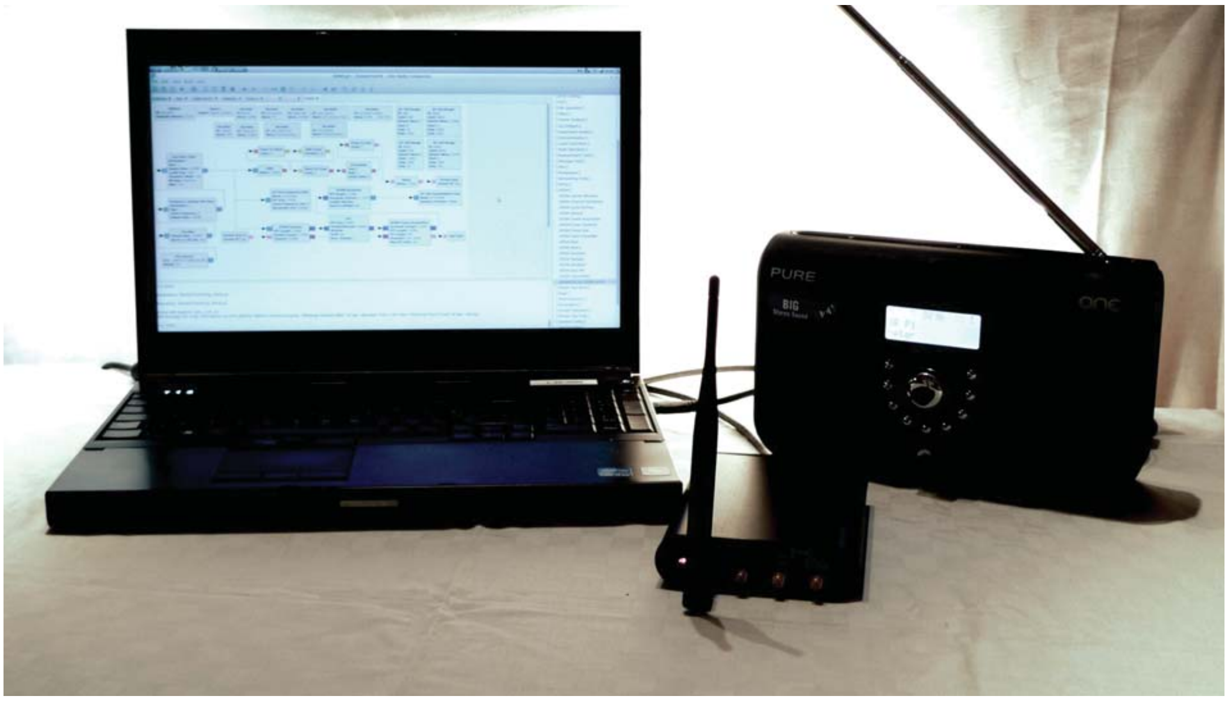
CHALMERS



DAB radio design and implementation

FREDRIK FORSBERG

Department of Signals and systems
Chalmers University of Technology
Göteborg, Sweden 2015

DAB radio design and implementation

FREDRIK FORSBERG

Department of Signals and systems
Chalmers University of Technology
Göteborg, Sweden 2015

DAB radio design and implementation
FREDRIK FORSBERG

Department of Signals and systems
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Cover:
Picture showing the set-up used during the verification of the DAB baseband signal (chapter 5.3) and for testing the interference injector (chapter 5.15).

Department of Signals and systems
Göteborg, Sweden 2015

Acknowledgements

This degree project was done in cooperation with Chalmers, Altran and Volvo PV.

I would like to express my gratitude to all people who supported me in this degree project.

I would like to offer my special thanks to:

- Kimmo Luomala at Altran for coming up with the idea for this project and for being my contact person at Volvo.
- Thomas Andersson at Altran for supporting the idea and for providing me with hardware.
- Dick Telder and Björn Bergqvist at Volvo for technical support.
- Göran Hult at Chalmers for being my supervisor and for giving me feedback.
- Manne Stenberg at Chalmers for being my examiner.
- Arto Heikkilä at Chalmers for helping me with some of the administrative work.

Abstract

This project explores the concept SDR (Software Defined Radio) using the free software GNU-radio and the graphical environment GRC (GNU Radio Companion). One goal was to evaluate the potential of GNU-radio and GRC in combination with the RF-hardware Ettus USRP B210 to find out what can easily be made and what cannot. Another goal was at the beginning to implement a DAB (Digital Audio Broadcasting) receiver in GNU-radio. Due to the complexity of a DAB receiver and the fact that this was my first contact with SDR, the project was limited to the use of the standard function blocks that comes with the GNU-radio package. A number of experiments resulted in list of blocks that can be used and blocks that are missing. As a good side effect of all experiments a useful RF-recorder / player was created.

The main purpose with implementing a DAB receiver in SDR is to achieve an affordable and flexible tool to examine how different interferences affects the different internal data streams and finally the sound reproduction in a DAB system. For this reason, creation and injection of interferences into a DAB-signal using SDR is also covered in this project.

The DAB system uses COFDM (Coded Orthogonal Frequency Division Multiplex), a modulation technology that has increased in popularity over the last years. This report aims to make it easier to understand the fundamentals of this technology.

Most of the practical work were done at Volvo PV Torslanda, Gothenburg.

Keywords: DAB, radio, SDR, Software Defined Radio, Ettus, USRP B210, GNU-radio, GRC, implementation

Table of contents

Glossary

ADC – Analogue Digital Converter
C/N – Current / Next
CA – Conditional Access
CIF – Common Interleaved Frame
COFDM – Coded Orthogonal Frequency Division Multiplex
CRC – Cyclic Redundancy Check
CU – Capacity Unit
DAB – Digital Audio Broadcasting
DAC – Digital Analogue Converter
DSP – Digital Signal Processing
EEP – Equal Error Protection
ETSI - European Telecommunications Standards Institute
FEC – Forward Error Correction
FFT – Fast Fourier Transform
FIB – Fast Information Block
FIC – Fast Information Channel
FIDC – Fast Information Data Channel
FIG – Fast Information Group
FIR – Finite Impulse Response
FM – Frequency Modulation
FPGA – Field Programmable Grid Array
GNU – GNU's Not Unix!
GPIO – General Purpose Input Output
GRC – Gnu Radio Companion
GUI – Graphical User Interface
IFFT – Inverse Fast Fourier Transform
ISI – Inter-Symbol Interference
LNA – Low Noise Amplifier
MCI – Multiplex Configuration Information
MPEG – Moving Pictures Expert Group
MSC – Main Service Channel
OE – Other Ensemble
OFDM – Orthogonal Frequency Division Multiplex
PAD – Programme Associated Data
PRBS - Pseudo-Random Binary Sequence
PRS – Phase Reference Symbol
PWM – Pulse Width Modulation
RF – Radio Frequency
RX - Reception
SDR – Software Defined Radio
SNR – Signal Noise Ratio
SI – Service Information
TII – Transmitter Identification Information
TX – Transmission
UEP – Unequal Error Protection
UHD – USRP Hardware Driver
USB – Universal Serial Bus
USRP – Universal Software Radio Peripheral

1. Introduction

1.1. Background

The trend of today is that more and more countries change their radio broadcasts to the DAB (Digital Audio Broadcasting) standard [8](516). The main reason is that DAB is superior to FM when it comes to the use of bandwidth. There are also other advantages like more data services, better sound quality (at high bit rates) and the possibility to use the same frequency at all transmitting sites. To meet these needs Volvo provides DAB receivers for their cars. The demands on a automotive DAB receiver is slightly higher than for domestic ones. They must work satisfyingly when the car moves and all challenges that comes with that. The signal strength and quality will change all the time due to changes in distance to the transmitter, doppler shift, new multipath situations etc. Interferences from different sources will also change under movement and also the car itself can produce interferences. The electrical system in vehicles becomes more and more complex. One specific source of interference is the PWM (Pulse Width Modulation) controllers in electric / hybrid vehicles where high currents are switched on and off rapidly creating a lot of harmonics. To optimize the DAB receiver for automotive use and to develop the vehicle in a DAB-friendly way more tools are needed. Anechoic chambers and spectrum analysers are great resources for these purposes but what happens inside the DAB decoder chip set is still a bit of a black hole. The old analogue FM systems give early warnings like increased noise, switching to mono etc. In a DAB receiver the first warning is that the audio signal becomes unusable because of cut outs, skipping and high pitched noises. With a SDR (Software Defined Radio) model of a DAB receiver with free access to all decoding stages it will be possible to monitor error rates, signal levels, phase errors and more to be aware of the margins to that stage where the audio signal becomes unusable. There are equipment available today for these purposes but the devices are very expensive and not very configurable. Another advantage is that SDR devices are very generic. They can be used in almost any application as long as the bandwidth is sufficient.

1.2. Purposes

- Investigate how much of a DAB-receiver that can be created using the graphical environment GNU-radio and the graphical environment GRC and the standard blocks that comes with the package.
- Evaluate GNU-radio, GRC and the hardware device Ettus USRP B210.
- Create a interference injector using GNU-radio and GRC.

1.3. Limitations

- The newer standard DAB+ is not covered.
- Only transmission mode 1 is treated.
- Creation of new function blocks is outside this project. All implementation shall be done using the standard GNU-radio blocks. One exception is the MPEG audio layer 2 decoder.

2. The DAB system

DAB (Digital Audio Broadcasting) is a radio standard for transmission of digital audio and data. The transmissions can be terrestrial, satellite or cable distributed [1](p164). The development of the DAB standard dates back to late 80:s and the first test transmissions took place in 1991 [8](p515). The DAB standard is specified by ETSI (European Telecommunications Standards Institute) in a number of documents that can be found at worlddab's home page. The documentation is vast and it may be hard to get a good overview. This chapter is intended as a quick overview for making it easy to understand the basics of DAB. All facts in chapter 2 can be found in [1] and [8].

2.1. System overview

A DAB transmission consists of a whole ensemble instead of a single station. This means that a single frequency carries many radio stations providing different services. DAB ensembles are transmitted in the VHF band I to V or the L-band. See annex A for VHF band III frequency table. A typical service provides audio and PAD (Programme Associated Data). With a rough comparison with FM transmissions PAD would correspond to programme related messages provided by RDS (Radio Data System). However, the service labels doesn't come with the PAD data. The audio is compressed using MPEG1 audio layer 2 resulting in audio frames which also contains the PAD. This is done separately for every service.

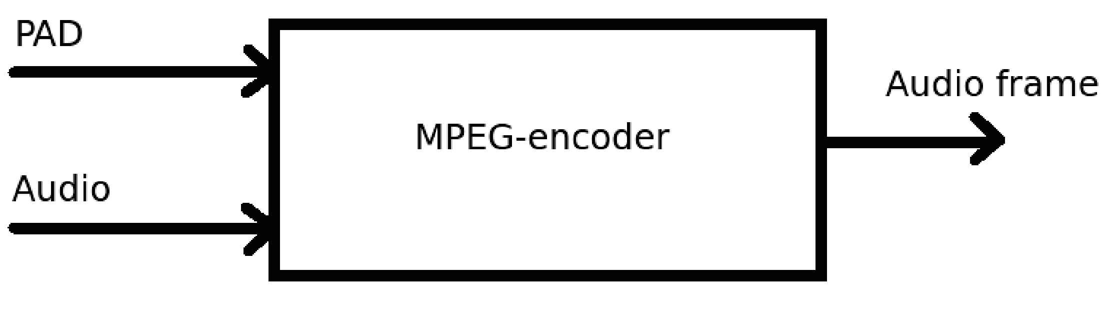


Figure 2.1.1: Audio frames are generated separately for each service [1](p22).

It is also possible to transmit general data services that goes outside the MPEG encoder using packet mode or stream mode [1](p22).

The audio frames, the packet data and the streamed data are then separately processed by an energy dispersal scrambler, a convolutional encoder and a time interleaver. The energy dispersal scrambler scrambles the data in a random way to avoid transmission of repeating patterns[1](p24).

The convolutional encoder adds redundancy to the signal for error protection. UEP (Unequal Error Protection) or EEP (Equal Error Protection) can be used. UEP means that important parts of the audio-frames (such as the header) gets more protection than less important parts (such as sub band samples).

The time interleaver shuffles data bits in the time dimension according to a pre-defined pattern. This adds protection against short interferences.

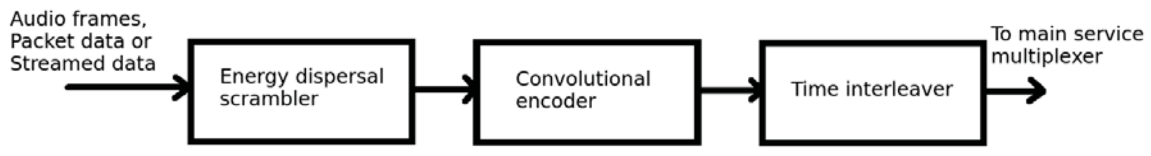

Audio frames, Packet data or Streamed data
Energy dispersal scrambler
Convolutional encoder
Time interleaver
To main service multiplexer

Figure 2.1.2: All signals are processed to add protection and robustness against interferences [1](p22).

The next step is to mix all services in the ensemble. This is done by a multiplexer (MUX) and the result is CIF:s (Common Interleaved Frames)[1](p22). The structure of the multiplexing varies depending on bit-rate and protection level for each service.

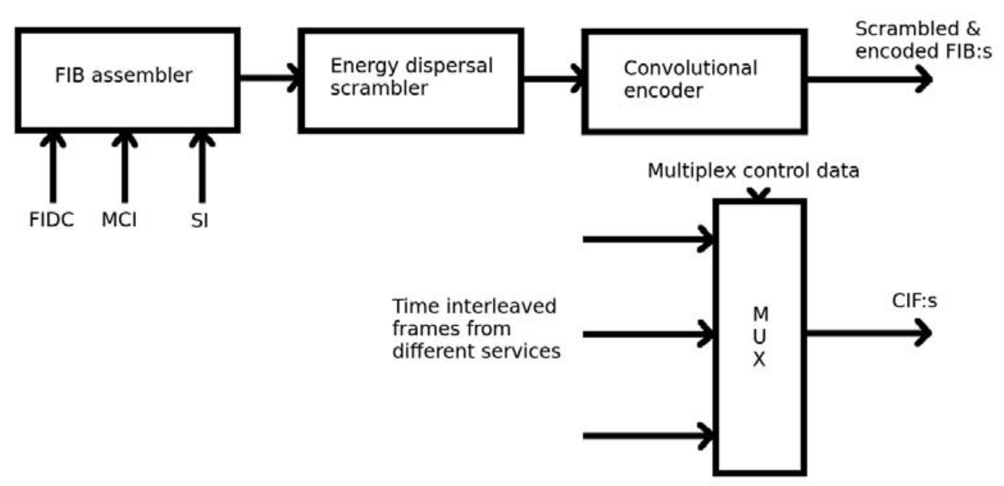

FIB assembler
Energy dispersal scrambler
Convolutional encoder
Scrambled & encoded FIB:s
FIDC
MCI
SI
Multiplex control data
Time interleaved frames from different services
M U X
CIF:s

Figure 2.1.3: All services are multiplexed [1](p22). All information about the MUX is provided by a FIC which is built up of FIB:s [1](p29).

When the ensemble is received the receiver needs to know the structure of the multiplexed data to be able to pick the right data for the selected service. For that reason, a FIC (Fast Information Channel) is generated[1](p21). The FIC contains service labels and the MCI (Multiplex Configuration Information). It is also possible to send data through the FIC using FIDC (Fast Information Data Channel). Typical usages are traffic and warning messages that are independent of the selected service. The FIC is built up of several FIB:s (Fast Information Blocks). The FIB:s are energy dispersal scrambled and convolutional encoded but not time interleaved[1](p22).

The CIF:s and the scrambled and convolutional encoded FIB:s are then put together to form transmission frames according to transmission mode. Serial data is converted to parallel data and a symbol generator generates sub-carriers modulated with π/4 D-QPSK (Differential Quadrature Phase-Shift Keying) (explained in 2.5).

The subcarriers are frequency interleaved which gives protection against narrow-banded interferences.

A synchronisation channel that consists of a null-frame and a PRS (Phase Reference Symbol) is added to complete the COFDM (Coded Orthogonal Frequency Multiplex) signal.

A TII-signal (Transmitter Identification Information) is added to the COFDM signal before it is up-converted to radio frequency, amplified and transmitted via antenna or cable.

A simplified block diagram showing the generation of a DAB ensemble can be found in annex B.

