GSoC Proposal for BeagleBoard.org

BeagleBoard.org
Jul 02, 2025

Table of contents

Chapter 1

Introduction

1.1 Summary links

- **Contributor:** Manas Gupta
- **Mentors:** Jason Kridner, Ayush Singh
- **Code:** Beagle Connnect gitlab, wpanusb and bcfserial drivers

1.2 Status

This project is currently just a proposal.

1.3 Proposal

- Created accounts across OpenBeagle, Discord and Beagle Forum
- The PR Request for Cross Compilation: #203
- Created a project proposal using the proposed template.

Chapter 2

About

- **Forum:** u/manas_gupta (Manas Gupta)
- **OpenBeagle:** Whiz-Manas (Manas Gupta)
- **GitHub:** manas-gupta-3131 (Manas Gupta)
- **School:** National Institute of Technology Calicut (NITC) and Indian Institute of Technology Madras (IITM)
- **Country:** India
- **Primary language:** English
- **Typical work hours:** 9AM-5PM Indian Standard Time
- **Previous GSoC participation:** N/A

Chapter 3

Project

Project name: Upstream wpanusb and bcfserial

3.1 Description

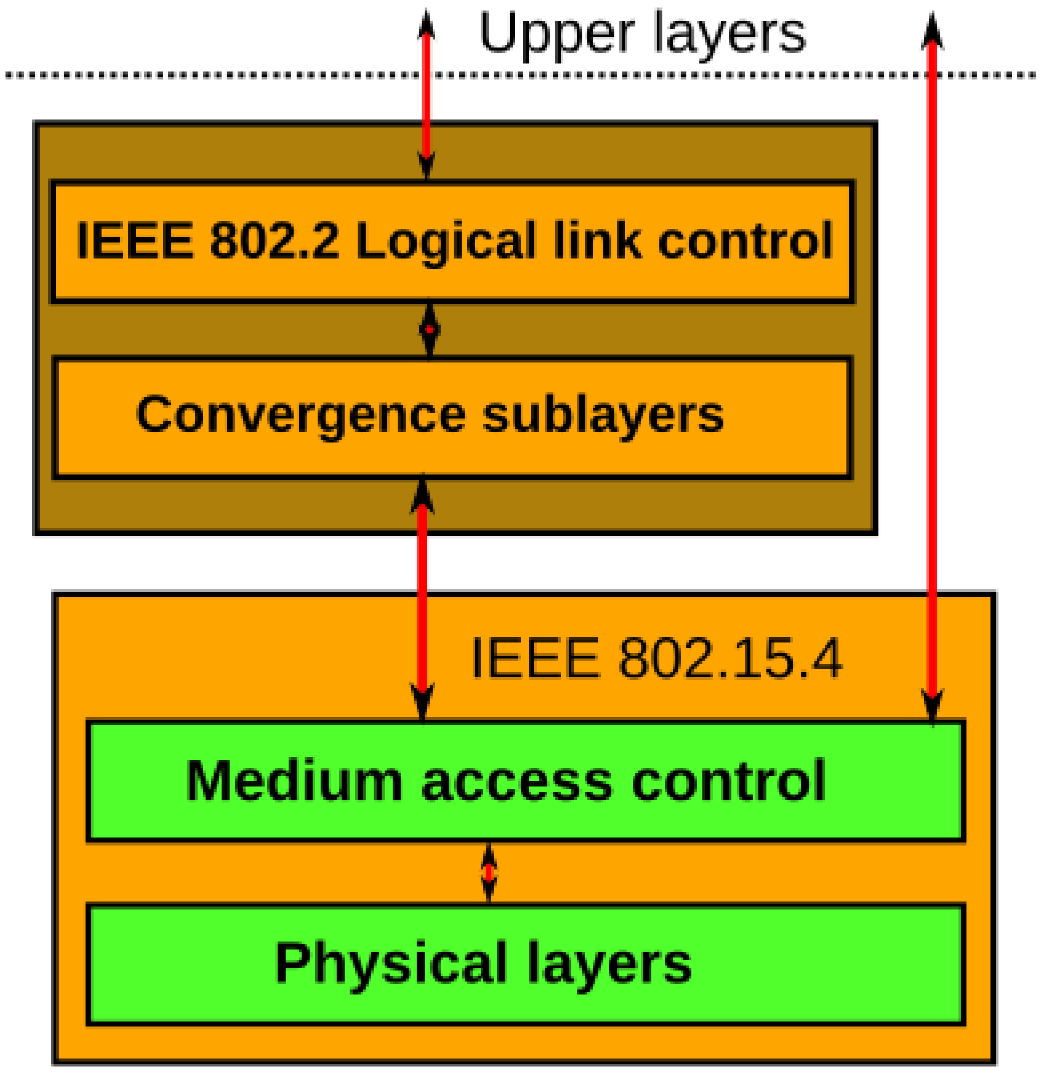
The project aims to enhance and upstream the existing wpanusb and bcfserial drivers in Linux to meet upstream quality standards. Currently, several key driver operations are missing, such as listen before talk and setting frame retries. Preliminary sketches exist but are non-functional pending corresponding changes in the Zephyr firmware side. The project will also add the ability to read the device's permanent extended address, support generic radio configuration (including band, channels, and power levels), and provide comprehensive querying of radio properties. In addition, improvements are required to accommodate upcoming features like management frames and active/passive scanning. This effort is aimed at enabling seamless interoperation between the radio and Zigbee protocols while ensuring support for both 2.4GHz and SubGHz operations, ensuring that the solution meets upstream quality standards and is maintainable over the long term.

