

Implementation of a DRM+ transmitter in the GNU Radio software radio framework

Bachelorarbeit

Felix Wunsch

Hauptreferent : Prof. Dr.rer.nat. Friedrich Jondral
Betreuer : Jens Elsner, M. Sc.

Beginn : 02.04.2012
Abgabe : 02.10.2012

Erklärung

Ich versichere hiermit, dass ich meine Bachelorarbeit selbständig und unter Beachtung der Satzung der Universität Karlsruhe (TH) zur Sicherung guter wissenschaftlicher Praxis in der aktuellen Fassung angefertigt habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und wörtlich oder inhaltlich übernommene Stellen als solche kenntlich gemacht.

Karlsruhe, 2. Oktober 2012

Felix Wunsch
Karlsruher Straße 38
76139 Karlsruhe

Abstract

This thesis presents an open source implementation of the digital broadcasting standard Digital Radio Mondiale (DRM) and its extension DRM+ in the Software Defined Radio (SDR) framework GNU Radio.

DRM operates in the AM bands below 30 MHz whereas DRM+ is intended for the use in the VHF range up to 174 MHz. Both are explicitly designed to coexist with existing analog technology and to replace it in the future. It offers immense benefits such as data stream and text news services, multiple services on one carrier and better performance in low SNR environments. The maximum achievable data rate is 186.4 kbps for DRM+.

The new module gr-drm is fully integrated into GNU Radio Companion (GRC) to make comprehension of the transmitter as easy and intuitive as possible. Various preconfigured flow graphs are supplied that can directly be used in combination with a Universal Software Radio Peripheral (USRP) and a daughterboard matching the desired frequency range. No deep knowledge about DRM is required and live transmissions should be possible without any further configuration of the flow graphs.

New blocks have been coded as generic and modular as possible to ensure high reusability for other programmers. The new blocks include e. g. an interleaver and a generic bit-to-symbol mapper. This document also serves as documentation for the released code that is publicly available on www.github.com/kit-cel/gr-drm [1].

The DRM transmitter has been optimized for low CPU usage and was successfully tested against a commercial receiver and Dream [2], a software receiver for DRM. DRM+ is also implemented, but still untested. This will be done in the near future when the appropriate hard- and software are available.

Zusammenfassung

Diese Arbeit präsentiert eine quelloffene Implementierung des digitalen Rundfunkstandards Digital Radio Mondiale (DRM) und seiner Erweiterung DRM+ im Software Defined Radio (SDR) Framework GNU Radio.

DRM wurde für Übertragungen unter 30 MHz entwickelt, während DRM+ das VHF-Band bis 174 MHz unterstützt. Sowohl DRM als auch DRM+ sind explizit dafür ausgelegt, parallel zu bestehenden, analogen Übertragungen eingesetzt werden zu können. Auf diese Weise soll ein sukzessiver Übergang zur digitalen Rundfunktechnik ermöglicht werden, da diese immense Vorteile bietet. Beispiele sind Datenstreams und Textnachrichtenservices, mehrere Dienste auf einer Trägerfrequenz sowie ein besseres Verhalten bei niedrigem Signal-Rauschverhältnis. Die maximal erreichbare Nutzdatenrate beträgt bei DRM+ 186.4 kbps.

Das neue Modul gr-drm ist vollständig in GNU Radio Companion (GRC) integriert, um das Verständnis der Funktionsweise des Senders so einfach und intuitiv wie möglich zu gestalten. Außerdem sind mehrere Flowgraphs enthalten, die in Kombination mit einem Universal Software Radio Peripheral (USRP) sowie einem passenden Daughterboard direkt ausgeführt werden können. Es ist kein spezielles Wissen über den DRM-Standard erforderlich und Übertragungen sollten ohne weitere Anpassung der Flowgraphs möglich sein.

Neue Signalverarbeitungsblocks wurden so generisch und modular wie möglich gehalten, um eine hohe Wiederverwendbarkeit für andere Programmierer zu erreichen. Unter den neuen Blocks sind z. B. ein Interleaver sowie eine generische Zuordnung von Bits zu Symbolen. Die vorliegende Arbeit soll auch als Dokumentation für den Code, der auf www.github.com/kit-cel/gr-drm [1] verfügbar ist, dienen.

Der DRM Sender wurde auf niedrige Prozessorlast hin optimiert und erfolgreich gegen einen kommerziellen Empfänger sowie Dream [2], eine Softwareimplementierung eines DRM-Empfängers, getestet. DRM+ wurde ebenfalls implementiert, ist aber noch ungetestet. Dies wird in naher Zukunft, wenn die erforderliche Hard- bzw. Software verfügbar ist, erfolgen.

Contents

1. Introduction	1
1.1. Why digital broadcasting?	1
1.2. What is DRM?	2
2. The DRM Standard	3
2.1. Overview	3
2.2. Transmitter Structure	5
2.3. Source Coding	5
2.4. Multiplexer	6
2.5. Energy Dispersal	8
2.6. Channel Coding	8
2.7. Cell Interleaver	9
2.8. OFDM Cell Mapping	10
2.8.1. Reference cells	10
2.9. Orthogonal Frequency Division Multiplexing (OFDM) signal generation . .	11
3. Implementation in GNU Radio	13
3.1. About GNU Radio	13
3.2. Design Paradigm	13
3.3. Design Process	14
3.4. New Module gr-drm	14
3.5. Transmitter Initialization	15
3.5.1. Determining Parameters	15
3.5.2. transm_params Description	17
3.6. New Basic GNU Radio Blocks	18
3.6.1. drm_add_tailbits_vbvb	18
3.6.2. drm_audio_decoder_svb	18
3.6.3. drm_audio_encoder_fvb	18
3.6.4. drm_cell_interleaver_vcvc	19
3.6.5. drm_cell_mapping_vcvc	19
3.6.6. drm_generate_fac_vb	19
3.6.7. drm_generate_sdc_vb	20
3.6.8. drm_interleaver_vbvb	20
3.6.9. drm_partitioning_vbvb	20
3.6.10. drm_punct_vbvb	21

3.6.11. <code>drm_qam_map_vbvc</code>	21
3.6.12. <code>drm_scrambler_vbvb</code>	22
3.7. New Hierarchical Blocks	22
3.8. GNU Radio from a Beginner's Point of View	23
4. Performance	25
4.1. Testbed	25
4.2. Profiling	26
4.3. Results	26
5. Conclusion	28
A. GRC Transmitter Flow Graph	30

1. Introduction

1.1. Why digital broadcasting?

Up to now, analog AM (Amplitude Modulation) and FM (Frequency Modulation) receivers are still very popular, although the technology they are based on is quite outdated. But the reasons are simple: These radios are less complex than their digital counterparts, inexpensive and known to work reliably.

Also the number of broadcasters, and that is what in the end defines the attractiveness of the technology for the consumer, is very large whereas digital broadcast is still a niche product. Another economical reason is that broadcasters and consumers would be obliged to invest in new hardware in order to be able to receive and transmit digital broadcast.

Obviously, all this makes the transition to the digital domain for both broadcasters and consumers less attractive. So why change at all?

The development of audio compression algorithms, e. g. MPEG-4 AAC (Advanced Audio Coding) [3], allows an enormous reduction of the audio bit rate while maintaining a high audio quality. The application of adaptable channel encoding and special modulation techniques offers further advantages and flexibility especially in low SNR environments.

The recent development e. g. in the smartphone field has shown that consumers increasingly request more and more complex features that cannot be provided by analog technology, e. g. video streams or text news. Digital technology offers the required flexibility to keep up with the consumers' needs.

Today, there are several competing standards for digital broadcasting available, one of them is Digital Radio Mondiale (DRM) [4].

1.2. What is DRM?

“DRM is the world’s only open standard, digital system for long-wave, medium-wave and short-wave and the VHF bands, including the FM bands, with the ability to use existing frequencies and bandwidth across the globe.” [5]

The inaugural transmission using the DRM standard took place in Geneva during the International Telecommunication Union (ITU)’s annual World Radio Conference in Geneva (Switzerland) on June 16, 2003. At that time, DRM only supported frequencies below 30 MHz and was designed to replace the Amplitude Modulation (AM) broadcasts.

In 2005, the DRM consortium decided to extend the standard to work with frequencies up to 174 MHz. This addition was called DRM+. DRM+ transmissions use a fixed channel bandwidth of 100 kHz and thus fit exactly in the existing channel pattern for Frequency Modulation (FM) broadcasts around 100 MHz.

