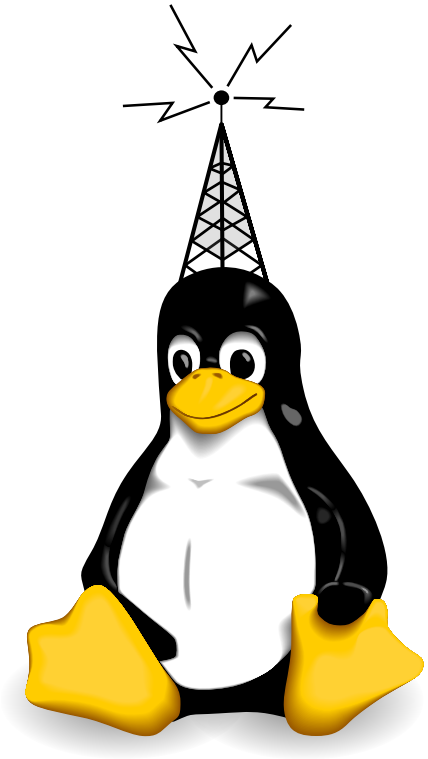
ODR-mmbTools
Open-Source Software-Defined DAB$^+$ Tools

Project Documentation

Opendigitalradio
http://opendigitalradio.org
2014–2022



Contents

Acronyms

1PPS	One pulse per second
CFR	Crest Factor Reduction
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
MER	Modulation Error Rate
MNSC	Multiplex Network Signalling Channel
NTP	Network Time Protocol
OCXO	Oven-Controlled Crystal Oscillator
OFDM	Orthogonal Frequency-Division Multiplexing
PAPR	Peak-to-Average Power Ratio
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 Introductions

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

This documentation assumes that you are already familiar with the basic concepts of the DAB system. Understanding how the DAB transmission chain is structured is a prerequisite for getting started with the ODR-mmbTools. The "DAB Bible" by Hoeg and Lauterbach [8], and the "Guide to DAB standards" from the ETSI [2] 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 individual programs that make up the ODR-mmbTools each have their own documentation for 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 toolset.

This document aims to fill this gap, by first outlining general concepts, then presenting different usage scenarios, and finally, detailing a complete transmission setup. With this document in hand, you should be able to understand all of the elements which go into the ODR-mmbTools transmission chain, and how to set one up.

Please refer to the bibliography for references on any individual topic that may need clarification, to the README files in the repositories of the tools that are going to be presented in this guide, and if you have further questions, get in touch with us through the mailing-list mentioned on our website.

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 was able to create baseband complex quadrature (I/Q) samples from files or streams in the ETI format. This I/Q data could then be sent to a hardware device (for broadcast or laboratory RF measurements) using another tool. For driving the universal software-defined radio peripherals (USRP) made by the company Ettus Research,

[1] `http://opendigitalradio.org`
[2] `http://crc.ca`

a "wave player" script was necessary to interface with 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.

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 ultimately never included in the CRC-mmbTools, and as time passed, 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 so, with the approval of CRC, the tools were officially forked 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 source. 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. The encoder ODR-AudioEnc can use this library to encode for DAB$^+$.

3.2 Included Tools

The ODR-mmbTools are composed of several software projects: ODR-DabMux, ODR-DabMod, ODR-AudioEnc, ODR-PadEnc, and other scripts, bits and pieces that are useful when setting up a transmission chain.

3.2.1 ODR-DabMux

ODR-DabMux implements a DAB multiplexer that combines all audio and data inputs and outputs them in the form of a file in ETI format. This can be used offline (i.e. not in real time) to generate ETI data for processing later, or for use in a real-time streaming scenario (e.g. in a transmitter).

ODR-DabMux can read input audio or data from files (".mp2" for DAB, ".dabp" for DAB$^+$), FIFOs (also called "named pipes"), or from a network connection. This network connection can use UDP (STI-D) or EDI.

The configuration of the multiplexer is given in a configuration file, whose format is defined in the example files in the `doc/` folder inside the ODR-DabMux repository.

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

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

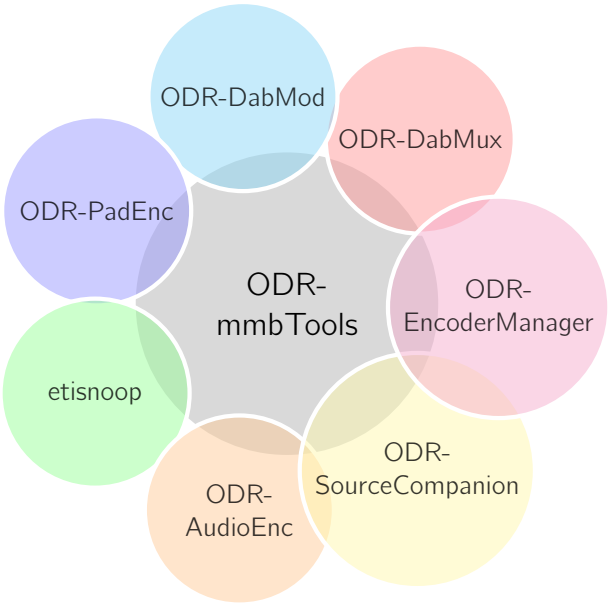


Figure 1: The family of ODR-mmbTools

3.2.2 ODR-DabMod

ODR-DabMod is a software-defined DAB modulator that receives or reads ETI data in streams or from files, and generates modulated I/Q data which can be used for transmission.

This I/Q data which is encoded as complex floats (32bits per complex sample) can be written to a file or pipe, sent to a USRP device using the integrated output for the open-source USRP Hardware Driver (UHD) or to other software-defined radio (SDR) devices using the SoapySDR[5] library.

The output of the modulator can also be sent to a GNURadio flow-graph for further processing, conversion or analysis using a ZeroMQ network connection.

3.2.3 ODR-AudioEnc

The ODR-AudioEnc encoder can be used to encode for DAB and DAB$^+$. It includes a TooLAME-based MPEG encoder, and uses the fdk-aac library as an external dependency to encode DAB$^+$.

The integrated TooLAME library is an MPEG-1 Layer II audio encoder that is used to encode audio for the DAB standard. Another encoder called twolame is not compatible with DAB even though it is more recent than TooLAME, and cannot be used for our application.

The framing and error correction which are needed for DAB$^+$, as well as the programme-associated data (PAD) insertion, the output EDI protocol, and the input from Advanced Linux Sound Architecture (ALSA) were then added by different parties.

[5] `https://github.com/pothosware/SoapySDR/wiki`

3.2.4 ODR-PadEnc

This encoder is able to generate programme-associated data (PAD) that can be injected into ODR-AudioEnc. It supports reading and encoding Dynamic Label Segment (DLS) from a text file, and reads images from a folder for MOT Slideshow.

3.2.5 ODR-EncoderManager

The ODR-EncoderManager presents a web-based interface that allows the user to create, manage and run audio and PAD encoders, and presents a HTTP API to update Dynamic Label Segment and Slides. One instance can handle several audio encoders simultaneously, and offers a simpler way to manage the audio encoding part of the DAB$^+$ transmission chain.

3.2.6 ODR-SourceCompanion

This tool allows using third party audio encoders with the ODR-mmbTools.

3.2.7 etisnoop

Etisnoop is not used in the broadcasting chain directly, but is an analysis tool for ETI, described in the ETSI standard [1]. ODR-DabMux can write an ETI file that can be analysed with etisnoop. The tool can be used to verify the multiplex signalling, the presence of data in the subchannels, and it can decode audio into files.

