FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FPGA implementation of a baseband processor for FBMC transmision

Miguel Nuno Marques Vaz de Carvalho

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Advisor: João Canas Ferreira

Co-Advisor: Mário Lopes Ferreira

June 25th, 2017

Resumo

Nos últimos tempos a investigação na área das telecomunicações tem estado focada nos sistemas de 5ª geração (5G). Estes sistemas necessitam de apresentar melhor performance que os atuais sistemas de 4ª geração (4G) em diferentes métricas, como por exemplo uma utilização mais eficiente do espectro de frequência. Uma das soluções propostas para cumprir estes novos objetivos é o uso de uma *waveform* diferente, *Filter Bank Multicarrier* (FBMC), ao invés da usada atualmente, *Orthogonal Frequency-Division Multiplexing*, já que apresenta melhores resultados de eficiência espectral.

As FPGAs constituem uma boa plataforma de desenvolvimento, pois têm um bom equilíbrio entre poder computacional, flexibilidade e consumo de potência. Assim, são vistas como uma boa plataforma para o implementações em hardware de processadores de banda base. Assim, o objetivo deste projeto é desenvolver e validar uma implementação em FPGA de um processador banda base para transmissão FBMC. Tendo em vista este objetivo, um estudo do estado da arte também foi feito, analisando diversas propostas de arquiteturas e os seus resultados. Da informação resultante deste estudo foi escolhida uma arquitetura para ser implementada. Uma vez implementado, o sistema foi validado através de comparações com um modelo em *software* já existente.

A performance do sistema foi medida através de diferentes métricas, tal como a utilização de recursos do sistema, o consumo de potência e o débito do sistema. Com estes resultados, comparando com os de outras implementações de sistemas FBMC e OFDM foi possível analisar a qualidade da implementação. O sistema obteve boa performance em termos de utilização de recursos e também do seu consumo de potência, quando comparado com outras implementações de sistemas FBMC e também de OFDM. O débito do sistema está também na mesma gama do obtido em sistemas OFDM.

Foi também possível concluir que os sistemas FBMC apresentam resultados competitivos quando comparados com sistemas OFDM e a *waveform* FBMC consiste, assim, numa alternativa válida para sistemas 5G.

Abstract

Recently, research in the area of wireless communications has been focused in fifth generation (5G) systems. They present new demands, with higher requirements than existing fourth generation (4G) ones, such as a more efficient use of the frequency spectrum. One of the proposed solutions to fulfil this requirement is the use of a new waveform, Filter Bank Multicarrier, as opposed to the existing Orthogonal Frequency-Division Multiplexing one, as FBMC achieves better spectral efficiency.

FPGAs provide a good development platform as they present a good relationship between their processing power, flexibility and power consumption. As a consequence, they are seen as a good platform for the development of a hardware implementation of baseband processors, as is the case. Therefore, the goal of this project was to develop and validate an FPGA implementation of a baseband processor for FBMC transmission. With this implementation in mind a study of the current state of the art was done, assessing several architecture proposals and their results. From the information yielded from this study the architecture to be implemented was chosen. Once implemented, the developed system was validated through a comparison with an existing software model.

The performance of the developed implementation was measured through different key metrics, such as its resource usage, power usage and its throughput. From these results, and comparing with those of other FBMC and OFDM implementations it was possible to assess the quality of the implementation. This system presents good resource usage and power consumption results when compared to other FBMC and OFDM implementations. Its throughput is also comparable to that of existing OFDM systems. It is also possible to conclude that FBMC systems obtain competitive results when compared to OFDM systems and that FBMC is a viable candidate waveform for 5G systems.

Acknowledgements

I want to thank my advisors, Professor João Canas Ferreira and MSc Mário Lopes Ferreira for their guidance and advice throughout the development of this project.

I wish to express my gratitude to my parents for the support and encouragement throughout my academic and personal life.

Lastly, I want to thank my friends and colleagues for their camaraderie throughout these years.

Miguel Carvalho

"Life grants nothing to us mortals without hard work."

Horace

Contents

List of Figures

List of Tables

Abbreviations

4G	Fourth Generation
5G	Fifth Generation
BER	Bit Error Rate
BRAM	Block RAM
CP	Cyclic Prefixing
DFT	Discrete Fourier Transform
DUT	Device Under Test
FBMC	Filter Bank Multicarrier
FF	Flip Flop
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
FS	Frequency Spreading
FSM	Finite State Machine
HDL	Hardware Description Language
ICI	Inter Carrier Interference
IFFT	Inverse Fast Fourier Transform
ISI	Inter Symbol Interference
LUT	Look Up Table
OFDM	Orthogonal Frequency-Division Multiplexing
OOB leakage	Out-of-band leakage
OQAM	Offset Quadrature Amplitude Modulation
PAM	Pulse Amplitude Modulation
PPN	Polyphase Network
QAM	Quadrature Amplitude Modulation
RAM	Random Access Memory

Chapter 1

Introduction

1.1 Context

In recent years research in the area of wireless communications has been focused on fifth generation (5G) mobile systems, more commonly known as 5G systems, and how they can be implemented. These systems aim to improve the previous generation (4G)'s performance and adapt to more recent needs, imposed by recent technologies and applications. These include high speed or high mobility scenarios and an increasing number of users that require an even more efficient use of the frequency spectrum, since this is a limited resource.

To comply with these new requirements several waveforms have been proposed. One of the stronger candidates is Orthogonal Frequency-Division Multiplexing (OFDM), which is already in use in 4G systems and, as such, is easier to implement and transition from 4G to 5G. OFDM's other main benefit relates to its implementation being less complex than its alternatives. Another strong candidate is Filter Bank Multicarrier (FBMC), as it presents good results in key metrics, such as its higher spectral efficiency.

Like other multicarrier techniques, FBMC works by splitting the frequency channel into smaller subchannels and transmitting the signals using each of those subchannels. Its distinguishing characteristic from other multi-carrier schemes comes from the fact that it includes filter banks both in the receiving and the transmitting sides.

Filter Bank Multicarrier (FBMC) is a technique whose development traces back to the 1960's, when filter banks were first proposed to be used with amplitude modulation signals. It has taken decades for it to be considered as a viable implementation due to its higher computational complexity. With the evolution of hardware it is now a viable and relevant waveform. This waveform's benefits include a higher spectral efficiency but also better performance in high speed scenarios and improved robustness to mobility.

This higher spectral efficiency is very important in the context of 5G communications and is achieved through reduced out-of-band leakage. Out-of-band leakage consists in the transmission of signal outside the desired frequency band. This causes a reduction in spectral efficiency as, due to this noise being transmitted outside the desired signal bands, larger guard bands are

needed to prevent interference. The use of larger guard bands implies a reduction of the available spectrum to transmit information. This effect has been measured and it has been concluded that FBMC presents a 20% gain in spectral efficiency when compared to OFDM [1]. Implementations of systems of this kind have been studied and there are some simulations with results for its expected efficiency and general performance, although hardware implementations are scarce. This makes for an absence of real measurements. A hardware implementation of this type of system is, therefore, a useful project, to be able to study this type of systems more in detail.

1.2 Goals and Tasks

The main goal of this project is to implement a baseband processor for FBMC transmission, in a field programmable gate array (FPGA). The developed implementation should also be validated to ensure its correct function. With a functional version of this implementation a comparative study of the performance of FBMC systems can be done.

To achieve this goal a study of 5G systems and its candidate waveforms in particular will be done. Existing architectures of OFDM and FBMC systems will be studied, so as to understand their characteristics and to draw conclusions as to which alternative is the best option for 5G. In the end, the characteristics of the implemented design will be compared with those in the state of the art in order to assess its validity and the relevance of FBMC as a candidate waveform.

The FBMC baseband processor will be represented using a Hardware Description Language (HDL), Verilog in this case. Once validated this implementation will be synthesised and implemented, targeting Xilinx's Zynq-7000 FPGA. With a functional implementation it will be possible to analyse its resource usage as well as its performance details, such as its throughput or transmitting power.

1.3 Contributions

The baseband processor for FBMC transmission produced in this project allows the study of an architecture different from existing ones. The study of this architecture enables some conclusions to be drawn regarding the performance of this architecture when compared to that of existing FBMC systems and of OFDM ones. As will be seen in chapter 2 several architectures have been proposed for systems of this kind, however most of them are either aimed for simulation or are high level proposals. There is a lack of post-synthesis or post-implementation results. This project provides this data for a specific architecture to be chosen and will help fill this void.

From the initial study of FBMC and OFDM candidates and an analysis of the current state of the art an idea about which waveform is the most suited for the new 5G systems can already be formed. The data resulting from the implementation, namely its usage and timing performance metrics helps assessing whether FBMC is a suitable and legitimate choice for these new systems. The conclusions stemming from this project provide additional data to take into account when choosing the waveform to be used in 5G communications.

1.4 Structure

This document is divided into 6 chapters. Initially, in chapter 1 the project is presented, contextualising the problem and describing the project's main goals and its most important contributions. Chapter 2 explains the most important theoretical notions that should be understood in order to fully comprehend the work described further. A thorough description of the current state of the art is also done in this chapter, covering several current implementations for FBMC systems. The implemented architecture is presented in chapter 3, through a description of its constituting blocks. The validation strategy to ensure the correct function of the developed system is also described, along with the methodology employed. Chapter 4 lists the practical implementation details of the developed system. This consists of a low level view of the different blocks developed and of the full system. In this chapter it is possible to see the more important details of the system. The outcomes of the project are presented in chapter 5, both listing the practical results of the implementation, such as its usage or power measurements. Some important conclusions about the systems are also listed, taking into account some experiments performed with the system's parameters. These results are analysed through a comparison with other systems' architectures. Finally, chapter 6 presents the most important conclusions about the project.

Chapter 2

Background

In this chapter the fundamental concepts related to this work will be presented. A review of the current state of the art in the implementation of Filter Bank Multicarrier systems will also be done by presenting different alternatives and their characteristics, as well as describing and comparing various implementations. The blocks employed by these implementations will also be studied, so as to understand their function and how they can be developed.

After this study it is possible to draw some conclusions regarding the performance of the different proposed implementations and analyse which of them might be the best option to implement in the current project. These conclusions are presented and consequently some decisions are made about the architecture to be implemented.

2.1 Filter Bank Multicarrier waveform

Filter Bank Multicarrier (FBMC) is a waveform technique which intends to improve the performance of the previous technique in use, Orthogonal Frequency-Division Multiplexing (OFDM). The first developments for this technology date back to the 1960's when a bank of filters was used to process a parallel set of Pulse Amplitude Modulation (PAM) symbol sequences. However, only recently has the interest around it increased, due to its perceived higher computational complexity.

As the name implies, FBMC is used for digital multicarrier modulation. The main benefits in the utilisation of this type of modulation are the minimisation of inter symbolic interference (ISI) and inter carrier interference (ICI) that might be introduced when using channels with varying gain in their frequency band. This is possible because the use of multicarrier modulation eases the task of channel equalisation, which is used to diminish the effects of the ISI and ICI.

An FBMC waveform is obtained through the transmission of data through a filter bank, which is the main difference when compared to the existing OFDM waveform. While in OFDM systems an IFFT coupled with Cyclic Prefixing (CP) is used, in FBMC this is replaced by a synthesis filter bank. This change allows it to present better performance than its multicarrier competitors, due to the ability to perform per subcarrier filtering.

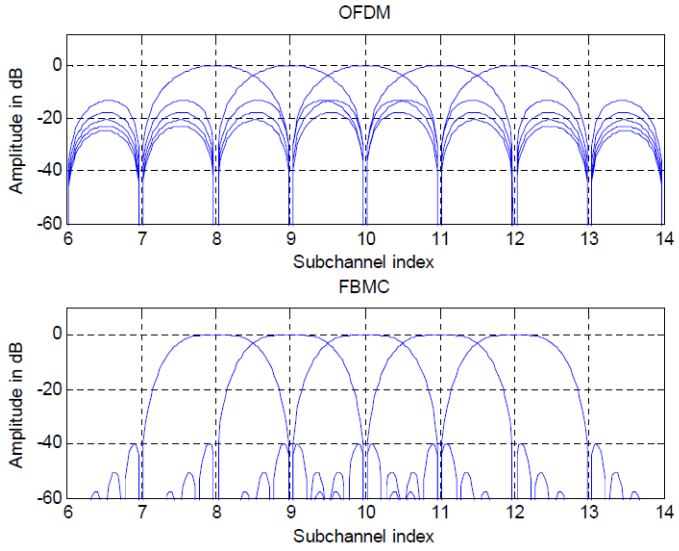

Figure 2.1: FBMC and OFDM's frequency response (source:[2])

Another of FBMC's most important advantages is its higher spectral efficiency, when compared with OFDM systems. The previously mentioned absence of cyclic prefixing in FBMC systems allows it to increase this performance, as CP originates additional overhead and, as a consequence, a loss in bandwidth efficiency. To add to this, FBMC shows a severe reduction in out-of-band leakage. Out-of-band leakage (OOB leakage) consists in the transmission of signal outside the designated frequency band and it plays a big part in the system's spectral efficiency. Having higher OOB leakage introduces the need for guard bands which are wasteful towards the spectrum use.

OFDM systems present strong sidelobes in their frequency response, while FBMC aims to avoid these. Both FBMC and OFDM systems' frequency responses are shown in figure 2.1.

As can be seen in this figure, sidelobes in FBMC are considerably smaller than the ones obtained with OFDM. In FBMC a subchannel only overlaps its adjacent subchannels, making it easier to achieve two independent multicarrier signals simply by leaving an empty subchannel between them. This result also helps to avoid inter carrier interference. The higher selectivity and spectral containment of FBMC subchannels also contribute to achieving good resistance when faced with narrowband interference. It has been calculated in [1] that FBMC presents a 20% gain in spectral efficiency when compared to OFDM.

The filter bank used in FBMC consists of a set of filters, that, when combined together, form the impulse response of the prototype filter. The choice of this prototype filter is one of the characteristics of FBMC which allows it to present better frequency containment. The prototype filter, parametrised by the overlapping factor K, can be chosen in a way to achieve higher selectivity and low adjacent channel leakage. The use of longer and well-shaped filters makes the non-adjacent subcarriers almost perfectly separated. This result varies with the filters' overlapping factor K. As mentioned above, this result contributes to one of FBMC's most important characteristics, its high

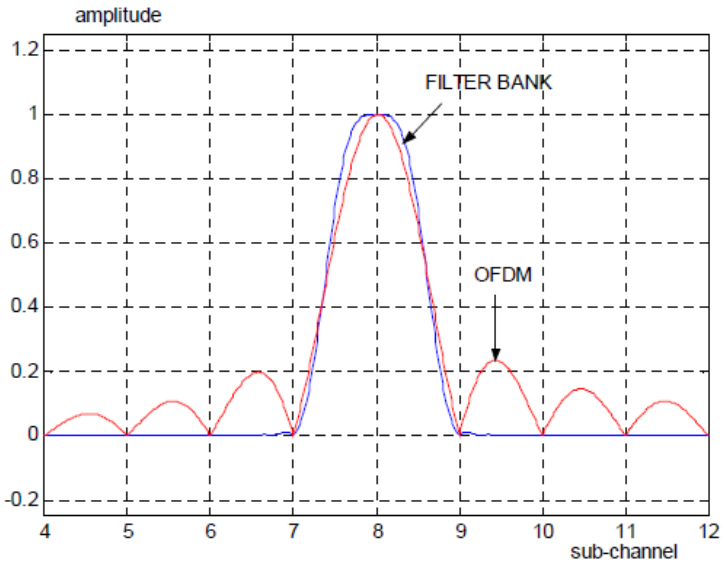

Figure 2.2: FBMC and OFDM's frequency response around a single subchannel(source:[2])

spectral efficiency: by having the energy concentrated in the frequency range of a subcarrier the wasteful guard bands can be little to non-existent.

FBMC's higher selectivity when compared with OFDM can easily be seen when analysing the frequency response around a single subcarrier. This is represented in figure 2.2.

From this figure it is possible to observe that OFDM exhibits ripples in the frequency domain, while FBMC's frequency response has little to no amplitude outside the desired frequency range.

The use of near-perfect subcarrier filters also allows FBMC to avoid multiple access interference (MAI) without the need to perform synchronisation. This is another result that helps FBMC stand out from OFDM, as in OFDM to achieve no MAI some cancellation is needed, which greatly increases its complexity.

Mathematically, FBMC's baseband signal can be represented by the following expression:

$$s(t) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} j^{(m+n)mod2} d_{m,n} g(t - n\tau_0) e^{j2\pi m f_0 t} \tag{2.1}$$

with M corresponding to the number of subchannels; $d_{m,n}$ to the QAM data transmitted on the mth subchannel of the nth FBMC symbol, f_0 represents the frequency domain spacing between two adjacent subchannels; τ_0 corresponds to the time offset between FBMC symbols; $g_{m,n}(t)$ corresponds to the filter of the mth subchannel of the nth symbol.