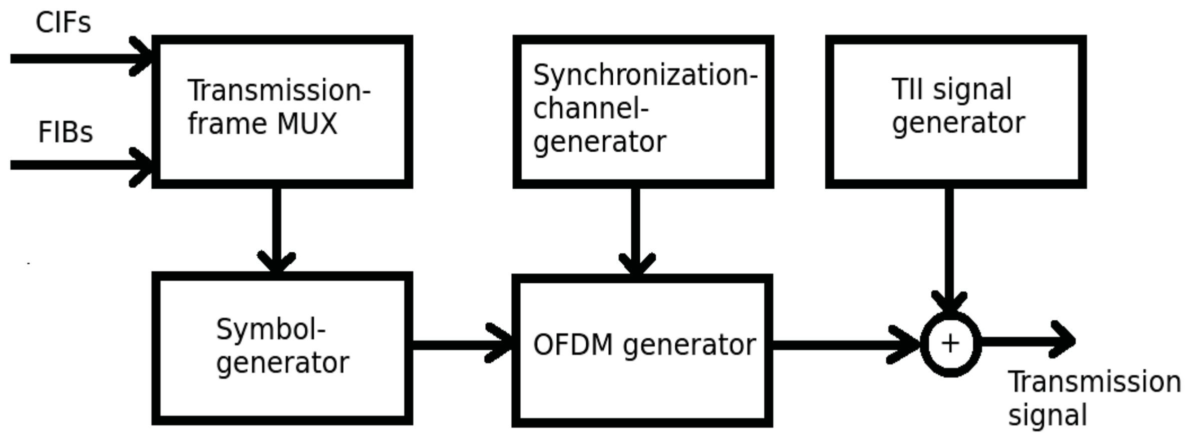


Figure 2.1.4: The generation of the transmission signal [1](p22). The transmission signal is then amplified and sent to the antenna (or cable).

2.2. The DAB transmission frame

The COFDM signal is divided into transmission frames which structure is shown in figure 2.2.1 [1](p144). The frame length varies depending on transmission mode. Transmission mode 1 uses a frame length of 96 ms which corresponds to 77 COFDM symbols including the null symbol.
The synchronization channel which is explained in chapter 2.5 is used for frame synchronization, phase reference and frequency correction.
The FIC (Fast Information Channel) which is used to carry configuration information and other information that is common for the whole ensemble is covered in chapter 2.3.
The Main Service Channel (MSC) transports data for all services and is covered in chapter 2.4.

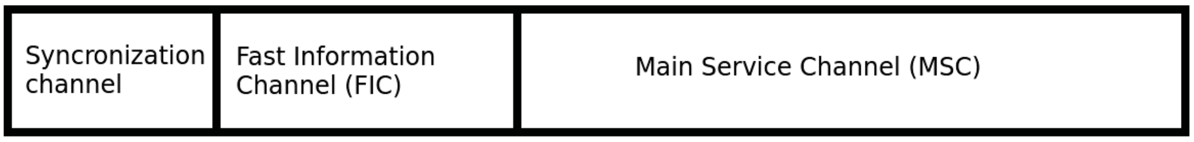


Figure 2.2.1: The structure of the DAB data frame [1](p28).

2.3. Fast Information Channel (FIC)

Instead of occupying one frequency for each radio station a number of stations are multiplexed and transmitted on a single frequency called an ensemble. The old way to select station was to change the received frequency. Transmitting several stations multiplexed at one frequency created two new needs:

- The user must be informed about all available stations / services on the ensemble.
- The user must be able to select one of these.

The solution is the use of a FIC (Fast Information Channel). The FIC contains data about all services on the ensemble [1]. The service labels shown in the display of the receiver are transmitted through the FIC together with the MCI used by the receiver to find the selected service in the MSC (Main Service Channel) [1](p29). The FIC contains a number of FIB:s (Fast Information Blocks). The number of FIB:s depends on transmission mode. DAB mode 1 uses 12 FIB:s per data frame. Every FIB is 32 byte long. The first 30 bytes consists of one or more FIG:s (Fast Information Group) and the last 2 bytes is used for CRC (Cyclic Redundancy Check) which is used for error detection. There are different FIG types for different purposes (see table).

FIG type	**Function**	**Explanation**
0 (000)	MCI and part of the SI	Carries the multiplex configuration
1 (001)	Labels	Service labels to be shown in the display of the receiver.
2 (010)	Labels (long)	Same as 1 but for longer labels that doesn't fit in one FIG.
3 (011)	Reserved	-
4 (100)	Reserved	-
5 (101)	FIDC	FIC Data channel
6 (110)	CA	Conditional Access
7 (111)	Reserved	Except for length 31 which is used for end marker

Table 2.3.1: The different FIG types and their functions [1](p30).

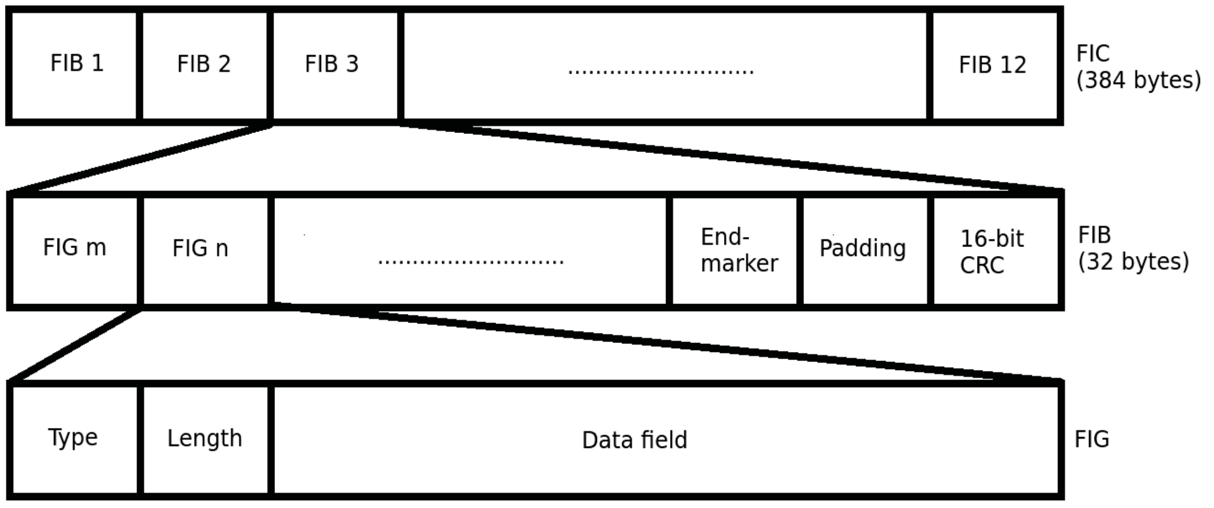


Figure 2.3.2: Structure of the FIC [1](p29).

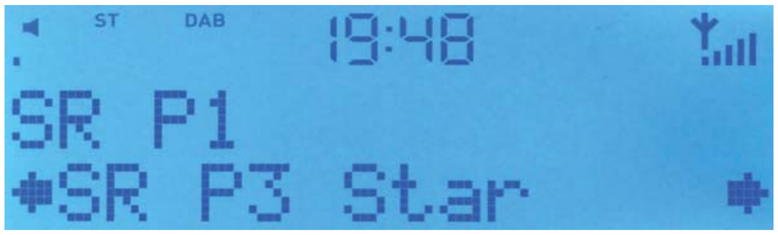


Figure 2.3.3: The service labels are provided by the FIG type 1 and 2 [1](p30).

It is possible to use different bit-rates and protection levels (explained in the error protection chapter) for sub-channels inside one ensemble.

A change in the sub-channel organisation is called a reconfiguration. If a reconfiguration is near by, this is announced in the FIG type 0 before the actual reconfiguration by sending the new configuration in combination with the flag C/N (Current / Next) set to 1.

2.4. Main Service Channel (MSC)

The MSC contains data for all services. The data is grouped in a number of CIF:s (Common Interleaved Frame). DAB mode 1 uses 4 CIF:s per data frame and each CIF contains 55 296 bits split into 864 CU:s (Capacity Unit)[1](p34). Data can be transmitted in packet mode or stream mode. Stream mode is used for transmission of signals without a obvious beginning or end such as audio. The data rate is always a fixed multiple of 8 kb/s. In stream mode the data is divided in logical frames.

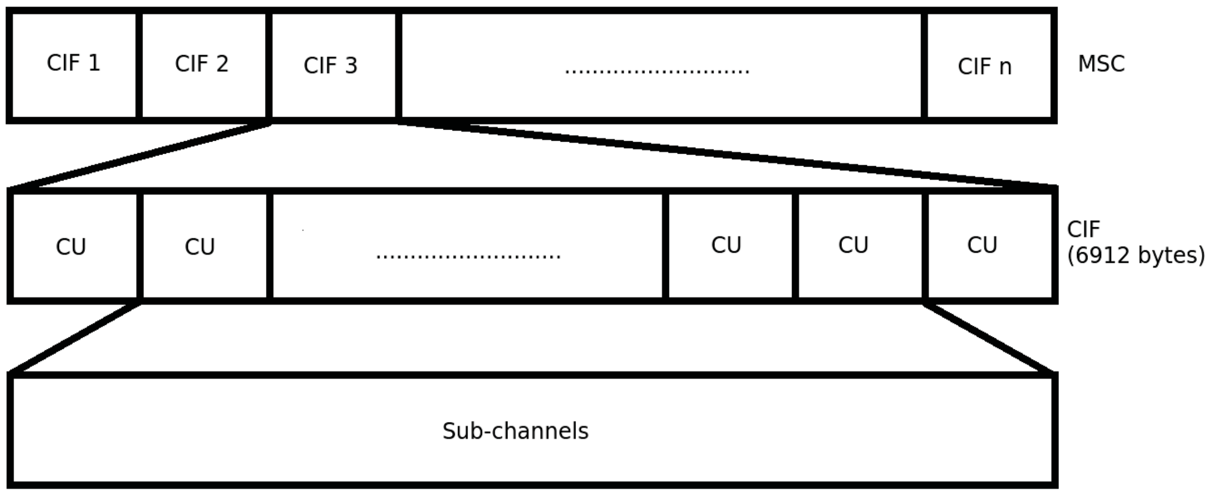


Figure 2.4.1: The MSC is divided into a number of CIF:s. Every CIF consists of 864 CU:s. The sub-channels occupies different number of CU:s depending on bit-rate, protection level etc...

2.5. Transmission signal

The data frames described in 2.2 are transmitted using COFDM (Coded Orthogonal Frequency Division Multiplex) [8](p515). The word “Coded” refers to the error protection used (convolutional coding for DAB). While most modulation techniques modulates one carrier COFDM uses a large number of sub-carriers, each modulated with a modulation scheme.

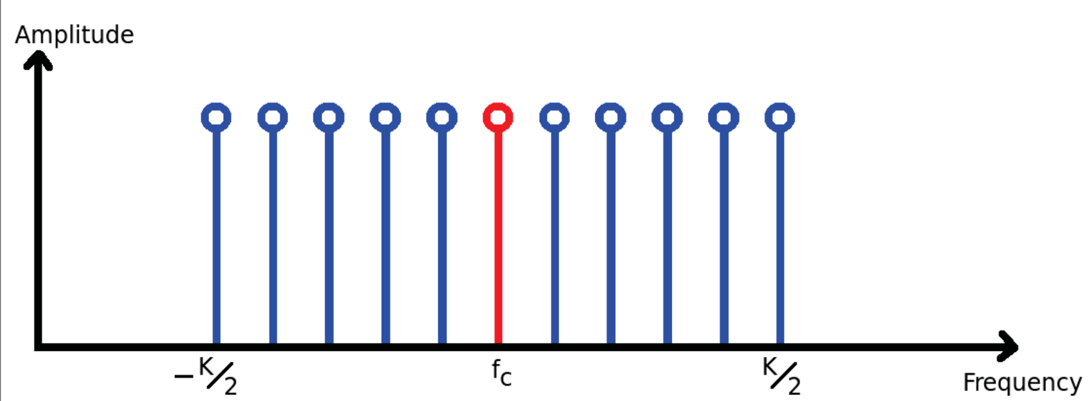


Figure 2.5.1: Graph showing the the sub-carriers in the frequency domain. The f_c is the centre frequency and K is the number of carriers. DAB mode 1 uses 1536 carriers, 768 on each side of f_c [1](p145, 148).

The main advantage with multiple carriers is that the guard area between each symbol can be much longer in the time domain compared with a single carrier system with the same data rate [8](p367). The reason is that each symbol contains more data. Large guard areas gives better protection against ISI (Inter-Symbol Interference) caused by multipath propagation, which is reflections of the same signal with a time-delay and attenuation.

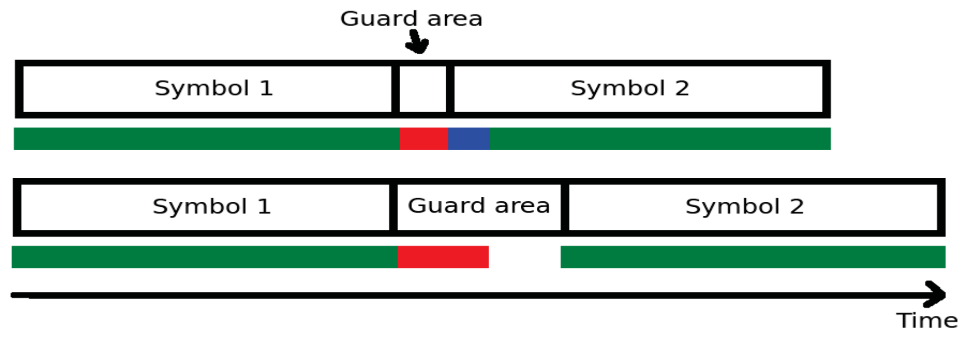


Figure 2.5.2: The larger guard area, the better protection against ISI. Green: Valid data. Red: ISI. Blue: Data corrupted by ISI [8](p362).

During each guard interval a cyclic prefix is transmitted. A cyclic prefix is a copy of the first samples of the previous symbol [8](p362). The cyclic prefix helps the receiver to detect the exact start of every symbol. It also helps the receiver to pick the right data within a symbol in case there are delays caused by multipath propagation.

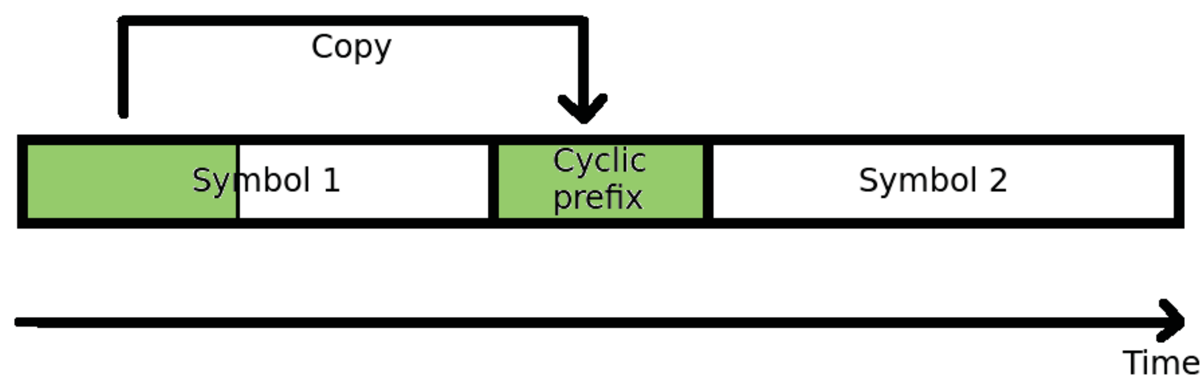


Figure 2.5.3: The cyclic prefix is a copy of the first samples of the previous symbol and is sent during the guard interval [8](p362).

Orthogonal refers to the mathematical term orthogonality. To save bandwidth, the sub-carriers must be as close to each other as possible without interfering each other. The minimum carrier-spacing is where the carriers are orthogonal to each other. Every modulated sub-carrier produces side-lobes. There are null-points between the side-lobes and orthogonality in this case is when other sub-carriers are placed in the null-points [8](p354).

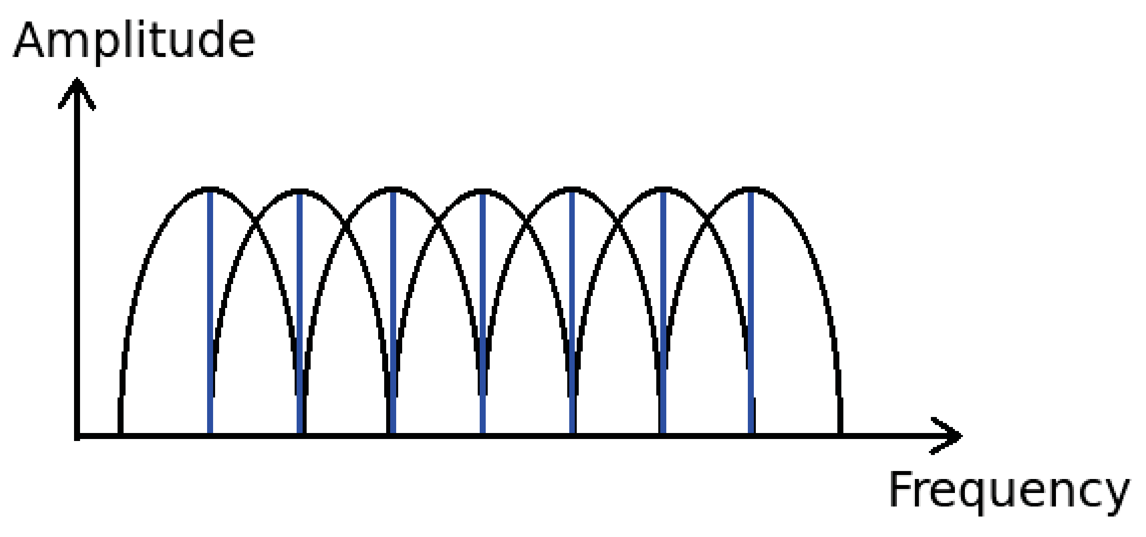


Figure 2.5.4: An orthogonal arrangement of the sub-carriers optimizes the use of bandwidth. The spacing is chosen so every sub-carrier (blue) is placed at null-points.