Additionally, it can output statistics in YAML format, which is useful in combination with an RTLSDR receiver and the `dab2eti` tool to monitor transmissions.

4 Installation of the Tools

There are 3 ways to install the tools.

4.1 Debian Binary Packages

If your host is running a debian-Bullseye-based operating system and its architecture is one of amd64, arm64 or arm/v7, then you can easily install ODR-AudioEnc, ODR-PadEnc, ODR-DabMux and ODR-DabMod from the ODR package repository by applying the following steps:

```
1  curl -fsSL http://debian.opendigitalradio.org/odr.
       asc | sudo tee /etc/apt/trusted.gpg.d/odr.asc
       1>/dev/null
2  curl -fsSL http://debian.opendigitalradio.org/odr.
       list | sudo tee /etc/apt/sources.list.d/odr.
       list 1>/dev/null
3  apt update
4  sudo apt install --yes odr-audioenc odr-padenc odr
       -dabmux odr-dabmod
```

Remarks The `odr-dabmux` and `odr-dabmod` packages do not include the web-based GUI Mux Manager and the GUI and Digital Predistortion Computation engine. If you need those, then you should look at the other 2 installation options below.

4.2 Opendigitalradio-provided Installation Script

This option allows you to compile and install:

- the above 4 main components of the tools
- the web-based GUI Encoder Manager, GUI Mux Manager, GUI and Digital Predistortion Computation engine
- sample configuration files

Apply the following steps:

```
1  sudo apt update && sudo apt upgrade --yes
2  sudo timedatectl set-timezone your_timezone
3  cd ${HOME}
4  git clone https://github.com/opendigitalradio/dab-
       scripts.git
5  bash ${HOME}/dab-scripts/install/mmbtools-get
       install
```

Remarks The installation script will compile the tools with all the possible features in order to give you the greatest configuration flexibility. It will also install and configure the `supervisord` package with the sample configuration files.

4.3 Manual Compilation

If you wish to compile and install some tools only and reduce disk usage by selecting the needed features, then you should follow the instructions given with each tool:

- `odr-audioenc` https://github.com/opendigitalradio/odr-audioenc
- `odr-padenc` https://github.com/opendigitalradio/odr-padenc
- `odr-dabmux` https://github.com/opendigitalradio/odr-dabmux
- `odr-dabmod` https://github.com/opendigitalradio/odr-dabmod

5 Interfacing the Tools

5.1 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
- one DAB$^+$ programme at 88kbps

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:

```
odr-audioenc --dab -b 128 -i prog1.wav -o 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:

```
odr-audioenc -i prog2.wav -b 88 -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 $5000 * 6144 \approx 30\text{MB}$ 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`

5.2 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.

The standard protocol to carry both contribution (From audio encoder to multiplexer) and distribution (from multiplexer to modulators) is EDI, specified by ETSI [4] [6].

EDI can be carried over UDP or other unreliable links, and offers a protection layer to correct bit-errors. Over network connections where the occasional congestion can occur, EDI can also be carried over TCP, which will ensure lost packets get retransmitted. Unless you are able to guarantee reserved bandwidth for the EDI traffic, using TCP is the safer option.

While the main reason to use EDI is to put the different tools on different computers, it is not necessary to do so. It is possible, and even encouraged to use this interconnection locally on the same machine, for increased flexibility.

5.2.1 Between Encoder and Multiplexer

Between ODR-AudioEnc and ODR-DabMux, the EDI protocol carries DAB$^+$ superframes or DAB frames, with additional metadata that contains the audio level indication, a version field and a free-form identifier string for monitoring purposes.[7] The multiplexer cannot easily derive the audio level from the encoded bitstream without decoding it, so it makes more sense to calculate this in the encoder and carry it along the encoded data.

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

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

    inputproto edi
    inputuri "tcp://*:7002"
    buffer-management prebuffering
}
```

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

- `inputuri`: This defines the interface and port on which to listen for incoming data. It must be of the form `<proto>://*:<port>`, with `proto` may be either `tcp` or `udp`.
- `buffer-management`: Two buffer management approaches are possible with EDI: The other option `timestamped` will take into account the timestamps carried in EDI, inserting the audio into the ETI frame associated to that same time stamp. `prebuffering` ignores timestamps and pre-buffers some data before it starts streaming. This allows to compensate for network jitter.

[6] For a summary about the ZeroMQ interface used before EDI, see the section 5.3 below.

[7] This metadata is carried in the custom EDI TAGs `ODRv` and `ODRa`.

- `buffer`: (Both buffer management settings) The input contains an internal buffer for incoming data. The maximum buffer size is given by this option, the units are 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 when the buffer is full.
- `prebuffering`: (Only in buffer management `prebuffering`) When the buffer is empty, the multiplexer waits until this amount of 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. In both buffer management techniques, it is a trade-off between absolute delay and robustness. When using pre-buffering, you directly control size of the buffer, and you set it to a value depending on your network delays. When using timestamped buffer management, the size of the input buffer is a consequence of the effective delay you set in the timestamps.

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 input buffer in the multiplexer will fill up to the amount given by `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 fast, 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 frame 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 ODR-AudioEnc, which insures that the encoder outputs the encoded 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.

5.2.2 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.

5.2.3 Between Multiplexer and Modulator

The EDI protocol can also carry data of a complete ensemble from ODR-DabMux to one or more instanced of ODR-DabMod.

In case you with to interface ODR-DabMux with a modulator that does not support EDI over TCP, but your network is not stable enough to use UDP, you can use ODR-EDI2EDI. See `http://github.com/Opendigitalradio/ODR-EDI2EDI` for information about that tool.

5.3 ZeroMQ

Previous versions of this guide described an IP protocol based on ZeroMQ, a library that permits the creation of a socket connection with automatic connection management (connection, disconnection, error handling). This has now been deprecated in favour of the standards compliant EDI, which, when carried over TCP, can also be used over congested networks like the Internet.

ZeroMQ was used both between for encoded audio contribution and for distribution of ensemble data, and was a custom, non-specified protocol. It was created before EDI got implemented in the ODR-mmbTools, and because carrying ETI inside IP was not sufficient for carrying complete timestamps, and because there was no other obvious option for audio contribution besides inventing a custom protocol.

Now that it is proven that EDI-over-TCP can work satisfactorily over congested networks, there is no reason for a protocol that is not conforming to ETSI specifications to exist anymore.

5.4 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 EDI.

6 Usage Scenarios

6.1 Experimentation

6.1.1 Creation of Non-Realtime Multiplex

The creation of a ETI file containing two programmes, one DAB and one DAB$^+$ is covered in section 5.1.

6.1.2 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 -n 1
```

This will create a file containing 32-bit interleaved I/Q at 2048000 samples per second. Where the maximal amplitude is bounded by 1.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]
format=complexf
normalize=1
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 available in the `doc/` folder.

6.2 Interfacing Hardware Devices

6.2.1 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]
digital_gain=0.8

[firfilter]
enabled=1

[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.
- `digital_gain=0.8` reduces the output sample deviation, to reduce compression in the USRP.
- `firfilter enabled=1` enables the FIR filter to improve the spectrum mask. If you want to customise the filter used, you can create your own filter taps file using `ODR-DabMod/doc/fir-filter/generate-filter.py`.
- `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. If you go above 70dB you will start to see nonlinearities.

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 depicted in figure 2 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.

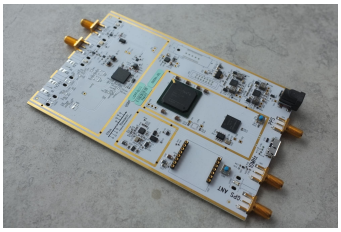


Figure 2: Ettus USRP B200

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.[8]

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.

