MASTER

Time synchronising in Digital Audio Broadcasting receivers

van Loon, P.A.C.J.

Award date:
1996

Link to publication
Time Synchronising in Digital Audio Broadcasting Receivers

master thesis

P.A.C.J. van Loon

21st June 1996
Abstract

In the current information era, all kinds of information are communicated. One method of communication is radio broadcast. The most used technical means until now has been AM and FM reception.

Digital Audio Broadcasting or DAB is a digital radio standard for transmitting CD quality stereo audio together with all kinds of extra information. It could be called the multi-media radio, for also video and computer data can be transmitted via DAB. Furthermore, DAB is intended for mobile error-free reception that discards well-known reception problems like multi-path, fading and Doppler effects.

The theory behind the Digital Audio Broadcasting system is investigated to provide a background for understanding time synchronisation in DAB receivers.

A high-level system modelling and simulation tool, DSP Station is used to describe a DAB system of encoder, channel and decoder. This DAB system description is setup in order to study time synchronisation in DAB receivers.

DSP Station is evaluated for its use as a high level system modelling and simulation tool, based upon the experiences of the author and several other DSP Station users. Several aspects in DSP Station need improvements in order to get a full-fledged time-saving, design-supporting CAD/CAE tool.

Several methods for time synchronisation are looked at and this author has proposed a new promising time synchronisation algorithm for implementation in a new DAB receiver chip-set.

This proposal has made it into the DAB V3 receiver chip-set that will be available at the end of this year. Field tests then can show the better performance due to this algorithm, relative to previous used algorithms.
# Contents

1  Introduction .................................................. 1

1  Digital Audio Broadcasting System Theory ................... 3

2  Introduction .................................................. 5

2.1  History .................................................... 5

2.2  Concepts ................................................... 6

   2.2.1 Digital Broadcasting for Mobile Reception ............... 7

   2.2.2 Audio Bandwidth Reduction .................................. 11

   2.2.3 Additional Features ........................................ 11

3  Analysis ...................................................... 15

3.1  System View ............................................... 15

3.2  Sender ..................................................... 16

   3.2.1 Services Layer ............................................ 16

   3.2.2 Channels Layer ........................................... 17

   3.2.3 Frames Layer ............................................. 19

   3.2.4 Symbols Layer ............................................ 20

   3.2.5 Signals Layer ............................................ 21

3.3  Channel ................................................... 24

   3.3.1 Determining the Channel Impulse Response ............... 25

   3.3.2 Probability Distributions for Channel Parameters .......... 27

3.4  Receiver .................................................. 29

   3.4.1 Signals Layer ............................................ 29

   3.4.2 Symbols Layer ............................................ 31

   3.4.3 Frames Layer ............................................ 32
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.4.4 Channels Layer</td>
<td>32</td>
</tr>
<tr>
<td>3.4.5 Services Layer</td>
<td>33</td>
</tr>
<tr>
<td>3.5 The DAB System, an Overview</td>
<td>33</td>
</tr>
<tr>
<td>II DSP Station High Level System Modelling and Simulation</td>
<td>37</td>
</tr>
<tr>
<td>4 Introduction</td>
<td>39</td>
</tr>
<tr>
<td>5 DSP Station</td>
<td>41</td>
</tr>
<tr>
<td>5.1 Design Route Covered by DSP Station</td>
<td>41</td>
</tr>
<tr>
<td>5.2 Description of Used Tools</td>
<td>42</td>
</tr>
<tr>
<td>5.2.1 Bold Browser</td>
<td>42</td>
</tr>
<tr>
<td>5.2.2 Design Manager</td>
<td>44</td>
</tr>
<tr>
<td>5.2.3 Design Architect</td>
<td>45</td>
</tr>
<tr>
<td>5.2.4 DSP Lab</td>
<td>48</td>
</tr>
<tr>
<td>6 DAB System Modelling and Simulation</td>
<td>51</td>
</tr>
<tr>
<td>6.1 Introduction</td>
<td>51</td>
</tr>
<tr>
<td>6.2 Original Bench</td>
<td>51</td>
</tr>
<tr>
<td>6.3 Time Synchronisation Sheet</td>
<td>52</td>
</tr>
<tr>
<td>6.4 Sender Components</td>
<td>53</td>
</tr>
<tr>
<td>6.4.1 Data Generator</td>
<td>53</td>
</tr>
<tr>
<td>6.4.2 Differential Modulator</td>
<td>54</td>
</tr>
<tr>
<td>6.4.3 Encoder</td>
<td>55</td>
</tr>
<tr>
<td>6.5 Channel Components</td>
<td>55</td>
</tr>
<tr>
<td>6.5.1 Channel</td>
<td>55</td>
</tr>
<tr>
<td>6.6 Receiver Components</td>
<td>57</td>
</tr>
<tr>
<td>6.6.1 Front-End</td>
<td>57</td>
</tr>
<tr>
<td>6.6.2 Cordic</td>
<td>57</td>
</tr>
<tr>
<td>6.6.3 Decoder</td>
<td>58</td>
</tr>
<tr>
<td>6.6.4 Differential Demodulator</td>
<td>58</td>
</tr>
<tr>
<td>7 DSP Station Evaluation</td>
<td>61</td>
</tr>
<tr>
<td>7.1 Expectations of the Philips ADC concerning DSP Station</td>
<td>61</td>
</tr>
</tbody>
</table>
7.2 Experiences of ing. B. van Steenbrugge ........................................ 61
7.3 Experiences of other users .......................................................... 62
7.4 Experiences of the author ........................................................... 62
7.4.1 Design Transfer ................................................................. 63
7.4.2 Design Specification .......................................................... 63
7.4.3 Design Simulation .............................................................. 64
7.4.4 Simulation Result Analysis ...................................................... 64
7.4.5 General ................................................................. 64
7.5 Evaluation .............................................................. 65

III DAB Fine Time Synchronisation Optimisation ........................................ 67

8 Introduction ............................................................. 69

9 Analysis ................................................................. 71
9.1 Frequency Synchronisation ....................................................... 71
9.1.1 Coarse Frequency Synchronisation ..................................... 71
9.1.2 Fine Frequency Synchronisation ........................................ 71
9.2 Time Synchronisation .......................................................... 72
9.2.1 Coarse Time Synchronisation ............................................. 72
9.2.2 Fine Time Synchronisation .............................................. 72

10 Design ................................................................. 75
10.1 Conditions for Optimal Fine-Time Synchronisation ....................... 75
10.1.1 Window Positioning for One Signal Path ................................. 75
10.1.2 Window Positioning for Multiple Signal Paths ....................... 76
10.1.3 Optimal Positioning of Window ......................................... 76
10.1.4 Calculation of Centre of Gravity ..................................... 78
10.2 Weigh Functions .............................................................. 79
10.3 Adjustable Weigh Function .................................................... 83
10.4 Validness of the Adjustable Weigh Function ............................. 86

11 Results ............................................................. 89
Introduction

Society is unthinkable without communication. Modern man’s way to establish the exchange of information has grown from the telegraph via telephone, radio, television, to all kinds of modern communication means. Radio broadcasting makes it possible to reach millions of people with one message. Quality of radio transmission has evolved, together with the diversity in content of the messages.

In the information era new interests for a broader information content have arisen. Multimedia is the magic mantra. Digital radio opens the horizon to unlimited transmission of unlimited messages of unlimited content.

Maybe with this in mind, a new radio broadcasting norm has been devised; Digital Audio Broadcasting or DAB. Philips is one of the participants in a European effort to establish that norm as a de-facto standard in the broadcasting industry, replacing current AM and FM receivers.

At the Philips Research Lab and at the Philips Advanced Development Centre experience has been gained in developing a chip-set for DAB receivers. Currently a third generation chipset is almost available on the market.

At the Philips Advanced Development Centre specifications for that chipset have been designed. The author of this report has done research on synchronisation algorithms for DAB reception with the use of a high-level DSP CAD/CAE tool. This research has been done as basis for the author’s master thesis and was initiated from the Digital Systems section, a part of the department of electrical and information engineering at the Eindhoven University of Technology.

This report can be divided in three parts. The first part contains an analysis of the Digital Audio Broadcasting system concept. The second part contains an evaluation of the use of the CAD tool DSP Station for fitness of use at the Philips Advanced Development Centre. The final part contains analysis and design of time synchronising mechanisms in DAB receivers.
Part I

Digital Audio Broadcasting System Theory
2

Introduction

In this chapter we will briefly discuss the history of Digital Audio Broadcasting and give a short explanation of the concepts used in Digital Audio Broadcasting, followed by a short overview of the complete Digital Audio Broadcasting system.

2.1 History

In the first half of this century the radio was introduced. Broadcasting companies emerged and sent their programs to the aether that were picked up by receivers and listened to by a public that grew to millions and millions. Other companies started developing receivers for that large market. During the years several organisations agreed upon standards for transmission and for reception.

In the late 1950's this led to the since then widespread used FM standard. The FM standard was originally intended for stationary receivers in combination with a directional antenna. It has however been more and more used in car radios. Its use in a mobile receiver in combination with a omni-directional rod antenna gives a decrease in the quality of the received signal.

In 1982 Philips introduced Compact Disc. It was the first consumer product that offered superior audio quality by digital processing of information stored on such CD's. It has been a great success and has pushed the analog record out of the market. Besides its audio quality it offered the customer more convenience and ease-of-use than the traditional record.

With the introduction of the CD a new era began, the era of digital audio. It was to be expected that one day digital radio would appear, and it came. In 1990 Digital Satellite Radio was introduced. This is a system suitable for reception of digital audio transmitted by a satellite. It is intended however for stationary use only and requires a large bandwidth and is therefore not suited for use in a car.

Meanwhile another system for digital radio had been demonstrated, although in its experimental stage: Digital Audio Broadcasting. In 1988 at the World Administrators Radio Conference the first DAB system was shown to the participants by its inventors of the Centre Commun d'Etudes de Télédiffusion et Télécommunications in France. It used a bandwidth of 8 MHz and was that large it fitted only in the back of a car. Although in its experimental stage it was designed for both digital audio and mobile reception.
Introduction

Since that event several other companies from all over the world saw its advantages and participated in its further developments. In 1991 a second generation DAB receiver was shown at the North American Broadcasters conference in Las Vegas. By that time the use of bandwidth was already reduced to 3.5 MHz. In 1992 the World Broadcasting Union agrees on the need of a DAB standard and to allocate a frequency slot for satellite transmission to DAB. In that year the first European Broadcasters Union DAB newsletter appeared. In 1993 third generation receivers became available within the EUREKA project 147 with coding and modulation according to the preliminary standard. In 1994 the standard was established by the European Telecommunications Standards Institute and the first generation DAB channel decoder chip set was available from the JESSI AE89 project. In 1995 Philips introduced the DAB 452 receiver, the first integrated DAB receiver based on this ETSI standard. At the IFA in Berlin several DAB demonstrations were given.

Digital Audio Broadcasting is aimed to be a successor to FM. The advantages of DAB are: CD audio quality, even in a mobile environment, operational comfort and numerous possibilities for additional services such as multi-language programs, paging, continuous weather and traffic information and audio-visual advertising. Further it has the potential for Multi-Media broadcasting, where audio-, data- and video-streams are combined in one channel and the user can interactively select parts of that information stream.

Future plans include large field tests in several European countries in 1996, the availability of a DAB channel decoder chip for consumer products in the same year, official DAB broadcasting following up the field tests in 1997 and the introduction of a DAB car receiver probable for less then 1000 DM in that year.

2.2 Concepts

The desire for mobile reception of stereo audio at CD quality and with additional services is tempered by technological problems.

The known major problems with mobile reception are the constant changing characteristics of the transmission channel caused by the rapidly changing position of the receiving antenna, electro-magnetic interference from domestic and industrial sources, and the reception of multiple waves originating from one sender caused by reflections of the original wave.

The problem with the desired audio quality is its bandwidth. If one assumes sampling of audio at 44.1 kHz with 16 bits per sample as needed for CD quality, the bit rate for both the left and right channel equals \(2 \cdot 16 \cdot 44.1 \cdot 10^3 = 1.4112 \cdot 10^6\) bits per second. For mobile, portable reception this is feasible, but expensive.

We will now look at the solutions found for dealing with the problems occurring in mobile reception. After that we will address the solutions to the problem of large bandwidth. We will end this section with a discussion of additional features and what conditions these impose upon the design to properly implement them.
2.2.1 Digital Broadcasting for Mobile Reception

The mentioned problems with mobile reception can be classified in several effects that can be mathematically modelled. The electro-magnetic interference causes burst-like errors in the channel, they can be dealt with by proper channel coding. The reception of multiple waves together with the problem of a mobile antenna is a more serious problem, which needs further investigation.

In a configuration of one sender and a stationary receiver, the emitted waves travel at the speed of light in air in all directions. In a mountainous terrain some of these waves will be reflected. In an urban area, nearby the receiver, waves are reflected by buildings or scattered by objects that are not simple flat reflectors. The antenna therefore receives a sum of delayed and attenuated waves as depicted in figure 2.1. This can lead to channel selectivity causing frequency dependent amplitude- and phase distortion of a signal transmitted through such a channel.

![Figure 2.1: Reception of multiple waves](image)

When the receiver is moving the antenna receives a sum of signals that are all frequency shifted in some way by the Doppler effect.

It has been shown (e.g. in [4]) that the channel described above can be reduced to a model of M independent Rayleigh channels, each associated with a delay $T_m$ with respect to an arbitrary origin. The delays $T_m$ have an exponential distribution.

Several observations have been made that lead to conditions that should be met to efficiently transmit information along such a channel:

- to reduce the selective effect of the channel, the period of one transmitted symbol should be much larger than the longest delay of a wave. This reduces however the data rate.

- to increase the amount of information per symbol, and thus the data rate, one could use frequency multiplexing to have several narrow-band channels along which the symbol
is transmitted. These narrow-band channels have such a reduced selectivity that they can be considered non-selective.

- to reduce influence of narrow-band channels onto each other independent of the speed of the vehicle, the frequency spacing between these channels should be above a threshold dependent on the average delay of reflected waves.

- by frequency interleaving these channels, they attain a further reduced selectivity.

- by time interleaving symbols the independence between two consecutive symbols can be increased and thus channel effects can be reduced. This has no effect when the vehicle isn’t moving.

- the use of convolutional coding and Viterbi decoding, further enhances the robustness of the system against transmission errors.

A very efficient way to implement a frequency multiplexing scheme is to use Orthogonal Frequency Division Multiplexing. If one considers a finite duration constant amplitude sinusoidal signal, its frequency spectrum has the shape of a $\text{sinc}(x)$ curve$^1$. At equidistant points, determined by the duration of the signal, this curve has zero-crossings except at one point. One can compose a spectrum of $\text{sinc}(x)$ curves translated along the frequency axis such that at equidistant points exactly one of the curves has a non-zero value. Therefore at equidistant places at the frequency axis, this spectrum can be considered as a collection of independent channels. In figure 2.2 an Orthogonal Frequency Division Multiplexing spectrum with six channels is depicted with one channel drawn thicker, $f_S$ is the carrier spacing. In figure 2.3 the resulting shape of the spectrum is depicted. In both pictures the central frequencies are marked.

![Figure 2.2: OFDM Spectrum](image)

In the time domain the Orthogonal Frequency Division Multiplexing spectrum translates to a sum of constant amplitude sinusoidal signals with equidistant frequencies. As a result this gives a large collection of signals onto which symbols can be mapped. In figure 2.4 the sinusoidal signals belonging to each channel are shown, $T_S$ is the symbol duration, $f_i$ are the six central frequencies. Note that $f_S = 1/T_S$.

$^1 \text{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$
By an appropriate choice of the symbol duration, one can set the spacing between the sub-channels at a sufficient distant to remove Doppler-effects in the frequency domain. Frequency interleaving can be simply done by changing the mapping of symbols onto signals.

A problem with Orthogonal Frequency Division Multiplexing at the receiver side is that for proper decoding the orthogonality of the sub-channels should be preserved. By Inter Symbol Interference this orthogonality is disturbed. A solution to this is to introduce a guard interval, a period of time that is prepended to the transmitted signal, and that is thrown away at the receiver. The prepended signal is a cyclic extension of the original signal. Although this leads to a slightly less efficient transmission, reducing Inter Symbol Interference in this way has clearly advantages for synchronisation at the receiver. In figure 2.5 three delayed waves are depicted with a corresponding channel impulse response. The useful symbol period is $t_s$, $\Delta$ is the guard interval duration and $T_S$ is again the (total) symbol period.

Errors can still occur caused by interference or temporal extreme channel situations. Therefore it's necessary to introduce a channel coding strategy. It has been shown ([3]) that codes of the convolutional type perform very well in combination with Orthogonal Frequency Division Multiplexing. The resulting system of channel coding and Orthogonal Frequency Division
Introduction

Multiplexing is called Coded Orthogonal Frequency Division Multiplexing. At the input of the channel coder one "sees" a virtually error-free channel with two states and a transition area; from perfect to non-perfect operation via a rapidly degrading transition.

