Universidade de Aveiro
Departamento de
Electrónica, Telecomunicações e Informática
2009

Lúcia Margarida
da Mata Antunes

# Software Defined Radio em FPGA

**Lúcia Margarida da Mata Antunes**

# Software Defined Radio em FPGA

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Electrónica e Telecomunicações, realizada sob a orientação científica do Doutor José Manuel Neto Vieira e do Doutor Arnaldo Silva Rodrigues de Oliveira, Professores do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

**o júri / the jury**

presidente / president

**Professor Doutor Nuno Miguel Gonçalves Borges de Carvalho**
Professor Associado da Universidade de Aveiro (por delegação da Reitora da Universidade de Aveiro)

vogais / examiners committee

**Professor Doutor José Carlos dos Santos Alves**
Professor Associado da Faculdade de Engenharia da Universidade do Porto (Arguente Principal)

**Professor Doutor José Manuel Neto Vieira**
Professor Auxiliar da Universidade de Aveiro (Orientador)

**Professor Doutor Arnaldo Silva Rodrigues de Oliveira**
Professor Auxiliar Convidado da Universidade de Aveiro (Co-orientador)

**agradecimentos / acknowledgements**

This dissertation would not have been possible without the help and support of so many people.

First of all, I would like to thank my coordinators for their help, guidance and sometimes patience during this last year.

Dr. Manuel Violas and *Instituto de Telecomunicações* for providing me the opportunity to develop my work in their facilities and with their XtremeDSP Development Kit-IV platform.

Daniel Albuquerque for having helped me to take the first step in the OFDM theory and implementation, as well as for having taught me how to work with latex.

My present and past roommates, but above all my friends! With them I passed a lot of memorable moments that I will never forget!

All my friends that I have met in the *University*, for their support on the good and bad times. Without them, these last years would not have been the best ever.

My friends from *Abrantes* for all weekends and summers that we spent together. Thank you for having been always ready for me, whether to have fun, to talk or just to relax.

And the most important: my Family! To have always supported me, listened me, encouraged me, helped me, advised me,..., but above all: loved me! *Obrigado por tudo! Sem vocês não teria sido possível chegar aqui!*

**Resumo**

Esta dissertação teve como objectivo o desenvolvimento de parte de um receptor para *Digital Audio Broadcasting* (DAB) recorrendo aos conceitos ditados por *Software Defined Radio* (SDR). O receptor de rádio inclui a conversão de digital para analógico e a subsequente desmodelação de banda-base, pelo que é possível aceder à *bit stream* em qualquer ponto do sistema.

A dissertação foi dividida em duas fases. Na primeira, o receptor completo foi simulado em MATLAB. Na segunda, o mesmo sistema foi implementado e testado numa placa *XtremeDSP Development Kit-IV*, a qual contém um *Field-Programmable Gate Array* (FPGA).

O sistema simulado foi testado com dois tipos de amostras. As primeiras consistiram em sinais DAB gerados em MATLAB e posteriormente distorcidos por diferentes canais também simulados pelo mesmo software. Foi assim possível fazer um estudo da probabilidade de erro quando o sinal é exposto a diferentes perturbações, como ruído, desvios na frequência e no tempo. O sistema foi ainda testado com amostras DAB reais. As constelações desmodelados mostraram o correcto funcionamento do sistema.

Apenas parte do receptor simulado foi implementado no FPGA. A parte já desenvolvida consiste nas funções de desmodelação: desmodelação OFDM, desmodelação diferencial, *frequency deinterleaving* e demapeamento QPSK. O sistema de sincronização DAB não foi implementado. O sistema já desenvolvido é assim capaz de desmodelar um sinal DAB gerado no MATLAB, desde que este não contenha qualquer distorção.

**Abstract**

The aim of this dissertation was the development of part of a *Digital Audio Broadcasting* (DAB) receiver by means of *Software Defined Radio* (SDR). This radio receiver includes the Intermediate Frequency (IF) to baseband conversion and the subsequent baseband demodulation, thus one may access the *bit stream* in any point of the system.

This dissertation was divided in two phases. In the first one, the whole DAB system was simulated in MATLAB. In the second, the receiver was implemented and tested in an *XtremeDSP Development Kit-IV* platform, which includes a *Field-Programmable Gate Array* (FPGA).

The simulated system was tested with two kinds of samples. The first ones were generated in MATLAB and subsequently distorted by different channel conditions also simulated in the same software. This well known DAB digital signal allowed us to perform a Bit Error Rate (BER) study with several channel conditions, such as noise, multipath, frequency and time offsets. Further on, real DAB samples were used for testing. The demodulated QPSK constellations showed the correct operation of the system.

Only part of the simulated receiver was implemented in the FPGA. This part consists in the channel demodulation functions: OFDM demodulation, differential demodulation, frequency deinterleaving and QPSK demapper. The DAB synchronization block was not implemented. The developed system is able to recover the modulated bit stream from the digital signal produced in MATLAB, since this signal is free of noise, frequency and time offsets.

# Contents

# List of Figures

# List of Tables

# Acronyms

**ADC** Digital-to-Analog Converter.

**AGC** Automatic Gain Control.

**AM** Amplitude Modulation.

**ASIC** Application-Specific Integrated Circuit.

**AWGN** Additive White Gaussian Noise.

**BER** Bit Error Ratio.

**CA** Conditional Access.

**CIF** Common Interleaved Frame.

**CPU** Central Processing Unit.

**CRC** Cyclic Redundancy Check.

**CU** Capacity Unit.

**DAB** Digital Audio Broadcasting.

**DAB+** Digital Audio Broadcasting plus.

**DAC** Analog-to-Digital Converter.

**DFT** Discrete Fourier Transform.

**DMB** Digital Multimedia Broadcasting.

**DQPSK** Differential Quadrature Phase-Shift Keying.

**DSP** Digital Signal Processor.

**DTV** Digital Television.

**Eb/No** Symbol Energy to Noise Ratio.

**EEP** Equal Error Protection.

**Es/No** Symbol Energy to Noise Ratio.

**ETSI** European Telecommunications Standards Institute.

**FDM** Frequency Division Multiplexing.

**FFT** Fast Fourier Transform.

**FIB** Fast Information Block.

**FIC** Fast Information Channel.

**FIDC** Fast Information Data Channel.

**FIFO** First-In/First-Out.

**FIR** Finite Impulse Response.

**FM** Frequency Modulation.

**FPGA** Field Programmable Gate Array.

**GPP** General Purpose Processor.

**I** In-phase.

**I/O** Input/Output.

**IC** Integrated Circuit.

**ICI** Intercarrier Interference.

**IDFT** Inverse Discrete Fourier Transform.

**IEEE** Institute of Electrical and Electronic Engineers.

**IF** Intermediate Frequency.

**IFFT** Inverse Fast Fourier Transform.

**IIR** Infinite Impulse Response.

**ISI** Intersymbol Interference.

**ITU** International Telecommunication Union.

**LFSR** Linear Feedback Shift Register.

**LNA** Low Noise Amplifier.

**LPF** Low Pass Filter.

**MCI** Multiplex Configuration Information.

**MMIC** Monolithic Microwave Integrated Circuit.

**MP2** MPEG Audio Layer II.

**MSC** Main Service Channel.

**NCO** Numerical Controlled Oscillator.

**OFDM** Orthogonal Frequency Division Multiplexing.

**P/S** Parallel/Serial.

**PRBS** Pseudo Random Binary Sequence.

**PRS** Phase Reference Symbol.

**Q** Quadrature.

**QPSK** Quadrature Phase-Shift Keying.

**RAM** Random Access Memory.

**RF** Radio Frequency.

**ROM** Read Only Memory.

**RTP** Rádio e Televisão de Portugal.

**S/P** Serial/Parallel.

**SDR** Software Defined Radio.

**SI** Service Information.

**SiGe** Silicon Germanium.

**SNR** Signal to Noise Ratio.

**SoC** System-on-Chip.

**TII** Transmitter Identification Information.

**UEP** Unequal Error Protection.

**VCO** Voltage-Controlled Oscillator.

# Chapter 1

# Introduction

## 1.1 Overview

The concept of *Software Defined Radio (SDR)* has been acquiring a great deal of attention in the past several years. Created in 1991 by Joseph Mitola, it is *"a radio whose channel modulation wave-forms are defined in software"* [III00]. This will allow users to operate the radio in different environments and applications. This system is characterized by its flexibility and reconfigurability, since changing its behaviour only requires modifying or replacing the software [CLA+09]. Due to actual technology progress limits, the ideal concept is not completely achievable since it is still necessary an analog front-end to downconvert the Radio Frequency (RF) signal to an Intermediate Frequency (IF). The resulting signal can be sampled by an Digital-to-Analog Converter (ADC) and finally be fully processed using digital techniques.

Since our major goal is to study the SDR concept and its implementation related problems, we have decided to develop a receiver for the radio transmission standard *Digital Audio Broadcasting (DAB)* using that concept.

DAB is a digital radio transmission standard developed by the European Project *Eureka 147*. Unlike the traditional Frequency Modulation (FM) radio, DAB permits to tune up to 10 radio stations within the same frequency and in any place of the country. Some of the DAB advantages include easy program selection, improved reception, program-associated data, information services and the most important "one receiver does it all". *Orthogonal Frequency Division Multiplexing (OFDM)* is one of the main features of this system. It divides a high data rate stream in $K$ parallel lower data rate streams, which will modulate $K$ different subcarriers. Hence, the symbol duration will be increased by a factor of $K$ and the system will be more robust against multipath and noise. DAB has already regular services in 30 countries, more than 990 DAB receivers are commercially available and more than 12 million have already been sold all around the world. DAB is seen as the future of the radio!

## 1.2 Objectives

The objective of this dissertation was to develop part of a DAB receiver by means of SDR using an XtremeDSP Development Kit-IV, which includes a *Field Programmable Gate Array (FPGA)*. The receiver should comprise the physical layer related functions, i.e., the downconversion from RF to IF, the time and frequency synchronization mechanisms and the baseband signal demodulation functions.

The receiver development was thought to be done in two parts. In the first one, the whole DAB system might be simulated in MATLAB. In the second, the receiver should be implemented and tested in the FPGA. The final system shall be able to recover the modulated bit stream from the digital signal produced in MATLAB.

## 1.3 Contributions

The complete physical layer of a DAB receiver was simulated in MATLAB. The final system is able to synchronize and subsequently demodulate a real DAB baseband signal. The synchronization block performs frame time synchronization, as well as estimates and corrects possible carrier or sampling frequency offsets. The remaining system demodulates the DAB signal in order to recover the original bit stream. The following demodulation functions are performed: OFDM demodulation, differential demodulation, frequency deinterleaving and Quadrature Phase-Shift Keying (QPSK) demapper.

The signal demodulation functions of the DAB receiver were also implemented in an FPGA using the Xilinx System Generator tool. The developed system is able to recover the modulated bit stream from the digital signal produced in MATLAB, since this signal is free of noise, frequency and time offsets.

To my contributions I would like to add the very helpful contribution of others to my work.

Thus, I will start with Andreas Muller. While searching on the Internet, I found his semester thesis dissertation called *"DAB Software Receiver Implementation"* [Mul08]. Its main goal was to develop a complete real-time software receiver for DAB using GNU Radio, an open-source toolbox to learn about, build and develop SDR systems. After contacting him, he kindly provided me several real DAB samples and the contact of Nicolas Alt, who has implemented a complete DAB receiver in MATLAB. Moreover, Andreas' semester thesis [Mul08] provided a lot of helpful information.

Nicolas Alt has also gently shared his receiver with me, as well as some DAB samples. Although none of the code and used techniques of his receiver implementation were used in this dissertation, they were an essential starting point to my DAB synchronization implementation. On the other hand, his DAB samples were intensively used through this dissertation. Furthermore, having a working software model and "known to work samples" was very useful to verify each step of my implementation. Additionally, the idea of using the distance between two Null symbols to estimate the sampling frequency was adopted from Nicolas' code.

The OFDM knowledge of Daniel Albuquerque was also fundamental for developing this master thesis dissertation. He kindly discussed with me some important DAB implementation issues and provided several MATLAB functions, such as OFDM implementation examples and a channel simulator. These were intensively used in this dissertation's MATLAB implementation.

Paulo Ferreira, who is responsible for the DAB ensemble in Rádio e Televisão de Portugal (RTP), has also provided me very useful information about this digital radio transmission in Portugal.

Finally, Afonso Silva has also nicely given me his Xilinx WebCase access and Xilinx System Generator projects [Sil08]. Although the former one was not very helpful, the later ones were a very good starting point for my FPGA receiver implementation. My co-coordinator kindly helped me with the first problem.

## 1.4 Outline

The dissertation is organized through five chapters.

**Introduction** presents the project idea, objectives and contributions.

**Software Defined Radio** explains the concept of SDR, its related problems and possible architectures.

**Digital Audio Broadcasting** gives an overview of the DAB standard, its features and transmitter architecture, as well as explains the OFDM modulation technique and its main concerns.

**Receiver Simulation in MATLAB** provides a detailed explanation of the DAB receiver, its MATLAB implementation and achieved results.

**Receiver Implementation in FPGA Technology** describes the implementation of the same DAB receiver in the XtremeDSP Kit-IV development platform and presents the achieved results.

**Conclusions and Future Work** reviews the dissertation, draws some conclusions and, finally, explains what can still be done in a future work.

# Chapter 2

# Software Defined Radio

## 2.1 Introduction

During the last two decades, the fast advances in the semiconductor field made possible the emergence of a new concept called *Software Defined Radio* (SDR). SDR was first introduced by Joseph Mitola in 1991, which he described as:

*"A software radio is a radio whose channel modulation waveforms are defined in software. That is, waveforms are generated as sampled digital signals, converted from digital to analog via a wideband Analog-to-Digital Converter (DAC) and then possibly upconverted from IF to RF. The receiver, similarly, employs a wideband ADC that captures all of the channels of the software radio node. The receiver then extracts, downconverts and demodulates the channel waveform using software on a general purpose processor."* [Mit95]

Basically, SDR is a radio that uses as much digital technology as possible. Typical components that were usually implemented using analog hardware, such as amplifiers, filters, mixers, modulators/demodulators and detectors (among others), are now implemented using software in a Digital Signal Processor (DSP). This system is characterized by its flexibility and adaptability, due to the fact that changing its behavior only requires modifying or replacing the software.

A typical radio transmitter or receiver has a dedicated analog circuit for filtering, tuning and modulating/demodulating. If one needs to use two or more standards, the radio would need a dedicated circuit for each of these. As one can imagine it would not be feasible in terms of size, power consumption and cooler system. SDR can be the solution for this problem, since a single radio can support different standards just by changing the transmission/reception software on the system without the need of dedicated circuits. Ideally, the receiver samples the signal just after the antenna and the remaining radio signal processing would be done by a DSP, as represented in Figure 2.1. This "ideal" SDR would be able to demodulate a signal with any carrier frequency (carrier flexibility), bandwidth (wideband) and kind of modulation (modulation flexibility). As will be explained in the next subsections, the state of the art in what concerns both analog and digital domains does not allow this simplified version of a SDR. In order to overcome this problem, several solutions have been proposed to get partial solutions to the SDR concept.

One early implementation of SDR systems was the United States military project called

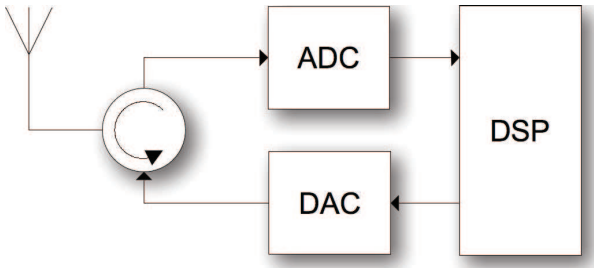


Figure 2.1: *The "ideal" SDR architecture. [Alb09]*

SpeakEASY that was started in 1991. SpeakEasy handled different modulations techniques and frequencies to make possible to communicate with more than 10 different types of military radio standards. SpeakEASY allowed digital frequency conversion and wideband signal processing, and showed that modular radio components, such as analog front-ends, DACs, ADCs and DSPs, can be combined in an open architecture bus. This open-architecture approach is able to increase production volume and reduce costs [Mor00]. Since then we have witnessed an explosion of interest in SDR, which is currently being developed for both communication and broadcasting applications, such as second-generation cellular phones, third generation CDMA cellular phones, the Digital Televisions (DTVs) and DAB.

SDR is seen as the future of the wireless communications. Since it will use public radio wave spectrum, it is highly necessary to create standards and rules about SDRs specifications. A group called SDR Forum, working in collaboration with the Institute of Electrical and Electronic Engineers (IEEE), has worked to establish the definition of SDR that will provide consistency and a clear overview of the technology and its associated benefits [For09a].

## 2.2 Software Defined Radio Configurations

One of the main steps of a SDR system is digitalization, which is the process of converting an analog signal into a digital sequence. This process is done in accordance with the sampling theorem, which states:

*"If the highest frequency contained in an anolog signal $x_a(t)$ is $F_{max} = B$ and the signal is sampled at a rate $F_s > 2F_{max} \equiv 2B$, then $x_a(t)$ can be exactly recovered from its sample values."* [PM06]

The sampling rate $F_N = 2F_{max}$ is called the *Nyquist rate*.

Ideally, the digitization would take place right after the antenna, since it allows processing the signal fully in software. However, this SDR configuration is currently impracticable due to the state-of-the-art of ADCs and DACs and the limitations on computational capacity of contemporary processors. In order to overcome this problem, several solutions have been proposed to get partial solutions to the SDR paradigm. These have increased the flexibility of the radios and many of them are already in use.

Considering the frequency in which the digitalization takes place, three different types of SDR configurations can be defined: Baseband digitalization, IF digitalization and RF digitalization. The following subsections will take a quick look at each of these configurations.

For better readability, only the receiver side is discussed. The transmitter side is based on the same configuration, but in the reverse order.

### 2.2.1 Baseband Digitalization

Digitization at baseband level is the most common used in nowadays radios. The simplified baseband radio configuration is illustrated in Figure 2.2.

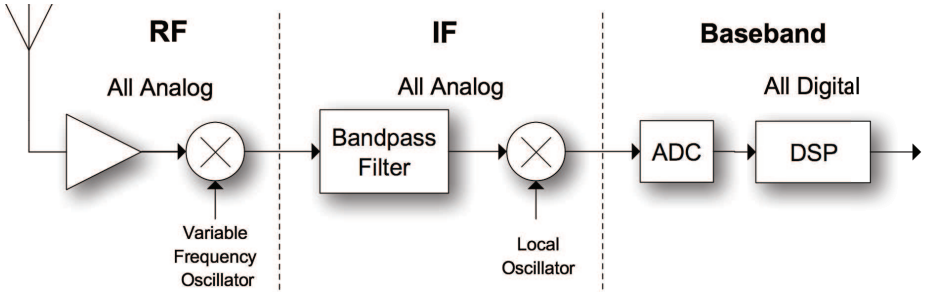


Figure 2.2: *Baseband digitalization architecture.*

The analog front-end downconverts the RF signal to an IF and, subsequently, to the baseband. Thus, the analog front-end will have at least one stage for each band frequency transition. Following the chain, the signal will be sampled at the ADC and the remaining radio functions will be implemented in a programmable processing technology. Thus, the signal modulation, channel and source decoding functions will be completely performed through digital means.

This SDR configuration should not be confused with a common analog radio that also samples the signal at the baseband. In that case, the *bitream* is directly extracted from the analog signal and fed into subsequent digital stages to profit from digital signal processing techniques such as music equalization. This is a common practice in widely used devices such as stereo music equipment. Since none of the demodulation functions is carried out in software, these radios are not considered to be a SDR [Man07].

### 2.2.2 IF Digitalization

Thanks to recent advancement of semiconductor technology, the SDR has being moved toward the antenna. Thus, the most emergent architecture implemented in SDRs is the IF digitization. This architecture is depicted in Figure 2.3 and consists of an antenna, an analog front-end, an IF stage and software to perform all the baseband computation.

The analog front-end will downconvert the RF signal to a frequency which can be processed by the ADC. It is comprised by, among others, a Low Noise Amplifier (LNA), a filter and a mixer. Following the chain, the signal will be sampled at the ADC and the remaining radio functions will be implemented through reconfigurable software operating on programmable processing technologies. These devices include DSPs, FPGAs, General Purpose Processors (GPPs), System-on-Chip (SoC) or other application specific programmable processors [For09a].

Since part of the frequency translation process takes place in the digital domain, the number of received channels can be much bigger than in the previous architecture, making this architecture more flexible and, therefore, desirable.

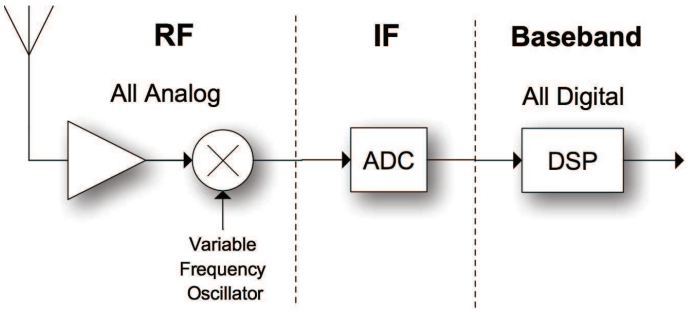


Figure 2.3: *IF digitalization architecture.*

### 2.2.3 RF Digitalization

As already mentioned, an "ideal" SDR system would sample the RF signal directly from the wideband antenna. The remaining signal processing would be done by digital means, including the band pass filtering, automatic gain control, frequency translation, low-pass filtering, and demodulation of the desired signal [Mor00]. The simplified block diagram of this configuration can be seen in Figure 2.4. With this configuration, the radio system would be

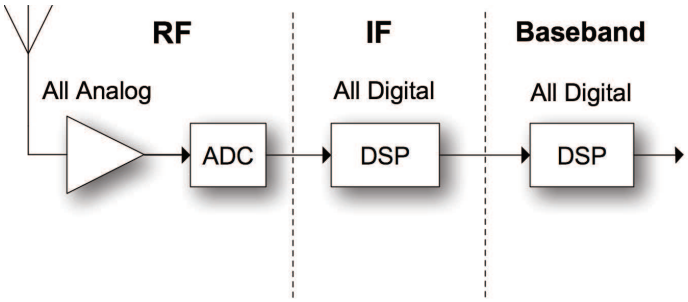


Figure 2.4: *RF digitalization architecture.*

able to process the whole spectrum and support any standard just by changing its software. Although this configuration is the most desirable, the actual ADCs and DACs capacities make it unfeasible. Nevertheless, significant research efforts are being taken to surmount this problem.

The several configurations standardized by SDR Forum can be found in [Gro02].

## 2.3 Key Components of Software Defined Radio

While the concept of SDR is not new, there are several factors that have enabled its large development over the last few years. First, the emerging standards require a lower dynamic range than some of earlier standards because of coding and processing gain. This made possible some "relaxations" in terms of overall performance. Second, and more important is the continued and rapid advancement of semiconductor processes and systems [BB08]. The end result is the availability of key components that enable SDR technology, which provides a flexible, robust and less expensive solution when compared with traditional analog radios.

### 2.3.1 Analog Front-Ends

The analog front-end is one of the most critical parts of the radio system. It has a main role in the overall system performance, power consumption and size, and it is where the receiver can, within limits, be designed for the best Signal to Noise Ratio (SNR) [Bow08].

Thanks to recent advances of semiconductor technology, it is now possible to incorporate most if not all of the analog radio functions such as filters, amplifiers and mixers, in a single chip. Monolithic Microwave Integrated Circuit (MMIC) is the technology used for integrating RF components in one chip [BB08]. It is a type of Integrated Circuit (IC) device that operates at microwaves frequencies (300 MHz to 300 GHz) and it may be based on widely different semiconductor processes, such as conventional silicon CMOS and advanced Silicon Germanium (SiGe) technologies. Many efforts are being made to reduce the size and improve the performance of these circuits, with a key focus on lowering the noise and improving the linearity [Bow08].

Next, we will take a quick look at the analog front-end configurations that are usual used in digital telecommunications, the homodyne and superheterodyne front-ends. It will be also presented the multiband front-end configuration already used by several SDR systems. Normally, the transmitter design challenges are very similar to those of the receiver. Hence, the following subsections will only present the receiver side of the different architectures.

#### Superheterodyne Front-End

The superheterodyne receiver is divided in two main stages, an IF stage and a subsequent baseband stage. Its simplified configuration is depicted in Figure 2.5.

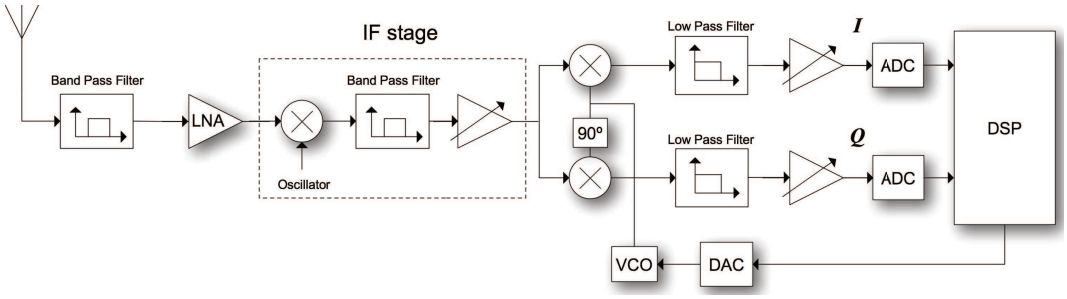


Figure 2.5: *Superheterodyne front-end architecture.*

First, the RF signal is downconverted to an IF frequency. Subsequently, the signal is filtered and amplified in order to select the desired channel. In the next stage, the signal is downconverted toward the baseband and amplified by an Automatic Gain Control (AGC) to fit the ADCs dynamic range. However, this configuration has some disadvantages. Since it has two stages, more components are needed and, therefore, an increased power consumption. Moreover, the superheterodyne must be designed according to a specific standard, which do not make this architecture suitable for wideband receivers as an SDR system. [Alb09]

To overcome this problem, a superheterodyne architecture derivation was proposed. Its simplified configurations can be seen in Figure 2.6. As one can see through the figure, instead of place the ADC at the output of the second downconverter stage, it is placed after the first down-converter stage. Since the quadrature downconversion is performed on the digital domain, this approach simplifies the hardware receiver design and, mainly, can deal with

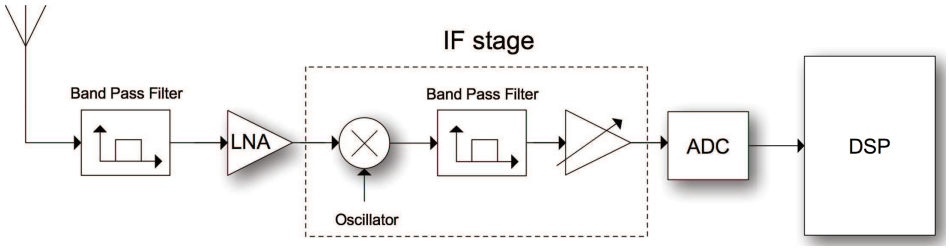


Figure 2.6: *Superheterodyne front-end derivation architecture.*

wideband signals. This superheterodyne derivation is one of the simplest front-ends that can be built for use in SDR. [Alb09]

### Homodyne Front-End

The simplified configuration of a homodyne receiver is depicted in Figure 2.7.

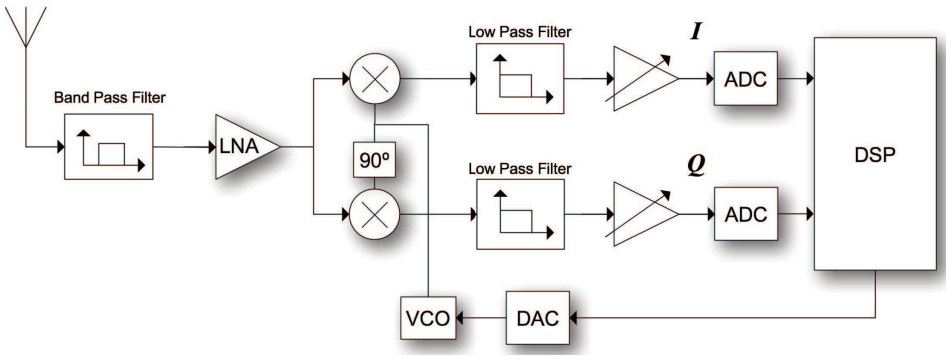


Figure 2.7: *Homodyne front-end architecture.*

In the homodyne receiver, the signal is firstly filtered and amplified in order to choose a specific channel. Subsequently, the signal is directly downconverted to the baseband by a Voltage-Controlled Oscillator (VCO). The resulting signal is filtered and amplified by an AGC to fit the signal to the ADCs dynamic range. This approach is very efficient because it only needs a few components when compared to the superheterodyne architecture. Although the homodyne architecture is not flexible, it can receive wideband signals. Thus, this front-end architecture is suitable for SDR systems. [Alb09]

### Multiband Front-End

One of the main aims of a SDR receiver is to be a multiband system, thus a high bandwidth ADC is highly necessary. Since these components are not yet available, several solutions were proposed to overcome this problem. One of those architectures is based on several front-ends connected to the same DSP, each one receiving a different frequency band. Its simplified configuration is illustrated in Figure 2.8.

Although this system approach be fixed to the implemented front-ends, it is possible to add a new one if necessary. Thus, it is inflexible, but it has also some level of scalability.

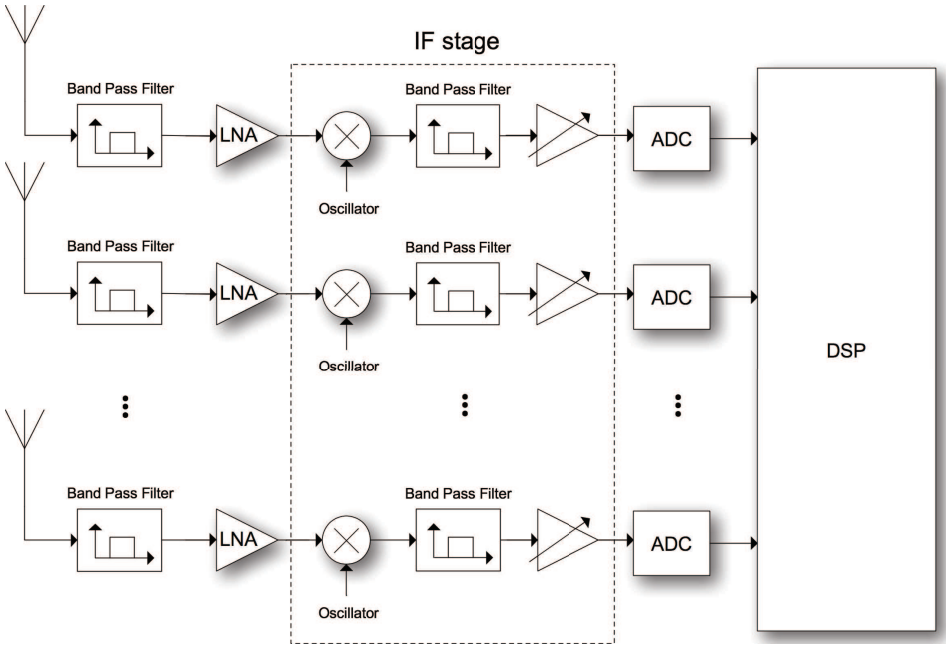


Figure 2.8: *Multiband front-end architecture.*

Moreover, this architecture can be simplified if the system does not have to receive all the bands at the same time. [Alb09]

### 2.3.2 Analog-to-Digital and Digital-to-Analog Converters

ADCs and DACs have a fundamental role in the evolution of SDR. The continued progress in this area made possible the signal conversion at higher and higher analog input frequencies, which allowed SDRs to move to the IF frequency.

The major parameters that define the performance of ADCs and DACs are the dynamic range and sampling rate. While dynamic range has not improved as in other technologies, a focused effort on increasing performance at higher frequencies has finally enable us to perform the signal digitalization at an IF frequency. Hence, while bit precision has only improved two bits (12 to 14 bits) over the last 5 years, the linearity has significantly improved by 40 dB (60 to 100 dBc for harmonics) for IF sampling [BB08].

Speed and power consumption are also an important tradeoff in what concern ADCs and DACs. If an ADC is fast, its power consumption is higher than a slower one. This is a very important issue in nowadays radios, mainly in mobile devices where cooling systems cannot be installed and the battery life is an extremely limiting factor. Currently, there are two trends among ADCs' researchers, some focus on increasing speeds and others on reducing power consumption [Man07].

### 2.3.3 Digital Signal Processing Circuits

After signal digitalization, the remaining radio functions in the signal chain must be processed digitally. Hence, the digitalized IF signal must be firstly downconverted, filtered and decimated. Next, the same or another digital signal processor performs the slower speed

signal processing such as channel and source decoding. The transmitter side is similar, but follows the reverse path.

Signal processing of high speed and wideband signals requires a very powerful and high-speed signal processing circuit. Thanks to the continuing reduction in feature size, more functions can be packaged into a single die while at the same time the total cost of those chips is also reduced. This technology has been notably used to increase the processing power of general purpose DSPs, but it has also been used to create cost effective high speed devices such as FPGAs and Application-Specific Integrated Circuits (ASICs), all key components used within a software defined radio system [BB08]. A DSP chip does signal processing by fetching instructions and data from memory, performing operations and storing the results back to memory [Mor00]. It has a higher speed than a regular Central Processing Unit (CPU), more resources and provides also some degree of parallelism. An ASIC is an integrated circuit that is designed to perform a fixed specific task such as digital downconversion and digital filtering. An ASIC has a better performance and higher speed then a DSP, however the user cannot change its functionalities. The FPGA comes to bridge the gap between DSPs and ASICs. One can perform any task in an FPGA just by mapping it to the hardware wherever needed. Thus, it provides the flexibility and complexity of an ASIC, but with the shorter turn-around time and reconfigurability of a programmable device.

Since reconfigurability is one of the most important features of a SDR, the key components to perform the signal processing tasks are DSPs and FPGAs. Usually, the FPGA performs the radio functions, which follow the ADC such as downconverting, filtering and decimating. The DSP comes after in the radio chain to perform the slower speed signal processing such as channel and source decoding.

### 2.3.4 SDR Platforms

Summarizing, the radio receiver chain starts with the antenna and is followed by an analog front-end where the signal is downconverted toward an IF frequency or the baseband. If we are in the presence of an IF configuration, the signal must be fed into an FPGA that downconverts the signal to the baseband. The same FPGA or a different DSP comes next to perform the slower speed signal processing such as signal demodulation, channel and source decoding. Moreover, an alternative DSP can be used to perform the radio synchronization functions, since they are independent from the rest. Figure 2.9 illustrates a typical SDR signal chain.

From the previous subsections, one could see that the development of high-performance SDR systems can present significant challenges in both design complexity and time to market. Thus, specific platforms are available in order to provide complete development environments and to enable faster and easier designs. Depending on the platform, all or some of the previous SDR key components are provided. The following platforms are some commercially available examples:

***LYRtech Virtex-4 FPGA SDR Development Platform*** - two Xilinx Virtex-4 FPGAs, TI DSP, two ADCs @ 125 MS/s (14 bits), two DACs @ 500 MS/s (16 bits) and a RF module.

***Texas SFF SDR Development Platform*** - Xilix Virtex-4 FPGA, 2 TI DSPs, two ADCs @ 125 MS/s (14 bits), two DACs @ 500 MS/s (16 bits). Modular architecture, one

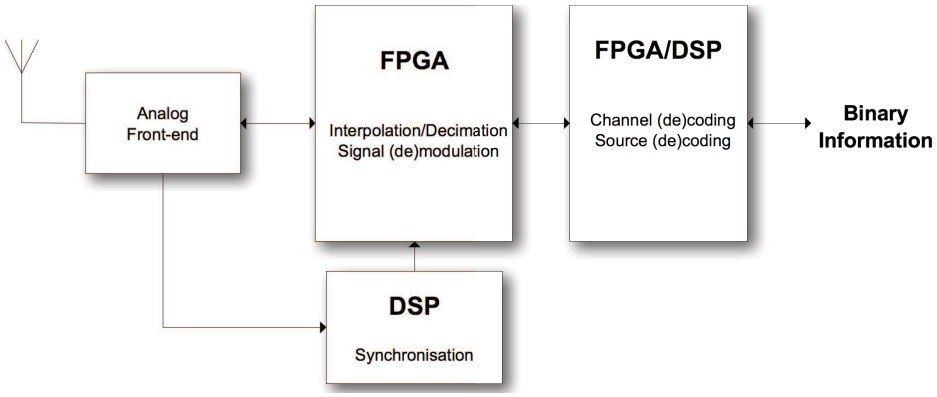


Figure 2.9: *Typical SDR signal chain.*

RF module and an optional second RF module for full-duplex operation or to cover additional bands.