From this expression it is possible to identify the OQAM modulation, as the signal alternates between real and imaginary; it is also possible to see that the transmitted signal is given by the sum of the application of its subcarriers to a filter, $g(t)$.

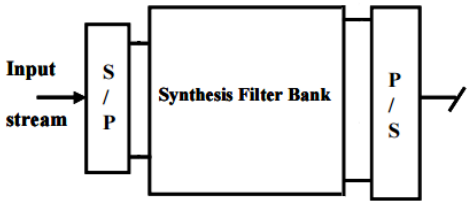

Figure 2.3: FBMC's generic architecture

2.2 FBMC architecture

In regards to its architecture, FBMC's general constitution is, as was seen above, made out of an initial modulation block, followed by a synthesis filter bank and a parallel to serial conversion. This generic architecture is represented in figure 2.3.

It is possible to distinguish between two major FBMC alternatives according to the used modulation: Quadrature Amplitude Modulation (QAM) based implementations and Offset QAM (OQAM). FBMC implementations making use of QAM modulation present a lower spectral efficiency as orthogonality between subcarriers is obtained through a reduction in the frequency domain overlap; OQAM ones, on the other hand obtain orthogonality in the real domain only, which guarantees maximum spectral efficiency. The use of OQAM modulation also has the impact of doubling the processing rate of the system. As a consequence of its better spectral performance OQAM based FBMC implementations are considered as the baseline modulation for FBMC and this will be the variant more in focus throughout this document.

The generic architecture of an FBMC/OQAM follows that of a generic FBMC one and adds an OQAM modulator. Therefore, one benefit of FBMC is its similarity to OFDM systems. The main difference, as expected, lies in the introduction of filter banks in FBMC. In the case of FBMC/OQAM there is also a need to introduce an OQAM processing block, as OFDM employs QAM modulation. Figure 2.4 shows the differences and the modules common to both systems.

In this figure the vertically dashed blocks are used in OFDM systems, while the horizontally dashed ones are used in FBMC transmitters. The remaining blocks are common to both FBMC and OFDM implementations. Although an IFFT block is used in both waveform architectures,

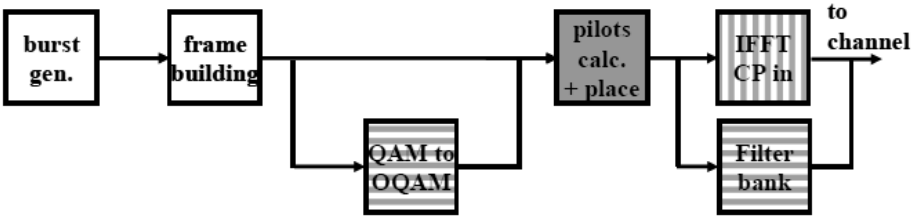

Figure 2.4: Transmission chain OFDM and FBMC (source:[1])

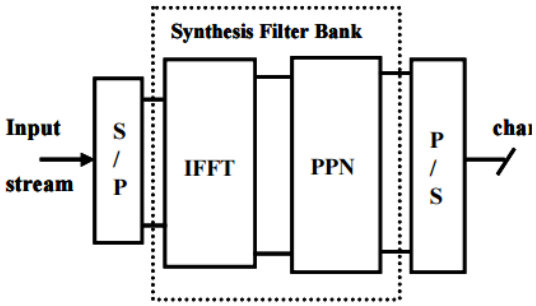

Figure 2.5: PPN-FBMC architecture (source:[1])

the inclusion of cyclic prefixing (CP) is only done in OFDM, as was mentioned previously in this chapter. This is one of the key characteristics of FBMC that allows it to present better spectral efficiency. This figure shows that, in order to transition from OFDM to FBMC few adjustments are needed, as there are several blocks in common and the system follows the same overall layout. This makes for an easier introduction of new systems, as the changes needed to be made in the systems' architecture is minimised.

Regarding FBMC/OQAM systems there are two main alternatives for its implementation, according to the way the filter bank is implemented. This leads to the two main alternatives for FBMC/OQAM implementations: if the filter bank is implemented making use of an inverse FFT operation, followed by a poly-phase network the system corresponds to the PPN-FBMC alternative; if it is generated through frequency spreading followed by an inverse FFT operation performed in the frequency domain then it is an FS-FBMC system.

Typical high level architectures for both alternatives can be seen in figures 2.5 and 2.6.

As can be seen in figure 2.5, in the PPN-FBMC alternative the input OQAM symbols are fed to an IFFT block with size N . The resulting data is then applied to a polyphase network; to obtain the final signal to be transmitted the parallel values output by the PPN are serialised and sent to the channel. The IFFT and PPN correspond to the system's synthesis filter bank.

The polyphase block used in this variation of FBMC implementations consists in a set of filters which, when combined, form the impulse response of a prototype filter which can be configured in such a way to achieve better localisation in the frequency domain. The shape and length of the prototype filter will change the time and frequency domain localisation.

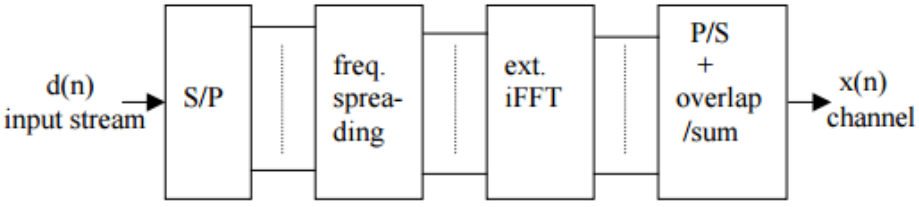

Figure 2.6: FS-FBMC architecture (source: [3])

Figure 2.6 represents a general frequency spreading implementation of FBMC. It is possible to observe that the architecture of an FS-FBMC is quite similar to that of a PPN-FBMC system, except in the composition of its synthesis filter bank. Instead of an IFFT followed by a PPN this alternative employs a frequency spreading block, because, as mentioned previously, frequency spreading filter the OQAM symbols in the frequency domain and an extended IFFT of size $K \times N$. Both these blocks constitute the chain's synthesis filter bank. Finally, the signal to be transmitted is obtained through an overlap and add block. It should be noted that there isn't an OQAM block in the depicted architecture neither in figure 2.6 nor figure 2.5. If an FS-FBMC or a PPN-FBMC system was to be implemented using OQAM this block should be added in the beginning of the chain.

The frequency spreading block used in FS-FBMC implementations consists of a filter designed through the frequency sampling technique. Using this technique the number of multicarrier symbols which overlap in the time domain is given by K, the overlap factor of the system. Using a filter of this kind each data symbol applied to a sub-channel corresponds to several frequency components, depending on the K. With this alternative the prototype filtering is implemented in the frequency domain by using an IFFT of size $K \times N$. To obtain the output signal the output of the IFFT is serialised through the application of an overlap and add operation. In this operation K of the IFFT's samples are added together to form to output, when the transient period is finished.

Comparing both the frequency spreading and polyphase network FBMC alternatives it can be concluded that FS-FBMC presents some additional benefits: it has better performance in timing offset compensation and achieves high equalisation. It also presents high performance equalisation capability. An additional benefit of this alternative is its flexibility, as various schemes can be implemented for a particular IFFT; a system with $K = 4$ can be reconfigured into a system with $K = 2$ with double the subchannels N. These benefits come at the cost of its increased computational complexity when compared with PPN-FBMC, due to the introduction of an IFFT block of size $K \times N$, opposed to a block of size N in PPN-FBMC. Another item in favour of FS-FBMC is that it simplifies the concept of FBMC, as it consists of an IFFT block with simple processing at its input and output.

2.3 FBMC blocks

Despite some variations, the blocks employed to produce an FBMC transmitter are usually the same in a typical FBMC architecture. OQAM modulation blocks are present in all, as they are needed to modulate the input information; according to the alternative the systems then use either a polyphase network coupled with an IFFT block or an IFFT block and a frequency spreading block. For both an IFFT block and an FIR filter need to be implemented. Lastly, to serialise the output either a parallel to series converter is used, in the case of PPN-FBMC, or an overlap and add block is implemented, for FS-FBMC systems. These will be presented in further detail in the following subsections.

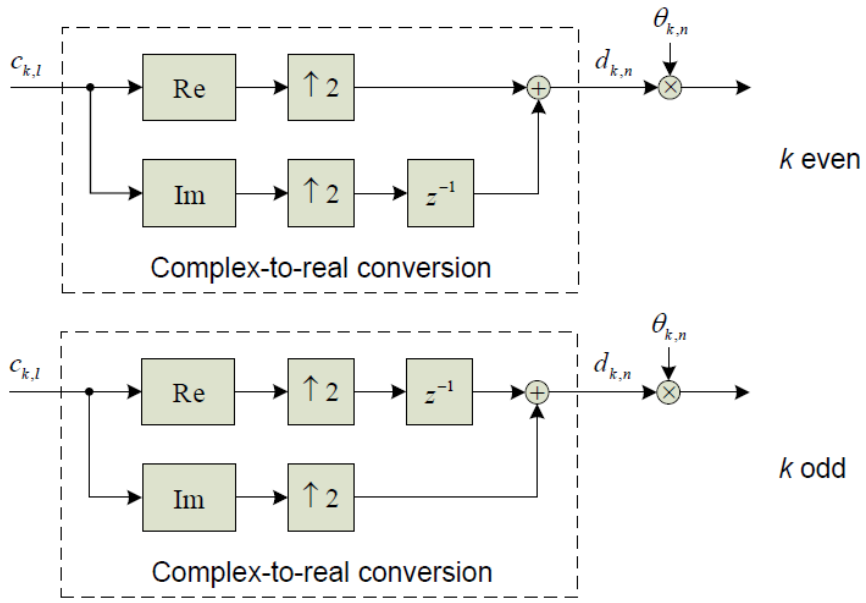

Figure 2.7: OQAM pre-processing architecture (source:[2])

2.3.1 OQAM pre-processing

Earlier in this document it was seen that the most spectrally efficient implementation of FBMC makes use of OQAM. Therefore, both FS-FBMC and PPN-FBMC make use of an OQAM modulator in order to perform the pre-processing operations which modulate their input data.

Offset Quadrature amplitude modulation (OQAM) is a popular modulation scheme similar to Quadrature Amplitude Modulation with the added characteristic of its values being either purely real or purely imaginary.. It is composed of two carrier waves with a 90^{o} phase shift, which transmit data through variation of their amplitude.

OQAM modulation is done in a processing block which performs a complex-to-real conversion. In this operation the real and imaginary parts of a symbol $c_{k,l}$ form two different new symbols, $d_{k,2l}$ and $d_{k,2l+1}$. The resulting values are then multiplied by a complex sequence which can be obtained from the following expression:

$$\theta_{k,n} = j^{(k+n)} \tag{2.2}$$

where k represents the subcarrier index with $k \in [0; M-1]; k = 0$ corresponds to the center subcarrier and n represents the time index at OQAM subsymbol rate.

This leads to an alternation between purely real and imaginary values in subchannel signals. The corresponding architecture can be seen in figure 2.7.

Besides k and n, which have already been defined in equation 3.2, l represents the time index at OQAM symbol rate. After this pre-processing values are either purely real or imaginary.

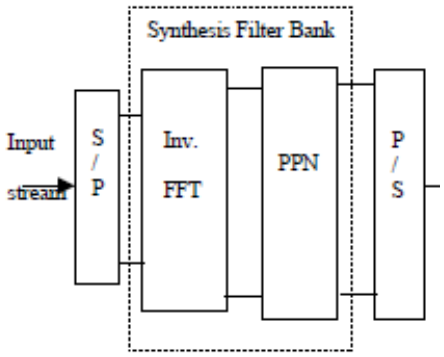

Figure 2.8: Architecture of a synthesis filter bank (source:[2])

2.3.2 Filter Bank

As was mentioned before, one of the distinguishing factors in FBMC's architecture when compared to other multicarrier modulations is the introduction of a filter bank in both the transmitting and receiving sides of the system. This is one of the most important blocks of the entire chain and, as such, its characteristics must be carefully chosen in order to satisfy the system's requirements.

Earlier in this chapter, when the distinction was made between FS-FBMC and PPN-FBMC it was seen that this distinction primarily came from the implementation of the filter bank. There are two main alternatives for its implementation, both of which will be detailed further.

The general architecture of a synthesis filter bank in a polyphase network FBMC system can be seen in figure 2.9.

As can be observed, in this alternative, a synthesis filter bank is comprised of a transform block, which performs an IFFT operation and a polyphase network block, which in turn consists of a set of digital filters. This set of digital filters, as a whole, forms a prototype low-pass filter. After the transform and polyphase network blocks a parallel to serial conversion is performed.

In the transmitter side of an FBMC system a synthesis filter bank is used, while in the receiving side it is contains an analysis filter bank.

An implementation for a synthesis filter bank structure is presented in [2] and can be seen in figure 2.9.

This implementation is one of a direct form synthesis filter bank, comprised of M upsamplers and M synthesis filters. The output signal is given by the addition of all subsignals. However, this implementation isn't computationally efficient, due to the high sampling rate introduced by the filtering operations, which leads to several unnecessary calculations.

To improve the performance of this block a different implementation for a synthesis filter bank has been proposed in [2], using polyphase structures. Polyphase structures are one of various possible implementations for filter banks. These also include lapped transforms or lattice structures. All these possible implementations consist of a filter section and a transform section, although only the polyphase structures are applicable when working with nearly perfect reconstruction filters.

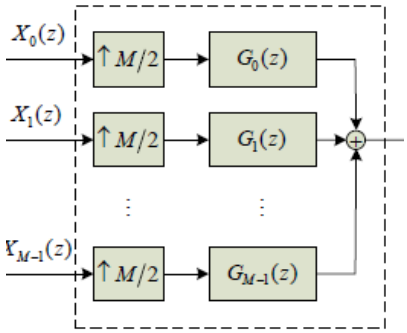

Figure 2.9: Implementation of a synthesis filter bank (source:[2])

In [2] the implementation proposed to improve the performance of the already presented direct form of a synthesis filter bank uses polyphase structures. This choice will allow a major reduction of the computational complexity as these function with lower sampling rates and so avoid unnecessary calculations. This alternative, more efficient implementation can be seen in figure 2.10.

As mentioned before, the characteristics of each block should be carefully calculated, to ensure the correct function of the system and to follow the set functional requirements. Some of the most important choices relate to the prototype filter used.
The prototype filter used in the implementations proposed in [2] is chosen to be a causal real-valued symmetric FIR filter with high frequency selectivity.

2.3.3 FIR Filter

Both the alternatives in the implementation of filter banks include the use of FIR filters in their architecture. FIR filters, meaning finite impulse response filters, are a type of digital filter used in digital signal processing. Filters of this type have an impulse response of finite duration which lasts N+1 samples, N being the order of the filter, with its output being the result of a weighted sum of the input values. A standard implementation architecture of an FIR filter is shown in figure 2.11.

An alternative for this architecture is the FIR's transposed form, which is preferred in the context of hardware development[5]. This alternative architecture is presented in figure 2.12.

