "Low-power iot communication security: On the performance of DTLS and TLS 1.3," in *2020 9th IFIP International Conference on Performance Evaluation and Modeling in Wireless Networks (PEMWN)*, pp. 1–6, 2020.

[35] T. Claeys, M. Vucinic, T. Watteyne, F. Rousseau, and B. Tourancheau, "Performance of the transport layer security handshake over 6TiSCH," *Sensors*, vol. 21, no. 6, 2021.

[36] "Boron lte cat-m1 (noram) with ethersim." https://store.particle.io/collections/cellular/products/boron-lte-cat-m1-noram-with-ethersim-4th-gen. Last accessed 2022-11-12.

[37] "Conexio stratus." https://www.conexiotech.com/. Last accessed 2022-11-12.

[38] "Icarus IoT board v2." https://www.actinius.com/icarus. Last accessed 2022-11-12.

[39] "Nordic thingy:91." https://www.nordicsemi.com/Products/Development-hardware/Nordic-Thingy-91/GetStarted. Last accessed 2022-11-12.

[40] "Particle docs: Libraries." https://docs.particle.io/firmware/best-practices/libraries/. Last accessed 2022-11-13.

[41] "4th generation, high-accuracy, ultra-low-power, 16-bit relative humidity and temperature sensor." https://sensirion.com/media/documents/33FD6951/624C4357/Datasheet_SHT4x.pdf. Last accessed 2022-11-13.

[42] "MEMS digital output motion sensor:ultra low-power high performance 3-axis femto accelerometer." https://www.st.com/en/mems-and-sensors/lis2dh.html. Last accessed 2022-11-12.

[43] "Aem10941 solar energy harvesting." https://e-peas.com/product/aem10941/. Last accessed 2022-11-12.

[44] "Revie flex series cellular antennas." https://www.lairdconnect.com/internal-antennas/multibandcellular-iot-and-m2m/revie-flex-series-cellular-antennas. Last accessed 2022-11-13.

[45] "The rust programming language, foreword." https://doc.rust-lang.org/book/foreword.html, Aug 2019.

[46] D. M. Ritchie, "The development of the C language," *ACM Sigplan Notices*, vol. 28, no. 3, pp. 201–208, 1993.

[47] L. Cen, R. Marcus, H. Mao, J. Gottschlich, M. Alizadeh, and T. Kraska, "Learned garbage collection," in *Proceedings of the 4th ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pp. 38–44, 2020.

[48] F. Wang, F. Song, M. Zhang, X. Zhu, and J. Zhang, "Krust: A formal executable semantics of rust," in *2018 International Symposium on Theoretical Aspects of Software Engineering (TASE)*, pp. 44–51, 2018.

[49] J. Gjengset, *Rust for Rustaceans: Idiomatic Programming for Experienced Developers*. No Starch Press, 2021.

[50] S. Klabnik and C. Nichols, *The Rust Programming Language*. No Starch Press, 2019.

[51] "nrfxlib." https://developer.nordicsemi.com/nRF_Connect_SDK/doc/1.4.2/nrfxlib/README.html. Last accessed 2022-11-13.

[52] "Jr & Sr R3D dials." https://rochestersensors.com/product/jr-and-sr-r3d-dials-lp/. Last accessed 2022-11-13.

[53] M. Majima, S. Ujiie, E. Yagasaki, K. Koyama, and S. Inazawa, "Development of long life lithium ion battery for power storage," *Journal of Power Sources*, vol. 101, no. 1, pp. 53–59, 2001.

[54] V. Khomenko and V. Barsukov, "Lithium-ion capacitor for photovoltaic energy system," in *Materials Today: Proceedings*, no. 6, pp. 116–120, Elsevier, 2019.

[55] J. Ronsmans and B. Lalande, "Combining energy with power: Lithium-ion capacitors," in *2015 International Conference on Electrical Systems for Aircraft, Railway, Ship Propulsion and Road Vehicles (ESARS)*, pp. 1–4, IEEE, 2015.

[56] "Solar resource maps and data." https://www.nrel.gov/gis/solar-resource-maps.html. Last accessed 2022-11-13.

[57] J. Aparicio, J. Munns, and Knurling-team, “Probe-run.” `https://crates.io/crates/probe-run`. Last accessed 2022-11-06.

[58] N. Husser, D. Boehi, A. Greig, and E. Fresk, “Probe-rs.” `https://probe.rs/`. Last accessed 2022-11-06.

[59] “Develop connected hardware with a cloud built for you.” `https://golioth.io/`. Last accessed 2022-11-13.

2 Absolute Maximum Ratings

Parameter	Rating
Vsrc	5.5 V
Operating junction temperature	-40 °C to +125 °C
Storage temperature	-65 °C to +150 °C

Table 2: Absolute maximum ratings

3 Thermal Resistance

Package	θ_{JA}	θ_{JC}	Unit
QFN28	38.3	2.183	°C/W

Table 3: Thermal data

ESD CAUTION

ESD (ELECTROSTATIC DISCHARGE) SENSITIVE DEVICE
These devices have limited built-in ESD protection and damage may thus occur on devices subjected to high-energy ESD. Therefore, proper ESD precautions should be taken to avoid performance degradation or loss of functionality.

