📑 Project Documentation (Protocol Decoder App)

1. Cover Page

- Title: **Protocol Decoder App**

---

2. Abstract

- A mobile application built with **Android (Kotlin + Jetpack Compose)**.
- It decodes **protocol frames in HEX format**.
- Features: parse header, extract payload, validate checksum & CRC.
- Helps developers and testers quickly verify communication frames.

---

3. Introduction

- **Background**: In embedded systems and IoT, devices communicate using custom protocols with start/end flags, checksums, and CRC validations.
- **Problem Statement**: Manual decoding of protocol frames is error-prone and time-consuming.
- **Objective**: Build an Android app that parses, validates, and visualizes protocol frames.
- **Scope**: Useful for IoT engineers, testers, and students working on serial/network communication.

---

4. Literature Review / Existing System

- Existing tools: desktop protocol analyzers (Wireshark, Serial Monitor).
- Limitation: Not mobile-friendly, require setup.
- Gap: A lightweight **mobile decoder** for quick checks.

---

5. Proposed System

- **Overview**: Android app that accepts hex input and validates frames
- **Features**:
  - Paste predefined example
  - Parse header (device ID, command, length)
  - Extract payload
  - Validate checksum & CRC
  - Display results in a clean UI
- **Advantages**: Portable, lightweight, fast validation on mobile.

---

6. System Analysis

- **Functional Requirements**:
  - User can enter hex frame
  - System must parse header
  - System must extract payload
  - System must validate checksum and CRC
  - Display results in readable format

- **Non-Functional Requirements**:
  - Usability: Simple UI with Compose
  - Performance: Real-time parsing
  - Portability: Android support
- **Constraints**: Requires valid hex input, frame must start/end with `0x7E`.

---

7. System Design

- **Architecture**:
  - UI Layer → `ProtocolDecoderScreen`

    - Logic Layer → `ProtocolUtils` + parsing functions
    - Data Models → `ProtocolFrame`, `ParseResult`
- **Diagrams**:
    - UML Class Diagram (MainActivity, ProtocolDecoderScreen, ProtocolUtils, ProtocolFrame)
    - Flowchart (Input HEX → Parse → Validate → Display Result)
- **Database**: Not applicable (stateless app).

---

8. Implementation

- **Technology Stack**:
    - Language: Kotlin
    - Framework: Jetpack Compose
    - Tools: Android Studio, Gradle
- **Core Modules**:
    - `MainActivity`: Entry point
    - `ProtocolDecoderScreen`: Compose UI
    - `ProtocolUtils`: Helper functions (header parse, payload extraction, CRC validation)
    - `ProtocolFrame`: Data class to hold frame details
    - `ParseResult`: Rich validation result model
- **Code Highlights**:
    - Hex sanitization and conversion (`hexToBytes`)
    - Checksum calculation (`calcChecksumOver`)
    - CRC16-CCITT validation (`crc16Ccitt`)

---

9. Testing

- **Test Strategy**:
  - Unit Testing (hex parsing, checksum validation, CRC)
  - Integration Testing (UI + Utils interaction)
- **Sample Test Cases**:
  - Input: `"7E 01 10 02 1A 2B 58 12 34 7E"` → Expected: Valid frame ✅
  - Input: Missing `0x7E` → Expected: Error "Missing start/end"
  - Input: Wrong checksum → Expected: "Invalid Frame"
- **Results**: App correctly validates frames and displays results.

---

10. Results & Discussion

- Successfully parses and validates protocol frames.
- Shows Device ID, Command, Payload, and Validation results.
- UI is clean, responsive, and user-friendly.

---

11. Conclusion

- Project achieved its objective of decoding protocol frames on Android.
- Reduces manual errors and improves productivity for developers.

---

12. Future Scope

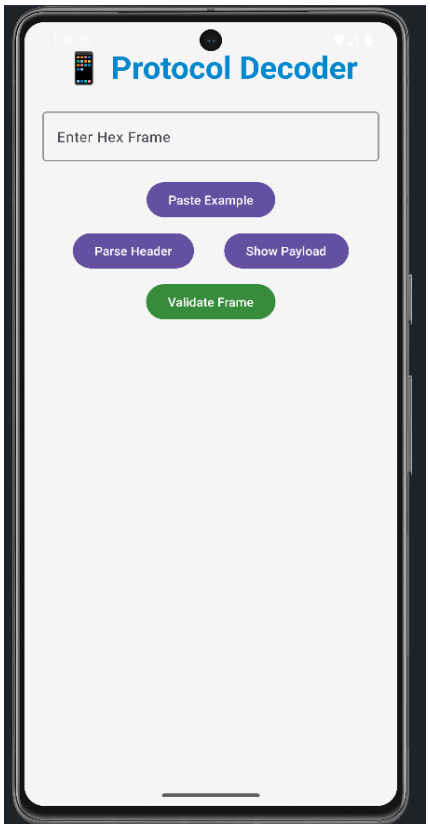
- Support for more protocol formats (Modbus, CAN, custom IoT).
- Save decoded frames history.
- Export results as CSV/PDF.
- Dark mode support.
- Integration with external devices (via Bluetooth/USB).

