Single-Frequency Network Support for the CRC mmbTools Open-Source Software-Defined DAB$^+$ Transmitter

Master project

Matthias P. Brändli
matthias.braendli@epfl.ch
EPFL IC LAP – 2012

Supervised by:
René Beuchat – EPFL IC LAP
http://lap.epfl.ch

Mathias Coinchon – European Broadcasting Union
http://tech.ebu.ch

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

The world of audio broadcasting is slowly moving away from the well-established analog FM standard to a new digital standard called "Digital Audio Broadcasting" (DAB), in the same way television is doing it. This digital switchover brings better signal quality to the listener and is more robust, especially in the case of mobile reception. For broadcasters, this new technology also brings improvements in regard to coverage planning and energy consumption, thanks to the ability to create single-frequency networks, using which a large coverage is achieved by running several transmitters in unison, simultaneously transmitting the same signal on the same frequency.

However, these improvements come at a certain cost: while in analog broadcasting it is possible for a small community radio to set up a small FM transmitter for a modest fee, it is much more complex in digital transmission. For one, it is not possible anymore to build and run a transmitter cheaply, partly because digital technology is more complex, but also because it is more recent. Secondly, the commercial solutions that exist on the market are too expensive for small broadcasters, who do not have the means to invest into a new technology, especially if the old one is still working fine. But DAB is especially interesting for small broadcasters, which often lose the fight for FM spectrum against larger entities, because it opens up much more spectrum, allowing a more diverse and larger set of radio programmes to be broadcast in a given region.

The Communications Research Centre Canada has developed a set of tools that may very well change this situation. Based on the software-defined radio principle, their open-source "Mobile Multimedia Broadcasting Tools" (mmbTools) can be used to transform any normal Linux computer into a DAB transmitter. These tools, when combined with a hardware interface that allows the computer to output a radio-frequency signal, compose a complete but affordable DAB transmitter. This solution has enabled several trial broadcasts in different countries.

While this solution is very interesting for experimentation and broadcasts with a small coverage, it does not support the creation of single-frequency networks, which is necessary for larger coverages. In this project, I have added this feature to the mmbTools, so as to open up the possibility to use these tools for larger projects.

2 The Digital Audio Broadcasting Standard

2.1 History and design

Digital Audio Broadcasting, or DAB for short, is a technology developed in the late eighties with the intent to replace analog FM radio by a more robust and feature-rich digital broadcasting technology. It is standardised by ETSI, which gives free access to the documents describing the whole transmission and reception chain [6]. It has already been deployed in several countries, notably in the United Kingdom, and in several European countries.

The radio programmes are encoded with a digital audio codec, and are transmitted using a digital modulation technique. The original DAB standard has been upgraded in 2007 with a new and more efficient audio encoder: In the original standard, MPEG Audio Layer 2 (MP2) is used to encode the audio programme, whereas DAB$^+$ uses MPEG-4 High Efficiency Advanced Audio Coding (HE-AAC). The ETSI standard TS 102 563 [12] describes the changes between DAB and DAB$^+$. The DAB$^+$ extension integrates well in the original standard, and many components are identical between DAB and DAB$^+$. Newer DAB$^+$ receivers must also be able to receive the older DAB standard, but old receivers cannot since they lack the HE-AAC decoder.

Contrary to analogue FM transmission, several radio programmes are put together in one *Ensemble*, which is then transmitted from one or several locations. Each ensemble contains up to 18 programmes–the number depending on the desired audio quality–which are transmitted at once.

It is also possible to transmit data, pictures and text information, which can for instance be used to show programme-related content on a receiver. For instance, a radio programme might show the CD cover of the currently playing song, or a camera picture of traffic on congested highway segments. Since these are auxiliary channels to the radio programme, the bit-rate of such picture channels is quite low, and the picture only changes every few seconds. This feature is called a *slideshow*.

The same modulation chain can also be used to transmit video, that is described in a related standard called *Digital Multimedia Broadcasting*, formalised in two ETSI standards [7, 8]. In that case, a MPEG transport stream is transmitted as a data service over DAB.

2.2 Modulation

The modulation scheme used in DAB has been chosen to cope with the adverse channel conditions that arise for mobile reception in urban environments, as illustrated in figure 1. Due to the environment that creates reflections, these channels are characterised by multipath propagation with long echoes. Several out of phase contributions resulting from those reflections will reach the receiver with different delays, and will interfere with each other. In some situations, reflections with a length of up to 50 km are possible; this corresponds to a delay of $\frac{50000}{c} \approx 165\,\mu s$. This interference pattern is frequency selective, with some parts of the spectrum that are subjected to constructive interference, while other parts interfere destructively. This results in a non-flat spectrum at the receiver.

For a mobile receiver, this effect does not only vary in frequency, but also in time. Furthermore, the receiver will experience a Doppler shift for each contribution–each of which arriving at the receiver from a different direction–leading to what is called a Doppler spectrum.

These channel conditions make it impossible to use a single-carrier modulation (e.g. a single QPSK modulation) without advanced techniques, because for a 1 Mbps bandwidth,

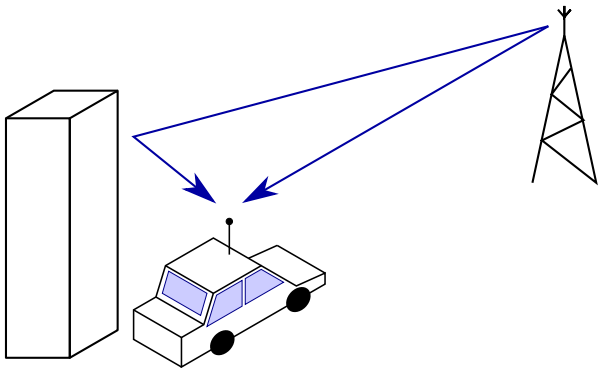

Figure 1: Reflections are extremely common in urban environments.

the symbol duration will be shorter than the reflections, leading to strong inter-symbol interference. For the QPSK example, which carries two bits per symbol using this bandwidth, the symbol duration $T_s = \frac{2\,bpsym}{1\,Mbps} = 2\,\mu s$. In that case, even a reflection of one kilometer–each kilometer corresponds to a delay of 3.3 µs–will lead to inter-symbol interference that is very difficult to cope with. Several techniques can be used in this situation, and the technique used in DAB to overcome these constraints is multi-carrier modulation.

In the multi-carrier approach, the total data-rate is split among several carriers, which therefore carry only a fraction of the total data-rate. If there are K carriers, the data-rate can be split in K, and the symbol duration will be K times larger. With enough carriers, the symbol duration can be made longer that the longest expected echoes.

The most frequently used technique used to implement a multi-carrier approach is Orthogonal Frequency-Division Multiplexing (OFDM), which uses the orthogonality of the Fourier transform to create the multiple channels. Thanks to the FFT algorithm for the calculation of the Fourier transform, this approach is very efficient in terms of computational complexity.

OFDM is a modulation technique designed to be resistant against multipath propagation channels. For a wide signal, like DAB which uses 1.5 MHz of spectrum, self-interference from multipath reflections results in a non-flat fading, i.e. the attenuation of the signal will not be the same for all frequencies. This leads to the aforementioned problems if a single carrier is used. However, for DAB, between 256 and 2048 OFDM carriers are used[1], each of which is then modulated with data. These sub-carriers will be narrow, and each will therefore only experience flat fading. Thus, instead of being unable to demodulate the whole signal, it will be possible to decode most sub-carriers, except those that as subjected to the strongest fading. This missing data can be compensated using forward error correction.

Furthermore, the total data throughput is distributed over all sub-carriers. Each sub-carrier transmits only a low throughput, and therefore has a low symbol rate. This makes it possible to insert a guard interval between symbols, and each sub-carrier will be much less affected by inter-symbol interference (ISI). The guard interval in this case is a cyclic prefix, and its length is related to the longest reflection that can accepted without degradation. The time for a radio signal to propagate over 1 km is about 3.3 µs, from which the required guard interval is calculated for a given maximum reflection length that the system must be able to cope with. The guard interval must be larger than

[1]The value depends on the transmission mode: TM 1: 2048, TM 2: 512, TM 3: 256, TM 4: 1024.

the delay of the latest reflection. In DAB, the guard interval T_g is 24.6% of the symbol duration T_s, which depends on the transmission mode:

Mode	T_s [µs]	T_g [µs]	max reflection [km]
TM 1	1000	246.00	75
TM 2	250	61.50	19
TM 3	125	30.75	9
TM 4	500	123.00	37

Table 1: The maximum reflection length depends on the mode-specific guard interval length.

The maximal length of the reflections can be calculated using the formula $c \cdot T_g$, where c is the speed of light.

From a time perspective, a DAB transmission looks like a sequence of transmission frames that are composed of the null symbol, the phase reference symbol and the data symbols. Their respective lengths depend on the transmission mode, and are given in *elementary periods*. The elementary period is defined as $\frac{1}{2'048'000}$ s. Section 15.2 of EN 300 401 [10] gives the durations of those elements. On an oscilloscope, it is difficult to see the difference between phase reference symbol and the data symbols, but the null symbol is easily recognisable because the transmitter is off during that symbol. This symbol is used for receiver synchronisation and it is very important to respect it's timing, otherwise reception will break down. Some receivers are not even able to resynchronise without user intervention. On the oscilloscope, this can be seen as shown on figure 2 and figure 3, where two null symbols can be seen.

Thanks to this null symbol, it is easy to measure the duration of a transmission frame using an oscilloscope, which is given here :

- TM 1: $T_f = 96\,\text{ms}$
- TM 2: $T_f = 24\,\text{ms}$
- TM 3: $T_f = 24\,\text{ms}$
- TM 4: $T_f = 48\,\text{ms}$

2.3 Ensemble Transport Interface

In the DAB specification, there is also a standard defining the data interface between ensemble multiplexers and modulators. This interface is called the *Ensemble Transport Interface*, and is defined in ETS 300 799 [5].

Since the ensemble can possibly be transported over several types of communication channels, the ETI uses a layered definition, where first a logical definition is given that can then be adapted to several physical layers. The logical interface is called ETI(LI). It contains all information required to modulate a DAB signal, such that two transmitters that receive the same ETI(LI) will output identical modulated RF waveforms. The information contained in one ETI(LI) frame cannot be changed when the interface is adapted to a physical layer.

The ETI(LI) frames do not have a constant length, and their nominal rate is one ETI frame each 24 ms. In transmission mode 2 and 3, one transmission frame can be

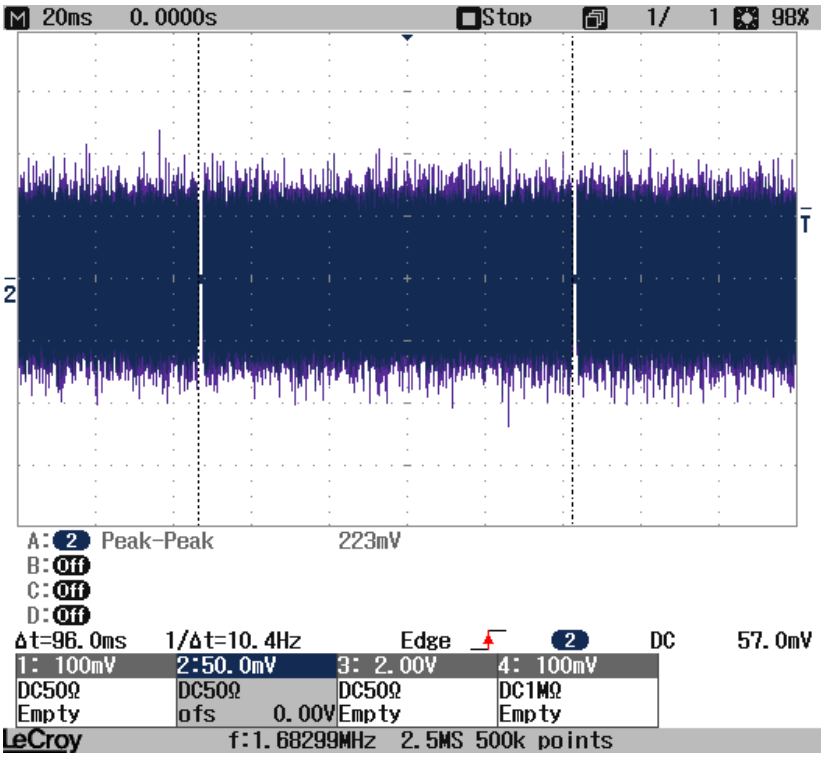

Figure 2: In TM 1, each transmission frame is 96 ms long.

modulated from the data contained in one ETI frame. In TM 1, four ETI frames for one transmission frames are taken together, whereas TM 4 combines two.

There are too many different fields in each frame to enumerate them all here, but some are more important for this project, and will be explained in detail. They are the following:

FCT The *frame counter* is a counter, that is in the range 0 to 249. This counter is incremented by one for each ETI frame, and therefore has a periodicity of 6 s.

MNSC One part of the time synchronisation data is transmitted in the *Multiplex Network Signalling Channel*. This channel contains additional information that is parallel to the programme data. It can contain either frame synchronous signalling information (FSS), or frame asynchronous signalling (ASS). In the case of frame synchronous signalling, the data transmitted in the MNSC is related to a particular frame. The MNSC is a low bandwidth channel, that uses only two bytes in each ETI frame. A signalling group consists of eight bytes, and is transmitted in the MNSC fields of four consecutive ETI frames. Time information is only one kind of data that can be transmitted as one signalling group, but will be the only information used in this project. Its encoding is defined in annex A of ETS 300 799 [5], and represents time with a one-second accuracy. It is frame synchronous, and the time information applies to the ETI frame containing the first two bytes of the signalling group.

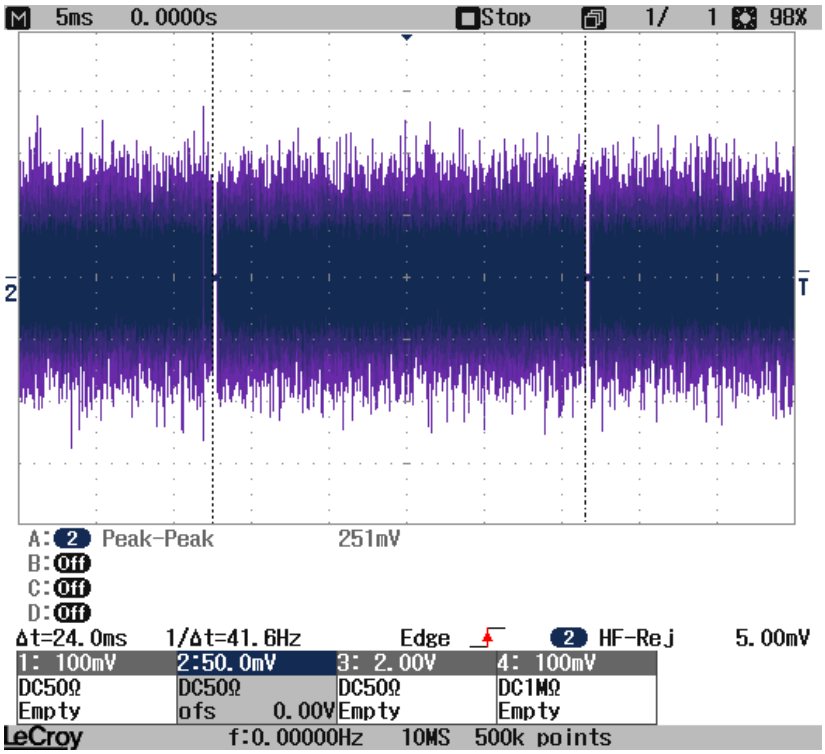

Figure 3: In TM 2, each transmission frame is 24 ms long.