4 Typical Electrical Characteristics at 25 °C

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
Power conversion						
$Psrc_{CS}$	Source power required for cold start.	During cold start	3			μW
Vsrc	Input voltage of the energy source.	During cold start	0.38		5	V
		After cold start	0.05		5	
V_{CS}	Custom cold-start voltage.	During the cold start (see page 12)	0.5		4	V
Vboost	Output of the boost converter.	During normal operation	2.2		4.5	V
Vbuck	Output of the buck converter.	During normal operation	2	2.2	2.5	
Storage element						
Vbatt	Voltage on the storage element.	Rechargeable battery	2.2		4.5	V
		Capacitor	0		4.5	V
Tcrit	Time before shutdown after STATUS[1] has been asserted.		400	600	800	ms
Vprim	Voltage on the primary battery.		0.6		5	V
Vfb_prim_u	Feedback for the minimal voltage level on the primary battery.		0.15		1.1	V
Vovch	Maximum voltage accepted on the storage element before disabling the boost converter.	see Table 7	2.3		4.5	V
Vchrdy	Minimum voltage required on the storage element before enabling the LDOs after a cold start.	see Table 7	2.25		4.45	V
Vovdis	Minimum voltage accepted on the storage element before switching to primary battery or entering into a shutdown.	see Table 7	2.2		4.4	V
Low-voltage LDO regulator						
Vlv	Output voltage of the low-voltage LDO.	see Table 7	1.2		1.8	V
Ilv	Load current from the low-voltage LDO.		0		20	mA
High-voltage LDO regulator						
Vhv	Output voltage of the high-voltage LDO.	see Table 7	1.8		Vbatt - 0.3 V	V
Ihv	Load current from the high-voltage LDO.		0		80	mA
Logic output pins						
STATUS[2:0]	Logic output levels on the status pins.	Logic high (VOH)	1.98	Vbatt		V
		Logic low (VOL)	-0.1		0.1	V

Table 4: Electrical characteristics

APPENDIX B: DRV5055-Q1 DATASHEET

DRV5055-Q1 Automotive Ratiometric Linear Hall Effect Sensor

1 Features

- Ratiometric Linear Hall Effect Magnetic Sensor
- Operates From 3.3-V and 5-V Power Supplies
- Analog Output With V_{CC} / 2 Quiescent Offset
- Magnetic Sensitivity Options (At V_{CC} = 5 V):
  – A1: 100 mV/mT, ±21-mT Range
  – A2: 50 mV/mT, ±42-mT Range
  – A3: 25 mV/mT, ±85-mT Range
  – A4: 12.5 mV/mT, ±169-mT Range
  – A5: –100 mV/mT, ±21-mT Range
- Fast 20-kHz Sensing Bandwidth
- Low-Noise Output With ±1-mA Drive
- Compensation For Magnet Temperature Drift
- AEC-Q100 Qualified for Automotive Applications:
  – Temperature Grade 0: –40°C to 150°C
- Standard Industry Packages:
  – Surface-Mount SOT-23
  – Through-Hole TO-92

2 Applications

- Automotive Position Sensing
- Brake, Acceleration, Clutch Pedals
- Torque Sensors, Gear Shifters
- Throttle Position, Height Leveling
- Powertrain and Transmission Components
- Absolute Angle Encoding
- Current Sensing

3 Description

The DRV5055-Q1 is a linear Hall effect sensor that responds proportionally to magnetic flux density. The device can be used for accurate position sensing in a wide range of applications.

The device operates from 3.3-V or 5-V power supplies. When no magnetic field is present, the analog output drives half of V_{CC}. The output changes linearly with the applied magnetic flux density, and five sensitivity options enable maximal output voltage swing based on the required sensing range. North and south magnetic poles produce unique voltages.

Magnetic flux perpendicular to the top of the package is sensed, and the two package options provide different sensing directions.

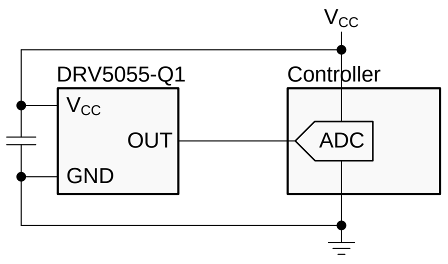
The device uses a ratiometric architecture that can eliminate error from V_{CC} tolerance when the external analog-to-digital converter (ADC) uses the same V_{CC} for its reference. Additionally, the device features magnet temperature compensation to counteract how magnets drift for linear performance across a wide –40°C to +150°C temperature range.

Device Information[1]

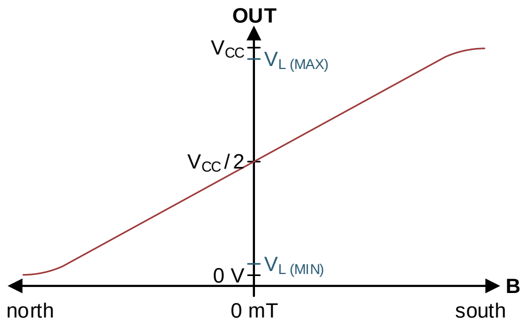
PART NUMBER	PACKAGE	BODY SIZE (NOM)
DRV5055-Q1	SOT-23 (3)	2.92 mm × 1.30 mm
	TO-92 (3)	4.00 mm × 3.15 mm

(1) For all available packages, see the orderable addendum at the end of the data sheet.