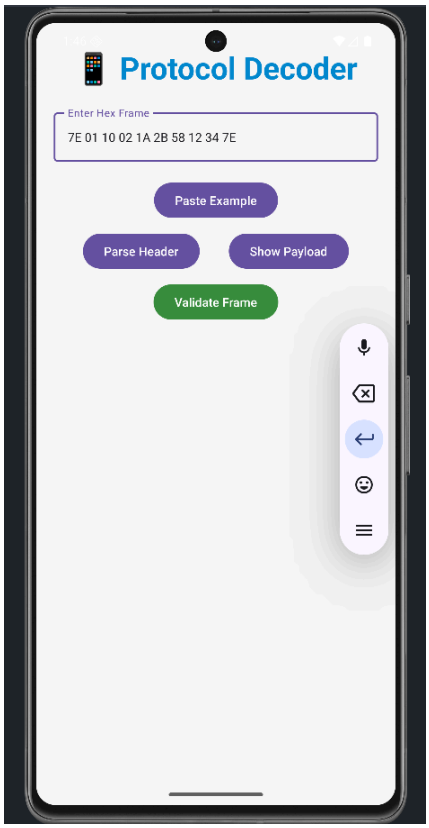
13. References

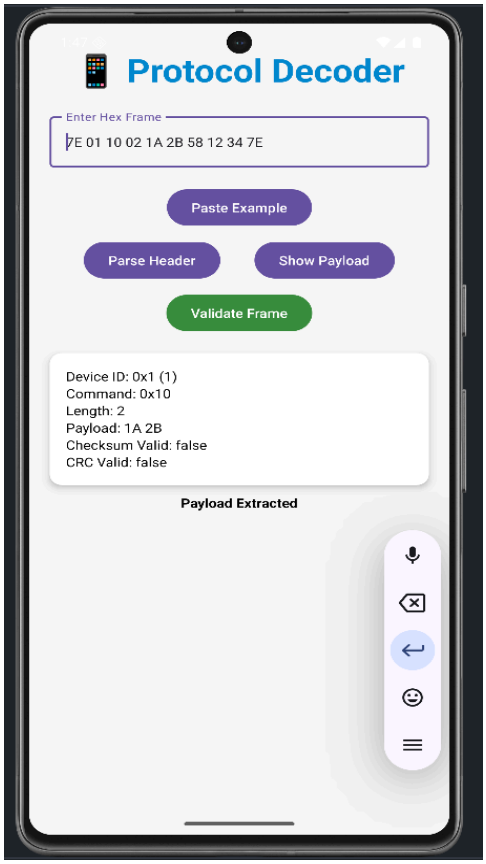
- Android Developers Docs (Jetpack Compose, Kotlin)
- CRC16-CCITT standard documentation
- Networking/IoT protocol references

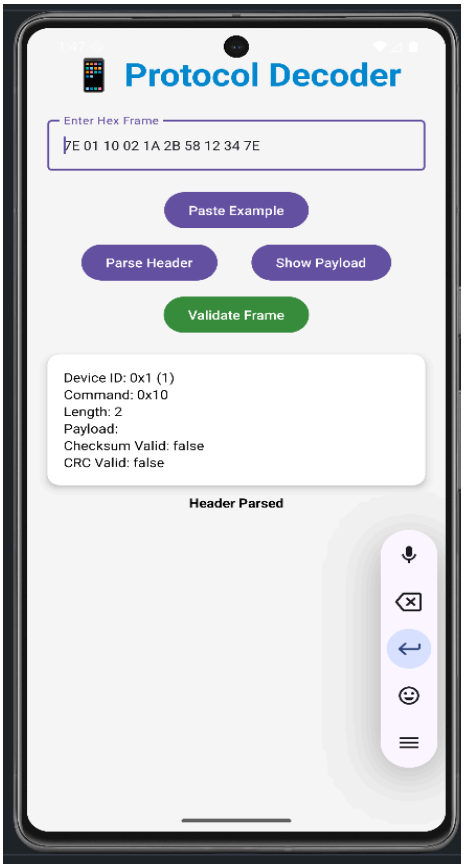
14. Appendix

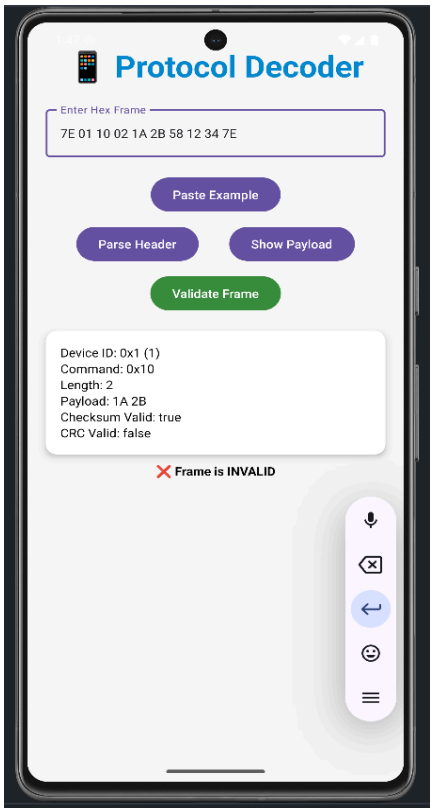
- Full source code (already provided).
- Screenshots of the running app.






Protocol Decoder
Enter Hex Frame
7E 01 10 02 1A 2B 58 12 34 7E
Paste Example
Parse Header
Show Payload
Validate Frame
Device ID: 0x1 (1)
Command: 0x10
Length: 2
Payload: 1A 2B
Checksum Valid: false
CRC Valid: false
Payload Extracted


Protocol Decoder
Enter Hex Frame
7E 01 10 02 1A 2B 58 12 34 7E
Paste Example
Parse Header
Show Payload
Validate Frame
Device ID: 0x1 (1)
Command: 0x10
Length: 2
Payload:
Checksum Valid: false
CRC Valid: false
Header Parsed


Protocol Decoder
Enter Hex Frame
7E 01 10 02 1A 2B 58 12 34 7E
Paste Example
Parse Header
Show Payload
Validate Frame
Device ID: 0x1 (1)
Command: 0x10
Length: 2
Payload: 1A 2B
Checksum Valid: true
CRC Valid: false
Frame is INVALID