***Ettus Research Universal Software Radio Peripheral (USRP)*** - Altera FPGA, gigabit Ethernet interface, two ADCs @ 64 MS/s (12 bits), two DACs @ 128 MS/s (14 bits), high-speed USB 2.0 controller and several RF module options. The USRP has an open design with freely available schematics, drivers and free software to integrate with GNU Radio. GNU Radio is a free software toolkit for learning about, building, and deploying SDR systems. The USRP is developed by a team led by Matt Ettus.

***Nallatech Virtex-4 XtremeDSP Development Kit Platform*** - XilinxVirtex-4 FPGA, Spartan-II FPGA, two ADCs @ 105 MS/s (14 bits), two DACs @ 160 MS/s (14 bits), two banks of ZBT-SRAM and status LEDs. Support for external clock, on board oscillator, and programmable clocks.

These are only some examples of the many options available on the market. Some of them provide a FPGA, a DSP and a RF modulo, while others only provide one or two of these components (among others). A SDR developer should choose the one that most fits his SDR system. The *Virtex-4 XtremeDSP Development Kit* was the platform used to develop our DAB receiver

### 2.3.5 Cognitive Radio

The concept of SDR brought up a new idea, the *Cognitive Radio* (CR). Also thought by Mitola [IM99, III00], it is essentially a SDR that senses its environment, tracks changes, and reacts according to its findings [CLA+09]. Thus, a CR should be able to detect holes on the frequency spectrum (opportunistic), estimate channel conditions and adapt its modulation parameters based on it, as well as transmit power controlled signals to reduce interference among systems. CRs should have a wideband front-end in order to sense the frequency spectrum and communicate with large bandwidth waveforms.

CRs are widely regarded as being the next step in the evolution of radio system, since it can adapt itself to every situation, replacing the traditional single function radios. Figure 2.10 illustrates a rough time evolution of the radio technology.

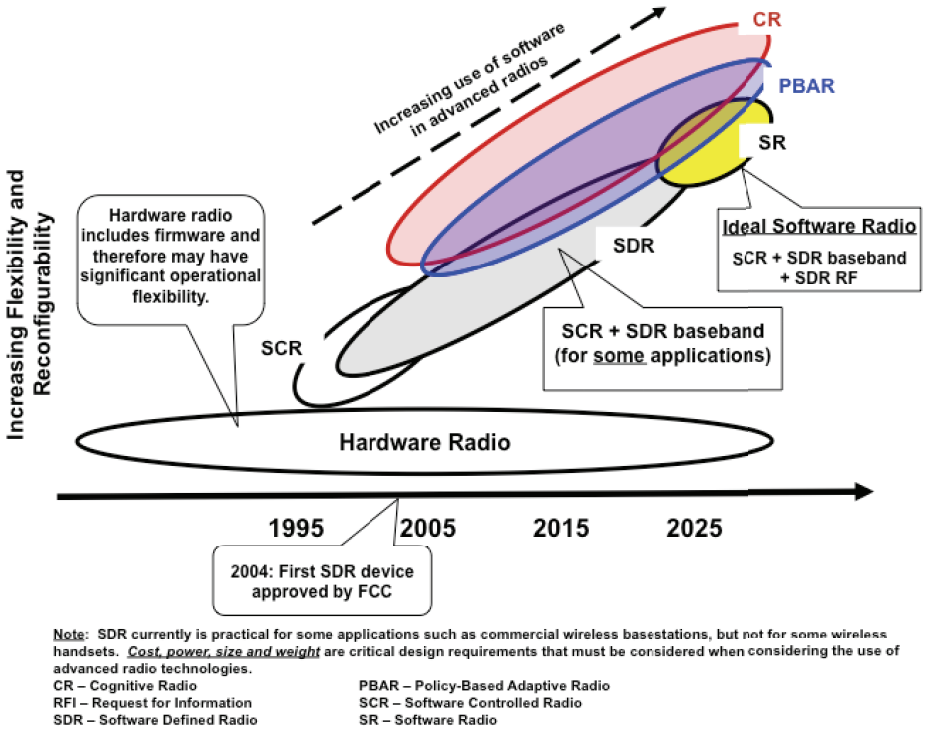


Figure 2.10: *Time evolution of the radio technology. [Gue07]*

---

Through Figure 2.10, one can observe that radio technology is moving toward the digital domain. Nowadays, some SDR approaches are already in use, but its ideal concept was not reached yet. Moreover, CRs are from all radio technologies the most flexible one.

# Chapter 3

# Digital Audio Broadcasting

## 3.1 Introduction

*Digital Audio Broadcasting* (DAB) is a digital radio broadcasting system developed within the *Eureka 147* project in the 1980s. The protocol specification was finalized in 1993 and it was recognized as a world standard by International Telecommunication Union (ITU) in 1995 and by European Telecommunications Standards Institute (ETSI) in 1997 [Eur06].

Unlike the traditional analog radio, DAB combines multiple services onto a single broadcast frequency through the application of multiplexing and compression techniques. Besides the indispensable audio channels, it can also transmit programme-associated data such as song titles, lyrics, weather maps, traffic information and general advertisements, as well as information services from other sources which include news headlines, weather information or even the latest stock prices [For09b].

Since DAB was originally designed for terrestrial mobile reception it is able to eliminate the multipath interferences that commonly disturb analog systems (e.g. Amplitude Modulation (AM) and FM) while on move. This feature could not be possible without the so-called technique OFDM. This technique distributes the information to be transmitted over a great number of orthogonal carriers. Since selective frequency interferences only affect some of those carriers, with additional information redundancy and digital error correction techniques it is possible to reconstruct the sequence of bits in the receiver. For this reason, a driver is able to cross an entire country staying tuned to the same station with no signal fade and without the need of changing the frequency [For09b].

However, it has been shown that the audio quality on DAB systems is sometimes lower than on FM systems for stationary receivers due to the use of lower bit rates in order to increase the number of channels within the same DAB ensemble. To solve this problem, an upgraded version of the system was realized in February 2007. Digital Audio Broadcasting plus (DAB+) is based on the original DAB standard and provides the same services, but uses a more efficient audio codec. The MPEG Audio Layer II (MP2) was replaced by the HE-ACC v2 (AAC+), which allows an equivalent or better subjective audio quality while broadcasting at lower bit rates [For09b].

The *Eureka 147* Family of standards, which includes DAB, DAB+ and Digital Multimedia Broadcasting (DMB), has experienced a great progress throughout the past few years. Its key area of development is Europe, but also China, Korea and the majority of the Asia Pacific Region including Australia [For08]. Nowadays, DAB has regular services in more than 30

countries, more than 990 DAB receivers are commercially available and more than 12 million have already been sold all around the world.

In Portugal, the DAB pilot broadcasts started in January of 1998 and several initiatives promoting DAB were developed during Expo 98. *RTP* won the license for installing and operating the national DAB ensemble in March of 1999. These radio transmissions operate on the 12B channel at the frequency of 225.648 MHz. RTP is also the provider of the five audio channels currently transmitted within the DAB ensemble, which are *Antena 1*, *Antena 2*, *Antena 3*, *RDP África* and *RDP Internacional*. According to Paulo Ferreira, who is working on RTP, the five audio channels are being transmitted in stereo with a bit rate of 192 kbit/s. Currently, there are 42 transmitters that cover more than 75% of the population, mainly in the coastline [For08].

The *Eureka 147* Family of standards has the potential to replace the existing AM and FM radio broadcasting systems in a near future, some say that it is the future of the radio!

## 3.2 The DAB System

The DAB system can be divided in three parts. Two of them are related with the main transport mechanisms, the Main Service Channel (MSC) and the Fast Information Channel (FIC). Starting with the MSC, each service signal is encoded individually at source level, error protected and time interleaved in the channel encoder. Then the services are multiplexed in the MSC according to a pre-determined multiplex configuration. In the FIC side, the multiplex control and service information are also encoded and error protected. Finally, the MSC is combined with the FIC to form the transmission frame. The third part comes in the transmission chain just after the previous two. This is responsible for the physical layer, which includes frequency interleaving, Differential Quadrature Phase-Shift Keying (DQPSK) and OFDM modulations. At the end, the signal is converted to the radio frequency band, amplified and transmitted [For09b].

The conceptual block diagram of a DAB transmitter is given in Figure 3.1. The complete block diagram that is provided in [Eur06] can be found in Appendix A.

The next subsections shall describe each part of the DAB transmitter, as well as its transport mechanism and transmission modes. The receiver is similar to the transmitter, but follows the reverse path. Further information can be found in [Eur06]. Moreover, the last subsection shall present a brief overview of OFDM given that it is one of the main features of the DAB system.

## 3.3 DAB Modes

As will be discussed later, the OFDM parameters must be chosen according to the requirements of the channel. A long symbol length $T_s$ with a long cyclic prefix make the system robust against long echoes, but sensitive against high Doppler frequencies [SL05]. Since DAB was design to operate in a wide range of frequencies (300MHz - 3GHz) and in several channels (terrestrial, satellite, cable), four transmissions modes have been defined with different sets of parameters (see subsection 3.5.5). The different modes are specified to accommodate different frequency ranges and operating conditions. In Section 15.1 of the DAB standard [Eur06], suitable conditions are described as follows (quote):

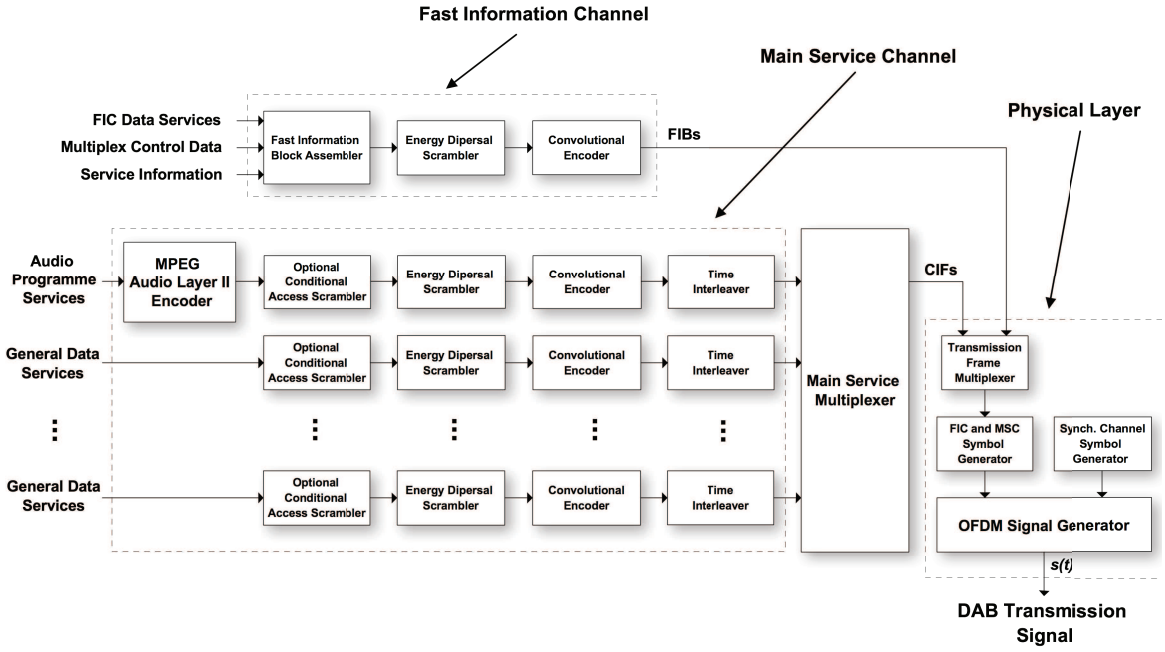


Figure 3.1: *Conceptual block diagram of the DAB transmitter.*

- Transmission mode I is intended to be used for terrestrial Single Frequency Networks (SFN) and local-area broadcasting in Bands I, II and III.
- Transmission modes II and IV are intended to be used for terrestrial local broadcasting in Bands I, II, III, IV, V and in the 1452 MHz to 1492 MHz frequency band (i.e. L-Band). It can also be used for satellite only and hybrid satellite-terrestrial broadcasting in L-Band.
- Transmission mode III is intended to be used for terrestrial, satellite and hybrid satellite-terrestrial broadcasting below 3 000 MHz.
- For cable distribution, transmission mode III is the preferred mode because it can be used at any frequency available on cable. However, transmission modes I, II and IV may also be used, depending on the chosen frequency band.

One of the most important DAB features is the possibility of using SFN, i.e, multiple geographically separated transmitters broadcasting at the same frequency. This requires that the transmitters have a maximum distance among them and be accurately time synchronized, by this way interferences at the receiver can be avoided [Mul08].

Although these different transmission modes also affect the FIC and the MSC, the main differences are on the physical layer.

## 3.4 Transport Mechanisms

As already said, there are two mechanisms for data transporting: the MSC and the FIC. The first one carries user information, such as audio data and other services. The FIC is

intended to transport the information that will allow the receiver to understand the MSC multiplex structure and to know which programmes are transmitted in the ensemble. Furthermore, a third mechanism called synchronization channel is used within the transmission system for synchronization and demodulation purposes. The next subsections shall describe each of these three channels.

### 3.4.1 Main Service Channel

The MSC is made up of Common Interleaved Frames (CIFs), each of these transports Audio Programme Services, Service Information or general Data Services. Obviously, the main service carried in the MSC is the stream based audio data. Depending on the service requirements, a packet based or a stream based mode can be used. All services contained in the MSC are multiplexed according to the information carried in the FIC. The simplified block diagram of the MSC chain is illustrated in Figure 3.2.

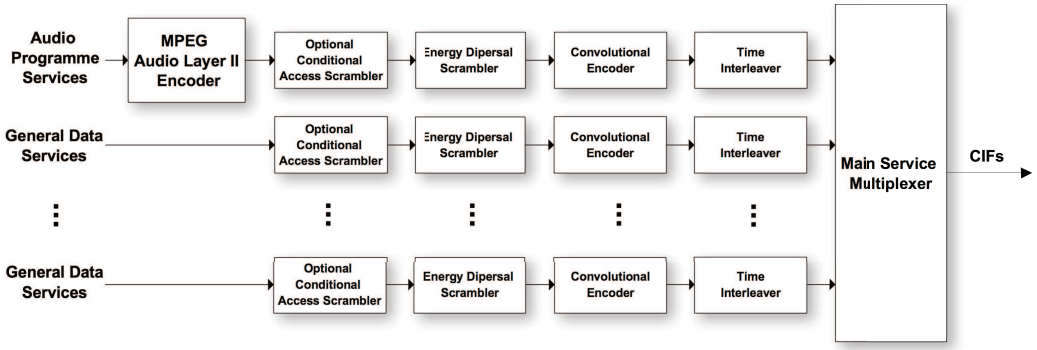


Figure 3.2: *Conceptual block diagram of the MSC encoder.*

Each sub-channel is individually encoded at source level, scrambled, error protected and time interleaved in the channel encoder. The next subsections shall describe each of this functions.

#### MPEG Audio Layer II codec

The available MSC gross bit rate is about 2.3 Mbit/s, including the audio programme data and error protection data. The balance of these two kinds of data establishes a trade-off between ruggedness of mobile reception and programme capacity. The ideal balance for terrestrial transmission would be an equal amount for both kinds of data, in this case 1.2 Mbit/s. However, a single stereo audio signal requires a bit rate of at least 1.5 Mbit/s [Gan03]. The source encoder is used to solve this problem by reducing the bit rate of the audio programme data. The DAB system uses MP2, which can reduce the required bit rate by a factor of 6 or more. The encoder can operate in stereo or mono mode and the output bit rate ranges from 8 kbit/s to 384 kbit/s [Eur06].

#### Conditional access scrambler

After source encoding, one can optionally provide Conditional Access (CA) to the bit stream. The CA is used to make the services incomprehensible to unauthorized users.

### Energy dispersal scrambler

Following the chain one can find the energy dispersal scrambler. Since it results in an outputted data with discontinuous energy, it might avoid unwanted regularity in the transmitted signal. For this purpose, a Pseudo Random Binary Sequence (PRBS) is generated by the Linear Feedback Shift Register (LFSR) with the following polynomial:

$$P(x) = x^9 + x^5 + 1 \tag{3.1}$$

with initial registers' states set to one. Subsequently, it is added modulo 2 to the data stream. Since the XOR operation is symmetric, the energy dispersal scrambler is exactly the same in the transmitter's and receiver's paths [Mul08].

### Convolutional encoder

To protect the data against errors, coding is applied. The encoder appears just after the energy dispersal scrambler and it is based on the punctured convolutional coding. Thus, the data is firstly encoded and subsequently punctured. It adds redundancy to the bit stream and allows both Equal Error Protection (EEP) and Unequal Error Protection (UEP).

As specified in [Eur06], the used code has constraint length 7, i.e., the current input bit and the 6 past bits are used. The mother convolutional encoder generates from the vector $(x_i)_{i=0}^{I-1}$, a codeword $\{(y_{0,i}, y_{1,i}, y_{2,i}, y_{3,i})\}$ defined by:

$$\begin{aligned} y_{0,i} &= x_i \bigoplus x_{i-2} \bigoplus x_{i-3} \bigoplus x_{i-5} \bigoplus x_{i-6} \\ y_{1,i} &= x_i \bigoplus x_{i-1} \bigoplus x_{i-2} \bigoplus x_{i-3} \bigoplus x_{i-6} \\ y_{2,i} &= x_i \bigoplus x_{i-1} \bigoplus x_{i-4} \bigoplus x_{i-6} \\ y_{3,i} &= x_i \bigoplus x_{i-2} \bigoplus x_{i-3} \bigoplus x_{i-5} \bigoplus x_{i-6} \end{aligned} \tag{3.2}$$

with $i = 0, 1, 2, , I + 5$ [Eur06]. Hence, the encoder produces four output bits from each input bit. The code is then punctured by taking out some bits generated by the mother's code. Thus, it is possible to achieve higher code rates and, consequently, different levels of code protection (UEP). Using UEP, it is possible to save capacity and add just the redundancy that is needed. The decoder can be implemented with Viterbi algorithm.

### Time interleaving

In order to spread the bits over a wider time span, each sub-channel is time interleaved. Mixing up the information in time intends to protect the data against short-time interferences that across all frequencies [Mul08]. This process is based on a convolutional interleaver.

### Main Service Multiplexer

Finally, the several sub-channels are multiplexed in the MSC. This channel is made up of CIFs which are characterized by an elementary 24 ms time period. Each CIF carries 55296 bits that are divided into 864 Capacity Units (CUs) of 64 bits. The total capacity of 864 CUs has to be shared by all the sub-channels. Since each of these sub-channels have a different data rate and error protection, the net data rate of the DAB system can not be defined. However, one can estimate a gross data rate of 2.304 Mbits/s.

### 3.4.2 Fast Information Channel

The FIC is made up of Fast Information Blocks (FIBs) and its primary purpose is to carry the control information necessary to set up the receiver circuitry before audio signals of the MSC be decoded. The essential part of this information is the Multiplex Configuration Information (MCI), which contains the multiplex structure and, sometimes, its re-configuration. Other data carried in the FIC includes the Service Information (SI), the CA management information and Fast Information Data Channel (FIDC) [Eur06]. The conceptual block diagram of the FIC is illustrated in Figure 3.3.

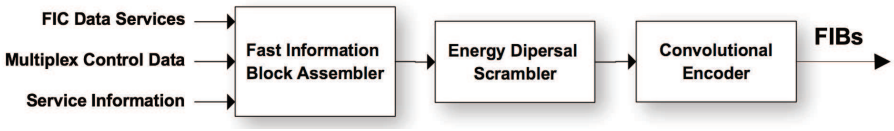


Figure 3.3: *Conceptual block diagram of the FIC encoder.*

The FIB assembler packs up the information of the FIC into packets of 256 bits called FIBs. Each FIB contains 30 bytes of data and a 16 bit Cyclic Redundancy Check (CRC) sum, which allows the receiver to verify correct decoding. The CRC is calculated with the following polynomial:

$$G(x) = x^{16} + x^{12} + x^5 + 1 \tag{3.3}$$

Subsequently, the data is scrambled and convolutionally encoded with fixed EEP. In order to allow the rapid and safe accesses to the MCI, the FIC is encoded with a high level of protection and it is not time interleaved so it do not suffer from an inherent delay. Hence, the receiver can firstly decode this channel and learn how to decode the MSC.

### 3.4.3 Synchronization Channel

The synchronization channel is used within the transmission system for demodulation purposes, such as transmission frame synchronization, automatic frequency control and channel state estimation [Eur06]. This channel is also used to carry optional Transmitter Identification Information (TII). The synchronization channel is composed by the first two OFDM symbols of each transmission frame.

#### Null symbol

The first of those symbols is the null symbol, which is used for rough time synchronization. During the time interval $[0, T_{NULL}]$, the transmitted signal is set to zero to indicate the beginning of a frame in the physical layer.

#### Phase reference symbol

The second OFDM symbol is called Phase Reference Symbol (PRS). The complex *Fourier* coefficients *sk* of this OFDM symbol were chosen in such way so that it serves as a frequency reference, as well as for echo estimation for the fine tuning of frame time synchronization. Furthermore, it is the start phase for the differential phase modulation [SL05].

## 3.5 Physical layer

The transmitted signal is built up around a transmission frame structure corresponding to the juxtaposition in time of the synchronization channel, the FIC and the MSC. Before the multiplexing of these three channels, the FIC and MSC are QPSK mapped and frequency interleaved. Then, they are differentially modulated with reference to the PRS. Finally, the Null symbol is added and the transmission frame is OFDM modulated to generate the transmission signal $s(t)$. The conceptual block diagram of the physical layer is shown in Figure 3.4 for transmission mode I.

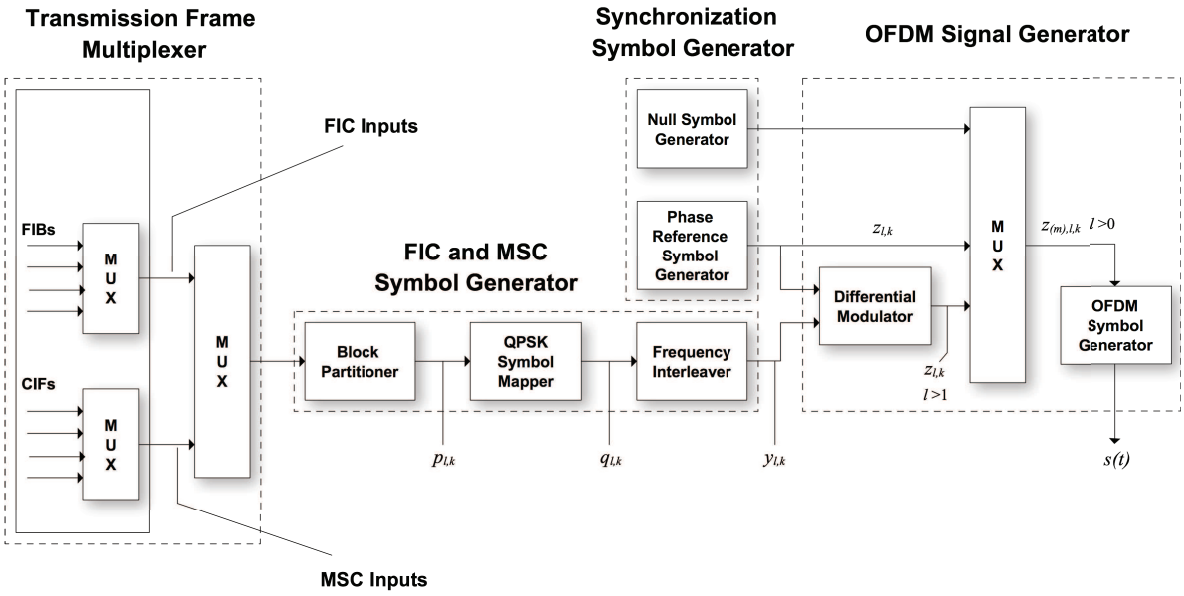


Figure 3.4: *Conceptual block diagram of the Physical Layer. [Eur06]*

The next subsections shall describe the several blocks of the physical layer.

### 3.5.1 Transmission Frame

In this block, both organization and length of a transmission frame depends on the transmission mode. For each of these, a transmission frame is defined as a periodically repeating sequence of OFDM symbols, each symbol consisting of a number of carriers. Therefore, it is essential that the time periods on the physical level and on the logical level be matched together. Thus, the period of a transmission frame is either the same as the audio frame length of 24 ms or an integer multiple of it [SL05]. The general structure of a transmission frame is depicted in Figure 3.5.

The transmission mode I is the one currently used in Portugal. Its transmission frame has a duration of 96 ms and is made up of 76 OFDM symbols. The first two symbols carry the synchronization channel, which consists in the Null symbol and the PRS. Then, 12 convolutionally encoded FIBs are transmitted within the next 3 OFDM symbols. At last, the remaining 72 OFDM symbols are made up of 4 CIFs that carry the data of the MSC. The 12 FIBs are divided in four groups, each one assigning one CIF of the same transmission frame.

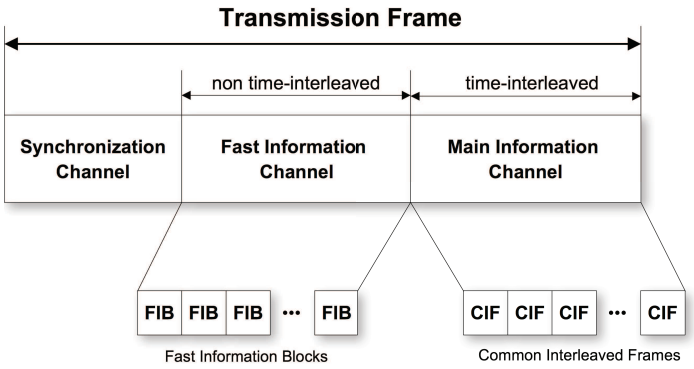


Figure 3.5: *DAB transmission frame structure.*

### 3.5.2 QPSK Symbol Mapper

For any OFDM symbols of index $l = 2, 3, 4, ..., L$, the $2K$-bit vector $(p_{l,n})_{n=0}^{2K-1}$ where $K$ is the number of carriers, shall be mapped on the $K$ complex QPSK symbols $q_{l,n}$ according to the rule:

$$q_{l,n} = \frac{1}{\sqrt{2}}[(1 - 2p_{l,n}) + j(1 - 2p_{l,n+K})] \quad for \quad n = 0, 1, 2, ..., K - 1[\text{Eur06}] \tag{3.4}$$

The resulting QPSK constellation is represented in Figure 3.6 a). In transmission mode I, the number of carriers $K$ is 1536 and the number of OFDM symbols $L$ is 76. Hence, a $2K = 3072$ bits from both FIC and MSC shall be mapped onto the 1536 complex modulation symbols for each OFDM symbol. The first 1536 bits will be mapped on the real part of the 1536 QPSK symbols and the last 1536 bits will be mapped on the correspondent imaginary part. In order to obtain a $\pi/4$ - DQPSK modulation, the reference symbol of that differential modulation must be mapped on a different constellation. Thus, the PRS constellation is the one represented in Figure 3.6 b).

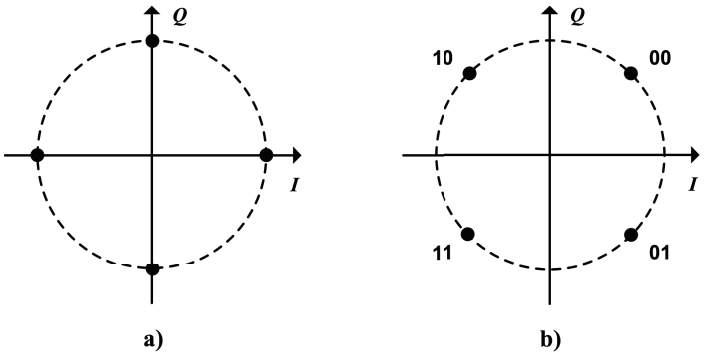


Figure 3.6: *QPSK constellations used by the PRS a) and the remaining symbols b).*

### 3.5.3 Frequency Interleaving

Next, the QPSK symbols shall be frequency interleaved. This process is done in order to introduce frequency diversity into the transmission signal, since the fading amplitudes of

adjacent OFDM subcarriers are highly correlated [SL05]. Thus, the frequency interleaving will define the correspondence between the index $n$ of the QPSK symbols $q_{l,n}$ and the carrier index $k$, where $k = \pm 1, \pm 2, \pm 3, ..., \pm K/2$. This is done by re-ordering the QPSK symbols according to a fixed pseudorandom permutation define by the following relation:

$$y_{l,k} = q_{l,n} \quad for \quad l = 2, 3, 4, ..., L \tag{3.5}$$

In the equation $k = F(n)$, where $F$ is a different function for each transmission mode. For further information see [Eur06].

### 3.5.4 Differential Modulation

The QPSK symbols of each carrier will be differentially modulated according to the following rule:

$$z_{l,k} = z_{l-1,k} z_{l,k} \quad for \quad l = 2, 3, 4, ..., L \quad and \quad k = \pm 1, \pm 2, \pm 3, ..., \pm K/2 \tag{3.6}$$

Thus, the phase of the current symbol is always added to the phase of the previous symbol, starting with reference to the PRS. This results in output symbols with constellations alternating between the constellations depicted in Figure 3.6 a) and b). Since each constellation is shifted by $\pi/4$ relatively to the previous one, this modulation is called $\pi/4$ - DQPSK.

### 3.5.5 OFDM Modulation

After multiplexing the Null symbol with the differentially modulated ones, the main signal $s(t)$ can be finally defined using the following formula:

$$s(t) = Re\{e^{j2\pi f_c t} \sum_{m=0}^{+\infty} \sum_{l=0}^{L} \sum_{k=-K/2}^{+K/2} z_{m,l,k} \times g_{k,l}(t - mT_F - T_{NULL} - (l-1)T_s)\} \tag{3.7}$$

with

$$g_{k,l}(t) = \begin{cases} 0 & for \quad l = 0 \\ e^{j2\pi k(t-\Delta)/T_U} \times Rect(t/T_s) & for \quad l = 1, 2, ..., L \end{cases} \tag{3.8}$$

and $T_s = T_U + \Delta$ [Eur06].

The complex $\pi/4$ - DQPSK symbols $z_{l,k}$ are the *Fourier* coefficients of the *lth* OFDM symbol of each frame. The several parameters and variables are defined as follows [Eur06]:

$L$ is the number of OFDM symbols per transmission frame (the Null symbol being excluded)

$K$ is the number of transmitted carriers

$T_F$ is the transmission frame duration

$T_{NULL}$ is the Null symbol duration

$T_S$ is the duration of OFDM symbols of indices $l = 1, 2, 3, ..., L$

$T_U$ is the inverse of the carrier spacing

$\Delta$ is the duration of the time interval called cyclic prefix

$z_{m,l,k}$ is the complex DQPSK symbol associated to carrier $k$ of OFDM symbol $l$ during transmission frame $m$

$f_c$ is the central frequency of the signal

Please note that $z_{m,l,k} = 0$ for $k = 0$, which means that the central subcarrier is not used. Moreover, the value $g_{l,k}(t) = 0$ for $l = 0$ represents the Null symbol. These parameters are specified in Table 3.1 for transmission modes I, II, III and IV. The values of the several time related parameters are given in multiples of the elementary period $T = 1/2\ 048\ 000$ seconds, and approximately in milliseconds or microseconds. Note that 2048 kHz is the sampling frequency if the smallest possible Fast Fourier Transform (FFT) length is used for each transmission mode [Eur06].

| Parameter | Transmission mode I | Transmission mode II | Transmission mode III | Transmission mode IV |
|---|---|---|---|---|
| $L$ | 76 | 76 | 153 | 76 |
| $K$ | 1536 | 384 | 192 | 768 |
| $T_F$ | 196 608 $T$<br>~ 96 ms | 49 152 $T$<br>~ 24 ms | 49 152 $T$<br>~ 24 ms | 98 304 $T$<br>~ 48 ms |
| $T_{NULL}$ | 2 656 $T$<br>~ 1,297 ms | 664 $T$<br>~ 324 ms | 345 $T$<br>~ 168 ms | 1 328 $T$<br>~ 648 ms |
| $T_S$ | 2 552 $T$<br>~ 1,246 ms | 638 $T$<br>~ 312 µs | 319 $T$<br>~ 156 µs | 1 276 $T$<br>~ 623 µs |
| $T_u$ | 2 048 $T$<br>1 ms | 512 $T$<br>250 µs | 256 $T$<br>125 µs | 1 024 $T$<br>500 µs |
| $\Delta$ | 504 $T$<br>~ 246 µs | 126 $T$<br>~ 62 µs | 63 $T$<br>~ 31 µs | 252 $T$<br>~ 123 µs |

Table 3.1: *DAB transmission signal parameters. [Eur06]*

Since transmission mode I is the one currently used in Portugal, the remaining chapters will use its corresponding parameters. The remaining modes can be obtained by changing the parameters on the table.

## 3.6 Orthogonal Frequency Division Multiplexing

Initial proposals for *Orthogonal Frequency Division Multiplexing* (OFDM) were made between 1960 and 1970. It took more than a quarter of a century for this technology to move from the research domain to the industry. The concept of OFDM is simple, but its implementation brings many complexities. The next subsections will describe its advantages, implementation considerations and additional protection techniques.

### 3.6.1 OFDM as a Multicarrier Transmission

Let us consider a digital transmission scheme with a linear carrier modulation and a symbol duration $T_S$. Let the occupied bandwidth $B$ be approximately equal to $T_S^{-1}$. For a transmission channel with a delay spread $\tau_m$, a reception free of interference will be only

possible if the condition

$$\tau_m << T_S \tag{3.9}$$

is fulfilled. Thus, the bit rate $R_b = log_2(M)T_S^{-1}$ for a given single carrier modulation scheme will be limited by the delay spread of the channel [SL05].

The concept of multicarrier transmission comes to solve this problem. A single stream of data is split into $K$ parallel substreams of lower data rates each of which is coded and modulated onto a different subcarrier. This can be regarded as a transmission parallel in the frequency domain, and it does not affect the total bandwidth that is needed. Each subcarrier has a bandwidth $B/K$, while the symbol duration $T_S$ is increased by a factor of $K$, which allows for a $K$ times higher data rate for a given delay spread [SL05]. That is, a transmission parallel in the frequency adds some additional immunity to impulse noise and reflections.

The concept explained above is the main idea of a Frequency Division Multiplexing (FDM) system. FDM is much efficient than single carrier modulation in the presence of narrowband frequency interferences. Since this interference will affect a specific frequency band, only a few subcarriers are lost. In a single carrier modulation at that frequency, one or more symbols would be completely lost. This concept is illustrated in Figure 3.7.

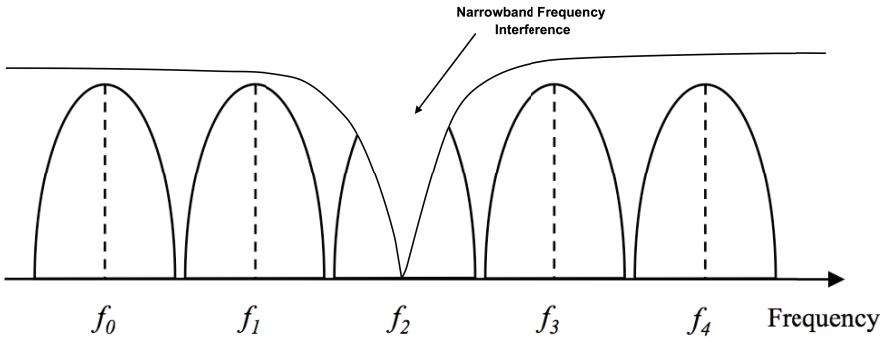


Figure 3.7: *Narrowband frequency interference effects in a FDM modulation system.*

The FDM carriers are all independent of each other. Hence, an FDM system requires a guard band between modulated subcarriers to prevent one subcarrier's spectrum from interfering with another. However, these guard bands lower the bit rate $R_b$ when compared to a single carrier system with similar modulation [LP01].

If the FDM system subcarriers were orthogonal to each other, the subcarriers could have overlap increasing the spectral efficiency. As long as orthogonality is maintained, it is still possible to recover each subcarrier signal despite their overlapping spectrums [LP01]. This is what makes an OFDM system one of the most adopted modulation scheme in emergent standards. Figure 3.8 depicts both FDM and OFDM frequency spectrums.

### 3.6.2 OFDM System Implementation

Conceptually, the process of generate an OFDM system is identical to the one of generate a common FDM system. Thus, considering a system with $K$ subcarriers the data stream is split up into $K$ parallel substreams, and each one is modulated with its own subcarrier at frequency $f_k$ in the complex baseband, described by the complex harmonic wave $e^{j2\pi fkt}$. The complex baseband signal is then given by the summation of all $K$ complex harmonic waves.