Typical Schematic



Magnetic Response (A1, A2, A3, A4 Versions)



6.2 ESD Ratings

			VALUE	UNIT
$V_{(ESD)}$	Electrostatic discharge	Human body model (HBM), per AEC Q100-002(1)	±2500	V
		Charged device model (CDM), per AEC Q100-011	±750	

(1) AEC Q100-002 indicates that HBM stressing shall be in accordance with the ANSI/ESDA/JEDEC JS-001 specification.

6.3 Recommended Operating Conditions

over operating free-air temperature range (unless otherwise noted)

		MIN	MAX	UNIT
V_{CC}	Power-supply voltage(1)	3	3.63	V
		4.5	5.5	
I_O	Output continuous current	–1	1	mA
T_A	Operating ambient temperature(2)	–40	150	°C

(1) There are two isolated operating V_{CC} ranges. For more information see the *Operating V_{CC} Ranges* section.
(2) Power dissipation and thermal limits must be observed.

6.4 Thermal Information

THERMAL METRIC(1)		DRV5055-Q1		UNIT
		SOT-23 (DBZ)	TO-92 (LPG)	
		3 PINS	3 PINS	
$R_{\theta JA}$	Junction-to-ambient thermal resistance	170	121	°C/W
$R_{\theta JC(top)}$	Junction-to-case (top) thermal resistance	66	67	°C/W
$R_{\theta JB}$	Junction-to-board thermal resistance	49	97	°C/W
Y_{JT}	Junction-to-top characterization parameter	1.7	7.6	°C/W
Y_{JB}	Junction-to-board characterization parameter	48	97	°C/W

(1) For more information about traditional and new thermal metrics, see the *Semiconductor and IC Package Thermal Metrics* application report.

6.5 Electrical Characteristics

for V_{CC} = 3 V to 3.63 V and 4.5 V to 5.5 V, over operating free-air temperature range (unless otherwise noted)

PARAMETER		TEST CONDITIONS(1)		MIN	TYP	MAX	UNIT
I_{CC}	Operating supply current				6	10	mA
t_{ON}	Power-on time (see Figure 18)	B = 0 mT, no load on OUT			175	330	µs
f_{BW}	Sensing bandwidth				20		kHz
t_d	Propagation delay time	From change in B to change in OUT			10		µs
B_{ND}	Input-referred RMS noise density	V_{CC} = 5 V			130		$nT/\sqrt{Hz}$
		V_{CC} = 3.3 V			215		
B_N	Input-referred noise	$B_{ND} \times 6.6 \times \sqrt{20\ kHz}$	V_{CC} = 5 V		0.12		mT_{PP}
			V_{CC} = 3.3 V		0.2		
V_N	Output-referred noise(2)	$B_N \times S$	DRV5055A1, DRV5055A5		12		mV_{PP}
			DRV5055A2		6		
			DRV5055A3		3		
			DRV5055A4		1.5		

(1) B is the applied magnetic flux density.
(2) V_N describes voltage noise on the device output. If the full device bandwidth is not needed, noise can be reduced with an RC filter.

6.6 Magnetic Characteristics

for V_{CC} = 3 V to 3.63 V and 4.5 V to 5.5 V, over operating free-air temperature range (unless otherwise noted)

PARAMETER		TEST CONDITIONS[1]		MIN	TYP	MAX	UNIT
V_Q	Quiescent voltage	B = 0 mT, T_A = 25°C	V_{CC} = 5 V	2.43	2.5	2.57	V
			V_{CC} = 3.3 V	1.59	1.65	1.71	
$V_{Q\Delta T}$	Quiescent voltage temperature drift	B = 0 mT, T_A = –40°C to 150°C versus 25°C			±1% × V_{CC}		V
V_{QRE}	Quiescent voltage ratiometry error[2]				±0.2%		
$V_{Q\Delta L}$	Quiescent voltage lifetime drift	High-temperature operating stress for 1000 hours			< 0.5%		
S	Sensitivity	V_{CC} = 5 V, T_A = 25°C	DRV5055A1	95	100	105	mV/mT
			DRV5055A2	47.5	50	52.5	
			DRV5055A3	23.8	25	26.2	
			DRV5055A4	11.9	12.5	13.2	
			DRV5055A5	–105	–100	–95	
		V_{CC} = 3.3 V, T_A = 25°C	DRV5055A1	57	60	63	
			DRV5055A2	28.5	30	31.5	
			DRV5055A3	14.3	15	15.8	
			DRV5055A4	7.1	7.5	7.9	
			DRV5055A5	–63	–60	–57	
B_L	Linear magnetic sensing range[3] [4]	V_{CC} = 5 V, T_A = 25°C	DRV5055A1, DRV5055A5	±21			mT
			DRV5055A2	±42			
			DRV5055A3	±85			
			DRV5055A4	±169			
		V_{CC} = 3.3 V, T_A = 25°C	DRV5055A1, DRV5055A5	±22			
			DRV5055A2	±44			
			DRV5055A3	±88			
			DRV5055A4	±176			
V_L	Linear range of output voltage[4]			0.2		V_{CC} – 0.2	V
S_{TC}	Sensitivity temperature compensation for magnets[5]				0.12		%/°C
S_{LE}	Sensitivity linearity error[4]	V_{OUT} is within V_L			±1%		
S_{SE}	Sensitivity symmetry error[4]	V_{OUT} is within V_L			±1%		
S_{RE}	Sensitivity ratiometry error[2]	T_A = 25°C, with respect to V_{CC} = 3.3 V or 5 V		–2.5%		2.5%	
$S_{\Delta L}$	Sensitivity lifetime drift	High-temperature operating stress for 1000 hours			<0.5%		

