Project Specification

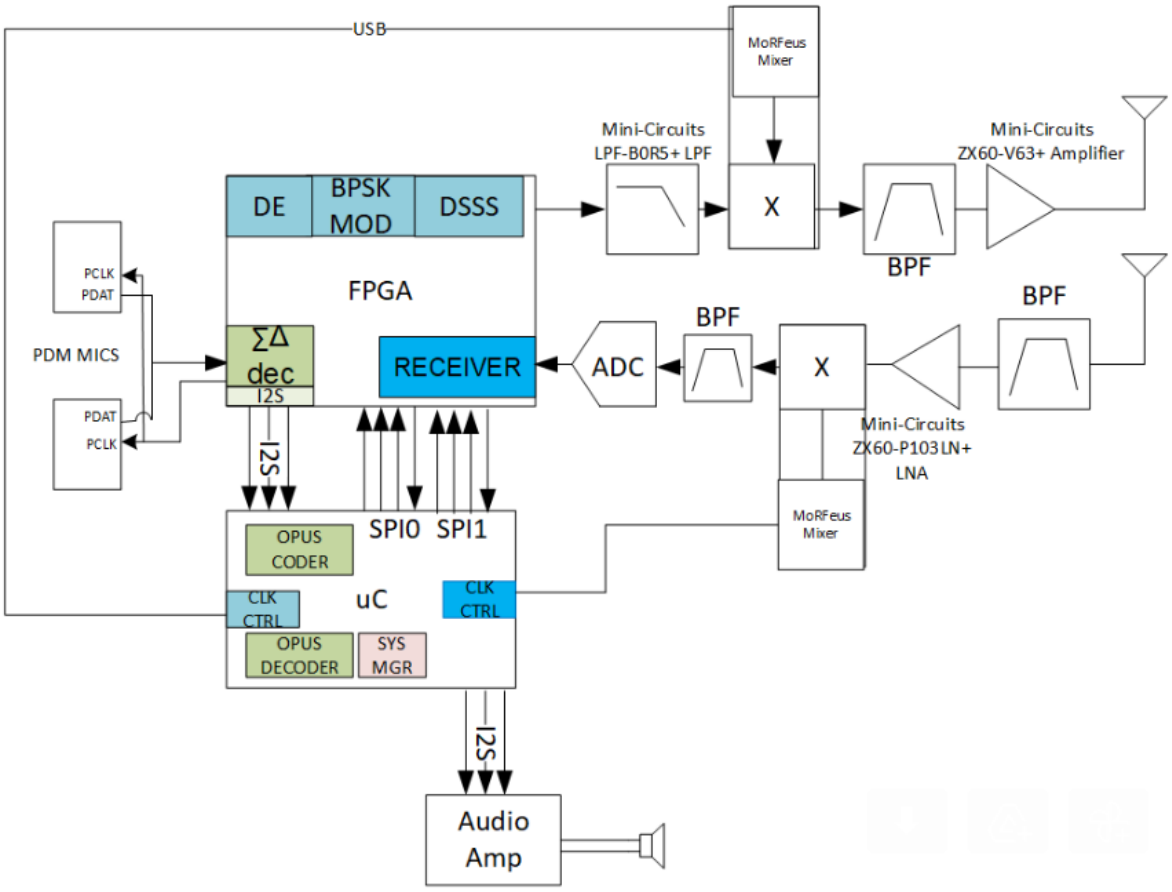
Jay Cordaro, Samuel Weismuller, Sang Ryul Pae

Project Charter

Overview

The project will be a simple “walkie-talkie” capable of communicating with other walkie-talkies in the immediate vicinity. The walkie-talkie will utilize IEEE’s 802.15.4 standard to make the system highly extendable. For instance, the system could be extended to tunnel voice calls over the internet using a commercially available 802.15.4 internet gateway. The system will be created “from scratch” as much as possible rather than using existing hardware/software solutions for 802.15.4 with the objective being to create an open-source and low-level implementation of the standard that is highly modifiable.