6.2.2 Other Hardware with SoapySDR

ODR-DabMod supports other radio interfaces using the SoapySDR[9] vendor-neutral and platform independent library to drive SDR devices. It can be used to drive the LimeSDR boards, the HackRF from Great Scott Gadgets and the Fairwaves XTRX devices, among others. Installation dependencies are shown in the `INSTALL` file, and an example configuration is in `doc/example.mux`.

The available sampling rates, the TX gain range and the antenna selection are device-specific, and can be discovered using the `SoapySDRUtil -probe` command.

For example, the LimeSDR mini requires the `tx_antenna` to be set to BAND2.

6.2.3 HackRF Through Stdout Pipe

For devices that are not offering a SoapySDR device driver, the last resort is to use standard output or a fifo to carry the IQ data from ODR-DabMod to a tool that can drive the device. The fifo must be created prior to runtime with the `mkfifo` command.

We are going to illustrate this with the HackRF, even though there is a SoapySDR device driver for it, but it is equally applicable for other devices.

[8] `http://wiki.opendigitalradio.org/index.php/USRP_B200_Measurements`
[9] `https://github.com/pothosware/SoapySDR/wiki`

The HackRF is an entry level yet versatile SDR which provides coverage between ≈ 10MHz to 6GHz, and DAB signals been successfully generated with it in VHF Band III (174–240MHz), L-Band (1462–1467.5MHz) and even the worldwide ISM Band (2400–2500MHz). The latter (subject to local regulations) is a licence exempt band which may be useful for performing freely radiating tests at low power. Cheap MMDS converters are currently available which helpfully provide a Band III IF output providing a direct feed to the aerial input of a receiver. Before choosing a converter it is important to pay close attention to the specifications. The local oscillator phase noise performance, and the dynamic range (due to the heavy use of the band) are both particularly important.

To use the HackRF through stdout (i.e. without SoapySDR), the output of ODR-DabMod must be set (in the configuration file) to produce 8-bit signed integers, rather than the default complex floats.[10] The HackRF has selectable baseband filters, however the lowest filter setting (1.75MHz) does not provide adequate image rejection at the native sampling rate of 2048k samples per second. An appropriate rate to start with is 4096k, and for some purposes this may well be adequate as this moves the image signals generated within the radio far enough into the stop-band of filter to attenuate them significantly. Since ODR-DabMod v1.0.1, the digital gain setting is not be influenced by the sample rate anymore, and should be set below 1, with some margin, to avoid digital clipping on modulation peaks.

Example of the settings in the `mod.ini` file suitable for use with HackRF:

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

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

[modulator]
digital_gain=0.8
rate=4096000

[firfilter]
enabled=1

[output]
output=file

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

[10] UHD versions before 3.12 output a version string to standard output at startup. This interferes with all ODR-DabMod usage with `/dev/stdout` as output, and will only function correctly if ODR-DabMod is configured at compilation time with UHD disabled.

The output fifo has to be created beforehand, and the `hackrf_transfer` utility is then used to transmit the signal to the device.

Depending on the capabilities of the host computer, using higher sampling rates (6144k, and even 8192k) may be possible. This oversampling is desirable as it helps to produce a cleaner spectral output. At higher rates one needs to ensure that samples are not being dropped on the USB and that CPU resources are not being contended.

The shoulder performance has been measured with a value at a little better than 35dB, which is roughly equivalent to that obtained from first generation commercial modulator equipment. This can be increased to a relatively respectable $\approx$ 40dB by enabling the FIR baseband filter in ODR-DabMod. The maximum output power available to meet these performance figures is approximately -10dBm RMS.

Example of using ODR-DabMod with the `hackrf_transfer` utility:

```
mkfifo /tmp/ofdm.fifo
odr-dabmod mod.ini &
hackrf_transfer -t /tmp/ofdm.fifo -f 216928000 -x 47 \
    -a 1 -s 4096000 -b 1750000
```

6.3 Advanced Signal Processing

6.3.1 Crest Factor Reduction

ODR-DabMod contains a prototype for crest factor reduction (CFR), which allows you to reduce peak-to-average power ratio (PAPR), trading off with modulation error rate (MER). The DAB OFDM signal has a very high PAPR, which directly translates to a decrease in power amplifier efficiency. The power amplifier has to be driven such that the peaks do not drive it into compression, but the overall efficiency is calculated from the average power. Reducing the PAPR makes it possible to drive the amplifier more.

The CFR algorithm works in the following way: all the generated OFDM samples go through a limiter, which clips the amplitude to a configurable value. This directly reduces PAPR, but impacts both amplitude and phase of the constellation points, thereby degrading MER. To compensate for this, a second step of the algorithm calculates the error vector for each constellation point, and clips the error to some maximum amplitude. The clipped error is then subtracted from the signal, which corrects part of the distortion created by the limiter.

To summarise: a low clipping value distorts the signal more; a high error clipping value corrects the distortion again. In the constellation plot view, the first is seen as an increase in spread of the points; the second is visible because it pulls the constellation points back into a circle of radius proportional to the error clipping value.

Settings and some statistics are available through the remote control.

6.3.2 OFDM Symbol Windowing

One technique to improve the shoulder performance is to avoid generating abrupt transitions between the OFDM symbols, but apply cross-fading from one symbol

to the next using a windowing function.[11]

This feature can be enabled by setting the number of samples to use for overlapping one symbol with the next one, the default value is 10 (out of a total symbol length of 2552 samples in Transmission Mode I). Refer to the parameter `ofdmwindowing` in the file `doc/example.ini` for instructions.

As this windowing feature modifies samples that are in the guard interval, the trade-off is reduced resilience against delayed reflections, reduced maximum transmitter delay difference in an SFN scenario, and potentially diminished robustness against Doppler spread.

6.3.3 Digital Pre-Distortion

An ongoing activity is the development of a method to characterise a power amplifier and predistort the I/Q samples to invert the distortion behaviour of the amplifier. More information about this work is in the `dpd/README.md` file in ODR-DabMod.

6.4 Audio Sources

Preparing a DAB multiplex with different programmes requires that we are able to read and encode several audio sources. We have seen in section 5.2.1 how the encoders can be interfaced to the modulator. In this section we'll go through the different ways to carry the audio data to the encoder.

6.4.1 Local Audio Card

It is possible to use an audio card connected to the computer as source. For very simple scenarios, the ALSA input for ODR-AudioEnc is easiest to set up. This however limits the usage of a single encoder per sound-card, and will not scale well if more than one programme has to be encoded on the machine. It is however ideal for dedicated encoding machines that can contribute the encoded audio over an IP network.

An alternative to using ALSA is JACK[12] that can be used with a multi-channel sound card. JACK will expose every audio input channel, and several encoders can be launched that also connect to JACK. The input channels can be freely connected to the encoders thanks to the virtual JACK patch panel.

It might be possible to use the libVLC input too, to be defined.

6.4.2 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.

[11] As of ODR-DabMod v1.0.1, this feature is still in the `next` development branch, and not part of a released version.

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

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. By far the easiest way is to use the libVLC binding that can be compiled for ODR-AudioEnc. This library has the same features as the VLC audio player, but the audio data is directly passed to the encoding routines. This allows the encoder to receive all network sources VLC supports, not only HTTP web-streams but also less common setups e.g. encoded audio inside multicast UDP MPEG-TS. This is illustrated in "Studio A" in figure 3.

