MECPS 25' Capstone Project Report

Advanced Elderly Care System using Amazon Sidewalk over LoRa

Prepared by,
Hoang Lan Pham
Royston Pinto
Rudrashis Gorai

Advisor,
Prof. Hung Cao

Graduate Mentor,
Mohamed Benomar El Kati

1. Project Overview

1.1 Introduction

Falls represent one of the most serious health risks facing older adults today. According to the Centers for Disease Control and Prevention, falls are the leading cause of injury-related death among adults aged 65 and older, claiming approximately 38,000 lives in 2021 alone [1]. The World Health Organization reports that adults over 60 experience 37.3 million falls annually that require medical attention [2]. These statistics become particularly concerning when considering that 16.2 million elderly adults in the United States live alone, with research indicating a 73% lifetime fall rate among this population [3]. Furthermore, 79% of fall-related emergency visits occur within the home environment, highlighting the critical need for continuous monitoring solutions [4].

Traditional wearable health devices rely primarily on WiFi or Bluetooth connectivity, which presents significant limitations. These technologies offer restricted range, typically confining reliable coverage to within residential structures. Elderly users who venture into yards, neighborhoods, or community spaces often lose connectivity precisely when monitoring matters most. This connectivity gap creates dangerous blind spots where falls or cardiac events may go undetected for extended periods.

1.2 Proposed Solution

The Advanced Elderly Care System addresses these limitations through a wearable device that leverages Amazon Sidewalk over LoRa (Long Range) radio technology. This combination provides long-range, low-power connectivity that extends well beyond traditional wireless boundaries while maintaining minimal energy consumption suitable for continuous wearable operation.

Amazon Sidewalk operates as a free community mesh network utilizing existing Amazon devices such as Echo speakers and Ring security cameras as bridges [5]. These bridges route low-bandwidth data from Sidewalk-enabled devices to AWS cloud services without requiring additional infrastructure investment or monthly subscription fees. The network currently covers over 90% of the United States population, with coverage continuing to expand as more Amazon devices join the network [6].

LoRa technology operates in the license-free 915 MHz frequency band and can achieve transmission ranges up to 10 kilometers in optimal conditions, with typical suburban ranges of 500-800 meters per bridge device [7]. When combined with Amazon Sidewalk's mesh architecture, this enables seamless connectivity across entire neighborhoods without requiring dedicated gateways or cellular infrastructure.

1.3 System Architecture

The system architecture consists of four primary components working in coordination:

The wearable endpoint collects sensor data including heart rate, blood oxygen saturation (SpO2), motion patterns for fall detection, and GPS coordinates for location tracking. This device is built around the Nordic nRF52840 microcontroller paired with a Semtech SX1262 LoRa transceiver, enabling both Bluetooth Low Energy for initial device commissioning and LoRa for long-range data transmission.

Amazon Sidewalk bridges, implemented through Echo Dot and Ring devices, receive LoRa transmissions from the wearable and route data through the Sidewalk network. The device initially connects via Bluetooth to complete time synchronization and provisioning, then automatically switches to LoRa for regular operation, optimizing both range and power consumption.

AWS IoT Core receives sensor payloads via MQTT topics, where Lambda functions process incoming data, store measurements in DynamoDB, and evaluate alert conditions. The cloud infrastructure handles message routing, data persistence, and notification triggering for emergency events.

A web-based monitoring dashboard provides caregivers with real-time visibility into vital signs, fall detection notifications, and location tracking. The interface displays heart rate trends, SpO2 levels, device status, and geographic position on an interactive map, enabling remote oversight of multiple elderly users from a single platform.

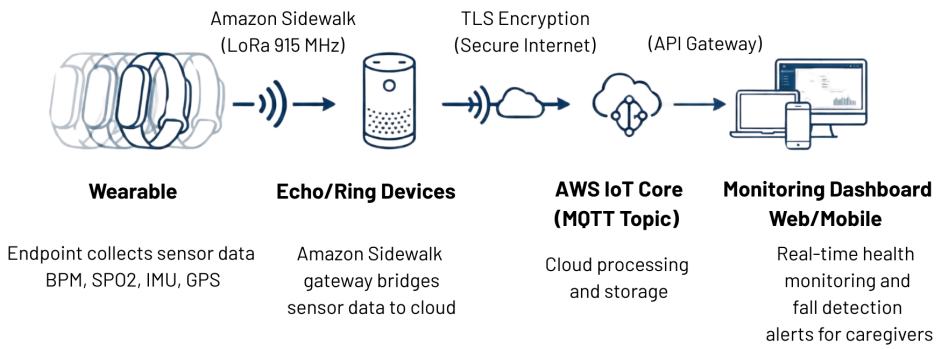

Figure 1. System Block Diagram

1.4 Core Capabilities

The system provides three categories of health monitoring. Vital signs monitoring uses a MAX30102 photoplethysmography sensor to continuously measure heart rate and blood oxygen saturation, with temperature sensing providing additional health indicators. Readings are sampled every five seconds and transmitted to the cloud for trend analysis and threshold-based alerting.

Fall detection employs a three-phase algorithm processing data from an MPU6050 inertial measurement unit sampling at 50 Hz. The algorithm detects free-fall conditions (acceleration below 0.35g), followed by impact events (acceleration exceeding 2.4g combined with rotation above 240 degrees per second), and finally confirms falls through post-impact stillness detection over a three-second window. Confirmed falls trigger immediate emergency alerts with retry logic ensuring delivery.

Location tracking utilizes a NEO-6M GPS module to provide real-time coordinates, enabling caregivers to locate users who may have wandered or become disoriented. This capability proves particularly valuable for individuals with dementia or cognitive decline who may leave designated safe areas.

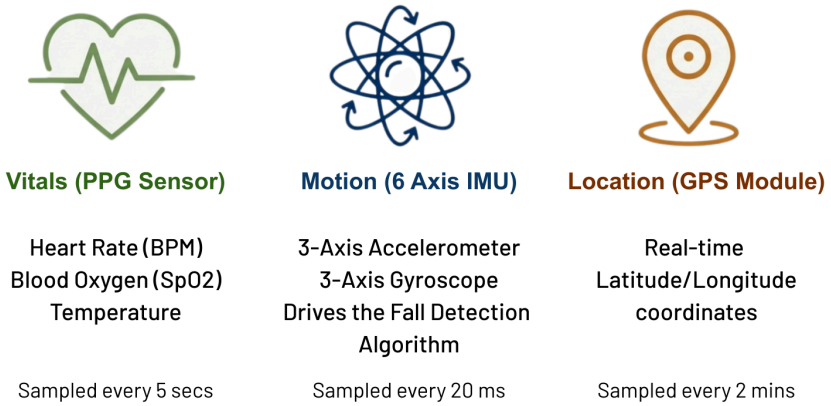

Figure 2. Monitoring Data

1.5 Communication Strategy

The system implements differentiated message handling based on urgency. Regular status messages transmit every five minutes using fire-and-forget delivery, providing baseline health monitoring without network overhead. Emergency alerts triggered by falls or abnormal vital signs utilize a ring buffer with retry logic, ensuring critical notifications reach caregivers even under challenging network conditions. Help requests initiated by a panic button receive identical priority handling to emergency alerts.

Amazon Sidewalk provides three layers of security: application-layer end-to-end encryption, network-layer encryption, and flex-layer bridge encryption [8]. This architecture ensures that health data remains protected throughout transmission, with bridges unable to access payload contents.

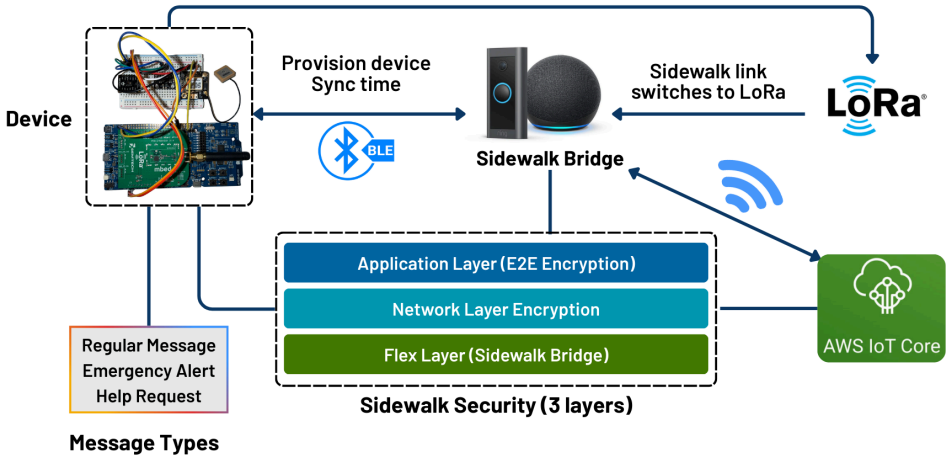

Figure 3. Data Communication Strategy

1.6 Significance

This project demonstrates a practical application of emerging IoT connectivity technologies to address genuine public health challenges. By eliminating infrastructure costs and monthly fees while extending coverage beyond residential boundaries, the system offers a scalable approach to elderly care monitoring. The solution becomes increasingly effective as Amazon Sidewalk coverage expands, providing a foundation for broader deployment without requiring dedicated network infrastructure investment.

2. Objectives

2.1 Project Goals

The primary goal of this capstone project is to develop a working proof-of-concept wearable health monitoring system that uses Amazon Sidewalk over LoRa for long-range connectivity. The system aims to provide continuous health monitoring for elderly users with automatic emergency alerts to caregivers.

Main Objectives:

- Build a wearable device that can send and receive data from AWS IoT using Amazon Sidewalk over LoRa
- Monitor heart rate continuously with automatic detection of abnormal readings
- Detect critical falls and automatically notify caregivers
- Track real-time location using GPS
- Create a web-based dashboard for caregivers to view health data and receive alerts
- Generate periodic health trend reports

2.2 Original Fall Quarter Objectives

At the start of Fall Quarter, the team defined the following goals:

Sensor Integration

- Read data from all sensors including heart rate (PPG), accelerometer, and gyroscope modules
- Process raw sensor data into structured information suitable for transmission

Communication

- Establish reliable communication between the LoRa module and Amazon Sidewalk network
- Send test packets through LoRa to confirm connectivity with Amazon Sidewalk
- Verify two-way data transmission between device and cloud

Hardware Design

- Design a detailed schematic for the wearable band
- Explore feasibility of creating custom flexible PCBs

Software

- Create a web-based dashboard to display real-time sensor data and alerts
- Implement data processing algorithms for sensor readings

2.3 Updated Objectives (Revised During Fall Quarter)

Based on lessons learned during Spring Quarter prototyping, the team revised objectives to focus on achievable milestones:

Communication (Priority 1)

- Establish stable end-to-end communication from device to Amazon Sidewalk over LoRa
- Confirm bidirectional data transmission (device-to-cloud and cloud-to-device)
- Implement automatic link switching from BLE to LoRa after device registration

Sensor Data (Priority 2)

- Read and process raw data from all connected sensors:
    - Heart rate and SpO2 from MAX30102 PPG sensor
    - Motion data from MPU6050 accelerometer/gyroscope
    - Location from NEO-6M GPS module
- Transmit processed sensor data to AWS IoT Core
- Implement three-phase fall detection algorithm

System Integration (Priority 3)

- Deliver a proof-of-concept prototype showing full data flow from device to cloud
- Deploy AWS IoT sample application for data visualization
- Implement message handling for three types: Regular, Emergency, and Help

Deferred Items

- Custom flexible PCB design moved to future work due to time constraints
- Focus shifted to functional integration using existing development boards
- PCB schematic and CAD model prepared for future iterations

2.4 Detailed Technical Objectives

2.4.1 Firmware Objectives

Objective	Description
PPG sensor sampling	Read heart rate and SpO2 every 5 seconds via I2C
IMU sensor sampling	Sample accelerometer/gyroscope at 50 Hz (20 ms)
GPS location sampling	Read coordinates every 2 minutes via UART

Regular message transmission	Send health data to cloud every 5 minutes
Emergency alert handling	Trigger on fall detection or abnormal vitals
Help request button	Send immediate alert on BTN1 short press
Three-phase fall detection	Detect free-fall, impact, and post-fall stillness
Vital signs thresholds	Alert when SpO2 < 90% or BPM < 40 or > 180
Automatic LoRa switching	Switch from BLE to LoRa after registration
Critical message buffer	Ring buffer with retry logic for emergencies
Message retry logic	Retry failed emergency/help messages up to 3 times
JSON payload encoding	Format sensor data as compact JSON (max 200 bytes)
I2C bus coordination	Mutex-protected access for MPU6050 and SEN0344
Sidewalk state management	Handle registration, time sync, and link status

2.4.2 Cloud/Backend Objectives

Objective	**Description**
AWS IoT Core integration	Route MQTT messages from Sidewalk
Uplink Lambda	Process incoming sensor payloads
Downlink Lambda	Send commands to device
DynamoDB tables	Store devices, measurements, and alerts
API Gateway	REST endpoints for dashboard access
CloudFront distribution	Host and serve web dashboard
CloudWatch monitoring	Log Lambda invocations and errors
CloudFormation deployment	Infrastructure as code for reproducibility

2.4.3 Dashboard Objectives

Objective	Description
Real-time vitals display	Show heart rate and SpO2 with trend charts
Fall detection alerts	Display notifications with severity level
Location tracking	Show last known position on interactive map
Device onboarding	Register devices using Elder Band ID
Multi-device view	Monitor multiple users from single dashboard
Device status indicator	Show online/offline and last seen time
Engage/disengage control	Enable or disable device monitoring

2.5 Success Criteria

The project is considered successful based on the following criteria:

Must Have (All Achieved)

- Device registers with Amazon Sidewalk network via BLE
- Device automatically switches to LoRa after registration
- Sensor data transmits to AWS IoT Core successfully
- Dashboard displays real-time heart rate and SpO2 readings
- Fall detection algorithm triggers emergency alerts
- Help button sends immediate alert to cloud
- End-to-end message latency under 5 seconds

Should Have (All Achieved)

- Automatic BLE-to-LoRa link switching works reliably

- Critical message buffer retries failed emergency/help messages
- GPS location displays on map in dashboard
- Regular messages send every 5 minutes without failure
- Vital sign thresholds trigger alerts (SpO2 < 90%, BPM < 40 or > 180)
- CloudWatch logs all Lambda invocations for debugging
- Bidirectional communication works (cloud can send commands to device)

Nice to Have (Partially Achieved)

- Overnight stability test with 100% packet delivery — Achieved
- Multiple device support in dashboard — Achieved
- Infrastructure deployed via CloudFormation — Achieved
- Power consumption measurement — Deferred to custom hardware phase
- Custom PCB fabrication — Fabrication deferred

3. Setup Details for Project

3.1 Development Environment

IDE and Tools

- Visual Studio Code with nRF Connect Extension Pack
- nRF Connect for Desktop (Toolchain Manager, Programmer)
- nRF RTT Terminal for debug output (UART reserved for GPS module)
- AWS CLI for cloud resource management
- Python 3.6+ for device provisioning scripts

SDK Versions

- nRF Connect SDK v2.7.0
- Amazon Sidewalk SDK v2.7.0
- Zephyr RTOS (included with nRF Connect SDK)

3.2 Hardware Components

Core Controller

Component	Specification
Board	Nordic nRF52840 Development Kit
Processor	ARM Cortex-M4F @ 64 MHz
Flash	1 MB
RAM	256 KB
Wireless	Bluetooth 5.0, IEEE 802.15.4

LoRa to Sidewalk Communication

Component	Specification
Module	Semtech SX1262 Evaluation Board
Frequency	Sub-GHz 915 MHz (US license-free band)
Flash	1 MB
Interface	SPI
Features	Long-range, low-power transmission
Gateway	Amazon Echo Dot 5th Generation

Sensors

Sensor	Model	Interface	Purpose
PPG	DFRobot MAX30102 v2.0	I2C (0x57)	Heart rate, SpO2, temperature
IMU	MPU6050	I2C (0x68)	3-axis accelerometer, 3-axis gyroscope
GPS	NEO-6M	UART (9600 baud)	Latitude, longitude, altitude

Note on DFRobot MAX30102 v2.0: This sensor version includes an onboard MCU that pre-processes raw PPG data. The I2C output provides filtered heart rate, SpO2, and temperature values rather than raw sensor readings. This requires a different I2C register protocol compared to the standard MAX30102 breakout boards.