Project Approach



The walkie-talkie will be composed of 3 main subsystems:

Microcontroller: This system will be responsible for encoding/decoding audio data and implementing the non-physical layer portions of the 802.15.4 standard. The microcontroller will also be responsible for sending received audio to a speaker for output.

FPGA: Handles the heavy digital signal processing tasks, modulating and demodulating signals in accordance with the 802.15.4 PHY. The FPGA will also receive audio data from the microphone and format it in order to be sent to the microcontroller.

RF front end: Responsible for taking the digitally modulated data generated by the FPGA, converting it into analog and transmitting it at RF. On the other end, it is also responsible for receiving bursts at RF and converting to digital samples to provide to the FPGA.

Each block is intended to be as modular as possible, allowing development/test independently. A blocker in one module does not interrupt development with the other modules. Dependencies will only be introduced later into the development process when we begin working on integration milestones.

Audio data will be encoded using Skype's Opus codec. The Opus codec is made for lossy, low data rate use cases such as what will be necessary for our project. Encoding and decoding audio data with the codec will be the main processing load of the microcontroller.

The system will operate on a "push to talk" system that will be managed by the microcontroller. When the "talk" button is pressed, the microcontroller will read audio data buffered and streamed from the FPGA to the microcontroller. The FPGA receives 1-bit digital sigma delta modulated audio data from Pulse Density Modulation (PDM) microphone(s), decimates the PDM data to 16kHz 16-bit Pulse Code Modulation (PCM), buffers it and sends it over I2S to the microcontroller.

The 802.15.4 standard can operate at both the 900 MHz and 2.4 GHz ISM bands. For maximum range, we will be focusing on Clause 13 which specifies differentially encoded BPSK on the 900 MHz band, with support for Clause 12 on the 2.4 GHz band being a stretch goal. The only mandatory modulation scheme for the 900 MHz band is raised cosine pulse-shaped, differentially encoded BPSK. This is a relatively simple modulation scheme that should be achievable within our time constraint.

Each data bit in 802.15.4 is represented by a 15-bit Direct Sequence Spread Spectrum (DSSS) chip sequence. The 0 and 1 chip sequences are inverses of each other, so the data

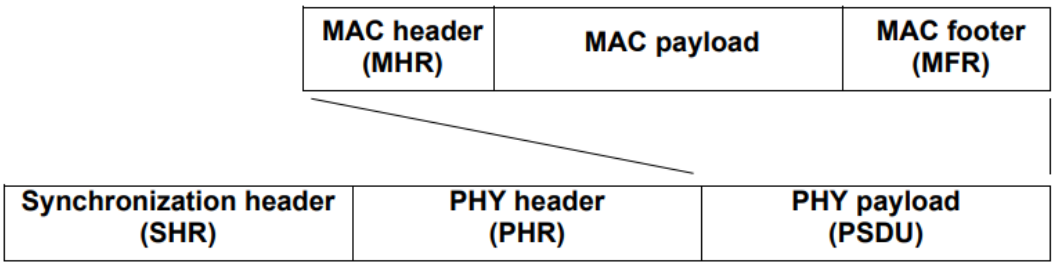
can be demodulated with a single matched filter. The 802.15.4 preamble is several repeating octets of 0 pattern, meaning signal presence can be detected using the same matched filter and a reasonable threshold. Frequency and phase errors will be corrected with a phased-locked loop and timing errors will be corrected with a delay-locked loop measuring the timing error from matched filter outputs.

All modulation and demodulation tasks, including the correction loops, will be carried out by the FPGA. All demodulated data bits following the preamble will be sent to the FPGA for further processing.

Minimum Viable Product

The minimum viable product is simply a 40kbps 802.15.4 Clause 12 transceiver in the 902-928MHz band, capable of sending and receiving data to and from a second, identical transceiver. On top of this baseline, we want to demonstrate the ability to send and receive voice data from one transceiver to the other in real time.

