Google Summer of Code 2017: DAB/DAB+ transceiver app

Moritz Luca Schmid

March 17, 2017

1 Introduction

The project "DAB/DAB+ transceiver app" aims to produce a DAB/DAB+ tranceiver app that is capable of the transmission and reception of audio and data according to the DAB/DAB+ standard. It builds on the existing GNU Radio Out-Of-Tree module `gr-dab`. The goals for this project are the completion of the `gr-dab` module and the creation of a full-fleshed application including a graphical user front end that is capable of transmitting and receiving DAB/DAB+.

2 Digital Audio Broadcasting

Digital Audio Broadcasting (DAB) is a standard for radio broadcasting of audio and data services. With the digital processing of data comes a whole new spectrum of opportunities in providing a large amount of service information to a channel for example emergency warnings, traffic information and even moving pictures like weather maps. Furthermore a DAB service which is embedded in a DAB ensemble, can supply a diversity of audio and data streams each sent in a separate sub channel that can be optionally shared among services. This flexibility of the system and its service structure demands complex data multiplexing. Therefore the DAB standard provides the Fast Information Channel (FIC) which mainly exists for carrying the multiplex control data safely and quickly.
The digital transmission of the audio data enables error protection and therefore better audio quality with less noise in comparison to FM. To reach a more efficient use of bandwidth the DAB system encodes audio using a modified MPEG Audio Layer II with an additional error protection. Typical data rates are 128 kbit/s or 160 kbit/s which try to achieve the subjective quality of an audio CD. In order to maintain this quality in even more noisy channels, DAB was further developed to DAB+. It contains the audio codec HE AAC v2 which uses spectral band replication and provides a stronger error correction with Reed-Solomon codes that achieves this subjective CD quality with a data rate of 80 kbit/s.
The transmission signal is modulated in D-QPSK and OFDM. There are four different transmission modes in DAB which are each optimized for specific network configurations different operating frequencies. Besides the common terrestrial transmission, transmission mode III for example is optimized for low frequency cable transmission and also there are specially optimized modes for Singlefrequency networks(SFN) in transmission mode I and IV.