DAB uses π/4 D-QPSK (Differential Quadrature Phase-Shift Keying) as sub-modulation[8](517). QPSK uses four different phase-angles which represents four different numbers (2 bits each) [2] (p118). I and Q stands for in phase and quadrature.

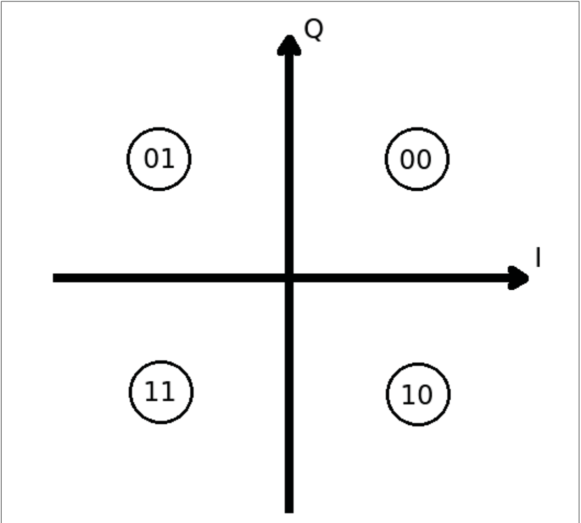


Figure 2.5.5: Each sub-carrier is modulated with a QPSK-scheme, and contributes with 2 bits / symbol.

Differential modulation means in this case that the complex number that is represented by each sub-carrier is multiplied with the same sub-carrier index from the previous symbol [1](p161)(the arguments are added). Since the constellation points are placed in 45°, 135°, 225° and 315° the constellation graph will shift 45° every symbol which is equal to π/4 in radians. The second symbol in the synchronization channel is called PRS (Phase Reference Symbol). One purpose with the PRS is to act as a phase reference for the first symbol that carries data. The advantage with differential modulation is that phase errors that occurs during transmission doesn't affect the transmitted data.

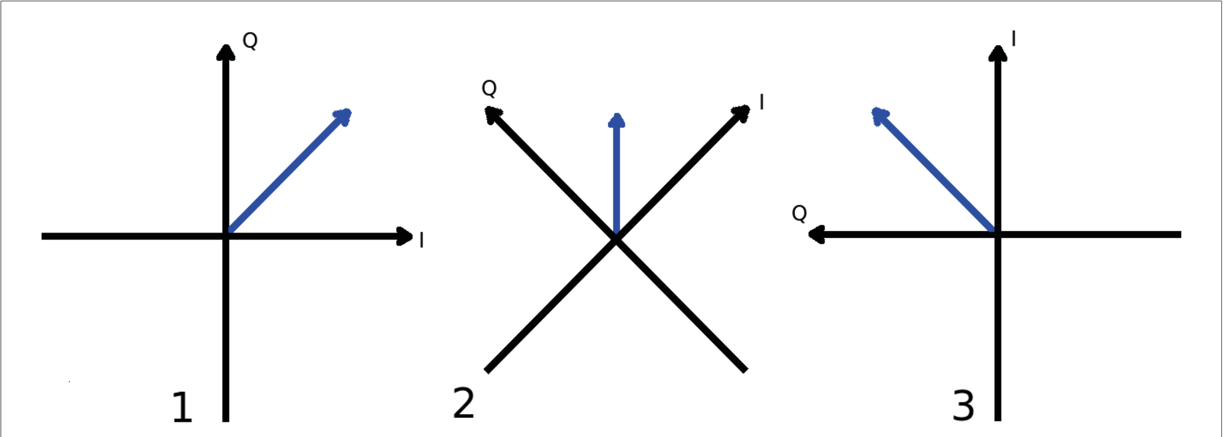


Figure 2.5.6: Phasors showing one sub-carrier transmitting the phase 45° tree symbols in row. Using differential modulation the previous symbols value is used as phase reference.

A DAB mode 1 transmission frame consists of 77 symbols including the null-symbol [1](p145). The null-symbol indicates the beginning of the frame by setting the amplitude of all carriers to zero [8](p528). However, a few sub-carriers can be used for transmitter identification during the null-symbol [8](p533). The next symbol is the PRS which, as mentioned before, acts as phase reference for the first data carrying symbol. Figure 2.5.7 visualizes how the symbols are arranged in a transmission frame. Every column represents a symbol according to figure 2.5.1 seen from above.

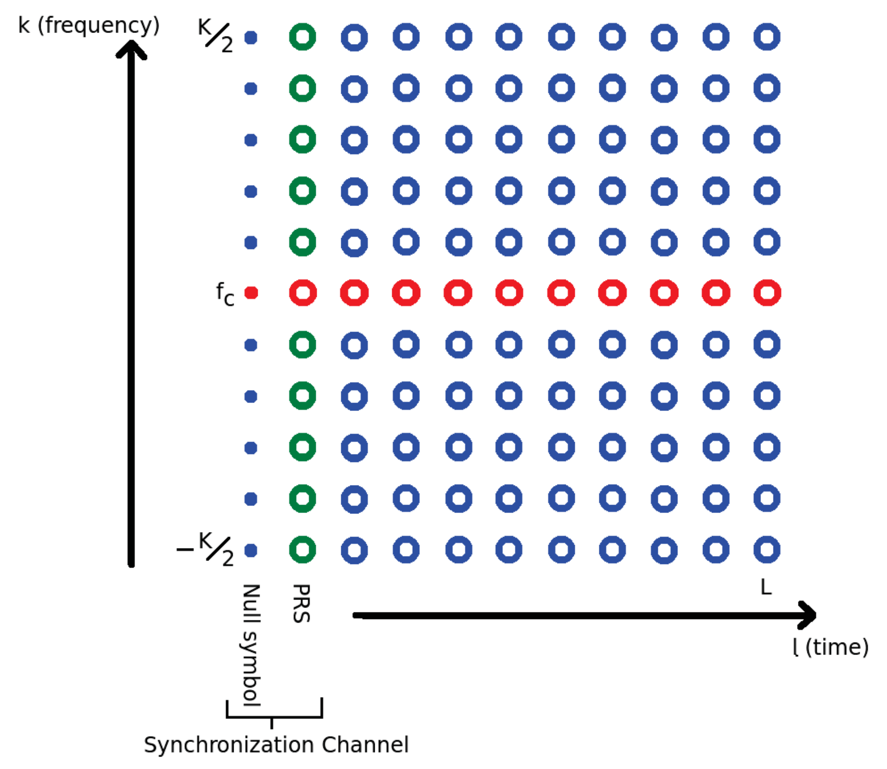


Figure 2.5.7: A set of symbols builds a data frame. The null-symbol synchronizes the receiver by indicating the start of the frame. The PRS acts as a phase reference for the first OFDM symbol. K = total number of sub-carriers, k = carrier index, L = total number of symbols / frame (null symbol excluded). l = symbol index. f_c = centre frequency. In the guard area between every symbol in the time dimension a cyclic prefix is transmitted (not shown).

Parameter	Definition	Value for transmission mode 1
L	Number of symbols / frame (null symbol excluded)	76
K	Number of used carriers	1536
T_F	Frame duration	96 ms
T_{NULL}	Null symbol duration	≈1,297 ms
T_S	Symbol duration (except null)	≈1,246 ms
T_u	Inverse of carrier spacing	1 ms
Δ	Guard interval duration	≈0,246 ms

Table 2.5.1: COFDM parameters for transmission mode 1 [1](p145).

2.6. Energy dispersal

The energy dispersal step scrambles the data with a PRBS (Pseudo-Random Binary Sequence). This is done to prevent transmission of regular patterns [1](p24, 128). The receiver must recover the data by scrambling with the PRBS again [8](p538).

A shift-register is loaded with the polynomial $P(x)=x^9+x^5+1$ (bit 9, 5 & 1 set to 1, other bits 0). The incoming data bits are the xor:ed with the shift-register content according to figure 2.6.1. The content in the shift-register shifts every bit. The shift-register resets every 24 ms [8](p537).

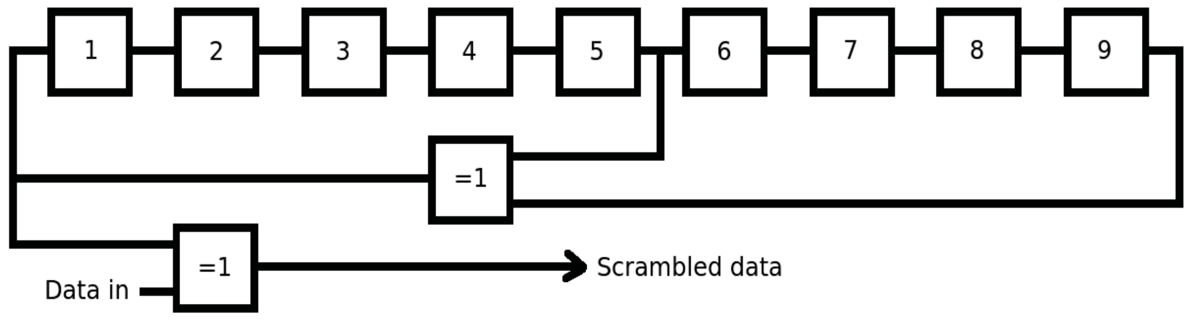


Figure 2.6.1: The PRBS generator generates a sequence based on the polynomial $P(x)=x^9+x^5+1$. The data stream is scrambled with the PRBS to avoid transmission of patterns.

2.7. Error protection

- Frequency interleaving
- Time interleaving
- CRC
- Convolutional coding

Frequency interleaving means that the information carried by each sub-carrier is re-ordered between sub-carriers before transmission according to a pre-defined pattern [1](p157). This gives protection against narrow-banded interferences. Example: Imagine that the text “Hello world” is transmitted with 11 sub-carriers, one carrier for each letter. If the first 3 carriers are interfered the transmitter would detect: “   lo world”. If the letters are shuffled around prior to transmission resulting in the message “lodHrlwe ol” and then shuffled back in the receiver the result could be: “He lo w rl ” (depending on shuffle sequence). The last message is easier to understand because it is easier to compensate for many small errors than one big. The same is true for error protection mechanisms.

The idea with time interleaving are similar to frequency interleaving except for that fact that information is shuffled in the time dimension instead. This minimizes the consequences caused by interferences over time. Time interleaving is only applied for the MSC [1](p137).

CRC (Cyclic Redundancy Check) is used at FIB, MSC and MPEG level for error detection. The data is bit-wise shifted trough a shift-register with 16 stages. The information is XOR:ed with a 16-bit polynomial between every stage. The final result in the shift-register represents the 16-bit CRC word. The CRC words are transmitted along with the corresponding data. It is then possible for the receiver to check the data by performing the same CRC calculation again and compare the result with the transmitted CRC word. CRC is just used for checking data, not correcting errors.

Convolutional coding is the FEC (Forward Error Correction) used in DAB [1](p129). The process adds redundancy to the transmitted signal by transmitting more data than needed. By doing so it is possible to correct errors caused by different channel conditions.

Two different protection modes are used UEP (Unequal Error Protection) and EEP (Equal Error Protection). The difference is that UEP uses more protection in critical parts of the signal like the header in the audio frame.

It is possible to use different protection levels for different sub-channels. UEP offers 5 protection levels and EEP 4. Higher protection level uses more bandwidth to gain more redundancy. The protection level affects the code rate where 8/32 is the highest and 8/9 is the lowest. The code rate is the ratio data in / data out. Code rate 8/32 means that the data is expanded by a factor four which means that four bits are transmitted for every input bit. In fact, the convolutional encoder always expands the amount of data with a factor four and then uses puncturing vectors to reduce the number of bits depending on selected code rate. In other meanings, every usable coding rate has its own puncturing vector.

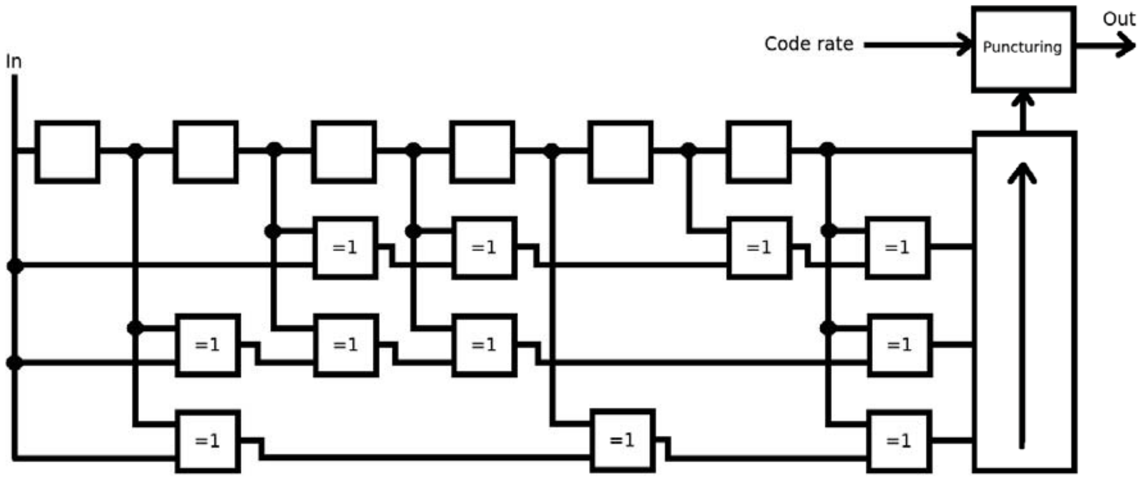


Figure 2.7.1: The mother code is generated by a network of shift registers and xor gates. The selected code rate controls the puncturing process [8](p537).

2.8. Audio and PAD

The audio and the PAD (Programme Associated Data) are encoded during transmission using MPEG Audio Layer II (ISO/IEC 11172-3 and ISO/IEC 13818-3) [1](p60). The audio frames varies in size depending on audio mode and the actual bit-rate (8 kb/s to 384 kb/s) [1](p72). The following audio modes are available:

- Mono
- Dual channel
- Stereo
- Joint stereo (Stereo mode with lower quality to save bandwidth)

Possible sampling rates are 24 kHz and 48 kHz.

The compression is done by filtering the audio signal into 32 sub-bands [1](175). A psycho-acoustic model processes the audio spectrum to determine how much the quality in every sub-band can be reduced without affecting the audible result significantly. The main idea is to take advantage of the weaknesses of the human hearing. For example, loud sounds masks away silent sounds. A reduction of the quality or, in some cases, removal of the silent sound will make a minor audible difference.

3. The RF-hardware

In this project the Ettus USRP B210 is used as the primary RF-hardware. The USRP B210 is a circuit board used to transmit and receive radio signals. It acts like an interface between the computer and the antenna. Most of the RF functions are handled by a single AD9361 chip [5]. For signal processing a Spartan 6 FPGA is used [3]. When used as a receiver all signals from the antenna are first amplified by a LNA (Low Noise Amplifier) [4] and down converted to a baseband. The down conversion is done in both 0° and 90° phase of the LO-signal (Local Oscillator) which creates a complex base band signal with I and Q components (In Phase and Quadrature). The I and Q components are converted to digital by two 12-bit ADC:s. Both digital signals are fed into the FPGA [3] which converts the signals into one UHD (USRP Hardware Driver) transport protocol. The UHD stream is sent to the computer through a USB 3.0 interface. Parameters like Gain, input channel, local oscillator frequency, sampling rate and other settings are also sent over the USB. When transmitting signals the operation is reversed. Two 12-bit DAC:s converts from digital to analogue and a PA (power amplifier) amplifies the signal that feeds the antenna. [4]

The device doesn't contain any FPGA software. Instead, the FPGA is loaded with software from the computer each power up. This is done automatically by the UHD driver. However, the driver need to be set up which is covered in the “installation” chapter.

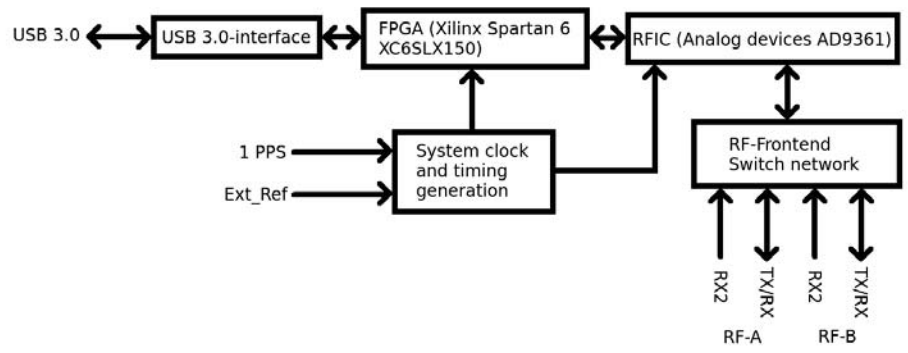

USB 3.0
USB 3.0-interface
FPGA (Xilinx Spartan 6 XC6SLX150)
RFIC (Analog devices AD9361)
1 PPS
Ext_Ref
System clock and timing generation
RF-Frontend Switch network
RX2
TX/RX
RX2
TX/RX
RF-A
RF-B

Figure 3.1: The architecture of the USRP B210.