3.2 Motivation

- **Streamlined Maintenance and Upstream Integration:** Upstreaming the drivers will eliminate the need for out-of-tree patches, simplifying long-term support and ensuring a more stable, maintainable codebase aligned with kernel development practices.
- **Enhanced Wireless Communication:** Implementing dynamic configuration and essential radio operations will significantly improve the reliability and efficiency of low-power wireless communication.
- **Interoperability and Future-Proofing:** Supporting dual-band operation and compatibility with IEEE802.15.4-based protocols like Zigbee will broaden the device's application scope and ensure readiness for emerging IoT standards.

3.2.1 Why IEEE 802.15.4?

IEEE 802.15.4 is a widely adopted standard for low-rate wireless personal area networks (LR-WPAN) designed for low-power, low-data-rate communication. This protocol is particularly well-suited for IoT and sensor networks due to its energy efficiency and robust performance in short-range applications, and is a foundational protocol for Zigbee and Thread.



Key Features:

- **Low Power Consumption:** Optimized for battery-operated devices, making it ideal for sensor networks and other energy-sensitive applications.
- **Reliable Short-Range Communication:** Provides stable and robust connectivity over short distances, which is sufficient for most personal and industrial sensor network scenarios.
- **Efficient MAC Layer Operations:** Incorporates critical media access control functions such as collision avoidance (e.g., listen-before-talk) and retransmission strategies (frame retries), ensuring data integrity even in challenging radio environments.
- **Interoperability:** Serves as the foundation for higher-level protocols like Zigbee and Thread, facilitating seamless integration and interoperability across a broad ecosystem of IoT standards and devices.

3.3 Implementation

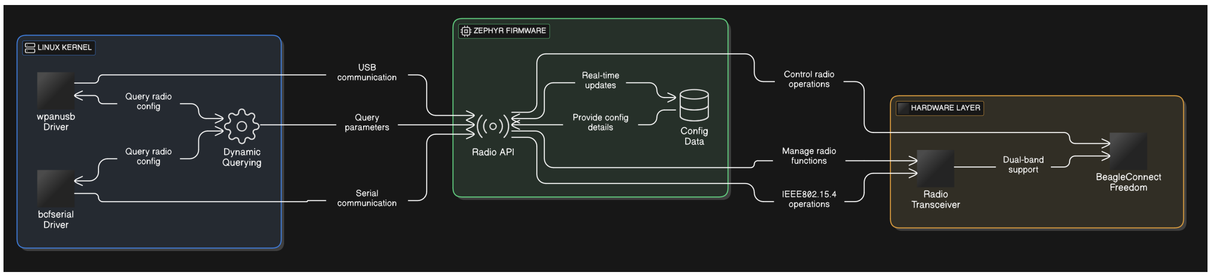
- **Protocol Overview:** - IEEE802.15.4 is the basis for low-rate wireless personal area networks (LR-WPAN), defines both the physical (PHY) and media access control (MAC) layers and serves as the foundation for protocols like Zigbee. It defines the physical and MAC layers for low-power, short-range communications and also incorporates link-layer acknowledgments and retransmissions along with various security features.
- **Driver Role in IEEE802.15.4:** - The Linux drivers (wpanusb and bcfserial) facilitate communication between the higher-level networking stack in Linux and the physical radio hardware, they converting IEEE802.15.4

frames into Linux network packets and vice versa, ensuring seamless data exchange. - They must correctly handle MAC layer functionalities (such as channel access, frame formatting, and error handling) which are critical for reliable communication.