FP The *frame phase* is a modulo-8 counter, that is used for two important aspects: the alignment of frames in TM 1 and TM 4, and in the transmission of the MNSC. In TM 2 and TM 3, one ETI frame is modulated into exactly one transmission frame, consisting of the null symbol, the phase reference symbol and the data symbols. However, in TM 1, four ETI frames are taken together to create one transmission frame. In TM 4, two ETI frames create one transmission frame. The grouping of the ETI frames is defined by the frame phase: in TM 1, a group of frames is composed either of four consecutive frames with FP = 0, 1, 2, 3 or with FP = 4, 5, 6, 7. In TM 4, it is analogous for FP values of 0, 1; 1, 2; 3, 4; 5, 6 and 7, 8. For the MNSC, the frame phase is used to define which bytes of the signalling group are contained in a specific ETI frame.

TIST This *Timestamp* field contains a value between 0 and 0.999 999 939 seconds, in 61 ns resolution, that specifies the time at which a frame must be transmitted. Combined with the time information transmitted in the MNSC, it defines the absolute time when a transmission frame must be sent, up to a constant offset. This field is also called the "one pulse per second" (1PPS) offset, because it can be understood as the offset between the rising edge of the 1PPS signal and the time when the frame has to be transmitted.

The ETI(LI) cannot be directly used to save ensemble data to a file[2] or to send it over some channel to another device, because there is no mechanism to synchronise frames. The layered approach defines an ETI network independent layer called ETI(NI), that adds synchronisation data and padding to the ETI(LI). ETI(NI) frames contain exactly

[2]Except maybe if one file per frame is used, which is by no means practical.

6144 bytes per frame, which gives a bitrate of 2048 kbps. This format is used in this project.

2.4 Single-Frequency Networks

From the point of view of the receiver, it is impossible to distinguish a reflection from a second transmitter when both transmitters send exactly the same signal. Figure 4 gives a simple illustration of this. The nice side-effect of reflection tolerance is therefore that it is possible to create a network of transmitters that can reuse the same frequency, have overlapping coverage, and the receivers will be able to take advantage of the multiple contributions.

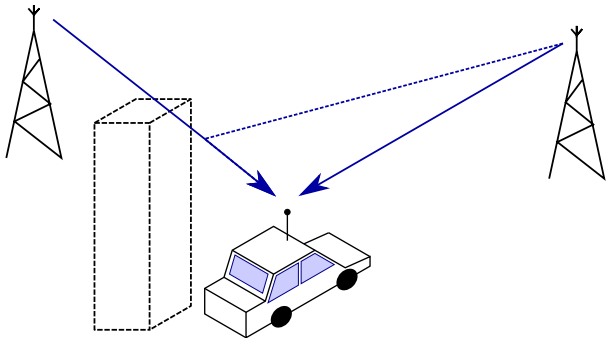

Figure 4: The reflection from the building has been replaced by another transmitter. There is no difference from the point of view of the receiver.

One first advantage of these networks, especially when considering mobile receivers, is that there is no need to switch frequencies at coverage boundaries, as it is the case for analog FM reception. The mobile receiver can stay tuned to the same frequency while moving through a region much larger than the coverage of a single transmitter and does not have to compare the quality of the transmission on different frequencies. Furthermore, the absence of switchover will create a better listener experience, because there will be no audible effect.

There are also less obvious advantages of single-frequency networks. For one, coverage planning is somewhat simplified, because it is not necessary to find free frequencies if the service quality has to be improved in an existing network. In FM broadcasting, frequency channels are a very scarce resource because a given frequency can only be reused at another transmitter if the coverage zones do not interfere with each other. The ability to create single-frequency networks in DAB makes a better use of this spectrum resource. Additionally, the fact that the receiver can combine different incoming signals leads to what is called *network gain*. The result of this combination is that it is possible to receive and decode correctly even in places where the individual incoming signals have a too low signal-to-noise ratio to be decoded independently. Compared to a multi-frequency network, where the receiver must switch from one transmitter to another, less power is necessary for the same coverage. This is illustrated in Figure 5, where (1) represents the network gain, (2) the minimum signal strength required for reception and (3) the transmit power of the transmitter. Thanks to the network gain, it is possible to lower transmission power while still guaranteeing enough signal strength, this would be represented as a downward translation of the dashed line.

These advantages come at the cost of higher transmitter complexity. Naturally, the

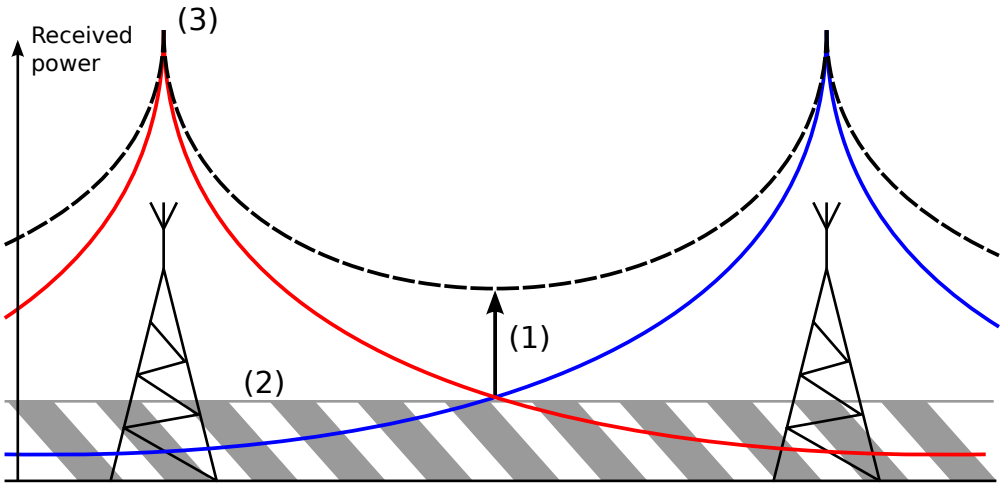

Figure 5: The red and blue lines represent the signal strength for two transmitters using different frequencies. The dashed line corresponds to signal strength in a SFN with identical transmitter power.

considerations on the relative delays that apply to reflections also apply to the incoming transmissions from different transmitters. This implies that for a good single-frequency network, the transmitters have to be perfectly synchronised on two key aspects:

1. Each frame transmission must start at the same absolute time for all transmitters. If there is a delay between the two transmitters, the whole system becomes less robust to multi-path, or breaks down if the delay is too large.

2. The transmission frequencies must be identical, so that the sub-carriers align properly. Otherwise, resistance against Doppler shift will be degraded.

Some delay and some frequency deviation can be tolerated, which depends on the modulation parameters used. The maximum delay that can be accepted is the length of the guard interval T_g, that is shown page 5. Since one kilometer corresponds to 3.3 µs propagation time, each millisecond additional delay worsens the possible coverage boundary by up to 300 m. The less relative delay there is between transmitters, the better the network will be, but in order to achieve good performance, a delay shorter than 1 µs should be guaranteed. The carrier frequency should be kept within 1% of the carrier spacing according to section 7.3.2 of Hoeg and Lauterbach [14].

To achieve the required frequency stability and precision, several approaches can be considered. Essentially, an identical clock must be present at each transmission site. A precise frequency standard alone (e.g. a rubidium or a cesium standard) is not sufficient, because it cannot be used to define a precise absolute time, unless it is used to drive a calibrated clock. Another approach is to use a system that receives synchronous time information from a single source. For the latter, DCF77 and GPS are possible solutions, with GPS being preferable because a GPS receiver has an internal clock that is aligned to GPS time and many receivers have a one pulse per second (1PPS) signal that is aligned to the GPS second change. In comparison, if DCF77 is used, an additional compensation must be applied at the receiver, depending on its location to correct for the signal propagation delay.

The transmitter must then use this timing information to align its carrier frequency, and to apply a controlled delay on the transmissions.

2.5 Existing transmission systems

Transmission of digital audio broadcasting is usually done with professional equipment that uses dedicated integrated circuits to process the data (e.g. for encoding, multiplexing, modulation, etc.). The cost of such devices is driven up by the fact that they are very specific to their application and are not mass-produced, because there is no large demand for such equipment. Also, broadcasters also want to guarantee the reliability of their transmission, preferring transmission equipment that offers redundancy and comes with quality guarantees. According to research done by OpenDigitalRadio,[3] such equipment can easily cost over 20'000 EUR for one transmitter.

The software-defined radio approach, in which the radio-frequency samples are not generated by dedicated hardware but using software is becoming more and more prevalent in this domain. This is true with commercial equipment[4] but also with more experimental systems. The mmbTools are of these experimental toolsets. Created by the Communications Research Centre Canada, the mmbTools are a set of PC-based software that build up software-defined radio transmission chain. Being open-source, I have been able to base my project on it. It is described in the next section.

[3] See slide 7 of the presentation `http://www.opendigitalradio.org/files/unikom_presentation_pub.pdf`.

[4] Unique Broadcast Systems Ltd. (`http://www.uniquesys.com/`) for instance sells a transmitter that can modulate DVB-T, DAB and many other modulations using different on-board software.

3 The CRC Open-Source DAB+ Transmitter

3.1 Overview of the CRC mmbTools transmission chain

The open-source transmitter for DAB and DAB+ that is part of the CRC mmbTools[5] has been developed by the Communications Research Centre Canada. It is built upon the software-defined radio principle, where the RF signal is not generated and modulated using dedicated hardware, but is calculated by a general-purpose processor–in this case, a normal PC is used–and then fed to a modulation-agnostic hardware transmitter. The main advantage of this solution, for the specific case of a DAB, is to drastically lower the price for a transmitter by replacing dedicated hardware for which the market is small by general purpose and readily available computers. The open-source nature of this software also enables modifications and adaptation of these tools to changing needs, and render the tools interesting for studying DAB multiplexing and modulation. Compared to a commercial transmission chain built out of dedicated hardware, the software-defined radio approach inherits from all the complexity coming from a general-purpose computer with its operating system. That makes it more difficult to set up, and requires more tinkering and testing to achieve an error-free function.

These tools have been used for several test and demonstration transmissions by the OpenDigitalRadio[6] project, and both the mmbTools and the OpenDigitalRadio project are still evolving.

In this section, I present the versions of the mmbTools on which this project is built. When joined together, the CRC mmbTools compose a complete transmission chain for one transmitter. In order to create a single-frequency network, I have cut the linear chain described below, so that several modulators can run simultaneously and drive more than one transmitter. Furthermore, I have modified and improved these elements to make sure the timing constraints described in section 2.4 are met.

In section 4, I will explain in detail what these modifications consist of.

3.2 CRC mmbTools Transmission chain

We will now go through the whole transmission chain shown in figure 6.

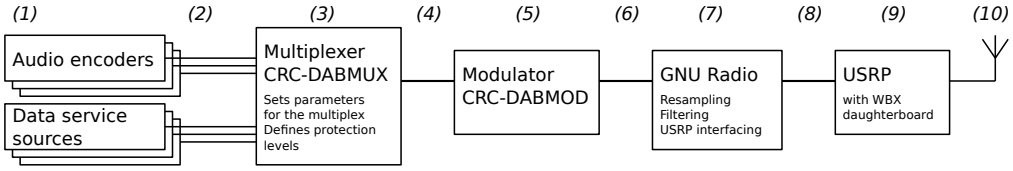

Figure 6: DAB transmission chain used with the CRC mmbTools

Short summary The first section, comprising parts 1 to 7, is entirely in software and is composed of the preparation of the content that is to be transmitted (audio coding), the multiplexing the services (audio and data), the error-correction and the modulation to baseband.

This modulated sampled baseband signal must then be transmitted. The signal must be converted to analog form using a high-speed digital-analog converter, and up-mixed

[5] http://mmbtools.crc.ca/

[6] http://www.opendigitalradio.org

to the desired transmission frequency. The USRP family of devices from Ettus Research is used for this (9). These devices are generic software-defined radio platforms, and consist of all components needed to receive and create RF signals. The hardware is connected either through USB or through Gigabit Ethernet (8) to a computer running GNU Radio (7), which interfaces the CRC mmbTools with the USRP driver.

Programme and data sources (1) Since one ensemble contains several programmes, it is necessary to gather them on one computer to assemble them into a single data stream. Each of those programmes must be properly encoded, either into MP2 or into HE-AAC, with suitable parameters.

The audio source can be a sound card, an audio file or a network stream of audio samples. To encode the source into MP2, the open-source *toolame* encoder can be used. To encode into HE-AAC, CRC-DABPLUS can be used. The latter is not open-source because of patent issues.

Additional data sources (slide-show pictures, text, etc.) can also be defined. CRC has developed a closed-source software that can be used to generate slide-shows.

Contribution (2) of each encoded audio programme to the multiplexer (3) The multiplexer, implemented in CRC-DABMUX, receives the encoded audio programmes from each source. The contributions can be done using a network stream, a fifo or a file.

This allows the radio stations to encode their programme in their own studios, but at the same time allows encoding on the computer on which the multiplexer runs if it is necessary.

Multiplexing (3) The multiplexer, *CRC-DABMUX*, creates the ensemble by combining all input sources. When this program is called, all multiplex parameters (name, programmes, bitrates, etc.) must be specified.

One ensemble contains several services, which can be selected by the user on the receiver. Each service has one or more components, which carry data. For instance, a radio programme called "Radio Foo Bar" which transmits both audio and slide-show, will have one service (which is likely to be called "FOO BAR") with two components. Each component is then mapped to a sub-channel, which has a specific bit-rate and protection. An example structure is given in figure 7.

The output is an *Ensemble Transport Interface* (ETI) stream, which is described below. CRC-DABMUX is written in C++, and has been released under an open-source license.

ETI Stream (4) All data concerning an ensemble is contained in an ETI stream, which can be saved to a file, or transmitted to a modulator. This ETI stream already contains the error-corrected data and is at a constant rate of 2.048 Mbps.

The multiplexer output (4) is an ETI stream, which is standardised in the specification ETS 300 799 [5]. However, the file format containing an ETI stream is not standardised, and CRC-DABMUX supports three formats, with different encapsulations. The supported formats are enumerated in appendix C. This becomes an issue only if the output is saved into a file.

This stream can be transported through a pipe if the multiplexer is on the same computer as the modulator, over IP networks or even over special interfaces.

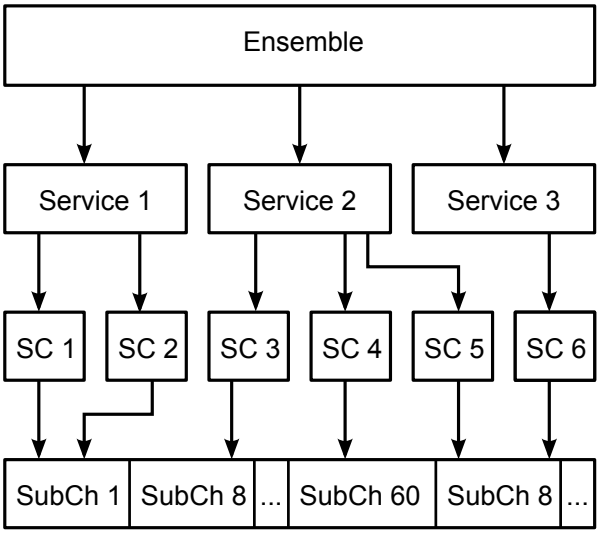

Figure 7: An example composition of an ensemble, showing three services, six service components mapped onto four sub-channels, taken from the CRC-DABMUX documentation [4].