We have also achieved good results with mplayer, and the dab-scripts repository[13] contains the script `encode-jack.sh` that uses mplayer, and illustrates how it is possible to encode a web-stream to DAB$^+$. JACK is used to interconnect the stream decoder to the DAB$^+$ encoder. This is illustrated in "Studio B".

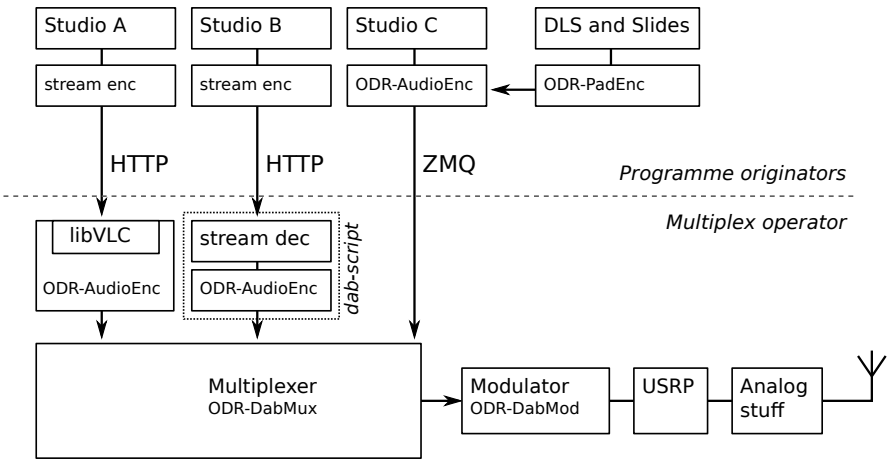


Figure 3: Three common ways to encode a remote audio sources.

The scripts are designed for production use, and also contain automatic restart logic in case of a failure. They send an email and write a message into the system log.

6.4.3 Encoders at Programme Originator Studios

In order to avoid the unavoidable encoder cascading when using mp3 web-streams, the DAB or DAB$^+$ encoder has to be moved to the programme originator's premises, and should directly encode the audio signal coming from the studios. This is illustrated in "Studio C" in figure 3.

If "Studio C" is able to prepare slides for MOT Slideshow and text to be sent as DLS, ODR-PadEnc can be used to prepare the PAD data for ODR-AudioEnc.

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

7 Data Features

7.1 Programme-Associated Data

It is possible to encode Dynamic Label Segment and Slideshow using ODR-PadEnc, and to inject the PAD data into the audio encoder.

ODR-AudioEnc and ODR-PadEnc need to be launched with the same PAD socket identifier, and they will be able to communicate. The PAD length specifies the amount of data that is taken away from audio and used for PAD. Valid values are 6, and the range 8 to 196. When not transmitting slides, small PAD lengths are perfectly suitable. When using slides, it is better to use values around 30. Higher lengths will of course accelerate the transmission of the slide at the expense of reduced audio quality during the transmission time.

ODR-PadEnc will itself only take DLS and slides from files on the system it runs on. If your playout system is able to push updates using FTP, you will need to configure and FTP server to present the right folder.

A more modern approach is offered by ODR-EncoderManager, which will not only configure and run your encoders, but also present you an HTTP API to update DLS and upload slides. More information is available in its README.

7.2 FIG 1 Labels and FIG 2 Extended Labels

The specification offers two ways to carry ensemble, service and component labels: through FIG 1 and through FIG 2, specified in clauses 5.2.2.2 and 5.2.2.3 of ETSI EN 300 401 [6].

Most receivers are only able to show FIG 1 Labels encoded in the Complete EBU Latin character set (defined in ETSI TS 101 756 clause 5.2 [7]). Some are able to display Unicode FIG 1 Labels, encoded either in UTF-8 or UCS-2, and, as of early 2019, receiver support for FIG 2 Extended Labels is practically absent.

The main downside of carrying Unicode FIG 1 Labels is the length limitation: 16 bytes will only encode eight characters in alphabets that require two bytes per character. FIG 2 supports up to 32 bytes labels to alleviate this.

The intention is that new ensembles in countries requiring labels in non-latin alphabets transmit only FIG 2 Extended Labels, whereas currently operating ensembles keep transmitting FIG 1 Labels. This entices receiver manufacturers to support FIG 2 without impacting functionality of receivers currently in use. Transmitting both FIG 1 and FIG 2 is discouraged by the specification.

The way FIG 2 is encoded has been redefined, which is why ODR-DabMux supports two variants: FIG 2 with character flag being the old variant, and FIG 2 with text control that will become the default variant.

7.3 Announcements

The ODR-DabMux multiplexer supports the insertion of FIG 0/18 and FIG 0/19 that are used to define and trigger announcements according to ETSI TR 101 496-2 Clause 3.6.8 [3]. An example configuration is available in the ODR-DabMux repository, in `doc/advanced.mux`.

The best known application for announcements is traffic information, but other kinds of announcements can also be signalled. ODR-DabMux allows triggering the announcements through the telnet and ZMQ remote control interfaces.

7.4 Service Linking

ODR-DabMux also supports the ability to inform receivers about other ways to receive a given service, through the FIGs 0/6, 0/21 and 0/24. FIG 0/6 communicates the identifiers of services linked together, 0/21 informs the receiver about other frequencies, and 0/24 includes information about other DAB ensembles carrying the linked service. Their interaction is outlined in ETSI TS 103 176 [5].

You will find an example configuration in the ODR-DabMux repository, in `doc/servicelinking.mux`.

8 System Environment

In this section, we describe the system configuration requirements for the continuous operation of the tools. The production environment differs in some respects to those used for experimentation and in laboratory testing. Monitoring, automatic recovery (in case of errors) and resilience are crucial in 24/7 operations. The term *production environment* will be used here to refer to such use.

8.1 Processing requirements

The tools have differing requirements regarding CPU performance and amount of memory, and while the performance of most desktop PCs is sufficient to run the tools, it is important to take the requirements in consideration when setting up a system. Memory requirements are easily met with 1GB of RAM, so we'll look at CPU more in depth.

The most resource-consuming part is the modulator ODR-DabMod. The following impact its CPU usage: number of sub-channels; enabling of the resampler; enabling crest factor reduction; enabling the predistorter. Compilation options to optimise ODR-DabMod for your system are described in the README. While you should have no trouble running it even on an older desktop PC, the computing power of embedded ARM boards (like the Raspberry Pi) could be insufficient, especially if the resampler is needed.[14] When using a USB SDR device, the USB controller can have a large impact on the robustness of the transmission, even if CPU usage is low. Such issues are visible as underruns during operation: with a good controller, less than one underrun per day is easily achievable on a machine dedicated to only this task. When using a graphical interface at the same time, interaction with the user interface can also trigger underruns. For a production system, it is better if no graphical user interface is running. In any case, it is required to evaluate a given system over several days if reliable operation is to be proven.

The multiplexer ODR-DabMux mostly rearranges data internally, and doesn't do much processing. Its resource requirements are low and it runs well on small systems. The same goes for ODR-PadEnc, ODR-EDI2EDI, ODR-zmq2edi and ODR-zmq2farsync.

Audio encoding using ODR-AudioEnc is in-between ODR-DabMux and ODR-DabMod in terms of resource usage, and running one encoder is not a problem even on small embedded ARM boards. However, you might want to run a dozen encoders on a single machine, where you will have to plan for more headroom.

In general, for a robust 24/7 system, you should strive for a CPU usage below 50%, regardless of which tools you are using. This gives you headroom for monitoring, remote administration and background jobs run by cron. Once your system is in operation, monitoring performance and observing logs is essential to assess the health of your transmission.

