European University Cyprus

School of Sciences | Department of Computer Science and Engineering

Post-Operation Vital Sign Monitoring Device

Bachelor of Science in Computer Science

Martin Aziz

Acknowledgements

Write some text for someone that you would like to say thank you for the completion of this work

Abstract

Write a page of text explaining the project, what you wanted to do, what was done, how it was done and what has been achieved.

Contents

Chapter 1

Introduction

1.1 Introduction

It is crucial to monitor patient vital signs in the post-operative period to ensure detection of any complications early on, and increase patient safety. Traditionally, nurses and clinical staff carry out this monitoring at regular intervals, which may leave gaps in observation and increase the risk of missing early warning signs. Advances in the Internet of Things (IoT) space have created new possibilities for real-time, continuous monitoring of patients using wireless sensors and communication protocols.

One of the main difficulties in implementing IoT systems in healthcare is selecting appropriate technologies for communication, that balance power consumption, range, and reliability, at a reasonable cost. Bluetooth, which is a widely used technology in consumer health devices, offers short-range, low-energy connectivity but it doesn't scale well to the size of a hospital, for the purpose of monitoring hundreds of patients at a time.The Long Range Wide Area Network (LoRaWAN) protocol) protocol on the other hand, is capable of communication at long ranges, using low power to transmit small amounts of data (perfect for sending the kinds of data sensors produce) from multiple devices to a central gateway.

This thesis presents the design and implementation of a device intended for the purpose of monitoring patient vital signs in the post-operative context, that integrates consumer medical sensing devices, and incorporates both Bluetooth as well as the LoRaWAN communication protocols. It is designed to collect data from commercially available medical devices and send that data efficiently to the cloud for analysis and alerting purposes.

1.2 Aims and Objectives

The main aim of this project is to design and develop an IoT based device to monitor post-operative vital signs. Several specific aims were defined, with clear objectives for each:

1. Investigate the clinical requirements and difficulties in the context of post-operative vital sign monitoring.
    - Conduct a literature review on the importance of post-operative monitoring.
    - Conduct a literature review on the clinical importance of each vital sign in the post-operative context.

2. Evaluate and compare appropriate commercially available Bluetooth capable medical devices for possible integration with the system.
    - Identify commercially avaialble Bluetooth capable devices for measuring temperature, blood pressure, and heart rate.
    - Assess their Bluetooth connectivity and openness, and as such determine their likely degree of integration with a custom system such as the one being developed.

3. Analyse and select suitable communication technologies for device integration and data transmission to cloud.
    - Review relevant communication protocols for device integration.
    - Review relevant communication protocols for long range data transmission to the cloud.
    - Justify the selection of technologies to be used in the system.

4. Design and implement an IoT system to monitor vital signs post-operation.
    - Develop the system architecture, consisting of the device itself as well as a simple backend cloud platform.
    - Make the device capable of connecting to and communicating with Bluetooth devices such as medical sensors.
    - Establish communication between the device itself and the backend cloud platform using LoRaWAN.

5. Test the system under realistic conditions and evaluate its performance.
    - Conduct testing of the system with simulated data representing data incoming from a medical sensor, and test transmission via the LoRaWAN protocol to the cloud.
    - Evaluate metrics such as data reliability, scalability, and energy efficiency.
6. Analyse the project outcomes and propose possible future improvements.
    - Discuss strengths, weaknesses, and potential points of improvement of the system.
    - Discuss any challenges or difficulties faced during design and development.
    - Suggest possible further research and enhancements.

1.3 Structure of the thesis

This thesis is organised into six main chapters, following the natural development process (starting with research and design, followed by implementation, ending with an analysis of the outcomes).

- **Chapter 1: Introduction** — Introduces the background, aims, objectives, and structure of the thesis.
- **Chapter 2: Background** — Reviews the clinical significance of monitoring vital signs after surgery, the role of IoT in this task, analyses current communication protocols and available medical sensing devices, and discusses some points of consideration regarding integration of said devices with the system.
- **Chapter 3: Analysis and Design** — Describes the 'user needs', requirements analysis, system architecture, and give a detailed design of the system.
- **Chapter 4: Implementation and Testing** — Presents the implementation of features, implementation of communication with backend cloud via LoRaWAN, and testing procedures.
- **Chapter 5: Discussion** — Discusses the resulting system, difficulties encountered, lessons learned, and possible improvements.
- **Chapter 6: Bibliography** — Lists all the references cited throughout the thesis.

1.4 Summary

This chapter gave an introduction to the reasoning behind developing such an IoT based system for the post-operative scenario, gave an outline for the aims and objectives of the project, and briefly covered the general structure of the thesis. The next chapter will dive deeper into the background of the project including clinical signigicance of vital sign monitoring, the role of IoT, relevant communication protocols, and compare and evaluate commercially available Bluetooth-capable medical sensing devices for this project.

Chapter 2

Background

2.1 Introduction

The healing process does not end after surgery is complete, but instead it is often a long effort for weeks or months post-operation. Furthermore, it is not just the physical health of the patient which must be considered, but also and perhaps equally important their mental health and quality of life moving forward. In recent decades it has become clearer to medical professionals that extended monitoring following operations greatly impacts the results of surgeries in a positive manner, as stated in [1]. Of course, there are multiple factors affecting the extent to which complications will arise in a patient after receiving surgery, such as their age, previous health issues, or conditions which may develop in parallel to the primary diagnosis even if they are in fact unrelated. In particular, the elderly are quite vulnerable to more complications, as stated by the authors in [2], specifically referring to elderly patients who underwent hip fracture surgery. Such complications affect quality of life longterm as well as general ability to function normally, not just physical health, degree of recovery, and survivability [2].

The role of the surgeon in the postoperative context is to firstly make sure the patient is receiving the necessary support to sustain a healthy balance in their body, and do whatever they can to avoid the development of any complications that may appear as a result of the procedure. Should any complications arise despite the medical staff's best efforts, it is then the surgeon's responsibility to recognise the signs pointing to the development of said complications and take appropriate actions to quickly and effectively manage them, allowing the patient to recover to their preoperative state eventually [3]

2.2 Clinical significance of monitoring each vital sign post-surgery

Postoperative vital sign monitoring is essential for both patient safety and the early identification of problems. Specific information on the patient's physiological state and course of recovery is provided by each measure.

Blood Pressure

To detect hypotension or hypertension, which can both result in major problems, postoperative blood pressure monitoring is crucial [4]. By enabling early diagnosis and control of blood pressure variations, continuous monitoring lowers the risk of postoperative haemorrhage and other unfavourable outcomes [5]. Continuous monitoring, according to studies, can identify hypotensive events that intermittent measures could overlook, allowing for prompt responses [6].

Temperature

After surgery, maintaining normothermia is essential to avoiding problems including surgical site infections and extended hospital stays. Hypothermia can raise the risk of infection and hinder the healing of wounds. Thus, it is advised to regularly check the patient's temperature during the postoperative phase in order to guarantee the best possible results [7].

Oxygen Saturation

Monitoring oxygen saturation is essential for identifying hypoxaemia, which, if left untreated, can result in organ dysfunction. Early management in cases of respiratory compromise is made possible by pulse oximetry, which offers a non-invasive way to continuously measure oxygen levels. When compared to sporadic inspections, continuous monitoring has been demonstrated to improve the detection of oxygen desaturation [8].

Heart Rate

In the postoperative phase, heart rate monitoring is crucial for spotting physiological alterations that could indicate patient decline. As stated in [9] heart rate changes, like bradycardia or tachycardia, can occur hours or even days before serious occurrences like cardiac arrest, particularly in the 48 hours after surgery. This emphasises the necessity of regular or ongoing monitoring. As is typical in many wards, manual spot checks conducted every 4–8 hours are not enough to accurately identify these occurrences. Continuous

monitoring, on the other hand, has been demonstrated to detect heart rate anomalies more frequently and with greater severity than would otherwise be detected, enabling medical professionals to take early action and possibly avert problems or death [9].

Respiratory Rate

Since respiratory rate (RR) is a sensitive predictor of early physiological decline, it is clinically important to monitor RR in postoperative patients. According to studies, irregular RR frequently precedes changes in other vital signs like blood pressure or heart rate and is the first obvious indication of a patient's decline. In clinical settings, RR is commonly undermonitored despite its significance. Elliott [10] pointed out that RR evaluation is frequently disregarded because of factors including a lack of automated measurement methods, time restrictions, and insufficient clinician understanding. This error raises the possibility of unfavourable outcomes by delaying the identification of respiratory impairment.

Furthermore, it has been demonstrated that ongoing respiratory rate (RR) monitoring improves patient safety during the recovery phase. In a retrospective observational analysis, patients who had laparoscopic colon surgery were contrasted with those who had conventional intermittent monitoring in terms of continuous RR monitoring. Two life-threatening respiratory episodes occurred in the intermittently watched group (2 out of 126 patients), while none occurred in the continuously monitored group (0 out of 69 patients), according to the study. According to these results, ongoing RR monitoring may help identify respiratory depression early, enabling prompt treatment and better patient outcomes [11].

It is crucial to remember that the type of operation done and the patient's unique risk factors can affect each vital sign's clinical significance. In order to solve this, the suggested system offers vital sign threshold customisation, allowing medical professionals to adjust monitoring settings to the unique requirements of every patient and surgical situation.

2.3 Role of IoT in vital sign monitoring

By enabling continuous, real-time tracking of physiological parameters, the Internet of Things' (IoT) incorporation into vital sign monitoring has significantly improved postop-

erative patient care. Vital signs like heart rate, blood pressure, respiration rate, temperature, and oxygen saturation are measured by IoT-based systems using wearable sensors. By wirelessly sending data to cloud-based platforms, these sensors enable medical professionals to keep an eye on patients from a distance and act quickly to stop any decline [12].

As an example, an IoT-driven system for continuous post-surgery monitoring of patients with cardiac disease was established in a study by Harez et al. In order to enable prompt medical interventions, the system used sensors to gather real-time vital sign data, which was subsequently examined to identify any issues [13].

An Internet of Things (IoT)-based vital signs monitoring system that combines several sensors to detect different health indicators was also presented by [14]. The architecture of the system facilitates effective data collection, transfer, and analysis, all of which enhance patient outcomes.

IoT-based wearable technology has the potential to provide individualised care and remote monitoring, but issues with data security, patient compliance, and compatibility with current medical systems limit its use in clinical settings. These technologies run the danger of increasing complexity without ensuring better results if they are not carefully implemented and clinically validated [15].

In summary, real-time, remote observation and early intervention capabilities provided by IoT integration into vital sign monitoring show promise for improving postoperative care. Research indicates that it can enhance results by facilitating ongoing monitoring of physiological indicators. Data confidentiality, patient adherence, and system interoperability are some of the issues that still need to be resolved for practical deployment in order to guarantee dependable therapeutic impact and avoid needless complexity.

2.4 Communication protocols in IoT-based monitoring

This section explores the communication protocols relevant to IoT-based monitoring of post-operative patients, focusing on Bluetooth and LoRaWAN. Reliable and efficient data transmission is crucial in this context, where vital signs such as heart rate, temperature, and blood pressure need to be collected with minimal power consumption and transmitted securely over short or long distances.

LoRa is a wireless modulation technique derived from chirp spread spectrum technology (CSS). It enables long-distance, low-power communication by encoding information on radio waves using chirp pulses. This makes it robust against interference and highly suitable for IoT applications that transmit small data packets at low bit rates [16]. Compared to technologies like WiFi, Bluetooth, or ZigBee, LoRa supports data transmission over significantly longer distances, particularly in sub-gigahertz bands, making it well suited for both indoor hospital and outdoor monitoring environments [17].

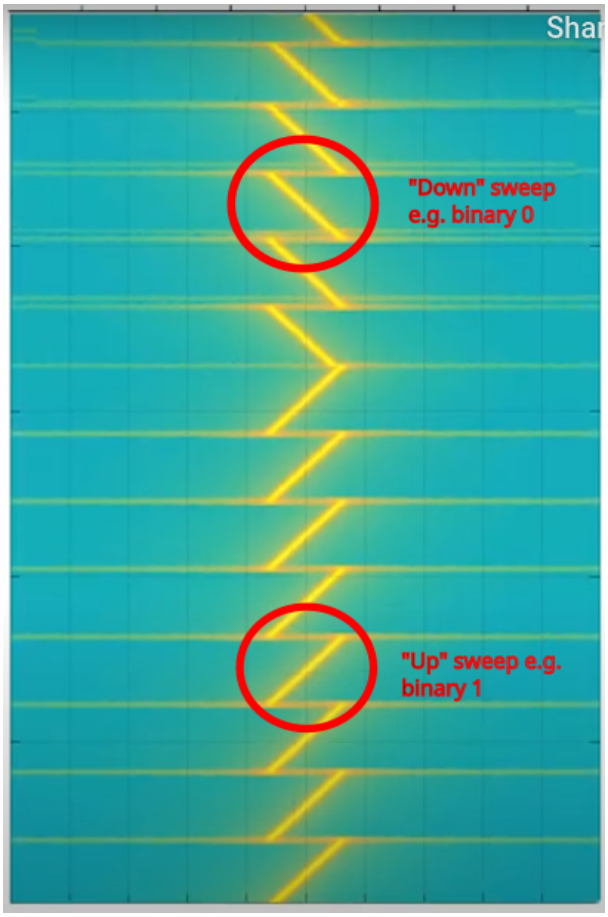


Figure 2.1: Example of a chirp spread spectrum transmission, highlighting the up and down sweeps which represent binary 1 and 0. [18]

Figure 2.1 provides a visual representation of the chirp spread spectrum technology. This technique encodes data in the frequency variation of the signals over time. A "chirp" refers to a timeframe of the signal where the frequency is increasing or decreasing within a certain bandwidth. This technique makes the signal significantly more resistant to interference compared to simply having two distinct frequencies which are pre-defined to represent a binary 0 or 1, and simply switching between them when transmitting the signal, an example of which can be seen in figure 2.2. The up and down sweeps in figure

2.1 visually reflect the bitstream and how it is encoded into radio frequencies.

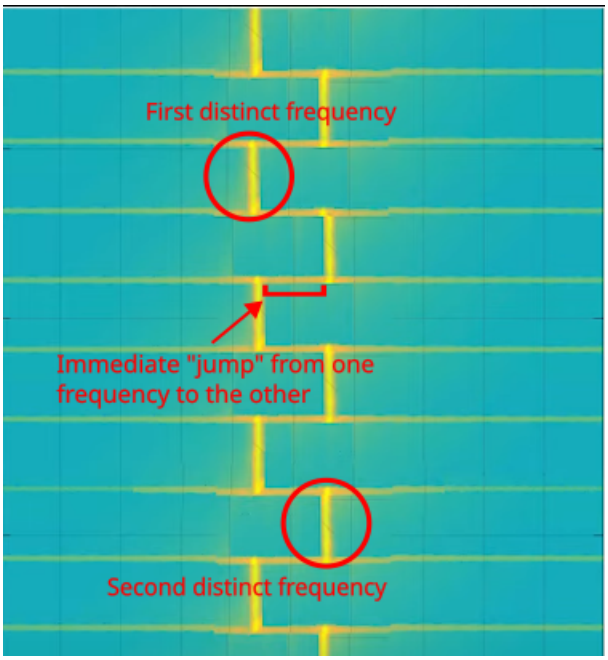


Figure 2.2: Example of having two distinct frequencies representing binary 0 and 1, and "jumping" between them instead of sweeping. [18]

Each character transmitted with LoRa lasts a duration determined by the spreading factor, defining how long each chirp lasts. As spreading factor increases, chirps last longer and have improved sensitivity at the expense of having a lower data transmission rate.