Modulation (5) and baseband signal (6) Modulation consists of creating the RF waveform that will be used to transmit the ensemble. Details about how the DAB modulation is built are given in section 2.2.

The open-source C++ project CRC-DABMOD is used for this task. Its input and output can be files or pipes.

The modulator output (6) is the RF waveform is represented in quadrature (I/Q) baseband ($2 * 16$ bits per sample), with a parametrisable sample rate. The nominal sample rate is 2048 kSps, but can be changed by resampling.

GNU Radio (7) Before this baseband signal gets sent to the hardware platform, it must be filtered to improve the transmission quality, and to respect the spectrum allocation. Furthermore, it is necessary to parametrise the hardware platform. Important settings are:

- The carrier centre frequency;
- The upsampling factor on the FPGA;
- The sample rate used on the USB;
- Which antenna to use, and the analog transmission gain.

These steps are performed by *GNURadio*, a software-defined radio framework written in C++, whose DSP blocks can be instantiated and connected using python code. It also includes a graphical designer called *GNURadio Companion*.

GNU Radio uses the UHD driver to interface to the USRP hardware platform.

Hardware platform (9) and its connection (8) Most hardware platforms from the USRP family have been used for trial transmissions. The first models (USRP1) have been most

used. The newer models (USRP2, USRP B100) are working, but have not been used for 24/7 transmissions yet. Their stability still has to be evaluated.

The USRP1 and the USRP B100 are connected to their host computer using USB 2.0. The USRP2 has a Gigabit Ethernet connection. Both connection types are abstracted by the UHD driver, and can be used in the same way with GNU Radio.

The hardware platform contains a communication interface, an FPGA used to buffer and upsample data, one or two digital-analog converters, each of which is connected to a RF daughterboard. There are several daughterboards available, the most interesting one for DAB transmission in Band III is the WBX board, which has adequate frequency coverage (50 MHz to 2.2 GHz). This daughterboard then mixes the baseband transmission with a local oscillator (LO) so that the transmission is done at the right frequency.

Antenna system (10) The RF output is connected to an antenna system which may include additional filters, amplifiers or to laboratory test equipment.

3.3 Ensemble multiplexing using CRC-DABMUX

As mentioned above in the overview, CRC-DABMUX can be used to create an ETI stream. CRC-DABMUX is a command-line tool that is written in C++. First, its invocation will be presented using a simple example, where two programmes (called "Prog1" and "Prog2") are multiplexed into one ensemble called "Test Ensemble". Both programmes are pre-encoded `.mp2` files that have been prepared in advance using toolame, according to instructions on opendigitalradio.[7] Both `prog1.mp2` and `prog2.mp2` are encoded at 128 kbps.

The output ETI stream is written to standard output, in "raw" format, also called ETI(NI, G.703) in the standard ETS 300 799[5]. The raw format is composed of 6144-byte ETI frames, without any additional headers, with padding bytes according to the standard. CRC-DABMUX also supports other output formats, that are described in appendix C.

The invocation one could use in this case is the following:

```
CRC-DabMux -L "Test Ensemble" \
-A prog1.mp2 -b 128 -i 10 -S -L "Prog1" -C \
-A prog2.mp2 -b 128 -i 3 -S -L "Prog2" -C \
-O fifo:///dev/stdout?type=raw
```

Each line beginning with `-A` creates a new subchannel, with one service using `-S` and one component `-C`. The other options define parameters for the subchannels and the parameters, and are described in detail in the CRC-DABMUX manpage.

In this example, the multiplexer will generate ETI frames as fast as possible, and not at the nominal rate of one frame every 24 ms. This is not an issue if the multiplexer is part of a blocking chain of programs, where the subsequent elements in the chain will back-pressure the multiplexer to the right speed. However, when used with a non-blocking output (e.g. a UDP multicast socket), the `simul` output can be added to the list of outputs as follows. This will rate-limit the ETI stream to one frame every 24 ms.

```
CRC-DabMux -L "Test Ensemble" \
-A prog1.mp2 -b 128 -i 10 -S -L "Prog1" -C \
-A prog2.mp2 -b 128 -i 3 -S -L "Prog2" -C \
```

[7] `http://www.opendigitalradio.org/index.php/DAB/DAB%2B_encoding`

```
-O fifo:///dev/stdout?type=raw \
-O simul://
```

3.4 OFDM modulation using CRC-DABMOD

The multiplexed ensemble, carried in the ensemble transport interface (ETI) stream, has to be modulated in order to be transmitted. The modulator offers much less options than the multiplexer, because most settings are defined in the ETI stream.

The modulators supports three options related to modulation:[8]

- `-g gainMode` selects how the modulator computes the OFDM symbol gain;
- `-r samplingRate` defines the output sample rate;
- `-c clockrate` enables the pre-emphasis for the interpolation filter in the FPGA.

The modulator is written as a flow graph, that is built as shown on figure 8.

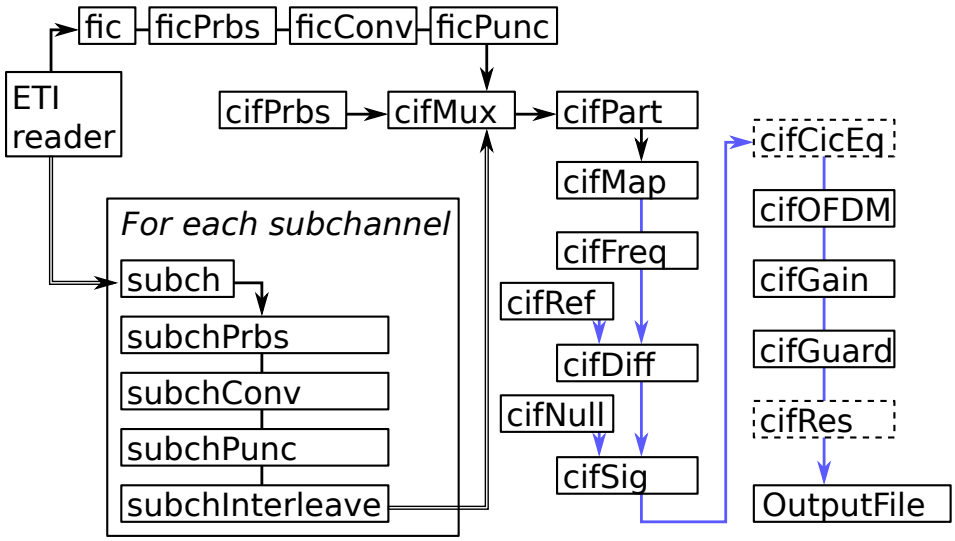

Figure 8: The flowgraph in the CRC-DABMOD modulator (version 0.2.0), black arrows represent data, blue arrows are complex floating-point samples. Dashed boxes are only enabled in some circumstances.

Each of the blocks will now be presented:

- ETI reader: reads one ETI frame into structures;
- fic (Fast information channel): This fic source block is created by the ETI reader, and is used as a flowgraph source for FIC data;
- ficPrbs, subchPrbs and cifPrbs are Pseudo-random bit sequence generators, used energy dispersal;
- ficConv and subchConv are convolutional encoders for error correction;

[8]The options are described on `http://www.opendigitalradio.org/index.php/CRC-DabMod`

- ficPunc and subchPunc apply puncturing after the convolutional encoders;
- subchInterleave takes several data streams from the subchannels, and interleaves them together into one single data stream;
- cifMux: This frame multiplexer writes the PRBS, the FIC and the subchannels into one CIF frame;
- cifPart: The CIF block partitioner takes the CIF frame and creates blocks of a size suitable for modulation. It only supports TM 2;
- cifMap is the QPSK symbol mapper, that maps two bits on each symbol. It outputs complex float I/Q data;
- cifFreq applies a permutation of the subcarriers, so as to disperse data among them;
- cifDiff transforms the QPSK symbols to DQPSK, and adds a phase offset to each symbol defined by cifRef;
- cifNull creates the NULL symbols, which get combined to the DQPSK symbols by the signal multiplexer cifSig;
- If the `-c` command line option was given, cifCicEq is used to equalise the CIC filter in the FPGA;
- Then the QPSK symbols are OFDM-modulated by cifOFDM, that takes the inverse Fourier transform of each frame. cifOFDM outputs I/Q samples.
- The guard interval is inserted by cifGuard at the beginning of each symbol;
- Finally, if the output sampling rate is not 2.048 Msps, cifRes is used to resample the I/Q data to the desired sample rate.

The output data (composed of 32 bits per complex sample, interleaved I/Q at the specified sample rate) is then written to a file or a pipe.

3.5 Conversion to analog using the USRP

In the original transmission chain, the output data from CRC-DABMOD is fed into a GNURadio script (written in python or designed using gnuradio-companion). This script, called a "baseband wave player" serves as interface between the modulator and the USRP, and applies two transformations to the I/Q signal:

- (optional) Filtering: applies a 800 kHz low pass filter to the I/Q signal, so as to have less energy out of band. The transmitted signal will be limited to about 1600 kHz bandwidth. This value has been empirically found. With this filter, less energy will be wasted in the mask-filter between the power amplifier and the antenna system;
- Normalisation: The USRP driver expects the samples to be in the range $[-1.0, 1.0]$, but DABMOD outputs samples in the range $[-32768.0, 32768.0]$. These values are normalised by the "Multiply const" block in the wave player shown in figure 9.

These two transformations can be seen in gnuradio-companion, as shown in the screenshot in figure 9.

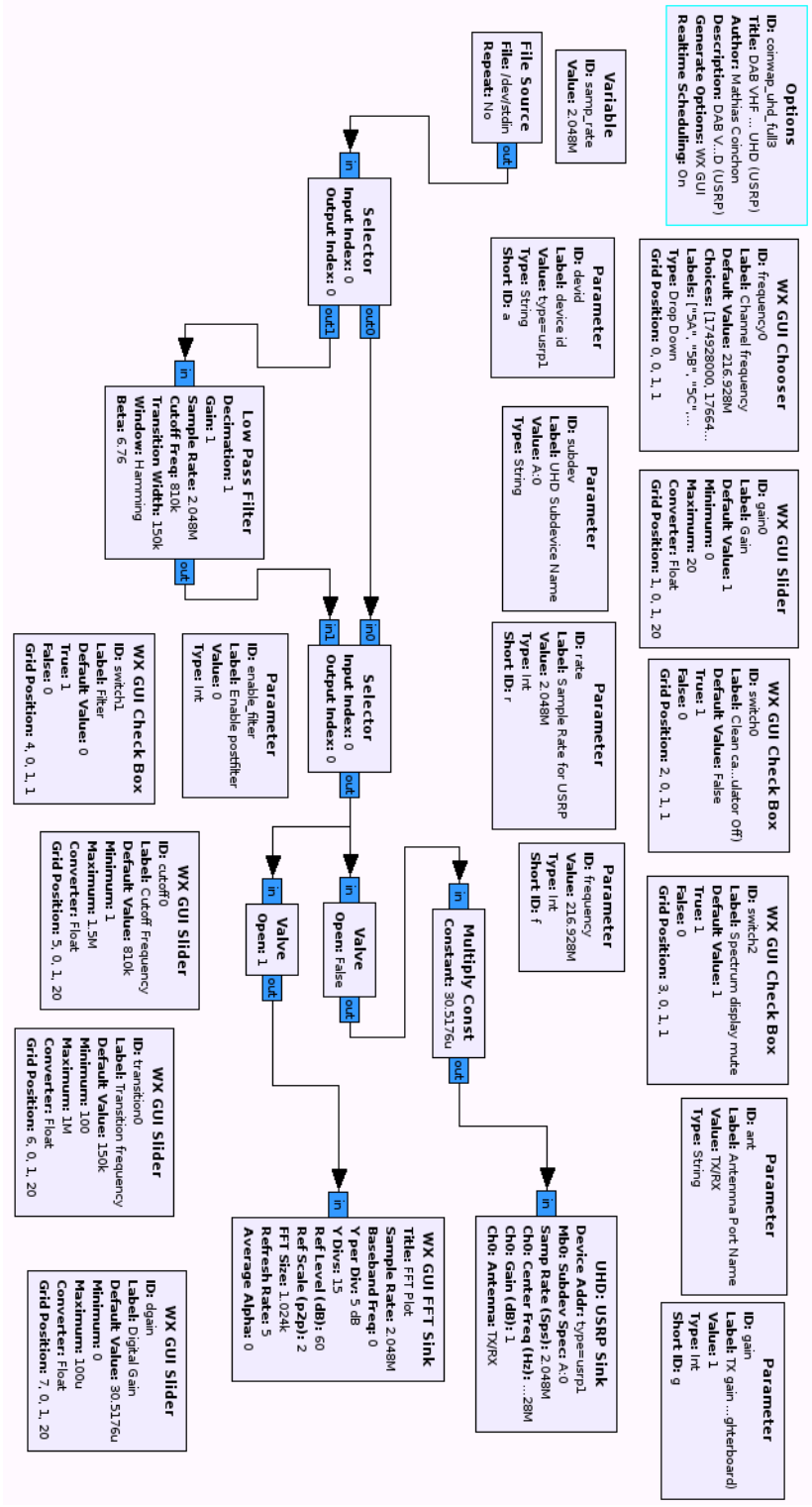

Figure 9: The `coinwap-uhd` wave player as seen in gnuradio-companion.

4 Adding SFN Support to the CRC mmbTools

4.1 Motivation and goals

The main goal of this project is to make it possible to use the existing CRC mmbTools to create an single-frequency network, using the Ettus Research USRP family of devices. In the present state, the mmbTools form a rich set of tools for experimentation with DAB, trials and short temporary transmissions in DAB and DAB$^+$. Without SFN support, they however lack an important feature necessary in larger projects requiring extended coverage areas. By adding this feature, these open-source tools will become more suitable for a larger range of broadcasting projects, while still keeping the cost-effectiveness of the USRP hardware platform.

In order to achieve this goal, I have developed a way to synchronise the modulators and the transmitters to a master clock. Also, the ETI stream gets distributed to all transmitters, while respecting important timing constraints, using a small program I wrote for this. A normal IP network is used for this.

With these modifications, the setup described in figure 10 is now working. In this setup, each transmitter has one modulator and all of them receive the same ETI stream over an IP network. The whole system must cope with the additional delay caused by the IP network.

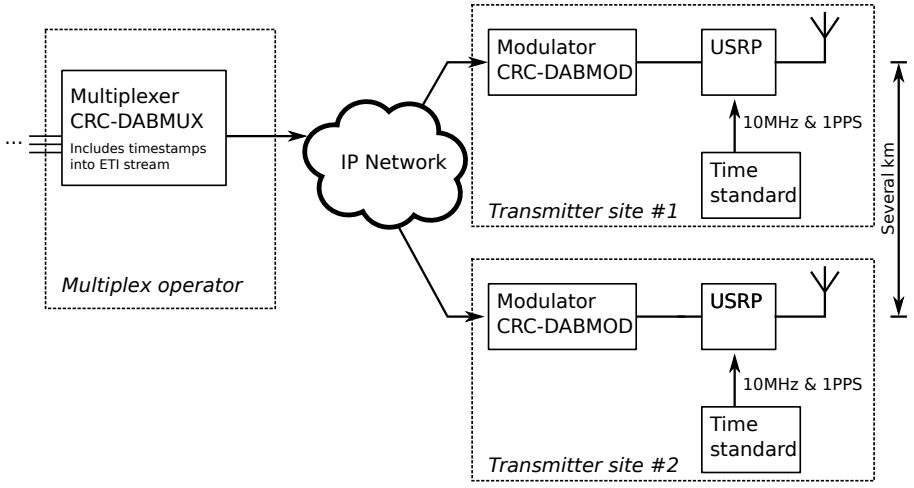

Figure 10: Proposed system for a time-synchronised SFN transmission with two transmitters.