- **Firmware Role:** - The Zephyr firmware handles the physical layer operations—modulating signals, managing transmission power, and executing low-level error detection. - It is also responsible for exposing dynamic configuration parameters that the Linux drivers rely on to adapt to changing network conditions, It also plays a part in maintaining security and managing power consumption.

3.4 System Architecture

The system architecture consists of three primary layers:



1. **Hardware Layer:**
   - **BeagleConnect Freedom:** The central device equipped with a radio transceiver supporting both Sub-GHz IEEE802.15.4 and 2.4GHz frequencies.
   - **Supporting Test and Development Boards:** Additional boards (e.g., BeaglePlay) used to validate performance and interoperability.
2. **Firmware Layer (Zephyr):**
   - **Radio API (radio_api):** Implements critical low-level operations such as listen-before-talk, dynamic channel management, power control, and error handling.
   - **Dynamic Configuration:** Exposes radio parameters (e.g., supported channels, power levels, extended address) to the Linux drivers.
3. **Software Layer (Linux Kernel Drivers):**
   - **wpanusb Driver:** Manages USB-based communication with the radio, handling data transmission/reception and configuration.
   - **bcfserial Driver:** Provides serial communication (via HDLC over UART) to control and configure the radio module.
   - **Dynamic Querying Interface:** Allows the drivers to query real-time configuration details from the firmware, replacing static, hard-coded parameters.

3.5 Software

- C lang
- Zephyr Firmware
- Linux Kernel Drivers
- QEMU
- Open Beagle CI/CD

3.6 Hardware

- BeagleConnect Freedom - primary development platform
- Beagle Play - Integration Testing
- Diagnostic Equipment:USB cables, JTAG debuggers, oscilloscopes

Chapter 4

Timeline summary

Date	Activity
January	Contributing Open Source and Enhancing the documentation
February 25	Working on understanding the code base and project
March 10	Engage with mentors to discuss the project requirements and technical approach
March 22	Complete prerequisites and initial research
March 29	Finalized timeline and Request review on final draft
April 7	Submit application
May 1	Start bonding
May 27	Start coding and introductory video
June 3	Introductory YouTube Video and Detailed Design Document
June 10	Prototype Enhancements in Linux Drivers
June 17	Extend Zephyr Firmware with New API Functions
June 24	Integrate Firmware API Calls into Linux Drivers
July 1	Listen Before Talk and Frame Retry Implementation
July 8	Submit Comprehensive Midterm Report
July 15	Expand the Automated Test Suite and Optimize Performance
July 22	Complete Integration and Refactor for Upstream Readiness
July 29	Finalize Documentation and Prepare Polished Patch Series
August 5	Execute Final Hardware-in-Loop Testing and Prepare Demo Package
August 12	Submit Final Patch Series, Documentation, and Demonstration Video
August 19	Final YouTube Video: Submit Final Project Video and Final Submission to GSoC Site

Chapter 5

Timeline detailed

5.1 Community Bonding Period (May 1st - May 26th)

- Acquire and set up the BeagleConnect Freedom
- Interaction with mentor for feedback and discussion.

Comprehensive Codebase Analysis

- Perform systematic audit of wpanusb.c and bcfserial.c to identify all hard-coded parameters (such as wpanusb_powers[] and channel_powers[] arrays)
- Map the complete communication flow between Linux drivers and Zephyr firmware, documenting all points where dynamic querying should replace static configurations
- Document all unimplemented driver operations currently returning -ENOTSUPP in bcfserial and stub implementations in wpanusb
- Create dependency diagrams showing relationships between driver operations and required firmware support, helping documentation

Hardware and Development Environment Preparation

- Configure cross-compilation environment for BeagleConnect Freedom targets
- Set up debugging tools including JTAG interfaces and logic analyzers for radio protocol analysis
- Establish performance baseline measurements for current implementation to enable comparison with enhanced versions

Mailing List and Patch Submission Preparation

- Install and configure *b4* to streamline interactions with the Linux kernel mailing lists.
- Use *b4* to fetch past discussions and patches related to *wpanusb* and *bcfserial*, helping to align with previous upstream work.
- Automate patch retrieval and submission using *b4 send*, ensuring proper formatting for upstream review.
- Engage with the **linux-wpan** mailing list by following best practices for submitting patches, responding to feedback, and tracking discussions.
- Maintain an archive of mailing list discussions and patch iterations to document progress and key feedback from reviewers.