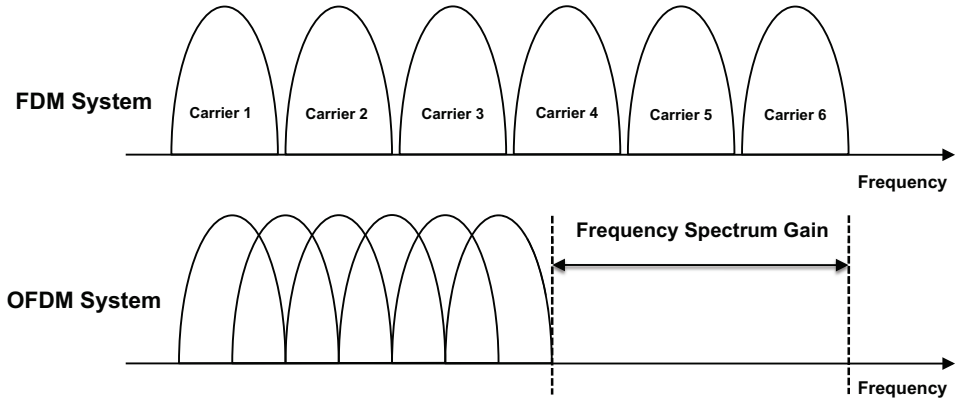


Figure 3.8: *FDM and OFDM frequency spectrums. [Sil08]*

This scheme can be implemented using the Discrete Fourier Transform (DFT). Recall from digital signal processing theory that the sinusoids of the DFT form an orthogonal basis set, and a signal in the vector space of the DFT can be represented as a linear combination of the orthogonal sinusoids [LP01]. This transform is therefore used at the OFDM transmitter to map an input signal onto a set of orthogonal subcarriers. Similarly, the same transform is used at the receiver side to demodulate the received subcarriers. Figure 3.9 shows the block diagram of an OFDM transmitter.

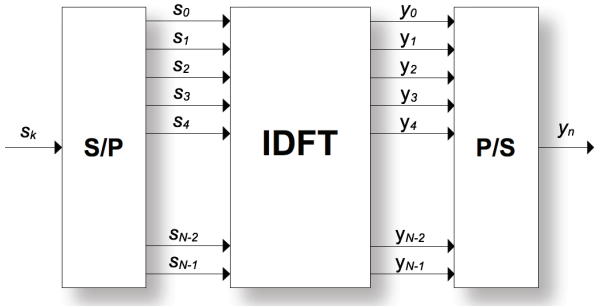


Figure 3.9: *Block diagram of an OFDM transmitter.*

Thus, the data stream is fed into a Serial/Parallel (S/P) converter which outputs $K$ parallel complex symbols $s_k$ (e.g. PSK or QAM). Each of these is subsequently modulated on a different subcarrier through the Inverse Discrete Fourier Transform (IDFT) and reconverted to a serial data stream. The complex time domain signal is given by the following expression:

$$y_n = \sum_{k=0}^{N-1} s_k e^{j\frac{2\pi}{N}kn} \tag{3.10}$$

where $N$ is the IDFT size, which is in this case equal to the number of carriers $K$. At the receiver, the signal follows the reverse path. The baseband signal is converted into parallel and subsequently fed into the DFT that converts it to the corresponding frequency domain symbols $s_k$. Finally, the complex data steam is recovered through a Parallel/Serial (P/S) converter.

In practice, OFDM systems are implemented using a combination of both FFT and Inverse Fast Fourier Transform (IFFT) blocks, which are more efficient versions of the DFT and IDFT, respectively. Moreover, an OFDM system can experience fades just as any other system. Coding is therefore required for all subcarriers, as well as a frequency interleaving scheme in order to avoid frequency selective fading.

### 3.6.3 Multipath Channels and the use of a Guard Interval

As already said, one of the major reasons to use an OFDM scheme is its performance in multipath channels. In these channels, the transmitted signal reflects on several objects. As a result, multiple delayed versions of the transmitted signal arrive at the receiver. This results in two main problems for an OFDM system: Intersymbol Interference (ISI) and Intercarrier Interference (ICI).

#### Intersymbol interference

In OFDM, the symbol duration $T_s$ is sufficiently long so that all delayed versions of the signal approximately arrive within the symbol. They do not spill over to subsequent symbols and therefore there is no ISI. As data rates go up or the channel delay increases, ISI starts to occur. Since the symbol duration $T_s$ is much longer then the delay spread $\tau_m$, the ISI effect will only distort the first samples of the received OFDM symbol. Thus, one can consider the use of a time domain guard interval to allow for multipath delay spread $\tau_m$. Since this guard interval would only contain zeros, it can be discarded at the receiver. Thus, delayed echoes wont interfere with the subsequent symbols. Figure 3.10 illustrates the use of a guard interval and its role in a multipath channel.

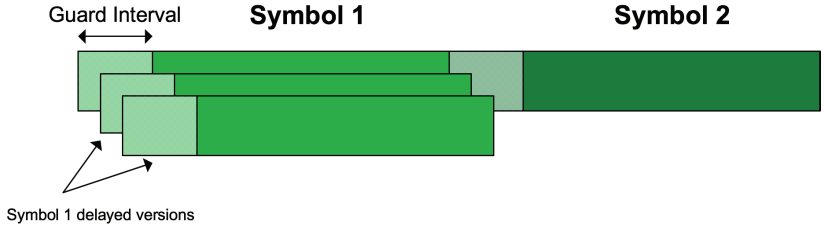


Figure 3.10: *Guard interval and its role in a multipath channel. [Sil08]*

The guard time must be designed to be larger than the expected delay spread. Reducing ISI from multipath delay spread leads to deciding on the number of subcarriers and the length of the guard interval [LP01].

#### Interrcarrier interference

The ICI problem is unique to multicarrier systems [LP01]. It is related with interferences amongst OFDM subcarriers. Adding the guard time with no information brings problems to the FFT, which results in ICI. A delayed version of one subcarrier can interfere with another subcarrier in the next symbol period. This is avoided by replacing the guard interval with something known as cyclic prefix. The cyclic prefix is a replica of the last samples of the OFDM symbol. Since this cyclic prefix only contains redundant information, it can also be

discarded at the receiver. It ensures that delayed symbols will have an integer number of cycles within the FFT integration interval. One should note that the FFT integration period excludes the cyclic prefix [LP01]. Summarizing, the cyclic prefix can remove both ISI and ICI as long as the delay spread is shorter than it.

### 3.6.4 OFDM Synchronization

Since subcarriers orthogonality must be preserved, OFDM synchronization in frequency and time domains must be extremely well performed.

If the OFDM scheme does not use cyclic prefix, time synchronization must be highly accurate. Otherwise, it would imply ISI. When using a cyclic prefix, a wrong synchronization within the cyclic prefix interval only originates a rotation of the complex symbols' constellation, as depicted in Figure 3.11 for a QPSK modulation with 10 carriers and a time offset of one sample. Since an OFDM system is usually differentially modulated, this rotation is

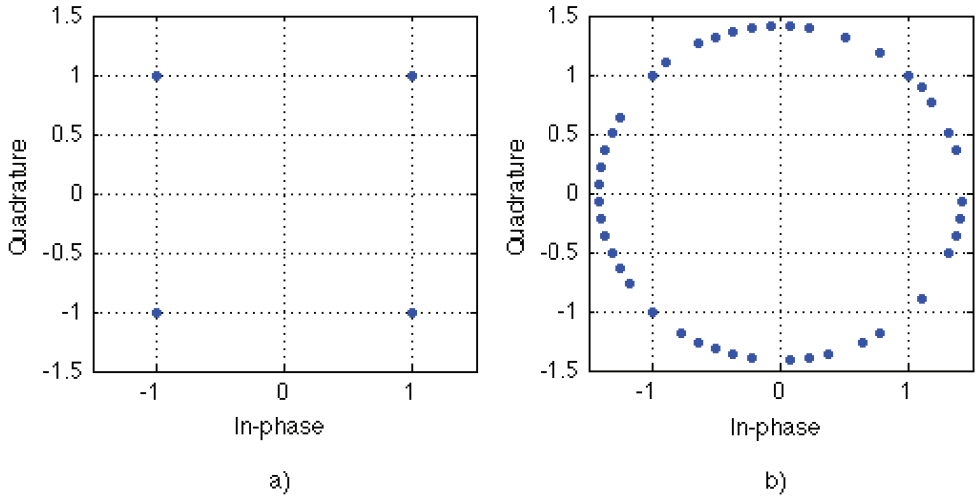


Figure 3.11: *QPSK symbols before OFDM modulation a) and after OFDM demodulation with a time offset of one sample b).*

acceptable.

On the other hand, the effects of an imperfect frequency synchronization are much dangerous. If a frequency offset is not corrected, the orthogonality among carriers is lost originating ICI. It happens because the ODFM demodulated symbols will be recovered at the wrong samples of its corresponding carriers' sinusoids. Figure 3.12 illustrates a frequency offset demodulated signal. Thus, large frequency offsets will originate a complete distortion of the signal. One should note that frequency shifts are mainly originated by receiver imperfections.

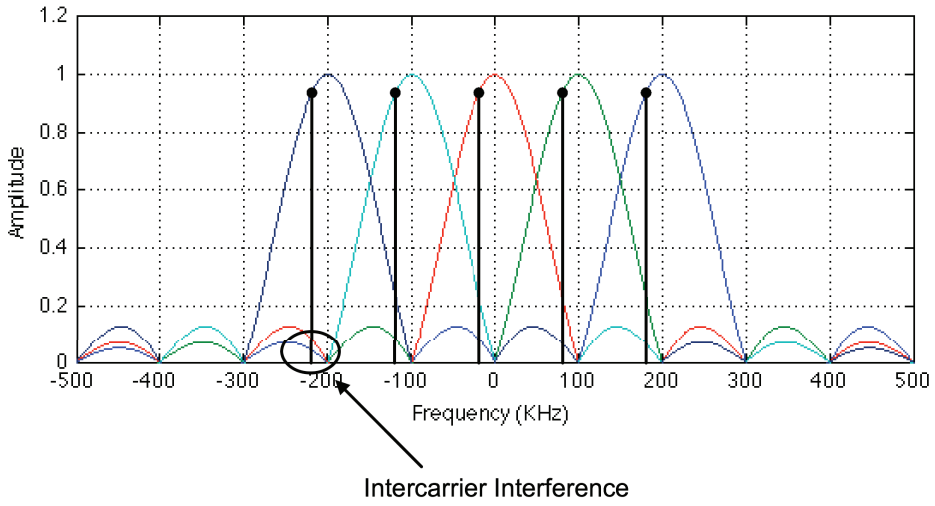


Figure 3.12: *Effect of a frequency offset in an OFDM demodulated signal.*

# Chapter 4

# Receiver Simulation in MATLAB

## 4.1 Introduction

As described in the previous chapter, the physical layer of a DAB transmitter consists in a chain of blocks that performs the QPSK mapper, the frequency interleaving, the differential modulation and the OFDM modulation. In the receiver side, the same functions are performed in the reverse order: OFDM demodulation, differential demodulation, frequency deinterleaving and QPSK demapper.

However, due to channel response and front-end imperfections the signal received is not the same as the one transmitted. Noise, frequency offsets and time delays are some of the problems which one has to deal when developing a radio receiver. Therefore, to the receiver blocks already mentioned we must add the most challenging one, the synchronization block. The receiver block diagram can be seen in Figure 4.1.

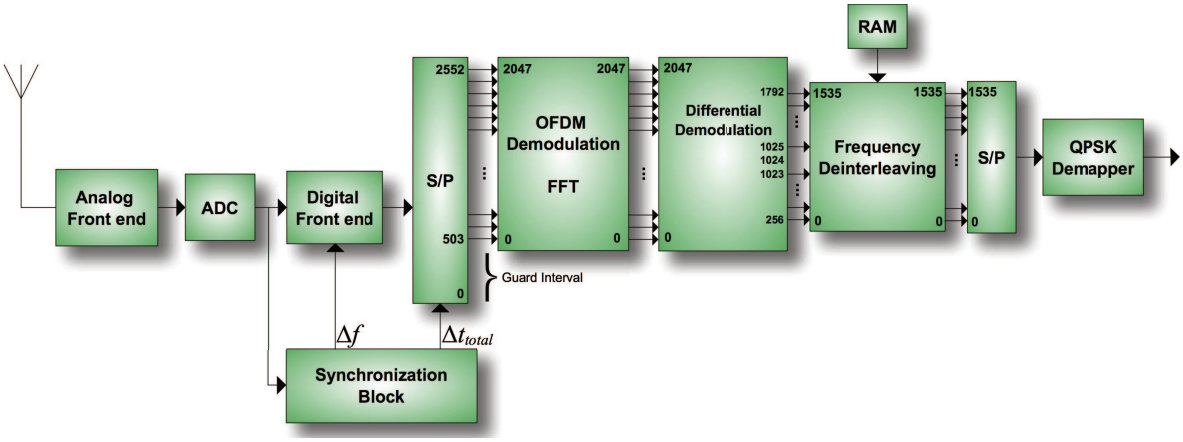


Figure 4.1: *General block diagram.*

In order to show a complete DAB receiver, the analog front-end, ADC and digital front-end blocks are also included in Figure 4.1. The former one is not included in the objectives of this work and it is being developed in the scope of another master thesis. The latter two blocks, ADC and digital front-end, will also depend on the analog front-end features. Therefore, they will be implemented in a future work. The remaining blocks were already designed and simulated in MATLAB.

The next subsection presents several digital front-end implementation options concerning the analog front-end properties. The subsequent two subsections explain each of the signal demodulation blocks and the synchronization block, as well as the several choices that were made when developing them. The last subsection presents the results of the several tests such as channel simulation and real DAB signal demodulation.

Appendix B presents the several DAB functions that were implemented in MATLAB, as well as their brief description and signal flow graph.

## 4.2 Digital Front-End

The digital front-end configuration will depend on the analog-to-digital domains interface. There are two possible types of interface [HL01]: I/Q interface and IF interface.

### 4.2.1 I/Q Interface

When the receiver is based on the I/Q interface, the In-phase (I) and Quadrature (Q) components of the complex baseband signal are generated in the analog domain. Thus, the digital front-end is only necessary if the ADC sampling frequency does not match with the desired one, i.e., if the signal is oversampled. In that case, the digital front-end has two branches, each one composed by a Low Pass Filter (LPF) and a digital decimator.

As mentioned above, the I/Q interface is usually related with baseband signals. Therefore, the superheterodyne front-end of Chapter 2 is one of the many architectures that can be use with this kind of interface. Figure 4.2 shows the I/Q receiver with both superheterodyne front-end and digital front-end.

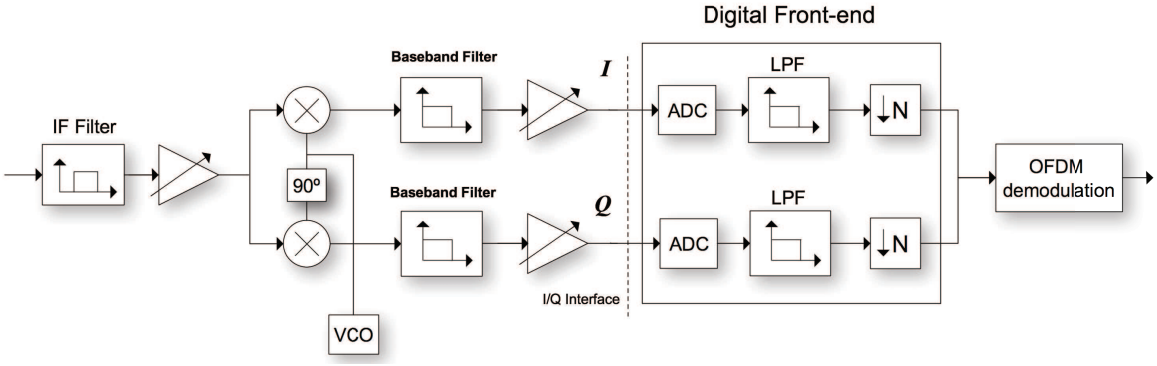


Figure 4.2: *Receiver architecture with an I/Q interface.*

Thus, the in-phase and quadrature signal components are fed into the ADCs in order to be digitalized. Then, the sampled signal is low pass filtered. Finally, the signals are decimated toward the sampling rate which best fits the OFDM demodulation implementation, normally the smallest power of two is chosen is order to optimize the hardware.

Moreover, frequency adjustments might be performed in order to enable correct signal demodulation. In this case, the digital front-end will need an additionally Numerical Controlled Oscillator (NCO) in order to change the signal frequency.

### 4.2.2 IF Interface

A receiver with an IF interface implies the signal digitalization at an IF frequency. The required receiver architecture can be seen in the Figure 4.3.

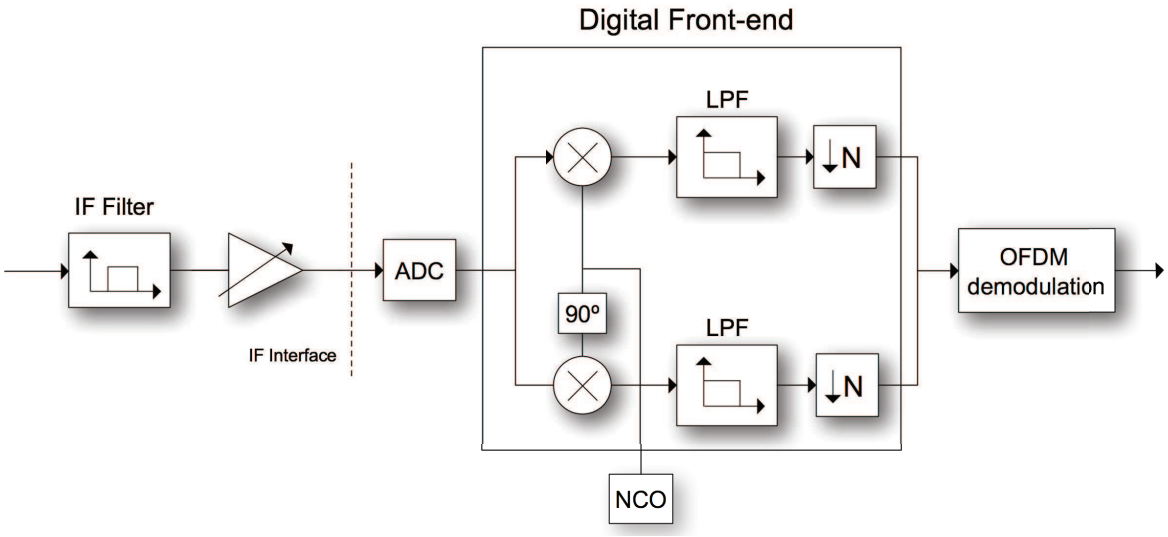


Figure 4.3: *Receiver architecture with an IF interface.*

Through the figure, one can see that the superheterodyne front-end derivation was used. The homodyne architecture could have also been used, among others. Thus, the IF signal of the analog front-end is fed into the ADC and the in-phase and quadrature components are generated in two different branches of the digital domain. In each of these, the same NCO is used to generate the complex components and to shift the signals toward to zero frequency. Following the branches, one can find LPFs and decimators which have the same aim as in the I/Q receiver.

The major advantage of this architecture is the possibility to choose the IF that best fits a specific analog front-end architecture as long as it is compatible with the ADC bandwidth and sampling rate. It also allows an easier frequency correction, which can be performed by the digital front-end multipliers. Unlike the previous subsection, it does not need an additionally NCO.

The actual choice of the ADC sampling rate is a trade-off between the filtering in the analog and digital domains [HL01]. Moreover, the hardware implementation cost can be minimised by choosing the sampling rate of the ADC as an integer multiple of the sampling frequency of the complex baseband signal, otherwise the signal must be fractionally decimated which is much more computationally intensive.

## 4.3 Signal Demodulation System

The signal demodulation system is obviously composed by all functions that are related with signal demodulation. In the transmitter side, these functions perform QPSK mapper, frequency interleaving, differential modulation and OFDM modulation. In the receiver, the

functions are the same but in the reverse order: OFDM demodulation, differential demodulation, frequency deinterleaving and QPSK demapper. Figure 4.4 illustrates the signal flow through these receiver functions.

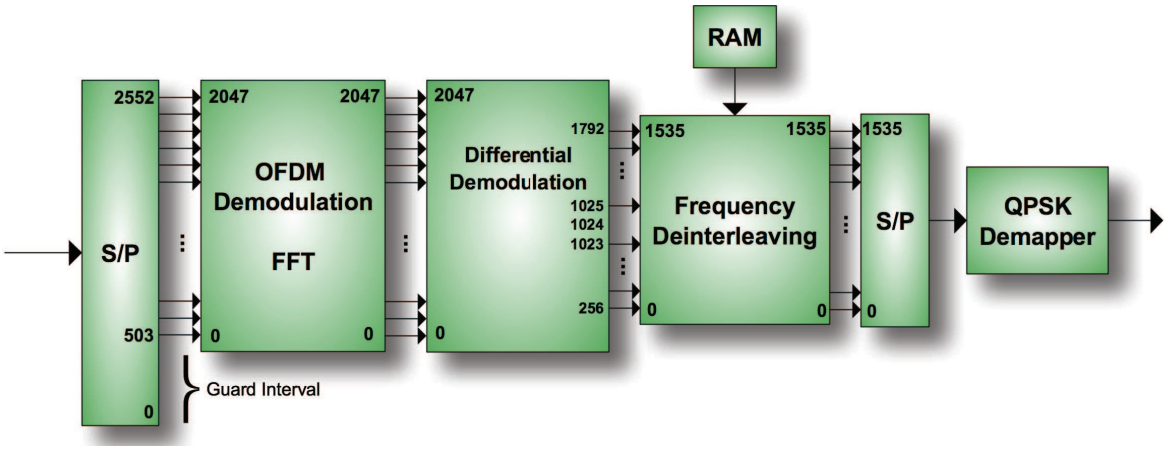


Figure 4.4: *Signal demodulation system block diagram.*

In what concerns the signal demodulation system, both transmitter and receiver were developed in MATLAB according to Chapter 3. One should note that there are a main diference between Chapter 3 and our implementation. Here, the DAB signal spectrum must be shifted 1537 KHz to the right in order to have no negative frequencies. By this way, the FFT from MATLAB can be used to (de)modulate the OFDM symbol. Thus, our range of carriers will be:

$$k = 0, 1, 2, ..., K-1 \quad where \quad K = 1536 \tag{4.1}$$

instead of:

$$k = 0, \pm 1, \pm 2, ..., \pm K/2 \quad where \quad K = 1536 \tag{4.2}$$

Moreover, the letters' size will be used to distinguish time and frequency domains. Lower case letters will be used to represent the time domain, while upper case ones to represent the frequency domain. Thus, some of Chapter 3 lower case variables will be used here with an upper case representation, but the nomenclature remains the same.

Additionally, DAB transmitter's and receiver's MATLAB objects were created [McG08]. These allow an easy and flexible reuse, which can help another users to develop DAB based systems or, perhaps, be used in signal processing subjects with an educational purpose. Appendix C explains both objects and gives a brief example of their use.

The following subsections shall explain the several functions of the signal demodulation system.

### 4.3.1 OFDM Demodulation

The demodulation of the OFDM symbol is performed by applying FFTs to calculate the complex amplitudes of the carriers of the DAB spectrum The FFT is a computationally efficient algorithm for computing the DFT. Thus, the OFDM demodulated signals can be defined as follows:

$$Z_k = \sum_{n=0}^{N-1} r_n e^{j\frac{2\pi}{N}nk} \tag{4.3}$$

where $r_n$ is the received OFDM symbol and $N$ is the transform size. These complex amplitudes contain the information of the modulated data by means of a DQPSK modulation [HSH99]. The OFDM demodulation block diagram can be seen in Figure 4.5.

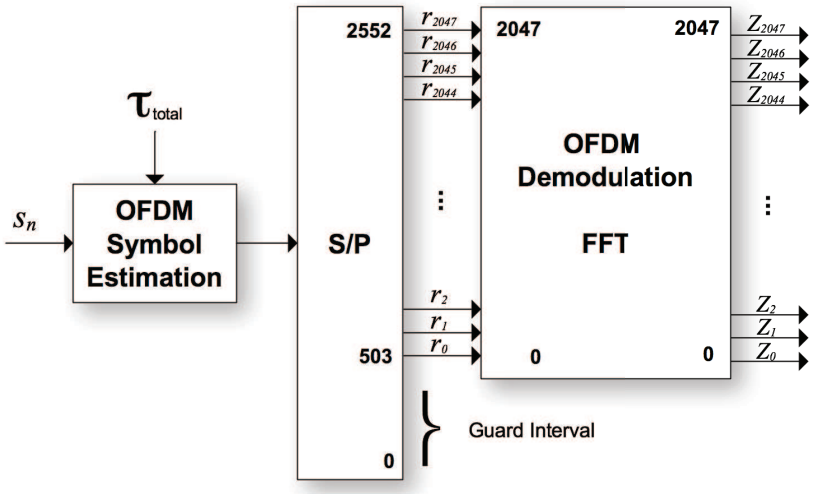


Figure 4.5: *OFDM demodulation block diagram.*

The first block is responsible for estimating the OFDM symbol based on the temporal information $\tau_{total}$. As will be seen in the next subsection, this temporal value is given by the synchronization block and is made up of the coarse and fine frame timing estimations:

$$\tau_{total} = \tau_c + \tau_f \tag{4.4}$$

Knowing the frame beginning, this block can easily estimate the remaining 76 OFDM symbols. Next, the serial to parallel block slices the sample stream into subsequent vectors with one OFDM symbol each, i.e. each vector has the length of one OFDM symbol. Finally, the cyclic prefix of each OFDM symbol is removed and the FFT is performed in order to transform those symbols into the frequency domain.

Since our intention is to implement a DAB mode I receiver, the size of the FFT $N$ must have a value larger than 1536. The smallest power of two was chosen in order to optimize the hardware, i.e., 2048. The remaining DAB mode related values are given in the table 3.1 of Chapter 3

### 4.3.2 Differential Demodulation

Since DAB uses a DQPSK modulation, the phase difference of consecutive symbols must be calculated. Hence, differential demodulation is usually performed by complex multiplying each OFDM symbol with the stored complex conjugate amplitude of the previous symbol. That is:

$$Y_{l,k} = Z_{l,k} \times Z^*_{l-1,k} \quad k = 0, 1..., 1535 \tag{4.5}$$

where $Y_l$ is the demodulated symbol, $Z_l$ is the incoming symbol and $Z_{l-1,k}$ is the previous symbol. The multiplication must be applied to each of the active carriers $k$, i.e, the ones that carry information. As explained in Chapter 3, this process is initialized in every frame by using the PRS. Figure 4.6 gives an overview of the algorithm. Through the figure, one can

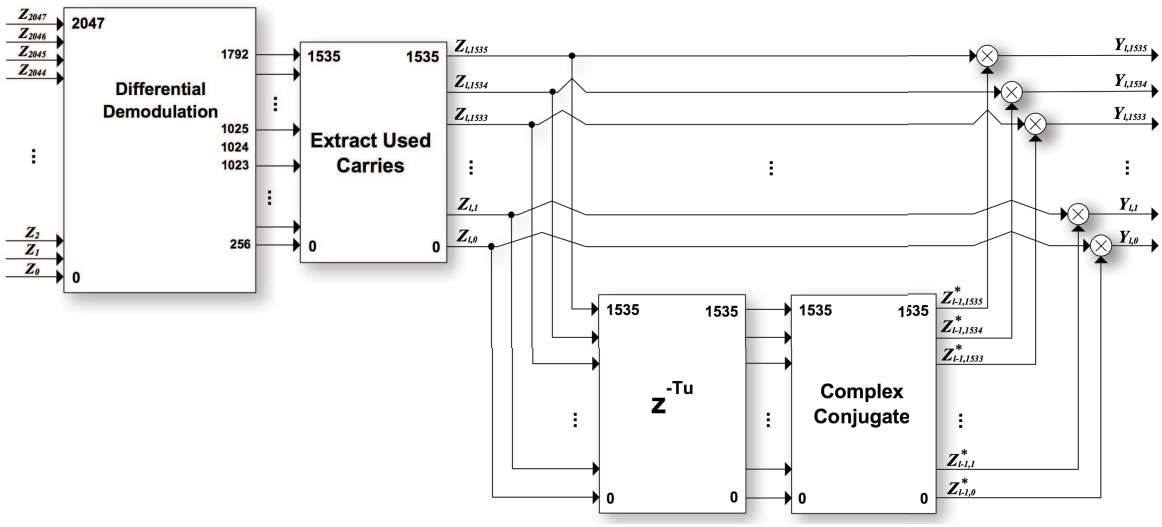


Figure 4.6: *Differential demodulation block diagram.*

see that the differential block also removes the unused carriers. Since we are implementing a DAB receiver mode I, this block must output vectors with 1536 samples, i.e., carriers. The PRS is also removed because it is no longer needed.

### 4.3.3 Frequency Deinterleaving

In order to cope with transmission disturbances, two interleaving mechanisms are used: frequency and time interleaving.

Frequency interleaving is a rearrangement of the digital bit stream of each OFDM symbol over the several subcarriers, which is done according to a fixed sequence specified in the DAB standard (see Chapter 4). This mechanism helps to eliminate the effects of selective fades. Once the bits are spread in the frequency spectrum, a short narrowband interfering signal is less likely to disturb two bits that belong to the same codeword [Mul08].

The frequency deinterleaving can thus be implemented by addressing the output of the differential demodulation block according to the deinterleaver fixed sequence.

### 4.3.4 QPSK Symbol Demapper

QPSK symbol demapping decides whether a bit is "1" or "0". Since, both real and imaginary parts of a given differential modulated symbol correspond to two independent bits, these can be directly evaluated as depicted in Figure 4.7.

Thus, QPSK symbol demapping simply checks whether the real and the imaginary parts of a given symbol are positive or negative. That is:

$$\begin{cases} I < 0 & \Rightarrow & bit_k = 1 \\ I > 0 & \Rightarrow & bit_k = 0 \\ Q < 0 & \Rightarrow & bit_{K+k} = 1 \\ Q > 0 & \Rightarrow & bit_{K+k} = 0 \end{cases} \quad where \quad k = 0, 1, ..., K-1 \quad and \quad K = 1536 \tag{4.6}$$

By this way, it is not necessary to calculate the phase of each symbol, which is very slow and computationally intensive. Per each OFDM symbol, the real parts of the several QPSK

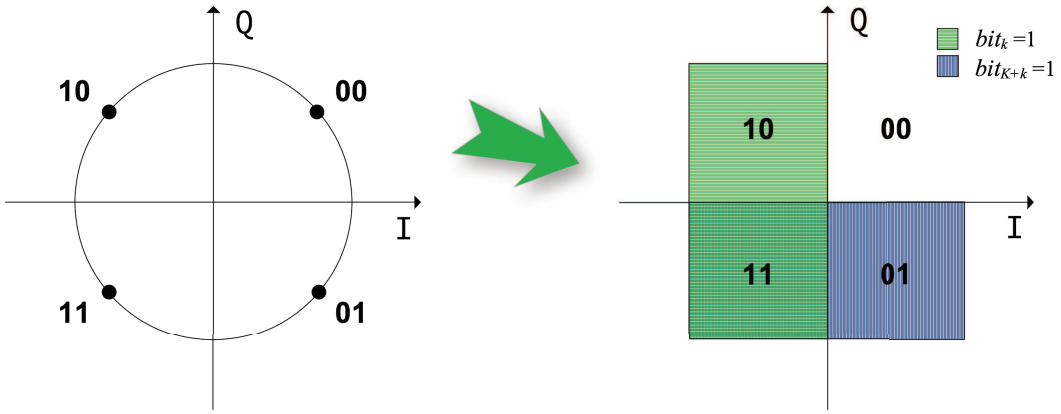


Figure 4.7: *QPSK symbol demapper illustration.*

symbols correspond to the first $K$ bits, while the corresponding imaginary parts correspond to the last $K$ bits.

## 4.4 Synchronization System

### 4.4.1 Introduction

The DAB system requires an accurate synchronization in what concerns frequency and timing of the incoming signal. As it was designed to work in mobile reception conditions, an OFDM scheme was adopted due to its robustness against the effects of multipath propagation. However, the following three aspects of synchronization need to be considered in order to guarantee correct OFDM demodulation (quote) [Gan03]:

1. The rate at which FFT computations must be carried out must be the reciprocal of the active OFDM symbol duration of the incoming signal, i.e., the symbol duration without the cyclic prefix.

2. The FFT baseband input signal must be such that the FFT output carriers frequencies lie precisely on the correct "teeth" of the comb with frequency separation equal to the reciprocal of the active symbol duration.

3. The FFT processing window must be chosen in such a way that the active symbol duration makes the greatest constructive use of the received multipath signals.

The first aspect must be assured in order to avoid inter-carrier crosstalk, i.e., ISI. It is related to the sampling frequency, which must obey to the following relation:

$$f_s = \frac{N}{T_s - \Delta} \tag{4.7}$$

where $f_s$ is the sampling frequency, $N$ is the FFT size, $T_s$ is the duration of the OFDM symbol and $\Delta$ is the duration of the cyclic prefix.

The second aspect must be guaranteed to avoid corruption of the differential coding and ISI. Since the OFDM system is very sensitivity to frequency offset, the carrier frequencies of the receiver must be kept consistent with the transmitter's ones. Thus, to correctly demodulate the signal one has to estimate and correct the frequency offset $\Delta f$. This offset can be divided as follows:

$$\Delta f = \delta_{in} f_U + \delta_{fr} f_U \tag{4.8}$$

where $f_U$ is the carrier spacing, $\delta_{in}$ is an integer and $\delta_{fr}$ is a decimal fraction less than 0.5. The term $\delta_{in} f_U$ is called coarse frequency offset and covers integral multiples of the distance between carriers. While the term $\delta_{fr} f_U$ is called fine frequency offset and covers frequency offsets that are less than half of carrier spacing.

The last aspect concerns to timing synchronization. A multipath channel has many adversities such as signal reflections and refractions, which originates multiple paths and different arrival times. However, this wireless channel has an impulse response that changes with time. Usually the time spread is used to give an average measure of the "length" of that impulse response. The cyclic prefix was introduced in order to support these several signal delays, thus if this delay spread is within the cyclic prefix, ICI is avoided and it is still possible to select a symbol interval without multipath adversities. To choose the start point in which the active symbol makes the greatest use of the received signal one has to perform an accurate time synchronization process [Gan03].

As mentioned in Chapter 3, the beginning of each DAB frame carries the synchronization channel whose main purpose is to provide means for both time and frequency synchronizations. It consists in the first two OFDM symbols, the Null symbol followed by the PRS. The former one provides coarse time synchronization, while the latter one is used by some of the remaining synchronization processes, as well as a reference symbol for differential (de)modulation. The PRS has a fixed pattern that was chosen in order to help its recognition when in the presence of noise and interferences. It is based on a CAZAC sequence.

### 4.4.2 Synchronization Scheme

Many methods have been proposed on estimation of time and frequency offsets. Usually, each of these offsets estimation is divided in two steps: fine and coarse.

While coarse time synchronization, also called frame synchronization, uses the Null symbol to quickly obtain the frame beginning, fine time synchronization gives a more accurate timing synchronization based on the PRS. As mentioned above, the frequency synchronization is also divided in two parts: integral and fractional offsets estimation/correction also called fine and coarse frequency synchronizations. The first one can be evaluated through the PRS or the cyclic prefix, while the former one usually uses the PRS.

In this dissertation, for each kind of synchronization one of the several options was chosen. The block diagram of the developed synchronization scheme is illustrated in Figure 4.8. Through that figure, one can see the synchronization chain:

1. First, the digital front-end moves the signal toward the baseband.

2. Next, one must perform coarse time synchronization, which simply detects the Null symbol in order to provide easily and quickly rough frame synchronization.

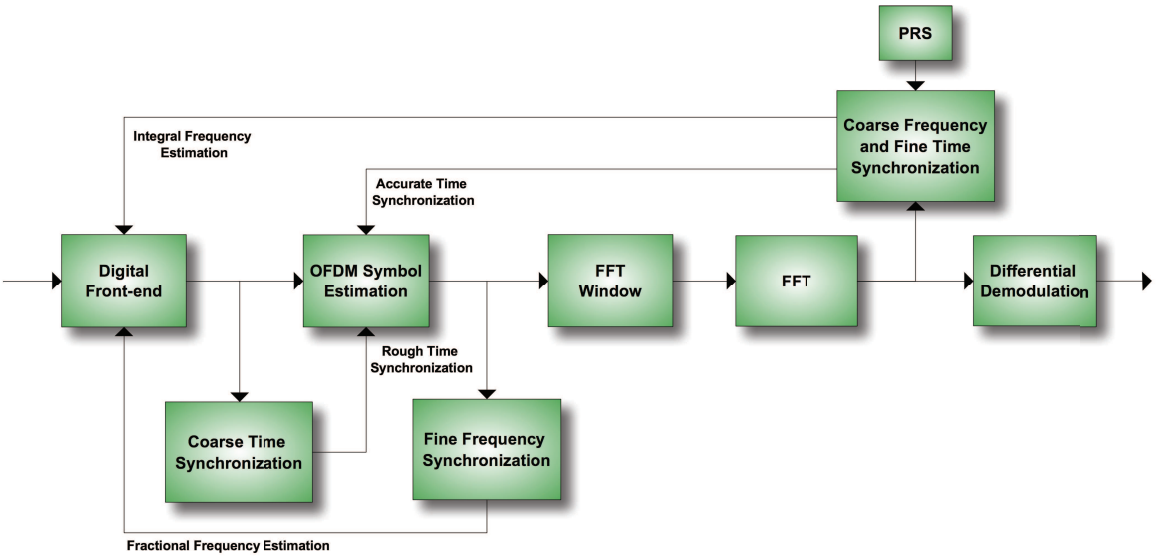


Figure 4.8: *Synchronization block diagram.*

3. Once the frame beginning is known, it is now possible to estimate the symbol interval and perform the remaining synchronization processes.