Both DRM and DRM+ are explicitly designed to be able to coexist with the existing analog broadcasting technologies. Thus a "soft" transition to the digital domain is possible and intended. For simplification, DRM and DRM+ will further be addressed as DRM and divided up into DRM30 and DRM+ where it is needed.

An introduction to how the standard works is given in the next chapter.

2. The DRM Standard

2.1. Overview

The standard can roughly be divided up into two parts: DRM30 and DRM+. DRM30 covers the frequencies in the Long Wave, Medium Wave and Short Wave range whereas DRM+ is to be applied at frequencies higher than 30 MHz, up to 174 MHz.

A DRM stream consists of three logical channels. The Main Service Channel (MSC) conveys the payload whereas the Service Description Channel (SDC) offers additional descriptive data, e. g. the current time and date, the station label and some technical parameters that are needed to decode the MSC. The third channel, the Fast Access Channel (FAC), carries only the core information about the station but is very fast decodable and therefore suitable for station scanning.

In addition to the usual audio services DRM offers a wide range of features such as Multimedia Object Transfer (MOT) slide shows, an Electronic Program Guide (EPG), text news (called Journaline) and even low-quality videostreams. It is also possible to have multiple audio streams parallelly. Especially the text news is a very valuable addition as it is by nature perfectly suited for broadcast, uses only very little bandwidth and therefore does not impact the parallelly transmitted audio service(s). By specification, a maximum of four parallel services, audio or data, are allowed.

DRM is designed as OFDM system to cope with many different channel conditions. This is why transmitter parameters are highly customizable. They can be roughly classified in two types defining signal bandwidth and transmission efficiency. Whereas the first type determines the amount of used spectrum, the efficiency related type takes noise, multipath and Doppler effects into account and allows a trade-off between capacity and error robustness.

The most important parameters in this context are the Spectrum Occupancy (SO) controlling the signal bandwidth and the Robustness Mode (RM) that sets parameters related to the error robustness in different channel conditions. All stages of the transmitter/receiver chain are dependent on these parameters to optimize performance. An overview over the different RM and SO values is given in table 2.1 and 2.2.

Table 2.3 gives an impression of the basical physical layer parameters.

Robustness Mode	Description
A	Gaussian channels, with minor fading
B	Time and frequency selective channels, with longer delay spread
C	As robustness mode B, but with higher Doppler spread
D	As robustness mode B, but with severe delay and Doppler spread
E	Time and frequency selective channels (DRM+)

Table 2.1.: Robustness Modes

Spectrum Occupancy	Bandwidth
0	4.5 kHz
1	5 kHz
2	9 kHz
3	10 kHz
4	18 kHz
5	20 kHz (DRM30) / 100 kHz (DRM+)

Table 2.2.: Spectrum Occupancies

	DRM30	DRM+
Center frequency	< 30 MHz	30 Mhz ... 174 MHz
Bandwidth	4.5 kHz ... 20 kHz	100 kHz
Symbol duration	9.33 ms ... 24 ms	2.25 ms
Guard interval	2.66 ms ... 7.33 ms	0.25 ms
Frame duration	400 ms	100 ms
Bit rate	4.8 kbps ... 72 kbps	37.3 kbps ... 186.4 kbps

Table 2.3.: Basic physical layer parameters

2.2. Transmitter Structure

Fig. 2.1 shows a conceptual DRM transmitter flow graph¹. It clearly shows the three logical streams and how they are independently processed.

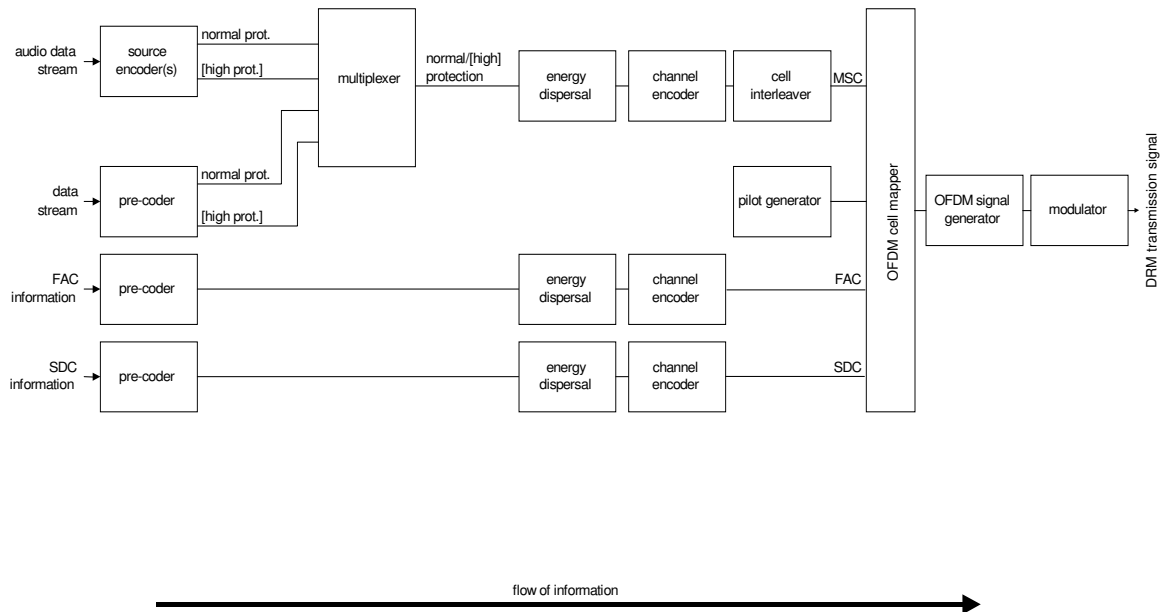


Figure 2.1.: Conceptual DRM transmitter flow graph [4, p. 15]

The single blocks will be discussed in the next sections.

2.3. Source Coding

DRM uses a subset of the MPEG-4 standard [3] for audio coding. There is one encoder for music, Advanced Audio Coding (AAC), and two for speech, depending on the available bit rate. For bitrates from 4 to 20 kbps (kilobit per second), Code Excited Linear Prediction (CELP) can be used and for very low bit rate applications at 2 or 4 kbps Harmonic Vector Excitation Coding (HVXC) can be applied.

All encoders compose audio super frames out of the encoded audio frames that can be mapped directly to the transmission frames as they are of the same duration for DRM30. In DRM+, a super audio frame fits exactly onto two transmission frames. This spares the receiver a separate synchronization to the audio payload.

¹Usage of all figures taken from the standard document has been permitted:

©European Telecommunications Standards Institute 2009. Further use, modification, copy and/or distribution are strictly prohibited. ETSI standards are available from <http://pda.etsi.org/pda/>.

The DRM AAC encoder uses the Low Complexity (LC) profile that can be combined with up to three additional modules to enhance the audio quality.

When encoding an audio signal it is desirable to keep a high audio bandwidth even at low bit rates. This can be achieved by the use of Spectral Band Replication (SBR). This tool analyzes the high frequency components of the full bandwidth signal and passes these as low bit rate side information to the receiver where these components can be reconstructed.

Parametric Stereo (PS) allows a stereo simulation out of a mono signal. Again, low bit rate data describing the stereo image is transmitted as side information. This tool is preferably applied where standard stereo transmissions cannot be afforded due to a limited bit rate.

Even multi-channel audio is possible with an extension named MPEG Surround (MPS). By intelligent encoding of the input signal a much lower data rate can be achieved than by simply transmitting all channels.

A conceptual DRM audio encoder block diagram can be seen in fig. 2.2.

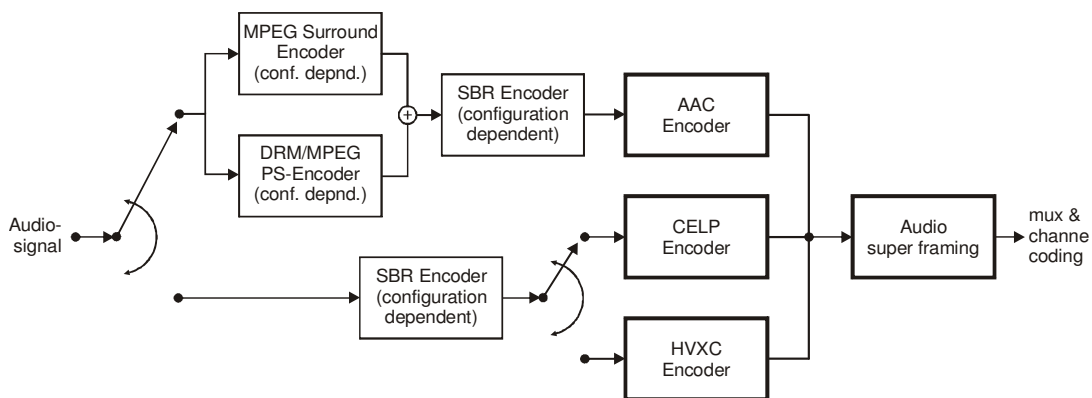


Figure 2.2.: DRM audio source encoding [4, p. 19]

As the audible effects of bit errors are not independent from the position in the bitstream, Unequal Error Protection (UEP) has been introduced. It allows defining a higher and a lower protected part to cope with this problem and increasing the audio quality by protecting the more sensitive bits stronger than the rest of the audio stream. UEP is only intended for the use with AAC and CELP.