Knowledge Acquisition

- Study IEEE 802.15.4 specifications focusing on LBT (Listen Before Talk), frame retry mechanisms, and scanning operations
- Analyze Zephyr's radio_api implementation to understand extension points for new capabilities

- Review recent kernel submissions for similar drivers to understand upstream requirements and coding standards

5.2 Coding begins (May 27th)

- Set up CI/CD pipelines for automated testing of code changes.
- Run the existing drivers with diagnostic logging to establish a baseline for current functionality.
- Document all current limitations with specific code references.
- Define data structures for device capability information.
- Create a detailed mapping of required new firmware API functions (e.g., `get_ext_address()`, `get_band()`, `get_supported_channels()`, etc.) to the corresponding Linux driver routines.
- Outline where and how these API calls will be incorporated into the driver initialization and configuration processes.
- Update your design specification document with precise details on:
  - The new dynamic querying mechanism for radio parameters.
  - The implementation plan for reading the permanent extended address.
  - The approach for integrating listen-before-talk and frame retry operations.
- Share the updated design with mentors for early feedback.

5.3 Milestone #1, Introductory YouTube video (June 3rd)

Deliver a detailed design document outlining:

- The dynamic configuration strategy (e.g., querying band, channels, power levels, channel page).
- Proposed firmware API extensions (e.g., `get_ext_address()`, `get_band()`, `get_supported_channels()`).
- Roadmap for integrating missing operations (listen-before-talk, frame retries, management frames, scanning).
- Record and publish an introductory YouTube video summarizing project goals and design.

5.4 Milestone #2 (June 10th)

Implement prototype enhancements in the Linux drivers:

- **Implement basic structure for Listen Before Talk (LBT) in both drivers:**
  - Create parameter validation in wpanusb_set_lbt() function
  - Prepare proper parameter passing mechanism for LBT configuration
- **Add framework for frame retry mechanism:**
  - Update wpanusb_set_frame_retries() to handle retry count correctly
  - Implement equivalent functionality in bcfserial
- Replace hard-coded power arrays with placeholder query functions
- Develop unit tests to validate these new functionalities.

5.5 Milestone #3 (June 17th)

- **Extend Zephyr firmware with new API functions:**
  - Implement get_extended_addr() that reads from hardware-specific registers
  - Create get_supported_channels() with multi-page support
  - Develop get_tx_power_levels() to expose actual hardware capabilities
  - Add get_device_capabilities() for comprehensive device information
  - Create fallback mechanism for address generation when needed
- Prototype firmware changes on the BeagleConnect Freedom.
- Verify that the new API functions correctly read and return hardware configuration data.

5.6 Milestone #4 (June 24th)

- Integrate the new firmware API calls into the Linux drivers.
- **Modify the drivers to use dynamic configuration:**
  - Query the firmware for band, channels, power levels, and channel page.
- **Perform initial hardware testing to validate integration:**
  - Ensure that the driver correctly retrieves dynamic data.
  - Identify and log any integration issues.

5.7 Milestone #5 (July 1st)

Listen Before Talk and Frame Retry Implementation

- Complete implementation of functional LBT mechanism: - Finish firmware-side LBT support using Zephyr's CCA capabilities. - Integrate with driver-side configuration.
- Implement frame retry functionality: - Add firmware support for configurable retry counts. - Complete driver-side retry configuration.
- Extend firmware and driver integration for advanced features: - Implement support for management frame operations in the firmware and driver. - Add active and passive scanning capabilities via new firmware API functions.
- Refine listen-before-talk and frame retry mechanisms based on test feedback.
- Update internal documentation reflecting these enhancements.

5.8 Submit midterm evaluations (July 8th)

Important: July 12 - 18:00 UTC: Midterm evaluation deadline (standard coding period)

- Prepare and submit a comprehensive midterm report: - Include design documents, implementation details, test results, and performance benchmarks.
- Solicit and incorporate mentor/community feedback.