This is not the only possible setup. It could also be possible to transmit the modulated baseband RF signal to each transmitter location, and have only one modulator. However, this leads to a thirty-two-fold increase in required network bandwidth: The ensemble transport interface, designed for the transmission of ensemble data from the multiplexer to the modulator, has a bit-rate of up to 2.048 Mbps, whereas the RF signal is represented 32 bit samples at a rate of 2.048 Msps. Furthermore, it would also require a non-standard encapsulation to transmit the timestamps.

This reason, combined with the desire to stay compatible with professional multiplexing and modulation equipment that is interconnected by the ensemble transport interface has rendered the first approach preferable.

In this section, I will define what are the requirements for the creation of a single-frequency network, show what features of the drivers are useful, and finally explain the modifications done on the mmbTools.

4.2 Requirements in a single-frequency network

4.2.1 Tolerable relative delay between transmitters

As explained in section 2.4, the relative delay between different transmitters must be below 1 µs. This means that the granularity of the delay control for each transmitter offset must be smaller that this value and that each transmitter must respect the specified offset with this precision.

4.2.2 Distribution of the ETI data to all transmitters

Until now, all experiments and test transmissions using the CRC mmbTools have used a single PC for multiplexing and modulation, and the communication between the two has only been done through UNIX pipes and files.

In a single-frequency network, and more generally in the case where there are several transmitters, it is necessary to place multiplexing and modulation at distinct physical locations. This implies that a distribution network for the multiplexed ensemble must be put in place. This is the network described as "IP Network" in figure 10. The distribution network connects all transmission sites to the multiplex site, and carries the ETI stream.

The ETSI standards TS 102 693 [11] and TS 102 821 [9] define a transport mechanism for the ETI distribution, and a transport protocol, but do not discuss the implications of using them on an existing IP network.

For cost-efficiency, it is preferable to use an existing network over a dedicated one (leased lines). The most interesting network in this case is the Internet, because of the following advantages it offers:

- Enough bandwidth theoretically available, even in more rural areas;
- Lower price than leased lines thanks to the wide spread deployment of xDSL residential Internet access connections;
- IP-based, enabling the use of diverse transport protocols.

However, being a packet-switched network, there are some drawbacks:

- No real-time guarantees concerning packet delivery times, leading to unpredictable delays;
- All traffic is best-effort, and shared among several users, which might pose bandwidth constraints.

Then, a suitable transport protocol has to be chosen. It is important for CRC-DABMOD that one ETI frame arrives completely, because there is no resynchronisation possible if a part of a frame is lost. In this happens, the modulator will crash. Also, the ETI stream must not suffer errors. Therefore, the chosen transport protocol must guarantee that one ETI frame reaches either correctly and in its entirety, or not at all.

If some of frames are missing, the modulator will resume modulation afterwards without problems, because the frame numbering is not verified. However, this also means that if two frames arrive in the wrong order, they will be considered in the wrong

order too. In the transmission modes where several ETI frames are taken together to modulate (TM 1 and TM 4), this can pose problems even if timestamps are defined.

Therefore, when using TM 1 or TM 4, the transport protocol must also ensure that all frames arrive at the modulator, and that they are in order.

4.2.3 Delay management and fault handling

Delay planning is an important aspect of single-frequency network planning. In a network of several transmitters, it must be possible to control the relative delay for each transmitter. Section 7.6.4 of Hoeg and Lauterbach [14] gives an example of an existing DAB SFN with the corresponding transmitter delays.

Figure 11 shows the timing of the ETI data at three different places: At the multiplexer, where the frames are created, and at two modulators, connected through a network to the multiplexer. We will consider frame number 51, but the same applies to all frames. We assume that the three entities possess a synchronised clock. The multiplexer generates frame i at time t_i, and includes that information in the timestamp, in the frame itself. We want all modulators to output the modulated RF waveform corresponding to the ETI frame at time $t_{\mathrm{TX},i}$, defined by a constant offset:

$$t_{\mathrm{TX},i} = t_i + \mathrm{offset}$$

Because of the network, each modulator receives the frame with a different delay t_{NET}. Since all modulators share the same clock, they will be able to delay transmission until exactly the right point in time.

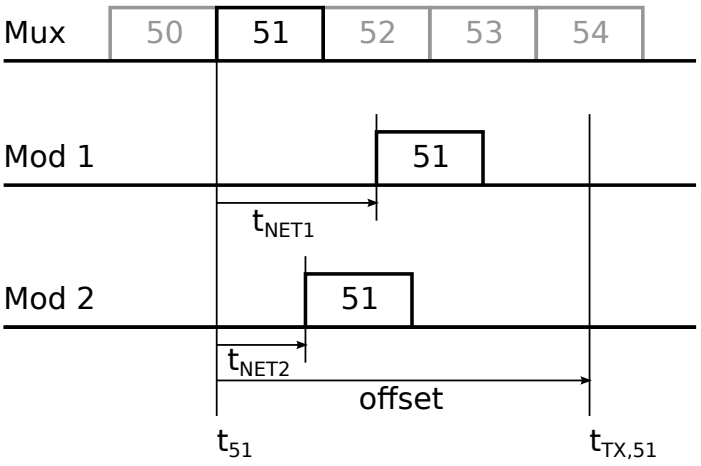

Figure 11: The ETI frame arrival time depends on the network delay.

It is important to note that the offset cannot be shorter than the maximal network delay, which comes from the fact that the network link with the highest latency defines the minimal latency between multiplexing and modulation.

4.3 Time handling in the USRP

There are two ways of using a precise time reference for the USRPs. The simple solution is to add the official, internal GPSDO that is available from Ettus Research.[9] This GPSDO is based on the Jackson Labs Firefly-1A device. It is also possible to use an external time source, thanks to the 10 MHz REFCLK and 1PPS inputs.

In both cases, the reference clock is used as main clock for the FPGA, the D/A converters and the analog circuit parts. On the USRP2, the clock is used by the Analog Devices AD9510 PLL and clock distribution IC.[10] On the USRP B100, the Analog Devices AD9522-4 is used for the same purpose. Because the REF clock is directly used in a phase-locked loop, which is sensitive to phase noise, care must be taken to have a reference clock of good quality.

In order to tell the USRP to use the external reference clock and the external 1PPS source (both on SMA connectors), the following must be executed by the program using the device (`usrp` is a pointer to the usrp device object):

```
uhd::clock_config_t clock_config;
clock_config.ref_source = uhd::clock_config_t::REF_SMA;
clock_config.pps_source = uhd::clock_config_t::PPS_SMA;
clock_config.pps_polarity = uhd::clock_config_t::PPS_POS;
usrp->set_clock_config(clock_config,
        uhd::usrp::multi_usrp::ALL_MBOARDS);
```

Listing 1: Setting clock configuration for the USRP

The 1PPS input is used to define the time inside the FPGA. The idea is to tell the FPGA to start the internal clock at a precise moment in time, while the clock advances at the speed defined by the reference clock input. The information about what time it is is given over another communication channel, either internally when the GPSDO is used, or by the computer (over USB or Gigabit Ethernet) otherwise. The UHD driver exports functions that allow the time to be set synchronously to the 1PPS rising edge. The simplest function sets the time at the next 1PPS rising edge:

```
usrp->set_time_next_pps(const time_spec_t &time_spec,
        size_t mboard = ALL_MBOARDS);
```

Listing 2: Specifying the time at the next 1PPS rising edge

This function can be used if the host computer also receives the 1PPS input, and knows when the rising edges occur. As this is not always the case,[11] it is also possible to synchronise the USRP time using another approach. Using the network time protocol (NTP), computer time can by synchronised to UTC, with a precision of at least 100 ms, often much better[12] as seen on figure 12.

When synchronised using NTP, the clock difference between the host computer and the GPS receiver is less than the estimated NTP offset, with a very high probability. This

[9] More details about the Ettus GPSDO: `https://www.ettus.com/product/details/GPSDO-KIT`

[10] Schematics for select models of USRPs and daughterboards have been published, from which this information has been taken. See the website `http://code.ettus.com/redmine/ettus/projects/public/documents` for more details.

[11] Additional interfacing circuitry between the GPS receiver and the PC is required, e.g. a serial RS232 connection, a parallel port or a dedicated 1PPS interface card. See `http://linuxpps.org` and the relevant documentation in the linux kernel source tree (`Documentation/pps/pps.txt`) for more details about how to interface a 1PPS signal to a linux computer.

[12] This depends on the PC clock quality and the manner in which the internal PC clock is handled. Often, an estimate error smaller than 10 ms can be achieved.

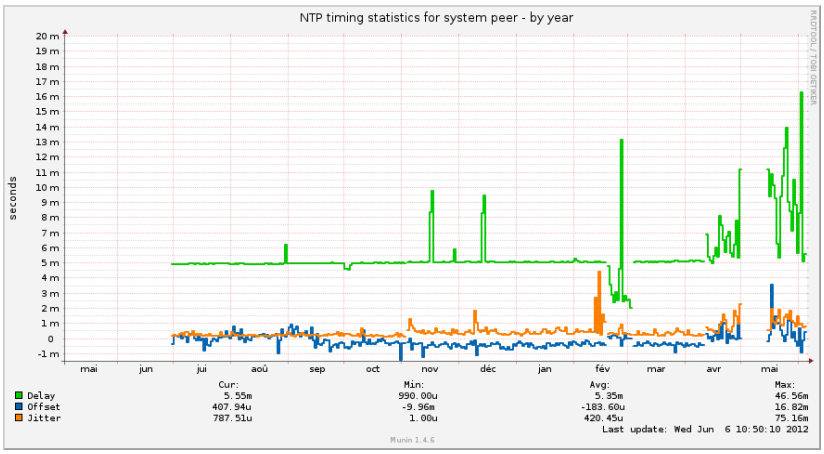

Figure 12: Network delay, clock offset and jitter monitored and graphed on a standard Linux based PC. Over the course of one year, the clock offset always was below 5 ms even in varying network conditions. Three different NTP servers are used on this client. The outage in may was due to other reasons.

fact can then be used to synchronise the USRP clock using the protocol outline in figure 13. The middle arrow shows code flow in listing 3. It is assumed that the GPS time has a negligible offset to NTP time, and I will use the term *absolute time* to represent either of those. Also, the GPS receiver offset relative to absolute time is assumed to be less than ±100 ns. The host computer time is also assumed to be synchronised to absolute time with a maximal (positive or negative) offset of 20 ms. The protocol used is the follwing:

- First, wait for the host computer clock to change second;
- Then, wait for a timespan that guarantees that the 1PPS rising edge has passed;
- Finally, call `set_time_unknown_pps`, which waits on for the next 1PPS rising edge, and sets the time in the USRP on the subsequent one by calling `set_time_next_pps`.

```
1  void set_usrp_time() {
2      struct timespec now;
3      time_t seconds;
4      clock_gettime(CLOCK_REALTIME, &now);
5
6      seconds = now.tv_sec;
7
8      while (seconds + 1 > now.tv_sec) {
9          usleep(1);
10         if (clock_gettime(CLOCK_REALTIME, &now)) {
11             error();
12         }
13     }
14     /* We are now shortly after the second change. */
15
16     usleep(200000); // 0.2s, we want the PPS to be later
17     usrp->set_time_unknown_pps(uhd::time_spec_t(seconds + 2));
```

```
18 }
```

Listing 3: Synchronising the USRP clock using a NTP synchronised host and a GPS as 1PPS source

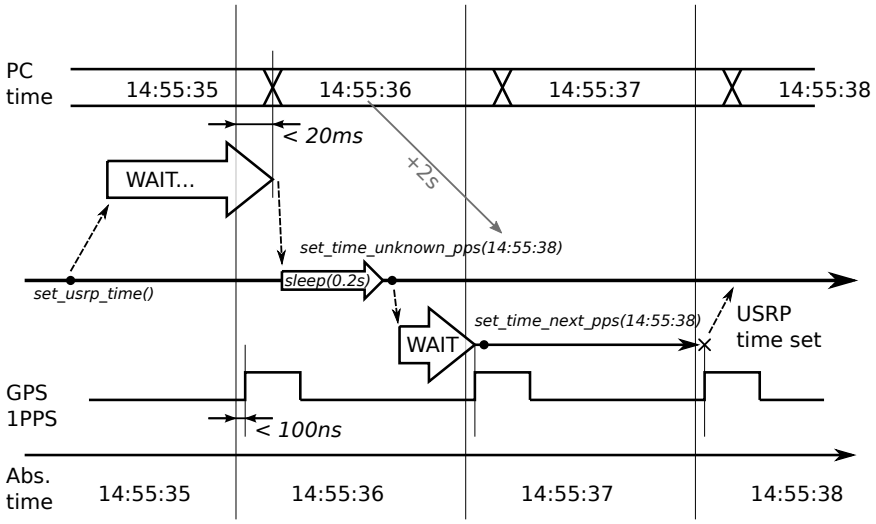

Figure 13: Setting time in the USRP from a NTP synchronised host, with the 1PPS generated by a GPS receiver requires several steps.

Once the time is set, it is possible to add a timestamp to samples that have to be transmitted. Each sample that is sent to the USRP is accompanied by metadata structure. The metadata in UHD is defined by:

```
1 struct tx_metadata_t {
2     bool has_time_spec;
3     time_spec_t time_spec;
4     bool start_of_burst;
5     bool end_of_burst;
6     tx_metadata_t(void);
7 };
```

Listing 4: UHD transmit metadata structure

The *start of burst* (*end of burst*) field is used to signal the USRP when a series of samples, called a burst starts (stops), which is used when transmitting non-continuously. This is not used in the DAB transmitter application, because it uses continuous streaming.

The `has_time_spec` specifies if the `time_spec` field is valid and should be considered. This contains the exact time at which the first sample of the burst shall be sent. These timestamps are considered each time streaming begins and at each beginning of a burst.

Assuming the precise time at which a sample must be transmitted is given as the sum of an unsigned `tx_second` and a floating point value `pps_offset` between 0 and 1, timestamped transmission can be achieved as follows:

```
1 struct tx_metadata_t md;
2 md.has_time_spec = true;
3 md.time_spec = uhd::time_spec_t(tx_second, pps_offset);
```

```
uhd::stream_args_t stream_args("fc32"); //complex floats
uhd::tx_streamer::sptr myTxStream = usrp->get_tx_stream(
    stream_args);
size_t bufsize = myTxStream->get_max_num_samps();
while (running && (num_acc_samps < samplebuffer_size)) {
    size_t samps_to_send = std::min(samplebuffer_size -
        num_acc_samps, bufsize);
    //send a single packet
    size_t num_tx_samps = myTxStream->send(
            &in[num_acc_samps], samps_to_send, md, timeout);
    num_acc_samps += num_tx_samps;
    md.time_spec = uhd::time_spec_t(tx_second, pps_offset)
                 + uhd::time_spec_t(0, num_acc_samps/sampleRate)
                     ;
}
```

Listing 5: Setting time in the transmit metadata structure

Since it is not known in advance how often `send` must be called, a loop is required. In each iteration, I accumulate the number of transmitted samples into `num_acc_samps`, and update the time specification in the metadata according to the sample rate.

In the FPGA fabric on the USRP, the `vita_tx_control.v` module reads the timestamp, encoded on 64-bit, and compares it to a counter defined in `time_64bit.v`. This counter is incremented in each cycle in the FPGA. The translation from numeric time (as represented in `struct tx_metadata_t`) to 64-bit raw form is done by the host code in UHD.