The filter banks used in [2] have the following characteristics: The number of subchannels is arbitrary, although it is common to use a power of 2, as it is one of the conditions for some efficient IFFT algorithms; The filter length should be given by one of the following equations:

$$L = KM \qquad L = KM + 1 \qquad L = KM - 1 \tag{2.3}$$

where L is the filter length, K the overlap factor and M the number of subcarriers used. The roll-off parameter shall be set as $\alpha = 1$, so that the transition band of a subchannel is limited at the center

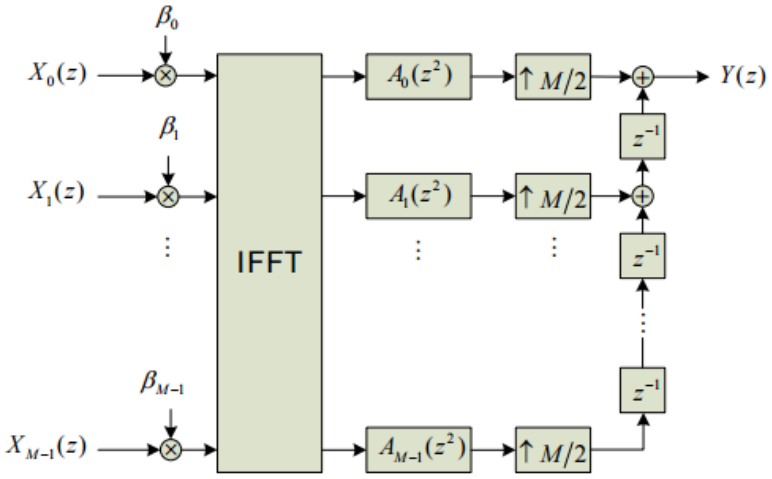

Figure 2.10: Synthesis Filter Bank architecture (source:[2])

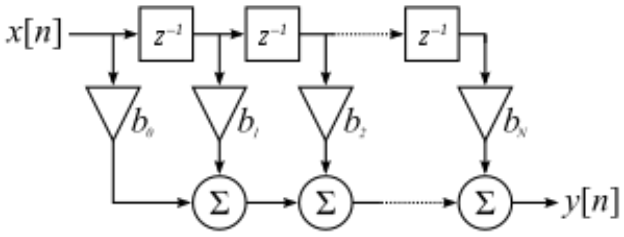

Figure 2.11: Architecture of an FIR filter (source: [4])

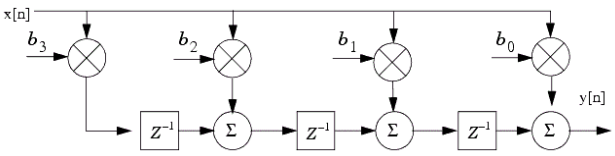

Figure 2.12: Transposed architecture of an FIR filter (source: [4])

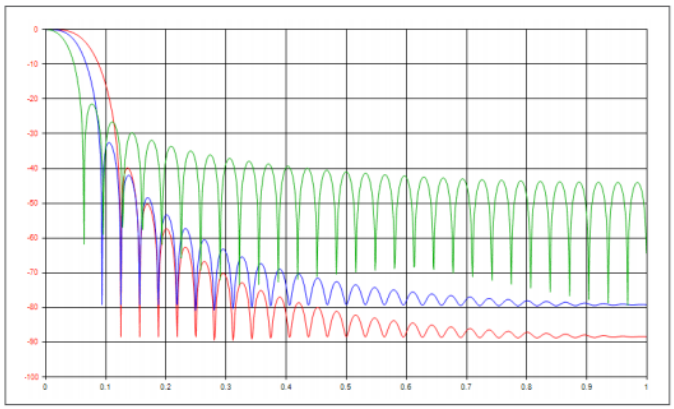

Figure 2.13: FBMC's spectral profile's variation with K (source:[6])

of the adjacent subchannel. This guarantees that only adjacent subchannels interfere with each other.

The value chosen for the overlap factor K is directly related to the achieved selectivity and as a consequence, the out-of-band leakage. In figure 2.13 it can be observed that as K decreases the out-of-band performance worsens, until a result similar to that of OFDM is seen.

A study of the impact of the overlap factor on some important transmission performance metrics was done in [7] and can be seen in table 2.1:

As was expected, the performance of the system improves with the increase of the overlap factor, as we can clearly observe that all interference metrics decrease substantially with K. These results were obtained for a filter length of $L = KM - 1$ and with $M = 64$.

According to [2] the overlap factor used for the filters must be 3 or higher.

Having defined the values of L, M and K to design the prototype filter one can use the frequency sampling technique. This technique starts by obtaining L target values in the frequency domain and then obtaining the prototype filter coefficients using the following expression:

$$h(m) = 1 + 2\sum_{k=1}^{K-1}(-1)^k H(k/L)cos(2\pi km/L)$$

$$h(0) = 0$$

As an example, for $L = 2048$; $M = 512$ and $K = 4$ the prototype filter obtained exhibits the frequency response seen in figure 2.14.

Table 2.1: Impact of the overlap factor on transmission performance (source: [7])

Performance metric	$K = 2$	$K = 3$	$K = 4$
ISI	-45.1 dB	-53,3 dB	-67,8 dB
ICI	-30,2 dB	-43,9 dB	-68,7 dB
TOI = ISI + ICI	-30,0 dB	-43,4 dB	-65,2 dB

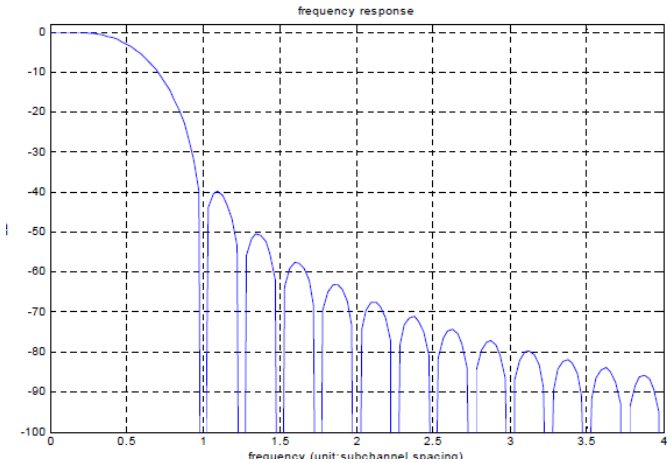

Figure 2.14: Example of a prototype filter's frequency response (source:[2])

In this graph the x axis is shown in units of subchannel spacing, so one can observe that above 2 subchannel spacings the attenuation exceeds 60 dB.

2.3.4 Fast Fourier Transform - FFT

A Fourier transform is an operation which converts a signal from its time domain to the frequency domain. This operation is defined as

$$F(k) = \int_{-\infty}^{\infty} f(x) e^{-2\pi jkx} dx \tag{2.4}$$

However, this formula applies to continuous signals. In the context of digital signal processing the discrete version of the Fourier Transform is more common, which is given by

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-\frac{j2\pi}{N} nk}, k \in [0, N-1] \tag{2.5}$$

The Fast Fourier Transform (FFT) is an alternative that computes the Discrete Fourier Transform (DFT) of a sequence in a more efficient way than applying the definition.

FFT algorithms are able to calculate this transform in a more efficient way by factorising the DFT matrix into a product of factors. There are several of these algorithms, which follow the same method: they divide the original DFT into several, smaller DFTs easier to compute.

The most common FFT algorithms are the Cooley-Tukey algorithm, the Good-Thomas algorithm and the Winograd algorithm. These can be compared and some conclusions can be drawn regarding the implementation of each one.

Some important characteristics to be analysed for each algorithm are the number of multiplications and additions that are needed to calculate the result, as well as the computation effort.

As can be seen in table 2.2, although the Cooley-Tukey algorithm presents the worst performance in terms of number of multiplications it is the one with the best performance in terms of

Table 2.2: Relative comparison of FFT algorithms (source:[5])

Property	Cooley-Tukey	Good-Thomas	Winograd
Number of multiplications	bad	fair	best
Number of additions	fair	fair	fair
Index computation effort	best	fair	bad

index computation overhead. Considering other factors, such as the data and coefficient memory sizes or the run-time code length the Cooley-Tukey proves to be the best overall solution [5]. It is also the only algorithm applicable to transforms of any length and it presents the advantage of using a small butterfly structure.

It can, therefore, be concluded that this algorithm provides the best overall solution and should be the one used in the system to be developed.

2.3.4.1 FFT architectures

In what concerns the FFT algorithm's hardware implementation there are two main options: memory-based architectures and pipelined ones.

Memory-based architectures use memory banks to store the input values and the intermediate calculations that are performed. It uses few processing elements and, as such, presents the advantage of requiring a low amount of resources and circuit area. This architecture can be seen in figure 2.16.

The main downside of this architecture, however, is that it does not use parallel execution.

The pipelined architecture alternative solves this problem, as all FFT operations in a single stage are done using only one instance of the required butterfly and complex multiplier (rotator). This can be seen in the generic architecture presented in figure 2.16, where each block represents one butterfly and rotator and belongs to each stage of the FFT.

Although this architecture allows for better performance it has, however, the downside of requiring more resources and circuit area. This is due to the fact that in order to properly synchronise their operations each pipeline stage requires memory.

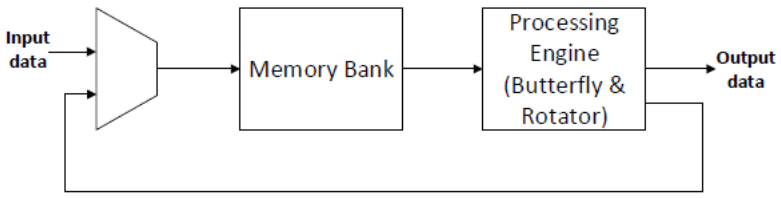

Figure 2.15: Memory-based FFT architecture (source:[8])

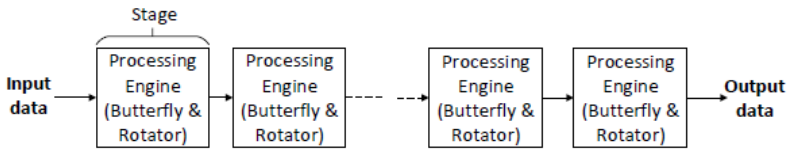

Figure 2.16: Pipelined FFT architecture (source:[8])

2.3.5 Overlap and Add

A block which is exclusive to the frequency spreading FBMC alternative is the overlap and add block. As was seen before the need for this block's use comes from the overlap in the FBMC symbols, which is a consequence of the frequency spreading operation performed in this alternative.

The number of symbols which overlap in the time domain is given by the overlapping factor K. To obtain the signal to be transmitted the overlapping FBMC symbols must be summed together.

This operation is depicted in figure 2.17, for $K = 4$. In this figure M consists of a sample.

2.4 FBMC implementations

Several implementations of FBMC systems have been proposed, with different computational complexities and with different results. They differ mainly in the parameters chosen, in the characteristics of each operation and in the organisation of the composing blocks, as they follow the generic structures already presented. A small description and review of some of these proposed implementations will be presented next. Initially some implementations of PPN-FBMC will be described followed by some implementations of FS-FBMC systems, so as to be possible to compare the different characteristics and their performances. This way it is possible to draw some conclusions about which one is better and most suited to be implemented in the scope of this project.

2.4.1 PPN-FBMC implementations

Three alternatives for a PPN-FBMC transmitter are proposed in [10]: A straightforward one, according to [2], as well as a computationally efficient implementation and an innovative technique with even less complexity.

The proposed straightforward implementation uses NK-IFFTs operating in parallel. With this architecture each symbol is calculated with an extended IFFT and the resulting data is overlapped to form the signal to be transmitted. This option is easier to implement, however it is not computationally efficient, as it requires larger IFFT operations occurring simultaneously. For this implementation an overlap factor of $K = 4$ is used and the length of the IFFT is set at $L = NK$ samples.

To reduce this computational complexity, [10] propose the use of two N-IFFTs, along with polyphase filtering. This new architecture can be seen in figure 2.18.

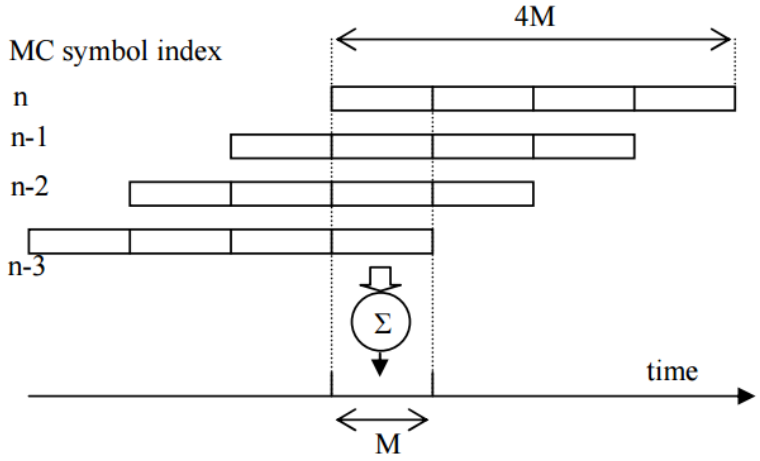

Figure 2.17: Overlap and Add operation (source:[9])

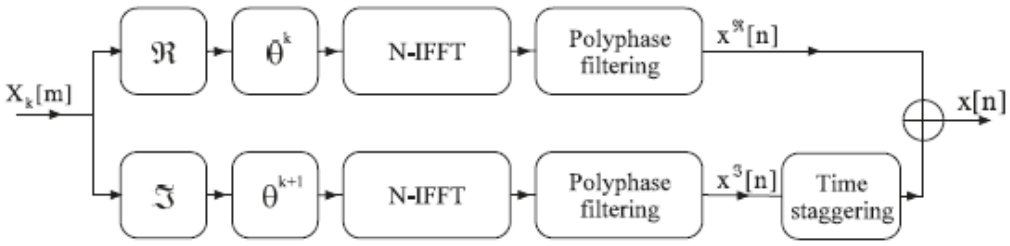

Figure 2.18: Implementation using two N-IFFTs (source:[10])

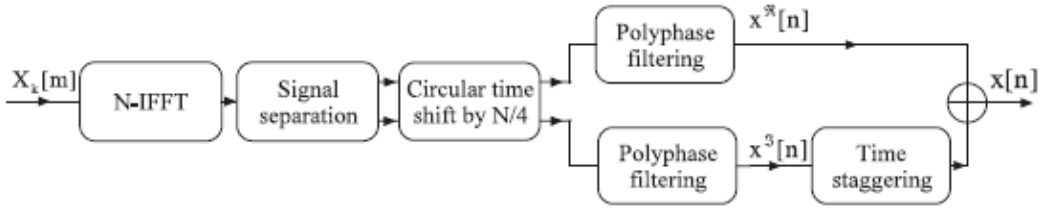

Figure 2.19: Implementation using one N-IFFT (source:[10])

This architecture improves the straightforward's approach complexity as it makes use FBMC's typical synthesis filter bank architecture. This allows the filtering to be done using an IFFT block and polyphase filtering, as opposed to the calculations using only IFFTs presented before.

Lastly, the final implementation proposed in [10] intends to further reduce the complexity by using a single N-IFFT along with polyphase filtering. It is observed that the complexity of an implementation can be greatly reduced by using additional signal processing blocks. The complexity of the proposed single N-IFFT implementation is calculated to be half that of the 2 N-IFFT. This implementation is represented in figure 2.19.

The reduction of complexity presented by this architecture comes from employing a single IFFT, as opposed to two, as in the previous one. To be able to use only one IFFT this architecture takes advantage of the property described in [11] which allows the computation of the discrete IFFT of two real functions simultaneously, with only one IFFT. This architecture has the increased advantage of allowing an OFDM transmitter to be easily reconfigured solely by adding certain signal processing blocks.

Also with the purpose of reducing the complexity of a transmitter's implementation an architecture is proposed in [12] that takes advantage of the pruned IFFT algorithm.

In order to reduce the number of computations, this architecture calculates only half of the IFFT outputs. To obtain the values of the remaining samples these are deduced from the calculated ones. This is possible due to the relation between the IFFT outputs. As an example, for a filter length of $L = KM$, the IFFT output of sample $u_n(2m+1)$ is obtained by

$$u_n(2m+1) = (-1)^n u_n^* \left(\frac{M}{2} - 2(m+1) \right) \tag{2.6}$$

knowing the value of the output of sample $u_n(m)$. From this result one can conclude that it is unnecessary to calculate the values of $\text{IFFT}_M^{even}/2$ and $\text{IFFT}_M^{odd}/2$, as they can be obtained through this relation. The calculation of the odd samples from the even ones resulting from the IFFT block is done in the reordering block. Besides this function, this block is also tasked with the parallelisation of two time-domain sequences, to feed the PPNs.

There is, however, the need to calculate the pruned IFFT twice, for real and imaginary OQAM symbols. The complexity of this operation is still half of that of a classic IFFT. At the end of the pre-processing block, the samples are parallelised for the two PPNs. We can see this architecture in figure 2.20.

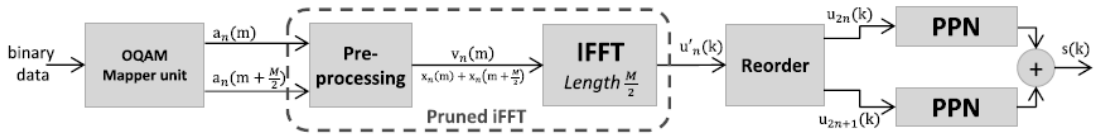

Figure 2.20: Implementation using pruned IFFT (source:[12])

In [12] two variations of this architecture are studied: one using a short prototype filter and one using the prototype filter proposed in [2]. An OFDM transmitter was also implemented, which can serve as a comparison to assess FBMC's performance. Both these alternatives were prototyped using a Zedboard and the post-synthesis results are presented. From these we can compare its performance to that of a typical FBMC transmitter, as well as to an OFDM one. These results were measured for an implementation using an IFFT size of $M = 512$. It is possible to conclude that any of the implementations using the different prototype filters mentioned is less complex than the typical one. It is also possible to observe that in the implementation using a longer filter the PPN introduces a bottleneck in hardware complexity.

