High-Level Design (HLD) Document
Project: foip_direct_demo
Version: 1.0
Author: [Author Name]
Date: [Insert Date]
Reviewers/Approvers: [Insert Names]
Related Documents/References: See Section 7
Source Document Reference: ORIGINAL_README.md

---

1. System Overview

1.1 Purpose
- Demonstrate a simplified Fax Over IP (FOIP) workflow using stubbed PJSIP (SIP signaling) and SpanDSP (T.30 fax protocol) libraries.
- Showcase the basic process of establishing a SIP call, encoding fax data, transmitting via UDPTL, and terminating the session.

1.2 Scope
- Targeted for embedded systems and demonstration environments.
- Designed for single-session, sender-only FOIP operation.
- Uses stubbed protocol libraries for demonstration, not production.

1.3 Primary Functions
- Establish SIP calls (simulate SIP INVITE/BYE).
- Encode and transmit fax pages using T.30/T.38 protocols.
- Send encoded fax data over UDPTL transport.
- Orchestrate the complete FOIP workflow from call setup to teardown.

1.4 Key Characteristics
- **Language:** C++17
- **Portability:** POSIX-compliant; buildable on Linux-like systems (CMake 3.10+).
- **Memory Footprint:** Not explicitly defined; expected to fit within standard embedded Linux constraints (RAM ≤ 256 KB assumed for stub/demo).
- **Supported Protocols:** SIP (RFC 3261, simulated), T.30 (ITU-T, simulated), T.38 (ITU-T, simulated), UDPTL (RFC 3362, simulated).

---

2. Architecture Diagram Description

Explanation
- The system is structured in three main layers:
  1. **SIP Signaling Layer (PJSIP Stub):** Handles call setup, UDPTL packet transmission, and call teardown.
  2. **Fax Protocol Layer (SpanDSP Stub):** Manages T.30 fax session, page encoding, and session termination.
  3. **Main Application:** Orchestrates the workflow, coordinating between SIP and fax protocol layers.

Interactions
- The Main Application invokes the SIP layer to establish a call, then initializes the fax session via the Fax Protocol layer.
- Encoded fax pages are passed from the Fax Protocol layer to the SIP layer for UDPTL transmission.
- The Main Application manages session and call termination by invoking the appropriate APIs in both layers.

---

3. Module Breakdown

3.1 PJSIP Stub Module
- **Responsibility:** Simulate SIP signaling and UDPTL transport.
- **Inputs:** Phone number (for call), encoded fax data (for UDPTL).
- **Outputs:** Boolean status for each operation.
- **Key Dependencies:** None (stubbed).
- **API Examples:**
  - `bool pjsip_make_call(const std::string& number);`
  - `bool pjsip_send_udptl(const uint8_t* data, size_t len);`
  - `bool pjsip_hangup();`

3.2 SpanDSP Stub Module
- **Responsibility:** Simulate T.30 fax session management and page encoding.
- **Inputs:** Raw page data (as vector of bytes).
- **Outputs:** Encoded T.30 frames (as vector of bytes), boolean status.
- **Key Dependencies:** None (stubbed).
- **API Examples:**
  - `bool spandsp_start_fax_session();`
  - `bool spandsp_encode_page(const std::vector<uint8_t>& page, std::vector<uint8_t>& outFrames);`
  - `bool spandsp_end_fax_session(std::vector<uint8_t>& finalFrames);`

3.3 Main Application
- **Responsibility:** Orchestrate the FOIP workflow.
- **Inputs:** User triggers, sample page data.
- **Outputs:** Console logs, workflow status.
- **Key Dependencies:** PJSIP Stub, SpanDSP Stub.
- **Workflow Steps:**
  1. Establish SIP call.
  2. Start fax session.
  3. Encode and send fax page.
  4. Finalize session and send termination frames.
  5. Hang up call.

---

4. Interfaces

4.1 Hardware Interfaces
- No direct hardware interfaces; operates at application layer.
- MAC driver abstraction not required for stub/demo.

4.2 Communication Buses
- No physical buses (UART, SPI, I2C, CAN, Ethernet) directly used.
- All communication is simulated within the application.

---

5. Constraints and Assumptions

5.1 Hardware Constraints
- **CPU:** Standard embedded Linux-class CPU.
- **Memory:** RAM ≤ 256 KB (assumed for stub/demo; not explicitly specified).
- **Storage:** Sufficient for application binary and logs.

5.2 Timing Constraints
- No real-time or latency requirements specified.
- Throughput and latency are not measured in the stub/demo.

5.3 Power/Environment Limits
- No explicit voltage or temperature constraints.
- Assumed to run in standard lab/office environment.

5.4 Assumptions
- IPv4 only; no IPv6 support.
- Single default route for network traffic.
- All protocol stacks are stubbed; no actual network or hardware dependencies.
- Only sender role is implemented; no receive functionality.

---

6. Non-Functional Requirements (NFRs)

6.1 Performance
- No explicit throughput or latency targets.
- Expected to complete demo workflow in <1 second on standard desktop hardware.

6.2 Reliability
- Boolean return codes for error handling.
- RAII principles for resource cleanup.
- Console logging for all operations.