Thanks to this setup, the USRP hardware driver allows to transmit samples at an accuracy up to one FPGA clock cycle, and forms the basis on which the DAB transmission generated by the CRC mmbTools can be synchronised.

4.4 Modifications in the CRC mmbTools

In order to support timestamped transmission, I have modified several parts of the transmission chain. These modifications are illustrated in figure 14, in which the shaded parts represent support for timestamps. CRC-DABMUX has partial support because it defines the TIST field, but does not transmit time information. With nothing else that the TIST, there is an ambiguity about which second the frame has to be transmitted. The CRC-DABMOD modulator ignores the timestamps, and cannot give any information to GNU Radio about timing. The UHD driver and the USRP support timestamps as described above.

The arrow summarises the changes I have done to the different parts, mainly in CRC-DABMUX and CRC-DABMOD. I have added precise time information in CRC-DABMUX, and included the UHD driver into CRC-DABMOD. The latter takes care of the delay calculation that is communicated to the USRP.

Thanks to these modifications, it is possible to transmit synchronously using the mmbTools and the USRP platform.

4.4.1 Encoding time into the ETI data

In order to support synchronised transmission, the transmission time must be embedded into the ETI frames. As explained in section 2.3, the time information is transmitted in two fields in the ETI frame: the MNSC contains the absolute date and time up to one-second resolution and the TIST field contains the 1PPS offset in 61 ns resolution. The

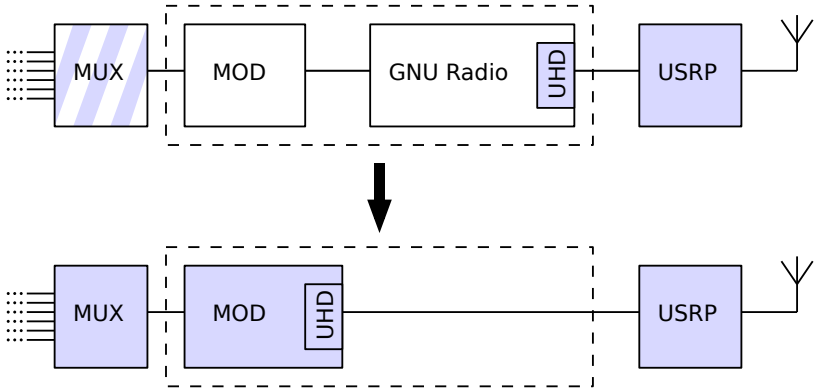

Figure 14: timestamp support has been added to CRC-DABMOD, that directly interfaces with the hardware.

precise details about TIST encoding are given in annex C in ETS 300 799 [5]. The PPS offset is encoded as a 24-bit unsigned integer, representing a multiple of 16.384 MHz clock periods. If all TIST bits are set to "1", the timestamp is disabled. TIST encoding can be selectively enabled using command-line arguments.

I have modified CRC-DABMUX such that these two fields are correctly defined. They are synchronously updated, so that the MNSC time gets incremented each time the TIST rolls over. The TIST is incremented by 24 ms for each frame, and taken modulo 1 s.

For the time in the MNSC, the information must be split over several frames, according to sections 5.5.1 and A.2.2 of ETS 300 791 [5]. Table 2 lists the information contained in the MNSC for four ETI frames, that compose a complete time signalling group.

FP & 0x3	MNSC byte 0	MNSC byte 1
0	type=Time	–
1	second/accuracy	minute/sync-to-frame
2	hour	day
3	month	year

Table 2: The time is encoded into four ETI MNSC fields.

This time information is synchronous to the frame with FP (mod) = 0 (i.e. the first of the four frames), and does not have to be included regularly. The MNSC can also be used to send other information than time to the modulators. This is not a big issue, since the modulators can keep track of time using the TIST field, incrementing the time each time the TIST rolls over.

The time encoded in MNSC is the actual time when the multiplexer creates the frame, and is represented in UTC. Care has to be taken that all devices reading this time are properly synchronised in regard to the UTC leap seconds.

These modifications respect the standard, and allow the combination of CRC-DABMUX with commercial modulators. At first, I have implemented a non-standard way of transmitting time in the padding of ETI(NI). This approach was simpler because it was not subjected to the bitrate and coding constraints of the MNSC, and it was possible to encode the full timestamp into each ETI frame. After discussing the implications of the non-standard encoding, I have then decided to implement the standard way. However, this first encoding has acted as a proof of concept for the synchronisation of two DAB

transmitters.

4.4.2 Decoding timestamps in CRC-DABMOD

The timestamp that CRC-DABMUX encodes into the ETI data obviously has to be decoded at the modulator. This is done at two different places in CRC-DABMOD. for this, I have modified the `ETIReader` module, and created a new `TimestampDecoder` module. The first module extracts the TIST and the MNSC fields into C structures, and then calls the function `updateTimestampEti(int framephase, uint16_t mnsc, double pps)` defined in the `TimestampDecoder` module. This decoder combines the information carried in the MNSC with the TIST that is included in each frame. Each transmission frame is then associated with a complete timestamp.

The decoder does not assume that the time is sent regularly using the MNSC, because that same channel can carry other kinds of data. Therefore, the rollover of TIST field is used to know when the second changes. The data from the MNSC is however required to set the time initially.

In order to avoid transmitting frames at the wrong time, transmission is inhibited as long as no complete timestamp has been decoded.

4.4.3 Multi-threaded modulation and USRP driver output

In the transmission chain used in the previous trials and experiments, GNU Radio was used to interface the modulator with the USRP. The GNU Radio script (also called "Baseband wave player") performs filtering and normalisation of all samples to the range $[-1.0; 1.0]$. As explained in section 4.3, the USRP platform supports timestamped transmission, but this feature is not yet available in GNU Radio, even though it also uses the UHD driver.

Therefore, a way must be found to give the time information to UHD. For synchronisation, it is better to carry the time information along with the data itself, and not over distinct paths. This makes it complicated to keep GNU Radio. Putting the UHD interface into CRC-DABMOD is a much cleaner approach, because the management of the time information is done only in one program. For this reason, GNU Radio has been removed.

Using UHD directly from CRC-DABMOD has several advantages:

- Less buffers between modulation and transmission device, leading to better timing control of the modulation chain;
- Tighter control of timestamps, because both data and time information are available in the same code.

The disadvantages are that the filtering part has to be re-engineered in CRC-DABMOD, and that showing the digital spectrum display is not possible anymore. I have reimplemented the filtering, which is described in section 4.4.5.

I have added an additional output module called `OutputUHD` to CRC-DABMOD. This output modules creates a new "UHD Worker" thread that sends data to the USRP. The main thread receives data from the other modules in the modulator and writes them into a double-buffer. The data to be sent is read from the double-buffer by the worker thread. Both threads are synchronised by a barrier, that blocks thread execution until both threads have called `wait()` on the barrier. At each barrier, the two threads exchange their buffers, so that one buffer is written to by the main thread while the other buffer is being read and sent by the worker thread. This principle is shown in figure 15. We do not know how often `send` must be called to transmit a complete DAB transmission frame.

In this example, we assume three times. I have used the threading and synchronisation primitives from the Boost C++ library `boost_thread`.[13]

On the timing diagram, upward pointing arrows represent call to the barrier `wait()` function, and downward pointing arrows show when the thread can go on after the barrier. In order for this system to work, the main thread must wait first, which is equivalent to saying that modulation must be faster than transmission.

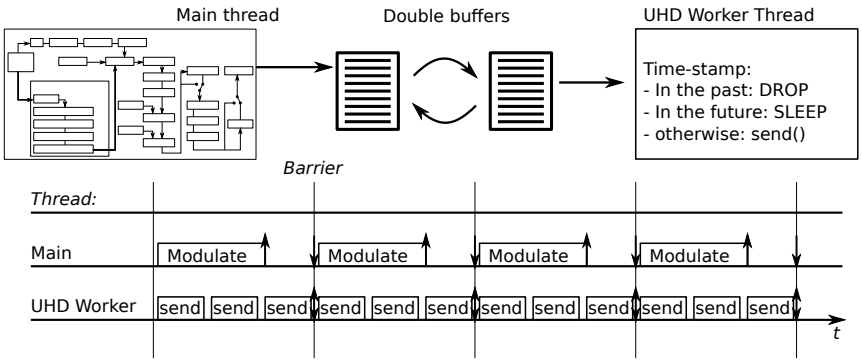

Figure 15: Multi-threading in the UHD output module allows that modulation and data transmission to the USRP can happen simultaneously.

The worker thread not only sends the frames to the USRP, but also verifies the timestamps. If the timestamp is in the past, then the frame is dropped. When this happens, the barrier will be reached in a time shorter than 24 ms, and the input buffer of the modulator will be emptied faster than it is filled.[14] The modulator will thus catch up on the late frames, until the timestamps are not in the past anymore.

On the other hand, if the timestamps are too far in the future, the worker sleeps so as to fill the input buffer, and to bring the timestamp to the near future. Then, the frame is sent with its accompanying metadata. The FPGA fabric in the USRP then takes care to free the samples at the right time.

The `OutputUHD` module also sets the following parameters: Transmission frequency, sample-rate for the USRP, transmission gain of the programmable-gain amplifier of the WBX daughterboard and the clock configuration given in listing 1.

I have modified the main module `DabMod.cpp` that handles command line arguments parsing, so as to offer additional options:

- -f *name*: Use file output with given filename. (use /dev/stdout for standard output)
- -u *device*: Use UHD output with given device string. (use for default device)
- -F *frequency*: Set the transmit frequency when using UHD output. (mandatory option when using UHD)
- -G *txgain*: Set the transmit gain for the UHD driver (default: 0)
- -o *offset*: (UHD only) Set the timestamp offset added to the timestamp in the ETI. The offset is a double.

[13] `http://www.boost.org/libs/thread`

[14] The fill-rate is defined by the ETI data rate, which is one frame every 24 ms.

- -O *offset-file*: (UHD only) Set the file containing the timestamp offset added to the timestamp in the ETI. The file is read every six seconds, and must contain a double value. Specifying either -o or -O makes DABMOD mute frames that do not contain a valid timestamp.
- -T *taps-file*: Enable filtering before the output, using the specified file containing the filter taps.

4.4.4 Handling per-modulator transmission delay

Delay management is very important in single-frequency networks. We must be able to give a different offset to each modulator, that is then added to the full timestamp.

There are two different ways that have been implemented to achieve this. The simple way is to set an offset on the command-line, when launching CRC-DABMOD. The offset is then constant for the whole runtime. This is done using the `-o <offset>` option, where `offset` is a floating point value in seconds. The second way, more flexible because it allows changing the offset without stopping CRC-DABMOD, is using the `-O <offsetfile>` option. The offset is written into a file (a single floating point value, on one line), which is then read by CRC-DABMOD in regular intervals. When the offset changes, transmission is interrupted and restarts at the right time.

In both cases, the `TimestampDecoder.cpp` module takes care of adding the offset to the decoded time information.

Information specific to transmitters is usually conveyed in the MNSC, and the standard defines a way to transmit delay information to modulators. However, since CRC-DABMUX does not yet support multiplex reconfiguration without a restart, it would be awkward to include this feature in the MNSC. Implementing the offset definition at the transmitter site makes tuning more complicated, but the multiplexer can be left running. If the multiplexer would have to be stopped each time the offset for one transmitter has to be changed, it would hamper delay management.

4.4.5 Filtering

Because of the missing timestamp support in GNU Radio, the modulator must work without it. While the normalisation is trivial to implement since the modulator already includes a gain module (`cifGain`), filtering is more complex.

In GNU Radio, the low-pass filter is a finite impulse response filter (FIR filter). The filter taps are calculated using the `firdes` function that is included in GNU Radio. Listing 6 shows how to calculate the taps for a low-pass filter described by the cutoff frequency, and the transition width. The resulting filter coefficients are shown on figure 16. These filter taps are then convolved with the sample data to apply the low-pass filter.

```
1 import gnuradio
2 from gnuradio import gr
3
4 gain = 1
5 rate = 2.048e6
6 cutoff = 810e3
7 transition_width = 150e3
8 beta = 6.76
9
10 taps = gr.firdes_low_pass(gain, rate, cutoff, transition_width,
     gr.firdes.WIN_HAMMING, beta)
```

```
11 print(len(taps))
12 for t in taps:
13     print(t)
```

Listing 6: Using gr.firdes to calculate the taps for a low-pass filter

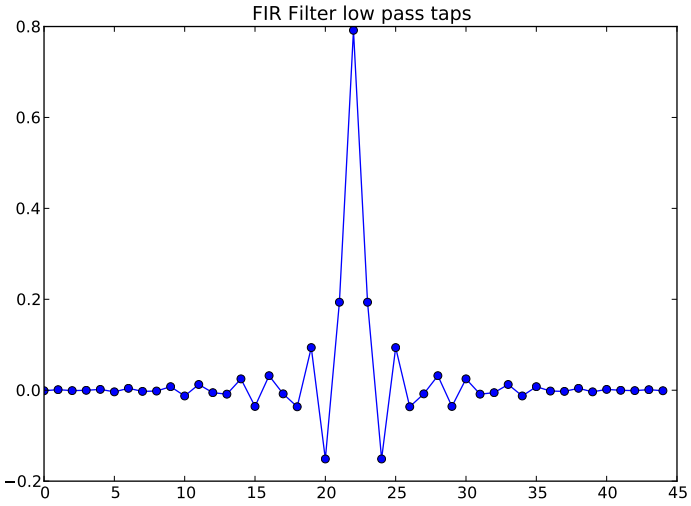

Figure 16: The default low-pass filter generated using firdes has length 45.

Implementing this filtering in CRC-DABMOD is simplified by the fact that the filter length is short compared to the length of the null symbol, that is shortest in TM 1 with length 345. Thanks to this, it is possible to filter each frame independently, because we can consider that the frame boundaries represent samples that are zero.

A FIR filter block has been implemented that calculates the convolution in two different manners, depending on the availability of the SSE extensions: One takes advantage of the SSE SIMD instruction set extension available in x86 processors by executing four multiplications with one instruction. If SSE is not available a slower unrolled loop is used. On my ThinkPad T420 (Intel Core i7-2640M, 2.8 GHz), the SSE version is 1.5 times faster than the unrolled loop. The `FIRFilter` module includes a timing code for this, measuring time passed in the processing loop. The values have been averaged over ten runs.

In order to avoid doing both modulation and filtering in a single thread, the FIRFilter module creates another thread. Data synchronisation is achieved through a thread-safe, blocking queue, as shown in figure 17.

The read-only filter coefficients are read from a file at startup, and need not be synchronised across the threads. The python program in listing 6 can directly be used to create a file containing the coefficients. The file must be in the following format:

- The file is saved in a human-readable format, with one value per line. Line endings are system dependent;
- The first line must contain an integer, specifying the number of taps `n_taps`;
- Then, the file must contains `n_taps` lines, with one real floating point value on each line.

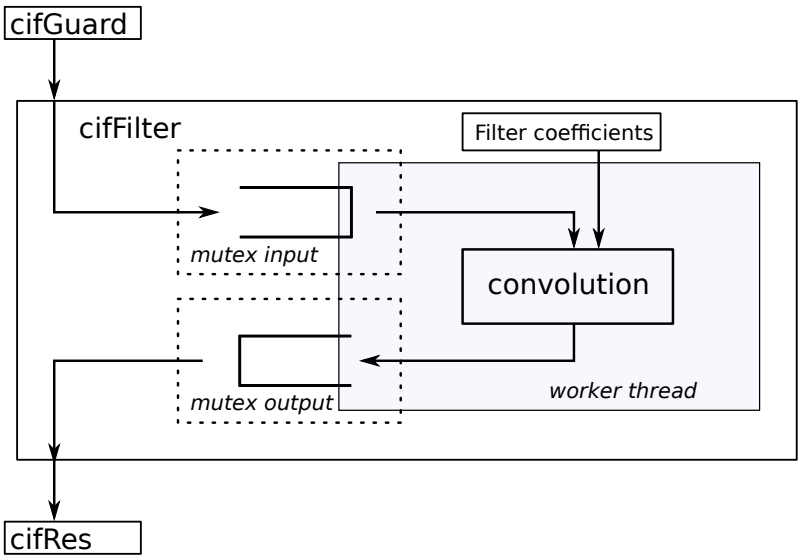

Figure 17: The basic structure of the FIRFilter block, with the worker thread in the shaded area.

