IOT-BASED HEALTHCARE MONITORING AND ALERT SYSTEM

By

Rachel Leung Onn Yneng

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF INFORMATION TECHNOLOGY (HONOURS) COMPUTER ENGINEERING

Faculty of Information and Communication Technology

(Kampar Campus)

FEBRUARY 2025

COPYRIGHT STATEMENT

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my project supervisor, Dr Teoh Shen Khang who has given me this bright opportunity to engage in an IoT-based project. It is my first step to establish a career in the IoT field. I would like to thank him for his insightful guidance and patience throughout the development of this research.

Finally, I must say thanks to my parents and my family and friends for their love, support, and continuous encouragement throughout the whole process.

ABSTRACT

In order to safeguard people's health, we want efficient healthcare solutions as the number of health issues worldwide rises. One proposed solution for this problem is a real-time Internet of Things (IoT)-based health monitoring system. People may now easily check their vital signs from home using new technology, eliminating the need for them to visit hospitals. This is especially helpful for those who reside in remote or rural areas with limited access to healthcare. This lowers healthcare costs while simultaneously saving time. Additionally, hospitals may monitor their patients' vital signs with this equipment. With the use of IoT, doctors can rapidly and efficiently get relevant patient data, enabling them to make the best judgements. The goal of this project is to present an intuitive IoT-based healthcare monitoring and alarm system that gives people proactive tools to effectively manage their health. The system makes use of sensors to record vital indications such as body temperature, heart rate, and activity level, giving the user useful information. a central processing unit with the capacity to manage several inputs from different sensors. Comprehensive health data is collected using an accelerometer, temperature sensor, and pulse sensor. For accurate analysis, these inputs are transformed from analog to digital format The collected data is processed and analyzed in real-time, and efficient data storage and retrieval are accomplished by utilizing a robust database management system. A user-friendly interface makes it easy and straightforward for patients and medical professionals to access and evaluate health data.

Area of Study (Minimum 1 and Maximum 2): Internet of Things, Healthcare Technology

Keywords (Minimum 5 and Maximum 10): Data Collection in IoT, Monitoring Application, Biomedical Sensor Technology, Vital Signs Measurement, Real-time Data, Data Analysis, Alert System, Health Data Logging, Data Visualization

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

IoT	Internet of Things
ECG	Electrocardiogram
CVD	Cardiovascular diseases
DWT	Design Rule Checker
MLP	Scalable CMOS N-Well Analog
FrFT	Application Specific Integrated Circuit
RT	Hardware Description Language
PR	Protease
IT	Information Technology
RPM	Remote Patient Monitoring
GUI	Graphical User Interface
SQL	Structured Query Language
LED	Light Emitting Diode
DC	Direct Current
LAN	Local Area Network
RDBMS	Relational Database Management System
IEEE	Institute of Electrical and Electronics Engineers
LPDDR	Low-Power Double Data Rate
Hz	Hertz
GB	Gigabyte
ARM	Advanced RISC Machine
SoC	System-on-a-Chip
USB	Universal Serial Bus
IDE	Integrated Development Environment
API	Application Programming Interface
FYP	Final Year Project
I^2C	Inter-Integrated Circuit
GPIO	General Purpose Input / Output
BPM	beats per minute
SpO_2	Peripheral Oxygen Saturation

VCC	Voltage at the Common Collector
GND	Ground
AWG	American Wire Gauge
SDA	Serial Data Line
SCL	Serial Clock
HDMI	High-Definition Multimedia Interface
VIN	Input Voltage
FIFO	First-In, First-Out
VGA	Video Graphics Array

Chapter 1
Introduction

This chapter will provide background information on the research on the IoT-based healthcare monitoring system, including information on how remote healthcare monitoring was developed, how IoT technology was introduced, and the project's objectives.

In today's interconnected world, the Internet of Things (IoT) has become an innovative force that is transforming various aspects of our daily lives. IoT refers to the collective network of connected devices including sensors, software, and other various technologies. The goal is to enable seamless connection, transfer, and exchange of data with other systems over the Internet autonomously without requiring human intervention (Kamarozaman et al., 2021). IoT can be used in nearly every area, including agriculture, energy, transportation, manufacturing, and healthcare.

IoT technology has significantly changed how we interact with our environment. We no longer live, work, or play the same way thanks to the Internet of Things. IoT devices have increased human well-being and convenience in our daily lives by allowing us to remotely operate our home appliances and security systems.

In the past, people have depended on routine check-ups at hospitals and clinics, which had the disadvantage of not providing real-time information on a patient's condition or acting quickly when there was a major health issue. Fortunately, IoT has opened up new choices for healthcare management because of technical advances.

This project aims to solve the inadequacies of conventional healthcare monitoring techniques. My suggested idea intends to enable people to take charge of their well-being by using IoT technology to monitor their health remotely and make informed decisions. Simultaneously, lowering healthcare expenses and facilitating prompt intervention by healthcare professionals when needed, guaranteeing prompt resolution of any health concerns. Through these efforts, the project aims to improve overall healthcare efficiency by focusing on patients’ needs.

1.1 Problem Statement and Motivation

Even with the advances in medical technology, there is still a significant gap in terms of offering patients outside of hospitals comfortable and efficient monitoring options. Patients with chronic diseases or limited mobility may find it burdensome to attend healthcare institutions on a frequent basis for check-ups or monitoring, which is a common need of the traditional method. The dependence on recurring visits restricts the prompt identification of health problems and could impede the application of preventative care, thereby leading to unfavorable outcomes for the patient. Therefore, it is important to develop cutting-edge solutions that utilize IoT technology to offer smooth and continuous remote patient monitoring in order to close the gap between clinical treatment and home-based monitoring.

Due to extended hospital stays and numerous in-person visits, the traditional method of health monitoring is costly and time-consuming. Both healthcare systems and individual patients are heavily burdened by these expenditures, which causes financial hardship and restricts access to essential medical services for those with low incomes.

Additionally, traditional monitoring methods often capture vital signs of the heart rate and blood pressure only at specific times during scheduled appointments. However, this periodic approach to data collection only provides a partial picture of the patient's health, leading to a disjointed knowledge of their general state of health. This disjointed data gathering may miss small variations or patterns in vital signs, delaying the detection of underlying illnesses or issues.

Even though traditional ways of keeping tabs on health have been a cornerstone of medical care for decades, they are not without drawbacks. These include high healthcare costs, limitations on prompt diagnosis and treatment, and dispersed data collecting. Identifying these obstacles emphasizes the need for creative methods of healthcare monitoring that put the needs of patients, the economy, and thorough data collecting first in order to improve patient outcomes.

1.2 Objectives

The aim of this thesis is to propose an IoT-based healthcare monitoring system that leverages new technologies and efficient algorithms to improve patient care. The objectives include:

- To develop a health monitoring system by using affordable platforms and sensors.

- To enable remote health monitoring to allow healthcare providers to access patient information anytime.

- To incorporate real-time vital sign analysis for timely healthcare intervention to reduce healthcare expenses and unnecessary hospital visits.

- To combine a Raspberry Pi database with a Python graphical user interface module to provide effective data management and user interaction within the interface.

1.3 Project Scope and Direction

The goal of this project is to create an IoT-based healthcare monitoring and alarm system that is fully operational at the conclusion. The project's title, "IoT-based Healthcare Monitoring and Alert System," accurately describes our goal of utilizing IoT technology to address the drawbacks of traditional healthcare monitoring systems, as stated in the problem statement. This system will be composed of wearable sensors, smart medical devices, a centralized data platform, and related algorithms for data analysis and alarm production. It will also include software components.

1.3.1 System Design and Architecture:

- Based on a thorough examination of the needs for healthcare monitoring, specify the functions, goals, and requirements for the system.
- Create the overall system architecture, taking into account the data storage, user interfaces, hardware, and software components.
- Determine how different sensors, gadgets, and IoT technologies can be integrated to allow for the real-time monitoring of health indicators and vital signs.

1.3.2 Hardware and Software Development:

− Create or acquire the required hardware, such as microcontrollers, communication modules, and sensors that work with Raspberry Pi.

− Create new libraries and software or utilize current ones to process the information gathered from every sensor, making sure the Raspberry Pi can generate an extensive report from the variety of inputs.

− Provide patient and healthcare provider-friendly interfaces for the monitoring system, such as mobile applications, web-based dashboards, and alarm systems.

1.3.3 Data Management and Integration:

− Install and set up the Raspberry Pi's database system to securely store and handle the massive amounts of medical data that the monitoring system has gathered.

− Provide database management features to ensure that data is efficiently arranged, queried, and retrieved while adhering to the specifications of the monitoring system.

− To safeguard patient privacy and stop unwanted access to private health information, put security measures in place and make sure data privacy laws are followed.

1.3.4 Integration and Testing:

− Integrate the Raspberry Pi, database, sensors, and GUI with the other hardware and software elements of the monitoring system.

− Prior to integration into the system, every sensor must be separately checked to ensure operation. In order to make sure that every sensor produces reliable and consistent data, use simple test codes.

− Integrate every sensor with the Raspberry Pi, making sure that the connections are safe and that no sensor affects the readings of any other sensor.

1.3.5 Deployment and Evaluation:

− Make sure the system is flexible and simple to use in a variety of contexts by concentrating the deployment approach on portability.

− Test the system on people to see how well it works for a wide range of people with different demands for health monitoring.

- Analyze the system's performance using quantitative measurements and qualitative input to enhance patient outcomes, healthcare delivery, and operational efficiency.

The project's overall goal is to create a novel and workable IoT-based healthcare monitoring solution, with an emphasis on data collecting, storage, and graphical interface presentation.

1.4 Contributions

Through creative methods and modern technology, this project significantly improves healthcare administration and monitoring. First of all, by giving patients more control over their health through remote monitoring capabilities, it empowers them. This encourages patient autonomy and improves general well-being by allowing patients to examine their vital signs and health data from the comfort of their homes, particularly for those managing ongoing medical illnesses or recovering from surgery.

Additionally, the initiative provides proactive healthcare management through real-time data analysis and continuous vital sign monitoring. Healthcare providers can optimize treatment plans and act quickly by adopting a proactive approach that helps them spot anomalies or changes in a patient's health state early on. Thus, fewer unnecessary in-person consultations and hospital readmissions are required, which enhances patient outcomes and reduces healthcare costs.

Moreover, the idea offers potential cost benefits for individuals as well as healthcare organizations by reducing healthcare expenditures associated with prolonged hospital stays and frequent visits. The project improves resource usage and simplifies healthcare delivery by utilizing IoT technology to facilitate remote monitoring, leading to increased efficiency and cost-effectiveness.

Additionally, the project improves the quality of healthcare services by offering possibilities for thorough and ongoing monitoring. Healthcare professionals may make educated decisions and provide individualized care by gathering vital signs and health data in real-time. This provides them with important insights into patients' health status. Together, these efforts modernize remote patient monitoring, boost the quality of healthcare services, and improve healthcare outcomes.

1.5 Report Organization

The details of this research are shown in the following chapters. Chapter 2 reviews the existing literature on IoT technology and its application in healthcare. It explores the evolution of remote healthcare monitoring, key technological advancements, and the current state of IoT-based solutions in the medical field. Then, the method for developing the IoT-based healthcare monitoring system is proposed in Chapter 3. It includes the system's design and architecture, the selection of hardware and software components, and the integration of various sensors and data management techniques to enable real-time health monitoring. Chapter 4 presents the preliminary work, including the development of the prototype, initial testing results, and challenges encountered during implementation. Finally, Chapter 5 concludes the thesis by summarizing the key findings of the research, discussing the contributions of the project, and suggesting potential directions for future research. It also evaluates the impact of the proposed system on healthcare delivery and patient outcomes.

Chapter 2
Literature Review

2.1 Previous Works on Deep Learning

2.1.1 ECG-based Machine-learning algorithms

Electrocardiogram (ECG) is a test that records the electrical activity of the heart over a period of time using electrodes placed on the skin. This test helps to diagnose various heart conditions by measuring the electrical impulses that control heart rhythm. The Electrocardiogram (ECG) signals illustrate the electrical behavior of the human heart and comprise various waveforms.

S. Aziz in [7] presents a novel methodology for diagnosing cardiovascular diseases (CVD) using electrocardiogram (ECG) signals without the need for a probe or a cardiologist. The proposed system involves placing probe-less ECG sensors on a patient's body, transmitting the signals via Bluetooth to a processing device like a mobile phone, and using a machine learning algorithm for automatic CVD diagnosis.

The methodology discussed in the paper involves various key components. Signal filtering is crucial due to the non-stationary nature of ECG signals and the presence of noise and artifacts. The Discrete Wavelet Transform (DWT) is used for baseline drift removal. Feature extraction and classification are then performed using machine learning techniques. Researchers have previously used the MIT-BIH arrhythmia database for ECG signal classification, employing techniques like DWT and Multi-Layer Perceptron (MLP) for high accuracy but with a large number of features.

The proposed algorithm aims to overcome the limitations of existing algorithms by combining event-related moving averages with Fractional Fourier Transform (FrFT) for improved detection of ECG waveform components. The classification of CVD involves feature extraction and model selection. The SVM and MLP classifiers are used, with features like PR and RT intervals, age, and sex showing promising results. The methodology is tested on the MIT-BIH database and the Shaoxing SPH database, achieving high accuracies with reduced feature sets.

Overall, the paper presents a detailed methodology for probe-less CVD diagnosis using ECG signals, combining signal processing, feature extraction, and machine learning classification techniques. The proposed algorithm shows promise for accurate and efficient CVD diagnosis, paving the way for future developments in remote healthcare monitoring systems.

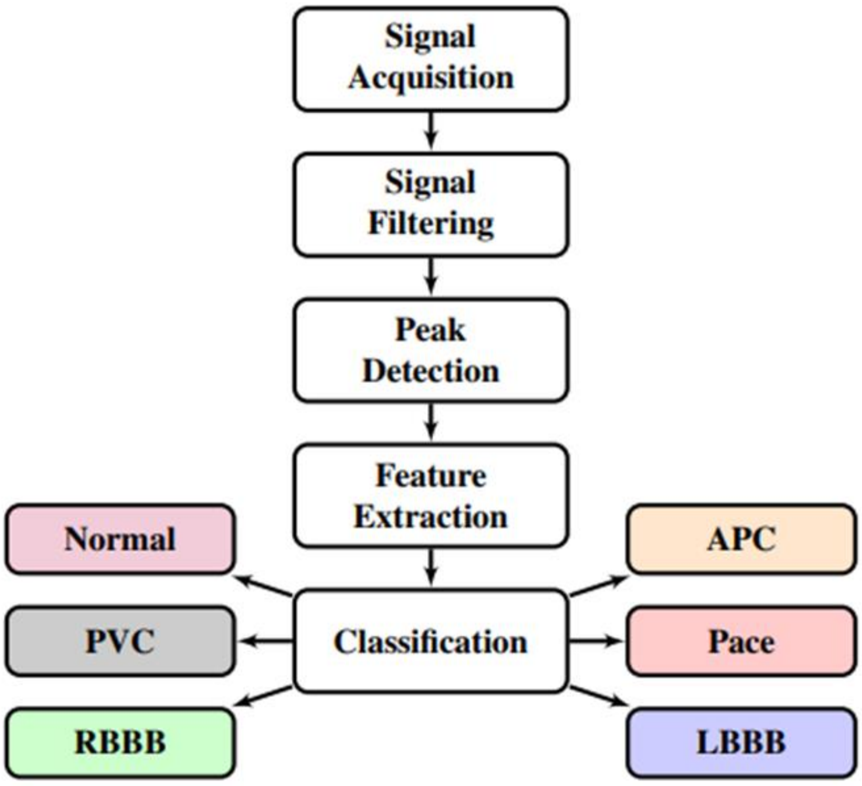


Figure 2.1 Block Diagram of the Proposed Methodology

2.1.2 Telemedicine in Healthcare

M. Stoltzfus et al. in [8] discusses the technical workings of telemedicine, detailing its reliance on advanced telecommunications and information technologies to deliver healthcare services remotely. Telemedicine employs a variety of digital communication tools, such as video conferencing, health informatics data, and real-time audio communications, to facilitate remote clinical services. These technologies enable healthcare providers to perform consultations, diagnostics, treatment planning, and patient monitoring without the need for physical presence. The system's architecture is designed to support seamless data flow between patients and providers, ensuring that critical health information is accessible and actionable in real time. This setup not only enhances the capability of healthcare systems to provide services to remote or underserved populations but also increases the overall efficiency and effectiveness of medical care.

Technologically, telemedicine integrates with existing healthcare IT systems, requiring robust cybersecurity measures to protect patient data and privacy. As telemedicine continues to evolve, it incorporates newer technologies like artificial intelligence and machine learning to further refine the accuracy and responsiveness of remote healthcare delivery.

2.1.3 Remote Patient Monitoring (RPM) Systems

RPM systems leverage IoT to enhance healthcare delivery, allowing for continuous patient monitoring outside traditional medical settings. These systems are particularly beneficial for chronic disease management and post-operative care, where regular monitoring is crucial. By integrating with cloud services, RPM systems can store vast amounts of medical data and provide healthcare professionals with real-time access. This integration supports advanced analytics and helps in making informed medical decisions (Muhammad Waleed et al. [9]) (S. Iranpak et al. [10]).

Advanced RPM systems utilize multi-hop IoT networks and biosensors. These networks enhance the range and reliability of data transmission, crucial for monitoring patients across different environments. Biosensors play a key role in collecting accurate health data, which is then processed to provide insights into the patient's health status (R. Uddin et al. [11]).

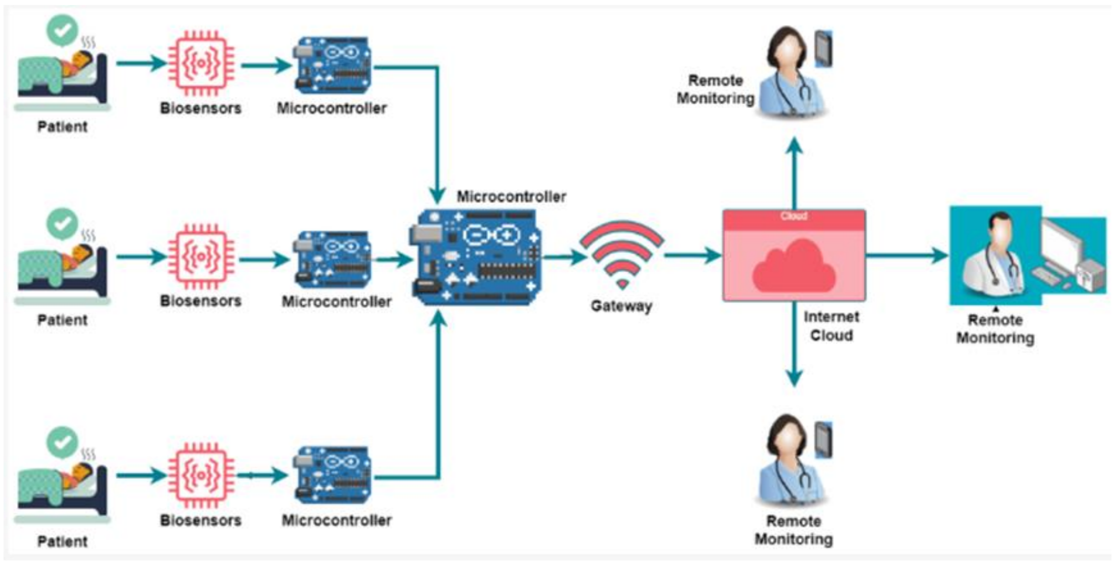


Figure 2.2 Remote Monitoring via IoT Healthcare System

2.1.4 Machine Learning and Wearable Biomedical Devices

A. Olyanasab in [12] delves into how wearable devices equipped with machine learning technologies are revolutionizing personalized health monitoring. These devices are broadly classified into three categories based on the type of measurements they perform: bio-electrical, bio-impedance and electro-chemical, and electro-mechanical.

The bio-electrical devices analyze electrical signals from the body, like heart activity from ECGs, using machine learning to detect cardiac health issues. Bio-impedance and electrochemical devices measure body tissue impedance and chemical changes, such as glucose levels, helping manage conditions like diabetes. Electro-mechanical devices assess physical activity and muscle function through sensors like EMG, with machine learning helping to analyze movement patterns to prevent injuries and optimize physical health.

In each case, machine learning is central to processing and interpreting the vast amounts of data these devices collect, enabling them to offer personalized health insights and interventions based on real-time data analysis. This approach not only enhances the accuracy of health monitoring but also enables more personalized healthcare by modifying actions according to unique data trends and forecasts.

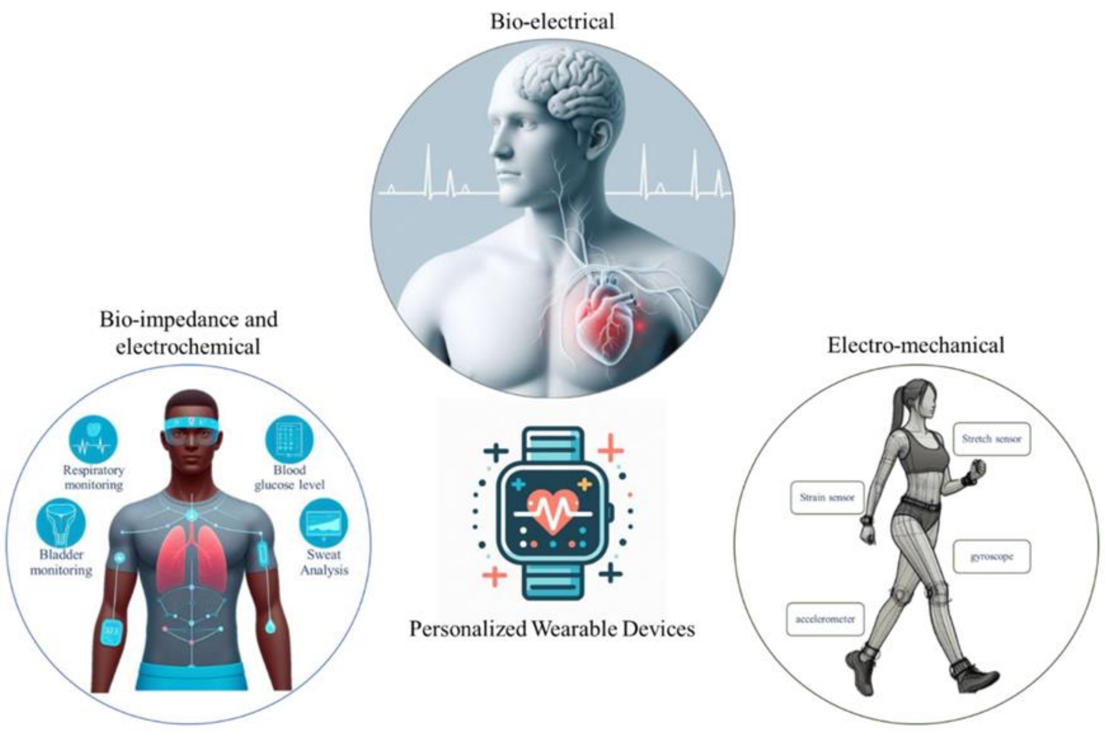


Figure 2.3 Wearable Biomedical Devices

2.1.5 Summary of previous work on deep learning

The literature study showcases technological advancements and their uses while providing a thorough overview of healthcare systems for cardiac monitoring. The first step is the creation of an IoT-based ECG monitoring system that provides real-time monitoring of vital signs including heart rate and ECG signals. This system is designed to address healthcare difficulties, particularly in low- and middle-income nations.

As a key element of remote healthcare delivery, telemedicine combines innovative information and communications technology with strong cybersecurity safeguards to deliver clinical services virtually. It also incorporates artificial intelligence and other advanced technologies.

Furthermore, RPM systems are emphasized for their function in ongoing patient monitoring beyond conventional healthcare environments, utilizing IoT networks and biosensors for information gathering and analysis.

Finally, real-time data analysis across several wearable categories allows for specific health monitoring and actions through the integration of machine learning with wearable biomedical equipment.

All things considered, these evaluations of the literature highlight how heartbeat monitoring technologies are developing and how they have the potential to improve patient care and healthcare delivery worldwide.

2.2 Limitations of Previous Studies

2.2.1 ECG-based Machine-learning algorithms

The electrocardiogram (ECG) is a tool used to record cardiac electrical activity. It is not always suitable for identifying other medical diseases or heart conditions that call for additional testing or imaging methods. Interpreting ECG data can be difficult and requires certain knowledge and experience. Making the wrong diagnosis or treatment choice can result from misinterpreting ECG signals. Specialized testing and advanced ECG monitoring devices may not be widely available or affordable in all healthcare settings, which restricts their utilization. Accurate diagnosis may be hindered by a patient's weight, lung illness, or anatomical differences, which can all have an impact on the quality and interpretation of ECG readings. False-positive or false-negative ECG readings can occasionally result in unneeded follow-up testing or missed diagnoses. An ECG may miss intermittent or temporary abnormalities that could be missed during the test since it records the heart's electrical activity at a single moment in time. Although traditional ECG recordings are usually brief, they may not be able to detect anomalies that occur infrequently or capture long-term heart activity.

2.2.2 Telemedicine in Healthcare

There are various barriers to the widespread adoption and efficacy of telemedicine in the healthcare industry. Since telemedicine does not allow for in-depth physical examinations like in-person consultations do, missed diagnoses could occur. This represents a significant drawback of the technique. Moreover, the kind of interpersonal and emotional connection that occurs during in-person visits is sometimes not possible for patients and clinicians in telemedicine.

The technology and infrastructure needs for efficient telemedicine present another set of difficulties. Among these is the requirement for dependable internet access, which is not always available, particularly in rural or underprivileged places. Since telemedicine involves communicating private medical data over the Internet, there are additional worries regarding patient data security and privacy.

The costs associated with setting up and maintaining telemedicine equipment, training medical staff, and managing these networks are high. Finally, due to varying regulations and payment rules, the financial side of telemedicine deployment may provide greater difficulties.

2.2.3 Remote Patient Monitoring (RPM) Systems

The limitations of Remote Patient Monitoring (RPM) systems can have a significant impact on their effectiveness and use in healthcare settings. Certain RPM systems might not be able to provide precise diagnostic data, which could result in significant categorization errors. Some RPM systems may have an intricate model or architecture, which can make computation time longer and present difficulties for users. RPM systems may have trouble correctly categorizing patient data, which could lead to significant classification errors. RPM systems can generate an enormous amount of data, which can be overwhelming to store and analyze, especially when connected to IoT devices. Particularly in mobile devices, RPM systems that depend on constant data transfer and processing may use a lot of energy. It is imperative that efforts be made to reduce energy use.

2.2.4 Machine Learning and Wearable Biomedical Devices

A significant limitation is the reliance on large-scale data collection, which is necessary to guarantee the precision and dependability of machine learning models but presents practical difficulties in situations involving ongoing health monitoring. Because handling and processing sensitive health information requires robust security measures to protect user privacy, privacy issues are especially significant. Furthermore, these devices may become less accessible and user-friendly to the general population due to the complexity added by advanced machine learning algorithms. The expensive cost of sophisticated sensors and complex algorithms is another crucial factor to take into account. This drives up the cost of these devices, making them inaccessible to a wider range of individuals.

Despite the benefits that machine learning presents, attaining steady precision and dependability in many real-life scenarios still poses a challenge to the successful integration of these technologies in wearable medical equipment.

Chapter 3

System Methodology/Approach

Project pre-development was the first of the project's several phases. To guarantee dependable data collection and transmission, the hardware components of the IoT-based healthcare monitoring system are carefully chosen and integrated. The software architecture is made to facilitate user engagement and communication in addition to data processing, storage, and visualization.

3.1 System Design Diagram

3.1.1 System Architecture

The system architecture for the IoT-based healthcare monitoring and alert system is carefully structured to facilitate seamless data collection, processing, storage, and communication among the different system components. The architecture is organized into several layers, each dedicated to a particular function, ensuring an efficient and integrated operation.

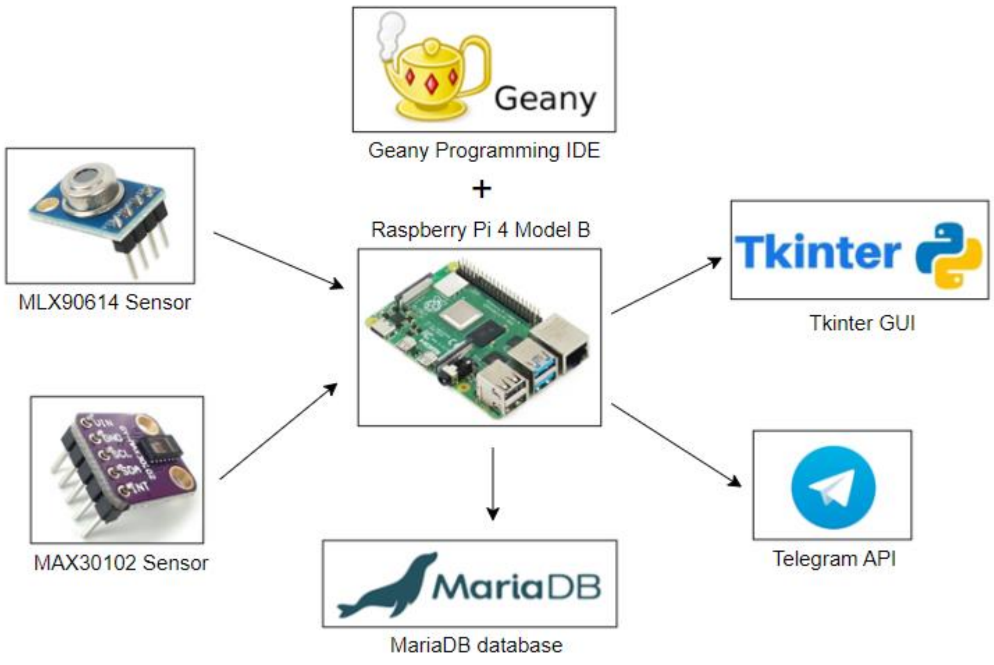


Figure 3.1 Block Diagram of the Health Monitoring System

The system architecture is composed of the following essential layers:

- **Sensor Layer**

This layer is responsible for capturing vital signs and environmental data through sensors like the MAX30102 and MLX90614. The MAX30102 sensor monitors heart rate (BPM) and blood oxygen levels (SpO_2), while the MLX90614 sensor measures body and ambient temperatures. These sensors transmit the collected data to the processing layer for further analysis.

- **Processing Layer**

The core of the system's processing activities occurs within the Raspberry Pi 4 Model B, using Python as the primary programming language. The code, developed in the Geany IDE, handles data acquisition, processing, and control logic. Python's extensive libraries enable efficient implementation of the algorithms needed to process sensor data and determine if any health anomalies require immediate attention.

- **Storage Layer**

Data management is handled by the MariaDB database, which stores all the vital sign readings and related health information. This layer ensures that the system maintains a comprehensive history of patient data, enabling healthcare providers to review past records and identify trends or changes in the patient's health over time.

- **Communication Layer**

The system uses the Telegram API to manage communication between the system and healthcare providers. When vital signs are detected, the system automatically sends alerts via Telegram, ensuring that medical professionals are promptly informed and can take necessary actions when abnormal readings are detected. This layer is crucial for real-time intervention and continuous patient monitoring.

- **User Interface Layer**

The user interface is built with Python's Tkinter library, offering a graphical interface that is intuitive and easy to navigate. This GUI allows patients and healthcare providers to view real-time data, access historical records, and interact with the system effortlessly. The interface is designed to be user-friendly, ensuring that even those with minimal technical knowledge can operate the system effectively.

3.1.1 Use Case Diagram and Description

The use case diagram shows the main interactions between users (doctors, patients, and IoT sensors) and the healthcare monitoring system. It helps in identifying the system's functional requirements and user interactions. The primary user is the patient or individual monitoring their health. The secondary user is the healthcare provider receiving alerts via Telegram.

Use Cases:
Measure vitals: User initiates data reading.
Store data: Data is automatically saved into the database after each check.
Receive alerts: Telegram sends notifications for every vital check.
View real-time data: GUI displays the user's latest health records.
View history data: GUI displays history data for the user and healthcare provider can request for history data through Telegram.

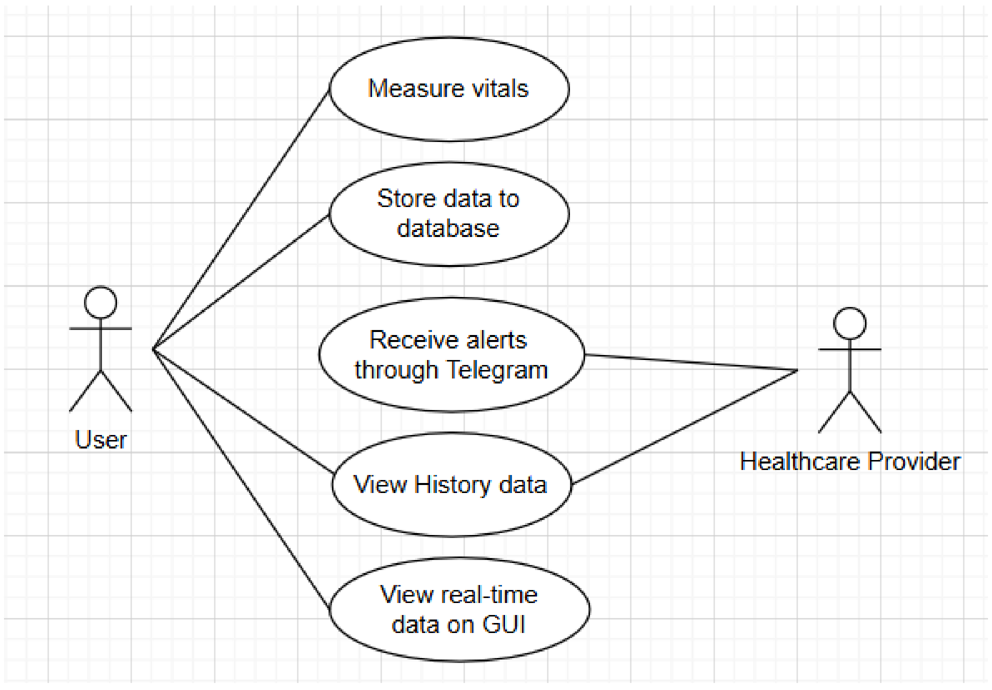


Figure 3.2 Use Case Diagram between User and Healthcare Provider

3.1.2 Activity Diagram

The activity diagram outlines the process flow of a system, showing the sequence of actions and decisions taken throughout a process.