Two other architectures, with two different implementations for the filter bank are proposed in [2]. The first one, with a direct form implementation can be seen in figure 2.21.

This implementation consists of an OQAM pre-processing block connected to a direct form synthesis filter bank. This direct form SFB is comprised of M upsamplers and M synthesis filters, as the filter banks studied in [2] are M-subchannel ones. The input signals are first upsampled and then applied to the synthesis filter bank, where the data is filtered with the synthesis filters, represented as $G_k(z)$ in 2.21. The output signal is obtained from the addition of all subsignals resulting from the filtering operations. This implementation is then improved by using a more efficient version of the filter bank. This can be seen in figure 2.22.

The increase in efficiency presented by this alternative comes from the avoidance of several unnecessary calculations that were done in the previous architecture. These calculations were a consequence of the high sampling rate at which the filtering operations were performed. In

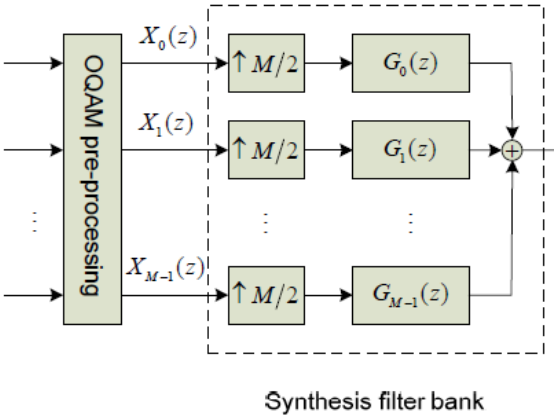

Figure 2.21: Phydyas implementation (source:[2])

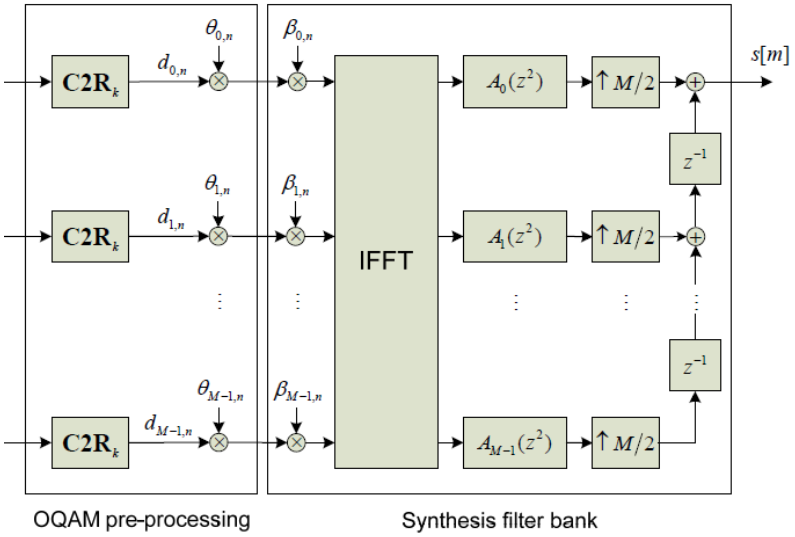

Figure 2.22: Phydyas implementation (source:[2])

this alternative, shown in 2.22 the filtering operations are performed using polyphase structures. These perform filtering operations at a lower sampling rate and, as a consequence, unnecessary calculations are avoided.

In [13] several waveform alternatives were implemented, including both variations of the FBMC waveform, so as to assess which one presents better performance. The PPN-FBMC architecture implemented can be seen in figure 2.23. There is not a lot of detail available regarding this architecture, but it is possible to see that it follows the generic architecture of PPN-FBMC presented earlier.

A different PPN-FBMC implementation is proposed in [14] , with the intent of studying the validity of this alternative. The architecture of this proposal is shown in figure 2.24.

As can be seen from this figure, the presented architecture is for FBMC/QAM instead of FBMC/OQAM in an effort to reduce the signal processing complexity. The major change from this architecture to the generic one is the division of the information symbols into odd and even numbered carriers. From this point on the system functions in the same way as a generic PPN-FBMC system, an IFFT transformation is applied to the symbols, followed by pulse shaping with two prototype filters, one for each branch of the chain. Finally, the overlapping symbols are added and the signal is transmitted.

This proposed PPN-FBMC system was simulated and according to the results the authors

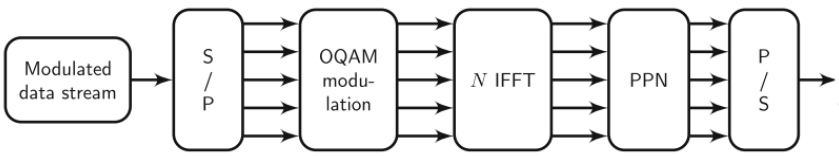

Figure 2.23: Polyphase network FBMC implementation (source:[13])

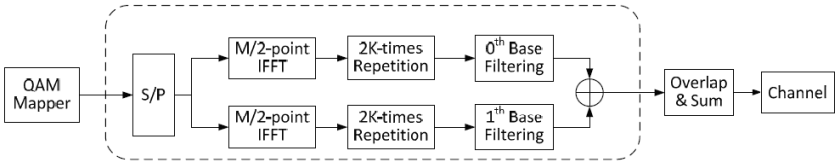

Figure 2.24: Architecture implemented in Qi (source:[14])

drew some conclusions about the merits of this implementation versus one of OFDM. Due to the organisation of the IFFT operation this implementation presents lower complexity than the one of OFDM, due to the reduced IFFT length. In terms of Bit Error Rate (BER) performance this system presents a result similar to that of OFDM. These results are obtained for $M = 1000$ and $K = 4$ The authors conclude that FBMC/QAM can achieve the same performance in various metrics as OFDM, while benefiting from FBMC's characteristics.

2.4.2 FS-FBMC implementations

All the aforementioned implementations follow the PPN-FBMC alternative. It is possible to observe that their general architecture is mostly the same, following the generic architecture presented in the beginning of this chapter. Their variations are mostly related to the parameters used and some minor architecture changes to its constituting blocks, so as to increase the overall performance and efficiency. Several FS-FBMC implementations have also been already developed, which will be presented next.

In [15] a study is performed about an iterative channel estimation scheme of FS-FBMC system. An implementation of a transmitter of this kind is presented and studied, of which a high level view is seen in figure 2.25.

From this figure it is possible to observe that this architecture follows the general architecture of FS-FBMC systems presented earlier in this chapter. The OQAM modulated data is applied to the frequency spreading blocks, where the results from this operation are applied to an extended IFFT module to generate the FBMC symbol. Finally the FBMC symbols are overlapped and the FBMC/OQAM signal is generated and transmitted to the channel.

The synthesis filter bank, making use of frequency spreading, employed in this project is represented in greater detail in figure 2.26. In this figure it is possible to see more clearly the operations performed in the system's synthesis filter bank. The OQAM symbols are conveyed on different frequency points, subchannels, to the IFFT operation. This is the results of an operation of frequency spreading, which employs a filter with carefully calculated coefficients. The coefficients of this prototype filter correspond to the frequency points of the different OQAM data. After this operation of frequency spreading the symbols are applied to an IFFT operation, and the FBMC symbols generated are overlapped to generate the FBMC/OQAM signal.

The study conducted in [15] simulated this system using different parameter values and drew conclusions about the system's performance. Results from this simulation show that the best performance was obtained for a configuration with 4096 subchannels. Other relevant configurations

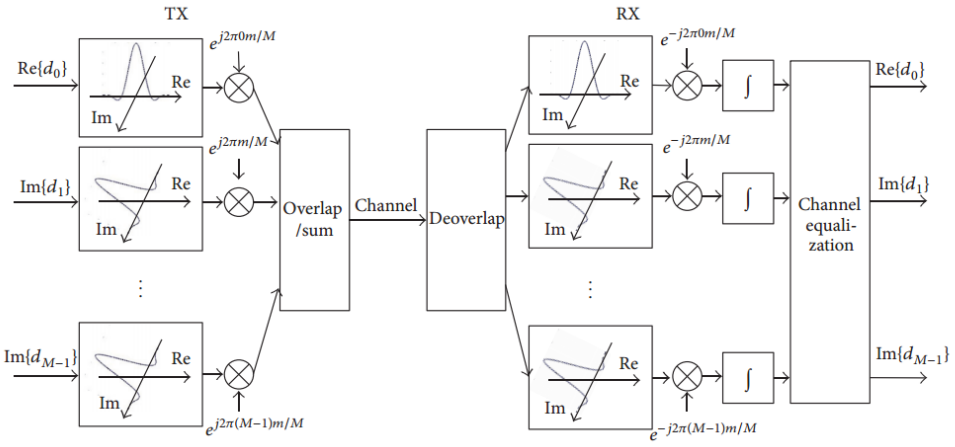

Figure 2.25: FS-FBMC architecture implemented in [15] (source:[15])

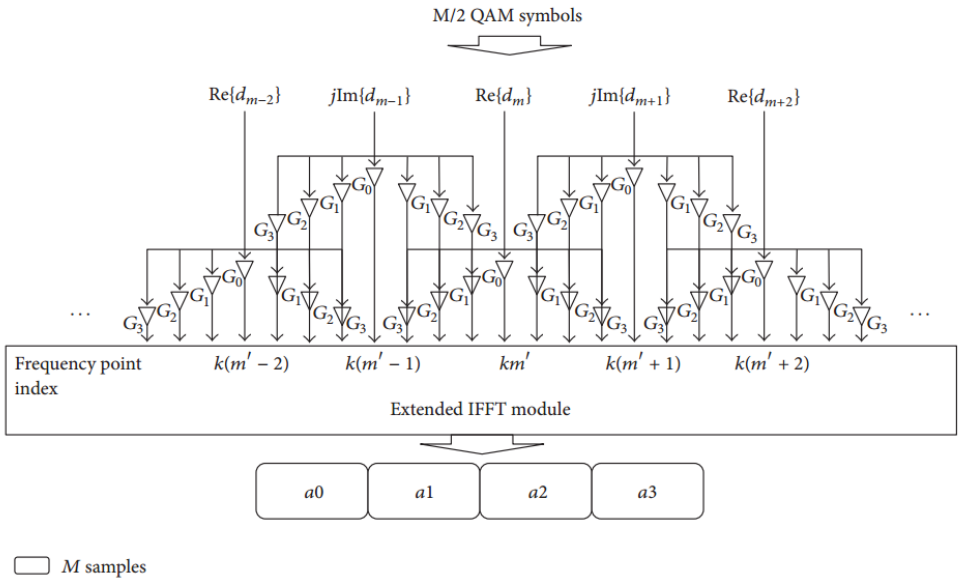

Figure 2.26: Synthesis filter bank in [15] (source:[15])

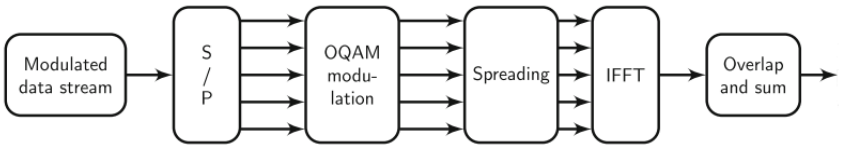

Figure 2.27: Frequency spreading FBMC implementation (source:[13])

for the proposed system included the use of a 256-QAM constellation.

As was mentioned in the previous subsection, [13] present several waveform alternatives, including both FBMC alternatives being studied. The implemented PPN-FBMC architecture was already presented, while the FS-FBMC one can be seen in figure2.27. Like the PPN-FBMC architecture, [13] does not describe in detail this architecture, it is only possible to see that it follows the generic architecture of FS-FBMC systems.

In this study several results were presented regarding the comparison between the different waveforms. These practical results were achieved for implementations with an overlapping factor, K, of 4 and with a number of subcarriers, M, of 2048. The first results relate to the computational complexity of the implemented architectures. As expected, FS-FBMC presents a worse performance than PPN-FBMC (due to the size of the IFFT block, which is K times larger in FS implementations), which in turn is worse than existing OFDM systems. In terms of resource usage the results show that FBMC's usage is similar to that of OFDM.

Finally, there is also a software model which simulates the function of an FS-FBMC processor. This software model is presented in [16] and follows the typical FBMC transmitter architecture presented above in figure 2.27. More in detail, this architecture makes use of a QAM mapping block, followed by an OQAM pre-processing block, which alternates real and imaginary symbols. The signal is then applied to a frequency spreading block and then to an extended IFFT. Finally, to obtain the transmit FBMC signal the data is applied to an overlap and sum block. This architecture is represented in figure 3.2.

There are other studies relating FBMC transmission that do not present specific architectures for the transmitter but include useful information. Such is the case of [Berg], which studies the reception part of the FBMC transmission. From this study it is possible to see the parameters in use, which will be shown in table 3.1.

The parameter values used in the aforementioned studies and articles can be seen in table 3.1. From this table it is possible to compare the different proposals in terms of their parameters and see which ones are most common in the current state of the art.

Analysing table 3.1 it is possible to conclude that both the overlap factor, K, and the number

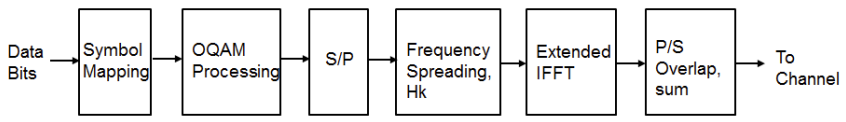

Figure 2.28: Architecture modelled in [16] (source:[16])

Table 2.3: Parameters used in the presented architectures

Architecture	M, subchannels	K, Overlap Factor	Implementation Level
Varga et al. [10]	NA	4	proposal only
Nadal et al. [12]	512	4	post-synthesis
Phydyas [2]	256	4	simulation
Mathworks [16]	1024	2,3,4	software model
Gerzaguet et al. [13]	1024	4	post-synthesis
Weitkemper et al. [17]	1024	NA	simulation
Won et al. [15]	256,1024,4096	4	simulation
Schaich [1]	512,1024	3,4	proposal
Berg et al. [18]	1024	4	post-synthesis (reception)

of subchannels, M has a small range of values in use. The value of M ranges from 256 to 4096, with 1024 being the most common value, while K varies from 2 to 4 with 4 being using in almost all implementations.

Some of the proposed architectures listed above also present post-synthesis or post-implementation usage results which are listed in table 2.4.

From table 2.4 it is possible to compare the resources consumed by these proposals and draw conclusions on the most efficient or complex proposals. It can be concluded that either of the proposals in [12] are less complex than the one used in [13]. Unfortunately all the implementations that have available usage data correspond to PPN-FBMC implementations so a comparison with FS-FBMC is not possible. These values will also prove to be valuable at the end of this project, to compare the obtained values of the resources spent by the developed implementation.

2.5 Summary

As a summary it can be concluded from the study described in the previous sections it is now possible to draw conclusions about the typical FBMC transmitter architectures in use, as well as the parameters used.

First of all, one can conclude that the most widespread alternative for the architecture of an FBMC transmitter is by employing polyphase networks, as opposed to frequency spreading. It is also possible to conclude that OQAM modulation should be used, as it presents better spectral performance when compared to QAM, without the introduction of too much complexity overhead.

Table 2.4: Post-synthesis result of used resources

Architecture	Slice Registers	LUTs	DSP48E1	RAM Blocks	FPGA family
Gerzaguet et al. [13]	11300	7990	30	19	Xilinx Kintex-7
Nadal et al. [12]	2828	4011	20	NA	Xilinx Zynq
Nadal et al. [12]	3788	5585	32	NA	Xilinx Zynq

From the architectures described the range of some parameters in use can also be deduced, such as the IFFT length or the overlap factor to be employed. These correspond to $K = 4$ for the overlap factor and a number of subchannels varying from 256 to 1024.

Chapter 3

Design overview

In this chapter the most important aspects of the FBMC solution will be presented. The system will be described by presenting a high level view of its architecture and various decisions regarding the system and its architecture will also be presented and justified.

The system will also be shown in further depth through the presentation of its main submodules' characteristics and their function. The testing environment employed to validate the system will be in focus, showing a high level view of its organisation.

Finally, the implementation strategy applied to the development of this project will also be described.

3.1 General architecture

As already stated, the problem consists in the development and validation of a baseband processor for FBMC transmission. In chapter 2 an overview of different existing architectures was made and a comparison of their characteristics was presented. From this information it is possible to draw some conclusions about the benefits of each alternative.

Although it is the alternative with higher computational complexity, FS-FBMC presents better performance than PPN-FBMC in several metrics, such as its high performance equalisation capability and its simplicity. Due to this simplicity this alternative is more straightforward to implement. For these reasons this was the alternative chosen to be implemented in the scope of this project. Another reason for this selection was the fact that a software model was available in [16] (presented earlier in chapter 2) which could be used to validate the developed implementation.