At the receiver side synchronisation to the time basis used for transmission is necessary for proper demodulation. This is one of the reasons to transmit symbols on a frame basis with a so-called null-symbol as the first symbol. This null-symbol carries no energy and can be used to synchronise the receiver to the frame. Figure 2.6 shows the frame basis and the null-symbol. In this picture $T_{NULL}$ is the duration of the null-symbol and $T_F$ is the frame duration.

The first modulated symbol in a frame is a special synchronisation symbol and is called the TFPR symbol. This fixed symbol is used for time- and frequency synchronisation and as a phase reference symbol. A picture of both the TFPR symbol and $J - 1$ normal symbols are in figure 2.7. The null-symbol and the TFPR symbol can be seen as a frame multiplexed sub-channel, called the synchronisation channel.

This ends our discussion of the concepts used in transmission and brings us to the discussion of audio bandwidth.
2.2.2 Audio Bandwidth Reduction

As earlier explained, a reduction in the needed bandwidth for stereo audio at CD quality is needed.

In the past years research has led to compression techniques that are based upon psychoacoustic properties of human hearing. The ISO-MPEG layer II sound compression, also known as MUSICAM, is already used in other consumer products, like the Philips Digital Compact Cassette players and recorders. It accepts as raw input a pulse code modulated audio signal sampled at 48 kHz, using a bit rate of 1.536 Mbit per second.

This compression technique is roughly based upon two psychoacoustic properties; that of thresholding and of masking. Thresholding is the effect that sound below a certain pressure level threshold isn't perceived any more. This threshold is frequency dependent; both low and high frequencies need more sound pressure to be heard. Audio can be processed to analyse sound below that threshold and then discard that information.

Masking is the effect that a tone of certain frequency introduces a increase in threshold around that tone, such that other tones in the neighbourhood fall below that threshold, or are masked by that tone. This effect can also be used to discard certain parts of the frequency spectrum.

Discarding information together with appropriate coding of the useful information, leads to a reduction of the bit rate to about 256 kbit per second without hearable loss in audio quality. This is a reduction by a factor 6. It is possible to reduce this rate to even 32 kbit per second, but this leads to a reduction in audio quality. The sound quality at 32 kbit per second can be qualified as telephone-speech quality.

2.2.3 Additional Features

With the ongoing developments in Digital Audio Broadcasting, several additional features to be implemented were proposed. These additional features imposed certain additional conditions to be met within DAB. We will first look at the concept of DAB modes, followed by a discussion of extra services that can be offered by broadcasters. Finally we will discuss what design consequences the implementation of these features cause.

A major feature that was introduced, was the introduction of several broadcast modes. This allows for several special applications for Digital Audio Broadcasting. These modes have been labelled I, II and III. Currently a new proposed mode – mode IV – is investigated for standardisation. The modes have been designed with a particular application area in mind. All these modes use the same bandwidth per symbol.
Introduction

<table>
<thead>
<tr>
<th>Band</th>
<th>Frequency Range (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>47 — 68</td>
</tr>
<tr>
<td>II</td>
<td>88 — 108</td>
</tr>
<tr>
<td>III</td>
<td>174 — 230</td>
</tr>
<tr>
<td>IV/V</td>
<td>470 — 862</td>
</tr>
<tr>
<td>L</td>
<td>1452 — 1492</td>
</tr>
</tbody>
</table>

Mode I has a relatively long guard interval that is suited to eliminate relatively long delays in reflected waves. Mode I is intended for terrestrial broadcasting in a so called Single Frequency Network and for local-area broadcasting at frequencies in band I, II and III (for frequency allocation see table 2.1. In such a configuration all senders are operating on the same carrier frequency and are emitting the same DAB frames. In general, the signal of a sender can only be received in a certain region. Therefore several senders are used to cover for instance a country. To ensure proper reception of the signal at every position these regions do overlap. If these senders emit their programmes on a single frequency, interference occurs at these spots. The DAB system is protected against such interference, for the signals of the different senders can also be considered as reflections of one sender. The benefit of a Single Frequency Network is that one hasn’t to switch frequencies when one is moving from one such region to another.

Mode II and mode IV are intended for terrestrial local broadcasting in bands I-III, IV-V and L. They can also be used for satellite-terrestrial broadcasting in band L. These modes doesn’t need a long guard interval, but there carrier spacing is increased to reduce Doppler effects.

Mode III is intended for terrestrial, satellite and hybrid satellite-terrestrial broadcasting below 3000 MHz. It has a still larger carrier spacing, but it has a relatively larger frame length.

Field-tests have shown that for almost all purposes, modes I, II and IV have sufficient carrier spacing for proper operation.

Besides these modes, several extra services were envisaged to offer a surplus in service compared to current radio services like FM.

By its digital nature, its fairly easy to transmit other information than audio via Digital Audio Broadcasting. This has been exploited in several services and used to transmit all kind of extra information. First of all it’s possible to transfer simple binary data to the receiver which can be used for other equipment that is connected to the receiver. Then there’s all kind of data that can be displayed for instance at a display on the receiver. Data associated with audio programs like artist, title of the song, but also the song-text or intended audience. Or various data that are supplied irregularly like traffic information or paging information, also used in so called “buzzers”.

Other data that is supplied regularly includes local time and date, country identification, geographical location of the sender and so-called announcements, messages supplied for instance by commercial organisations. Another extra service is that of conditional access. This makes it possible that certain programmes can only be listened to if one has the appropriate key.
to decode certain scrambled data. This feature is intended to be used by organisations who want to be paid by whoever wants to listen to their programs.

To transmit more than one program in a Single Frequency Network, one can use the concept of multiplexing. With multiplexing a DAB frame can be divided into a maximum of 63 sub-channels. The extra services also need allocation in a DAB frame. This leads to the concept of the frame divided into a control section and a data section, the control section describing the format of a data section. In Digital Audio Broadcasting the control section together with other services that don’t need much bandwidth, are contained in a frame multiplexed sub-channel called the Fast Information Channel. The data section is contained in a frame multiplexed sub-channel called the Main Service Channel. The Fast Information Channel together with the Main Service Channel and the Synchronisation Channel can be seen in figure 2.8, in this figure \( f \) is a frame number.

![Figure 2.8: DAB frame multiplexed channels](image)

In table 2.2 an overview of often used symbols is presented and table 2.3 presents characteristic parameters of the DAB system.

**Table 2.2: Frequently used symbols**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T )</td>
<td>Elementary period</td>
</tr>
<tr>
<td>( L )</td>
<td>The number of symbols per transmission frame excluding the NULL symbol</td>
</tr>
<tr>
<td>( K )</td>
<td>The number of transmitted carriers</td>
</tr>
<tr>
<td>( f_s )</td>
<td>The spacing between adjacent carriers</td>
</tr>
<tr>
<td>( T_F )</td>
<td>Transmission frame duration</td>
</tr>
<tr>
<td>( T_{NULL} )</td>
<td>The NULL symbol duration</td>
</tr>
<tr>
<td>( T_s )</td>
<td>The duration of OFDM symbols other than the NULL symbol</td>
</tr>
<tr>
<td>( T_U )</td>
<td>The inverse of the carrier spacing</td>
</tr>
<tr>
<td>( \Delta )</td>
<td>The duration of the guard interval</td>
</tr>
</tbody>
</table>
Table 2.3: DAB mode parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Mode I</th>
<th>Mode IV</th>
<th>Mode II</th>
<th>Mode III</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>1/2048000 s</td>
<td>1/2048000 s</td>
<td>1/2048000 s</td>
<td>1/2048000 s</td>
</tr>
<tr>
<td>$L$</td>
<td>76</td>
<td>76</td>
<td>76</td>
<td>153</td>
</tr>
<tr>
<td>$K$</td>
<td>1536</td>
<td>768</td>
<td>384</td>
<td>192</td>
</tr>
<tr>
<td>$f_s$</td>
<td>1 kHz</td>
<td>2 kHz</td>
<td>4 kHz</td>
<td>8 kHz</td>
</tr>
<tr>
<td>$T_F$</td>
<td>196608T (96 ms)</td>
<td>98304T (48 ms)</td>
<td>49152T (24 ms)</td>
<td>49152T (24 ms)</td>
</tr>
<tr>
<td>$T_{NULL}$</td>
<td>2656T (1.3 ms)</td>
<td>1328T (648 µs)</td>
<td>664T (324 µs)</td>
<td>345T (168 µs)</td>
</tr>
<tr>
<td>$T_S$</td>
<td>2552T (1.2 ms)</td>
<td>1276T (623 µs)</td>
<td>638T (312 µs)</td>
<td>319T (156 µs)</td>
</tr>
<tr>
<td>$T_U$</td>
<td>2048T (1 ms)</td>
<td>1024T (500 µs)</td>
<td>512T (250 µs)</td>
<td>256T (125 µs)</td>
</tr>
<tr>
<td>$\Delta$</td>
<td>504T (246 µs)</td>
<td>252T (123 µs)</td>
<td>126T (62 µs)</td>
<td>63T (31 µs)</td>
</tr>
</tbody>
</table>
In this chapter we will have a detailed look at the complete Digital Audio Broadcasting system. We will mainly discuss it from a mathematical point of view, without looking much at implementation details. We will first briefly discuss some principles, followed by a short overview of the system. Then we will travel through the Digital Audio Broadcasting system from sender through channel to receiver, going from high-level to low-level and back again.

3.1 System View

We will consider a configuration of one sender and one receiver. We can make a layered model of sender, channel and receiver that expresses the symmetry between concepts in both sender and receiver and that also shows the levels of functionality. This model is in figure 3.1.

At the top of the sender several services are presented, this is the level of highest functionality from the view of a user. They are presented to a layer labelled "services". This layer is
responsible for processing and assembling services into a suitable internal format. The next layer is labelled “channels”. It stands for the processing of data-streams, called sub-channels, from the service layer so that it can be sent along a channel. The layer labelled “frames” assembles slotted data from the channels layer into a frame; a unit of transmission adapted to processing and synchronisation in the receiver. One layer lower, we find the “symbols” layer. This layer divides frames into symbols and adds synchronisation information. A symbol is a unit of transmission adapted to the behaviour of the channel. At the lowest level we find the layer “signals”. It converts symbols to electrical signals suitable for transmission by electro-magnetic waves.

The channel transports the electro-magnetic waves to the receiver. During this transport information is corrupted.

The receiver is presented a signal that carries several kinds of information. Its task is to reconstruct services. The user can select certain services to be offered, others being discarded.

The sender has added information at all layers, with the purpose to be able to transport them by the channel and to reconstruct the original services by the receiver.

At the “signals” layer of the receiver the extra information of the guard interval is used to optimally guess the information presented to the “signals” layer in the sender. The information in the NULL-signal is used to coarsely synchronise to a new frame. At the “symbols” layer the TFPR symbol is used as a phase reference for the other symbols in a frame. The TFPR symbol is also used to finely synchronise symbols. At the “frame” layer information from the Fast Information Channel is used to recreate channels. One level higher at the “channels” layer, information added at the sender is used to nearly perfectly reconstruct the information presented to the “channels” layer at the sender. At the highest level, the “services” layer, services are reassembled into their original format and according to the selections made by the user, presented to him.

The levels above the “services” layer are not represented in this model and are of little interest for understanding DAB. At the sender side these levels can contain A\D conversion of audio, composing of service information and switching from one service to the other. At the receiver side D\A conversion can exist, or a user interface that displays information or that selects services according to the user’s choices.

We will now start our journey at the sender.

3.2 Sender

3.2.1 Services Layer

At the input of this layer both stream oriented and packet oriented data is presented. In DAB terms these are called service components. Sampled audio, program associated data, service information and general data are such service components.

The control inputs select which service components are assembled into services and select which services are sent to the receiver. It also selects various characteristics for service components like audio compression rate or protection level.
At this level MPEG layer II audio compression is performed. Data can be encrypted and given conditional access properties. Service information is assembled with multiplex and fast-information-channel data into the Fast Information Channel. Other service information and audio and data streams form a maximum of 64 sub-channels to be contained in the Main Service Channel.

### 3.2.2 Channels Layer

At the inputs of the “channels” layer several sub-channels are presented. Sub-channels are bit-streams with an associated bit-rate. Control information is provided that selects channel properties of these sub-channels.

The main task of this level is to encode the sub-channels to anticipate on errors that occur in the real transmission channel. This is done by both energy-dispersal, convolutional coding and time-interleaving. Energy dispersal pseudo-randomly changes a signal in such a way that its energy spectrum is more uniform across its entire frequency range. Convolutional encoding adds redundant bits to a bit stream and makes it possible that the receiver can reconstruct the original bit stream if its corruption is below a certain bit error rate. Time-interleaving delays certain bits from a channel over a maximum of 384 ms in order to get independence in the time domain. In that way the effect of short burst-like errors are spread in time when a sub-channel is de-interleaved at the receiver again. Time-interleaving is used only for channels in the Main Service Channel.

Both the energy disperser and the convolutional encoder need to be initialised for each sub-channels, to obtain independence between the sub-channels. To be able to offer other services while broadcasting, it must be possible to change sub-channels regularly, and therefore to initialise the energy disperser and the convolutional encoder regularly.

This is obtained by dividing the bit-streams into slots of equal duration. The duration of a such a slot is 24 ms. The collection of slots of all sub-channels is called a logical frame. Because all sub-channels can have different bit-rates, these slots can consist of a different number of bits. It follows from this that the bit-rates of the sub-channels must be chosen in such a way that a slot contains a whole number of bits.

In DAB energy dispersal is done by the unit pictured below.

![Energy dispersal unit](image)

**Figure 3.2: Energy dispersal unit**

In this picture the inputs are bits, the blocks labelled 'D' are registers and the encircled pluses denote exclusive or gates. This unit generates a pseudo-random bit sequence and scrambles
the input line labelled 's' to generate a sequence labelled 'c'. In $\mathbb{Z}_2$ this can be mathematically expressed as:

$$x = D^9 x \oplus D^5 x$$

$$c = x \oplus s$$

Where $c, s, x \in \mathbb{Z}_2$ and the unary operator $D^n$ delays a signal $n$ time steps. The energy dispersal unit is initialised by feeding one bits to all registers. This initialisation is done at the beginning of each logical frame.

The convolutional encoder used in DAB is shown in the picture below. It implements a convolutional code with constraint length seven. The registers are initialised with zero bits for each sub-channel at the start of each logical frame.

It generates four polynomials in $x$ that can be expressed in $\mathbb{Z}_2$ as

$$P_1(x) = D^6 x \oplus D^5 x \oplus D^3 x \oplus D^2 x \oplus x$$

$$P_2(x) = D^6 x \oplus D^3 x \oplus D^2 x \oplus D^1 x \oplus x$$

$$P_3(x) = D^6 x \oplus D^4 x \oplus Dx \oplus x$$

$$P_4(x) = D^5 x \oplus D^5 x \oplus D^3 x \oplus D^2 x \oplus x$$

Thus for every source bit, four coded bits are generated, giving a code rate of $R = \frac{1}{4}$.

To get other code rates puncturing is used. This procedure groups 32 source bits into a slot, and discards bits of the 128 bits at the output of the convolutional encoder according to several puncturing schemes. The 24 defined puncturing schemes give code rates of $R = \frac{8}{8+i}$ for $i \in \{1, \ldots, 24\}$. For the bit-rates of the sub-channels it means that they have to lead to a multiple of 32 bits in a slot.

At the end of a slot six zero bits are appended, so that the convolutional encoder is initialised for the next subchannel. These six bits give 24 bits at the output of the encoder. These are coded with puncturing index 8, giving 12 bits at the output.
After a full slot is convolutional encoded and punctured, 0 to 63 zero bits are appended to the result in order to get a multiple of 64 bits per sub-channel.

The Fast Information Channel is convolutional encoded with a fixed rate of $\frac{1}{3}$. The other channels can be encoded with several so-called protection schemes. With equal error protecting, a fixed code rate is used for a channel. Unequal error protection allows a flexible code rate within a channel. For 64 combinations of channel data rate and a so-called protection level, this unequal error protection is defined.

After energy dispersal, convolutional encoding, puncturing and zero-padding, the resulting slots are time-interleaved for all channels in the Main Service Channel.

Time-interleaving takes 16 bits at a time and delays each bit a number of logical frames according to its position. If $i$ is the position of a bit in a row of 16 bits, with $0 \leq i < 16$ and $i = 0$ being the first bit, then the number of logical frames $d$ that bit is delayed before it is passed to the output of the time-interleaver, is given by

$$d = 2^3 \cdot i_3 + 2^2 \cdot i_2 + 2^1 \cdot i_1 + 2^0 \cdot i_0$$

or otherwise said, $d$ is the bit-reversal of $i$. This is done for all groups of 16 bits. Time-interleaving preserves the order of bits in a slot. Initially, when at some bit-positions no valid bits are yet available, the time-interleaver yield zero bits. At the output of the time-interleaver we have again multiples of 64 bits, called capacity units (CU's), thus one CU stands for 64 bits.