Features of the USRP B210 [3]:

- RF coverage 70 MHz – 6 GHz
- USB 3.0 interface
- 12 bit ADC / DAC
- 2 TX and 2 RX channels with full or half duplex
- GPIO capability
- Up to 56 MHz instantaneous bandwidth (1 TX and 1 RX used)
- Up to 30.72 MHz of instantaneous bandwidth (2 TX and 2 RX used)

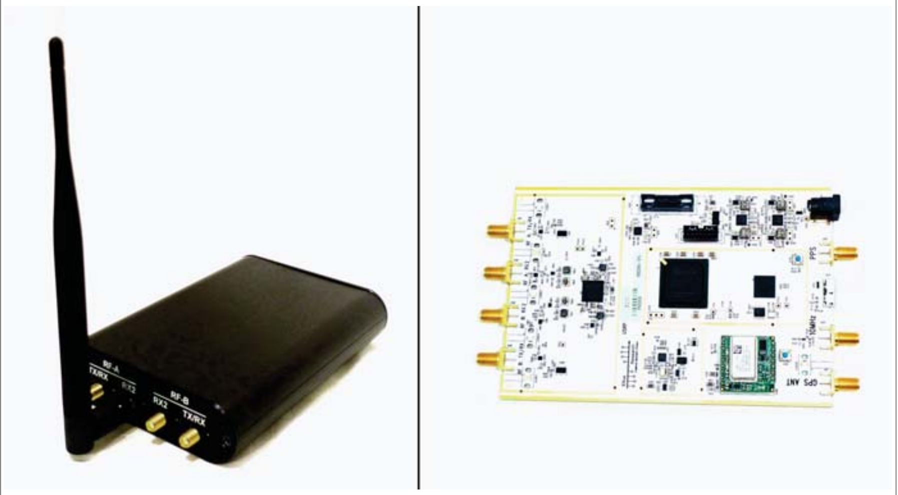
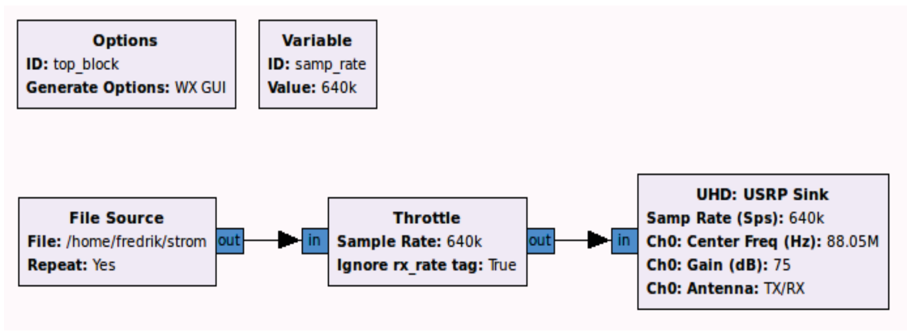


Figure 3.2: The USRP B210 with and without metal case.

4. GNU radio and GNU radio companion

GNU radio is a software development tool-kit for creation of software defined radios. It comes with a large number of function blocks that provides functions like signal sources and sinks, signal processing, mathematical and logical operations. The signal processing blocks are written in C++ [9]. SDR:s can be written in python after importing required GNU radio modules but the most user friendly way is to use the graphical interface GRC (Gnu Radio Companion). In GRC blocks are picked from a library and placed on a work area. Connections between blocks are made by drawing lines between their input and output ports which gives a very good overview over the signal flow [7]. There are a number of different data types used between blocks. To visualize which data type is associated with a specific in- or output, colour codes are used [10]. Most blocks have a number of parameters that can be set by the user. Examples of such parameters is frequency and amplitude of an oscillator or cut off frequency of a low pass filter. Many blocks have a brief documentation that can be viewed inside the setting menu.



Options
ID: top_block
Generate Options: WX GUI

Variable
ID: samp_rate
Value: 640k

File Source
File: /home/fredrik/strom
Repeat: Yes

Throttle
Sample Rate: 640k
Ignore rx_rate tag: True

UHD: USRP Sink
Samp Rate (Sps): 640k
Ch0: Center Freq (Hz): 88.05M
Ch0: Gain (dB): 75
Ch0: Antenna: TX/RX

Figure 4.1: Example of a simple GRC application for transmission of a pre-recorded FM baseband.

Colour	Data type	Name in GRC
🟫	Complex Float 64	
🟦	Complex Float 32	Complex
🟩	Complex Integer 64	
🟩	Complex Integer 32	
🟧	Complex Integer 16	
🟪	Complex Integer 8	
🟦	Float 64	
🟥	Float 32	Float
🟩	Integer 64	
🟩	Integer 32	Int
🟨	Integer 16	Short, IShort
🟪	Integer 8	Byte, Char, UChar
◼	Message Queue	
⬜	Asynchronous Message	
	Wildcard	

Table 4.1: The different data types used in GRC and their colour codes [10].

4.1. Sources and sinks

Source blocks, often referred to as "sources" are used as signal sources. Signals can be generated, read form a file or captured with a hardware device. There are also null source blocks that can be used to connect unused inputs, which in most cases is necessary to make the flow graph executable. Waveform generators and blocks that captures signals from a hardware are always set to a specific sample rate. This is not possible with file or null source blocks. In those cases the sample rate can be limited either by connecting a throttle block or by connecting it to a hardware sink. If a file source is used in a flow graph without a rate limiting block GRC will use 100% of the processor and the sample rate will be limited by the speed of the computer. The rate limiting block can be connected anywhere in the flow graph as long as the signal path from the source to the limiting block is synchronous.

Sink blocks are used to output signals from the flow graph to hardware, a file or a null source. When using hardware sinks the sample rate must be set to match both the incoming signal and the hardware. If previous signal processing blocks needs a higher sample rate than supported the hardware or a specific rate is needed the sample rate has to be converted. This can be done by using decimation, which many blocks offers, or by using a re-sampling block.

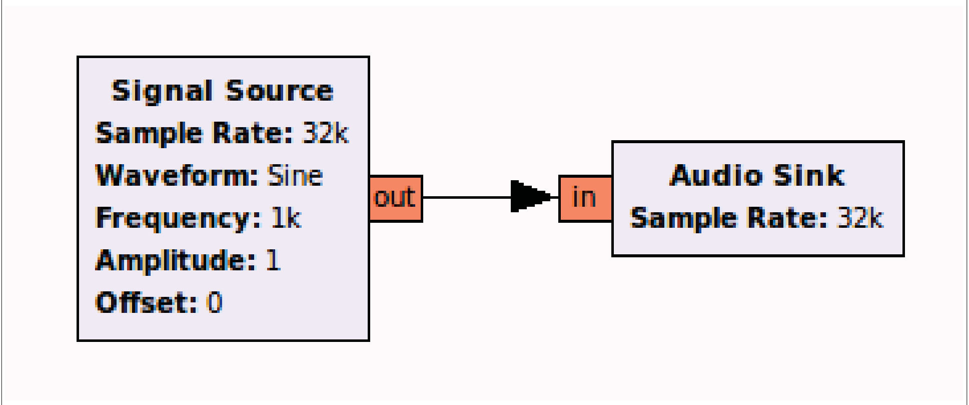


Figure 4.1.1: Simple flow graph that generates a 1 kHz sine wave and outputs it as audio through the sound card.

4.2. Signal processing blocks

The main feature with SDR and GNU radio is the possibility to process signals with software. This strategy is often referred to as DSP (Digital Signal Processing). GNU radio provides a lot of DSP blocks that in GRC are categorized in groups in the block library. It is also possible to search for blocks. When connecting blocks together there are some rules. The data type must match in every connection. Many blocks have configurable ports which makes it possible to set data type. There are also a number of blocks called type converters which are used to convert form one data type to another. The sample rate must always be considered when connecting blocks. First, the sample rate must be high enough to handle the actual signal. According to the sampling theorem the minimum sampling rate is defined as $f_s > 2 \cdot f_{max\ signal}$ [6] (p 122). If a block has more than one input port the sample rate must in many cases be the same on both inputs ports.

4.3. Tagged streams

There are some situations when it is necessary to pass information between blocks in sync with a signal that doesn't exist in the signal itself. For example, one block extracts data packages and sends it downstream as a data stream. To be able to interpret the data the next block needs to know the beginning and length of every data packet. If both blocks supports tagged streams the solution is to tag the beginning of every packet with a tag containing the packet length. One feature is that tags are shown when using some of the instruments which makes debugging easier.

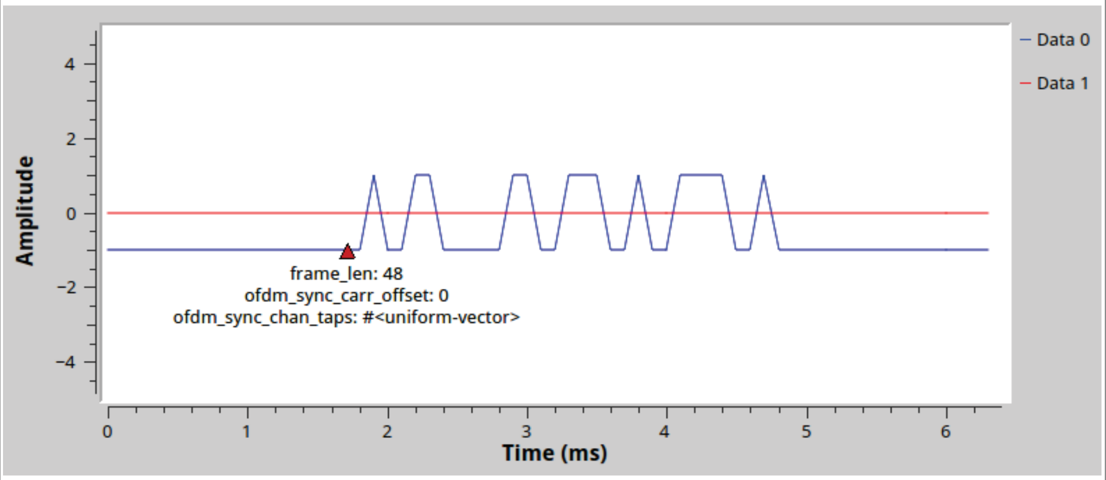


Figure 4.3.1: Time scope showing a data burst containing tags at the beginning.

4.4. GUI and instrumentation

GRC offers some GUI (Graphical User Interface) Widgets for creating simple GUI:s with check boxes, sliders, choosers etc. The parameter "ID" sets the name of the variable that carries the value from the widget. For example, if the ID of a chooser is set to "freq" and the user selects 100 using that chooser the variable "freq" will carry the value 100. Using the variable "freq" as parameter in blocks instead of numerical values allows the user to control that parameter from the GUI in real time. The instrumentation blocks are used to create instruments like oscilloscopes, spectrum analysers etc. They have only input ports and the output is plotted in a window. GRC offers widgets and instrumentation using both WX and QT framework. The "option" block is set to use either WX or QT.

5. Method

Receiving and decoding DAB signals is complicated and the receivers are very complex. A first study of the specifications were done and a planning report were written. The next step was to install programs and preform simple test projects to be familiar with the tools.
Following hardware and software were used during the project:

- RF-hardware: Ettus USRP B210
- Computer: Dell Precision M4600 (32 MB ram)
- Operating system: Lubuntu linux 14.10 kernel 3.16.0-33-generic (x86_64) and Windows 7
- DAB receiver: Pure One Elite
- USB-dongle: FM+DAB+DVB-T SDR with RTL2832 R820T Chipset
- GNU Radio Companion version 3.7.3

Following preparing tasks were done:

- Setting up the computer
  - Installation of Lubuntu Linux
  - Setting up the UHD (USRP Hardware Driver)
  - Installing GNU radio Companion
- Understanding the basics of the tools
  - Generation of simple signals
  - Using sources, sinks and instrumentation
  - Creation of simple graphical user interfaces using WX and QT
  - Creation of a mono / stereo FM receiver with pre-sets
- Generation of DAB baseband signals
  - Creation of a baseband recording application
  - Capture DAB signals using the USRP and the recording application
  - Creation of a playback application
  - Verifying captured base bands

- Generation of a authentic FIC data file and MPEG audio file
    - Installing “DAB player von Andreas Gsinn”
    - Saving FIC data
    - Saving MEPG audio

All tasks above were done with success and some of them are described in details later. It is obvious that DAB baseband signals are needed for testing purposes. The other test files were strategically chosen so other blocks of the receiver can be developed stand alone without a working OFDM receiver.

The main part of the project was the implementation of the receiver and the interference injector. The idea was first to define the blocks in such a way that the output can be predicted and measured easily after applying a appropriate input signal. One example is the synchronization algorithm which shall output a sync pulse every 96 ms after applying a DAB mode 1 signal at the input (according to the frame length). This can be measured using the GNU-radio instrumentation.

However, as more knowledge about the technology was collected along the way following main and sub-areas were defined:

- OFDM receiver
    - Frame synchronization
    - Frequency correction
    - OFDM sampling
    - FFT
    - Differential decoding and frequency interleaving
    - Serialization
    - Constellation demodulator

- Stream manipulation
    - FIC detection
        - FIB detection and CRC
        - FIG detection and presentation of service labels
        - Multiplex configuration
    - MSC detection
        - CIF detection and CRC
        - PAD detection and presentation
        - MPEG audio decoding and playback

- Interference injection
    - Sine wave interferer
    - Square wave interferer
    - Multipath simulation
    - Recorded interferences
        - Recording
        - Playback

Details about the areas mentioned above can be found later in this chapter.
It should be mentioned that discoveries during the project caused changes in the defined sub-areas.

There are some blocks that comes with GNU-radio that builds a fully functional OFDM transmitter or receiver with a single block. It was found early that none of them supported the parameters used by DAB. The only choice then was to try to build the OFDM receiver using discrete blocks.

One major mistake during the implementation was caused by a misunderstanding about how phase errors in the sub-carriers are corrected. The most common approach in OFDM is to use a known symbol (which in this case would be the PRS) to measure the phase error in every sub-carrier. The result is then used to equalize all sub-carriers the rest of the frame. Most OFDM blocks that comes with GNU-radio are designed that way.

5.1. Planning report

The planning report contained the following tasks:

1. Preparations
2. Reception of a DAB signal
3. Demodulation of the DAB signal
4. FIG detection
5. MSC detection
6. Detection of PAD data and audio stream
7. MPEG decoding
8. Interference handling
9. Integration
10. Completion of report

Notable changes from the planning report was that step 5 and 6 was excluded since it was found in step 4 that important function blocks were missing. Step 9 was also excluded since it requires that all previous steps are completed. Also the writing of the report were delayed about 3 weeks caused by personal issues. The progress were communicated to the supervisor continuously.

A Gantt chart can be found in annex E.

5.2. Installation

This guide assumes that Linux is installed on the computer and that it has access to the internet. This guide is verified to work with the distribution Lubuntu 14.10. To install GNU Radio, open a terminal window and type:

```
sudo apt-get install gnuradio
```

The user will first be prompted to enter password and later confirm the installation by typing “j” or “y” and enter. After the installation the program can be launched from the start menu under the name GRC. To launch GNU Radio from the terminal type:

```
~$ gnuradio-companion
```

At this point GNU Radio should be up and running but it will probably not connect to the USRP B210. Check the driver by connecting the device to the computer, open the terminal and type:

```
~$ uhd_find_devices
```

If the UHD driver is missing the user will get the message “command not found”. Solve this with:

```
~$ sudo apt-get install uhd
```

If the device is not detected the user will get the message “No UHD Devices Found”. The main problem is likely that the FPGA images for the device are missing. To download and install these type:

```
~$ cd /usr/lib/uhd/utils
~$ sudo ./uhd_images_downloader.py
```

Check again with:

```
~$ uhd_find_devices
```

If everything went well the UHD-driver should upload the FPGA image to the device and then output something like:

```
linux; GNU C++ version 4.8.3; Boost_105500; UHD_003.007.001-0-unknown

--------------------------------------------------
-- UHD Device 0
--------------------------------------------------
Device Address:
    type: b200
    name:
    serial: F5000B
```

If the device still is undetected type:

```
~$ lsusb
```

Disconnect the device and type again:

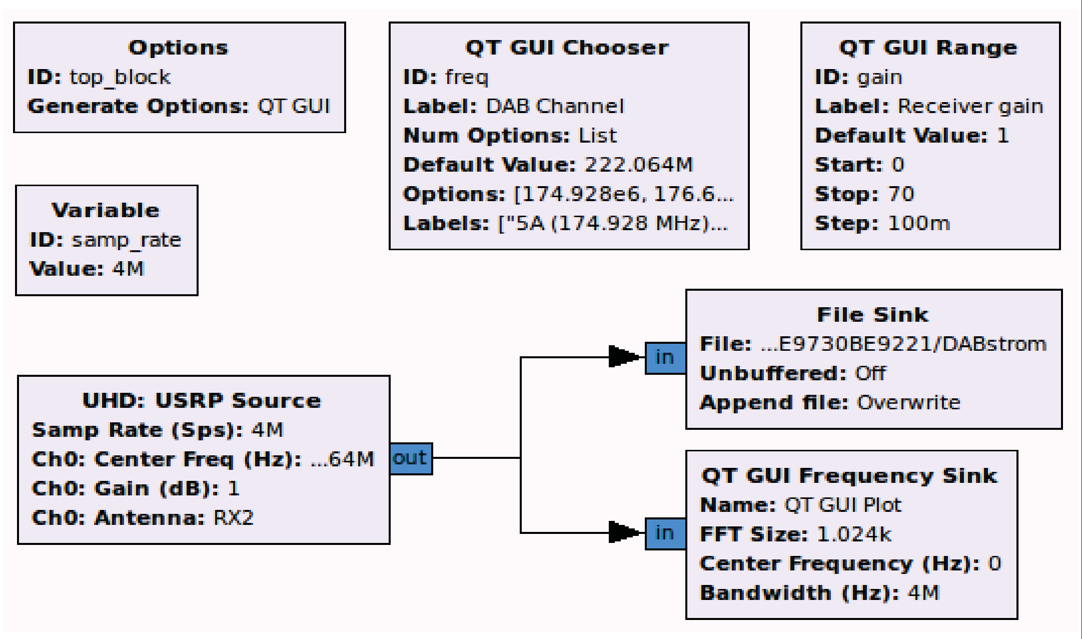
```
~$ lsusb
```

If one device is missing compared to the first list the conclusion should be that the USB connection is working. For further troubleshooting, visit Ettus homepage.

