ODR-mmbTools
Open-Source Software-Defined DAB$^+$ Tools

Project Documentation

Matthias P. Braendli, for Opendigitalradio
http://opendigitalradio.org
2014

Figure 1: Okay, maybe put a real logo here. . .

Contents

Acronyms

1PPS	One pulse per second
CIF	Common Interleaved Frame
CRC	Communications Research Centre Canada
DAB	Digital Audio Broadcasting
DMB	Digital Multimedia Broadcasting
ETI	Ensemble Transport Interface
ETSI	European Telecommunications Standards Institute
FIC	Fast Information Channel
HE-AAC	High Efficiency Advanced Audio Codec
mmbTools	Mobile Multimedia Broadcasting Tools
MNSC	Multiplex Network Signalling Channel
NTP	Network Time Protocol
OCXO	Oven-Controlled Crystal Oscillator
OFDM	Orthogonal Frequency-Division Multiplexing
PRBS	Pseudo-Random Bit Sequence
SFN	Single-Frequency Network
TCXO	Temperature-Compensated Crystal Oscillator
TIST	Timestamp field in the ETI frame
TM	Transmission Mode
UHD	USRP Hardware Driver
USRP	Universal Software-Radio Peripheral

1 Introduction

This is the official documentation for the ODR-mmbTools. These tools can be used to experiment with DAB modulation, learn the techniques behind it and setup a DAB or DAB$^+$ transmitter.

This documentation assumes that you are already familiar with base concepts of the DAB system. To get started with the ODR-mmbTools, understanding how the DAB transmission chain is structured is a prerequisite. The "DAB Bible" by Hoeg and Lauterbach [2] and the "Guide to DAB standards" from the ETSI [1] can be used as a starting point.

In this document, the terms "DAB" and "DAB$^+$" are used somewhat interchangeably, since many parts of the transmission chain are identical between the two variants. In most cases, "DAB" will be used, and "DAB$^+$" when talking about specific details about the newer version of the standard.

2 Purpose

The different programs that are part of the ODR-mmbTools each have their own documentation regarding command-line options and configuration settings, and the opendigitalradio.org wiki[1] contains many explanations and pointers, but there is no single source of documentation available for the whole tool-set.

This document aims to solve this, by first outlining general concepts, presenting different usage scenarios and detailing a complete transmission setup.

With this document in hand, you should be able to understand all elements composing a ODR-mmbTools transmission chain, and how to set one up.

3 Presentation of the Tools

3.1 Origins

Before we begin with technical details, first a word about the history of the mmbTools. In 2002, Communications Research Centre Canada[2] started developing a DAB multiplexer. This effort evolved through the years, and was published in September 2009 as CRC-DabMux under the GPL open-source licence.

CRC also developed a DAB modulator, called CRC-DABMOD, which could create baseband I/Q samples from an ETI file. This I/Q data could then be set to a hardware device using another tool. For the Ettus USRPs, a "wave player" script was necessary to interface to GNURadio. Only DAB Transmission Mode 2 was supported. CRC-DABMOD was also released under the GPL in early 2010.

As encoders, toolame could be used for DAB, and CRC developed a closed-source CRC-DABPLUS DAB$^+$ encoder.

These three CRC- tools, and some additional services available on the now unreachable website[3] `http://mmbtools.crc.ca` were part of the CRC-mmbTools. These tools made it possible to set up the first DAB transmission experiments.

[1]`http://opendigitalradio.org`

[2]`http://crc.ca`

[3]There are some snapshots of the website available on `http://archive.org`.

In 2012, these tools received experimental support for single-frequency networks, a functionality that has been developed by Matthias P. Brändli during his Master's thesis[4]. Because SFNs are mainly used in TM 1, CRC subsequently released a patch to CRC-DABMOD that enabled all four transmission modes.

At that point, involvement from CRC started to decline. The SFN patch was finally never included in the CRC-mmbTools, and as time passed by, the de-facto fork on `http://mpb.li` was receiving more and more features. Having two different programs with the same name made things complicated, and the tools were officially forked with the approval of CRC in February 2014, and given the new name ODR-mmbTools. They are now developed by the Opendigitalradio association.

In April 2014, the official CRC-mmbTools website went offline, and it has become very difficult, if not impossible to acquire licences for the CRC-DABPLUS encoder. Luckily there is an open-source replacement available, which was part of Google's Android sources. This encoder has been extended with the necessary DAB$^+$-specific requirements (960-transform, error correction, framing, etc.), and now exists under the name FDK-AAC-DABplus.