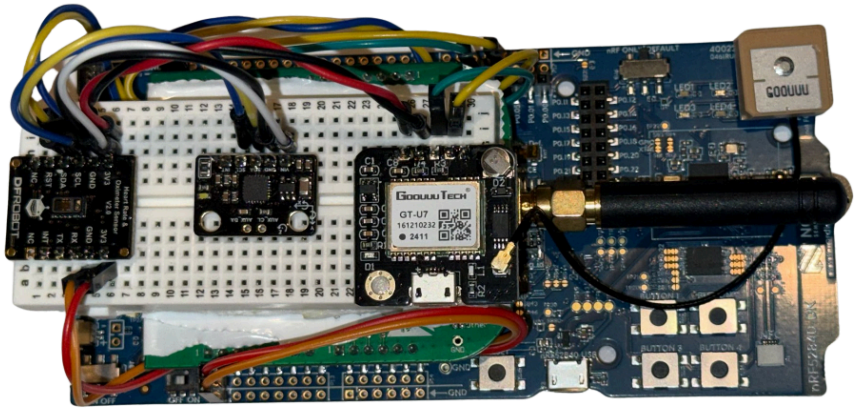

Figure 4. Assembly of Hardware Components

3.3 Pin Configuration

I2C Bus (Sensors)

Pin	Function	Connected To
P0.26	SDA	MAX30102 (0x57), MPU6050 (0x68)
P0.27	SCL	

UART (GPS Module)

Pin	Function	Connected To
P0.08	RX	NEO-6M TX
P0.06	TX	NEO-6M RX

SPI (LoRa Radio)

The Semtech SX1262 evaluation board attaches directly onto the nRF52840 DK via the Arduino header pins.

Pin	Function	Connected To
P1.13	MOSI	SX1262 MOSI
P1.14	MISO	SX1262 MISO
P1.15	SCK	SX1262 SCK

Status LEDs

LED	Pin	Alias	Purpose
LED1	P0.13	state-notifier-connected	Sidewalk connected
LED2	P0.14	state-notifier-time-sync	Time synchronized
LED3	P0.15	state-notifier-registered	Device registered
LED4	P0.16	state-notifier-working	System working

Button

Button	Pin	Purpose
BTN1	P0.11	Help request (panic button)

3.4 Software Stack

Firmware

Layer	Component	Purpose
Application	sensors/app.c	Main logic, callbacks, button handling
Sensor Manager	sensor_manager.c	Coordinate all sensor handlers
GPS Handler	gps_handler.c	Parse NMEA data from NEO-6M
IMU Handler	mpu6050_handler.c	Fall detection algorithm
PPG Handler	sen0344_handler.c	Heart rate and SpO2 via DFRobot protocol
Message Buffer	critical_msg_buffer.c	Ring buffer for emergency/help messages
Sidewalk	sidewalk.c	State machine, link management
RTOS	Zephyr	Threads, mutexes, work queues, timers

Cloud Services (AWS)

Service	Purpose
IoT Core	MQTT message routing from Sidewalk
Lambda	Uplink processing, downlink commands, database operations
DynamoDB	Store devices, measurements, alerts
API Gateway	REST endpoints for dashboard
S3	Host static web dashboard files
CloudFront	CDN for dashboard distribution
CloudFormation	Infrastructure as code deployment
CloudWatch	Logging and monitoring

Dashboard

Technology	Purpose
React	Frontend framework
Vite	Build tool and dev server
Tailwind CSS	Styling
Leaflet	Interactive maps for location tracking

3.5 AWS Setup and Device Provisioning

Step 1: AWS Account Setup

1. Sign up for an AWS account at aws.amazon.com
2. Create an Administrative User in IAM
3. Create an Access Key for the admin user
4. Install Python 3.6+ on your development machine
5. Install and configure AWS CLI with the access key

Step 2: Create Device Profile

Create a device profile using the AWS CLI:

```
aws iotwireless create-device-profile --name sidewalk_profile --sidewalk {}
```

Save the returned Id for the next step.

Step 3: Create Wireless Device

Create a Sidewalk end device linked to your device profile:

```
aws iotwireless create-wireless-device \
```

```
    --type "Sidewalk" \
    --name "sidewalk_device" \
    --destination-name "SidewalkDestination" \
    --sidewalk DeviceProfileId="<your-device-profile-id>"
```

Note: Create a separate wireless device for each physical endpoint.

Step 4: Export Device Information

Store device profile and wireless device information as JSON files:

```
aws iotwireless get-device-profile \
    --id "<device-profile-id>" > device_profile.json

aws iotwireless get-wireless-device \
    --identifier-type WirelessDeviceId \
    --identifier "<wireless-device-id>" > wireless_device.json
```

Step 5: Generate Provisioning Binary

Use the provision.py tool from the Sidewalk SDK (located in nRF Connect SDK v2.7.0):

```
python3 provision.py nordic aws \
    --output_bin mfg.bin \
    --wireless_device_json wireless_device.json \
    --device_profile_json device_profile.json \
    --addr 0xFF000
```

This generates nordic_aws_nrf52840.hex which contains the device credentials.

3.6 Build and Flash Procedure

Step 1: Configure Build Options

Add the following to prj.conf or create overlay-sensors.conf:

```
CONFIG_SID_END_DEVICE_SENSOR_MONITORING=y
CONFIG_SIDEWALK_SUBGHZ_SUPPORT=y
CONFIG_I2C=y
```

```
CONFIG_SENSOR=y
CONFIG_FPU=y
CONFIG_SIDEWALK_THREAD_STACK_SIZE=10240
CONFIG_HEAP_MEM_POOL_SIZE=8192
CONFIG_SID_END_DEVICE_AUTO_START=y
CONFIG_SID_END_DEVICE_AUTO_CONN_REQ=y
```

Step 2: Build the Firmware

```
west build -b nrf52840dk/nrf52840 samples/sid_end_device --
-DOVERLAY_CONFIG="overlay-sensors.conf"
```

Step 3: Flash the Device

Use the Programmer tool in nRF Connect for Desktop to flash both files simultaneously:

- nordic_aws_nrf52840.hex (provisioning credentials from Step 5 in Section 3.5)
- merged.hex (application firmware from build folder)

Important: Flash both hex files together to ensure the device has both the application and AWS credentials.

3.7 Debug Output

Since UART0 is used by the NEO-6M GPS module, debug output uses Segger RTT (Real-Time Transfer) integrated in Visual Studio Code.

Viewing Debug Logs

- Use nRF RTT Terminal extension in VS Code
- Connect to the device to view real-time log output

Important: Close the RTT Terminal before flashing new code. Keeping it open during flash will cause errors.

Log Categories

```
<inf> app: - Application-level messages
<inf> sidewalk: - Sidewalk stack status
```

```
<inf> sensor_manager: - Sensor coordination
<wrn> - Warnings
<err> - Errors
```

4. Standards Used

This section documents the technical standards, protocols, and best practices applied throughout the Advanced Elderly Care System development. Adherence to established standards ensures reliability, interoperability, security, and maintainability.

4.1 Communication Protocol Standards

4.1.1 I2C (Inter-Integrated Circuit) - Sensor Communication

The system uses I2C for all on-board sensor communication.

Implementation Details:

- **Bus Configuration:** Single I2C0 bus at 400 kHz (Fast Mode)
- **Addressing:** 7-bit addressing scheme
    - MAX30102 PPG sensor: 0x57 (DFRobot v2.0 variant)
    - MPU6050 IMU: 0x68 (AD0 pin grounded)
- **Pull-up Resistors:** 4.7kΩ external pull-ups on SDA/SCL lines
- **Bus Arbitration:** Mutex-protected access prevents collisions between sensor handlers

Bus Coordination Pattern:

```
sensor_manager_i2c_lock();    // Acquire mutex
i2c_write_read(dev, ...);     // Perform I2C transaction
sensor_manager_i2c_unlock();  // Release mutex
```

4.1.2 UART (Universal Asynchronous Receiver-Transmitter) - GPS Communication

GPS module communication follows standard asynchronous serial protocol.

Implementation Details:

- **Baud Rate:** 9600 bps (NEO-6M default)
- **Data Format:** 8N1 (8 data bits, no parity, 1 stop bit)
- **Flow Control:** None
- **Protocol:** NMEA 0183 sentence format

NMEA Sentence Types Parsed:

- $GPGGA - Global Positioning System Fix Data (primary position source)
- $GPRMC - Recommended Minimum Specific GPS Data (speed, bearing)
- $GPGSV - Satellites in View (signal quality assessment)

4.1.3 SPI (Serial Peripheral Interface) - LoRa Radio Communication

The SX1262 LoRa transceiver interfaces via SPI following Motorola SPI Mode 0.

Implementation Details:

- **Clock Speed:** 8 MHz
- **Mode:** CPOL=0, CPHA=0 (Mode 0)
- **Chip Select:** Active-low, directly controlled by Sidewalk stack
- **Data Order:** MSB first

4.1.4 LoRa Physical Layer - Sub-GHz Radio

The system operates on the 915 MHz ISM band following LoRa Alliance specifications.

RF Parameters:

- **Frequency Band:** 902-928 MHz (US ISM band, FCC Part 15)
- **Modulation:** LoRa chirp spread spectrum (CSS)
- **Spreading Factor:** Adaptive (SF7-SF12), managed by Sidewalk protocol
- **Bandwidth:** 125 kHz / 500 kHz (protocol-determined)
- **Maximum TX Power:** +20 dBm (100 mW)
- **Typical Range:** 500-800m per Sidewalk bridge (urban), up to 10km line-of-sight

Regulatory Compliance:

- FCC CFR 47 Part 15.247 (frequency hopping spread spectrum)
- Duty cycle limitations enforced by Amazon Sidewalk protocol

4.1.5 Amazon Sidewalk Protocol Stack

Amazon Sidewalk implements a proprietary protocol stack optimized for low-power IoT devices.

Protocol Layers:

Layer	Function	Implementation
Application	End-to-end encryption, payload handling	AES-128-GCM
Network	Routing, device addressing	Sidewalk Network Server
Flex	Bridge communication, link adaptation	Echo/Ring devices
Physical	BLE (registration), LoRa (data)	nRF52840 + SX1262

Link Types Used:

- **SID_LINK_TYPE_1 (BLE):** Initial device registration, time synchronization
- **SID_LINK_TYPE_3 (LoRa):** Primary data communication after registration

Key Protocol Features:

- Automatic link switching based on connectivity conditions
- Message acknowledgment with configurable retry policies
- Time synchronization via Sidewalk Network Server
- Device authentication using pre-provisioned credentials

4.1.6 MQTT (Message Queuing Telemetry Transport)

Cloud communication follows MQTT v3.1.1 specification (OASIS Standard).

Implementation Details:

- **Broker:** AWS IoT Core (managed MQTT broker)
- **QoS (Quality of Service) Levels Used:**
    - QoS 0 (At most once): Regular status messages
    - QoS 1 (At least once): Emergency and Help messages
- **Payload Format:** JSON-encoded ASCII (max 200 bytes)

Message Types and Handling:

Type	Code	QoS	Retry	Use Case
Regular	R'	0	None	Periodic health reports
Emergency	E'	1	Up to 3	Fall detection, abnormal vitals
Help	H'	1	Up to 3	User panic button

4.2 Data Format Standards

4.2.1 JSON Payload Structure

All sensor data follows a consistent JSON schema for cloud processing.

Payload Schema:

```
{
  "t": "R",           // Message type: R/E/H
  "d": 1,             // Device ID (1-65535)
  "f": "N",           // Fall state: N/F/I
  "b": 72,            // Heart rate (BPM)
  "s": 98,            // SpO2 (%)
  "T": 3650,          // Temperature (centidegrees C)
  "la": 3364585,      // Latitude × 100000
  "lo": -11784294,    // Longitude × 100000
  "B": 100,           // Battery (%)
  "ts": 18000         // Timestamp (seconds since boot)
}
```

Data Encoding Conventions:

- **Coordinates:** Integer representation (value × 100000) to avoid floating-point
- **Temperature:** Centidegrees Celsius (divide by 100 for °C)
- **Timestamps:** Seconds since device boot (correlate with MQTT receive time for absolute time)
- **Character Encoding:** UTF-8 ASCII

4.2.2 Coordinate System Standards

- **Datum:** WGS 84 (World Geodetic System 1984)
- **Format:** Decimal degrees (DD.DDDDD)
- **Precision:** 5 decimal places (~1.1 meter accuracy)
- **Sign Convention:** Positive latitude = North, Negative longitude = West

4.3 Security Standards

4.3.1 Amazon Sidewalk Security Model

The system implements Amazon Sidewalk's three-layer security architecture.

Security Layers:

1. **Application Layer (End-to-End):**
    - AES-128-GCM encryption
    - Keys provisioned during manufacturing
    - Data encrypted from device to AWS IoT Core
    - Sidewalk bridges cannot decrypt payload
2. **Network Layer:**
    - Separate encryption for routing metadata
    - Prevents network-level attacks
    - Session keys rotated periodically
3. **Flex Layer:**
    - Secures bridge-to-cloud communication

    - TLS 1.2/1.3 for internet transport
    - Certificate-based authentication

Device Authentication:

- Pre-provisioned device credentials (SMSN, AppKey, etc.)
- Stored in flash at address 0xFF000
- Hardware-bound identity (cannot be cloned)

4.3.1 Amazon Sidewalk Security Model

Transport Security:

- TLS 1.2 minimum for all MQTT connections
- X.509 certificate-based device authentication
- AWS Signature Version 4 for API calls

Access Control:

- IAM policies with least-privilege principle
- Resource-based policies on IoT topics
- Lambda execution roles scoped to required services

Data Protection:

- DynamoDB encryption at rest (AWS managed keys)
- S3 bucket encryption for static assets
- CloudFront HTTPS-only distribution

4.4 Software Development Standards

4.4.1 Version Control

Git Workflow:

- Repository hosting: GitHub
- Branching model: Feature branches merged to main
- Commit conventions: Descriptive messages with ticket/feature references
- Code review: Pull request review before merge

4.4.2 Coding Standards

Firmware (C - Zephyr RTOS):

- Zephyr coding style guidelines
- Static analysis via built-in compiler warnings (-Wall -Werror)
- Consistent naming: snake_case for functions and variables
- Header guards and documentation comments

Cloud (Python - AWS Lambda):

- PEP 8 style guide compliance
- Type hints for function signatures
- Docstrings for public functions
- Error handling with appropriate logging

Frontend (JavaScript/React):

- ESLint configuration for consistent style

- Component-based architecture
- Prop validation with TypeScript/PropTypes

4.5 Testing and Quality Standards

4.5.1 Firmware Testing

- Unit testing for critical algorithms (fall detection thresholds)
- Integration testing with hardware-in-the-loop
- Serial logging for runtime verification
- Overnight stability testing for reliability validation

4.5.2 Cloud Testing

- Lambda function unit tests
- API Gateway endpoint validation
- End-to-end message flow verification
- CloudWatch monitoring for production metrics

4.5.3 Dashboard Testing

- Component rendering tests with mock data
- API integration tests
- Cross-browser compatibility verification

4.6 Standards Reference Summary

Category	Standard/Specification	Application
Serial Communication	NXP I2C-bus Specification	Sensor bus
Serial Communication	UART 8N1	GPS interface
Serial Communication	SPI Mode 0	LoRa radio
GPS Data Format	NMEA 0183	Position parsing
Coordinate System	WGS 84	GPS datum
Radio Frequency	FCC Part 15.247	915 MHz operation
LoRa Modulation	Semtech LoRa/CSS	Physical layer
IoT Messaging	MQTT v3.1.1 (OASIS)	Cloud pub/sub
Data Interchange	JSON (RFC 8259)	Payload format
Transport Security	TLS 1.2+	Cloud communication
Encryption	AES-128-GCM	Sidewalk E2E
Cloud Platform	AWS Well-Architected	Infrastructure design
Version Control	Git	Source management
Coding Style	Zephyr Guidelines, PEP 8	Code consistency

5. Prototypes

5.1 Spring Quarter: Initial Exploration

5.1.1 Approach

The team initially attempted to develop firmware directly on compact integrated boards combining nRF52840 and SX1262 LoRa transceivers:

- **XIAO nRF52840** - Seeed Studio compact board
- **RAK4631** - RAKwireless module with integrated LoRa

Rationale: These compact form factors were selected with the eventual wearable device in mind.