2.4. Multiplexer

The multiplexer combines the audio and data streams to one single bit stream. In this process, data services can be placed in the higher or in the lower protected part allowing

different error protection for different services. An example of how three different streams (one audio with UEP and two data) can be mapped to a multiplex frame is shown in fig. 2.3.

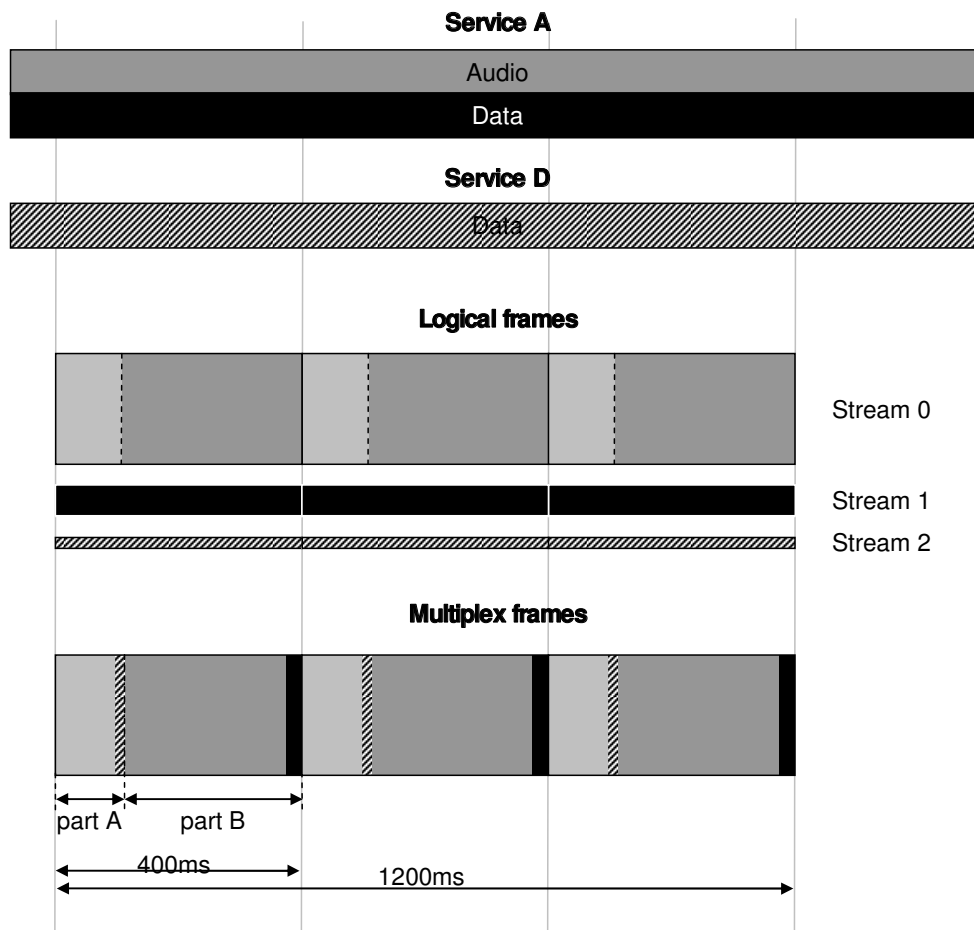


Figure 2.3.: Assigning streams to multiplex frames [4, p. 163]

The three simultaneously transmitted, continuous services are first broken up into a number of segments. If UEP is applied, the data is further divided up into the higher (A) and the lower protected part (B). One segment of each service is assigned to a transmission frame. As a last step, the parallel bitstreams are multiplexed into one bitstream that represents the actually transmitted bit sequence. In DRM30 there are typically three transmission frames that form a super transmission frame with a total length of 1.2 seconds.

2.5. Energy Dispersal

As systematic patterns might result in unwanted regularity affecting the transmitted signal or the signal processing negatively, bits are scrambled by the addition of a Pseudo Random Bit Sequence (PRBS). The aim is to break long chains of constant ones or zeros and to create a uniform distribution between the bits.

The PRBS sequence is generated with a Linear Feedback Shift Register (LFSR) that uses the following polynomial:

$$P(x) = x^9 + x^5 + 1 \quad (2.1)$$

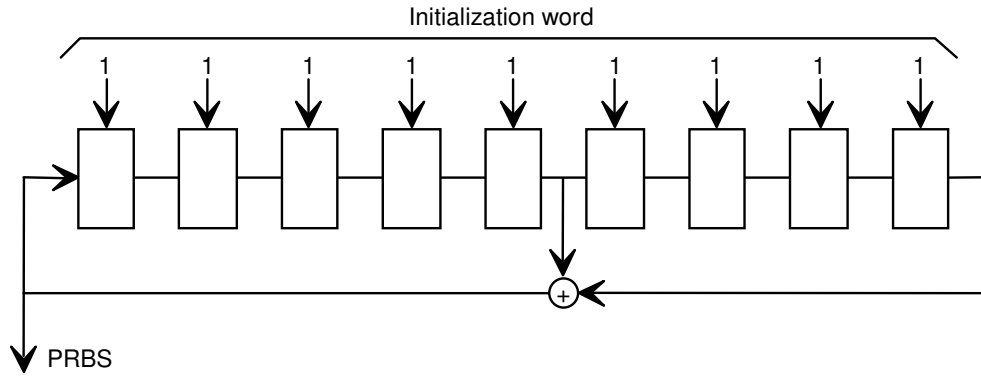


Figure 2.4.: PRBS generator using shift registers [4, p. 113]

2.6. Channel Coding

To optimize the performance of the coding, a multilevel coding scheme is used. This way the more error prone positions in the Quadrature Amplitude Modulation (QAM) mapping get a higher protection.

These different levels of error protection are realized by the use of different code rates. The core encoder (fig. 2.6) is a convolutional encoder with a mother code rate of $R = \frac{1}{6}$ that is combined with different puncturing schemes resulting in code rates up to $\frac{8}{9}$. The encoder polynomial in octal form is $133_8, 171_8, 145_8, 133_8, 171_8, 145_8$. It can be seen that the second half of the code word is a simple copy of the first half. The initial state as well as the final state are all-zeros.

After the puncturing bit interleaving is applied to ensure that no consecutive bits are mapped to the same symbol and thus improving robustness against burst errors.

As modulation scheme QAM is used (fig. 2.5). The number of signal points depends on the configuration and is either 4, 16, or 64. For 64-QAM there are two more hierarchical

modulation schemes available. These hierarchical modulation schemes introduce another level of error protection dividing the bitstream up into a Standard Protected Part (SPP) and a Very Strongly Protected Part (VSPP). DRM does not use Gray mapping. Especially when crossing the real or imaginary axis, multiple bits are inverted.

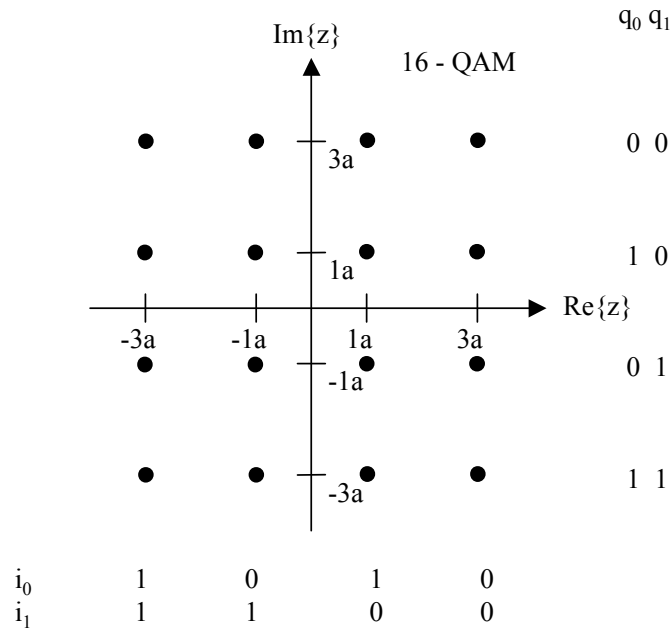


Figure 2.5.: 16-QAM constellation diagram [4, p. 131]

2.7. Cell Interleaver

This interleaver is only used for the MSC and performs a frequency-domain interleaving. It uses the same interleaving algorithm as the bit interleaver but also offers a “long interleaving” mode. In this mode, a convolutional scheme is used to increase the robustness of transmissions for channels suffering from severe time- and frequency-selective fading as it is typical for frequencies below 30 MHz. The interleaver delay is always an integer multiple of the length of a transmission frame. DRM30 uses a factor of $D = 5$, for DRM+ a value of $D = 6$ has been determined as the optimal tradeoff between performance and processing delay. In the case of DRM30 this results in a total delay of 2.4 seconds.