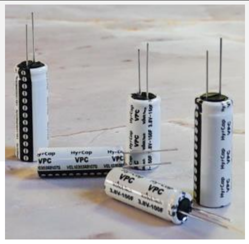
(1) B is the applied magnetic flux density.
(2) See the *Ratiometric Architecture* section.
(3) B_L describes the minimum linear sensing range at 25°C taking into account the maximum V_Q and Sensitivity tolerances.
(4) See the *Sensitivity Linearity* section.
(5) S_{TC} describes the rate the device increases Sensitivity with temperature. For more information, see the *Sensitivity Temperature Compensation for Magnets* section.

3.8V 250F (1335)

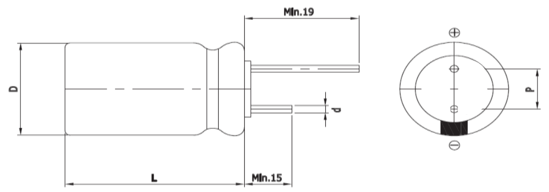
VINATech

Features

VPC (Vina Pulse Capacitor)
- Low Self Discharge
- Wide Temperature Range
- High Operating Voltage
- High Capacitance



Drawing



Size	1335
D (Φ)	12.5 +1.0 Max
L (㎜)	35.0 ±1.5
d (Φ)	0.8 ±0.1
P (㎜)	5.0 ±0.5

Specification

Items	Characteristics	
Rated Voltage (VR)	3.8V	
Operating voltage	3.8V ~ 2.5V	
Surge voltage	4.0V	
Operating temperature	-25℃ to +70℃	
Capacitance Tolerance	-10% +30%	
High Temperature Load Life	After 1,000 hours at VR loaded at 70℃, capacitor shall meet the following limits	
	Capacitance change	≤ 30% of initial value
	ESR change	≤ 200% of initial spec. value
Projected cycle life	20,000 Cycle (100% DoD, at 25℃, cut-off voltage: 2.5V, C/D current: 1.8A)	
	Capacitance change	≤ 30% of initial value
	ESR change	≤ 200% of initial spec. value
Shelf life	3 Years (No electrical charge, Temperature below 25℃)	
	Capacitance change	≤ 10% of initial value
	ESR change	≤ 100% of initial spec. value

Part Number	Capacitance (F) #1	Capacity (mAh)	ESR (mΩ)		Leakage Current (μA)	Rated Current (A)	Pulse Discharge Current (A) #2	Pulse Charge Current (A)	Max Charge Voltage (V) #3	Weight (g)
			AC	DC						
VEL13353R8257G	250 @25℃	90 @25℃	50 @25℃, 1KHz	100 @25℃	10 @25℃, 72hr	0.75 @25℃	5.0 @25℃	8.0 @25℃	3.85 @25℃	8.2±0.3

#1 : Reference IEC62813 4.2
#2 : 1sec. Discharge to 3.2V
#3 : If the charging voltage is continuously used at 3.85V, the lifespan is reduced by 10%
WARNING : precautions must be taken to ensure that device leads are not shorted

Version 1.4 2022.10.05.

APPENDIX D: ADC TO TANK LEVEL CONVERSION RAW DATA

Level	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
5	945	948	955	957	954	951	952	959	952	950
10	946	938	944	950	942	942	943	948	945	941
15	1021	1021	1024	1018	1022	1024	1020	1021	1022	1023
20	1109	1105	1107	1107	1109	1105	1109	1106	1103	1104
25	1202	1194	1201	1197	1202	1194	1195	1194	1195	1194
30	1280	1275	1278	1276	1281	1274	1276	1267	1272	1273
40	1460	1458	1460	1455	1460	1452	1459	1455	1453	1456
50	1648	1646	1649	1651	1651	1646	1653	1645	1645	1649
60	1874	1864	1866	1866	1864	1865	1869	1864	1863	1864
70	2069	2069	2064	2065	2065	2062	2062	2066	2068	2059
80	2259	2253	2260	2260	2346	2345	2343	2342	2340	2340
90	2401	2400	2400	2397	2389	2398	2396	2397	2397	2399
95	2490	2495	2496	2493	2492	2496	2494	2494	2495	2494