The minimum viable product will transmit and receive valid 802.15.4 MAC frames, described in 802.15.4 Clause 7. The frame structure is described in 802.15.4 Clause 5.7.3, Frame Structure, and is shown, below



MVP Frame Structure

MAC Payload Packets will be generated in the microcontroller. The format for the packet is described in Clause 7.1 and shown, below

Octets: 1/2	0/1	0/2	0/2/8	0/2	0/2/8	variable	variable		variable	2/4
Frame Control	Sequence Number	Destination PAN ID	Destination Address	Source PAN ID	Source Address	Auxiliary Security Header	IE		Frame Payload	FCS
		Addressing fields					Header IEs	Payload IEs		
MHR							MAC Payload			MFR

Figure 7-1—General MAC frame format

Beyond our MVP, we would like to implement 6LoWPAN, specifically RFC 4944 and RFC 6282 as an adaptation layer for encapsulating IPv6 packets into IEEE 802.15.4 MAC Service Data Units (MSDUs). If audio packets are implemented, they will be sent over UDP packets.

Clause 5.5 describes 802.15.4 network topologies. For the MVP we will implement Peer-to-Peer topology

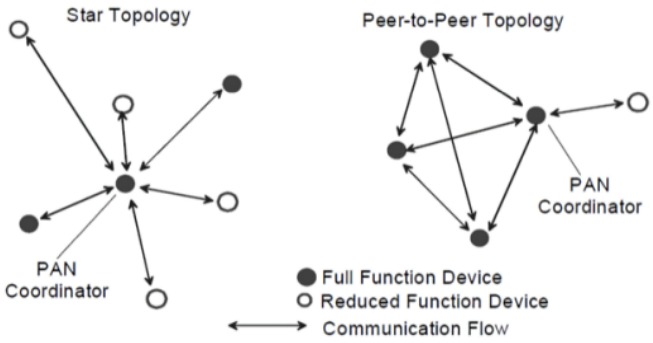


Figure 5-1—Star and peer-to-peer topology examples

The Thread protocol has many good ideas but the protocol is complicated, it only specifies operation over 2.4GHz, and it seems to be very locked down for an 'open' standard. We will not implement Thread in this project, however, we will borrow several ideas we learned about Thread in our WES269 IoT Class to build our MVP:

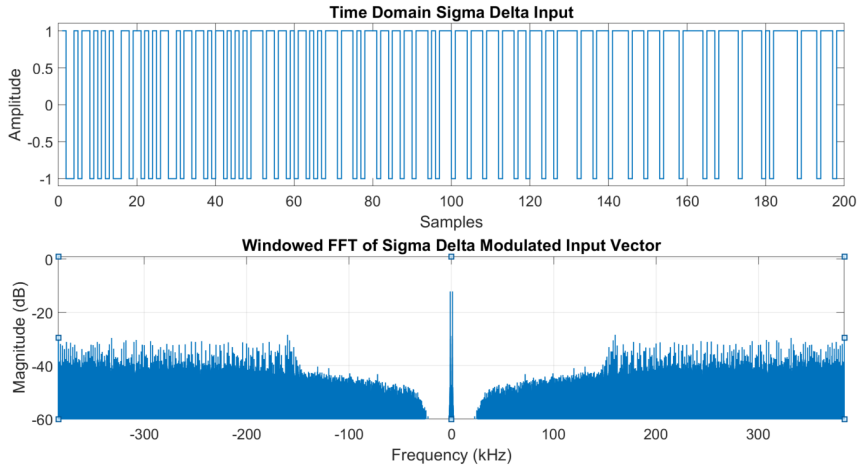
- No periodic beacons
- No guaranteed Time Slots.
    - Non-beacon enabled PAN (Personal Enabled Network) only
    - No Priority Channel Access for critical Events
    - No Slotted CSMA-CA beacon enabled PAN only
- Remove guaranteed Time Slot request
- Use THREAD frame types:
    - Beacon

- MAC Command
- Data
- Acknowledgement

We will utilize Carrier-Sense Multiple Access with Collision Avoidance (CSMA-CA) per Clause 6.2.5.1.