The typical architecture of an FS-FBMC system using OQAM modulation was already presented in chapter 2 and can also be seen in figure 3.1.

The blocks described in figure 3.1 are the most important ones in an FS-FBMC processing chain and will be the ones further in detail throughout the project's development. The ones with higher complexity are the frequency spreading block, the IFFT and the Overlap and Sum blocks. These will be the ones presented in further detail in this chapter.

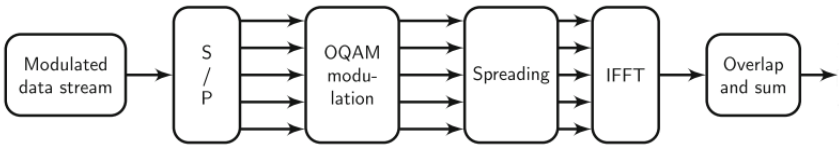

Figure 3.1: Frequency spreading FBMC implementation (source:[13])

Some of the project's main requirements for the produced FBMC processing chain are its ability to be easily configurable and expanded. This is achieved through the use of user defined and adjustable parameters. These parameters are listed in table 3.1.

The parameter K corresponds to the system's overlap factor, which affects both the filter length and the IFFT size. The length of the FIR filter used will correspond to $2 \times K - 1$ and the IFFT size to $K \times M$. This is one of the characteristics that make FS-FBMC more complex than PPN-FBMC as the IFFT size in PPN-FBMC implementations corresponds to M. The M parameter, corresponding to the number of subchannels used affects the size of each symbol and, as such, every block bar the OQAM modulator varies with its value. The IFFT block, logically is the one most affected by this parameter's value, as the number of calculations needed for IFFT operations of different sizes varies greatly. The size of each input symbol is dependant on the number of subchannels, M, and also on the guard band's size, G, as this value corresponds to $M/2 - G$.

As was mentioned above, FS-FBMC was chosen as the FBMC alternative to be implemented in this project. Of all the different proposals for an implementation of this kind it was decided to follow that of [16], as it follows the generic architecture of a system of this kind. By being a generic architecture it has the benefit of easing any change that one might want to make to the architecture. As it is similar to OFDM systems it also makes comparing the two easier. Finally, as this architecture is presented through a software model it provides theoretical results which can be used to validate the system's function. This architecture was already presented in chapter 2 but can also be seen in figure 3.2.

As can be seen in figure 3.2, bar the addition of some blocks, the presented architecture contains the main blocks shown in figure 3.1. The additional blocks perform simple but necessary signal processing operations, such as upsampling or zero padding. In a general view over this system it can be seen that the operations performed are an initial OQAM modulation, followed by frequency spreading, coupled with an IFFT operation and the generation of the output signal through an overlap and add operation, as is expected from an FS-FBMC system.

Some values for the adjustable parameters are also presented in [16] and are listed in table 3.2.

The presented software model accepts a value of the overlap factor K between 2 and 4, while

Table 3.1: User-defined parameters

Parameter	**Variable**
K	Overlap factor
M	Number of subchannels
G	Guard band size in units

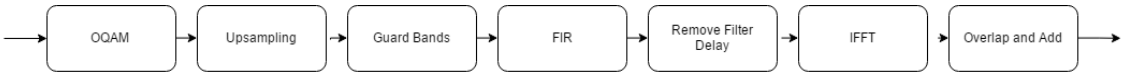

Figure 3.2: Implemented architecture

the remaining values are configurable and can be changed to observe how the system varies with these parameters. The values shown in table 3.2 present an example for a possible configuration of the system. As the implemented system is to be parametrisable a change in these parameters should also be able to be made in the developed hardware implementation. By doing this there is the possibility to implement several different alternatives, some of which were presented in chapter 2. By analysing the different characteristics of each configuration it is possible to study how the system varies with its parameters. This study will be presented in chapter 5. The values of table 3.2 are in accordance with the typical values in the current state of the art, seen in chapter 2.

Other than the system's constituting blocks, an important feature is the way these blocks communicate and exchange data. To ensure the correct function of the system one must make sure that each block only receives data when it can be correctly processed and only generates valid data when the following block is ready to process it. There must be a way to guarantee this synchronisation, through a handshake protocol, using control signals. The correct exchange of these signals and their application in regards to the calculations is vital to ensure the processing chain's synchronisation.

As the IFFT block which was available from previous work [8] was provided with AMBA's AXI4-Stream interfaces this was the protocol chosen to be used to synchronise the system's submodules. It is a simple and widespread handshake-like protocol which makes use of several control signals to exchange data between various master/slave blocks. The details of the AXI4-Stream protocol will be described further in this chapter, showing the timing of the different signals and the restrictions in their generation that ensure the correct synchronisation of the system. The implemented signals of this standard protocol and their function can be seen in table 3.3.

Other signals are used in the full AXI4-Stream protocol however only these were implemented as they are the fundamental ones to ensure the chain's correct function. Each block of the processing chain has an interface comprising these signals, which eases the task of synchronising the exchange of information. A connection between two AXI4-Stream compliant modules is exemplified in figure 3.3. It should be noted that the clock and reset signals aren't depicted in this figure, as these are common to all blocks.

The complete FBMC processing chain produced implements an AXI4-Stream interface both in its input and in its output, so as to simplify its integration in higher level projects. The resulting

Table 3.2: Parameter values in in the software model[16]

Parameter	**Value(s)**
M	1024
K	2, 3, 4
G	212

Table 3.3: AXI4-Stream control signals

Signal	**Function**
Clock	Clock source
Reset	Reset source, active low
Valid	Signals the validity of the output/input data
Ready	Signals the readiness of the block to receive new valid data
Last	Signals the last valid word to be received

IP block follows the architecture seen in figure 3.4.

As can be seen in this figure, the final block which implements the FBMC processing chain should provide an AXI4-Stream interface, in order to ease its inclusion in higher level projects. This figure doesn't show the clock and reset signals, which are also used in this block.

Due to the fact that every block of the processing chain implements the AXI4-Stream interface, they have the same input and output signals, bar some exceptions. These signals are the AXI4-Stream ones listed in table 3.3, both as input and output and also the data signals in the input and output, the only ones which aren't a simple wire, but a bus with a bit width of 16 bits each. These signals are listed in table 3.4.

The only block that doesn't follow this table is the OQAM submodule, where the input is a single 4 bit signal coming from the processing chain's input.

Another characteristic of the system which interferes with the implementation of its modules is the fact that its calculations are performed using fixed point. To be coherent with the provided IFFT block, the remaining submodules of the processing chain are to be implemented using Q(5,11) representation. This means that each 16 bit value contains 5 bits for its integer part and 11 bits for the fractional component. A representation of this kind introduces some error, as its resolution is $1/2^{11} = 0,00048828125$.

The use of fixed point entails some changes to some submodules' architecture, due to the

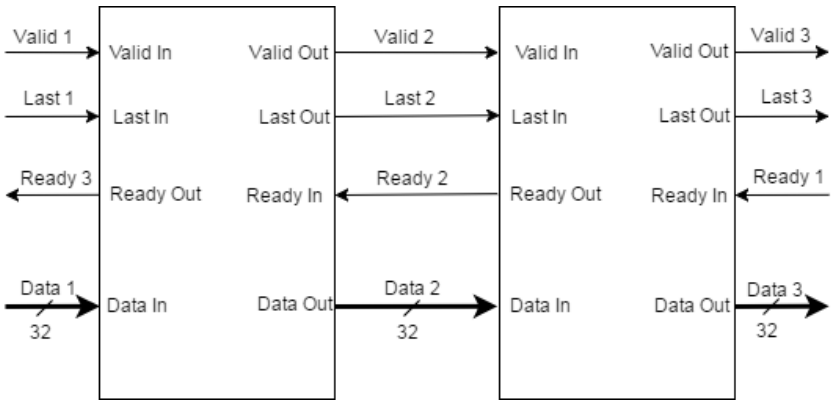

Figure 3.3: AXI4-Stream example

Figure 3.4: Final IP block desired architecture

rounding performed in some operations. In a signed multiplication between two fixed point numbers the result is given as in the following expression:

$$A(a_1,b_1)\times A(a_2,b_2) = A(a_1+a_2+1,b_1+b_2) \tag{3.1}$$

This means that, in order to obtain a 16 bit value in the output of an operation of this nature the result needs to be truncated. This operation will be needed, for example, in the multiplications performed in the calculation of the FIR filter. This truncation introduces rounding errors, which result in some error in the final transmitted signal.

3.2 General view of the system's submodules

In the following subsections the most complex blocks in a typical FBMC implementation will be described in further detail, along with an explanation of their function.

3.2.1 OQAM modulation

As was already described in chapter 2, as FBMC makes use of OQAM modulation there is a need to implement a block which modulates the input signal using this technique.

The first step in this modulation consists of modulating the input signal using simple QAM modulation. 16-QAM was chosen as the QAM modulation as it is the most common choice, as seen in chapter 2. To modulate the input signal a multiplexer is used, which outputs the value of the real and imaginary components according to the input value. The constellation selected is the one used in [16] and can be seen in figure 3.5. The chosen constellation makes use of Gray Mapping, in an effort to reduce errors.

From the real and imaginary values of the constellation points obtained from the initial QAM modulation, to obtain the desired OQAM modulated result these values should be upsampled by a factor of 2. Then a delay is introduced to either real or imaginary component, according to the index input, guaranteeing the offset required for OQAM modulation.

A high level view of the OQAM block's architecture can be seen in figure 3.6. There we can see the operations performed and the way the input signals control them.

As can be seen in figure 3.6 the OQAM modulator block can be divided into three submodules: the QAM mapping, which transforms the input value into a complex value, according to the QAM constellation in use; a block which performs the 2 factor upsampling and finally a block which

Table 3.4: Input and Output signals of each implemented submodule

Signal	**Data type**
Clock	Input
Reset	Input
ValidIn	Input
ReadyIn	Input
LastIn	Input
RealIn	Input [15:0]
ImagIn	Input [15:0]
ValidOut	Output
ReadyOut	Output
LastOut	Output
RealOut	Output [15:0]
ImagOut	Output [15:0]

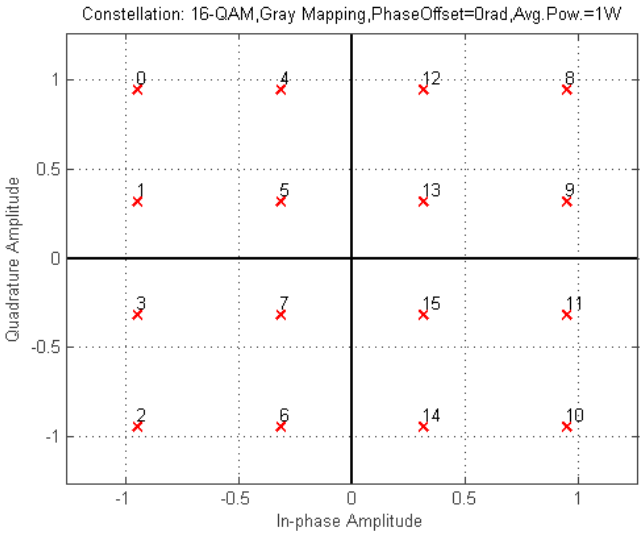

Figure 3.5: QAM constellation

Figure 3.6: OQAM high level architecture

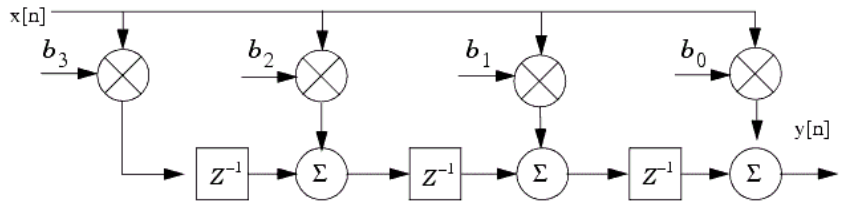

Figure 3.7: FIR architecture

applies the 1 clock delay to one of the signals, according to the index, which is one of the OQAM block's inputs.

Mathematically, the QAM to OQAM is achieved by a multiplication of the QAM data by the following expression:

$$\theta_{k,n} = j^{(k+n)} \tag{3.2}$$

with k and n corresponding to the subchannel index and the time index, respectively. This means, as expected, that the OQAM modulated data is either purely real or purely imaginary.

To exemplify the function of this block one can look at the operations performed for an input of 0, followed by an input of 1: Initially these inputs are converted into a complex number corresponding to their point in the constellation, in this case -0.9487 + 0.9487j for 0 and -0.9487 + 0.3162j for 1. The real and imaginary signals from the QAM mapper will then be upsampled by a factor of 2, which means a 0 will be inserted in between two valid values. This upsampling operation also means that the OQAM block generates two valid outputs for each valid input. Then, according to the value of the block's input *index* either real or imaginary component will be delayed. For the purpose of this example index is assumed to be 0, so the imaginary signal is delayed by one clock cycle. This means the output will be -0.9487, 0.9487j, -0.9487 and finally 0.3162j.

3.2.2 FIR filter

One of the main differences between frequency spreading and polyphase network FBMC implementation is the use of a spreading block to perform the filtering as opposed to a polyphase network block. The spreading block employed in FS-FBMC requires the use of an FIR filtering operation.

For the present implementation it was chosen to use a transposed version of an FIR filter as it allows an easier and more efficient implementation [5]. This version of a FIR architecture can be seen in figure 3.7.

The coefficients used in the implemented filter are the ones used in the Matlab script[16]. These cause the filter response seen in figure 3.8.

The use of an FIR filter implies the introduction of a filter delay which must be removed. Hence, a block needs to be added to the chain next to the FIR filter which is tasked with removing this delay. This is a simple operation as its only function is to remove the first $K-1$ values of the input signal and add $K-1$ 0s at the end.

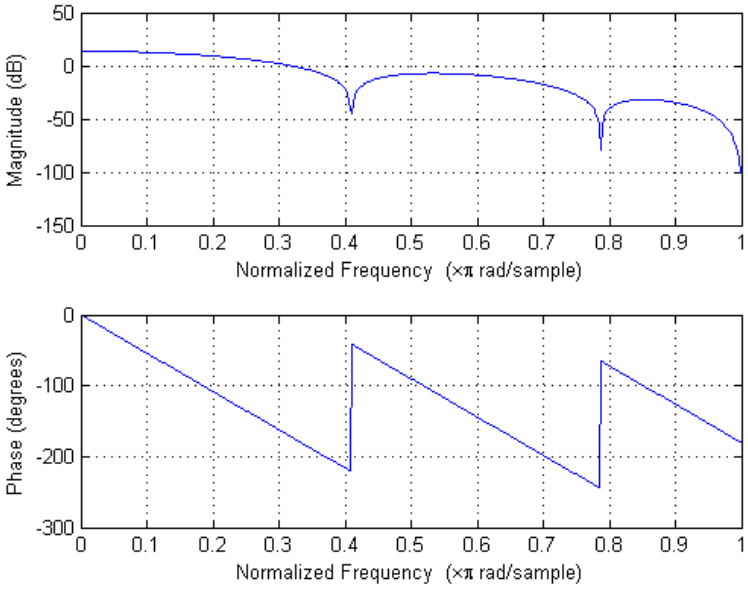

Figure 3.8: FIR filter response

3.2.3 IFFT

The most computationally expensive block of the processing chain of an FS-FBMC transmitter is the IFFT block. This block was previously developed for a different project[8] and, as such, was not developed in the scope of this work.

The parameters in use for this block are consistent with those chosen for the processing chain which in turn were chosen according to the literature review previously done. These consist of a number of subchannels M of 1024 and an overlapping factor K of 4. The frequency spreading alternative for FBMC systems entails the use of an IFFT operation of size $K \times M$, instead of M, as is the case in PPN-FBMC implementations, which means this specific configuration entails an IFFT size of $1024 \times 4 = 4096$.

This makes this block more expensive but makes the full processing chain simpler, when compared with OFDM systems.

3.2.4 Overlap and add

The final block in the FBMC processing chain is the one which performs the overlap and add operation. The main difficulty with this block is the need to store the input data, as this information will only be used in the following symbol. This requires a thoughtful use of the FPGA's resources and a careful timing of the read and write operations which will be performed.