4. Thus, the cyclic prefix can be used to carry out fine time synchronization.

5. Following the chain, the FFT window is estimated and the FFT is evaluated.

6. Once in the frequency domain, coarse frequency and fine time synchronizations can be finally performed using the PRS. Fine time synchronization will allow an accurate timing estimation for the remaining symbols of the frame be correctly demodulated.

7. Both frequency offsets will be finally used in the digital front-end for frequency correction.

One should note that the difference between OFDM symbol and the FFT window estimations is the cyclic prefix. The former one is needed in order to perform fine frequency synchronization through the cyclic prefix.

Since the coarse frequency offset is mainly originated by front-end imperfections, its slowly changing conditions allow us to use the same frequency offset in upcoming frames. Once this offset changes, it will be detected by the respectively synchronisation block and added to the previous one. Then, this novel coarse frequency offset will be used for frequency correction of the current and upcoming frames.

The following subsections provide a detailed explanation of the several types of synchronization.

### Coarse Time Synchronization

At the start of the whole synchronization process, coarse frame synchronization is required to find the beginning of each frame. As many of the remaining blocks rely on the frame start, it is very important that its evaluation be fast and accurate.

In the DAB system, a Null symbol that has no energy or considerable less energy than the others symbols (noise) is placed at the beginning of each frame. Thus, the DAB frame timing can be easily estimated by detecting this discontinuity through a power estimation circuit. The block diagram describing the coarse time synchronization process is shown in Figure 4.9.

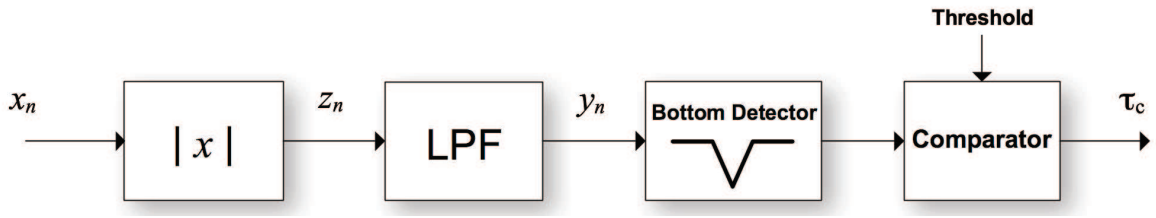


Figure 4.9: *Coarse time synchronization block diagram.*

Firstly, the envelope of the incoming RF signal is evaluated in order to estimate its energy. With that purpose, the module of the signal should be calculated as follows:

$$z_n = |x_n| = \sqrt{a_n^2 + b_n^2} \tag{4.9}$$

where $x_n = a_n + jb_n$. However, this block would be very slow and computationally intensive since it requires two multiplications and a square root operation. In order to simplify its implementation the following approximation was made:

$$z_n = |x_n| \approx |a_n| + |b_n| \tag{4.10}$$

Both possibilities were simulated in MATLAB and tested with real DAB baseband samples. The resulting signals can be seen in Figure 4.10. Both implementations simply have an

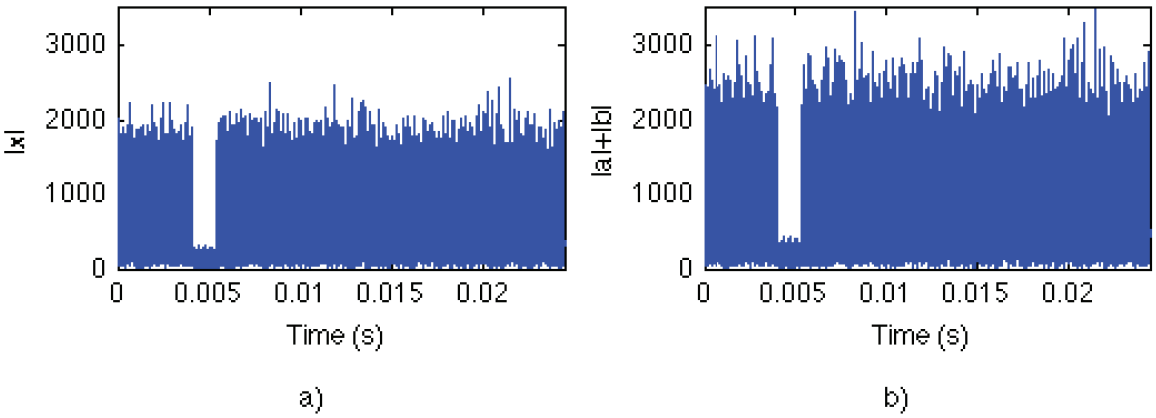


Figure 4.10: *Module of the DAB signal a) and its approximation b).*

amplitude difference. Later on, we will see that this difference can be ignored.

Once the module operation is finish, the signal envelope can be evaluated through a LPF. The following first order Infinite Impulse Response (IIR) filter was used with that purpose:

$$y_n = Ay_{n-1} + B(z_n + z_{n-1}) \tag{4.11}$$

In order to obtain the filter coefficients $A$ and $B$, the cutoff frequency $w_c$ should be firstly estimated. Thus, the following relation was used:

$$w_c = \frac{1}{\tau} \quad \Rightarrow \quad f_c = \frac{1}{2\pi\tau} \tag{4.12}$$

where $w_c$ is the cutoff frequency in radians per second, $f_c$ is the cutoff frequency in Hertz and $\tau$ is the time constant of this filter. To guarantee the detection of the Null symbol, its interval duration must be taken into account. Thus, we have:

$$T_{NULL} = 1ms \quad \Rightarrow \quad \tau < 1ms \tag{4.13}$$

where $T_{NULL}$ is the duration of the Null symbol in DAB mode I. Consequently:

$$f_c = \frac{1}{2\pi\tau} > \frac{1000}{2\pi} \approx 159Hz \tag{4.14}$$

A cutoff frequency $f_c$ of 1024 Hz was chosen, so the filter response shall be sufficiently fast to guarantee the Null symbol detection.

The filter was simulated in MATLAB and tested with both DAB signal modules. Figure 4.11 shows the resulting envelopes. Through the figure, it can be seen that the amplitude is

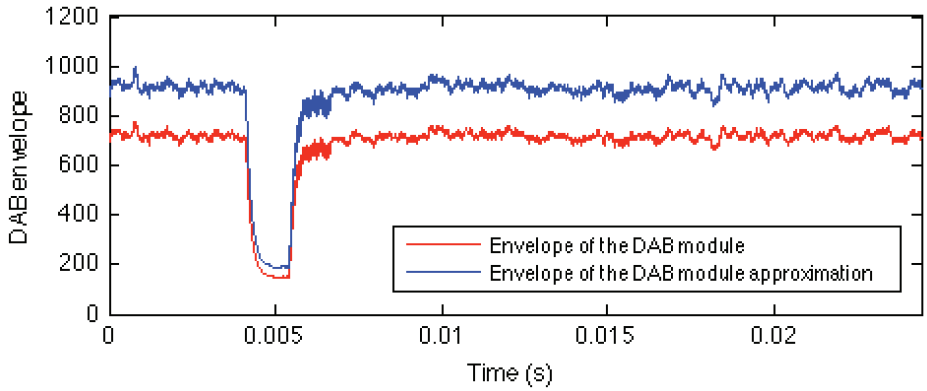


Figure 4.11: *Envelopes of the DAB signal module and its approximation.*

the only difference between the two envelopes. Thus, the peak detector and comparator can be adjusted to work with both signals. Since the operation that determines the DAB module is slower and more computationally exigent, its approximation is used.

The Null symbol can thus be detected by finding the minimum value of the envelope within an observation window smaller than the DAB frame duration, which is 96 ms for DAB mode I (since we are using a sampling frequency of 2048 MHz it corresponds to 196608 samples).

Finally, the start of the frame can be estimated if the received signal power rises above a predetermined threshold. Therefore, a histogram of the signal was evaluated in order to choose the most appropriate threshold. The histogram can be seen in Figure 4.12.

The histogram was calculated over a DAB envelope interval that includes the Null symbol and two times the duration of its duration for each side. It is easily understandable that the transition between the Null symbol ($y \simeq$200) and the useful part of the DAB signal ($y \simeq$900) is relatively slow. If we add the fact that this time synchronization is only the first coarse

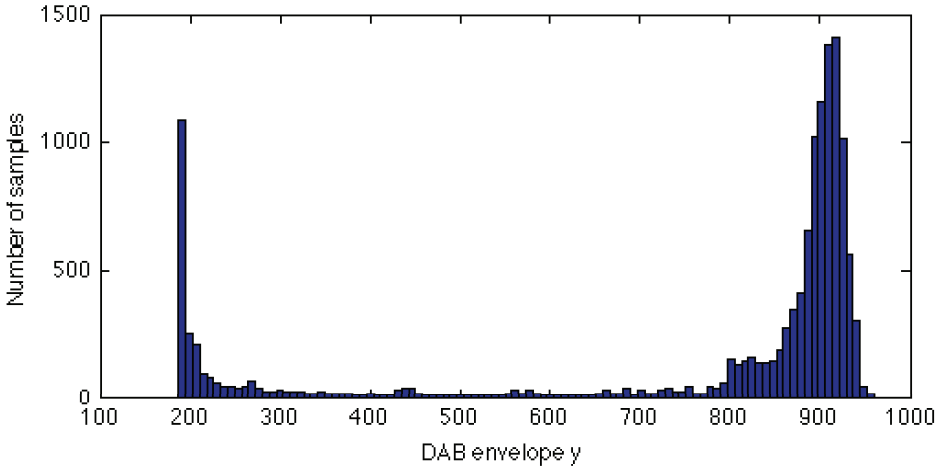


Figure 4.12: *DAB signal histogram.*

estimation and it only needs to guarantee detection inside the cyclic prefix (504 samples), the threshold can be roughly estimated.

However, it is important to note that the amplitude of the Null symbol, i.e., of the noise, depends on the channel conditions. Thus, the threshold was implemented in order to change with these conditions:

$$Threshold = y_{min} + (y_{mean} - y_{min})/I \tag{4.15}$$

where $y_{min}$ and $y_{mean}$ are, respectively, the minimum and the mean values of the DAB envelope. Since our intention is to detect the beginning of the cyclic prefix, the threshold must be closed to the Null symbol. So, in order to obtain a small interval of $(y_{max} - y_{mean})/I$, the variable $I$ was asserted through simulation. Figure 4.13 shows the envelope of a real DAB signal and its respectively threshold.

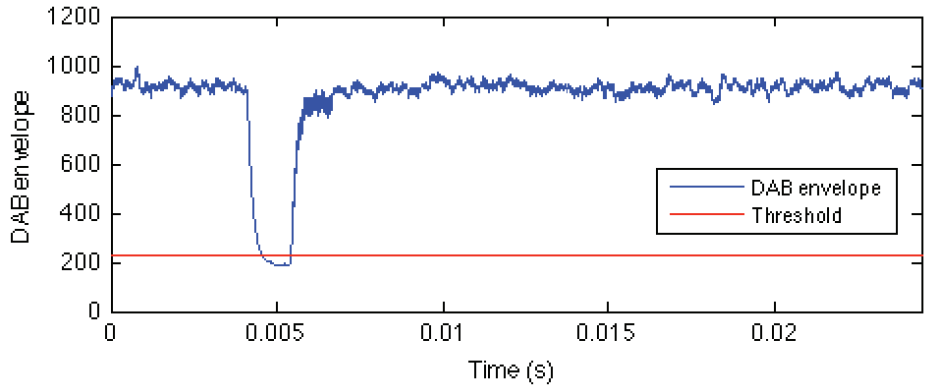


Figure 4.13: *Envelope of the DAB signal and coarse time synchronization threshold.*

The second connection between the threshold and the DAB envelope corresponds to the sample $\tau_c$ which states the desired beginning of the frame. Note that this is only a coarse estimation of the beginning of the frame so the remaining synchronizations can be performed.

Further on, a fine time synchronization shall be made in order to find an accurate start of the frame.

**Fine Frequency Synchronization**

Fine frequency synchronization estimates the fractional part of the frequency offset normalized to the subcarrier spacing, i.e., offsets that are smaller than the subcarrier spacing. As derived in [HSH99], the fractional part of the frequency offset rotates the complex constellation and causes ICI that resembles Additive White Gaussian Noise (AWGN) at the FFT output. Accurate fine frequency correction must be therefore performed before OFDM reconstruction, especially in a signal with a low SNR.

In OFDM, to avoid ISE, a cyclic prefix is employed in the beginning of each symbol that is exactly the same as the backend interval of the symbol. As described in [HD03, HSH99, CCB+01, TTT+96], this cyclic prefix can be used to estimate the fractional frequency offset. Figure 4.14 illustrates the symbol and its cyclic prefix.

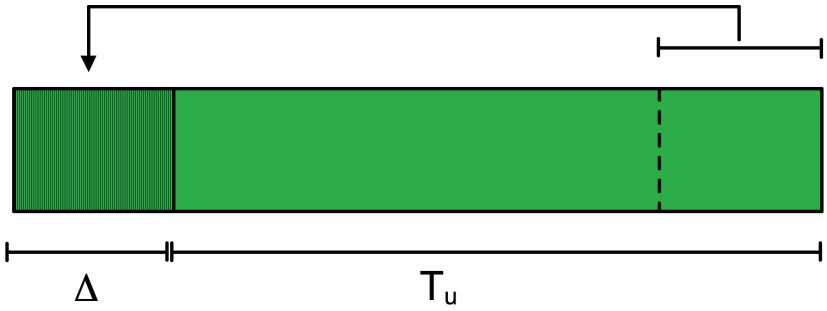


Figure 4.14: *DAB symbol and its cyclic prefix.*

As already mentioned in Chapter 4, $\Delta$ is the duration of the cyclic prefix and $T_u$ is the duration of an OFDM symbol as well as the reciprocal of the carrier spacing, that is $f_u = 1/T_u$. Assuming that $r(t)$ is cyclic prefix signal and $r(t - T_u)$ is the corresponding backend signal of the OFDM symbol, in ideal channel conditions:

$$r(t) = r(t - T_u) \quad t \in [0; \Delta] \tag{4.16}$$

If we let the fractional part of the frequency offset be $\delta_{fr} f_u$, there will be the following phase difference between the two intervals before the reconstruction of the OFDM symbol:

$$r(t) = r(t - T_u)e^{j2\pi\delta_{fr} f_u T_u} = r(t - T_u)e^{j2\pi\delta_{fr}} \quad t \in [0; \Delta] \tag{4.17}$$

For better understanding, the relation above is illustrated in Figure 4.15.

Once in the digital domain, this phase angle can thus be evaluated by doing the inner product between the cyclic prefix and the complex conjugate of its corresponding useful part of the symbol. Knowing that $r_n e^{-j2\pi\delta_{fr} n/N}$ is the $nth$ sample of the received OFDM symbol, we have the following inner product:

$$\sum_{n=125}^{380} r_n e^{-j2\pi\delta_{fr} n/N} \times r^*_{N-n} e^{j2\pi\delta_{fr}(N-n)/N} = \sum_{n=125}^{380} C e^{j2\pi\delta_{fr}} \tag{4.18}$$

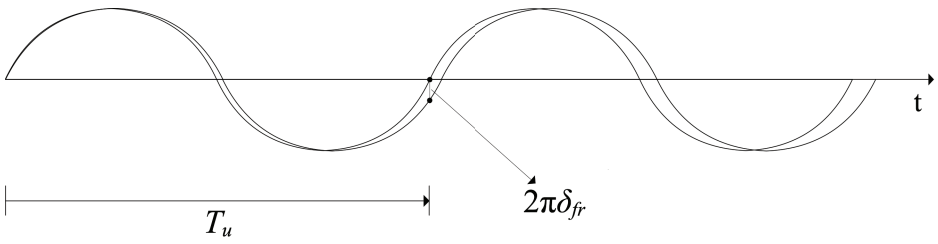


Figure 4.15: *Fractional frequency offset Illustration.*

where $C$ is a real number equal to the square of the amplitude of the OFDM symbol and $N$ is the FFT size (2048) [CCB+01]. In order to avoid timing errors, the inner product is only performed over the 256 samples in the middle of the cyclic prefix. The fractional part of the frequency offset can finally be estimated from the phase angle as follow:

$$\delta_{fr} = \frac{1}{2\pi} arg \left\{ \sum_{n=125}^{380} C e^{j2\pi\delta_{fr}} \right\} \tag{4.19}$$

where arg{} is an operator that extracts the phase term inside the braces.

The fine time synchronization algorithm is depicted in Figure 4.16.

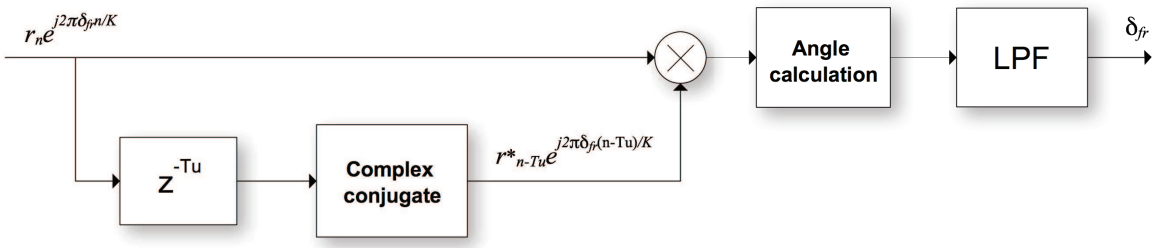


Figure 4.16: *Fine frequency synchronization block diagram.*

The fractional frequency offset that was previously estimated is applied to the received DAB signal in the digital front-end. Further on, the remaining fractional frequency offset is estimated and averaged with the previous one through a simple IIR filter. The process is also performed among frames.

### Coarse Frequency Synchronization

Coarse frequency synchronization estimates the integral part of the frequency offset normalized to the subcarrier spacing, i.e., offsets that are multiples of the subcarrier spacing. As derived in [HSH99], the effect of this frequency offset is a cyclic shift of the FFT output. For example, the FFT output is shifted by 1 when the frequency offset lies between 0.5 and 1.5, which occurs in the counterclockwise or clockwise direction depending on the signal [HSH99]. Hence, the subcarriers are moved to the wrong "teeth" of the comb and interpreted as the wrong symbols. Since a single shift would usually result in complete loss of the information, coarse frequency synchronization must be done with high accuracy.

When estimating the integral frequency offset the most common used techniques are based on the PRS, as can be find in [HD03, HSH99, CCB+01, TTT+96]. Since the PRS has a

property that turn it orthogonal to itself with any integral frequency offset [Gan03], coarse frequency synchronization can be based on a matched filter. When using this technique, the received PRS is matched to several integral frequency shifted PRSs generated in the receiver. The one that most closely matches the received PRS produces a peak in its output, while the others produce noise-like outputs [HSH99]. The coarse frequency synchronization process is described in Figure 4.17.

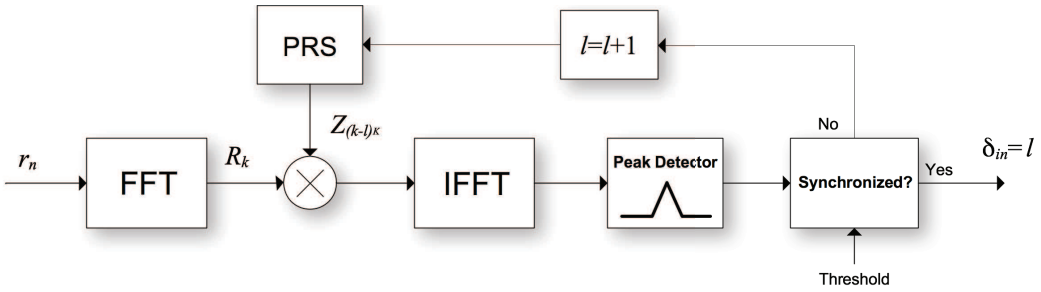


Figure 4.17: *Coarse frequency synchronization block diagram.*

As depicted in Figure 4.17, the matched filter is not more than the cross-correlation between the received PRS and the several integral frequency shifted PRSs. In order to reduce computation complexity, this time domain cross-correlation is perform through multiplication in the frequency domain:

$$y(n) = \sum_t h(k)x^*(n+k) \overset{FourierTransform}{\Leftrightarrow} Y(jw) = H(jw)X^*(jw) \tag{4.20}$$

Assuming that the integral frequency offset is $\delta_{in} f_u$ and the PRS is $z_k$ in the time domain, the received signal $r_k$ will be equal to $z_k e^{-j2\pi\delta_{in}}$. The received PRS can be reconstructed at the FFT output as follows:

$$R_k = \sum_{n=0}^{N-1} r_n e^{-j2\pi kn/N} = \sum_{n=0}^{N-1} z_n e^{-j2\pi\delta_{in}} e^{-j2\pi kn/N} = Z_{(k-\delta_{in})_N} \quad k = 0, 1, ..., N-1 \tag{4.21}$$

where $N$ is the FFT size and $()_K$ stands for modulo $K$. Through the above expression one can see that the received PRS is reconstructed as a shifted version of the known PRS. If we multiply the reconstructed symbol with several versions of the know PRS shifted by different $l$'s, the correlation can be thus obtained at the IFFT output as follows:

$$corr_n = \sum_{k=0}^{N-1} R_k Z^*_{(k-l)_N} e^{j2\pi nk/N} = \sum_{k=0}^{N-1} Z_{(k-\delta_{in})_N} Z^*_{(k-l)_N} e^{-2\pi nk/N} \quad n = 0, 1, ..., N-1 \tag{4.22}$$

When $l = \delta_{in}$, the cross-correlation between the two symbols is high and the IFFT output resembles an impulse [CCB+01]. Thus, the amount of shifts $l$ necessary to obtain this impulse is the integer frequency offset normalized to the subcarrier spacing. Figure 4.18 shows the process concept.

In order to determine whether the system is in synchronization or not, the IFFT peak output is compared with a predetermined threshold. Unlike the coarse time synchronization threshold, the current one does not require to be noise adaptable. This fact can be explained

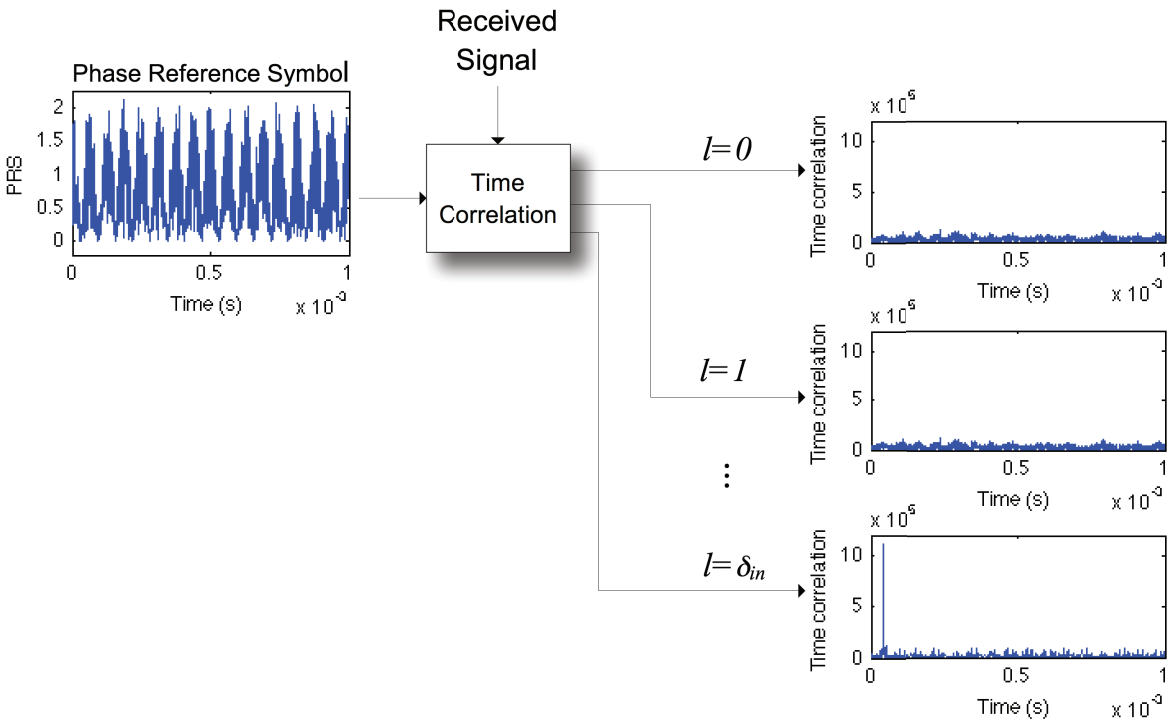


Figure 4.18: *Coarse frequency synchronization illustration.*

through the linear proprieties of correlation. The previous cross-correlation can be written as follow when in the time domain:

$$corr_n = \sum_{k=0}^{N-1} r_k z_{n+k}^* \tag{4.23}$$

In ideal channel conditions the received OFDM symbol $r_k$ is equal to the stored PRS $z_n$, so the expression will resemble an impulse for $k$=0:

$$n = 0 \quad \Rightarrow \quad corr_0 = \sum_{k=0}^{N-1} r_k z_k^* = \sum_{k=0}^{N-1} z_k z_k^* = \sum_{k=0}^{N-1} r_k^2 = E_z \tag{4.24}$$

However, in real channel conditions the receiver OFDM symbol has a noise component $n_k$, i.e., $r_k = n_k + z_k$. Thus, the following cross-correlation will take place:

$$corr_n = \sum_{k=0}^{N-1} r_k z_{n+k}^* = \sum_{k=0}^{N-1} (z_k + n_k) z_{n+k}^* = \sum_{k=0}^{N-1} z_k z_{n+k}^* + \sum_{k=0}^{N-1} n_k z_{n+k}^* \tag{4.25}$$

Due to its linear proprieties, the cross-correlation will be made up of two independent components: the PRS and the noise. Therefore, the desired impulse amplitude $E_z$ will be constant for every amount of noise, while the difference between this amplitude and the noise amplitude will be variable. So, the current threshold must depend on the signal energy instead the noise.

Thus, to determine the threshold according to the signal energy, the mean value of the envelope that was evaluated in the coarse time synchronization will be used as follows:

$$Threshold = I \times y_{mean} \tag{4.26}$$

Since our intention is to detect the correlation impulse, the threshold must be as closed to the peak as possible. So, the factor $I$ was asserted through simulation. If the several correlations between the signal and all possible shifted PRSs do not produce any peak higher than the $Threshold$, the highest peak is used for synchronization.

The MATLAB simulation searches for the highest peak by covering all the possible shifted PRSs. However, in a real DAB receiver the number of integral shifts will only cover the range of frequency uncertainty of the oscillators used in the transmitter and receiver.

The integral frequency offset that is firstly estimated is subsequently applied to the incoming DAB signal in the digital front-end. Since this frequency offset is mainly originated by oscillator's instability, its slowly changing conditions allow us to correct the upcoming frames and, therefore, to reduce the synchronization time. Further on, coarse time synchronization will be always performed since it is also used for fine time synchronization (see next subsection). If the integral frequency offset changes, it will be immediately detected and the frequency offset value that is stored in the digital front-end will be updated.

**Fine Time Sync**

Fine time synchronization accurately estimates the timing offset. The effect of this offset is the rotation of the OFDM symbols constellation, which can normally be ignored due to the differential modulation. Although, in the presence of multipath channels most suitable timing control is required in order to minimize the deterioration of receiver performance.

Assuming that $\tau_f$ is the timing offset which remains from the coarse time synchronization and the stored PRS is $z_n$ in the time domain, the received signal $r_n$ will be:

$$r_n = z_{n-\tau_f} \tag{4.27}$$

i.e., there are a time domain shift between both signals. Thus, the OFDM symbol $R_k$ can be reconstructed at the FFT output as follows:

$$R_k = \sum_{n=0}^{N-1} r_n e^{-j2\pi kn/N} = \sum_{n=0}^{N-1} z_{n-\tau_f} e^{-j2\pi kn/N} = Z_k e^{-j2\pi\tau_f k/N} \quad k = 0, 1, ..., N-1 \tag{4.28}$$

From the expression above, one can see that $kth$ subcarrier of the PRS constellation is rotated by $-2\pi\tau_f k/N$. Since coarse frequency synchronization was already done, the same algorithm of the last subsection can be used to estimate the timing offset $\tau_f$. The cross-correlation can thus be performed through a multiplication in the frequency domain and a subsequent IFFT:

$$\begin{aligned} corr_n &= IFFT\{R_k Z_k^*\} = \sum_{k=0}^{N-1} R_k Z_k^* e^{j2\pi nk/N} = \\ &= \sum_{k=0}^{N-1} Z_k e^{-j2\pi\tau_f k/N} Z_k^* e^{j2\pi nk/N} = \sum_{k=0}^{N-1} Z_k^2 e^{j2\pi k(n-\tau_f)/N} \end{aligned} \quad n = 0, 1, ..., K-1 \tag{4.29}$$

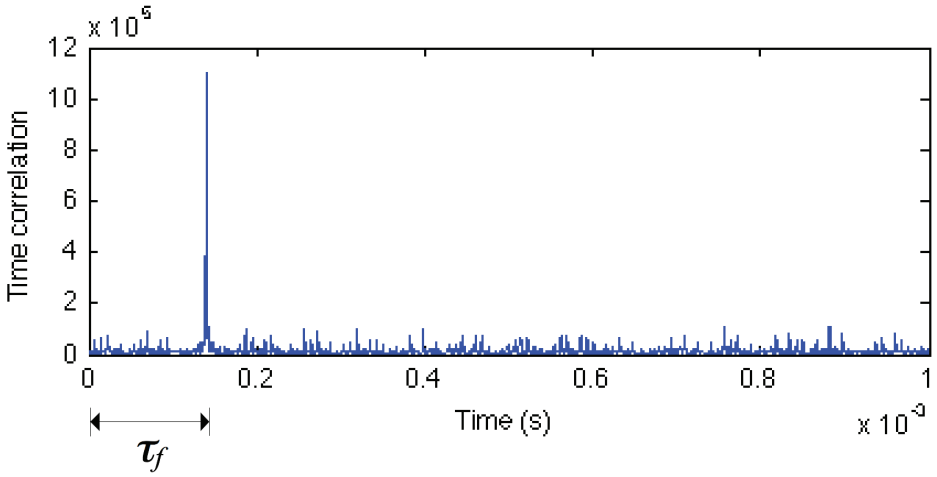


Figure 4.19: *Fine time synchronization illustration.*

So the impulse signal appears precisely at the start of the effective symbol duration, the timing offset can thus be found by examining the position of the impulse signal contained in the IFFT output [TTT+96]. Figure 4.19 shows an example.

Fine time synchronization is performed in the beginning of every frame.

### Frequency Correction

Finally, the received signal frequency must be corrected based on the estimated frequency offset. This can be done in both analog and digital domains as depicted in Figures 4.20 and 4.21.

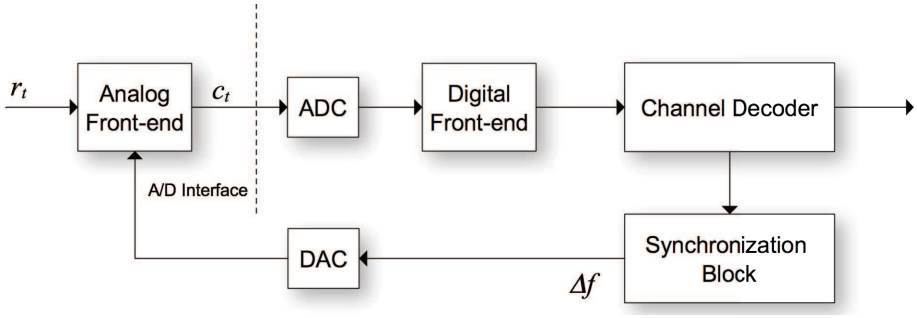


Figure 4.20: *Analog frequency correction.*

The analog correction adjusts the local-oscillator to make the carriers accordant. This method requires the local oscillator to have high precision, high stability and to be adjustable [HD03]. Since our main purpose is to work with SDR, the alternative digital correction was chosen. It is by far more flexible and do not require any special oscillator.

The principle of digital correction is to counteract the frequency offset before OFDM demodulation by means of digital signal processing. This is done as follows:

$$c_n = r_n \times e^{-2\pi n \Delta_f / N} \tag{4.30}$$

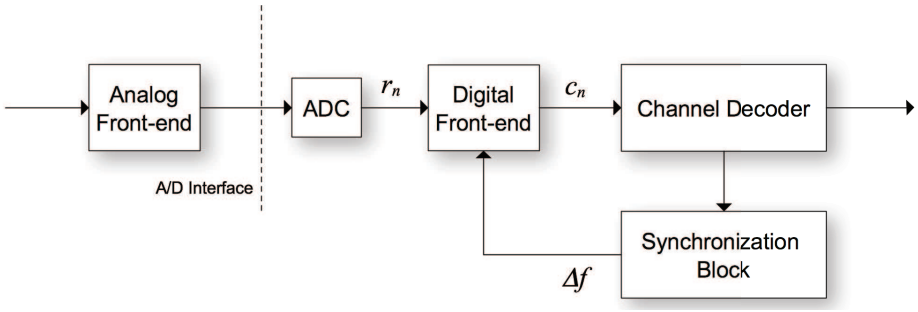


Figure 4.21: *Digital frequency correction.*

where $c_n$ is the corrected signal, $r_n$ is the received signal and $\Delta_f$ is the estimated frequency offset normalized to the subcarrier spacing. This complex multiplication must be applied to each sample of the received signal.

While MATLAB simulation of the frequency correction block is relatively straightforward, its FPGA implementation is one of the most complex and computationally exigent blocks. Therefore, special care should be taken when implementing this block.

### Sampling Frequency Estimation and Correction

Once time and frequency synchronization were already developed, a plot of the QPSK symbols should show four separate clouds with symbols belonging to the four constellation points. However, this was not the case as one can see in Figure 4.22.

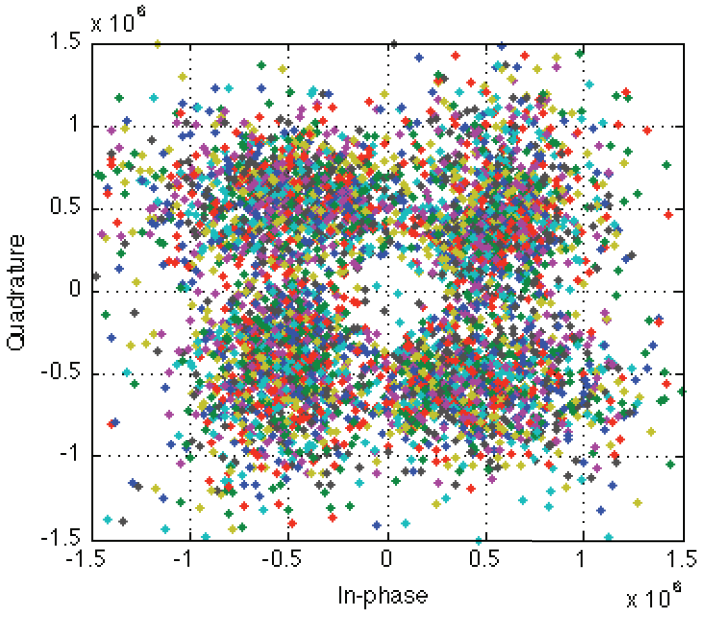


Figure 4.22: *QPSK symbols constellation of three OFDM symbols.*

Although the decoded QPSK symbols actually belong to four different clouds, this are unified through the decision limits, i.e., the axis. This constellation will originate more errors and, therefore, a deterioration of receiver performance. In order to overcome this

problem, more plots with different parameters were made. Figure 4.23 shown the QPSK symbol constellations of three different subcarriers: the central carrier and two carriers of opposite extremes. The three plots clearly show a phase difference among the several carriers.