LoRaWAN builds upon the LoRa physical layer by adding a media access control (MAC) protocol. It defines how devices connect, how data is encrypted, and how the network is managed. LoRaWAN is designed for ultra-low-power operations, allowing devices to last several years on small batteries, and offers deep indoor penetration, which is particularly valuable in hospital settings where walls and medical equipment can attenuate signals [17].

To assess the suitability of LoRaWAN, it is important to compare it to other LPWAN technologies, including SigFox, NB-IoT, and LTE-M.

SigFox is a proprietary ultra-narrowband communication technology designed for transmitting small data payloads over long distances, while consuming small amounts of power. It operates on unlicensed Industrial, Scientific, and Medical (ISM) bands, and works best when the application requries one-way, infrequent transmissions. These attributes make it great for low-throughput applications, such as smart city metering, and environmental monitoring. Due to some issues with non-fixed environments such

as frequency inaccuracies and interference, SigFox works best when employed in fixed locations [19].

NB-IoT on the other hand operates in the licensed spectrum. Developed by 3GPP, it offers powerful indoor penetration, is reliable, and allows for connecting a large number of devices. It is mainly suited for applications needing regular (unlike SigFox), low data rate communication, like utility metering and building automation [20].

Once again developed by 3GPP, LTE-M supports higher data rates than NB-IoT and allows for voice communication using Voice over Long-Term Evolution (VoLTE). It ensures seamless handover between cell towers, which makes its use cases more dynamic and mobile, such as asset tracking, pet tracking, and point-of-sale devices [21].

Below is a table summarising their main advantages and disadvantages.

Table 2.1: LPWAN technologies - Advantages & Disadvantages [19, 20, 22, 23]

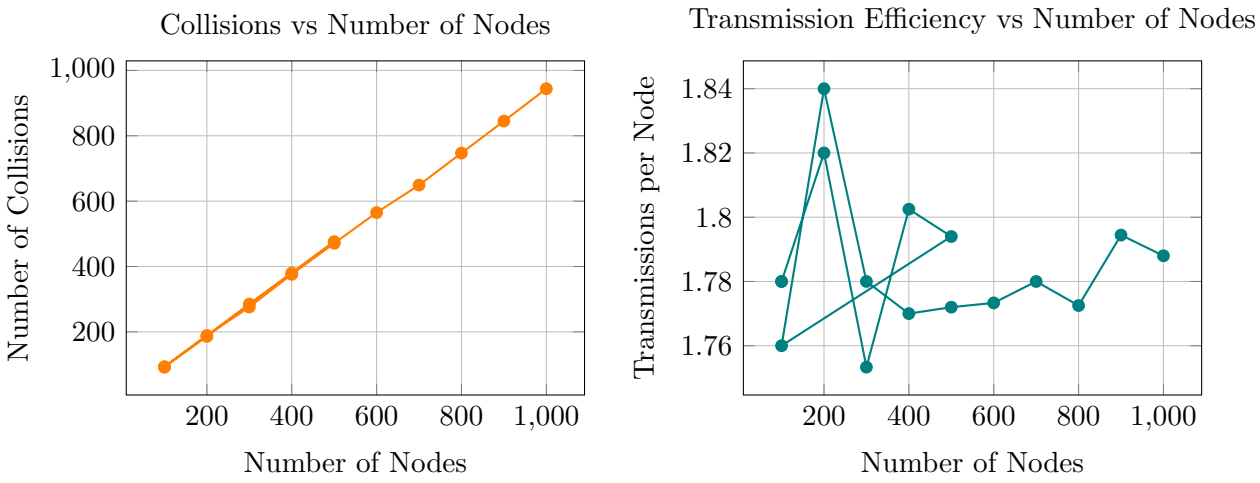
Technology	**Advantages**	**Disadvantages**
LoRaWAN	• Long range • Low power consumption • Adaptive data rates • Deep indoor penetration • Low cost	• Limited by duty cycle • Unsuitable for real-time or low latency applications
SigFox	• Lightweight protocol • Minimal overhead • Long battery life • Wide coverage • Underground support	• One-way communication • Limited data rates • One operator per country
NB-IoT	• High scalability • Robust QoS • Extended range • Structure penetration • Backed by operators	• Licensed spectrum costs • Lower data rates than LTE-M • No roaming support
LTE-M	• High data rates • Excellent coverage • LTE network integration • Efficient power use	• Higher costs • Firmware updates consume power • Not for high-volume data

Bluetooth is also widely used in medical IoT devices. While it offers short-range connectivity and is well-suited for personal health devices like heart rate monitors and themometers, its power consumption and range limit its usefulness for hospital-wide or remote patient monitoring. Bluetooth Low Energy (BLE) improves power efficiency and is increasingly integrated into wearable medical devices. As such, it will be used in this project to connect the main system with the medical sensors which take actual readings.

Testing

While the focus of this project was not to benchmark the LoRaWAN protocol itself, preliminary simulations were conducted to validate its suitability for post-operative monitoring applications. A single base station scenario was simulated using open-source software provided by Lancaster University [24]. The simulation was adapted to run on Python 3, and tests were conducted with 100 to 1000 node configurations, increasing by 100 each experiment. The parameters for the simulations where fixed as such: `AVGSEND = 100ms`, `SIMTIME = 2000ms`, and `COLLISION = 1` to enable full collision detection. The parameters used correspond to the simulated nodes transmitting small packets roughly every 100 milliseconds over a total simulation time of two seconds. Furthermore, `EXPERIMENT = 0` was used, which configures all nodes to transmit with the slowest LoRa modulation settings. The key objective was to confirm that LoRaWAN could handle multiple nodes transmitting small amounts of data at low power over long distances.

The first set of results explore how node density affects transmission quality:



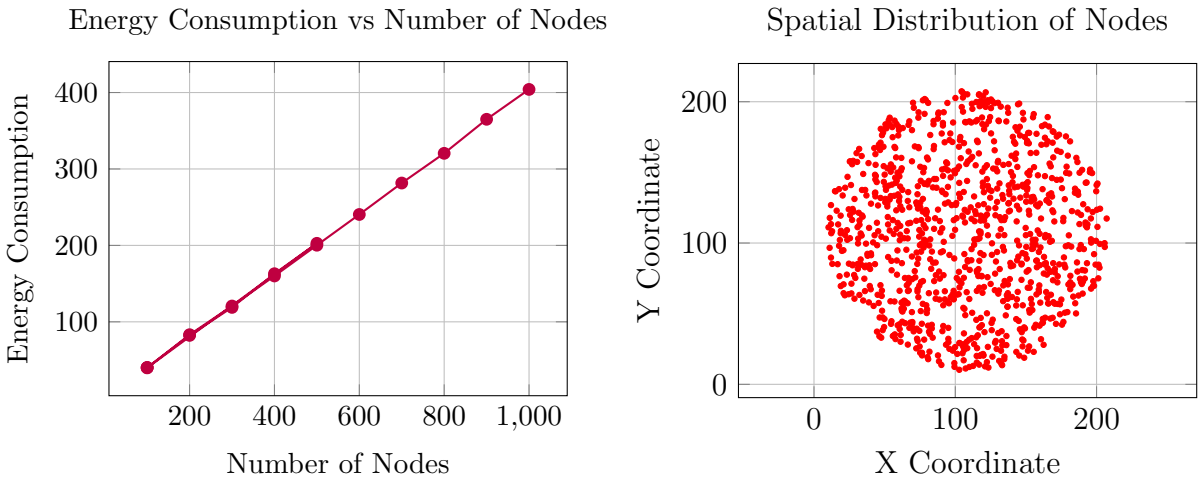
(a) Collisions vs Number of Nodes

(b) transmission Efficiency per Node

Figure 2.3: Impact of node count on number of collisions and transmission efficiency.

As per Figure 2.3a the number of collisions increases with the number of nodes linearly, something that is consistent with LoRaWAN's unslotted ALOHA-based access scheme. Regardless, transmission efficiency (Figure 2.3b) remained relatively stable, even if it did drop slightly around the 500 node point. This demonstrates LoRaWAN's robustness even under increased load.

The next two plots look at power efficiency against the number of nodes, and the physical spatial distribution of the simulated nodes.

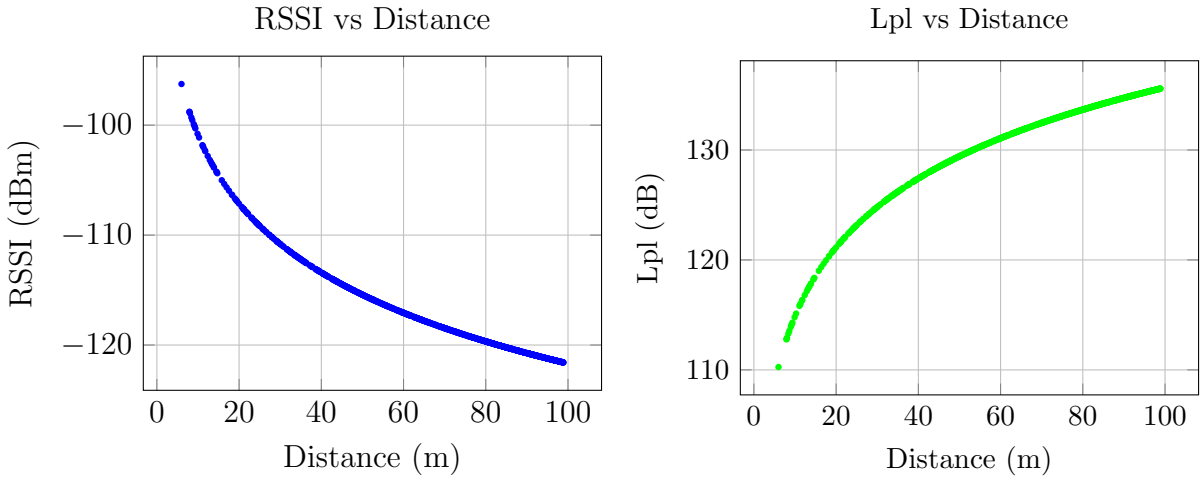


(a) Energy Consumption vs Number of Nodes (b) Spatial Distribution of Nodes

Figure 2.4: Energy scalability and simulation topology.

Figure 2.4a shows that energy consumption of each node scaled linearly once more with the number of nodes. This is to be expected as all nodes use the same transmit settings and a fixed sending interval. Provided duty cycles are properly managed, these results support LoRaWAN's use in battery-powered applications. A duty cycle is the time a device is permitted to transmit on a given frequency band, within a set time period [25]. The spatial distribution (Figure 2.4b) simply confirms that the simulation used a random deployment of the nodes across the simulated physical area, approximating a real-world environment such as a hospital ward.

Path loss and signal strength trends were also considered to better understand signal behaviour.



(a) RSSI values relative to distance from the gateway.

(b) RSSI vs Distance

(c) Path loss (Lpl) values relative to node distance from the gateway.

(d) Path Loss (Lpl) vs Distance

Figure 2.5: Signal strength and path loss trends relative to node distance from the gateway

Both path loss and Received Signal Strength Index (RSSI) degrade as the distance increases, which is to be expected based on theoretical radio propagation models. RSSI is a representation of the measurement of power level for a received radio signal. It is typically expressed in dBm, and as the values get more negative, it indicates a weaker signal.

Results showed that as the number of nodes doubled, the number of collisions also increased, which is consistent with LoRaWAN's access behaviour. In this context, a *collision* refers to the event where multiple decvices transmit at the same time, on the same frequency, and with the same spreading factor, at overlapping times, leading to signal interference at the gateway. When collisions occur, the gateway will likely have trouble demodulating the incoming transmitted packets, causing data loss.

Despite the increase in collisions, transmission efficiency per node remained stable, and energy consumption scaled predictably with the node count — highlighting LoRaWAN's suitability for applications requiring scalability, wide coverage, and energy efficiency, such as this project. Compared to Wi-Fi, LoRaWAN uses significantly less power and has a vastly longer communication range. Wi-Fi needs to have constant connectivity, and frequently send beacons out informing end devices of available access points and services [26]. These two facts mean Wi-Fi consumes substantially more power than LoRaWAN making it unsuitable for battery operated devices intended for long-term use (in the order

of years). Furthermore, Wi-Fi was designed for high-throughput applications in a limited range in indoor environments, unlike LoRaWAN which was designed for sparse, low data rate transmission covering large areas. As for GSM-based technologies like NB-IoT and LTE-M, while they do offer broader coverage and existing infrastructure backed and maintained by mobile providers, they come with significantly higher costs, greater power needs, and licensing requirements (meaning legally only mobile network operators can operate them, or a partnership with a mobile operator is requried), making them not as suitable for this application as LoRaWAN is. The comparison in Table 2.2 illustrates the advantages of LoRaWAN in terms of its efficient power consumption, cost, and flexibility in deployment and operation.

These outcomes support the choice of LoRaWAN as a communication backbone in the system, balancing scalability and energy efficiency in a medical monitoring context.

2.5 Commercially available medical devices

Because Bluetooth technology combined with technologies allowing integration with the cloud allow for continuous, real-time monitoring of vital signs including heart rate, blood pressure, respiration rate, temperature, and oxygen saturation, they have completely changed postoperative treatment. Such devices enable smooth data transfer to medical professionals, enabling prompt interventions and individualised patient treatment. Numerous Bluetooth-enabled medical devices are available on the market, with differences in performance, compatibility, and data access transparency. Since it dictates how easily these devices may be integrated with other healthcare systems and apps such as the one discussed in this thesis, the level of Bluetooth openness is especially important. I will examine a number of commercially available devices that demonstrate these characteristics in the sections that follow, emphasising their features and clinical application considerations.

Numerous commercially available medical devices can be used in clinical settings and are made for the continuous monitoring of vital signs. Selected devices are summarised in the table below, with an emphasis on those that have been certified for clinical use by the US Food and Drug Administration (FDA) or that bear the CE mark in accordance with the EU Medical Device Regulation. It is important to keep in mind that FDA approval

entails a more stringent examination procedure and is frequently thought of as a higher quality of clinical validation, even if the CE mark certifies regulatory conformity.

The table features the certification granted to the device, the measured vital signs, and its Bluetooth openness level. Each device listed is capable of measuring one or more of the following vital signs:

- **HR:** Heart Rate
- **BP:** Blood Pressure (may be traditional or cuffless)
- **RR:** Respiratory Rate
- **SpO_2:** Blood Oxygen Saturation
- **Temp:** Body Temperature
- **ECG:** Electrocardiogram

The term "Bluetooth openness" indicates how simple it is to access and incorporate data into customised solutions. Interoperability may be a constraint for customised IoT solutions because the majority of clinically certified devices function within proprietary ecosystems.

Table 2.2: Medically Certified Bluetooth Capable Devices

Device Name	**Manufacturer**	**Certification**	**Vital Signs it Measures**	**Bluetooth Openness**
GE Portrait Mobile	GE Health-Care	FDA	HR, RR, SpO_2, Temp	Closed
Berry Multi-Parameter Monitor	Berry Medical	CE (MDR)	HR, BP, SpO_2, Temp, ECG	Semi-open
Contec CMS8000	Contec Medical	CE (MDR)	HR, BP, SpO_2, Temp, RR, ECG	Closed
Biobeat Chest Patch	Biobeat	FDA, CE (MDR)	HR, BP (cuffless), SpO_2, RR, Temp	Closed
Caretaker4	Caretaker Medical	CE (MDR)	BP, HR, RR, SpO_2	Closed
Checkme Pro Vital Signs Monitor	Wellue (Viatom)	CE (MDR)	HR, BP, SpO_2, Temp, ECG	Semi-open
CardiacSense Watch	CardiacSense	CE (MDR)	HR, ECG, SpO_2, Temp	Closed
SIFVITAL-1.2	SIFSOF	CE (MDR)	HR, BP, SpO_2, Temp, Glucose	Closed
CARER HC-03 6-in-1 Monitor	CARER Medical	CE (MDR)	HR, BP, SpO_2, Temp, Glucose, ECG	Closed
Lepu Vital Signs Monitor	Lepu Medical	CE (MDR)	HR, BP, SpO_2, Temp, ECG	Closed

Even though a lot of devices fulfil the requirements for medical certification, as the

table shows, they frequently lack open Bluetooth interfaces. When incorporating them into custom IoT systems, this poses a problem, which will be discussed in the next section along with suggestions for dealing with it.