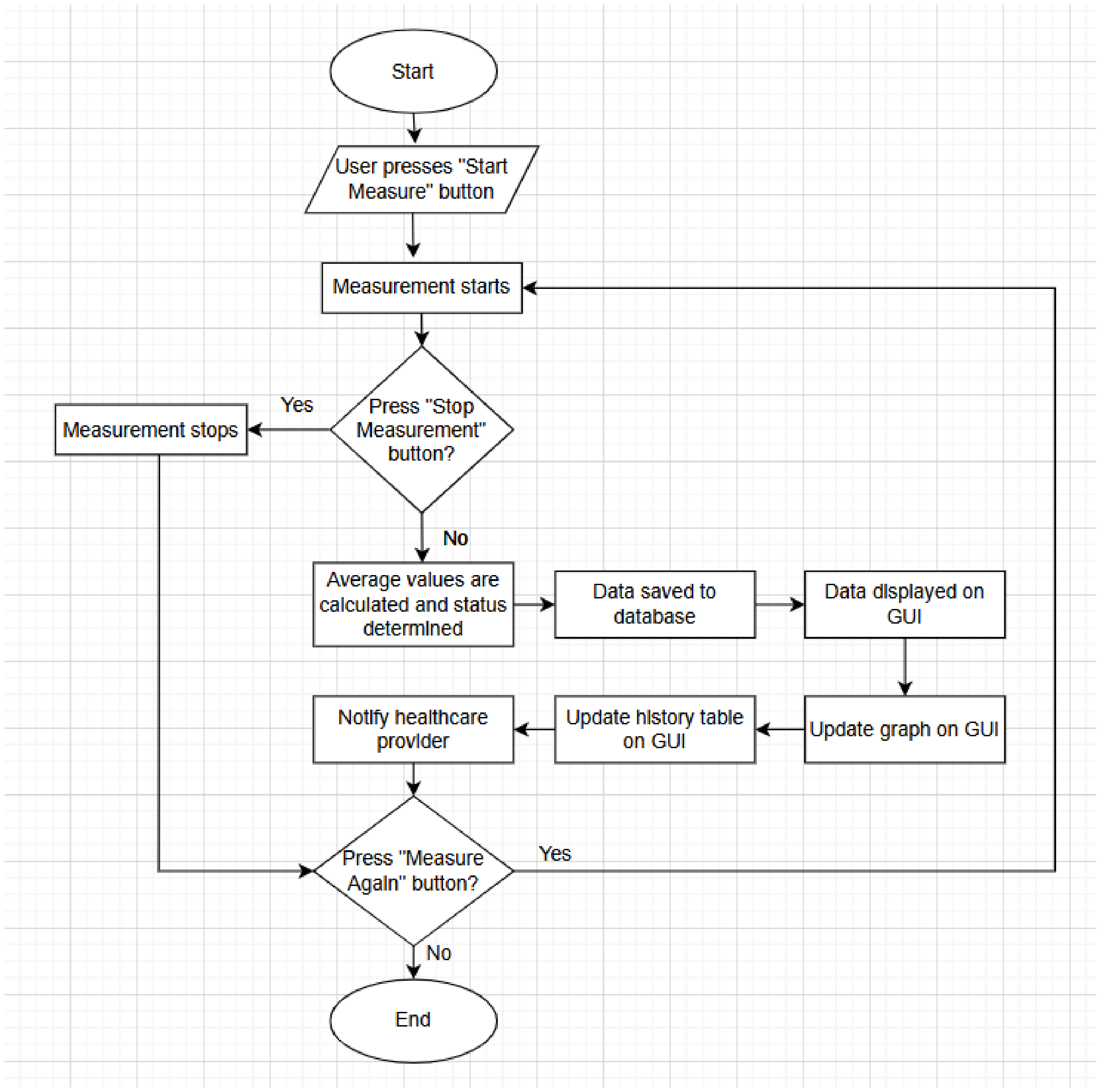

Start
User presses "Start Measure" button
Measurement starts
Press "Stop Measurement" button?
Yes
Measurement stops
No
Average values are calculated and status determined
Data saved to database
Data displayed on GUI
Update graph on GUI
Update history table on GUI
Notify healthcare provider
Press "Measure Again" button?
Yes
No
End

Figure 3.3 Flowchart of the Whole System

3.2 Timeline

The timeline for the project is divided into two main phases: FYP1 and FYP2. Each phase spans multiple weeks, focusing on different aspects of the system development and research. The Gantt Chart for both parts of the project is shown at Table 3.4 and Table 3.5.

The first phase of the project (FYP1) focused on laying the foundation for the system by defining the project scope and objectives over the initial two weeks. Following this, research on sensors and IoT systems was conducted for three weeks to identify suitable components and technologies. Simultaneously, a literature review was undertaken to understand the existing knowledge and gaps in the field. The system design, including hardware and software selection, was carried out over a four-week period to ensure all components were well-integrated. By Week 6, the hardware and software setup began, taking three weeks to complete. Integration with the database and testing of the sensor data was done in the subsequent four weeks to ensure functionality. Finally, the preliminary results were gathered, analyzed, and documented in a report. This phase also included the preparation of a presentation and poster for the project, with these tasks running concurrently with report writing in the final four weeks.

Table 3.1 Gantt Chart for FYP1

No.	**Project Activities**	**W1**	**W 2**	**W 3**	**W 4**	**W 5**	**W 6**	**W 7**	**W 8**	**W9**	**W10**	**W11**	**W12**
1	Project Planning, Scope & Objectives	■	■										
2	Research On Sensors & IoT System		■	■	■								
3	Literature Review		■	■	■								
4	Components Selection & System Design				■	■	■	■					
5	Hardware & Software Setup						■	■	■				
6	System Integration with Database & Testing								■	■	■	■	
7	Obtain Preliminary Results & Report Writing									■	■	■	■

FYP2 builds upon the foundation established in FYP1, focusing on enhancing the system by adding a Graphical User Interface (GUI) using Tkinter. The GUI development spans five weeks, providing users with a more accessible and user-friendly way to interact with the system. Additionally, an alert system using Telegram API to connect to the system is developed over five weeks, starting from Week 3, allowing real-time notifications for health monitoring. System integration and testing take place from Week 6 to Week 10, ensuring that all components, including the GUI and alert system, function cohesively. In parallel, data collection and system analysis begin in Week 7, running for six weeks to provide insights into the system’s performance. Lastly, to ensure the Raspberry Pi has better protection and a more professional look, a case is designed and printed out within a few weeks. The project concludes with four weeks dedicated to report writing and presentation preparation, where the final system's results and analysis are documented in a comprehensive report and prepared for presentation.

Table 3.2 Gantt Chart for FYP 2

No.	**Project Activities**	**W1**	**W2**	**W3**	**W4**	**W5**	**W6**	**W7**	**W8**	**W9**	**W 10**	**W 11**	**W 12**	**W 13**
1	GUI Development	■	■	■	■	■								
2	Alert System Development			■	■	■	■	■						
3	System Integration & Testing						■	■	■	■	■			
4	Data Collection & System Analysis							■	■	■	■	■	■	
5	Casing design & 3D printing									■	■	■	■	
6	Report Writing & Presentation										■	■	■	■

This timeline ensures that the project progresses in an organized manner, balancing research, development, testing, and documentation within the period for both FYP1 and FYP2.

Chapter 4
System Design

This chapter explains the structure and internal workings of the IoT-based healthcare monitoring system. It outlines the specifications of the hardware and software components, how the circuits were designed, and how the system components interact to carry out data collection, processing, and communication.

4.1 System Components Specifications

This section outlines the hardware and software components used in the IoT-based healthcare monitoring and alert system.

4.1.1 Hardware

- **Raspberry Pi 4 Model B**

The IoT-based healthcare monitoring system's central processing unit was a Raspberry Pi 4 Model B. It enabled both data collecting and remote data transmission by offering the computing capacity and adaptability needed for data processing, system control, and communication with external hardware components via GPIO pins, USB ports, and additional connection choices including Wi-Fi and Bluetooth.

Figure 4.1 Raspberry Pi 4 Model B

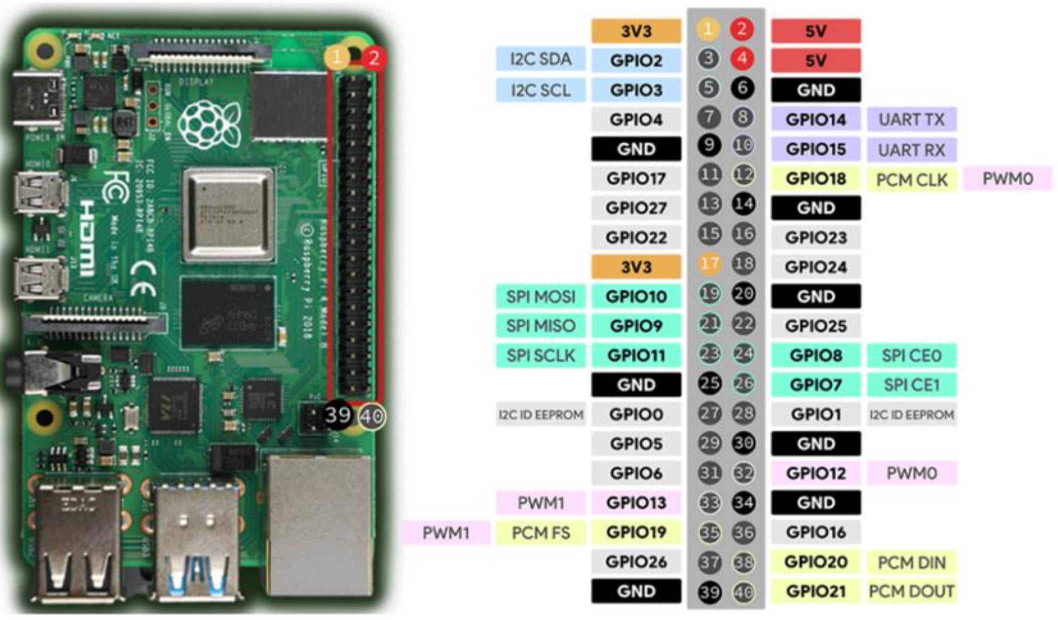


Figure 4.2 Raspberry Pi 4 Model B Pinout Diagram

Table 4.1 Specifications of Raspberry Pi 4 Model B

Specifications	Parameters
Processor	Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit SoC @1.5GHz
Memory	1GB, 2GB or 4GB LPDDR4 (depending on model)
Connectivity	2.4GHz and 5.0GHz IEEE 802.11b/g/n/ac wireless LAN, Bluetooth 5.0, BLE Gigabit Ethernet
GPIO	Standard 40-pin GPIO header
Input Power	5V DC

- **MAX30102 Sensor**

The MAX30102 is a specialized sensor designed for the non-invasive monitoring of heart rate (BPM) and blood oxygen saturation levels (SpO_2). It integrates two LEDs (one emitting red light and the other infrared light), a photodetector, and optimized optics. The red and infrared light are absorbed differently by oxygenated and deoxygenated blood, allowing the sensor to measure SpO_2 levels based on the ratio of absorbed light. Additionally, the sensor uses the light absorption variation caused by the pulsing of blood in the arteries to calculate the heart rate. The MAX30102's compact design and low power consumption make it ideal for wearable health monitoring devices. The sensor is interfaced with the Raspberry Pi through I^2C communication, ensuring efficient data transfer and real-time monitoring capabilities.

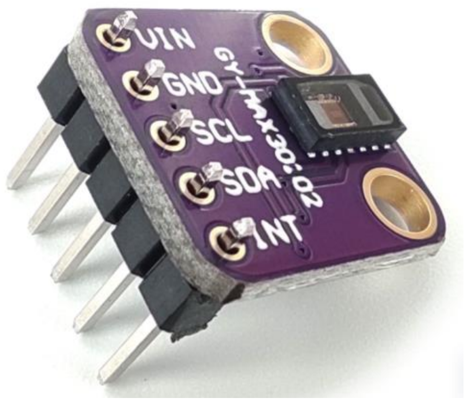


Figure 4.3 MAX30102 Sensor

Table 4.2 Specifications of MAX30102 Sensor

Specifications	**Parameters**
Power consumption	Very low, suitable for mobile devices
Programmable sample rate and LED current	Energy-saving
Sampling rates	High
Communication protocol	Standard I^2C, 0x57 fixed 7-bit address
Detection Signal Type	Optical Reflection Signal (PPG)
Immunity to movement artifacts	Solid
Signal-to-Noise Ratio (SNR)	High SNR
Power supply	3.3-5V
Operating temperature range	-40°C to 85°C
PCB size	14 x 14mm

- **MLX90614 Sensor**

The MLX90614 is an infrared thermometer sensor that measures both body and ambient temperatures without direct contact. It operates based on the principle of infrared radiation, where the sensor detects the infrared energy emitted by objects within its field of view and converts it into a temperature reading. This sensor features a built-in digital signal processor and a 17-bit analog-to-digital converter, providing high accuracy and sensitivity. It is particularly useful for continuous monitoring of body temperature in a healthcare setting, as it can detect even small variations in temperature, which may be indicative of changes in a patient's condition. The MLX90614 is connected to the Raspberry Pi via the I²C interface, allowing for seamless integration into the healthcare monitoring system and enabling real-time temperature data collection and analysis.

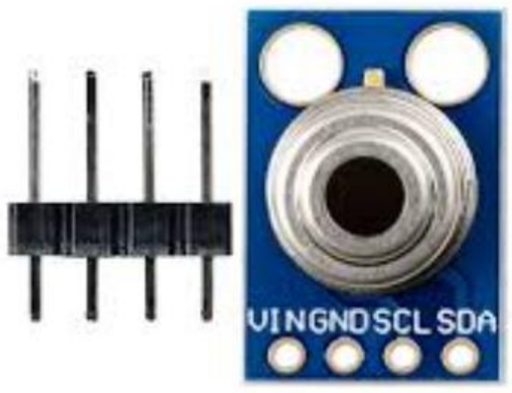


Figure 4.4 MLX90614 Sensor

Table 4.3 Specifications of MLX90614 Sensor

Specifications	**Parameters**
Size	11.5 x 16.5mm (+-2mm)
Power supply	3-5V DC (internal low dropout regulator)
Communication protocol	Standard I²C, 0x5A fixed 7-bit address
Calibration	Factory calibrated
Sensor temperature range	-40°C to +125°C
Object temperature range	-70°C to +380°C
Temperature accuracy	±0.5°C around room temperatures
Field of view	90°
Power-saving feature	Power saving mode
Grade	Automotive-grade

- **Dupont jumper wires**

Dupont jumper wires are essential to connect the Raspberry Pi’s GPIO pins to the sensors via the breadboard. The flexible male-to-female and male-to-male jumper wires are used to enable the sharing of some GPIO pins, ensuring efficient use of available connections on the Raspberry Pi. Female-to-female jumper wires are used mostly to connect Raspberry Pi’s GPIO pins directly to the sensors. Y-split jumper wires are used to split the I²C connection to accommodate both sensors using the same GPIO pins without connecting to a breadboard.

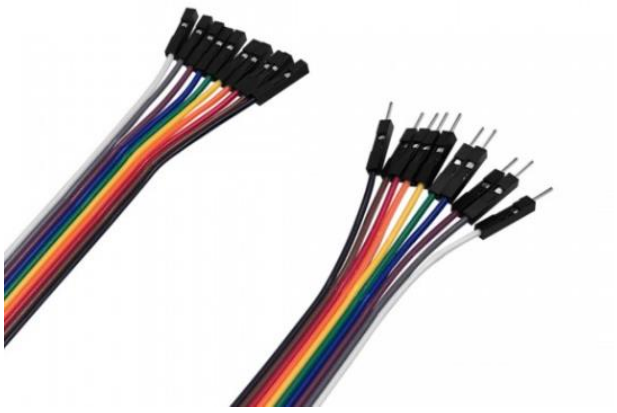


Figure 4.5 Dupont jumper wires

Table 4.4 Specifications of Dupont Jumper Wires

Specifications	**Parameters**
Connector type	DuPont (2.54 mm pitch)
Wire Type	28 AWG (standard), some are 26 or 30 AWG
Wire Material	Usually tinned copper or copper-clad aluminum (CCA)
Insulation Material	PVC (Polyvinyl Chloride)
Voltage Rating	Up to 300V DC (depending on insulation)
Current Capacity	Around 1A (for 28 AWG wires)
Temperature Range	-20°C to +80°C
Pin Type	Male to Male (M-M), Male to Female (M-F), Female to Female (F-F)

- **3D-printed case**

The 3D-printed case is necessary for protecting the Raspberry Pi or holding sensors in place during operation.

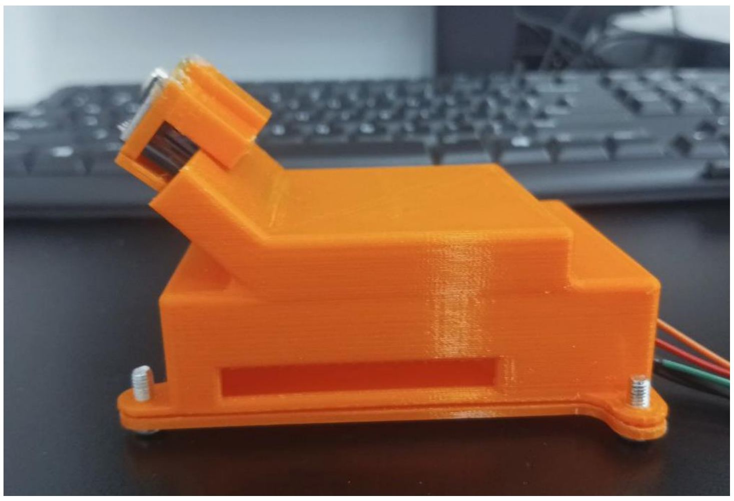


Figure 4.6 3D-printed Raspberry Pi Case

Table 4.5 Specifications of 3D-printed Raspberry Pi Case

Specifications	Parameters
Material	PLA (Polylactic Acid)
Color	Orange
Design Software	SolidWorks
Printing Technology	Fused Deposition Modeling (FDM)
Printer Used	Creality Ender 3 & Raise3D N2
Layer Height	0.2 mm
Infill Density	20%
Print Time	6 hrs 22 mins
Mounting Features	Holes for sensor brackets, slots for ventilation

4.1.2 Software

The IoT-based healthcare monitoring system's software architecture consists of a number of essential parts that make data processing, management, and visualization easier.

- **Python programming**

Python is the main programming language used to create the system's software components. Python is a flexible and user-friendly programming language that can be used to construct a wide range of functionalities, such as data processing methods, communication protocols, and user interfaces. It also has a large library support.

Figure 4.7 Python Logo

- **Geany Programming IDE**

Geany is a lightweight and fast integrated development environment (IDE) used for writing and editing the Python code that powers the system. It provides essential features like syntax highlighting, code folding, and an integrated terminal, making it a convenient tool for software development. Its simplicity and efficiency make it ideal for managing the project's codebase.

Figure 4.8 Geany Programming Logo

- **Tkinter**

Tkinter is the standard Python library used for creating the graphical user interface (GUI) of the healthcare monitoring system. It provides a simple yet powerful toolkit for designing the front end of the system, allowing users to visualize real-time health data, access historical records, and interact with various features of the application. The interface is designed to be user-friendly, ensuring that both patients and healthcare providers can easily navigate the system.

Figure 4.9 Tkinter Python Logo

- **MariaDB**

MariaDB is the chosen relational database management system (RDBMS) for this project, providing a robust platform for storing and managing the healthcare data collected from the sensors. MariaDB is known for its high performance, security, and compatibility with other systems, making it an excellent choice for handling sensitive medical data. In this system, MariaDB stores critical information such as heart rate, SpO_2 levels, and temperature readings, along with timestamps and patient identifiers. The database structure is designed to allow easy querying and retrieval of historical data, which is essential for tracking patient health trends over time.

Figure 4.10 MariaDB Logo

- **Telegram API**

The Telegram API is utilized to implement the alert system within the healthcare monitoring application. When the system detects that any health parameters fall outside of the normal range, it will automatically send an alert to healthcare providers via Telegram. This real-time notification system ensures that healthcare professionals are promptly informed of any potential issues, allowing them to take immediate action if necessary.

Figure 4.11 Telegram API Logo

4.2 Circuits and Components Design

The hardware components in this system are arranged within a custom-designed 3D-printed Raspberry Pi case. Instead of using the initial breadboard-based design during FYP1 prototyping, the final prototype utilizes custom wire connections neatly routed inside a 3D-printed case.

The sensors are connected to the Raspberry Pi through the GPIO and I²C interface. To achieve dual I²C sensor communication, the I²C interface is shared between the two sensors via a Y-split jumper cable as shown in Figure 4.12, allowing both sensors to share the same SDA and SCL lines while maintaining individual power and ground connections through additional female-to-female jumper wires.

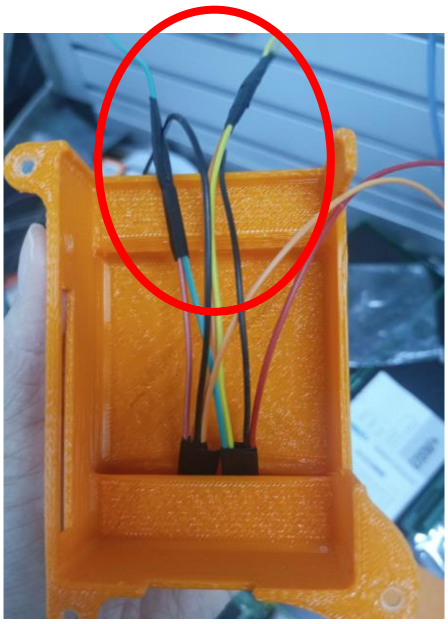


Figure 4.12 Y-split jumper cables

The MAX30102 sensor, used for measuring heart rate and oxygen saturation levels, and the MLX90614 sensor, used for recording both body and ambient temperatures, are strategically mounted at a 45-degree angle on top of the case. This is to allow users to place their fingers comfortably and consistently while measuring their vitals. This ergonomic orientation ensures more accurate readings while enhancing user experience. The internal wiring is concealed within the case to minimize visual clutter and prevent accidental disconnections, resulting in a cleaner and more professional appearance. The case plays a crucial role in stabilizing the sensors, organizing internal wiring, and improving the overall usability and aesthetics of the device.

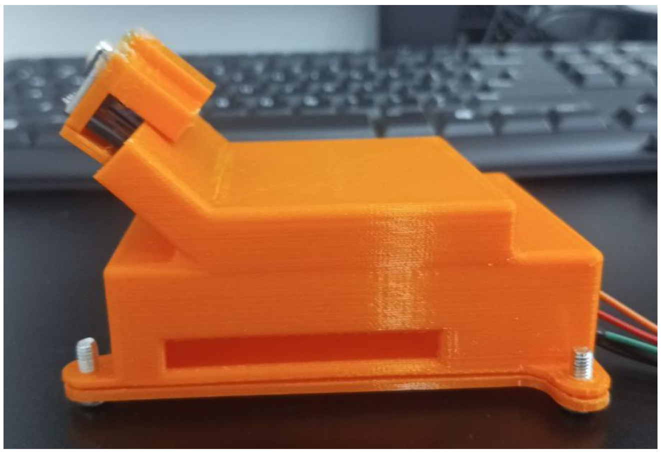


Figure 4.13 3D-Printed Raspberry Pi Case

Power is supplied to the Raspberry Pi via the USB-C port. A mouse and keyboard are connected through the USB ports, while a micro-HDMI to VGA adapter links the Raspberry Pi to a monitor for GUI display. This compact and organized layout not only enhances system durability and portability but also supports efficient sensor operation and user interaction.

4.3 System Components Interaction Operations

The healthcare monitoring and alert system is designed to provide real-time tracking and analysis of vital health parameters, specifically focusing on heart rate (BPM), blood oxygen saturation (SpO_2), and body temperature. The system integrates various sensors, a Raspberry Pi for real-time processing, Tkinter-based GUI for user interaction, MariaDB database for data storage, and Telegram for instant health data notifications to healthcare providers, enabling an effective solution for continuous health monitoring.

At the core of the system are the MAX30102 and MLX90614 sensors. When the user initiates a measurement through the GUI, the system begins by activating both sensors via the I^2C interface. The MAX30102 sensor continuously measures heart rate and oxygen saturation, while the MLX90614 sensor monitors body and ambient temperature. These sensors are connected to a Raspberry Pi, which acts as the system’s central processing unit. The Raspberry Pi collects raw data from the sensors, processes it to calculate average values, and then classifies the readings as normal, high, or low based on predefined thresholds.

The processed data is stored in a MariaDB database, which records each measurement along with a timestamp and status indicators for historical tracking and analysis. Simultaneously, the real-time data is then displayed on a user-friendly interface built with Tkinter. The Tkinter-based GUI presents current readings, the last 10 database entries in a table format, and a visual graph showing the trends of all vitals, allowing users to view real-time and historical data.

The system also includes an alert mechanism. Every time a measurement is taken, the system will send a formatted message containing the vitals to healthcare providers via Telegram. If any health parameters fall outside of the normal range, a warning message will be sent along with the vital data, allowing healthcare providers to receive instant updates. This ensures remote monitoring is not only possible but proactive.

The interaction between hardware and software components is coordinated by Python scripts, from sensor activation to GUI output and Telegram alerts. These scripts also include logic for interpreting sensor signals, performing basic validations, and ensuring data consistency before database entry and message transmission.

In summary, this system provides a robust and scalable solution for monitoring key health metrics, offering users detailed insights and timely alerts. By combining reliable hardware components with efficient data processing and storage, the system ensures that critical health information is both accessible and actionable.

Chapter 5
System Implementation

This chapter outlines the steps taken to implement the IoT-based healthcare monitoring and alert system, covering the hardware and software setup, configuration procedures, system operation, encountered challenges, and final remarks on the implementation phase.

5.1 Hardware Setup

The initial step in setting up the healthcare monitoring and alert system involved assembling the hardware components. The Raspberry Pi 4 Model B was selected as the central processing unit due to its versatility and sufficient processing power for handling the sensor data and running the necessary software. The following components were connected:

- **MAX30102 Sensor**: This sensor is connected directly to the Raspberry Pi's I²C interface using a Y-split jumper wire. It is mounted on top of a custom 3D-printed case at a 45-degree angle to provide a more ergonomic position for users to place their finger for measurements of heart rate (BPM) and SpO_2 levels.
- **MLX90614 Sensor**: Connect to the same I²C interface using the shared Y-split jumper wire, this sensor is fixed beside the MAX30102 on the 3D-printed case. This sensor was used to measure both body temperature and ambient temperature.
- **Jumper Wires**: Essential for simplifying the wiring and ensuring stable connections between the Raspberry Pi and the sensors. **Y-type** jumper wires are used to split the SDA and SCL I²C lines between both sensors. Additional **female-to-female** jumper wires are used to connect the VCC and GND pins from the Raspberry Pi to the sensors. The wiring is organized within the 3D-printed case to maintain a neat layout.
- **Custom 3D-Printed Case**: Designed to securely hold the Raspberry Pi and both sensors, this enclosure hides internal wiring for a cleaner look and includes angled mounting (45°) for better finger placement and user comfort during measurements.
- **Power Supply and Peripheral Connections**:
  - The Raspberry Pi is powered via the USB-C power port.
  - A mouse and keyboard are connected through the USB ports for input control.
  - A monitor is connected to the Raspberry Pi's micro-HDMI port, using a micro-HDMI to VGA adapter to support VGA displays during setup and operation.

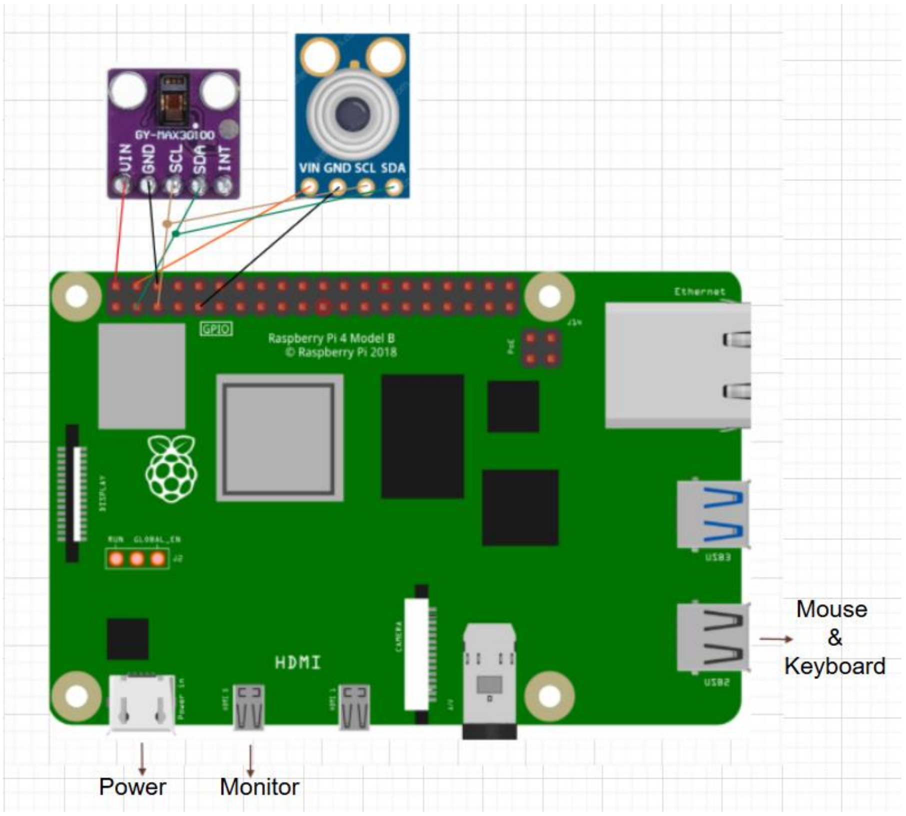


Figure 5.1 Hardware Pinout Diagram

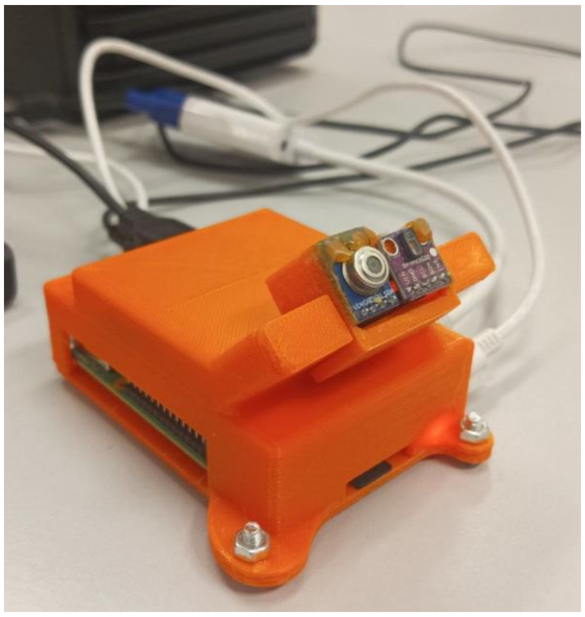


Figure 5.2 Hardware Setup

Table 5.1 Pin Connection of Raspberry Pi and MAX30102

Raspberry Pi	**MAX30102**
5V (Pin 2)	VIN
I²C SDA (GPIO 2; Pin 3)	SDA
I²C SCL (GPIO 3; Pin 5)	SCL
GND (Pin 6)	GND

Table 5.2 Pin Connection of Raspberry Pi and MLX90614

Raspberry Pi	**MLX90614**
5V (Pin 3)	VIN
I²C SDA (GPIO 2; Pin 3)	SDA
I²C SCL (GPIO 3; Pin 5)	SCL
GND (Pin 9)	GND

5.2 Software Setup

The software setup for the healthcare monitoring and alert system involved the integration of various Python scripts that interact with the hardware components (MAX30102 and MLX90614 sensors) and manage the system's operations. The Geany programming IDE is used to provide a user-friendly environment for writing and debugging the Python code.
Essential libraries are imported to enable certain functionalities:
time: Used for introducing delays, managing time intervals between measurements, and timestamping data entries.
numpy: Employed for numerical computations and calculating the average values of heart rate and SpO_2 from collected samples.
argparse: Included to handle command-line arguments, if needed, particularly useful during testing or debugging phases.
smbus2: This library is critical for I²C communication between the Raspberry Pi and the connected sensors (MAX30102 and MLX90614).

mysql.connector: Enables direct interaction with the MariaDB database to store and retrieve sensor data entries.
tkinter: Provides the framework for the system's graphical user interface (GUI), allowing users to view real-time readings, graphs, and historical data.
matplotlib: Used to generate and display graphs that visualize trends in vital signs, enhancing user insight.
telepot: Handles Telegram bot communication, allowing the system to send vital sign readings to caregivers or remote users instantly upon each measurement.

Below is a breakdown of the essential Python scripts and the libraries used within each.

- **MAX30102**

The MAX30102 sensor is responsible for measuring heart rate (BPM) and blood oxygen saturation (SpO_2). The following scripts were developed to handle data collection, processing, and storage from the MAX30102 sensor:

- *hrcalc.py*: This script contains the necessary functions to calculate heart rate and SpO_2 from the raw data collected by the MAX30102 sensor.
  Library used:
  - ➔ *'numpy'*: For efficient numerical computations.

- *max30102.py*: This script provides the necessary functions to interface with the MAX30102 sensor. It includes functions to read data from the sensor's FIFO registers and handle the sensor's configuration.
  Library used:
  - ➔ *'time'*: For introducing delays in the data collection process.
  - ➔ *'smbus'*: For I^2C communication with the MAX30102 sensor.

- *heartrate_monitor.py*: This script acts as a wrapper around the MAX30102 sensor, providing a class that manages the sensor's operation, including starting, stopping, and running the sensor in a separate thread. It processes the raw data, calculates the BPM and SpO_2, and handles alerts for abnormal readings.
  Library used:
  - ➔ *'numpy'*: For processing and averaging BPM and SpO_2 values.
  - ➔ *'max30102'*: Import *max30102.py* file to interface with MAX30102 sensor.

  - ➔ *'hrcalc'*: Import *hrcalc.py* file to process raw data and calculate the BPM and SpO_2.
  - ➔ *'threading'*: To run the sensor in a separate thread, enabling concurrent data collection.

'time': For timing and delays in the data collection loop.

- **MLX90614**

The MLX90614 sensor is responsible for measuring body and ambient temperature. The following script was developed to interface with the MLX90614 sensor:

- *mlx90614.py*: This script provides functions to read temperature data from the MLX90614 sensor, including both object (body) and ambient temperature readings.

  Library used:

'time': For introducing delays as necessary during data collection.

- **Combine Operation of MAX30102 and MLX90614**

The main script that combines the functionalities of both the MAX30102 and MLX90614 sensors, integrating their data collection and storage processes:

- *combined.py*: This script integrates the heart rate, oxygen saturation level, and temperature monitoring functionalities. It reads data from both the MAX30102 and MLX90614 sensors, processes the data, and stores it in the MariaDB database. It also provides the user with real-time feedback and stores results in a way that includes alerts for high, low, or normal readings.

  Library used:
  - ➔ *'numpy'*: For data processing and calculations.
  - ➔ *'heartrate_monitor'*: Import *heartrate_monitor.py* file for managing the heart rate and SpO_2 data collection.
  - ➔ *'mlx90614'*: Import *mlx90614.py* file for interfacing with the MLX90614 sensor.
  - ➔ *'smbus2'*: For I^2C communication with both sensors.
  - ➔ *'mysql.connector'*: To connect to the MariaDB database and executing SQL queries to store and retrieve data.
  - ➔ *'time'*: For timing and delays in data collection and overall operation.

- ➔ *'argparse'*: For parsing command-line arguments to customize sensor operation (e.g., duration of data collection).
- ➔ *'tkinter'*: To provides the framework for the system's GUI
- ➔ *'matplotlib'*: To generate and display graphs of the data collection.
- ➔ *'telepot'*: To handle Telegram bot communication

These scripts collectively manage the healthcare monitoring and alert system, ensuring accurate data collection, storage and user interaction.
All scripts were tested and executed in Python 3 on the Raspberry Pi OS. The GUI and all backend processes run locally on the Raspberry Pi to ensure offline capability and low latency. Proper error handling was implemented to ensure the system remains responsive even if the sensor temporarily fails or network connectivity to Telegram is disrupted.

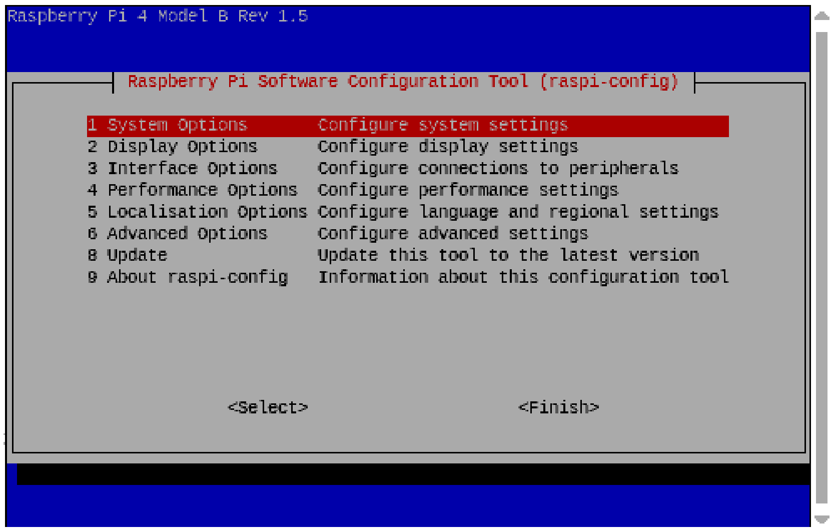
5.3 Setting and Configuration

The system required several essential configurations to ensure seamless integration between the hardware, software, database, and external services like Telegram. These settings were carefully implemented on the Raspberry Pi to establish full system functionality.