At the output of the “channels” layer we now have coded channels; the Fast Information Channel, and the Main Service Channel with a maximum of 64 sub-channels. These coded channels consist of a whole number of CU's.

### 3.2.3 Frames Layer

At the input of the frames layer, a maximum of 64 sub-channels in the Main Service Channel and sub-channels in the Fast Information Channel present their CU’s. The frames layer gathers these CU’s and assembles them in a so-called Common Interleaved Frame or CIF. A total of 864 CU’s are gathered in one CIF, excluding the data in the Fast Information Channel. Per Common Interleaved Frame a number of Fast Information Blocks or FIB’s, that are part of the Fast Information Channel, are assigned, that encode the position of a sub-channel in the Common Interleaved Frame.

Another important task at the frames layer is to introduce the concept of a mode. A mode is designed to anticipate upon certain transmission characteristics. Four modes are defined in Digital Audio Broadcasting; labelled I, II, III and IV.

At the frames layer a mode defines how many Common Interleaved Frames are assembled into a so-called Transmission Frame. A Transmission Frame is a frame suitable for transmission, it consists of the Synchronisation Channel, the Fast Information Channel and the Main Service Channel. The Synchronisation Channel is used at the receiver to synchronise to a Transmission Frame.
Modes I, IV and II are modes that can be described very regularly; mode III takes in that sense a different position.

In mode I four Common Interleaved Frames are assembled in one Transmission Frame, in mode IV two Common Interleaved Frames, in mode II one Common Interleaved Frame and in mode III also one Common Interleaved Frame. Per CIF, three FIB's are used to describe the position of a sub-channel in the CIF. All CU's of a sub-channel are grouped together and inserted at one of the 864 starting positions in the CIF. This is called the start address of the sub-channel, and it's encoded in a FIB related to that CIF. Unused CU's are grouped together and are filled with the output of the energy disperser, fed with zero bits.

This description holds for modes I, IV and II, mode III however has a peculiarity. All modes are designed to carry the same amount of information per second, regardless of the mode chosen. This must also be true for the Synchronisation Channel. That dictates the number of Transmission Frames that are transmitted per second, for per mode the number of information carried in the Synchronisation Channel per Transmission Frame is different. For mode III however, the standard defined the same number of Transmission Frames as for mode II. The reason for this is to get a minimum frame duration of 24 ms. That means that per second less synchronisation information is transmitted in mode III. Per mode III Transmission Frame this capacity is used to carry extra information in the Fast Information Channel; one extra FIB is transmitted per Transmission Frame.

At the output of the frames layer, one Transmission Frame is presented to the symbols layer, it is one long bit-string that contains all FIB's grouped together, appended with all CIF's grouped together, and place holders for the Synchronisation Channel at the beginning of the frame.

3.2.4 Symbols Layer

At the input of the "symbols" layer frames are presented.

The task of the "symbols" layer is to divide a frame into slots of equal width, to map these slots onto frequencies in the OFDM spectrum and to perform frequency interleaving. Another task is the generation of the TFPR symbol.

The frequency slots have a different width in each DAB mode. Let \( K \) be the maximum number of carriers in a OFDM spectrum and \( A \) the number of used carriers. For all modes these numbers are listed in table 3.2.4, the ratio \( \frac{A}{K} = \frac{3}{4} \).

<table>
<thead>
<tr>
<th>Mode</th>
<th>I</th>
<th>IV</th>
<th>II</th>
<th>III</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>2048</td>
<td>1024</td>
<td>512</td>
<td>256</td>
</tr>
<tr>
<td>A</td>
<td>1536</td>
<td>768</td>
<td>384</td>
<td>192</td>
</tr>
</tbody>
</table>

Now let \( S_K = \{0, \ldots, K - 1\} \), \( S_A = \{0, \ldots, A - 1\} \), \( S_C = \{\frac{K}{8}, \ldots, \frac{K}{2} - 1, \frac{K}{2} + 1, \ldots, \frac{7K}{8}\} \) and \( S_F = \{-\frac{A}{2}, \frac{A}{2}\} \setminus \{0\} \). Note that \( S_C \subset S_K \) and \( |S_A| = |S_C| = |S_F| \). \( S_K \) is the set of indices of the carriers in the OFDM spectrum and \( S_C \) is the set of indices of the active carriers in that
spectrum. The sets $S_A$ and $S_F$ are used to number the active carriers independent of the the OFDM spectrum.

A frame is divided into slots of $2A$ bits each. Such a slot is called a symbol. Mapping to frequency slots is done by assigning the first $A$ bits to the most significant bit and the next $A$ bits to the least significant bit of a number associated with a carrier $i$ of the active carriers. Call these bits $s_i$ and $t_i$ respectively and call the number $c_i$ with $s_i, t_i \in \mathbb{Z}_2, c_i \in \mathbb{Z}^2_2, i \in S_A$, then

$$c_i = (s_i, t_i)$$

Frequency interleaving is done by re-mapping $c_i$ to a number $d_k \in \mathbb{Z}^2_2, k \in S_C$ in the full OFDM spectrum. For this frequency interleaving a permutation is used on the active carriers. This isn’t done directly with a permutation on $S_A$, but a permutation $P$ on $S_K$ is used to construct a mapping $R$ from $S_A$ to $S_C$ that is then used to map the active carriers in the OFDM spectrum. Let

$$P = \{(m, n) \in S_K \times S_K \mid n = \Pi(m)\}$$

with $\Pi : \mathbb{Z}_K \rightarrow \mathbb{Z}_K$ a permutation defined by

$$\Pi(m \oplus 1) = 13 \oplus \Pi(m) \oplus C \land \Pi(0) = 0$$

with $C = \frac{K}{4} - 1$.

We now construct a function $R : S_A \rightarrow S_C$ from $P$ with a reduced range and another domain but with the same ordering (expressed by $f$) on its domain.

$$R = \{(p, q) \in S_A \times S_C \mid \exists (a, b) \in P : f(p) = a \land q = b\}$$

$$f : S_A \rightarrow S_K \land \forall a_1, a_2 \in S_A : f(a_1) < f(a_2) \iff a_1 < a_2$$

Now $d_k$ is defined by

$$d_{R(i)} = c_i$$

Frequency interleaving can be informally described as follows. Generate a row of numbers $\Pi(0), \Pi(1), \ldots, \Pi(K-1)$, delete every number not in $S_C$. This results in a row of $A$ numbers that give the resulting index in the OFDM spectrum.

At the output of the symbols layer for all $L$ symbols a set of $A$ numbers per symbol is generated that are mapped onto the frequency spectrum of the OFDM carrier spectrum.

### 3.2.5 Signals Layer

At the input of the "signals" layer frequency interleaved symbols are presented. This layer processes these spectra to generate a signal suitable for transmission.
First the elements of a symbol are differential modulated using QPSK. This is done by generating a complex phase \( p_k \in \mathbb{C} \) for each carrier \( k \in S_C \) in the OFDM spectrum from the set of numbers \( \{ d_k \in \mathbb{Z}_2 \} \) associated with each symbol. Since we’re using QPSK, we’re mapping \( d_k \) to one of four phases according to the following rule. Let \( d_k = (s_k, t_k) \) then

\[
\sqrt{2} \cdot p_k = (1 - 2s_k) + j \cdot (1 - 2t_k)
\]

Note that \( |p_k| = 1 \) and that \( \arg p_k = n \pi \frac{\pi}{4} \). This complex phase is then added to the phase of a carrier. Call a carrier with index \( k \) in symbol \( s \) with \( s \in \{1, \ldots, L\} \), \( x_{k,s} \) then for \( s > 1 \)

\[
x_{k,s} = x_{k,s-1} \cdot p_k \Rightarrow \arg x_{k,s} = \arg x_{k,s-1} + \arg p_k \pmod{2\pi}
\]

and as we look at the phase relationship we see the differential modulation reflected in

\[
\arg x_{k,s} - \arg x_{k,s-1} = \arg p_k \pmod{2\pi}
\]

For \( s = 1 \), \( x_{k,1} \) is specified only by the TFPR symbol, and has a different phase than the carriers of the other symbols.

\[
x_{k,1} = t(k) \land t : S_C \rightarrow \{1, j, -1, -j\}
\]

Thus \( |x_{k,1}| = 1 \) and \( \arg x_{k,1} = n \pi \frac{\pi}{2} \) and therefor also \( |x_{k,s}| = 1 \).

Now we map can generate a time signal from the OFDM spectrum. The active carriers are the ones defined by \( x_{k,s} \), that is with their indices \( k \in S_C \). We will consider one symbol only, therefor we’re at this moment not interested in the index \( s \) and will write \( x_k \) instead of \( x_{k,s} \). Now define a value \( Y_k \) for all carriers \( k \) in the OFDM spectrum by setting the non-active carriers to zero,

\[
Y_k = \begin{cases} 
  x_k & \text{for } k \in S_C \\
  0 & \text{for } k \notin S_C 
\end{cases}
\]

The resulting spectrum is processed by performing a inverse Fourier transform on it, giving a digital cyclic signal \( \tilde{y}[n] \) of period \( K \) in the time domain.

\[
\tilde{y}[n] = \frac{1}{K} \sum_{k=0}^{K-1} Y_k W_K^{kn}
\]

With \( W_K = e^{j\frac{2\pi}{K}} \).

To extend this signal with a guard interval is quite straightforward. The length of the guard interval \( D \) is defined as \( D = \frac{K}{4} - \frac{K}{56} \). Let \( y[k] \) be the resulting signal of the signal \( \tilde{y}[n] \) extended with the guard interval, then the following simple relation defines \( y[k] \)

\[
y[k] = \tilde{y}[k - D] \text{ for } 0 \leq k < K + D
\]
As a result of limiting $\tilde{y}[k]$ to $y[k]$, the spectrum of $y[k]$ is distorted. To see this, write

$$y_w[k] = \tilde{y}[k - D] \cdot w[k]$$

with $w[k]$ a so-called rectangular window function,

$$w[k] = \begin{cases} 1 & \text{for } 0 \leq k < K + D \\ 0 & \text{otherwise} \end{cases}$$

Now for the spectrum $Y_w(e^{j\theta})$ of $y_w[k]$ that means – if $Y(e^{j\theta})$ is the spectrum of the signal $\tilde{y}[k]$ – that

$$Y_w(e^{j\theta}) = W(e^{j\theta}) \ast (Y(e^{j\theta}) \cdot e^{-j\theta D})$$

with

$$W(e^{j\theta}) = \frac{\sin \left( \frac{1}{2} (K + D) \theta \right)}{\sin \left( \frac{1}{2} \theta \right)} e^{-j \frac{1}{2} \theta (K + D - 1)}$$

Look at figure 3.4 to see a plot of the function $\frac{\sin \left( \frac{1}{2} N \theta \right)}{\sin \left( \frac{1}{2} \theta \right)}$. This function has periodicity $4\pi$.

![Figure 3.4: Plot of $\frac{\sin \left( \frac{1}{2} N \theta \right)}{\sin \left( \frac{1}{2} \theta \right)}$ for $N = 20$](image)

Now the complex discrete signal $y[k]$ is converted to an analog signal and modulated upon a transmission carrier as follows.

Call the real and imaginary parts of $y[k]$, $r[k]$ and $i[k]$ respectively. We shall assume that D/A conversion is ideal, we can then describe D/A conversion as presenting a series of Dirac pulses to an ideal low-pass filter. Represent both $r[k]$ and $i[k]$ as a series of Dirac pulses.
Analysis

\[ r_p(t) = \sum_{n=0}^{K+D} r[n] \delta(t - nT) \]
\[ i_p(t) = \sum_{n=0}^{K+D} i[n] \delta(t - nT) \]

With \( T \) the sample duration.

Now these signals are presented to a low-pass filter, giving resulting signals

\[ r_c(t) = \sum_{n=0}^{K+D} r[n] \text{sinc}(\frac{t}{T} - n) \]
\[ i_c(t) = \sum_{n=0}^{K+D} i[n] \text{sinc}(\frac{t}{T} - n) \]

For the impulse response \( h(t) \) of an ideal low-pass filter with frequency response

\[ H(j\omega) = \begin{cases} T & \text{for } |\omega| < \frac{1}{2T} \\ 0 & \text{otherwise} \end{cases} \]

equals \( h(t) = \text{sinc}(\frac{t}{T}) \).

Now the transmitted signal \( y(t) \) is defined to be

\[ y(t) = \text{Re}\{y_c(t) \cdot e^{j\omega_0 t}\} \]

With \( y_c(t) = r_c(t) + j \cdot i_c(t) \) and \( \omega_0 \) the angular transmission frequency. If we rewrite \( y(t) \) we get

\[ y(t) = \text{Re}\{[r_c(t) + j \cdot i_c(t)] \cdot e^{j\omega_0 t}\} \]

or

\[ y(t) = r_c(t) \cdot \cos(\omega_0 t) - i_c(t) \cdot \sin(\omega_0 t) \]

This last operation is known as I/Q modulation. Finally \( y(t) \) is amplified and transmitted through the æther.

3.3 Channel

The electro-magnetic waves, transmitted by the sender, travel in all directions and can be reflected or diffracted by the terrain or can be distorted by other electro-magnetic waves.
The behaviour of these electro-magnetic waves can be modelled by a channel model. This is a simplification of the real-world situation, but gives us the opportunity to understand and predict several phenomena that occur when these waves are transmitted.

What we’ll include in our channel model are multi-path effects, Doppler frequency shifts, local scattering, signal attenuation and noise. What we don’t model is for example dispersion caused by a medium and burst-like interference.

The channel model is split into two parts; one part models several signal distortions that can be described by additive white Gaussian noise, the other part describes the ‘impulse response’ of the channel, and includes the other effects.

We then have a model of sender-channel-receiver as can be found in figure 3.5.

\[
\begin{align*}
\text{Sender} & \quad \text{h(t)} \quad \text{Receiv}\ldots \n(t) \\
\end{align*}
\]

Figure 3.5: Channel model

In this figure \( h(t) \) is the channel impulse response and \( n(t) \) is the additive white Gaussian noise.

We will now determine the channel impulse response.

### 3.3.1 Determining the Channel Impulse Response

In a model of sender, channel and receiver, we can characterise the channel by its impulse response. Such channel modelling has been done by several authors. We will here present the results of that work, and a slightly different description of the channel that will fit our needs better.

Let the physical signal at the sender be expressed by

\[
s_e(t) = r_c(t) \cos(\omega_0 t) - i_c(t) \sin(\omega_0 t)
\]

Now write \( y_e(t) = r_c(t) + j \cdot i_c(t) \) and let \( s(t) = |y_e(t)| \) and \( \phi(t) = \arg\{y_e(t)\} \), then we can express \( s_e(t) \) also as

\[
s_e(t) = s(t) \cos(\omega_0 t + \phi(t))
\]

with \( s(t) \) and \( \phi(t) \) real signals, being the amplitude and phase modulators of the signal modulated at angular frequency \( \omega_0 \).

We can model this signal by a complex signal \( z_e(t) \), with \( s_e(t) = \text{Re}\{z_e(t)\} \) and
\[ z_e(t) = s(t)e^{j(\omega_0 t + \phi(t))} \]

The mobile channel can be characterised by several aspects. Due to reflections, several paths exist at which a transmitted signal can be picked up by the receiver antenna. Number these paths \( 1 \ldots M \). Due to the different lengths of these paths, the transmitted signal is delayed with a different delay for each path, call these delays \( T_m \) with \( m \in \{ 1 \ldots M \} \). Locally around a receiver a signal from a path can be scattered, introducing sub-paths with delays grouped around the delay of the originating path. Number these sub-paths \( 1 \ldots N \) (\( N \) being the greatest number of sub-paths for all paths), and call the associated delays \( T_{mn} \) with \( n \in \{ 1 \ldots N \} \). All these paths can attenuate the signal with a different factor, call that \( C_{mn} \).

Another effect is that of Doppler shift. Due to the motion of the receiver's antenna relative to the signal paths, a frequency shift is introduced. Call that frequency shift \( \omega_{mn} \) for all paths. That frequency shift can be expressed in terms of the velocity of the vehicle (and thus the antenna), \( v \), the angle of incidence of a wave to the direction the vehicle is moving to, \( \alpha_{mn} \), and, finally, to the transmission frequency of the sender, \( \omega_0 \). This is expressed by

\[ \omega_{mn} = 2\pi \frac{v}{c} \omega_0 \cos(\alpha_{mn}) \]

With \( c \) being the speed of light in the medium the antenna is moving in. We can model each received path by a complex signal \( z_{mn}(t) \) as follows

\[ z_{mn}(t) = C_{mn} z_e(t - T_{mn})e^{j\omega_{mn}t} \]

Where we have included respectively the attenuation, the delay and the Doppler shift. The total received signal can thus be modelled by \( z_r(t) \) and

\[ z_r(t) = \sum_{m=1}^{M} \sum_{n=1}^{N} z_{mn}(t) = \sum_{m=1}^{M} \sum_{n=1}^{N} C_{mn} z_e(t - T_{mn})e^{j\omega_{mn}t} \]

Of course the received physical signal, \( s_r(t) \), is related to that complex signal by \( s_r(t) = \text{Re}\{z_r(t)\} \).