[14] See section 6.2.2 for an example.

8.2 Launching the tools

Services running in a production environment are usually administered remotely, and must be able to run without user intervention, or connection. Traditionally, such services are implemented (in UNIX terminology) as 'daemons'. These are started and stopped using the init system contained within the distribution. As the ODR-mmbTools cannot daemonise themselves, a process supervisor is used.

supervisord One possibility is to use `supervisord`[15] which can launch the tools and monitor their proper execution. It will restart the processes and optionally inform the operator by email.

Once installed, supervisord reads its configuration file in `/etc/supervisor.conf` and launches the processes that are to be monitored. Each process is described by a file. The following example assumes the tools are run as user `odr`, and that the multiplex configuration is in `/home/odr/config.mux`, and that ODR-DabMux is to be launched. The standard output and standard error streams of ODR-DabMux are written to the specified log files.

```
[program:ODR-DabMux]
command=odr-dabmux config.mux
directory=/home/odr
user=odr
autostart=true
autorestart=true
stdout_logfile=/home/odr/logs/mux.out.log
stderr_logfile=/home/odr/logs/mux.err.log
```

Once this configuration has been added to the supervisord configuration, the settings have to be re-read using:

```
supervisorctl reread
```

In order for supervisord to start managing and running this process, it needs to be added:

```
supervisorctl add ODR-DabMux
```

Setting up more processes (such as any of the other tools) can be easily achieved by customising the configuration template above. Examples are provided in the `mmbtools-aux` repository, under the `supervisor` folder - these need to be changed to reflect the paths that are in use on your system.

supervisord also includes a small web-server that can display the state of the managed processes. It is enabled with the `[inet_http_server]` setting in the configuration file.

systemd Most recent GNU/Linux distributions use `systemd` as init system, which also can handle the supervision of processes. To achieve this, systemd unit files have to be written for the tools. For more information, see the systemd documentation.

Give an example unit file

[15] `http://supervisord.org`

8.3 Logging

Collecting information about events is essential within a production environment. This information is essential forensic analysis, and tracing sources of trouble. This is achieved through the logging of important messages that can be sent by the tools.

ODR-DabMux and ODR-DabMod both support logging to standard error, to a file and to the system logger `syslog`. Logging to syslog is the most flexible approach; log information can be forwarded over the network to a centralised logging server - where logs can then be filtered according to the priority of each message. Both tools log to the LOCAL0 facility which in turn can be redirected into an ODR-mmbtools specific log file.

Describe rsyslog configuration

In order to avoid the log files from becoming undesirably large, `logrotate` should be set to rotate the files automatically.

Describe logrotate configuration

8.4 Timing

The ODR-mmbTools require the system time to be accurate in order for them to function correctly - this is especially important when running a SFN, but is also important for standalone transmitters when in a production environment. It is also important to remember that most receivers have a clock that is synchronised to the clock time which is being transmitted by the multiplex to which it has been tuned.

The system needs to run a NTP client that synchronises the system time over the network. Correct synchronisation can be checked using `chronyc tracking` or the the `lpeers` command of the `ntpq` tool, depending on if you use chrony or openntp. The magnitude of the offset should be below 10 ms.

The performance of the NTP synchronisation should also be monitored permanently during operation.

ODR-DabMux can run a command at startup to verify if the NTP client is properly working, using the `startupcheck` setting. This can be used to call `ntp-wait` or `chronyc waitsync` to wait for proper NTP sync.

8.5 Monitoring through SNMP

There is ongoing work to make the monitoring of the tools possible using SNMP. Please see `https://wiki.opendigitalradio.org/SNMP` for information about this effort.

8.6 Monitoring using munin

The Munin[16] monitoring tool can create graphs for essential system health parameters. It can also send emails if values transgress the defined bounds - this assists the operator in the assessment of system status, as well as the health of the services.

[16] `http://munin-monitoring.org/`

In addition to basic system measurements like CPU, RAM and disk usage, NTP synchronisation, disk and network performance, there are custom data sources available for ODR-DabMux and ODR-DabMod.

The ODR-DabMux data sources include EDI and ZMQ input buffer monitoring (buffer level, underruns and overruns) and the peak audio input levels (mono, or stereo). The plugin for ODR-DabMod can monitor SDR device statistics among others.

The plugins are written in python and are located in the `doc` folder in the repositories. Copy them to `/etc/munin/plugins.d` and restart munin-node. They require that the ODR-DabMux management server, and the ZeroMQ remote control enabled on the ports give in the example configurations.

8.7 Monitoring using Xymon

The xymon monitoring tool[17] is used to monitor the health of many types of systems. It can present the results in text, tables and/or graphs. It supports the basic health checks directly out of the box, and can be extended with scripts to perform non-standard health checks. The default mode of operation is that clients retrieve data and send it to the xymon server, which interprets the results, displays them and generates alerts if thresholds are exceeded. An alert can be send in an e-mail, an SMS or a tweet.

The Perl script `retodrs.pl`[18], retrieves the status and statistics of an Opendigitalradio service and it reports the results to xymon. The information is retrieved from the management server within ODR-DabMux. The information presented includes a table with the status of each sub-channel and the underrun and overrun rates on the sub-channels. If needed an alert can be generated depending on the subchannel status or a rate exceeding a threshold.

The script needs to be installed on the same server running ODR-DabMux, as the management service within it is only accessible from the same computer. This implies that the xymon client software also needs to be installed on the same machine. The client is configured to run the script. The configuration and the scripts can typically be found in subdirectory `/usr/lib/xymon/client`, although that may depend on your distribution.

Once the client is set up, it needs to connect to a xymon server, which may or may not be on the same machine. The server is configured to limit the altering to specific sub-channels, to store the statistical data and to generate graphs. The configuration and the scripts on a xymon server are usually stored in the subdirectory `/usr/lib/xymon/server`.

8.7.1 Installation of the Xymon Client

The perl script has additional requirements: `App::cpanminus`, `ZMQ::LibZMQ3`, and `JSON::PP`. They can be installed through your distribution packages or using CPAN.

Once the script has been copied to `/usr/lib/xymon/client/ext`, the configuration of the launcher within the xymon client needs to be extended. Create a

[17] `http://xymon.sourceforge.net/`

[18] The script name stands for "Retrieve Opendigitalradio Status"

new file named odrmux.cfg in /usr/lib/xymon/client/etc/clientlaunch.d containing the following lines:

```
#
# Test odrmux checks the state and the statistics
# of the ODR-DabMux service.
#
[odrmux]
ENVFILE $XYMONCLIENTHOME/etc/xymonclient.cfg
CMD $XYMONCLIENTHOME/ext/retodrs.pl
LOGFILE $XYMONCLIENTLOGS/retodrs.plog
INTERVAL 5m
```

After a restart of the xymon client, the script retodrs.pl will be invoked once every 5 minutes.

8.7.2 Server Configuration

By default all subchannels will be monitored, and will raise alerts if the status or the statistics are in outside of a valid operational range. The alerting can be limited to a subset of the sub-channels by adding a tag to the hosts-entry in the configuration file /usr/lib/xymon/server/etc/hosts.cfg. The additional tag is:

```
ODR:select(<SubChannelName0>;<SubChannelName1>;...)
```

The sub-channels not named will still be shown, but no alerts will be generated for those sub-channels. This is visible as the green/yellow/red icons are missing for those sub-channels.