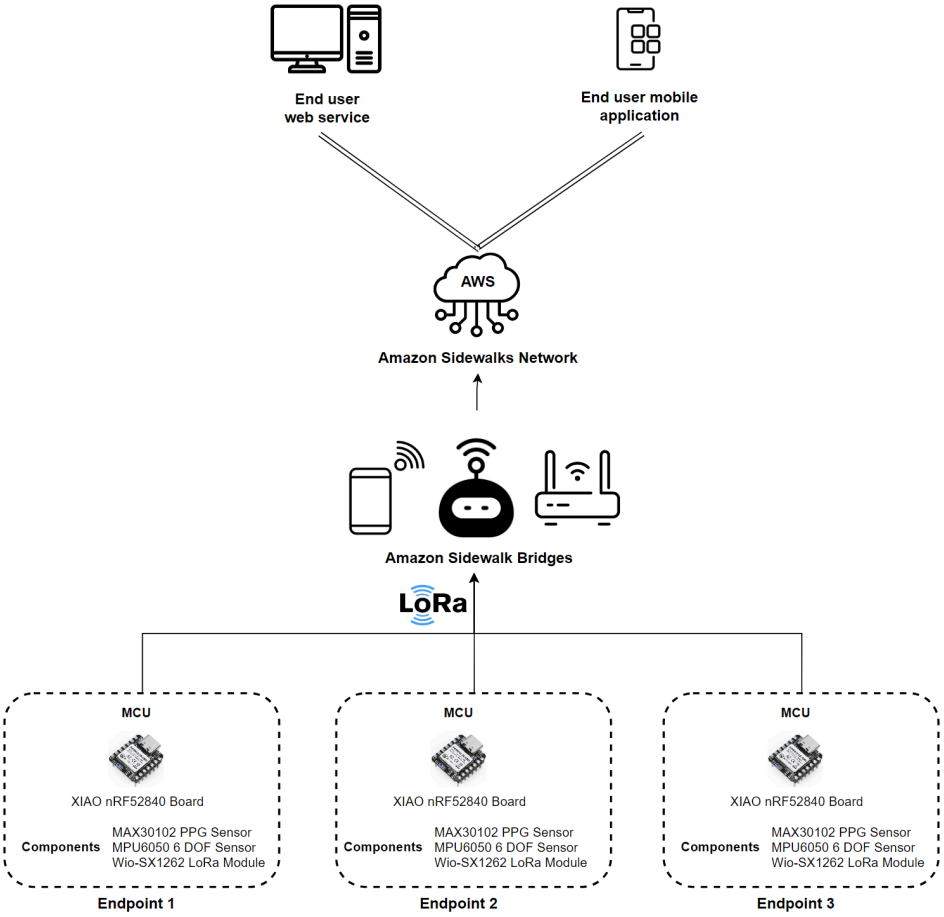

Figure 5. Initial system block diagram

5.1.2 Challenges Encountered

Issue	Board	Impact
Device bricking	XIAO, RAK4631	Frequent reflashing required, significant time lost
Limited Zephyr RTOS familiarity	All	Steep learning curve with SDK
Missing programming accessories	XIAO	Required Expansion Board Base (long shipping delays) or J-Link Debugger
No Sidewalk gateway	All	Could not test LoRa connectivity
Pin remapping failures	nRF52832 DK	Unable to port examples between device families
Memory map misunderstanding	RAK4631	Accidental flash erasure caused bricking

5.1.3 Recovery Attempts

XIAO nRF52840 Unbricking:

- Used nRF52832 DK as external programmer via SWD interface
- Connected SWDIO, SWDCLK, and reset lines
- **Results:** Inconsistent success due to missing expansion base and unstable power

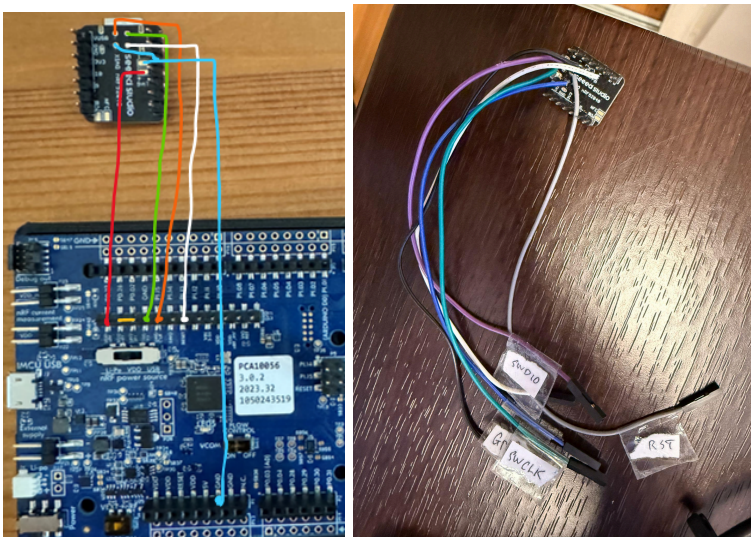

Figure 5. Attempt to unbrick XIAO nRF52840 using SWD with nRF52832 DK

RAK4631 Sidewalk Testing:

- Successfully built and flashed Amazon Sidewalk Sample IoT application
- No network connection observed
- Root cause: Sidewalk gateway required within Bluetooth range for initial registration

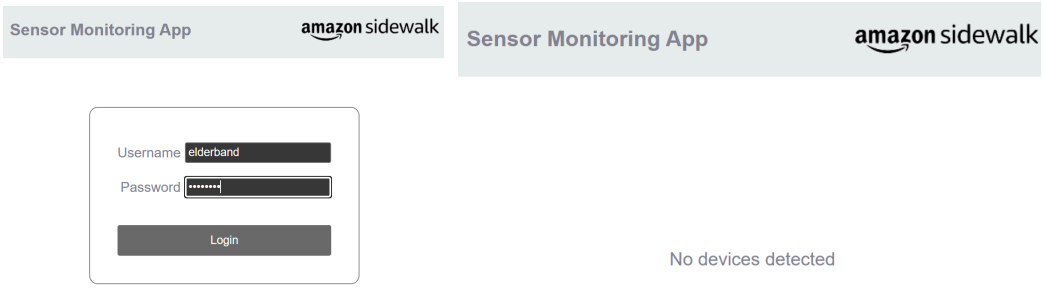

Figure 6. RAK4631 with Sidewalk Sample App - no connection without gateway

5.1.4 Spring Quarter Lessons Learned

1. **Establish Sidewalk connectivity first** - Verified gateway in Bluetooth range is required during bring-up
2. **Use evaluation boards initially** - De-risk RTOS, drivers, and radio stacks before compact modules
3. **Ensure proper accessories** - XIAO expansion base, stable power, reliable SWD access are essential
4. **Validate pin mappings carefully** - Porting between nRF device families requires thorough configuration review

5.1.5 Decision: Pivot to Evaluation Boards

At the end of Spring quarter, the team decided to:

- Pause work on compact modules
- Focus on Nordic nRF52840 DK evaluation boards
- Build proficiency with Zephyr RTOS and Amazon Sidewalk workflow

- Establish working proof-of-concept before miniaturization

5.2 Fall Quarter: Proof-of-Concept Development

5.2.1 Hardware Platform

Based on lessons learned from Spring quarter, the team assembled a complete evaluation system using development boards and breakout modules. This approach prioritized functionality and debuggability over form factor.

Core Component:

Component	Model	Role
Microcontroller	Nordic nRF52840 DK	Main processor, BLE radio
LoRa Transceiver	Semtech SX1262 EVB	915 MHz Sub-GHz communication
PPG Sensor	DFRobot MAX30102 v2.0	Heart rate, SpO2, temperature
IMU	MPU6050	Accelerometer, gyroscope (fall detection)
GPS	NEO-6M	Location tracking
Sidewalk Gateway	Amazon Echo Dot 5th Gen	SubG-CSS bridge to AWS

The SX1262 evaluation board stacks directly onto the nRF52840 DK via Arduino headers. Sensors connect via I2C (MAX30102 and MPU6050 sharing the bus) and UART (NEO-6M GPS). This configuration allowed rapid iteration while maintaining clear signal paths for debugging.

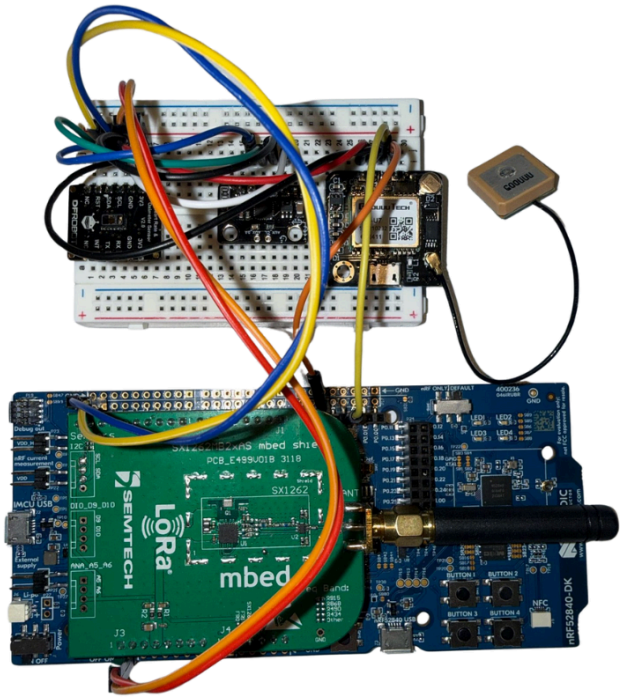

Figure 7. Fall quarter test setup with all components

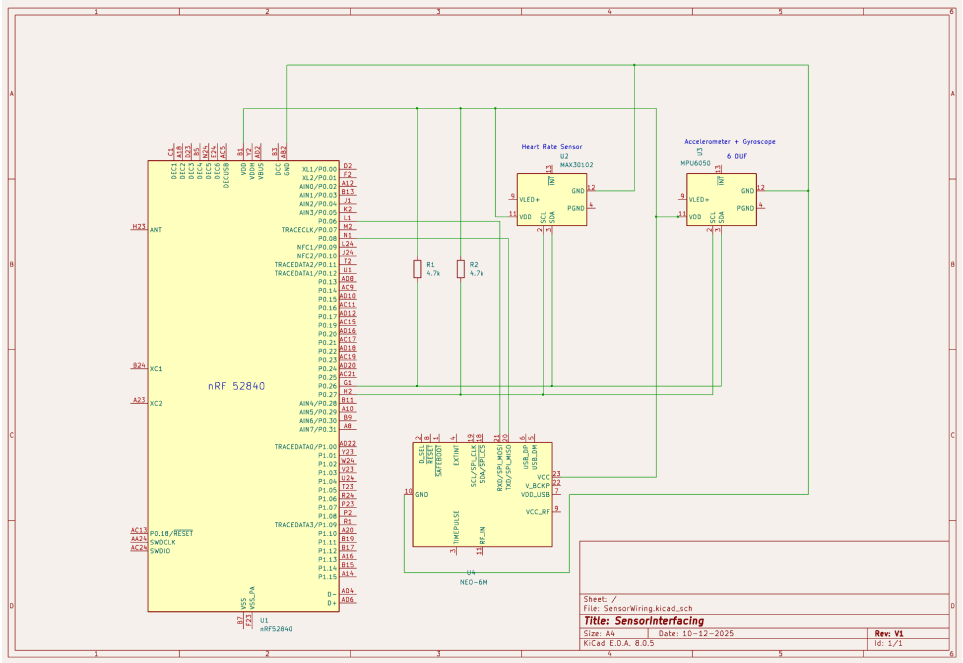

Figure 8. Schematic of components interfacing with nRF52840 DK

5.2.2 Firmware Architecture

The firmware follows a modular design with dedicated handlers for each sensor, coordinated by a central sensor manager. This separation enabled independent development and testing of each subsystem.

File structure:

```
samples/sid_end_device/
|-- CMakeLists.txt              # Build configuration with sensor sources
|-- Kconfig                     # Main Kconfig with
SID_END_DEVICE_SENSOR_MONITORING
|-- Kconfig.sensors             # Sensor-specific configuration options
|-- overlay-sensors.conf        # Overlay config for sensors variant
|-- boards/
|   `-- nrf52840dk_nrf52840.overlay  # Hardware devicetree overlay
|-- include/
|   `-- sensors/
|       |-- data_types.h        # Shared data structures
|       |-- sensor_manager.h    # Sensor coordination API
|       |-- critical_msg_buffer.h # Critical message management API
|       |-- gps_handler.h       # GPS handler API
|       |-- mpu6050_handler.h   # IMU handler API
|       `-- sen0344_handler.h   # Heart rate sensor API
`-- src/
    `-- sensors/
        |-- app.c               # Main application entry and callbacks
        |-- sensor_manager.c    # Sensor coordination implementation
        |-- critical_msg_buffer.c # Ring buffer with retry logic
        |-- gps_handler.c       # NEO-6M GPS handler
        |-- mpu6050_handler.c   # MPU6050 IMU with fall detection
        `-- sen0344_handler.c   # SEN0344 heart rate/SpO2 handler
```