Security is described in 802.15.4 Clause 9 and is an optional requirement which we will not implement in this project.

Audio will be captured from Pulse Density Modulated (PDM) microphones, decimated in the FPGA and transported over I2S to the microcontroller for compression with the Opus codec and packetization for transport. A PDM microphone digital waveform and its frequency spectrum are shown in the figure below.



The FPGA will decimate this audio using an efficient DSP filtering mechanism to produce 16-bit Pulse Code Modulated (PCM) audio data.

As stretch goals, we would like to demonstrate interoperability with other 802.15.4 devices and provide a display screen and possibly a graphical user interface.

Constraint, Risk, and Feasibility

One main concern is compatibility between our hardware components. In particular, the exact method in which the FPGA should communicate with the RF front end and what sample rates should be used. A lot of implementation details depend on this decision and will be difficult to change afterwards. In addition the RF front end hardware can get expensive, so we want to be able to prototype it as cheaply as possible with parts that we have on-hand.

Another potential stumbling block is signal integrity. We'll be using this device in a real environment, which means we will need to be robust against multipath fading, additive noise,

timing and frequency errors. All these factors could have an impact on BER which can only reach up to a certain threshold before audio becomes unintelligible. However, given that our modulation scheme is BPSK, we are not likely to run into any serious issues as long as we have a sufficiently high SNR.

Another concern is the FPGA. The FPGA has to perform a lot of signal processing tasks that need to be completed in a timely manner to support our fairly high bit rate. An appropriate balance will have to be struck between timeliness and the area being consumed on the FPGA. To alleviate the risk of running into constraints late in the project's development, the FPGA that is chosen should be as large as reasonably possible.

The final concern is the Opus codec. This needs to be able to run in our embedded environment. It also needs to be able to run in a timely manner without consuming 100% of the processing time on the microcontroller. To alleviate this risk, the chosen microcontroller should be reasonably powerful.

Group Management

Each subsystem will have a group lead:
Microcontroller: Sam Weismulelr
FPGA: Jay Cordaro
RF Front End: Sang Ryul Pae

The module lead is responsible for managing deliverables and milestones for that module. Others may be pulled in as necessary. For cross module interfacing the module leads will work together to debug resolve issues. Verification of milestones will also be a joint effort so all members have an understanding of the entire system.

Milestones will have a pre-set completion date of the schedule. If it appears an item on the schedule won't be completed in time, the module lead will raise the concern as early as possible so the group can work together to resolve the issue. Most of this communication will occur through Slack and any necessary face-to-face meetings will occur on Google Meet.

While each module lead is responsible for progress on their module remaining on track, all design decisions will still be agreed upon by consensus to ensure all dissenting opinions and concerns are addressed.

Project Development

Each of the modules roughly correspond to a different skillset:

- Microcontroller: Embedded software development
- FPGA: SystemVerilog RTL development
- RF Front End: RF hardware and ensuring signal integrity

Each module lead is the individual with the most experience in each of these tasks. The other members will assist in the development of the module in order to gain experience with these other domains as well. In addition to these components, Matlab models will need to be created to simulate our transmission and reception algorithms, along with simulating the channel, to ensure our implementation is viable before it is implemented in hardware. Since this is an aspect of design that has been covered heavily by this program the task of creating these models will be split between all group members.

The microcontroller we intend to use is the Teensy 4.1. This board features an NXP ARM Cortex-M7 device and uses a plug-in on the Arduino IDE called “Teensyduino”. The microcontroller should take advantage of the open-source implementation of the Opus codec and get it to work in the embedded environment.