3.2 Included Tools

The ODR-mmbTools are composed of several software projects: ODR-DabMux, ODR-DabMod, Toolame-DAB, FDK-AAC-DABplus, and other scripts, bits and pieces that are useful for the setup of a transmission chain.

3.2.1 ODR-DabMux

ODR-DabMux implements a DAB multiplexer that combines all audio and data inputs into an ETI output. It can be used off-line (i.e. not real-time) to generate ETI data for later processing, or in a real-time streaming scenario (e.g. in a transmitter).

It can read input audio or data from files (".mp2" for DAB, ".dabp" for DAB$^+$), FIFOs (also called "named pipes") or a network connection. The network connection can use UDP or ZeroMQ. The CURVE authentication mechanism from ZeroMQ can also be used to authenticate the encoder, in order to avoid that a third-party can disrupt or hijack a programme.

The ensemble configuration can be specified on the command line using the options described in the manpage, or using a configuration file. The command line options are kept to be compatible with CRC-DABMUX, but using the configuration file is preferred, because it supports more options.

3.2.2 ODR-DabMod

ODR-DabMod is a software-defined DAB modulator that receives or reads ETI, and generates modulated I/Q data usable for transmission.

This I/Q data which is encoded as complex floats (32bits per complex sample) can be written to a file or pipe, or sent to a USRP device using the integrated UHD output. Other SDR platforms can be used if they are able to accept the

[4]The corresponding report is available at `http://mpb.li/report.pdf`

I/Q data. The output of the modulator can also be used in GNURadio if format conversion or graphical analysis (spectrum) is to be done.

3.2.3 Toolame-DAB

TooLAME is a MPEG-1 Layer II audio encoder that is used to encode audio for the DAB standard. The original project has been unmaintained since 2003, but the twolame fork that pursues the development removed the DAB framing. Because of this, twolame is not suitable for DAB.

The Toolame-DAB fork includes the ZeroMQ output and PAD insertion support, but the audio coder is the same as the one in tooLAME.

3.2.4 FDK-AAC-DABplus

The FDK-AAC-DABplus encoder can be used to encode for DAB$^+$. The encoder itself comes from the Android sources, and was written by Fraunhofer.

The necessary framing and error-correction that DAB$^+$ mandates, the PAD insertion, the ZeroMQ output and the ALSA input were then added by different parties.

4 Interfacing the Tools

4.1 Files

The first versions of these tools used files and pipes to exchange data. For offline generation of a multiplex or a modulated I/Q, it is possible to generate all files separately, one after the other.

Here is an example to generate a two-minute ETI file for a multiplex containing two programmes:

- one DAB programme at 128kbps, encoded with Toolame-DAB
- one DAB$^+$ programme at 88kbps, encoded with FDK-AAC-DABplus

We assume that the audio data for the two programmes is located in uncompressed 48kHz WAV in the files `prog1.wav` and `prog2.wav`. The first step is to encode the audio. The DAB programme is encoded to `prog1.mp2` using:

```
toolame -b 128 prog1.wav prog1.mp2
```

The DAB+ programme is encoded to `prog2.dabp`. The extension `.dabp` is arbitrary, but since the framing is not the same as for other AAC encoded audio, it makes sense to use a special extension. The command is:

```
dabplus-enc -i prog2.wav -b 88 -a o prog2.dabp
```

These resulting files can then be used with ODR-DabMux to create an ETI file. ODR-DabMux supports many options, which makes it much more practical to set the configuration using a file than using very long command lines. Here is a short file that can be used for the example, which will be saved as `2programmes.mux`:

```
general {
    dabmode 1
    nbframes 5000
}
remotecontrol { telnetport 0 }
ensemble {
    id 0x4fff
    ecc 0xec ; Extended Country Code

    local-time-offset auto
    international-table 1
    label "mmbtools"
    shortlabel "mmbtools"
}
services {
    srv-p1 { label "Prog1" }
    srv-p2 { label "Prog2" }
}
subchannels {
    sub-p1 {
        ; MPEG
        type audio
        inputfile "prog1.mp2"
        bitrate 128
        id 10
        protection 5
    }
    sub-p2 {
        type dabplus
        inputfile "prog2.dabp"
        bitrate 88
        id 1
        protection 1
    }
}
components {
    comp-p1 {
        label Prog1
        service srv-p1
        subchannel sub-p1
    }
    comp-p2 {
        label Prog2
        service srv-p2
        subchannel sub-p2
    }
}
outputs { output1 "file://myfirst.eti?type=raw" }
```

This file defines two components, that each link one service and one subchannel. The IDs and different protection settings are also defined. The bitrate defined in each subchannel must correspond to the bitrate set at the encoder.