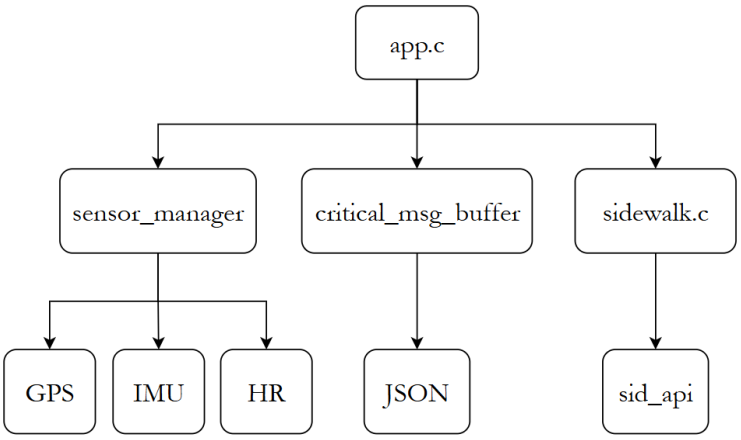

Figure 9. Component Dependencies

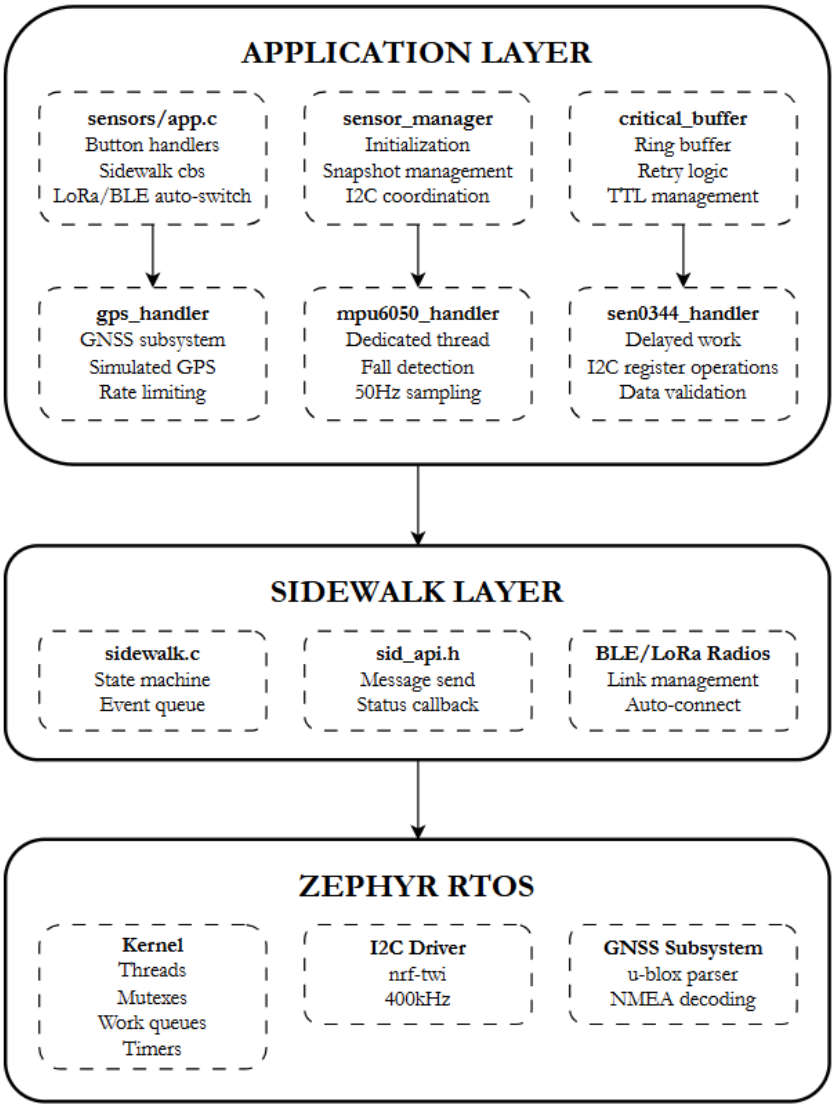

Figure 10. High-Level Architecture Diagram

The system runs three primary execution contexts:

Thread	Stack Size	Priority	Responsibility
sidewalk	10 KB	Configured	Sidewalk protocol state machine
imu_thread	1.5 KB	5	MPU6050 sampling, fall detection
sysworkq	2 KB	Configured	Deferred sensor reads, message handling

The IMU thread runs at 50 Hz to capture motion data for fall detection. Heart rate readings use the system workqueue with a 5-second interval. GPS updates are rate-limited to every 2 minutes to conserve power. All I2C transactions are protected by a shared mutex to prevent bus conflicts.

5.2.3 Sensor Data Processing

Vital Signs (MAX30102)

The DFRobot MAX30102 v2.0 module includes an onboard microcontroller that pre-processes PPG signals, providing heart rate (BPM), SpO2 percentage, and temperature via a custom I2C register protocol. This differs from the standard MAX30102 which outputs raw optical data requiring software processing.

I2C Register Protocol:

Register	Address	Data	Description
Heart Rate/SpO2	0x0C	8 bytes	SpO2 at byte[0], BPM at bytes[2-5] (big-endian)
Temperature	0x14	2 bytes	Whole degrees[0], fractional[1]
Start Collection	0x20	Command	Initiates measurement cycle

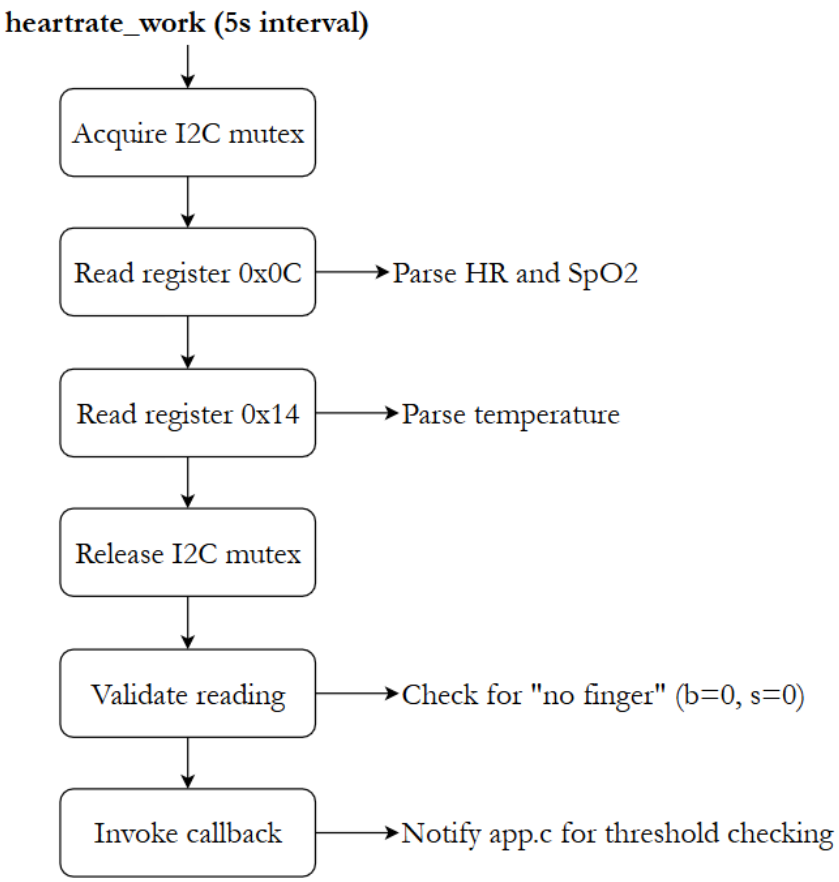

Figure 11. MAX30102 Data Acquisition Flow

Readings are validated before transmission - zero values for both heart rate and SpO2 indicate no finger contact and are flagged as invalid. The callback mechanism allows the main application to evaluate vital sign thresholds and trigger emergency alerts.

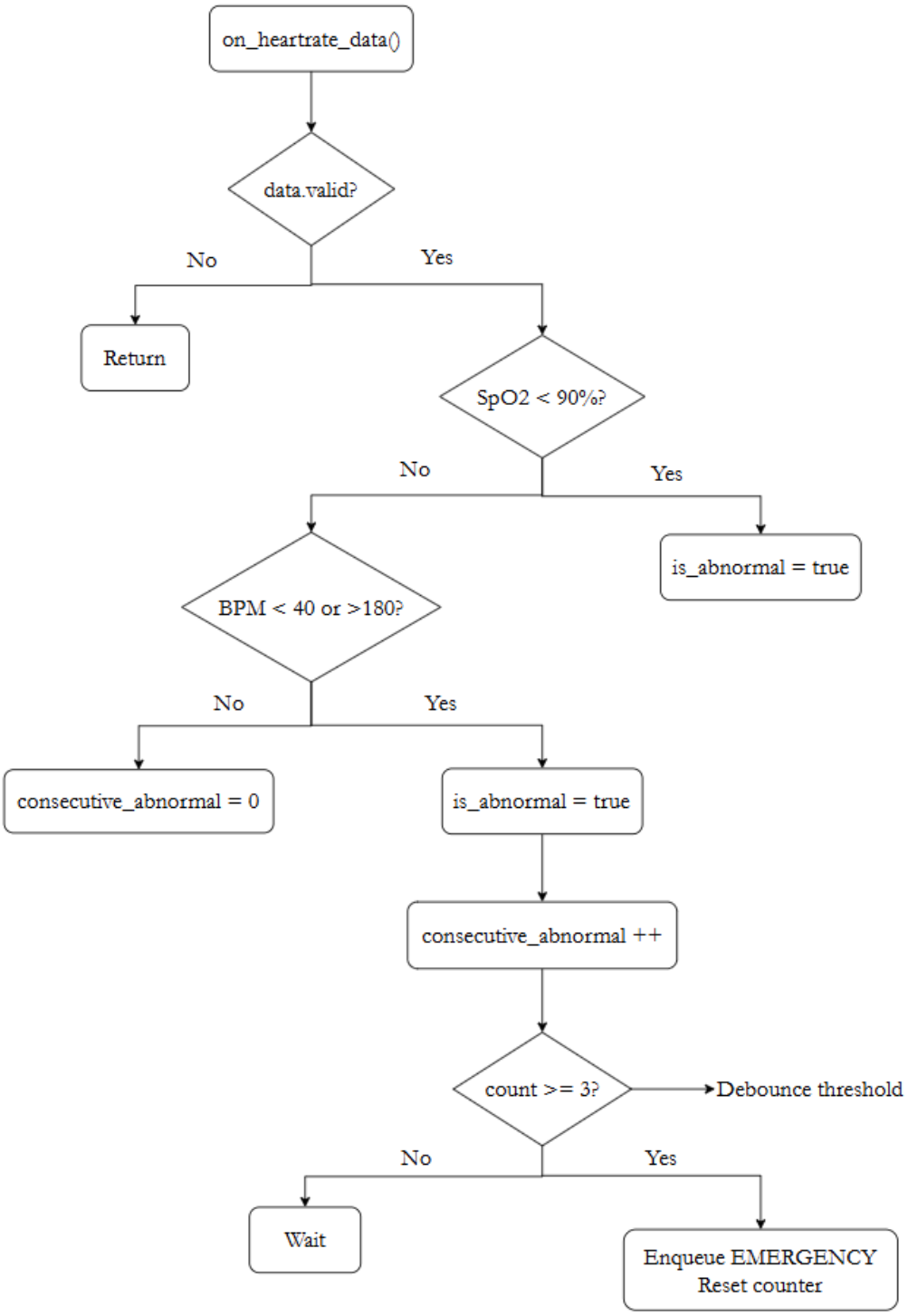

Figure 12. Vital Signs Monitoring State Machine

The debounce logic requires three consecutive abnormal readings before triggering an emergency alert, reducing false positives from momentary sensor noise.

Fall Detection (MPU6050)

The IMU handler runs a dedicated thread sampling accelerometer and gyroscope data at 50 Hz (20ms period). Raw sensor values are converted to physical units and processed through a three-phase state machine.

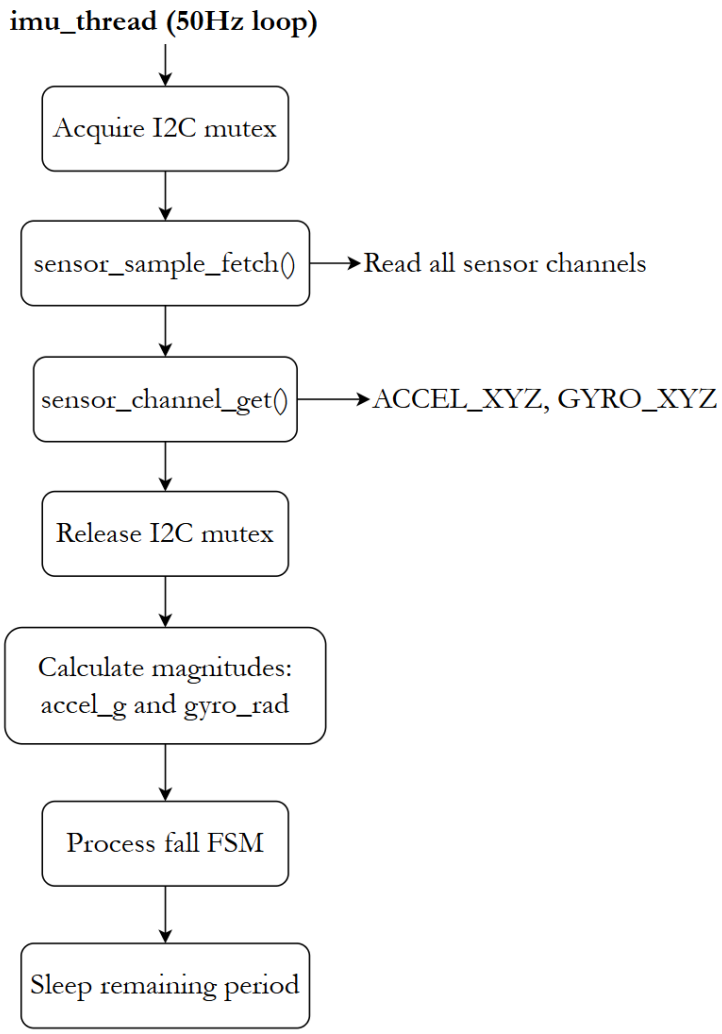

Figure 13. MPU6050 Data Acquisition Flow

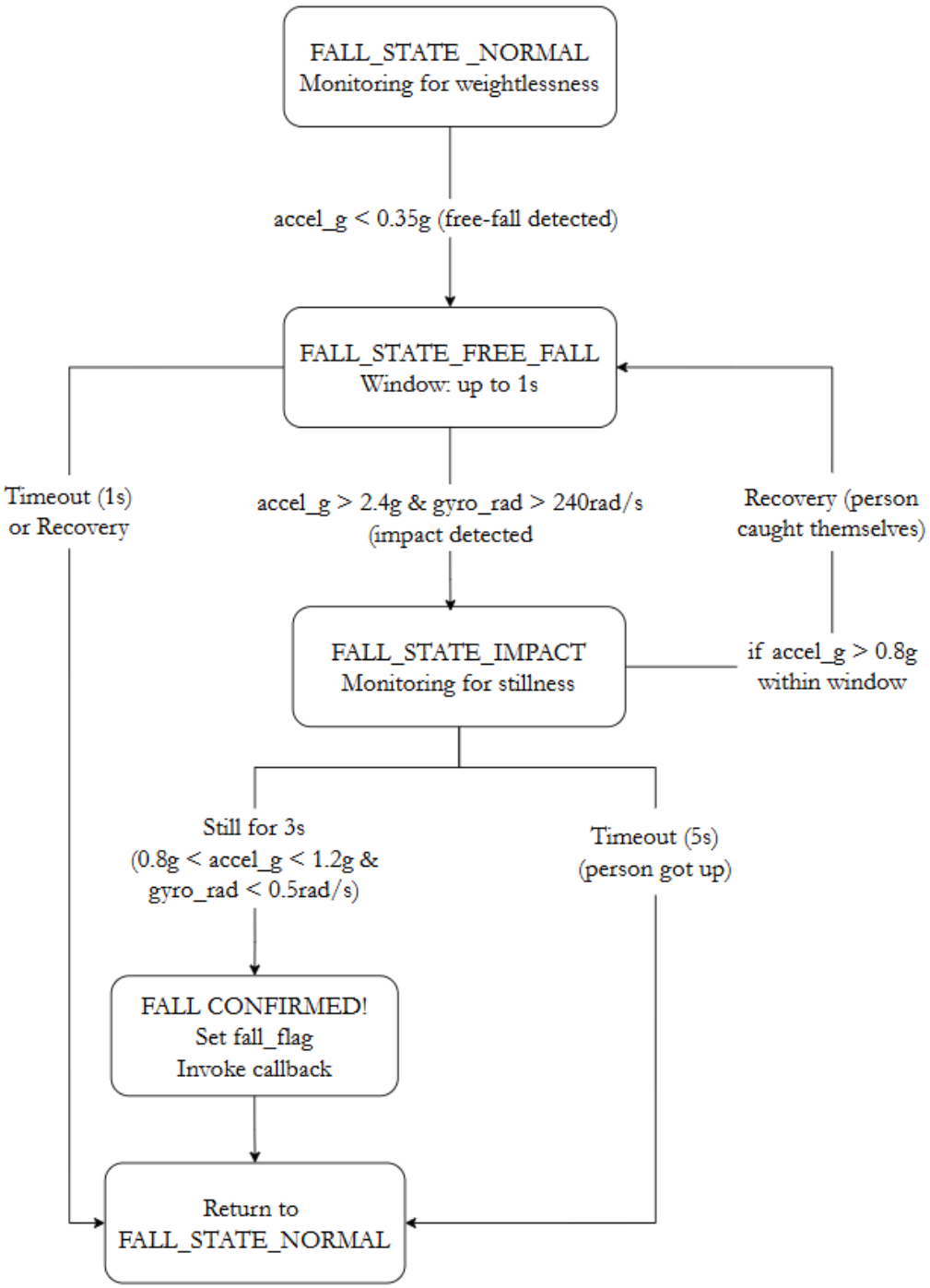

Figure 14. Fall Detection State Machine

Fall detection thresholds:

Parameter	Value	Description
ACCEL_FREE_FALL_G	0.35g	Weightlessness threshold
ACCEL_IMPACT_G	2.4g	Impact detection threshold
GYRO_IMPACT_RAD_S	4.19 rad/s	Rotation threshold (240°/s)
ACCEL_STILL_MIN_G	0.8g	Minimum for lying still
ACCEL_STILL_MAX_G	1.2g	Maximum for lying still
GYRO_STILL_RAD_S	0.5 rad/s	Maximum rotation for stillness
FREE_FALL_WINDOW_MS	1000 ms	Maximum free-fall duration
POST_FALL_DURATION_MS	5000 ms	Post-impact monitoring timeout
POST_FALL_STILL_SAMPLES	150	Samples for stillness (3s at 50Hz)

When a fall is confirmed, the handler sets a flag and invokes a callback to notify the main application, which then enqueues an emergency message.