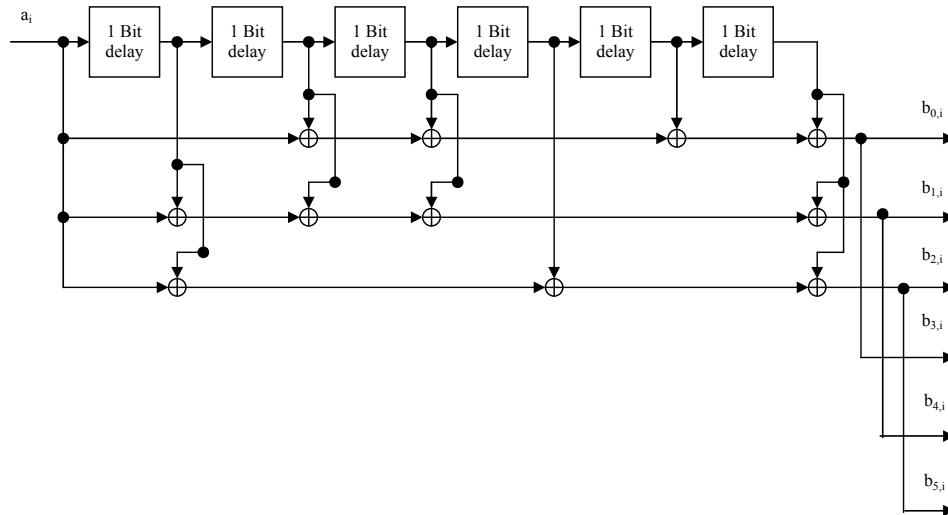


Figure 2.6.: Convolutional encoder [4, p. 119]

2.8. OFDM Cell Mapping

The OFDM cell mapping places the MSC, SDC and FAC cells on the time-frequency plane that is called transmission frame. A transmission frame consists of a concatenation of a certain number of OFDM symbols that is determined by the choice of the RM. Furthermore, multiple transmission frames are grouped to a super transmission frame.

2.8.1. Reference cells

In addition to the three channels, there are four different pilot or reference symbols that aid demodulation in the receiver.

The **frequency reference cells** represent continuous sine waves at 750 Hz, 2250 Hz and 3000 Hz. They are used for detection of a DRM signal and frequency offset correction. These cells are not present in RM E, i. e. DRM+.

Time reference cells can be found in every first symbol of a transmission frame and are therefore used for frame synchronization and for ambiguity resolution caused by the guard time correlation.

As QAM is used, **gain reference cells** are inserted to aid the coherent demodulation. This is why they are scattered all over the time-frequency plane to allow proper calculation of the channel coefficients. The cells located on the edge carriers are overboosted to improve performance at low Signal to Noise Ratio (SNR) levels.

The last type of reference cells are the **AFS** ones. Alternative Frequency Switching (AFS) is a special system in DRM that allows telling the receiver - if available - alternative frequencies where the same content is being transmitted. Thereby receivers have multiple frequencies at their disposal from which they can choose the one that offers the best reception quality.

Fig. 2.7 shows the position of the pilot and data cells on an exemplary super transmission frame (RM B, SO 3).

The different types of complex symbols are color-coded as follows:

- black: DC carrier (not used)
- light grey: MSC cells
- blue: SDC cells
- red: FAC cells
- yellow: time reference cells
- green: frequency reference cells
- purple: gain reference cells
- pink: boosted gain reference cells

2.9. OFDM signal generation

To convert the signal from the frequency domain to the time domain, the Inverse Discrete Fourier Transform (IDFT) or its more computation efficient version, the Inverse Fast Fourier Transform (IFFT), is used.

Typical for OFDM is also the insertion of a cyclic prefix for multipath handling. The duration varies with the choice of the RM. In DRM30, it can be up to 7.33 ms long and thus cover multipaths with a length of 2200 km. The cyclic prefix in DRM+ is fixed to a much shorter duration of 0.25 ms and therefore handles multipath components with a length of up to 75 km. This is nevertheless a sensible choice as the propagation conditions for frequencies above 30 MHz are different, explicitly not including reflections at the ionosphere that allow transmissions all around the globe for short wave radio stations.

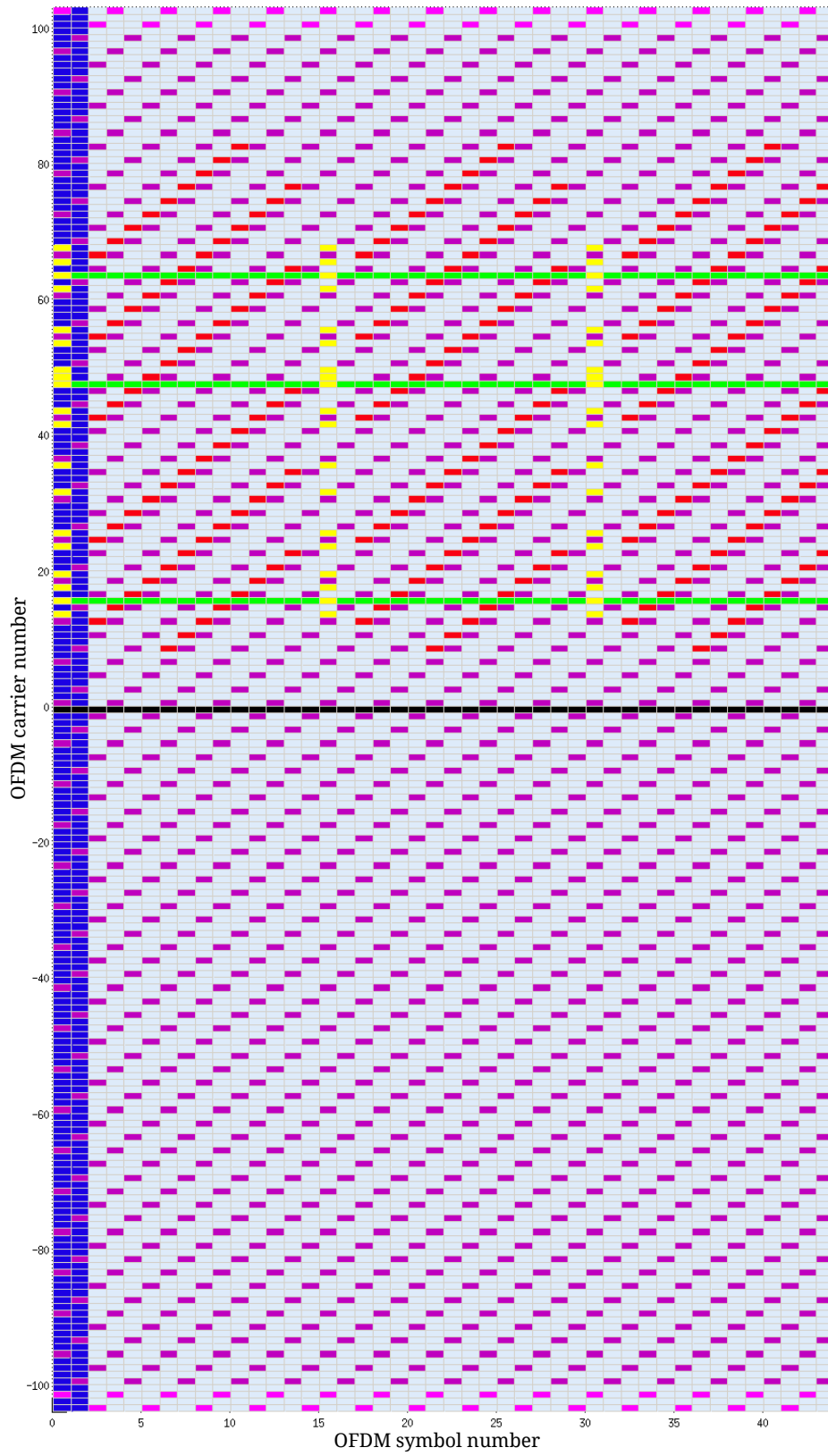


Figure 2.7.: Super transmission frame (RM B, SO 3)

3. Implementation in GNU Radio

3.1. About GNU Radio

GNU Radio is an open source framework for Software Defined Radios (SDRs). It offers a rich blockset of already fully integrated signal processing routines and is arbitrarily extendable. It also offers a graphical front-end called GNU Radio Companion (GRC) that allows visual programming by inserting and connecting blocks in a drag & drop manner. Further it features graphical sinks that allow live monitoring of the data being processed.