5.3. Generation of DAB baseband signals

The experimenting work with modelling the demodulator requires a valid DAB baseband signal source. The solution was to capture a DAB ensemble with the USRP, record it to file, verify the file and then use the file as signal source. One alternative solution could be to receive a live DAB transmission with the USRP and to use that signal as source. One major disadvantage with that is that it is hard to know if the signal quality is good enough.

GNU radio comes with blocks that makes recording and playback of baseband signals easy. A RF recording and playback equipment was used as signal source. The output of the signal source was connected to input RX2 on the USRP. The recording was done with the USRP source block connected to a file sink block. A OFDM demodulator requires I and Q components of the signal so both blocks were set to complex data type. A simple GUI that provides possibilities to adjust gain, set frequency and to monitor the received spectrum was also made. The gain was adjusted for the best SNR and the frequency so the centre frequency of the OFDM spectrum was close to zero. Ensembles were first recorded with three different sample rates: 2, 4 and 8 MHz. Even 2,048 MHz was used later to match the FFT window with the FFT length. Since the sample rate must be a multiple of the clock rate this was changed from the default 32 MHz to 32,768 MHz when sampling with 2.048 MHz.


Options
ID: top_block
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 4M

QT GUI Chooser
ID: freq
Label: DAB Channel
Num Options: List
Default Value: 222.064M
Options: [174.928e6, 176.6...
Labels: ["5A (174.928 MHz)...

QT GUI Range
ID: gain
Label: Receiver gain
Default Value: 1
Start: 0
Stop: 70
Step: 100m

UHD: USRP Source
Samp Rate (Sps): 4M
Ch0: Center Freq (Hz): ...64M
Ch0: Gain (dB): 1
Ch0: Antenna: RX2

File Sink
File: ...E9730BE9221/DABstrom
Unbuffered: Off
Append file: Overwrite

QT GUI Frequency Sink
Name: QT GUI Plot
FFT Size: 1.024k
Center Frequency (Hz): 0
Bandwidth (Hz): 4M

Figure 5.3.1: Flow graph of the baseband recorder

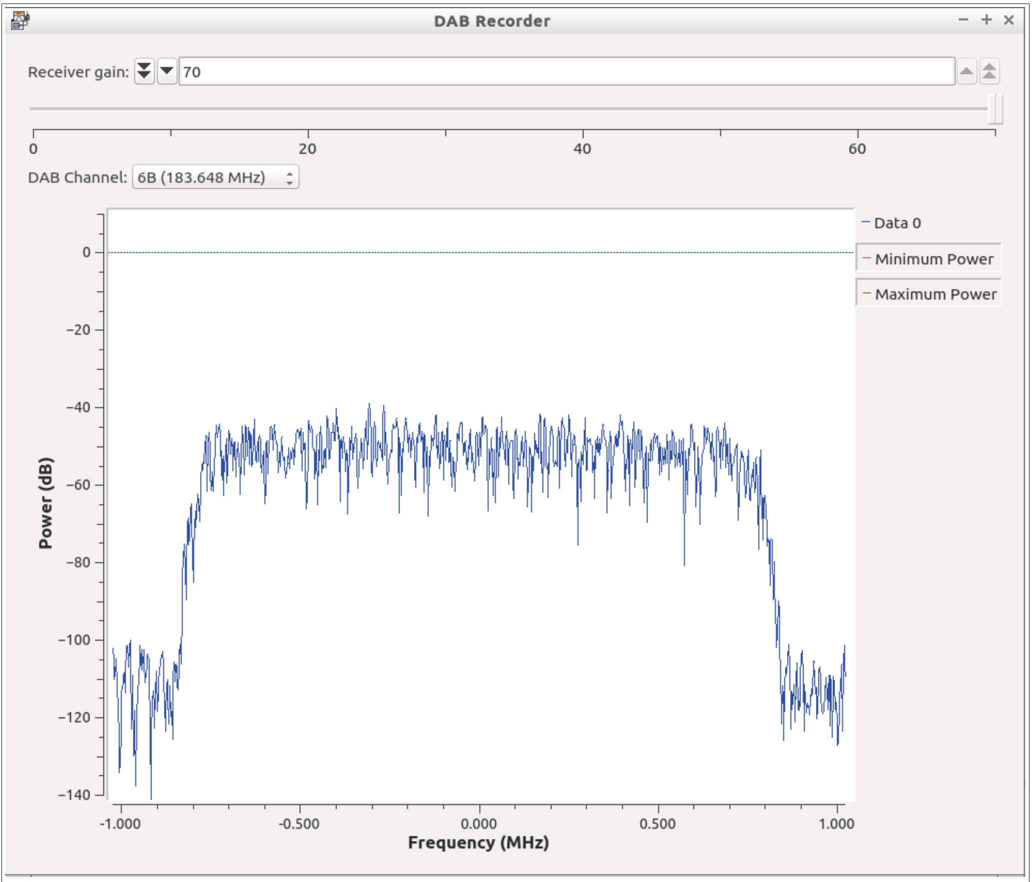
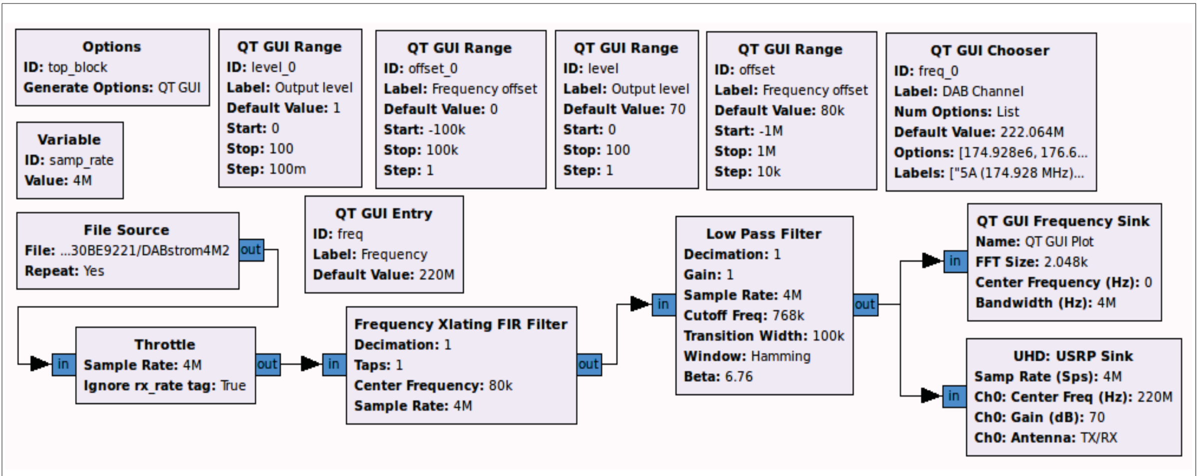


Figure 5.3.2: GUI of the baseband recorder.

To verify the recorded baseband signal a transmitter application was built with GRC. In this case the USRP was used as a transmitter by using the USRP Sink block. The transmitted signal was received with a domestic DAB receiver. If the ensemble could be received and all services reproduced with all service labels intact and the audio without any audible errors the file was considered as usable for further experiments. For fine tuning of the frequency offset a Frequency Xlating FIR filter was used. A low pass filter was added to suppress frequencies outside the sub-carriers. The reception quality were flawless using 8 and 4 MHz sampling rate. 2 MHz worked poorly (skipping audio). 2 MHz should be enough but the explanation can be that it was a second generation copy. The 2,048 MHz recording were hard to verify because GNU-radio froze all the time when combining that frequency with the USRP sink. However, the signal looked fine in the FFT scope.


Options
ID: top_block
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 4M

QT GUI Range
ID: level_0
Label: Output level
Default Value: 1
Start: 0
Stop: 100
Step: 100m

QT GUI Range
ID: offset_0
Label: Frequency offset
Default Value: 0
Start: -100k
Stop: 100k
Step: 1

QT GUI Range
ID: level
Label: Output level
Default Value: 70
Start: 0
Stop: 100
Step: 1

QT GUI Range
ID: offset
Label: Frequency offset
Default Value: 80k
Start: -1M
Stop: 1M
Step: 10k

QT GUI Chooser
ID: freq_0
Label: DAB Channel
Num Options: List
Default Value: 222.064M
Options: [174.928e6, 176.6...
Labels: ["5A (174.928 MHz)...

File Source
File: ...30BE9221/DABstrom4M2
Repeat: Yes

QT GUI Entry
ID: freq
Label: Frequency
Default Value: 220M

Throttle
Sample Rate: 4M
Ignore rx_rate tag: True

Frequency Xlating FIR Filter
Decimation: 1
Taps: 1
Center Frequency: 80k
Sample Rate: 4M

Low Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 4M
Cutoff Freq: 768k
Transition Width: 100k
Window: Hamming
Beta: 6.76

QT GUI Frequency Sink
Name: QT GUI Plot
FFT Size: 2.048k
Center Frequency (Hz): 0
Bandwidth (Hz): 4M

UHD: USRP Sink
Samp Rate (Sps): 4M
Ch0: Center Freq (Hz): 220M
Ch0: Gain (dB): 70
Ch0: Antenna: TX/RX

Figure 5.3.3: Flow graph of a DAB transmitter using a baseband file source

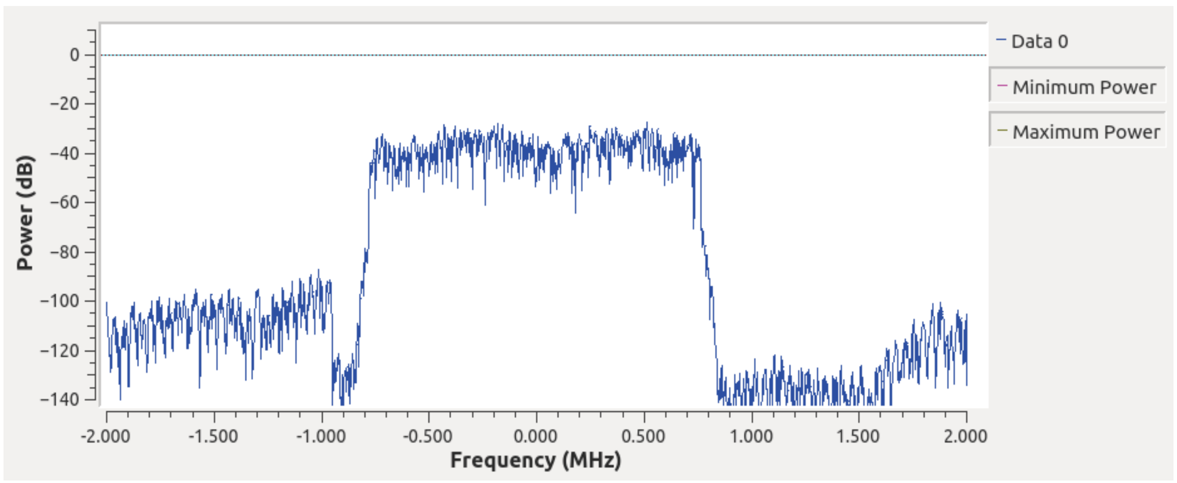


Figure 5.3.4: FFT plot of the transmitted signal (before up conversion). The higher noise level on the left side is in fact a second DAB-ensemble suppressed by the low pass filter.

5.4. Generation of FIC data file and MPEG audio file

After the demodulation the next milestone is to detect the FIC and decode it. The first real proof that the OFDM demodulator is working would be to extract the service labels from the FIC and print them on the screen. Developing a FIC decoder requires authentic FIC data for testing purposes. It is a benefit if the test FIC contains the same data as the FIC in the baseband file. For generation of a authentic FIC file a DAB receiving software called “DAB player von Andreas Gsinn” was used. The software player supports a large number of USB dongles, for example those using the Realtek RTL2832U chip. It also offers possibilities to save FIC data. It is also possible to save MPEG audio from the selected service. A number of FIC files were created from real broadcasts and from the earlier developed DAB baseband transmitter. MPEG audio files were also created for testing of the audio decoding step.

Figure 5.4.1: The dongle used together with "DAB player von Andreas Gsinn" to capture FIC and MPEG data.

5.5. OFDM frame synchronization

To be able to extract any valid data from the stream a frame synchronization algorithm is needed. GNU radio provides 3 different OFDM frame synchronization methods, Cox & Schmidt synchronization, PN-synchronization and a third method that detects the start of the frame by two known symbols. None of these blocks are able to sync on the null symbol. The two first blocks uses a sync-symbol with every even or every odd sub-carrier set to zero which makes the first half identical to the second half in the time domain. They also measures the frequency offset in the signal. The third block relies on two known symbols in row according to its documentation. It was tested that a null-symbol can't be used for this block as one of the two known symbols.

For detection of the null symbol the sync algorithm must detect energy dips in the signal. This can be done using standard GNU radio blocks. To prove that a RMS block was used to measure the energy in the signal over a given time, in this case the length of the null symbol. The energy signal is then inverted so the energy dips are converted to energy tops. A threshold block forms pulses with amplitude 1 by comparing the input signal with the high and low threshold values. Some more blocks were used for data type conversion and offset adjustment. This algorithm was able to reproduce the synchronization pulses. However, it struggled with some problems and a custom made block would do the job more accurate. This algorithm is sensitive to shifts in the signal level and SNR (Signal Noise Ratio). This was revealed by shifting the signal level and offset manually. The explanation is that this algorithm uses fixed threshold and offset. The suggestion is to build a custom block that carries out both the synchronization and the frequency offset measurement. Creating new blocks is outside the limitations of this project so this solution were used.

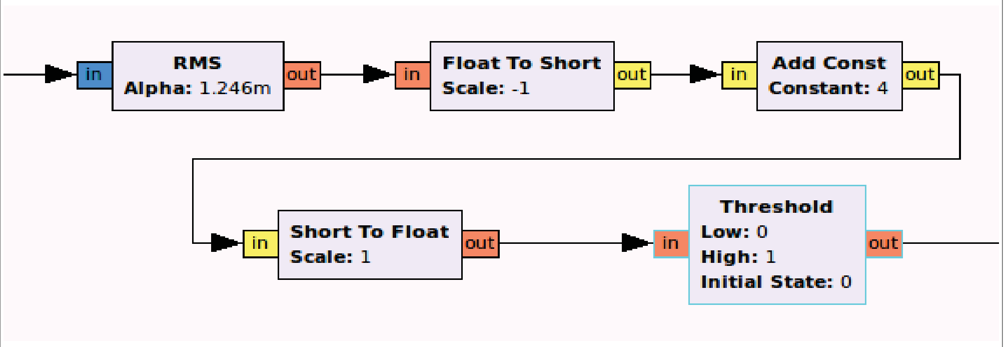


Figure 5.5.1: Flow graph showing the frame sync algorithm.