APPENDIX E: BUILD.RS

```rust
use std::env;
use std::fs::File;
use std::io::Write;
use std::path::PathBuf;

fn main() {
    // Put `memory.x` in our output directory and ensure it's
    // on the linker search path.
    let out = &PathBuf::from(env::var_os("OUT_DIR").unwrap());
    File::create(out.join("memory.x"))
        .unwrap()
        .write_all(include_bytes!("memory.x"))
        .unwrap();
    println!("cargo:rustc-link-search={}", out.display());

    // By default, Cargo will re-run a build script whenever
    // any file in the project changes. By specifying `memory.x`
    // here, we ensure the build script is only re-run when
    // `memory.x` is changed.
    println!("cargo:rerun-if-changed=memory.x");

    println!("cargo:rustc-link-arg-bins=--nmagic");
    println!("cargo:rustc-link-arg-bins=-Tlink.x");
    println!("cargo:rustc-link-arg-bins=-Tdefmt.x");
}
```

APPENDIX F: CARGO.TOML

```toml
[package]
name = "propane_monitor_embassy"
version = "0.4.2"
edition = "2021"

[features]
default = ["nightly"]
nightly = [
    "embassy-executor/nightly", "embassy-nrf/nightly",
    "embassy-nrf/unstable-traits"
]

[dependencies]
alloc-cortex-m = "0.4.2"
at-commands = "0.5.2"
coap-lite = {version = "0.11.2", default-features = false}
cortex-m = {version = "0.7.6",
    features = ["critical-section-single-core"]
}
cortex-m-rt = "0.7.0"
defmt = "0.3"
defmt-rtt = "0.3"
embassy-futures = { version = "0.1.0"}
embassy-sync = { version = "0.1.0", features = ["defmt"] }
embassy-executor = {
    version = "0.1.0", features = ["defmt", "integrated-timers"]
}
embassy-time = {
    version = "0.1.0", features = ["defmt", "defmt-timestamp-uptime"]
}

```

```
embassy-nrf = {
    version = "0.1.0",
    features = ["defmt", "nrf9160-ns", "time-driver-rtc1",
    "gpiote", "unstable-pac"]
}
futures = {
    version = "0.3.17", default-features = false,
    features = ["async-await"]
}
heapless = { version = "0.7.16", features = ["serde"] }
nrf-modem = {
    git = "https://github.com/diondokter/nrf-modem.git",
    features = ["defmt"]
}
panic-probe = { version = "0.3", features = ["print-defmt"] }
serde = { version = "1.0", default-features = false,
    features = ["derive"]
}
serde_json = {
    version = "1.0", default-features = false, features = ["alloc"]
}
static_cell = "1.0"
tinyrlibc = {
    git = "https://github.com/diondokter/tinyrlibc.git",
    branch = "ncs-2.0.1"
}

[patch.crates-io]
embassy-futures = { git = "https://github.com/embassy-rs/embassy" }
embassy-sync = { git = "https://github.com/embassy-rs/embassy" }
embassy-executor = { git = "https://github.com/embassy-rs/embassy" }
embassy-time = { git = "https://github.com/embassy-rs/embassy" }
embassy-nrf = { git = "https://github.com/embassy-rs/embassy" }
```

```

# cargo build/run --release
[profile.release]
codegen-units = 1
debug = 2
debug-assertions = false
incremental = false
lto = true
opt-level = 'z'
overflow-checks = false
```

APPENDIX G: AT.RS

```rust
use at_commands::parser::CommandParser;
use crate::Error;

/// Parse AT+CESQ command response and return a signal strength in dBm
/// Signal strength = -140 dBm + last int_parameter
async fn get_signal_strength() -> Result<i32, Error> {
    let command = nrf_modem::at::send_at::<32>("AT+CESQ").await?;

    let (_, _, _, _, _, mut signal) = CommandParser::parse(command.as_bytes())
        .expect_identifier(b"+CESQ:")
        .expect_int_parameter()
        .expect_int_parameter()
        .expect_int_parameter()
        .expect_int_parameter()
        .expect_int_parameter()
        .expect_int_parameter()
        .expect_identifier(b"\r\n")
        .finish()
        .unwrap();
    if signal != 255 {
        signal += -140;
    }
    Ok(signal)
}

```

APPENDIX H: CONFIG.RS

```rust
pub const SERVER_URL: &str = "coap.golioth.io";
pub const SERVER_PORT: u16 = 5684;

pub const PSK_ID: &str = "conexio-stratus-id@propane-monitor";

pub const PSK: &[u8] = b"";

pub const SECURITY_TAG: u32 = 1;
```

APPENDIX I: GNSS.RS

```rust
use super::Error;
use defmt::info;

pub async fn config_gnss() -> Result<(), Error> {
    // confgiure MAGPIO pins for GNSS
    info!("Configuring XMAGPIO pins for 1574-1577 MHz");
    nrf_modem::at::send_at::<0>("AT%XMAGPIO=1,0,0,1,1,1574,1577").await?;
    nrf_modem::at::send_at::<0>("AT%XCOEX0=1,1,1574,1577").await?;
    Ok(())
}

/// Function to retrieve GPS data from a single GNSS fix
pub async fn get_gnss_data() -> Result<(), Error> {
    let mut gnss = nrf_modem::gnss::Gnss::new().await?;
    let config = nrf_modem::gnss::GnssConfig::default();
    let mut iter = gnss.start_single_fix(config)?;

    if let Some(x) = futures::StreamExt::next(&mut iter).await {
        info!("{:?}", Debug2Format(&x.unwrap()));
    }
    Ok(())
}
```