To be able to efficiently implement this operation this storage should be done using Block RAMs, as it would be too expensive to store using LUTs. Since this implementation stores a temporary array of size $2 \times K \times M$ for both the real and imaginary components, containing both the current symbol to be transmitted and the temporary symbol where the IFFT results are summed

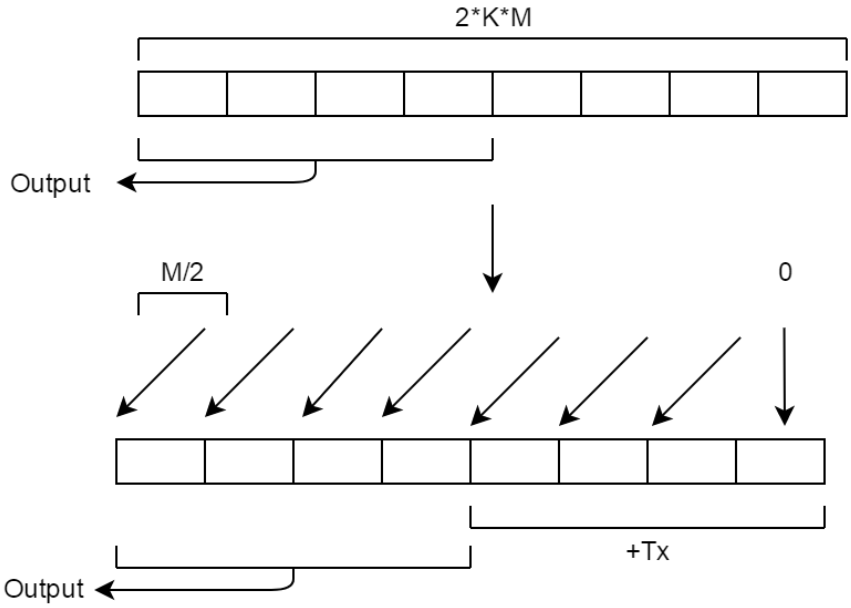

Figure 3.9: Overlap and Add

it requires a large amount of resources. As each value stored in this array is 16 bit its total size is $2 \times K \times M \times 16$bits, hence why it would be prohibitive to implement without block RAMs or other efficient memory uses.

The operations performed by this block are exemplified in figure 3.9.

From figure 3.9 we can see that the first $K \times M$ values from the overlap array are sent to the output. A shift operation of $M/2$ values is then applied to this array, where the last $M/2$ positions are filled with 0s. Finally, the received signal is added to the last $K \times M$ values of the array. This process is repeated for as long as the system is in function.

3.2.5 Other blocks

From figure 3.2 we can see that the chosen architecture makes use of blocks other than the ones previously presented in this section. These are simpler and perform less complex operations and, as such, will only be briefly described by stating their main characteristics.

3.2.5.1 Upsampling

The upsampling block is tasked with adding $K-1$ 0s between each two valid values. Effectively, this consists in an overlap operation without changing the signal's rate. The number of zeros introduced is controlled by the user defined parameter of the processing chain K.

This block, with an input signal of M valid values will generate an output signal with $K \times M$ valid values.

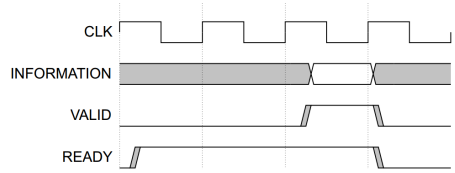

Figure 3.10: AXI example with Ready asserted before Valid

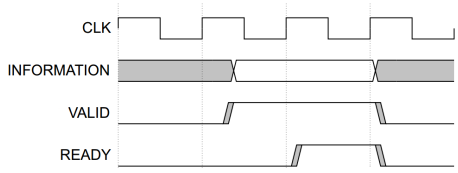

Figure 3.11: AXI example with Valid asserted before Ready

3.2.5.2 Guard Bands

The block following the upsampling employed in the chosen architecture corresponds to the one tasked with adding zeros at the beginning and at the end of each symbol.

In the architecture implemented this operation is performed by the "Guard Bands" block, which pads the input signal with zeros. This padding occurs both in the beginning and at the end of the signal, adding $K \times G$ zeros in both ends.

3.2.6 Synchronisation

Due to the characteristics of the provided IFFT block, which makes use of AMBA's AXI4-Stream's protocol in its interface, this is the protocol chosen for the remaining blocks of the FBMC processing chain, to ensure their correct synchronisation. As was seen, this interface was partially implemented, with its fundamental signals being implemented in every block. These are the signals listed in the handshake process described in the protocol's specification [19], the ready and valid signals both in the blocks' inputs and outputs. The last signal was also implemented as it is useful when the system is integrated in a higher level testing environment.

The correct synchronisation and control of the data flow is ensured through the valid and ready signals' handshake. The data transfer follows these restrictions, listed in the protocol's specification: modules can only exchange data once both valid and ready signals are asserted; when valid is asserted it must remain that way until the ready signal is also asserted; a module can wait for the ready signal to be asserted before asserting its ready signal. In a broad sense these signals ensure that a block doesn't generate new outputs when the next one isn't ready to process them and also prevents a block from reading values that aren't valid. This allows the system not to lose important valid data and also not to introduce invalid information in its calculations.

Two examples of the handshake between the Ready and Valid signals can be seen in figures 3.10 and 3.11. These restrictions were implemented in all blocks, some of them through simple logic dealing with its inputs and in others making use of the block's internal finite state machine. Likewise to the FBMC's core implementation, the development of this implementation was done on a per-block basis, along its validation.

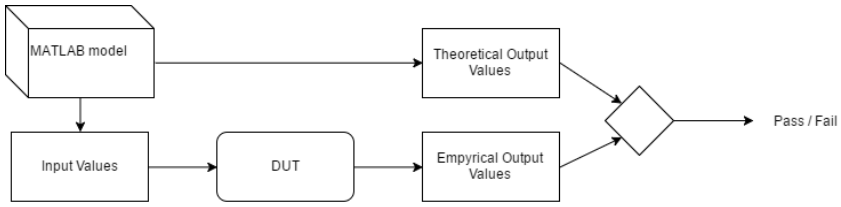

Figure 3.12: Validation environment

3.3 Validation environment

In chapter 2 it was mentioned that one of the benefits of implementing a frequency spreading FBMC alternative when compared to polyphase networks is the availability of a Matlab software model which can be used to validate the implemented system. With this software model the values obtained from the physical implementation can be compared to the ones obtained in a Matlab simulation.

This possibility enabled the development of a simple validation environment which is depicted in figure 3.12.

As can be observed in this figure, the Matlab model generates both the input symbols which will be applied to the device under test (DUT), the FBMC transmitter chain, in this case, and the theoretical output values. These values can then be compared to the ones generated by the DUT to verify its correct function. Both the software model and the implemented chain's parameters can easily be changed to test the system for different configurations.

One of the system's characteristics that should be taken into account when comparing the obtained results with the ones obtained from software models is the fact that, working with fixed point, these results will vary, as software models work with floating point.

The IFFT block also works with fixed point with a Q(5,11) format, which means each 16 bit value uses 5 bits for the integer part's representation and 11 bits for the fractional part. Some operations performed across the system's processing chain will need to be carefully executed due to the limitations of fixed point. An example of this is the multiplication performed during the filtering operation. When 2 Q(5,11) fixed point numbers are multiplied the result is represented in Q(10,22). This means that, in order to obtain a 16 bit result for this operation a truncation must be performed. As a consequence of this truncation some rounding errors are introduced. Errors are also introduced when converting floating point numbers to fixed point, as is the case with the values of the FIR's coefficients or the QAM constellation's points.

As a consequence of this use of fixed point representation, the results shouldn't be expected to correspond exactly to those of the software model used as a comparison, as this works with floating point. As such, a tolerance shall be established to assess whether the produced values are correct, taking into account the error expected to be introduced from these rounding errors.

3.4 Strategy for development

The development of this project followed the usual design flow seen in digital systems. The initial step in the development of this system consists in the HDL description of the aforementioned blocks. To ensure their correct function their development is followed by behavioural simulations. As the implemented system comprises several different, independent blocks its implementation should follow a previously defined strategy, so as to ease the development and validation of each block and, as a consequence, of the entire system. This makes for a more efficient use of the time available for the project. This strategy consisted in a step by step development of the system where a full implementation of a block would be produced and immediately validated. By doing this it becomes easier to detect any eventual errors in the block and correcting them. Once all the system's constituting blocks are developed and validated their integration and synchronisation can be done. This step is somewhat time consuming, as the synchronisation between blocks isn't trivial, as all AXI4-Stream control signals must be correctly generated in the right times. The development of each of the system's blocks comprised different stages: firstly a study of its function must be done; then the HDL development, which can only start when the block's details are fully understood and its architecture is carefully thought out; a software model is then produced using Matlab; the block is implemented and validated through comparison between its outputs and the ones obtained from the software model. Finally, the block's interface is tested ensuring that, not only is it producing the correct results, it is also producing them at the right times and working properly with the input AXI4-Stream signals.

Once the HDL description of the system's constituting blocks is completed and validated the next step consists in running Synthesis and checking if the result produced by the synthesizer is acceptable. Before advancing it is important to run a post-synthesis simulation, to ensure that the synthesised design produces the correct results. There are multiple scenarios in this step which might entail a step back into HDL development: some of the code might not be synthesisable; the post-synthesis simulation's results might not be the pretended; the produced system might be using too much of the available resources and should be implemented in a more resource efficient way.

Once the synthesised hardware produces the correct results and presents acceptable usage results the next step is to run the implementation of the system. This step consists in the place and route of the generated netlist onto the FPGA resources. Likewise to the previous step, a post-implementation simulation should be run to ensure the system's function.

Once all these steps are concluded an IP block can be generated and included in a high level project ready to be prototyped.

In this project all these steps are done making use of Xilinx's Vivado software.

Chapter 4

Implementation

In this chapter the implementation of the developed system will be presented, through a more detailed view of its constituting submodules. Their architecture and its development will be described and explained in a lower level view than the one presented in the previous chapter. The system's timing characteristics will also be described, via the presentation of several timing diagrams.

4.1 Description of the developed submodules

In the previous chapter a more general view of the architecture of the system and its constituting submodules was already presented, explaining their function in the overall FBMC processing chain. In the following subsections a more detailed description of each module will be presented, through their corresponding hardware and timing diagrams. An overall view of the implemented architecture for the complete processing chain will also be presented.

The IFFT block isn't included in these descriptions as, as was mentioned previously, it was developed in the context of another project and was provided for use.

4.1.1 OQAM

As was already seen in chapter 3, the first block in the FBMC processing chain is the one responsible for modulating the input signal, using the OQAM technique. To perform this OQAM modulation there is the need for an initial QAM mapping of the input signal, followed by an upsampling of the resulting signal by a factor of 2 and finally a delay is applied to either real or imaginary component.

The implemented architecture for this block can be seen in figure 4.1. From this figure it is possible to observe that the architecture of the OQAM block can be divided into three main sections: a QAM mapper, an upsampling block and some additional logic which applies a delay to one of the output signals.

The QAM mapper consists of a simple look up table where the values of the real and imaginary components of the used constellation are stored. This block receives the 4 bit input that is applied

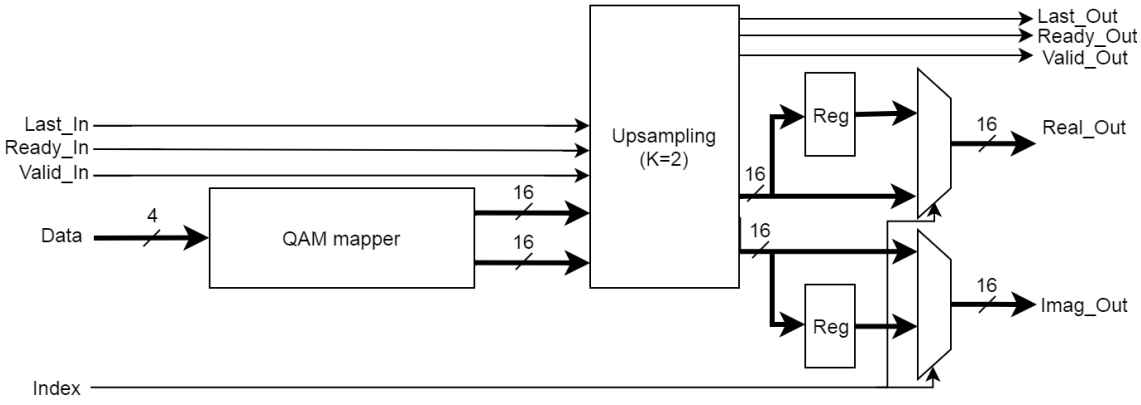

Figure 4.1: OQAM block architecture

to the OQAM block and, according to its value, different values for the real and imaginary output are selected and fed to the upsampling block.

The upsampling module instanced in this block is tasked with upsampling the resulting signal of the QAM mapper by a factor of 2. This block consists of an instace of the upsampling block developed having in mind the upsampling operation that is needed for the complete chain and, as such, will be described in further detail later in this section. However, as this operation is always done with a factor of 2, this block isn't dependant of the system's K parameter, unlike the other upsampling block, as will be seen later.

Finally, to obtain the OQAM modulated signal either the real or the imaginary component is delayed 1 clock cycle, according to the value of the input signal "index". This delay ensures that the output signal is either purely real or purely imaginary, which is the goal of OQAM modulation. To obtain this delay two multiplexers are used, one for each component which, according to the value of the "index" input choose either a normal or delayed version of the real/imaginary symbol to be sent to the OQAM block's output.

Due to the timing properties of the operations performed in this block the generation of the synchronisation control signals is quite simple, as it is done solely in the upsampling module instanced in this block.

An example of this block's operation can be seen in the timing diagram shown in figure 4.2. As can be seen, this block has a latency of one clock cycle, meaning that the outputs are valid one clock cycle after the application of any valid input. Due to the function of the upsampling block, which isn't ready to receive new valid inputs while it is outputting 0s, the inputs for the OQAM must remain constant for 2 clock cycles. This operation is guaranteed by the synchronisation

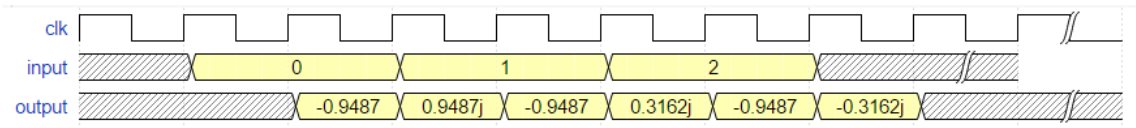

Figure 4.2: Example of OQAM modulation

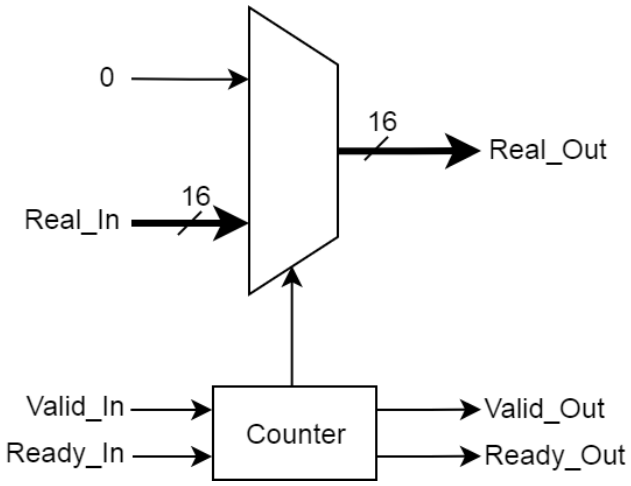

Figure 4.3: Architecture of the Upsampling block

signals generated inside this block.

4.1.2 Upsampling

The next block in the FBMC processing chain is the one which performs the upsampling operation. The upsampling performed is dependant of the system's parameter K, which determines the factor of this operation. This way, this block inserts $K-1$ 0s between each two valid inputs.

The architecture of this block can be seen in figure 4.3. As can be seen from this figure the implementation of this block is made simple, mainly due to the use of a counter controlled by the block's input synchronisation signals. It should be noted, however, that in this figure only the architecture for the real signal is depicted, omitting the signals related to the imaginary part of the signal, in order to simplify the diagram. The function of this block for these signals is exactly the same as the one depicted.

To perform the upsampling operation this block must output either a 0 or the input value originating from the OQAM block. To select this value a multiplexer is used, controlled by a counter. This counter is set at K when the input signal is valid and decreases at each clock cycle. This way it is possible to obtain $K-1$ 0s for each valid input sample. The operation of this counter block is controlled by the ready and valid input signals, to guarantee a correct synchronisation of this block. From the value of this counter the output control signals valid and ready out are also set to 1 or 0.

An example of this block's function can be seen in figure 4.4, where its timing behaviour is shown, in a situation where its *ReadyIn* input is constantly 1, which means the following block is always ready to receive a new value. This example is for an upsampling factor of $K=3$ so, as was explained earlier, the upsampling block will insert $K-1=2$ 0s in between each two valid input values.

From this diagram it is also possible to see that this block presents minimal latency (1 clock cycle), due to the way it is implemented.