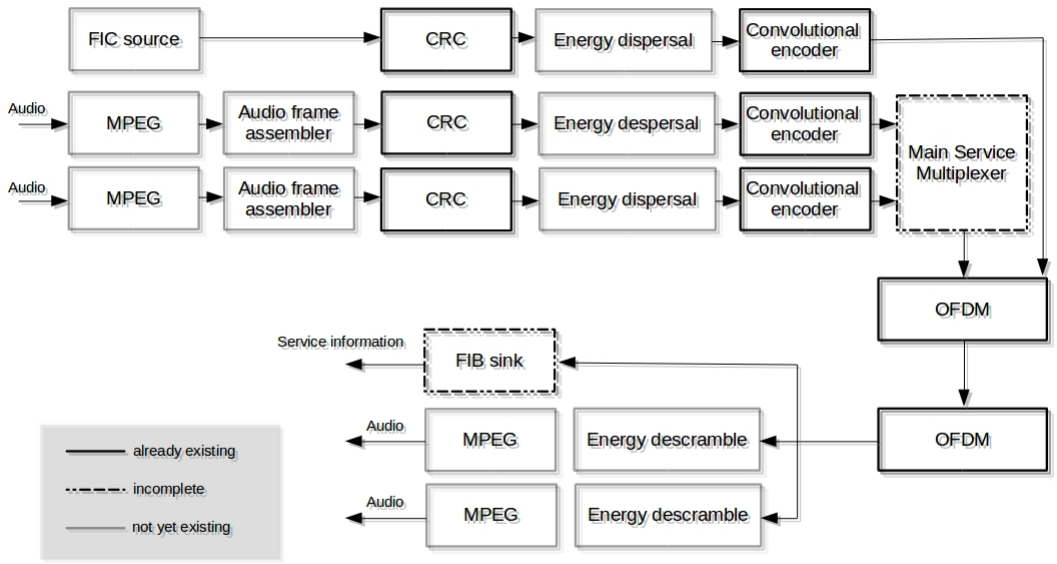

Figure 1: Flowgraph of a DAB transceiver

3 Deliverable features

3.1 Out-Of-Tree module gr-dab

A dab module (`gr-dab`) already exists, impemented by Andreas Mueller which can be found on GitHub [1]. It is compatible with `gr_modtool` and can be modified and extended.
The `gr-dab` module contains the implementation of the physical layer including OFDM and error correction. It does not provide any interpretation of the received audio frames due to a missing audio decoding. It can extract basic information for example service labels, but a lot of features are not implemented yet. The handling of the specific messages and creation of the datastream which is necessary for a transmittion signal is also not existing.
To provide a functional transmission and extended reception of DAB signals including audio coding, I am planning to extend and modify the `gr-dab` module by improving existing blocks and adding new ones.
A comprehensive overview of the blocks, existing and planned to add, is shown in figure 1. In detail, I am planning to implement the following features (written in C++):

1. **FIC source**: This block creates the Fast Information Groups (FIGs) of the incoming parameters (transmission mode, number of subchannels and strings of labels of the ensemble) containing multiplexing information (MCI) and service information (SI). In addition

it handles the assembling of the Fast Information Blocks (FIBs) made out of the FIGs It prefers those FIGs containing essential MCI and subsequently filling the FIBs with optional SI. Finally tags are added to the stream to mark the beginning of each FIB for the following cyclic redundancy check (CRC) which completes the FIB. The implementation of the CRC is not necessary because it is included in GNU Radio. In `gr-dab` exists as well a convolutional coding (`crc16`) which seems to have the identic functionality like the one in `gr-fec`.

2. **FIB sink**: This block already exists in the module `gr-dab` and is able to read the different FIBs out of FIC. It includes CRC and energy descrambling. Some interpretation of data is already done, like the recognition of labels. Other features of the service information are not supported yet. The missing parts for interpretation of the FIB data is added, including service information and parts of the fast information data channel.

3. **Energy dispersal** and **energy descramble**: An energy dispersal scrambler ensures the appropriate energy dispersal of the transmitted audio frames and the transmitted FIBs. It can be derived from the existing energy dispersal of the DVB-T module, changing the pseudo-random binary sequence and the polynomial. The class accepts the transmission mode (and FIB/MSC) as an input argument according to change the length of the vector that is being scrambled which depends on the number of FIGs which are included in one transmission frame. The descrambling in the receiver follows the same process.

4. The **convolutional coding** which completes the coding process is already included in GNU Radio (`gr-fec`) and can be used.

5. Audio services are coded in **MPEG Audio Layer II** in DAB, and **HE-AAC v2** in DAB+. For this issue MPEG encoding and decoding blocks are added using C/C++ libraries. To preserve the flexibility of easily changing the audio coding, for example to switch between DAB and DAB+, and possibly try new codecs in the future, the audio frame assembler is implemented separately.

6. **Audio frame assembler**: The block accepts the encoded audio stream and adds header information and zero padding and tags the stream for the following CRC.

7. **main service multiplexer**: It combines all of the sub channel streams to a common interleaved frame (CIF). This class is fractionally implemented in `gr-dab` (`dab_concatenate_signals`) with a comment, that samples of the input vectors get lost in some cases. I am planning to solve the problem of long input vectors using tagged streams instead.

8. The OFDM is in mainly parts already implemented in `gr-dab` and is going to be used.

3.2 Graphical user interface

To provide comfortable usage of the transceiver module, a user front end is implemented with the PyQt5 framework. It displays the received service information and lets the user select the transmission mode and the channel out of the ensemble. In addition it offers certain information for developers with interest in the physical transmission, for example a waterfall diagramm. A draft of how the GUI could look like, is given in figure 2.

3.3 Documentation

A detailled documentation is a essential requirement especially for a open-source project that is likely to be continued and extended after GSoC. Therefore a doxygen documentation is written for the `gr-dab` module. In addition I will comment my source code in order make my thoughts easily comprehensible for everyone interested.