- **I²C Interface Activation**

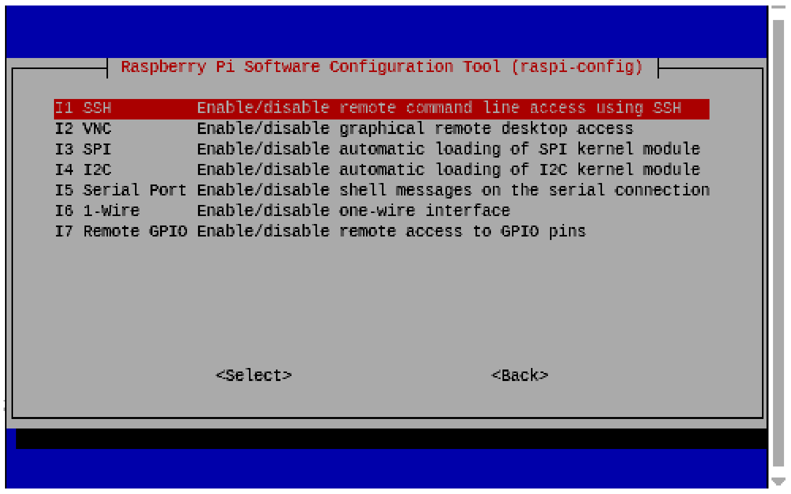
The I²C interface was enabled through the Raspberry Pi Configuration Tool to allow communication with the MAX30102 and MLX90614 sensors.
This was done by:

1. Opening the terminal and running: *sudo raspi-config*
2. Navigating to *Interface Options* > *I2C* and enabling it



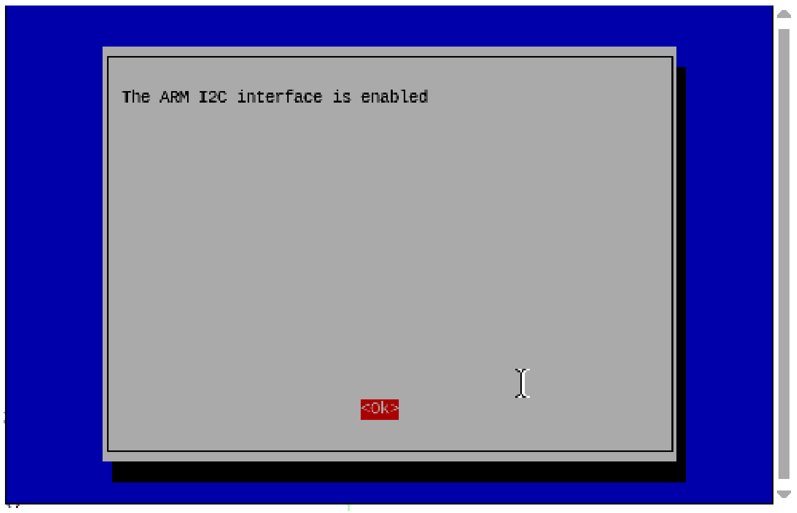
```
Raspberry Pi 4 Model B Rev 1.5

        Raspberry Pi Software Configuration Tool (raspi-config)

        1 System Options       Configure system settings
        2 Display Options      Configure display settings
        3 Interface Options    Configure connections to peripherals
        4 Performance Options  Configure performance settings
        5 Localisation Options Configure language and regional settings
        6 Advanced Options     Configure advanced settings
        8 Update               Update this tool to the latest version
        9 About raspi-config   Information about this configuration tool

                    <Select>                    <Finish>
```

Figure 5.3 Raspberry Pi Configuration Tool



```
        Raspberry Pi Software Configuration Tool (raspi-config)

    I1 SSH         Enable/disable remote command line access using SSH
    I2 VNC         Enable/disable graphical remote desktop access
    I3 SPI         Enable/disable automatic loading of SPI kernel module
    I4 I2C         Enable/disable automatic loading of I2C kernel module
    I5 Serial Port Enable/disable shell messages on the serial connection
    I6 1-Wire      Enable/disable one-wire interface
    I7 Remote GPIO Enable/disable remote access to GPIO pins

                    <Select>                    <Back>
```

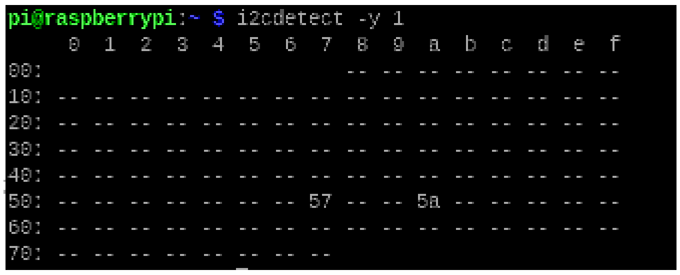
Figure 5.4 Raspberry Pi Interface Options



```
The ARM I2C interface is enabled

                     <Ok>
```

Figure 5.5 ARM I²C Interface Enabled Notification

3. Verifying I²C connections with the command: *i2cdetect -y 1*



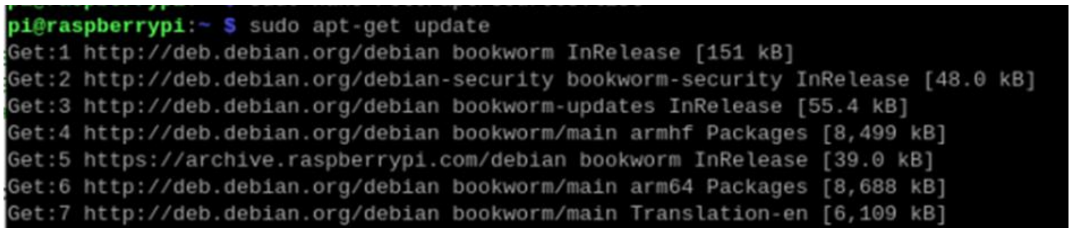
```
pi@raspberrypi:~ $ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:                         -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- 57 -- -- 5a -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Figure 5.6 Verification of I²C connections

- **MariaDB**

MariaDB was set up as the database management system to store the collected health data. The database schema was designed to efficiently handle the storage of sensor readings and other relevant patient information.

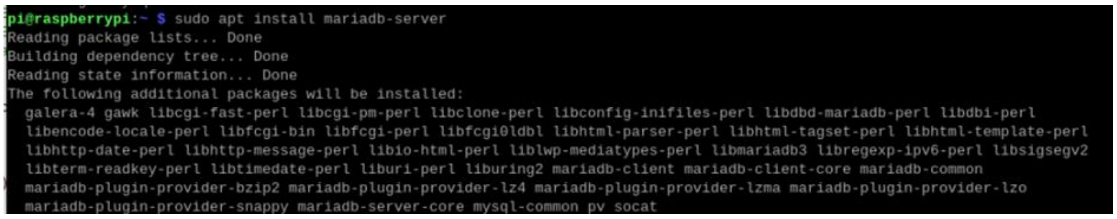
1. Before installing MariaDB, update the system to the latest packages:
   - *sudo apt-get update*



```
pi@raspberrypi:~ $ sudo apt-get update
Get:1 http://deb.debian.org/debian bookworm InRelease [151 kB]
Get:2 http://deb.debian.org/debian-security bookworm-security InRelease [48.0 kB]
Get:3 http://deb.debian.org/debian bookworm-updates InRelease [55.4 kB]
Get:4 http://deb.debian.org/debian bookworm/main armhf Packages [8,499 kB]
Get:5 https://archive.raspberrypi.com/debian bookworm InRelease [39.0 kB]
Get:6 http://deb.debian.org/debian bookworm/main arm64 Packages [8,688 kB]
Get:7 http://deb.debian.org/debian bookworm/main Translation-en [6,109 kB]
```

Figure 5.7 Update Command Result

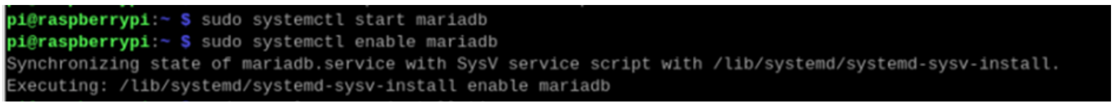
2. Install MariaDB database:
   - *sudo apt install mariadb-server*



```
pi@raspberrypi:~ $ sudo apt install mariadb-server
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  galera-4 gawk libcgi-fast-perl libcgi-pm-perl libclone-perl libconfig-inifiles-perl libdbd-mariadb-perl libdbi-perl
  libencode-locale-perl libfcgi-bin libfcgi-perl libfcgi0ldbl libhtml-parser-perl libhtml-tagset-perl libhtml-template-perl
  libhttp-date-perl libhttp-message-perl libio-html-perl liblwp-mediatypes-perl libmariadb3 libregexp-ipv6-perl libsigsegv2
  libterm-readkey-perl libtimedate-perl liburi-perl liburing2 mariadb-client mariadb-client-core mariadb-common
  mariadb-plugin-provider-bzip2 mariadb-plugin-provider-lz4 mariadb-plugin-provider-lzma mariadb-plugin-provider-lzo
  mariadb-plugin-provider-snappy mariadb-server-core mysql-common pv socat
```

Figure 5.8 Install MariadDB Command Result

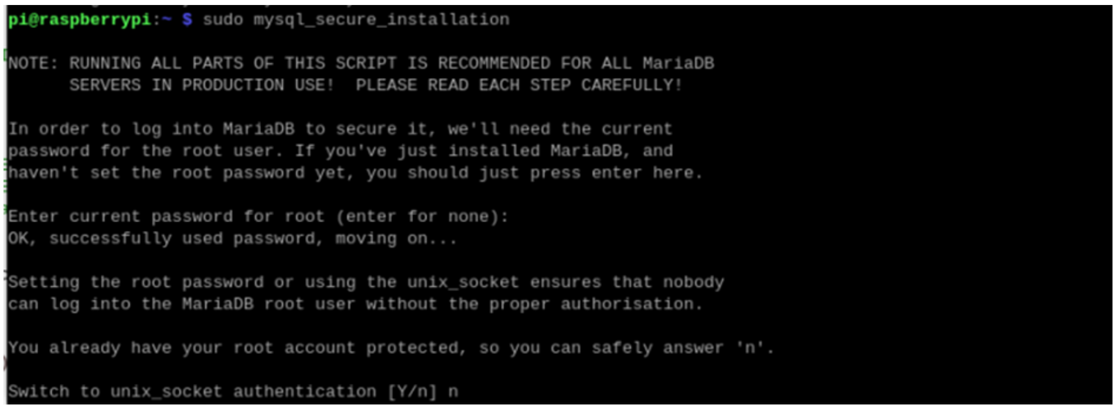
3. After installing MariaDB Server, start the service with these two commands:
   - *sudo systemctl start mariadb*
   - *sudo systemctl enable mariadb*



```
pi@raspberrypi:~ $ sudo systemctl start mariadb
pi@raspberrypi:~ $ sudo systemctl enable mariadb
Synchronizing state of mariadb.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable mariadb
```

Figure 5.9 Start MariaDB Service Command Result

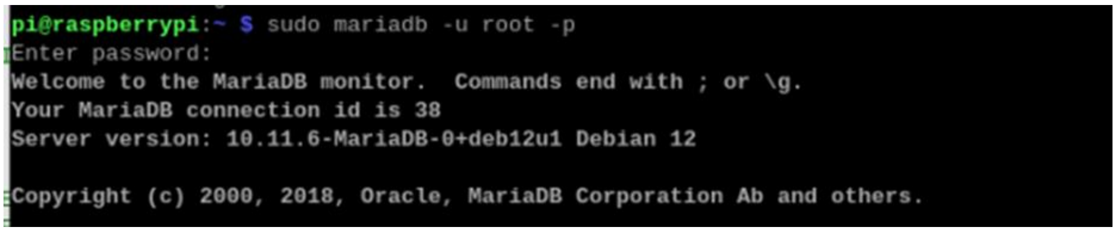
4. Run the security script:
   - *sudo mysql_secure_installation*



```
pi@raspberrypi:~ $ sudo mysql_secure_installation

NOTE: RUNNING ALL PARTS OF THIS SCRIPT IS RECOMMENDED FOR ALL MariaDB
      SERVERS IN PRODUCTION USE!  PLEASE READ EACH STEP CAREFULLY!

In order to log into MariaDB to secure it, we'll need the current
password for the root user. If you've just installed MariaDB, and
haven't set the root password yet, you should just press enter here.

Enter current password for root (enter for none):
OK, successfully used password, moving on...

Setting the root password or using the unix_socket ensures that nobody
can log into the MariaDB root user without the proper authorisation.

You already have your root account protected, so you can safely answer 'n'.

Switch to unix_socket authentication [Y/n] n
```

Figure 5.10 Secure Script Command Result

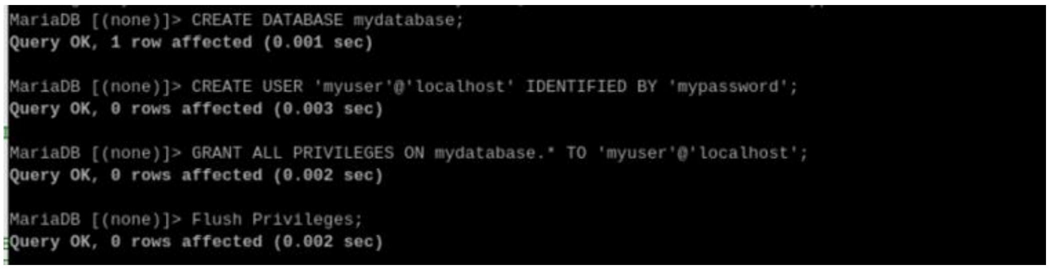
5. Login to MariaDB as root user:
   - *sudo mariadb -u root -p*



```
pi@raspberrypi:~ $ sudo mariadb -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 38
Server version: 10.11.6-MariaDB-0+deb12u1 Debian 12

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.
```

Figure 5.11 MariadDB Login Command Result

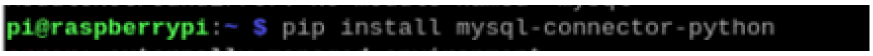
6. Create a new database and a new user and grant privileges:
   - *CREATE DATABASE mydatabase;*
   - *CREATE USER 'myuser'@'localhost' IDENTIFIED BY 'mypassword';*
   - *GRANT ALL PRIVILEGES ON mydatabase.* TO 'myuser'@'localhost';*
   - *FLUSH PRIVILEGES;*



```
MariaDB [(none)]> CREATE DATABASE mydatabase;
Query OK, 1 row affected (0.001 sec)

MariaDB [(none)]> CREATE USER 'myuser'@'localhost' IDENTIFIED BY 'mypassword';
Query OK, 0 rows affected (0.003 sec)

MariaDB [(none)]> GRANT ALL PRIVILEGES ON mydatabase.* TO 'myuser'@'localhost';
Query OK, 0 rows affected (0.002 sec)

MariaDB [(none)]> Flush Privileges;
Query OK, 0 rows affected (0.002 sec)
```

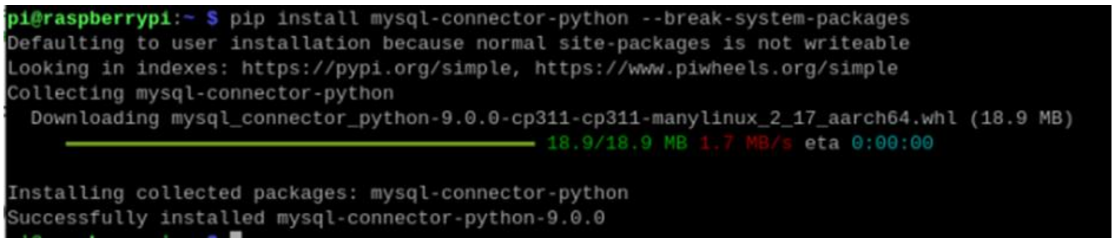
Figure 5.12 Create Database and User Command Result

7. Exit the MariaDB shell by typing *'exit'*
8. Install the MySQL connector for Python:
   - *pip install mysql-connector-python*



```
pi@raspberrypi:~ $ pip install mysql-connector-python
```

Figure 5.13 SQL Connector Command



```
pi@raspberrypi:~ $ pip install mysql-connector-python --break-system-packages
Defaulting to user installation because normal site-packages is not writeable
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting mysql-connector-python
  Downloading mysql_connector_python-9.0.0-cp311-cp311-manylinux_2_17_aarch64.whl (18.9 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 18.9/18.9 MB 1.7 MB/s eta 0:00:00

Installing collected packages: mysql-connector-python
Successfully installed mysql-connector-python-9.0.0
```

Figure 5.14 SQL Connector Command Result

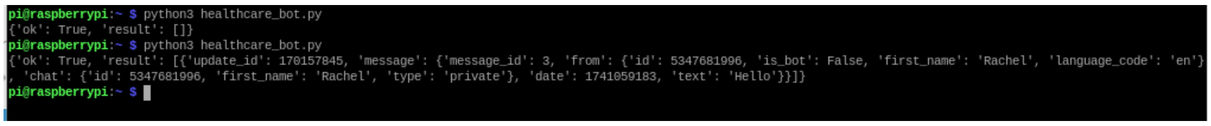
Import *'mysql.connector'* library at *combined.py* and include the *mysql.connector.connect()* method to establish a connection to your MySQL database using the credentials and database name you provide.

- **Telegram**

A Telegram bot was set up to enable real-time remote data sharing. The bot was created using 'BotFather' on Telegram, and the generated bot token was integrated into the Python script using the *telepot* library. To enable the bot to send messages to the correct recipient, the chat ID is required.

There are two ways to obtain the telegram bot *@rachelHealthcareBot* chat ID:

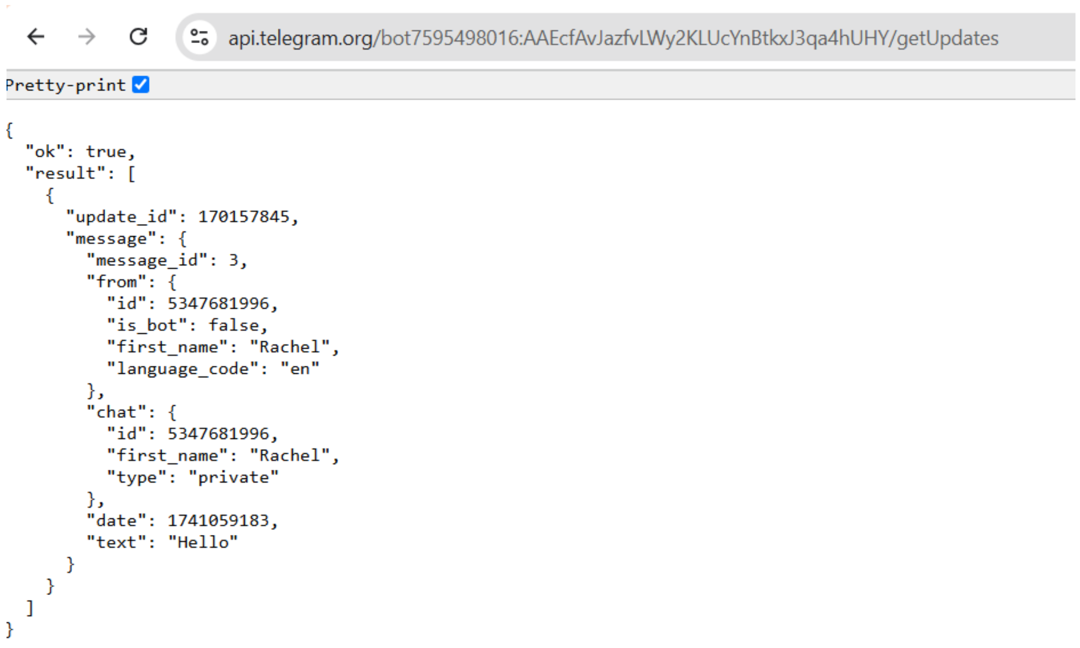
Method 1 (Terminal): Send a message to the bot and use a script or *telepot* to print the chat ID from incoming messages.



```
pi@raspberrypi:~ $ python3 healthcare_bot.py
{'ok': True, 'result': []}
pi@raspberrypi:~ $ python3 healthcare_bot.py
{'ok': True, 'result': [{'update_id': 170157845, 'message': {'message_id': 3, 'from': {'id': 5347681996, 'is_bot': False, 'first_name': 'Rachel', 'language_code': 'en'}
, 'chat': {'id': 5347681996, 'first_name': 'Rachel', 'type': 'private'}, 'date': 1741059183, 'text': 'Hello'}}]}
pi@raspberrypi:~ $ 
```

Figure 5.15 Terminal Method to obtain chat ID

Method 2: (Web Tool): Use a third-party website such as *https://api.telegram.org/bot<your_token>/getUpdates* to view the chat ID after sending a message to the bot.



api.telegram.org/bot7595498016:AAEcfAvJazfvLWy2KLUcYnBtkxJ3qa4hUHY/getUpdates

Pretty-print ☑

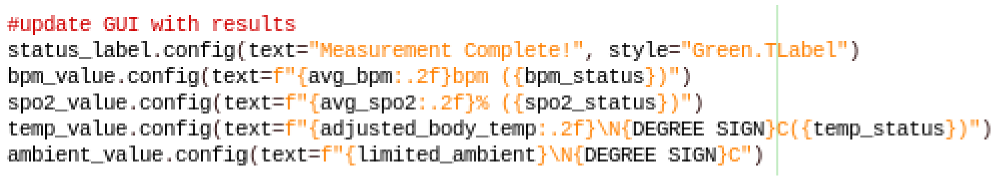
```
{
  "ok": true,
  "result": [
    {
      "update_id": 170157845,
      "message": {
        "message_id": 3,
        "from": {
          "id": 5347681996,
          "is_bot": false,
          "first_name": "Rachel",
          "language_code": "en"
        },
        "chat": {
          "id": 5347681996,
          "first_name": "Rachel",
          "type": "private"
        },
        "date": 1741059183,
        "text": "Hello"
      }
    }
  ]
}
```

Figure 5.16 Web Tool Method to obtain chat ID

The bot was programmed to automatically send formatted messages containing the user's vital signs after each user-initiated measurement, allowing healthcare providers or users themselves to receive immediate updates remotely.

- **Tkinter GUI**

The Tkinter GUI was configured to run on the Raspberry Pi desktop environment:
The GUI windows were designed to be resolution-friendly for the connected HDMI monitor. It allows users to interact with the system without needing to use the terminal or command line. The main interface includes buttons to initiate real-time measurements and displays live data for heart rate, SpO2, and body temperature.

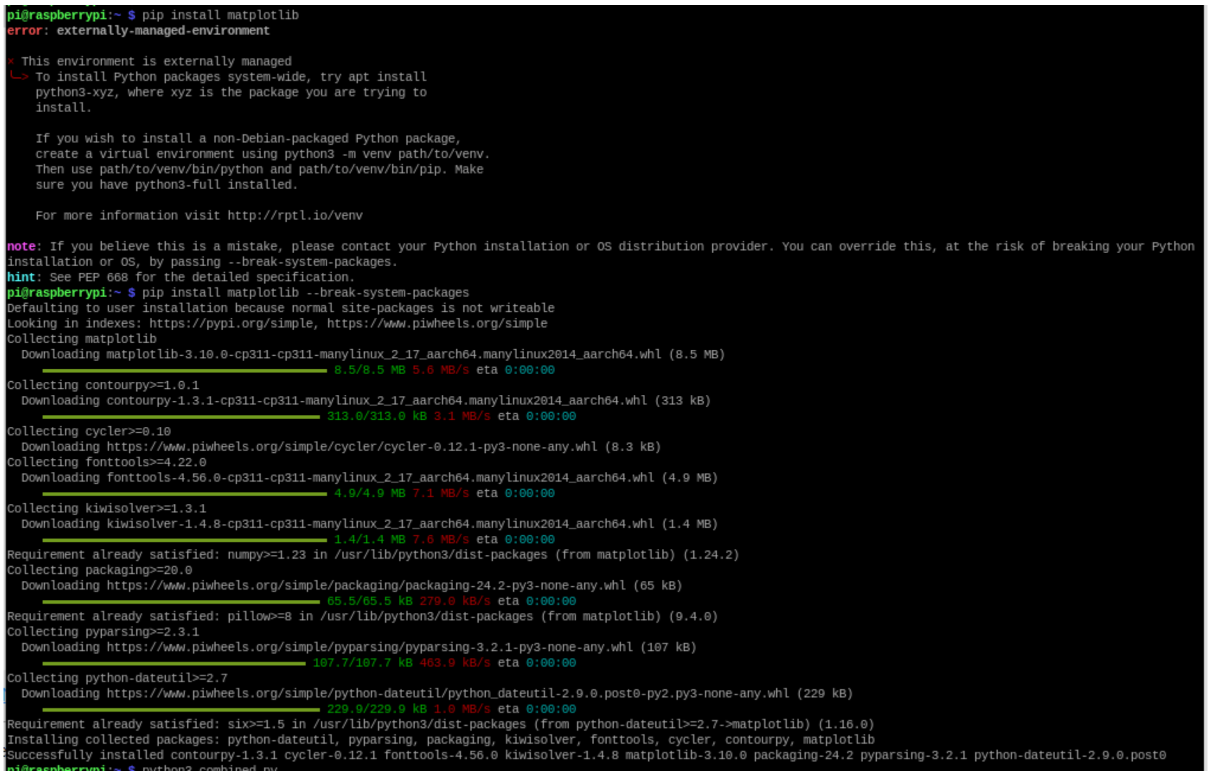


```
#update GUI with results
status_label.config(text="Measurement Complete!", style="Green.TLabel")
bpm_value.config(text=f"{avg_bpm:.2f}bpm ({bpm_status})")
spo2_value.config(text=f"{avg_spo2:.2f}% ({spo2_status})")
temp_value.config(text=f"{adjusted_body_temp:.2f}\N{DEGREE SIGN}C({temp_status})")
ambient_value.config(text=f"{limited_ambient}\N{DEGREE SIGN}C")
```

Figure 5.17 Codes for updating results on GUI

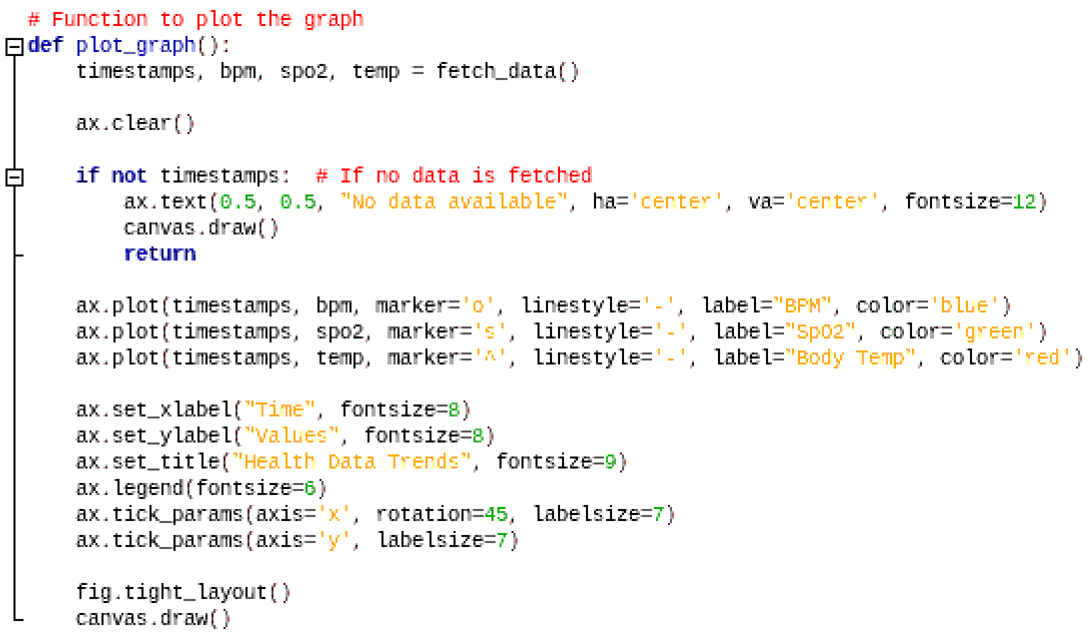
Moreover, a graphing section was implemented using the *matplotlib* library to visualize trends in the vital signs over time. A command is used to install the *matplotlib* library:

- *pip install matplotlib –break-system-packages*



```
pi@raspberrypi:~ $ pip install matplotlib
error: externally-managed-environment

× This environment is externally managed
╰─> To install Python packages system-wide, try apt install
    python3-xyz, where xyz is the package you are trying to
    install.

    If you wish to install a non-Debian-packaged Python package,
    create a virtual environment using python3 -m venv path/to/venv.
    Then use path/to/venv/bin/python and path/to/venv/bin/pip. Make
    sure you have python3-full installed.

    For more information visit http://rptl.io/venv

note: If you believe this is a mistake, please contact your Python installation or OS distribution provider. You can override this, at the risk of breaking your Python
installation or OS, by passing --break-system-packages.
hint: See PEP 668 for the detailed specification.
pi@raspberrypi:~ $ pip install matplotlib --break-system-packages
Defaulting to user installation because normal site-packages is not writeable
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting matplotlib
  Downloading matplotlib-3.10.0-cp311-cp311-manylinux_2_17_aarch64.manylinux2014_aarch64.whl (8.5 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.5/8.5 MB 5.6 MB/s eta 0:00:00
Collecting contourpy>=1.0.1
  Downloading contourpy-1.3.1-cp311-cp311-manylinux_2_17_aarch64.manylinux2014_aarch64.whl (313 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 313.0/313.0 kB 3.1 MB/s eta 0:00:00
Collecting cycler>=0.10
  Downloading https://www.piwheels.org/simple/cycler/cycler-0.12.1-py3-none-any.whl (8.3 kB)
Collecting fonttools>=4.22.0
  Downloading fonttools-4.56.0-cp311-cp311-manylinux_2_17_aarch64.manylinux2014_aarch64.whl (4.9 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.9/4.9 MB 7.1 MB/s eta 0:00:00
Collecting kiwisolver>=1.3.1
  Downloading kiwisolver-1.4.8-cp311-cp311-manylinux_2_17_aarch64.manylinux2014_aarch64.whl (1.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.4/1.4 MB 7.6 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.23 in /usr/lib/python3/dist-packages (from matplotlib) (1.24.2)
Collecting packaging>=20.0
  Downloading https://www.piwheels.org/simple/packaging/packaging-24.2-py3-none-any.whl (65 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 65.5/65.5 kB 279.0 kB/s eta 0:00:00
Requirement already satisfied: pillow>=8 in /usr/lib/python3/dist-packages (from matplotlib) (9.4.0)
Collecting pyparsing>=2.3.1
  Downloading https://www.piwheels.org/simple/pyparsing/pyparsing-3.2.1-py3-none-any.whl (107 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 107.7/107.7 kB 463.9 kB/s eta 0:00:00
Collecting python-dateutil>=2.7
  Downloading https://www.piwheels.org/simple/python-dateutil/python_dateutil-2.9.0.post0-py2.py3-none-any.whl (229 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 229.9/229.9 kB 1.0 MB/s eta 0:00:00
Requirement already satisfied: six>=1.5 in /usr/lib/python3/dist-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
Installing collected packages: python-dateutil, pyparsing, packaging, kiwisolver, fonttools, cycler, contourpy, matplotlib
Successfully installed contourpy-1.3.1 cycler-0.12.1 fonttools-4.56.0 kiwisolver-1.4.8 matplotlib-3.10.0 packaging-24.2 pyparsing-3.2.1 python-dateutil-2.9.0.post0
pi@raspberrypi:~ $ python3 combined.py
```

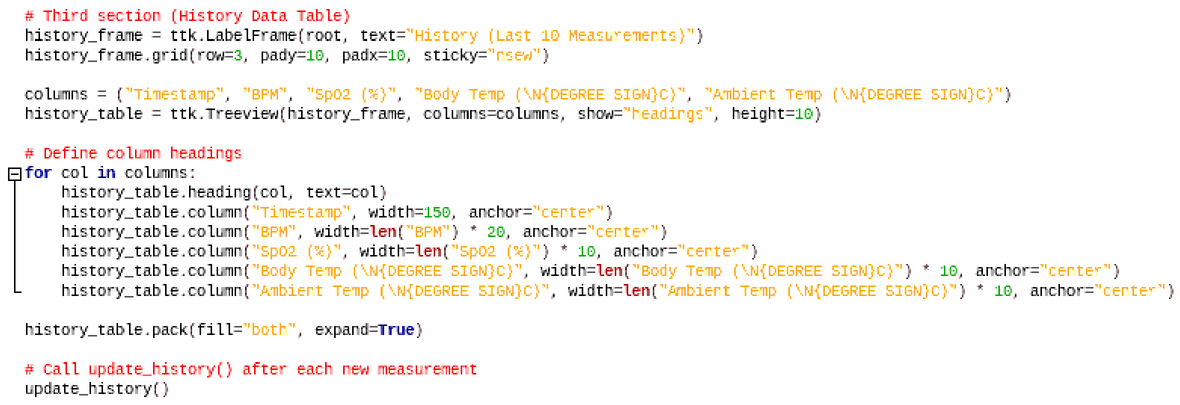
Figure 5.18 Installation of matplotlib Library



```python
# Function to plot the graph
def plot_graph():
    timestamps, bpm, spo2, temp = fetch_data()

    ax.clear()

    if not timestamps:  # If no data is fetched
        ax.text(0.5, 0.5, "No data available", ha='center', va='center', fontsize=12)
        canvas.draw()
        return

    ax.plot(timestamps, bpm, marker='o', linestyle='-', label="BPM", color='blue')
    ax.plot(timestamps, spo2, marker='s', linestyle='-', label="SpO2", color='green')
    ax.plot(timestamps, temp, marker='^', linestyle='-', label="Body Temp", color='red')

    ax.set_xlabel("Time", fontsize=8)
    ax.set_ylabel("Values", fontsize=8)
    ax.set_title("Health Data Trends", fontsize=9)
    ax.legend(fontsize=6)
    ax.tick_params(axis='x', rotation=45, labelsize=7)
    ax.tick_params(axis='y', labelsize=7)

    fig.tight_layout()
    canvas.draw()
```

Figure 5.19 Function Code to Plot Graph on GUI

In addition to real-time display, the GUI features a table showing the 10 most recent data entries retrieved from the MariaDB database, providing users with a quick historical overview.