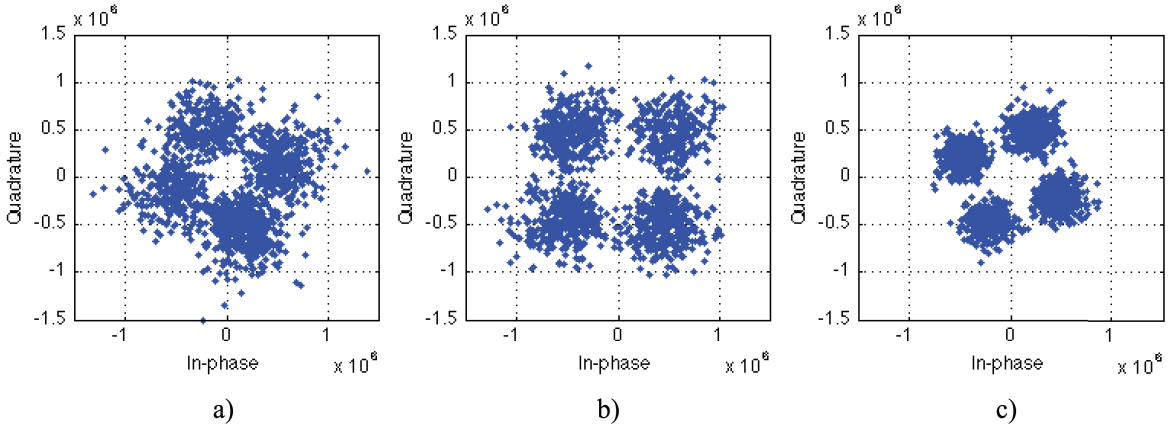


Figure 4.23: *QPSK symbols constellations of three different subcarriers: the* 36th *carrier a), the* 768th *carrier b) and the* 1500th *carrier c).*

Thus, it let us to assume that there might be a frequency dependent phase offset introduced by an inaccurate sampling rate. The plot displayed in Figure 4.24, which shows several symbols phases from different subcarriers, confirms this assumption.

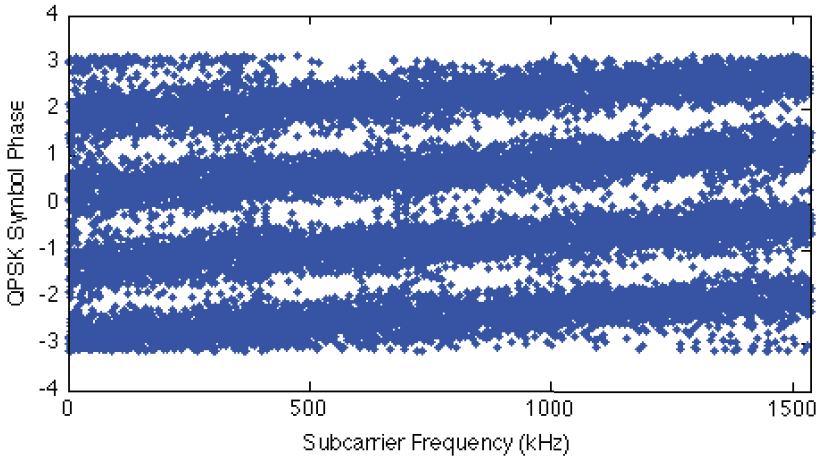


Figure 4.24: *QPSK symbol phase over the subcarrier frequency.*

As derived in [Sli01], a sampling frequency offset leads to a reduction of the amplitude of the symbols, a phase shift of each QPSK symbol that depends on its distance to the central carrier (easily seen through the Figures 4.23 and 4.24) and ICI due to the loss of orthogonality between subcarriers.

The first step toward sampling rate correction is its estimation. Thus, the approach used by Nicolas Alt in his DAB receiver was adopted with that purpose. Since every two PRSs are always interleaved by the same amount of OFDM symbols, the sampling rate can be easily estimated by looking at the length of the received frame. Figure 4.25 illustrates the sampling

frequency estimation process.

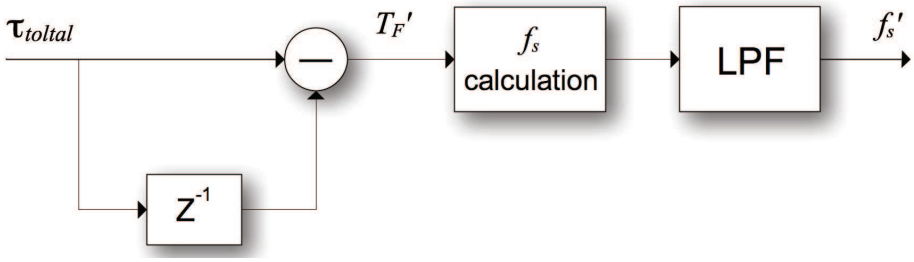


Figure 4.25: *Sampling frequency estimation block diagram.*

The sampling frequency estimation is based on the temporal information $\tau_{total}$, which is made up of the coarse and fine time synchronization outputs:

$$\tau_{total} = \tau_c + \tau_f \tag{4.31}$$

where $\tau_c$ is the coarse frame timing estimation and $\tau_f$ is the accurate one. Firstly, we must determine the length of the received frame:

$$T'_F(n) = \tau_{total}(n) - \tau_{total}(n-1) \tag{4.32}$$

where $T'_F$ is the duration of the received frame. The sampling frequency can be finally estimated according to the following expression:

$$f'_s(n) = T'_F(n) \times \frac{f_s}{T_F} \tag{4.33}$$

where $f'_s$ is the current sampling rate, $f_s$ is expected sampling rate (2048 KHz) and $T_F$ is the expected frame duration. Further on, the sampling rate is averaged over all frames through a simple IIR filter.

Once the actual sampling rate is known, it can be used to correct the incoming signal. This process can be done through several ways, such as subcarrier phase offset estimation and further correction, signal resampling or ADC sampling rate adjustment. The former one was used when simulating the system in MATLAB, whereas the last one will be probably used in the FPGA implementation.

When using the first method, the individual phase offset of each subcarrier can be easily estimated after OFDM symbol reconstruction as follows:

$$\theta_s = f_s \times T_s - f'_s \times T_s \tag{4.34}$$

where $T_s$ is the symbol duration. The correction of the several subcarriers phase offset can finally take place:

$$C_k = R_k \times e^{-j2\pi\theta_s k/N} \quad k = -N/2, ..., -1, 0, 1, ..., N/2 - 1 \tag{4.35}$$

where $C_n$ is the corrected OFDM symbol, $r_n$ is the reconstructed OFDM symbol and $N$ is the FFT size. Note that the original carriers' frequency symmetry must be respect.

However, when the actual sampling rate differs more than 1Hz from the expected one, i.e.:

$$|\Delta f_s| = |f_s - f'_s| < 1Hz \tag{4.36}$$

the FFT size might be changed to $f_s/f_u$ where $f_u$ is carrier spacing. A further fine correction can be performed through subcarrier phase offset estimation and correction, as it was explained above.

Although the previous algorithm was used when simulating the system in MATLAB, it is not feasible when using a FPGA device. This device does not allow us to change the FFT size and the subcarrier phase offset correction is very slow and computationally intensive. Since we are using a platform that includes ADCs and the System Generator allows us to change their sampling rate, its real-time adjustment will be probably our implementation choice.

## 4.5 Results

Once the DAB receiver has already been designed, developed and simulated, one must take the next step and test it with several channel conditions. With that purpose, two different kinds of DAB samples were used: MATLAB generated samples and real DAB samples.

In order to generate DAB samples for testing, a DAB transmitter was fully developed in MATLAB. Since we knew the original bit stream, the Bit Error Ratio (BER) performance of the receiver could be easily evaluated for different channel conditions. Thus, the *channel* function of Daniel Albuquerque [Alb09] and some additionally channel conditions were used to simulate different channel environments, such as frequency and time shifts, multipath and sampling frequency errors.

Further on, the real DAB samples of Nicolas Alt were also used for testing. Since we do not know the original bit stream, two approaches were made. In the first one, the DAB signal was demodulated and the evaluated constellations were analyzed. While in the second one, the demodulated bit stream was fed into the MSC and FIC decoders of Nicolas Alt receiver in order to listen the audio channels carried in that stream.

The next subsection describes the DAB transmitter implementation, while the last two subsections present the several results obtained with both MATLAB and real DAB samples.

### 4.5.1 DAB Transmitter

To allow BER performance study of the DAB receiver, a DAB transmitter was developed. As the transmitter needs no synchronization or frequency correction, this implementation is quite straightforward and was done in a reduced amount of time than the receiver. Its block diagram can be seen in Figure 4.26.

The DAB transmitter signal chain is the same of the receiver, but in the reverse order. Thus, the bit stream is firstly QPSK mapped to be subsequently split up in several vectors corresponding to an OFDM symbol each. Then, those vectors are frequency interleaved, differentially coded and OFDM modulated as described in Chapter 3. As already explained, the DAB signal spectrum will be shifted 1537 KHz to the right in order to have no negative frequencies. Therefore, our *1537th* carrier will correspond to the zero frequency of Chapter 3. By this way, the FFT from MATLAB can be used to modulate the OFDM symbol. At the end, the several 76 OFDM symbols are converted from parallel to serial, the Null symbol is

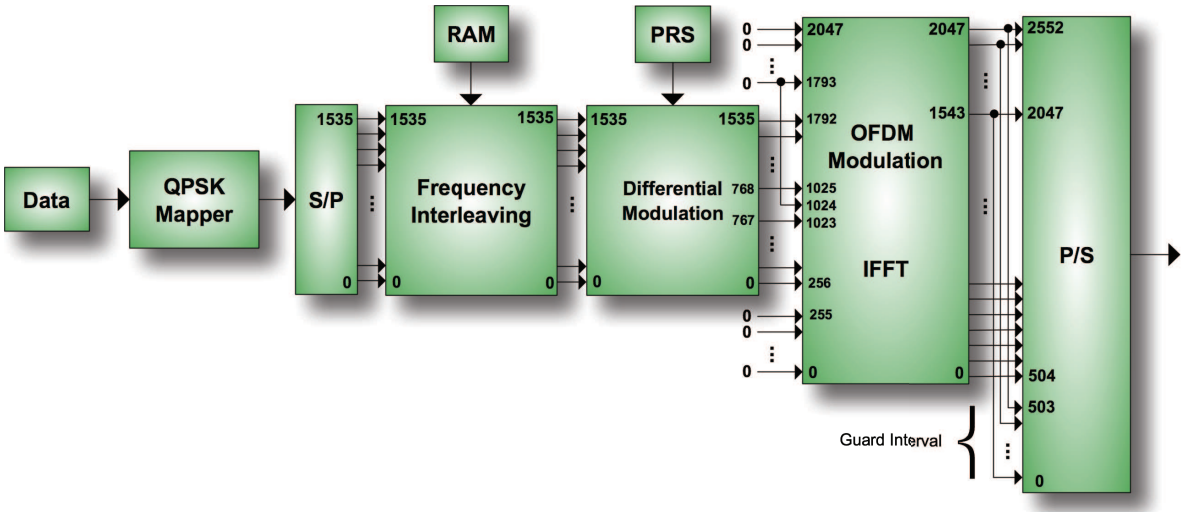


Figure 4.26: *DAB transmitter block diagram.*

added and the signal is ready to be "transmitted". A DAB frame in both time and frequency domain can be seen in Figures 4.27 and 4.28, respectively.

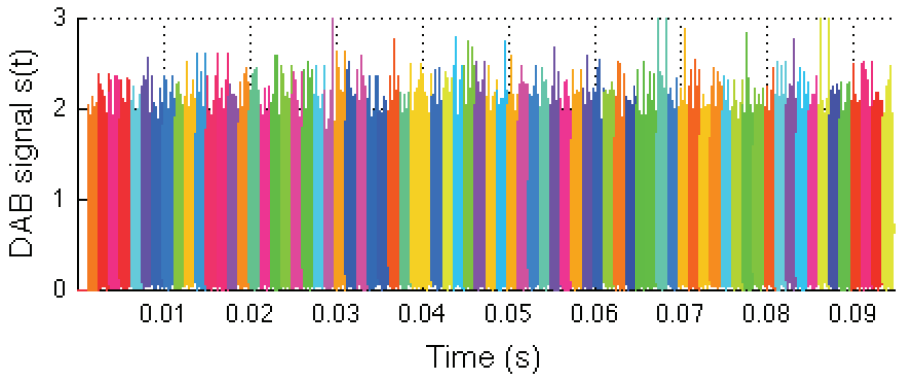


Figure 4.27: *DAB frame in the time domain.*

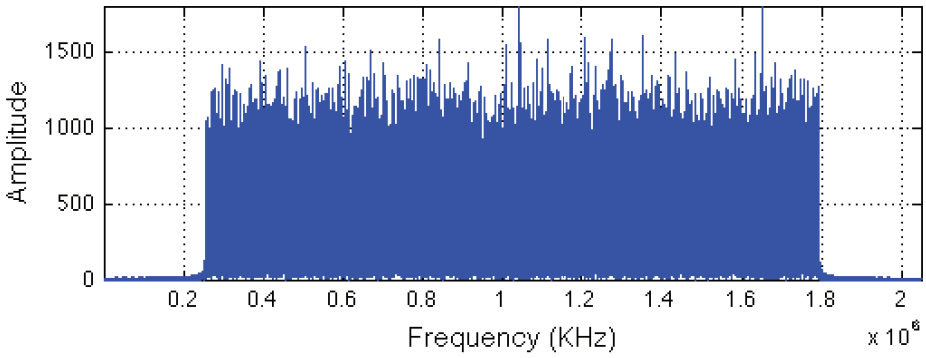


Figure 4.28: *DAB frame in the frequency domain.*

In the first figure, the different OFDM symbols are represent through different colours. Thus, one can see that a DAB transmission Mode I frame is made up of 77 symbols. The first symbol is the Null symbol and carries no information, the second is the PRS and the remaining 75 carry the several FIC and MSC services.

Figures 4.28 shows the DAB frame baseband frequency spectrum. Through that figure, one can roughly see that only the 1536 middle carriers carry information. Moreover, the zero frequency, in this case the *1537th* carrier, carries no information.

### 4.5.2 Channel Simulation

In this section, several channel conditions are simulated in order to perform BER analyses. Thus, the DAB transmitter and receiver, connected through several channel models, are wrapped up in a test bench. This test bench generates random bits, feds them though transmitter, channel and receiver and estimates the BER. Figure 4.29 illustrates the test bench block diagram.

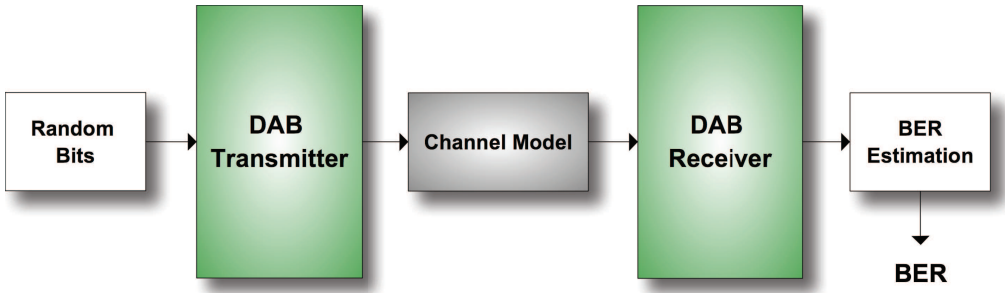


Figure 4.29: *Test bench block diagram.*

The channel models are either the *channel* function implemented by Daniel Albuquerque or the *channel_sim* function implemented during this thesis. Thus, the following channel conditions are simulated:

**AWGN -** modeled by a noise source and an adder

**Time Offset -** modeled by introducing zeros on the beginning of each vector

**Carrier frequency offset -** modeled by a sine source and a multiplier

**Sampling frequency offset -** modeled by a fractional interpolator

**Multipath propagation -** modeled by a simple Finite Impulse Response (FIR) filter

The first three are implemented through Daniel's function with some additional DAB specific modifications, while the last two are implemented through the *channel_sim* function. The time offset does not model any channel condition, but only demonstrate the performance of the synchronization algorithm on frame time offset estimation. One should note that these imperfections are modeled in a channel, but in practice they may be introduced in the transmitter or receiver.

### Additive White Gaussian Noise

In this subsection, the BER performance of the DAB receiver will be analyzed for an AWGN channel. With that purpose, a specific amount of AWGN must be added to the DAB signal in order to change its Symbol Energy to Noise Ratio (Eb/No) in each simulation. Therefore, this DAB specific amount of noise must be firstly derived.

As already mentioned, the DAB receiver uses an $\pi/4$ - QPSK modulation with symbol's amplitude $A$ and duration $T_b$ constants. Thus, the Symbol Energy to Noise Ratio (Es/No) of the DAB receiver can be expressed as:

$$\frac{Es}{No} = \frac{A^2 T_b}{No} \tag{4.37}$$

In order to derive Eb/No, we will apply to the following relation:

$$\frac{Es}{No} = \frac{Eb}{No} R_c log_2(M) \tag{4.38}$$

where $R_c$ is the code rate and $M$ is the size of the code alphabet. Since the $\pi/4$ - QPSK modulation transmits 2 bits per each symbol interval, it is a rate - 1/2 code and uses $M$=4 levels per symbol [Cou06]. Thus, the Eb/No can be obtain as:

$$\frac{Eb}{No} = \frac{Es/No}{R_c log_2(M)} = \frac{Es/No}{\frac{1}{2} log_2(4)} = \frac{Es}{No} = \frac{A^2 T_b}{No} \tag{4.39}$$

Next, the noise energy $No$ should be also derived. With this purpose, [Hay00] propose the receiver model represented in Figure 4.30.

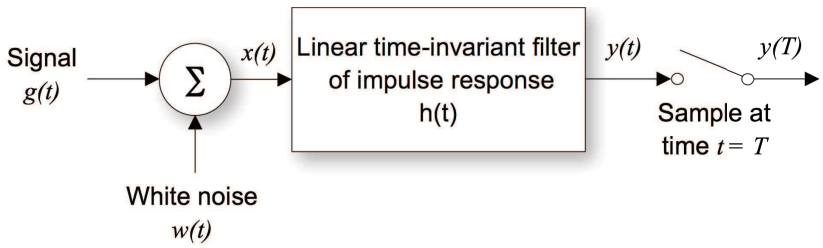


Figure 4.30: *Receiver model. [Hay00]*

The receiver consists of a linear time-invariant filter of impulse response $h(t)$. The filter input $x(t)$ is made of a pulse signal $g(t)$ corrupted by additive channel noise $w(t)$:

$$x(t) = g(t) + w(t) \quad 0 \leq t \leq T \tag{4.40}$$

where $T$ is an arbitrary observation interval. The pulse signal $g(t)$ may be a binary symbol "1" or "0", while $w(t)$ is the sample function of a white noise process of zero mean and power spectral density $No/2$. The linear output $y(t)$ of the filter may be expressed as:

$$y(t) = g_o(t) + n(t) \tag{4.41}$$

where $g_o(t)$ and $n(t)$ are produced by the signal and noise components of the input $x(t)$, respectively. Since the function of the receiver is to maximize the output signal component

$g_o(t)$ as much as possible, the filter design must be optimized in some way so the effects of the noise are minimized. The optimum filter was therefore derived in [Hay00]. The evaluated filter impulse response corresponds to the time-reversed and delayed version of the impulse signal, i.e., it is "matched" to the input signal. The linear time-invariant filter derived is therefore called a matched filter. When this optimum filter is matched to a rectangular pulse of amplitude $A$ and duration $T_b$, it can be shown that the variance of the white noise $w(t)$ is given by:

$$\sigma^2 = \frac{No}{2T_b} \tag{4.42}$$

In order to apply the previous result to the DAB receiver, we will make use of the OFDM proprieties. Since the several carriers are orthogonal to each other, in ideal conditions there wont be interference among them. Therefore, we will match the receiver filter to the emitted $\pi/4$ - QPSK symbol in each carrier. Thus, substituting the previous equation into equation 4.39 yields:

$$\frac{Eb}{No} = \frac{A^2 T_b}{No} = \frac{A^2}{2\sigma^2} \tag{4.43}$$

with $A = 1$. Once knowing the variance of the noise, we can easily generate the correct amount of noise for a specific Eb/No, using the following MATLAB command:

$$AWGN = \sigma \times randn(size) + j \times \sigma \times randn(size); \tag{4.44}$$

where $\sigma^2$ is the AWGN variance and $randn()$ is a MATLAB function that returns an matrix of size $size$ containing pseudorandom values drawn from the standard normal distribution. The AWGN channel simulation was finally performed over 36 DAB frames. The results can be seen in Figure 4.31.

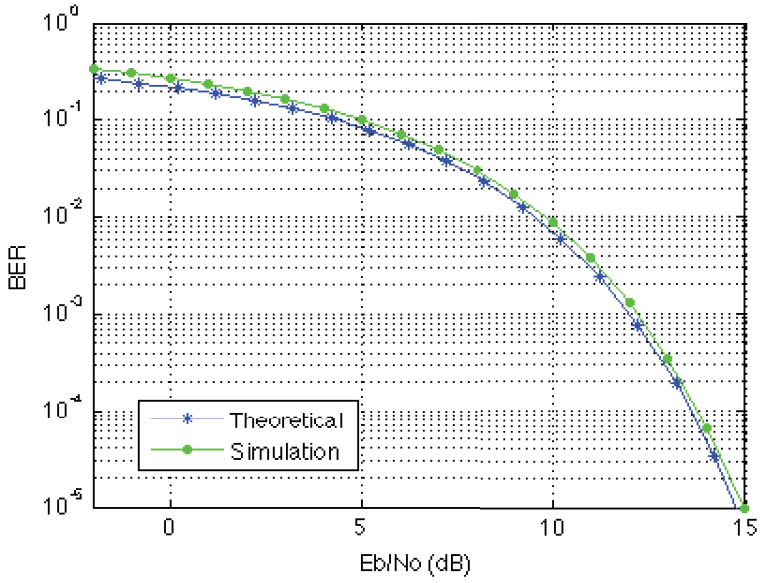


Figure 4.31: *BER performance in an AWGN channel (simulated with 1MB data per transmission).*

The theoretical BER curve was evaluated through several approximations. We started by using the theoretical BER of a QPSK modulation, that is:

$$BER_{QPSK} = Q\left(\sqrt{2\frac{Eb}{No}}\right) \tag{4.45}$$

In order to adapt the theoretical BER of a QPSK modulation to the specific DAB case, that curve must be shifted to the right. This occurs due to three main reasons:

1. For the same BER differentially detected $\pi/4$ - QPSK requires about 3 decibels more Eb/No than that for QPSK [Cou06]. Thus, the BER performance curve must be shifted by 3 decibels to the right.

2. For an OFDM modulation with cyclic prefix, all performance curves as functions of Eb/No or Es/No must be shifted by $10log_10(T/T_S)$ decibels to the right [SL05], where $T_s$ is the total symbol duration (cyclic prefix included) and $T$ is the active symbol duration.

3. The energy of the signal has a frequency bandwidth about 0.75 smaller than the noise bandwidth. Thus, the BER performance curve must be shifted by $10log_{10}(f/f_S)$ decibels to the right, where $f_s$ is the sampling rate and $f$ is the signal bandwith.

Summarizing, the DAB BER performance curve can be obtain by shifting the QPSK BER curve as follows:

$$\begin{aligned} BER_{DAB} &= BER_{QPSK}\left(\tfrac{Eb}{No} - (3 + 10log_10(T/T_S) + 10log_{10}(f/f_S))\right) \\ &\qquad\qquad with \quad T_s = T + \Delta \\ &\approx BER_{QPSK}\left(\tfrac{Eb}{No} - 5dB\right) \end{aligned} \tag{4.46}$$

where $\Delta$ is the cyclic prefix. With this 5 dB shift over the QPSK BER performance curve, one can see that the curve obtained through simulation is almost coincident with the theoretical one. The remaining difference can be originated by several factors. The first on concerns to the fact that the theoretical curve is just an approximation. Second, the DAB system includes two synchronization symbols that are not used to carry data and, therefore, consume power that also shifts the performance curves to the right. Furthermore, the synchronization process may also be affected by the noise and introduce further errors.

**Time Offset**

This subsection shall demonstrate the performance of the synchronization algorithm on frame timing offset estimation.

Thus, we will firstly show the effect of time offset on BER performance when no synchronization is performed. We define the time offset as negative when sampling time lags the cyclic prefix, and we define the time offset as positive when sampling time leads the cyclic prefix. The effect of several time offsets on BER performance in an AWGN channel is shown in Figure 4.32. Through the figure, one can see that as long as sampling time is within the cyclic prefix, there is no BER degradation as compared with the perfect timing case. On the other hand, if the initial sampling time is out of the cyclic prefix, even a few samples, performance degradation occurs. Moreover, negative and positive offsets with equal length

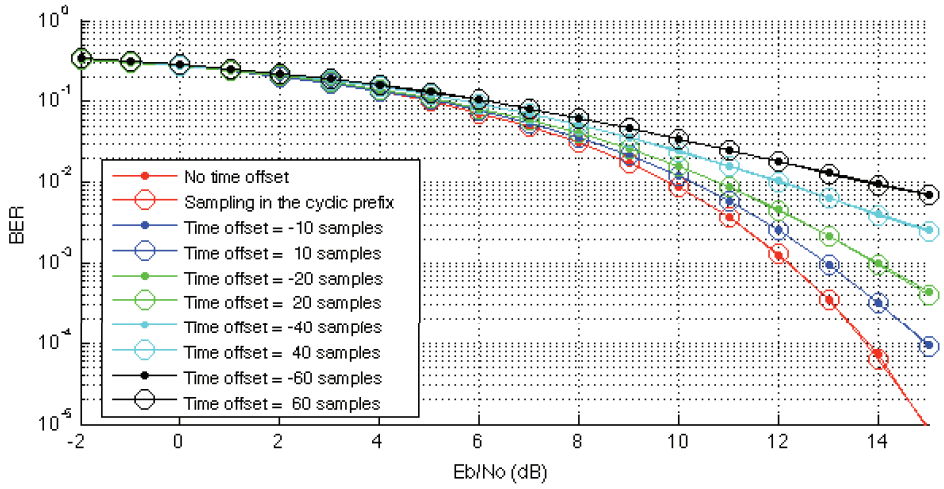


Figure 4.32: *BER versus time offset in an AWGN channel (simulated with 1MB data per transmission).*

have the same BER performance. This is easily understood since the same amount of ISI occurs, i.e., the interference with the previous or subsequent symbol is the same.

In order to demonstrate the performance of our time synchronization algorithm on time offset estimation, Table 4.1 shows the timing estimations for several frames in an AWGN channel with Eb/No of 8 dB. The results were averaged over 37 frames (1 MB). Coarse time

| | Frame 1 | Frame 2 | Frame 3 | Frame 4 | Frame 5 | Frame 6 |
|---|---|---|---|---|---|---|
| **Theoretical Value (samples)** | 395 873 | 592 481 | 789 089 | 985 697 | 1 182 305 | 1 378 913 |
| **Coarse Time Sync. (samples)** | 395 881 | 592 491 | 789 100 | 985 705 | 1 378 923 | 1 575 532 |
| **Fine Time Sync. (samples)** | 395 873 | 592 481 | 789 089 | 985 697 | 1 182 305 | 1 378 913 |

Table 4.1: *Frame timing estimation results in an AWGN channel (Eb/No = 8 dB).*

synchronization, as the name suggest, performs a rough frame timing estimation. However, its estimated value is still within a range of 100 samples around the frame beginning. If the receiver makes use of the cyclic prefix (504 samples), no performance degradation will occur. Moreover, coarse time synchronization is only used by the synchronization system. Fine time synchronization perfectly estimates the first sample of each frame, which guarantees a correct signal demodulation.

Table 4.2 contains the results of Frame 1 timing estimation for different AWGN channel conditions. When the Eb/No decreases the performance of fine and coarse time synchronizations decreases. Since both of them remain in the cyclic prefix, no BER degradation occurs.

| Time (samples) \ Eb/No (dB) | 2 | 4 | 6 | 8 | 10 | 12 |
|---|---|---|---|---|---|---|
| Coarse Time Sync. | 395 732 | 395 848 | 395 869 | 395 881 | 395 885 | 395 887 |
| Fine Time Sync. | 395 821 | 395 873 | 395 873 | 395 873 | 395 873 | 395 873 |

Table 4.2: *Frame 1 timing estimation results in several AWGN channel conditions.*

We can conclude that our synchronization algorithm performs well even when Eb/No is only 2dB.

### Carrier Frequency Offset

This subsection shall demonstrate the performance of the synchronization algorithm on carrier frequency offset estimation. Although a frequency offset is usually introduced by an inaccurate carrier frequency in the receiver, it can be modeled in the channel.

Firstly, we will show the importance of carrier frequency synchronization. Thus, the effect of different frequency offsets on BER performance in an AWGN channel is shown in Figure 4.33 when no synchronization is performed. Through the figure, one can conclude

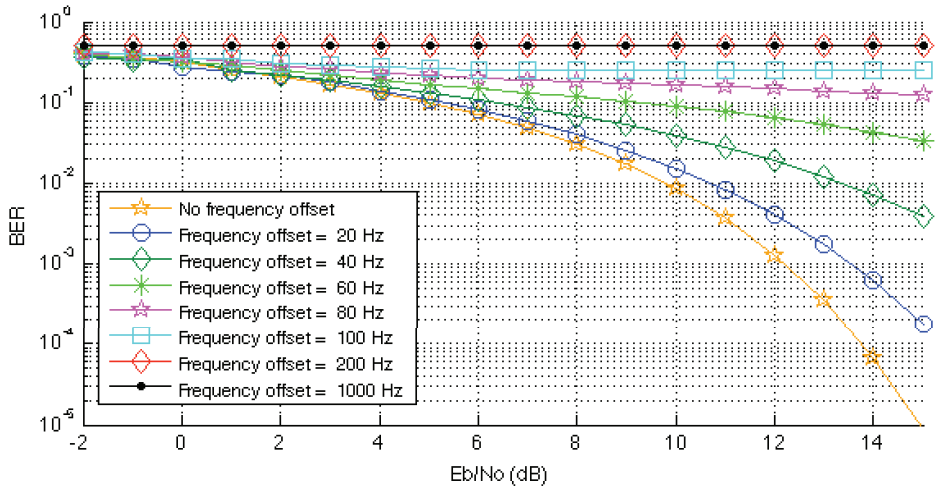


Figure 4.33: *BER versus carrier frequency offset in an AWGN channel (simulated with 1 MB data per transmission).*

that larger frequency offsets cause more BER degradation due to ICI. We can also observe that even small carrier frequency offsets have a dramatic effect on BER performance. When frequency offset approaches 200 Hz, i.e., 0.2 subcarrier spacing, the average BER approaches 0.5. Furthermore, the sign of the frequency offset does not influence the BER performance (not illustrated in the figure).

In the following, we investigate the effect of frequency offsets on BER performance when

the synchronization system is turned on. Thus, Table 4.3 shows the frequency estimations for different carrier frequency offsets in an AWGN channel with Eb/No of 8 dB. The results were averaged over 37 frames (1 MB). It can be observed that our algorithm achieves an estimation

| Frequency shift (Hz) | 400 | 800 | 1 200 | 1 600 | 2 000 | 20 000 |
|---|---|---|---|---|---|---|
| Fine Frequency Sync. | 399.90 | -200.04 | 200.13 | -399.95 | -0.13 | 0.03 |
| Coarse Frequency Sync. | 0 | 1000 | 1 000 | 2 000 | 2 000 | 20 000 |
| Total Estimation | 399.90 | 799.96 | 1 200.13 | 1 600.05 | 1 999.87 | 20 000.03 |

Table 4.3: *Frequency estimation results in an AWGN channel (Eb/No = 8 dB).*

accuracy always smaller than 0.01 of the subcarrier spacing (1 Hz).

Furthermore, simulations of the BER with different frequency offsets revealed that the BER gets quite high if the offset is close to half of the subcarrier spacing. This occurs due to the fact that fine frequency offset can either be corrected up or down to the nearest subcarrier. With an offset of half the subcarrier spacing, both are equally likely. Our algorithm solves this problem. Since we are using an IRR filter this will have a very light effect in the average frequency effect. Moreover, that offset is very unlikely to occur and if it might occur it is subsequently compensated in upcoming frames. However, if we set this frequency offset in the beginning of the simulation, it would have a great weight in the IRR filter and would take time to recover from it. Due to this fact this frequency offset is not shown in our simulation results. However, in a real environment where the demodulation time is much longer, the averaged domain would be much larger and it would work well.

Table 4.4 contains the results of carrier frequency estimation for different AWGN channel conditions for a carrier frequency offset of 1200 Hz.

| Eb/No (dB) / Frequency (Hz) | 2 | 4 | 6 | 8 | 10 | 12 |
|---|---|---|---|---|---|---|
| Fine Frequency Sync. | 200.29 | 199.93 | 200.20 | 200.13 | 199.92 | 200.06 |
| Coarse Frequency Sync. | 1 000 | 1 000 | 1 000 | 1 000 | 1 000 | 1 000 |
| Total Estimation | 1 200.29 | 1 199.93 | 1 200.20 | 1 200.13 | 1 199.92 | 1 200.06 |

Table 4.4: *Frequency estimation results in several AWGN channel conditions for a carrier frequency offset of 1200 Hz.*

We can observe that even when the Eb/No is only 2 dB, the synchronization algorithm achieves an estimation accuracy smaller than 0.01 of the subcarrier spacing (1 Hz).

Finally, the effect of our frequency synchronization algorithm on BER performance in an AWGN channel is shown in Figure 4.34.

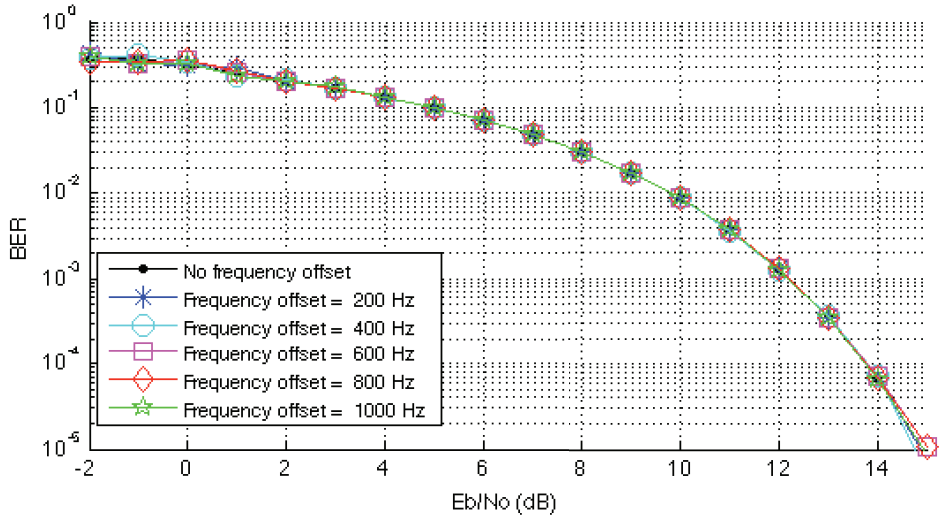


Figure 4.34: *BER performance with carrier frequency offset compensation in an AWGN channel. (simulated with 1 MB data per transmission).*

We can conclude that the BER performance with our synchronization algorithm almost coincides with the perfect synchronization case in an AWGN channel. The effectiveness of the developed synchronization algorithm can also be seen when comparing Figure 4.33 with Figure 4.34.

### Sampling Frequency Offset

This subsection shall demonstrate the performance of the developed synchronization algorithm on sampling frequency offset estimation and subsequent correction.

As in the previous sections, we will firstly show the effect of a sampling frequency offsets when no correction is performed. Figure 4.35 illustrates the simulation results in an AWGN channel for several sampling frequency offsets. We can observe that even small sampling frequency offsets have a dramatic effect on BER performance. Since crystal oscillators are often specified to an accuracy of around 50 ppm, this is critical.

Our algorithm solves this problem. Figure 4.36 shows the effect of our sampling frequency synchronization algorithm on BER performance in an AWGN channel. We can observe that sampling frequency offsets that are integral multiples of the subcarrier spacing have no BER degradation when compared with the perfect case. However, if the sampling frequency is a fractional part of the carrier spacing, some degradation occurs. This happens because both offsets are corrected through two different algorithms. Thus, the first one only changes the FFT size, while the second estimates the several carriers' offset in order to correct them. The latter one is only an approximation, but stills better than if no synchronization is performed. Furthermore, the sign of the frequency offset does not influence the BER performance.