Now, we want to determine the impulse response, \( h(t) \) of the channel. We therefore look at the response at the receiver to a delta-impulse at the sender. We thus have

\[ s_e(t) = \delta(t) \]

and

\[ h(t) = s_r(t) \]

If we substitute this in the formulas we’ve earlier formulated, we find

\[ h(t) = \text{Re}\left\{ \sum_{m=1}^{M} \sum_{n=1}^{N} C_{mn} \delta(t - T_{mn})e^{j\omega_{mn}t} \right\} \]
And that equals, because of the independence of the delta pulses, to

\[ h(t) = \sum_{m=1}^{M} \sum_{n=1}^{N} C_{mn} \delta(t - T_{mn}) \cos(\omega_{mn} t) \]

And that is the channel impulse response we were looking for.

### 3.3.2 Probability Distributions for Channel Parameters

Measurements in real-life mobile channels have been the ground for specifying probability distribution functions of several channel parameters. These models can help us to get a little more insight in the behaviour of such a channel under different circumstances.

The delays \( T_{mn} \) can be thought to be grouped around a delay \( T_m \), for the delays introduced by scattering are very small compared to the delays caused by reflections. Thus \( T_m \) is defined as

\[ T_m = \frac{1}{N} \sum_{n=1}^{N} T_{mn} \]

The delays \( T_m \) can be modelled by an exponential distribution,

\[ p_{T_m}(x) = \begin{cases} \frac{1}{T_0} e^{-\frac{x}{T_0}} & \text{for } x \geq 0 \\ 0 & \text{otherwise} \end{cases} \]

With \( T_0 \) representing both the mean and standard deviation.

Using \( T_m \) instead of \( T_{mn} \) the channel impulse response becomes

\[ h(t) = \sum_{m=1}^{M} \delta(t - T_m) \sum_{n=1}^{N} C_{mn} \cos(\omega_{mn} t) \]

If we look at the earlier defined angle of incidence \( \alpha_{mn} \), it is a good approximation to consider that value to be uniformly distributed on \([0, 2\pi)\), notated by a probability density function

\[ p_{\alpha_{mn}}(x) = \begin{cases} \frac{1}{2\pi} & \text{for } x \in [0, 2\pi) \\ 0 & \text{otherwise} \end{cases} \]

Now, if we introduce a new random variable, \( \beta_{mn} \), with

\[ \beta_{mn} = \cos(\alpha_{mn}) \]

Then \( \beta_{mn} \) is distributed on \((-1, 1)\) with a probability density function of
And a probability distribution function of

$$p_{\beta_{mn}}(x) = \begin{cases} \frac{1}{\pi} \frac{1}{\sqrt{1-x^2}} & \text{for } x \in (-1, 1) \\ 0 & \text{otherwise} \end{cases}$$

Both the probability density function (pdf) and the probability distribution function (PDF) can be seen in figure 3.6

Now for the pdf and the PDF of $\omega_{mn}$ we find the following expressions

$$p_{\omega_{mn}}(x) = \begin{cases} \frac{1}{\pi} \frac{1}{\sqrt{\omega_D^2 - x^2}} & \text{for } x \in (-\omega_D, \omega_D) \\ 0 & \text{otherwise} \end{cases}$$

With $\omega_D = 2\pi \omega_0$ and $\omega_{mn} = \omega_D \cdot \beta_{mn}$ and
\[ P_{\omega_{mn}}(x) = \begin{cases} 
0 & \text{for } x \leq -\omega_D \\
\frac{1}{2} + \frac{1}{\pi} \arcsin \left( \frac{x}{\omega_D} \right) & \text{for } x \in (-\omega_D, \omega_D) \\
1 & \text{for } x \geq \omega_D 
\end{cases} \]

The pdf and PDF of \( \omega_{mn} \) have the same shape as the pdf and PDF of \( \beta_{mn} \), except for a scaling factor in both domain and range.

### 3.4 Receiver

Now that we've looked at the transmission channel, we'll enter the receiver.

#### 3.4.1 Signals Layer

The receiving antenna can pick up the transmitted waves if it is close enough to the sender. Not only the transmitted waves are picked up by the receiver, but also all kinds of other electro-magnetic waves. Therefore an analog tuner is used that locks to a specific Digital Audio Broadcasting transmission.

In this tuner the received signal is converted back to an intermediate frequency (IF). A mechanism exists that controls the signal amplification; the Automatic Gain Control (AGC) tries to keep the received signal at a constant amplitude. Another mechanism assures that the tuner keep locked to the signal; the Automatic Frequency Control (AFC).

After the tuner the signal is sampled and converted to a digital signal. This digital signal must be assembled again into a symbol.

The recovering of the real and imaginary parts that were associated with the transmitted signal, is done with I/Q demodulation.

Let \( s_r(t) \) be the received signal and \( s_r(t) = r_r(t) \cos(\omega_0 t) - i_r(t) \sin(\omega_0 t) \) then \( r_r(t) \) and \( i_r(t) \) can be reconstructed as follows

\[
s_r(t) \cdot \cos(\omega_0 t) = r_r(t) \cos^2(\omega_0 t) - i_r(t) \sin(\omega_0 t) \cos(\omega_0 t) \]
\[
= r_r(t) \frac{1}{2} + \left[ r_r(t) \frac{1}{2} \cos(2\omega_0 t) - i_r(t) \sin(2\omega_0 t) \right] \]

\[
s_r(t) \cdot \sin(\omega_0 t) = r_r(t) \cos(\omega_0 t) \sin(\omega_0 t) - i_r(t) \sin^2(\omega_0 t) \]
\[
= i_r(t) \frac{1}{2} + \left[ r_r(t) \frac{1}{2} \sin(2\omega_0 t) - i_r(t) \cos(2\omega_0 t) \right] \]

Now if both these signals are passed through a low-pass filter, the resulting signals will be \( r_r(t) \frac{1}{2} \) and \( i_r(t) \frac{1}{2} \). Note that for this demodulation to work correctly, the signal must be multiplied with exactly the same frequency \( \omega_0 \) as used in the transmitter.

After demodulation these signals are sampled, giving two signals \( r[n] \) and \( i[n] \) with

\[
r[n] = r_r(nT) \quad i[n] = i_r(nT)\]
With $T$ the sample duration. In the new Digital Audio Broadcasting chipset, the I/Q demodulation is done in the digital domain, from a sampled signal $s_r(nT)$.

To be able to reconstruct a symbol without errors the duration of $T$ must be the same as in the transmitter. What we don’t consider is the effect of the exact sampling moment, for the right moment is hard to define in a surroundings where multi-path effects are present. In the Digital Audio Broadcasting receiver these signals are also quantised, but we won’t look at that effect and consider the sampled signals to be real numbers. If word-lengths are chosen right this isn’t a bad approximation.

Define the complex signal $y[n]$ to be

$$y[n] = r[n] + j \cdot i[n]$$

Note that $y[n]$ is a stream of data, that is, it’s defined for $-\infty < n < \infty$. Of course this is again an approximation to the real-world situation where at a certain moment that signal is present and at a certain moment it disappears.

Suppose that we have an ideal situation where the channel transports the transmitted wave without modification. In that case, at the receiver the same analog signal is present as at the sender. Suppose further that the receiver is synchronised perfectly to the signal, both in time and in frequency. In that case the $y[n]$ we have just defined is the same as the concatenation of the signals $y[k]$ as defined in section 3.2.5.

Of course this is an ideal case, and we will in chapter 9 look at synchronisation and at the effects of the channel on the received signal, but for now we will use these assumptions.

If we extract at exactly the right moment a sequence of $K$ samples from $y[n]$, and then skip the next $D$ samples and then repeat this process, then we have regained one period $y_0[n]$ of the cyclic signal $\tilde{y}[n]$. This holds when we don’t take in account the effect of the window function on the signal. This assumption is valid if the signal strength is large enough.

By an FFT we can reconstruct $Y_k$ as follows

$$Y_k = \sum_{n=0}^{K-1} \tilde{y}_0[n] W_{K}^{-kn} \quad \text{for} \quad 0 \leq k < K$$

Now we have our original spectrum, we can use differential demodulation to reconstruct frequency interleaved symbols that will be presented to the symbols layer.

Therefor the phase information in the first modulated symbol in a received frame is used as a reference to all subsequent received symbols. The operation that is performed is

$$p_k = x_{k,s} \cdot x_{k,s-1}^* \quad \Rightarrow \quad \arg p_k = \arg x_{k,s} - \arg x_{k,s-1} \mod 2\pi$$

With $k \in S_C$ and thus we have reconstructed our transmitted complex phase.

Suppose however that an error is introduced in $p_k$, let

$$p'_k = p_k \cdot e_k$$
with $e_k$ the error factor. In the transmitter, as explained in section 3.2.5, two bits $s_k$ and $t_k$ were assembled into $p_k$ according to a QPSK scheme. Now the error $e_k$ changes the argument of $p_k$, so we can't tell with certainty what the original phase was. Now let's look at figure 3.7

![Figure 3.7: Regions used by QPSK](image)

The numbers between parenthesis indicate the $(s_k, t_k)$ pairs. The bold dots represent the original positions in the complex plane of the phases $p_k$. The best guess we can make for $s_k$ and $t_k$ is that the error $e_k$ hasn't moved $p_k$ out of a quadrant. We construct two numbers from $p_k'$, the real and imaginary parts

$$s'_k = \text{Re}\{p'_k\} \quad t'_k = \text{Im}\{p'_k\}$$

We call the pair $d'_k = (s'_k, t'_k)$ a measure. The purpose of this measure will be clear when it's explained in the "Channels Layer" section. From this measure a guess $\hat{d}_k = (\hat{s}_k, \hat{t}_k)$ can be made

$$\hat{s}_k = \begin{cases} 1 & \text{for } s'_k < 0 \\ 0 & \text{for } s'_k > 0 \end{cases} \quad \hat{t}_k = \begin{cases} 1 & \text{for } t'_k < 0 \\ 0 & \text{for } t'_k > 0 \end{cases}$$

note that for either $s'_k = 0$ or $t'_k = 0$ the pair $(\hat{s}_k, \hat{t}_k)$ isn't defined.

The measure $s'_k$ and $t'_k$ are quantised to 4 bits in the Digital Audio Broadcasting receiver. The guess $\hat{d}_k$ is not constructed in a Digital Audio Broadcasting receiver, it is presented here for didactic purposes.

We now have a complete set of $A$ numbers $d'_k$ per symbol that are presented to the next layer; the symbols layer.

### 3.4.2 Symbols Layer

The inputs $d'_k$ per symbol are first frequency de-interleaved such that
\[ c'_i = d'_R(i) \]

for \( i \in S_A \) and \( c_i \in \mathbb{Z}_2^2 \).

The numbers \( c'_i \) are gathered in a slot of \( 2A \) numbers, such that the first half of that slot contains the numbers \( s'_i \) and the second half of the slot contains the number \( t'_i \).

These slots are then assembled into frames and presented to the "Frames Layer".

### 3.4.3 Frames Layer

At the frames layer, a Transmission Frame is split into Common Interleaved Frames and accompanying FIB’s, according to mode information that is supplied by the user. The information from the FIB’s that encoded the sub-channels’ positions is used to reconstruct the encoded sub-channels from the CU’s in a Common Interleaved Frame. Because of time-interleaving at the sender, a delay of 16 Common Interleaved Frames is introduced at the receiver, in order to avoid acausalities. The channels layer supplies the frames layer with information from the FIB’s, so that the frames layer can properly reconstruct the sub-channels.

To the next layer, the channels layer, sub-channel slots are presented as an integral number of CU’s.

### 3.4.4 Channels Layer

The sub-channels presented to the "Channels Layer" consists of streams of frequency- and time-de-interleaved measures.

At the transmitter, bits were convolutional encoded. The measures have a strong correlation to the output of the convolutional encoder.

These measures are used in decoding with the Viterbi algorithm, giving a very good approximation to the bit-stream presented at the input of the convolutional encoder in the transmitter.

The punctured output of the convolutional encoder in the transmitter can be considered as a convolutional code with a bigger code rate, but strictly speaking it isn’t a convolutional code itself. In the Viterbi decoding algorithm, the punctured bits are considered to be transmitted bits with value \( \frac{1}{2} \), so they carry no information and don’t influence the decoding process. The zero-padded bits at the end of a slot of a sub-channel are ignored. As a result the combination of convolutional encoding, puncturing and modified Viterbi decoding, gives a slightly degradation in performance compared to a 'pure' convolutional code with 'real' Viterbi decoding. In practice this degradation is so little, it isn’t important.

Note that the code rate for a certain sub-channel is given by information in the Fast Information Channel. The Fast Information Channel itself is decoded with an at the receiver known code rate of (approximately) \( \frac{1}{3} \).

The Viterbi decoder gives again a stream of bits per sub-channel. To get the bit-stream we
applied to the channels layer in the transmitter, we must reverse the energy dispersal that was used in the transmitter.

Because energy dispersal was a modulo-2 operation on the original bit-stream, it can be reversed by again feeding this bit-stream through an energy dispersal unit. If \( c' \) is the received sequence, \( x \) the same pseudo-random bit sequence as in the transmitter and \( s' \) the reconstructed bit-stream, then

\[
s' = c' \oplus x = (x \oplus s) \oplus x = (x \oplus x) \oplus s = 1 \oplus s = s
\]

so that it’s clear that the reconstructed bit-stream is indeed equal to the original bit-stream. This of-course under the assumption that \( c' \) is the same as \( c \), that is no errors exist in the bit-stream at the output of the Viterbi decoder.

The resulting bit-streams are presented to the services layer.

### 3.4.5 Services Layer

The bit-streams from the channels layer is processed according to the type of the bit-streams that is encoded in the Fast Information Channel. An MPEG layer II stream is audio-decompressed, conditional access properties are checked, data is decrypted and packet data is reassembled. The resulting data is presented to the user, according to some selection criterion, that the user has communicated to the receiver.

This concludes our travel through the entire DAB system. We now present some practical information.

### 3.5 The DAB System, an Overview

Now that we’ve looked at most aspects from the DAB system, we’ll here present an overview of the DAB system from a more practical and technical point of view.

A picture of a DAB transmitter can be seen in figure 3.8

In this picture we can see the ingoing signals to the left and the transmitted signal to the right.

A picture giving an example of the signals present at several layers in the transmitter is presented in figure 3.9.

Note that this picture presents the signals at the transmitter, but of course those signals are present in the receiver also and the receiver is responsible to try to reconstruct a selected (audio) program as good as possible, as explained earlier.
Figure 3.8: DAB System Overview
The DAB System, an Overview

Figure 3.9: DAB Signals Overview

- Sampled Audio
  - Audio Encoding
  - MPEG Layer II Compressed Audio
  - Convolutional Encoding
  - Error-protected Encoded Audio
  - Time Interleaving
  - Sub-Channels
    - Frame Multiplex
  - Frame Multiplexed Sub-Channels
  - Symbol Mapping
  - Symbols
    - Frequency Interleaving
  - Partial Filled Spectrum
    - Differential Modulation
  - Modulated OFDM Spectrum
    - Inverse Fourier Transform
  - Basic OFDM Signal
    - Extend with Guard Interval
  - OFDM Signal with Guard Interval
Part II

DSP Station High Level System Modelling and Simulation
4

Introduction

At the Philips Advanced Development Centre, specifications for a DAB receiver application specific chip set are developed. This is done with traditional digital design techniques and from experience gained with a DSP implementation of a DAB receiver. Another group at the Philips Research Laboratories uses these specifications to develop an actual implementation of the chip set. The specifications are in the form of PASCAL source code that is used to simulate the behaviour of the Digital Audio Broadcasting system, together with all kinds of documents that describe formally or informally parts of implementation details or various parameters.

At the Philips Research Laboratories this collection of material is used to construct a description in a hardware description language that models the desired chip set and is suitable for simulation and eventually (partial) synthesis.

This part of this report evaluates the use of a high level system specification tool for its fitness to be used at the Advanced Development Centre to model DAB now, and possibly other projects in the future.

The high level system tool is the same as used at the Philips Research Laboratories, the expectation is that a better coupling between developmental tools will result in a more efficient route from idea via specification to implementation.