The signal processing part is mainly kept in C++, whereas connecting the flow graphs and other top-level tasks are carried out in Python. As interface between these two layers SWIG (Simplified Wrapper and Interface Generator) is used.

GNU Radio is constantly being developed further and has a very active community.

3.2. Design Paradigm

There were two major design rules when this transmitter was designed:

- Full integration into GRC.
- Maximizing reusability for other, possibly not DRM-related projects.

The integration into GRC was chosen as high priority because this project also has an educational character and is very well suited for demonstration. The visual clearness of the information flow definitely profits by a nice flow graph in GRC.

This is also why a vector-oriented approach has been chosen as it supports the thought of working on blocks of data as it is done in the DRM standard.

Following one of the core design concepts in GNU Radio, a high modularity, new blocks were coded as generic as possible in the hope of being of use for other programmers.

3.3. Design Process

As a first step, a DRM transmitter using a fixed configuration was implemented in Matlab to ensure a correct understanding of the standard. For testing purposes, an inverse transmitter was implemented, too. Then this was tested against Dream and a commercial DRM receive, the DR-111 by Chengdu Newstar Electronics. For further information please refer to <http://www.cdnse.com/products/dr111>. The device used for testing is a pre-production sample and therefore had some initial bugs, but many of them were resolved in recent firmware versions. A major bug that still exists is that the radio does not decode SO 4 and 5, i. e. bandwidths of 18 and 20 kHz. Also the display sometimes gets stuck and does not display text messages or station labels.

After successful testing the Matlab code was divided into single blocks to maximize the modularity. These blocks were ported to GNU Radio one by one, implementing the signal processing routines in C++ and the GRC representation in XML. Additionally, Python unit tests were assigned to each block taking reference values from the Matlab implementation.

For the setup of the build environment “gr_modtool.py” [6] was used so that it was possible to focus on the development of the actual signal processing.

3.4. New Module gr-drm

All source code was bundled in a new module: gr-drm. It contains many new GNU Radio blocks and some additional classes and routines that are used for calculating DRM-related parameters.

The transmitter parameters have a hierarchical structure. There are several “major” parameters that the rest is derived from. The most important are RM, SO and the mapping scheme used for the MSC. All other parameters such as the symbol duration, guard interval length, payload length etc. are derived from these. The calculation of all these variables is done before the actual flow graph is started in order to reduce the CPU usage later on.

The software is publicly hosted on <http://www.github.com/kit-cel/gr-drm> [1]. There are also fully configured flow graphs available that can be used as-is without any deep knowledge about DRM or even digital broadcasting at all.

3.5. Transmitter Initialization

3.5.1. Determining Parameters

The DRM transmitter configuration is determined by a wide variety of parameters. Most of these parameters depend on one or more other parameters and thus create a complex hierarchical structure. This allows the configuration of multiple low-level parameters such as the symbol duration and the guard interval length through higher-order parameters, in this special case the RM. Following this structure, it is possible to define a relatively small group of user-definable parameters that allow a complete configuration of the transmitter. In the receiver, the inverse is possible: Low-layer parameters, that are characteristic for a higher-level parameter allow the derivation of other parameters that have the same dependencies.

The top-level parameters are:

- RM
- SO
- UEP flag and number of higher protected bytes
- Type of audio codec and input audio sample rate
- MSC and SDC mapping scheme
- MSC and SDC code rate
- Long interleaving flag
- Station label and text message

In gr-drm, this structure is represented by several classes and child classes, e. g. for MSC, SDC and FAC parameters. There is one top-level class that expects all the “key parameters” as constructor arguments and instantiates subclasses that use these parameters to calculate the lower-layer parameters. These subclasses hold the actual transmission configuration. The purpose of this top-level class is mainly an improved convenience because many blocks need parameters from more than one subclass and like this, simply the top-level class can be passed to the block. The needed values are then extracted in the block’s constructor. All necessary parameter calculations are performed before the actual flow graph is started to keep CPU consumption low during transmission.

Fig. 3.1 illustrates the initializing process. The key parameters are passed from GRC to the top-level class *transm_params* that hands them down to the *config* class which performs some range and plausability checks and tries to fix invalid configurations. If the configuration passes these tests, the other subclasses are initialized by passing the *config* instance, that holds the - possibly corrected - key parameters.

To reduce redundancy, all *_params classes are derived from the same parent *global_params* class and its child class *channel_params* which itself has another child class *control_channel_params*.

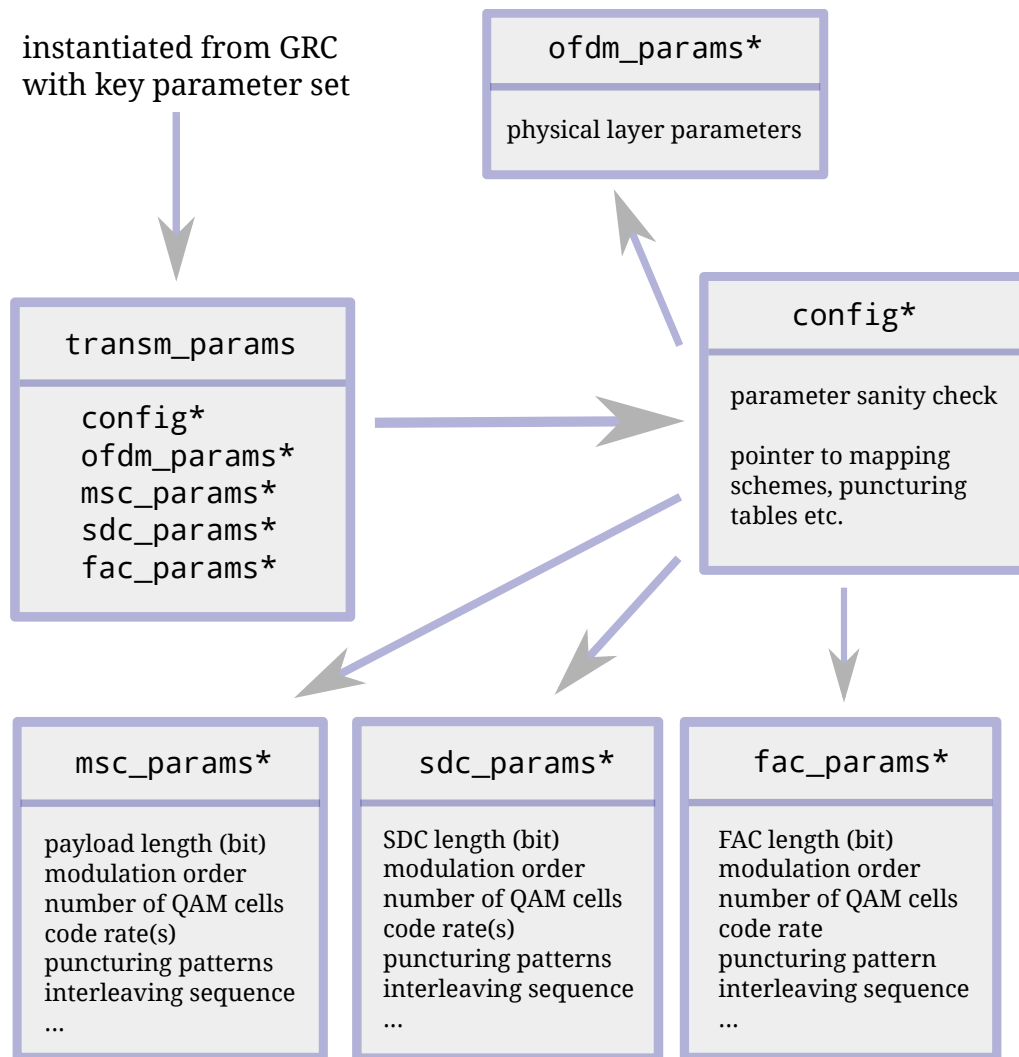


Figure 3.1.: Transmitter initialization

3.5.2. `transm_params` Description

Class definition:

```
class transm_params
{
    config* d_cfg;
    ofdm_params* d_ofdm;
    msc_params* d_msc;
    sdc_params* d_sdc;
    fac_params* d_fac;

public:
    config cfg();
    ofdm_params ofdm();
    msc_params msc();
    sdc_params sdc();
    fac_params fac();

    transm_params(unsigned short RM,
        unsigned short S0,
        bool UEP,
        unsigned int n_bytes_A,
        bool text,
        unsigned short msc_mapping,
        unsigned short msc_prot_level_1,
        unsigned short msc_prot_level_2,
        unsigned short sdc_mapping,
        unsigned short sdc_prot_level,
        bool long_interl,
        unsigned int audio_samp_rate,
        std::string station_label,
        std::string text_message);

    ~transm_params(){};
};
```

A list of valid parameter values for the constructor can be found in *gr-drm/include/config.h*.