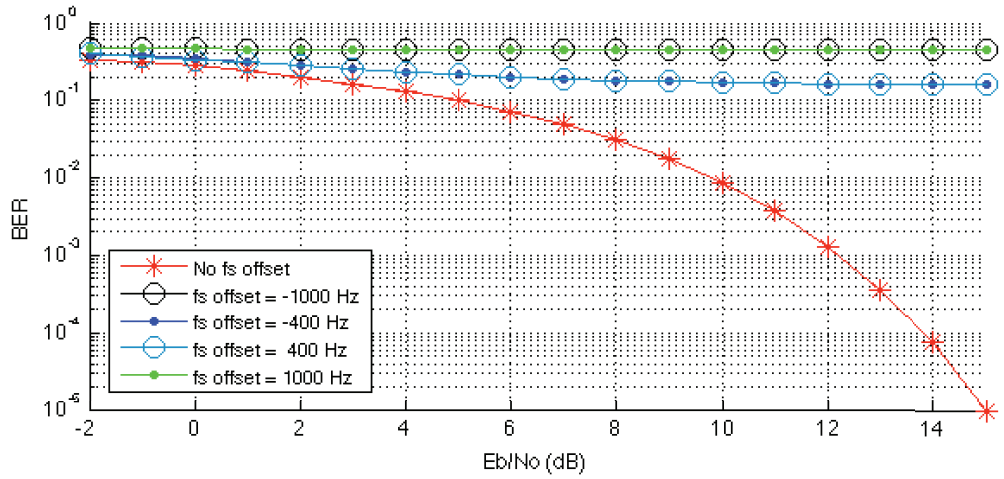


Figure 4.35: *BER versus sampling frequency offset in an AWGN channel (simulated with 1 MB data per transmission).*

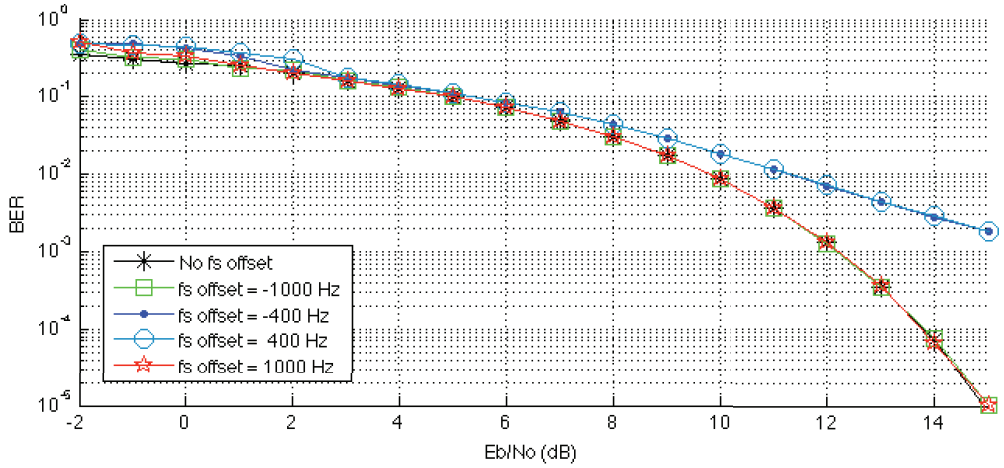


Figure 4.36: *BER performance with sampling frequency offset compensation in an AWGN channel (simulated with 1 MB data per transmission).*

---

### 4.5.3 Multipath Propagation

This subsection shall demonstrate the performance of the developed synchronization algorithm when multipath propagation occurs.

The effect of multipath propagation can be modeled with a FIR filter, which can model different echo delays. In this simulation, a single echo was modeled. Figure 4.37 shows the simulation results in an AWGN channel for several echo delays with half of the signal amplitude. This plot is particularly interesting, when the length of the cyclic prefix is kept in mind. We can observe that as soon as the delay is longer than the cyclic prefix (504 samples), the effect of an echo increases drastically.

Figure 4.38 shows the simulation results in an AWGN channel for several echo amplitudes relatively to the original signal, the echo delay is fixed at 400 samples.

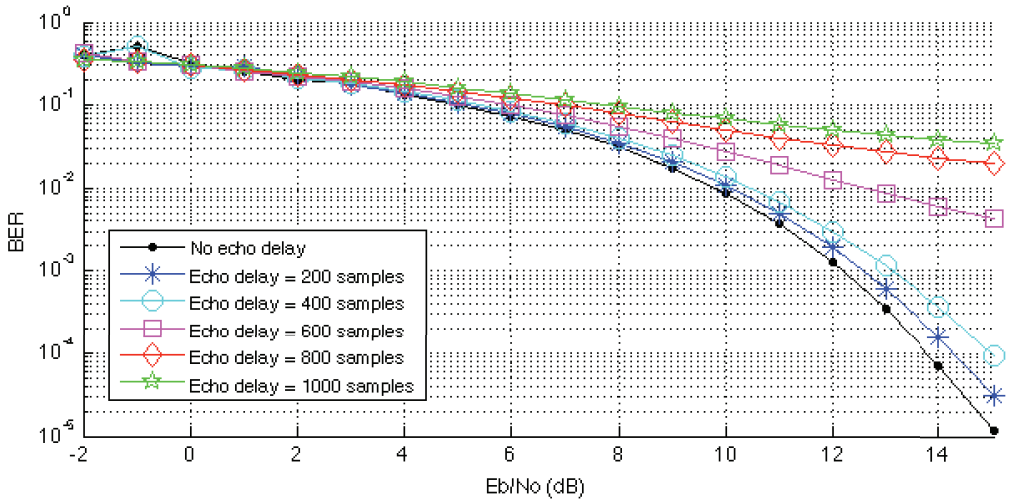


Figure 4.37: *BER versus echo delay in an AWGN channel (simulated with 1 MB data per transmission).*

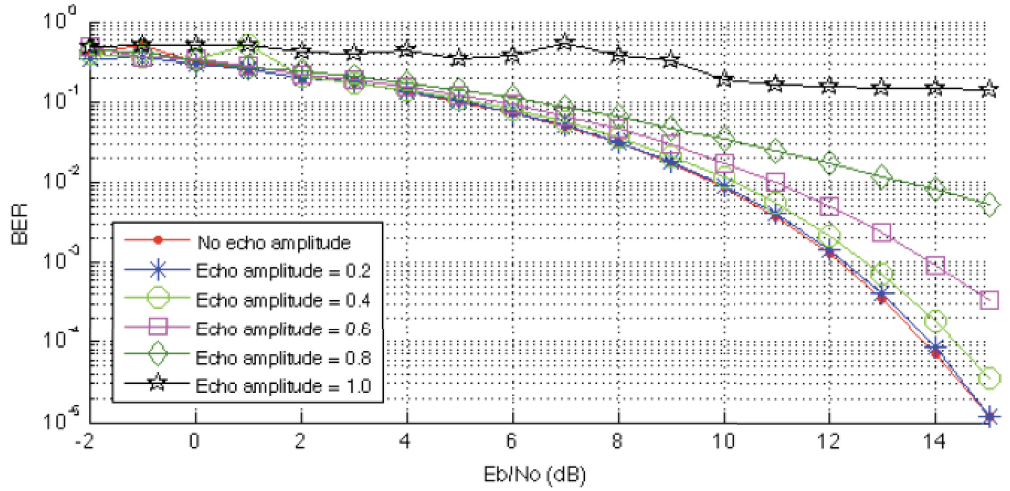


Figure 4.38: *BER versus echo amplitude in an AWGN channel (simulated with 1 MB data per transmission).*

---

As one might conclude, a delay of 400 samples should not have any effect on BER performance. However, fine frequency synchronization uses part of the cyclic prefix to estimate the fractional part of the frequency offset. Thus, if the echo delay is longer than half of the cyclic prefix this synchronization process will be corrupted.

We can finally conclude that echoes with amplitudes smaller than 0.2 have little effect on BER performance. As these amplitudes start to increase, BER degradation occurs. Moreover, when the echo arrives with the same amplitude of the signal, the receiver cannot demodulate any information (BER = 0.5).

### 4.5.4 Real DAB Signal

This subsection shall demonstrate the performance of the developed DAB receiver when demodulating a real DAB signal.

With that purpose, Nicolas Alt's DAB mode I samples were used. They were recorded with a Lyrtech SDR platform. Our synchronization algorithm showed the following signal features:

**Carrier frequency offset ≈ 74 290 Hz**
**Sampling frequency ≈ 2 047 846 Hz**
**Sampling frequency offset ≈ 75 ppm**

As one could understand through this chapter, the Lyrtech samples' main challenge was its sampling frequency correction. Figure 4.39 shows the demodulated signal constellation after sampling frequency correction.

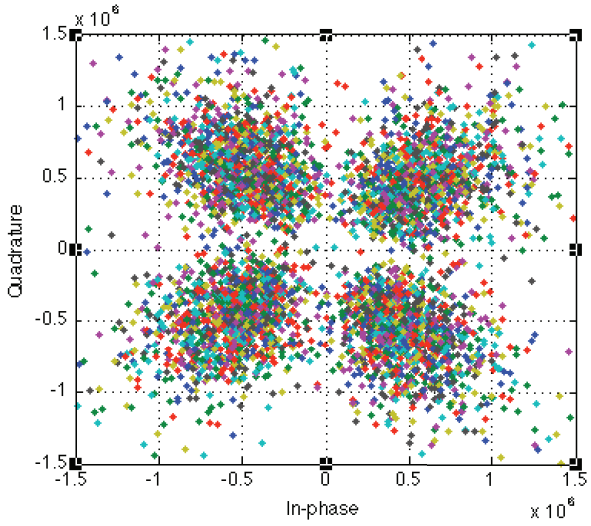


Figure 4.39: *QPSK symbols constellation of three OFDM symbols after sampling frequency correction.*

One can observe four separate clouds with QPSK symbols belonging to the four constellation points, as it was expected. For better understanding of the sampling frequency correction effectiveness, Figure 4.23 and 4.24 of Section 4.4.2 are reproduced here after synchronization correction. Figure 4.40 and 4.41 show both plots, respectively.

One can conclude that frequency correction is performing well, since the several constellations are not rotated anymore and the carriers' phase plot resembles four straight lines.

Moreover, when the demodulated bit stream was fed into the MSC and FIC decoders of Nicolas Alt receiver, one could perfectly listen the audio channels carried in that stream.

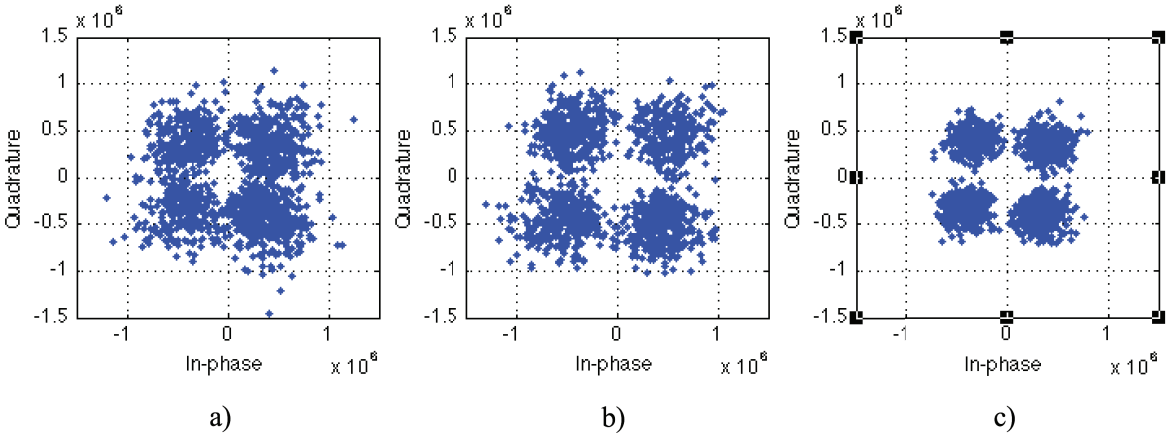


Figure 4.40: *QPSK symbols constellations of three different subcarriers after sampling frequency correction: the 36th carrier a), the 768th carrier b) and the 1500th carrier c).*

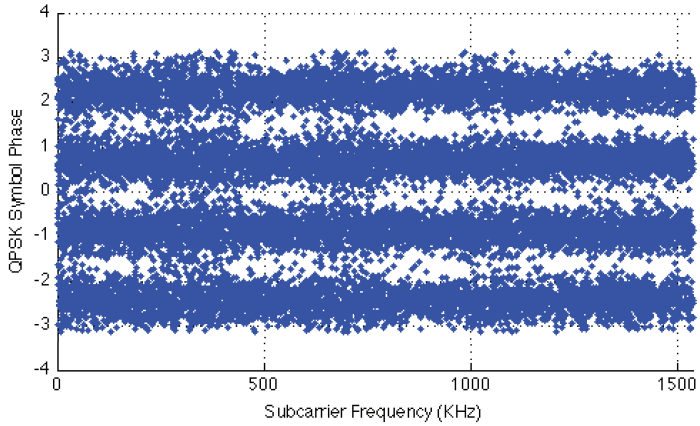


Figure 4.41: *QPSK symbol phase over the subcarrier frequency after sampling frequency correction.*

# Chapter 5

# Receiver Implementation in FPGA Technology

## 5.1 Introduction

In order to develop a real time SDR system, its design must be transferred into hardware. Hence, the simulated DAB receiver of the previously chapter should be implemented in one of the available technologic options. *Field programmable gate array (FPGA)* technology was the chosen one.

As previously mentioned in Chapter 2, the FPGA bridges the gap between DSPs and ASICs. It provides the flexibility and complexity of an ASIC but with the shorter turn-around time and reconfigurability of a programmable device. A FPGA is a logic device that contains a two-dimensional array of generic logic cells and programmable switches. The conceptual structure of an FPGA device is shown in Figure 5.1.

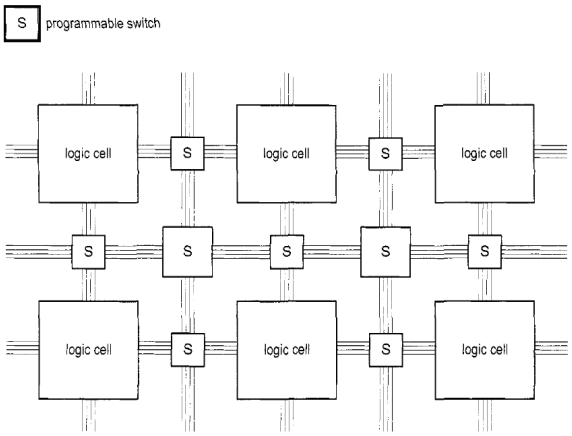


Figure 5.1: *Conceptual structure of an FPGA device [Chu08].*

A logic cell can be programmed to perform a simple function like a logic gate, or more complex combinational functions such as decoders or mathematical operations. The several programmable switches allow logic cells to be interconnected as needed by the system designer. The custom design can thus be implemented by specifying the function of each logic cell and

by setting the connection of each programmable switch. Once the design and synthesis are completed, one can transfer the desired configuration into the FPGA device and obtain the custom circuit. Since this process can be done "in the field" rather than "in a fabrication facility", these devices are known as *field programmable* [Chu08].

One should note that nowadays FPGAs are much more complex than the simplified conceptual architecture showed in Figure 5.1. Additionally resources, such as DSP48 and block memories, are provided to enable a wider range of systems.

Since creating extremely high-performance SDR systems can present significant challenges in both design complexity and time to market, specific platforms are available in order to provide complete development environments and make the designs faster and easier. Analog front-end's, FPGAs, DSPs, DACs and ADCs are examples of the components that can be included in these platforms. The XtremeDSP Development Kit-IV from Xilinx is one of those platforms and it was used in the scope of this work to implement the DAB receiver.

Developed by Nallatech, the XtremeDSP Development Kit-IV is a development platform for the Virtex-4 FPGA technology. The kit consists of a motherboard populated with a daughter card, which provides a dual channel, high performance ADCs and DACs, as well as a user programmable Virtex-4 device [Nal07]. In addition, the kit can also be combined with the *Xilinx System Generator for DSP* software tool. System Generator is a visual dataflow design environment based on MATLAB/Simulink's visual modeling tool set that provides an easy environment to develop high-performance signal processing systems. This programming interface allows the system developers to work at a suitable level of abstraction from the target hardware platform, and use the same model not only for simulation and verification, but for FPGA implementation as well [DH03]. Last but not least, the System Generator tool also interfaces with MATLAB/Simulink and enables us to perform hardware co-simulation on the XtremeDSP Development Platform via PCI or JTAG. This provides simulation acceleration and allows the debug and verification of the design on the FPGA [Xil06b]. Appendixes D and E give an overview of the several features of both the XtremeDSP Development Kit-IV and Xilinx System Generator tool, respectively.

Although the whole physical layer of the DAB receiver is intended to be implemented on the XtremeDSP Development Kit-IV, only part of the system was developed during this dissertation. The receiver block diagram of the previous chapter is reproduced in Figure 5.2 in order to identify which blocks are already implemented. One can see that the implemented

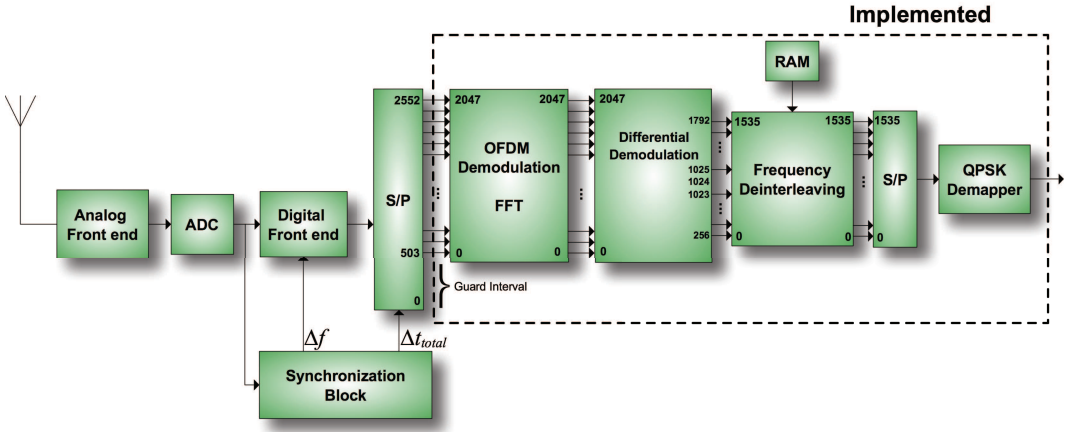


Figure 5.2: *General block diagram.*

blocks correspond to the several signal demodulation functions.

During the remaining chapter, one should not confuse DAB synchronization with FPGA signal control synchronization. DAB synchronization was not performed in the FPGA, but only simulated in MATLAB.

## 5.2 Signal Demodulation System

As previously mentioned, the signal demodulation system comprises all functions that are related to signal demodulation, which are OFDM demodulation, differential demodulation, frequency deinterleaving and QPSK demapper. Each of these functions was implemented through the System Generator tool in FPGA technology. Figure 5.3 shows the resulted parent system.

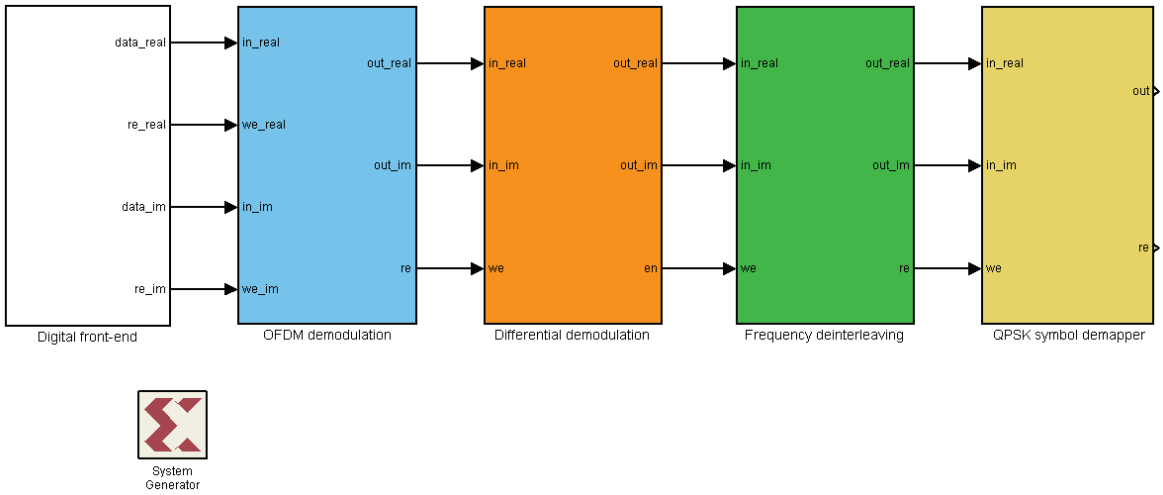


Figure 5.3: *System Generator parent system.*

The main system is divided in several subsystems that are represented in the Figure 5.3 through different colours. Each of these corresponds to one of the several signal demodulation functions. In addition, the white subsystem was developed in order to supply the data to be decoded. Thus, this subsystem simply consists of two different Read Only Memories (ROMs) for both In-phase (I) and Quadrature (Q) components of the complex data. The stored data is cyclically repeated. Since the digital front-end is the block behind the OFDM demodulation, its name was adopted by this subsystem.

The working frequency of the system was not chosen yet. However, the programmable clock sources on the board that drive the user FPGA clock nets are generated by PLL oscillators which have a minimum operation frequency, so the overall system must work at a higher frequency than this [Nal07]. Thus, the system was initially thinking to work at the lowest clock frequency provided, which is 16MHz. Nevertheless, this will depend on the frequency of the incoming signal and, consequently, on the analog front-end features.

Moreover, the ADC working frequency should be an integer multiple of 2.048 MHz, which is the DAB sampling frequency. Since none of the programmable clock frequencies fulfils this requirement, an onboard crystal oscillator which feds a clock signal directly to the Clock FPGA must be used to derive the differential clock signals used to clock the ADCs. The

crystal oscillator supplied has a low jitter characteristic and its speed can be matched to the sampling frequency of the ADCs [Nal07]. The derivation of this clock will be done in a future stage of the implementation.

The following subsections presents each of the signal demodulation subsystems, as well as explains the several options that were made when developing them.

### 5.2.1 OFDM Demodulation

Since the synchronization block was not implemented, we assume that the DAB signal is already temporally synchronized and frequency compensated. Thus, the digital front-end outcoming signal is directly fed into the OFDM demodulation block.

As explained in the last chapter, the demodulation of the OFDM symbols is based on the determination of the DAB carrier's complex amplitudes through the FFT. Since we have already seen that the working frequency of the DAB receiver will be probably greater than the sampling frequency (2.048 MHz), the incoming signal should be synchronized in order to correctly demodulate the OFDM symbols. Figure 5.4 shows the OFDM demodulation subsystem implemented in the System Generator.

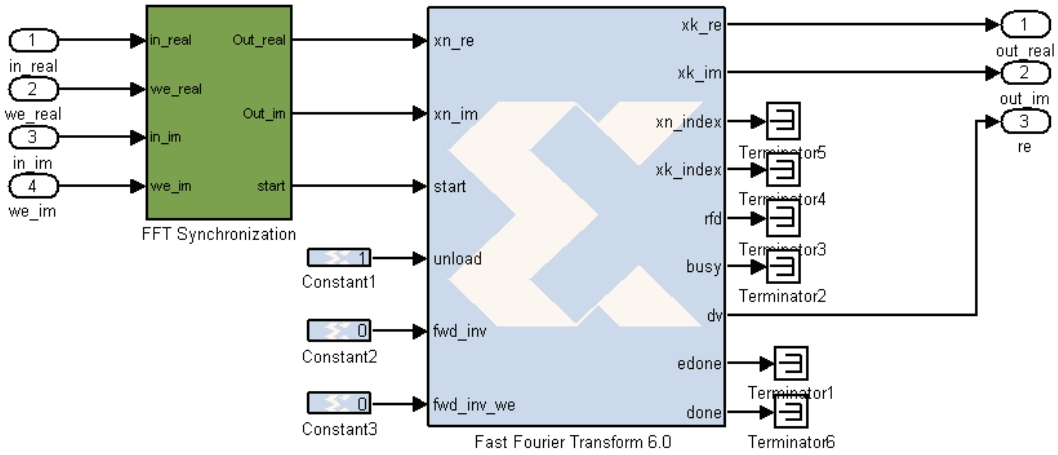


Figure 5.4: *OFDM demodulation subsystem in the System Generator*

The synchronization block is made up of two First-In/First-Out (FIFO) memories and a synchronization circuit based on a counter. The two FIFO accumulate the incoming real and imaginary complex components. The data is only read by the FFT when both FIFOs are completely filled, so the FFT can process the data at once. Hence, the FIFO should have the following size:

$$N_{FIFO} = \frac{f}{fs} \tag{5.1}$$

where $N_{FIFO}$ is the FIFO size, $f$ is the working frequency and $f_s$ is the sampling frequency. The FIFOs and the *start* output of the synchronization block are controlled through a cyclic counter.

The *Xilinx Fast Fourier Transform 6.0 block* is used to perform the FFT operation. This block implements a computationally efficient algorithm for computing the DFT of sample

sizes that are a positive integer power of 2. The DFT of a sequence is defined as follows:

$$x_k = Z_k e^{-j2\pi\tau_f} = \sum_{n=0}^{N-1} x_n e^{-j2\pi kn/N} \quad k = 0, 1, ..., N-1 \tag{5.2}$$

where $N$ is the transform size and $j = \sqrt{-1}$. The FFT core uses the *Cooley-Tukey algorithm* for computing the FFT, which breaks the DFT into smaller DFTs in order to reduce the computation time.

The FFT core provides four different architectures in order to offer a trade-off between core size and transform time (quote) [Xil08a].

- *Pipelined, Streaming I/O* - Allows continuous data processing.
- *Radix-4, Burst I/O* - Loads and processes data separately, using an iterative approach. It is smaller in size than the pipelined solution but has a longer transform time.
- *Radix-2, Burst I/O* - Uses the same iterative approach as Radix-4, but the butterfly is smaller. This means it is smaller in size than the Radix-4 solution, but the transform time is longer.
- *Radix-2 Lite Burst I/O* - Based on the Radix-2 architecture, this variant uses a time multiplexed approach to the butterfly for an even smaller butterfly, at the cost of longer transform time.

Since the working frequency of the system will be probably greater than the sampling frequency, the FFT will be easily executed in less than 1 ms as required. Thus, our concerns are more related with size than time. By this way, the *Radix-2 Burst I/O* architecture was chosen. This architecture uses the Radix-2 butterfly processing engine as represented in Figure 5.5.

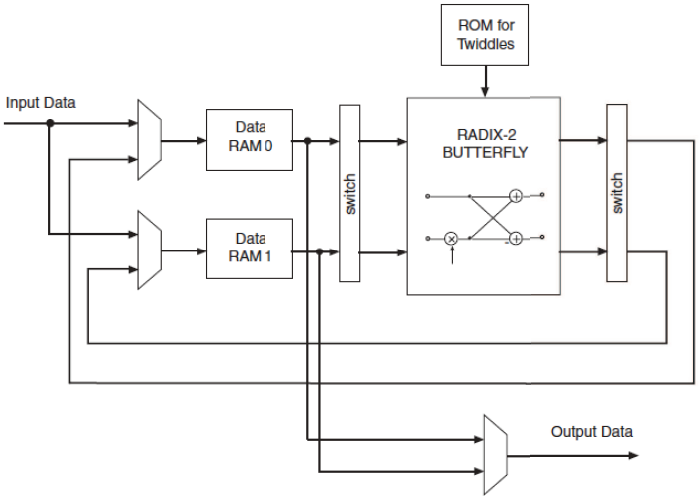


Figure 5.5: *Radix-2, Burst I/0 architecture [Xil08a].*

The $N$ - point FFT of a Radix-2 architecture consists of $log_2(N)$ stages, with each stage containing $N/2$ Radix-2 butterflies. This architecture does not allow the performance of

simultaneous operations, i.e., data input, FFT computation and data output. Thus, the data is loaded when the FFT is started. After a full frame has been loaded, the core computes the transform. When the computation finishes, the evaluated data is finally at the output ports [Xil08a].

Since we are using fixed-point inputs, the input data is a vector of $N$ complex values represented as two's-complement numbers. The real and imaginary components of the input data samples are loaded separately through the *xn_re* and *xn_im* ports, respectively. The two's-complement representation implies the previous normalization of the OFDM symbols, otherwise the input samples are interpreted incorrectly. The *start* port marks the beginning of each OFDM symbol. It was noticed that the input samples are only loaded three CPU cycles after the start signal be asserted. The several signal paths can be seen in Figure 5.6 and its temporal behaviours in Figure 5.6.

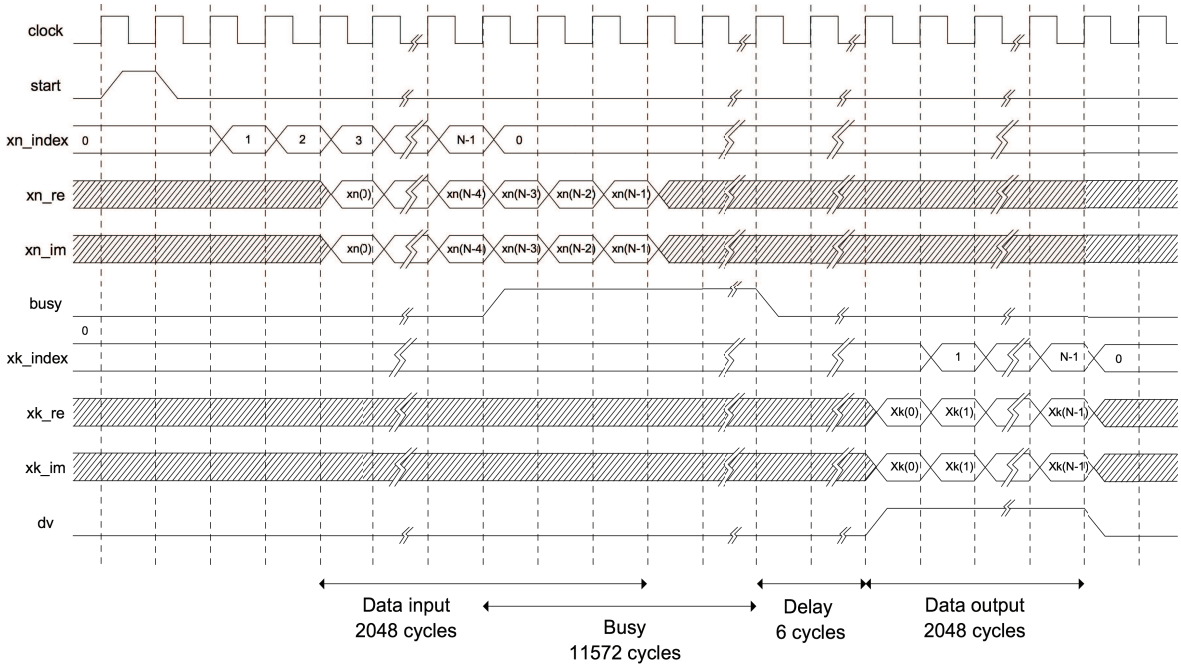


Figure 5.6: *Temporal behaviour of the FFT input and output signals.*

For a better understanding of the several steps of the FFT operation, Figure 5.7 shows the plots of the most important FFT control signals. One can see that its operation is divided in three main steps: data input, FFT computation and data output. Thus, the *xn_index* signal value indicates the number of the sample that is being loaded, the *busy* signal indicates that the FFT is being computed and, finally, the *xk_index* signal value indicates the number of the sample that is ready to read. Since the *dv* signal is set when valid data is at the output ports, it is used by the OFDM demodulation subsystem in order to indicate whether the output data is valid or not. The real and imaginary components of the demodulated signal are outputted in natural order through the *xk_re* and *xk_im* ports, respectively.

The FFT computation takes 13627 clock cycles to process each symbol. Thus, if the sampling frequency was the same as the system frequency operation, the OFDM demodulation subsystem would also take 15675 clock cycles. However, the system frequency operation will be probably greater, so the FFT has to wait for the incoming samples to perform the OFDM

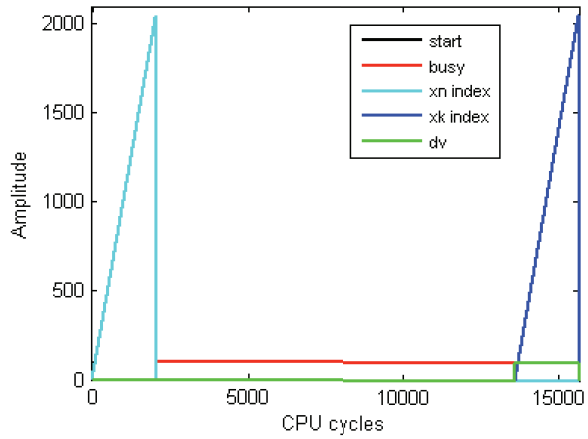


Figure 5.7: *FFT block control signals.*

symbol at once.

### 5.2.2 Differential Demodulation

Once the signal is OFDM demodulated, differential demodulation must take place. As mentioned in the previously chapter, differential demodulation is usually performed by complex multiplying each OFDM symbol with the stored complex conjugate amplitude of the previous symbol. That is:

$$Y_{l,k} = Z_{l,k} \times Z^*_{l-1,k} \quad k = 0, 1..., 1535 \tag{5.3}$$

where $Y_l$ is the demodulated symbol, $Z_l$ is the incoming symbol and $Z_{l-1}$ is the previous symbol. This complex multiplication can be done in both complex notations:

- In the polar notation by multiplying the amplitudes and adding the respective phases.
- In the rectangular notation by performing several multiplication between the real and imaginary components.

While the first option is computationally exigent because implies the determination of the phase and amplitude components, the second option implies more multiplications. However, since the upcoming QPSK demapper uses a rectangular notation, the second option was implemented. Assuming that $Z_{l,k} = a + bj$ and $Z_{l-1,k} = c + dj$, the complex multiplication will be implemented as follows:

$$Y_{l,k} = Z_{l,k} \times Z^*_{l-1,k} = (a + bj) \times (c - dj) = (ac + bd) + (cb + ad)j \tag{5.4}$$

Figure 5.8 shows the differential demodulation subsystem implemented in the System Generator. Through the figure, one can see that four FIFOs are used for both real and imaginary components of the incoming signal. The first two removes the unused carriers, while the second two store the previous symbol. In order to synchronize the system, control signals are highly necessary. Thus, a synchronization block was implemented to perform the following tasks:

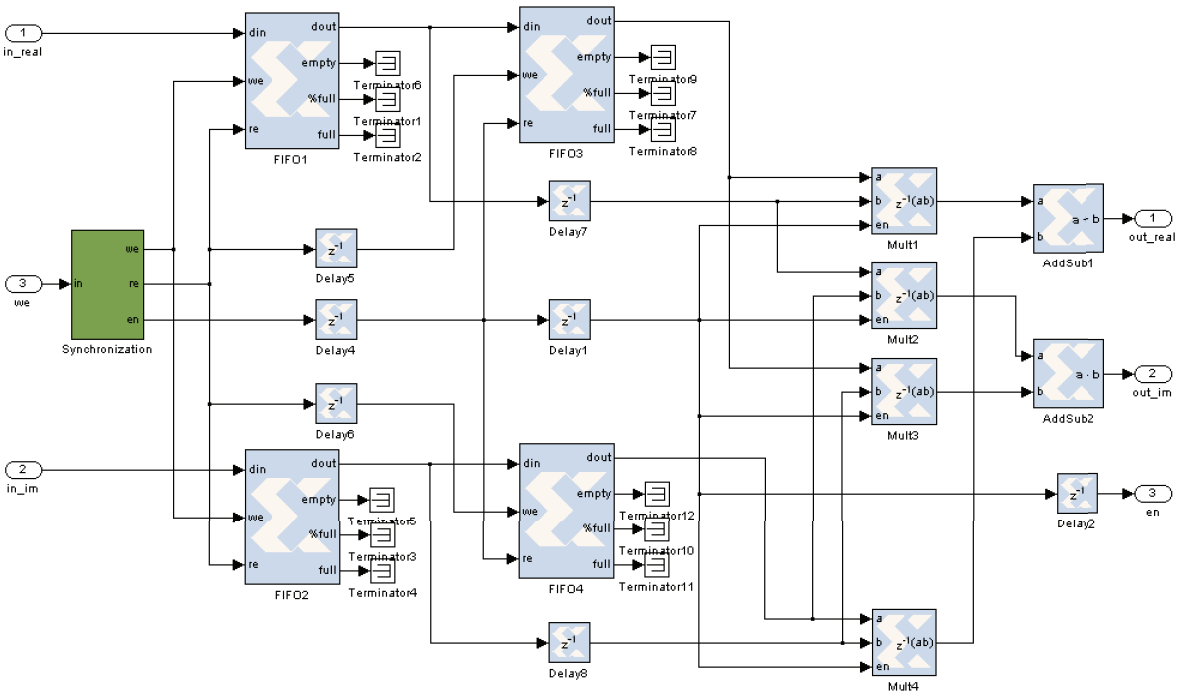


Figure 5.8: *Differential demodulation subsystem in the System Generator.*

---

- Decide whether the carriers are useful or not, which originates the write enable signals of the first two FIFOs
- Synchronize the multiplications, which originates the read enable signals of the first two FIFOs, the write and read enable signals of the last two FIFOs and the enable of the multiplications.

The first task is based on a counter, whose output $counter_{out}$ corresponds to each of the 2048 OFDM carriers. Thus, the write enable signals of the first two FIFOs are controlled through three comparators as follows:

$$\begin{cases} counter_{out} <= 255 & \Rightarrow \quad FIFO_{we} = 0 \\ 255 < counter_{out} < 1024 & \Rightarrow \quad FIFO_{we} = 1 \\ counter_{out} = 1024 & \Rightarrow \quad FIFO_{we} = 0 \\ 1204 <= counter_{out} < 1793 & \Rightarrow \quad FIFO_{we} = 1 \\ counter_{out} >= 1793 & \Rightarrow \quad FIFO_{we} = 0 \end{cases} \tag{5.5}$$

Note that the counter's initial value is 0, so the index of each carrier is equal to the index of the corresponding carrier of the previous chapter minus one.