The tool under consideration is called “DSP Station”. It is effectively an environment consisting of several tools each with its own application range in the design route. DSP Station runs under the UNIX operating system with the XWindows graphical user interface system.

In the following chapters we will discuss respectively DSP Station in general, the modelling of the DAB system in DSP Station and we conclude this part with an evaluation of DSP Station and its fitness for use at the Advanced Development Centre.
5

DSP Station

5.1 Design Route Covered by DSP Station

As earlier mentioned, DSP Station is in fact a set of cooperating tools each with a specific application area. The purpose of DSP Station is to facilitate devising and testing a DSP algorithm and to generate DSP code or data suitable for chip synthesis.

When using DSP Station, a design route typically for high level system modelling tools is traversed.

This route is depicted in figure 5.1 consist of the following phases; modelling, compilation, simulation, analysis and synthesis.

\[ \text{Modelling} \rightarrow \text{Compilation} \]

\[ \text{Simulation} \rightarrow \text{Synthesis} \]

\[ \text{Analysis} \]

\[ \text{Figure 5.1: Design Flow in DSP Station.} \]

In the modelling phase the designer can use several methods in DSP Station to express his design. One possibility is to use schematics to compose more and more complex hierarchical components built from standard library components. Another method is to use a hardware description language to describe the behaviour of a design. These methods can be mixed at will. The hardware description languages that can be used are DFL and a kind of C called
DSPC. DSP Station also offer tools that generate code in the hardware description language for several kinds of digital filters.

During the compilation phase DSP Station converts a hierarchical design composed of possibly a mixture of components and behavioural descriptions, into an internal format suitable for the next phase.

That phase is the simulation phase. During this phase DSP Station runs the design, consuming user-supplied stimuli and producing corresponding output. DSP Station can simulate in the time domain but also in the frequency domain.

In the analysis phase the designer can evaluate the outcome of the simulation and decide about the design if it has to be altered, upon which he enters the modelling phase again.

DSP Station offers analysis of several important characteristics of signal processing devices. It can give noise figures, an indication of the sensitivity of the algorithm to changes in parameters, an analysis of group delay, of amplitude slope and finite word length effects due to overflow and quantisation. It can calculate the minimum required word length and optimise constants of digital filters. Another important feature DSP Station offers is the extensive possibilities for post-processing and viewing of signals.

If the design has met the criteria of the designer, the synthesis phase is entered. The designer can select a specific digital signal processor for which DSP Station then generates code. He can also choose to use his algorithmic design as a basis for an ASIC implementation, in that case he can choose to let DSP Station generate VHDL code, C code or even to generate data that is suitable for implementing the algorithm in either a bit-serial or a bit-parallel architecture.

5.2 Description of Used Tools

We will discuss in this section the tools used when simulating and enhancing the DAB system in DSP Station. In figure 5.2 an overview of the DSP Tools for the different phases in the design route is given.

Beside these tools two other tools will be discussed. One tool is for viewing tutorials and manuals, the other one is a design management tool that is a uniform interface to all design data and all design tools.

5.2.1 Bold Browser

The Bold Browser is a tool that let the user view and print online documentation. Tutorials will be discussed separately from other manuals for they contain material that offer a short hands-on course that uses examples that can be accessed and altered in DSP Station.

In figure 5.3 a typical view of a Bold Browser session is given.
Tutorials

The Bold Browser contains for all major subject like DFL programming, working with Design Architect and so on, tutorials suitable for self-study that offer a short hands-on course in the subject. These so-called workbooks contain several lessons that introduce the reader to the subject and teaches him how to accomplish certain tasks using the concerned tool. For this purpose a simple introduction to the material is given together with some elementary examples. Each introduction is followed by some so-called lab-exercises, detailed step-by-step descriptions of how to perform a certain common task on the basis of an example. Of most examples a file is available, so the amount of work is most of the time restricted to pointing, clicking and a little typing. Each lesson is concluded with a small exercise to test the knowledge of the reader.

If the reader has finished the tutorial he is able to perform the most common tasks with a tool.

Manuals

The manuals available within the Bold Browser tool cover more subjects than the tools and concepts of DSP Station only. The quality of the manuals is good, the information is thorough
44   DSP Station

Figure 5.3: Typical Bold Browser Configuration

enough, and besides the table of contents also a list of figures and of tables, a glossary and an index is included. Figures are of good quality, but sometimes there could have been some more of them. The Bold Browser has mechanisms for cross-references and a search engine. For often read manuals it is more practical to print them than to read them on-line.

Although there's a search engine, it's sometimes hard to find detailed information on a subject. Some important topics - like properties - are not good enough documented.

5.2.2 Design Manager

The Design Manager offers a uniform interface to the tools of DSP Station and facilitates the managing of designs and transport of data between different tools.

In figure 5.4 a typical view of a Design Manager session is given.

Managing Designs

As can be seen from figure 5.4, one window in the Design Manager shows all available tools, the other window is used to navigate through the design hierarchy. From the tool window, tools can be started, the navigate window can be used to select designs and perform operations on
them. Designs can be copied, moved in the directory structure and deleted. Other operations include showing the hierarchical relations of a component and showing info on a component, like the pin list, the registered models or the body properties.

In practice the Design Manager is little used. All tools can be run separately from the Design Manager and simple functions on designs like copying, moving and deleting can be done inside most tools. One option that only the Design Manager offers and that was used when the DAB design was copied from the Philips Research Laboratories, was the Check References operation. This operation checks the design for symbolic references. Symbolic references are variables that point to directories. For the directory structure is different between machines, it is important not to include references to absolute directory names. Only a symbolic reference should be made for the top component in a design; all other references should be relative to that symbolic reference. The Check References operation can change invalid symbolic references only at the component level. If symbolic references are made inside DFL or DSPC code, these have to be changed by hand.

### 5.2.3 Design Architect

Design Architect allows textual entry of DFL and DSPC code and drawing of schematics by choosing symbols, placing them on a sheet and connecting them by wires. Besides that it allows all kinds of checking and reporting on the design entries.
In figure 5.5 a typical view of a Design Architect session is given.

We will here discuss schematic entry and entry of DFL and DSPC code, for that is what was used when extending the DAB system.

**Schematic Entry**

With schematic entry one can describe a design as a hierarchical collection of so-called sheets. Each sheet represents a component and consists of components at one hierarchical level deeper. These sub-components are represented by symbols; rectangles with line-segments representing in- and output ports, with text describing the name of the component, the names of the ports and possibly a description of the properties attached to this component. A picture of a generic symbol is in figure 5.6.

In this picture there's no difference between in- and output ports. That's because in a sheet only certain properties are visible. Default the property that defines a port to be an in- or output port is not visible. The visibility of properties can be modified for all properties. When editing a symbol, which is done inside the symbol editor of Design Architect, all properties are visible, and of each property the default visibility can be defined.

A sheet is thus composed of components that are represented by symbols and these symbols can be chosen from a standard library or can be drawn by the user. When a symbol is placed
on a sheet, it must be connected to other components or to an in- or output port symbol that represents the in- and output ports of the component associated with the sheet.

This connecting is done by drawing lines – representing single wires or busses – between ports of components. When a sheet is drawn it must be checked for unconnected wires, connections that connect ports of different types, etcetera. This is done automatically by Design Architect, that then generates a report of found errors. After a sheet is free of errors it can be saved.

When one wants to use the component as a sub-component in another sheet, one has to have a symbol for that component. There are several ways to do this in Design Architect, but there’s only one way that will finally work! First use DSP Update Viewpoint with the “Copy Extracted DFL into DFL model” option enabled but with the “Generate DSP Symbol from extracted DFL” option disabled! When this is done, use a “Open DFL” to view to generated DFL code. Then select from the menu bar under the “Compile” option the “Create DSP Symbol” option. The user still has to add the signal type at each pin, but any other method will eventually fail.

**DFL**

DFL is the preferred language to use in Design Architect. Schematics must be converted to DFL to be used in other sheets. DFL stands for Data Flow Language, it is an applicative language. An applicative language, in contrast to a functional language like Cor Pascal, has no variables and no assignments. No state can be defined at any place in a piece of applicative code. Only a relation between signals is defined. Otherwise said; it is a functional language with anything removed that defines an execution order.

A specification of the DFL syntax can be found in appendix A. In Design Architect DFL can
be edited and compiled.

An in-depth discussion of DFL will be given in chapter 7.

**DSPC**

A valid DSPC program is an element of a specific subset of the set of all valid C programs. Otherwise said, it is a C program with extra function calls that are used when simulating the component. DSPC is the language into which DFL is compiled for simulations. A component written in DSPC doesn’t need to be compiled in Design Architect, but is linked when starting a simulation, thus saving time. Another reason to describe a component’s behaviour in DSPC is that some constructs like serial-parallel conversion or state-dependent behaviour, are easier to describe and faster to simulate in DSPC.

Most DSPC programs are written by letting Design Architect generate a main body of code as specified by an interface that’s described in a form, and then adding code at certain parts of the generated code. This can be done by using the “use form to define interface” option when using the “Open C (DSP)” feature.

**5.2.4 DSP Lab**

After a design is compiled in Design Architect with the “Update DSP Viewpoint” option, it can be simulated and the simulation results can be viewed and processed in DSP Station.

In figure 5.7 a typical view of a DSP Lab session is given.

We will discuss the tools used in DSP Lab, that is the tools for time-domain analysis, viewing the results and for processing and storing the results.

When DSP Lab is started one first has to enter the working database, a database into which all simulation results are (temporary) stored.

**Simulation**

We only used the time-domain simulation tool TSim for our simulations, so that’s the tool we’ll describe. When starting TSim, one has to enter the component one wants to simulate. After some processing a form appears on which the user can enter information necessary for the simulation. The user can select which behaviour he wants to simulate (high-level or bittrue), the number of time-steps, the sample frequency and the behaviour of the simulation at the beginning and the end of the simulation. In this form the user must also enter input stimuli for at least one input signal by either creating a signal generator or by importing signal data from a database. He can also select which signals he wants to view. Only when he has completely filled in this form, the simulation can be started.

After the simulation is ready, the observed signals are stored in the working database and the user can use other tools to use these results.
Viewing Results

The user can view the results from the simulation with different tools. With the listing tool a list of all values of a signal can be viewed. The 2-D tool let the user plot a diagram where one signal is plotted as x-coordinate and another signal is plotted as y-coordinate. But most of the time the trace tool is used; it plots a signal along the time axis. Several signals can be plotted in one window to ease comparing these signals.

Processing and Storing Results

When doing several simulation runs, it’s very useful to be able to compare the outcomes of different runs. In DSP Lab one can permanently store results and manipulate. This makes it possible to look at both present and past results and draw conclusions from then. Several databases can be used at once and each database can be organised to contain for instance all significant results of a certain design.

Processing of results can be done with an expression editor that has built-in functions for doing signal conversions. Another option is to use standard processing tools that can be invoked from the menu bar inside DSP Lab. These conversion tools include for instance Discrete Fourier Transform to analyse results from a time-domain simulation in the frequency domain.
6

DAB System Modelling and Simulation

6.1 Introduction

We used DSP Station to model a complete DAB system, consisting of sender, channel and receiver. Because the purpose of using DSP Station was to use it as an aid for specifying part of a chip set for the receiver, the receiver is modelled in more detail than the sender.

In this chapter we’ll discuss how the DAB system is modelled and look at some important details. Where appropriate we’ll look at how descriptions evolved and what problems in DSP Station were encountered.

6.2 Original Bench

In figure 6.1 the sheet is depicted of the DAB system as was used by the Philips Research Laboratories and that was used as a starting point of system modelling at the Advanced Development Centre.

When the design was started, it was originally aimed to work for all three (and later four) DAB modes. However at the time the design was transferred to the Advanced Development Centre it was suitable only for mode I. This is not a big problem for mode I is the mode that needs the most system resources and if mode I works correctly, the other modes will probably work as well.

In this picture we can discern the sender at the top left of the sheet; it is composed of the components TFPRdata_gen, TFPRdiff_mod and TFPRencoder. The names of these components are slightly misleading for they don’t model the working of the TFPR symbol only. These components are responsible for respectively generation of a bit stream at the input of the symbol generator, a differential modulator and a component responsible for the inverse Fourier transform and the extension of the signal with a guard interval.

The sender is followed by a channel, modelled by the component TFPRchannel.

The signal is then presented to a component called TFPRfrontend that models the behaviour of the front-end of a DAB receiver. Then the signal is routed through the components TF-
PRcordic, TFPRdecoder and finally TFPRdiff_demod. These components model respectively the frequency error correction component, the component performing a Fourier transform, frequency error calculation and calculation of the channel impulse response that will be used for fine-time synchronisation, and finally the differential demodulator.

The block TFPRtime_sync stands for a component that models the behaviour of coarse-time synchronisation. The other components are used for signal conversion so that when the simulation is finished some important signals can be viewed without having to post-process intermediary signals.

### 6.3 Time Synchronisation Sheet

In figure 6.2 the sheet as developed at the Advanced Development Centre is shown. This sheet has added components to the original sheet, to model the fine-time synchronisation behaviour.

In comparison to the original sheet, much more conversion components are added so that amplitude and phase of all complex signals can be viewed directly after simulating the design. The most important components that are added are TFPRcalcCIR, TFPRfilter and TFPRcalcOptEff. These components respectively calculate the centre of impulse response, filter the result for use as a control signal to the front-end and calculate the optimal position for time-synchronisation.

In the rest of this chapter we will look at all components used in the fine-time synchronisation sheet.
6.4 Sender Components

6.4.1 Data Generator

The data generator component is depicted in figure 6.3. As input it accepts the size of a frame in symbols, as output it generates an array of 3072 bits per symbol. For the NULL symbol and the TFPR symbol these bits are all zero, in all other cases random bits are generated. These random bits model part of time- and frequency interleaved scrambled channels. Because the overall simulation doesn't focus on levels higher than the symbol layer, this is an appropriate model.

Note that the simulation sampling frequency is the same as the symbol rate. This is done
to get a sufficient simulation speed. Note also that the NULL symbol has a length equal to
the other symbols, which is not according to the standard. This should not influence our
simulation appropriateness because we’re mainly interested in the fine-time- and frequency
synchronisation behaviour of the system.

The data generator is modelled using 177 lines of DSPC code.

6.4.2 Differential Modulator

![Figure 6.4: Differential Modulator Component](image)

The differential modulator encodes the incoming 3072 bits onto 1536 carriers in a spectrum of
2048 carriers. At the output of the differential modulator an array of 2048 complex numbers
is presented. The complex numbers are in 24 bits signed format but are composed of a signed
real part of 12 bits and a signed imaginary part of 12 bits.

When looking sec at these numbers, for example in DSP Lab, they’re almost meaningless.
First the real and imaginary must be split and where necessary converted to amplitude and
phase to derive meaningful information from them. This is done by some conversion compo­
nents.

For the NULL symbol, only zeroes are presented at the output. The TFPR symbol causes
a fixed pattern of values to appear at the output. During the other symbols the 3072 bits
are split and assigned at a frequency slot as earlier explained. These slots are then mapped
to a phase shift and finally multiplied with the previous value of the differential modulator
output. This last operation performs the differential modulation as we have shown earlier in
this report.

One peculiarity of the mapping of the bits must be addressed here. The central frequency of
the spectrum is defined around zero, threfor the high part of the frequency slots are mapped
to frequencies 1 to 768, and because of the cyclicaty of the spectrum, the lower part of the
frequency slots are mapped to frequencies 1280 to 2047.

The code for the differential modulator uses 21891ines of DFL code, but only 141 lines if the
definition of the TFPR symbol is left out.

6.4.3 Encoder

The encoder transforms its input to a time-signal by performing an inverse Fourier transform
on it. It also should prepend a guard interval to the signal. That reads “should” because it
doesn’t implement this feature for unknown reasons.

At the output of the encoder an array of 2048 complex numbers is presented that represent the time signal without guard interval and only as baseband signal.

The inverse Fourier transform is performed by using eleven stages of radix-2 butterflies, that is by using an inverse fast Fourier transform. The constants used in the radix-2 butterfly are stored in an array of 2048 complex numbers.

The encoder is written in 1346 lines of DFL code, but 1024 lines are used for the array of complex constants.

6.5 Channel Components

6.5.1 Channel

The channel component uses C code used in other simulations to model the behaviour of different channel profiles. These channel profiles are standardised sets of parameters for testing DAB.

There are eight standard channel profiles defined, the channel component let the user select two additional profiles also. The eight channel profiles are modelled for different situations and for conditions ranging from fairly easy to difficult reception.

The channel profiles are defined by four parameters per profile. The first parameter is the number of groups. A group is a set of delayed transmitted signals clustered around a central
value. This central value has a delay that is the second parameter; each group has a mean delay associated to it. The third parameter is the spread of the delays per group. The fourth and final parameter is the relative attenuation of a cluster of signals.

For the eight channel profiles these values can be found in table 6.5.1.