APPENDIX J: PSK.RS

```rust
use core::fmt::write;
use defmt::Format;
use heapless::String;
use crate::config::{PSK, PSK_ID, SECURITY_TAG};
use crate::Error;

/// Credential Storage Management Types
#[derive(Clone, Copy, Format)]
#[allow(dead_code)]
enum CSMType {
    RootCert = 0,
    ClientCert = 1,
    ClientPrivateKey = 2,
    Psk = 3,
    PskId = 4,
    // ...
}

/// This function deletes a key or certificate from the nrf modem
async fn key_delete(ty: CSMType) -> Result<(), Error> {
    let mut cmd: String<32> = String::new();
    write(
        &mut cmd,
        format_args!("AT%CMNG=3,{},{}", SECURITY_TAG, ty as u32),
    )
        .unwrap();
    nrf_modem::at::send_at::<32>(cmd.as_str()).await?;
    Ok(())
}

/// This function writes a key or certificate to the nrf modem
```

```
async fn key_write(ty: CSMType, data: &str) -> Result<(), Error> {
    let mut cmd: String<128> = String::new();
    write(
        &mut cmd,
        format_args!(r#"AT%CMNG=0,{},{},"{}""#, SECURITY_TAG, ty as u32, data),
    )
        .unwrap();

    nrf_modem::at::send_at::<128>(&cmd.as_str()).await?;

    Ok(())
}

/// Delete existing keys/certificates and loads new
/// ones based on config.rs entries
pub async fn install_psk_id_and_psk() -> Result<(), Error> {
    assert!(
        !&PSK_ID.is_empty() && !&PSK.is_empty(),
        "PSK ID and PSK must not be empty. Set them in the `config` module."
    );

    key_delete(CSMType::PskId).await?;
    key_delete(CSMType::Psk).await?;

    key_write(CSMType::PskId, &PSK_ID).await?;
    key_write(CSMType::Psk, &encode_psk_as_hex(&PSK)).await?;

    Ok(())
}

fn encode_psk_as_hex(psk: &[u8]) -> String<128> {
    fn hex_from_digit(num: u8) -> char {
        if num < 10 {
            (b'0' + num) as char
        } else {
            (b'a' + num - 10) as char
        }
    }

    let mut s: String<128> = String::new();
    for ch in psk {
        s.push(hex_from_digit(ch / 16)).unwrap();
        s.push(hex_from_digit(ch % 16)).unwrap();
    }

    s
}
```

APPENDIX K: LIB.RS

```rust
#![no_main]
#![no_std]
#![feature(alloc_error_handler)]

extern crate alloc;
extern crate tinyrlibc;

mod config;
pub mod psk;
mod at;
mod gnss;

use crate::config::{SECURITY_TAG, SERVER_PORT, SERVER_URL};
use alloc_cortex_m::CortexMHeap;
use at_commands::parser::ParseError;
use coap_lite::error::MessageError;
use coap_lite::{CoapRequest, ContentFormat, RequestType};
use core::mem::MaybeUninit;
use core::sync::atomic::{AtomicBool, Ordering};
use defmt::{info};
use embassy_nrf as _;
use embassy_time::TimeoutError;
use heapless::{Vec};
use nrf_modem::dtls_socket::{DtlsSocket, PeerVerification};
use serde::Serialize;
use {defmt_rtt as _, panic_probe as _};
use crate::at::*;

/// Once flashed, comment this out along with the SPM
///entry in memory.x to eliminate flashing the SPM
/// more than once, and will speed up subsequent builds.
```

```
///Or leave it and flash it every time
#[link_section = ".spm"]
#[used]
static SPM: [u8; 24052] = *include_bytes!("zephyr.bin");

/// Crate error types
#[derive(Debug)]
pub enum Error {
    Coap(MessageError),
    Json(serde_json::error::Error),
    NrfModem(nrf_modem::error::Error),
    Timeout(TimeoutError),
    ParseError(ParseError),
}

impl From<MessageError> for Error {
    fn from(e: MessageError) -> Self {
        Self::Coap(e)
    }
}

impl From<serde_json::error::Error> for Error {
    fn from(e: serde_json::error::Error) -> Self {
        Self::Json(e)
    }
}

impl From<nrf_modem::error::Error> for Error {
    fn from(e: nrf_modem::error::Error) -> Self {
        Self::NrfModem(e)
    }
}

```

```
impl From<TimeoutError> for Error {
    fn from(e: TimeoutError) -> Self {
        Self::Timeout(e)
    }
}

impl From<ParseError> for Error {
    fn from(e: ParseError) -> Self {
        Self::ParseError(e)
    }
}

/// Payload to send over CoAP (Heapless Vec of Tanklevel Structs)
#[derive(Debug, Serialize)]
pub struct Payload<'a> {
    pub data: Vec<TankLevel, 6>,
    pub signal: i32,
    pub timeouts: u8,
    location: &'a str,
}

/// Payload constructor
impl Payload<'_> {
    pub fn new() -> Self {
        Payload {
            data: Vec::new(),
            signal: 0,
            timeouts: 0,
            location: "Lowes3",
        }
    }
}