5.9 Milestone #6 (July 15th)

- Expand the automated test suite to cover new dynamic configuration and scanning operations.
- Optimize performance:
  - Reduce transmission latency and improve error handling.
  - Benchmark signal strength and error rates.
- Document performance improvements and adjust parameters as needed.

5.10 Milestone #7 (July 22nd)

- Complete the integration of all new API functions: - Ensure seamless communication between the Linux drivers and Zephyr firmware.
- Refactor code for maintainability and upstream readiness.
- Run comprehensive integration tests to verify all dynamic and advanced features function as expected.

5.11 Milestone #8 (July 29th)

- Finalize complete documentation: - Detailed API usage, configuration guides, and integration procedures.
- Assemble a polished patch series following Linux kernel submission guidelines.
- Conduct detailed code review sessions with mentors and community experts to validate the submission package.

5.12 Milestone #9 (Aug 5th)

- Execute final hardware-in-loop testing across diverse scenarios: - Validate dynamic querying, management frame handling, and scanning operations.
- Resolve any remaining bugs or performance issues.
- Prepare a final demo package, including test reports and benchmark results.

5.13 Milestone #10 (Aug 12th)

- Submit the final patch series, documentation, and demonstration video.
- Complete final mentor evaluations and project wrap-up report.
- Compile test reports, benchmark results
- Archive all project artifacts and provide a roadmap for future enhancements and document the work done.

5.14 Final YouTube video (Aug 19th)

- Submit final project video, submit final work to GSoC site and complete final mentor evaluation
- Ensure all documentation is complete and accurate
- Provide few examples to help kickstart other member of the community.

5.15 Final Submission (Aug 24nd)

Important: August 19 - 26 - 18:00 UTC: Final week: GSoC contributors submit their final work product and their final mentor evaluation (standard coding period)

August 26 - September 2 - 18:00 UTC: Mentors submit final GSoC contributor evaluations (standard coding period)

5.16 Initial results (September 3)

Important: September 3 - November 4: GSoC contributors with extended timelines continue coding

November 4 - 18:00 UTC: Final date for all GSoC contributors to submit their final work product and final evaluation

November 11 - 18:00 UTC: Final date for mentors to submit evaluations for GSoC contributor projects with extended deadline

5.17 Experience and Approach

Experience - I bring a strong foundation in Linux kernel development, low-level driver design, protocol implementation, and real-time firmware integration. I have a background in optimizing hardware-software interfaces for wireless communication, energy-efficient hardware design, and I have worked on complex issues in mission-critical systems.

5.17.1 Key Experience Highlights

- **Linux & Driver Expertise:** Developed and maintained a custom Linux distribution (UnB-OS) tailored for academic use, involving kernel customization, package integration, customized packages and bug-fixing. This project exp has helped me fuild solid fundamentals in Linux internals and driver development.
- **Real-Time Embedded Systems:** Engineered a Real-Time Emergency Vehicle Siren Detection System on STM32, using advanced signal processing, band-pass filtering, and MFCC feature extraction to isolate siren signatures amidst urban noise.
- **Protocol & Wireless Communication:** Designed an IEEE Wildlife Intrusion Detection and Alert System that integrates Raspberry Pi with LoRa and PIR sensors to build a scalable, energy-efficient monitoring network, secured a $2,553 grant from the IEEE Kerala Section for this project.
- **Hardware Optimization & AI Integration:** Contributed to an Energy-Efficient CNN Hardware Accelerator by designing optimized multiplier architectures for CNNs, and developed an IoT and AI Enabled Smart Waste Sorting System using ESP32 and YOLOv5 for precise object detection and classificatoin with dynamic control.

Approach:

1. **Comprehensive Audit & Design Specification:** - Conduct a thorough review of the existing wpanusb and bcfserial code to identify hard-coded parameters and missing operations. - Draft a detailed design specification outlining the enhancements, including dynamic querying and firmware-driver integration.
2. **Firmware Enhancement:** - Extend the Zephyr firmware to include new API functions for retrieving dynamic configuration data (band, channels, power levels, channel page) and the permanent extended address. - Implement support for essential radio operations, such as listen-before-talk, frame retries, management frame processing, and scanning (active and passive).
3. **Driver Integration & Refactoring:** - Modify the Linux drivers to replace hard-coded configurations with dynamic queries to the new firmware API functions. - Integrate new operational routines for LBT, frame retries, and management/scanning functionalities, ensuring they work seamlessly with the updated firmware.