```
# Third section (History Data Table)
history_frame = ttk.LabelFrame(root, text="History (Last 10 Measurements)")
history_frame.grid(row=3, pady=10, padx=10, sticky="nsew")

columns = ("Timestamp", "BPM", "SpO2 (%)", "Body Temp (\N{DEGREE SIGN}C)", "Ambient Temp (\N{DEGREE SIGN}C)")
history_table = ttk.Treeview(history_frame, columns=columns, show="headings", height=10)

# Define column headings
for col in columns:
    history_table.heading(col, text=col)
    history_table.column("Timestamp", width=150, anchor="center")
    history_table.column("BPM", width=len("BPM") * 20, anchor="center")
    history_table.column("SpO2 (%)", width=len("SpO2 (%)") * 10, anchor="center")
    history_table.column("Body Temp (\N{DEGREE SIGN}C)", width=len("Body Temp (\N{DEGREE SIGN}C)") * 10, anchor="center")
    history_table.column("Ambient Temp (\N{DEGREE SIGN}C)", width=len("Ambient Temp (\N{DEGREE SIGN}C)") * 10, anchor="center")

history_table.pack(fill="both", expand=True)

# Call update_history() after each new measurement
update_history()
```

Figure 5.20 Code to Update History Table on GUI

The integration of Tkinter enhances accessibility and ease of use, making the system more practical for non-technical users.

5.4 System Operation (with Screenshot)

The system begins operation through a graphical user interface (GUI) developed using the Tkinter library. When the application is first used for the first time, the database does not contain any records yet, the GUI will display a message indicating that no data is available.

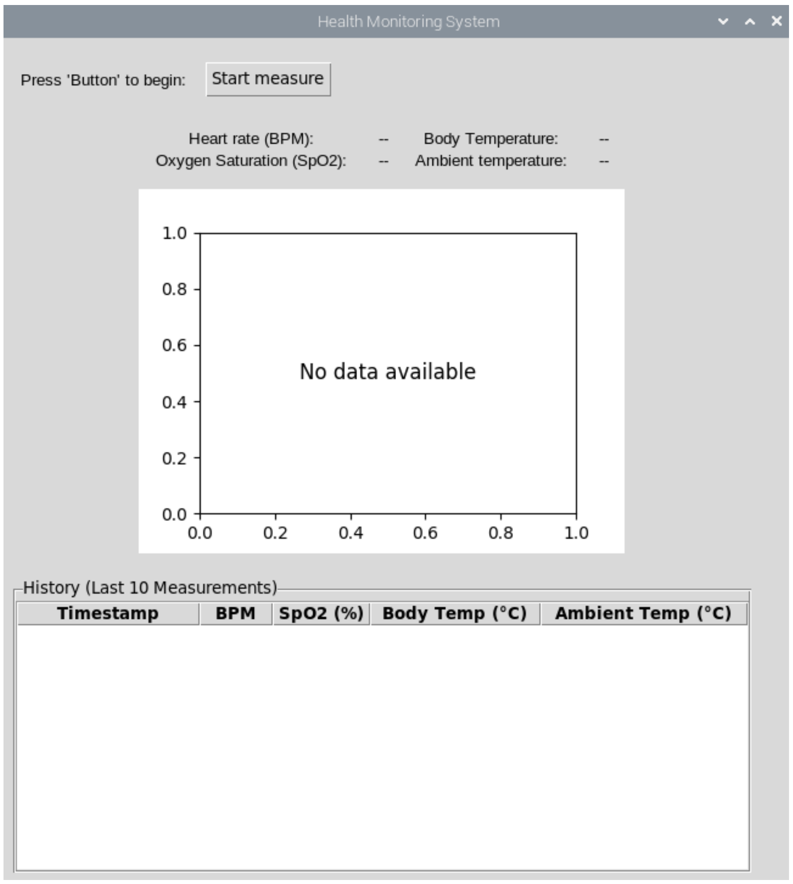


Figure 5.21 GUI When No Data is Available in The Database

When the user presses the "Start Measure" button on the Tkinter interface, the button itself will be disabled to prevent multiple readings from being initiated simultaneously. At the same time, a "Stop measure" button will appear, allowing the user to stop the measurement process at any point before it completes.

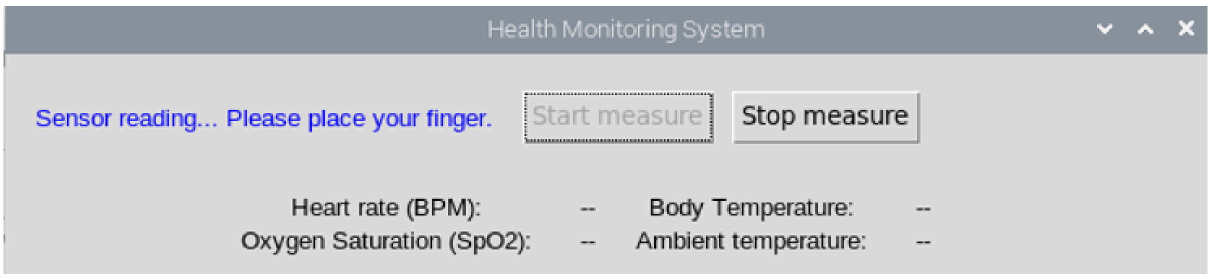
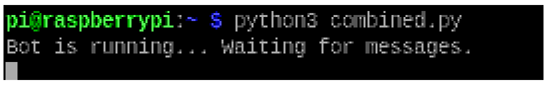


Figure 5.22 GUI Indicating Sensor is Reading

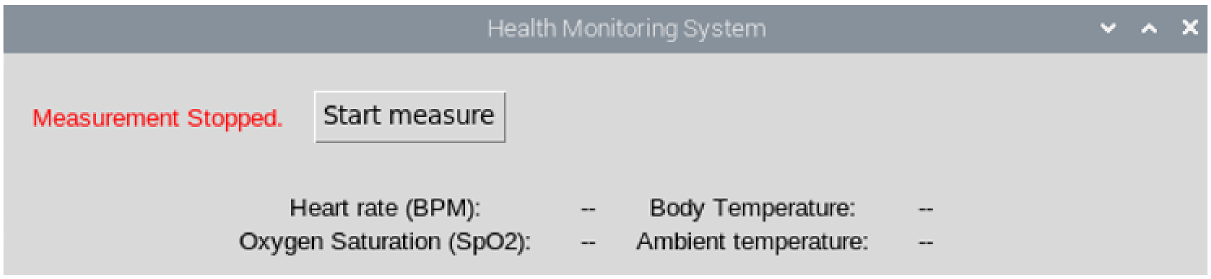
The system will then begin its operation by initializing the sensors and setting the Telegram bot to standby mode, ready to send messages.



```
pi@raspberrypi:~ $ python3 combined.py
Bot is running... Waiting for messages.
```

Figure 5.23 Terminal Indicating Telegram in Standby Mode

The user is then required to place their fingers on the sensor surface angled at 45 degrees for the MAX30102 and MLX90614 sensors to begin collecting data. If the user decides to press the “Stop measure” button, the ongoing measurement will be halted, a message “Measurement stopped” will be displayed, and the “Start measure” button will be re-enabled, giving the user the option to initiate a new measurement.



Health Monitoring System

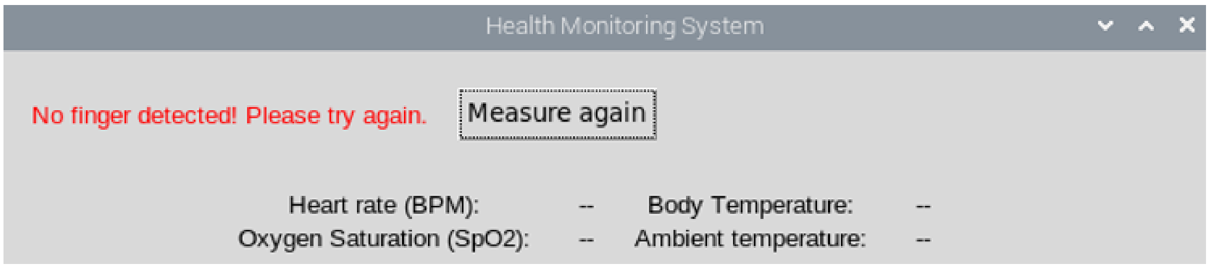
Measurement Stopped.

Start measure

Heart rate (BPM): -- Body Temperature: --
Oxygen Saturation (SpO2): -- Ambient temperature: --

Figure 5.24 GUI Indicating Measurement Stopped

If fingers are not detected during sensor reading, the system will display “Finger not detected” and there will be a “Measure again” button displayed for user to start a new measurement. This process ensures that only accurate and valid health data is processed and stored.



Health Monitoring System

No finger detected! Please try again.

Measure again

Heart rate (BPM): -- Body Temperature: --
Oxygen Saturation (SpO2): -- Ambient temperature: --

Figure 5.25 GUI Indicating No Finger Detected

To ensure accurate readings, proper finger placement is shown in the picture below.

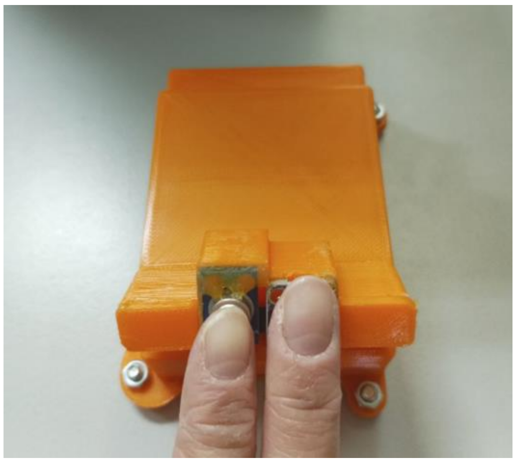
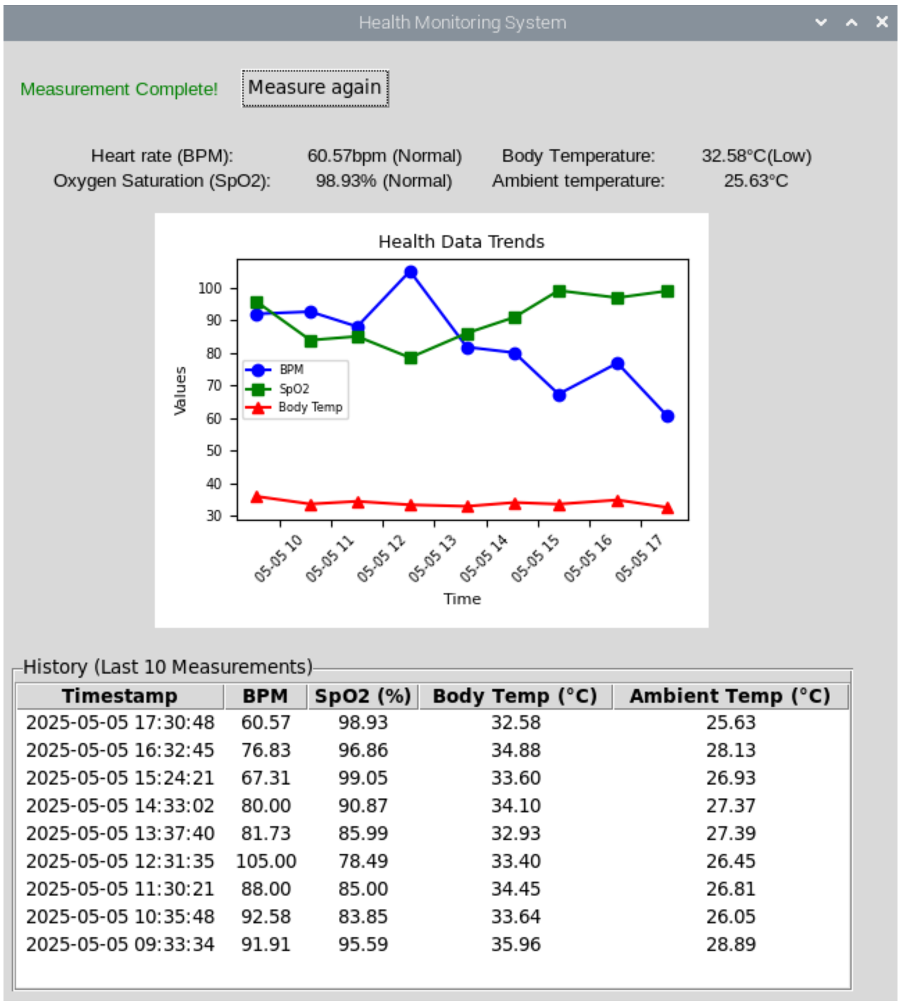


Figure 5.26 Finger Placement on Both Sensors

During the sensor reading period, provided the user's finger is correctly placed on the sensors, the system actively collects heart rate and oxygen saturation data for 20 seconds. Throughout this duration, multiple individual readings are taken, and the system calculates the average BPM and SpO2 values from these readings. Both the BPM and SpO2 data are then classified as either "Normal," "Low," or "High" based on predefined thresholds. Immediately following this, the system reads the user's body temperature using the MLX90614 sensor. The temperature value and status are displayed immediately after the reading is taken.

When readings for heart rate, oxygen level, and body temperature are obtained, it will display “Measurement Complete!” at the GUI, and a “Measure again” button will appear for the user to initiate another new measurement. At the same time, the system will display the values in real-time on the GUI and store them in the MariaDB database. Simultaneously, the graph that visualizes recent data trends and a table showing the last ten records will also be updated on the GUI, as shown below.



Measurement Complete!

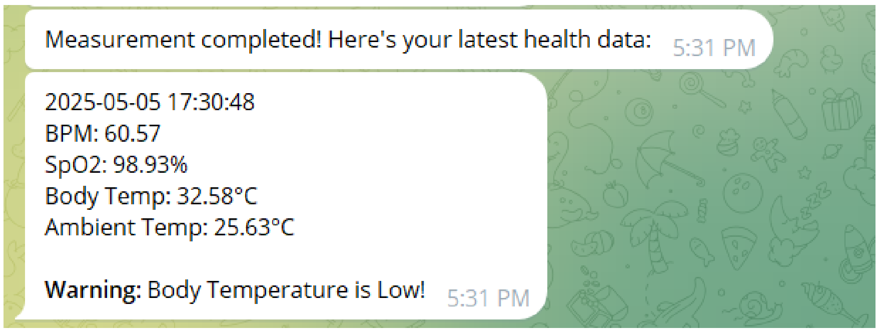
Heart rate (BPM): 60.57bpm (Normal) Body Temperature: 32.58°C(Low)
Oxygen Saturation (SpO2): 98.93% (Normal) Ambient temperature: 25.63°C

History (Last 10 Measurements)

Timestamp	BPM	SpO2 (%)	Body Temp (°C)	Ambient Temp (°C)
2025-05-05 17:30:48	60.57	98.93	32.58	25.63
2025-05-05 16:32:45	76.83	96.86	34.88	28.13
2025-05-05 15:24:21	67.31	99.05	33.60	26.93
2025-05-05 14:33:02	80.00	90.87	34.10	27.37
2025-05-05 13:37:40	81.73	85.99	32.93	27.39
2025-05-05 12:31:35	105.00	78.49	33.40	26.45
2025-05-05 11:30:21	88.00	85.00	34.45	26.81
2025-05-05 10:35:48	92.58	83.85	33.64	26.05
2025-05-05 09:33:34	91.91	95.59	35.96	28.89

Figure 5.27 Tkinter GUI of the Measurement

A Telegram alert containing the results is instantly sent via Telegram to the designated healthcare provider. This ensures timely communication and facilitates remote health supervision.

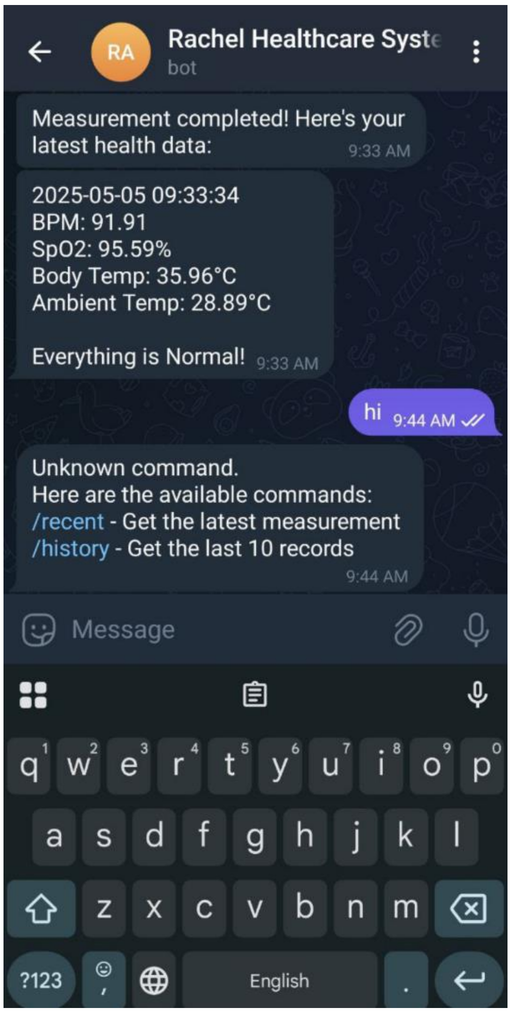


Measurement completed! Here's your latest health data: 5:31 PM

2025-05-05 17:30:48
BPM: 60.57
SpO2: 98.93%
Body Temp: 32.58°C
Ambient Temp: 25.63°C

Warning: Body Temperature is Low! 5:31 PM

Figure 5.28 Telegram Message for the Measurement


Rachel Healthcare Syste
bot

Measurement completed! Here's your latest health data: 9:33 AM

2025-05-05 09:33:34
BPM: 91.91
SpO2: 95.59%
Body Temp: 35.96°C
Ambient Temp: 28.89°C

Everything is Normal! 9:33 AM

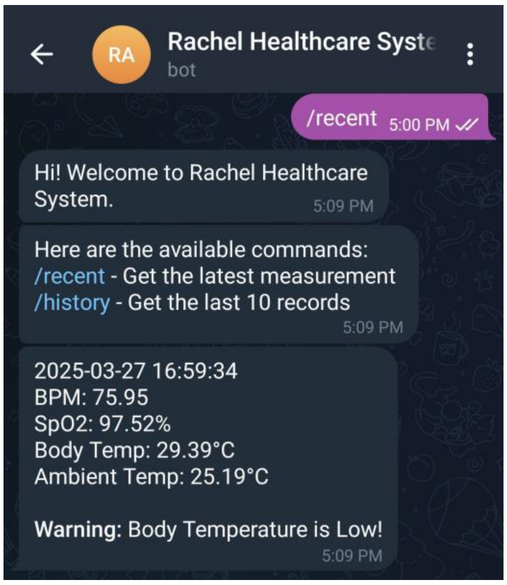
hi 9:44 AM

Unknown command.
Here are the available commands:
/recent - Get the latest measurement
/history - Get the last 10 records
9:44 AM

Figure 5.29 Mobile View of Telegram Message Sent by the System

The results are saved in the database and can be accessed through a secured login portal, ensuring data privacy while providing extended access to comprehensive health history.

In addition to automatic alerts, users or healthcare providers can manually interact with the Telegram bot using the */recent* command to retrieve the latest measurement or */history* to request a short list of past results. These commands make it convenient for healthcare professionals to check vital signs remotely without accessing the full system interface.



/recent 5:00 PM

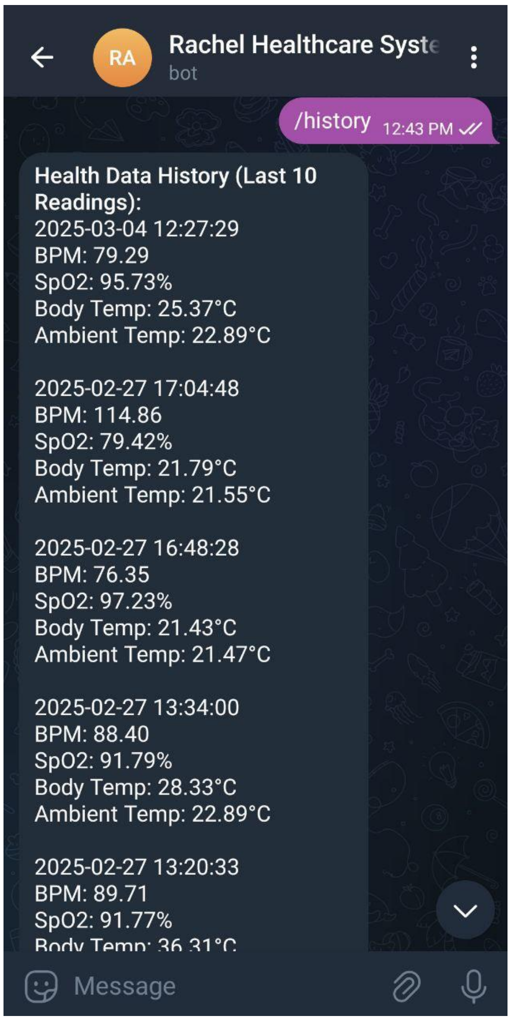
Hi! Welcome to Rachel Healthcare System. 5:09 PM

Here are the available commands:
/recent - Get the latest measurement
/history - Get the last 10 records
5:09 PM

2025-03-27 16:59:34
BPM: 75.95
SpO2: 97.52%
Body Temp: 29.39°C
Ambient Temp: 25.19°C

Warning: Body Temperature is Low!
5:09 PM

Figure 5.30 '/recent' Command in Telegram



/history 12:43 PM

Health Data History (Last 10 Readings):
2025-03-04 12:27:29
BPM: 79.29
SpO2: 95.73%
Body Temp: 25.37°C
Ambient Temp: 22.89°C

2025-02-27 17:04:48
BPM: 114.86
SpO2: 79.42%
Body Temp: 21.79°C
Ambient Temp: 21.55°C

2025-02-27 16:48:28
BPM: 76.35
SpO2: 97.23%
Body Temp: 21.43°C
Ambient Temp: 21.47°C

2025-02-27 13:34:00
BPM: 88.40
SpO2: 91.79%
Body Temp: 28.33°C
Ambient Temp: 22.89°C

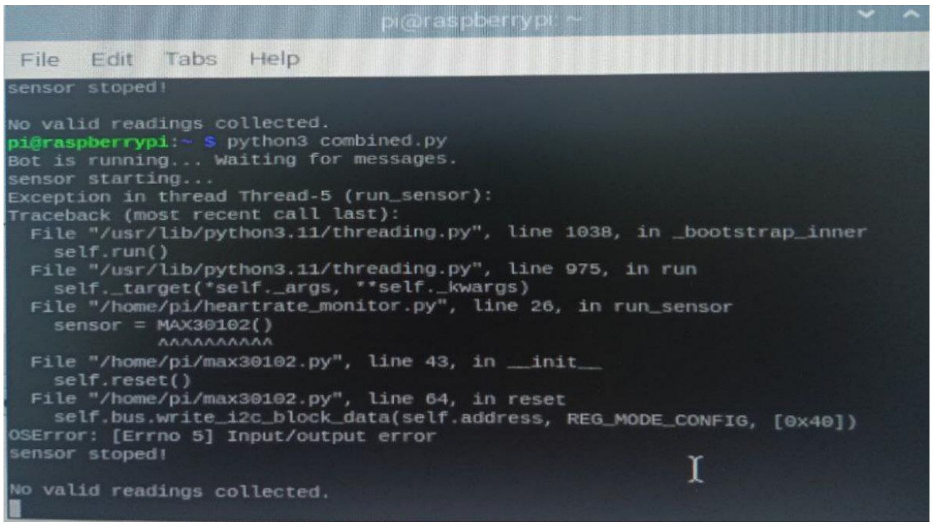
2025-02-27 13:20:33
BPM: 89.71
SpO2: 91.77%
Body Temp: 36.31°C

Figure 5.31 '/history' Command in Telegram

5.5 Implementation Issues and Challenges

The development of the health monitoring system encountered several implementation issues and practical challenges that required iterative debugging and hardware adjustments. One significant challenge involved the internet connection during the software setup phase. At several points, the installation of crucial Python libraries such as *matplotlib*, *telepot*, and *mysql.connector* failed repeatedly, leading to confusion about possible compatibility or package errors. After considerable troubleshooting, it was discovered that the root cause was an unstable Wi-Fi connection. Once the Raspberry Pi was rebooted and connected to a reliable Wi-Fi network, the required libraries were downloaded and installed successfully. This issue highlighted the importance of stable internet connectivity during software configuration, especially on Linux-based systems like Raspberry Pi OS.

Hardware configuration presented another set of difficulties, particularly in wiring both the MAX30102 and MLX90614 sensors to the Raspberry Pi via I2C. Since these sensors share the same communication bus, it was necessary to build a custom insulated Y-splitter that allowed both to operate concurrently. This involved manually cutting wires, connecting three insulated wires together, and wrapping them securely with insulation tape. However, this solution led to inconsistent sensor communication and triggered I/O errors:



```
sensor stoped!

No valid readings collected.
pi@raspberrypi:~ $ python3 combined.py
Bot is running... Waiting for messages.
sensor starting...
Exception in thread Thread-5 (run_sensor):
Traceback (most recent call last):
  File "/usr/lib/python3.11/threading.py", line 1038, in _bootstrap_inner
    self.run()
  File "/usr/lib/python3.11/threading.py", line 975, in run
    self._target(*self._args, **self._kwargs)
  File "/home/pi/heartrate_monitor.py", line 26, in run_sensor
    sensor = MAX30102()
             ^^^^^^^^^^
  File "/home/pi/max30102.py", line 43, in __init__
    self.reset()
  File "/home/pi/max30102.py", line 64, in reset
    self.bus.write_i2c_block_data(self.address, REG_MODE_CONFIG, [0x40])
OSError: [Errno 5] Input/output error
sensor stoped!

No valid readings collected.
```

Figure 5.32 Input/Output Error in Terminal

After repeated failures, the issue was traced back to faulty or loosely connected wires within the splitter. The wiring was disassembled and tested one by one to see which is the faulty wire. The damaged segment was replaced with a new workable Y-split cable, which restored stable data transmission.

Space constraints inside the 3D-printed case also made cable management difficult, but internal wire organization helped maintain cleanliness and functionality.

These challenges, though time-consuming and occasionally frustrating, led to a deeper understanding of both the hardware and software aspects of the system. Through persistence and testing, each issue was gradually resolved, resulting in a stable and reliable health monitoring platform.

5.6 Concluding Remark

The implementation phase of the health monitoring system marked a significant milestone in transforming the conceptual design into a fully functional and interactive solution. Through the integration of hardware components such as the Raspberry Pi, MAX30102, and MLX90614 sensors, along with the development of a comprehensive software system using Python, the system successfully achieved its intended functionality. A user-friendly graphical interface was created using Tkinter, real-time health data were processed and displayed, and a Telegram bot was integrated to support remote notifications and data retrieval.

Despite encountering challenges in both hardware configuration and software setup, each obstacle was addressed through iterative troubleshooting and refinement. The use of a custom-designed 3D-printed case helped organize internal components and improve user ergonomics, while the database integration and Telegram bot enabled both local and remote data access.

The implementation outcomes demonstrate the system's capability to support user-friendly, real-time health monitoring in a compact and affordable form.

Chapter 6
System Evaluation and Discussion

This chapter evaluates the developed health monitoring system, focusing on its functionality, performance, and the challenges encountered. The effectiveness of the system is measured through a series of test scenarios, real-time data readings, and data visualization. The chapter also discusses how well the system met the initial objectives and how user experience was considered in its design.

6.1 System Testing and Performance Metrics

To evaluate the functionality and reliability of the IoT-based healthcare monitoring system, Testing was conducted to evaluate its accuracy, responsiveness, and stability.

The primary components tested include the MAX30102 sensor for heart rate and SpO_2 monitoring, the MLX90614 sensor for body temperature measurement, the Raspberry Pi as the processing unit, the MariaDB database for data storage, the GUI built using Tkinter, and the Telegram notification system.

The performance metrics included:

Measurement Duration and Sampling: Each reading was captured over a 20-second interval, allowing the system to compute average values and reduce the impact of noise or motion artifacts.

Sensor Accuracy: Measurements were compared against reference devices such as a Xiaomi smartwatch (for BPM and SpO_2) and an infrared forehead thermometer (for body temperature).

Wi-Fi Dependency: A stable internet connection was essential for retrieving accurate timestamps and enabling real-time Telegram communication.

Data Logging Reliability: The system's ability to store and retrieve data without failure was assessed.

Response Time: The interval between measurement completion and Telegram message delivery was measured.

These metrics demonstrate the system's capability to offer timely and accurate health monitoring in a compact and affordable setup.

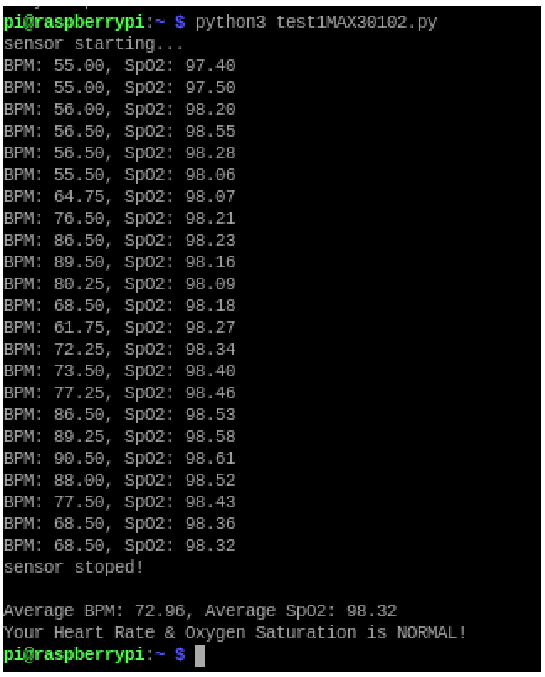
6.2 Testing Setup and Result

Before conducting hourly monitoring, the system's sensor accuracy was evaluated by comparing its readings to commercially available devices in a controlled indoor environment. This preliminary testing aimed to ensure the sensors were capable of providing consistent and reliable health measurements.

6.2.1 Sensor Accuracy Testing

1. Heart Rate (BPM) and Oxygen Saturation (SpO₂):

- Compared with: Xiaomi smartwatch
- Results:



```
pi@raspberrypi:~ $ python3 test1MAX30102.py
sensor starting...
BPM: 55.00, SpO2: 97.40
BPM: 55.00, SpO2: 97.50
BPM: 56.00, SpO2: 98.20
BPM: 56.50, SpO2: 98.55
BPM: 56.50, SpO2: 98.28
BPM: 55.50, SpO2: 98.06
BPM: 64.75, SpO2: 98.07
BPM: 76.50, SpO2: 98.21
BPM: 86.50, SpO2: 98.23
BPM: 89.50, SpO2: 98.16
BPM: 80.25, SpO2: 98.09
BPM: 68.50, SpO2: 98.18
BPM: 61.75, SpO2: 98.27
BPM: 72.25, SpO2: 98.34
BPM: 73.50, SpO2: 98.40
BPM: 77.25, SpO2: 98.46
BPM: 86.50, SpO2: 98.53
BPM: 89.25, SpO2: 98.58
BPM: 90.50, SpO2: 98.61
BPM: 88.00, SpO2: 98.52
BPM: 77.50, SpO2: 98.43
BPM: 68.50, SpO2: 98.36
BPM: 68.50, SpO2: 98.32
sensor stoped!

Average BPM: 72.96, Average SpO2: 98.32
Your Heart Rate & Oxygen Saturation is NORMAL!
pi@raspberrypi:~ $
```

Figure 6.1 MAX30102 Sensor Reading for Both BPM and SpO2

- BPM readings of the MAX30102 sensor showed slight fluctuations but remained within a reasonable physiological range. Minor discrepancies were attributed to hand motion or variability in heart rhythm.

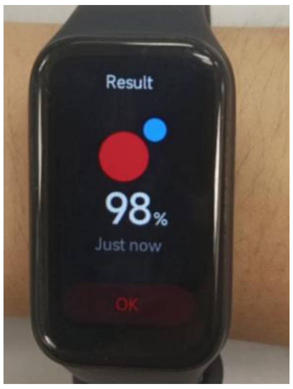


Figure 6.2 Smartwatch's SpO2 reading

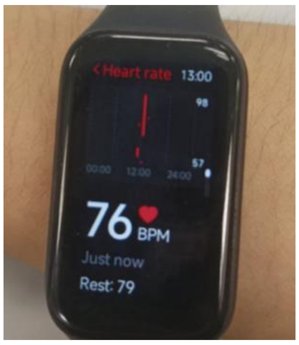
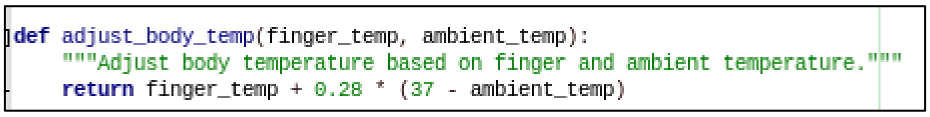


Figure 6.3 Smartwatch's BPM reading

- SpO_2 readings were stable, while BPM showed mild variations but stayed within a normal physiological range.

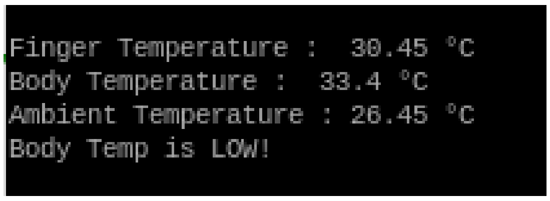
2. Body Temperature:

- Compared with: Infrared thermometer
- Results:
  - The system that measures temperature from the user's finger recorded lower values than the forehead thermometer due to ambient temperature influence on peripheral skin (fingers), especially in an air-conditioned environment.
  - To improve accuracy, a formula was applied in the software to adjust the finger-based temperature readings closer to core body temperature estimates.



```
def adjust_body_temp(finger_temp, ambient_temp):
    """Adjust body temperature based on finger and ambient temperature."""
    return finger_temp + 0.28 * (37 - ambient_temp)
```

Figure 6.4 Formula to Adjust the Body Temperature

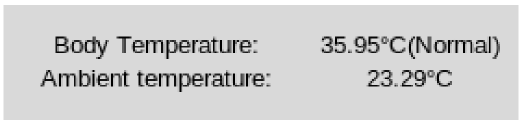


```
Finger Temperature :  30.45 °C
Body Temperature :  33.4 °C
Ambient Temperature : 26.45 °C
Body Temp is LOW!
```

Figure 6.5 MLX90614 Sensor Reading for Temperatures

Finger temperature in Figure 6.5 is the temperature without using a formula. Body temperature Figure 6.5 is the adjusted temperature by using the predefined formula.

It is still too low even after using the formula when staying in an air-conditioned room for a period of time.