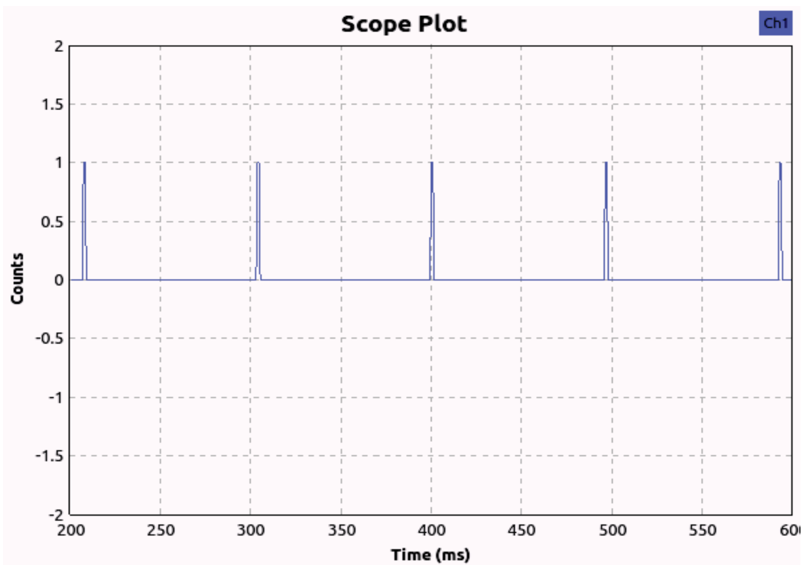


Figure 5.5.2: Time scope view of the frame synchronization pulses. The frame duration is 96 ms in mode 1 [1].

5.6. Frequency correction

It is very important that f_c is located as close as possible to 0 Hz after the down conversion. Since the spacing between the sub-carriers is only 1 kHz in DAB mode 1 [1](p27) even small frequency shifts can cause sub-carriers to be detected in the wrong frequency bin or interfering each other resulting in corrupted data. Both the Cox & Schmildt- and the PN-synchronization block detects small frequency errors and outputs those as a frequency offset signal which is used externally for frequency correction by controlling a frequency modulator. The measured frequency offset then results in a generated frequency that is used to convert the baseband signal to the right frequency. These blocks are unusable in this case. They are not able to detect the null symbol, see chapter 5.5. Unfortunately, they were also unable to detect the frequency offset which was tested. The explanation is likely that they need to find the PRS to measure the frequency offset. This is impossible without synchronization.

The suggestion is to, as mentioned before, build a custom made block that does both the frame sync extraction and measures the frequency offset just like the excising OFDM sync blocks.

The suggestion is to use the PRS for frequency offset measurement [8](p534).

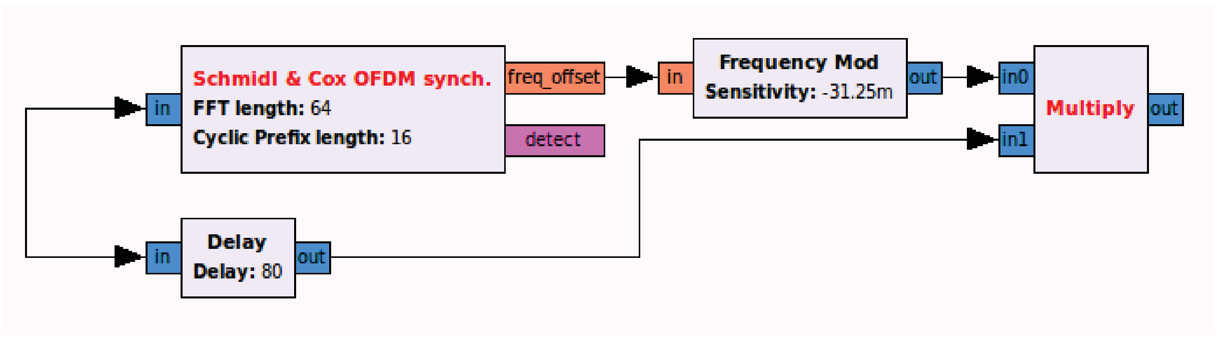
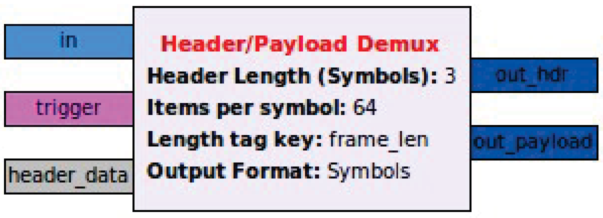


Figure 5.6.1: Frequency correction as it is done in the example "rx_ofdm.grc" included in the GRC bundle. The detect signal outputs the frame synchronization. The suggestion is to build a "DAB-sync" block that replaces the Schmidl & Cox block above.

5.7. OFDM sampling

The FFT block requires a vector at the input which size is equal to the FFT length. The conversion to vectors must be synchronized with the signal. GNU radio offers a block called OFDM sampler for that purpose. It requires the parameter FFT size which in this case is 2048 and the parameter symbol length which is equal to the FFT size + cyclic prefix. The cyclic prefix is transmitted in the guard area between two symbols[8](p362). It is given in the standard that the symbol time is 1 ms and the guard time is 0,246 ms for DAB mode 1 [1](145). Therefore the cyclic prefix length must be: $\frac{(0.246 \cdot 2048)}{1} \approx 504$ The synchronization was done by connecting the OFDM frame sync signal to the flag input. It is unclear if that's the right way to use the flag input port since there is no documentation about that port. It is likely that the OFDM sampler block can be used for DAB.

Another block that does the same thing is the header / payload demux. This block is used in the OFDM packet receiver example “rx_ofdm.grc”. In addition to the sampling it also provides a demux function intended for receivers with separate decoding paths for header and payload. The block were tested but discarded since it didn't accept the DAB signal.



Header/Payload Demux
Header Length (Symbols): 3
Items per symbol: 64
Length tag key: frame_len
Output Format: Symbols

Figure 5.7.1: The block "header/payload demux" preforms sampling to OFDM symbols. However, it was discarded because it didn't accept the DAB signal.

5.8. FFT (Fast Fourier Transform)

The next signal processing steps must be done in the frequency domain. The FFT block transforms the incoming signal from time to frequency domain when it is set to forward mode (and opposite direction in reverse mode). The FFT size refers to the frequency resolution. Setting a FFT length of 2048 will divide the FFT window in 2048 frequency bins. The FFT window is limited by the sampling rate. In this case the window was defined by the sampling rate 2,048 MHz. Dividing 2,048 MHz with 2048 gives a spacing of 1 kHz between the frequency bins which is equal to the spacing between the sub-carriers in DAB mode 1. The FFT block outputs a vector with a length corresponding to the FFT size. Each element consists of a complex number representing what was detected in the corresponding frequency bin. It is then possible to reproduce amplitude and phase of each sub-carrier. The FFT block also supports the float data type but this is not interesting in this case since the phase information is necessary. It is very likely that the FFT- block provided with GNU-radio can be used in a DAB implementation without any modifications. The proof for that is that the constellation points were detected using this FFT block, see figure 5.10.1.

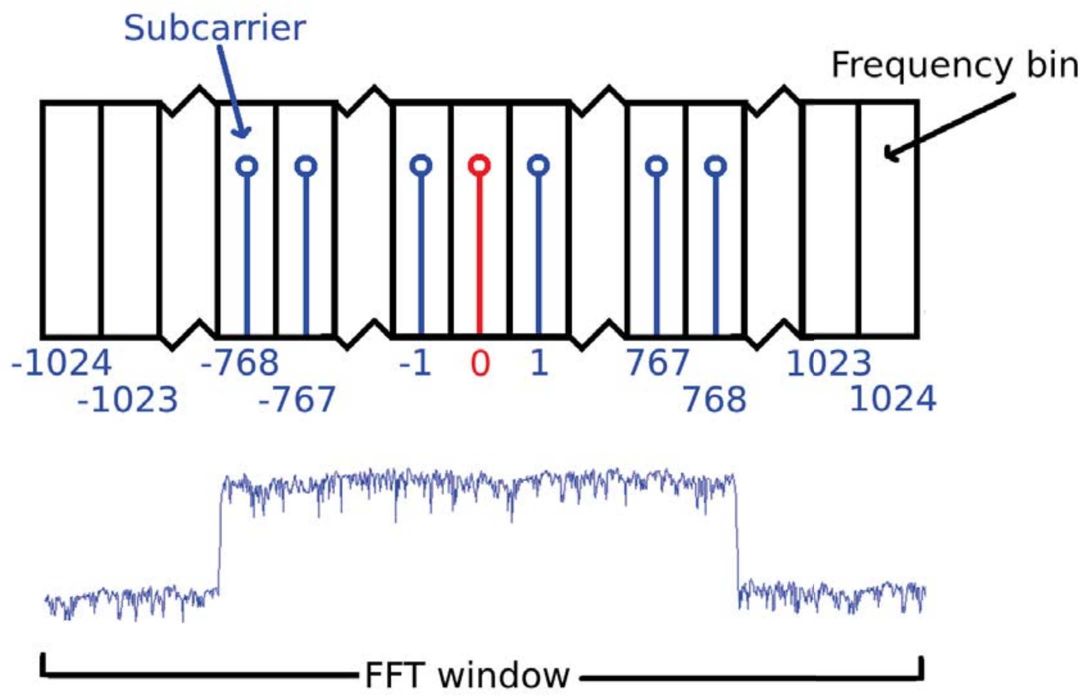


Figure 5.8.1: Illustration showing how sub-carriers are detected in frequency bins using the FFT algorithm. Dividing the FFT window in 2048 bins represents the FFT length 2048. At the bottom the corresponding OFDM spectrum is shown.

5.9. Differential demodulation and frequency deinterleaving

DAB uses Differential modulation and for that reason no equalization is needed. Every sub-carrier uses the same sub-carrier index from the previous symbol as phase reference. The PRS acts as phase reference for the first symbol after the synchronization channel. The differential modulation is described by the following formula [1](p161):

$$z_{l,k}=z_{l-1,k}\cdot y_{l,k}$$

Z is a differential modulated item, l is the symbol number, k is the sub-carrier index and y is a item without differential modulation.

GNU-radio provides a differential phasor block. Unfortunately it doesn't support vector mode. Differential modulation is applied on sub-carriers with same index from the previous symbol. For that reason the differential phasor block must handle vectors to be able to keep the whole previous symbol in memory. Such a block would fit best between the FFT block and the serializer block. It could be combined with the frequency deinterleaving function which also fits best where the signal is in vector form and in the time domain.

Frequency interleaving means that the data that modulates each sub-carrier are shuffled around in each frame according to a pattern described in the DAB specification [1](p157). The purpose is to slice narrow-banded interferences to many small errors, which later can be corrected by the error protection mechanism. The frequency deinterleaving shuffles the data back. There is no obvious block for that function in GNU Radio. There are interleave and deinterleave blocks that splits a vector to separate streams and vice versa. They are unusable in this case for at least two reasons:

- It is impractical with blocks having 2048 ports.
- The shuffle sequence also depends on the frame number. Using the existing blocks will result in a static pattern.

The missing block(s) that should fit between the FFT block and the serializer block shall take a complex vector with the size equal to the FFT-length as input. It shall apply differential demodulation and frequency deinterleaving according to the DAB standard. The PRS and the null-symbol must be removed to prevent the to propagate in the data stream. It shall be possible to set FFT-size and interleave pattern. Alternatively, the parameter “ transmission mode” would be enough since both FFT-size and interleave pattern are associated with the transmission mode as described in the DAB standard.

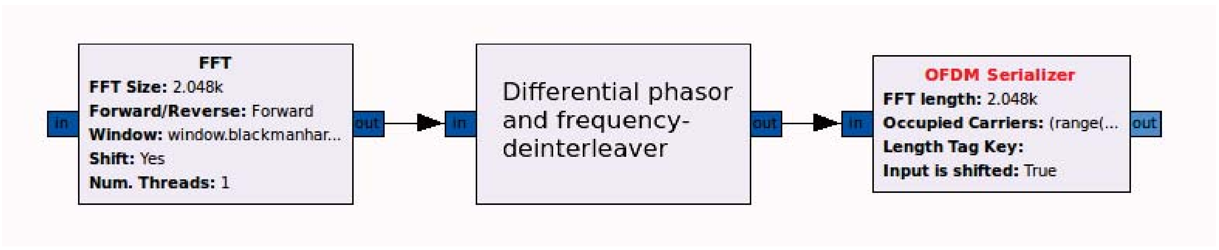


Figure 5.9.1: The best place to preform differential demodulation and frequency deinterleaving is probably between the FFT and the OFDM Serializer blocks. Dark blue ports symbolizes complex vectors.

5.10. Serialization

At this stage the received signal is represented as a vector containing complex values in the time domain. Before the constellation demodulator the signal has to be serialized. This can be done with the “OFDM serializer” block. Required parameters are FFT length and occupied carriers. Occupied carriers are -768 to -1 and 1 to 768 (0 is unused). The easiest way to set the occupied carriers-parameter is to enter: (range(-768,-1) + range(1,768)). The FFT length should be 2048 for DAB mode 1. The “OFDM serializer” block can probably be used for DAB as it is without any modifications.

To support all previous statements about blocks and their usability for DAB it should be possible at this point to plot the QPSK constellation. To compensate for the missing differential phasor (see chapter 5.9) a work around was done. The idea was to use the existing differential phasor block even if it doesn't support vectors. This was done by delaying the output of the FFT one symbol and then connect the delayed signal to a second OFDM serializer block. This produces two streams: One from the current symbol and one from the previous. The two streams were merged into one stream using a interleave block. This produces one stream where every second item comes from the current symbol and the other second comes from the delayed previous symbol. By using this method the amount of data is doubled. This shouldn't be a problem if the only purpose is to apply differential demodulation so the QPSK constellation can be viewed.

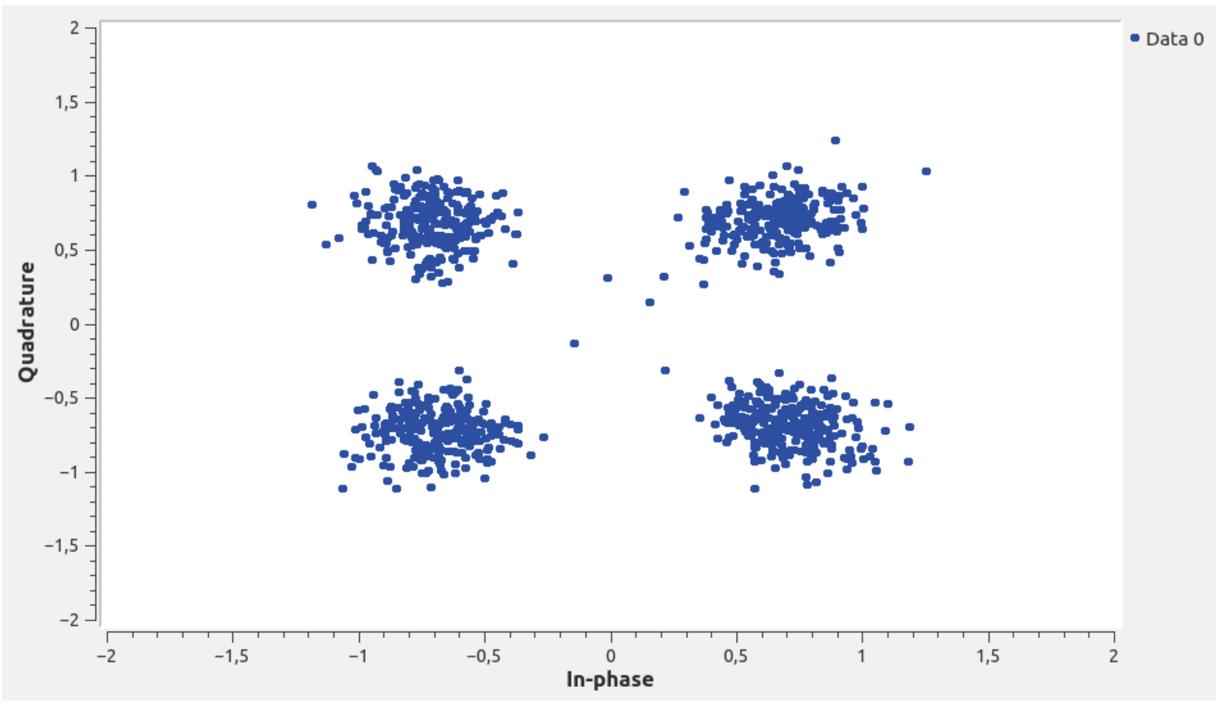


Figure 5.10.1: Constellation diagram showing the QPSK dots extracted from the signal after the serialization. The synchronization worked poorly, the frequency correction is missing and the null-symbol wasn't removed so the dots were unstable. A work around was also done to compensate for the missing differential phasor block. However, this test shows the potential of the OFDM sampler, FFT and OFDM serializer blocks. A flow graph of the code that generated this image can be found in annex C.

5.11. Constellation decoder

At this stage the complex signal from the serializer shall be turned into a bit-stream. This is done by a constellation decoder. There are some blocks provided with GNU-radio that probably can be used as they are. However they couldn't be tested since there were important blocks missing earlier in the chain. One of the interesting blocks is the constellation decoder block. It is a very generic block that takes a constellation object as parameter to map the incoming signal against. The differential demodulation is done earlier so the constellation map shall be a simple QPSK-sheme.

5.12. Stream manipulation

Most of the stream manipulation could not be done with the existing blocks. Data must be picked in different positions in the data frames and in many cases on conditional criteria. There is a block called “keep M in N” which outputs M items from every N items at the input. It is also possible to set a offset which makes it possible to pick the M items anywhere from the N items. This could, for example, be used to separate the FIC from every frame. The problem is that the block must be synchronized with the stream so every N items corresponds to a data frame. The obvious way to do that is to use a tag as sync-pulse. Unfortunately, this block doesn't support tag synchronization. There are many situations where a conditional statement is needed. For example, if a FIG is flagged as a FIG type 0 the information should be used to select sub-channel. If the FIG is of type 1 the contents should be addressed to the display etc. None of the blocks were suitable for these conditional operations.