### Table 6.1: Channel Profile Characteristics

<table>
<thead>
<tr>
<th>Profile</th>
<th>Group 1</th>
<th>Group 2</th>
<th>Group 3</th>
<th>Group 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RA</td>
<td>Δ</td>
<td>S</td>
<td>RA</td>
</tr>
<tr>
<td></td>
<td>μs</td>
<td>μs</td>
<td>μs</td>
<td>μs</td>
</tr>
<tr>
<td>1</td>
<td>1.0</td>
<td>0.11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1.0</td>
<td>0.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.66</td>
<td>1.0</td>
<td>0.64</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>0.91</td>
<td>0.28</td>
<td>0.09</td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td>0.33</td>
<td>0.33</td>
<td>0.50</td>
<td>30</td>
</tr>
<tr>
<td>6</td>
<td>0.20</td>
<td>0.33</td>
<td>0.60</td>
<td>20</td>
</tr>
<tr>
<td>7</td>
<td>0.0</td>
<td>1.0</td>
<td>0.05</td>
<td>100</td>
</tr>
<tr>
<td>8</td>
<td>0.0</td>
<td>1.0</td>
<td>100</td>
<td>1.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Profile</th>
<th>Group 5</th>
<th>Group 6</th>
<th>Group 7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RA</td>
<td>Δ</td>
<td>S</td>
</tr>
<tr>
<td></td>
<td>μs</td>
<td>μs</td>
<td>μs</td>
</tr>
<tr>
<td>7</td>
<td>2.5e-3</td>
<td>385</td>
<td>1.0</td>
</tr>
<tr>
<td>8</td>
<td>0.16</td>
<td>385</td>
<td>1.0</td>
</tr>
</tbody>
</table>

**Legend**

- RA : Relative Amplitude
- Δ : Delay of Group Centre
- S : Spread around Group Centre

The other two profiles the channel component allows are a Gaussian channel and a back-to-back implementation in which the channel passes the signal without distorting it.

The inputs of the channel consist of a signal input, an input for the selection of a channel profile and an input for the selection of the signal over noise ratio. To select a profile enter the number associated with it as presented in table 6.5.1, for the Gaussian channel enter 0 and for the back-to-back profile enter 11. The signal over noise ratio is measured in Decibels.

The channel component falsely models what the encoder component should have modelled; extension of the signal with a guard interval.

The component contains 1302 lines of DSPC code.

### 6.6 Receiver Components

#### 6.6.1 Front-End

The front-end mimics some functions of the analog front-end in a DAB receiver. At the
input of the front-end a complex signal is presented, together with three control inputs; one labelled FREQoffset, one WindowStart and one Gain. Let we relabel them $F$, $W$ and $G$ respectively, then the front-end models the following relation between input samples $a[i]$ and output samples $b[i]$ for $i \in \{0, \ldots, 2047\}$

$$b[i] = G \cdot a[i + W] \cdot e^{j \frac{2\pi}{2048} i \cdot F}$$

That is, the guard interval is discarded, the signal is attenuated and possibly frequency shifted.

The frequency offset is not used in the new chip set, another part of the chip, the CORDIC, is responsible for frequency error correction. The gain is controlled by an auto-gain circuit, but in our simulation it’s not important and has a constant value.

The window start is important for it let us select part of the signal that has suffered least from inter symbol interference.

The front-end component is written in DSPC and has 199 lines of code.

### 6.6.2 CORDIC

The input of the CORDIC component consists of 2048 complex samples, just as it’s output. The control input of the CORDIC labelled “shift” let the CORDIC do a frequency shift. The “shift” input is measured in Hertz.

The CORDIC is part of the DAB chip set and is the active component for performing frequency shifts to correct for frequency-errors. Another part of the chip is responsible for determining the frequency error.
6.6.3 Decoder

The decoder component is a relatively complex component. It does the conversion of the time-signal to a spectrum by a fast Fourier transform, it performs calculation of the channel impulse response and it calculates the frequency error of the received signal.

The inputs of the component are the frequency shifted signal from the CORDIC, the TFPR symbol on the TFPRref input, the current symbol number on input FrameSize and an input labelled “q” that we don’t use but can be used to scale intermediary results from the frequency error calculation.

As outputs we have “TFPRrecFREQ”, the spectrum of the input signal, “impulse”, the calculated impulse response of the channel, “freqerror”, the frequency error in Hertz, “CFE”, the frequency error in units of the carrier spacing, and “Power”, the squared amplitude of the signal input of the decoder.

The frequency error is used in a control loop to correct for the frequency error on a frame basis, that is, the frequency error is calculated by using the TFPR symbol only and the output port has that value for all symbols after the TFPR symbol.

The impulse response is used for fine-time synchronisation as we will see in subsection 6.11.

The spectrum is used to distil the original bit stream presented at the differential modulator. We will therefor now first look at the differential demodulator.

The decoder consists of 1924 lines of DFL code, when excluding the constants for the Fourier transform, 900 lines.

6.6.4 Differential Demodulator

The differential demodulator reconstructs the original bit stream from the spectrum offered by the decoder. Besides that input, it has inputs for the number of the current symbol and for the original bit stream.

At the output we find the output bit stream, but also the differential demodulated complex value associated with a carrier in the spectrum. Another output counts the number of bit errors per symbol.
This component is relatively simple and consists of 123 lines of DFL code.

In the chip set a Viterbi decoder is implemented, however the Differential Demodulator component doesn’t use the extra accuracy in the spectrum at the input to do soft-decision; instead it uses only the top-bit of the real and the imaginary part of a complex value associated with a carrier to output a single bit.

Another component could be devised that uses the earlier mentioned differential demodulated complex values to do soft-decision. For our purpose – simulating fine-time synchronisation behaviour – an extra component isn’t needed, for we only need a measure of the relative performance.
7

DSP Station Evaluation

7.1 Expectations of the Philips ADC concerning DSP Station

At the Philips Advanced Development Centre, several expectations concerning DSP Station were made, before a part of the DAB project was specified using DSP Station. Other groups had experiences with the development of a chipset for MPEG, this group used DSP Station for the first time. Previously designs were done on traditional tools; for instance simulations were done in Pascal programs with the aid of libraries containing specific DSP functionality.

Expectations of DSP Station were:

- Easy specification of designs
- Easy modifications in design
- Simple to use tools
- Easy exchange of the design with the Philips Research Laboratorium
- Should save time relative to the current used design methods

7.2 Experiences of ing. B. van Steenbrugge

At the Philips Advanced Development Centre ing. B. van Steenbrugge already had experience with DSP Station. He used DSP Station to develop an MPEG I and later an MPEG II decoder IC. The author interviewed ing. B. van Steenbrugge to learn about his experiences with DSP Station. This section is an overview of his experiences, neither complete nor extensive.

Both the MPEG I and the MPEG II design were written in procedural DFL. This embodies a different model than the pure applicative use of DFL. Filter generation with several tools worked easy and simulations were simple to do; porting to Mistral 2 architecture worked fine.

The MPEG I decoder gave no problems, but during simulation of the MPEG II decoder, the simulation gave problems; it wasn't usable because of the big size of the MPEG II decoder DFL description. With MPEG II procedural DFL had to be manually mapped onto applicative
DFL else compilations would give errors. Better support is needed for procedural DFL, for instance global variables should be supported.

The iterative design cycle of specification, compilation, simulation and evaluation takes too much time. It’s not clear how certain DFL constructs are synthesized into architectural constructs, this leads to obfuscation.

For certain desired behaviour, more than one method exist to accomplish that. However some methods won’t lead to the desired result, without being clear where such a method fails. The error messages are cryptic in their origin, it’s not clear what caused a certain error.

It’s difficult to impossible to come to a hierarchical design. Regular structures like filters are relatively easy to implement, but irregular structures are difficult to get to work. Synthesizing of regular structures seems to lead to nearly optimal solutions.

The manuals aren’t up-to-date; while using a new software release with new features one still has to use older manuals that don’t describe those features.

It seems that for DSP like algorithms with a data-oriented design, DSP Station works fine; designs that are irregular or incorporate significant control sections seems to lead to problems in DSP Station.

7.3 Experiences of other users

In february of this year a DSP Station user’s meeting was organised. Several users presented their designs made with the aid of DSP Station. Some users presented their experiences with DSP Station. Some of these experiences, related to the work the author has done in DSP Station, will be described in this section.

The designers of the DAB V2 chipset at the Philips Research Laboratorium remarked the lack of debugging facilities for DFL.

The SHOARMA2 team made the following remarks. A DFL debugger is needed for procedural DFL. The simulation iteration cycle has to be shortened. There’s a need for an improved DFL editor, DFL online syntax checker/formatter, DFL make, improved display performance, extended listing tool and extended logging functionality. The tool quality needs to be improved for all tools.

The team had the following overall impressions. DFL and TSIM are allright for the system design of the ADPCM (Adaptive Differential Pulse Code Modulation). If the processor design had be done in the classical VHDL way, the design would have finished at least a month earlier. A lot of problems had to be solved, much energy had to be spent on non-design issues. Child diseases in DSP Station disturb the design process.

7.4 Experiences of the author

The use of DSP Station for simulation of the Digital Audio Broadcasting system has revealed both strengths and weaknesses. Some experiences are shared with other DSP Station users, some experiences are looked at with a more detailed inspection of the workings of DSP Station.
that account for certain (un)wanted behaviour.

The experiences shall be grouped into five categories: transfer of design, specification of design, simulation of design, analysis of simulation results and general tool experiences. The used version of DSP Station was the A4.8 release.

7.4.1 Design Transfer

Part of the DAB system was already designed at the Philips Research Laboratorium. That design was transferred to the Philips Advanced Development Centre via internet. Basically all hierarchies used in DSP Station are built upon the hierarchical directory structure of UNIX. To transfer a design, the directory containing the design and all its subdirectories have to be transferred.

A problem with this method is that references to parts of a design are related to pathnames. Those pathnames however change when a transfer is moved to another machine. The design is then no longer valid. A solution to this is to use a symbolic name via the UNIX 'set VARIABLE PATH' command, that points to the directory in which the design is copied, and use that symbolic name inside DSP Station.

The design from the Philips Research Laboratorium contained part of a fixed path. To resolve that, a similar directory structure had to be set up with a symbolic link to the 'real' design directory.

Other problems that were encountered were compilation problems that had to do with properties. These problems were so severe, that DSP Station a couple of times completely crashed and the design files were severely damaged and had to be redone again. Later on in the design, problems were encountered that had to to with properties of the initial design, but were only then revealed.

7.4.2 Design Specification

A main problem with specifying a design in DSP Station is the difficulty in mixing sheets and DFL or DSPC code. Another main problem is the difficulty in doing hierarchical design in DFL with possible reusable components. One of the things that should be possible is to pass parameters (especially type parameters) to functions. Parameter passing is in the case of types not possible. This is a serious drawback of DSP Station that should be changed to be able to use hierarchical designs with components chosen from libraries that are devised by other DSP Station users, e.g. in a IC design group.

Another problem is the use of properties. The design tools don't indicate what properties can be used and don't put a restriction on those properties. However when a design is saved or compiled wrong properties can crash DSP Station.

It is also possible to perform a task (for instance generating a symbol for a DFL description of a component) in several ways, but only one way will work, some other ways will lead to a crash of DSP Station.

Another problem, of the same order as the parameter passing problem, is the mixing of
pure applicative and declarative DFL code. Both are based on a different underlying model. Another similar problem are the component info blocks. Component info blocks communicate information to the 'environment’ but a better mechanism would be to make such information exchange part of the language.

7.4.3 Design Simulation

Simulation in DSP Station is easy to do. The only problem is simulation speed. For each change, compilation has to be done in Design Architect and a simulation run must be setup in DSP Lab. The total simulation time grew to an unacceptable level. At one point DSP Station had to be abandoned totally, for simulation times were in the order of half an hour, where simulations on a PC could be done in a couple of minutes.

Another point is that for certain parts, DSPC does give an increase in simulation speed, but DPSC isn’t synthesizable itself. However, one is almost forced to use DSPC instead of DFL for operations like serial-parallel conversion to get reasonable simulation speeds.

No real debugging facilities are available to debug or trace a design’s execution, for this purpose 'print’ statements must be used at the moment.

7.4.4 Simulation Result Analysis

DSP Lab is a good tool to view simulation results. It has a lot of functions for viewing results. Also the ability to store, view and post-process simulation results in a database are very handy. A tool that would allow easy graphical viewing of arrays would be very handy. The listing tool doesn’t function correctly when viewing a lot of signals.

7.4.5 General

The installation of a new version of DSP Station resulted in some problems. Some of the problems that couldn’t be solved even after a considerable amount of effort was put in it were printing problems. It was nearly impossible to print documents, sheets or screens from DSP Station without errors that either made the result unusable or gave no result at all.

The time to learn to use DSP Station was about three to four weeks. The training workbooks provided a good way to learn basic concepts quickly. The online documentation is overall fine, but sometimes out of date. A printed manual for often used tools is to be preffered. A manual that gives an overview of all concepts and tools and their relation is lacking but is needed. The very important topic of properties is not clearly covered by the online manuals.

The window manager doesn’t work properly in all cases. Sometimes some parts of important information disappears or can’t be accessed. The functioning of the program concerning selections should be changed; for instance when selecting a signal for observing, the viewing window shows again the first signal in a list instead of the selected signal. A lot of such improvements could be made to make the program easier and faster to use. Another example is the good idea of strokes; however some strokes are nearly impossible to perform with a
mouse. To succeed in the 9-5-1-2-3 stroke for example, one has to repeat it about ten times before being successful.

7.5 Evaluation

We'll end this chapter with some overall conclusions and recommendations concerning DSP Station.

DSP Station has the potential to become a very useful tool for modeling and synthesizing DSP algorithms. In order to reach that position all kinds of smaller and larger errors should be removed from the tools. The speed of simulation must be improved.

If DSP Station must be suitable for modeling algorithms in general, some real changes in the DFL language and the mixing of schematics and DFL should be made. The language must be made suitable for hierarchical design and a typing system must be introduced. Hierarchical design is a must for large designs and for designing in groups where reusable components must be available. The problem with this however is, that this may introduce a lot of unwanted trouble and may lead to a language that isn't as simple and easy to use as DFL now is.

Maybe one has to choose for another language — like VHDL, that is suited for hierarchical design and has a good typing system — to design non-DSP algorithms in, and use DSP Station for those (parts of) algorithms that are typically DSP-like. For non-DSP like algorithms behaviour can also be modelled via one of the available mathematical packages like Mathematica, Maple or Mathcad. That has however the disadvantage that it isn't integrated like DSP Lab.
Part III

DAB Fine Time Synchronisation Optimisation
8

Introduction

In this part we will look at the concepts necessary for correct functioning of a Digital Audio Broadcasting receiver. As we examined earlier, several intermediate data structures are defined to let Digital Audio Broadcasting work. These data structures must be reconstructed at the receiver. This reconstruction has to be done at precisely the right moment, else data is corrupted or even lost.

What we’re aiming at is the concept of synchronisation; we use certain parts of the received information to determine when we must transform data into another form.

In a Digital Audio Broadcasting receiver several kinds of synchronisation take place. First there’s synchronisation at the analog front-end. This synchronisation assures that the receiver is tuned to a certain frequency and that the signal is kept at a certain strength (level?).

In the digital domain, both frequency- and time synchronisation are used. We will here shortly introduce the concepts behind these synchronisation mechanisms, and will pay our attention to time-synchronisation in the rest of this part.

Frequency synchronisation uses the Time Frequency Phase Reference symbol to calculate a possible frequency error. Coarse frequency synchronisation can determine a frequency error in an whole number of carrier steps. Fine frequency synchronisation uses that as a basis to estimate the error to an accuracy of about 10 Hz. This information is used to drive the CORDIC, that can perform a frequency transposition of the received signal.

Time synchronisation is divided in coarse- and fine time synchronisation. Coarse time synchronisation uses the envelope of the signal to determine the NULL-symbol. This is used to coarsely synchronise to a transmitted Digital Audio Broadcasting frame. Fine time synchronisation uses the Time Frequency Phase Reference symbol to calculate the impulse response of the transmission channel that is then used to finely synchronise to a transmitted symbol.

Synchronisation has shown to be of crucial importance to the performance of a Digital Audio Broadcasting receiver. The constantly changing characteristics of the channel, together with Doppler effects caused by moving of the vehicle, makes it necessary for the receiver to adapt to the signal fast and accurate.

We shall see how this can be implemented for time-synchronisation, especially fine time synchronisation, for that is our main design problem. It will be the main topic in the rest of this part.
9

Analysis

In this chapter we will look at several aspects concerning synchronisation. We’ll start with a short overview of frequency synchronisation and then pay our attention to time synchronisation.