```
Body Temperature:      35.95°C(Normal)
Ambient temperature:      23.29°C
```

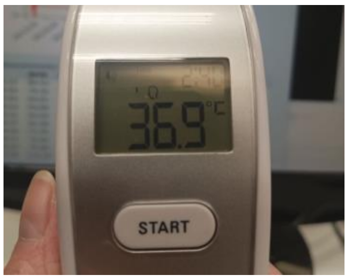


Figure 6.6 Comparison between MLX90614 Sensor and Infrared Thermometer

- While the values remained slightly lower than forehead thermometer results, the adjustment provided a more realistic baseline for health tracking. The readings followed a consistent trend, suitable for non-critical health monitoring.

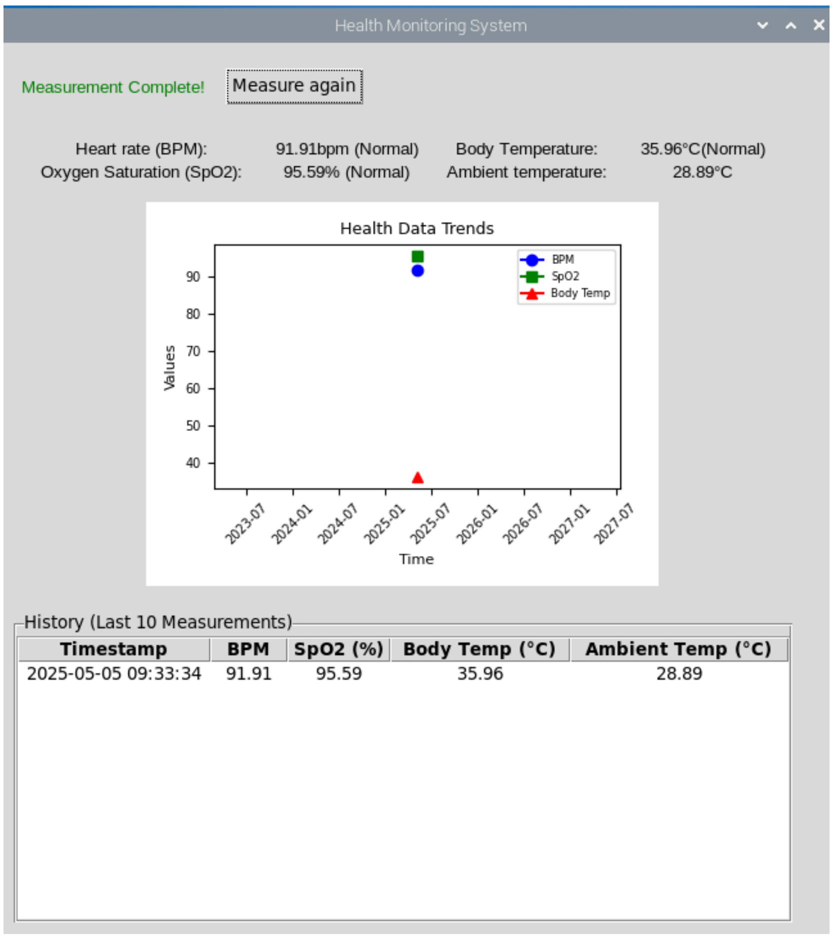
These tests validated that the system's sensors could produce reasonable and consistent outputs for non-critical, home-based health monitoring. With calibration in place, the system proceeded to perform scheduled hourly measurements from 9:30 AM to 5:30 PM.

6.2.2 Hourly Monitoring Results

Following the initial sensor accuracy testing, the system was used to perform hourly health monitoring throughout a typical day, from 9:30 AM to 5:30 PM. This process is repeated hourly, and the user is required to place their finger on the sensors at the start of each new hour to generate new readings. At the end of the day, the data collected from multiple intervals will allow healthcare providers or the user to analyze their health status trends throughout the day.

The system was evaluated in a typical indoor environment, primarily in an air-conditioned room with consistent lighting and stable power supply. A stable Wi-Fi connection was essential to retrieve accurate timestamps and enable real-time communication through the Telegram bot. The testing covered an entire working day, with health measurements taken hourly from 9:30 AM to 5:30 PM. Each measurement included heart rate (BPM), oxygen saturation (SpO_2), and body temperature.

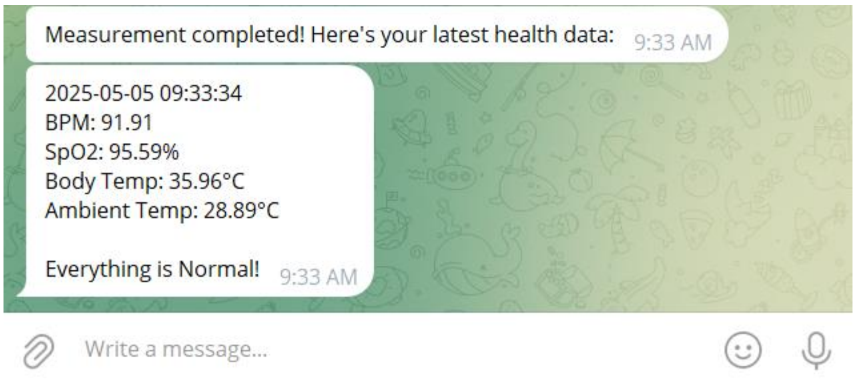
Below are the hourly monitoring results of the GUI and Telegram alerts:



Heart rate (BPM): 91.91bpm (Normal) Body Temperature: 35.96°C(Normal)
Oxygen Saturation (SpO2): 95.59% (Normal) Ambient temperature: 28.89°C

Timestamp	BPM	SpO2 (%)	Body Temp (°C)	Ambient Temp (°C)
2025-05-05 09:33:34	91.91	95.59	35.96	28.89

Figure 6.7 GUI for the First Measure at 9.33 am

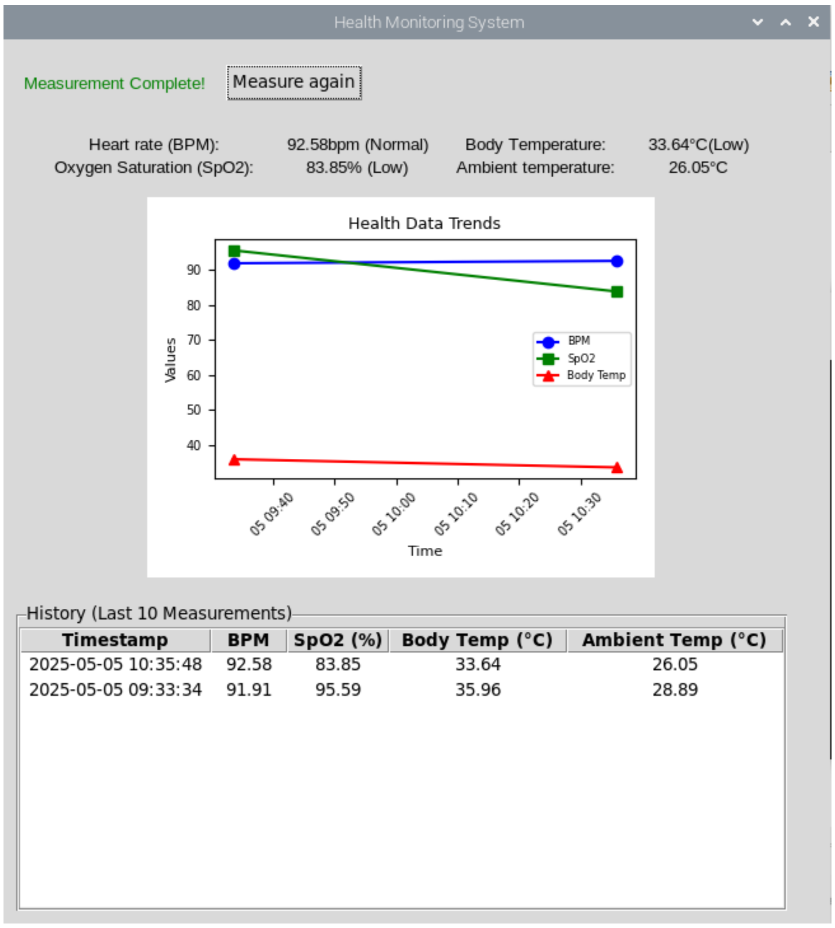


Measurement completed! Here's your latest health data: 9:33 AM

2025-05-05 09:33:34
BPM: 91.91
SpO2: 95.59%
Body Temp: 35.96°C
Ambient Temp: 28.89°C

Everything is Normal! 9:33 AM

Figure 6.8 Telegram Message for the First Measure



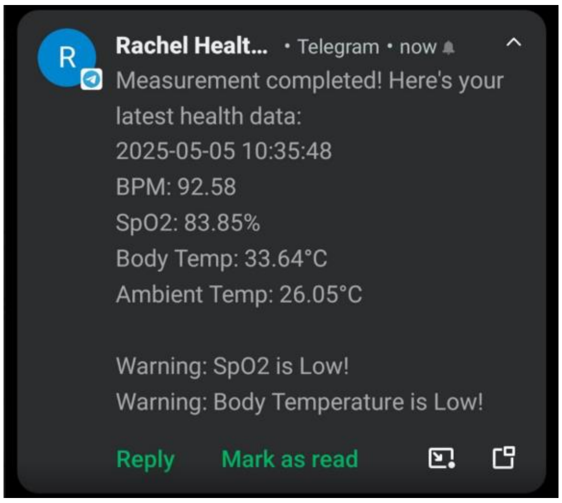
Measurement Complete!

Heart rate (BPM):	92.58bpm (Normal)	Body Temperature:	33.64°C(Low)
Oxygen Saturation (SpO2):	83.85% (Low)	Ambient temperature:	26.05°C

History (Last 10 Measurements)

Timestamp	BPM	SpO2 (%)	Body Temp (°C)	Ambient Temp (°C)
2025-05-05 10:35:48	92.58	83.85	33.64	26.05
2025-05-05 09:33:34	91.91	95.59	35.96	28.89

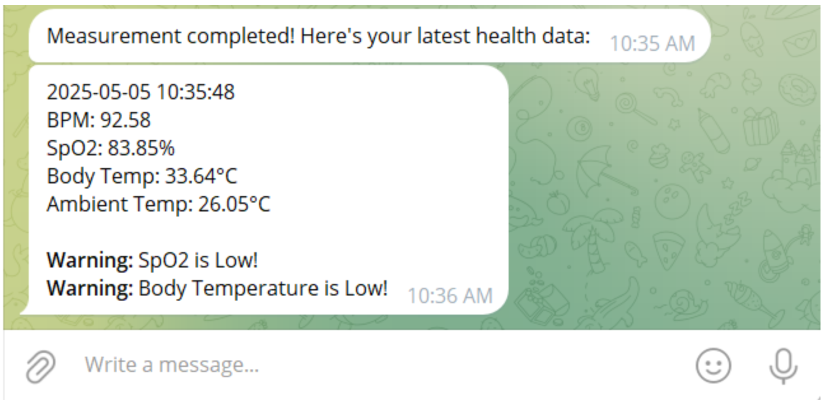
Figure 6.9 GUI for the Second Measure at 10.35 am



Rachel Healt... • Telegram • now
Measurement completed! Here's your latest health data:
2025-05-05 10:35:48
BPM: 92.58
SpO2: 83.85%
Body Temp: 33.64°C
Ambient Temp: 26.05°C

Warning: SpO2 is Low!
Warning: Body Temperature is Low!

Figure 6.10 Notification Shown on Mobile

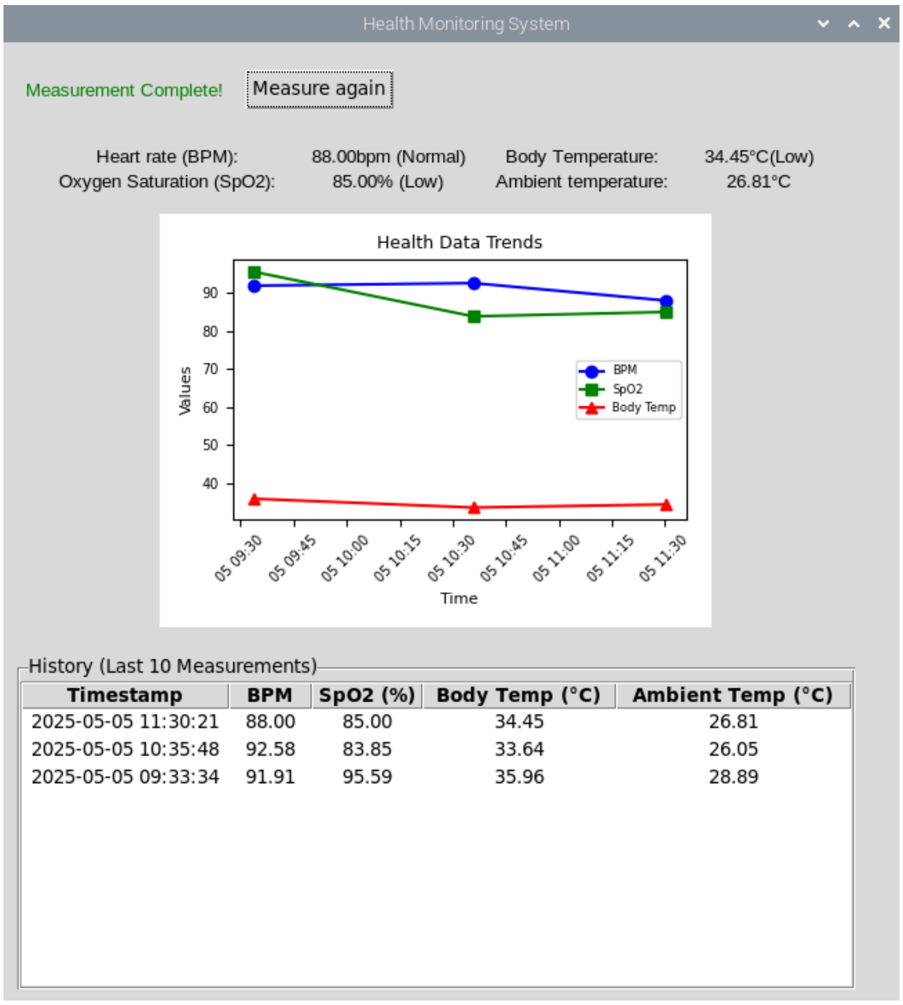


Measurement completed! Here's your latest health data: 10:35 AM

2025-05-05 10:35:48
BPM: 92.58
SpO2: 83.85%
Body Temp: 33.64°C
Ambient Temp: 26.05°C

Warning: SpO2 is Low!
Warning: Body Temperature is Low! 10:36 AM

Figure 6.11 Telegram Message for the Second Measure

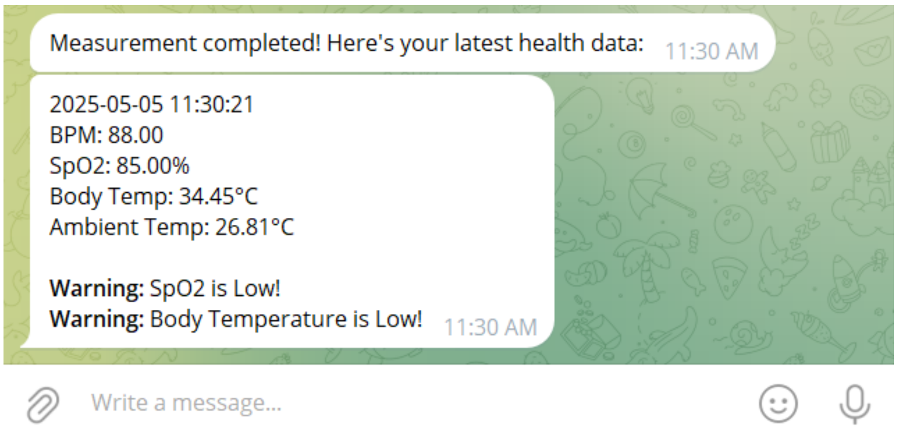


Heart rate (BPM): 88.00bpm (Normal) Body Temperature: 34.45°C(Low)
Oxygen Saturation (SpO2): 85.00% (Low) Ambient temperature: 26.81°C

History (Last 10 Measurements)

Timestamp	BPM	SpO2 (%)	Body Temp (°C)	Ambient Temp (°C)
2025-05-05 11:30:21	88.00	85.00	34.45	26.81
2025-05-05 10:35:48	92.58	83.85	33.64	26.05
2025-05-05 09:33:34	91.91	95.59	35.96	28.89

Figure 6.12 GUI for the Third Measure at 11.30 am

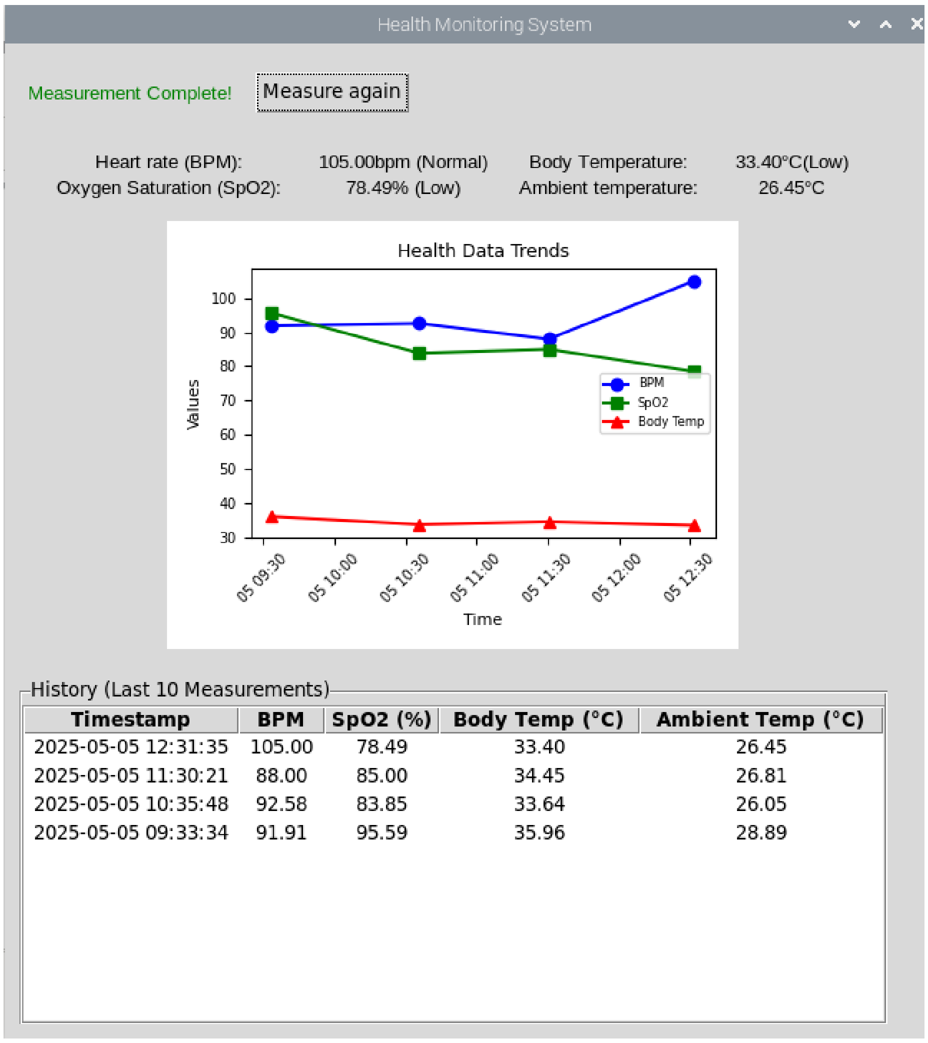


Measurement completed! Here's your latest health data: 11:30 AM

2025-05-05 11:30:21
BPM: 88.00
SpO2: 85.00%
Body Temp: 34.45°C
Ambient Temp: 26.81°C

Warning: SpO2 is Low!
Warning: Body Temperature is Low! 11:30 AM

Figure 6.13 Telegram Message for the Third Measure

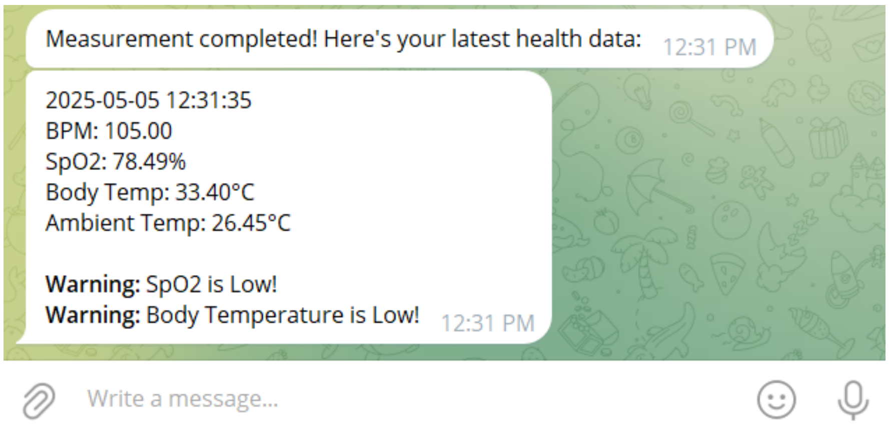


Heart rate (BPM):	105.00bpm (Normal)	Body Temperature:	33.40°C(Low)
Oxygen Saturation (SpO2):	78.49% (Low)	Ambient temperature:	26.45°C

History (Last 10 Measurements)

Timestamp	BPM	SpO2 (%)	Body Temp (°C)	Ambient Temp (°C)
2025-05-05 12:31:35	105.00	78.49	33.40	26.45
2025-05-05 11:30:21	88.00	85.00	34.45	26.81
2025-05-05 10:35:48	92.58	83.85	33.64	26.05
2025-05-05 09:33:34	91.91	95.59	35.96	28.89

Figure 6.14 GUI for the Fourth Measure at 12.31 pm

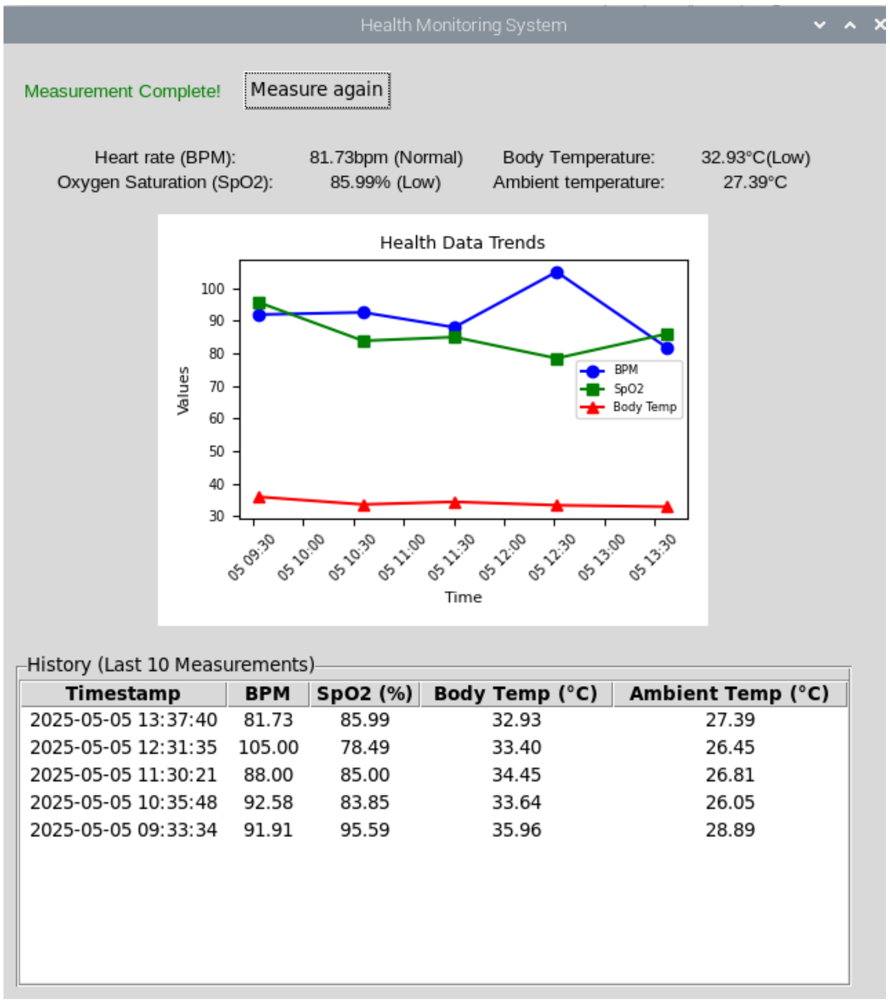


Measurement completed! Here's your latest health data: 12:31 PM

2025-05-05 12:31:35
BPM: 105.00
SpO2: 78.49%
Body Temp: 33.40°C
Ambient Temp: 26.45°C

Warning: SpO2 is Low!
Warning: Body Temperature is Low! 12:31 PM

Figure 6.15 Telegram Message for the Fourth Measure



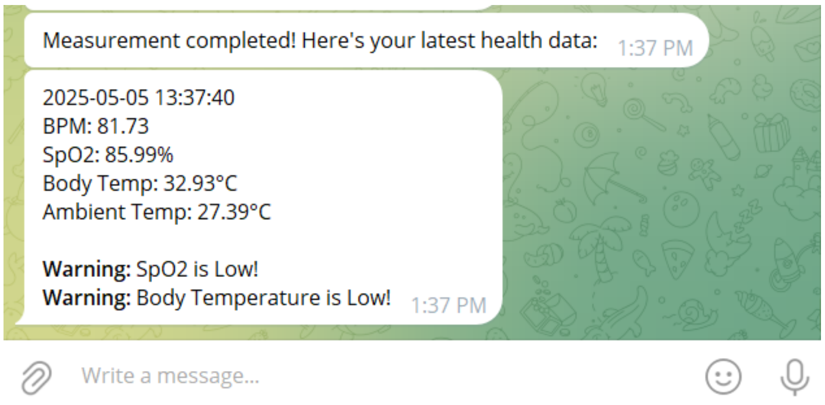
Measurement Complete!

Heart rate (BPM):	81.73bpm (Normal)	Body Temperature:	32.93°C(Low)
Oxygen Saturation (SpO2):	85.99% (Low)	Ambient temperature:	27.39°C

History (Last 10 Measurements)

Timestamp	BPM	SpO2 (%)	Body Temp (°C)	Ambient Temp (°C)
2025-05-05 13:37:40	81.73	85.99	32.93	27.39
2025-05-05 12:31:35	105.00	78.49	33.40	26.45
2025-05-05 11:30:21	88.00	85.00	34.45	26.81
2025-05-05 10:35:48	92.58	83.85	33.64	26.05
2025-05-05 09:33:34	91.91	95.59	35.96	28.89

Figure 6.16 GUI for the Fifth Measure at 1.37 pm

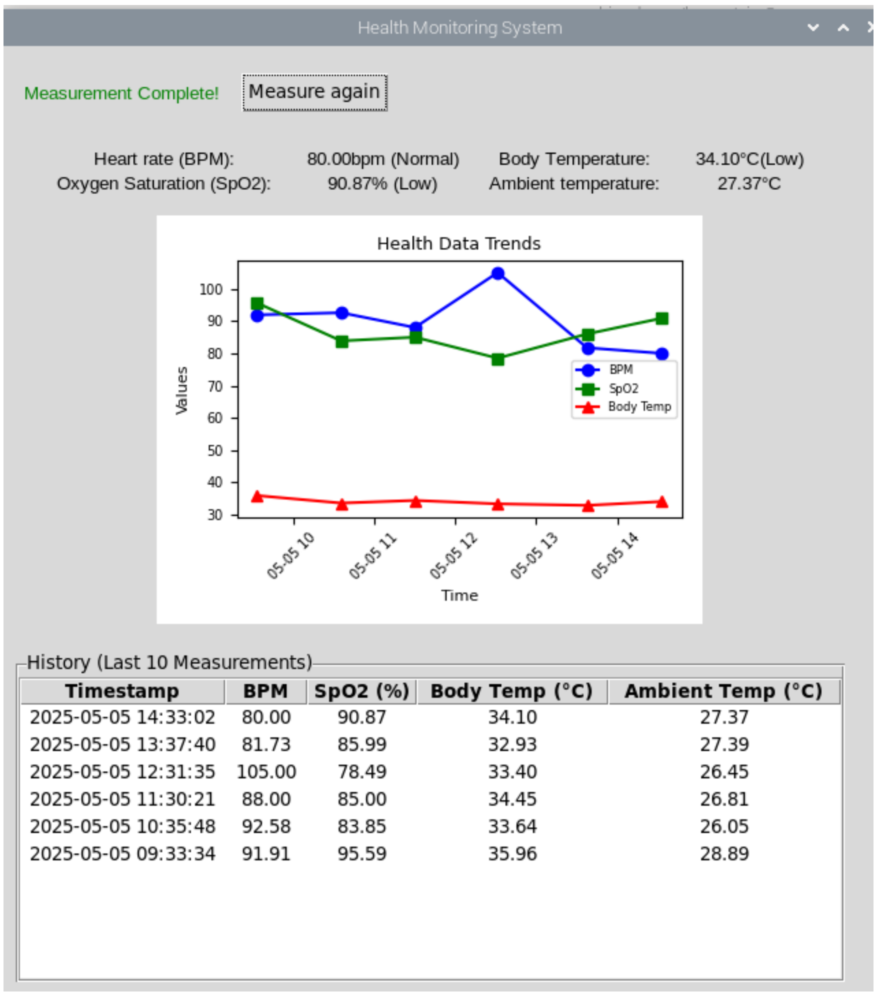


Measurement completed! Here's your latest health data: 1:37 PM

2025-05-05 13:37:40
BPM: 81.73
SpO2: 85.99%
Body Temp: 32.93°C
Ambient Temp: 27.39°C

Warning: SpO2 is Low!
Warning: Body Temperature is Low! 1:37 PM

Figure 6.17 Telegram Message for the Fifth Measure



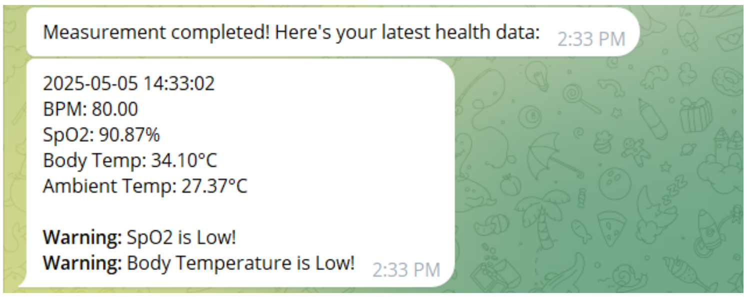
Measurement Complete!

Heart rate (BPM):	80.00bpm (Normal)	Body Temperature:	34.10°C(Low)
Oxygen Saturation (SpO2):	90.87% (Low)	Ambient temperature:	27.37°C

History (Last 10 Measurements)

Timestamp	BPM	SpO2 (%)	Body Temp (°C)	Ambient Temp (°C)
2025-05-05 14:33:02	80.00	90.87	34.10	27.37
2025-05-05 13:37:40	81.73	85.99	32.93	27.39
2025-05-05 12:31:35	105.00	78.49	33.40	26.45
2025-05-05 11:30:21	88.00	85.00	34.45	26.81
2025-05-05 10:35:48	92.58	83.85	33.64	26.05
2025-05-05 09:33:34	91.91	95.59	35.96	28.89

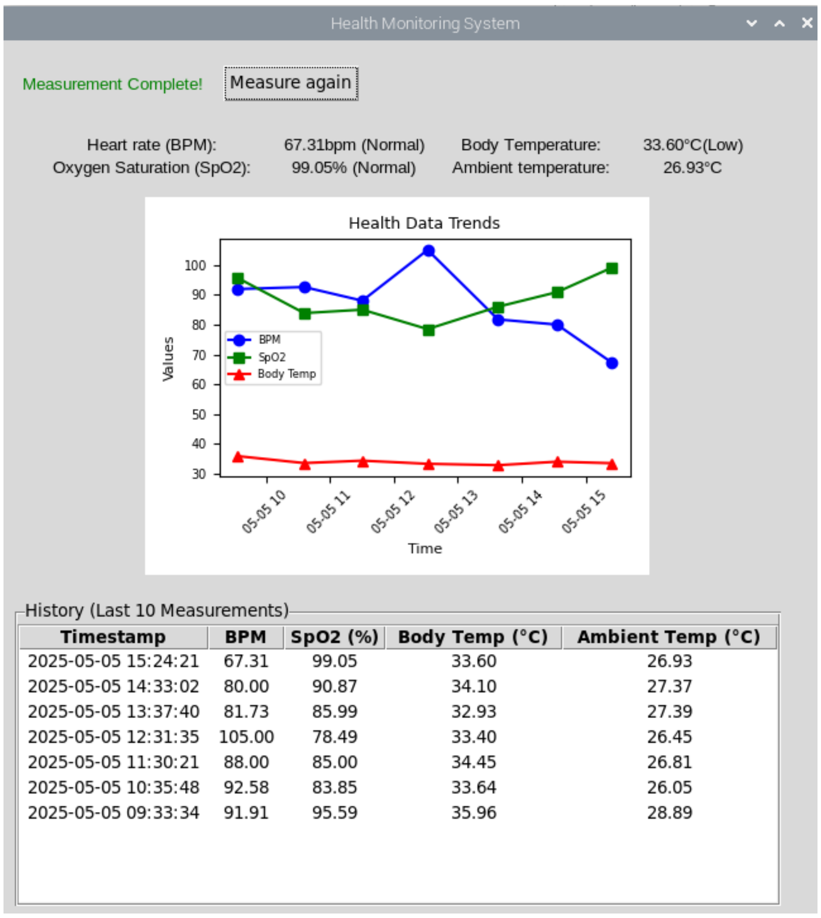
Figure 6.18 GUI for the Sixth Measure at 2.33 pm


Measurement completed! Here's your latest health data: 2:33 PM

2025-05-05 14:33:02
BPM: 80.00
SpO2: 90.87%
Body Temp: 34.10°C
Ambient Temp: 27.37°C

Warning: SpO2 is Low!
Warning: Body Temperature is Low! 2:33 PM

Figure 6.19 Telegram Message for the Sixth Measure

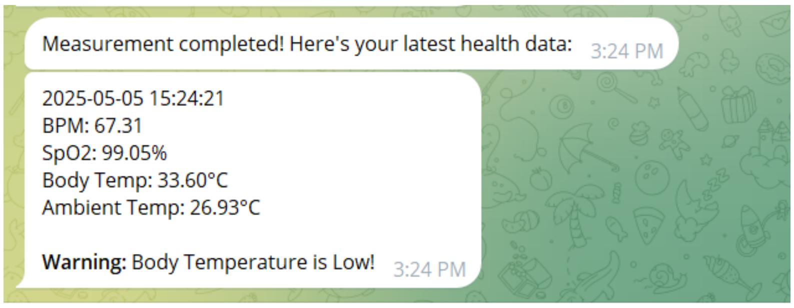


Heart rate (BPM): 67.31bpm (Normal) Body Temperature: 33.60°C(Low)
Oxygen Saturation (SpO2): 99.05% (Normal) Ambient temperature: 26.93°C

History (Last 10 Measurements)

Timestamp	BPM	SpO2 (%)	Body Temp (°C)	Ambient Temp (°C)
2025-05-05 15:24:21	67.31	99.05	33.60	26.93
2025-05-05 14:33:02	80.00	90.87	34.10	27.37
2025-05-05 13:37:40	81.73	85.99	32.93	27.39
2025-05-05 12:31:35	105.00	78.49	33.40	26.45
2025-05-05 11:30:21	88.00	85.00	34.45	26.81
2025-05-05 10:35:48	92.58	83.85	33.64	26.05
2025-05-05 09:33:34	91.91	95.59	35.96	28.89

Figure 6.20 GUI for the Seventh Measure at 3.24 pm

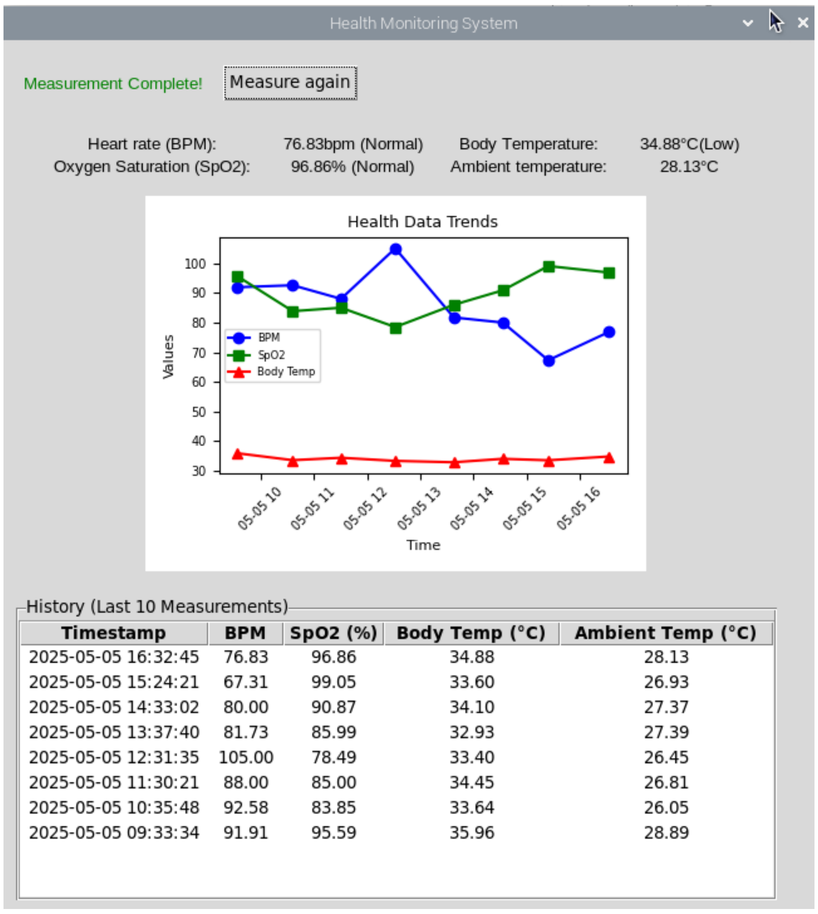


Measurement completed! Here's your latest health data: 3:24 PM

2025-05-05 15:24:21
BPM: 67.31
SpO2: 99.05%
Body Temp: 33.60°C
Ambient Temp: 26.93°C

Warning: Body Temperature is Low! 3:24 PM

Figure 6.21 Telegram Message for the Seventh Measure



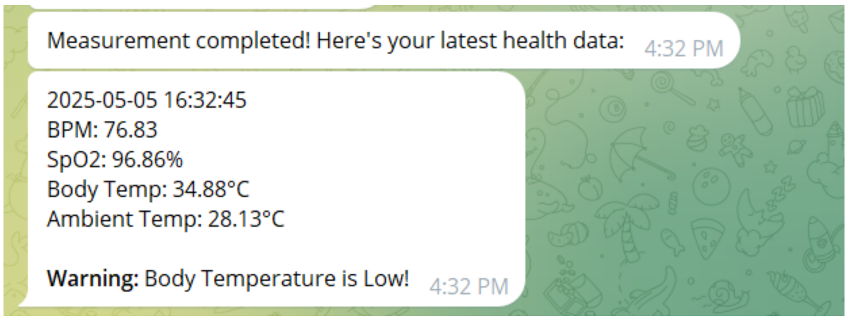
Measurement Complete!

Heart rate (BPM):	76.83bpm (Normal)	Body Temperature:	34.88°C(Low)
Oxygen Saturation (SpO2):	96.86% (Normal)	Ambient temperature:	28.13°C

History (Last 10 Measurements)

Timestamp	BPM	SpO2 (%)	Body Temp (°C)	Ambient Temp (°C)
2025-05-05 16:32:45	76.83	96.86	34.88	28.13
2025-05-05 15:24:21	67.31	99.05	33.60	26.93
2025-05-05 14:33:02	80.00	90.87	34.10	27.37
2025-05-05 13:37:40	81.73	85.99	32.93	27.39
2025-05-05 12:31:35	105.00	78.49	33.40	26.45
2025-05-05 11:30:21	88.00	85.00	34.45	26.81
2025-05-05 10:35:48	92.58	83.85	33.64	26.05
2025-05-05 09:33:34	91.91	95.59	35.96	28.89

Figure 6.22 GUI for the Eighth Measure at 4.32 pm

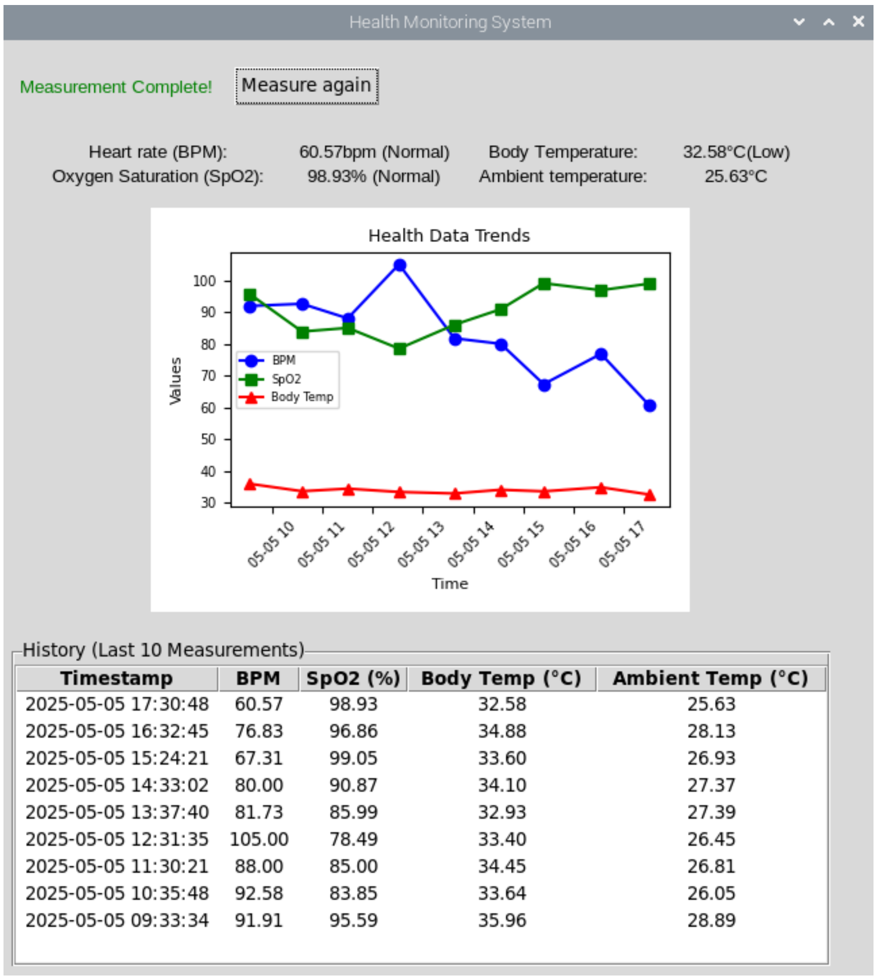


Measurement completed! Here's your latest health data: 4:32 PM

2025-05-05 16:32:45
BPM: 76.83
SpO2: 96.86%
Body Temp: 34.88°C
Ambient Temp: 28.13°C

Warning: Body Temperature is Low! 4:32 PM

Figure 6.23 Telegram Message for the Eighth Measure



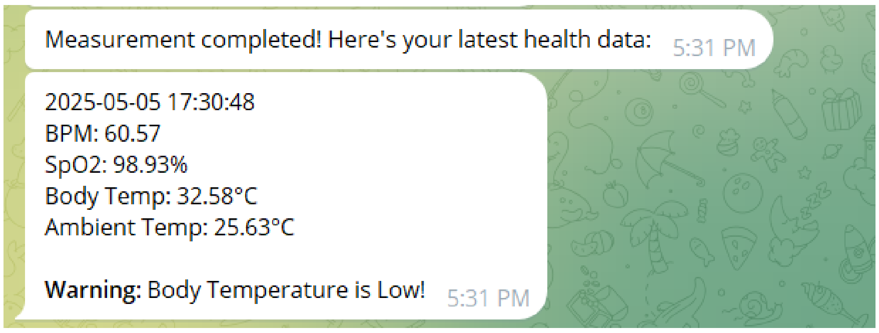
Measurement Complete!

Heart rate (BPM): 60.57bpm (Normal) Body Temperature: 32.58°C(Low)
Oxygen Saturation (SpO2): 98.93% (Normal) Ambient temperature: 25.63°C

History (Last 10 Measurements)

Timestamp	BPM	SpO2 (%)	Body Temp (°C)	Ambient Temp (°C)
2025-05-05 17:30:48	60.57	98.93	32.58	25.63
2025-05-05 16:32:45	76.83	96.86	34.88	28.13
2025-05-05 15:24:21	67.31	99.05	33.60	26.93
2025-05-05 14:33:02	80.00	90.87	34.10	27.37
2025-05-05 13:37:40	81.73	85.99	32.93	27.39
2025-05-05 12:31:35	105.00	78.49	33.40	26.45
2025-05-05 11:30:21	88.00	85.00	34.45	26.81
2025-05-05 10:35:48	92.58	83.85	33.64	26.05
2025-05-05 09:33:34	91.91	95.59	35.96	28.89

Figure 6.24 GUI for the Ninth Measure at 5.30 pm

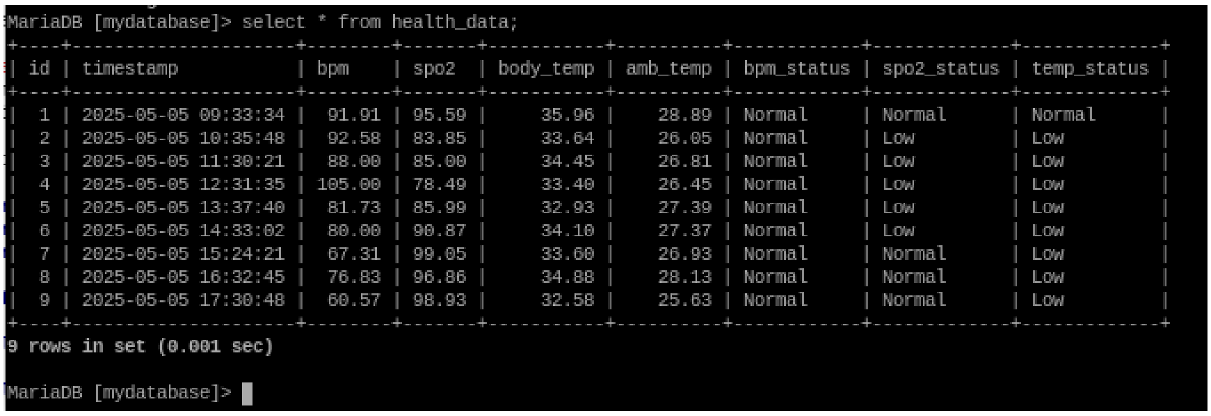


Measurement completed! Here's your latest health data: 5:31 PM

2025-05-05 17:30:48
BPM: 60.57
SpO2: 98.93%
Body Temp: 32.58°C
Ambient Temp: 25.63°C

Warning: Body Temperature is Low! 5:31 PM

Figure 6.25 Telegram Message for the Ninth Measure