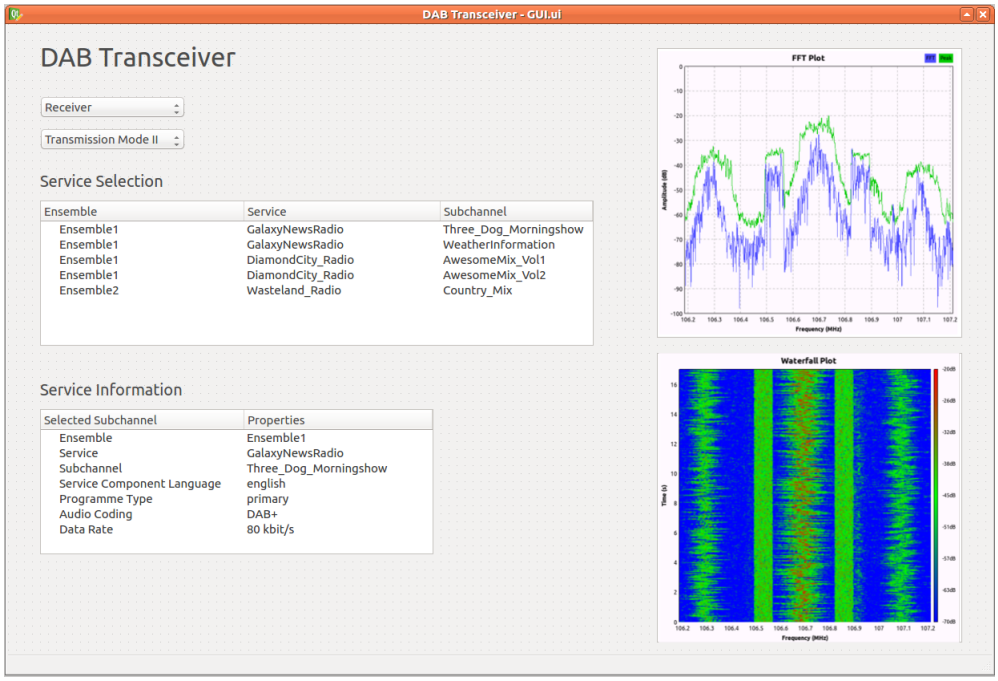

Figure 2: GUI

4 Timeline

I am a third-year Bachlor student at Karlsruhe Institute of Technolgy (KIT), Germany and I am planning to write my bachelor thesis in fall 2017. I have the possibility to write my thesis at Communications Engineering Lab (CEL) about DAB/DAB+ which would include the implementations coded in GSoC. Therefore I could work full-time on the GSoC project DAB/DAB+ and write my Bachelor thesis afterwards. At CEL I have got access to SDRs for transmission and reception of DAB signals. Furthermore the GSoC mentor Felix Wunsch works also at CEL which would ensure an intense coummunication while GSoC. The CEL fully supports the open-source idea. Therefore the whole project will be open-source available on GitHub, included the GPLv3 licensed code of GSoC and the thesis written after GSoC. I am also planning to publish my weekly progress in a internet blog in order to keep my work transparent.

According to the timeline of GSoC 2017 there are 13 weeks of coding period including 3 evaluation periods. I scheduled the deliverables into 1- or 2-week periods, planning to work full-time from Monday to Friday and using the weekends as additional time buffers.

Table 1: Timeline GSoC 2017

May 4 - May 30	**Community Bonding** - coordination of the project with mentor.
May 30 - June 11	Coding, 2 weeks - implement transmitter blocks in C++ and QA tests in Python.
June 12 - June 18	Coding, 1 week - add MPEG Audio Layer II and HE-AAC v2 encoder and decoder block and optimize transmitter.
Jun 19 - Jun 25	Coding, 1 week - implement receiver blocks in C++ and QA tests in Python.
June 26 - June 28	**Phase 1 evaluation**, 3 days - prepare results for evaluation including transmitter and QA tests.
29 Jun - Jul 16	Coding, 2.5 weeks - implement receiver blocks in C++ and QA tests in Python .
Jul 17 - Jul 23	Coding, 1 week - wrap up the *gr-dab* module and create QA tests for transceiver in Python and example for GRC.
Jul 24 - Jul 26	**Phase 2 evaluation**, 3 days - prepare results for evaluation.
July 27 - Aug 13	Coding, 2.5 weeks - GUI implementation in Python.
Aug 14 - Aug 20	Coding, week - final bug fixes and optimization.
Aug 21 - Aug 29	**Final evaluation** - 1 week, submission of the final code and evaluation .