2.6 Integration with custom IoT systems

Because of the restricted or proprietary nature of their Bluetooth communication protocols, it is still very difficult to integrate commercially available vital signs monitors into custom Internet of Things (IoT) systems. As mentioned in the preceding section, the majority of devices offer little to no support for open data access, despite the fact that many of them are medically certified and give accurate physiological readings. Some, at most, provide semi-open APIs that are restricted to partner software or vendor-specific apps. This makes it more difficult to integrate seamlessly with specially designed systems that seek to independently gather, process, or send data. The impacts of these restrictions are examined in this section, along with possible interoperability tactics for medical IoT applications.

Challenges and Implications

Even though there are medically recognised vital sign measurement devices available, Bluetooth accessibility issues make it difficult to integrate them into custom IoT systems. The majority of medical equipment intended for professional usage rely on closed data ecosystems or proprietary Bluetooth protocols. The functionality is frequently confined to vendor-approved apps or limited APIs, preventing complete integration with customised microcontroller-based systems, even in situations where some Bluetooth access is offered.

This lack of openness presents a number of practical and technical difficulties. First, there are significant limitations on compatibility. Raw vital sign data cannot be directly accessed by custom systems that depend on local gateways or lightweight edge devices (like Arduino or ESP32) unless the device supports standard Bluetooth Generic Attribute Profile (GATT) profiles or offers a published Application Programming Interface (API). Consequently, the two main advantages of edge-based IoT architectures—real-time processing and on-device decision-making—become impracticable. Second, vendor lock-in develops into an extremely serious issue. System designers are left with no control over

data flow, privacy policies, and long-term data preservation when device data is only available via the cloud platforms or mobile apps of the manufacturer, which almost certainly makes it more difficult to comply with data protection laws like GDPR or HIPAA in clinical or research settings where data sovereignty, repeatability, and traceability are crucial.

Furthermore, development time and integration complexity also rise. Reverse engineering, unapproved Software Development Kits (SDKs), or intricate workarounds are frequently required in order to retrieve useful data from these devices. These endeavours are laborious, prone to mistakes, and possibly fragile, as they can fail with firmware changes or violate license agreements.

These difficulties show how the requirements for adaptable, responsive IoT-based monitoring solutions differ from those of authorised commercial devices. To overcome these constraints while maintaining system dependability and regulatory compliance, potential workarounds and integration techniques are examined in the immediately following paragraphs.

Possible Solutions

Developers of custom IoT systems must think of alternate ways to enable integration while preserving clinical dependability and compliance, given the limitations imposed by proprietary Bluetooth implementations in medically approved vital sign monitors. While there isn't a single solution that works for all, there are a few workarounds that could enable significant interoperability despite the existing constraints.

1. **Use of intermediate devices with vendor APIs**

   Data can be accessed by official desktop software or mobile applications in certain closed systems. In these situations, middleware bridges that communicate with the vendor software through open SDKs or REST APIs can be created, such as companion Raspberry Pi services or smartphone apps. These intermediary devices can then use standardised protocols to send the necessary information to customised IoT devices. This maintains data accuracy and honours the device's intended usage paradigm, even if it adds another layer of complexity and can cause latency to increase.

2. **Preferring semi-open devices to entirely closed devices**

   Vendors occasionally offer limited Bluetooth access via documented GATT profiles or custom services. Although they are not completely open, devices like the Wellue Checkme Pro provide restricted data access that can be adequate for everyday uses. When feasible, system designers should give priority to these semi-open devices and use reverse-engineered documentation or community-developed libraries to retrieve readings with the least amount of intrusion to prevent any legal problems regarding licensing agreements.

3. **Passive signal interception (may violate licensing agreements)**

   Certain specialised tools can be used to determine the transmission structure of the Bluetooth signals sent from closed sensor devices, and reverse-engineer the communication protocol in devices without access to an API or access point. Although this method can provide complete access to sensor data, it is not recommended for production systems or environments that need to adhere to legal or clinical requirements since it runs the risk of breaking user agreements and regulatory restrictions.

4. **Develop sensor systems in parallel**

   By constructing parallel sensor modules with clinically proven open hardware or certified discrete sensors (e.g., MAX30101 for SpO_2 and pulse, MLX90614 for temperature), it is also possible to entirely eliminate reliance on closed devices. This provides complete firmware-level control over sensing, communication, and data handling, but it is more taxing in terms of development resources. Full-stack customisation is made possible by the direct integration of such customised sensors with system infrastructure when paired with LoRa, BLE, or Wi-Fi transmission modules.

5. **Collaboration with vendors and enterprise partnerships**

   Organisations can contact vendors for OEM agreements or developer access when commercial devices are preferred. On request, certain manufacturers might provide enterprise SDKs or documentation, particularly for system integration pilot programs, academic research, or clinical trials. Even though it could take more time, this approach guarantees long-term support and compliance.

The expense, complexity, and legality of each of these workarounds are trade-offs. The best strategy frequently calls for a hybrid system, in which open or semi-open alternatives are utilised for system development or simulation, while medically certified but closed devices are utilised for clinical validation. In the end, market forces and regulatory pressure may push manufacturers towards greater openness in upcoming device generations as the need for patient-specific, interoperable healthcare systems increases.

2.7 Summary

There are technological and legal issues with integrating medically approved vital sign monitoring devices into unique IoT systems. Even if a lot of commercial devices provide precise and trustworthy measurements, smooth integration is hampered by their closed or semi-open Bluetooth implementations, which restrict direct access to raw data streams. These restrictions may affect custom healthcare platforms' adaptability, reactivity, and independence. However, there are a number of practical alternatives, such as using semi-open devices, developing parallel sensing modules, using middleware programs, or forming direct relationships with manufacturers. For IoT healthcare solutions to be scalable, interoperable, and patient-tailored, these integration hurdles must be removed.

Chapter 3

Analysis and Design

3.1 Introduction

This chapter will present the analysis and design porition of the project. Identified are the system users and their needs, which are then translated into technical requirements, and finally the system's functional and architectural design is described. Several diagrams are included to aid with visualising the behaviour, workflow, and structure of the system.

3.2 User Needs and Requirements

When developing any technology it is imperative to have a solid foundation upon which to build on and refer to throughout the development process. It should help provide a clear understanding of the problem, the needs and expectations of the people who will be using it, and give some insight into how the original problem will be solved in the implementation stage. In the case of this project, the users include nurses and doctors, and the patients who will be using the device. There are several different users each with different required levels of control and functionalities, therefore it is crucial to translate their needs into measurable and clearly defined requirements that will steer the design and eventually implementation of the system.

The following pages present two key tables accomplishing that task: the **User Needs table** (Table 3.1), and the **User Requirements table** (Table 3.2). The User Needs table outlines what the system should accomplish in simple terms, similar to how an average non-technical person might describe what they need to solve their problem. Each row

consists of a unique **ID** for that particular need, a brief **Description** of what is needed, and a **Source** which represents what part of the system this need might be addressed in. The User Requirements table expands upon the User Needs table by translating each need into one or more clear and actionable system requirements. These requirements can be measured as satisfied or not in the final system. Each requirement has a unique **Req. ID**, a **User Requirement Name** giving a very short overview of what that requirement is supposed to do, and a **Description** of the requirement with more specific technical details such as how it might be implemented, what specific numeric constraints are present for it, or what exactly is required to satisfy it from a technical perspective. Following these are the **Justification/Comment** and the **Reference** columns. The Justification/Comment column expands a bit on the description and states *why* some of the technical details are as they are, or why the requirement was necessary in the first place. The Reference column links the User Requirements Table to the User Needs table by stating what user need (or what multiple needs) a particular requirement is intended to satisfy.

These tables together give a high-level perspective of what the system must achieve, with some justification as to why certain decisions where taken instead of others.

Table 3.1: User Needs Table

ID	**Description**	**Source**
UN-01	Accurate capture of data from various medical devices in order to monitor patient status	Arduino
UN-02	Simple pairing of the device to other medical devices through Bluetooth in order to easily setup and guarantee reliable data transfer	Arduino
UN-03	Updating/setting of limits/conditions for different measurement types in order to customise each device to a particular patient's needs	Arduino/ Cloud
UN-04	Process data according to preset conditions in order to send only relevant data to the cloud	Arduino
UN-05	Storing of data in the short-term in the case of high network congestion to prevent loss. (Only store after processing if values are abnormal/relevant to be sent to the cloud)	Arduino
UN-06	Alert medical staff of patient status if critical or abnormal reading gathered in order to allow them to check on said patient in person	Cloud
UN-07	Alert users about device status in order to allow them to keep it in a state where it is running normally	Arduino
UN-08	Provide a general picture about the patient's status daily in order to track it over time (daily averages viewed in context of weeks, months, years)	Cloud

Table 3.2: User Needs Table

Req. ID	**User Requirement Name**	**Description**	**Justification/Comment**	**Reference**
UR-01	Temperature Reading	5 times/12 hours Accuracy: to nearest 0.1°C	5 readings in a single 12 hour period should be enough to detect any changes in temperature early enough after they get out of bounds	UN-01
UR-02	Blood Pressure Reading	1 time/12 hours Accuracy: ±3mmHg for both systolic and diastolic	The recommended number of times to measure BP is once per day	UN-01

Req. ID	User Requirement Name	Description	Justification/Comment	Reference
UR-03	Heart Rate Reading	3 readings spaced 2 minutes apart / hour, and keeping the average of the 3 readings as final BPM value Accuracy: ± 5 BPM	In a clinical/hospital environment there would be constant EKG measurement so there is no reason medically not to measure heart rate as often as possible. However to conserve battery, avoid network congestion, and keep the processor constantly working, 3 readings averaged on the hour is sufficient for this system	UN-01
UR-04	Bluetooth Pairing with Medical Devices	Visual interface to display BT connection status, and allow force disconnecting a device which has connected if it is not a medical device	Pairing with a device should be incredibly easy seeing as a likely user group will be elderly individuals	UN-02

Req. ID	User Requirement Name	Description	Justification/Comment	Reference
UR-05	Per Vital Sign Configuration	Visual interface to be able to select a vital sign that will be measured, and set the upper and lower acceptable limits for it. This must persist across power loss and general restarting of the device. In addition, the limits must make sense in the context of the vital sign they relate to. For example it should be impossible to set an upper limit of 60°C for temperature	Each patient has different medical needs and as such will need customised ranges for the acceptable readings. Furthermore, the patients should not be required to have to set these limits in the event of a power loss of the device or device restart	UN-03

Req. ID	User Requirement Name	Description	Justification/Comment	Reference
UR-06	Data Processing and sending to the cloud	Process each data reading in the context of the device type's preset limits (lower/upper). Readings which are below/above the limits will be added to a queue. The front data will be sent to the cloud using the LoRaWAN protocol. Once confirmation is received that the data is successfully received, it will be dequeued and the next data (if it exists) will be transmitted. Allow up to 3 attempts before considering that there is a problem with the device/connection to the network and alerting the user.	Data that is within the acceptable limits does not need to be sent. The queue's purpose is to ensure any critical readings get to the cloud platform where the medical professionals can respond accordingly.	UN-04 UN-05

Req. ID	User Requirement Name	Description	Justification/Comment	Reference
UR-07	Notification of medical staff for abnormal reading	After sending data through the LoRaWAN protocol to the cloud, create some kind of notification for the medical staff in the form of of an email or "ticketing" system	A popup notification is not sufficient because it might be missed	UN-05 UN-06
UR-08	Device status information	Constantly monitor device status (BT connection, state) and display this information to the user	Due to the fact that status information must always be visible, and we do not want to occupy the LCD screen with it all the time, the information will be available through coloured LED lights	UN-07
UR-09	Patient status tracking over time	Regardless of whether an abnormal reading is gathered, at fixed times of the day and for a set number of times, take readings and send them to the cloud even if they are not abnormal.	Patient history could be vital for providing medical staff with insight into the patient's condition, however there is no need to take more than one non-abnormal reading per day	UN-08

3.3 Use Cases and System Interactions

The Vital Sign Monitoring System allows for on-demand monitoring of key physiological parameters in postoperative patients, enabling prompt medical intervention and remote supervision. Medical personnel like doctors and nurses, as well as patients are the intended users of the system, and they engage with the it in several ways.

3.3.1 Medical Staff Use Cases

- **Set Vital Sign Thresholds Remotely**

  Each patient's vital signs have upper and lower limitations that can be set by medical personnel remotely. By using these thresholds, out-of-bounds readings are automatically flagged and sent to the cloud platform.
- **Send Alert to Take Reading**

  The system can be manually prompted by a physician or nurse to ask the patient for a new measurement.
- **View Readings on Cloud Platform**

  Medical staff can view current and past readings remotely, via a web dashboard.
- **Receive Notification if Patient Status Abnormal**

  The system notifies the clinician when a vital sign is below or above the predetermined threshold.
- **Take Reading from Medical Sensor Device**

  Medical staff can initiate a reading automatically from the cloud platform. If there is no medical sensor connected with Bluetooth at that time, it will switch to prompting the user to connect one and take the reading themselves.

3.3.2 Patient Use Cases

- **Receive Alert to Take Reading**

  When a measurement is required, the patient is prompted (both visually through the LCD screen and the LEDs, and audibly through the buzzer) that a reading is requested and what vital sign they should measure.
- **Control Medical Sensor Device Bluetooth LE Connection Status**

  Can connect or disconnect vital sign sensors to the device through Bluetooth LE.

- **Take Reading from Medical Sensor Device**

  Patients can manually take readings at times of their choosing, and take readings when directed to remotely by medical staff.

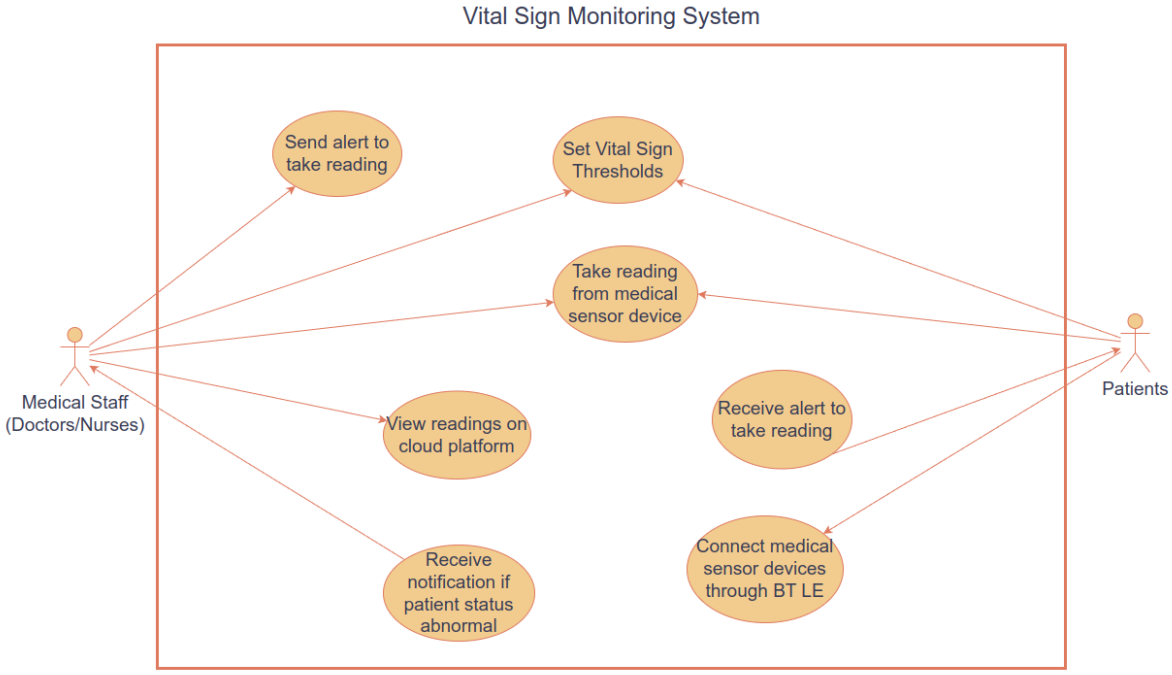


Figure 3.1: How medical staff and patients each interact with the system

Following these exchanges, the system behaviour depicted in Figure 3.1 shows how the two user categories and the underlying monitoring functions are related. Every user role has a different workflow, as the diagram illustrates. Notably, the system will ask the patient to connect a compatible device and take the reading manually if medical staff requests a reading but there isn't a Bluetooth-connected medical sensor available at the moment. However, the use case diagram does not depict this internal fallback behaviour because these diagrams are meant to depict the actions that each actor can take that are visible to the outside world, not the internal logic or decision flows of the system. Nonetheless, this system ensures that clinical workflows are robust even when there are disconnections and that monitoring capability is maintained.