3.6. New Basic GNU Radio Blocks

3.6.1. `drm_add_tailbits_vbvb`

Description

This block appends a given number of tailbits (zeros) at the end of the input vector.

Block API

```
drm_add_tailbits_vbvb (int vlen_in, int n_tailbits)
```

- `int vlen_in`: Input vector length
- `int n_tailbits`: Number of tail bits

3.6.2. `drm_audio_decoder_svb`

Description

Decodes MPEG-4 AAC-LC encoded data, outputs a stream of shorts (raw PCM samples). For the actual decoding, the FAAD2 library has been interfaced [7].

Block API

```
drm_make_audio_decoder_vbs (transm_params* tp)
```

- `transm_params* tp`: Transmitter parameters

3.6.3. `drm_audio_encoder_fvb`

Description

Audio source encoder using FAAC [8] to produce AAC samples (mono, LC, no SBR/PS). Input is a PCM stream (float), output is a vector of bits (unpacked) coded as unsigned characters.

Block API

```
drm_make_audio_encoder_fvb (transm_params* tp)
```

- `transm_params* tp`: Transmitter parameters

3.6.4. `drm_cell_interleaver_vcvc`

Description

Performs the cell interleaving. The long interleaving flag determines the interleaving mode.

Block API

```
drm_cell_interleaver_vcvc (std::vector<int> interl_seq,  
    bool long_interl, int depth)
```

- `std::vector<int> interl_seq`: Vector of interleaved indexes, its length denoting the input and output vector length
- `bool long_interl`: Flag determining interleaver mode
- `int depth`: number of consecutive vectors the convolutional scheme shall be applied to (only for long interleaving)

3.6.5. `drm_cell_mapping_vcvc`

Description

Performs mapping of MSC, SDC, FAC and pilot cells onto the OFDM super transmission frame.

Block API

65

```
drm_cell_mapping_vcvc (transm_params* tp, std::vector< int > input_sizes)
```

- `transm_params* tp`: Transmitter parameters
- `std::vector< int > input_sizes`: Vector denoting the vector lengths of the inputs (MSC, SDC, FAC)

3.6.6. `drm_generate_fac_vb`

Description

Generates the Fast Access Channel (FAC) data (with Cyclic Redundancy Check (CRC)).

Block API

`drm_generate_fac_vb (transm_params* tp)`

- `transm_params* tp`: Transmitter parameters

3.6.7. `drm_generate_sdc_vb`

Description

Generates the Service Description Channel (SDC) data (with CRC).

Block API

`drm_generate_sdc_vb (transm_params* tp)`

- `transm_params* tp`: Transmitter parameters

3.6.8. `drm_interleaver_vbvb`

Description

Interleaves a sequence of the same length as the interleaver sequence as specified in the sequence (representing the interleaved indexes).

Block API

`drm_interleaver_vbvb (std::vector<int> interl_seq)`

- `std::vector<int> interl_seq`: Vector of interleaved indexes. Its length also denotes the input and output vector length.

3.6.9. `drm_partitioning_vbvb`

Description

Partitions a vector in an arbitrary number of output streams.

Block API

`drm_partitioning_vbvb (unsigned int vlen_in, std::vector<int> vlen_out)`

- `unsigned int vlen_in`: Input vector length
- `std::vector<int> vlen_out`: Vector of output vector lengths. Its size denotes the number of output ports, the single values the length of the respective port. The sum of all elements has to equal the input vector length.

3.6.10. `drm_punct_vbvb`

Description

Punctures a vector of (unpacked) bits according to `punct_pat_1`. For the last `num_tailbits` bits `punct_pat_2` is used.

Block API

`drm_punct_vbvb (std::vector<unsigned char> punct_pat_1,
std::vector<unsigned char> punct_pat_2, int vlen_in,
int vlen_out, int num_tailbits)`

- `std::vector<unsigned char> punct_pat_1`: Main puncturing pattern. Zeros denote bits to be kept, ones bits to be dropped. The pattern is repeated until the tail bits are reached.
- `std::vector<unsigned char> punct_pat_2`: Tail bit puncturing pattern. No repetition, its length must equal the number of tail bits.
- `int vlen_in`: Input vector length
- `int vlen_out`: Output vector length
- `num_tailbits`: Number of tail bits

3.6.11. `drm_qam_map_vbvc`

Description

General QAM mapping. Input is a matrix with `bits_per_symbol` (indexing through decimal interpretation of the bits) rows and 2 columns (I/Q). Works only for symmetrical signal constellations.

Block API

```
drm_qam_map_vbvb (const float map_table[][2], int bits_per_symbol,  
                  int vlen_out, int n_inputs)
```

- const float map_table[][2]: Mapping table
- int bits_per_symbol: Number of bits per symbol
- int vlen_out: Output vector length
- int n_inputs: Number of inputs

3.6.12. drm_scrambler_vbvb

Description

Scrambles the input vector with a PRBS sequence. Scrambler gets reset after block_len samples.

Block API

```
drm_scrambler_vbvb (unsigned int block_len)
```

- unsigned int block_len: Number of bits before scrambler gets reset.

3.7. New Hierarchical Blocks

The DRM channel encoder consists of large number of basic blocks and implements the structure shown in fig. 3.2. To maintain a clear flow graph structure, hierarchical blocks were created for the different encoder versions. There are up to three so-called levels in the encoder that could not be implemented in one single hier_block. The parameters they expect are just the collection of the parameters of the underlying basic blocks.

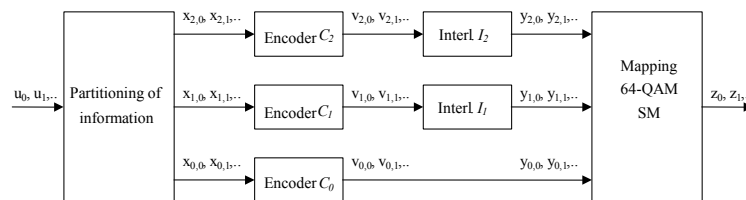


Figure 3.2.: DRM multilevel encoder [4, p. 115]

3.8. GNU Radio from a Beginner's Point of View

At this point I want to present my personal experience with GNU Radio as a beginner.

The main advantage for me was the data stream architecture GNU Radio offers. It allows to code blocks (more or less) without caring about how the samples get in and out. This way more time can be spent on the actual signal processing instead of thinking about buffering or scheduling of the sample flow.

GNU Radio Companion (GRC) is also a very convenient tool for the visualization of flow graphs and even visual programming. My flow graphs benefitted especially from the concept of hierarchical blocks (called "hier_blocks") that group multiple blocks into one top-layer block. This helped a lot in terms of avoiding chaos. Very recently and in response to my request at the GNU Radio Conference 2012 in Atlanta, support for looking into the underlying structure of hier_blocks from a higher layer and dynamic reloading of XML block definitions from within GRC has been added. Also resizing of blocks would be a great feature to add because - especially if they are passed pointers - they can get very large, blocking much space on the working plane. A good example for a flow graph with unnecessarily large blocks is the gr-drm transmitter flow graph itself (fig. A.1) that can be found in appendix A.

As GNU Radio is an open source project it is possible to access every block's implementation. This is especially for beginners a very good way to learn how to write efficient and nicely formatted code because it is a bit like looking over the professional programmer's shoulder.

Generally, the reusability of GNU Radio blocks is very high. So it was possible to perform the whole OFDM operation in GRC just by clicking the respective, already available blocks together.

Unfortunately there is also a downside. The documentation is very sparse. There exist automatically generated ones for C++ and Python but as they only list classes, their functions and parameters they are of little help for beginners that do not even exactly know what they are looking for. Also the system with C++ for signal processing, Python for top-level flow graph tasks and SWIG as the glue inbetween is quite complex and sometimes hard to understand for a beginner.