The duration of the ETI file is limited by the `nbframes` 5000 setting. Each frame corresponds to 24 ms, and therefore $120/0.024 = 5000$ frames are needed for 120 seconds.

The output is written to the file `myfirst.eti` in the ETI(NI) format. Please see Appendix A for more options.

To run the multiplexer with this configuration, run:

```
odr-dabmux 2programmes.mux
```

This will generate the file `myfirst.eti`, which will be $5000 * 6144 \approx 30$MB in size.

Congratulations! You have just created your first DAB multiplex! With the configuration file, adding more programmes is easy. More information is available in the `doc/example.mux`

4.2 Over the Network

In a real-time scenario, where the audio sources produce data continuously and the tools have to run at the native rate, it is not possible to use files anymore to interconnect the tools. For this usage, a network interconnection is available between the tools.

This network connection is based on ZeroMQ, a library that permits the creation of a socket connection with automatic connection management (connection, disconnection, error handling). ZeroMQ uses a TCP/IP connection, and can therefore be used over any kind of IP networks.

This connection makes it possible to put the different tools on different computers, but it is not necessary. It is also possible, and even encouraged to use this interconnection locally on the same machine.

4.2.1 Between Encoder and Multiplexer

Between FDK-AAC-DABplus and ODR-DabMux, the ZeroMQ connection transmits AAC superframes, with additional metadata that contains the audio level indication for monitoring purposes. The multiplexer cannot easily derive the audio level from the AAC bitstream without decoding it, so it makes more sense to calculate this in the encoder.

The Toolame-DAB encoder also can send MPEG frames over ZeroMQ, but is not yet able to calculate and transmit audio level metadata yet.

On the multiplexer, the subchannel must be configured for ZeroMQ as follows:

```
sub-fb {
    type dabplus
    bitrate 80
    id 24
    protection 3

    inputfile "tcp://*:9001"
    zmq-buffer 40
    zmq-prebuffering 20
}
```

The ZeroMQ input supports several options in addition to the ones of a subchannel that uses a file input. The options are:

- `inputfile`: This defines the interface and port on which to listen for incoming data. It must be of the form `tcp://*:<port>`. Support for the `pgm://` protocol is experimental, please see the `zmq_bind` manpage for more information about the protocols.

- `zmq-buffer`: The ZeroMQ input handles an internal buffer for incoming data. The maximum buffer size is given by this option, the units are AAC frames (24 ms). Therefore, with a value of 40, you will have a buffer of $40 * 24 = 960$ ms. The multiplexer will never buffer more than this value, and will discard data one AAC superframe (5 frames = 100 ms) when the buffer is full.

- `zmq-prebuffering`: When the buffer is empty, the multiplexer waits until this amount of AAC frames are available in the buffer before it starts to consume data.

The goal of having a buffer in the input of the multiplexer is to be able to absorb network latency jitter: Because IP does not guarantee anything about the latency, some packets will reach the encoder faster than others. The buffer can then be used to avoid disruptions in these cases, and its size should be adapted to the network connection. This has to be done in an empirical way, and is a trade-off between absolute delay and robustness.

If the encoder is running remotely on a machine, encoding from a sound card, it will encode at the rate defined by the sound card clock. This clock will, if no special precautions are taken, be slightly off frequency. The multiplexer however runs on a machine where the system time is synchronised over NTP, and will not show any drift or offset. Two situations can occur:

Either the sound card clock is a bit slow, in which case the ZeroMQ buffer in the multiplexer will fill up to the amount given by `zmq-prebuffering`, and then start streaming data. Because the multiplexer will be a bit faster than the encoder, the amount of buffered data will slowly decrease, until the buffer is empty. Then the multiplexer will enter prebuffering, and wait again until the buffer is full enough. This will create an audible interruption, whose length corresponds to the prebuffering.

Or the sound card clock is a bit slow, and the buffer will be filled up faster than data is consumed by the multiplexer. At some point, the buffer will hit the maximum size, and one superframe will be discarded. This also creates an audible glitch.

Consumer grade sound cards have clocks of varying quality. While these glitches would only occur sporadically for some, bad sound cards can provoke such behaviour in intervals that are not acceptable, e.g. more than once per hour.

Both situations are suboptimal, because they lead to audio glitches, and also degrade the ability to compensate for network latency changes. It is preferable to use the drift compensation feature available in FDK-AAC-DABplus, which insures that the encoder outputs the AAC bitstream at the nominal rate, aligned to the NTP-synchronised system time, and not to the sound card clock. The sound card clock error is compensated for inside the encoder.