9.1 Frequency Synchronisation

Frequency synchronisation is done by using the Time Frequency Phase Reference symbol as basis for a calculation of a frequency error, that is then used to transpose the incoming signal in frequency. Therefore it follows that this is done only once a frame, and is used for all symbols in that frame. We’ll start with coarse frequency synchronisation, followed by fine frequency synchronisation.

9.1.1 Coarse Frequency Synchronisation

With coarse frequency synchronisation, the stored TFPR symbol at the receiver is correlated with the received TFPR symbol for 15 different positions. Due to the special properties of the TFPR symbol this will give a maximum value for one of those values. This represents the frequency offset in number of carrier spacing steps from zero, that is no frequency error.

9.1.2 Fine Frequency Synchronisation

Fine frequency error calculation uses the result from the coarse frequency calculation to get two numbers related to the carriers located to the left and right of the calculated frequency error. This is then used to make an estimate of the fine frequency error, which can have an accuracy of about 10 Hertz.

To explain all the details isn’t necessary to understand fine-time synchronisation. We can assume that frequency errors will be neglectable small.
9.2 Time Synchronisation

We will now look at mechanisms used for time synchronisation, starting with coarse time synchronisation, followed by fine time synchronisation.

9.2.1 Coarse Time Synchronisation

Coarse time synchronisation uses the envelope of the received signal to determine the start of a frame. This is possible because at the start of a frame a NULL symbol is transmitted, a symbol with (almost\(^1\)) no carriers transmitted. That means that the transmitted power equals (nearly) zero and thus envelope detection can be used.

This was initially done on a sample basis and the values that are calculated are

\[
y_1[n] = c_2 \cdot y_1[n-1] + c_1 \cdot |x[n]|
\]

and

\[
y_2[n] = c_4 \cdot y_2[n-1] + c_3 \cdot |x[n]|
\]

With \(x[n]\) the sampled input signal. Now \(c_2\) and \(c_4\) determine the time constants of these low-pass filters; for \(y_1[n]\) it is chosen in the order of the length of a symbol. For \(y_2[n]\) it is chosen in the order of 10 symbol periods. The gain of the filters is given by the ratio \(\frac{c_1}{c_2}\) and \(\frac{c_3}{c_4}\), the second ratio must be smaller than the first ratio.

When these constants are properly chosen, the instants that these signals cross, that is \(y_1[n] = y_2[n]\), indicate the start or end of the NULL symbol.

In extreme circumstances, this scheme wouldn’t give correct answers. To solve this, the filter represented by \(y_1[n]\) has been replaced by a matched filter that uses a sliding window of 512 samples. That solution has proved to work even in aforementioned extreme circumstances.

9.2.2 Fine Time Synchronisation

When we’ve coarsely synchronised a new frame, we use the Time Frequency Phase Reference signal to finely synchronise. We do this by selecting part of a received symbol as useable part, and the remaining part we discard. The purpose of this operation is to eliminate or reduce inter-symbol interference (ISI). In this way we finely synchronise to a symbol.

Look at figure 9.1 to see how this principle works.

In this figure we see three signal paths. At the receiver the sum of these signals is picked up and converted to a stream of samples. The signal consist of a sequence of symbols, and each symbol consists of a guard- and a useful period. We see in the figure that ISI occurs in a period that the useful period of a symbol overlaps with the guard period of the next symbol.

\(^1\)except for TII carriers
If we discard part of the received signal where ISI occurs, and use the part of the signal where ISI is absent, we have the least errors.

That part is in this figure referred to as the useable part. But how do we determine what this useable part is?

We shall later see that we can use the Time Frequency Phase Reference signal to determine the channel impulse response and that the start of the useful symbol period of in a signal path causes a peak to appear in the impulse response.

The position of these peaks we can use to select the useable part, as we will later see, but first we shall look at the effects the channel has on the transmitted signal.

Although we can in this way reduce or even eliminate ISI, we have of course, as can also be seen from this figure, to deal with symbol interference.
10

Design

10.1 Conditions for Optimal Fine-Time Synchronisation

In this section we’ll look what conditions we must satisfy to get optimal fine-time synchronisation.

10.1.1 Window Positioning for One Signal Path

Let’s look what happens if one signal path is present, see figure 10.1.

In this picture we see two buffers; the full symbol buffer that contains \( K + D \) input samples, and the useful symbol buffer that contains \( K \) samples. The useful symbol buffer we’ll refer to shortly as ‘buffer’. The buffer is positioned within the full symbol buffer at position \( W \).

We assume that we haven’t yet synchronised the start of our buffer to the signal, but that we’ve synchronised the full symbol buffer with coarse-time synchronisation fairly well to the received frame. Further we assume for now that the channel is ideal and thus the received signal equals the transmitted signal.

At the bottom of the picture is the buffer that is filled with the useable part of the received signal and that is used as input to the FFT processing. Because the guard period of the signal is a cyclic extension of the useful period, the buffer contains the complete symbol. But the original signal is shifted to the right by \( k \) samples, that is, the buffer contains
\[ \hat{y}[n - k] \]

If the channel impulse response is calculated, it will give a result of

\[ \delta[n - k] \]

Thus we will find a delta peak at position \( k \) in the buffer that corresponds to the start of the IFFT window in the transmitter.

If we look where we can position the buffer so that no ISI occurs, then the buffer can be aligned to the right of the signal; in that case \( k = 0 \), or we shift the buffer to the left and then \( k = D \), thus

\[ 0 \leq k \leq D \]

Thus we can conclude that we must position the buffer in such a way, that the peak in the channel impulse response is positioned in the first \( D + 1 \) samples of the buffer.

### 10.1.2 Window Positioning for Multiple Signal Paths

We can extend the situation, to a situation where we have multiple signal paths with undistorted signals. We can write the received signal as

\[ \sum_{i=0}^{M} \hat{y}[n - k_i] \]

which will give a channel impulse response of

\[ \sum_{i=0}^{M} \delta[n - k_i] \]

Now define the spread \( G \) of the delays \( k_i \) as

\[ G = \max_{i,j} \{|k_i - k_j|\} \]

If \( G < D + 1 \) we can position the buffer in such a way that we don’t have ISI, by getting all the delta pulses in the first \( D + 1 \) samples of the buffer.

Now let’s look what the results are if we look at channel effects.

### 10.1.3 Optimal Positioning of Window

We’ve seen that if the spread in \( h(t) \) is less than \( D \), we can position the window that no ISI occurs. Of course we must now devise a way to calculate a value to set the window position.
to and we must take some action in order that the position of the buffer changes relative to
the transmitted signal.

Positioning of the Window

How do we position the window? We can only set the position of the window once a frame,
for we use the TFPR symbol to calculate the window position. We can use the calculated
value however only for the next frame. For slowly varying channel characteristics this method
has proved itself accurate enough to follow channel characteristics. One encountered problem
was that the rapid changing of the window position each frame caused other problems, so the
mentioned delay of one frame isn't a problem at all.

We're also limited with changes in window positioning each frame, for we can't shift the
window too much to the left for that would lead to acausalities in the next frame. Speaking
over a period of time where the channel is changing, but the channel profile more or less stays
the same, we can say that the mean position of the window will be more or less the same.
Therefore, we can't shift the window too much to the right, for we never know how much in
that case the window had to be shifted to the left again in the next frames, and shifting to
the left has a strict limit.

For modes I,IV,II and III the number of positions the window can be shifted left or right per
frame are respectively 126,63,32 and 16.

We'll now introduce a position shift in the channel impulse response buffer for convenient
interpreting of calculated window positions. Our concept of positioning the window has been
to move the window in order to collect all impulses in the first $D$ samples of the window.
We could also say that we were clustering samples around a point midway those samples, at
position $\frac{D}{2}$. We could consider it as a fixed centre and by changing the window start we can
'move' the impulses towards it.

Therefore we renumber the positions in the channel impulse response buffer and call the centre
position 0. This view is depicted in figure 10.2.

Our window is still $K$ positions wide. For reasons of symmetry – symmetry that will be
exploited in an efficient implementation – the size of the window is reduced to $K-1$ positions.
The window starts at $-K/2 + 1$ goes through 0 to end in $K/2 - 1$.

With this new concept we will look at several strategies to optimally or sub-optimally position the window.

**Maximise In-Guard Energy**

One strategy could be to calculate the energy over $D$ samples for all positions within the window and search for the position that gives the highest value. In a sense you then have the maximal energy in the 'guard' (let’s call the $D$ samples around the centre guard, realising that this is not the real guard interval that’s discarded, but is closely related to it).

Let $w$ be the position within the shifted window that gives that maximal energy, then the maximal energy is expressed by

$$\sum_{i=w}^{w+D-1} p[i \mod (K-1)]$$

To calculate that position $w$, we could use a sliding window, so that the total number of calculations involved would equal $2 \cdot K - D - 2$ additions ($D$ initial additions plus one addition and one subtraction for $(K-1-D)$ positions).

The problem with this calculation is that $w$ can take any value, and isn’t thus restricted to the earlier mentioned maximum shift per frame. The value of $w$ will also show a 'jumpy' behaviour, so we can’t use it to move the window-start even over a couple of frames to some optimal position, for the calculation of $w$ could give a very different values each frame.

**Maximise Ratio In-Guard/Out-Guard Energy**

A similar approach to the previous idea, is to maximise the ratio of the energy in-guard versus the energy out-guard. That is, calculate the total energy over $D$ points, calculate the energy of the remaining $K-1-D$ points, and divide the two. Then search for the maximum ratio along the entire buffer, and use that position for determining the window-start.

The problem with this approach is the same as with the previous one.

**10.1.4 Calculation of Centre of Gravity**

Another way to find the optimum position of the window, is to calculate the centre of gravity and subtract $D/2$ from that. The centre of gravity gives the position in the buffer that is central in the sense that $\frac{1}{2}(K-1)$ samples to the left of that point, the same amount of energy is accumulated, as $\frac{1}{2}(K-1)$ samples to the right of that point.

The calculation that is done, can be written down as
with \( p[n] \) the impulse response power and \( COG \) the resulting centre of gravity.

If no additive white Gaussian noise is present, then around the centre of gravity the impulse power will be clustered and we move the centre of the guard of our window to the centre of gravity.

If additive white Gaussian noise is present, it will have the same mean value for all positions in the window, so \( \frac{K}{2} \) points to the left and to the right of the centre of gravity (for the window is cyclic), roughly the same power will be present as a result of that noise. It doesn’t thus influence the centre of gravity.

The centre of gravity gives us a good way to calculate the optimal position of the window start. Fluctuations in the calculated impulse power will result in fluctuations around the centre of gravity, so stability is assured.

### 10.2 Weigh Functions

We’ll introduce a general way to calculate a value for a window position by a so-called weigh function. If \( w[n] \) is such a function \( p[n] \) is the calculated impulse power then let \( c \) be the calculated position according to

\[
  c = \frac{\sum_{k=-(\frac{K}{2}+1)}^{\frac{K}{2}-1} w[k] \cdot p[k]}{\sum_{k=-(\frac{K}{2}+1)}^{\frac{K}{2}-1} p[k]}
\]

If the weigh function \( v[n] \) is a scaled and transposed version of \( w[n] \) according to

\[
v[n] = a \cdot w[n] + b\]

then if \( c_w \) is the position given by the weigh function \( w[n] \) it follows that the position \( c_v \) given by the function \( v[n] \) satisfies
thus the position $c_v$ is also a scaled and transposed version of $c_w$.

Note that this is an important result. If we have found a weigh function that functions well, we can reduce the spread of a random variable that fluctuates around the origin of the weigh function, if we choose $a < 1$ and change the original weigh function by multiplying it by $a$. Also, $b$ can compensate for any constant offset in the window position, for instance caused by transposing the window to centre the window around zero.

In DAB the weigh function is chosen to be point-symmetrical around position 0. This has several nice properties.

The point-symmetrical symmetry can be expressed as

$$w[-n] = -w[n]$$

We can then rewrite $c$ as

$$c = \frac{\sum_{k=0}^{K-1} w[k] \cdot (p[k] - p[-k])}{\sum_{k=-K+1}^{K-1} p[k]}$$

and we have thus achieved a reduction in computational complexity, for per two samples we perform a subtraction $(p[k] - p[-k])$, a multiplication $(w[k] \cdot)$ and an addition to the
total value of the denominator. Furthermore the nominator can be simply implemented as a running total.

Another nice property is that we reduce the influence of noise. We can write $p[k]$ as

$$p[k] = i[k] + n[k]$$

With $i[k]$ being information and $n[k]$ additive white Gaussian noise with mean $E_{\text{Gauss}}$ and variance $\sigma^2$. If we look at noise only, we can write

$$p[k] = n[k]$$

Because white Gaussian noise is uncorrelated between samples, we can see the term $p[k] - p[-k]$ as a subtraction of two Gaussian distributed random variables, the resulting random variable has a different mean. If the term $p[k]$ is presented by a Gaussian random variable $P$, the term $p[-k]$ by $Q$ and the resulting term as $R$, then

$$R = P - Q$$

and thus

$$E[R] = E[P] - EQ = 0$$

but

$$\sigma_R^2 = \sigma_P^2 + \sigma_Q^2 = 2\sigma^2$$

thus we reduced our noise influence to a noise with mean 0 but with a doubled variance.

That last equation follows from


$$= E[P^2] - E^2[P] + E[Q^2] - E^2[Q] = \sigma_P^2 + \sigma_Q^2$$

If $i[k]$ is of the form

$$i[k] = C + z[k]$$

with $C$ a constant and $z[k]$ a variable with a mean of zero, we can see that the constant $C$ is eliminated in the calculation of $p[k] - p[-k]$, for that is then reduced to

$$p[k] - p[-k] = (C + z[k] + n[k]) - (C + z[-k] + n[-k]) = z[k] - z[-k] + n[k] - n[-k]$$

A further restriction we can lay upon a weigh function, is that it should be a monotonically non-decreasing function. It should be 0 for position 0, positive for positions to the right of
the central position and negative for positions to the left. In that way the direction of change of the window position will always be correct, whatever the exact form of the weigh function will be. That is a big advantage for stability in control of the window position.

Let’s look at the centre of gravity algorithm again. We can model the centre of gravity algorithm by a weigh function of

$$w[n] = n$$

that is, \(w[0] = 0\) and \(w[n]\) is less than zero for \(n\) left of the centre of the guard, and \(w[n]\) greater than zero for \(n\) right of the centre of the guard. The centre of gravity is also a symmetrical weigh function, and it is monotonically non-decreasing.

Our main goal is to keep all impulse power inside the guard. Any main energy concentration outside the guard must as soon as possible be controlled back into the guard. It therefore seems attractive to accentuate certain regions in the window, so that our weigh function becomes more sensitive to relevant energy concentrations in those regions.

We will use power shaping for this purpose; a function that operates on the calculated impulse response power. Let \(p[n]\) be the impulse power, \(s[n]\) be the shaping function, then \(p'[n]\) will be the power shaped impulse power according to

$$p'[k] = p[k] \cdot s[k] \text{ for all } k$$

The modified weigh function then becomes

$$c = \frac{\sum_{k=-\frac{K}{2}+1}^{K-1} w[k] \cdot p'[k]}{\sum_{k=-\frac{K}{2}+1}^{K-1} p'[k]}$$

The shaping function \(s[n]\) determines what parts are important. In DAB the power shaping function is chosen to be

$$s[n] = \begin{cases} 1 & \text{for } n \in \{-\frac{D}{2}, \ldots, \frac{D}{2}\} \\ P & \text{elsewhere} \end{cases}$$

with \(P\) a constant greater than one. The constant \(P\) will be set in relation to the noise energy measured; if noise is low, it can be high, if noise energy is high, we can’t make it too big, else we can’t discern between signal- and noise power.

The effect it has on the weigh function is as follows; Because it is symmetrical around 0, noise power has no influence on the calculated \(c\), signal power however is amplified if it is concentrated outside the guard. This has the desired effect that if a peak is located outside the guard, the algorithm will give a high value for \(c\) (speaking of the case of a symmetrical weigh function around 0) in the direction that the window will move so that the peak will move into the guard.
A problem that occurs in using the centre of gravity algorithm, is that it is accurate and fast, but due to the changing channel – however with a fixed profile – the calculated position will move wildly around a general constant mean value. This introduces several unwanted effects in other parts of the receiver. We therefore want to reduce the spread around a certain mean window position.

Another condition that we must meet is the maximum number of positions the window can be moved per frame. That makes it impossible to use a centre of gravity algorithm, for it uses a larger range than that limit.

A modified centre of gravity algorithm was tested for its performance; the piecewise linear centre of gravity algorithm. It divides the window in several regions; a region around the central position 0, where a weigh function \( w[n] = n \) is used, a region from the left and the right of the former region, to the ends of the guard interval, where a weigh function \( w[n] = a \cdot n \) is used \((0 < a < 1)\) and a third region to the ends of the window, that uses a weigh function \( w[n] = b \).