The filter coefficient file is given to CRC-DABMOD using the new `-T <filtertaps>` command line, that also enables the filter block.

4.4.6 Distribution of ETI over an IP network

The simplest transport protocol to use over IP is TCP. As a stream protocol, it considers that the transferred data is continuous. If it is used to transmit the ETI(NI) stream, the frame boundaries will be lost, but thanks to the synchronisation and the constant frame size, this does not pose a problem. Also, TCP guarantees that the stream will reach the receiver in the same form as it has been sent, without errors. To insure this, TCP retransmits data that was lost on the network or that is erroneous. TCP offers more than enough guarantees to be used to directly transport ETI(NI) data, without the need of an additional encapsulation. However, its retransmissions can potentially be harmful, because network resources are used to retransmit data for ETI frames that will possibly be too late to be useful on arrival.

To alleviate this, a UDP-based protocol could be used instead of plain TCP. UDP does not send a continuous stream of data, but sends units of data called "datagrams". However, these datagrams could get lost on the network, or reach the receiver in another order than they were sent. For UDP to be usable to transport ETI(NI) data, an application protocol that solves these issues has to be created. This application protocol could use a forward error correction code to compensate for lost packets, and must support reordering at the receiver.

Since the drawbacks of using TCP are not really significant, and because it is much simpler to set up, I have implemented a small TCP server to transmit ETI(NI) data. CRC-DABMUX possesses a TCP server output too, which does not support several connections. Furthermore, care has to be taken that the TCP sockets do not block multiplexing, which should always run at the nominal rate regardless of the ETI consumers. The small server uses queues whose writes are non-blocking to make sure that the multiplexer never

blocks, as illustrated in figure 18. The queue write either succeeds, writing all requested data into the queue, or fails when the queue cannot accept one more ETI frame. That way, no ETI frame is partially written. Furthermore, Eti-TCP disables Nagle's algorithm[15] to shorten the latency between server and client.

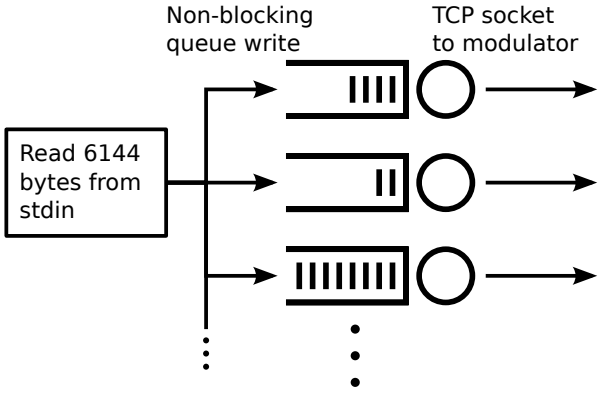

Figure 18: The Eti-TCP server sends data to several connections. The third connection has stalled and its queue is full, but this does neither harm the other connections nor the multiplexer. Each queue element represents one ETI(NI) frame (6144 bytes).

The Eti-TCP server is implemented in python and is multi-threaded. One thread is used to read from standard input and fill the queues, one thread listens for new incoming connections, and each connection creates one thread. In total, $N + 2$ threads are used when N connections are active.

With TCP, there is no need for a sophisticated client, because there is no additional application protocol. Thanks to this, the netcat tool[16] can be used to receive the ETI data. The data is then piped into CRC-DABMOD's standard input using an operating system pipe.

[15] Described in RFC 896, section "The solution to the small-packet problem" [15], and suggested by Baset et al. [3].

[16] Several versions exist of this old UNIX tool exist, e.g. GNU Netcat `http://netcat.sf.net`

4.4.7 Summary of changes to CRC-DABMOD

A summary of all changes done to CRC-DABMOD can be seen in figure 19.

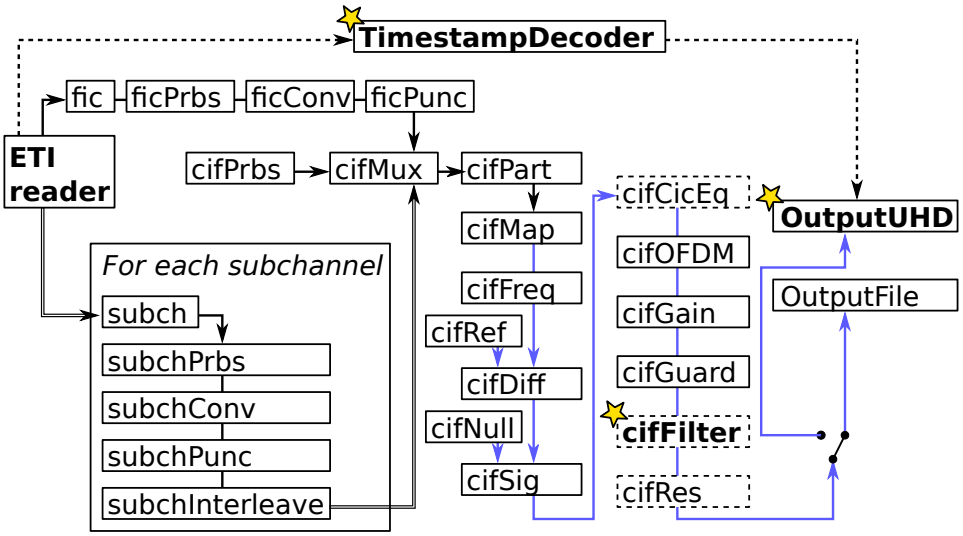

Figure 19: The flowgraph of the modified CRC-DABMOD modulator (version 0.3.3), black arrows represent data, blue arrows are complex floating-point samples and the dashed arrows show time information. Dashed boxes are only enabled in some circumstances. Modules with a bold font have been modified, and those with a star have been added.

5 GPS as Time Synchronisation Source

5.1 Overview

For a single-frequency network to work well, it is necessary to have a precisely synchronised time at each transmitter. I will discuss this aspect more in depth here, and compare several solutions. It will become clear why GPS is the preferred time source for such applications, and why most commercial solutions are based on it.

First, recall that two key parameters have to be synchronised:

- Frequency : carrier frequency, sampling frequency, symbol rate, etc. are derived from it;
- and Time : the point in time at which a specific frame gets transmitted.

Many types of oscillators exist that can be used as a frequency source, with a wide range of precision and cost. For the time synchronisation however, it is more difficult to find a simple source. These will be discussed here.

5.2 Comparison of oscillator types

Because they are so important in electronics, communications and many other fields, oscillators exist in a wide variety of technologies and performance. In order to be able to compare their performance, some metrics have to be defined first.

The first important metric is *short-term stability*. This characterises the deviation of an oscillator over a short timespan (of the order of seconds), that is mainly caused by noise in the active elements of the oscillator. Averaging over a longer time helps to remove this noise. A more important metric is *long-term stability*, also called *aging*. For a crystal oscillator, the Anritsu application note "Understanding Frequency Accuracy in Crystal Controlled Instruments" [1] mentions three important factors that influence its aging: "relaxation of mechanical stress", which is due to the fact that the crystal is subjected to mechanical stress when mounted in its enclosure ; "movement of impurities" that migrate through the crystal ; and the fact that "material comes loose from, or adheres to the crystal."

Finally, we want our oscillators to have a high *accuracy*, meaning that the actual frequency corresponds to the design frequency. One of the parameters that strongly influences accuracy is temperature. Crystal oscillators are very sensitive to temperature variations, and a good oscillator will have to control the crystal temperature.

Another important aspect for oscillators is *phase noise*. It is related to jitter, the term used for digital signals. The effect of phase noise, for a pure tone, is to spread out the energy around the centre frequency of the signal. This aspect is covered more in depth in chapter "Phase noise" in "Fractional/Integer-N PLL Basics" [2].

Oscillators can be used in several ways, which will be presented here.

Room temperature The simplest way to use a crystal oscillator is without any temperature control. The crystal oscillator will be subjected to temperature variations of the environment, and the accuracy will be fair. When used like this, the oscillator is simply called a "crystal oscillator" ("XO").

Temperature-compensated crystal oscillator A TCXO contains additional tuning circtuitry that measures temperature and compensates the measured variations. This compensation can be done with an analog or with a digital circuit. In both cases, it is important to know the temperature behaviour of the crystal precisely, in order to do a good compensation. These TCXO oscillators have a much better accuracy than simple crystal oscillators.

Oven-controlled crystal oscillator To get even better performance, it is necessary to control the operating temperature of the crystal. This is achieved using a small heater and an enclosure that stabilises the temperature at a specific value. Using this oven, the factor most strongly influencing accuracy is effectively controlled. These OCXO have the best performance practically achieveable using crystals.

For these three types of oscillators, quartz is the most commonly used crystal, because of its good characteristics and its high Q-factor. Oscillators based on other materials than piezo-electric crystals also exist.

Atomic standard Caesium atomic clocks are the most precise frequency standards existing today, and the second in the SI system of units has been defined using this clock. The same principle also exists with another material, rubidium, that can be used to create small frequency standards. A rubidium frequency standard has a better accuracy than OXCOs.

5.3 Using GPS to discipline oscillators

Using well-calibrated crystal oscillators, it is possible to have a very good long-term stability. But for the DAB SFN application, a synchronised pulse-per-second signal is also required at each transmitter site. This requires some common clock between all transmitters.

In order to calculate the position, GPS receivers need to do precise timing measurements, and need to align their internal clock to the global GPS clock. The availability of cheap GPS receivers makes it possible to have an affordable way to synchronise time in geographically distant places. Furthermore, many receiver modules have an output signal that is aligned to the GPS second change. The delay between the rising edge of this signal and the change of second is often shorter than 200 ns.

By combining a GPS receiver with a precise oscillator, it is possible to synchronise both time and frequency at several places. However, if the oscillator frequency is also used for timekeeping and not only for frequency alignment, then the time derived from the oscillator will drift away from the GPS time. In that case, it is either necessary to readjust the oscillator clock to the GPS clock periodically, which creates clock-jumps, or to link the oscillator against the gps clock. The latter solution is preferred, because the oscillator will be disciplined by the GPS receiver, which will increase the frequency accuracy by several orders of magnitude.

Figure 20 shows one way of building such a system using a phase-locked loop.

In theory, any oscillator type can be used in a GPSDO, but most often, OCXOs are used, because the offer the best performance in case of a GPS signal loss.

Some GPS receivers have additional outputs that can be used to lock a PLL. For instance, the Navman Jupiter-T has a 10 kHz output that can be useful to lock a PLL. A

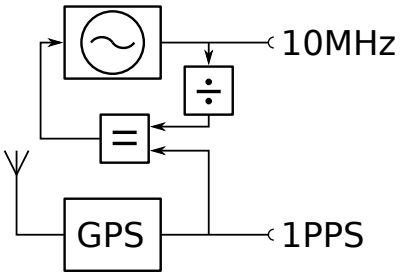

Figure 20: The oscillator output is divided to get a 1 Hz signal that is compared to the 1 Hz PPS signal from the GPS receiver. This phase comparison is then used to tune the oscillator, forming a feedback loop.

GPSDO design including schematics is available at James Miller's website.[17]

Type	Accuracy	Aging	Power	Weight
XO	10^{-5}	10–20 PPM	20 µW	20 g
TXCO	10^{-6}	2–5 PPM	100 µW	50 g
OXCO	10^{-8}	$2 \cdot 10^{-8}$–$2 \cdot 10^{-7}$	1–3 W	200–500 g
Rb std.	10^{-9}	$2 \cdot 10^{-10}$–$2 \cdot 10^{-9}$	6–12 W	1500–2500 g
GPSDO	10^{-8}–10^{-11}	10^{-13}	0.5–4 W	100–500 g

Table 3: This table compares some metrics of different oscillator types. Source: The Meinberg GPSDO product page `http://www.meinberg.de/english/specs/gpsopt.htm` and the "Precision Frequency Generation" tutorial [13]

5.4 Evaluation of the suitability of GPS receivers

On GPS receiver that seemed quite interesting for this application is the u-Blox LEA-6T. [18]

This GPS receiver can drive two timepulse outputs, whose timing and frequency can be configured. One timepulse can be used as a one-pulse-per-second output, such that the rising edge is aligned to the GPS time. The other can be configured to output a 10 MHz square wave, whose frequency is controlled by the GPS.

Compared to a GPSDO with an OXCO, the performance will be worse, especially in the case the GPS signal is lost. The oscillator in the LEA-6T is not designed for such applications, because it offers a limited short-time stability. However, if this approach works, this receiver will be usable as a very cost-effective time source for transmitter synchronisation.

To evaluate it, I have designed a small printed circuit board, which is described in appendix B. The board was manufactured at EPFL, and soldered by hand. Using the "u-center" evaluation software,[19] it is possible to control the correct function of the receiver and to configure the timepulses.

Using the LeCroy WaveJet 354A oscilloscope, I have analysed the TIMEPULSE2 output configured at 10 MHz. From figure 21, it is immediately clear that an important

[17] `http://www.jrmiller.demon.co.uk/projects/ministd/frqstd0.htm`
[18] `http://www.u-blox.com/en/gps-modules/u-blox-6-timing-module/lea-6t.html`
[19] `http://www.u-blox.com/en/evaluation-tools-a-software/u-center/u-center.html`

amount of jitter is present on the signal. When this signal is used as reference clock for the USRP, the PLL IC is not able to lock properly (both USRP2 and USRP B100).

Figure 21: The TIMEPULSE2 signal of the U-Blox LEA-6T receiver configured at 10 MHz is subjected to considerable jitter.

Section 2.3.2 of the U-Blox timing application note [17] mentions that the TIMEPULSE outputs are derived from a 48 MHz master clock, which explains the jitter on 10 MHz since 10 does not divide 48. When configured to 8 MHz, the obvious jitter is not present, as seen on figure 22. More precise jitter measurements require specialised equipment. The same application note [17] also shows this behaviour in Section 3.2 Example 2.

The USRP B100 is not validated for a reference clock other than 10 MHz, but it is possible to modify the B100 to accept a 8 MHz reference. More precisely, the PLL configuration is entirely handled by the host code that interacts with the USRP firmware. In the UHD driver code, by changing the constant `REFERENCE_INPUT_RATE` in `host/lib/usrp/b100/clock_ctrl.cpp`, it is possible to tell the B100 that another reference clock frequency is used. However, the B100 is unable to get a lock on the reference, both when using 8 MHz and 6 MHz. It is unclear why this is the case, but the GPS receiver alone might not have enough short-time stability to be suitable for this.

The GPS receiver could also be used to synchronise computer time using the NMEA-0183 messages over the serial line. In this project, this is done using NTP.

Figure 22: The TIMEPULSE2 signal of the U-Blox LEA-6T receiver configured at 8 MHz shows no obvious jitter.

6 Laboratory Setup Used for Functional Verification

This part describes how the system was set up in the laboratory in the absence of a GPS-based time source.