```

```
/// Structure to hold our individual measure data
#[derive(Debug, Serialize)]
pub struct TankLevel {
    pub value: u32,
    pub timestamp: u32,
    pub battery: u32,
}

/// TankLevel constructor
impl TankLevel {
    pub fn new(value: u32, timestamp: u32, battery: u32) -> Self {
        TankLevel {
            value,
            timestamp,
            battery,
        }
    }
}

/// Create CoAP request, serialize payload, and transimt data
/// request path can start with .s/ for LightDB Stream or
/// .d/ LightDB State for Golioth IoT
pub async fn transmit_payload(payload: &mut Payload<'_>) -> Result<(), Error> {
    // Create our DTLS socket
    let mut socket = DtlsSocket::new(PeerVerification::Enabled,
    &[SECURITY_TAG]).await?;
    info!("DTLS Socket created");
    socket.connect(SERVER_URL, SERVER_PORT).await?;
    info!("DTLS Socket connected");

    let sig_strength = get_signal_strength().await?;
    payload.signal = sig_strength;
    info!("Signal Strength: {} dBm", &sig_strength);

    let mut request: CoapRequest<DtlsSocket> = CoapRequest::new();
    request.set_method(RequestType::Post);
    request.set_path(".s/tank_level");
    request
        .message
        .set_content_format(ContentFormat::ApplicationJSON);
    let json = serde_json::to_vec(payload)?;
    // info!("Payload: {:?}", Debug2Format(payload));
    // info!("JSON Byte Vec: {:?}", Debug2Format(&json));
    request.message.payload = json;

    socket.send(&request.message.to_bytes()?).await?;
    info!("Payload done");

    // The sockets would be dropped after the function call ends,
    // but this explicit call allows them
    // to be dropped asynchronously
    info!("deactivate socket");
    socket.deactivate().await?;

    Ok(())
}

/// Convert sensor ADC value into tank level percentage
pub fn convert_to_tank_level(x: i16) -> u32 {
    let val = ((534 * x as u32) - 39_0634) / 10000;
    info!("Tank Level: {}", &val);
    if val > 100 {
        100
    } else if val < 10 {
        10
    } else {
        val
    }
}

/// Convert ADC value into a milli-volt battery measurement
pub fn convert_to_mv(x: i16) -> u32 {
    // Stratus: V_bat measurement multiplier = 200/100
    // Icarus: V_bat measurement multiplier = 147/100
    ((x * (200 / 100)) as u32 * 3600) / 4096
}

/// Terminates the application and makes `probe-run` exit with exit-code = 0
pub fn exit() -> ! {
    loop {
        cortex_m::asm::bkpt();
    }
}

/// An allocator is required for the coap-lite lib
#[global_allocator]
static ALLOCATOR: CortexMHeap = CortexMHeap::empty();

static mut HEAP_DATA: [MaybeUninit<u8>; 8196] = [MaybeUninit::uninit(); 8196];

pub fn alloc_init() {
    static ONCE: AtomicBool = AtomicBool::new(false);

    if ONCE
        .compare_exchange(false, true, Ordering::SeqCst, Ordering::SeqCst)
        .is_ok()
    {
        unsafe {
            ALLOCATOR.init(HEAP_DATA.as_ptr() as usize, HEAP_DATA.len());
        }
    }
}

/// Default alloc error handler for when allocation fails
#[alloc_error_handler]
fn alloc_error(_: core::alloc::Layout) -> ! {
    cortex_m::asm::udf()
}
```