Another necessary function is the CRC calculations. There are a block called “Stream CRC32” that can both generate and check CRC. Unfortunately, it is locked to 32-bit CRC and DAB uses 16 bits. It is also impossible to initialize the shift-registers to “1” which is necessary [1](p29, p36, p39).

When it comes to the output and control of the receiver the GUI-widgets provided with GRC is insufficient.

One thing that could be done was the audio decoding and playback. The audio stream that was captured as described in chapter 5.4 could be streamed to the media player VLC as a UDP stream using the block “UDP sink”. When feeding the UDP sink from a file a “trottle” block was needed to prevent excessive buffering. The conclusion must be that custom block(s) must be made for the all stream manipulation and GUI.

5.13. Interference injection

Another purpose with this project was to build a interferer in GRC that injects interferences into a DAB transmission. The interferences can both be synthesized or pre-recorded. A concept interferer, which easily can be expanded with more functions, were made (see annex D for flow graph).

The signal source consists of a pre-recorded DAB-ensemble down converted to baseband, a sine wave generator, a square wave generator and a file source for pre-recorded interferences. Each source except for the DAB-source have their own level controllers which can be controlled from the GUI. The level is adjusted by simply multiplying each signal with a constant using "multiply constant" blocks. The constants are connected to variables that can be controlled by sliders in the GUI. There are also sliders for controlling the frequency of each waveform generator. Another feature with the interferer is that multipath propagation can be simulated. A multipath occurs when the transmitted signal is reflected off surfaces on its way to the receiver. The reflections takes a longer path then the direct signal resulting in a delay in the reflected signal compared with the direct signal. The reflected signal is also attenuated as a result of the longer path and the attributes of the reflective surface. The multipath simulator in the interference injector simply delays the DAB source signal using a delay block. The delay time can be controlled by a slider to simulate different multipath distances. It is also possible to control the attenuation caused by the multipath with a level slider. All signals are then added using a multi input add block before it is transmitted by the USRP via the USRP sink block. The signal can then be received using a ordinary DAB receiver.

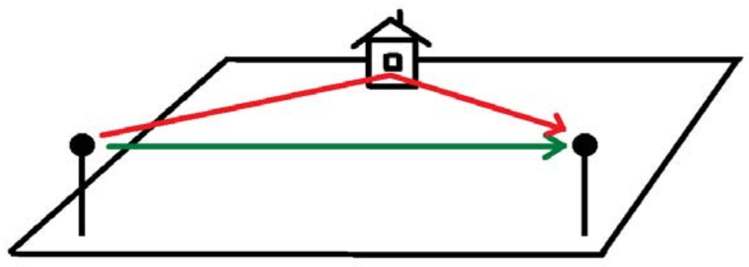


Figure 5.13.1: A simple multipath situation. The green signal is the direct signal and the red one is reflected off a wall. The reflected signal is delayed and attenuated as a result of the longer path. It is also likely that the ground would reflect the signal resulting in a second multipath (not shown).

The interference injector were tested by receiving the transmitted signal using the Pure One Elite DAB receiver. All functions functioned as predicted. Different interference sources were used one by one or combined and the audio began to skip when the interference levels became too high. It is not a accurate instrument by any means but it can still be used to evaluate a DAB receivers resistance to different interferences relative to each other. This can be done as follows:

1. A recorded interference is added to the DAB signal and transmitted to the receiver.
2. A second synthesized interference, a square wave for example, is added and the level is increased until the reception becomes unacceptable.
3. At the point when the reception were affected the set level of the second interference is noted down.
4. Repeat step 1 to 3 with a second recorded interference.
5. Comparison of the two levels reveals which one of the two interferences that affects the DAB signal in the most negative way.

It should be taken in consideration that the result can differ depending on the attributes of the second interference relative to the first one. It is also of great importance that the test is carried out in a environment free from other interferences. An anechoic chamber is preferable.

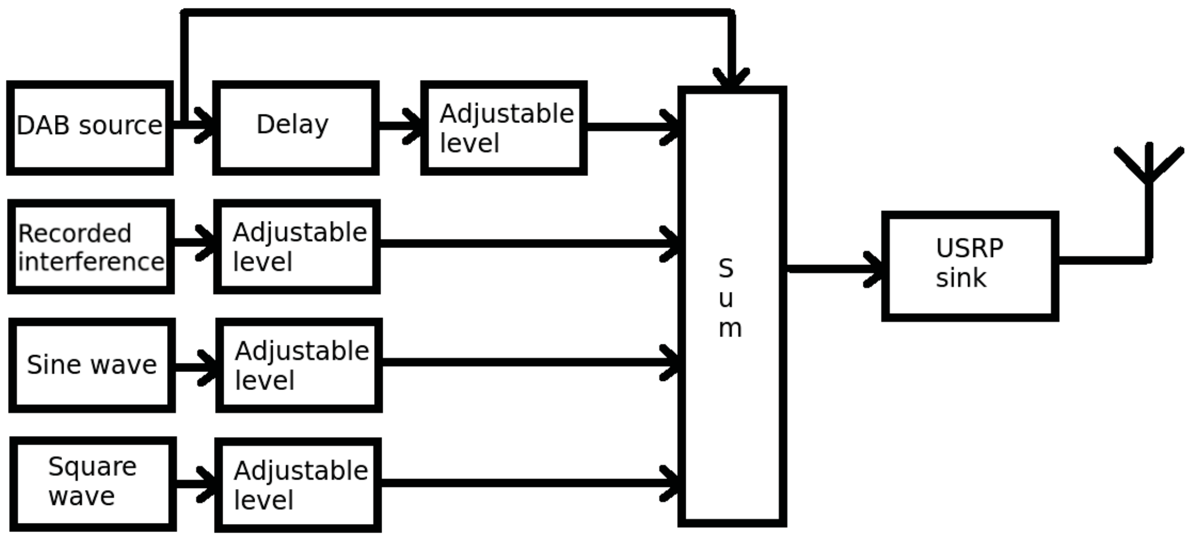


Figure 5.13.1: Block diagram of the interference injector. The GNU Radio flow graph can be found in annex D.

6. Results

It became clear at an early stage that several custom made blocks are needed to implement a DAB receiver with GRC. Blocks that definitively are missing:

- Frame synchronization and frequency correction that supports the DAB synchronization channel.
- Frequency deinterleaving- and differential decoder block that can handle vectors.
- Blocks for all stream manipulation and GUI.

Blocks that probably can be used as they are:

- USRP Source
- OFDM sampler
- FFT
- OFDM Serializer
- The constellation decoder in combination with a constellation object
- The UDP sink can be used to output the audio stream to an external media player

Other valuable results:

- The discoveries described in this report and the resulting GRC flow graphs may represent a good springboard for further research.
- A functional RF-recorder and player were made. These may, after some fine tuning, replace the large, heavy, expensive and power demanding RF-recorders / players used at Volvo today.
- A working interference injector was made.
- It has been proven how easy and cheap instruments like spectrum analysers can be made with SDR software and hardware. Many measurements done during product development are non verifying measurements and the GNU-radio instrumentation is more than enough in many cases. It is much worth that many engineers can have their own cheap instruments instead of sharing one expensive instrument.
- This project has contributed with knowledges about what can be easily done with GRC and what cannot.
- It was proven that VLC can be used to play MPEG layer 2 audio that origins from a data stream inside GNU-radio.
- Frame synchronization pulses were extracted and constellation plots became visible. The result wasn't perfect but it was valuable to find out the potential of the different blocks.

7. Discussion and conclusions

It is obvious that SDR will gain in popularity as a cheap and flexible development tool. This assumes that the software and hardware works well and is easy to use. During this project it was discovered how easy things can be made with SDR using GRC but also the opposite were experienced.

The concept with graphical programming and flow graphs is very user-friendly and gives a very good overview. GRC offers this for free but there are many things that could be improved. One of those things is the documentation and the examples. Some blocks are well documented and some doesn't have documentation at all. In general the documentation needs a major overview. Much of the progress was the result of days with trials and errors trying to find out exactly how different blocks behaved by probing inputs and outputs with different instrumentation and using appropriate test signals.

8. Further research

Except for creating all missing blocks and complete the DAB receiver a number of problems that came up during this project need to be investigated:

There were problems with the instrumentation provided with GNU-radio (both WX and QT). For example, when measuring the synchronization pulses with a time scope different results were obtained with the WX and the QT scope.

There were also problems related to the hardware / communication with the hardware. When using sample rates over 8 MHz the system became unstable and seemed to corrupt the FPGA software in the USRP. When that happened the solution was to re-power the USRP to force a software reload. The same thing happened more than one time when moving around the USRP when it was in operation which may indicate a bad cable or solder joints in the USRP.

Another problem was the selection of the transmission frequency. For example, it was possible to transmit a recorded DAB ensemble at 220 MHz and receive it on a DAB receiver but not at 220,352 MHz which corresponds to DAB channel 11C. The same phenomena occurred on other frequencies too. Typing the frequency in the form 220,352e6 or 220352000 gave the same result. When using real DAB frequencies the receiver in most cases was able to find some, but not all, of the services but it was impossible to play any audio. Shifting the DAB ensemble up and down within the baseband with different frequency offsets didn't solve anything. When transmitting on a working frequency like 220 MHz the receiver was able to compensate for frequency shifts made at baseband level even if it was shifted up 352 kHz generating a output frequency of 220,352 MHz. The conclusion is that this problem is related to GNU-radio or the USRP.

9. References

[1] ETSI EN 300 401 (2006-06): “Radio Broadcasting Systems; Digital Audio Broadcasting (DAB) to mobile, portable and fixed receivers”

[2] J. Dunlop and D.G. Smith: “Telecommunications Engineering” Third edition

[3] Ettus Research: “USRP B200/B210 Bus Series” b200-b210_spec_sheet.pdf (Product specification)

[4] Analog Devices: “RF Agine Transceiver AD9361 Data Sheet Rev. E”

[5] Ettus research – Product Detail [Internet]. Santa Clara: Ettus Research; 2015 [Cited 25 mar 2015]. Available from: http://www.ettus.com/product/details/UB210-KIT

[6] L. Bengtsson and B. Karlström: “Transformer – från jw till Wavelets” Edition 1:4

[7] GNURadio.org – GNURadioCompanion [Internet]. [Cited 15 apr 2015]. Available from: http://gnuradio.org/redmine/projects/gnuradio/wiki/GNURadioCompanion

[8] W. Fisher “Digital Video and Audio Broadcasting Technology” Third edition

[9] GNURadio.org – GNURadio [Internet]. [Cited 28 apr 2015]. Available from: http://gnuradio.org/redmine/projects/gnuradio

[10] GNURadio.org – Guided GNU Radio Tutorials [Internet]. [Cited 29 apr 2015]. Available from: http://gnuradio.org/redmine/projects/gnuradio/wiki/Guided_Tutorials

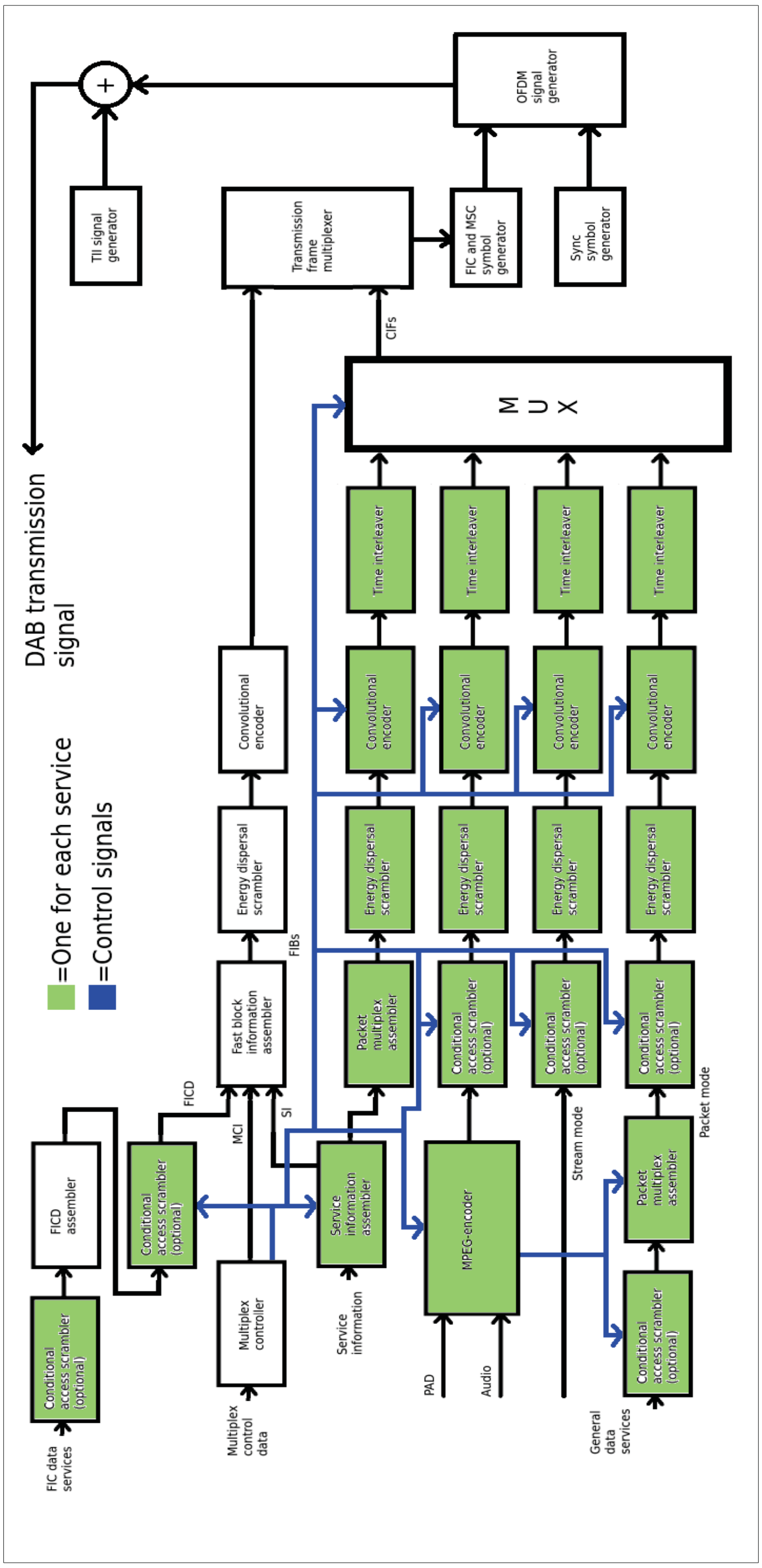
10. Annex A

Frequency table for VHF band III [8](p265).