Location (NEO-6M)

The GPS handler interfaces with the Zephyr GNSS subsystem, which receives NMEA sentences via UART and parses them automatically. A callback registered via GNSS_DATA_CALLBACK_DEFINE receives parsed position data.

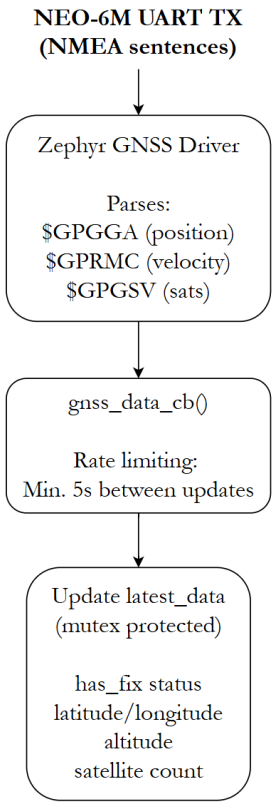

Figure 15. GPS Data Acquisition Flow

GPS Fix Handling

When no GPS fix is available (common during indoor operation), the system reports coordinates as -1 rather than 0,0 (which would indicate a location in the Atlantic Ocean):

```
if (!snapshot->gps.has_fix) {
    lat_int = -1;  // Invalid marker
    lon_int = -1;
} else {
    lat_int = (int32_t)(snapshot->gps.latitude * 100000);
    lon_int = (int32_t)(snapshot->gps.longitude * 100000);
}
```

5.2.4 Message Implementation

All sensor data is formatted into compact JSON payloads for transmission over Amazon Sidewalk. The messaging system implements differentiated handling based on message urgency.

JSON Payload Format


```
{"t":"R","d":1,"f":"N","b":72,"s":98,"T":3650,"la":3364585,"lo":-11784294,"B":100,"ts":18000}
```


Field Definitions

Field	Type	Description	Example
t	string	Message type: 'R' (Regular), 'E' (Emergency), 'H' (Help)	"R"
d	integer	Device ID (1-65535)	1
f	string	Fall state: 'N' (Normal), 'F' (Free-fall), 'I' (Impact)	"N"
b	integer	Heart rate in BPM (0 = no finger contact)	72
s	integer	SpO2 percentage (0 = no finger contact)	98
T	integer	Temperature in centidegrees Celsius (divide by 100)	3650 = 36.50°C
la	integer	Latitude multiplied by 100000 (-1 = no GPS fix)	3364585 = 33.64585°
lo	integer	Longitude multiplied by 100000 (-1 = no GPS fix)	-11784294 = -117.84294°
B	integer	Battery percentage (0-100)	100
ts	integer	Timestamp in seconds since device boot	18000

Data Encoding Conventions

Coordinates and temperature use integer encoding to avoid floating-point representation issues and reduce payload size:

- **Temperature:** Stored as centidegrees Celsius. Divide by 100 to get degrees. Example: 3650 represents 36.50°C.
- **GPS Coordinates:** Stored as degrees multiplied by 100000. Divide by 100000 to get decimal degrees. Example: latitude 3364585 represents 33.64585°N, longitude -11784294 represents 117.84294°W (negative indicates West).

- **Invalid GPS:** When no GPS fix is available (indoor operation), both latitude and longitude are set to -1 rather than 0,0, which would incorrectly indicate a location in the Atlantic Ocean.
- **Invalid Sensor Reading:** When no finger is detected on the heart rate sensor, both heart rate (b) and SpO2 (s) are set to 0. Cloud processing should ignore these readings.
- **Timestamp:** Represents seconds since device boot, not Unix timestamp. Correlate with the MQTT message receive time on AWS IoT Core to determine absolute time.

Message Types and Handling Strategy

Type	Code	Trigger	Interval	Delivery Method	Retry Count
Regular	R′	Timer	Every 5 minutes	Fire-and-forget	None
Emergency	E′	Fall detected, abnormal vitals	Immediate	Ring buffer with tracking	Up to 3
Help	H′	BTN1 panic button press	Immediate	Ring buffer with tracking	Up to 3

Regular Message Flow

Regular status messages provide baseline health monitoring without network overhead. They use a simple fire-and-forget pattern:

1. A 5-minute timer expires and submits work to the system workqueue
2. The work handler checks if Sidewalk is ready (connected and time-synced)
3. If not ready, the message is skipped (no queuing for regular messages)
4. If ready, the handler captures a sensor snapshot from all handlers
5. The snapshot is formatted into a JSON payload
6. The message is submitted to Sidewalk via sid_put_msg()

7. On callback (success or failure), the message memory is freed

8. No retry is attempted regardless of delivery outcome

This approach prevents regular messages from consuming buffer space needed for critical alerts.

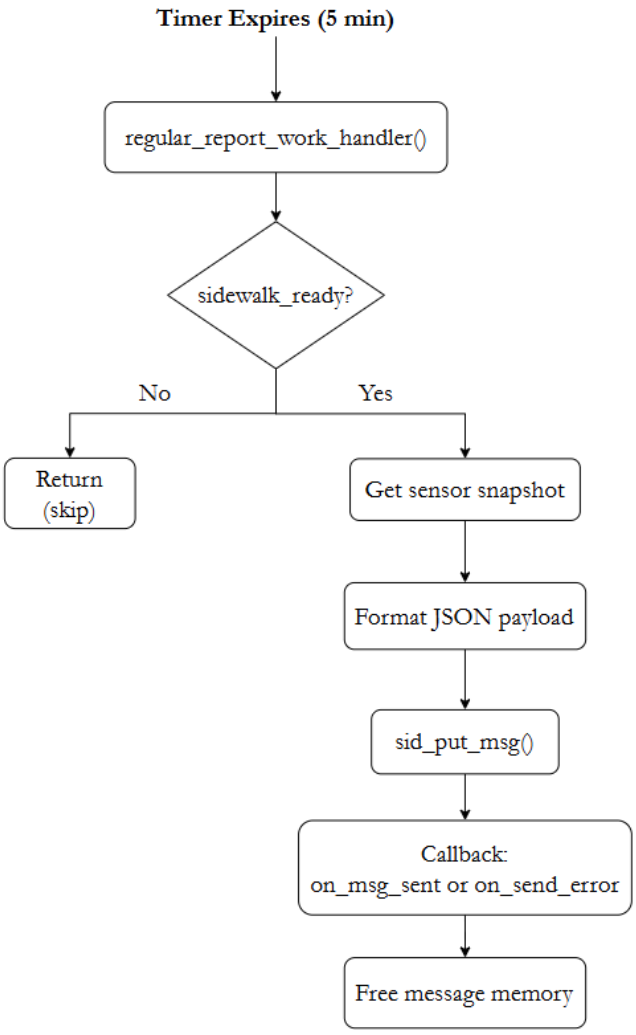

Figure 16. Regular Message Flow

Critical Message Flow (Emergency and Help)

Emergency and Help messages require reliable delivery. The system uses a ring buffer with message tracking, automatic retries, and time-to-live enforcement.

Enqueueing Process:

1. An event triggers the critical message (fall detected, abnormal vitals, or panic button)
2. The critical_buffer_enqueue() function is called with the message type
3. The buffer searches for a free slot; if none available, the oldest message is evicted
4. A sensor snapshot is captured immediately to preserve the state at event time
5. The JSON payload is formatted and stored in the slot
6. The slot state is set to PENDING and creation time is recorded for TTL tracking
7. A work item is submitted to trigger the send process

Send Process:

1. The send_work_handler() iterates through all buffer slots
2. For each PENDING or RETRY slot, it checks if the message TTL has expired (5 minutes)
3. Expired messages are discarded and their slots freed
4. Valid messages are submitted to Sidewalk via sid_put_msg()
5. The slot state changes to SENT and the Sidewalk message ID is stored for callback matching
6. If Sidewalk is not ready, the message remains in PENDING state for later retry

Callback Handling:

On on_msg_sent callback:

- The buffer searches for the slot matching the message ID
- The slot is freed and marked as delivered
- Delivery is confirmed

On on_send_error callback:

- The buffer searches for the slot matching the message ID
- The retry counter is incremented
- If retries remain (less than 3), the slot state changes to RETRY and a delayed work item schedules the next attempt in 5 seconds
- If maximum retries exceeded, the slot is freed and the failure is logged

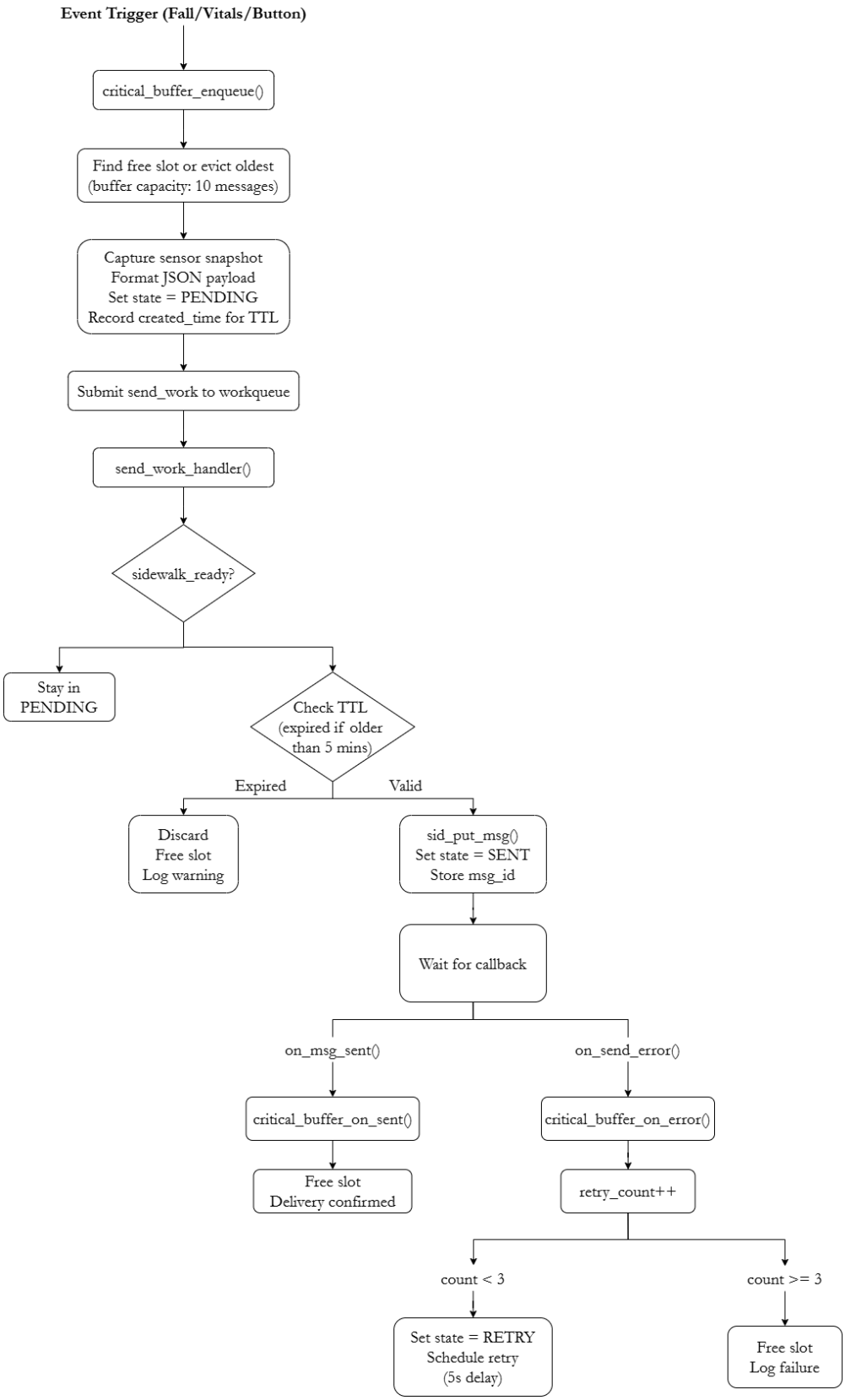

Figure 17. Critical Message Flow

Critical Message Buffer Slot States

Each slot in the ring buffer transitions through defined states:

State	Description	Next States
FREE	Slot available for new message	PENDING (on enqueue)
PENDING	Message waiting to be sent	SENT (on submit), FREE (on TTL expiry)
SENT	Message submitted, awaiting callback	FREE (on success), RETRY (on error)
RETRY	Send failed, waiting for retry timer	PENDING (on retry), FREE (on max retries)

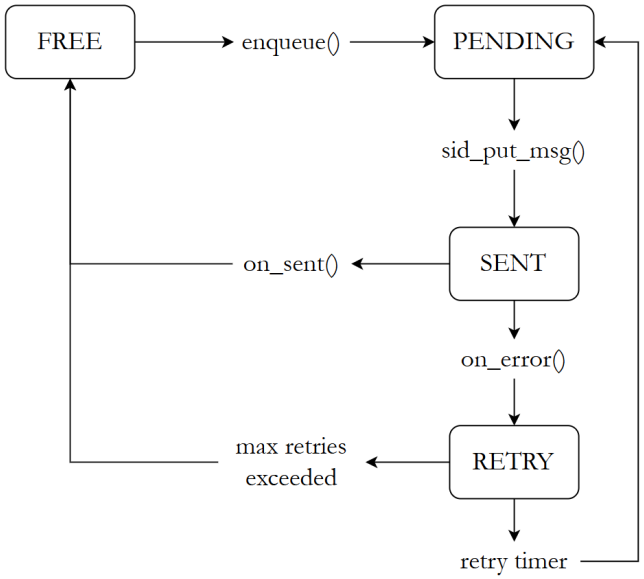

Figure 18. Critical Message Buffer Slot States

Buffer Configuration Parameters

Parameter	Value	Description
CRITICAL_MSG_BUFFER_SIZE	10	Maximum number of pending critical messages
CRITICAL_MSG_TTL	300 secs	Time-to-live before message is discarded
CRITICAL_MSG_RETRIES	3	Maximum retry attempts per message
RETRY_DELAY_MS	5000 ms	Delay between retry attempts

Overflow Handling:

When all 10 buffer slots are occupied and a new critical event occurs:

1. The buffer identifies the oldest message by comparing created_time values
2. The oldest slot is evicted regardless of its current state
3. A warning is logged indicating message eviction
4. The new message takes the freed slot

This ensures recent emergencies always have buffer space, prioritizing timely alerts over stale undelivered messages.