```
MariaDB [mydatabase]> select * from health_data;
+----+---------------------+--------+-------+-----------+----------+------------+-------------+-------------+
| id | timestamp           | bpm    | spo2  | body_temp | amb_temp | bpm_status | spo2_status | temp_status |
+----+---------------------+--------+-------+-----------+----------+------------+-------------+-------------+
|  1 | 2025-05-05 09:33:34 |  91.91 | 95.59 |     35.96 |    28.89 | Normal     | Normal      | Normal      |
|  2 | 2025-05-05 10:35:48 |  92.58 | 83.85 |     33.64 |    26.05 | Normal     | Low         | Low         |
|  3 | 2025-05-05 11:30:21 |  88.00 | 85.00 |     34.45 |    26.81 | Normal     | Low         | Low         |
|  4 | 2025-05-05 12:31:35 | 105.00 | 78.49 |     33.40 |    26.45 | Normal     | Low         | Low         |
|  5 | 2025-05-05 13:37:40 |  81.73 | 85.99 |     32.93 |    27.39 | Normal     | Low         | Low         |
|  6 | 2025-05-05 14:33:02 |  80.00 | 90.87 |     34.10 |    27.37 | Normal     | Low         | Low         |
|  7 | 2025-05-05 15:24:21 |  67.31 | 99.05 |     33.60 |    26.93 | Normal     | Normal      | Low         |
|  8 | 2025-05-05 16:32:45 |  76.83 | 96.86 |     34.88 |    28.13 | Normal     | Normal      | Low         |
|  9 | 2025-05-05 17:30:48 |  60.57 | 98.93 |     32.58 |    25.63 | Normal     | Normal      | Low         |
+----+---------------------+--------+-------+-----------+----------+------------+-------------+-------------+
9 rows in set (0.001 sec)