6.1 Using signal generators as time-source

At the start of the project, in order to facilitate development of the SFN functionality, it was simpler to use signal generators as time sources than to find or develop a time source based on GPS or any other technique.

In the lab, the following devices were available:

- The lab desktop computer (Core 2 Duo, Windows XP);
- Two signal generators (HP 33120A, Agilent 33250A);
- A LeCroy WaveJet 354A oscilloscope;
- Coaxial cabling and adapters.

Because the desktop computer was only running Windows, and because the CRC mmbTools are made to run on Linux machines, I have brought two additional laptop computers to the lab: One IBM Thinkpad T43p, and one Lenovo Thinkpad T420, both running a recent linux distribution. Also, I have also brought a Gigabit Ethernet switch to interconnect the computers and the USRP2.

Two USRPs were available for the tests: one USRP2 and one USRP B100, both with WBX daughterboards. They were provided by the European Broadcasting Union.

In the first laboratory setup, I have used one generator to create a 1PPS signal and the USRP internal oscillators as frequency reference. The T420 was running CRC-DABMUX to create the ensemble, the Eti-TCP program to distribute the ensemble over TCP and CRC-DABMOD to modulate for the USRP2. The T43p was only running CRC-DABMOD for the USRP B100.[20] Both modulators receive the ensemble over TCP/IP. The USRP2 and the two laptops are connected through the Gigabit Ethernet switch. I have used the LeCroy WaveJet 354A oscilloscope to display the output RF signals from the two USRPs. This first setup is shown in figure 23.

This setup cannot be used for longer tests, because the internal oscillators of the USRPs will drift away from each other.

For this reason, I have added a second signal generator to generate the reference clock. It is set to 10 MHz, sine with 1 $V_{peak\text{-}peak}$ amplitude. That way, both USRPs have a single time-base. Figure 24 shows this setup.

This setup is problematic because the two signal generators were not synchronised, and the 1PPS is not in the 10 MHz clock domain from the other generator. To solve this problem, the generators were swapped because the Agilent generator has a sync input at 10 MHz. The HP generator then is used as reference clock for both USRPs and for the Agilent generator. With the lab setup in figure 25, it was possible for the first time to have a consistent synchronisation over several trials.

[20] The T43p contains a Pentium M processor that is not fast enough to resample the 2.048 Msps stream, which is necessary for the USRP2. It could only be used to modulate using a USRP B100, thanks to the flexible master clock configuration it offers.

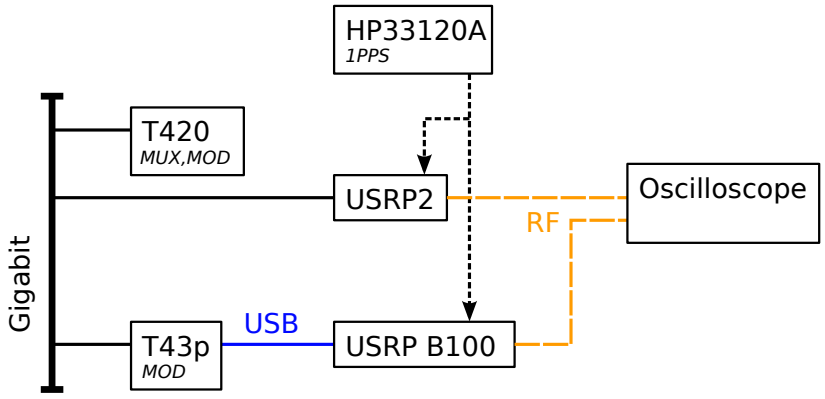

Figure 23: First lab setup with both USRPs

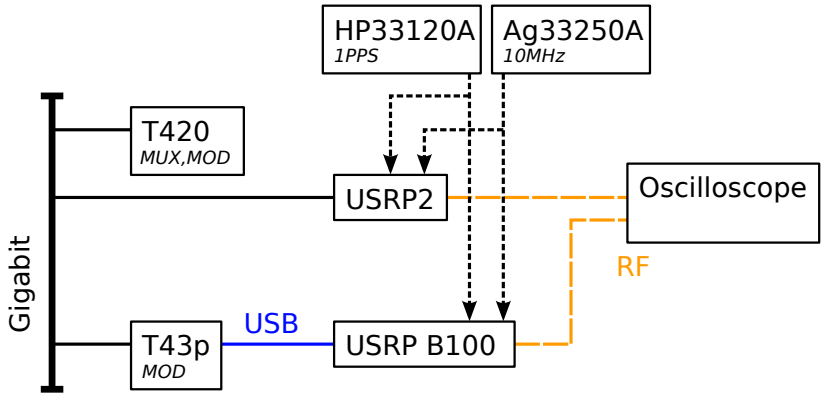

Figure 24: The second signal generator is used as REFCLK for both USRPs.

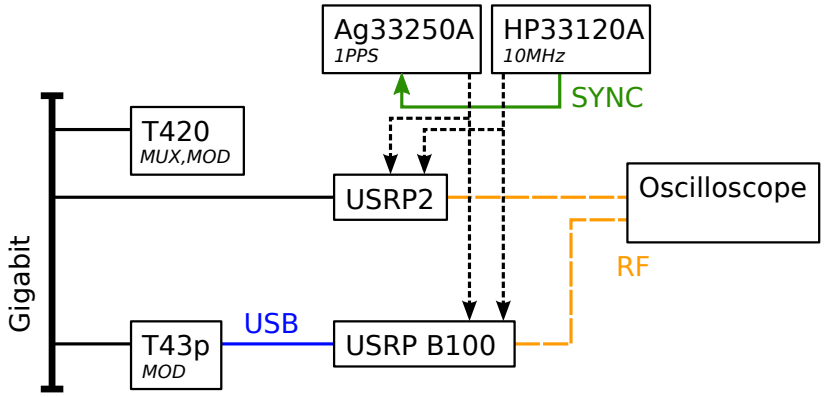

Figure 25: With the two signal generators synchronised, the 1PPS signal is always latched correctly.

6.2 Instantiation and communication between multiplexer and modulators

As explained above, a TCP/IP network is used between the multiplexer and the modulators. This section shows how these elements must be launched, when using the last laboratory setup above. It is assumed that the clocks of all computers are synchronised using NTP.

First, the CRC-DABMUX must be started to create the multiplex. Assuming that the current working directory contains the audio files and the Eti-TCP server, the following invocation will start the multiplexer:

```
CRC-DabMux -L "TuxMux" -s -r \
    -A funk.mp2 -b 128 -i 10 -S -L "Funk" -C \
    -A notfunk.mp2 -b 128 -i 3 -S -L "Not Funk" -C \
    -O fifo:///dev/stdout?type=raw \
    | ./eti_tcp.py 54001
```

This creates a multiplex composed of two programmes “Funk” and “Not Funk”, whose content is read from the corresponding `.mp2` files. The raw ETI(NI) output is piped into the Eti-TCP server, that will listen on the arbitrarily chosen TCP port 54001. The `-s` flag is required to set the TIST field, and the `-r` flag throttles multiplexing to the nominal rate of one frame every 24 ms. In the lab setup, this invocation is done on the T420 PC. We assume that the T420 possesses the IP address 192.168.0.2.

Now, the modulators must be started. If we follow the lab setup, there will be one modulator on the T420, and one on the T43p. The netcat tool is used to receive the ETI data, which is piped into CRC-DABMOD. For a USRP B100 that has a flexible master clock, we execute:

```
nc 192.168.0.2 54001 | crc-dabmod /dev/stdin -g2 -O ./modulatoroffset \
    -u "master_clock_rate=32768000,type=b100" \
    -F 234208000
```

In this example, the file-based offset setting is enabled. The file `./modulatoroffset` contains a value that is slightly longer that the network delay, plus a margin of about 1 s. In a local ethernet network, a value of 2 s works well. The `-u` option is given to UHD to configure the USRP B100 clock. The carrier frequency is chosen by the `-F` option. 234.208 MHz corresponds to DAB channel 13A. The `-g2` option selects transmission frame gain mode as described on opendigitalradio.org.[21]

When using a USRP2, the invocation is slightly different, because the master clock rate cannot be changed. We must enable resampling using the `-r` parameter.

```
nc 192.168.0.2 54001 | crc-dabmod /dev/stdin -g2 -O ./modulatoroffset \
    -r 4000000 -u "type=usrp2" -F 234208000
```

6.3 Results

Using the described setup with synchronous modulation, it is possible to see DAB transmission frame synchronisation on the oscilloscope, as shown in figures 26, 27 and 28. The relative delay between the two transmitters is less than 50 ns.

The transmission has been tested with several receivers:

[21] `http://opendigitalradio.org/index.php/CRC-DabMod`

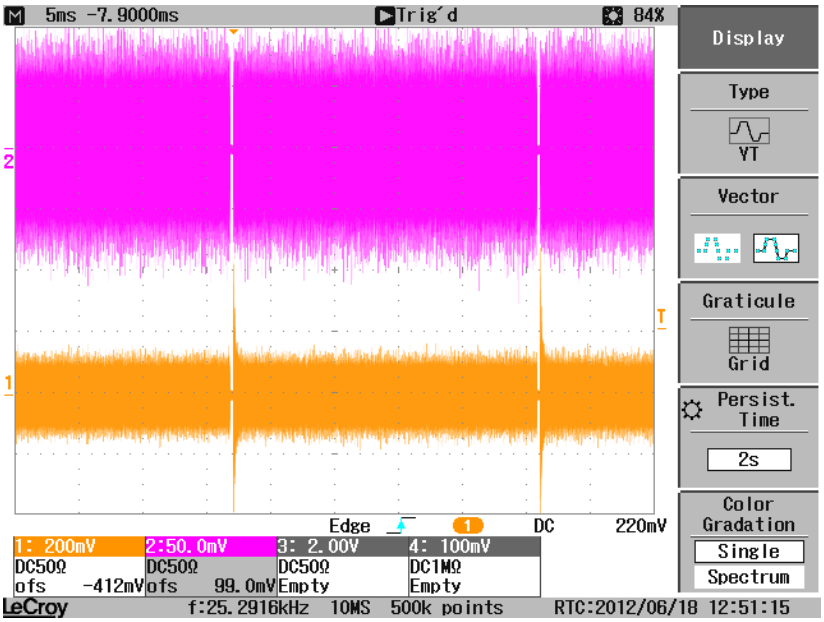

Figure 26: The oscilloscope shows a full frame for each of the two signals from the two USRPs.

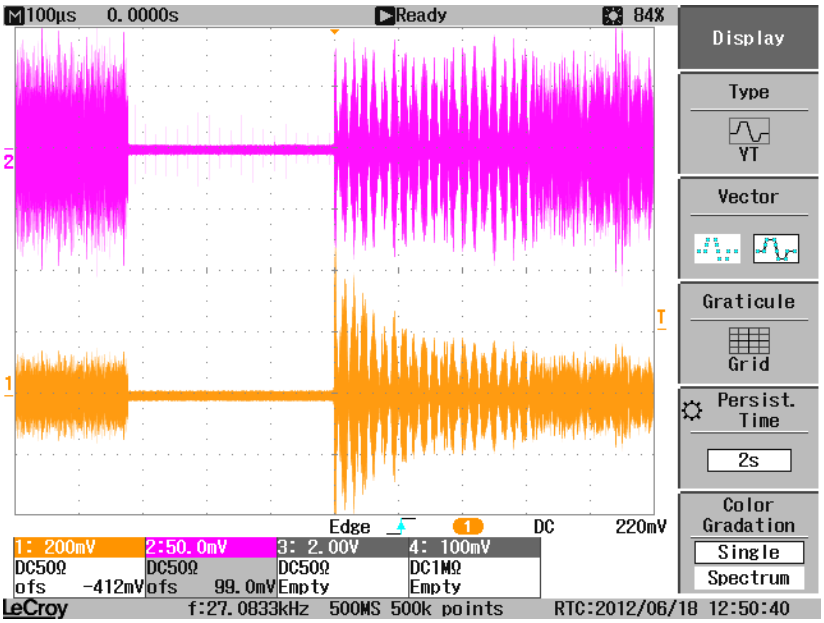

Figure 27: The null symbol and the phase synchronisation symbol are visible at the start of the transmission frame.

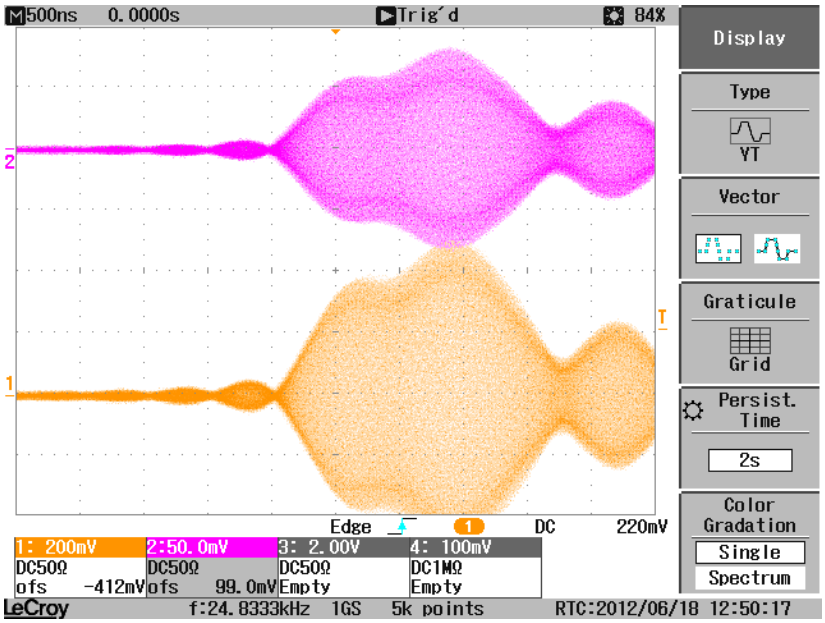

Figure 28: At a shorter time-span, the oscilloscope shows the synchronisation of the beginning of the phase synchronisation symbol.

- The United DAB receiver, that contains a Frontier Silicon Venice 7[22] Receiver;
- A portable Dual DAB4 receiver;
- A Enspert Identity E201 android tablet with a special application that shows bit-error rate, signal-to-noise ratio and data error correction information. It only works in TM 1.

All three receivers are able to decode the combined signal from the two synchronous USRPs, and do not experience glitches when one receiver is turned off. The single-frequency network thus created by the two USRPs satisfies the desired timing constraints.

In a more realistic setup, in which several transmitter locations are used, the transmitters will not share a common reference clock source. The 1PPS signals will also be generated independently by two GPS receivers. In the absence of a common clock source, the performance will likely be worse, since the generated reference clocks will not necessarily be in phase. If they are 180° out of phase, the additional delay incurred will be 50 ns. Furthermore, there might be an offset of up to 200 ns (according to datasheet) between the two 1PPS signals. Considering these factors, the total delay will still be below 1 µs, which is in the specification. How this holds up in a real setup has to be tested once GPS disciplined oscillators are available.

[22]Sadly, Frontier Silicon do not publish any data-sheets about their products, the only information that is accessible is on their website `http://www.frontier-silicon.com/products/modules/venice7.htm`

7 Further work

7.1 Evaluating other ETI transport protocols

While transporting the ETI(NI) data over TCP works well, it is no the best solution. There are other protocols that are more flexible, and that allow the user to choose what guarantees are really necessary. SCTP for instance can be used to transmit messages in contrast to TCP that transmits a stream, and similar to UDP that transports datagrams. However, it's behaviour for lost packets can be configured. The article "SCTP: What, Why, and How" by Natarajan et al. [16] gives an introduction of the features this protocol offers. Other protocols can also be evaluated for the purpose of transporting ETI data.