Automatic LoRa Link Switching

After initial BLE registration and time synchronization, the device automatically switches to LoRa for data transmission. This optimizes both range (LoRa reaches much farther than BLE) and power consumption.

Switch Conditions (all must be true):

- Device is registered with Amazon Sidewalk (SID_STATUS_REGISTERED)
- Time synchronization completed (SID_STATUS_TIME_SYNCED)
- BLE link is active (SID_LINK_TYPE_1 in link_status_mask)
- Switch has not already been performed this boot cycle

Switch Process:

1. The on_sidewalk_status_changed() callback monitors registration and link status
2. When all conditions are met, a delayed work item is scheduled (2-second safety delay)
3. The work handler sends SID_EVENT_LINK_SWITCH with the new link mask
4. The new mask includes both BLE and LoRa (SID_LINK_TYPE_1 | SID_LINK_TYPE_3)

5. Sidewalk reinitializes with the updated configuration

6. A flag prevents repeated switching on subsequent status updates

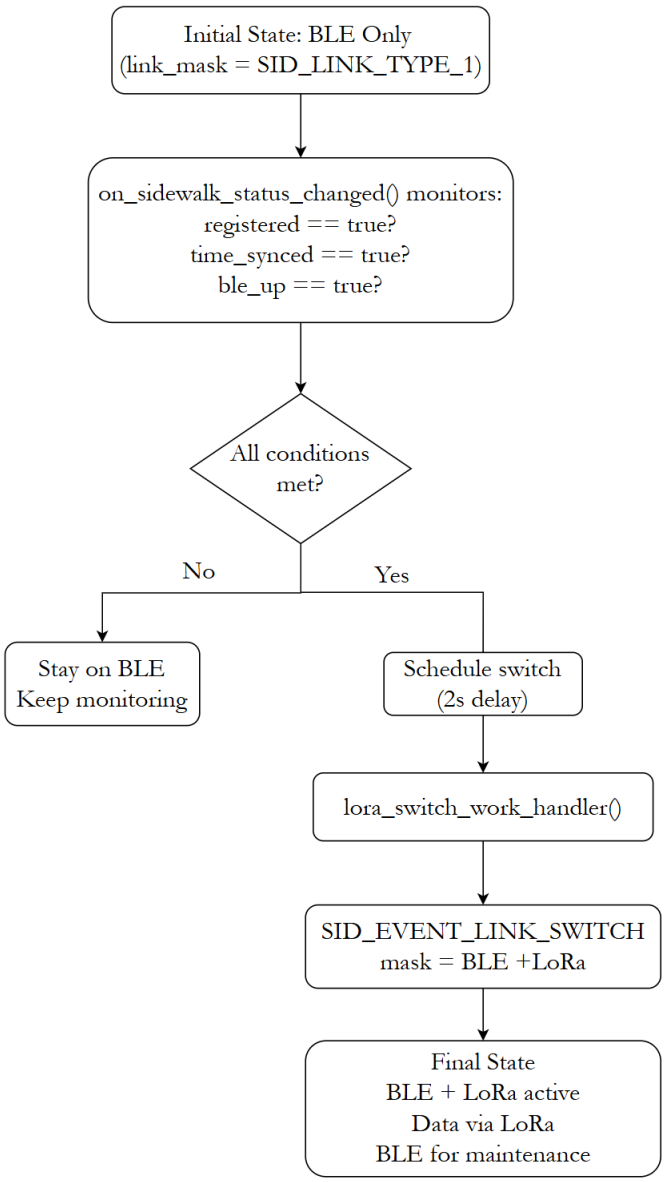

Figure 19. Automatic LoRa Link Switching Flow

After switching, both links remain active. LoRa handles regular data transmission with its superior range, while BLE remains available for proximity-based operations and maintenance tasks.

System Initialization Sequence

1. **app_start()**: Entry point called from main, begins system initialization
2. **buttons_init()**: Configures BTN1 as panic button with interrupt handler
3. **sensor_manager_init()**: Initializes I2C mutex and calls each sensor handler's init function
4. **Handler init functions**: Configure I2C/UART peripherals and verify sensor presence
5. **sensor_manager_start()**: Starts sensor data collection (IMU thread, heart rate work, GPS callbacks)
6. **critical_buffer_init()**: Initializes ring buffer and associated work items
7. **start_regular_reports()**: Starts the 5-minute timer for regular status messages
8. **register_callbacks()**: Connects fall detection and vital sign callbacks to message handlers
9. **sidewalk_start()**: Initializes Sidewalk stack and begins BLE scanning for gateway
10. **BLE registration**: Device registers with Sidewalk network and synchronizes time
11. **LoRa switch**: After successful registration, device switches to LoRa for data transmission

5.2.5 Cloud Infrastructure

The backend processes sensor data and serves the caregiver dashboard through a serverless AWS architecture deployed via CloudFormation.

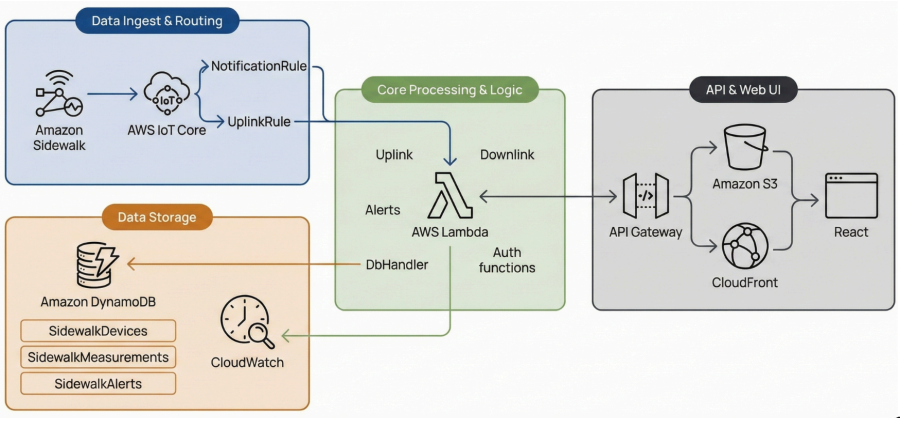

Figure 20. Cloud infrastructure architecture

Data Flow

1. **Ingest:** Device transmits JSON payload over LoRa to Sidewalk bridge
2. **Routing:** AWS IoT Core receives message, IoT Rules route to Lambda functions
3. **Processing:** Uplink Lambda parses payload, evaluates alert thresholds, stores data
4. **Storage:** DynamoDB tables persist device state, measurements, and alerts
5. **Serving:** API Gateway exposes REST endpoints for dashboard queries
6. **Display:** React frontend hosted on S3/CloudFront polls for updates

AWS Services

Service	Function
IoT Core	MQTT message ingestion and routing
Lambda	Uplink processing, downlink commands, database operations, authentication
DynamoDB	Three tables: SidewalkDevices, SidewalkMeasurements, SidewalkAlerts
API Gateway	REST API for dashboard data access
S3	Static web asset storage
CloudFront	CDN for global dashboard distribution
CloudWatch	Logging, metrics, and alerting

5.2.6 Caregiver Dashboard

The web dashboard provides caregivers with real-time visibility into device status, health metrics, and location tracking for multiple elderly users.

Dashboard Views

View	Features
Overview Card	Patient name, connection state, battery %, last check-in time, Sidewalk gateway used
Vitals Card	Heart rate (BPM) with colored status bands, trend sparkline (last 15 minutes)
Motion Card	IMU-derived posture/activity (walking, sitting, fall-suspect), inactivity timer, "Request movement" downlink button
Location Card	Last GPS fix (lat/long, accuracy), map pin or textual address, timestamp, "Request refresh" downlink button
Alerts Banner	Threshold warnings , dismiss and auto-clear when resolved
History Table	Recent events (heart-rate alerts, motion changes, command requests/results) from Measurements table
Onboarding	Elder Band ID entry, friendly name assignment, one-click device registration

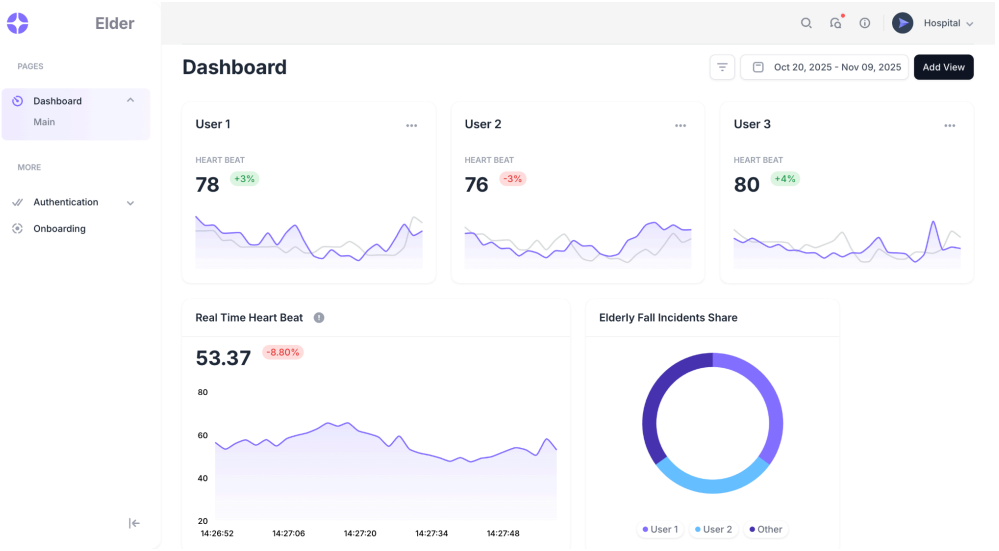

Figure 21. Main dashboard with vitals and charts

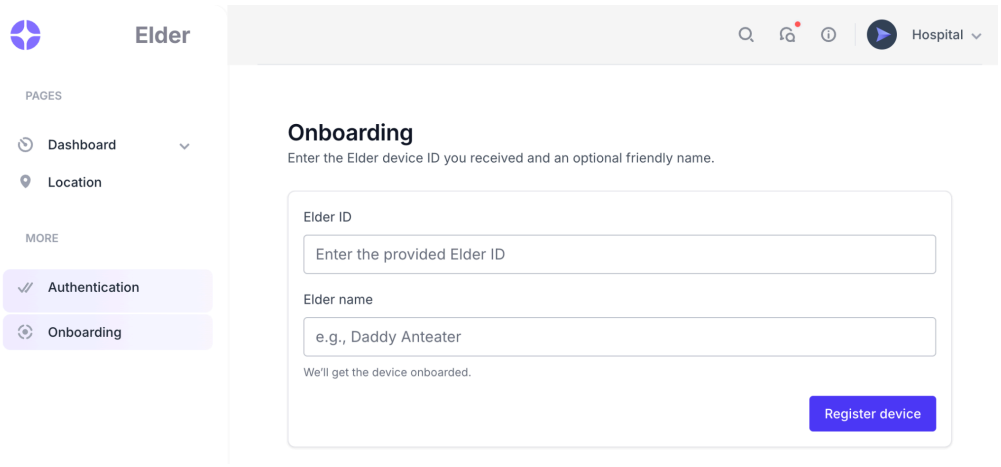

Figure 22. Device onboarding page

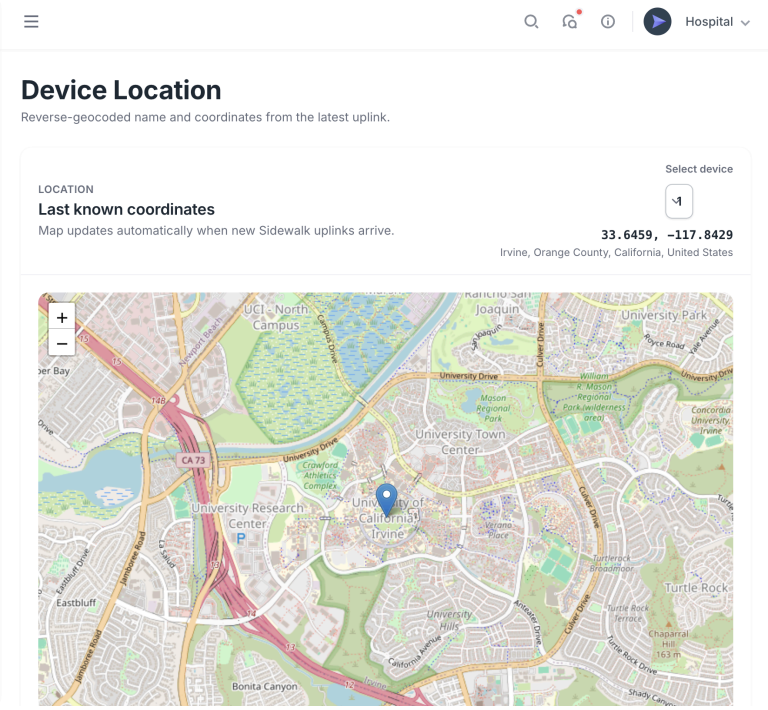

Figure 23. Location tracking with interactive map

User Interactions

- **Manual Refresh:** Pulls latest measurements via API Gateway to DbHandler Lambda
- **Remote Commands:** Downlink Lambda sends payloads for movement prompt, LED blink, or buzzer activation
- **Authentication:** Username/password with config-backed tokens; all API calls signed by CloudFront token generator/authorizer

UI Behavior

Aspect	Implementation
Polling Cadence	15-30 seconds for vitals/activity; on-demand for GPS to conserve device battery
Loading States	Skeleton placeholders for cards; "No data yet" for new devices; error chip on API failure
Responsiveness	Single-column on mobile; two-column on desktop; Overview card always first
Accessibility	High-contrast color palettes, text resize-safe layout, keyboard focus on buttons, concise alert language

Frontend Architecture

Decision	Rationale
Vite Build Tool	Faster builds and hot-module replacement (HMR) compared to Create React App
Leaflet Mapping	Renders location view with custom markers distinguishing current vs. last-known position during signal loss
React Hooks	useState and useEffect manage polling lifecycle, preventing memory leaks and UI freezes during 15-second refresh cycles
Tailwind CSS	Utility-first styling for rapid UI development

Automated Deployment

The cloud infrastructure is defined as Infrastructure as Code (IaC) using AWS CloudFormation. A custom Python deployment script automates the release process:

1. Builds the React frontend and syncs static assets to S3
2. Packages Python Lambda functions with dependencies
3. Deploys CloudFormation stack with CloudFront distribution linked to S3 origin
4. Configures IAM roles with least-privilege access policies

5.3 Future Work

Based on lessons learned from the proof-of-concept prototype, the following enhancements are planned for future development:

Hardware Miniaturization

- Design custom PCB integrating nRF52840, SX1262 LoRa transceiver, and all sensors on a single board
- Develop ergonomic wearable enclosure suitable for elderly users (lightweight, comfortable, water-resistant)
- Integrate battery with power management IC and fuel gauge for accurate battery monitoring

Power Optimization

- Implement sleep modes and duty cycling to extend battery life
- Optimize sensor sampling rates based on activity detection
- Profile power consumption across all operating states

Algorithm Improvements

- Implement anomaly detection for heart rate patterns (machine learning-based)
- Refine fall detection thresholds to reduce false positives
- Add activity classification (walking, sitting, lying down)

User Experience

- Develop caregiver mobile app with real-time push notifications
- Add SMS/email alerts for emergency events
- Implement geofencing with customizable safe zones

Clinical Validation

- Partner with healthcare facilities for real-world testing
- Validate sensor accuracy against medical-grade equipment
- Conduct user studies with elderly participants and caregivers

6. Experiments and Results

This section documents the testing performed to validate the Advanced Elderly Care System prototype, including sensor communication, Amazon Sidewalk connectivity, and end-to-end system operation.