The FPGA to be used will be Intel MAX10M08 using the MAX10M08 Evaluation Kit. The RTL will be written in SystemVerilog. The FPGA synthesis tool will be Quartus II with simulation using EDA Playground and Siemens Questa and ModelSim.

Determining the RF front end hardware that will be used is a major task in the early parts of the development process. We have access to a number of hardware components that could potentially be combined to create an initial prototype.

A breakdown the necessary hardware components is as follows:

- FPGA boards
  - Intel MAX10 EK-10M08E144 EVBs
- Teensy 4.1 Boards
- Connection wire
- 0.1” pitch Headers
- RF Front End
  - Analog Devices AD9839SDZ DDS
  - MoRFeus SigGen/Mixers
  - 915MHz SAW Filters CBPFS-0915
  - Antenna
  - LNA
  - Power Amp
  - AD9226 ADC Module

Currently we possess the Teensy boards and one version of the Intel MAX10 FPGA board. Additionally, we have various connection wires and RF components on hand. We have met and determined the additional hardware components required. Jay works at Analog

Devices and is trying to secure 3 RF Integrated Transmitters and RF Integrated Receivers to borrow, such as the ADAQ8092, or possibly a different model if it is available and has parallel CMOS inputs for the transmitter and CMOS outputs for the ADC.

Individual software components on the microcontroller will be unit tested to ensure that they all function individually. FPGA functionality will also be individually tested with a test bench. RF front end components can be tested individually with signal generators, o-scopes, and spectrum analyzer. As soon as the necessary interfaces are ready, integration tests will be performed to confirm communication between subsystems.

Documentation will be stored on the github repo or on Google Docs. The majority of documentation will be through clean, well commented code along with detailed descriptions of the hardware components and their connections.

Project Milestones and Schedule

Items in bold are **deliverables**. The majority of milestones on the schedule are required, however there are a few tasks we can technically do without that will be labeled as "nice-to-have". All stretch goals will be listed in a separate section at the end of the schedule. These are tasks we don't expect to be able to complete but can be done if time permits. Milestones will be tracked via a Google Doc checklist located here.

Week 1 (ends 4/9/2023):
Project Management:

- [Jay] Create a Github repository for Matlab simulations and source code.
- [Team] Create skeleton for Project Specification, including a rough schedule.

Week 2 (ends 4/16/2023):
Matlab Model:

- **[Jay] Transmitter Model: Converts input bits into direct sequence-spread spectrum pulse-shaped BPSK. Simulation produces a plot of the modulated signal.**

Microntroller:

- [Sam] Demonstration of development environment functioning, can build and load a simple blink program.

FPGA:

- [Jay] Selection of which FPGA board will be used for the project.

RF Front End:

- **[Eric] Determine the required components and their specifications. Create a document outlining the available options.**

Program Management:

- **[Team] Completion of the Project Specification.**

Week 3 (ends 4/23/2023):
Matlab Simulation:

- **[Sam] Simulation of Rx chain: including signal detection, matched filtering, and error-tracking loops. Demonstrates it can demodulate signals generated by the Tx simulation.**
- **[Jay] Audio decimation chain: can take a one-bit pulse density modulated signal at the microphone sample rate and convert it to a pulse code modulated signal at a sample rate usable by the Opus codec. This simulation should be able to take PDM input samples, convert it to PCM, and play the resulting audio.**

Microcontroller:

- [Sam] Build and run Opus codec decoder/encoder libraries on Teensy board. Confirm they are able to encode/decode canned audio samples.
- [Sam] Implement SPI and I2S interfaces so they can be used for testing implementation on the FPGA.

FPGA:

- [Jay] Implement I2S and SPI interfaces and empirically validate output pins.

RF front end:

- [Eric] Complete Rx Front End component specification for required parts.
- [Eric] Complete chain calculation with all components.