The second task is based on a finite state machine with two sates. The control signals are generated in order to allow a continuous differential demodulated output, i.e., without interruptions originated by unuseful carriers. Thus, two FIFOs will be needed as will be seen.

As mentioned, the differential demodulation process is initialized in every frame by using the PRS. Thus, at the beginning of each frame the finite state machine is restarted. The two stats perform the following tasks:

**First state of the finite state machine:**

1. The PRS of each DAB frame is stored in the last two FIFOs. Since the first two FIFOs are in the path, the PRS samples have to pass through them.

2. When the PRS is fully stored, the finite state machine changes for the second state in order to differentially decode the remaining symbols.

**Second state of the finite state machine:**

1. The first three active samples of each incoming symbol are written in the first two FIFOs.

2. At the fourth sample, both symbols are finally synchronized: the previous one stored in the second two FIFOs and the incoming one which is being stored in the first two FIFOs. Thus, the four FIFOs are read in order to perform the several multiplications.

The several signals' temporal behaviours are illustrated in Figure 5.9 for the second state of the finite state machine.

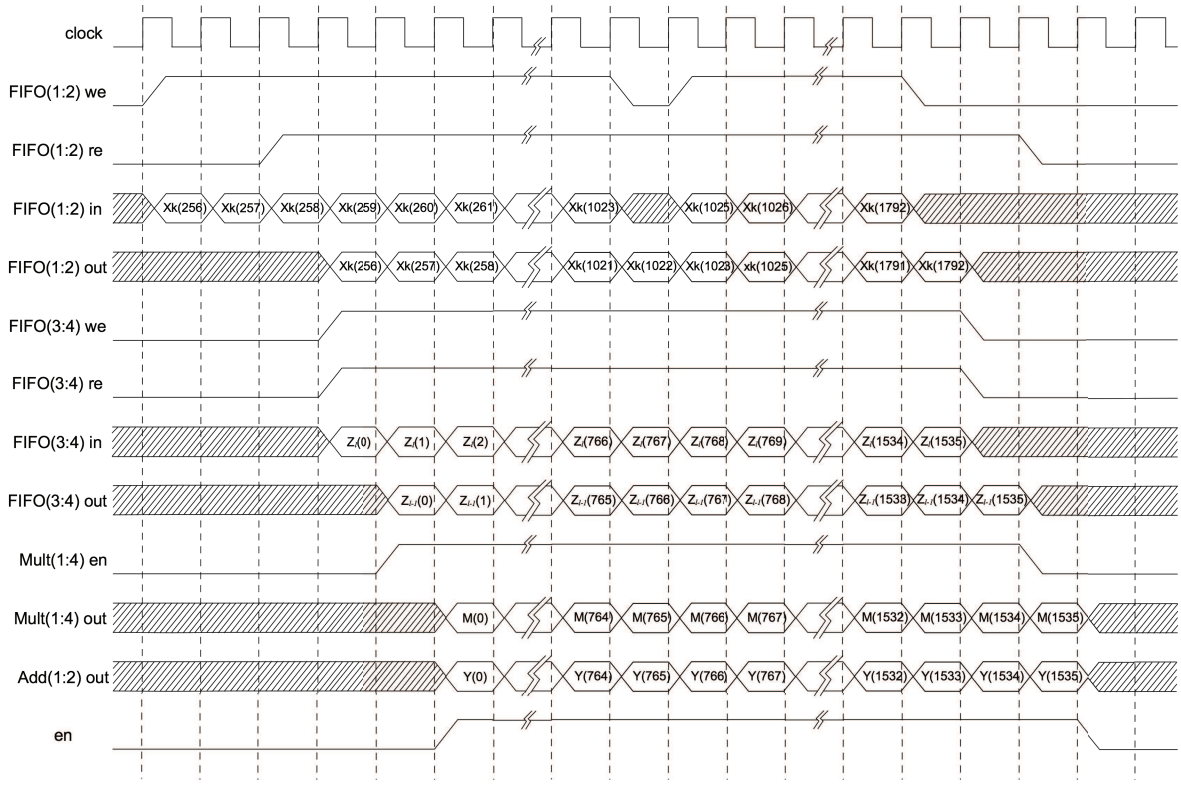


Figure 5.9: *Temporal behaviour of the differential demodulation subsystem signals.*

As one can see through Figure 5.9, the utilization of two pairs of FIFOs originates a continuous differential demodulated output. Since the available FIFOs only have sizes that are a power of two, one of the FIFOs will have a size of 4 and the other of 2048.

The differential demodulation subsystem takes 1597 clock cycles to process each sample.

### 5.2.3 Frequency Deinterleaving

The frequency interleaving mixes the different subcarriers according to a fixed sequence specified in the DAB standard. Therefore, the frequency deinterleaving can be implemented by addressing the output of the differential demodulation block according to the corresponding deinterleaving fixed sequence. Figure 5.10 shows the frequency deinterleaving subsystem that was implemented in the System Generator.

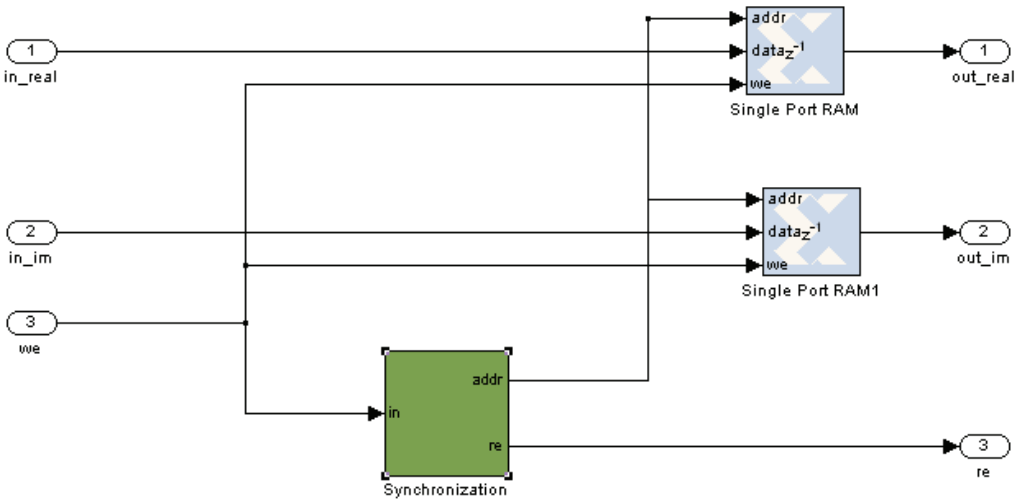


Figure 5.10: *Frequency deinterleaving subsystem in the System Generator.*

Two single port Random Access Memories (RAMs) are used for both real and imaginary parts of the incoming signal. First, the RAMs are filled with the incoming data, which is then read according to the deinterleaver fixed sequence. The synchronization block is based on a ROM where the fixed sequence is stored. So, when this ROM is read its outputs are going to address the two RAMs' data in the desired order. The temporal behaviour of the several signals are illustrated in Figure 5.11.

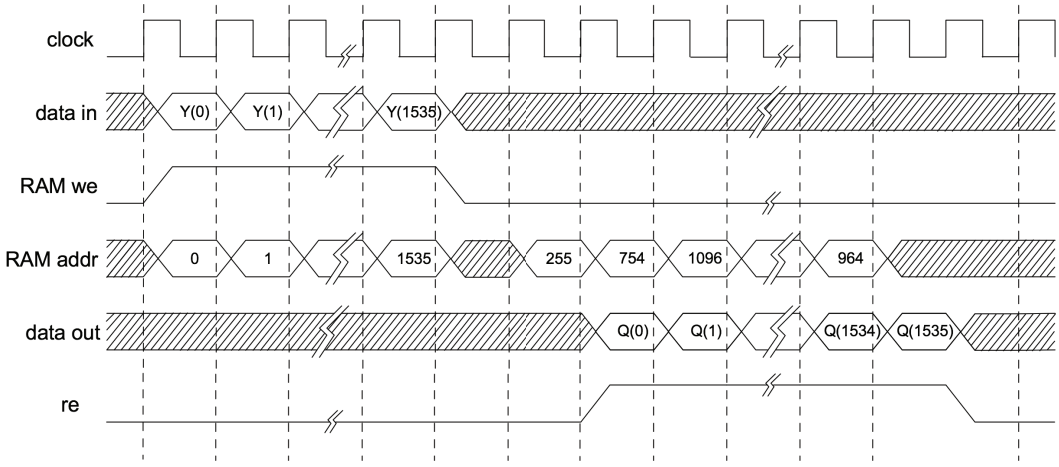


Figure 5.11: *Temporal behaviour of the frequency deinterleaving subsystem signals.*

As one can see through Figure 5.11, the first index of the frequency deinterleaving fixed sequence is 754, the second one is 1096 and the last one is 964. Meanwhile, it addresses all the 1536 samples stored in each RAM.

The frequency deinterleaving subsystem has a latency of 3074 clock cycles, i.e., takes 3074 clock cycles to process each symbol. This subsystem can be also used for frequency interleaving by simple storing the appropriate sequence.

### 5.2.4 QPSK Symbol Demapper

QPSK symbol demapping decides whether the bits carried by each QPSK symbol are "1" or "0". As explained in the last chapter, this process simply checks if the real and the imaginary parts of a given symbol are positive or negative. That is:

$$\begin{cases} I < 0 & \Rightarrow & bit_k = 1 \\ I > 0 & \Rightarrow & bit_k = 0 \\ Q < 0 & \Rightarrow & bit_{K-k} = 1 \\ Q > 0 & \Rightarrow & bit_{K-k} = 0 \end{cases} \quad where \quad k = 0, 1, ..., K-1 \quad and \quad K = 1536 \qquad (5.6)$$

The System Generator subsystem that implements this function can be seen in Figure 5.12.

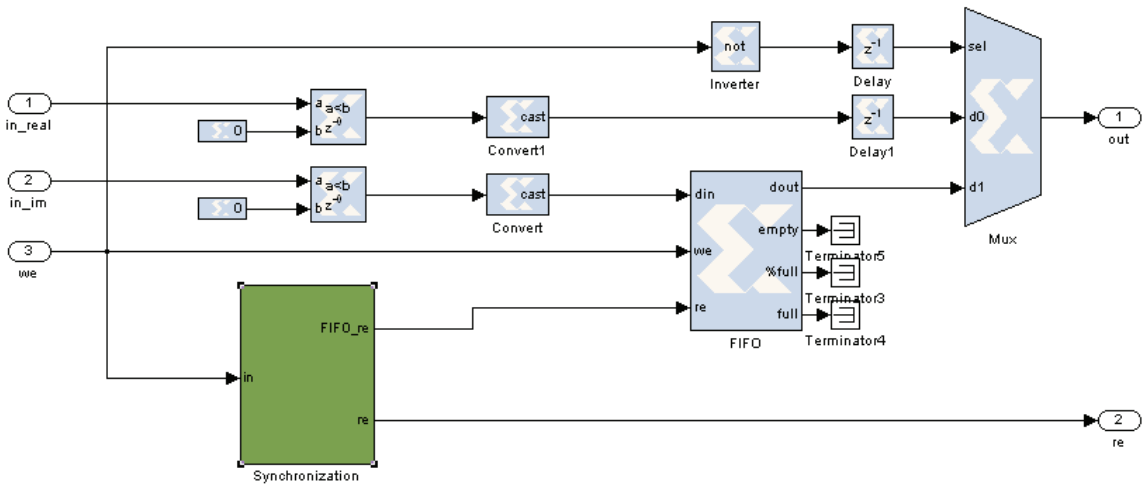


Figure 5.12: *QPSK symbol demapper subsystem in the System Generator.*

Per each OFDM symbol, the real parts of the several QPSK symbols correspond to the first $K$ bits and the corresponding imaginary parts correspond to the last $K$ bits. Thus, a FIFO is used to store the imaginary components while the real ones are being evaluated. The synchronization block simply controls the read enable of the FIFO as well as the read enable of the main system, since this is the final subsystem. The temporal behaviour of the several signals are illustrated in Figure 5.13. The *binary sequences* $A$ and $B$ are the output signals of both real and the imaginary comparators, respectively.

The QPSK symbol demapper subsystem has no latency, i.e. the first sample of each OFDM symbol does not take any clock cycle to be process. Therefore, the system will only have combinational delays.

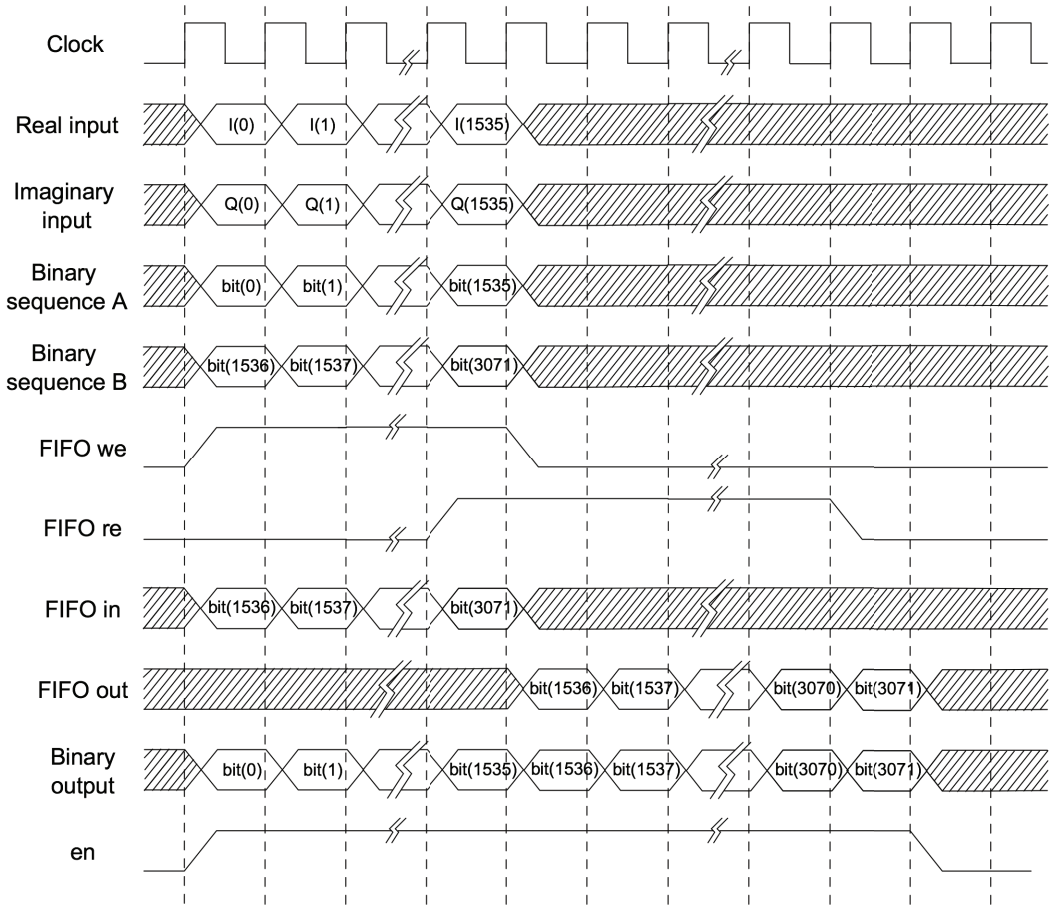


Figure 5.13: *Temporal behaviour of the QPSK symbol demapper subsystem signals.*

## 5.3 Results

Programming an FPGA using System Generator means describing a computation as a Simulink model, generating a hardware description from this model and then compiling this hardware description into an FPGA configuration file, called a *bitstream* [Nal07]. The first step is already done and allowed us to design an accurate model of an FPGA circuit in Simulink. However, a hardware engineer wants to see his design running in hardware. With that purpose, the last two steps must be taken.

System Generator provides hardware co-simulation interfaces that make it possible to incorporate an FPGA directly into a Simulink simulation. Thus, it allows the compiled design to be tested in actual hardware. The co-simulation tool was used in order to test the DAB receiver and its results are presented in the next subsection.

Finally, the last subsection presents the main FPGA resources that are used by the DAB receiver, as well as its time behaviour and frequency constraints.

### 5.3.1 Co-Simulation

Once the DAB receiver model was already simulated within Simulink, we are ready to start with the co-simulation flow. The System Generator code generator has *Hardware Co-*

*simulation* compilation targets that create the *netlist* of the design but also run the design through the synthesis and implementation tools to create a *bitfile* automatically. Once complete, a new library is created which contains a runtime block for the co-simulation of the design [Nal07]. Thus, after choosing the XtremeDSP Development Kit as the compilation target, the DAB receiver model was compiled and the library that is illustrated in Figure 5.14 was created.

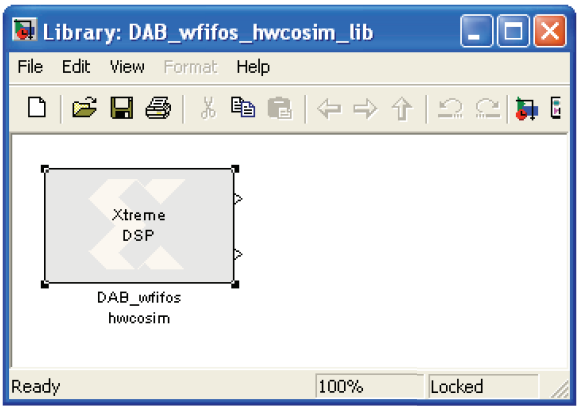


Figure 5.14: *Runtime block for DAB receiver co-simulation.*

If we add this block to our design, System Generator automatically incorporates an FPGA hardware platform configured with its *bitstream* back into Simulink as a run-time block. When the design is simulated in Simulink, results for the compiled portion are calculated in hardware.

There are two ways in which a System Generator hardware co-simulation block can be synchronized with its associated FPGA hardware: single-step mode and free-running mode. In the former mode, the FPGA is clocked from Simulink. Whereas in free-running clock mode, the FPGA runs off an internal clock and is sampled asynchronously when Simulink wakes up the hardware cosimulation block [Xil08d]. Both clock modes were used for DAB receiver co-simulation.

The next two subsections explain the algorithms that were developed to co-simulate the DAB receiver in both clock modes, as well as present their results. The co-simulation was performed through PCI interface.

### Co-simulation with Single-Step Clock

When using the single-step mode, the algorithm used to co-simulate the system is relatively straightforward. Since the hardware is kept in lock step with the software simulation, one can co-simulate the design at the same time that compare its results with the ones produced by the Simulink model simulation. The model used is illustrated in Figure 5.15.

Thus, the results of both simulations are first compared in order to increment a counter when an error occurs. The comparison between the resulting *bitstreams* is only activated when both read enables are set. The results did not show any error.

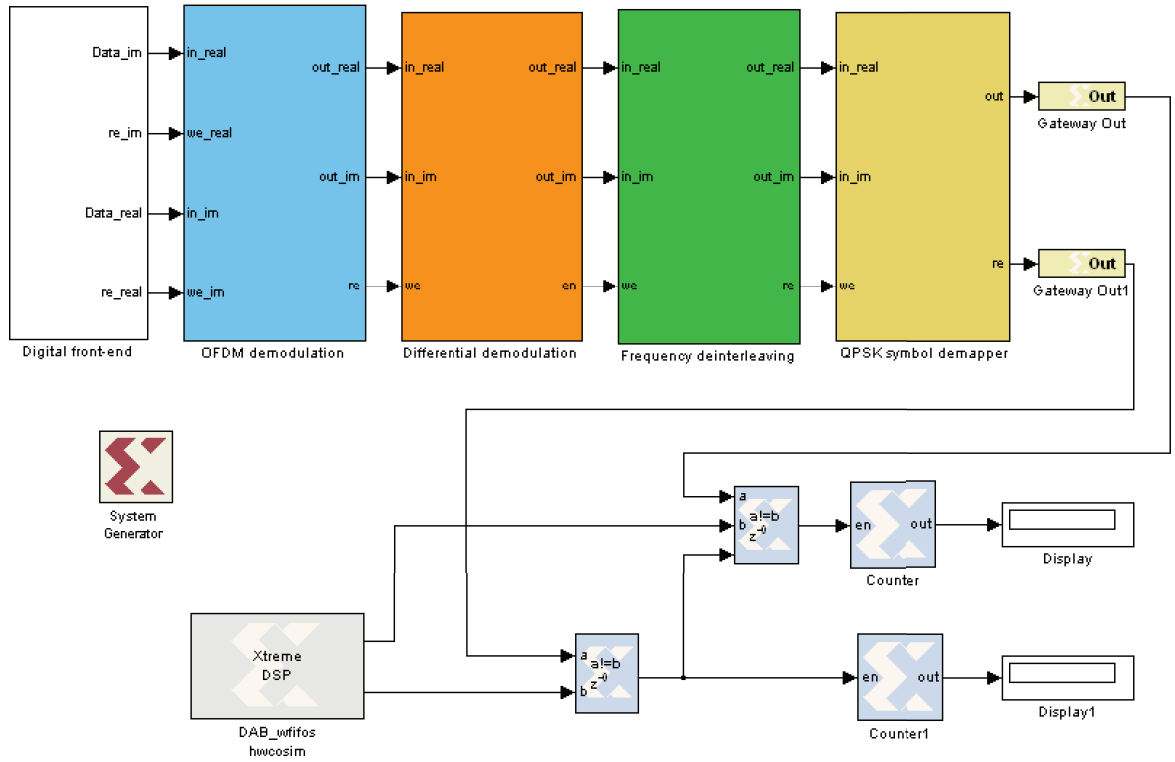


Figure 5.15: *Co-simulation with a single-step clock.*

### Co-simulation with Free-Running Clock

The development of an algorithm to obtain the co-simulation results when using the free-running mode clock is not so straightforward as with the single-step clock mode. Since the hardware runs asynchronously relatively to the software simulation, the FPGA port Input/Output (I/O) is no longer synchronized with the Simulink events. Thus, Simulink is only sampling the internal state of the hardware at the times when Simulink awakes the hardware co-simulation block [Xil08d].

The first approach that was developed to receive the demodulated data consisted in a feedback mechanism between the FPGA and Simulink. Thus, the first demodulated symbol was stored in an FIFO to be read by Simulink. When Simulink received a data sample, it would send an acknowledgment. Subsequently, the FPGA would send the next data sample. However, the temporal behavior between the inputs and outputs showed to be also asynchronous. Since the data samples were sometimes missed or repeated, this algorithm was discard.

When reading the System Generator user guide, another idea emerged based on the following statement:

"In free-running mode, you must build explicit synchronization mechanisms into the System Generator model. A simple example is a status register, exposed as an output port on the hardware co-simulation block, which is set in hardware when a condition is met. The rest of the System Generator model can poll the status register to determine the state of he

hardware." [Xil08d]

Why do not compare the simulation results inside the FPGA instead of compare them in the Simulink? Thus, the Simulink results were stored in a FIFO and compiled into the FPGA. The remaining approach was similar to the one used in the single-step clock mode: the simulation results were compared with the ones stored in the FIFO and the resulting errors were counted. The rest of the System Generator model could finally poll the counter value in order to determine the number of errors. The FPGA clock was set to 16 MHz, the minimum programmable frequency.

More than 10 symbols were stored and cyclic demodulated. The results did not show any error. To be sure, some errors were explicitly introduced into the stored data. The exact number of errors were counted by the FPGA and received through co-simulation.

### 5.3.2 FPGA Resources

In order to analyse the several FPGA resources used by the DAB receiver, as well as their distributions among the four subsystems, the DAB receiver and each of its subsystems were individually compiled into FPGA configuration *bitstreams*. Table 5.1 summarize the amount of main resources used by the several subsystems and their parent system. The several FPGA resources functionalities will not be presented in this work. For further information see [Xil06a] and [Xil07].

| | OFDM demodulation | Differential demodulation | Frequency deinterleaving | QPSK symbol demapper | Total (post-map) | Total available | Device usage |
|---|---|---|---|---|---|---|---|
| **Slices** | 1754 | 477 | 108 | 74 | 2385 | 15360 | 15% |
| **LUTs** | 2176 | 556 | 206 | 130 | 3009 | 30720 | 9% |
| **Flip Flops** | 2464 | 583 | 30 | 75 | 3153 | 30720 | 10% |
| **RAMs** | 19 | 20 | 18 | 1 | 46 | 192 | 23% |
| **DSP48s** | 28 | 36 | 0 | 0 | 64 | 192 | 33% |

Table 5.1: *FPGA resources used by the DAB receiver.*

For a better idea of the FPGA resources distribution among the four subsystems, Figure 5.16 illustrates these results through several pie graphs. It is easy to conclude that the OFDM demodulation subsystem is responsible for more than a half of the resources, which is easy to understand since its FFT block is the most complex block that is used. The differential demodulation subsystem comes next due to its four multiplications. The Frequency deinterleaving and QPSK system demapper subsystems are the most "light" ones. Moreover, one should note that each FPGA slice is made up of two Flip Flops and two LUTs. Thus, the minimum amount of FPGA slices is half of the most used of those resources. Table 5.1 is in accordance with this requirement.

Finally, Table 5.2 presents the DAB receiver time behaviour and frequency constraints.

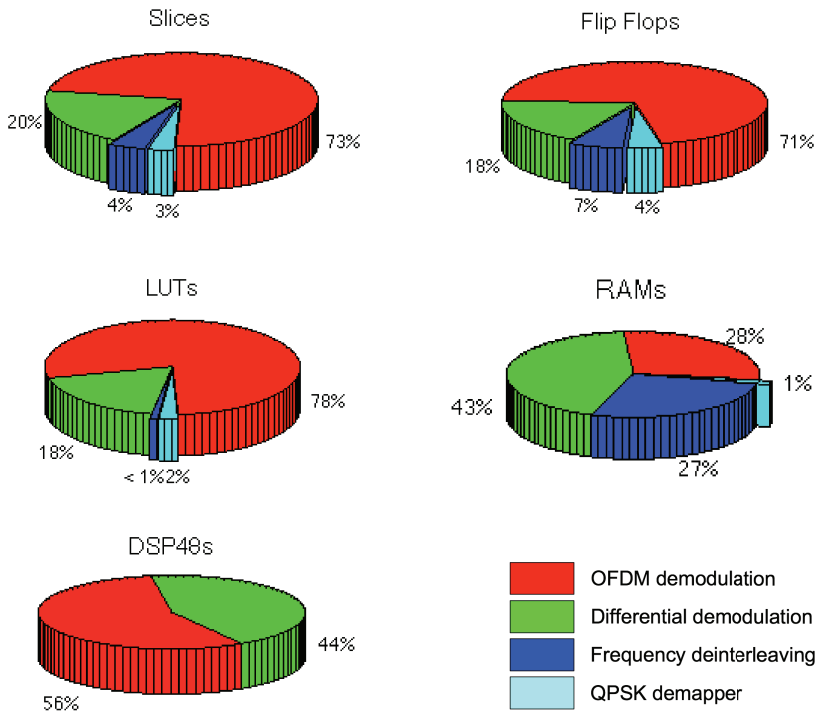


Figure 5.16: *Pie graphs of the several FPGA resources distribution among the DAB receiver subsystems.*

| OFDM demodulation clk cycles | Differential demodulation clk cycles | Frequency deinterleaving clk cycles | QPSK symbol demapper clk cycles | Total number of clk cycles | $f_{max}$ (MHz) |
|---|---|---|---|---|---|
| 13627 | 261 | 1538 | 0 | 15426 | 28.817 |

Table 5.2: *DAB receiver time behaviour and frequency constraints.*

The number of clock cycles is related to the latency of each subsystem, i.e., it is the number of cycles that the first input sample of each OFDM symbol takes to be processed and be ready to read. The maximum frequency allowed by the system is about 28 MHz.

# Chapter 6

# Conclusions and Future Work

## 6.1 Conclusions

The main goal of this dissertation was the development of part of a DAB receiver by means of SDR.

Thus, the physical layer of a DAB receiver was firstly developed in MATLAB. The simulated system was tested with known DAB samples that were passed through several simulated channel conditions. These results allowed us to perform a BER study with several SNR, frequency and time offsets channel conditions. The simulated AWGN channel BER rate curve was close to the theoretical one. The small mismatch was probably related with the approximation made when calculating the $\pi/4$ - DQPSK BER curve. The remaining channel conditions simulation, such as frequency and time offsets, wrong sampling frequency and multipath, showed that the synchronization system is working as expected since the corresponding BER curves were usually less then $10^{-4}$ when the Eb/No was higher then 15 dB. Only fractional sampling frequency offsets and strong multipath effects revealed higher BERs. The former one is related with a rough carrier offset estimation, which originates a BER of about $10^{-3}$. While the later one is related with fine frequency synchronization. Thus, when a delayed echo corrupts more than a half of the cyclic prefix, this synchronization process is affected originating a BER performance deterioration. Furthermore, the developed receiver was also tested with real DAB samples. The evaluated results showed the demodulated constellations as they were expected.

Only part of the DAB physical layer was implemented in the FPGA. This part corresponds to the channel demodulation functions, such as OFDM demodulation, differential demodulation, frequency deinterleaving and QPSK demapper. The DAB synchronization block was not implemented. The developed design was able to recover the modulated bit stream from the digital signal produced in MATLAB, since this signal was free of noise, frequency and time offsets. Xilinx System Generator provides a tool that allows us to perform both single-clocked and free-running hardware co-simulation. Thus, after intensive Simulink simulations the design was finally tested in the hardware through this tool. Both co-simulation clock modes showed that the DAB signal was demodulated without any error and, therefore, the implemented functions are working well. Further FPGA resources analysis revealed that the OFDM demodulation and differential demodulation subsystems require most of the FPGA resources. This fact is easily understood since the FFT used by the OFDM modulation subsystem is the most complex and computationally exigent block used in the overall imple-

mentation. Furthermore, the FFT is also responsible for more than half of the clock cycles. One should note that, if necessary, faster FFT algorithms can be used, however they would consume more resources. The four multiplication of the differential subsystem are responsible for most of the remaining resources.

When implementing the DAB synchronization system, several implementation choices can be redesigned in order to use the hardware and time resources as needed for the overall system to work.

## 6.2 Future Work

As a future work, the first step shall be the development of the synchronization block. Next, the digital front-end might be implemented, as well as the ADC be programmed in accordance to the analog front-end features. At last, the digital part of the receiver shall be combined with the analog front-end and the whole system be tested in real time with real channel conditions. Furthermore, if one wants to hear an audio signal, the implementation of the FIC and MSC will be required.

Finally, it would also be interesting to implement a DAB transmitter. Since a transmitter does not require such things as frequency synchronization, its implementation would be much easier than the implementation of the corresponding receiver. With both transmitter and receiver, innumerous studies can be made such as an alternative use of the Portuguese DAB ensemble, which is not being fully explored.

# Appendix A

# DAB Emission Block Diagram

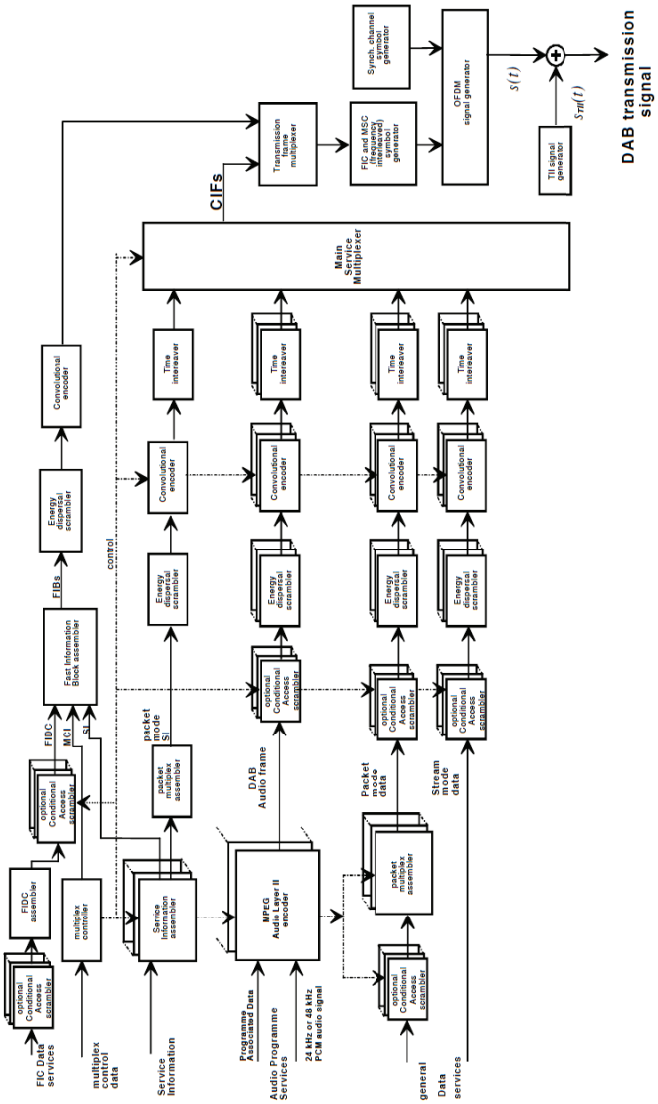


Figure A.1: *Conceptual DAB emission block diagram. [Eur06]*

# Appendix B

# MATLAB Functions

As explained in Chapter 4, the physical layer of both DAB transmitter and receiver was fully implemented in MATLAB. The resulting transmitter and receiver were connected through a channel model in order to form a test bench. Figure B.1 shows the several MATLAB functions, as well as the signal flow through them.

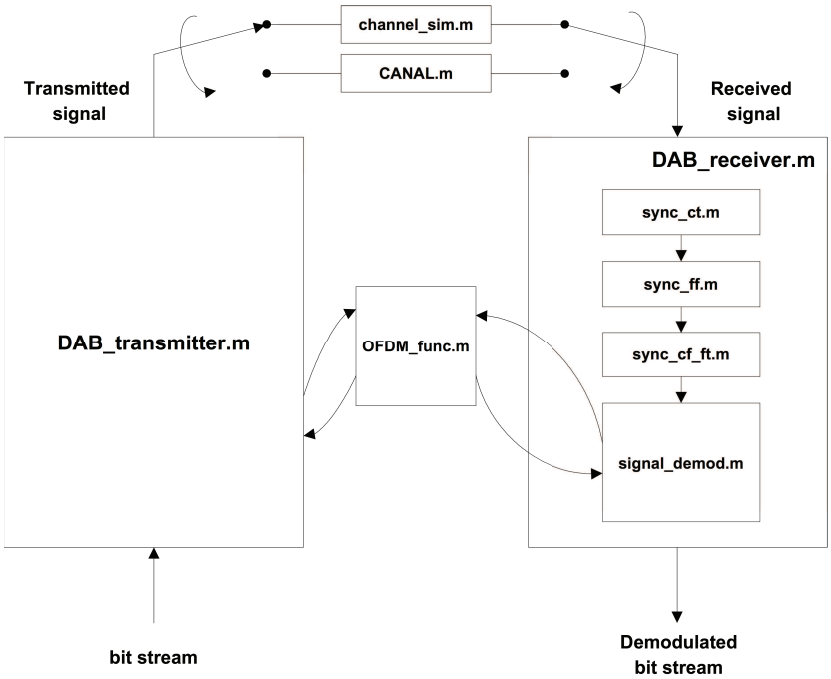


Figure B.1: *Signal flow through the several MATLAB functions.*

Thus, the bit stream is firstly fed into the *DAB_transmitter.m* function. This function performs the several signal modulation operations, such as QPSK mapper, differential modulation, frequency interleaving and OFDM demodulation. Once these operations are complete, the DAB signal is provided.

Subsequently, the DAB signal is fed into the channel model. Two functions were used with that purpose: *channel_sim.m* and *CANAL.m*. The former one was developed during this dissertation in order to model the following channel conditions:

**Sampling frequency offset -** modeled by a fractional interpolator

**Multipath propagation -** modeled by a simple FIR filter

*CANAL.m* is an MATLAB object (see Appendix C) implemented by Daniel Albuquerque [Alb09]. This object was also used to model the following channel conditions:

**AWGN -** modeled by a noise source and an adder

**Time offset -** modeled by introducing zeros on the beginning of each vector

**Carrier frequency offset -** modeled by a sine source and a multiplier

Once the DAB signal is distorted, the receiver should be used in order to recover the bit stream. The BAB receiver is implemented in the *DAB_receiver.m* function, which comprise both DAB synchronization and signal demodulation systems. The former one consists in the following functions:

***sync_ct.m* -** performs coarse time synchronization

***sync_ff.m* -** performs fine frequency synchronization

***sync_cf_ft.m* -** performs both coarse frequency and fine time synchronizations

After time synchronization and frequency correction, the DAB signal is finally ready to be demodulated by the *signal_demod.m* function. Moreover, both DAB transmitter and receiver functions call the *OFDM_func.m* function in order to OFDM modulate and demodulate the respectively signal.

The various functions are defined as follows:

***out = DAB_transmitter (in)* -** *in* is the input bit stream and *out* is the output DAB signal.

***out = channel_sim (in,delay,amp,fs)* -** *in* is the input signal, *delay* and *amp* are, respectively, the delay and amplitude of the signal echo, *fs* is the desired sampling rate and *out* is output distorted signal.