Six statistic values are gathered by the script, namely BufferMin, BufferMax, PeakLeft, PeakRight, UnderRun and OverRun. It is found that only the latter two seem to contain sensible values all the time, so those values are the only ones shown in a graph. Note that those values retrieved by the script are ever-increasing counters, showing the total number of over-runs or under-runs. In the graph, the average number of over-runs or under-runs per second, averaged over a period of 5 minutes, is shown.

The first step is to have the collected statistics to be moved into a database, a so-called *Round Robin Database*. This is accomplished by adding a file named odr.cfg in /usr/lib/xymon/server/etc/xymonserver.d containing the following lines:

```
TEST2RRD+=",odr_mux=devmon"
GRAPHS+=",odr_mux::1"
```

The next step is to define the layout of the graph. Create a file named graphs.odr.cfg in /usr/lib/xymon/server/etc/graphs.d containing the following lines:

```
#
# Graphs to show the statistics collected from an
# Opendigitalradio DabMux server.
#
```

```
[odr_mux]
FNPATTERN ^odr_mux\.(.+)\.rrd$
TITLE , Frame loss rate
YAXIS Rate [/s]
-l 0
DEF:ur@RRDIDX@=@RRDFN@:Underrun:AVERAGE
DEF:or@RRDIDX@=@RRDFN@:Overrun:AVERAGE
LINE1:ur@RRDIDX@#FF0000:@RRDPARAM@ underrun
GPRINT:ur@RRDIDX@:MIN:Min \: %5.1lf %s
GPRINT:ur@RRDIDX@:MAX:Max \: %5.1lf %s
GPRINT:ur@RRDIDX@:AVERAGE:Avg \: %5.1lf %s
GPRINT:ur@RRDIDX@:LAST:Cur \: %5.1lf %s\n
LINE1:or@RRDIDX@#00FF00:@RRDPARAM@ overrun
GPRINT:or@RRDIDX@:MIN: Min \: %5.1lf %s
GPRINT:or@RRDIDX@:MAX:Max \: %5.1lf %s
GPRINT:or@RRDIDX@:AVERAGE:Avg \: %5.1lf %s
GPRINT:or@RRDIDX@:LAST:Cur \: %5.1lf %s\n
```

8.8 Real-time Scheduling

As a general principle, it is prudent not to run tools (that do not need superuser privileges) as the `root` user. The same principle also applies to the ODR-mmbTools, but care has to be taken that the tools can still request real-time scheduling when it is needed.

This is achieved by adding the following to `/etc/security/limits.conf`, assuming the tools are run under the user `odr`.

```
1 odr            -        rtprio           65
2 odr            -        nice            -10
```

If you have installed JACK with real-time privileges, you may find this has already been configured for the 'audio' group, written as `@audio`, which should suffice providing your desired user is a member of the 'audio' group.

8.9 Accessing the USRP as Non-root

Superuser privileges are not required to access USB-connected USRP devices, but sometimes the system lacks the configuration to enable normal users to communicate with the device. In that case, it is necessary to add a rule file for `udev`. This file is included in the UHD sources, but might not have been automatically installed. The file is called `10-uhd-usrp.rules`, should be placed into `/etc/udev/rules.d/` and should contain

```
#USRP1
SUBSYSTEMS=="usb", ATTRS{idVendor}=="fffe", ATTRS{idProduct}=="0002", MODE:="0666"
#B100
SUBSYSTEMS=="usb", ATTRS{idVendor}=="2500", ATTRS{idProduct}=="0002", MODE:="0666"
#B200
SUBSYSTEMS=="usb", ATTRS{idVendor}=="2500", ATTRS{idProduct}=="0020", MODE:="0666"
```

9 A Production Broadcast Setup

9.1 Outline

We have now seen all necessary elements for a complete broadcast chain, and we will now consider what is necessary to use these elements in a 24/7 production environment. At this point, many previously considered topics come together to form a reliable system.

First, let us outline what our desired setup shall include:

- We want to transmit about a dozen programmes using a single transmission site, i.e. no SFN;
- Some audio sources are web-streams, some are using remote ODR-AudioEnc encoders;
- One machine is used for audio encoding web-streams, multiplexing and modulation;
- ODR-AudioEnc instances will connect over EDI to ODR-DabMux, ODR-DabMod will use EDI to connecto to ODR-DabMux;
- All audio encoders will insert PAD with DLS, and optionally slideshow;
- We are transmitting using a USRP B200, driving a power amplifier;
- We enable both telnet and zmq remote control interfaces for management purposes;
- The power amplifier will be driven linearly, no digital-predistortion is used;
- We must respect the spectrum mask given by the broadcast license;
- The setup must be resilient to program failure and restart them automatically, also informing the operator;
- We use munin to monitor the operation of the system.

This skips over planning considerations like choice of site location, antenna diagrams, appropriate transmit power or regulation aspects, as we assume these topics are were already taken care of. With the outline set, we will now go through a list of steps that will lead to a functional and reliable broadcast setup.

9.2 Setup steps

Choice of computer First, a suitable computer has to be chosen for running the tools. As this needs to be as reliable as possible, it is preferable to chose a server designed for reliability. Because we are driving a USRP over USB, it is essential to have a good USB controller on the motherboard. Sadly, there is no easy way to verify this besides actually testing it. See section 6.2.1 for more details. Redundant power supplies and the ability to use two hard drives in a RAID are also useful to have.

Operating System The operating system needs to be installed next. All the dependencies for the tools need to be installed, as well as the additional tools needed for the system: supervisord for process supervision, Munin for monitoring, logging and logrotate configuration, proper NTP setup, configuring real-time scheduling and additional topics discussed in section 8. If you need to prepare remote encoders, this has to be done for all the machines you will use.

Installation of the Tools The tools themselves need to be installed, according to instructions in the `README` and `INSTALL` files of the repositories. Then you need to prepare the configuration for the tools, using the examples and this guide. For every programme, create a folder for the slideshow images and gather the slides, and prepare the interfaces for DLS text. Write the supervisord configuration files that are used to launch all ODR-AudioEnc and ODR-PadEnc processes. Write the multiplex configuration, with all the entries for your programmes and an appropriate supervisor configuration. Setup ODR-DabMux munin monitoring as desired.

Verification of Multiplexer At this point you should already be able to launch the configured tools and verify that they start, connect properly and stay running. You can simulate process failures by killing any of the tools; the supervisor should restart it. You could use etisnoop and other ETI analysis tools to verify that your multiplex is valid, or listen in on the programmes by using netcat piped into dablin.[19] Also check that logging and munin monitoring works.

ODR-DabMod Configuration Next configure ODR-DabMod. For improved spectrum performance, configure it with FIR filter enabled, OFDM symbol windowing enabled (if available), with the frequency given in your license, and start with a digital gain of 0.5 and a TX gain of 60dB. If you have a disciplined 10MHz clock reference or a GPSDO, configure accordingly. This will ensure the modulator runs at the same rate as the rest of the transmission chain whose rate is in turn related to NTP.

Generate Exciter Signal Prepare the ODR-DabMod supervisor configuration. Connect the USRP to a spectrum analyser and launch the modulator. Before connecting the power amplifier, make sure to have a good spectrum at the USRP TX port, and use the remote control interface to modify TX gain and digital gain to see what RF power you can generate given the spectrum mask you want to achieve. Placing a DAB receiver next to your setup, you should also be able to verify that reception is possible, audio is present and that the DLS and slides are properly transmitted. Ideally, let this setup run for a couple of days and check for the absence of underruns. This step proves you can generate a valid exciter signal with good characteristics.

[19] `nc MUX 9200 | dablin_gtk` should work, assuming your ODR-DabMux serves ETI over TCP on port 9200. Replace `MUX` by the multiplexer IP address. See `http://github.com/Opendigitalradio/dablin` for information about dablin.