In conclusion, the system outlines the responsibilities of both patients and medical staff clearly, and takes into account situations where a reading is requested but there is no device connected to take it. Patients can also take readings throughout the day if they so desire, but when there is a remote request from a clinician it can not be ignored.

The system architecture and behavioural design, including state diagrams and the circuit diagram, are shown in the next section.

3.4 System Architecture and Design

The architecture and design of the system are described in this section. For long-distance, low-power data transmission and centralised monitoring, the system includes a microcontroller-based LoRa communication unit. To gather data from the vital sign sensors, a Bluetooth Low Energy module is used, and for the communication between the device and the cloud a LoRaWAN gateway is set up to transmit to TheThingsNetwork. The goal of the design is to make it capable of communicating with BLE devices as simply as possible (something that due to the current closed nature of commerically available medical BT devices is not as simple as would be ideal), while utilising LoRaWAN's low-power features to allow for scalable, remote vital sign collecting, which is perfect for post-operative settings or locations with inadequate infrastructure.

Three main phases make up the system design: data collection (using the Bluetooth module to collect readings from medical sensors), processing of said data and finally transmission using an Arduino-based LoRa node. To tie everything together, cloud integration (using a LoRaWAN gateway and backend) will consist of a simple platform to view data and send commands to the device from a regular browser.

A high-level overview of the system's design is shown in Figure 3.2. Blood pressure sensors, temperature sensors, and heart rate sensors are examples of medical Bluetooth capable devices that connect to a central unit (an Arduino-based microcontroller with a LoRa module) via Bluetooth LE. This central unit acts as a command centre and filter at the same time, where it filters out read data and only sends that which exceeds the set thresholds and is therefore considered critical data for a particular patient. It ignores data within bounds in all but a few cases — those being when medical staff specifically request a reading, or the reading is one of those that are to be taken at set times every day.

This clearly compartmentalised architecture allows for separation of responsibilities: sensor management and handling of data is done locally, while the data is stored and analysed on the cloud platform. The cloud platform is also responsible for sending commands

to the device when required by the medical staff.

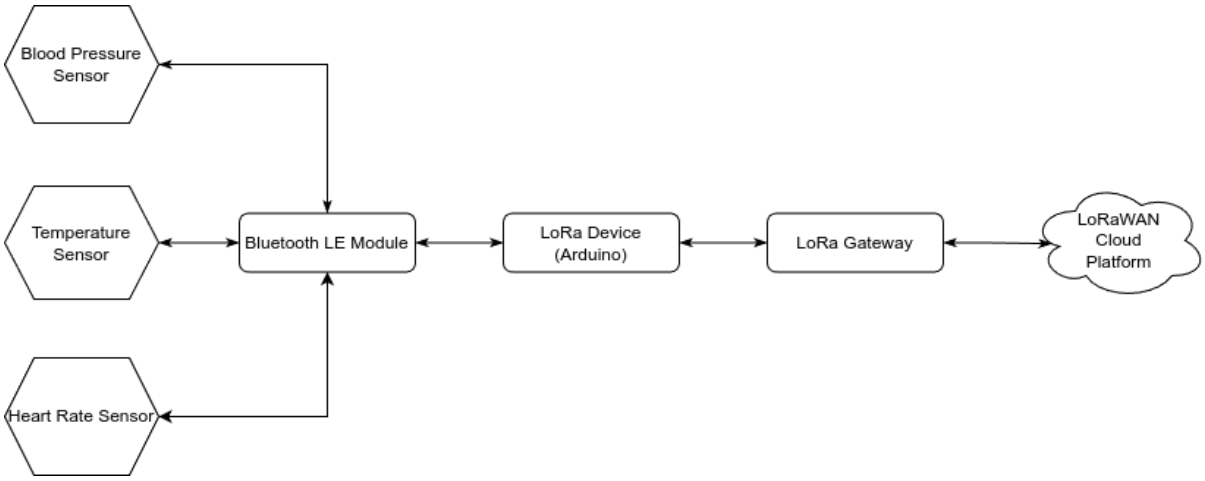


Figure 3.2: A high level view of the system architecture

On the right side of Figure 3.2 is the LoRa Gateway which is responsible for bridging the device and the cloud platform.

The internal architecture of the embedded system is depicted in Figure 3.3, which also highlights the hardware components' interactions on the Arduino-based microcontroller. The particular modules utilised in the finished implementation are shown in this diagram along with their functions in data collection, user interaction, processing, and communication.

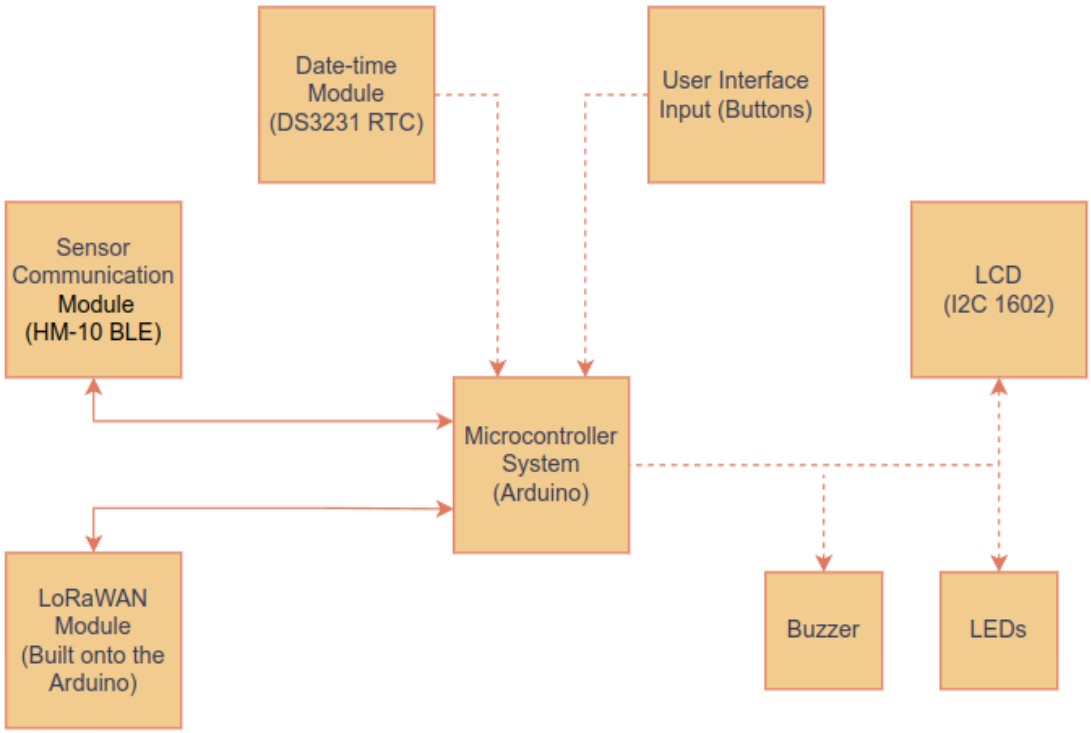


Figure 3.3: Block diagram showing relationships between high-level components of the system

The core of the system is the Arduino microcontroller, which controls all data flow and peripheral interactions.

Components for Sensing and Input

- Module for Sensor Communication (HM-10 BLE): Communication with external medical devices, such as temperature, heart rate, and blood pressure sensors, is managed by this Bluetooth Low Energy module. This module is used by the Arduino to read data from the paired sensors.
- Date-Time Module: Accurate timestamps for sensor readings and intitiation of data reading at pre-programmed times of the day is provided by a real-time clock module (DS3231 RTC). This is crucial for monitoring vital sign trends over time. This data is read by the Arduino through I^2C.
- User Interface Input (Buttons): The patient can initiate actions, including entering reading mode and waiting for a reading or exploring menus on the display, by pressing buttons. This provides simple interaction without the need for additional devices.

Output and Feedback

- LCD Display: Shows timestamps, sensor data, and UI prompts in real time. This display uses very little GPIO thanks to the I^2C protocol.
- Local alarms are provided by the buzzer and the LEDs. System states, such as connected or disconnected, reading, processing, or errors can be represented by LEDs, and in the event of unusual readings or the need for user action, their attention can be grabbed by the use of the buzzer.

Transmission and Communication

- LoRaWAN Module: This module manages wireless data transfer over great distances and is integrated into the Arduino board. After being taken and formatted, a reading is sent to the gateway via LoRa and uploaded to the cloud.

The system's behaviour can be varied based on the context and connected sensor thanks to its modular design using a Finite State Machine. Additionally, separated troubleshooting, streamlined development, and simpler future extensions (such incorporating SpO_2 sensors or supporting more sophisticated interfaces) are made possible by the separation of the communication, interface, and alarm modules. Next we will discuss the FSM that allows for this modularisation.

State Machine Diagrams

As soon as the system is turned on, it goes into the DISCONNECTED state. Awaiting user or environmental input, this state acts as the initial idle condition. In this state, two main conditions are assessed:

- Without additional user interaction, the system switches straight to the CONNECTED state whenever a Bluetooth connection is discovered. When a compatible medical sensor is already activated and within range, this guarantees a smooth interaction.
- As an alternative, the system enters the SETUP state when the user chooses the "Setup" option from the interface, enabling the setting of vital sign thresholds before any sensors are connected.

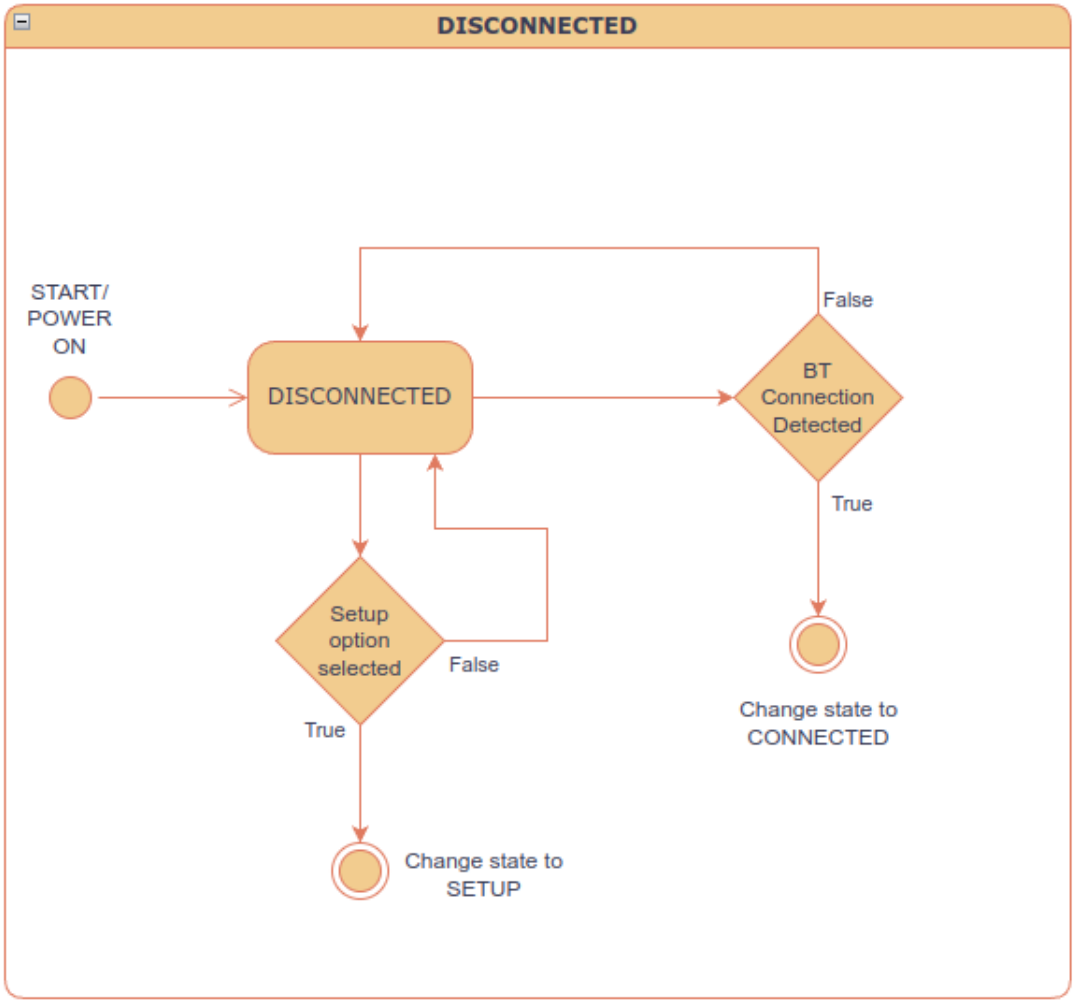


Figure 3.4: State diagram for the DISCONNECTED state

Whether waiting for an automatic connection or permitting pre-configuration prior to active use, the decision structure guarantees flexibility and user-friendliness in the beginning stages of powering on (not requiring the user to explicitly enter the CONNECTED state if a device is already connected). The system can go on to measurement and data handling capability after configuration is finished or a device connects.

This flow can be seen visually represented as a state diagram in Figure 3.4 above.

After a successful Bluetooth Low Energy (BLE) connection has been made with a medical sensor device, the CONNECTED state serves as the system's primary operating mode. In this condition, the user can take measurements or set up system parameters and has complete access to the system's essential features.