MariaDB [mydatabase]> 
```

Figure 6.26 Database Result of Hourly Data in A Day

As a summary of the result analysis, the system collected nine sets of health data from 9.30 AM to 5.30 PM, with each hour's readings stored in the database. The heart rate (BPM) ranged from approximately 60.57 to 105 bpm, with variations attributed to different activity levels or environmental conditions between measurement times. Oxygen saturation (SpO2) readings were generally stable, ranging from 78.49% to 99.05%, indicating the sensor's consistency. Lower readings occasionally occurred but remained within acceptable non-critical limits. The body temperature values ranged between 32.58°C and 35.96°C. These lower-than-expected readings were primarily due to measurement through the finger, especially in a cooled indoor setting. However, the application of a correction formula helped moderate this variance.

The system demonstrated strong data logging reliability, as all measurements were consistently saved to the MariaDB database without interruption. Telegram alert timing was also effective, with health data being successfully sent to the user's Telegram within 1–2 seconds after each measurement, reflecting the real-time capability of the system. Furthermore, environmental impact was evident in the body temperature data, with slight variability caused by finger-based measurements in an air-conditioned environment. Overall, the collected data illustrates both the sensor's sensitivity and accuracy, while also highlighting how external conditions can influence certain physiological readings.

6.3 Project Challenges

Several challenges were encountered during the development and testing of the system, especially when combining different parts like sensors, the graphical interface, and Telegram messaging.

One challenge was getting accurate readings from the sensors. For example, the body temperature sensor gave lower readings when used in an air-conditioned room, especially since the measurement was taken from the finger. To improve this, a correction formula was used, and testing was repeated several times to check the results against a standard infrared thermometer.

The user interaction part also brought some problems. For the Telegram bot, it took time to make sure messages were sent correctly after each reading. This involved fixing token-related issues, dealing with internet connection problems, and learning how to get the correct chat ID. On the GUI side, designing a Tkinter window that could update in real time without freezing was not easy. It had to be tested many times to make sure it stayed responsive and easy to use while showing live data and saving it to the database.

There were also hardware-related difficulties. The system was built into a small 3D-printed case, so all the wiring had to be arranged carefully to fit and stay neat. Jumper wires had to be adjusted and sometimes replaced to avoid loose connections. Another issue came from the micro HDMI to VGA adapter, which the monitor sometimes cannot display the Raspberry Pi interface because the connection was not firm. This needed manual checking and reconnecting the adapter when the screen didn’t turn on.

These challenges made the project more difficult, but they also helped improve the final system. Each problem was solved step by step, leading to better design, stronger connections, and more reliable results.

6.4 Objectives Evaluation

MariaDB and Tkinter are integrated for real-time display, history table, and graphing The system effectively met all its stated objectives, demonstrating its potential as a practical and affordable health monitoring solution. The system was developed using affordable components such as Raspberry Pi and low-cost sensors, meeting the goal of cost-effectiveness. Remote health monitoring was enabled through Telegram, allowing users or caregivers to access vital sign data in real time. Real-time analysis of heart rate, oxygen saturation level, and body temperature was implemented, supporting timely health decisions and potentially reducing unnecessary hospital visits. Lastly, the integration of a MariaDB database with a Python Tkinter GUI provided efficient data management and a responsive, user-friendly interface. The interface not only displayed real-time measurements but also maintained a history of recent data entries and provided graphical visualization to help users track trends over time. This ensured both accessibility and ease of understanding for users managing their health. Overall, the system fulfilled its intended purpose as a reliable and accessible health monitoring solution.

6.5 Concluding Remark

The evaluation of the system demonstrated that all the major components functioned as intended, with reliable performance across multiple test sessions. From data acquisition to real-time display and remote communication, each feature operates together to achieve the project's primary goals. The system showed that remote health monitoring using affordable hardware can be implemented effectively, offering users a convenient way to track their vital signs in everyday settings. This validation sets a strong foundation for future improvements and potential deployment in real-world scenarios, particularly for individuals who require regular monitoring without needing to visit a healthcare facility.

Chapter 7
Conclusion and Recommendation

7.1 Conclusion

This project aimed to develop a low-cost, real-time health monitoring system using readily available hardware and open-source software. The IoT-based healthcare monitoring and alert system presented in this project demonstrates the potential for leveraging technology to improve patient autonomy and healthcare efficiency. By integrating sensors such as the MAX30102 and MLX90614, the system continuously tracks vital health parameters like heart rate, oxygen saturation, and body temperature, providing real-time monitoring and data storage in a MariaDB database. The inclusion of a Tkinter-based GUI for user interaction and an alert system via Telegram bot was developed to notify users immediately after each measurement, extending the system's usability beyond physical presence.

Throughout the testing phase, the system consistently demonstrated its ability to record vital signs accurately and send updates within 1–2 seconds, ensuring that users remain informed of their health status in near real time. By logging hourly readings from morning to evening, the system helped simulate a daily health routine, supporting users in monitoring trends and detecting any irregularities.

In conclusion, this project demonstrates how embedded systems and IoT technologies can be applied to improve personal healthcare, especially in times when remote monitoring is crucial. The success of this implementation reflects the growing potential of personalized, technology-driven solutions in healthcare.

7.2 Recommendation

Although the health monitoring system achieved its primary objectives, several areas can be improved to enhance its practicality and long-term usability. Firstly, the sensor placement can be optimized by designing a proper wearable form, such as a clip-on or wrist-mounted enclosure, which would ensure consistent contact and improve reading accuracy.

The current setup involves manual wiring and the use of Y-splitters to share the I²C bus, but it has introduced risks of disconnection and I/O errors. Future versions should consider using a custom PCB or integrated sensor module to reduce wiring complexity and potential errors, improving the durability and compactness of the system.

Additionally, extending the interface to mobile or web platforms would make the system more accessible to a wider range of users, especially for healthcare providers and family members who want to monitor the user's health remotely.

Integrating a rechargeable battery would also allow for portable use, removing dependency on a fixed power source and making the system more practical for continuous health tracking.

From a software perspective, adding intelligent analytics such as health trend detection, abnormality warnings, or generating daily reports could support early intervention and long-term health planning.

Lastly, to ensure data privacy and integrity, especially if expanded to cloud storage or mobile apps, future development should include secure data transmission and user authentication features.

With these improvements, the system can evolve from a functional prototype to a more user-friendly and practical tool for daily health management. It has the potential to serve not only individuals with chronic conditions but also general users seeking to maintain their well-being through regular monitoring.

REFERENCES

[1] N. B. Kamarozaman and A. H. Awang, "IOT COVID-19 Portable Health Monitoring System using Raspberry Pi, Node-Red and ThingSpeak," IEEE Xplore, Aug. 01, 2021. https://ieeexplore.ieee.org/document/9587444

[2] S. Abdulmalek et al., "IoT-Based Healthcare-Monitoring System towards Improving Quality of Life: A Review," Healthcare, vol. 10, no. 10, p. 1993, 2022, doi: https://doi.org/10.3390/healthcare10101993

[3] Sharrath M, Saran V, Pradeep Kp and Rudrasamy K, "PATIENT HEALTH MONITORING SYSTEM USING IOT," International Journal of Creative Research Thoughts, vol. 10, no. 6, pp. 2320–2882, 2022, Accessed: Apr. 11, 2024. [Online]. Available: https://ijcrt.org/papers/IJCRT22A6788.pdf

[4] Dr. E. N. Ganesh, "Health Monitoring System using Raspberry Pi and IOT," Oriental Journal of Computer Science and Technology, vol. 12, no. 1, pp. 08-13, Apr. 2019, Available: https://www.computerscijournal.org/vol12no1/health-monitoring-system-using-raspberry-piand-iot/

[5] N. Lakkundi, "Health monitoring using Raspberry pi," www.skyfilabs.com, 2022 https://www.skyfilabs.com/project-ideas/health-monitoring-using-raspberry-pi

[6] J. Heaney, J. Buick, M. U. Hadi, and N. Soin, "Internet of Things-Based ECG and Vitals Healthcare Monitoring System," Micromachines, vol. 13, no. 12, p. 2153, Dec. 2022, doi: https://doi.org/10.3390/mi13122153

[7] S. Aziz, S. Ahmed, and M.-S. Alouini, "ECG-based machine-learning algorithms for heartbeat classification," Scientific Reports, vol. 11, no. 1, Sep. 2021, doi: https://doi.org/10.1038/s41598-021-97118-5

[8] M. Stoltzfus, A. Kaur, A. Chawla, V. Gupta, F. N. U. Anamika, and R. Jain, "The role of telemedicine in healthcare: An overview and update," The Egyptian Journal of Internal Medicine, vol. 35, no. 1, Jun. 2023, doi: https://doi.org/10.1186/s43162-023-00234-z

[9] Muhammad Waleed, T. Kamal, T.-W. Um, A. Hafeez, B. Habib, and Knud Erik Skouby, "Unlocking Insights in IoT-Based Patient Monitoring: Methods for Encompassing Large-Data Challenges," Sensors, vol. 23, no. 15, pp. 6760–6760, Jul. 2023, doi: https://doi.org/10.3390/s23156760

[10] S. Iranpak, A. Shahbahrami, and H. Shakeri, "Remote patient monitoring and classifying using the internet of things platform combined with cloud computing," Journal of Big Data, vol. 8, no. 1, Sep. 2021, doi: https://doi.org/10.1186/s40537-021-00507-w

[11] R. Uddin and I. Koo, "Real-Time Remote Patient Monitoring: A Review of Biosensors Integrated with Multi-Hop IoT Systems via Cloud Connectivity," Applied Sciences, vol. 14, no. 5, p. 1876, Jan. 2024, doi: https://doi.org/10.3390/app14051876

[12] A. Olyanasab and M. Annabestani, "Leveraging Machine Learning for Personalized Wearable Biomedical Devices: A Review," Journal of Personalized Medicine, vol. 14, no. 2, p. 203, Feb. 2024, doi: https://doi.org/10.3390/jpm14020203

[13] "Raspberry Pi 4 Model B (4GB)," www.autobotic.com.my.

https://www.autobotic.com.my/Raspberry-Pi-4-Model-B-4GB-1-Yr-Warranty

[14]"IoT Based Contactless Body Temperature Monitoring using MLX90614 Infrared Temperature Sensor, Raspberry Pi with Camera and Email Alert," *circuitdigest.com*. https://circuitdigest.com/microcontroller-projects/iot-based-contactless-body-temperature-monitoring-using-raspberry-pi-with-camera-and-email-alert

[15]"MAX30102 Oximeter and Heart Rate Sensor Module," *Cytron Technologies Malaysia*. https://my.cytron.io/p-max30102-oximeter-and-heart-rate-sensor-module?r=1

[16]"MLX90614 Non-Contact Infrared Temperature Sensor," *Cytron Technologies Malaysia*, 2024. https://my.cytron.io/p-mlx90614-non-contact-infrared-temperature-sensor

[17]"Interfacing MAX30100 Pulse Oximeter and Heart Rate Sensor with Arduino," *Last Minute Engineers*, Feb. 05, 2022. https://lastminuteengineers.com/max30100-pulse-oximeter-heart-rate-sensor-arduino-tutorial/

[18]D. Das, "How MAX30102 Pulse Oximeter and Heart Rate Sensor Works and how to Interface it with Arduino?," *circuitdigest.com*, May 04, 2022. https://circuitdigest.com/microcontroller-projects/how-max30102-pulse-oximeter-and-heart-rate-sensor-works-and-how-to-interface-with-arduino

[19]P. Valiya, S. Shinde, and M. Paraye, "GUI BASED HEALTH MONITORING SYSTEM USING RASPBERRY PI." Accessed: Sep. 04, 2024. [Online]. Available: https://repo.ijiert.org/index.php/ijiert/article/download/2243/2105/4217

[20]"Raspberry Pi 4 Model B Default GPIO Pinout with PoE Header - Documents - Raspberry Pi - element14 Community," *community.element14.com*. https://community.element14.com/products/raspberry-pi/w/documents/4317/raspberry-pi-4-model-b-default-gpio-pinout-with-poe-header

APPENDIX

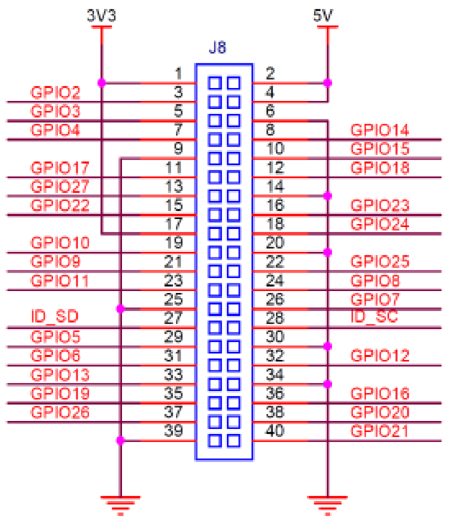
Raspberry Pi 4 Model B Datasheet

5 Peripherals

5.1 GPIO Interface

The Pi4B makes 28 BCM2711 GPIOs available via a standard Raspberry Pi 40-pin header. This header is backwards compatible with all previous Raspberry Pi boards with a 40-way header.

5.1.1 GPIO Pin Assignments



ID_SD and ID_SC PINS:

These pins are reserved for HAT ID EEPROM.

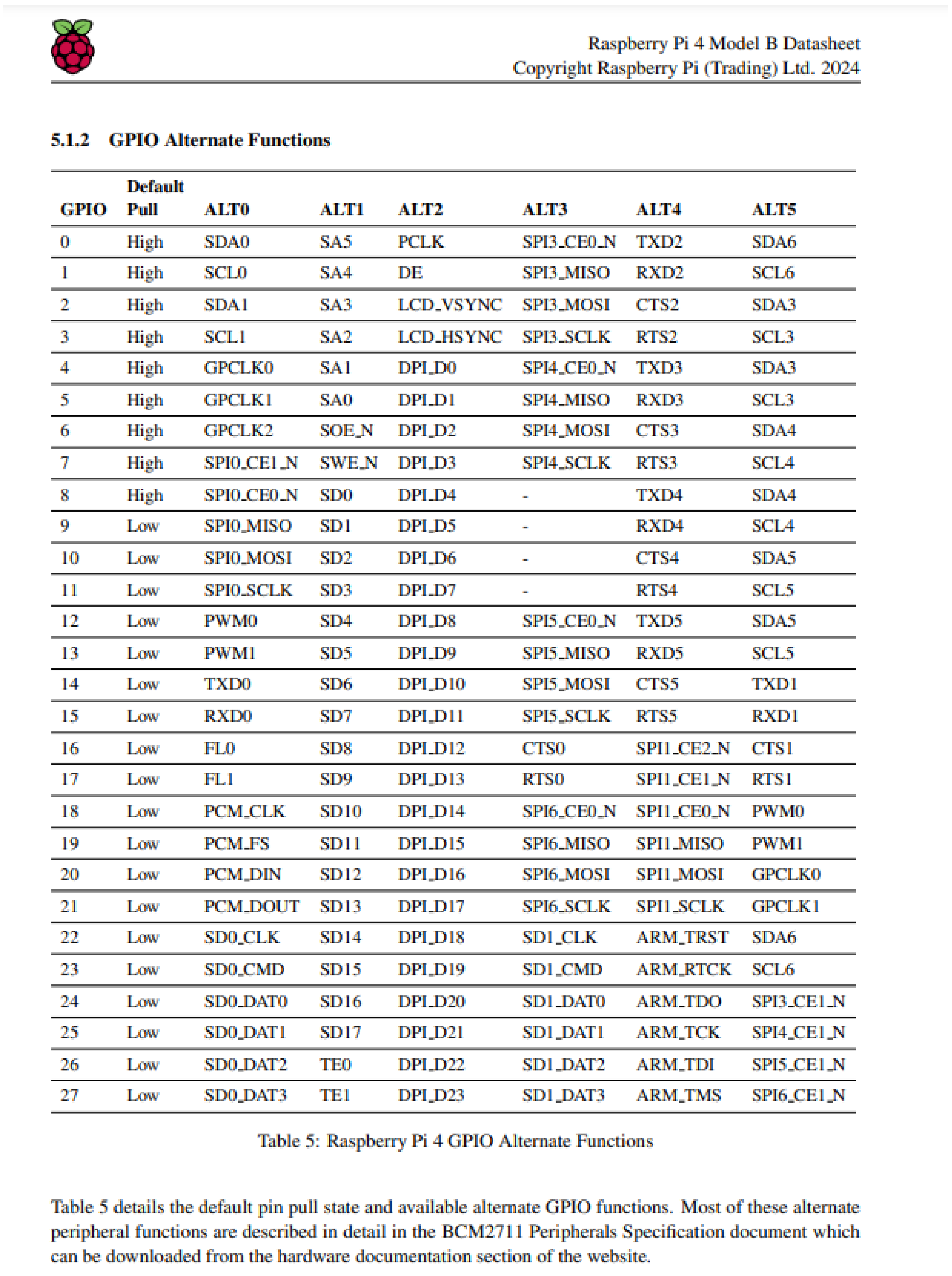
At boot time this I2C interface will be interrogated to look for an EEPROM that identifes the attached board and allows automagic setup of the GPIOs (and optionally, Linux drivers).

DO NOT USE these pins for anything other than attaching an I2C ID EEPROM. Leave unconnected if ID EEPROM not required.

Figure 3: GPIO Connector Pinout

As well as being able to be used as straightforward software controlled input and output (with programmable pulls), GPIO pins can be switched (multiplexed) into various other modes backed by dedicated peripheral blocks such as I2C, UART and SPI.

In addition to the standard peripheral options found on legacy Pis, extra I2C, UART and SPI peripherals have been added to the BCM2711 chip and are available as further mux options on the Pi 4. This gives users much more flexibility when attaching add-on hardware as compared to older models.



Raspberry Pi 4 Model B Datasheet
Copyright Raspberry Pi (Trading) Ltd. 2024

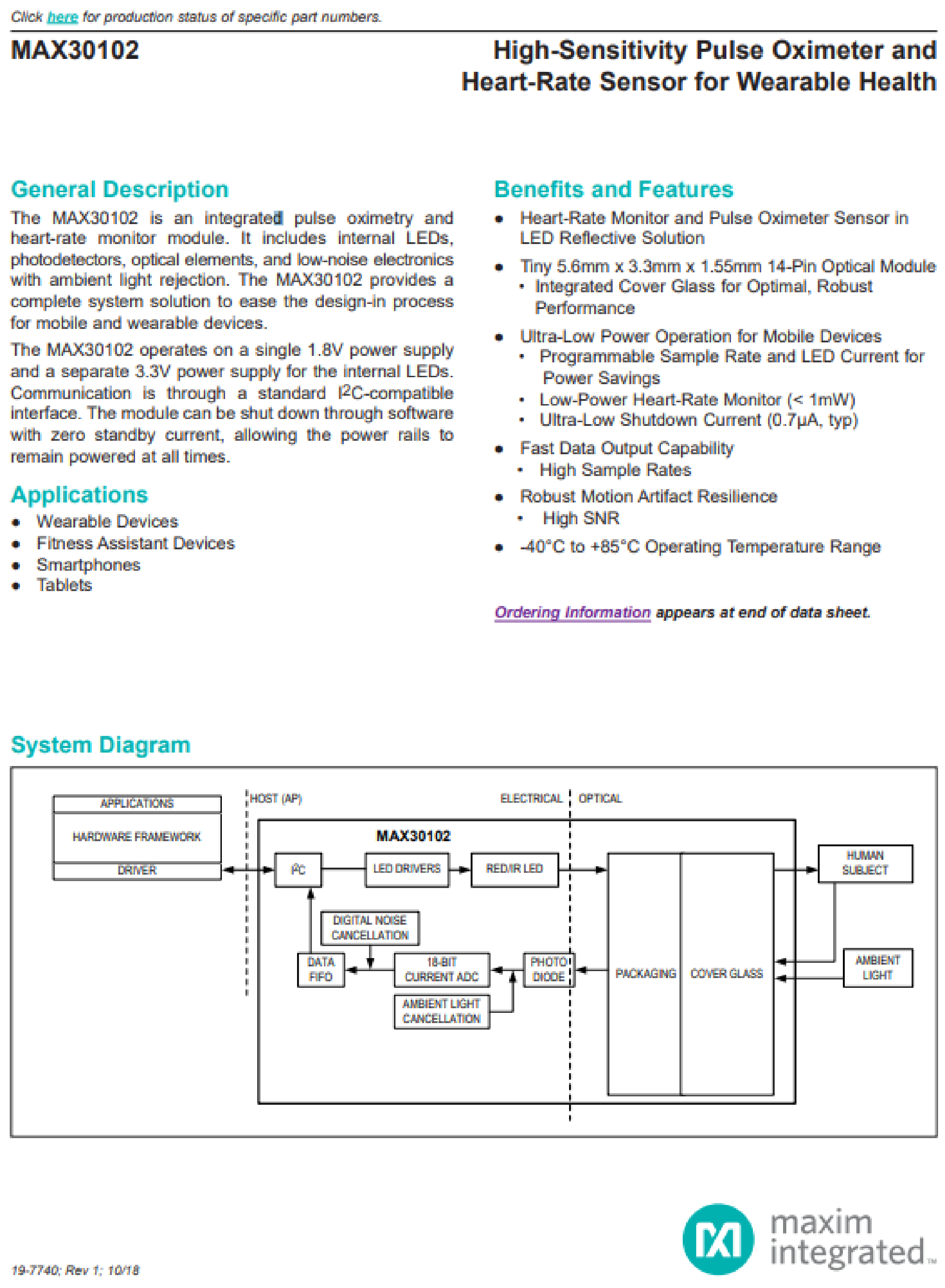
5.1.2 GPIO Alternate Functions

GPIO	Default Pull	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5
0	High	SDA0	SA5	PCLK	SPI3_CE0_N	TXD2	SDA6
1	High	SCL0	SA4	DE	SPI3_MISO	RXD2	SCL6
2	High	SDA1	SA3	LCD_VSYNC	SPI3_MOSI	CTS2	SDA3
3	High	SCL1	SA2	LCD_HSYNC	SPI3_SCLK	RTS2	SCL3
4	High	GPCLK0	SA1	DPI_D0	SPI4_CE0_N	TXD3	SDA3
5	High	GPCLK1	SA0	DPI_D1	SPI4_MISO	RXD3	SCL3
6	High	GPCLK2	SOE_N	DPI_D2	SPI4_MOSI	CTS3	SDA4
7	High	SPI0_CE1_N	SWE_N	DPI_D3	SPI4_SCLK	RTS3	SCL4
8	High	SPI0_CE0_N	SD0	DPI_D4	-	TXD4	SDA4
9	Low	SPI0_MISO	SD1	DPI_D5	-	RXD4	SCL4
10	Low	SPI0_MOSI	SD2	DPI_D6	-	CTS4	SDA5
11	Low	SPI0_SCLK	SD3	DPI_D7	-	RTS4	SCL5
12	Low	PWM0	SD4	DPI_D8	SPI5_CE0_N	TXD5	SDA5
13	Low	PWM1	SD5	DPI_D9	SPI5_MISO	RXD5	SCL5
14	Low	TXD0	SD6	DPI_D10	SPI5_MOSI	CTS5	TXD1
15	Low	RXD0	SD7	DPI_D11	SPI5_SCLK	RTS5	RXD1
16	Low	FL0	SD8	DPI_D12	CTS0	SPI1_CE2_N	CTS1
17	Low	FL1	SD9	DPI_D13	RTS0	SPI1_CE1_N	RTS1
18	Low	PCM_CLK	SD10	DPI_D14	SPI6_CE0_N	SPI1_CE0_N	PWM0
19	Low	PCM_FS	SD11	DPI_D15	SPI6_MISO	SPI1_MISO	PWM1
20	Low	PCM_DIN	SD12	DPI_D16	SPI6_MOSI	SPI1_MOSI	GPCLK0
21	Low	PCM_DOUT	SD13	DPI_D17	SPI6_SCLK	SPI1_SCLK	GPCLK1
22	Low	SD0_CLK	SD14	DPI_D18	SD1_CLK	ARM_TRST	SDA6
23	Low	SD0_CMD	SD15	DPI_D19	SD1_CMD	ARM_RTCK	SCL6
24	Low	SD0_DAT0	SD16	DPI_D20	SD1_DAT0	ARM_TDO	SPI3_CE1_N
25	Low	SD0_DAT1	SD17	DPI_D21	SD1_DAT1	ARM_TCK	SPI4_CE1_N
26	Low	SD0_DAT2	TE0	DPI_D22	SD1_DAT2	ARM_TDI	SPI5_CE1_N
27	Low	SD0_DAT3	TE1	DPI_D23	SD1_DAT3	ARM_TMS	SPI6_CE1_N

Table 5: Raspberry Pi 4 GPIO Alternate Functions

Table 5 details the default pin pull state and available alternate GPIO functions. Most of these alternate peripheral functions are described in detail in the BCM2711 Peripherals Specification document which can be downloaded from the hardware documentation section of the website.

MAX30102 Sensor Datasheet



Click here for production status of specific part numbers.

MAX30102 **High-Sensitivity Pulse Oximeter and Heart-Rate Sensor for Wearable Health**

General Description

The MAX30102 is an integrated pulse oximetry and heart-rate monitor module. It includes internal LEDs, photodetectors, optical elements, and low-noise electronics with ambient light rejection. The MAX30102 provides a complete system solution to ease the design-in process for mobile and wearable devices.

The MAX30102 operates on a single 1.8V power supply and a separate 3.3V power supply for the internal LEDs. Communication is through a standard I²C-compatible interface. The module can be shut down through software with zero standby current, allowing the power rails to remain powered at all times.

Applications

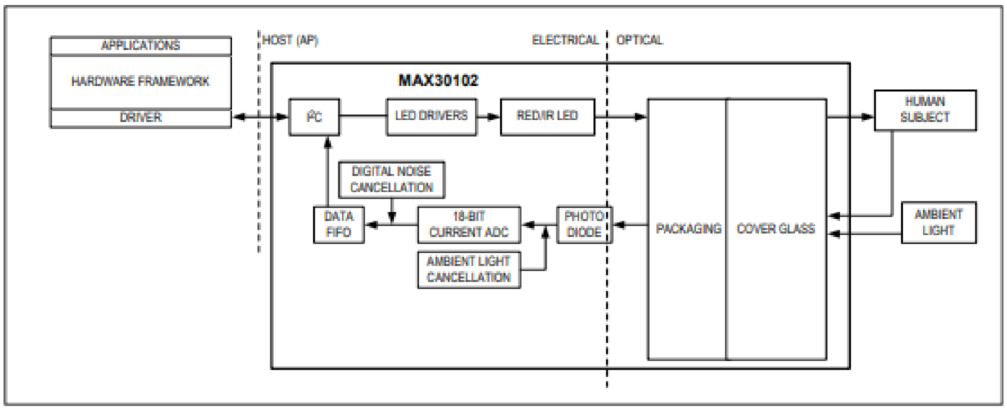
- Wearable Devices
- Fitness Assistant Devices
- Smartphones
- Tablets

Benefits and Features

- Heart-Rate Monitor and Pulse Oximeter Sensor in LED Reflective Solution
- Tiny 5.6mm x 3.3mm x 1.55mm 14-Pin Optical Module
  - Integrated Cover Glass for Optimal, Robust Performance
- Ultra-Low Power Operation for Mobile Devices
  - Programmable Sample Rate and LED Current for Power Savings
  - Low-Power Heart-Rate Monitor (< 1mW)
  - Ultra-Low Shutdown Current (0.7µA, typ)
- Fast Data Output Capability
  - High Sample Rates
- Robust Motion Artifact Resilience
  - High SNR
- -40°C to +85°C Operating Temperature Range

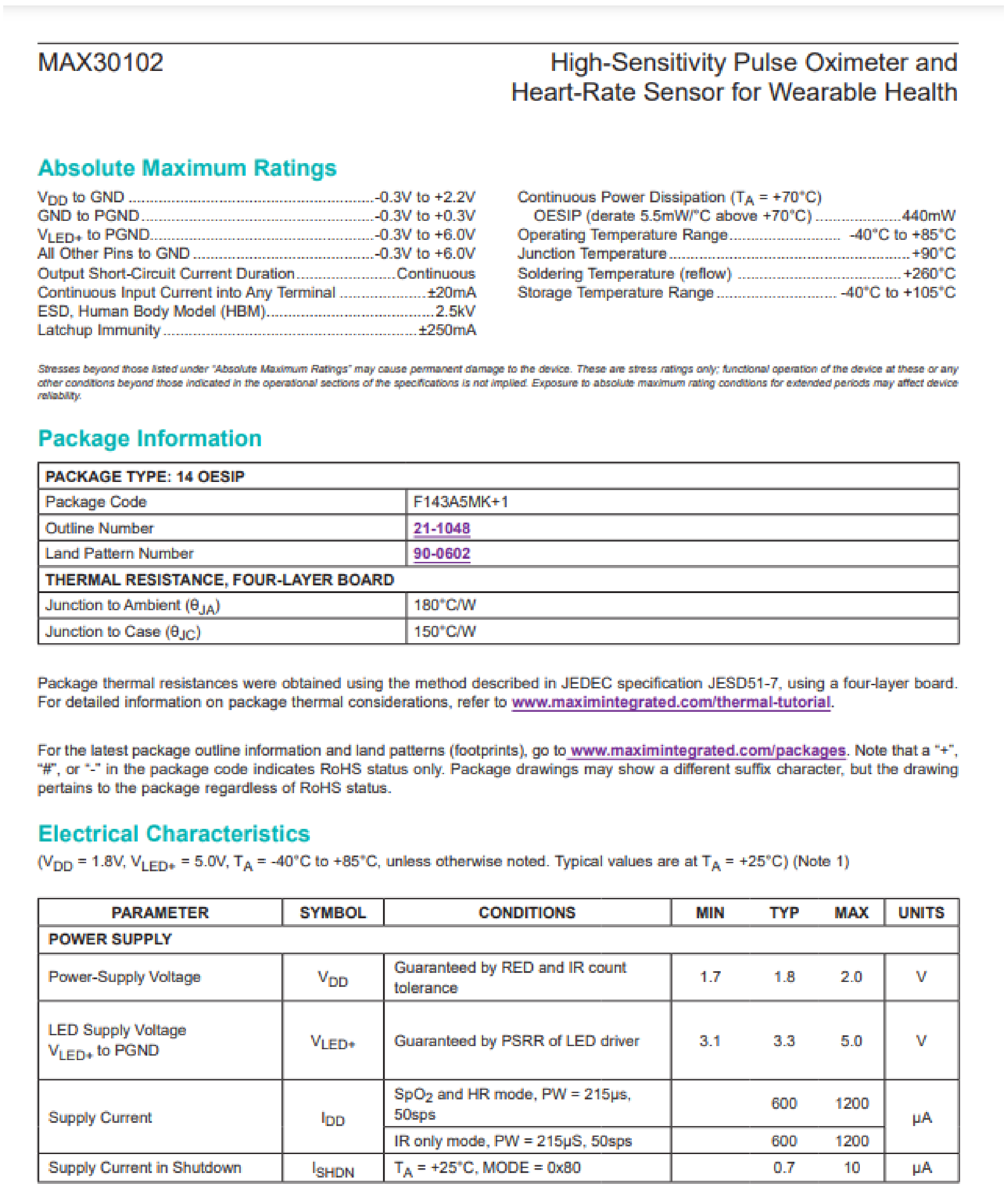
Ordering Information ***appears at end of data sheet.***

System Diagram



maxim integrated

19-7740; Rev 1; 10/18



MAX30102 — High-Sensitivity Pulse Oximeter and Heart-Rate Sensor for Wearable Health

Absolute Maximum Ratings

V_{DD} to GND	-0.3V to +2.2V
GND to PGND	-0.3V to +0.3V
V_{LED+} to PGND	-0.3V to +6.0V
All Other Pins to GND	-0.3V to +6.0V
Output Short-Circuit Current Duration	Continuous
Continuous Input Current into Any Terminal	±20mA
ESD, Human Body Model (HBM)	2.5kV
Latchup Immunity	±250mA
Continuous Power Dissipation (T_A = +70°C)	
OESIP (derate 5.5mW/°C above +70°C)	440mW
Operating Temperature Range	-40°C to +85°C
Junction Temperature	+90°C
Soldering Temperature (reflow)	+260°C
Storage Temperature Range	-40°C to +105°C

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only; functional operation of the device at these or any other conditions beyond those indicated in the operational sections of the specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Package Information

PACKAGE TYPE: 14 OESIP	
Package Code	F143A5MK+1
Outline Number	21-1048
Land Pattern Number	90-0602
THERMAL RESISTANCE, FOUR-LAYER BOARD	
Junction to Ambient (θ_{JA})	180°C/W
Junction to Case (θ_{JC})	150°C/W

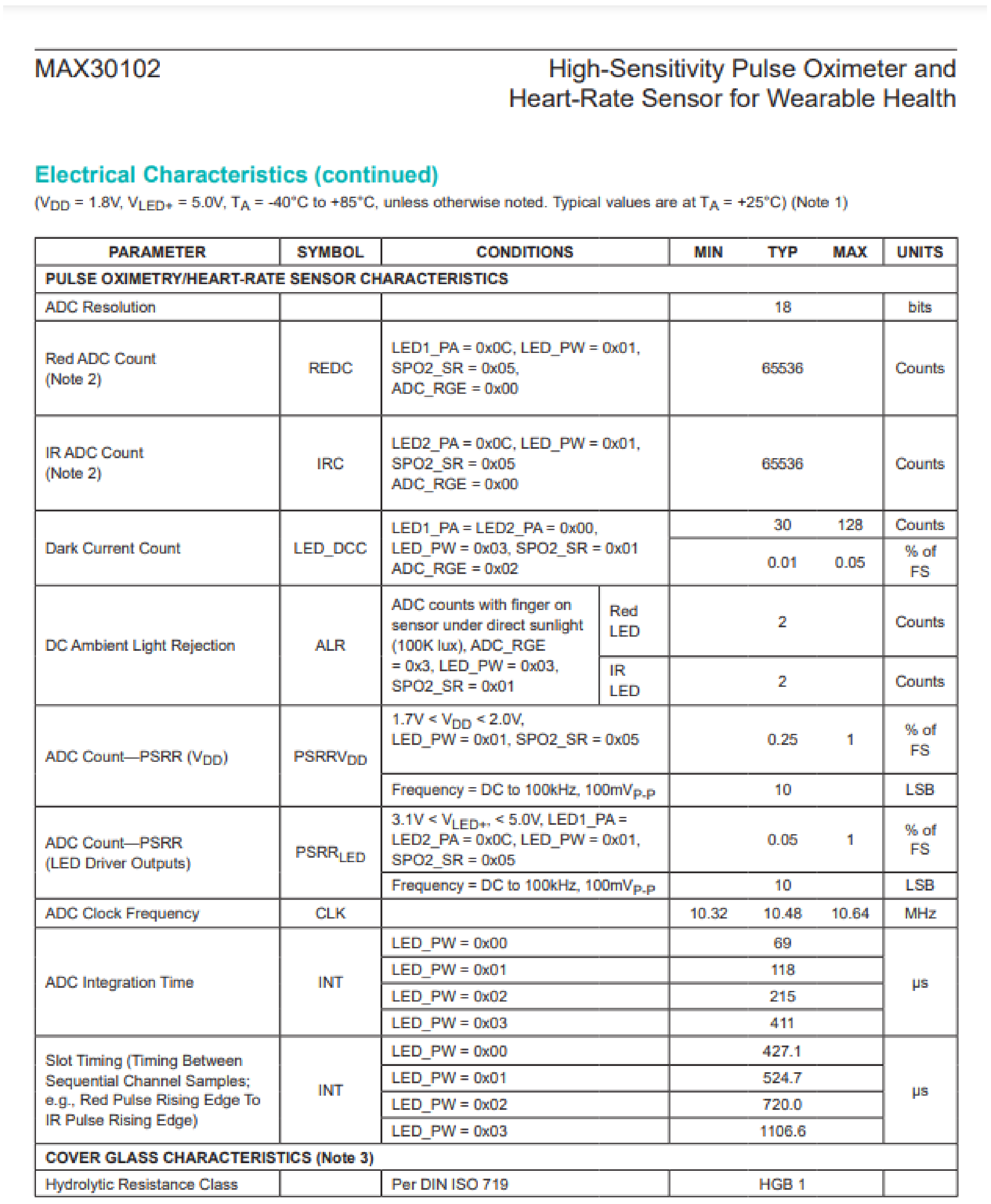
Package thermal resistances were obtained using the method described in JEDEC specification JESD51-7, using a four-layer board. For detailed information on package thermal considerations, refer to www.maximintegrated.com/thermal-tutorial.