In the stationary case, when the main power is concentrated in a small part of the guard, the region around 0 gives us accuracy, the region next to that gives us a reduced next window position (and thus a less bigger position change) and the last region limits the window change to the previously discussed limit.

This algorithm has proven to be both accurate and to reduce spread around a certain mean.

All weigh functions were to be implemented by a ROM table that would hold all values of \( w[n] \), for all four DAB modes. Even with a point-symmetrical weigh function, this would give a ROM table of 1916 positions or almost 2Kbyte if 8 bits words were used. This would occupy a relatively large portion of the chip. Besides that, once the weigh function is chosen, it can't be modified anymore.

### 10.3 Adjustable Weigh Function

A solution to the problem of a ROM table would be to calculate the weigh function itself. If a function is chosen with some parameters, the weigh function can be changed, even when implemented on chip. Of course, the calculation of the weigh function can't take too much time or resources on a chip.

A weigh function must be chosen that complies to the earlier mentioned characteristics of point-symmetry and monotical non-decrease. Possible easy to change and sensible parameters would be a desirable feature.

The author came up with a function that will meet those criteria and will function like the piecewise linear centre of gravity, a function that proved itself useful in practice.

The intended function is

\[
w[n] = K \cdot \left(1 - e^{-b \cdot n}\right)
\]

for \( n \geq 0 \). That function is easy to implement;
Design

\[ w[n] = K \cdot (1 - a[n]) \]

with

\[ a[n + 1] = a \cdot a[n] \land a[0] = 1 \land a = e^{-b} \]

per value of \( w[n] \) the costs are two multiplications and one subtraction. The value of \( K \) can be moved to the end of the calculation and thus only one multiplication and one subtraction per value of \( w[n] \) remains.

We will now look at the continuous version of \( w[n] \) to be able to perform some calculations. Let \( w(t) \) be the continuous function

\[
 w(t) = \begin{cases} 
 1 - e^{-b \cdot t} & \text{for } t \geq 0 \\
 -1 + e^{b \cdot t} & \text{for } t \leq 0 
\end{cases}
\]

We don't look at the effects of \( K \), and will later on look at its effect. Note that \( w(t) \) is continuous, also for values of \( t \) less than zero.

It fist derative is line symmetrical around 0 and continuous

\[
 w'(t) = \frac{d w(t)}{d t} = \begin{cases} 
 b \cdot e^{-b \cdot t} & \text{for } t \geq 0 \\
 b \cdot e^{b \cdot t} & \text{for } t \leq 0 
\end{cases}
\]

Note that for small values \( \epsilon \) of \( t \), this will be equal to

\[
 w'(\epsilon) = b
\]

that is the same as the linear centre of gravity where \( b = 1 \). The limit for large positive and negative values of \( t \) are

\[
 \lim_{t \to \infty} w(t) = \lim_{t \to \infty} 1 - e^{-b \cdot t} = 1 \\
 \lim_{t \to -\infty} w(t) = \lim_{t \to -\infty} -1 + e^{b \cdot t} = -1
\]

If we now take in account the value of \( K \), we'll see that the derative of \( w(t) \) will be for small values of \( t \)

\[
 w'(\epsilon) = K \cdot b \cdot t
\]

and the limits for large \( t \) will be

\[
 \lim_{t \to \infty} w(t) = K \\
 \lim_{t \to -\infty} w(t) = -K
\]

We thus have approximately a linear piece with the equation
\[ r[n] = K b \cdot n \]

and a part with nearly a constant value

\[ c[n] = K \]

The lines cross at \( t = \frac{1}{b} \), where \( w[n] \) will have the greatest deviation from this piecewise linear 'approximation'.

For \( K \) and \( b \) equal to 1 \( w(t) \) is plotted in figure 10.3

![Plot of w(t) together with approximating lines](image)

**Figure 10.3:** Plot of \( w(t) \) together with approximating lines

In the continuous case the difference between the approximation and \( w(t) \) at that point equals

\[ K - w\left(\frac{1}{b}\right) = K - K \left(\frac{e - 1}{e}\right) = \frac{K}{e} \approx 0.37 \cdot K \]

Thus two parameters with a clear interpretation can be used to control the maximum window offset per frame \( (K) \) and the slope of a COG-like nearly linear piece around zero \( (K \cdot b) \).

Note that \( K \) also controls the speed of convergence (the bigger \( K \) is, the sooner the window is controlled to zero) and that \( K \cdot b \) also controls both sensitivity (when \( K \cdot b = 1 \) small changes around zero are almost in one frame reduced to zero, but if \( K \cdot b \) is a lot smaller than one only large changes will result in a change in window position) accuracy (if \( K \cdot b \) is larger than 1 — not preffered — certain window positions could be unreachable ) and speed (if \( K \cdot b \) is larger, the algorithm will control quicker — in lesser steps — the window to zero).
10.4 Validness of the Adjustable Weigh Function

Now that we've introduced the weigh algorithm, we can ask ourselves, does it give us a sensible result? The original COG could be interpreted as calculating the centre of 'mass'; that is, it calculated the position that was the centre of energy concentration. The rationale behind it was that by positioning that centre in the middle of the window, probably all energy was located in the guard (of course the centre of gravity didn't tell us how the energy was spread around that centre).

The introduction of power shaping also seemed a sound principle; if any energy was located outside the guard, it would give an extra 'push' to the window position to quickly reduce that energy concentration.

What about the algorithm just introduced? Does it calculate something like a centre of mass? Does it reduce the spread of window jittering around zero? Does it control all values back to zero?

To answer this, consider the following calculations.

The centre of gravity had a formula like

\[
COG = \frac{\sum n \cdot p[n]}{\sum p[n]}
\]

and the weigh function had

\[
c = \frac{\sum w[n] \cdot p[n]}{\sum p[n]}
\]

Clearly not the same. If we rewrite \( w[n] \) however as

\[
w[n] = n \cdot d[n]
\]

we would get

\[
c = \frac{\sum n \cdot d[n] \cdot p[n]}{\sum p[n]}
\]

Now that looks closely like

\[
\frac{\sum n \cdot d[n] \cdot p[n]}{\sum d[n] \cdot p[n]}
\]

that could be interpreted as the COG of a power-shaped \( p[n] \). Now for our solution we can express \( d[n] \) as (for \( \geq 0 \)) continuous function as

\[
K \cdot \frac{1-e^{-bx}}{x} = K \cdot \frac{1-(1-bx+(bx)^2/2!)-(bx)^2/2!-...}{x} = K \cdot b \cdot \frac{1-(bx+(bx)^2/2!-...}{x}
\]
Note that for \( x = 0 \) this is not defined, but has a limit of \( K \cdot b \). A plot of this function for \( K \) and \( b \) equal to 1 is in figure 10.4

![Plot of function \( d(x) \)](image)

Thus \( d[n] \leq 1 \) for \( K \cdot b \leq 1 \). That means that

\[
0 \leq \sum d[n] \cdot p[n] \leq \sum p[n]
\]

and thus

\[
\left| \frac{\sum n \cdot d[n] \cdot p[n]}{\sum p[n]} \right| \leq \left| \frac{\sum n \cdot d[n] \cdot p[n]}{\sum d[n] \cdot p[n]} \right|
\]

and that indicates that our function gives a reduced value in comparison to a COG like function, and that is wanted behaviour for that reduces spread around zero. Now we've looked at the function \( d[n] \) we can also give a sensible and valuable interpretation of our derived formula.

We can interpret our calculation of \( c \) as a COG like situation with a reduced outcome, and with an interesting property. We could view \( d[n] \) as a function changing the 'weight' \( p[n] \). The weight is almost unchanged for values near zero, but the weight is reduced for values further away from zero. That means we can view our function as the calculation of the centre of mass, in which mass that is located further from the origin is lighter than mass near the origin. That tends to concentrate the centre of mass towards the origin. This is perfect wanted behaviour! We thus have devised a function that does a centre of mass like calculation, but attracts that centre towards zero.
The proposed algorithm hasn’t been implemented to be tested in 'real-life' situations. However, simulations with a DAB encoder, a physical channel simulator and a DAB decoder have been performed by other members of the DAB team at Philips Advanced Development Centre.

This setup made it possible to perform real-time simulations of near real-world situations. With the channel simulator six delay paths could be simulated, together with Doppler effects and attenuation of paths. A noise generator injected a Gaussian additive white noise into the receiver when wanted.

Several time synchronisation algorithms were implemented in a digital signal processor in the receiver. One was the original algorithm as implemented in the DAB452 test receiver. That algorithm, we label it A, searched the first peak above a certain threshold.

The second algorithm, B, used a table per DAB mode to implement the COG algorithm with clustering of power located outside the 'guard interval'. This algorithm had a better performance than algorithm A.

The third algorithm, C, is similar to algorithm B, but now a piecewise linear version of the COG algorithm is used, linear around zero, linear with a lesser slope in regions at both sides of the region around zero, and a constant value for large values.

The results of the simulation could be watched on an oscilloscope. The results could also be noted by the audio quality and by the calculated bit-error rate at the receiver. Both algorithm B and C showed an increase in performance over algorithm A. Algorithm C showed a more relaxed behaviour of the window position on the oscilloscope, with a reduced 'jerky' behaviour.

The power shaping function worked pretty well; when some peaks 'left' the guard interval, they were 'pushed' back quickly.

The algorithm this author proposes — let’s call it D algorithm — looks like algorithm C, but has a smoother appearance. Expected is that it will work better than algorithm C.

More important, it has several very attractive properties that will be listed here.

- Less chip area is needed, for the ROM table can be discarded
Results

- It's easy to implement
- It's not fixed; changes can be made after the IC is ready
- Parameters can be used to change the behaviour of the algorithm
- Maximum window offset per frame can be set
- Speed/accuracy/sensitivity can be set
- Simple relation between parameters and algorithmic behaviour; can be approximated by a first order function
- Easy to program a microcontroller to use the algorithm
- Microcontroller can use other data to adaptively change the parameters for still better performance
- COG algorithm can be approximated by right choice of parameters

These properties together with the confidence in the overall working of the algorithm, has led the DAB chip design team to incorporate this algorithm in the specifications for the DAB V3 chipset, even only a couple of days before the specifications were frozen. This algorithm will therefore be used in all Digital Audio Broadcasting receivers that will based upon the Philips DAB V3 chipset.
Part IV

Conclusions and Recommendations
Conclusions

The conclusions will be presented in two parts; first conclusions concerning DSP Station, then conclusion about DAB fine time synchronisation.

12.1 DSP Station

About DSP Station the following conclusions can be made.

- DSP Station is not yet very suited for non-DSP like algorithms
- DSP Station is not yet suited for full hierarchical designs

12.2 DAB Fine Time Synchronisation

The conclusion about DAB fine time synchronisation are

- A good synchronisation algorithm has been succesfully developped
- This algorithm is part of the specification of the new DAB V3 chip-set
13

Recommendations

In this chapter we will give recommendations for the two parts; DSP Station and DAB fine time synchronising.

13.1 DSP Station

We'll here list recommendations concerning DSP Station.

- Give up-to-date on-line manuals with every (major) software update
- Improve speed of simulation cycle (maybe this is already done in a new release)
- Eliminate the small but annoying errors
- Add DFL debugging facilities
- Possibly change DFL to make real hierarchical designs possible
- Improve mixed design of sheets and (DFL) code

13.2 DAB Fine Time Synchronisation

The recommendations for DAB fine time synchronisation are:

- Perform field tests to verify quality of new synchronisation algorithm
- Maybe devise adaptive micro-controller algorithm
A.

DFL Syntax Description

A.1 DFL Syntax Description

\[<\text{DFL character}> \rightarrow \text{<alphabetic character> | <graphic character> | <decimal digit>}\]

\[<\text{alphabetic character}> \rightarrow 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | 'i' | 'j' | 'k' | 'l' | 'm' | 'n' | 'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' | 'v' | 'w' | 'x' | 'y' | 'z' | 'A' | 'B' | 'C' | 'D' | 'E' | 'F' | 'G' | 'H' | 'I' | 'J' | 'K' | 'L' | 'M' | 'N' | 'O' | 'P' | 'Q' | 'R' | 'S' | 'T' | 'U' | 'V' | 'W' | 'X' | 'Y' | 'Z'\]

\[<\text{graphic character}> \rightarrow '!' | ',' | ';' | ':' | ',' | \{'\} | '@' | ,01, | '-' | '+' | '=' | '[' | ']' | '\' | '"' | '@' \]

\[<\text{decimal digit}> \rightarrow '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'\]

\[<\text{binary digit}> \rightarrow '0' | '1'\]

\[<\text{ternary digit}> \rightarrow '0' | '1' | 'T'\]

\[<\text{octal digit}> \rightarrow '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7'\]

\[<\text{hexadecimal digit}> \rightarrow <\text{decimal digit}> | 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'A' | 'B' | 'C' | 'D' | 'E' | 'F'\]

\[<\text{identifier}> \rightarrow <\text{letter}> \{ <\text{letter}> | <\text{decimal digit}> \}\]

\[<\text{letter}> \rightarrow <\text{alphabetic character}> | _\]

\[<\text{quoted string}> \rightarrow "" \{ <\text{DFL character}> | "" \} ""\]

\[<\text{label}> \rightarrow <\text{identifier}>'::'\]

\[<\text{constant}> \rightarrow <\text{integer constant}> | <\text{fixed-point constant}> | <\text{floating-point constant}>\]

\[<\text{integer constant}> \rightarrow <\text{decimal digit}>\]

\[<\text{fixed-point constant}> \rightarrow <\text{integer constant}> \cdot <\text{integer constant}> | <\text{signed-digit constant}> | <\text{binary constant}> | <\text{octal constant}> | <\text{hexadecimal constant}>\]

\[<\text{signed-digit constant}> \rightarrow '0sd' <\text{ternary digit}>\]

97
expression -> '<right hand side>}' 'tixe'
<exit statement> -> 'exit' '(' <boolean expression> ')' 
<block> -> [label] 'begin' { <local clause> ';' } { <unit> ';' } 'end'
<local clause> -> 'local' <identifier> [';', <identifier>] <procedural if> 'if' <boolean expression> [label] 'then' <statement list> ['elseif' <boolean expression> [label] 'then' <statement list>] '*' [[label] 'else' <statement list>] 'fi'
<statement list> := { <local clause> ';' } { <unit> ';' }
<procedural for loop> -> [label] 'for' <procedural iteration clause> 'begin' <loop body> 'end'
<procedural iteration clause> -> <identifier> '=' <expression> 'to' <expression>
<loop body> -> { <unit> ';' } | { <unit> ';' } 'exit clause' ';' <unit> | <exit statement> '
<function definition> -> 'func' <identifier> '(' [{formal input argument part}] ')' [{formal output argument part}] '=' 'begin' { <unit> ';' } 'end'
<formal input argument part> -> <formal argument> ';' <formal argument>*
<formal argument> -> [ 'var' | 'signal' | 'state' ]<identifier> [';', <identifier>] <type designator> ['{' [range] ']' '][' '{' [period] ']' ']' | <formal input argument part>
<formal output argument part> -> , <type designator> ['{' [range] ']' '][' '{' [period] ']' ']' | <formal input argument part>
<range> -> <positive manifest integer expression> | <manifest integer expression> '..' | <manifest integer expression>
<period> -> <manifest integer expression>
<phase> -> <manifest integer expression>
<formal declaration> -> 'func' <identifier> '(' [{formal input argument part}] ')' <formal output argument part> '=' 'extern'
<extern Node> -> 'extern' <identifier> 'component' <name> { [extern Node options] }
<extern Node options> -> 'interface' <name> | 'model' <name> | 'parameters' <argument list>
<program> -> { <unit> ';' }
<unit> -> <signal definition> | <variable assignment> | <compound signal definition> | <function definition> | <function declaration> | <operator redefinition> | <extern Node> | <variable declaration> | <signal declaration> | <function call>
<operator redefinition> -> 'oper' <operator> '(' [<identifier> [';', <identifier>]] ')' '=' <expression>
A.2 Syntax Description for Component Info Blocks

<identifier> → <letter> { <letter> | <decimal digit> }^*
<quoted string> → '"' { <DFL character> | '"'}^ '"'
<idorstring> → <identifier> | <quoted string>
<info block> → '/**' { <statement> ';' }'*/'
<statement> → <mainstat> | <paramstat> | <aspectstat> | <includestat> | <modestat>
<mainstat> → 'main' <idorstring>
<paramlistentry> → <name idorstring> ':' <value idorstring>
<paramlist> → { <paramlistentry> }^*
<paramstat> → 'parameter' <idorstring> <paramlist>
<aspectlist> → { <idorstring> ',' }^+
<aspectstat> → 'aspect' <idorstring> { 'variable' | 'function' | 'operation' } <aspectlist
#includestat > → 'include' <idorstring>
<modestat> → 'mode' { 'applicative' | 'procedural' | 'fine_mixed' | 'coarse_mixed' }
Bibliography