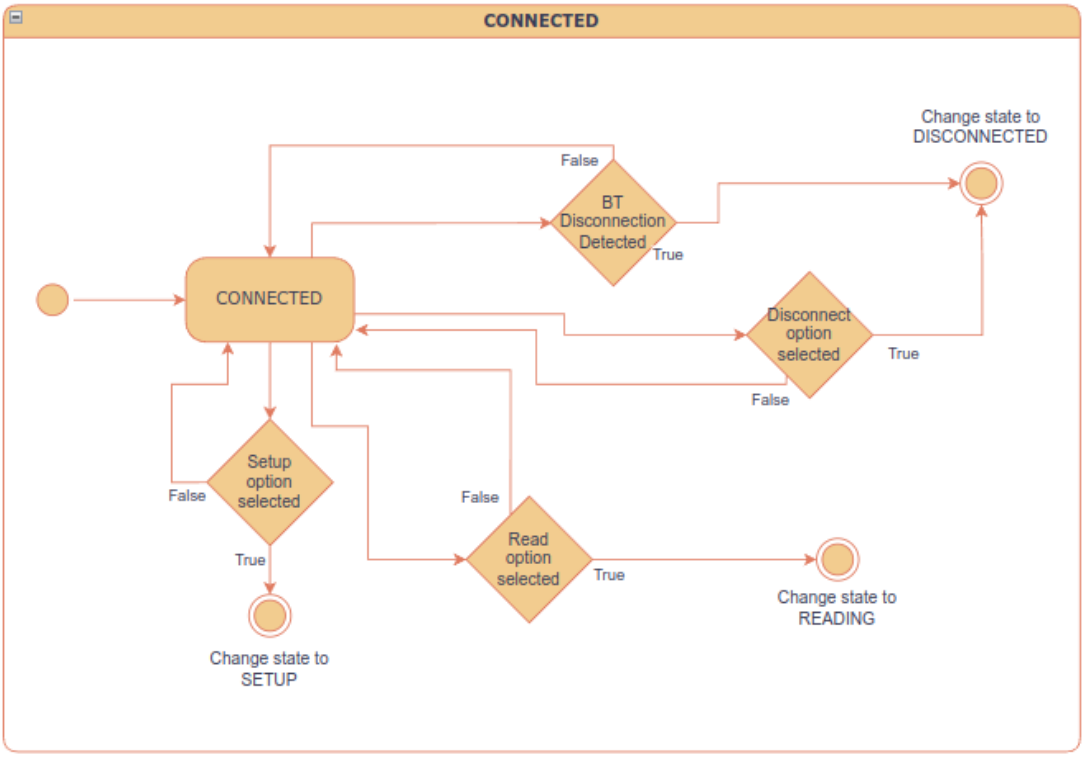


Figure 3.5: State diagram for the CONNECTED state

The system offers three options from this state:

- Setup: If the user selects the Setup option, the system enters the SETUP state, enabling the setup of vital sign thresholds.
- Read: Choosing the Read option puts the device in the READING mode and starts the process of gathering vital sign information from the linked medical sensor.
- Disconnect: The system returns to the DISCONNECTED state if the Disconnect option is chosen or if a Bluetooth disconnection is automatically detected. This guarantees that the availability of a linked sensor is always appropriately reflected by the system.

By ensuring that the system responds to user input as well as environmental changes (such a loss of connectivity), the state diagram preserves the system's resilience and user friendliness. Thus, the CONNECTED state acts as the centre for processes related to both configuration and data collection.

The interface for configuring the system's vital sign thresholds is accessible during the SETUP state. Depending on whether a Bluetooth-enabled sensor is currently cou-

pled with the system, one can enter this state from either the DISCONNECTED or CONNECTED states.

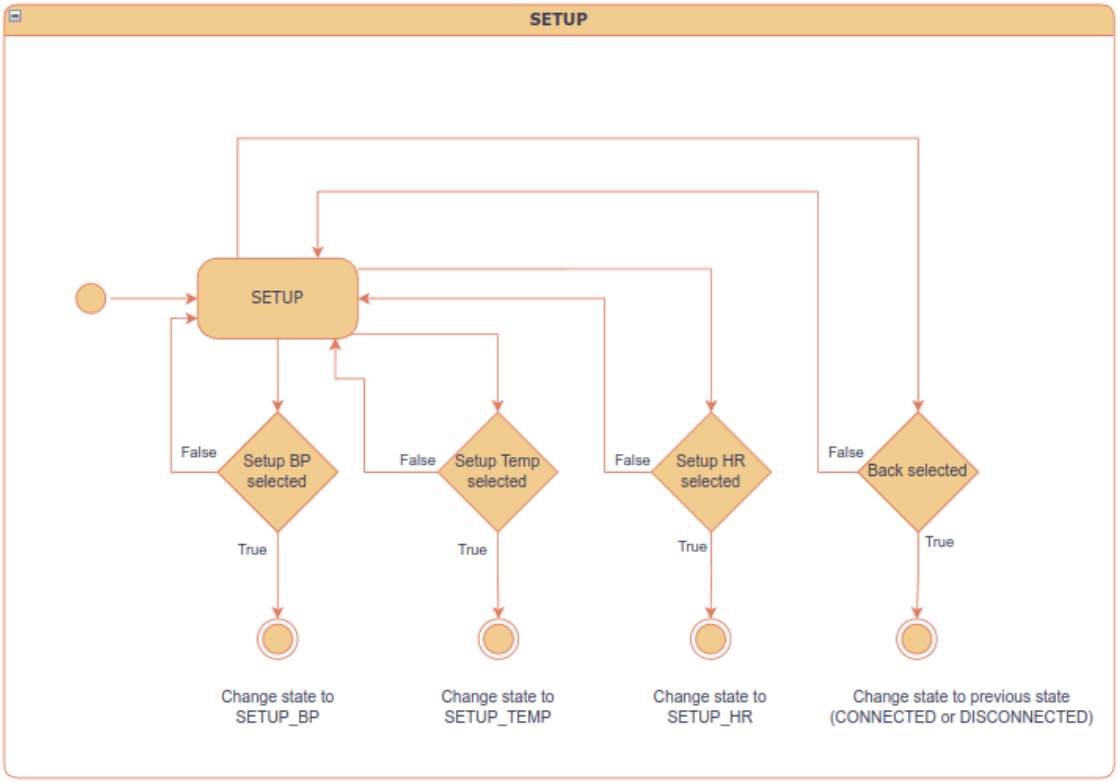


Figure 3.6: State diagram for the SETUP state

As is visible from Figure 3.6, after entering the SETUP state, the user is given four options:

- Setup BP (blood Pressure): This leads to the SETUP_BP state, where the user can set the minimum and maximum systolic and diastolic thresholds accordingly.
- Setup Temp (Temperature): Allows the user to set the minimum and maximum temperature values by leading to the SETUP_TEMP state.
- Setup HR (Heart Rate): This leads to the SETUP_HR state where the user can set the minimum and maximum values for heart rate.
- Depending on where the SETUP state was called from, selecting the "Back" option will return the user to the CONNECTED or DISCONNECTED accordingly.

By serving as a routing hub, this state makes it possible to set clinical thresholds for each patient or surgery type. Each configuration option leads to a specific sub-state that handles validation and data entry logic; no threshold adjustments are made within this

state itself.

The diagram below (Figure 3.7) goes into detail for each of the setup sub-states.

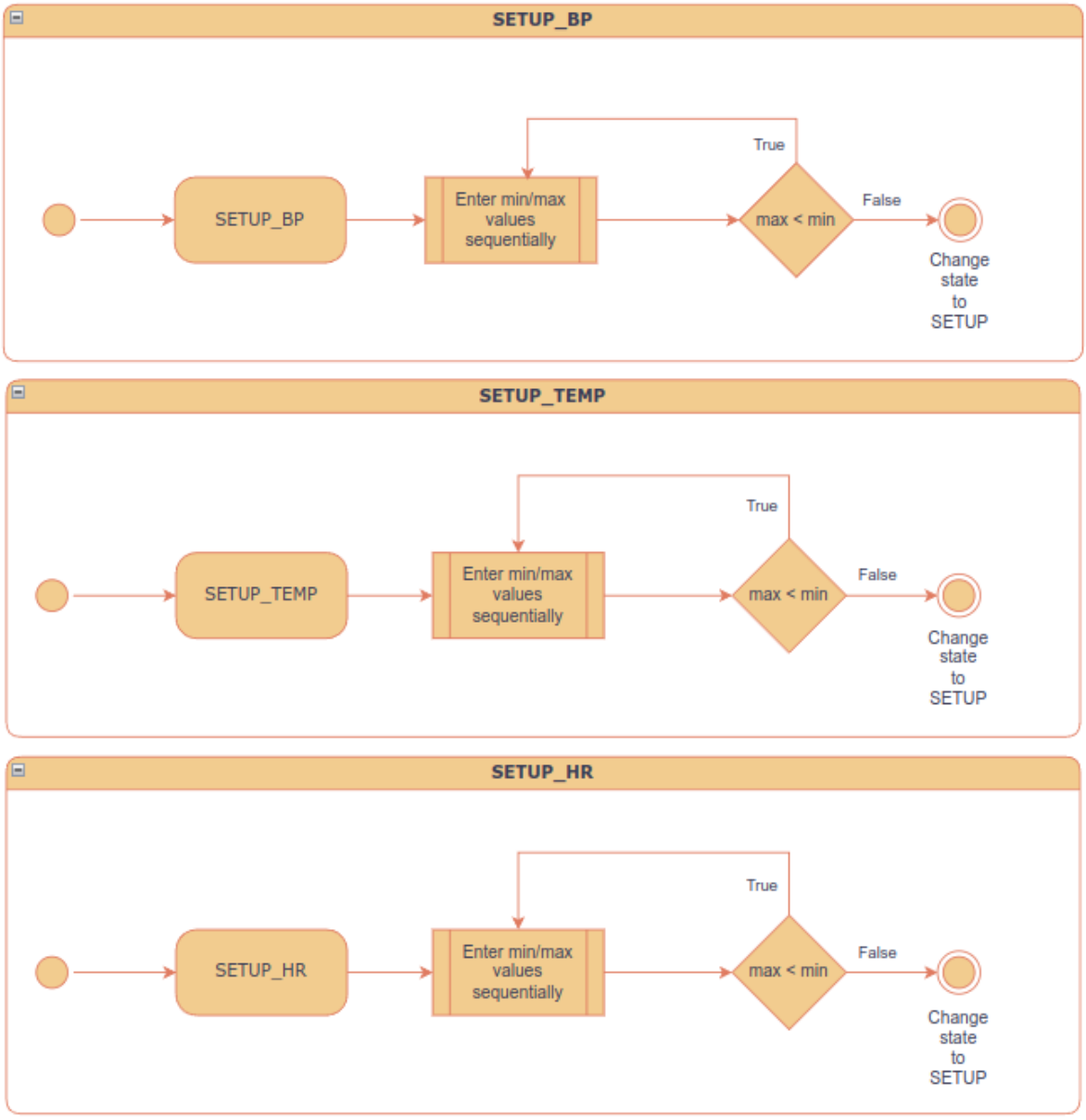


Figure 3.7: State diagram for the SETUP sub-states

SETUP_BP, SETUP_TEMP, and SETUP_HR are specifically responsible for setting the lowest and highest acceptable values for a given vital sign. The user is prompted to enter the values one after the other. To make sure the minimum is less than or equal to the maximum, each pair is separately checked. The system will re-prompt for both values if any validation is unsuccessful.

- All of these states are essentially identical in structure and operation. They ask the user to enter a minimum and maximum value, check that the maximum is not less

than the minimum, and if it is not return the user to the SETUP state.

These sub-states guarantee that threshold values used for measurement and analysis are reasonable, and enforce that thresholds are correctly set.

The READING state (Figure 3.8) is in charge of receiving vital sign information from a connected Bluetooth medical sensor. The system will wait for incoming data from the sensor device until something is received.

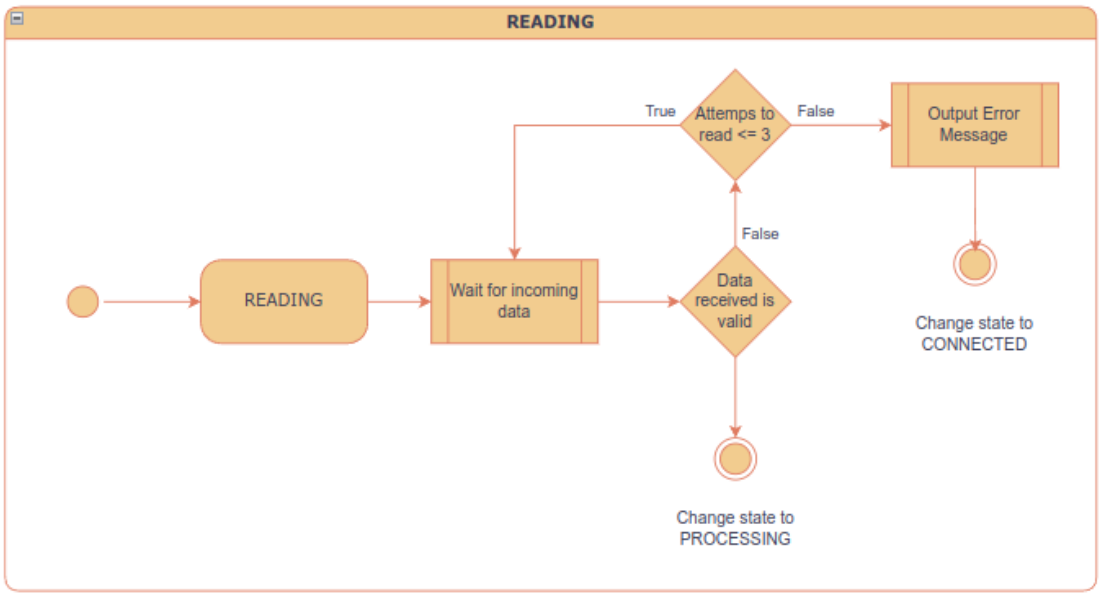


Figure 3.8: State diagram for the READING state

Following the receipt of data, it must confirm that it's valid:

- The system moves on to the PROCESSING state for more analysis if valid data is received.
- The system counts the number of read attempts if the data received is deemed invalid. Up to three attempts to read valid data are allowed.
- The system rejects the data and waits for a new reading if the number of attempts is less than or equal to three.
- The user is presented with an error message and the system returns to the CONNECTED state if the third attempt is unsuccessful as well.

The vital sign reading obtained in the previous READING state must be assessed by the PROCESSING state (Figure 3.9). The system determines if the newly received data is within the established acceptable bounds after it has been received. These thresholds,

which are unique to each type of vital sign (e.g., blood pressure, heart rate, temperature), are established by the medical team or patient during system setup.

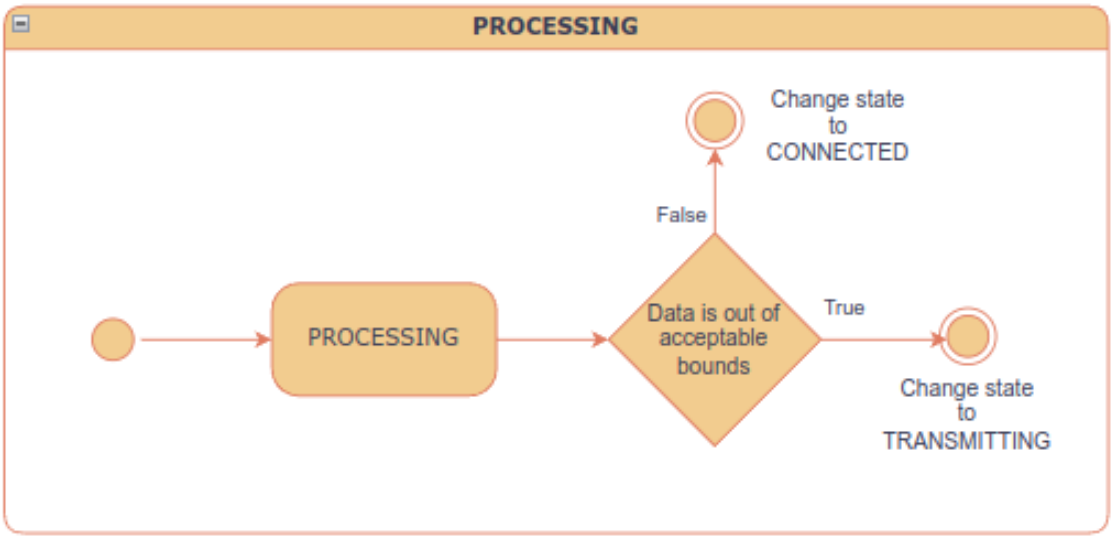


Figure 3.9: State diagram for the PROCESSING state

The system returns to the CONNECTED state, where additional user input or new readings may be analysed, if the data falls within acceptable bounds and it decides that no immediate action is necessary. This is done to not flood the network with unnecessary traffic.