6.3 Safety
- No explicit safety standards required for demo/stub.

6.4 Security
- No authentication or encryption in stub implementation.
- For production: SIP authentication, TLS/SRTP, access control required.

6.5 Power Usage
- Not optimized for low power; standard desktop/embedded usage.

6.6 Maintainability
- Modular design: clear separation between SIP, fax protocol, and application logic.
- Easy to replace stubs with real libraries.

6.7 Testing Strategy
- **Unit Testing:** Not explicitly implemented; recommended for real modules.
- **Integration Testing:** Verify interaction between PJSIP and SpanDSP stubs.
- **Protocol Compliance:** Simulated only.
- **Performance Testing:** Not applicable for stub/demo.
- **Stress Testing:** Not applicable for stub/demo.
- **Security Testing:** Not applicable for stub/demo.
- **Hardware-in-the-Loop Testing:** Not applicable.
- **Regression Testing:** Manual verification via console output.

---

7. Metadata

- **Document Title:** FOIP Direct Demo High-Level Design (HLD)
- **Version:** 1.0
- **Author:** [Author Name]
- **Date:** [Insert Date]
- **Reviewers/Approvers:** [Insert Names]
- **Related Documents/References:**
  - ITU-T T.30, T.38
  - RFC 3261 (SIP)
  - RFC 3362 (T.38 over SIP)
  - PJSIP and SpanDSP documentation (for real implementation)
- **Source Document Reference:** ORIGINAL_README.md

---

8. Compliance Matrix Section

Spec Requirement	HLD Section	Pass/Fail	Comments
C++17 Language	1.4	Pass	Explicitly stated
CMake 3.10+ Build System	1.4, 7	Pass	Explicitly stated
SIP Signaling (Stub)	3.1	Pass	API and responsibilities defined
T.30 Fax Protocol (Stub)	3.2	Pass	API and responsibilities defined
UDPTL Transport (Stub)	3.1	Pass	API and

responsibilities defined |
End-to-End FOIP Workflow	3.3	Pass	Workflow steps detailed
Error Handling via Boolean Returns	6.2	Pass	Described in NFRs
Console Logging	6.2	Pass	Logging format described
No Authentication/Encryption (Stub)	6.4	Pass	Security limitations noted
IPv4 Only, Single Route	5.4	Pass	Assumptions section
Memory Footprint (≤256 KB, assumed)	1.4, 5.1	Pass	Not explicit, but fits demo scope
Protocol Compliance (Simulated)	6.7	Pass	Simulated only
Hardware/Bus Interfaces	4	Pass	Not applicable, clarified
Performance Targets	6.1	Pass	Not required for demo
Modular Design	6.6	Pass	Explicitly described
Testing Strategy	6.7	Pass	Outlined for demo context

Deviation Summary Section

Deviation/Gaps	Recommended Actions/Resolution
No explicit memory/CPU constraints in spec	Confirm with stakeholders if demo limits suffice; add explicit targets for production
No receive (Rx) functionality implemented	Add receive path in future enhancement phase
No authentication/encryption in stub	Implement SIP authentication, TLS/SRTP for production
No multi-page or concurrent session support	Extend architecture for scalability as needed
No unit/integration test automation	Add automated tests for real implementation
No configuration file support	Add configuration management in future
No structured logging framework	Implement logging framework for production
No explicit power/environmental constraints	Confirm if required for target deployment

---

Actionable Review Notes

1. **Memory/CPU Constraints:**
   - Action: Confirm with stakeholders if the assumed RAM ≤ 256 KB and standard embedded CPU are sufficient for the demo and future production.
   - Next Step: Specify explicit memory and CPU targets for production if needed.

2. **Receive Functionality:**
   - Action: The current design is sender-only.
   - Next Step: Define requirements and design for receive (Rx) path if bidirectional FOIP is needed.

3. **Security Features:**
   - Action: Stub implementation lacks authentication and encryption.
   - Next Step: For production, specify and implement SIP authentication, TLS/SRTP, and access control.

4. **Testing Automation:**
   - Action: No automated unit/integration tests are defined.
   - Next Step: Develop and integrate automated tests for real modules.

5. **Configuration Management:**
   - Action: No configuration file support.
   - Next Step: Add configuration management for runtime flexibility.

6. **Scalability:**
   - Action: No support for multi-page or concurrent sessions.
   - Next Step: Extend architecture for scalability as per future requirements.

7. **Logging Framework:**
   - Action: Only console logging is implemented.
   - Next Step: Add structured logging with levels for production.

8. **Power/Environmental Constraints:**
   - Action: Not specified.
   - Next Step: Confirm if any are required for the intended deployment environment.

Optional Consolidated Summary

This HLD describes a modular, demonstration-focused FOIP system using stubbed protocol libraries. The design is suitable for embedded or desktop environments, with clear separation between SIP signaling, fax protocol handling, and application orchestration. While the current implementation is sender-only and lacks production-grade security, scalability, and configuration features, the architecture allows for straightforward extension. All requirements from the specification are mapped and met for the demo context, with clear next steps

provided for any missing or ambiguous details.