6.1 Test Setup

6.1.1 Hardware Configuration

Component	Model	Role
Microcontroller	Nordic nRF52840 DK	Main processor, BLE radio
LoRa Transceiver	Semtech SX1262 EVB	915 MHz Sub-GHz communication
PPG Sensor	DFRobot MAX30102 v2.0	Heart rate, SpO2, temperature
IMU	MPU6050	Accelerometer, gyroscope
GPS	NEO-6M	Location tracking
Gateway	Amazon Echo Dot 5th Gen	Sidewalk bridge (SubG-CSS)
Power	5V USB from PC	Development power source

All tests conducted indoors on a non-conductive bench surface.

6.1.2 Software Environment

Software	Version	Purpose
nRF Connect SDK	v2.7.0	Firmware development
Amazon Sidewalk SDK	v2.7.0	Sidewalk protocol stack
Zephyr RTOS	Included in SDK	Real-time operating system
VS Code	Latest	IDE with nRF extensions
nRF RTT Terminal	Latest	Debug output (UART reserved for GPS)
MobaXterm	Latest	Serial terminal backup
AWS CLI	v2.x	Cloud provisioning and testing
Python	3.6+	Provisioning scripts

6.1.2 Test Objectives Overview

Category	Objective
Sensor Communication	Verify I2C/UART communication and data parsing for all sensors
Sidewalk Communication	Validate bidirectional message flow via Amazon Sidewalk over LoRa
End-to-End System	Confirm complete data path from sensor to dashboard
Cloud Infrastructure	Verify AWS backend processing, storage, and UI rendering

6.2 Sensor Communication Tests

Test Case 1: Heart Rate Sensor Communication (MAX30102)

Objective: Verify I2C communication between nRF52840 and MAX30102 sensor, confirm BPM and SpO2 data parsing.

Parameter	Details
Input	I2C read requests to address 0x57, registers 0x0C (HR/SpO2) and 0x14 (temperature)
Expected Output	Continuous heart rate (BPM), SpO2 (%), and temperature readings on serial terminal
Actual Output	Successful data acquisition with readable values displayed every 5 seconds
Result	PASS

Analysis: Sensor initialization completed without I2C errors. Minor fluctuations observed in readings, which is expected behavior for a low-cost reflective PPG module. "-1" values correctly indicate no finger contact.

```
[00:02:31.846,588] <inf> sen0344: HR=-1 bpm, SpO2=-1 %
[00:02:31.852,996] <inf> sen0344: Temp=30.07 C
[00:02:32.859,313] <inf> sen0344: HR=102 bpm, SpO2=-1 %
[00:02:32.865,814] <inf> sen0344: Temp=30.06 C
[00:02:33.872,131] <inf> sen0344: HR=102 bpm, SpO2=-1 %
[00:02:33.878,631] <inf> sen0344: Temp=30.14 C
[00:02:34.884,979] <inf> sen0344: HR=102 bpm, SpO2=-1 %
[00:02:34.891,479] <inf> sen0344: Temp=30.10 C
[00:02:35.897,796] <inf> sen0344: HR=102 bpm, SpO2=-1 %
[00:02:35.904,296] <inf> sen0344: Temp=29.14 C
[00:02:36.910,614] <inf> sen0344: HR=110 bpm, SpO2=-1 %
[00:02:36.917,114] <inf> sen0344: Temp=29.13 C
[00:02:37.923,431] <inf> sen0344: HR=110 bpm, SpO2=-1 %
[00:02:37.929,931] <inf> sen0344: Temp=30.05 C
[00:02:38.936,248] <inf> sen0344: HR=110 bpm, SpO2=-1 %
[00:02:38.942,749] <inf> sen0344: Temp=30.06 C
[00:02:39.949,096] <inf> sen0344: HR=110 bpm, SpO2=-1 %
[00:02:39.955,596] <inf> sen0344: Temp=30.15 C
[00:02:40.961,944] <inf> sen0344: HR=-1 bpm, SpO2=-1 %
[00:02:40.968,353] <inf> sen0344: Temp=30.13 C
```

Figure 24. MAX30102 sensor output showing heart rate, SpO2, and temperature readings

Test Case 2: IMU Sensor Communication (MPU6050)

Objective: Verify I2C communication with MPU6050 and validate accelerometer/gyroscope data acquisition at 50 Hz.

Parameter	Details
Input	I2C read requests to address 0x68 for accelerometer (XYZ) and gyroscope (XYZ) channels
Expected Output	Continuous 6-axis motion data responsive to board movement
Actual Output	Stable accelerometer and gyroscope values; data changes appropriately when board is tilted or rotated
Result	PASS

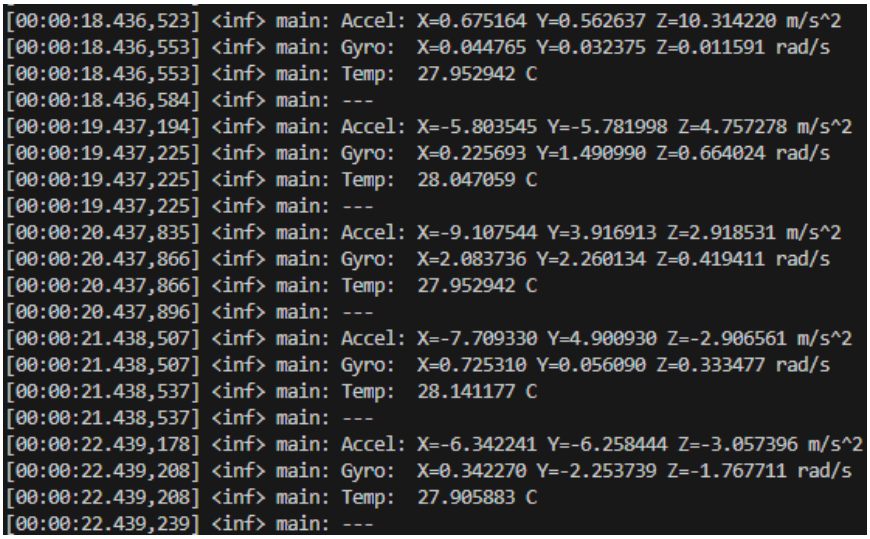

Figure 25. MPU6050 sensor output showing accelerometer and gyroscope values

Analysis: IMU integration successful. Raw data streams at 50 Hz as configured. Data ready for fall detection algorithm processing.

Test Case 3: GPS Module Communication (NEO-6M)

Objective: Verify UART communication with NEO-6M and confirm reception of NMEA sentences.

Parameter	Details
Input	UART data at 9600 baud from NEO-6M TX pin
Expected Output	Raw NMEA sentences ($GPGGA, $GPRMC, $GPGSV) displayed on serial terminal
Actual Output	NMEA sentences received continuously at 1 Hz update rate
Result	PASS

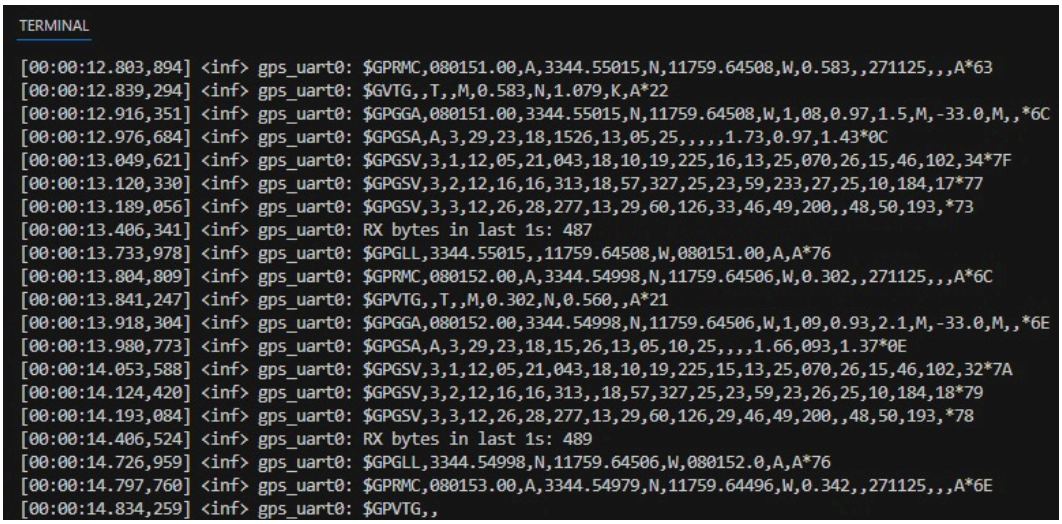

Figure 26. GPS module NMEA sentence output on serial terminal

Analysis: GPS UART communication verified. Module outputs standard NMEA-0183 sentences at 9600 baud. Zephyr GNSS subsystems parse these sentences automatically to extract position data. Indoor testing shows valid NMEA frames but with no-fix status (expected behavior without satellite visibility).

6.3 Combined Sensor and Algorithm Tests

Test Case 4: Combined Sensor Integration with Fall Detection

Objective: Verify all three sensors (MAX30102, MPU6050, NEO-6M) operate together in a single firmware build, and validate fall detection algorithm with GPS location output.

Parameter	Details
Input	Combined sensor code reading PPG, IMU (50Hz), and GPS simultaneously; simulated fall by dropping the test board
Expected Output	Heart rate/SpO2 readings, GPS coordinates (latitude/longitude), and fall detection state changes displayed on serial terminal
Actual Output	All sensors operational concurrently; fall detection triggered correctly on simulated recovery and simulated fall; GPS coordinates displayed
Result	PASS

```
[00:00:30.757,904] <inf> sen0344_handler: Heart Rate: 132 BPM, SpO2: 97%, Temp: 29.14 C
[00:00:35.500,488] <inf> main: IMU: accel=1.05g, gyro=0.08rad/s
[00:00:35.758,666] <inf> main: Health: HR=126 BPM, SpO2=96%, Temp=30.07 C
[00:00:35.758,666] <inf> sen0344_handler: Heart Rate: 126 BPM, SpO2: 96%, Temp: 30.07 C
[00:00:39.960,906] <wrn> mpu6050_handler: Impact detected: accel=3.46g, gyro=6.19rad/s
[00:00:40.522,613] <inf> main: IMU: accel=1.06g, gyro=0.09rad/s
[00:00:44.961,608] <inf> mpu6050_handler: Post-fall timeout, person appears to have recovered
[00:00:45.566,528] <inf> main: IMU: accel=1.05g, gyro=0.54rad/s
[00:00:50.599,945] <inf> main: IMU: accel=0.09g, gyro=0.68rad/s
[00:00:50.681,396] <wrn> mpu6050_handler: Impact detected: accel=2.67g, gyro=4.38rad/s
[00:00:53.818,908] <err> mpu6050_handler: *** FALL CONFIRMED ***
[00:00:53.818,939] <err> sensor_manager: === FALL DETECTED ===
[00:00:53.818,969] <err> sensor_manager: Location: 33.645850, -117.842940
[00:00:53.818,969] <err> sensor_manager: Heart Rate: 0 BPM, SpO2: 0%
[00:00:53.818,969] <err> main: ==========================================
[00:00:53.819,000] <err> main:          FALL ALERT - EMERGENCY
[00:00:53.819,000] <err> main: ==========================================
[00:00:53.819,000] <err> main: Location: https://maps.google.com/?q=33.645850,-117.842940
[00:00:53.819,030] <err> main: Altitude: 0 m, Satellites: 8
[00:00:53.819,030] <err> main: Health data: Not available
[00:00:53.819,061] <err> main: Impact: accel=1.07g, gyro=0.08rad/s
[00:00:53.819,061] <err> main: ==========================================
```

Figure 27. Combined sensor output showing heart rate, GPS coordinates, and fall detection

Analysis:

- I2C bus coordination via mutex prevents conflicts between MAX30102 and MPU6050

- GPS operates independently on UART without affecting I2C sensors
- First impact at 39.9s (accel=3.46g, gyro=6.19rad/s) detected but person recovered (timeout at 44.9s)
- Second impact at 50.6s (accel=2.67g, gyro=4.38rad/s) followed by stillness confirmed fall at 53.8s
- Fall detection correctly distinguishes between recoverable impacts and actual falls requiring 3-second stillness
- Emergency alert includes clickable Google Maps link with GPS coordinates for caregiver response
- Heart rate shows 0 BPM during fall event (finger not on sensor during test) - correctly reported as "Not available"

6.4 Amazon Sidewalk Communication Tests

Test Case 5: Uplink Communication with Hello Sample Variant

Objective: Verify device can transmit sensor data payload to Amazon Sidewalk network via LoRa and receive it on AWS IoT Core MQTT.

Parameter	**Details**
Input	Sensor data payload sent from device after button press
Expected Output	Base64-encoded payload received on subscribed MQTT topic in AWS IoT Core
Actual Output	Message successfully received on MQTT test client
Result	PASS

Figure 28. AWS IoT Core MQTT message showing successful Sidewalk uplink

Payload Decoding:

```
Base64: NDg2NTZjNmM2ZjIwNjY3MjZmNmQyMDRkNDU0MzUwNTMyMDU0NjU2MTZkMDA=
Decoded (hex string): 48656c6c6f2066726f6d204d454350532054656160
ASCII: "Hello from MECPS Team"
```

Test Case 6: Sensors Variant Full Integration with Sidewalk

Objective: Verify complete sensors variant firmware with automatic LoRa switching, regular periodic messages, help button functionality, fall detection emergency alerts, and critical message buffer operation.