The data may, however, point to a potentially hazardous or abnormal patient condition if it drops below or goes above the established threshold range. The system then switches to the TRANSMITTING state, where it tries to transmit the abnormal reading to the cloud platform so that medical professionals can monitor it remotely.

This design ensures that alerts are raised only for clinically relevant events, reducing false alarms while maintaining responsiveness to potentially critical conditions, and reduces load on the network by only sending relevant data to the cloud.

Abnormal vital sign readings must be sent via LoRaWAN to the cloud platform while in the TRANSMITTING state (Figure 3.10). This only happens when a reading is outside the configured acceptable range, which indicates a possible health concern. This is determined by the PROCESSING state.

When the system reaches this state, the LoRa module on the Arduino starts the data transfer, after which it awaits confirmation from the cloud platform that the data was

successfully received.

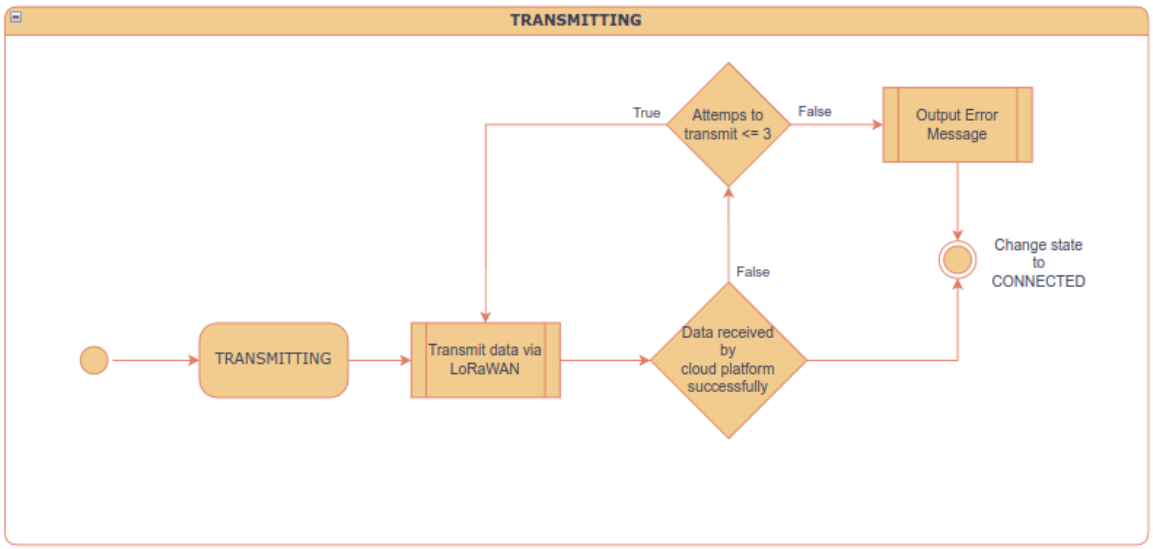


Figure 3.10: State diagram for the TRANSMITTING state

Data transmission is deemed successful if confirmation is obtained in the form of an acknowledgement from the cloud platform, and the system goes back to the CONNECTED state, prepared to accept more inputs.

The system will attempt to deliver the data up to three times in the case of a failed transmission. The system notifies the user of the failure with an error message and returns to the CONNECTED state if, after three unsuccessful attempts, the data is still unable to reach the cloud platform. In addition to keeping the system from becoming permanently trapped in a failed state, this three-retries process helps guarantee reliability in wireless data transmission, particularly in settings where LoRaWAN connectivity may be erratic.

Circuit Schematic

The hardware circuit schematic for the system is displayed in Figure 3.11. The system heart is the Arduino Leonardo microcontroller, which manages interfacing with peripheral components such the HM-10 Bluetooth Low Energy (BLE) module and the DS3231 real-time clock connected via I^2C. While the BLE module enables wireless connectivity to external sensor devices, the RTC maintains precise timekeeping for timestamping collected data, among other tasks. The three-button interface allows for very basic user interaction in the form of menu navigation and threshold value entry. Four status LEDs—green,

yellow, red, and blue—show the device's condition and Bluetooth connectivity. Additionally, a buzzer provides audible alerts for abnormal readings or transmission failures, and a 16x2 LCD display (as with the RTC, also connected via I^2C) is used to display menus. To guarantee a clear digital signal, a pull-down resistor is attached to each button, and 220 ohm resistors are used to limit the current flowing through all of the LEDs. This schematic represents a small, power-efficient design.

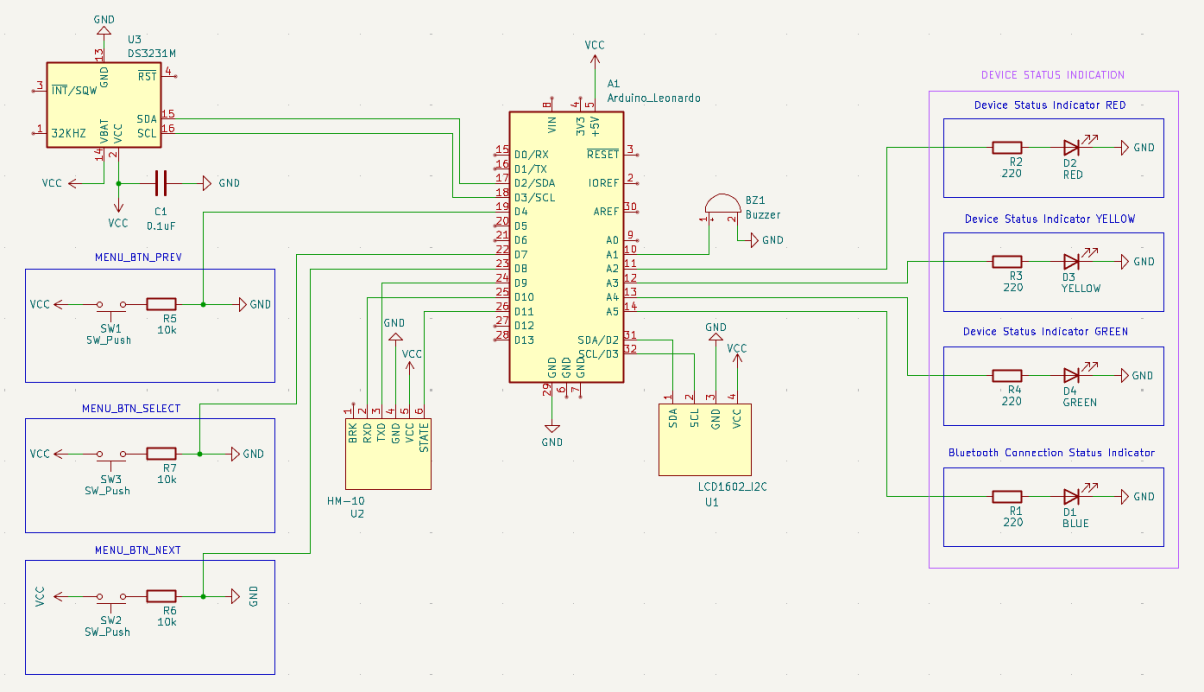


Figure 3.11: Detailed circuit shcemtatic of all all components in the device

3.5 User Interface Design

The device's user interface was made to be simple and easy to use, accommodating embedded technology limitations while guaranteeing medical staff and patients alike could easily interact with it. Push buttons, an LCD screen, a buzzer, and a group of LEDs make up the interface's four primary elements. Each one has a distinct function in communicating system status while occasionally their feedback cues may overlap to tell the user the same thing in different ways (e.g. Error displayed on the LCD while simultaneously a buzzer tone sounds corresponding to that error).

Input Mechanism: Push Buttons

Three push buttons are used by the system to facilitate option selection and menu navigation. These buttons are used to confirm selections and navigate through the available options (such as "Setup", "Read", or "Disconnect"). Only valid menu options are displayed at any given point thanks to the context-dependent functionality based on the current system state (as described in the state machine). Debouncing is handled in software to prevent unpredictable inputs.

Visual Feedback: LCD Display

A single 16x2 LCD module gives visual feedback to the user. This display, which shows menus, configuration options, and error alerts, acts as the primary means of communication between the user and the system. For instance, it shows any errors that may occur, or displays messages indicating to the patient that a request has come in from the cloud platform by a clinician. It is ideal for portable medical equipment due to its low power consumption, compact footprint, and microcontroller compatibility.

Visual Feedback: Status Indicator LEDs

Four LEDs are incorporated inside this device to give instantaneous, nonverbal feedback about connectivity and system status. Without looking at the LCD interface, users can rapidly determine operational conditions at a glance thanks to their clear and extremely simple nature. A different system state (not referring to states in the context of the state machine but rather operational conditions), such as regular operation, warnings, critical alerts, or connectivity status, is linked to each LED. By providing a quick overview of vital sign evaluations and device performance, this improves usability in clinical settings.

Auditory Feedback: Buzzer

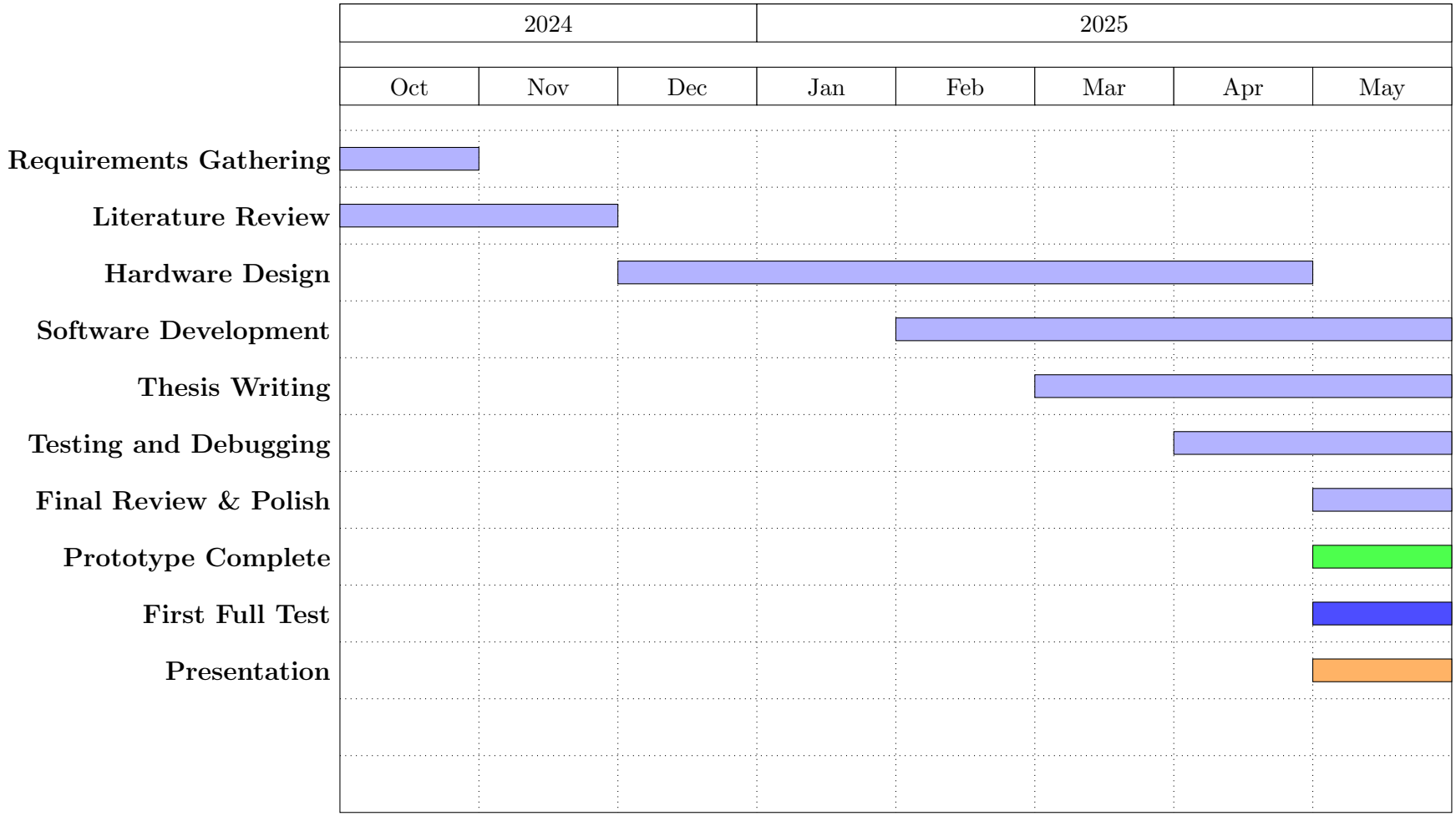
Feedback is also provided by the buzzer on the device, for instance for abnormal measurements or any possible errors that may occur. For example, the buzzer emits certain tone patterns to notify either medical staff or the patient (if they have been released from hospital and are using the device for continuous at-home monitoring) if three consecutive read attempts are unsuccessful or if a vital sign reading deviates from permitted limits and must be transmitted. In situations where visual cues could be overlooked or are

otherwise unavailable (e.g. if the patient is blind), the duality of having output both visually and audibly is vital.

The user interface design attempts to strike a balance between practicality and simplicity. It increases the system's dependability in a medical setting by reducing the need for user training and guaranteeing that important information is communicated quickly, while also providing the necessary functionality to control the system efficiently.

3.6 Project Planning

To make sure that development, testing, and documentation progressed in a timely and systematic manner, the project execution was divided into distinct phases with some phases overlapping if it made sense to do so. The Gantt chart below shows the timeframe, which spans the months of October 2025 through May 2025. The pace and priority of the system's development and academic submission process are reflected in this timeline.


2024
2025
Oct
Nov
Dec
Jan
Feb
Mar
Apr
May
Requirements Gathering
Literature Review
Hardware Design
Software Development
Thesis Writing
Testing and Debugging
Final Review & Polish
Prototype Complete
First Full Test
Presentation

- Gathering Requirements (October 2024): This initial phase concentrated on determining the system's objectives and functional requirements in light of postoperative care and Internet of Things-based medical monitoring.
- Review of Literature (October–November 2024): Research was done to comprehend the clinical and technical backdrop of vital sign monitoring systems, pertinent communication protocols, and medical device integration at the same time as requirements gathering.
- Hardware Design (December 2024–April 2025): This stage included choosing and integrating sensors, microcontrollers, and communication modules as well as designing and developing the device's electronics and circuit-level architecture.
- Software Development (February-May 2025): This stage, which overlapped with hardware design, concentrated on developing the Arduino-based system's firmware, putting state logic into practice, and providing preliminary support for receiving and transmitting sensor data.
- Thesis Writing (March–May 2025): As development stabilised, the thesis's composition and writing got underway. This made it possible to provide actual implementation information, system schematics, and assessment findings.
- Testing and Debugging (April–May 2025): This step involved testing system module integration, validating state transitions, and improving threshold configuration and error-handling procedures all at the same time as documentation.
- Thesis Final Evaluation and Polishing (May 2025): With the help of my supervisors, revisions were made to the thesis, and mistakes corrected.
- First complete version finalised (May 2025): First completely working version done and ready for full testing.
- Presentation (May 2025): Visual aids and practice sessions for the project's final presentation.

3.7 Summary

This section described the full foundational design of the system being developed, starting with an assessment of the user and technical requirements. These served as a guide for overall development and were drawn from the practical demands of postoperative patient

care.

The interactions between patients, medical personnel, and the system were made clear by the use case diagram. This section also highlighted the ways in which data is gathered, sent, and examined. The architecture of the system, including high-level data flow and hardware integration, was discussed introducing the main technological elements such as the LoRaWAN communication backbone and the microcontroller-based circuit architecture. Following that, the user interface design was discuseed in detail, analysing the function and reasoning behind the LEDs, the LCD screen, the buttons, and the buzzer. The project's development cycle was finally depicted by a Gantt chart, which demonstrated how the many design and implementation phases were arranged and overlapped to effectively fulfil deadlines. When combined, these components create a coherent and well-designed system that meets the specified technological and practical objectives while retaining extensibility and adaptability for future modifications.