APPENDIX L: APP.RS

```
#![no_std]
#![no_main]
#![feature(type_alias_impl_trait)]

use defmt::{error, info, unwrap};
use embassy_executor::Spawner;
use embassy_nrf::gpio::{Flex, Level, Output, OutputDrive};
use embassy_nrf::interrupt::{self, InterruptExt, Priority};
use embassy_nrf::pac::{UARTE0, UARTE1};
// use embassy_nrf::pwm::{Prescaler, SimplePwm};
use embassy_nrf::saadc::{ChannelConfig, Config, Saadc};
use embassy_time::{Duration, Ticker, Timer, with_timeout};
use futures::StreamExt;
use nrf_modem::{ConnectionPreference, SystemMode};
use propane_monitor_embassy::*;
use propane_monitor_embassy::psk::install_psk_id_and_psk;

#[embassy_executor::main]
async fn main(_spawner: Spawner) {
    // Set up the interrupts for the modem
    let egu1 = interrupt::take!(EGU1);
    egu1.set_priority(Priority::P4);
    egu1.set_handler(|_| {
        nrf_modem::application_irq_handler();
        cortex_m::asm::sev();
    });
    egu1.enable();

    let ipc = interrupt::take!(IPC);
    ipc.set_priority(Priority::P0);
    ipc.set_handler(|_| {
        nrf_modem::ipc_irq_handler();
        cortex_m::asm::sev();
    });
    ipc.enable();

    // // Disable UARTE for lower power consumption
    let uarte0: UARTE0 = unsafe { core::mem::transmute(()) };
    let uarte1: UARTE1 = unsafe { core::mem::transmute(()) };
    uarte0.enable.write(|w| w.enable().disabled());
    uarte1.enable.write(|w| w.enable().disabled());

    // Initialize heap data
    alloc_init();

    // Run our sampling program, will not return unless an error occurs
    match run().await {
        Ok(()) => unreachable!(),
        Err(e) => {
            // If we get here, we have problems
            error!("app exited: {:?}", defmt::Debug2Format(&e));
            exit();
        }
    }
}

async fn run() -> Result<(), Error> {
    // Handle for device peripherals
    let mut p = embassy_nrf::init(Default::default());

    // Stratus: Disconnect accelerometer for power savings
    Flex::new(&mut p.P0_29).set_as_disconnected();

    // Configuration of ADC, over sample to reduce noise (8x)
```

```rust
    let adc_config = Config::default();
    // Oversample can only be used when you have a single channel
    // adc_config.oversample = Oversample::OVER8X;

    // Pin 14 can be used on both Stratus and Icarus boards for Analog Input
    let sensor_channel = ChannelConfig::single_ended(&mut p.P0_14);
    // Stratus: Pin 20 for V_bat measurement
    // Icarus: Pin 13 for V_bat measurement
    let bat_channel = ChannelConfig::single_ended(&mut p.P0_20);

    let mut adc = Saadc::new(
        p.SAADC,
        interrupt::take!(SAADC),
        adc_config,
        [sensor_channel, bat_channel],
    );
    adc.calibrate().await;
    info!("ADC Initialized");

    // Icarus: Has an eSIM and an External SIM.
    // Use Pin 8 to select: HIGH = eSIM, Low = External
    // Only change SIM selection while modem is off (AT+CFUN=1)
    // let _sim_select =
        Output::new(p.P0_08, Level::Low,OutputDrive::Standard);

    // Hall effect sensor power, must be High Drive to provide enough current (6 mA)
    let mut hall_effect =
        Output::new(p.P0_31, Level::Low, OutputDrive::Disconnect0HighDrive1);

    // Stratus: Pin 25 to control VBAT_MEAS_EN, Power must connect to V_Bat to measure correctly
    // Icarus: Pin 07 to disable battery charging circuit
    let mut enable_bat_meas = Output::new(p.P0_25, Level::Low, OutputDrive::Standard);
    // let _disable_charging = Output::new(p.P0_07, Level::High, OutputDrive::Standard);

    // Stratus: Pin 3 for blue LED power when data is being transmitted
    // Stratus: Pin 12 for blue LED power when data is being transmitted,
    // (red: P_10, green: P_11)
    let mut led = Output::new(p.P0_03, Level::High, OutputDrive::Standard);

    // Initialize cellular modem
    unwrap!(
        nrf_modem::init(SystemMode {
            lte_support: true,
            nbiot_support: false,
            gnss_support: true,
            preference: ConnectionPreference::Lte,
        })
        .await
    );

    // Configure GPS settings
    // config_gnss().await?;

    // install PSK info for secure cloud connectivity
    install_psk_id_and_psk().await?;

    // Heapless buffer to hold our sample values before transmitting
    let mut payload = Payload::new();

    // Create our sleep timer (time between sensor measurements)
    // Set to 15 seconds for quick testing
    let mut ticker = Ticker::every(Duration::from_secs(15));
    info!("Entering Loop");
    loop {
        let mut buf = [0; 2];

        // get_gnss_data().await?;

        // Power up the hall sensor: max power on time = 330us
        // (wait for 500us to be safe)
        hall_effect.set_high();
        enable_bat_meas.set_high();

        Timer::after(Duration::from_micros(500)).await;
        adc.sample(&mut buf).await;

        hall_effect.set_low();
        enable_bat_meas.set_low();

        info!(
            "Tank level: {}%, Battery: {} mV",
            convert_to_tank_level(buf[0]),
            convert_to_mv(buf[1])
        );

        payload
            .data
            .push(TankLevel::new(
                convert_to_tank_level(buf[0]),
                1987,
                convert_to_mv(buf[1]),
            ))
            .unwrap();

        // Our payload data buff is full, send to the cloud, clear the buffer
        if payload.data.is_full() {
            // info!("TankLevel: {}", core::mem::size_of::<TankLevel>());
            info!("Payload is full");

            // Visibly show that data is being sent
            led.set_low();

            // If timeout occurs, log a timeout and continue.
            if let Ok(_) =
                with_timeout(
                    Duration::from_secs(30), transmit_payload(&mut payload)
                )
                    .await
            {
                payload.timeouts = 0;

                info!("Transfer Complete");
            } else {
                payload.timeouts += 1;
                info!(
                    "Timeout has occurred {} time(s), data clear and start over",
                    payload.timeouts
                );
            }

            payload.data.clear();

            led.set_high();
        }
        info!("Ticker next()");
        ticker.next().await; // wait for next tick event
    }
}

```