But nevertheless GNU Radio is worth all the time it takes to get up to speed as it offers a wide variety of possibilities and only few constraints. I personally found it - after some time - very convenient and very well suited for my project.

A probably good way for beginners to start with GNU Radio development is to read all the "Getting started" and "Using GNU Radio" material on <http://www.gnuradio.org>. Then experimenting with the graphical front-end GRC might be a good idea to learn what is possible with GNU Radio and what is already available. There is e. g. a FM receiver for

GRC available that uses only blocks that are shipped with a standard GNU Radio installation. But it is almost inevitable that sometime a point will be reached where a special functionality is needed that is not yet implemented in GNU Radio. This is a good time to start with the *gr-howto-write-a-block* tutorial that is well explained in the “Developing GNU Radio” section on the website.

From there on, the best way to get deeper into GNU Radio is to look at the source code and probably to modify existing blocks. This is by the way a reason not to use binary installations, even if they are available and very easy to install. But installing from source is not very complicated either. There is a build script called *build-gnuradio* that performs a complete installation of GNU Radio and UHD, the driver software for the USRP radio front-ends, without any user interaction. For the creation of new out-of-tree modules *gr_modtool.py* [6] is very recommendable as it creates a lot of skeleton code and saves the user setting up all the build system related code. The script allows creating new modules as well as adding and removing blocks from it very conveniently.

4. Performance

For a software transmitter or receiver being run in the background on a normal desktop PC or notebook it is of great importance to be CPU and memory efficient and thus to impact the running system as little as possible. This was also a major design rule in this project and the results will be presented in this chapter.

4.1. Testbed

All tests were performed on the following system:

- Intel i5-2520M dual core 2.5 GHz
- GNU Radio 3.6.1
- Ubuntu 12.04 (64 bit)
- USRP N210 with BasicTX

The transmitter was configured as follows:

- `drm_transmitter_64sm_4.grc`
- RM A, SO 3 (10 kHz bandwidth)
- Long interleaving
- Text message application
- 24 kHz input audio file
- 26.56 kbps AAC mono stream

4.2. Profiling

For finding bottlenecks in the unoptimized code, profiling tools were used, primarily “valgrind” [9]. This tool executes the program on a simulated CPU and records which function was called how often, from whom and how much time was spent executing it.

“KCachegrind” [10] was developed to conveniently display the logs produced by valgrind. It even creates annotated source code with call counters for every line. Another very useful feature is the possibility to produce call graphs showing how the different functions and their (sub-)calls are linked together.

4.3. Results

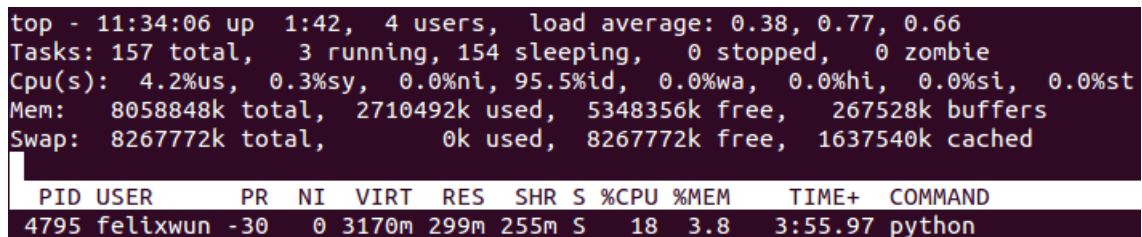
After applying the tools discussed in the previous section, CPU consumption could be reduced by approximately 50 %.

The remaining CPU load is to about 70 % caused by the FAAC encoder library. Therefore, a new open source version of the Fraunhofer Institute’s AAC encoder for Android will be integrated in the hope of better CPU efficiency. It also supports SBR and PS enhancing the audio quality at the same bit rate.

The test configuration showed about 20 % CPU consumption on one of four logical cores resulting in a system-wide average load of about 5 %. But unfortunately, almost 300 MB of buffers were allocated.

For memory alignment reasons the buffers are allocated with a certain, system-dependent granularity. In conjunction with the vector-based approach this leads to the allocation of far more memory than actually needed. Using internal buffers for the blocks working on the largest vectors might alleviate this problem and will be tested in the near future.