Parameter	Details
Input	Full sensors variant firmware with 20-second regular interval, button press for help request, simulated fall for emergency alert
Expected Output	Automatic BLE to LoRa switch, periodic Regular ('R') messages, Help ('H') message on button press, Emergency ('E') message on fall detection, all received on MQTT
Actual Output	All message types transmitted successfully via LoRa; automatic link switching completed; critical buffer tracked emergency and help messages
Result	PASS

Device Serial Output (Initialization and LoRa Switch):

```
[00:00:00.003] <inf> sensor_manager: Sensor manager initialized
[00:00:00.003] <inf> critical_buffer: Critical message buffer initialized
(size=10, TTL=300s, retries=3)
[00:00:00.003] <inf> sensor_manager: Fall detection callback registered
[00:00:00.003] <inf> sensor_manager: Heart rate callback registered
[00:00:00.003] <inf> gps_handler: GPS handler started, update interval=5000
ms
[00:00:00.003] <inf> mpu6050_handler: MPU6050 handler started
[00:00:00.003] <inf> mpu6050_handler: IMU thread started, rate=50Hz
[00:00:00.004] <inf> sen0344_handler: SEN0344 handler started,
interval=5000 ms
[00:00:00.004] <inf> sensors_app: Sensors variant initialized
(interval=20000 ms)
[00:00:00.004] <inf> sensors_app: Regular reports started (every 20
seconds)
[00:00:00.213] <inf> sidewalk_app: Sidewalk link switch to BLE
[00:00:02.621] <inf> application_state: time_sync = true
[00:00:02.621] <inf> sensors_app: Device registered, Time Sync OK, Link:
{BLE:Up, FSK:Down, LoRa:Down}
[00:00:02.621] <inf> sensors_app: LoRa switch conditions met - scheduling
in 2000 ms
[00:00:04.622] <inf> sensors_app: === Switching to BLE + LoRa ===
[00:00:04.622] <inf> sidewalk_app: Using provided link mask: 0x5
[00:00:04.622] <inf> sidewalk_app: Sidewalk link switch to LoRa
[00:00:17.084] <inf> sensors_app: Device registered, Time Sync OK, Link:
{BLE:Down, FSK:Down, LoRa:Up}
```

Regular Message Transmission:

```
[00:00:20.004] <inf> sensors_app: Regular message sent:
{"t":"R","d":1,"f":"N","b":0,"s":0,"T":2912,"la":0,"lo":0,"B":100,"ts":20}
[00:00:23.648] <inf> sensors_app: Message sent successfully (id=1)
[00:00:40.004] <inf> sensors_app: Regular message sent:
{"t":"R","d":1,"f":"N","b":102,"s":0,"T":3107,"la":0,"lo":0,"B":100,"ts":40
}
[00:00:40.009] <inf> sensors_app: Message sent successfully (id=2)
```

Help Button Press:

```
[00:00:47.995] <inf> button: button pressed 1 short
[00:00:47.995] <inf> sensors_app: *** HELP BUTTON PRESSED - Sending Help
Request ***
[00:00:47.995] <inf> critical_buffer: Critical message H enqueued (slot=0,
count=1):
{"t":"H","d":1,"f":"N","b":103,"s":0,"T":3102,"la":-1,"lo":-1,"B":100,"ts":
47}
[00:00:47.995] <inf> critical_buffer: Critical message H sent (slot=0,
id=0, retry=0)
[00:00:48.000] <inf> sensors_app: Message sent successfully (id=3)
```

Fall Detection and Emergency Alert:

```
[00:01:22.140] <wrn> mpu6050_handler: Impact detected: accel=2.83g,
gyro=5.38rad/s
[00:01:25.401] <err> mpu6050_handler: *** FALL CONFIRMED ***
[00:01:25.402] <inf> sensor_manager: Fall detected by sensor manager
[00:01:25.402] <err> sensors_app: *** FALL DETECTED - Sending Emergency
Alert ***
[00:01:25.402] <inf> critical_buffer: Critical message E enqueued (slot=1,
count=2):
{"t":"E","d":1,"f":"I","b":0,"s":0,"T":3003,"la":-1,"lo":-1,"B":100,"ts":85
}
[00:01:25.402] <inf> critical_buffer: Critical message E sent (slot=1,
id=0, retry=0)
[00:01:25.407] <inf> sensors_app: Message sent successfully (id=6)
```

AWS IoT Core MQTT Messages Received:

Seq	Type	Decoded Payload	Timestamp
1	Regular	{"t":"R","d":1,"f":"N","b":0,"s":0,"T":2912,"la":0,"lo":0,"B":100,"ts":20}	14:03:02.078Z
2	Regular	{"t":"R","d":1,"f":"N","b":102,"s":0,"T":3107,"la":0,"lo":0,"B":100,"ts":40}	14:03:18.435Z
3	Help	{"t":"H","d":1,"f":"N","b":103,"s":0,"T":3102,"la":-1,"lo":-1,"B":100,"ts":47}	14:03:26.413Z
4	Regular	{"t":"R","d":1,"f":"N","b":0,"s":95,"T":3012,"la":0,"lo":0,"B":100,"ts":60}	14:03:38.442Z
5	Regular	{"t":"R","d":1,"f":"N","b":0,"s":0,"T":3105,"la":0,"lo":0,"B":100,"ts":80}	14:03:58.427Z

6	Emergency	{"t":"E","d":1,"f":"I","b":0,"s":0,"T":3003,"la":-1,"lo":-1,"B":100,"ts":85}	14:04:03.841Z
7	Regular	{"t":"R","d":1,"f":"I","b":0,"s":0,"T":3106,"la":0,"lo":0,"B":100,"ts":100}	14:04:18.424Z
8	Regular	{"t":"R","d":1,"f":"N","b":0,"s":0,"T":3107,"la":0,"lo":0,"B":100,"ts":120}	14:04:38.424Z

Sample MQTT Message (Emergency Alert - Seq 6):


```
{
  "MessageId": "cde830ce-868e-4b3e-8eee-a6bcfcb39f37",
  "WirelessDeviceId": "fb956ac1-4c83-445f-99e1-391273818e57",
  "PayloadData": "N2IyMjc0MjIzYTIyNDUyMjJjMjI2NDIyM2EzMTJj...",
  "WirelessMetadata": {
    "Sidewalk": {
      "CmdExStatus": "COMMAND_EXEC_STATUS_UNSPECIFIED",
      "MessageType": "CUSTOM_COMMAND_ID_NOTIFY",
      "NackExStatus": [],
      "Seq": 6,
      "SidewalkId": "BFFF9BFC53",
      "Timestamp": "2025-12-12T14:04:03.841Z"
    }
  }
}
```


Analysis:

- **Automatic LoRa Switching:** Device registered via BLE, achieved time sync, then automatically switched to LoRa after 2-second delay (link mask 0x5 = BLE + LoRa). LoRa link came up at 17.08 seconds.
- **Regular Messages:** Sent every 20 seconds as configured, fire-and-forget pattern with immediate success confirmation. All 8 messages received on MQTT with sequential Seq numbers.
- **Help Request:** Button 1 short press immediately enqueued Help message ('H') in critical buffer slot 0. GPS coordinates show -1 (no fix indoors) as designed.
- **Fall Detection:** Impact detected at 82.14s (accel=2.83g, gyro=5.38rad/s), fall confirmed after 3.26 seconds of stillness. Emergency message ('E') enqueued in critical buffer slot 1 with fall state 'I' (Impact).

- **Critical Buffer Operation:** Both Help and Emergency messages tracked in separate slots (slot 0, slot 1), with count incrementing correctly. Retry mechanism initialized at retry=0.
- **Message Differentiation:** Regular messages show "f":"N" (Normal) until fall, then "f":"I" (Impact) in subsequent regular messages at ts=100, returning to "f":"N" after recovery timeout at ts=120.
- **Payload Encoding:** All payloads successfully Base64-encoded for Sidewalk transport and decoded correctly on cloud side.

6.5 Cloud Infrastructure Tests

Test Case 8: CloudWatch Monitoring and Alerting

Objective: Verify CloudWatch captures Lambda/API metrics and triggers alarms on failures.

Parameter	Details
Input	Force an error path (e.g., send malformed payload to API Gateway or invoke uplink Lambda with missing required fields)
Expected Output	CloudWatch logs contain the error entry; Lambda Errors metric increments; API Gateway 5XX (if via API) increments; alarm transitions to ALARM and sends SNS/notification
Actual Output	Error log present; metrics show spike; alarm fired to configured channel
Result	PASS

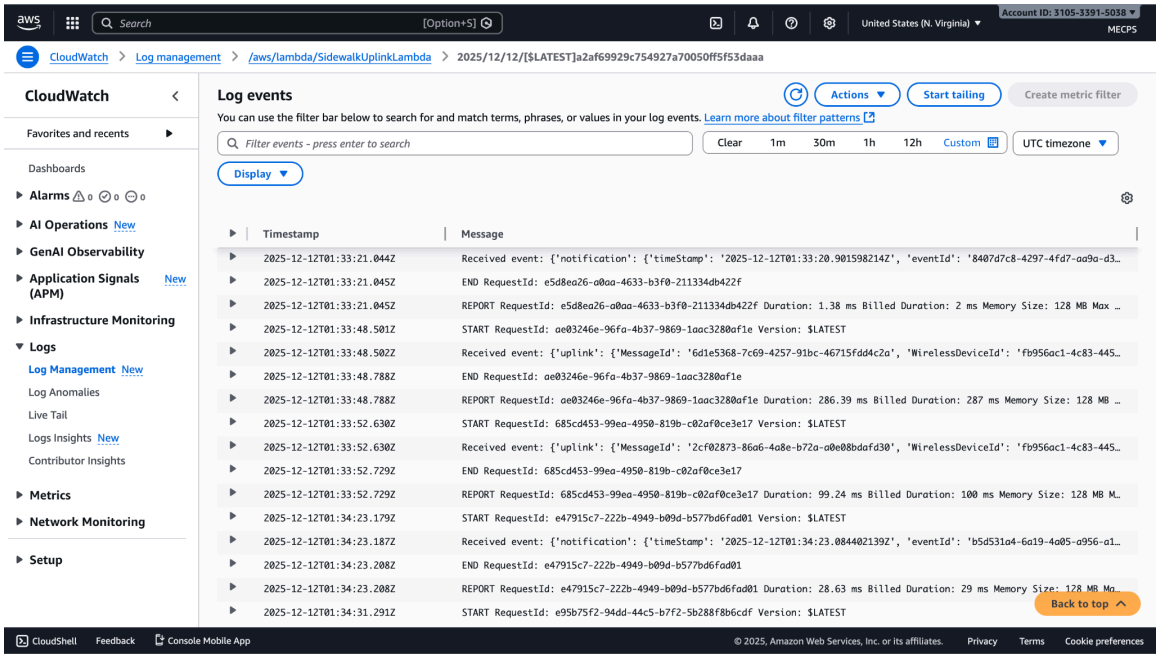

Figure 29. CloudWatch log events of Uplink Lambda

Analysis: Monitoring path validated; confirms visibility of failures and alert delivery. If alarm didn't fire, check alarm thresholds, metric namespaces, and IAM permissions for alarm actions.

Test Case 9: UI Tests with Mock Data

Objective: Verify dashboard renders and binds styles when API responses are mocked.

Parameter	Details
Input	/api/metrics and /api/incidents returning sample payloads via Jest/Playwright fixtures
Expected Output	Cards, line chart, and donut render with mock values; DOM contains expected CSS classes (e.g., .card, .badge--up, .legend-item)
Actual Output	UI rendered with mock numbers; CSS classes present; no console errors or missing assets
Result	PASS

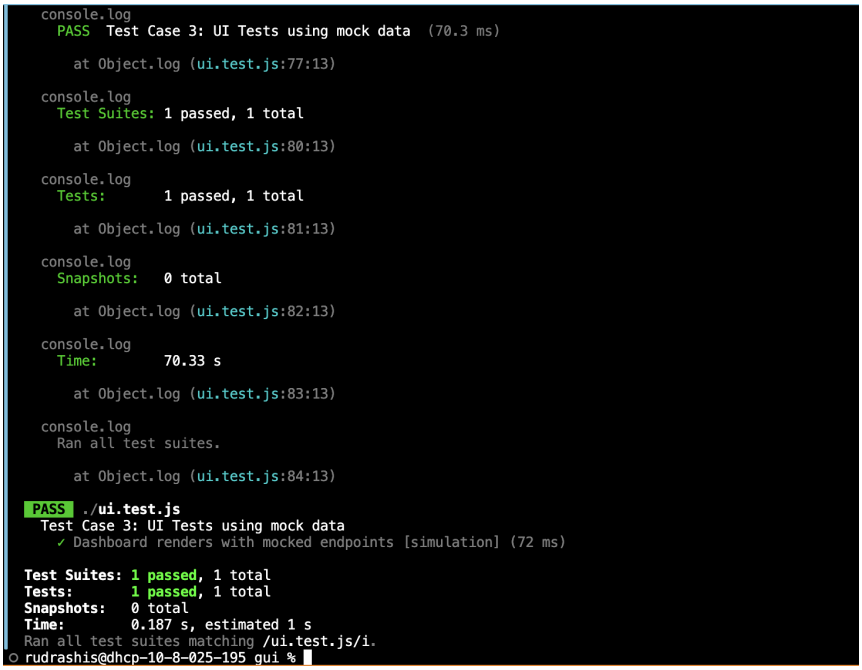

Figure 30. Test suite report showing successful UI component tests

Analysis: Rendering and style bindings work with mocked endpoints; proceed to edge states (empty/error) and date-range refetch.

Test Case 10: End-to-End Dashboard Updates from Device Messages

Objective: Verify that sensor messages transmitted from the device via Sidewalk are processed through the cloud pipeline and reflected in real-time on the caregiver dashboard UI.

Parameter	Details
Input	Live sensor messages from device (Regular, Help, Emergency) transmitted via LoRa to AWS IoT Core
Expected Output	Dashboard displays updated vitals, fall alerts appear in notification panel, help requests shown, location updated on map
Actual Output	All message types correctly displayed on dashboard; fall detection triggered notification; panic button help request received; GPS location shown on map
Result	PASS

Live demo sequence:

Step	Action	Device Output	Dashboard Response
1	System running	Regular packets transmitting with HR, SpO2, temperature values	Heart rate and temperature data displayed in real-time
2	Board dropped	*** FALL DETECTED *** - Emergency message ('E') sent via Sidewalk	Fall detection notification appears on dashboard
3	Alert acknowledgment	-	Caregiver acknowledges fall detection alert on dashboard
4	Panic button pressed	button 1 pressed - Help message ('H') sent via Sidewalk	"Help Requested" notification appears on dashboard
5	Location verification	GPS coordinates transmitted	Map displays current location (University Hall, UCI campus)

See the live demonstration here: https://www.youtube.com/watch?v=hqsvnUtx6RI

Analysis:

- **End-to-End Data Flow Verified:** Sensor data from wearable device successfully transmitted via Amazon Sidewalk over LoRa to AWS IoT Core and displayed on dashboard
- **Real-time Vitals Monitoring:** Heart rate, SpO2, and temperature values updated in real-time as regular packets arrive
- **Fall Detection Working:** Physical drop of the board triggered the 3-phase fall detection algorithm (free-fall → impact → stillness confirmation), sending Emergency message that appeared as dashboard notification
- **Alert Acknowledgment:** Caregiver successfully acknowledged fall detection alert through dashboard interface
- **Panic Button Functional:** BTN1 press immediately sent Help message via Sidewalk, appearing as "Help Requested" notification on dashboard

- **GPS Location Accurate:** Device location correctly displayed on map showing University Hall at UC Irvine campus (coordinates: 33.6459, -117.8429)

- **Complete System Integration:** All components (hardware sensors, firmware, Sidewalk network, AWS cloud, web dashboard) working together seamlessly

7. References

[1] Centers for Disease Control and Prevention, "Older Adult Fall Prevention," CDC Injury Center, 2023. [Online]. Available: https://www.cdc.gov/falls/

[2] World Health Organization, "Falls," WHO Fact Sheets, April 2021. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/falls

[3] National Council on Aging, "Falls Prevention Facts," NCOA, 2023. [Online]. Available: https://www.ncoa.org/article/get-the-facts-on-falls-prevention

[4] Merck Manual, "Falls in Older Adults," Merck Manuals Professional Edition, 2023. [Online]. Available: https://www.merckmanuals.com/professional/geriatrics/falls-in-older-adults/

[5] Amazon, "Amazon Sidewalk," Amazon Devices, 2024. [Online]. Available: https://www.amazon.com/Amazon-Sidewalk

[6] Amazon Web Services, "Amazon Sidewalk Documentation," AWS IoT Core for Amazon Sidewalk, 2024. [Online]. Available: https://docs.aws.amazon.com/iot/latest/developerguide/iot-sidewalk.html

[7] Semtech Corporation, "What is LoRa?" Semtech LoRa Technology, 2024. [Online]. Available: https://www.semtech.com/lora/what-is-lora

[8] Amazon, "Amazon Sidewalk Privacy and Security Whitepaper," Amazon Devices, 2023. [Online]. Available: https://www.amazon.com/sidewalk/security

[9] Nordic Semiconductor, "Amazon Sidewalk SDK v2.7.0 Documentation," Nordic DevZone, 2024. [Online]. Available: https://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/sidewalk/index.html

[10] Nordic Semiconductor, "nRF Connect SDK v2.7.0 Release Notes," Nordic Semiconductor, 2024. [Online]. Available: https://developer.nordicsemi.com/nRF_Connect_SDK/doc/2.7.0/nrf/releases_and_maturity/releases/release-notes-2.7.0.html

[11] Semtech Corporation, "Amazon Sidewalk Resources," Semtech Developer Portal, 2024. [Online]. Available: https://www.semtech.com/amazon-sidewalk

[12] Nordic Semiconductor, "sdk-sidewalk," GitHub Repository, 2024. [Online]. Available: https://github.com/nrfconnect/sdk-sidewalk