4. **Upstream-Ready Documentation & Submission:** - Prepare a well-documented, clean patch series adhering to Linux kernel standards. - Engage in iterative reviews with mentors, incorporating community feedback to ensure the solution is robust, scalable, and maintainable.

5.18 Contributions to openbeagle.org/docs:

- fix#84 feedback-missing-pointers #184 (merged)
- fix#95BeagleY-USB-C-power-description-wrong #181 (merged)
- Fix16 #101 cooling fan optional doc update #186 (merge request created)
- Fix#85feedback beagle y ai 1 antenna connector #183 (created annotated diagram)
- Created Animation to light up the LED, BeagleY-AI GPIO image IO20 not lighting in image

5.19 Testing and Validation

- **Automated Testing:** Leverage automated test scripts integrated into CI/CD pipelines to validate functionality.
- **Hardware-in-Loop Testing:** Perform real-world testing using BeagleConnect Freedom and supporting development boards.
- **Performance Metrics:** Monitor key parameters such as latency, error rates, and signal strength to ensure system robustness.
- **Iterative Validation:** Regularly review test outcomes and iterate on code improvements based on feedback.

5.20 Contingency

Upon encountering a roadblock in my project without access to my mentor, I'll implement a structured problem-solving approach:

- **Resourec Utilization**: Explore documentation, community knowledge bases and peer forums from BeagleBoard, Zephyr, and the linux-wpan mailing list archives.
- **Engage with peer networks**: I'll leverage the active BeagleBoard community through forums and Discord for platform-specific expertise, additionally reach out to community peers with experience upstreaming Zephyr support for Single Board Computers (SBCs).
- **Utilize the kernel mailing list** : Seek guidance from the broader Linux kernel community on upstreaming and technical challenges.
- **Maintain Documentation** : I'll maintain comprehensive documentation of all challenges and solutions for future contributors.
- **Problem Decomposition** : When dealing with complex probelms, I'll break them down problems into smaller, independently testable components, allowing me to simultaneously pursue progress on parallel work streams to ensuring project advancement while addressing specific challenges.

5.21 Benefits

This project offers significant advantages to both the developer community and end users, including:

- **Enhanced Maintainability:** Upstreaming the drivers into the Linux kernel will help ensure long-term support and will reduce the burden of maintaining out-of-tree patches, in-turn leading to a more stable and reliable system.

- **Improved Interoperability:** By enabling dynamic configuration and solidly integrating the firmware-driver, the solution supports dual-band (SubGHz and 2.4GHz) operations and smooth interoperability with protocols such as Zigbee, expanding the range of applications and compatibility with emerging IoT standards.
- **Performance and Reliability Gains:** Implementing critical features like listen-before-talk, frame retries, and dynamic power/channel management optimizes wireless communication, reducing latency and error rates while enhancing overall system performance.
- **Community Collaboration:** Well-documented, upstream-ready patch series will facilitate collaboration within the open-source community, encouraging contributions, code reviews, and continuous improvement.
- **Scalable and Future-Proof Design:** The modular architecture and dynamic configuration approach would enable easy adaptation to new hardware capabilities and evolving wireless standards, ensuring the solution remains relevant and scalable over time.

5.22 References

- https://en.wikipedia.org/wiki/IEEE_802.15.4
- https://www.kernel.org/doc/html/v5.10/networking/ieee802154.html
- https://www.kernel.org/doc/html/latest/process/submitting-patches.html
- https://openbeagle.org/beagleconnect/linux/wpanusb
- https://openbeagle.org/beagleconnect/linux/bcfserial
- https://docs.zephyrproject.org/apidoc/latest/structieee802154__radio__api.html
- https://lore.kernel.org/linux-wpan/e75f3fd8-fa0c-7708-e914-f757965920c0@datenfreihafen.org/T/
- https://github.com/statropy/wpanusb_bc
- https://lore.kernel.org/linux-wpan/f98b4515-9570-ad48-2d8f-dcc5482a21a1@datenfreihafen.org/
- https://github.com/finikorg/wpanusb
- https://www.kernel.org/doc/html/latest/process/submitting-patches.html
- https://github.com/jadonk/bcfserial
- https://github.com/jadonk/wpanusb/tree/beagleconnect
- https://sysplay.github.io/books/LinuxDrivers/book/Content/Part11.html
- https://b4.docs.kernel.org/en/latest/