Connect Power Amplifier After stopping the transmission, connect the USRP to a Band III filter[20] to suppress harmonics, connect to the power amplifier. Using suitable attenuation, connect the amplifier output to your spectrum analyser. Configure a low TX gain of 30dB and a digital gain of 0.5, and power up. Again do some experimentation with both TX gain and digital gain to find the optimal settings, now with the amplifier. Let the amplifier warm up to operational temperature before reaching conclusions. If your amplifier has a monitoring interface, make sure it works and integrate it into your setup.

Tune RF Settings Also experiment with settings that have an impact on the spectrum performance: OFDM Symbol Windowing and the FIR Filter settings. If you have measurement equipment that can demodulate and measure MER, make sure it is within bounds, ideally better than 25dB. You can trade-off MER against peak-to-average power ratio using the `normalise_variance` and CFR settings.

Justify this value.

Insert Mask Filter The final measurements before installation needs to be done with the mask filter connected after the power amplifier, to ensure that the spectrum mask is satisfied. The mask filter also needs some warm up time. It is also advisable to use a vector-network analyser to check the mask filter's S11 and S12 parameters.

Final Setup Finally, set up the system at your transmission site, power up to nominal power, do coverage measurements and compare them to the simulations. By now, you will also have to deploy all the remote encoders at the programme originators' studios.

Maintenance and Monitoring Running a multiplex is unlikely to be a "set up and ignore" project. Usually you will have to do many kinds of interventions, because of changes in your multiplex composition, requests from programmes you are carrying (e.g. change of web-stream URL, replacement of slides), or notify them in case of audio issues; equipment failure due to weather conditions requiring replacement; regular system updates that should made with low impact; changes of configuration related to announcements or service linking; modification of RF settings due to aging of RF components or due to seasonal thermal changes. All these are inherent to operating a broadcast infrastructure and create maintenance work that needs to be planned for.

[20] For example, a filter with similar characteristics as the Mini-Circuits RBP-220W+.

10 Single-Frequency Networks

10.1 Requirements

The DAB standard has been designed to enable the creation of transmission networks where several transmitters share the same frequency, and send the same signal synchronously. Such networks are called "Single-Frequency Networks". Each transmitter needs to be fed the same multiplex stream, which must include timing information required for synchronisation. This timing information implies that a time reference must be installed at each transmitter.

The requirements for a SFN can therefore be summarised in three points:

- The signal must be *identical* for each transmitter. This requires a common multiplexers, and a distribution network that carries the ETI to all modulators.
- All transmitters must transmit on the *same frequency*. The modulators require a frequency reference.
- The signal must be transmitted at the *same time*, which requires a time reference at each site. It also implies that the ETI stream must contain timestamps.

The figure 4 shows a SFN setup with two transmitters.

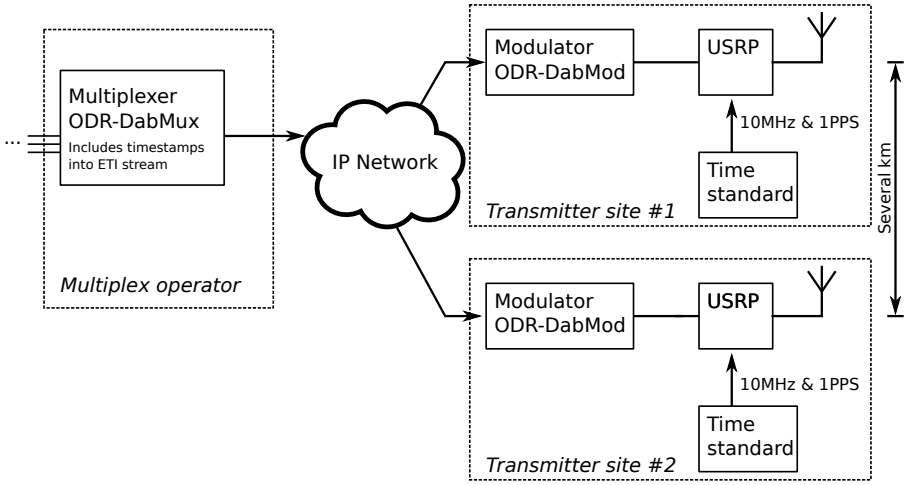


Figure 4: This outline for a SFN shows two transmission sites.

Explain requirements on system time, NTP

10.2 Multiplexer Configuration

On the ODR-DabMux configuration, there are not many options that are specific to an SFN setup. Most importantly, the timestamp feature must be enabled using the "tist" option in the "general" section.

Furthermore, it is recommended to use the EDI transport between the multiplexer and the modulators, which can be enabled in the "outputs" section. Care

has to be taken to have an output that slows ODR-DabMux down to nominal rate. The EDI output alone does not enforce this. The following listing shows the relevant options we just covered.

```
general {
    tist true
    ...
}

...

outputs {
    ; Accept connections on all interfaces, on port
       13000
    edi {
        destinations {
            tcp {
                protocol tcp
                listenport 13000
            }
        }
        enable_pft  false
    }
    ; This throttles muxing down to nominal rate
    throttle "simul://"
}
```

10.3 Modulator Configuration

Since the modulator has to ensure that the three SFN requirements are satisfied, its configuration is more complex.

We will assume, in this explanation, that one of the following USRP devices is used: USRP2, USRP B100, USRP B200. Other devices also support the necessary time and frequency synchronisation, but they have not been well tested. These USRP devices can accept different sources for the reference clock:

- The default "internal" source uses the non-disciplined clock generator inside the USRP. It is not suitable for SFN.
- The "external" source corresponds to the SMA connector on the USRP. A 10MHz signal from an external source must be connected to it.
- The optional GPSDO that can be mounted inside the USRP, and is selected as source with the "gpsdo" setting.

For the time reference, the "pps_source" option is used. Possible values are "none", "external" and "gpsdo", with analogous meaning as for the reference clock.

In case the USRP is connected to external references, the relevant configuration would be as follows:

```
[uhdoutput]
refclk_source=external
pps_source=external
```

These settings alone do not tell the modulator to enable synchronisation of the transmission, they only select how the USRP is configured. To enable timestamp decoding and the frame synchronisation logic in ODR-DabMod, the following settings must also be set:

```
[delaymanagement]
synchronous=1

; The constant offset to be added to the TIST, in seconds
offset=2.0
```

The "offset" setting deserves some further explanations. The ETI data stream contains TIST information, from which a time-stamp for each ETI frame can be derived. Each ETI frame (24 msinterval) is therefore associated with a precise point in time that defines the time of transmission of the corresponding transmission frame.[21] The TIST information is set to current time at ETI frame generation, and does not take in account the propagation delay across the distribution network. Therefore, we need to add an offset, called δ, to the TIST to define transmission time.

$$t_{transmission} = t_{TIST} + \delta$$

If this offset is set to a higher value, there will be a bigger delay (measured in absolute time) between the point in time a frame is multiplexed and the point in time the frame is transmitted. More frames therefore will be buffered in the ODR-DabMod input, increasing robustness against network latency fluctuations.

The offset already has two functions: it compensates for network delay and allows a trade-off between delay and robustness. But it also serves a third purpose: When doing coverage planning for an SFN, it is necessary to be able to control the relative delay between transmitters in the order of milliseconds. This tuning of relative delay is included in the "offset" setting. We can therefore rewrite the above equation as:

$$t_{transmission} = t_{TIST} + \delta_{network} + \delta_{planning}$$

$$\delta_{offset} = \delta_{network} + \delta_{planning}$$

When using the ZeroMQ input, the `max_frames_queued` setting must be large enough to contain enough ETI frames to accommodate the offset.