For the latest package outline information and land patterns (footprints), go to www.maximintegrated.com/packages. Note that a "+", "#", or "-" in the package code indicates RoHS status only. Package drawings may show a different suffix character, but the drawing pertains to the package regardless of RoHS status.

Electrical Characteristics

(V_{DD} = 1.8V, V_{LED+} = 5.0V, T_A = -40°C to +85°C, unless otherwise noted. Typical values are at T_A = +25°C) (Note 1)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
POWER SUPPLY						
Power-Supply Voltage	V_{DD}	Guaranteed by RED and IR count tolerance	1.7	1.8	2.0	V
LED Supply Voltage V_{LED+} to PGND	V_{LED+}	Guaranteed by PSRR of LED driver	3.1	3.3	5.0	V
Supply Current	I_{DD}	SpO_2 and HR mode, PW = 215μs, 50sps		600	1200	μA
		IR only mode, PW = 215μS, 50sps		600	1200	
Supply Current in Shutdown	I_{SHDN}	T_A = +25°C, MODE = 0x80		0.7	10	μA

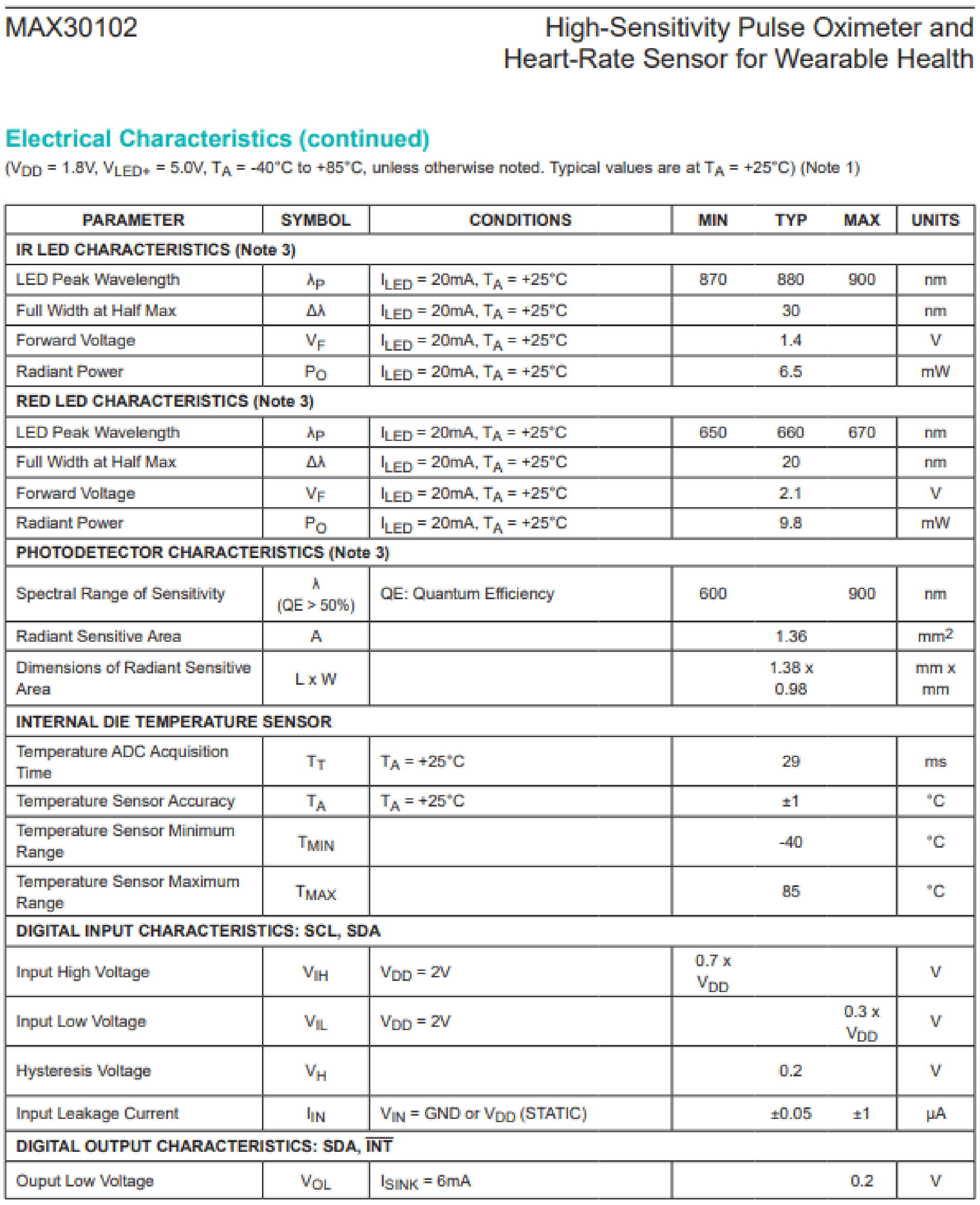


MAX30102 High-Sensitivity Pulse Oximeter and Heart-Rate Sensor for Wearable Health

Electrical Characteristics (continued)

(V_{DD} = 1.8V, V_{LED+} = 5.0V, T_A = -40°C to +85°C, unless otherwise noted. Typical values are at T_A = +25°C) (Note 1)

PARAMETER	SYMBOL	CONDITIONS		MIN	TYP	MAX	UNITS
PULSE OXIMETRY/HEART-RATE SENSOR CHARACTERISTICS							
ADC Resolution					18		bits
Red ADC Count (Note 2)	REDC	LED1_PA = 0x0C, LED_PW = 0x01, SPO2_SR = 0x05, ADC_RGE = 0x00			65536		Counts
IR ADC Count (Note 2)	IRC	LED2_PA = 0x0C, LED_PW = 0x01, SPO2_SR = 0x05 ADC_RGE = 0x00			65536		Counts
Dark Current Count	LED_DCC	LED1_PA = LED2_PA = 0x00, LED_PW = 0x03, SPO2_SR = 0x01 ADC_RGE = 0x02			30	128	Counts
					0.01	0.05	% of FS
DC Ambient Light Rejection	ALR	ADC counts with finger on sensor under direct sunlight (100K lux), ADC_RGE = 0x3, LED_PW = 0x03, SPO2_SR = 0x01	Red LED		2		Counts
			IR LED		2		Counts
ADC Count—PSRR (V_{DD})	$PSRRV_{DD}$	1.7V < V_{DD} < 2.0V, LED_PW = 0x01, SPO2_SR = 0x05			0.25	1	% of FS
		Frequency = DC to 100kHz, $100mV_{P-P}$			10		LSB
ADC Count—PSRR (LED Driver Outputs)	$PSRR_{LED}$	3.1V < V_{LED+} < 5.0V, LED1_PA = LED2_PA = 0x0C, LED_PW = 0x01, SPO2_SR = 0x05			0.05	1	% of FS
		Frequency = DC to 100kHz, $100mV_{P-P}$			10		LSB
ADC Clock Frequency	CLK			10.32	10.48	10.64	MHz
ADC Integration Time	INT	LED_PW = 0x00			69		µs
		LED_PW = 0x01			118		
		LED_PW = 0x02			215		
		LED_PW = 0x03			411		
Slot Timing (Timing Between Sequential Channel Samples; e.g., Red Pulse Rising Edge To IR Pulse Rising Edge)	INT	LED_PW = 0x00			427.1		µs
		LED_PW = 0x01			524.7		
		LED_PW = 0x02			720.0		
		LED_PW = 0x03			1106.6		
COVER GLASS CHARACTERISTICS (Note 3)							
Hydrolytic Resistance Class		Per DIN ISO 719			HGB 1		



MAX30102

High-Sensitivity Pulse Oximeter and Heart-Rate Sensor for Wearable Health

Electrical Characteristics (continued)

(V_{DD} = 1.8V, V_{LED+} = 5.0V, T_A = -40°C to +85°C, unless otherwise noted. Typical values are at T_A = +25°C) (Note 1)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
IR LED CHARACTERISTICS (Note 3)						
LED Peak Wavelength	λ_P	I_{LED} = 20mA, T_A = +25°C	870	880	900	nm
Full Width at Half Max	$\Delta\lambda$	I_{LED} = 20mA, T_A = +25°C		30		nm
Forward Voltage	V_F	I_{LED} = 20mA, T_A = +25°C		1.4		V
Radiant Power	P_O	I_{LED} = 20mA, T_A = +25°C		6.5		mW
RED LED CHARACTERISTICS (Note 3)						
LED Peak Wavelength	λ_P	I_{LED} = 20mA, T_A = +25°C	650	660	670	nm
Full Width at Half Max	$\Delta\lambda$	I_{LED} = 20mA, T_A = +25°C		20		nm
Forward Voltage	V_F	I_{LED} = 20mA, T_A = +25°C		2.1		V
Radiant Power	P_O	I_{LED} = 20mA, T_A = +25°C		9.8		mW
PHOTODETECTOR CHARACTERISTICS (Note 3)						
Spectral Range of Sensitivity	λ (QE > 50%)	QE: Quantum Efficiency	600		900	nm
Radiant Sensitive Area	A			1.36		mm^2
Dimensions of Radiant Sensitive Area	L x W			1.38 x 0.98		mm x mm
INTERNAL DIE TEMPERATURE SENSOR						
Temperature ADC Acquisition Time	T_T	T_A = +25°C		29		ms
Temperature Sensor Accuracy	T_A	T_A = +25°C		±1		°C
Temperature Sensor Minimum Range	T_{MIN}			-40		°C
Temperature Sensor Maximum Range	T_{MAX}			85		°C
DIGITAL INPUT CHARACTERISTICS: SCL, SDA						
Input High Voltage	V_{IH}	V_{DD} = 2V	0.7 x V_{DD}			V
Input Low Voltage	V_{IL}	V_{DD} = 2V			0.3 x V_{DD}	V
Hysteresis Voltage	V_H			0.2		V
Input Leakage Current	I_{IN}	V_{IN} = GND or V_{DD} (STATIC)		±0.05	±1	µA
DIGITAL OUTPUT CHARACTERISTICS: SDA, $\overline{INT}$						
Ouput Low Voltage	V_{OL}	I_{SINK} = 6mA			0.2	V

MAX30102 High-Sensitivity Pulse Oximeter and Heart-Rate Sensor for Wearable Health

Electrical Characteristics (continued)

(V_{DD} = 1.8V, V_{LED+} = 5.0V, T_A = -40°C to +85°C, unless otherwise noted. Typical values are at T_A = +25°C) (Note 1)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
I²C TIMING CHARACTERISTICS (SDA, SDA, $\overline{INT}$) (Note 3)						
I²C Write Address				AE		Hex
I²C Read Address				AF		Hex
Serial Clock Frequency	f_{SCL}		0		400	kHz
Bus Free Time Between STOP and START Conditions	t_{BUF}		1.3			µs
Hold Time (Repeated) START Condition	$t_{HD;STA}$		0.6			µs
SCL Pulse-Width Low	t_{LOW}		1.3			µs
SCL Pulse-Width High	t_{HIGH}		0.6			µs
Setup Time for a Repeated START Condition	$t_{SU;STA}$		0.6			µs
Data Hold Time	$t_{HD;DAT}$		0		900	ns
Data Setup Time	$t_{SU;DAT}$		100			ns
Setup Time for STOP Condition	$t_{SU;STO}$		0.6			µs
Pulse Width of Suppressed Spike	t_{SP}		0		50	ns
Bus Capacitance	C_B				400	pF
SDA and SCL Receiving Rise Time	t_R		20 + 0.1C_B		300	ns
SDA and SCL Receiving Fall Time	t_{RF}		20 + 0.1C_B		300	ns
SDA Transmitting Fall Time	t_{TF}				300	ns

Note 1: All devices are 100% production tested at T_A = +25°C. Specifications over temperature limits are guaranteed by Maxim Integrated's bench or proprietary automated test equipment (ATE) characterization.

Note 2: Specifications are guaranteed by Maxim Integrated's bench characterization and by 100% production test using proprietary ATE setup and conditions.

Note 3: Guaranteed by design and characterization. Not tested in final production.

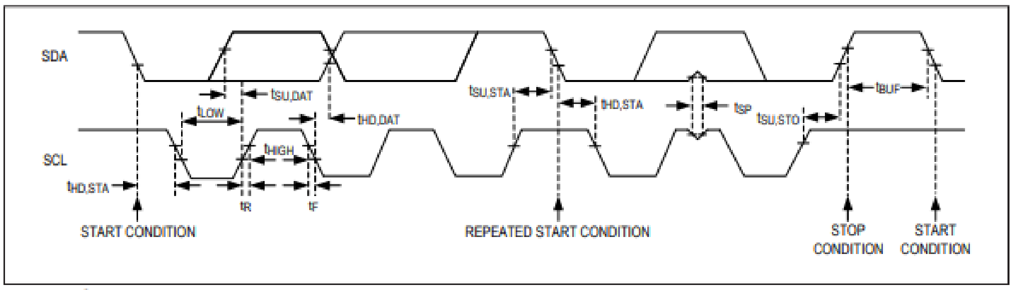
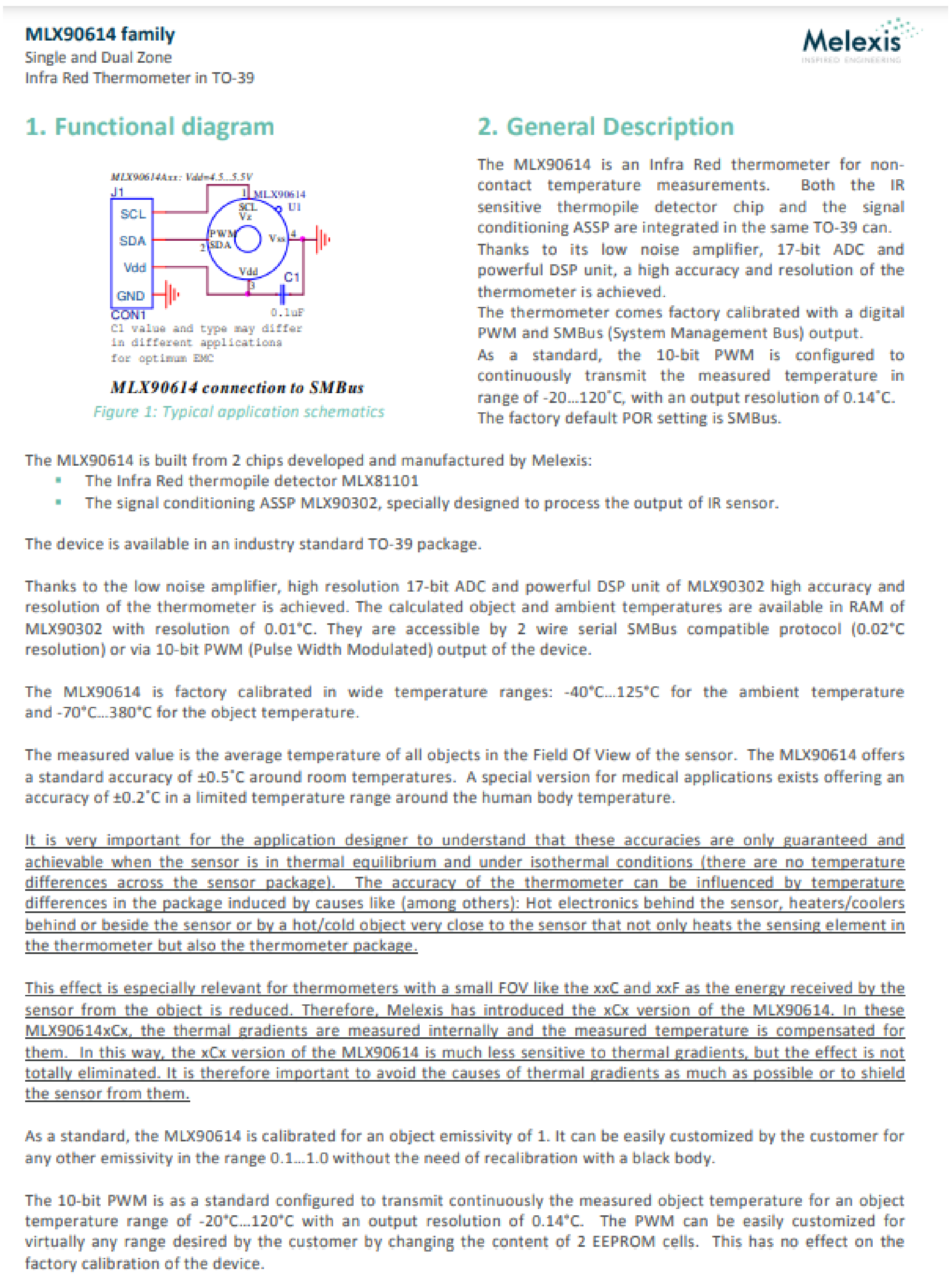


Figure 1. I²C-Compatible Interface Timing Diagram

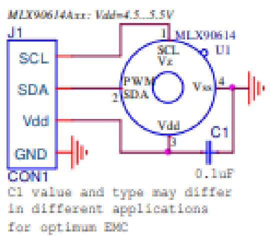
MLX90614 Sensor Datasheet



MLX90614 family
Single and Dual Zone
Infra Red Thermometer in TO-39

Melexis

1. Functional diagram


MLX90614Axx: Vdd=4.5...5.5V
J1
SCL
SDA
Vdd
GND
CON1
MLX90614
U1
SCL Vz
PWM SDA
Vss
Vdd
C1
0.1uF
C1 value and type may differ in different applications for optimum EMC

MLX90614 connection to SMBus

Figure 1: Typical application schematics

2. General Description

The MLX90614 is an Infra Red thermometer for non-contact temperature measurements. Both the IR sensitive thermopile detector chip and the signal conditioning ASSP are integrated in the same TO-39 can.
Thanks to its low noise amplifier, 17-bit ADC and powerful DSP unit, a high accuracy and resolution of the thermometer is achieved.
The thermometer comes factory calibrated with a digital PWM and SMBus (System Management Bus) output.
As a standard, the 10-bit PWM is configured to continuously transmit the measured temperature in range of -20…120˚C, with an output resolution of 0.14˚C.
The factory default POR setting is SMBus.

The MLX90614 is built from 2 chips developed and manufactured by Melexis:

- The Infra Red thermopile detector MLX81101
- The signal conditioning ASSP MLX90302, specially designed to process the output of IR sensor.

The device is available in an industry standard TO-39 package.

Thanks to the low noise amplifier, high resolution 17-bit ADC and powerful DSP unit of MLX90302 high accuracy and resolution of the thermometer is achieved. The calculated object and ambient temperatures are available in RAM of MLX90302 with resolution of 0.01°C. They are accessible by 2 wire serial SMBus compatible protocol (0.02°C resolution) or via 10-bit PWM (Pulse Width Modulated) output of the device.

The MLX90614 is factory calibrated in wide temperature ranges: -40°C…125°C for the ambient temperature and -70°C…380°C for the object temperature.

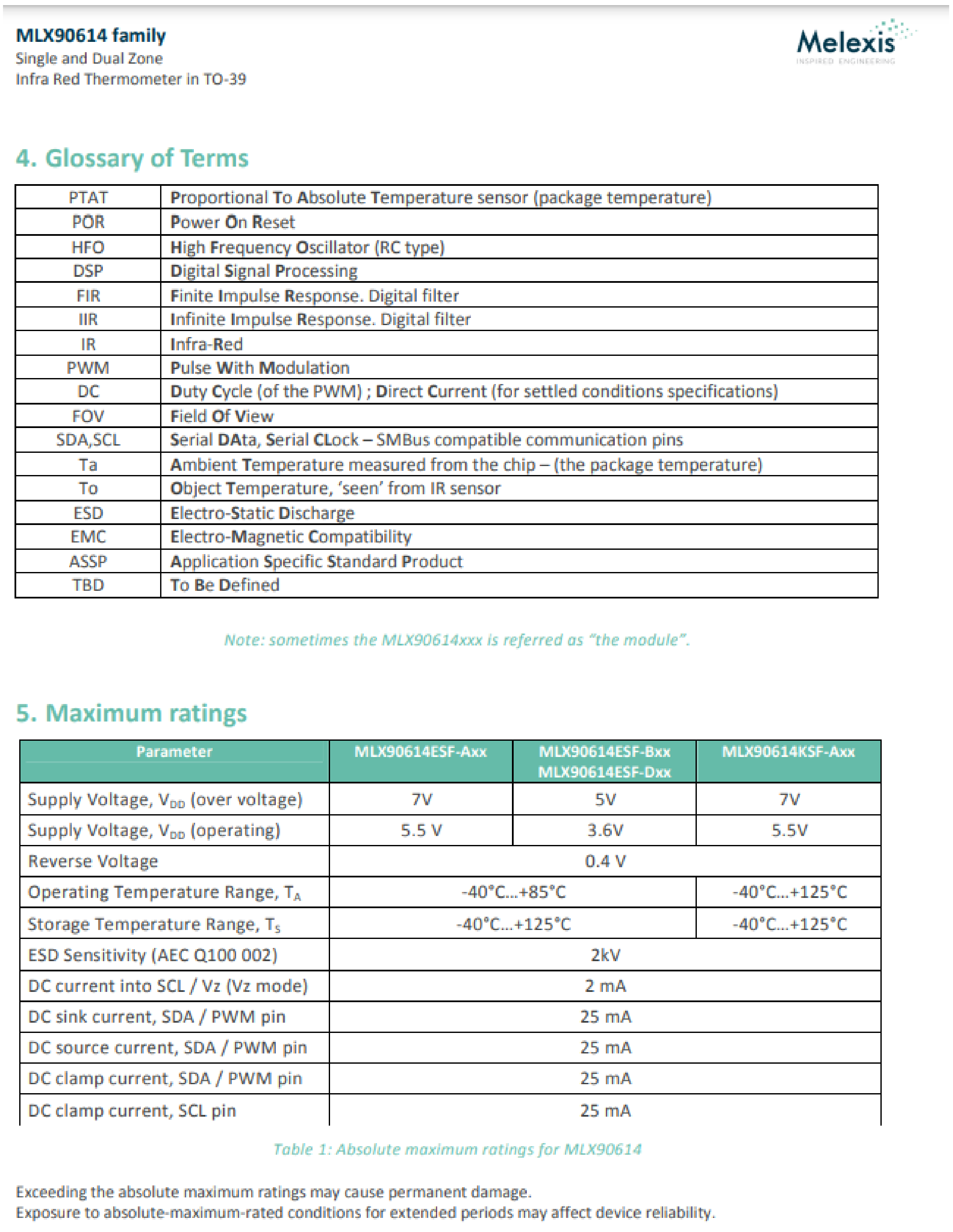
The measured value is the average temperature of all objects in the Field Of View of the sensor. The MLX90614 offers a standard accuracy of ±0.5˚C around room temperatures. A special version for medical applications exists offering an accuracy of ±0.2˚C in a limited temperature range around the human body temperature.

It is very important for the application designer to understand that these accuracies are only guaranteed and achievable when the sensor is in thermal equilibrium and under isothermal conditions (there are no temperature differences across the sensor package). The accuracy of the thermometer can be influenced by temperature differences in the package induced by causes like (among others): Hot electronics behind the sensor, heaters/coolers behind or beside the sensor or by a hot/cold object very close to the sensor that not only heats the sensing element in the thermometer but also the thermometer package.

This effect is especially relevant for thermometers with a small FOV like the xxC and xxF as the energy received by the sensor from the object is reduced. Therefore, Melexis has introduced the xCx version of the MLX90614. In these MLX90614xCx, the thermal gradients are measured internally and the measured temperature is compensated for them. In this way, the xCx version of the MLX90614 is much less sensitive to thermal gradients, but the effect is not totally eliminated. It is therefore important to avoid the causes of thermal gradients as much as possible or to shield the sensor from them.

As a standard, the MLX90614 is calibrated for an object emissivity of 1. It can be easily customized by the customer for any other emissivity in the range 0.1…1.0 without the need of recalibration with a black body.

The 10-bit PWM is as a standard configured to transmit continuously the measured object temperature for an object temperature range of -20°C…120°C with an output resolution of 0.14°C. The PWM can be easily customized for virtually any range desired by the customer by changing the content of 2 EEPROM cells. This has no effect on the factory calibration of the device.



MLX90614 family
Single and Dual Zone
Infra Red Thermometer in TO-39

Melexis
INSPIRED ENGINEERING

4. Glossary of Terms

PTAT	Proportional To Absolute Temperature sensor (package temperature)
POR	Power On Reset
HFO	High Frequency Oscillator (RC type)
DSP	Digital Signal Processing
FIR	Finite Impulse Response. Digital filter
IIR	Infinite Impulse Response. Digital filter
IR	Infra-Red
PWM	Pulse With Modulation
DC	Duty Cycle (of the PWM) ; Direct Current (for settled conditions specifications)
FOV	Field Of View
SDA,SCL	Serial DAta, Serial CLock – SMBus compatible communication pins
Ta	Ambient Temperature measured from the chip – (the package temperature)
To	Object Temperature, 'seen' from IR sensor
ESD	Electro-Static Discharge
EMC	Electro-Magnetic Compatibility
ASSP	Application Specific Standard Product
TBD	To Be Defined

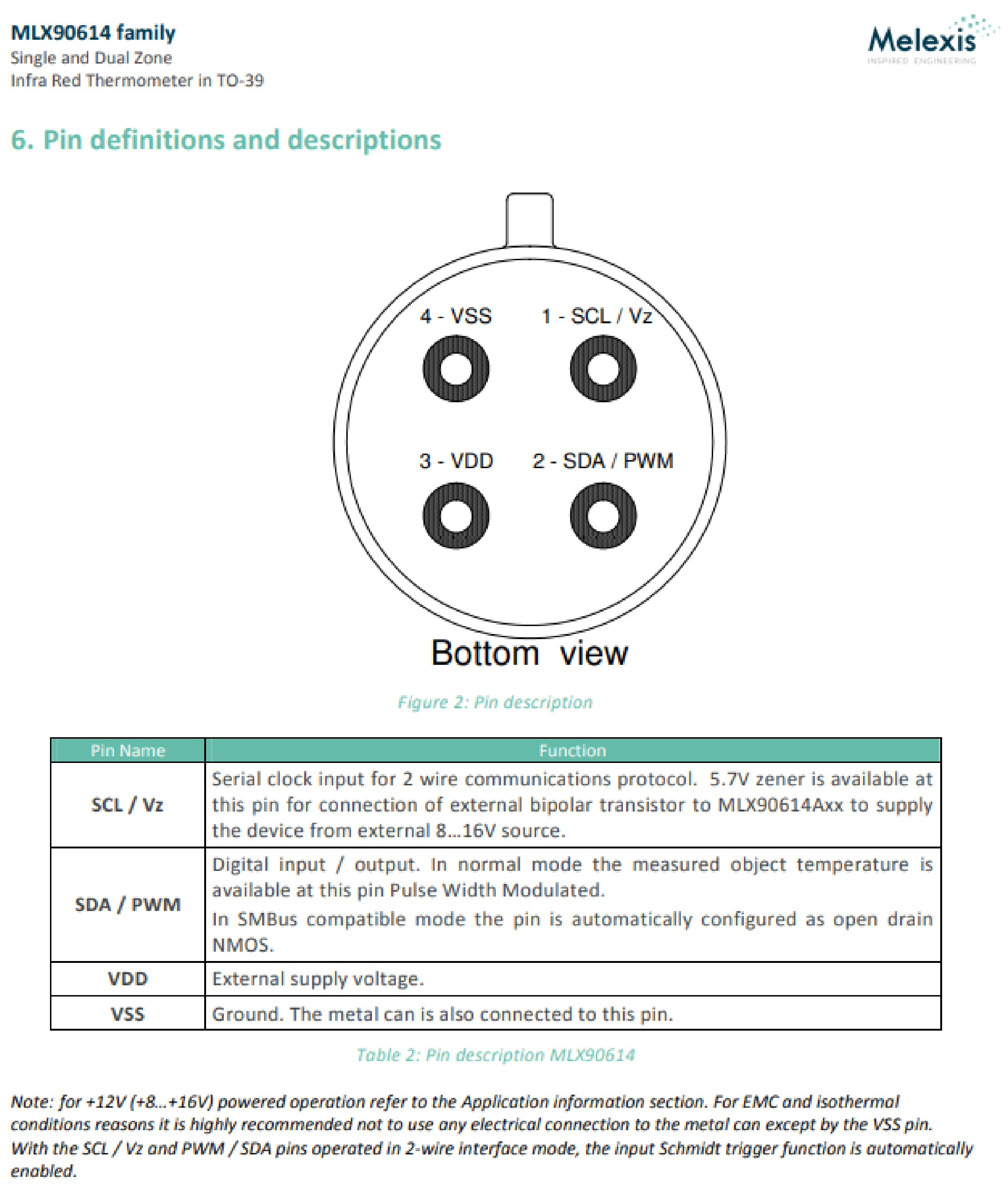
Note: sometimes the MLX90614xxx is referred as "the module".

5. Maximum ratings

Parameter	MLX90614ESF-Axx	MLX90614ESF-Bxx MLX90614ESF-Dxx	MLX90614KSF-Axx
Supply Voltage, V_{DD} (over voltage)	7V	5V	7V
Supply Voltage, V_{DD} (operating)	5.5 V	3.6V	5.5V
Reverse Voltage	0.4 V		
Operating Temperature Range, T_A	-40°C…+85°C		-40°C…+125°C
Storage Temperature Range, T_S	-40°C…+125°C		-40°C…+125°C
ESD Sensitivity (AEC Q100 002)	2kV		
DC current into SCL / Vz (Vz mode)	2 mA		
DC sink current, SDA / PWM pin	25 mA		
DC source current, SDA / PWM pin	25 mA		
DC clamp current, SDA / PWM pin	25 mA		
DC clamp current, SCL pin	25 mA		

Table 1: Absolute maximum ratings for MLX90614

Exceeding the absolute maximum ratings may cause permanent damage.
Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.



MLX90614 family
Single and Dual Zone
Infra Red Thermometer in TO-39

Melexis
INSPIRED ENGINEERING

6. Pin definitions and descriptions

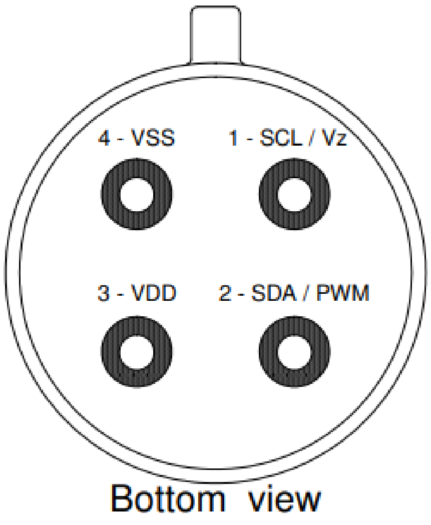
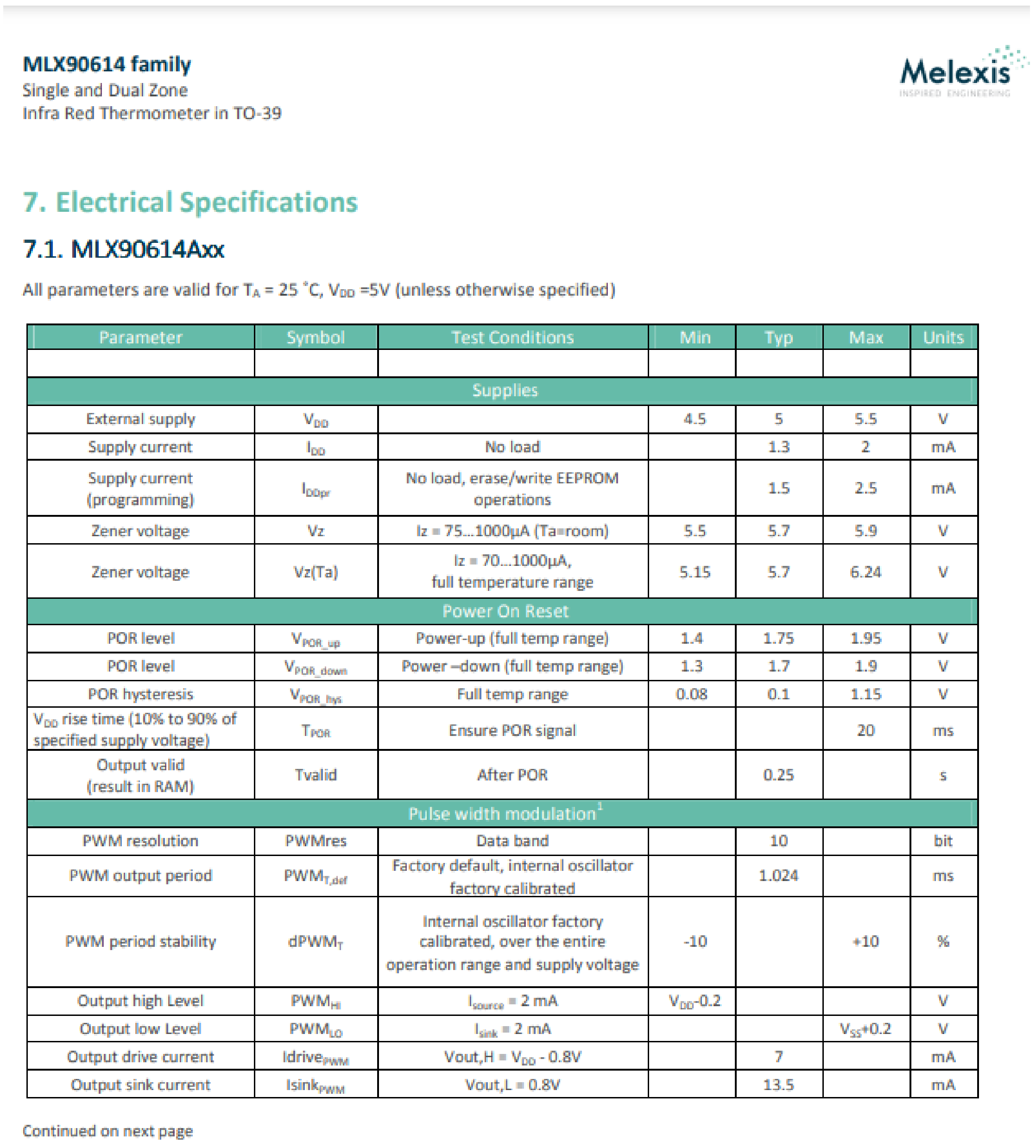


Figure 2: Pin description

Pin Name	Function
SCL / Vz	Serial clock input for 2 wire communications protocol. 5.7V zener is available at this pin for connection of external bipolar transistor to MLX90614Axx to supply the device from external 8…16V source.
SDA / PWM	Digital input / output. In normal mode the measured object temperature is available at this pin Pulse Width Modulated. In SMBus compatible mode the pin is automatically configured as open drain NMOS.
VDD	External supply voltage.
VSS	Ground. The metal can is also connected to this pin.

Table 2: Pin description MLX90614

Note: for +12V (+8…+16V) powered operation refer to the Application information section. For EMC and isothermal conditions reasons it is highly recommended not to use any electrical connection to the metal can except by the VSS pin. With the SCL / Vz and PWM / SDA pins operated in 2-wire interface mode, the input Schmidt trigger function is automatically enabled.



MLX90614 family
Single and Dual Zone
Infra Red Thermometer in TO-39

Melexis
INSPIRED ENGINEERING

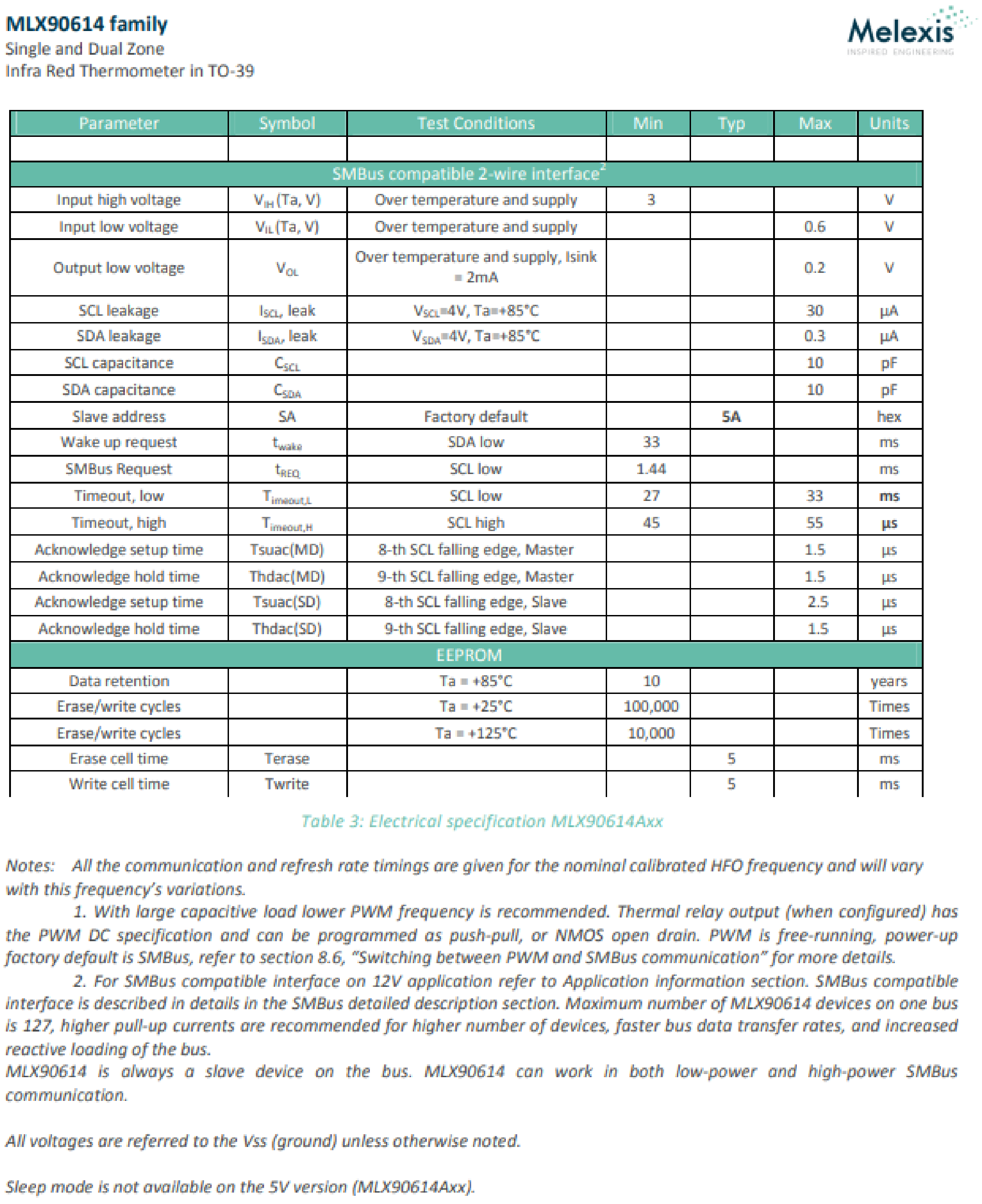
7. Electrical Specifications

7.1. MLX90614Axx

All parameters are valid for T_A = 25 ˚C, V_{DD} =5V (unless otherwise specified)

Parameter	Symbol	Test Conditions	Min	Typ	Max	Units
Supplies						
External supply	V_{DD}		4.5	5	5.5	V
Supply current	I_{DD}	No load		1.3	2	mA
Supply current (programming)	I_{DDpr}	No load, erase/write EEPROM operations		1.5	2.5	mA
Zener voltage	Vz	Iz = 75…1000µA (Ta=room)	5.5	5.7	5.9	V
Zener voltage	Vz(Ta)	Iz = 70…1000µA, full temperature range	5.15	5.7	6.24	V
Power On Reset						
POR level	V_{POR_up}	Power-up (full temp range)	1.4	1.75	1.95	V
POR level	V_{POR_down}	Power –down (full temp range)	1.3	1.7	1.9	V
POR hysteresis	V_{POR_hys}	Full temp range	0.08	0.1	1.15	V
V_{DD} rise time (10% to 90% of specified supply voltage)	T_{POR}	Ensure POR signal			20	ms
Output valid (result in RAM)	Tvalid	After POR		0.25		s
Pulse width modulation[1]						
PWM resolution	PWMres	Data band		10		bit
PWM output period	$PWM_{T,def}$	Factory default, internal oscillator factory calibrated		1.024		ms
PWM period stability	$dPWM_T$	Internal oscillator factory calibrated, over the entire operation range and supply voltage	-10		+10	%
Output high Level	PWM_{HI}	I_{source} = 2 mA	V_{DD}-0.2			V
Output low Level	PWM_{LO}	I_{sink} = 2 mA			V_{SS}+0.2	V
Output drive current	$Idrive_{PWM}$	Vout,H = V_{DD} - 0.8V		7		mA
Output sink current	$Isink_{PWM}$	Vout,L = 0.8V		13.5		mA

Continued on next page



MLX90614 family
Single and Dual Zone
Infra Red Thermometer in TO-39

Melexis
INSPIRED ENGINEERING

Parameter	Symbol	Test Conditions	Min	Typ	Max	Units
SMBus compatible 2-wire interface[2]						
Input high voltage	V_{IH}(Ta, V)	Over temperature and supply	3			V
Input low voltage	V_{IL}(Ta, V)	Over temperature and supply			0.6	V
Output low voltage	V_{OL}	Over temperature and supply, Isink = 2mA			0.2	V
SCL leakage	I_{SCL}, leak	V_{SCL}=4V, Ta=+85°C			30	µA
SDA leakage	I_{SDA}, leak	V_{SDA}=4V, Ta=+85°C			0.3	µA
SCL capacitance	C_{SCL}				10	pF
SDA capacitance	C_{SDA}				10	pF
Slave address	SA	Factory default		**5A**		hex
Wake up request	t_{wake}	SDA low	33			ms
SMBus Request	t_{REQ}	SCL low	1.44			ms
Timeout, low	$T_{imeout,L}$	SCL low	27		33	**ms**
Timeout, high	$T_{imeout,H}$	SCL high	45		55	**µs**
Acknowledge setup time	Tsuac(MD)	8-th SCL falling edge, Master			1.5	µs
Acknowledge hold time	Thdac(MD)	9-th SCL falling edge, Master			1.5	µs
Acknowledge setup time	Tsuac(SD)	8-th SCL falling edge, Slave			2.5	µs
Acknowledge hold time	Thdac(SD)	9-th SCL falling edge, Slave			1.5	µs
EEPROM						
Data retention		Ta = +85°C	10			years
Erase/write cycles		Ta = +25°C	100,000			Times
Erase/write cycles		Ta = +125°C	10,000			Times
Erase cell time	Terase			5		ms
Write cell time	Twrite			5		ms

Table 3: Electrical specification MLX90614Axx

Notes: All the communication and refresh rate timings are given for the nominal calibrated HFO frequency and will vary with this frequency's variations.

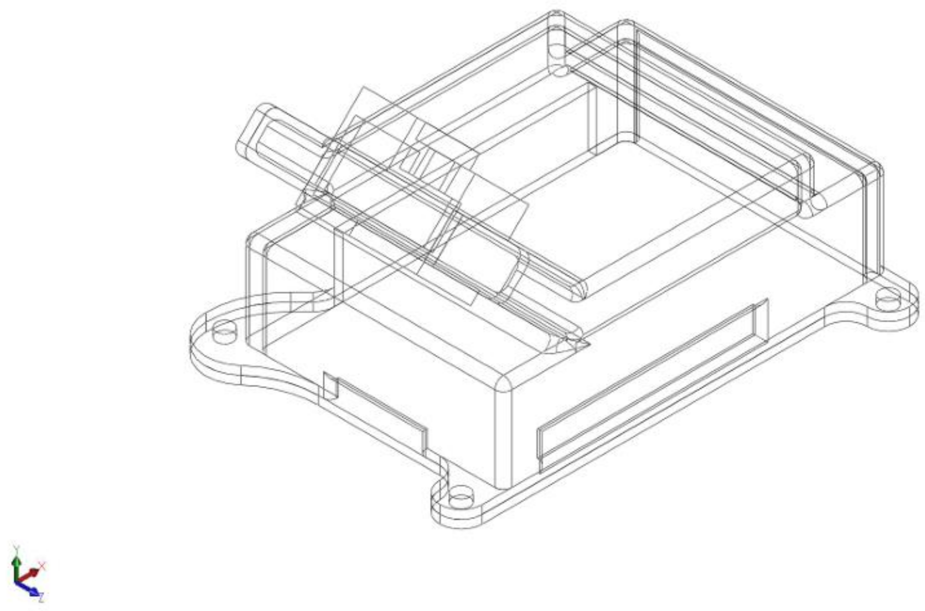
1. With large capacitive load lower PWM frequency is recommended. Thermal relay output (when configured) has the PWM DC specification and can be programmed as push-pull, or NMOS open drain. PWM is free-running, power-up factory default is SMBus, refer to section 8.6, "Switching between PWM and SMBus communication" for more details.

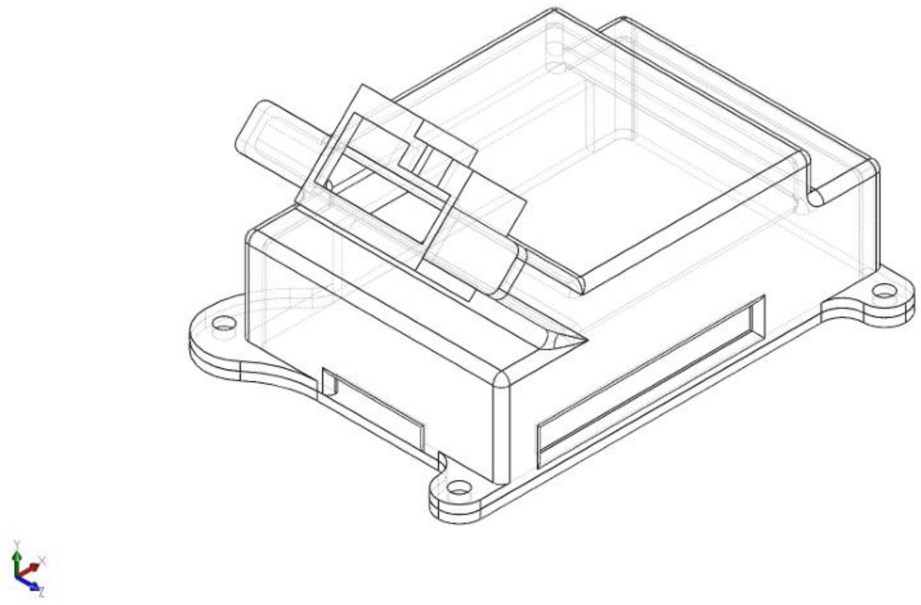
2. For SMBus compatible interface on 12V application refer to Application information section. SMBus compatible interface is described in details in the SMBus detailed description section. Maximum number of MLX90614 devices on one bus is 127, higher pull-up currents are recommended for higher number of devices, faster bus data transfer rates, and increased reactive loading of the bus.
MLX90614 is always a slave device on the bus. MLX90614 can work in both low-power and high-power SMBus communication.

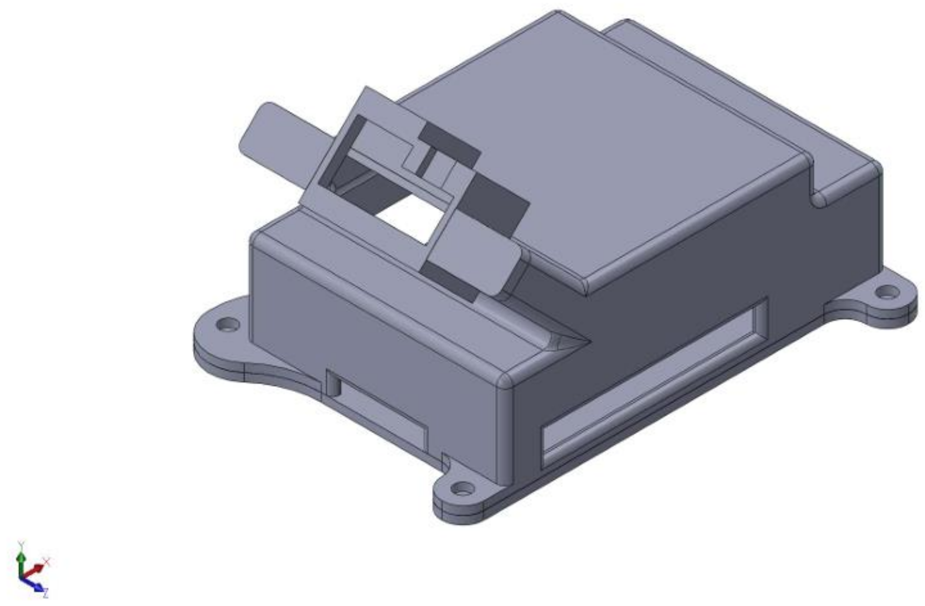
All voltages are referred to the Vss (ground) unless otherwise noted.

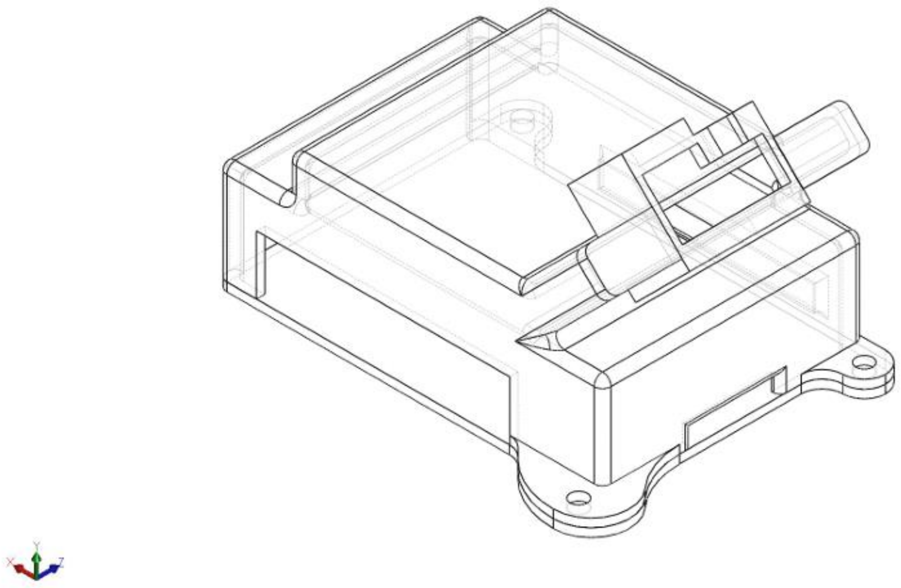
Sleep mode is not available on the 5V version (MLX90614Axx).

3D-Printed Raspberry Pi 4B Case Full Design

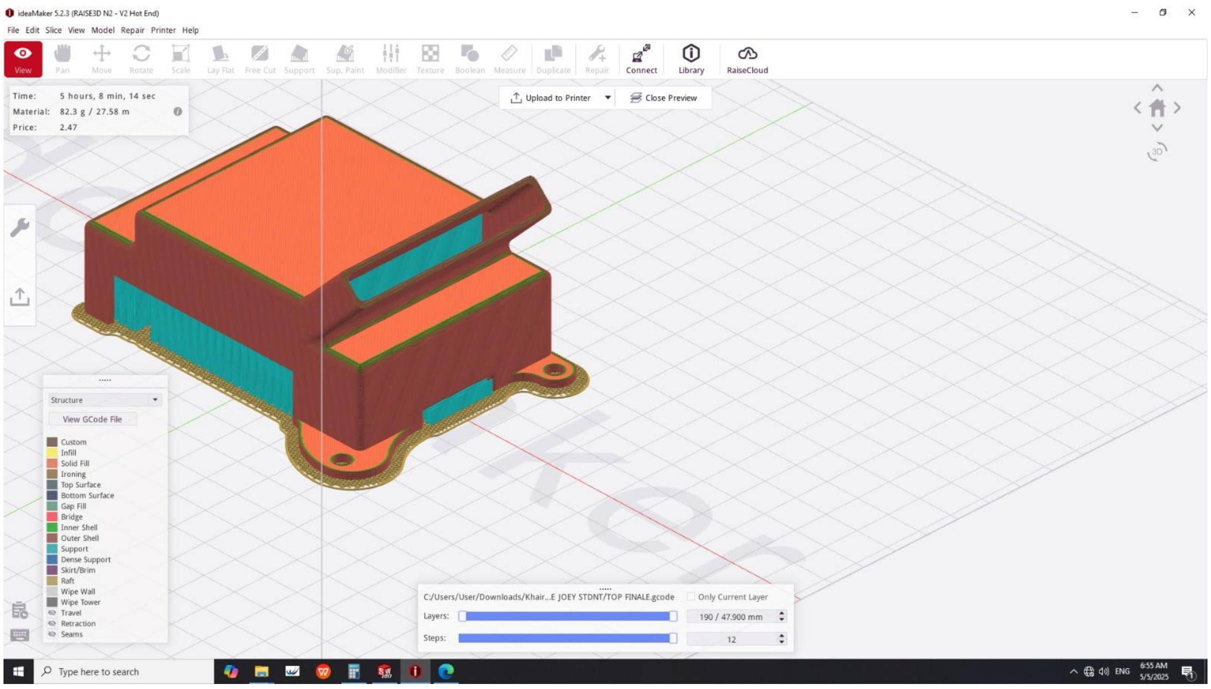






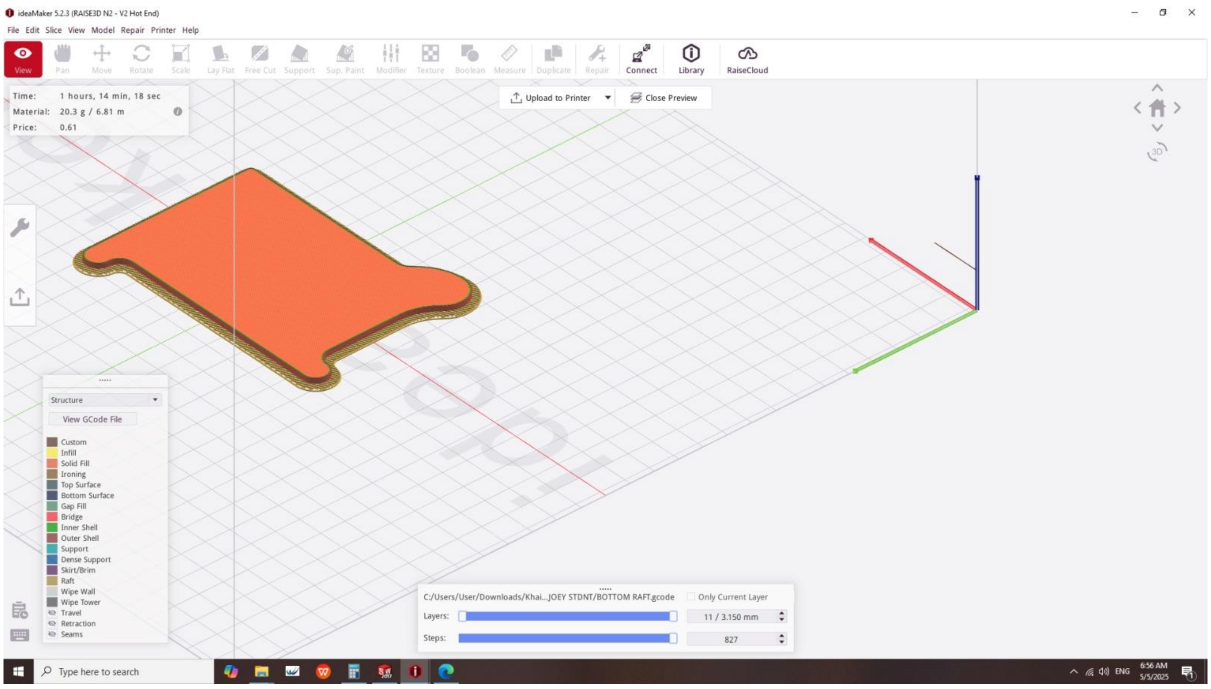


Appendix



Time: 5 hours, 8 min, 14 sec
Material: 82.3 g / 27.58 m
Price: 2.47

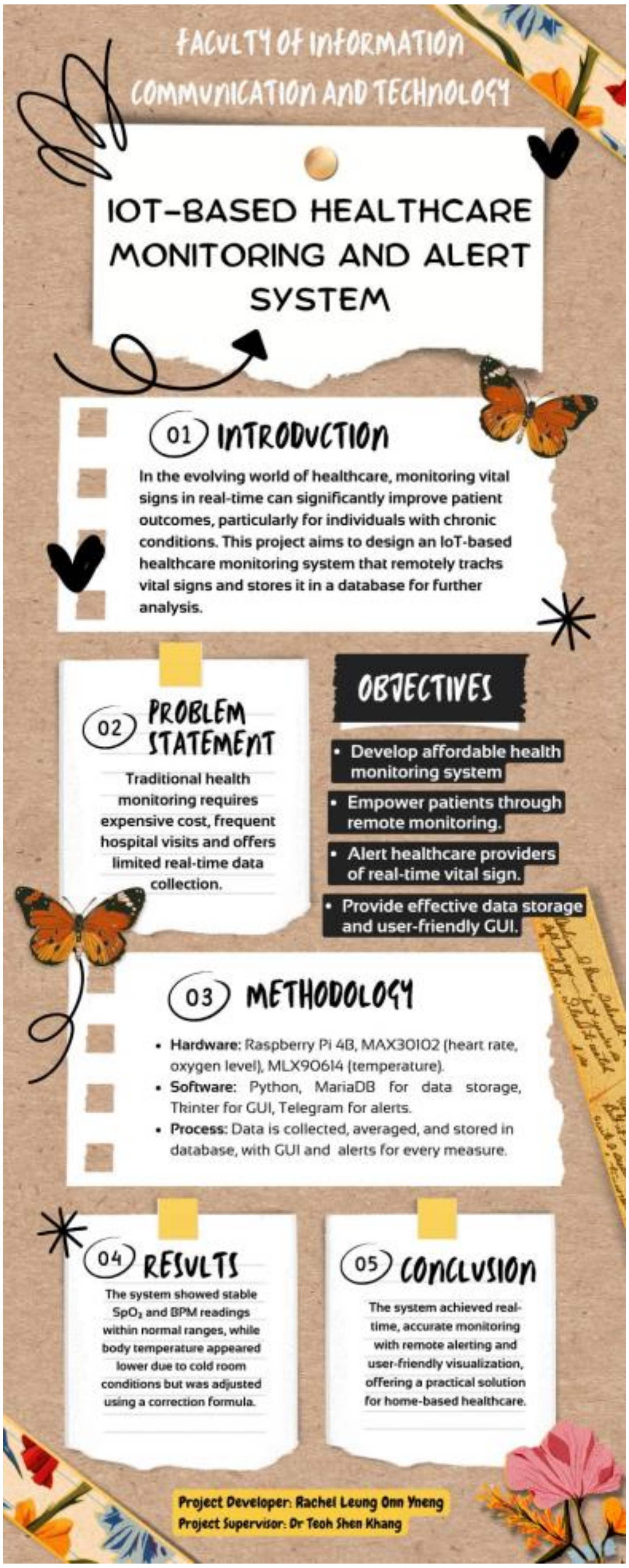
C:/Users/User/Downloads/Khair...E JOEY STDNT/TOP FINALE.gcode
Only Current Layer
Layers: 190 / 47.900 mm
Steps: 12



Time: 1 hours, 14 min, 18 sec
Material: 20.3 g / 6.81 m
Price: 0.61

C:/Users/User/Downloads/Khai...JOEY STDNT/BOTTOM RAFT.gcode
Only Current Layer
Layers: 11 / 3.150 mm
Steps: 827

POSTER



FACULTY OF INFORMATION COMMUNICATION AND TECHNOLOGY

IOT-BASED HEALTHCARE MONITORING AND ALERT SYSTEM

01 INTRODUCTION

In the evolving world of healthcare, monitoring vital signs in real-time can significantly improve patient outcomes, particularly for individuals with chronic conditions. This project aims to design an IoT-based healthcare monitoring system that remotely tracks vital signs and stores it in a database for further analysis.

02 PROBLEM STATEMENT

Traditional health monitoring requires expensive cost, frequent hospital visits and offers limited real-time data collection.

OBJECTIVES

- Develop affordable health monitoring system
- Empower patients through remote monitoring.
- Alert healthcare providers of real-time vital sign.
- Provide effective data storage and user-friendly GUI.

03 METHODOLOGY

- **Hardware:** Raspberry Pi 4B, MAX30102 (heart rate, oxygen level), MLX90614 (temperature).
- **Software:** Python, MariaDB for data storage, Tkinter for GUI, Telegram for alerts.
- **Process:** Data is collected, averaged, and stored in database, with GUI and alerts for every measure.

04 RESULTS

The system showed stable SpO_2 and BPM readings within normal ranges, while body temperature appeared lower due to cold room conditions but was adjusted using a correction formula.

05 CONCLUSION

The system achieved real-time, accurate monitoring with remote alerting and user-friendly visualization, offering a practical solution for home-based healthcare.

Project Developer: Rachel Leung Onn Yneng
Project Supervisor: Dr Teoh Shen Khang