Fig. 4.1 shows a screenshot from “top” (a tool to display running Linux tasks). In fig. 4.2 a list of functions, ordered by their CPU consumption, is given (column “Self”).



```
top - 11:34:06 up 1:42, 4 users, load average: 0.38, 0.77, 0.66
Tasks: 157 total, 3 running, 154 sleeping, 0 stopped, 0 zombie
Cpu(s): 4.2%us, 0.3%sy, 0.0%ni, 95.5%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 8058848k total, 2710492k used, 5348356k free, 267528k buffers
Swap: 8267772k total, 0k used, 8267772k free, 1637540k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
4795	felixwun	-30	0	3170m	299m	255m	S	18	3.8	3:55.97	python

Figure 4.1.: Screenshot from “top”

Incl.	Self	Called	Function	Location
■	25.27	■ 23.09	29 272 WriteReorderedSpectral...	libgnuradio-drm.so
■	38.90	■ 13.01	14 636 AACQuantize	libgnuradio-drm.so
	8.26	8.26	3 785 729 QuantizeBand	libgnuradio-drm.so
	9.47	6.88	8 968 306 __ieee754_pow_sse2	libm-2.15.so: e_pow.c
	3.52	3.52	4 254 524 __ieee754_log_sse2	libm-2.15.so: e_log.c
	3.47	3.47	1 118 074 ccomplex_dotprod_sse	libgnuradio-core-3.6.1git.so.0.0.0
	3.04	3.04	1 147 drm_punct_vbvb::gener...	libgnuradio-drm.so
	2.74	2.74	6 545 929 PutBit	libgnuradio-drm.so
	2.58	2.58	5 182 573 __exp1	libm-2.15.so: e_exp.c
	2.14	2.14	1 267 gr_unpack_k_bits_bb::w...	libgnuradio-core-3.6.1git.so.0.0.0
	1.96	1.96	1 111 486 enqueue_bits_dec(unsig...	libgnuradio-drm.so
	1.88	1.88	1 002 156 kf_work'2	libgnuradio-drm.so
	2.48	1.78	32 388 enqueue_crc(unsigned c...	libgnuradio-drm.so
	1.76	1.76	1 368 788 CalcBits	libgnuradio-drm.so
■	100.00	1.48	802 <cycle 9>	python2.7
	1.44	1.44	997 298 __memmove_ssse3_back	libc-2.15.so: memcpy-ssse3-back.S
	1.28	1.05	1 148 050 _int_malloc	libc-2.15.so: malloc.c, arena.c
	10.51	1.05	8 968 306 pow	libm-2.15.so: w_pow.c
	0.89	0.89	227 drm_qam_map_vbvb::w...	libgnuradio-drm.so
	1.59	0.87	2 269 566 __fread_chk	libc-2.15.so: fread_chk.c, libioP.h
	0.87	0.87	463 151 OutputBits	libgnuradio-drm.so
	0.87	0.86	2 337 TnsEncode	libgnuradio-drm.so
	2.61	0.84	14 636 NoiselessBitCount	libgnuradio-drm.so
	0.82	0.82	448 drm_scrambler_vbvb::w...	libgnuradio-drm.so
	0.81	0.81	8 532 098 memcov_ssse3_back	libc-2.15.so: memcov-ssse3-back.S

Figure 4.2.: Functions ordered by CPU consumption

5. Conclusion

Within the scope of this thesis, a functional DRM transmitter has been implemented and successfully tested against available software and hardware receivers. DRM+ is also implemented but still untested. This will soon be done when the appropriate testing software is available that already has been requested.

The software called *gr-drm* supports a wide range of transmitter configurations. It is also fully integrated into GNU Radio Companion (GRC), the graphical frontend for GNU Radio offering various preconfigured flow graphs to make a quick transmitter setup possible. Two major design targets were modularity and reusability to make the flow of information clearly visible and maximize the reusability of the code for other programmers.

Nevertheless, there are still features to be implemented. The current major constraints are:

- Only one mono audio service allowed
- Unequal Error Protection (UEP) not available
- Hierarchical mapping schemes not available
- No reconfiguration during runtime possible

So far, only one bug is known where the text message accompanying the audio stream gets corrupted for a special configuration (RM A, SO 5). These issues will be addressed in the near future as well as the implementation of a recently released AAC encoder library developed by the Fraunhofer Institute. This implementation offers more encoder features such as Spectral Band Replication (SBR) and is expected to be more CPU efficient.

The long-term goal following this project is the development of a DRM receiver in GNU Radio. This will hopefully allow a very low priced receiver setup for DRM+ using a computer and an RTL2832 SDR stick that is currently available for less than 20 USD. A complete transmitter/receiver chain in GNU Radio also offers a good opportunity for demonstrating the capabilities of the framework.

One of the biggest obstacles in the spread of digital broadcasting in general is that the public does only know very little about its advantages. To make the development and production of e. g. DRM radios profitable, first of all a demand for this product has to be aroused. Once consumers realize the benefits of digital broadcasting, analog services will be much less attractive.

There is already a significant number of broadcasters using digital transmission, e. g. Voice of Russia, British Broadcasting Corporation (BBC) World Service and many more, although they mostly also provide their short-wave services in analog AM.

One might ask why in times of mobile internet access there is still a need for digital broadcasting. In comparison to internet streaming services, broadcasting has the advantage of producing no additional traffic regardless of the number of recipients. Furthermore, it is a valuable source of information where no high-speed internet or no internet at all is available or it is too expensive due to roaming fees.

With CPU power getting cheaper and devices smaller from year to year it is increasingly feasible to integrate a digital broadcast receiver even into smartphones where demodulators for analog FM broadcast are already very common.

A very recent development that could set a lot into motion is that DRM could soon become the national digital broadcasting standard of Brazil [11]. This would surely boost the public recognition immensely and finally motivate manufacturers to produce low-cost receivers. The final governmental announcement is expected to be in December.

A. GRC Transmitter Flow Graph

Fig. A.1 shows one of the preconfigured flow graphs for GRC that come with gr-drm. Its structure is very similar to the standard's flow graph (fig. 2.1) and allows an easy transfer from the standard document to this implementation.

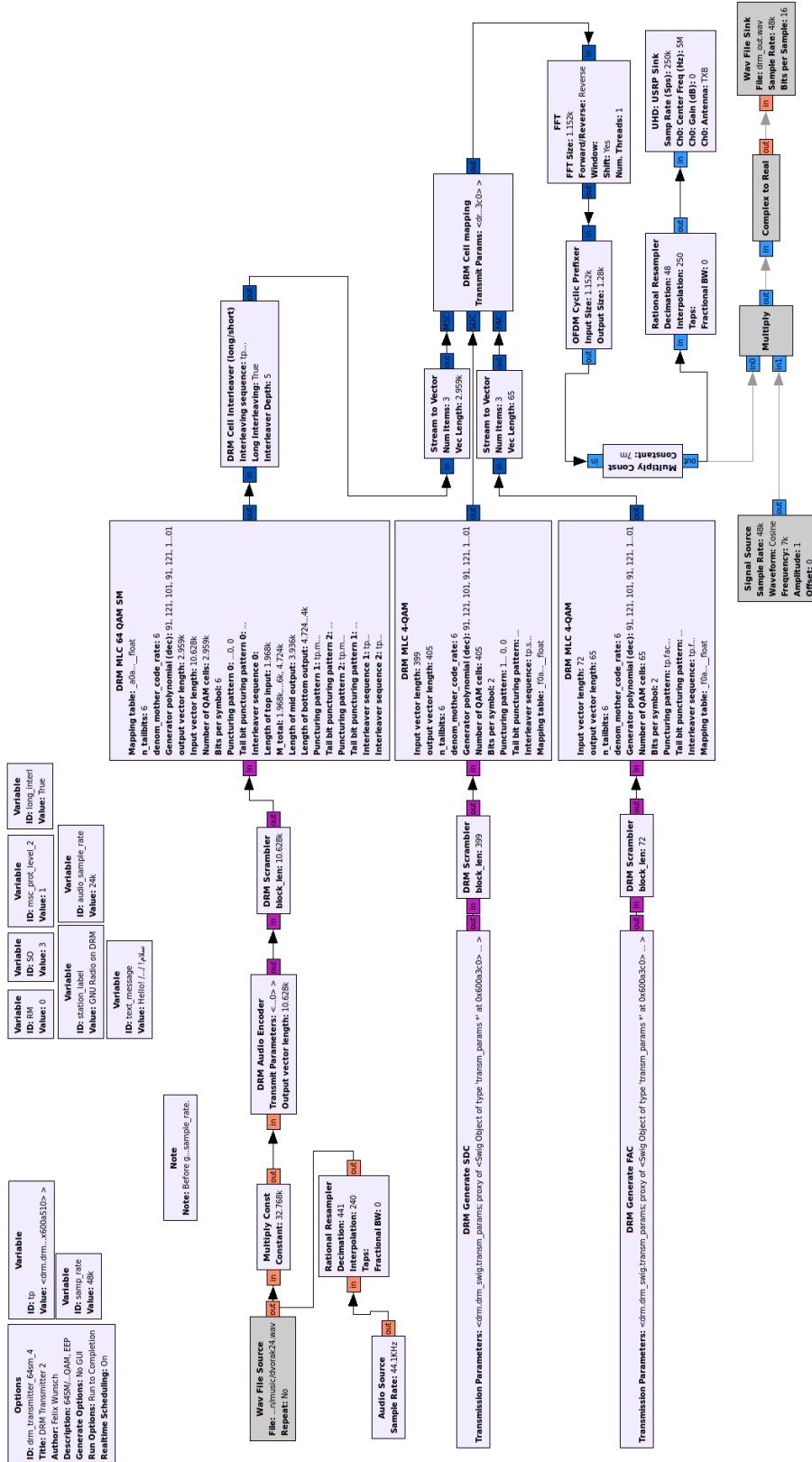


Figure A.1.: gr-drm transmitter flow graph (RM B, SO 3)

Bibliography

- [1] gr-drm Repository. <http://www.github.com/kit-cel/gr-drm>.
- [2] Dream Website on sourceforge.net. <http://www.drm.sourceforge.net>. Accessed 2012-09-10.
- [3] "MPEG-4 part 3, (ISO/IEC 14496-3:2009)," International Organization for Standardization, International Electrotechnical Commission, Tech. Rep., 2009.
- [4] "Digital Radio Mondiale (DRM); System Specification (ES 201 980)," European Standards Organisation, Tech. Rep., 2009.
- [5] Official Website of the DRM consortium. <http://www.drm.org>. Accessed 2012-09-05.
- [6] gr_modtool Repository. <http://www.github.com/mbant/gr-modtool>. Accessed 2012-09-17.
- [7] FAAD2 Website. <http://www.audiocoding.com/faad2.html>. Accessed 2012-09-14.
- [8] FAAC Website. <http://www.audiocoding.com/faac.html>. Accessed 2012-09-14.
- [9] Valgrind Homepage. <http://www.valgrind.org>. Accessed 2012-09-15.
- [10] KCachegrind Homepage. <http://www.kcachegrind.sourceforge.net>. Accessed 2012-09-15.
- [11] DRM Brasil Website.
<http://www.drm-brasil.org/en/content/abert-congress-19th-%E2%80%93-93-21st-june-%E2%80%93-brazil>. Accessed 2012-09-29.

Acronyms

AAC	Advanced Audio Coding
AFS	Alternative Frequency Switching
BBC	British Broadcasting Corporation
AM	Amplitude Modulation
CELP	Code Excited Linear Prediction
CRC	Cyclic Redundancy Check
DRM	Digital Radio Mondiale
EPG	Electronic Program Guide
FAC	Fast Access Channel
FM	Frequency Modulation
GRC	GNU Radio Companion
HVXC	Harmonic Vector Excitation Coding
IDFT	Inverse Discrete Fourier Transform
IFFT	Inverse Fast Fourier Transform
ITU	International Telecommunication Union
LC	Low Complexity
LFSR	Linear Feedback Shift Register
MOT	Multimedia Object Transfer
MPEG	Moving Pictures Experts Group
MPS	MPEG Surround
MSC	Main Service Channel
OFDM	Orthogonal Frequency Division Multiplexing
PRBS	Pseudo Random Bit Sequence
PS	Parametric Stereo

QAM	Quadrature Amplitude Modulation
RM	Robustness Mode
SBR	Spectral Band Replication
SDC	Service Description Channel
SDR	Software Defined Radio
SPP	Standard Protected Part
SNR	Signal to Noise Ratio
SO	Spectrum Occupancy
SWIG	Simplified Wrapper and Interface Generator
UEP	Unequal Error Protection
USRP	Universal Software Radio Peripheral
VHF	Very High Frequency
VSPP	Very Strongly Protected Part