***obj = CANAL (EbNo,dt/fs,fs,df)* -** *EbNo* is the Eb/No, *dt/fs* is the number of delay samples, *fs* is the actual sampling rate, *df* is the desired frequency offset and *obj* is the created object. In order to call the object's modulo the following command must be used: *out = filter(obj,in)*, where *obj* is the created object, *in* is the input signal and *out* is the signal after distortion.

***out = DAB_receiver (in)* -** *in* is the input signal and *out* is the demodulated bit stream.

***[ct,ymean,ymin] = sync_ct (in,start)* -** *in* is the input signal, *start* is starting sample index, *ymean* and *ymin* are, respectively, the mean and minimum of the DAB envelope (see Chapter 4) and *ct* is the coarsely estimated frame beginning.

**ff = sync_ff (in) -** *in* is one OFDM symbol and *ff* is the estimation of the fractional part of frequency offset.

***[cf,ft] = sync_ft_cf (in,ymean,ymin) -*** *in* is the PRS, *ymean* and *ymin* are, respectively, the mean and minimum of the DAB envelope (*sync_ct.m* function outputs), *cf* is the estimation of the integral part of the frequency offset and *ft* is the accurate estimation of the frame beginning.

***out = OFDM_func (in,fs,mod) -*** *in* is the input signal, *fs* is the actual sampling rate, *mod* defines if the signal is being modulated ($mod = 1$) or demodulated ($mod = 0$), and *out* is the OFDM (de)modulated signal.

***out = signal_demod (in, start) -*** *in* is the input signal, *start* is starting sample index and *out* is the demodulated bit stream.

The receiver functions that were not mentioned, such as $fs$ estimation and frequency offset correction, are performed in the *DAB_receiver.m* function.

# Appendix C

# DAB Objects

*"Object-oriented programming (OO) applies to software development the standard science and engineering practice of identifying patterns and defining a classification system describing those patterns. Classification systems and design patterns allow engineers and scientists to make sense of complex systems and to reuse efforts by others."* [McG08]

Thus, by applying to object-oriented programming one can improve the ability to develop and manage large applications and data structures. With that purpose, a DAB system object was created.

The DAB object is called *DAB.m* and includes both DAB transmitter's and receiver's physical layers. Only the several signal demodulation functions were implemented, i.e. the QPSK (de)mapper, frequency (de)interleaving, differential and OFDM (de)modulation functions. The DAB synchronization system is not included.

The DAB object can thus be crated as follows:

$$>> obj = DAB(fs);$$

where $fs$ is the desired sampling rate.

Once the DAB object is created, both transmitter and receiver methods can be performed. Methods are called just like functions, with the object *obj* passed in as one of the arguments [McG08]. Thus, the transmitter method can be called as follows:

$$>> out = transmitter(obj, in);$$

where *obj* is the DAB object, *in* is the input bit stream and *out* is the DAB signal. Please note that only one frame is modulated per each call, so the length of the input vector must be equal to number of data bits per transmission frame, i.e., 230400. Similarly, the receiver method is called as follows:

$$>> out = receiver(obj, in);$$

where *obj* is the DAB object, *in* is the received DAB signal and *out* is demodulated bit stream. The length of the input vector must be equal to the number of samples per transmission frame, which depends on the sampling frequency that was chosen.

In order to clarify any doubt, the following example of use is given:

```
>> fs = 2048000;                    % Sampling frequency
>> Nb = 230400;                     % Number of bits per transmission frame
>> x = randint(1, Nb);              % Creates the input bit stream
>> obj = DAB(fs);                   % Creates the DAB object
>> signal = transmitter(obj, x);    % Creates the DAB signal
>> y = receiver(obj, signal);       % Recovers the original bit stream
```

The produced bit stream will be equal to the original one, i.e., $y = x$.

The DAB object is now ready to be used in any user design, providing DAB signal demodulation functions abstraction and, therefore, allowing the construction of more complex systems.

# Appendix D

# Xtreme DSP Development Kit-IV

## D.1 Introduction

Developed by Xilinx and Nallatech, the XtremeDSP Development Kit-IV is a complete development platform for the Virtex-4 FPGA technology. The dual-channel, high-performance ADCs and DACs, coupled with a user-programmable Virtex-4 SX35-10 FPGA, make this platform ideal for implementing high-performance signal processing applications such as SDR systems. The SX35 FPGA features over 55,000 logic cells, 192 XtremeDSP slices [Xil06b].

Moreover, this platform can be combined with the Xilinx System Generator for DSP software tool providing a faster and easier transition from algorithm concept to hardware verification. The System Generator tool allows us to perform hardware co-simulation on the XtremeDSP Development Platform via PCI or JTAG. Since this tool interfaces with MATLAB/Simulink, it enables the development of algorithms, as well as the debugging and verification of the resulting designs on the actual hardware, and always within the same user-friendly environment.

The next subsections shall present the main features of this platform and its functionalities.

## D.2 Key Features

The XtremeDSP Development Kit-IV platform consists of a motherboard populated with a module (daughter card). The motherboard is referred to as the *BenONE-Kit Motherboard* and the module is referred to as the *BenADDA DIME-II module* [Nal07]. The complete board can be seen in Figure D.1.

The several features of both motherboard and module are enumerated in [Nal07] as follows (quote):

**BenONE-Kit motherboard:**

- Supports the supplied BenADDA DIME-II module only
- Spartan-II FPGA for 3.3V/5V PCI
- Host interfacing via 3.3V/5V PCI 32-bit/33-MHz interface
- Status LEDs

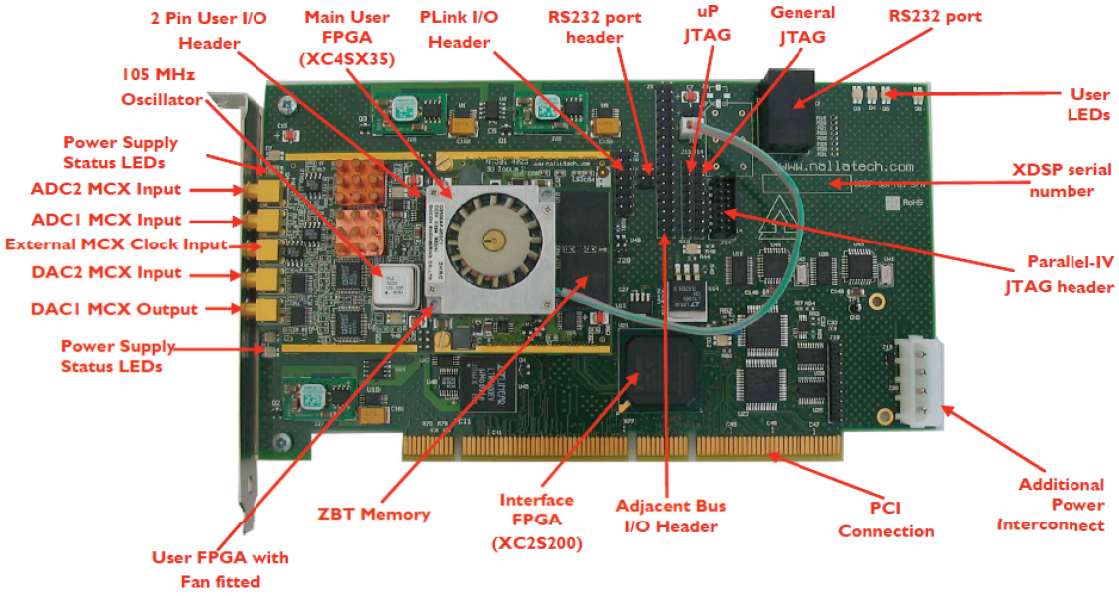


Figure D.1: *XtremeDSP Development Kit-IV. [Nal07]*

- JTAG configuration headers
- User 0.1 pitch pin headers connected directly to user programmable FPGA I/O
- RS232 ports

**BenADDA DIME-II module:**

- Virtex-4 User FPGA: XC4VSX35-10FF668
- Two independent ADC channels: AD6645 ADC (14-bits up to 105 MSPS)
- Two independent DAC channels: AD9772 DAC (14-bits up to 160 MSPS)
- Support for external clock, on board oscillator, and programmable clocks
- Two banks of ZBT-SRAM (133MHz, 512Kx32-bits per bank)
- Multiple clocking options: Internal and External
- Status LEDs

## D.3 Functional Diagram

The XtremeDSP Development Kit-IV functional diagram is illustrated in Figure D.2. Thus, this platform contains three Xilinx FPGAs: a Virtex-4 User FPGA, a Virtex-II FPGA for clock management and a Spartan-II Interface FPGA.

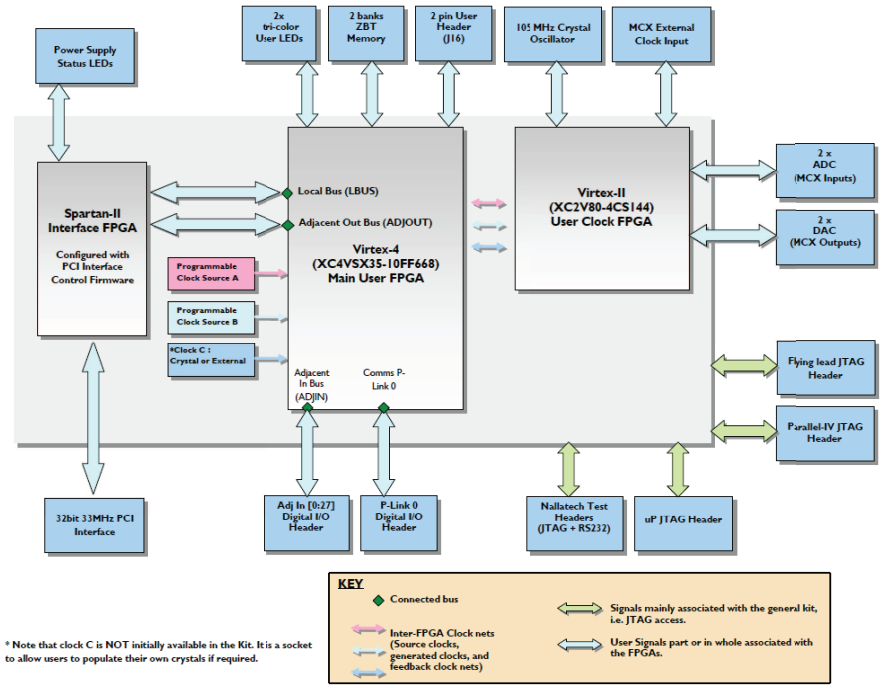


Figure D.2: *XtremeDSP Development Kit-IV functional diagram. [Nal07]*

The Virtex-4 XC4VSX35-10FF668 device is available exclusively for the main part of user's design [Nal07].

The Spartan-II enables PCI interfacing. This FPGA is supplied with pre-configured PCI interfacing firmware and low level drivers, which abstract the PCI interfacing from the user resulting in a simplified design process. The Interface FPGA communicates directly with the user FPGA via a dedicated communications bus that is made up of the LBUS and ADJOUT busses [Nal07].

Finally, the Virtex-II XC2V80-4CS144 is intended to be used as a clock configuration device in every user design [Nal07]. This FPGA is located on the underside of the module, so it is not visible in Figure D.1.

Further information on the several XtremeDSP Development Kit-IV's functionalities can be found in [Nal07].

## D.4 Nallatech and Xilinx Software Support

Nallatech motherboards and modules, such as the XtremeDSP Development Kit-IV platform's BenONE motherboard and BenADDA module, are designed to the modular DIME-II standard. The software support for DIME-II hardware is provided by Field Upgradeable Systems Environment (FUSE).

Nallatech's FUSE system software provides several configuration, control and communication functionalities, which enable developers to design complex processing systems with easy integration between software, hardware and FPGA applications. FUSE provides several interfaces, including the scripting language DIMEscript, FUSE Probe Tool, and the FUSE development APIs for C/C++. FUSE is available for both Windows and Linux operating

systems, and optional APIs are also available for Java and MATLAB [Nal07]. In [Nal07] the following Fuse's key features are presented:

- Fast and simple device configuration
- Multiple card support
- Interfacing and control of Nallatech hardware features

Furthermore, the XtremeDSP Development Kit-IV platform can be combined with Xilinx software, such as the Xilinx Foundation ISE, Xilinx System Generator for DSP and MATLAB/Simulink. Thus, the developers are able to implement, compile and simulate their designs always within the same user-friendly application.

Both Nallatech and Fuse software make the XtremeDSP Development Kit-IV platform the ideal environment to develop complex signal processing systems, such as SDR applications.

# Appendix E

# Xilinx System Generator for DSP

## E.1 Introduction

System Generator is a DSP design tool from Xilinx that enables FPGA hardware design and extends Simulink in several ways in order to provide a friendly and powerful modelling environment for signal processing systems development. Designs are therefore developed in the Simulink environment using a Xilinx specific blockset, which provides a high level of abstraction. Moreover, the System Generator also provides access to underlying FPGA resources through lower level abstractions, enabling the implementation of highly efficient FPGA designs [Nal07]. All of the downstream FPGA implementation steps including synthesis, place and route are automatically performed to generate an FPGA programming file [Xil08b].

Thus, Xilinx System Generator for DSP provides a complete environment for developing FPGA designs much easier and faster. The following subsections shall present some of the System Generator main features.

## E.2 System Generator Design Flow

Programming an FPGA using Xilinx System Generator means describing an algorithm as a Simulink model, generating a hardware description from this model, and then compiling this hardware description into an FPGA configuration file, called *bitstream* [Nal07]. Figure E.1 illustrates the System Generator design flow.

An executable spec can be firstly created using the standard Simulink blockset and without any hardware detail. Once the functionality and basic dataflow issues have been defined, the executable spec must be translate into a hardware model using the Xilinx DSP blockset for Simulink. System Generator will automatically invoke Xilinx Core Generator to generate highly optimized *netlists* for the DSP building blocks and optionally execute all the downstream implementation tools to produce a *bitstream* for programming the FPGA. Furthermore, an optional testbench can be created using test signals generated in the Simulink environment for use with ModelSim or the Xilinx ISE Simulator [Xil08b]. As already said, hardware co-simulation is also provided for design acceleration. Thus, if the platform that is being used supports this feature, the previous testbench is likely to be replaced by truly hardware simulation.

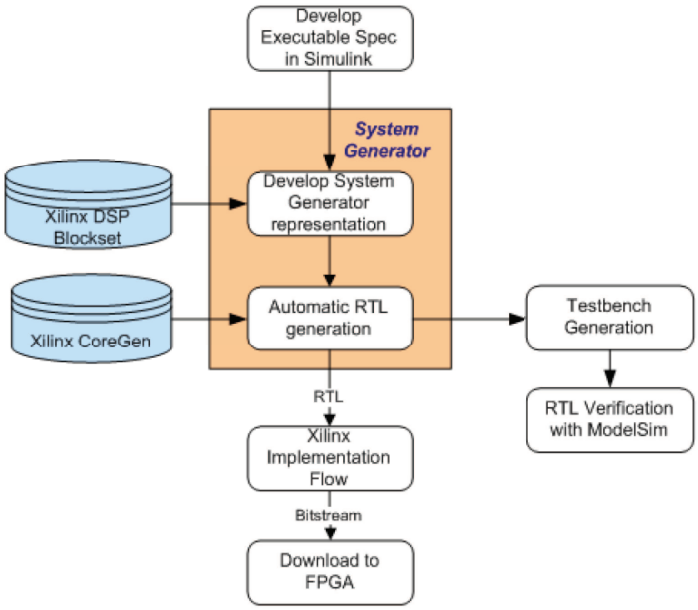


Figure E.1: *System Generator design flow. [Xil08b]*

## E.3 Xilinx DSP Blockset

The Xilinx DSP blockset can be accessed via the Simulink Library browser, as depicted in Figure E.2.

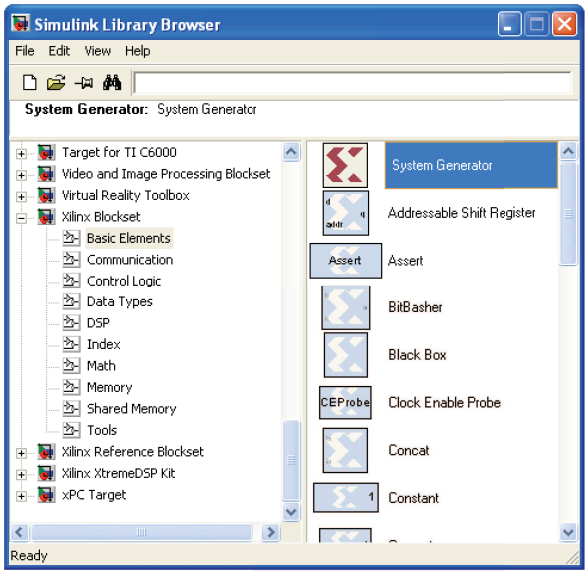


Figure E.2: *Simulink Library browser and Xilinx DSP blockset.*

Over 90 DSP building blocks are available, including common blocks such as adders, multipliers and registers, or more complex DSP blocks such as FFTs, filters and memories. The next subsections shall describe some of these blocks and their functionalities.

### E.3.1 Basic Blocks

Figure E.3 shows a simple System Generator design.

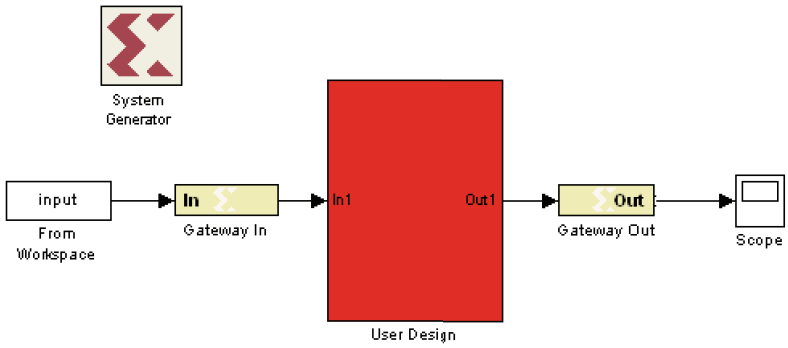


Figure E.3: *System Generator design example.*

Since System Generator works with standard Simulink models, specific blocks are needed to define the boundary of the FPGA from the Simulink simulation model. Thus, two blocks called *Gateway In* and *Gateway Out* are provided with that purpose, as depicted in Figure E.3. The *Gateway In* block converts the floating point input to a fixed-point number, whereas the *Gateway Out* block converts the FPGA outputs back to double precision [Xil08b].

Through Figure E.3, one can see an additional block called *System Generator Token*. Every System Generator design requires at least one of these blocks in order to drive the FPGA implementation process. The property editor of this block is illustrated in Figure E.4.

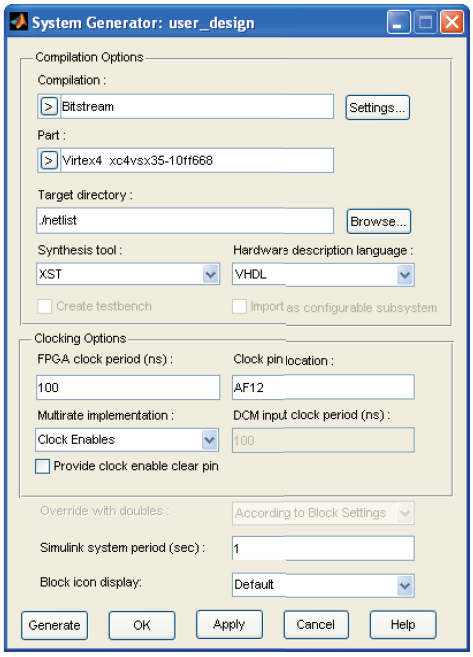


Figure E.4: *Property editor of the System Generator Token block.*

Thus, it allows specification of the target netlist, device, performance targets and system period [Xil08b].

Finally, the subsystem *User Design* of Figure E.3 can be developed using the several blocks of the Xilinx DSP blockset, such as filters, FFTs, FEC cores, memories, arithmetic, logical and bitwise blocks [Xil08b].

Morever, if one wants to generate input vectors or observe the produced results, standard Simulink blocks can be used before the *Gateway In* or after the *Gateway out* blocks with that purpose. In Figure XX, the Simulink *From Workspace* block is used to drive the input vector, while the Simulink *Scope* block is used to plot the output signal.

### E.3.2 System Control Blocks

When developing DSP systems in hardware, system control is highly required. This may include state dependent behavior, simply performing operations or bursty data such as non-streaming FFTs [Xil08b]. Thus, several System Generator blocks are provided with that purpose (see Figure E.5).

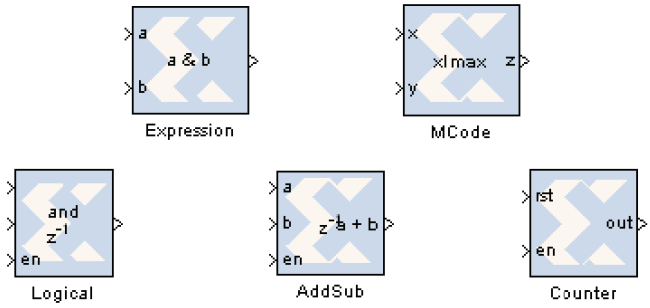


Figure E.5: *System Generator system control blocks.*

The *MCode* block supports the use of MATLAB code for developing state dependent and branch conditional control operations. This block provides the possibility of implementing finite state machines and complex muxing conditions [Xil08b].

The *Expression* block is provided in order to perform several bitwise operations, such as *not*, *and*, *or* and *xor*. This block can therefore be used to implement logical control in a DSP system [Xil08b].

Furthermore, storage elements can optionally include *reset* and *clock enable* ports that can be connected in the System Generator design. Thus, if a greater control over these functions is required, that ports might be used [Xil08b].

### E.3.3 Memory Blocks

Xilinx FPGAs offer two different memory options: Block RAM and Distributed RAM. Block RAM uses dedicated hardware resources, which offer high performance but larger routing delays. On the other hand, distributed RAM uses the lookup tables in the FPGA slices to implement memory. The routing delays can thus be minimized, but the number of slices available for logical operations is decreased. Both kinds of memories establish a trade-off between routing delays and resources allocation. Distributed RAMs are usually used for small memories.

The several System Generator memory blocks are illustrated in Figure E.6.

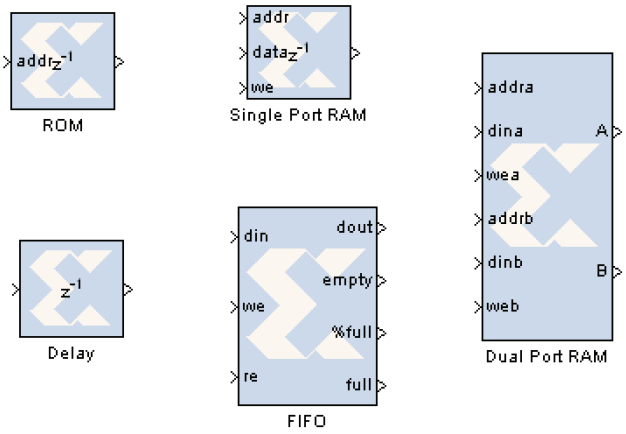


Figure E.6: *System Generator memory blocks.*

System Generator provides both single-port and dual-port *RAM* blocks, as well as *ROM* blocks. These two kinds of memories support both block and distributed RAM and are initialized through a $1 \times n$ vector that matches their depth. MATLAB statements can be used to set the initial value vector, e.g., the reading commands such as *imread*, *auread*, *wavread*, and *load* [Xil08b].

*Delay* and *FIFO* blocks are also available. The former one is usually used to synchronize dataflow through the FPGA, while the later one is used to store bursty data.

### E.3.4 Multirate System Blocks

Usually, radio communication systems, such as the developed DAB receiver, must down-convert the input signals prior to the digital signal processing performed during equalization and demodulation. These systems are known as *multirate systems* [Xil08b].

Figure E.7 shows some of the System Generator blocks that enable multirate systems.

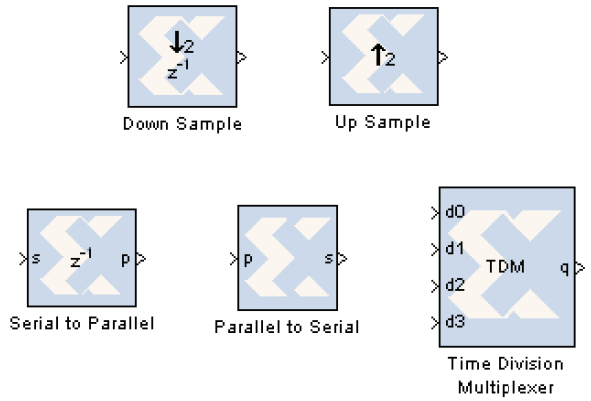


Figure E.7: *System Generator multirate system blocks.*

The *Up Sample* and *Down Sample* blocks are provided in order to change the system sample rate. The *Up Sample* block adds additional samples to the signal to achieve the

desired sampling rate, whereas the *Down Sample* block simply discards the necessary amount of samples [Xil08b].

Additional rate changing functional blocks are also provided. These blocks are used not only to change the sampling rate, but also to perform a specific function. Thus, the *Parallel to Serial* block will up sample, the *Serial to Parallel* block will down sample and the *Time Division Multiplexer (TDM)* block will up sample [Xil08b]. These three block are only a few examples of the many rate changing functional blocks avaiable.

Moreover, several debugging utilities are supplied in order to enable the debugging of complex multi-rate systems [Xil08b].

### E.3.5 Signal Processing Blocks

System Generator provides several specific signal processing blocks that make possible to develop complex radio communication systems such as SDR applications. This set of block includes FFTs, filters and encoders, among others. Figure E.8 illustrates some examples.

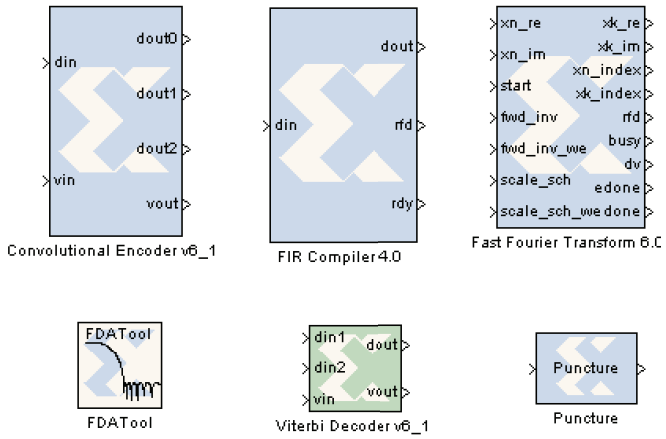


Figure E.8: *System Generator signal processing blocks.*

Further information on these blocks can be found in [Xil08b], [Xil08d] and [Xil08c].

## E.4 Co-simulation

System Generator enables hardware co-simulation of over 20 different hardware platforms. This tool allows a dramatically acceleration of the design simulation making it possible to incorporate an FPGA directly into a Simulink simulation. Both PCI and JTAG interfaces are supported. Since the XtremeDSP Dvelopment Kit-IV allows PCI simulation, the remaining section will be in accordance with this interface.

Thus, after simulating the System Generator model within Simulink, the next step is to make use of the co-simulation flow. This is described in [Nal07] as follows:

1. Ensure that the Sytem Generator model is open.

2. Double click on the *System Generator Token* block. Figure E.4 illustrates its properties editor.

3. Set the desired compilation target. In this case: Hardware cosimulation ⇒ XtremeDSP Development Kit ⇒ PCI'.

4. Set the desired part. In this case: 'Virtex-4 ⇒ XC4VSX35-10FF668'.

5. Set the FPGA Clock Period as this determines how fast the design is capable of running if using a free running clock.

6. Click on the *Generate* button to start the System Generator flow. The flow will create the *netlist* of the design, as well as run the design through the synthesis and implementation tools to create a *bitfile* automatically.

Once the System Generator flow finishes, a new library is created as depicted in Figure E.9.

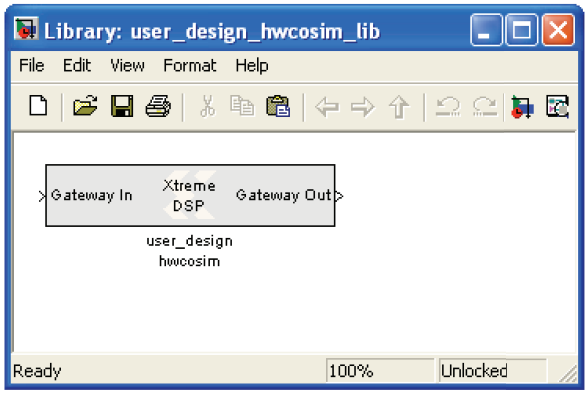


Figure E.9: *System Generator co-simulation library.*

This library contains a single runtime block for the co-simulation of the design. Thus, one can add this block to the System Generator model (see Chapter 5). If we double click on the co-simulation block, the control dialog box depicted in Figure E.10 will appear. The several block parameters control the selection of the *bitfile* and *clock mode* (see Chapter 5).

Finally, the co-simulation can be started by clicking on the Simulink *Start simulation* icon. By this way, the compiled design can be fully evaluated in actual hardware speeding up the simulation dramatically [Nal07].

Please note that the following requirements are highly necessary:

- Windows XP (32 bit only)
- ISE 10.1.03
- ISE 10.1 IP Update 3
- System Generator for DSP 10.1.03
- MATLAB R2007a or R2007b from The MathWorks

The ISE Service Pack 3 must be installed.

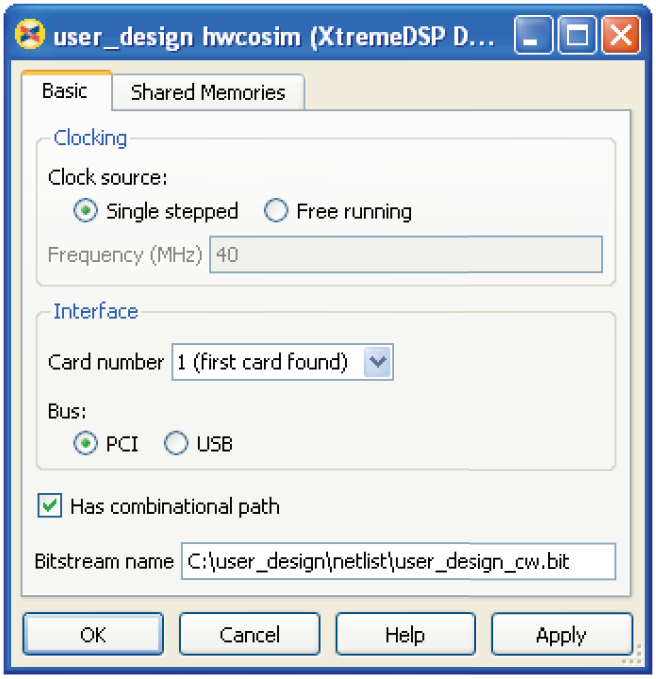


Figure E.10: *Property editor of the System Generator co-simulation block.*

# Bibliography


[Alb09] D. Albuquerque. "Front-Ends for Software-Defined Radio and it concerns". *Department of Electronics, Telecommunications and Informatics*, 2009.

[BB08] B. Brannon and D. Buchanan. "Software Defined Radio - The New Reality". Technical report, Analog Devices, Inc., 2008.

[Bow08] C. Bowick. "What's in an RF Front End?". *EE Times - India*, 2008.

[CCB+01] J. Cho, N. Cho, K. Bang, M. Park, H. Jun, H. Park, and D. Hong. "PC-Based Receiver for Eureka-147 Digital Audio Broadcasting". *IEEE Transactions on Broadcasting*, 47(2):95–102, June 2001.

[Chu08] P. Chu. *FPGA Prototyping by VHDL Examples Xilinx Spartan - Version 3*. John Wiley & Sons, Inc., 2008.

[CLA+09] A. Costa, J. Lima, L. Antunes, M. Souto, and N. Borges de Carvalho. "Spectrum analyzer with USRP, GNU Radio and MATLAB". *Conftele Conference on Telecommunications*, January 2009.

[Cou06] W. Couch. *"Digital and Analog Communication Systems"*. Prentice Hall, 7 edition, July 2006.

[DH03] C. Dick and F. Harris. "FPGA implementation of an OFDM PHY". *Conference Record of the Thirty-Seventh Asilomar Conference on Signals, Systems and Computers*, 1:905–909, November 2003.

[Eur06] European Telecommunications Standards Institute. *"ETSI EN 300 401: Radio Broadcasting Systems; Digital Audio Broadcasting (DAB) to Mobile, Portable and Fixed Receivers"*, 2006.

[For08] World DAB Forum. "Global Broadcasting Update", September 2008.

[For09a] SDR Forum. http://www.sdrforum.org/, March 2009.

[For09b] World DAB Forum. http://www.worlddab.org/, March 2009.

[Gan03] C. Gandy. *"DAB: An Introduction to the Eureka DAB System and a Guide to How It Works"*. BBC - Research & Development, June 2003.

[Gro02] Base Station Working Group. "Base Station System Structure". Technical report, SDR Forum, January 2002.

[Gue07] J. Guenin. "Jim Hoffmeyer on Overview of SCC41". http://www.scc41.org/, September 2007.

[Hay00] S. Haykin. *"Communication Systems"*. Wiley, 4 edition, May 2000.

[HD03] L. Huang and Z. Dong. "The Implementation of Estimation and Correction of Carrier Frequency Offset of COFDM System in DAB Receiver". *Proceedings. 5th International Conference on ASIC*, 2:890–893, October 2003.

[HL01] W. Hoeg and T. Lauterbach. *"Digital Audio Broadcasting: Principles and Applications"*. John Wiley & Sons, Ltd, March 2001.

[HSH99] Y. Huang, C. Sheu, and C. Huang. "Joint Synchronization in Eureka 147 DAB System Based on Abrupt Phase Change Detection". *IEEE Journal on Selected Areas in Communications*, 17(10):1770–1780, October 1999.

[III00] J. Mitola III. *"Cognitive Radio: An Integrated Agent Architecture for Software Defined Radio"*. PhD thesis, Department of Teleinformatics, Royal Institute of Technology (KTH), Stockholm, Sweden, May 2000.

[IM99] J. Mitola III and G. Q. Maguire. "Cognitive Radio: Making Software Radios More Personal". *IEEE Personal Communications*, 6(4):13–18, August 1999.

[LP01] L. Litwin and M. Pugel. "The Principles of OFDM". *RF Signal Processing*, January 2001.

[Man07] N. Manicka. "GNU Radio Testbed". Master's thesis, Faculty of the University of Delaware, 2007.

[McG08] S. McGarrity. *"Introduction to Object-Oriented Programming in MATLAB"*. The MathWorks, Inc, March 2008.

[Mit95] J. Mitola. "The Software Radio Architecture". *IEEE Communications Magazine*, 33(5):26 – 38, 1995.

[Mor00] N. Morinaga. *"Wireless Communication Technologies - New Multimedia Systems"*. Kluwer Academic Publishers, 2000.

[Mul08] A. Muller. "DAB - Software Receiver Implementation". Master's thesis, Department of Information Technology and Electrical Engineering, June 2008.

[Nal07] Nallatech Limited. *"XtremeDSP Development Kit-IV Reference Guide"*, 3 edition, July 2007.

[PM06] J. Proakis and D. Manolakis. *"Digital Signal Processing - Principles, Algorithms and Applications"*. Prentice Hall, 4 edition, April 2006.

[Sil08] A. Silva. "Geração de Sinais Vectoriais Baseada em FPGA". Master's thesis, Universidade de Aveiro, 2008.

[SL05] H. Schulze and C. Lueders. *"Theory and Applications of OFDM and CDMA: Wideband Wireless Communications"*. John Wiley & Sons, Ltd, July 2005.

[Sli01] M. Sliskovic. “Carrier and Sampling Frequency Offset Estimation and Correction in Multicarrier Systems”. 2001.

[TTT+96] K. Taura, M. Tsujishita, M. Takeda, H. Kato, M. Ishida, and Y. Ishida. “A Digital Audio Broadcasting (DAB) Receiver”. *IEEE Transactions on Consumer Electronics*, 42(3):322 – 327, August 1996.

[Xil06a] Xilinx, Inc. *“Programmable Logic Design - Quick Start Handbook”*, June 2006.

[Xil06b] Xilinx, Inc. “Virtex-4 SX 35 XtremeDSP Development Kit for Digital Communication Applications”. *DSP Magazine*, pages 60–61, May 2006.

[Xil07] Xilinx, Inc. *“Virtex-4 Family Overview”*, ds112 (v3.0) edition, September 2007.

[Xil08a] Xilinx, Inc. *“Fast Fourier Transform v6.0 - Product Specification”*, September 2008.

[Xil08b] Xilinx, Inc. *“System Generator for DSP Getting Started Guide”*, release 10.1 edition, March 2008.

[Xil08c] Xilinx, Inc. *“System Generator for DSP Reference Guide”*, release 10.1 edition, March 2008.

[Xil08d] Xilinx, Inc. *“System Generator for DSP User Guide”*, release 10.1 edition, March 2008.