I have read the rules of conduct on the GNU Radio webpage and acknowledge the tree strikes rule.

5 About myself/Coding history

I firstly came in touch with programming when I participated in the inventors competition "MikroMakro – kleine Köpfe große Ideen" of the foundation "Landesstiftung Baden-Württemberg". Together we were a team of six classmates. We got an amount of 5000 € to execute our project idea - "Mamas Voice", a little humanlike interactive speaking device that reminds students of deadlines and to do's – like a mother does, including the interaction with the human. We implemented the project on the electronics platform Arduino – using the Arduino microcontroller and software. Here I could gain first experience in programming.
In 2013 I collected geodata from roads and trails for hiking and biking with a gps device and the geographic information system GeoMedia working for the department of environment of Leinfelden-Echterdingen. In order to grant the outdoor community public access of the data, I managed to read the geodata together with metadata like road conditions out of geoMedia. I additionally implemented a front end that displays the tracks on a georeferenced aerial photograph and offers the user the possibility to create a route by selecting certain track sections and automatically generating a gpx file which can be used to navigate through the route with a gps device. I used the open-source environment Processing (based on java) in order to easily deal with graphical issues.
Since 2014 I have been studying electronics and information technology at KIT in Karlsruhe. I have improved my coding skills attending lectures in information technology (based on C/C++) and successfully completed a practical training. The objective was to implement the hardware abstraction layer for the control software of a segway called TivSeg. This included the control of a microcontroller, excluding the feedback control algorithm for the control of the segway. The code was written in C/C++ containing hardware-near programming.
After visiting several lectures to radio communication I got a job at CEL (Communication Engineering Lab) in 2016 in order to get more in touch with practical telecommunication engineering. I got familiar with GNU Radio and improved my coding skills in C++ and Python. I suscribed to the GNU Radio mailing list to get in touch with the community. So far I had no need to ask questions due to some really helpful colleagues at CEL but I have the intention to enrich the GNU Radio community in the future. to active I already dealt with the DAB standard in my job, doing researches and understanding the standard.
You can find the code of the mentioned projects on my GitHub account [2].

6 Conclusion

A DAB module in GNU Radio would not just be a great example of how a worldwide operating broadcasting system can be fully implemented in a software radio environment, but also a very useful basis for future projects with GNU Radio implementing new features or doing research. The developed DAB/DAB+ transceiver app allows a comfortable use of the transceiver for developers as well as for beginners.
Through my projects I came to realise the importance of open-source and how everyone can achieve more by sharing their ideas and thoughts. Therefore I would like to take this opportunity and participate in a GNU Radio project to contribute to the open-source idea. Finally I want to emphasize: Cyberspectrum is the best spectrum.

7 References

References

[1] https://github.com/andrmuel/gr-dab Visited March 13, 2017.

[2] https://github.com/MoritzLucaSchmid

[3] Digital Audio Broadcasting; DAB+ audio coding (ETSI TS 102 563 V2.1.1 (2017-01))

[4] Radio Broadcasting Systems; Digital Audio Broadcasting (DAB) ro mobile, portable and fixed receivers (ETSI EN 300 401 V2.1.1 (2017-01))

[5] Digital Audio Broadcasting (DAB); DAB audio coding (MPEG Layer II) (ETSI TS 103 466 V1.1.1 (2016-10))

[6] GNU Radio Documentation http://gnuradio.org/doc/doxygen/modules.html Visited March 10

[7] W. Hoeg, T. Lauterbach, "Digital Audio Broadcasting; Principles and Applications of Digital Radio", Wiley 2003

[8] Waterfall and FFT plot http://www.stargazing.net/david/GNUradio/RTLFMstations.html