Chapter 4

Implementation and Testing

4.1 Introduction

This chapter describes the system's practical implementation and the techniques used to verify its functionality, building on the architectural and design choices discussed in the preceding chapter. The focus now is on the actual construction and verification of the prototype, including hardware integration, firmware development, and system testing under realistic settings, whereas Chapter 3 concentrated on high-level structure, software logic, and system interactions.

There are three primary sections to the chapter. Both the software and hardware components development is outlined in the Implementation section, which also covers the difficulties encountered and how they were resolved during its development. The methods for assessing system performance, including sensor reading validation, Bluetooth connection stability, and state transition accuracy, are described in the Testing section. The chapter is concluded with a summary that highlights the main findings and lessons discovered that influenced later system improvements.

By bridging the gap between the operational device and the conceptual model, its implementation guarantees that the proposed solution not only satisfies user expectations but also operates dependably under real-world restrictions.

4.2 Implementation

This section describes the hardware integration and software development steps involved in creating the prototype system. The system design and user requirements previously outlined served as a guide for the implementation, guaranteeing that the final prototype had the desired capabilities in terms of connectivity, data gathering, processing, and user interaction.

4.2.1 Hardware integration

The design started with an Arduino-compatible microcontroller containing a built-in LoRa module. To this device, an Arduino-compatible Bluetooth Low-Energy module (HM-10) was connected, to facilitate communication to-and-fro the various Bluetooth capable medical sensors that might be used with the system. In addition to the BT module, a DS3231 RTC module was connected to be able to reliably track time and hence allow for some automation in terms of taking readings from the medical sensors. The rest of the hardware components used in the system were for the purposes of allowing users to interface with the system. These components are the LEDs, the buzzer, the 16x2 LCD screen, and the buttons. All but the buttons are for output purposes in varying forms (audio, and visual). The buttons are how the user will be able to provide input to the system. In combination with the LCD screen, the buttons can be formed into a simple yet powerful user interface. The LEDs and buzzer are secondary interfaces, serving mostly as quick indicators to the user regarding the system state, and allow for easier recognition of certain conditions (e.g. a red flashing LED along with a buzzer tone is a clearer indicator that there is some error with the system, rather than just printing text to the LCD). During development the system needs a USB connection to a computer for power, though it would be trivial to alter it to allow for everyday batteries to power it. A detailed schematic of the final circuit is shown in Figure 3.11

4.2.2 Firmware Logic

To develop the firmware, C++ was used in the Arduino development environment. As discussed in the previous chapter, a finite state machine was implemented to control all system operations and clearly separate functionality according to the state it is intended

to run in. State transitions are controlled both by active user input and by passivley monitoring conditions of the system (for example BT disconnection/connection, or reading invalid data a certain number of times). Each major state was implemented as a single function. Intentional state change is handled by using a helper function to explicitly call the state function belonging to the desired new state. This was done so that when first entering a state intentionally the initialisation for that state can occur in the helper function, and prevent it re-initialising if the state function gets called repeatedly every time the main Arduino loop runs (multiple times a second). It also makes state transition clear in code, so that somebody else reading the code can clearly differentiate when a state transition must happen. Another helper function was created to allow for easy logging to the serial monitor of the Arduino IDE, allowing for quick and easy debugging as well as general log messages which can remain after the system is completed (separation of log message levels into DEBUG, and other levels. DEBUG messages can be ignored by setting a single boolean variable to false, leaving general INFO, WARN, and other messages in the production version).

State transitions are event driven. Events that can trigger a state change are selection of a menu option, incoming valid data from the BT module, threshold evaluation, and success/failure conditions as in the READING and TRANSMITTING states.

4.2.3 Bluetooth Communication

Serial Bluetooth communication over the UART (Universal Asynchronous Receiver Transmitter) protocol was implemented to send and receive data to and from paired medical sensors. Sending data to the medical sensor is only to acknowledge valid data received, or request a retry of the transmission. To validate the received data, a simple parsing method was used to first determine what type of reading it represents (blood pressure, temperature, or heart rate), which will influence how the rest of the data must be parsed and interpreted. This was done by always sending a prefix corresponding to each vital sign measured, from the simulation app (BP for blood pressure, TEMP for temperature, HR for heart rate). Depending on the prefix and hence the type of reading represented, the data had to be parsed and split up in different ways. For instance, blood pressure contains two integer values representing systolic and diastolic blood pressure, separated by a forward slash '/', while temperature data is a two-digit integer, followed by a decimal

point '.' followed by another single digit integer. Heart rate is simply either a two or three digit integer.

4.2.4 User Interface Logic

The user interface logic is menu driven primarily and incorporates the LEDs and buzzer as secondary state indicators. The structure of the menus consists of the current menu option printed on the first line of the LCD, and any button functions printed on the second line, if applicable. The menu options are tightly coupled to the individual states, ensuring that the correct menu is displayed at all times depending on the current state. Menu option selection logic occurs within the state functions. As different events occur causing state changes, the LEDs and buzzer also provide indications of said events. The primary user interface elements are the buttons and the 16x2 LCD screen, which are used to interact with and display menu options. The three buttons' available functions are "Previous", "Select", and "Next" in all menus but the SETUP sub-states, where they instead function as "Decrease", "Confirm", and "Increase" buttons referring to the threshold values that can be set from those sub-states. In states with only one option such as the READING, PROCESSING, and TRANSMITTING states, the buttons are effectively disabled seeing as the operations performed by those states do not depend on any user interaction. The way those states disable user interaction is by only having a single menu option defined. In the menu handling logic, a state with only one option is considered "non-interactable" and just prints the single option to the LCD while not printing the menu button functions on the second line of the LCD.

4.3 Testing

System testing was done to verify the way it responded to both normal and extreme situations, as well as to guarantee that every component operated reliably. Testing was split into two categories: testing of local device behaviour, and testing of cloud data transmission.

4.3.1 Local Device Behaviour Testing

MIT App Inventor 2 was used to create a smartphone application that mimics real-world operating conditions without the need for genuine medical sensors. Vital sign data could be manually entered and sent to the device over Bluetooth thanks to this application. Further below in this subsection are included screenshots of the app in various modes.

A variety of test data were sent via the smartphone app, including:

- Valid readings for all vital signs (Blood Pressure, Temperature, Heart Rate)
- Values for boundary conditions near defined thresholds.
- Malformed strings, missing data fields, and values that are out of range (examples of invalid data).

This made it possible to thoroughly check that, given its current setup and status, the device could appropriately parse, assess, and react to incoming input. The logging functions written specifically for this system were used to verify that what was intended to happen on the device was actually what was happening. These logs were essential for observing:

- Raw data received via Bluetooth
- Internal state transitions and their triggers
- System responses such as accepting/rejecting readings, and warnings/errors

Transitions between the READING, PROCESSING, and TRANSMITTING states received special attention. Both standard and edge-case workflows were validated, including error recovery in cases when incorrect readings were received repeatedly.

4.3.2 Cloud Data Transmission Testing

After local data processing was successfully validated, focus shifted to confirming that the system could send data to to the cloud. Making sure the system could consistently offload out-of-bounds readings for additional analysis or clinical action and fail gracefully if required to, was the goal.

To assess this functionality's dependability and robustness, a specific testing strategy was used:

- **Trigger Conditions and State Transitions**

The system was tested to confirm that readings falling within bounds transitioned to the CONNECTED state without passing through the TRANSMITTING state first, whereas if the reading fell out-of-bounds, there would be a transition to the TRANSMITTING state. The mobile app was used to send various inputs simulating each case.

- **Data Formatting and Transmission**

  Each transmitted payload was verified to contain the correct parameters formatted correctly (e.g. vital sign type, value, timestamp).

- **Acknowledgement and Retry Handling**

  The system was programmed to expect a response from the cloud platform indicating successful receipt of the data, failure on malformed or incomplete data, and a timeout timer indicating the platform is cloud possibly unreachable. Each of these three cases was tested to work as expected, by sending different "readings" through the mobile app and observing the response received by the Arduino using the previously mentioned logging functions, and the data received on the cloud platform. To simulate an unreachable network, valid data was sent from the mobile app with the LoRa gateway disconnected.

- **Error Reporting**

  Transmission success should not bring up any messages, but failure should produce a message on the LCD, LEDs, and a buzzer tone.

- **Data Integrity**

  Valid data sent from the device and received on the cloud platform was compared to ensure that it was consistent across the entire chain, from the mobile app, to the device, and finally on the cloud platform.

The tests conducted showed that the cloud communication system can handle transmission failures gracefully without affecting the performance of the device in real-world scenarios, and it can reliably report abnormal readings.

4.4 Summary

This chapter covered the suggested system's practical implementation, from hardware integration and communication logic to software architecture and user interface design.

A state-driven paradigm served as the foundation for the system's implementation, guaranteeing responsive interaction with users and linked sensors as well as dependable transitions between operational modes. Adaptive threshold setup, real-time feedback through buzzers and LEDs, and strong input validation were prioritised.

A specially designed mobile application that simulated a variety of input circumstances was used for testing. The system's capacity to process both expected and incorrect inputs, appropriately handle state transitions, and send out-of-range values to a cloud service for additional evaluation was confirmed by these tests. In order to keep an eye on internal behaviour and confirm dependability under different circumstances, logging tools were extensively used.

With the implementation completed and verified in a controlled setting, the following chapter offers a critical analysis of the outcomes, looks at the system's advantages and disadvantages, and talks about its wider clinical applicability and room for development.

Chapter 5

Discussion

5.1 Introduction

Introduce what this chapter is going to present.

5.2 Discussion

5.3 Difficulties phased

5.4 Knowledge acquired

5.5 Future work

5.6 Conclusions

References

[1] T. A. D'Amico, "Defining and improving postoperative care," *The Journal of Thoracic and Cardiovascular Surgery*, vol. 148, no. 5, pp. 1792–1793, 2014.

[2] S. K. Kare, K. R. Kumar, L. P. Kumar, and N. Naidu, "Post-operative complications in elderly patients undergoing hip fracture surgery: An observational study," *Asian Journal of Medical Sciences*, vol. 15, no. 3, pp. 191–195, 2024.

[3] E. A. Surwit and T. Y. Tam, "Postoperative care," Jan 2008. [Online]. Available: https://www.glowm.com/section-view/heading/Postoperative%20Care/item/36

[4] E. Kachel, K. Constantini, D. Nachman, S. Carasso, R. Littman, A. Eisenkraft, and Y. Gepner, "A pilot study of blood pressure monitoring after cardiac surgery using a wearable, non-invasive sensor," *Front. Med. (Lausanne)*, vol. 8, p. 693926, 2021.

[5] M. Demetz, A. Krigers, R. Uribe-Pacheco, D. Pinggera, J. Klingenschmid, C. Thomé, C. F. Freyschlag, and J. Kerschbaumer, "The role of postoperative blood pressure management in early postoperative hemorrhage in awake craniotomy glioma patients," *Neurosurg. Rev.*, vol. 47, no. 1, p. 452, 2024.

[6] A. Noto, A. Chalkias, F. Madotto, L. Ball, E. G. Bignami, M. Cecconi, F. Guarracino, A. Messina, A. Morelli, P. Princi, F. Sanfilippo, S. Scolletta, L. Tritapepe, A. Cortegiani, and SIAARTI Study Group, "Continuous vs intermittent Non-Invasive blood pressure MONitoring in preventing postoperative organ failure (niMON): study protocol for an open-label, multicenter randomized trial," *J Anesth Analg Crit Care*, vol. 4, no. 1, p. 7, 2024.

[7] S. M. Frank and L. A. Fleisher, "Temperature's importance in patient safety reviewed," https://www.apsf.org/article/

temperatures-importance-in-patient-safety-reviewed/, Jun. 1999, accessed: 2025-5-11.

[8] A. K. Khanna, A. Banga, J. Rigdon, B. N. White, C. Cuvillier, J. Ferraz, F. Olsen, L. Hackett, V. Bansal, and R. Kaw, "Role of continuous pulse oximetry and capnography monitoring in the prevention of postoperative respiratory failure, postoperative opioid-induced respiratory depression and adverse outcomes on hospital wards: A systematic review and meta-analysis," *J. Clin. Anesth.*, vol. 94, no. 111374, p. 111374, 2024.

[9] A. K. Khanna, M. Flick, and B. Saugel, "Continuous vital sign monitoring of patients recovering from surgery on general wards: a narrative review," *Br. J. Anaesth.*, vol. 134, no. 2, pp. 501–509, 2025.

[10] M. Elliott, "Why is respiratory rate the neglected vital sign? a narrative review," *International Archives of Nursing and Health Care*, vol. 2, no. 3, 2016.

[11] H. Kawanishi, J. Egawa, S. Inoue, T. Shiota, and M. Kawaguchi, "Incidence of life-threatening respiratory events after laparoscopic colon surgery with or without continuous respiratory rate monitoring," *JA Clinical Reports*, vol. 3, pp. 1–4, 2017.

[12] J. Olusegun, M. Alonge, E. Ok, and B. Barnty, "Real-time patient monitoring with iot-integrated wearables," 02 2025.

[13] J. B. Harez, S. Akter, R. I. Sony, M. T. H. Amin, M. J. Anee, and A. Hossain, "IoT-Driven System for Continuous Monitoring of Heart Disease Patients Post-Surgery," 10 2024.

[14] K. Sundaravadivel, P. Malaiyarasan, H. Karuppiah, and N. Raja, "Iot based vital signs monitoring system for human beings," *3C Tecnología_Glosas de innovación aplicadas a la pyme*, pp. 611–621, 11 2021.

[15] H. Taherdoost, "Wearable healthcare and continuous vital sign monitoring with iot integration," *Computers, Materials & Continua*, vol. 81, no. 1, pp. 79–104, 2024. [Online]. Available: http://www.techscience.com/cmc/v81n1/58312

[16] "What are LoRa and LoRaWAN?" https://www.thethingsnetwork.org/docs/lorawan/what-is-lorawan/, accessed: 2025-5-8.

[17] "LoRa — LoRa documentation," https://lora.readthedocs.io/en/latest/, accessed: 2025-5-8.

[18] R. Wenner, "LoRa CHIRP," 17 2017.

[19] "Sigfox: Advantages and disadvantages," https://www.rfwireless-world.com/Tutorials/advantages-and-disadvantages-of-Sigfox-wireless-technology.html, accessed: 2025-5-8.

[20] "NB-IoT: Advantages and disadvantages," https://www.rfwireless-world.com/Terminology/Advantages-and-Disadvantages-of-NB-IoT.html, accessed: 2025-5-8.

[21] https://iot.telenor.com/technologies/connectivity/lte-m/, accessed: 2025-5-10.

[22] "LTE-M: Advantages and disadvantages," https://www.rfwireless-world.com/Terminology/Advantages-and-Disadvantages-of-LTE-M.html, accessed: 2025-5-8.

[23] "LoRaWAN: Advantages and disadvantages," https://www.rfwireless-world.com/Terminology/Advantages-and-Disadvantages-of-Lora-or-LoRaWAN.html, accessed: 2025-5-8.

[24] "No title," https://www.lancaster.ac.uk/scc/sites/lora/lorasim.html, accessed: 2025-5-8.

[25] "Duty cycle," https://www.thethingsnetwork.org/docs/lorawan/duty-cycle/, accessed: 2025-5-11.

[26] "Wi-Fi beacons & beacon interval: Everything you need to know," https://www.7signal.com/news/blog/controlling-beacons-boosts-wi-fi-performance, accessed: 2025-5-11.