7.2 Porting the modulator to an embedded platform

The idea of having a small box that acts as a self-contained DAB modulator is very appealing. It should be possible to get rid of the PC running CRC-DABMOD at each transmitter site by using an embedded system instead. The USRP E100 is a model that combines an USRP with a ARM-based embedded single-board computer. It also contains a Xilinx Spartan 3A-DSP 1800 FPGA, with most logic elements unused by the default UHD design.

This platform could be an interesting starting point for further developments. The way CRC-DABMOD is built (using a flow-graph) makes it quite simple to take some processing blocks and move them to dedicated logic in the FPGA.

7.3 Adding Transmitter Identification Information support to CRC-DABMOD

In a DAB transmitter network, it is difficult to distinguish the contributions from each transmitter if they do not have an unambiguous distinction. The Transmitter Identification Information, described in section 14.8 of the main DAB standard [10] defines how such a unique signal is created. Adding this to CRC-DABMOD would ease the creation and maintenance of a single-frequency network.

7.4 CRC-DABMUX multiplex reconfiguration

All options and parameters necessary to describe the DAB ensemble are given as command line arguments to CRC-DABMUX. When only one transmitter is used, it is a light annoyance to have to cut transmission when we want to change the ensemble. However, when several transmitters is used, it becomes much more delicate to shut all modulators off and turn them on again. We would like to keep the modulators running, and be able to reconfigure the ensemble without interruption, as described by the standard (Annex E in ETS 300 799 [5]).

To add this feature to CRC-DABMUX, it would be necessary to represent ensemble configuration in another format–using a configuration file or database would seem appropriate–and to implement the reconfiguration protocol.

This would also simplify the delay handling for each modulator, that could then be communicated using the MNSC.

7.5 Monitoring of CRC-DABMOD

In its present state, there is no feedback monitoring channel from the modulators to the network operator. It is impossible to get any health information from the modulators.

However, such information is crucial for a network operator, otherwise management and fault handling are very difficult if not impossible. What kind of information has to be monitored, and how this information should be communicated has to be studied.

8 Conclusion

Thanks to the synchronised transmission that I have implemented in this project, it is now possible to create single-frequency networks using the open-source mmbTools, making new ambitious projects possible. The first project taking advantage of this new feature will be a temporary broadcast during the event "Rencontres Mondiales du Logiciel Libre"[23] in which two transmitters will be used in this first real-world test of my modifications. This temporary transmission will certainly also bring new insights about the whole system, because it will use real GPS disciplined oscillators.

Since the u-Blox receiver can unfortunately not be used as time reference alone for the USRPs, used GPSDOs from old cellular base-stations, available on an internet auction site will be used. The u-blox receiver board can however still be used for other projects, also in combination with the FPGA4U[24] board thanks to the compatible serial UART connector.

I hope that these tools will be used for many projects, and am confident that their open-source nature will enable numerous improvements.

[23] http://2012.rmll.info
[24] http://fpga4u.epfl.ch

9 Acknowledgments

Working on this project has been highly interesting from the very beginning until the end. This work has been made possible by the great support I have received from all involved people.

I would first like to thank Pascal Charest and François Lefebvre from the Communications Research Centre Canada for the development of the CRC mmbTools, and for having published them under an open-source license. Their work has enabled many small broadcasters to create DAB transmissions, giving them the opportunity to participate in the switch to digital radio broadcasting. Furthermore, I have had the chance to be in contact with them during these developments. Thanks to this collaboration, the SFN functionality has also been implemented into the new, not-yet published version of the modulator that includes support for all transmission modes.

Also, my thanks go to Stan Röhrich and Mathias Coinchon (European Broadcasting Union), who have worked on the mmbTools, and have created the OpenDigitalRadio.org project. Their support has been very valuable to this project, and the two USRP platforms the EBU has lent me were very useful.

Many thanks also go to René Beuchat (LAP) for his advice. He always asks the right questions, those who require you to look at a specific problem from another perspective and allow you to find another way to the solution. I would also like to express my gratitude to all the people from the Processor Architecture Laboratory at EPFL, for the many projects I could participate in.

Finally, the motivation for doing this project also originates from my activity at Fréquence Banane, the student radio station of EPFL and Unil, that allowed me to step into the world of radio broadcasting.

A Software versions and equipment

I have used the following development platforms in this project:

Lenovo ThinkPad T420

- Gentoo Linux base system;
- GCC 4.5.6 with glibc 2.14.1;
- Boost libraries 1.48.0;
- UHD library version 003.004.001-129-g23344268;
- Python 2.7.3;
- Linux kernel 3.2.12.

IBM ThinkPad T43p

- Arch Linux base system;
- GCC 4.6.3 with glibc 2.15;
- Boost libraries 1.49.0;
- UHD library version 003.004.000-1-gea19de0b;
- Python 2.7.3;
- Linux kernel 3.3.7.

B u-blox LEA-6T GPSDO design

The printed circuit board used to evaluate the u-blox LEA-6T GPSDO[25] has been designed with Altium.

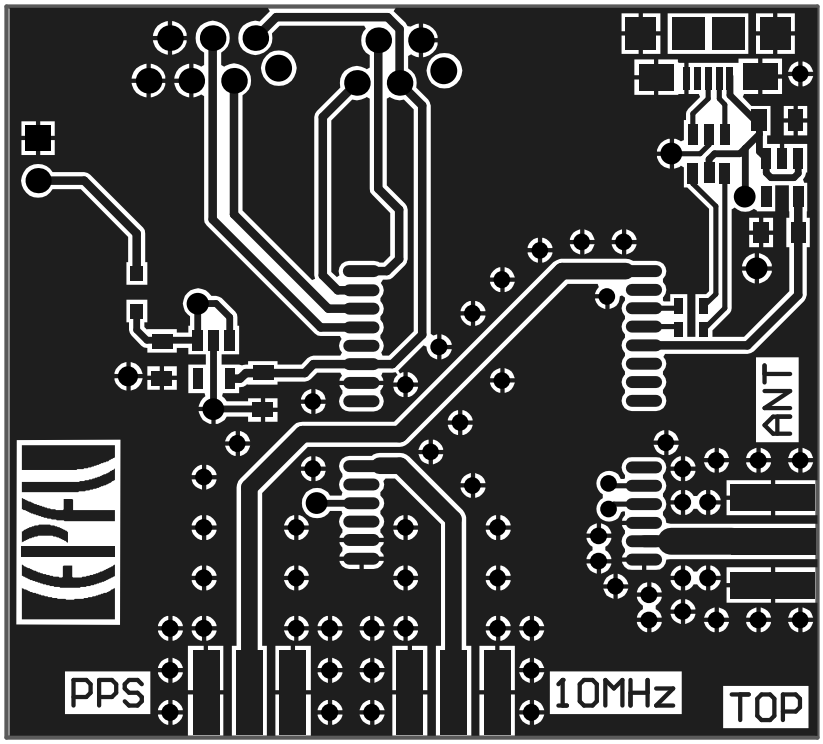

Figure 29: The TOP layout sent to production.

The PCB has been fabricated at the EPFL-ACI, and has been assembled by hand. Figures 31 and 32 show the completed receiver board.

[25] http://www.u-blox.com/en/gps-modules/u-blox-6-timing-module/lea-6t.html

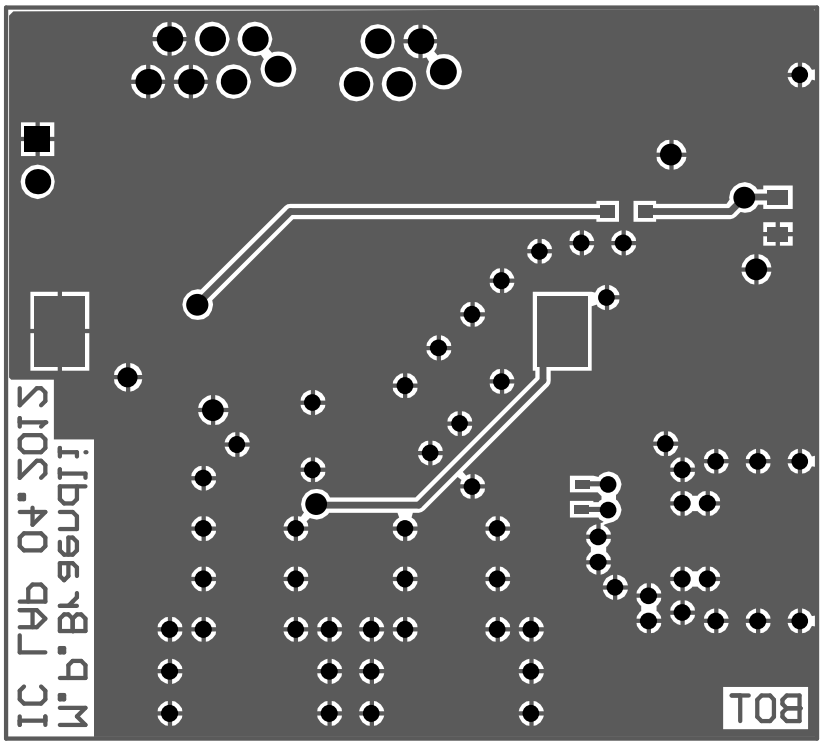

Figure 30: The BOT layout sent to production.

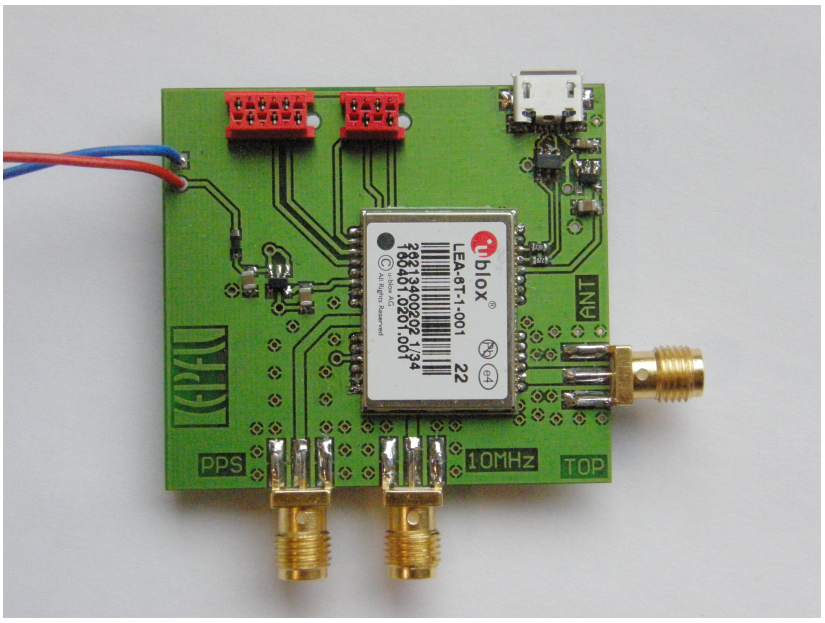

Figure 31: The u-blox LEA-6T receiver board seen from the top. On the upper half, from left to right: External 5V power, UART connection, I^2C connection, USB 2.0 device; The SMA connectors: 1PPS output, configurable clock output, and GPS antenna input.

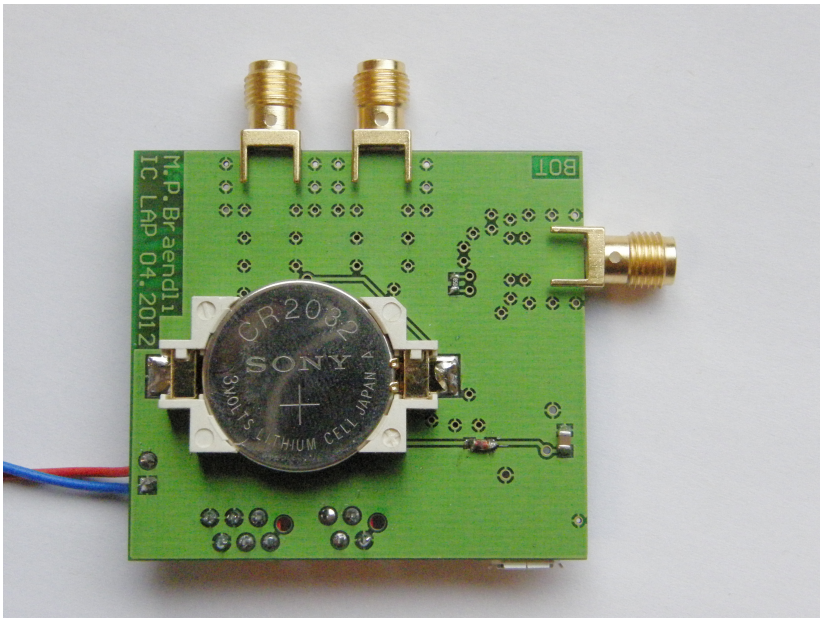

Figure 32: The bottom side of the receiver board holds the backup battery.

C CRC-DABMUX ETI file formats

CRC-DABMUX supports three output formats for the ETI stream, that have been described on the mmbTools forum website.[26]

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`.

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

References

[1] Anritsu. *Understanding Frequency Accuracy in Crystal Controlled Instruments*, March 2001. Application Note. Available at http://www.anritsu.com/en-GB/Downloads/Application-Notes/Application-Note/DWL2308.aspx.

[2] Barrett, C. *Fractional/Integer-N PLL Basics, Texas Instruments technical brief*. http://www.ti.com/lit/an/swra029/swra029.pdf.

[3] Baset, S. A., Brosh, E., Misra, V., Rubenstein, D., and Schulzrinne, H. Understanding the Behavior of TCP for Real-time CBR Workloads. *Proceedings of the 2006 ACM CoNEXT conference* (2006). http://dx.doi.org/10.1145/1368436.1368502.

[4] Charest, P. *CRC-DABMUX man page*. http://opendigitalradio.org/index.php/CRC-DabMux.

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

[6] 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.

[7] ETSI. *TS 102 427, Digital Audio Broadcasting (DAB); Data Broadcasting – MPEG-2 TS streaming*, July 2005. V1.1.1.

[8] ETSI. *TS 102 428, Digital Audio Broadcasting (DAB); DMB video service; User Application Specification*, June 2005. V1.1.1.

[9] ETSI. *TS 102 821, Digital Radio Mondiale (DRM); Distribution and Communications Protocol (DCP)*, October 2005. V1.2.1.

[10] ETSI. *EN 300 401, Digital Audio Broadcasting (DAB) to mobile, portable and fixed receivers*, June 2006. V1.4.1.

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

[12] ETSI. *TS 102 563, Digital Audio Broadcasting (DAB); Transport of Advanced Audio Coding (AAC) audio*, May 2010. V1.2.1.

[13] Frequency Electronics, Inc. *Tutorial Precision Frequency Generation Utilizing OCXO and Rubidium Atomic Standards with Applications for Commercial, Space, Military, and Challenging Environments*, March 2004. http://www.ieee.li/pdf/viewgraphs/precision_frequency_generation.pdf.

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

[15] Nagle, J. *Request For Comments 896, Congestion Control in IP/TCP Internetworks*, January 1984. http://tools.ietf.org/html/rfc896.

[16] Natarajan, P., Baker, F., Amer, P. D., and Leighton, J. T. Sctp: What, why, and how. *Internet Computing, IEEE 13*, 5 (sept.-oct. 2009), 81–85. http://dx.doi.org/10.1109/MIC.2009.114.

[17] U-Blox. *GPS-based Timing Considerations with u-blox 6 GPS receivers, Application Note*. http://www.u-blox.com/images/downloads/Product_Docs/Timing_AppNote_%28GPS.G6-X-11007%29.pdf.