Complete examples of such a setup are given in the Scenarios.

4.2.2 Authentication Support

In order to be able to use the Internet as contribution network, some form of protection has to be put in place to make sure the audio data cannot be altered by third parties. Usually, some form of VPN is set up for this case.

Alternatively, the encryption mechanism ZeroMQ offers can also be used. To do this, it is necessary to set up keys and to distribute them to the encoder and the multiplexer.

```
    encryption 1
    secret-key "keys/mux.sec"
    public-key "keys/mux.pub"
    encoder-key "keys/encoder1.pub"
```

Add configuration example

4.2.3 Between Multiplexer and Modulator

The ZeroMQ connection can also be used to connect ODR-DabMux to one or more instances of ODR-DabMod. One ZeroMQ frame contains four ETI frames, which guarantees that the modulator always assembles the transmission frame in a correct way, even in Transmission Mode I, where four ETI frames are used together.

4.3 Pipes

Pipes are an older real-time method to connect several encoders to one multiplexer on the same machine. It uses the same configuration as the file input but instead of using files, FIFOs, also called “named pipes” are created first using `mkfifo`.

This setup is deprecated in favour of the ZeroMQ interface.

5 Usage Scenarios

5.1 Experimentation

5.1.1 Creation of Non-Realtime Multiplex

The creation of a ETI file containing two programmes, one DAB and one DAB$^+$ is covered in section 4.1.

5.1.2 Modulation of ETI for Offline Processing

The ETI file generated before can then be used with ODR-DabMod to generate a file containing I/Q samples. Here, we must chose between using the command line or the configuration file. For a very simple example, using the command line makes sense, but for more advanced features it is preferable to use a configuration file. For illustration, we will present both.

To modulate the file `myfirst.eti` into `myfirst.iq`, with the default options, the command is simply

```
odr-dabmod myfirst.eti -f myfirst.iq
```

This will create a file containing 16-bit interleaved I/Q at 2048000 samples per second. The transmission mode is defined by the ETI file.

The equivalent configuration file would be

```
[input]
transport=file
source=myfirst.eti

[output]
output=file

[fileoutput]
filename=myfirst.iq
```

This is a very minimal file that defines only the necessary settings equivalent to the above command line options. The configuration file however supports more options that the command line, and becomes easier to manager once the set becomes more complex. It is best to use the example configuration availble in the `doc/` folder.

5.2 Interfacing Hardware Devices

5.2.1 Ettus USRP

ODR-DabMod integrates support for the UHD library that can interface with all USRP devices from Ettus. The following configuration file `mod.ini` illustrates how to send the `myfirst.eti` over a USRP B200 on channel 13C:

```
[remotecontrol]
telnet=1
telnetport=2121
```

```

[input]
transport=file
source=myfirst.eti
loop=1

[modulator]
gainmode=2
digital_gain=0.8

[firfilter]
enabled=1
filtertapsfile=simple_taps.txt

[output]
output=uhd

[uhdoutput]
master_clock_rate=32768000
type=b200
txgain=40
channel=13C
```

This example also shows more options that the example for the file output:

- `remotecontrol telnet=1` enables the Telnet server that can be used to set parameters while the modulator is running.
- `loop=1` rewinds the input file when the end is reached. The same ETI file will be transmitted over and over.
- `gainmode=2` sets the GainMode to VAR, which reduces overshoots in the output.
- `digital_gain=0.8` reduces the output sample deviation, to reduce compression in the USRP.
- `firfilter enabled=1` enables an additional FIR filter to improve the spectrum mask.
- `master_clock_rate=32768000` sets the USRP internal clock to a multiple of 2048000, which is required if we want to use the native DAB sample rate.
- `txgain=40` Sets the analog transmit gain of the USRP to 40dB, which is specific to the B200.

Some of these options are not necessary for the system to work, but they improve the performance.

Remarks concerning the USRP B200 The USRP B200 is the device we are using most. It's performance is proven in a production environment, it supports the transmit synchronisation necessary for SFN and is robust enough for 24/7 operation.

However, care has to be taken about the host system, especially about the USB controller. Using USB 2.0 is not a problem for a DAB transmission, both USB 2.0 and USB 3.0 host controllers can therefore be used. Since USB 2.0 has been around for longer and is more mature, it is sometimes preferable because it causes less USB errors. This heavily depends on the exact model of the USB controller inside the host PC, and has to be tested for each system.

The txgain on the B200 varies between 0dB and about 90dB. Experience shows that compression effects begin to appear at values around 85dB. This might be different from device to device and needs to measured.