Week 4 (ends 4/30/2023):
Matlab Simulation:

- **[Eric] Channel Simulation: Connects the Tx and Rx simulations with a reasonable channel simulation. Must introduce timing, phase, frequency error as well as multi-path fading. Demonstration must show that the channel introduces appropriate errors and can still be demodulated. (nice-to-have)**

FPGA:

- [Jay] Implementation of Tx signal chain from Matlab simulation. Signal is output from FPGA and can be evaluated for correctness.

Microcontroller:

- [Sam] Implementation of 802.15.4 packetization of Opus-coded data. All header elements must be included in accordance with the standard.
- **[Sam] Demonstration of speaker playing audio data using I2S interface.**

RF Front End:

- [Eric] Complete test of the whole chain and each front end component that demonstrates they will meet requirements.

Week 5 (ends 5/7/2023):
Microcontroller:

- [Sam] Implementation of I2S to FPGA interface. The Microcontroller should now be able to receive audio data from the FPGA on demand.
- [Sam] Implementation of SPI to FPGA interface for sending transmit data. The FPGA should now be able to receive packets from the Microcontroller.

FPGA:

- [Jay] Received packets from the Tx SPI are modulated and transmitted.
- [Jay] Pulse density modulated data from the microphone is converted to pulse code modulated data and sent to the Microcontroller at an appropriate sample rate for the Opus code.

RF front end:

- [Eric] Interoperability test with FPGA. Transmitted data from the FPGA should flow through the Tx front end. Received data samples on the Rx front end should be readable by the FPGA.

Week 6 (ends 5/14/2023):
Matlab:

- **[Eric] RF front end simulink model: Include a simulation of the RF front end to occur between the Tx chain, Rx chain, and channel simulations. Use the model to determine a rough estimate of the final bit error rate. (nice-to-have)**

FPGA:

- **[Jay] Rx packet detection: Demonstrate that the FPGA can detect a 802.15.4 packet using a matched filter and a SNR threshold.**
- [Jay] Differential decoder: Matched filter outputs can be converted into data bits by reversing the differential encoding. Demonstrate this allowing us to decode the 802.15.4 header.
- [Jay] Implement signal analyzer in FPGA to give us improved debugging capabilities.

Microcontroller:

- **[Sam] Demonstrate microphone data can be received from the FPGA, encoded and decoded, and then output to the speaker to show the interoperability of the components.**

Week 7 (ends 5/21/2023):
Microcontroller:

- [Sam] Listen to GPIO pin from FPGA to determine if data can be sent to it, otherwise buffer or drop pending packets.

FPGA:

- [Sam] Implement CSMA by buffering a transmit packet if a signal is detected. Use a GPIO pin to indicate to the microcontroller if the FPGA is ready to receive more Tx data.

RF Front End:

- **[Eric] RF front end demo: Demonstrate that a signal from the FPGA can be transmitted over-the-air through the RF front end and successfully demodulated by the receiving FPGA.**

Week 8 (ends 5/28/2023):
Microcontroller:

- [Sam] Receive Rx data from FPGA and output it to the speaker.

FPGA:

- [Jay] Finalize Rx chain including implementing any necessary error correction loops. Bit error rate must now be within an acceptable range.

Integration:

- [Eric] Integration of all components. Ensure Tx data from the microphone is transmitted by the Rx front end and ensure received signals are demodulated and audio is emitted by the speaker.

Week 9 (6/4/2023):
Program Management:

- **[Eric] End-to-end demonstration showing audio transmitted from one walkie-talkie being received by another walkie-talkie.**
- **[Jay] Creation of slides for final presentation.**
- **[Sam] Creation of video for final presentation.**

Stretch goals:

- Demonstrate interoperability of our walkie-talkies with another 802.15.4 device.
- Tunnel traffic through the internet using an 802.15.4 gateway.
- Implement the optional OQPSK waveform for 802.15.4.