10.4 Using ODR LEA-M8F GPSDO board

The ODR GPSDO board integrating a u-blox LEA-M8F module can be used as time and frequency reference for the USRP B200. The board design is available

TODO: Add Picture

[21] It is slightly more complex, because one transmission frame is composed of several ETI frames in some transmission modes, but the principle stays the same. It suffices for this explanation that we can derive the transmission time from the TIST information.

on the Opendigitalradio website, with a bill of materials describing how to source the components. The PCB itself can be manufactured in any PCB fab.

The module includes the correct pin header so that it can be mounted directly onto the USRP B200, but also includes footprints for SMA connectors for other usages. Communication between the PC and the GPS is possible through USB or over UART through the B200.

The u-blox LEA-M8F module is a GPS disciplined TCXO module, with a one-pulse per second and a reference clock output at a frequency of 30.72 MHz. This is different than what the normal USRP firmware expects.

Because the UART communication protocol and the reference clock frequency are different than for the GPSDO units Ettus supports, a modified version of UHD is necessary. This version includes new UHD sensors, used by ODR-DabMod to verify that the GPSDO is locked properly, and different configuration settings for the clock management PLL inside the USRP, making the USRP compatible to the 30.72MHz reference clock frequency.

The modified UHD version is available on the ODR GitHub[22] and is used in place of Ettus' UHD.

ODR-DabMod can be configured as follows:

```
[uhdoutput]
refclk_source=gpsdo
pps_source=gpsdo
behaviour_refclk_lock_lost=crash
max_gps_holdover_time=600
```

10.5 Using Ettus GPSDO

When using the GPSDO from Ettus, which is a Jackson Labs Firefly module, no special UHD version needs to be installed.

The configuration is:

```
[uhdoutput]
refclk_source=gpsdo-ettus
pps_source=gpsdo
behaviour_refclk_lock_lost=crash
max_gps_holdover_time=600
```

[22] http://www.github.com/Opendigitalradio/uhd.git

11 Supervision of Transmitted Ensembles

11.1 Introduction

We have previously seen a way to monitor the transmission infrastructure (or at least some of its essential parts) in chapter 8.6 about munin monitoring. These monitoring elements give an indication about ODR-DabMux and ODR-DabMod health from within the infrastructure itself, and may not be able to inform you about some issues happening outside of the software tools.

Monitoring the transmitted signal at a remote site within the coverage area can complement the internal monitoring and broaden the supervision coverage. In the end, we can only consider the broadcast system being in an operational state if a receiver can play all programmes, and being able to verify this automatically by placing a receiver in the field is the only way to ensure this.

In this chapter, we will see one way to achieve this.

11.2 Welle.io Software-Defined Receiver

The welle.io[23] project offers an SDR DAB receiver that can run both with a graphical user interface for ease of use, and as a command-line tool that can be used for automated systems. The command-line tool called `welle-cli` presents an HTTP API that makes ensemble parameters and audio content available to third party tools. Until this tool becomes part of a released version, checkout the `next` branch and compile it using CMake, as described in the readme. Execute it directly from the build folder, so that it also can access the `index.html` file.

welle-cli can present the ensemble data in more than one way, but we will focus on the HTTP interface. It is enabled with the `-w 7979` option, which will run the HTTP server on port 7979. Select the channel to receive, e.g. 10A, with `-c 10A`. When you point your browser to `http://localhost:7979`, you will see a simple web-page that shows a subset of the data available through the API. When pressing a play button, `welle-cli` will start decoding the selected sub-channel and stream it to the browser as an MP3 stream.[24]

Several options are available for decoding the programmes: use `-D` to decode all audio and PAD simultaneously. This requires a powerful PC. Use `-C` to decode the audio in a carousel, i.e. one-by-one. When using `-CP` the decoder remains up to 80s on a programme, but switches programmes once a slide was correctly decoded. Compared to `-C` alone, this improves the likelihood of decoding slideshow at the expense of a lower audio level update rate.

While this web-page already has some utility as-is, it mainly serves as an illustration of what can be done with the API, where the real value of `welle-cli` resides. The API is, for the moment, quite simple:

- `/mux.json` contains most information in JSON format. From this JSON you can extract the list of services, the ensemble parameters, TII decoding and other information.

[23] Project page: `http://welle.io`, sources on: `https://github.com/AlbrechtL/welle.io`

[24] MP3 is used because it is the only compressed audio format that is supported in all browsers. The AAC or MP2 audio inside the ensemble is re-encoded by `welle-cli` using the LAME encoding library.

- `/mp3/SID` will give you a live mp3 stream of the primary component of the given service id.
- `/fic` will send a data stream containing the FIC. This can be saved to file and analysed offline with other tools, among which etisnoop (using its `-I` option). etisnoop is also able to do live analysis of the FIC, e.g. with `curl -s http://localhost:7979/fic|etisnoop -I /dev/stdin` whose YAML output can in turn be processed further.
- `/channel` will return the currently tuned channel on receiving a GET request, and set the channel and restart the receiver on receiving a POST.

Other HTTP URLs give back information that needs to be processed further. See the script code inside `index.html` to understand how to work with it.

- `/spectrum` will send a sequence of float values that show the spectrum power density of the signal.
- `/nullspectrum` will send a sequence of float values that show the spectrum power density of the NULL symbol, where the TII carriers are visible.
- `/constellation` will send a sequence of complex float I/Q values corresponding to the demodulated constellation points.
- `/impulseresponse` will send a sequence of float values that represent the measured channel impulse response.

An example integration into a monitoring system is given in the `welle-cli-munin.py` script. This munin plugin fetches the `mux.json` and converts the audio levels in a format that munin can graph. In this way, an entire ensemble can be monitored at once.

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.[25]

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 Additional EDI TAGs used

ODR defined and uses two additional EDI TAGs, whose content is described here.

ODR-AudioEnc inserts audio level metadata into the "ODRa" TAG. The TAG item is in the following format:

```
  TAG Name="ODRa" [4 bytes]
  Length=4 [4 bytes]
  Left Audio Level [signed 16-bit integer]
  Right Audio Level [signed 16-bit integer]
```

The second EDI TAG "ODRv" contains version and uptime information for the EDI source.

```
  TAG Name="ODRv" [4 bytes]
  Length=N+4 [4 bytes]
  Version [String of N bytes, UTF-8 encoded, not zero
     terminated]
  Uptime [unsigned 32-bit integer representing number
     of seconds since program start]
```

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

C Bibliography

References

[1] ETSI. *ETS 300 799, Digital Audio Broadcasting (DAB); Distribution interfaces; Ensemble Transport Interface (ETI)*, September 1997.

[2] 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`.

[3] ETSI. *TR 101 496-2, Guidelines and rules for implementation and operation; Part 2: System features*, November 2000. V1.1.1.

[4] ETSI. *TS 102 693, Digital Audio Broadcasting (DAB); Encapsulation of DAB Interfaces (EDI)*, November 2009. V1.1.2.

[5] ETSI. *TS 103 176, Digital Audio Broadcasting (DAB); Rules of implementation; Service information features*, July 2013. V1.1.2.

[6] ETSI. *EN 300 401, Digital Audio Broadcasting (DAB) to mobile, portable and fixed receivers*, October 2016. V2.1.1.

[7] ETSI. *TS 101 768, Digital Audio Broadcasting (DAB); Registered Tables*, August 2017. V2.2.1.

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