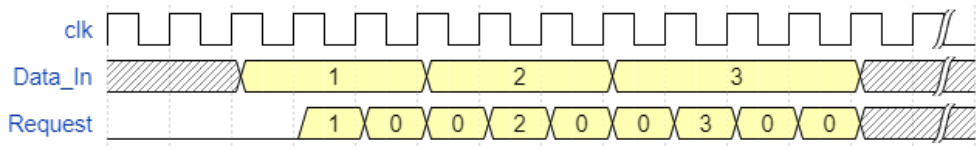

Figure 4.4: Timing diagram of the Upsampling block

4.1.3 Guard Bands

The operation that follows the upsampling is the introduction of guard bands in the symbol. Since this is a relatively straightforward operation the block that implements it is simple and makes use of a finite state machine, which controls the introduction of zeros at the beginning and at the end of each symbol.

This operation is simplified by the knowledge of the size of each symbol. It is known that $G \times K$ zeros should be introduced in the beginning and at the end of each symbol. The initial zeros can be introduced immediately, once it is known that a new symbol will be produced (signalled by an asserted *Valid_In* input); the last $G \times K$ zeros will be introduced once all valid inputs are received, which is known to be $K \times M - 2 \times G \times K$. The finite state machine developed to implement this operation is shown in figure 4.5.

As seen in this figure, there is an initial state where the system is idle. When a valid input is received the system knows that a new symbol is to be received, so it progresses to the next state, where the first $K \times G$ zeros are output. In this initial state the valid value received is stored, so it can be output as the first value after the zeros are output. Once all $K \times G$ zeros have been fed to the output the system advances to a state where the previously stored valid value is output. Afterwards the system enters a state where it receives valid inputs and feeds them to the output. Once all $K \times M - 2 \times G \times K$ valid values are received and output, the system enters a state where it outputs the final $K \times G$ zeros of the symbol. Lastly, the system either returns to the idle state if there is no other symbol waiting to be processed or advances immediately to the state where the initial zeros are padded.

The function of this block and its finite state machine is better understood with the help of its timing diagram, seen in figure 4.6. In this figure it is possible to see that, considering the *ReadyIn* input as constantly 1, which means the next block is always ready to receive new data, this block produces $K \times G$ zeros at the beginning and at the end of a symbol. The latency of this block is one clock cycle.

The example presented in this figure is for a configuration of $K = 4$ and $G = 1$. Due to the upsampling factor a Guard Band parameter of 1 will introduce 4 zeros at the beginning and at the end of the symbol. The symbol represented comprises only 3 values.

4.1.4 FIR filter

The output signal resulting from the Guard block is then applied to the FIR filter. As is implied by the name of the block, a filtering operation is performed.

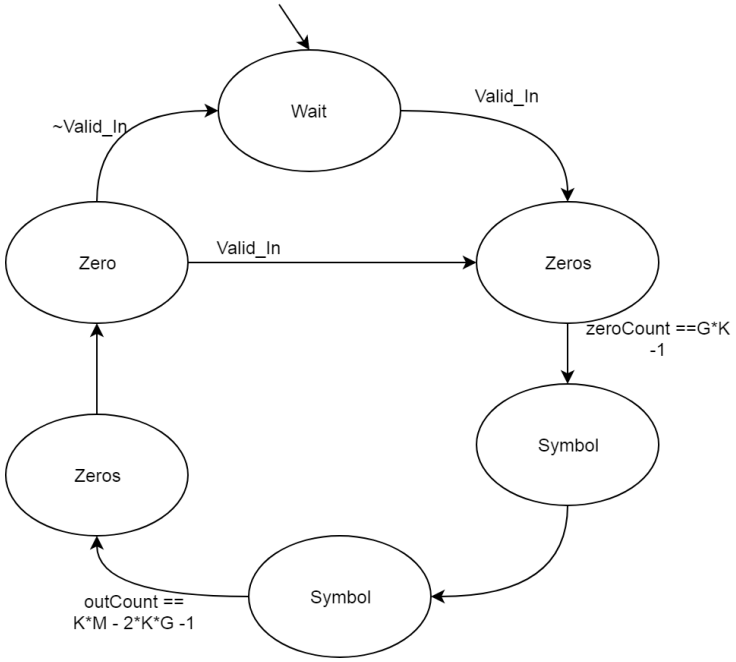

Figure 4.5: Guard Band block's FSM

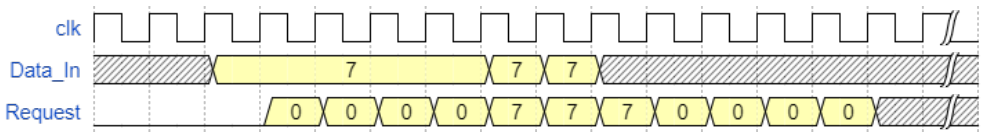

Figure 4.6: Timing diagram of the Guard Bands block

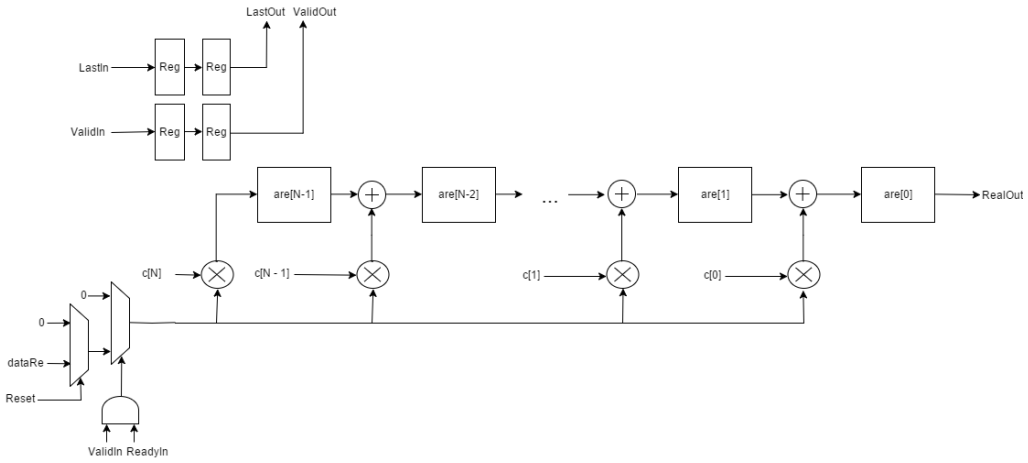

Figure 4.7: Architecture of the FIR block

To ensure a more efficient implementation, a transposed FIR filter architecture was chosen, as this is considered to be the preferred implementation for FIR filters[5]. This is due to the fact that this architecture avoids the introduction of additional shift registers for the input signal $x[n]$ when compared to the standard FIR architecture.

This alternative was implemented through the use of several adders and multipliers. The multiplications are done using the FPGA's internal DSP slices, in this case, the DSP48E1 slice, which has an internal multiplier.

Likewise to the other blocks implemented for the FBMC processing chain, the FIR block should be parametrisable. In this case the filter length is user-defined and depends of the parameter K. A value of K will entail a filter length of $2 \times K - 1$, according to what is done in [16]. To be able to extend the function of this block according to the parameter set by the user the logic which implements these operations is coded using Verilog's *generate* command. The coefficients in use for the filtering operation are stored in a previously loaded LUT, with values corresponding to those of [16].

The implemented architecture can be observed in figure 4.7. As can be seen, likewise to what was done for the upsampling block, the architecture of the FIR block is only represented for the real part of a symbol. The logic associated with the calculations for the imaginary part was omitted for simplicity, as it is identical to that of the real component.

One of the limitations of the implementation of this block is the use of fixed point representation for the 16-bit signal values. This introduces the need to truncate the result of some operations. Such is the case in the multiplication operation performed in this block. The truncation performed after this operation is also not depicted in the diagram.

Due to how multiplication works in fixed point, two 16 bit operands (Q(5,11))will generate a 32 bit result Q(10,22). To convert this result back into a 16 bit representation (with 5 integer bits and 11 fractional ones, as was chosen for this project, this word should be truncated according to the following instructions: The 5 less significant bits of the integer part should be the new integer part; the 10 most significant bits of the fractional part should correspond to the new fractional part.

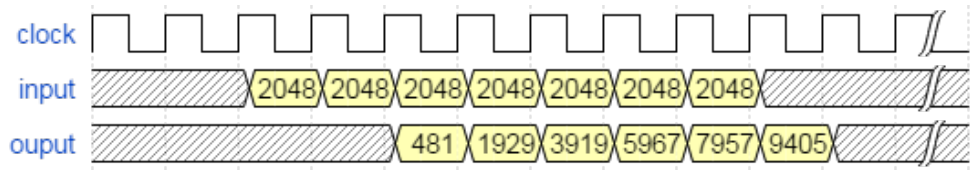

Figure 4.8: Example of FIR function

This operation will introduce a rounding error, which will be present in every calculation involving fixed point.

An example of this block's operation can be seen in figure 4.8. For this example the input was kept constant at 2048 to ease the understanding of what is happening. The parameters in use for this example are as follows: $K = 4$, which produces a filter length of 7. The coefficients used are the same as the ones in the Matlab model, represented in fixed point: $[481\ 1448\ 1990\ 2048\ 1990\ 1448\ 481]$. In this figure the block's timing can also be seen; the filter operation presents a latency of 2 clock cycles, during which the calculations still aren't valid. From then on the system outputs the result of the filtering operation in fixed point. In a similar fashion to the diagrams presented for the other blocks in this example it is assumed that the *ReadyIn* input is constantly 1, meaning that this block doesn't need to wait before outputting a new valid value.

A consequence of the use of an FIR filter is the introduction of a filter length delay. An operation to remove this delay must, then, be applied. This operation consists in removing the first $K-1$ values of a symbol and with adding $K-1$ zeros are added at its end. Being one of the blocks which implements one of the simpler operations of the entire processing chain its implementation is also straightforward and, like the previously described block which introduces the guard bands in the symbol, it is based on a finite state machine. This finite state machine is represented in figure 4.9.

From this diagram one can observe that the function of this block is divided into 5 states: an initial idle state where the system waits until a valid value is input, signalled by the *Valid_In* input; Once an initial valid value is received the system waits in a state where no output is set, effectively removing the first $K-1$ values of the symbol. The system then advances to a state where the output corresponds to the data received in this block. Once all values are output the system advances to one of two states which output the $K-1$ 0s needed. These two states differ in the state where the system will advance to; If a new symbol is to be received the system enters a state where it outputs 0s while ignoring the first $K-1$ values of the following symbol, otherwise it simply outputs $K-1$ 0s before returning to the idle state. The use of both of these states allows the output of a constant stream of data in a situation where there are more than one symbols to process, which is important to guarantee the maximum possible throughput for this system.

The function of this module can be seen in the timing diagram presented in figure 4.10.

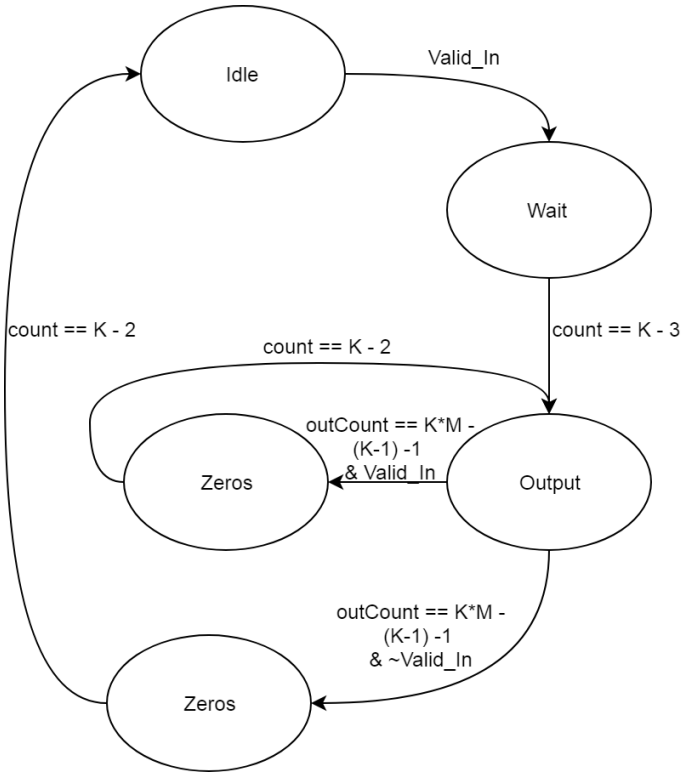

Figure 4.9: Finite state machine of the delay removal block

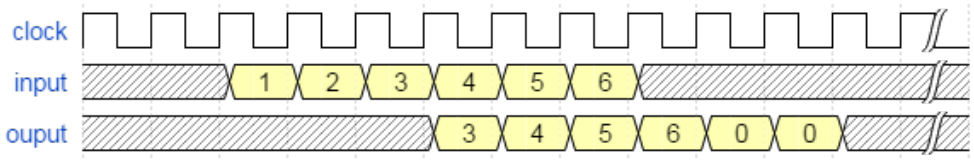

Figure 4.10: Timing diagram of the delay removal block

4.1.5 Overlap and Add

Finally, the last step in FBMC's processing chain is to submit the filter results to an overlap and add block. As was explained in the higher level description of this block in chapter 3, in order to function properly this block needs to store not only the values of the previous symbol but also those of the temporary array used to build the new symbol. This requires storing an array of values with double the size of a symbol, $2 \times K \times M$. This would mean a usage of $16 \times 2 \times K \times M \times 2$ bits of storage. To be able to efficiently implement this storage without this high resource usage Block RAMs were used. This was achieved through a specification of the memory type to be inferred by the Synthesis tool in use (Vivado).

The code used to specify the memory style was as presented in the synthesis tool's user guide[1]. The *ram_style* is an attribute which one can configure, in order to instruct the software on how to synthesize the memory. This attribute can instruct the synthesizer to infer the memory as being distributed, where it is mapped to LUT RAMs and also as a block which will map a block RAM. This is the option selected, as it allows a better resource usage.

```
(* ram_style = "block" *)
    reg signed [15:0] nextSymbolReal [2*M*K-1:0];
(* ram_style = "block" *)
    reg signed [15:0] nextSymbolImag [2*M*K-1:0];
```

As such, both these arrays will be mapped to one of the FPGA's internal RAM blocks, avoiding the use of an unreasonable amount of LUTs and other of the FPGA's available resources.

This type of RAMs in the Zynq-7000 FPGA is dual-port, which means that read and write operations can be performed at the same time, if given that these operations are not performed to/from the same address. This possibility eased the implementation of the Overlap and Add block, as, for example, during the shift operation, it is possible to read from one end of the memory and write to another. One operation where this caused problems is the operation of adding the input value to the already present one in the memory. To circumvent this issue and allow a simultaneous read and write one of these operations was delayed, so in reality what is happening is that the memory was being read from one address and written to another one.

The overlap and add operation can be split into three main actions: the reception of a new symbol; the transmission of the output symbol and the shift operation. The reception of the new symbol will consist in storing the new value in the memory or adding it to the existing value in memory, according to its index value. To be able to minimise the system's throughput this block should be able to perform these actions at the same time. This poses a problem as both the output and overlap operations need to access the memory, which isn't possible, as the BRAMs in use only allow 1 read operation per clock cycle. To circumvent this issue the memories in use were doubled. This allows one memory to be used for reading the output value while the other is used to read the current value and add in the new value. The action of writing new data into the memories is performed in all of them at the same time, so as to keep their content the same.

[1] Vivado Design Suite User Guide Synthesis, https://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_2/ug901-vivado-synthesis.pdf

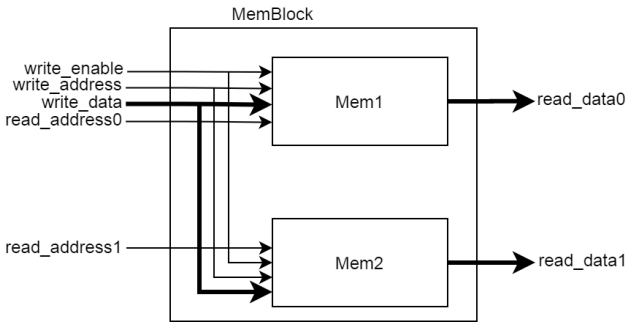

Figure 4.11: Memory block used in the Overlap and Add block

The structure of the memory block employed in this block can be seen in figure 4.11. As can be seen in this figure, the memory block has 5 input signals, which control both the write operation (with an enable signal, an address and the data to be written) and the read operation (two addresses, one for each memory). The memory block has two outputs, with each one containing the data read from each memory. There are two instances of this block in the Overlap and Add block, one for the real component of the signal and another for the imaginary component.