Channel	**Center frequency (MHz)**
5A	174,928
5B	176,640
5C	178,352
5D	180,064
6A	181,936
6B	183,648
6C	185,360
6D	187,072
7A	188,928
7B	190,640
7C	192,352
7D	194,064
8A	195,936
8B	197,648
8C	199,360
8D	201,072
9A	202,928
9B	204,640
9C	206,352
9D	208,064
10A	209,936
10B	211,648
10C	213,360
10D	215,072
11A	216,928
11B	218,640
11C	220,352
11D	222,064
12A	223,936
12B	225,648
12C	227,360
12D	229,072
13A	230,784
13B	232,496
13C	234,208
13D	235,776
13E	237,488
13F	239,200

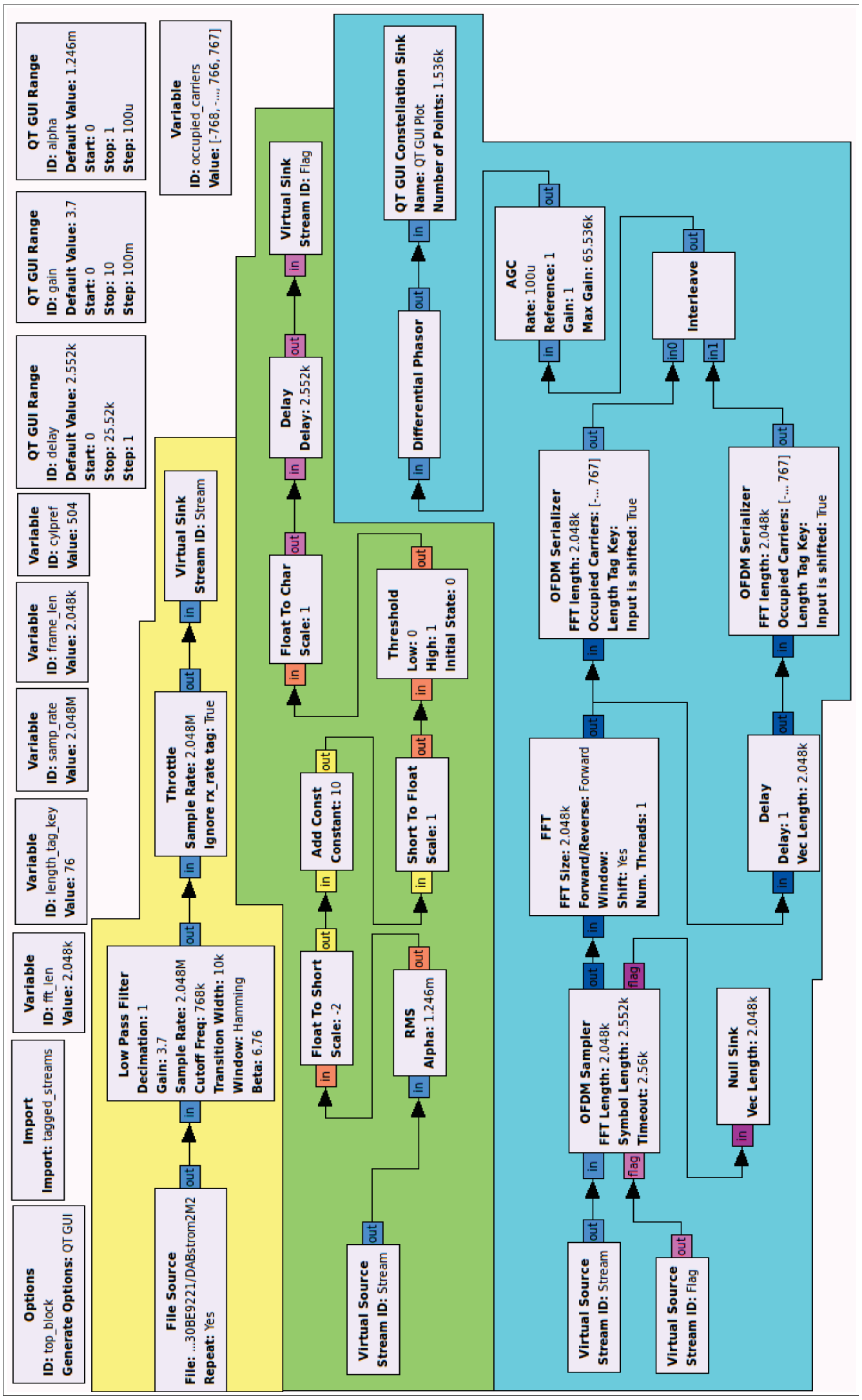
11. Annex B

Block diagram showing the generation of a DAB ensemble [1](p22).



12. Annex C

Flow graph of the test OFDM demodulator.



Options
ID: top_block
Generate Options: QT GUI

Import
Import: tagged_streams

Variable
ID: fft_len
Value: 2.048k

Variable
ID: length_tag_key
Value: 76

Variable
ID: samp_rate
Value: 2.048M

Variable
ID: frame_len
Value: 2.048k

Variable
ID: cylpref
Value: 504

QT GUI Range
ID: delay
Default Value: 2.552k
Start: 0
Stop: 25.52k
Step: 1

QT GUI Range
ID: gain
Default Value: 3.7
Start: 0
Stop: 10
Step: 100m

QT GUI Range
ID: alpha
Default Value: 1.246m
Start: 0
Stop: 1
Step: 100u

Variable
ID: occupied_carriers
Value: [-768, -..., 766, 767]

File Source
File: ...30BE9221/DABstrom2M2
Repeat: Yes

Low Pass Filter
Decimation: 1
Gain: 3.7
Sample Rate: 2.048M
Cutoff Freq: 768k
Transition Width: 10k
Window: Hamming
Beta: 6.76

Throttle
Sample Rate: 2.048M
Ignore rx_rate tag: True

Virtual Sink
Stream ID: Stream

Virtual Source
Stream ID: Stream

RMS
Alpha: 1.246m

Float To Short
Scale: -2

Add Const
Constant: 10

Short To Float
Scale: 1

Threshold
Low: 0
High: 1
Initial State: 0

Float To Char
Scale: 1

Delay
Delay: 2.552k

Virtual Sink
Stream ID: Flag

Differential Phasor

QT GUI Constellation Sink
Name: QT GUI Plot
Number of Points: 1.536k

Virtual Source
Stream ID: Stream

Virtual Source
Stream ID: Flag

OFDM Sampler
FFT Length: 2.048k
Symbol Length: 2.552k
Timeout: 2.56k

Null Sink
Vec Length: 2.048k

FFT
FFT Size: 2.048k
Forward/Reverse: Forward
Window:
Shift: Yes
Num. Threads: 1

Delay
Delay: 1
Vec Length: 2.048k

OFDM Serializer
FFT length: 2.048k
Occupied Carriers: [-... 767]
Length Tag Key:
Input is shifted: True

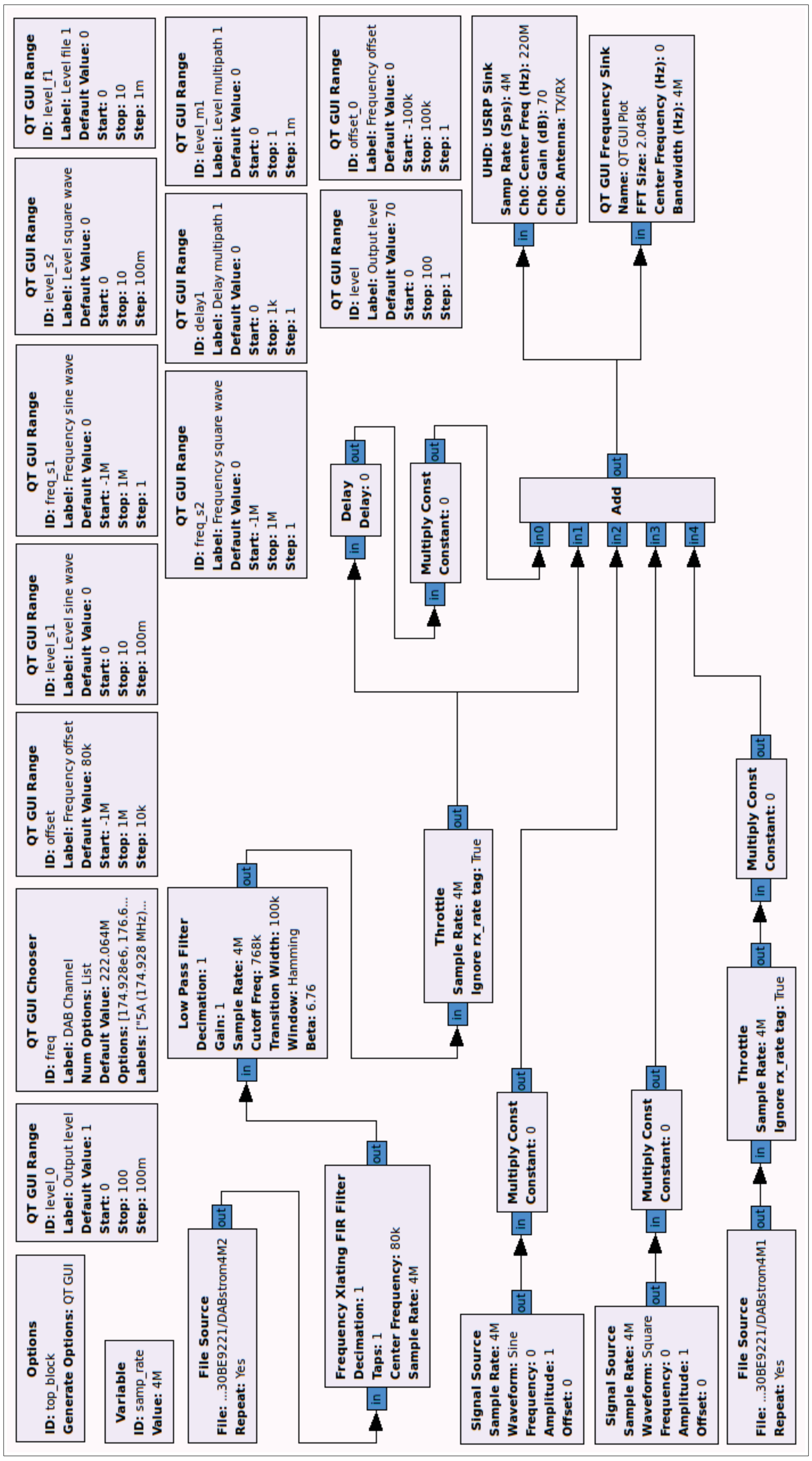
OFDM Serializer
FFT length: 2.048k
Occupied Carriers: [-... 767]
Length Tag Key:
Input is shifted: True

AGC
Rate: 100u
Reference: 1
Gain: 1
Max Gain: 65.536k

Interleave

13. Annex D

Flow graph of the interference injector.



Options
ID: top_block
Generate Options: QT GUI

QT GUI Range
ID: level_0
Label: Output level
Default Value: 1
Start: 0
Stop: 100
Step: 100m

QT GUI Chooser
ID: freq
Label: DAB Channel
Num Options: List
Default Value: 222.064M
Options: [174.928e6, 176.6...
Labels: ["5A (174.928 MHz)...

QT GUI Range
ID: offset
Label: Frequency offset
Default Value: 80k
Start: -1M
Stop: 1M
Step: 10k

QT GUI Range
ID: level_s1
Label: Level sine wave
Default Value: 0
Start: 0
Stop: 10
Step: 100m

QT GUI Range
ID: freq_s1
Label: Frequency sine wave
Default Value: 0
Start: -1M
Stop: 1M
Step: 1

QT GUI Range
ID: level_s2
Label: Level square wave
Default Value: 0
Start: 0
Stop: 10
Step: 100m

QT GUI Range
ID: level_f1
Label: Level file 1
Default Value: 0
Start: 0
Stop: 10
Step: 1m

Variable
ID: samp_rate
Value: 4M

File Source
File: ...30BE9221/DABstrom4M2
Repeat: Yes

Low Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 4M
Cutoff Freq: 768k
Transition Width: 100k
Window: Hamming
Beta: 6.76

QT GUI Range
ID: freq_s2
Label: Frequency square wave
Default Value: 0
Start: -1M
Stop: 1M
Step: 1

QT GUI Range
ID: delay1
Label: Delay multipath 1
Default Value: 0
Start: 0
Stop: 1k
Step: 1

QT GUI Range
ID: level_m1
Label: Level multipath 1
Default Value: 0
Start: 0
Stop: 1
Step: 1m

Frequency Xlating FIR Filter
Decimation: 1
Taps: 1
Center Frequency: 80k
Sample Rate: 4M

Delay
Delay: 0

QT GUI Range
ID: level
Label: Output level
Default Value: 70
Start: 0
Stop: 100
Step: 1

QT GUI Range
ID: offset_0
Label: Frequency offset
Default Value: 0
Start: -100k
Stop: 100k
Step: 1

Multiply Const
Constant: 0

Throttle
Sample Rate: 4M
Ignore rx_rate tag: True

Signal Source
Sample Rate: 4M
Waveform: Sine
Frequency: 0
Amplitude: 1
Offset: 0

Multiply Const
Constant: 0

UHD: USRP Sink
Samp Rate (Sps): 4M
Ch0: Center Freq (Hz): 220M
Ch0: Gain (dB): 70
Ch0: Antenna: TX/RX

Add

Signal Source
Sample Rate: 4M
Waveform: Square
Frequency: 0
Amplitude: 1
Offset: 0

Multiply Const
Constant: 0

QT GUI Frequency Sink
Name: QT GUI Plot
FFT Size: 2.048k
Center Frequency (Hz): 0
Bandwidth (Hz): 4M

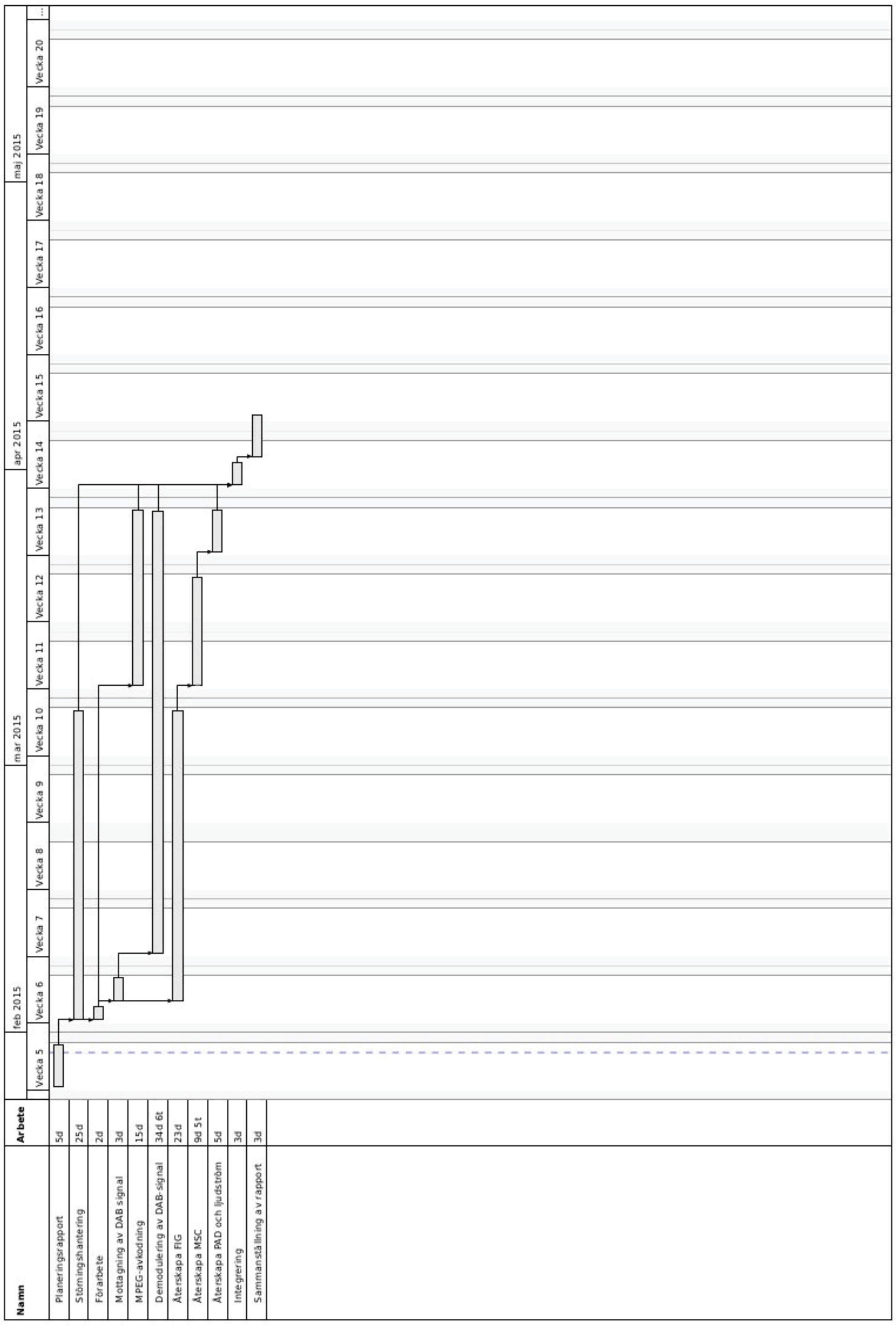
File Source
File: ...30BE9221/DABstrom4M1
Repeat: Yes

Throttle
Sample Rate: 4M
Ignore rx_rate tag: True

Multiply Const
Constant: 0

14. Annex E

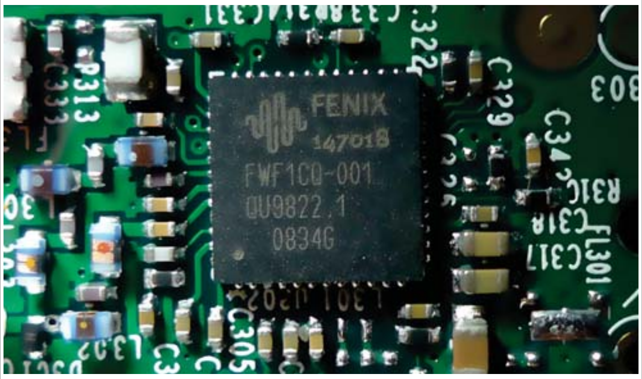
Gantt chart of the planning report.



DAB radio design and implementation

FREDRIK FORSBERG

Software designed radio (SDR) is a technique where the signal processing of the radio signal is done with software instead of hardware. This approach gives the designer maximum flexibility and it is also possible to use the same hardware for many different applications which is a benefit in many ways. It is not a new concept but it has become more widely used as a result of lower price of computer power. SDR:s can be made to run on different chips, mostly DSP:s but there are also software development kits for the PC. One of these development kits is GNU-Radio with the graphical shell GRC (Gnu Radio Companion). One focus in this project was to investigate how much of a DAB receiver that can be implemented in GRC using the standard blocks that comes with the package. Consumer receivers are built up of chip-sets that provides all DAB functions. A SDR model of a receiver gives access to all signal processing stages. This is usable when investigating interferences impact on a DAB receiver.



In most consumer DAB-receivers all functionality is provided by a few customized chips. This picture shows the Fenix 14701B-chip which functions as the tuner.