Similarly, the digital gain has to optimised for a given setting. It is important that there is no digital clipping in the chain, because that leads to problematic spurious spectrum components, that can disturb or even damage a power amplifier.

There are some performance measurements available on the Opendigitalradio wiki.[5]

Remarks concerning other USRP models We have used the USRP1, the USRP2 and the USRP B100 with the tools. The WBX is the most appropriate daughterboard for these models.

The txgain setting has another range, it is best to start at 0dB, and increase it in steps of 3dB or smaller while measuring the output signal, until the correct power is reached.

5.2.2 Other Hardware

The HackRF can also be used as a transmit platform. It's interfacing is not integrated with ODR-DabMod, and it only supports 8-bit samples. The configuration is a bit different. First, we must output signed 8-bit interleaved I/Q samples instead of 32-bit interleaved floats, and second, we cannot use UHD, but must go through a pipe to HackRF. The output settings inside the configuration are as shown:

```
[output]
output=file
[fileoutput]
format=s8
filename=/tmp/ofdm.fifo
```

The output fifo has to be created beforehand, and the `hackrf_transfer` utility is then used to transmit the signal to the device. The options needed for the transmission utility are not complete yet. The rough idea would be:

```
mkfifo /tmp/ofdm.fifo
odr-dabmod mod.ini &
hackrf_transfer -f <frequency> -t /tmp/odr.fifo
```

[5] http://wiki.opendigitalradio.org/index.php/USRP_B200_Measurements

5.3 Audio Sources

Preparing a DAB multiplex with different programmes requires that we are able to read and encode several audio sources. This audio data can reach the multiplexer in different ways. We will go over different possibilities in this part.

5.3.1 Using Existing Web-Streams

One common scenario is to transmit radio stations that already are available as web-radio streams. For simplicity, it makes sense to get these web streams, which are most often encoded in mp3 and available through HTTP, decode them, and use them as audio source for the DAB or DAB$^+$ encoder.

The advantage of this approach is that the radio itself does not need to setup a new infrastructure if the stream is of good quality. The main disadvantage is that the audio is encoded twice, and this coding cascading degrades the audio quality.

Often, web-streams are encoded in mp3 at 44100Hz sample-rate, whereas DAB is most often 48000Hz or sometimes 32000Hz. A sample-rate conversion is necessary in the stream decoder.

There are many different stream decoders, and gstreamer, mpg123 and mplayer have been tested. We have achieved good results with mplayer, and the dab-scripts repository[6] contain the script `encode-jack.sh` that uses mplayer, and illustrates how it is possible to encode a web-stream to DAB$^+$. JACK[7] is used to interconnect the stream decoder to the DAB$^+$ encoder.

This script is designed for production use, and also contains automatic restart logic in case of a failure.

5.3.2 Encoders at Programme Originator Studios

5.3.3 Multi-capture Audio Card

5.4 Single-Frequency Networks

[6] `http://github.com/Opendigitalradio/dab-scripts`

[7] The JACK Audio Connection Kit is a virtual audio patch, `http://www.jack-audio.org`

A ODR-DabMux ETI file formats

ODR-DabMux supports three output formats for the ETI stream, that have been described on the mmbTools forum website.[8]

The three formats are called *framed*, *streamed* and *raw*.

The *framed* format is used for saving a finite ETI stream into a file. Each frame does not contain any padding, and the format can be described as follows:

```
uint32_t nbFrames
// for each frame
  uint16_t frameSize
  uint8_t data[frameSize]
```

When streaming data, in which case the number of frames is not known in advance, the *streamed* format can be used. This format is identical to the first one except for the missing `nbFrames`.

```
// for each frame
  uint16_t frameSize
  uint8_t data[frameSize]
```

The *raw* format corresponds to ETI(NI), where each frame has a constant size of 6144 Bytes. The padding in this case is necessary.

```
// for each frame
  uint8_t data[6144]
```

In order to select the format, the following syntax for the `-O` option is used: `-O file://filename?type=format`, where `format` is one of `framed`, `streamed` or `raw`.

B Bibliography

References

[1] ETSI. *TR 101 495, Digital Audio Broadcasting (DAB); Guide to DAB standards; Guidelines and Bibliography*, November 2000. V1.1.1. All DAB standards are available at `http://www.etsi.org/WebSite/Technologies/DAB.aspx`.

[2] Hoeg, W., and Lauterbach, T. *Digital Audio Broadcasting; Principles and Applications of DAB, DAB+ and DMB*. John Wiley & Sons Ltd., 2009.

[8]`http://mmbtools.crc.ca/component/option,com_fireboard/Itemid,55/func,view/id,4/catid,13/#28`