Appendices

Appendix A

Write a few words about what is included in this appendix

Appendix B

Write a few words about the contents of the appendix

Manuals

User Manual

This device is designed to collect key health metrics in the post-operative phase via Bluetooth, and help you and your medical professionals monitor said vital signs. The three measurements it is capable of collecting and monitoring currently are blood pressure, temperature, and heart rate. It allows setting ranges for each metric, so that if a reading is outside these ranges it can automatically send it to the cloud and alert your medical professionals in order to allow them to more efficiently take care of you.

5.6.1 Getting Started

Power On

- Plug in device or add 6xAA standard batteries
- The LCD display will briefly display a message informing you it is booting, followed by "No connection" if there is no device currently set to auto-connect within Bluetooth range, or "Read" if there is a device within range and it automatically connects.

5.6.2 Menu Navigation

Buttons

- **Prev** (left)
- **Select** (middle)
- **Next** (right)

Basic Actions

- **Prev/Next:** Move the highlighted option or adjust values
- **Select:** Confirm choice or save a value

5.6.3 Main Menus

Device has no active Bluetooth connections

- **No connection**
  - Displays when there are no active Bluetooth connections
  - Pressing select on this option has no effect
- **Setup**
  - Allows setting up the acceptable range for each vital sign, where values read from Bluetooth which are within this range will be considered healthy readings
  - Pressing select on this option switches to the Setup menu

Device has at least one active Bluetooth connection

- **Read**
  - Enter mode where waiting for incoming Bluetooth data from a paired sensor
  - Buttons have no effect here. When data comes in, device will automatically check the data or remain in this mode if invalid data received.
- **Setup**
  - Allows setting up the acceptable range for each vital sign, where values read from Bluetooth which are within this range will be considered healthy readings
  - Pressing select on this option switches to the Setup menu
- **Disconnect**
  - Disconnect all connected sensors **Note: Not implemented yet**

Menu to set acceptable ranges for each vital sign

- **Setup BP**
  - Set minimum and maximum for systolic and diastolic blood pressure
  - Prev button decreases the value, Next increases it. Select confirms the value.
- **Setup Temp**

  - Set minimum and maximum for temperature.
  - Prev button decreases the value, Next increases it. Select confirms the value.
- **Setup HR**
  - Set minimum and maximum for heart rate.
  - Prev button decreases the value, Next increases it. Select confirms the value.
- **Back**
  - Returns to the previous menu
- The maximum can not be less than the minimum for any given vital sign. For example, concerning the systolic blood pressure maximum, it must be equal to or more than the minimum for systolic blood pressure.

5.6.4 Configuring Acceptable Ranges

Blood Pressure (BP)

1. Select **Setup BP**
2. Adjust **SYS min** (systolic mimimum) until reached desired value
3. Press the Select button to commit that value
4. Adjust **SYS max** (systolic maximum) until reached desired value
5. Press the Select button to commit that value
6. Adjust **DIA min** (diastolic mimimum) until reached desired value
7. Press the Select button to commit that value
8. Adjust **DIA max** (diastolic maximum) until reached desired value
9. Press the Select button to commit that value
10. After the fourth value, return to Setup

Note: If you accidentally set a "max" below its corresponding "min", the device will display a short error message (Err: Max < Min. Re-enter. It will then prompt you to re-enter both values.

Temperature (TEMP)

1. Select **Setup Temp**
2. Adjust **TEMP min** until reached desired value
3. Press the Select button to commit that value
4. Adjust **TEMP max** until reached desired value
5. Press the Select button to commit that value
6. Returns to Setup

Heart Rate (HR)

1. Select **Setup HR**
2. Adjust **HR min** until reached desired value
3. Press the Select button to commit that value
4. Adjust **HR max** until reached desired value
5. Press the Select button to commit that value
6. Returns to Setup

5.6.5 Taking Readings

1. **Ensure you have at least one paired external sensore of type BP/TEMP/HR**
2. In Connected menu, select **Read**
3. The device listens coninuously until it gets a valid reading. When a new reading arrives, it is automatically processed and transmitted to the cloud if necessary.

5.6.6 Readings In Range or Out of Range

- Within range:
  - No action - device remains in Connected state
- Out of range:
  - The device transitions to the Transmitting state and sends the reading to you cloud central console via LoRaWAN
  - You or your medical professionals can then review the reading remotely.

5.6.7 Maintenance & Care

- Keep the LCD and buttons free from dust and moisture.
- Store in a dry, room-temperature environment.
- Clear with a soft, dry cloth - do not use solvents or other liquids.

Technical Manual

This manual is intended to be read by people wishing to gain a better understanding of the structure of the codebase for this project, so that they can effectively extend, modify, or otherwise change it. This firmware implements a finite-state machine to manage all functions of the device. These include:

- Connection handling
- Configuration of the acceptable ranges for Blood Pressure, Temperature, and Heart Rate
- Data acquisition, validation, threshold checking, and transmission to cloud via LoRaWAN

All user interaction with the device takes place on a 16x2 I^2C LCD, navigated by 3 physical buttons.

5.6.8 Directory & File Structure

```
/src
 +-- globals.h           // Pin, global variable, & EEPROM
   ↪ address definitions, extern globals
 +-- menu.h/.cpp         // Menu definitions & handlers
 +-- states.h/.cpp       // State functions & FSM table
 +-- utils.h/.cpp        // Button debounce, logging, value
   ↪ adjustment, validation
 +-- Waveshare_LCD1602.* // Customized LCD driver
 +-- vital_monitor.ino   // setup() / loop() wiring FSM +
   ↪ peripherals
```

5.6.9 Global Definitions & Naming Conventions

Macros:

- `BTN_PREV`, `BTN_SELECT`, `BTN_NEXT`: GPIO pins connected to the Previous, Select, and Next buttons respectively

- `LED_BLUE`, `LED_GREEN`, `LED_YELLOW`, `LED_RED`: GPIO pins driving the four status LEDs
- `BT_STATE`: GPIO pin reading the Bluetooth module's connection status (HIGH = connected)
- `G_*_ADDR`: EEPROM addresses for storing each threshold, where * will have information about the vital sign, and whether it is a min or a max. All uppercase, and are prefixed with **G_**

    e.g. `G_TEMP_MIN_ADDR` (the address storing the minimum threshold for temperature) or `G_BP_SYS_MAX_ADDR` (the address storing the maximum threshold for blood pressure, specifically the systolic value)
- `G_*_THRESHOLD_MIN, G_*_THRESHOLD_MAX`: Default (factory) minimum and maximum allowable values for each vital sign. All uppercase, and are prefixed with **G_**

Global variables:

All lowercase, and are prefixed with **g_**

- **Button states:**
  - `g_prev_button_state`, `g_select_button_state`, `g_next_button_state`: Debounced current readings of the Prev/Select/Next buttons.
- **Menu navigation:**
  - `g_current_option_index`: Index of the currently highlighted menu entry.
  - `g_last_option_index_displayed`: Last index sent to the LCD (to avoid unnecessary redraws every loop).
  - `g_selection_pending`: Flag indicating a Select press is awaiting handling.
- **Threshold values:**
  - **Blood pressure**
    - `g_bp_systolic_threshold_min`, `g_bp_systolic_threshold_max`: User-configurable BP bounds (uint8_t)
  - **Temperature**
    - `g_temp_threshold_min`, `g_temp_threshold_max`: User-configurable TEMP bounds (uint16_t). Stored as the temperature×10 (e.g. 36.5°C stored as 365 to save 2 bytes in memory compared to storing it as a float or double

which would require 4 bytes).

  - **Heart rate**
    * `g_hr_threshold_min`, `g_hr_threshold_max`: User-configurable HR bounds (uint8_t).

- **Finite-State Machine Control:**
  - `g_current_state`, `g_previous_state`: Current and last state in the finite-state machine.
  - `g_setup_caller_state`: Remembers which state launched the Setup menu (for "Back" logic, after entering a setup sub-state such as "Setup BP", which would then set `g_previous_state` to the "setup" state instead of "Disconnected" or "Connected").
  - `g_multi_reset`: Signals entry into a multi-step threshold setup to initialise its values without needing to create a separate sub-state for each individual value (for example no need to create a sub-state for each of the following: systolic minimum, systolic maximum, diastolic minimum, diastolic maximum).
- **Data buffer:**
  - `g_received_data_buffer[G_RECEIVED_DATA_BUFFER_SIZE]`: Holds the latest NUL-terminated Bluetooth message (e.g. "BP:120/80").
- **Debug:**
  - `debug_enabled`: Enables or silences `log_msg("DEBUG", ...)` output.

State Enumeration (states):

The `states` enum defines each major step of the device's operation:

- `DISCONNECTED`

  No active Bluetooth connections; waiting to connect. User may enter `SETUP`.
- `CONNECTED`

  Bluetooth is up; user may begin reading data, enter `SETUP`, or disconnect the connected Bluetooth devices.
- `SETUP`

  Main configuration menu, where user can choose to edit BP, TEMP, or HR thresholds.
- `SETUP_BP`, `SETUP_TEMP`, `SETUP_HR`

Sub-menus for adjusting blood pressure, temperature, or heart-rate limits.

- `READING`

  Actively listening to receive a measurement string from the Bluetooth module.

- `PROCESSING`

  Parsing the incoming data and checking it against the configured thresholds.

- `TRANSMITTING`

  Sending an out-of-range measurement onward via LoRaWAN, then returning to `CONNECTED`.

5.6.10 State Machine Architecture

`stateTable[]`

An array mapping each `states` value to its handler `StateFunc`

```
struct StateTable stateTable[] = {
        {DISCONNECTED, state_disconnected},
        {SETUP, state_setup},
        {SETUP_BP, state_setup_bp},
        {SETUP_TEMP, state_setup_temp},
        {SETUP_HR, state_setup_hr},
        {CONNECTED, state_connected},
        {READING, state_reading},
        {PROCESSING, state_processing},
        {TRANSMITTING, state_transmitting}
};
```

`loop()` & **Transition Logic**

```
if (g_current_state in {DISCONNECTED, CONNECTED, SETUP})
        next_state = handle_menu(g_current_state);
else if (g_current_state == READING)
        next_state = state_reading();
... etc ...
next_state = check_bt_connection(next_state);
```

```
if (next_state != g_current_state)
        change_state(next_state);
```

`check_bt_connection()`

- Monitors the `BT_STATE` input pin to detect stable Bluetooth connection or disconnection events, and drives FSM transition accordingly.
- Suppresses connection checks during certain states where it is irrelevant (`PROCESSING`, `TRANSMITTING`, `SETUP`, `SETUP_BP`, `SETUP_TEMP`, `SETUP_HR`,).
- Returns a `states` enum value, either:
  - Unchanged (`current_state`)

    If still within the initial stabilisation period, or if in a state that should ignore Bluetooth changes.
  - `CONNECTED`

    Upon detecting a sustained HIGH on `BT_STATE`.
  - `DISCONNECTED`

    Upon detecting a sustained LOW on `BT_STATE` after a previous HIGH.

`change_state()`

- Updates `g_previous_state` and `g_setup_caller_state` (for Setup menu "Back").
- Clears LCD and resets menu indices when entering new state.
- Triggers LED update.

5.6.11 Menu System

Menu Tables (`menu_table[]`)

- Disconnected: `{"No connection", "Setup"}`
- Setup: `{"Setup BP", "Setup Temp","Setup HR", "Back"}`
- Connected: `{"Read", "Setup", "Disconnect"}`
- Reading/Processing/Transmitting: single-item static menus (no prev/select/next button functionality)

`handle_menu()`

1. Calls `handle_menu_options_buttons()`, which:
    - Edge-detects Prev/Select/Next buttons.
    - Wraps navigation index so that pressing Next on the last entry wraps around to the first entry and vice-versa.
    - Returns 0-N where N is the number of options for that state's menu. Return value is the index of the selected option.
    - Returns 255 if no selection was made.
2. Once Select is detected, dispatches to a new state:
    - In `DISCONNECTED`: "Setup" → `SETUP`.
    - In `SETUP`, enters corresponding `SETUP_*` sub-state or returns via `g_setup_caller_state`
    - In `CONNECTED`:
      "Read" → `READING`,
      "Setup" → `SETUP`,
      "Disconnect" → `DISCONNECTED`

5.6.12 Button & LCD Utility Functions

`debounceReadButton()`

Standard 20ms software debounce; tracks `stable_state`.

`log_msg()`

Takes two required parameters: the level of the log message, and the message itself. There are two overloaded versions which take a third optional argument (`const bool` and `unsigned`). Respects `debug_enabled`.

`handle_value_adjust_u8, handle_value_adjust_u16`

- Support tap for single increment/decrement, as well as hold for auto-repeat.
- Parameterised by step size, hold delay, and repeat interval. Step size is the amount to increment/decrement, hold delay is how long to wait before considering a button

press as a hold, and repeat interval is used to control speed of press and hold increment/decrement.

`validate_message()`

- Ensures `g_received_data_buffer` matches one of:
  - **BP:2-3 digits/2-3 digits**
  - **TEMP: dd.d**
  - **HR:2-3 digits**
- Rejects malformed strings before processing.

5.6.13 Threshold-Setup Framework

Two template functions:

- `multi_threshold_setup_u8()` for 8-bit thresholds.
- `multi_threshold_setup_u16()` for 16-bit temperature thresholds.

Behaviour on entry (`g_multi_reset` flag):

- Reset `step`, `last_drawn`, `last_select`
- Clamp existing `*values[i]` to `[lo[i]..hi[i]]` blocking the user from incrementing/decrementing past the default minimum and maximum allowable values for each vital sign (`G_*_THRESHOLD_MIN` and `G_*_THRESHOLD_MAX`).

Per-step UI loop:

1. Clear & redraw prompt + current `*values[step]`.
2. Call `handle_value_adjust`.
3. On **Select** button rising edge:
   - If max<min (for paired steps), display error and repeat this step.
   - Otherwise write changed values to EEPROM.
   - Advance `step`; if `step == count`, return to `previous_state`.

5.6.14 Bluetooth Data Flow

1. `state_reading()`
   - Monitors `BT_STATE` pin for disconnect.

- Reads bytes until `\n`.
- Strips optional `\r`.
- Validates via `validate_message()`.
- Sends `ACK` or `RETRY` over Bluetooth module's UART.
- On valid data, copies it into `g_received_data_buffer` and transitions to the `PROCESSING` state.

2. `state_processing()`
   - Parses buffer depending on prefix ("BP:", "TEMP:", "HR:").
   - Checks parsed values against `g_*_threshold_{min,max}`.
   - If out-of-range, returns `TRANSMITTING` state, otherwise returns `CONNECTED` state.

3. `state_transmitting()`
   - Placeholder to send `g_received_data_buffer` to cloud via LoRaWAN (Not implemented yet).
   - Always returns `CONNECTED` state.

5.6.15 EEPROM Storage

- Addresses:
  - **BP:** bytes 0-3 (one byte each, min/max for systolic and diastolic)
  - **TEMP:** 4-7 (two bytes each, min/max) (uint16_t little-endian)
  - **HR:** 8-9 (one byte each, min/max)
- Initialisation (in `setup()`):
  - Read each address; if uninitialised (0xFF/0xFFFF) as they will be on the very first boot of the device, load default macros (`G_*_THRESHOLD_MIN`, `G_*_THRESHOLD_MAX`)

    For temperature which is a uint16_t, need to read in a special way:

    ```
    g_temp_threshold_min = EEPROM.read(G_TEMP_MIN_ADDR) |
            (EEPROM.read(G_TEMP_MIN_ADDR + 1) << 8);
    ```

    Read the low byte first and logical OR it with the high byte, shifted 8-bits left.

- Writing:
  - Only write when new value $\neq$ stored value to minimise wear on the EEPROM cells (limited writes).

5.6.16 LCD Driver Adjustments

- Based on manufacturer's Waveshare_LCD1602 library
- Had to comment out the line: `Wire.setPins(4, 5);`