This memory block is the central point of the overlap and add module, as the overlap and add operation is performed through a careful control of the write and read operations performed in both these memories. To be able to easily perform the necessary $M/2$ shift operation the addresses in use are increased by $M/2$, every time a full symbol is received, effectively shifter the information simply by changing its address. To further simplify the read and write operations and also to allow a continuous function even between symbols without the need to reset the information both memories were implemented as circular buffers.

To perform the add operation the data to be written in the memories is chosen with a multiplexer which selects either the input value or the sum of the input value and the existing data in the memory. This multiplexer is controlled by a signal which counts the number of writes: the overlap only happens for the first $K \times M - M/2$ values of each symbol, as was seen in chapter 3. For the remaining $M/2$ values the input data is simply stored in the memories. In the case of an add operation the existing value in memory is read from one of the memories, while the other memory is used in parallel to obtain the output value. This possibility to read from to addresses at the same time (from two memories, although they have the same content) allows this block to present a lower latency, and also to produce a higher throughput. However, the use of 4 memories of $2 \times K \times M$ mapped to BRAMs needed for this purpose makes this block expensive in terms of resources used, however it allows the system to present higher throughput values, as it is capable of outputting one sample each clock cycle.

4.1.6 Top Level

The top level system makes use of all the previously described modules, joining them to produce the full FBMC processing chain. Besides the developed modules described above, the system also makes use of an IFFT block which, as was mentioned previously was provided from a different

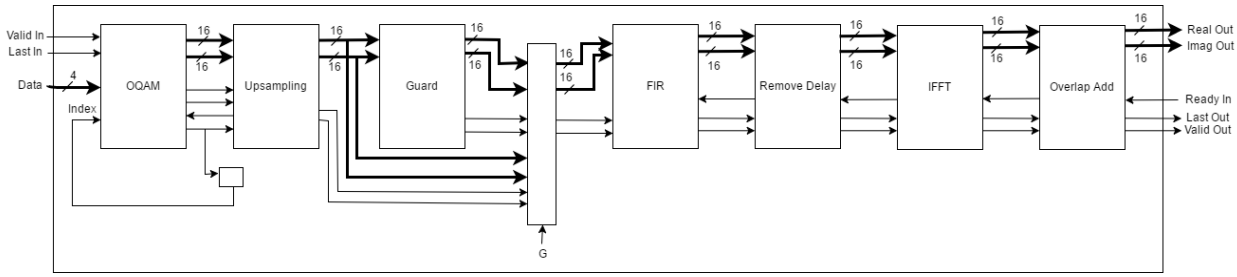

Figure 4.12: Detailed architecture of the overall processing chain

project[20]. One key aspect of this block that affected the development of other block is that, in order to obtain a constant stream of valid data at its output, the IFFT block should receive a constant stream of valid inputs. This means that there should be no intervals between each symbol's output.

Since the development of the processing chain was done having in mind this inclusion, its features are all compatible with those of the other modules. These features include both the data size (16 bit for the real component and 16 bit for the imaginary component); the fixed-point representation used (Q(5,11)); the use of an AXI4-Stream interface. This made for a smooth introduction of this module in the full processing chain.

The use of the AXI4-Stream protocol for synchronisation not only made the use of the IFFT block easier but also avoided additional work in order to synchronise the different modules, as this is done by the internal AXI4-Stream interface signals employed in each block.

The complete processing chain also includes AXI4-Stream signals in both its input and output, in order to ease its inclusion in higher level projects. Like what happens in a lower level, it will make this IP trivial to include in other projects, as long as they function with this interface. The system also accepts a 4 bit data input and produces two 16 bit data outputs, one for the real component and one for the imaginary component of the FBMC signal.

A more detailed, lower level view of the complete processing chain can be seen in figure 4.12 where these characteristics can be observed. From this figure it is possible to observe that the only processing done at this level is the one required to generate the OQAM's index, along with some additional logic to choose the input of the FIR block.

As was mentioned when describing the OQAM block, the value of the index input controls which component is delayed, either the real or imaginary component. This value should alternate between 1 and 0 with each transmitted symbol. To achieve this the number of valid outputs of the OQAM block is counted and when it reaches the number of outputs of a symbol (which is $M - 2 \times G$) it changes.

From this figure it is also possible to observe what was mentioned earlier: the connections between the different modules are straightforward and need no logic to achieve synchronisation, as this is achieved through the use of the previously described AXI4-Stream signals. The only connection in this chain which is not simply a wire connecting two modules is the one between the Upsampling, Guard and FIR blocks, where a multiplexer is used. This multiplexer is controlled

by the value of the user-defined parameter G, corresponding to the size of the guard bands to introduce; if this value is zero the Guard block will be bypassed and the FIR inputs will come from the Upsampling block; otherwise the chain functions "normally" with the FIR inputs coming from the Guard block.

One thing that should be noted is that the synchronisation efforts that were done ensure that all blocks can work at the same time, allowing for a lower latency of the complete processing chain.

Chapter 5

Project Results

The results from the development of this project are presented in this chapter. These are focused mainly on the developed FBMC baseband processor's resource and power utilisation, its timing properties and also the quality of the FBMC signal generated.

Once presented, a critical analysis of these results is made, by comparing the produced implementation with existing ones. Through this comparison conclusions will be made about the quality of this implementation. Comparisons with OFDM are also done, with the intent of assessing whether FBMC is a valid alternative waveform for 5G systems.

5.1 Results

The main outcome of this project is the development of a functionally correct FBMC baseband processor. The behaviour of this system was validated, as mentioned previously, by a comparison with the Matlab model available in [16]. Being a hardware implementation targeting an FPGA board, in this case the Zynq-7000, one of the most important results from this implementation is its resource usage. These values are obtained post-place and route implementation targeting Xilinx's Zynq - 7000 FPGA (both synthesis and implementation steps were run with the default settings) and with a system configuration with $K = 4$, $G = 10$ and $M = 1024$. This configuration was chosen as $K = 4$ and $M = 1024$ is one of the most common configurations found in the available literature, as seen in chapter 2. The value of G was chosen so as to be possible to keep it constant in all configurations (as an example, for a configuration of $K = 2$ and $M = 256$ it isn't possible to use a value of G over 128, due to the size of each symbol). It should also be noted that the LUTs mentioned in the following tables are the default ones for the target FPGA, which consist of 6-input LUTs.

Table 5.1 lists the usage results for this default configuration.

As can be seen, most of the system's blocks are not computationally expensive, with the IFFT block being the exception as the one which uses the most resources. Since this is an FS-FBMC implementation this result was expected as the IFFT operation has a size of $K \times M$, which entails a higher resource usage, due to its size.

Table 5.1: Post-Place and Route resource usage for the implemented FBMC baseband processor

Block	**Slice Regs**	**LUTs**	**DSP48E1**	**RAM BLKs**
FS-FBMC (total)	2204 (2.07%)	5334 (10.03%)	21 (9.54%)	55 (39.3%)
OQAM	68 (0.06%)	53 (0.1%)	0	0
Upsampler	36 (0.03%)	25 (0.05%)	0	0
Guard	112 (0.11%)	105 (0.19%)	0	0
FIR	275 (0.26%)	223 (0.42%)	6 (2.72%)	0
FIR delay	70 (0.06%)	49 (0.09%)	0	0
IFFT	1546 (1.45%)	4775 (8.98%)	15 (6.82%)	39 (27.86%)
Overlap and Add	81 (0.08%)	95 (0.18%)	0	16 (11.44%)

In chapter 3 it was mentioned that the FBMC system to be developed should have configurable user-defined parameters, to allow the designer to easily change the system's configuration. Using this capability it is possible to change the system's parameters and study how the usage results vary. By presenting configurations with different parameters it also becomes easier to compare this system with other FBMC and OFDM implementations, as the parameters employed by these might not be the same. For this study the parameters K and M were changed to different values, while G was kept constant as this parameter does not affect the system's usage in a significant way. The new usage results for these new configurations can be seen in table 5.2.

From this table a correspondence can be observed between the resource usage of the system and the product $K \times M$. This product is highlighted in a column of its own to make this relationship more evident. This result is to be expected since, as can be seen in table 5.1, the most expensive block in the processing chain is the IFFT block. The size of the IFFT operation performed by this block corresponds to $K \times M$ so it is expected that, as this product increases, the IFFT becomes more expensive and, as a consequence, the usage of the system also increases considerably, as a considerable amount of the system's results are spent in this block. Another key fact that explains this relationship is the fact that each symbol has a length of $K \times M$. This will affect some operations where the length of the symbol plays a key role, as is the case in the overlap and add block. As was seen in chapter 4, this block uses block RAMs to store a number of values consisting of double the size of a symbol. It is expected, then, that as the length of the symbol increases the block RAM usage of the system follows this increase. This result is also observable in table 5.2. It can also be

Table 5.2: Usage results for different configurations

Parameters	$K \times M$	**Slice Regs**	**LUTs**	**DSP48E1**	**RAM BLKs**
K = 4; M = 1024 (default)	4096	2204 (2.07%)	5334 (10.02%)	21 (9.54%)	55 (39.3%)
K = 2; M = 1024	2048	2118 (1.99%)	3505 (6.59%)	17 (7.72%)	29 (20.7%)
K = 4; M = 512	2048	2245 (2.11%)	3578 (6.73%)	21 (9.54%)	29 (20.7%)
K = 3; M = 512	1536	2705 (2.54%)	3905 (7.34%)	25 (11.36%)	30 (21.4%)
K = 4; M = 256	1024	1959 (1.84%)	3228 (6.07%)	18 (8.18%)	17 (12.1%)
K = 2; M = 256	512	1908 (1.79%)	2713 (5.09%)	14 (6.36%)	15 (10.7%)
K = 4; M = 128	512	2037 (1.91%)	2784 (5.23%)	18 (8.18%)	15 (10.7%)

Table 5.3: System's power consumption

	Power Consumption (W)
Total On-Chip Power	0.387
Dynamic	0.261
Device Static	0.126

observed that for different configurations with the same product $K \times M$ the system's usage remains approximately constant.

Another interesting result which can be extracted from this implementation is its power consumption. This information is obtained using Vivado's Power Analyser[1]. After a simulation the nodes' activity is extracted and stored in a SAIF file. This SAIF file contains information regarding the switching activity data of the design. In order to obtain more accurate results a post-implementation simulation is done; In this way the details about the routing resources and the timings for each path are complete. By performing the power analysis at this level the results obtained are the most accurate possible power estimation. The confidence level of this analysis is high, with a 100% matching of the design nets at an operating frequency of 100 Mhz. The results for the default configuration are presented in table 5.3.

This information can also be obtained for each block individually, so as to assess which blocks consume the most power. These results are presented in table 5.4.

A result similar to that of the resource usage is seen; The most expensive blocks of the implementation (in this case the ones with the highest dynamic power consumption) are the IFFT and Overlap and Add blocks. This is expected as these are the blocks that make use of block RAMs, which, as is possible to observe in table 5.5, are the resource with the highest power consumption. The power usage results of the system divided per resource type are shown in table 5.5.

From both the result of the system's total power consumption and of the consumption per block it is possible to conclude that the BRAMs are the resource with the highest power usage. This explains the results seen in table 5.4, where the overlap and add and the IFFT blocks present the highest power consumption, as these blocks are the ones which employ a larger number BRAMs.

[1] Xilinx, Vivado Design Suite User Guide Power Analysis and Optimization

Table 5.4: Dynamic power consumption per block

Block	**Dynamic power Consumption (W)**
FS-FBMC (total)	0.261
OQAM	<0.001
Upsampling	<0.001
Guard	0.001
FIR	0.006
FIR Delay	<0.001
IFFT	0.200
Overlap and Add	0.050

Table 5.5: Power consumption per component type

Component	**Power Consumption (W)**
Clocks	0.015
LUTs	0.027
Slice Regs	<0.001
Signals	0.031
Block RAMs	0.176
DSPs	0.007
Static Power	0.126

Varying the system's parameters a similar study to that of the system's resources can be done for the system's power consumption. Table 5.6 shows the results from this study.

The results for the system's power usage as its parameters vary follow the ones seen for its resource usage: as the product $K \times M$ increases so does its power consumption. This is easily explained having in mind the system's resource usage. As was seen, the system's usage increases as the product $K \times M$ increases. This increase in the system's resources will entail a higher power consumption, especially considering the higher use of BRAMs with a higher value of $K \times M$, which correspond to the resource with the highest power consumption.

Knowing how the system's power usage results vary with its parameters, it is also relevant to observe how this value behaves with its frequency. This result was measured for four different frequency values: the default value of 100 MHz, 50 MHz, 10 MHz and also the system's maximum frequency of 113.96 MHz. These values are shown in table 5.7.

The values shown in this table are obtained for the default configuration of $K = 4$, $M = 1024$ and $G = 10$. It is possible to observe from these values that the dynamic power consumption increases with the increase of the system's frequency. The total on-chip power of the system also increases, as the device static power is constant for all frequency values. This is expected for an SRAM-based FPGA implementation.

Besides the system's usage and power behaviour is is also important to measure its timing properties. By varying the frequency and analysing the system's worst slack of all timing paths it is possible to obtain the maximum frequency of the system. For this implementation, with the

Table 5.6: Power usage for different configurations

Parameters	$K \times M$	**Total on-chip power (W)**	**Dynamic (W)**	**Device Static (W)**
K = 4; M = 1024 (default values)	4096	0.387	0.261	0.126
K = 2; M = 1024	2048	0.281	0.158	0.123
K = 4; M = 512	2048	0.282	0.159	0.123
K = 3; M = 512	1536	0.279	0.156	0.123
K = 4; M = 256	1024	0.235	0.113	0.122
K = 2; M = 256	512	0.215	0.093	0.122
K = 4; M = 128	512	0.215	0.093	0.122

Table 5.7: Power usage for different frequency values for the default configuration

Frequency (MHz)	Total On-Chip Power (W)	Dynamic (W)	Device Static (W)
113.96 (maximum)	0.415	0.289	0.126
100 (default)	0.387	0.261	0.126
50	0.292	0.167	0.125
10	0.213	0.089	0.124

same configuration mentioned above of $K = 4$, $G = 10$ and $M = 1024$ this value is 113.96MHz.

It is also possible to measure the implemented FBMC baseband processor's latency, and the one of its constituting blocks along with its throughput. This result is also important to understand where the system's bottlenecks are located in the general processing chain. The results of the latency for the system's blocks are seen in table 5.8.

The results presented in this table are obtained for the default configuration. The latency of some of the system's blocks, such as the one which removes the delay from the FIR and the Overlap and Add block change with the system's parameters and can be expressed as K and $K \times M + 1$, respectively. From this table it is possible to observe that the system's latency comes mainly from both the IFFT and Overlap and Add blocks. The latency of the IFFT can be explained both with the way this calculation is done, which prevents this block from generating valid outputs immediately after the reception of valid inputs. This block also performs a shift operation to the results of the IFFT, which adds to this latency. The latency of the Overlap and Add block is caused by the generation of an initial non-valid symbol, which happens before any valid inputs are received. It is, then, necessary to wait $K \times M$ clock cycles, which is the size of one symbol, plus 1 clock cycle before obtaining valid outputs from this block.

In both the IFFT and Overlap and Add block's a shift operation is performed. Due to the use of Block RAMs the accesses to these memories must be carefully timed, entailing a higher number of clock cycles to perform this operation. The high latencies of these blocks are the system's bottleneck.

It is also important to quantify the latency of the complete FBMC baseband processor, which consists of the sum of the latencies of the different blocks. For the scenario used for the results in table 5.8 this corresponds to 20533 clock cycles, or 20.53 μs considering the system's operating frequency of 100 MHz.

Table 5.8: Latency of the system's blocks

Block	Latency (clock cycles)
OQAM	1
Upsampling	1
Guard	2
FIR	2
FIR Delay	4
IFFT	16426
Overlap and Add	4097

Along with the system's latency, another important characteristic that can be measured is its throughput. The implemented FBMC baseband processor is capable of generating a new sample each clock cycle, after the initial delay of 20.53 μs. This means that the throughput of this system is 100 MSamples/s, considering the operating frequency of 100 MHz. Taking into account the maximum frequency of the system, which corresponds to 113.96 MHz it can be concluded that the maximum throughput for this implementation is 113.96 MSamples/s.

Lastly, it is also important to verify if the result of the implemented FBMC system corresponds to the theoretical expected values. To confirm this, a comparison between the results from this system and the ones obtained from the aforementioned software model is done. For a configuration of $K = 4$, $M = 1024$ and $G = 10$ the maximum obtained error per symbol is 4.7×10^{-3}. As mentioned before in chapter 3, this result was to be expected due to the rounding errors introduced by the fixed point calculations. The software model uses floating point calculations, hence the difference in these values. With the implemented fixed point configuration, Q(5,11) a resolution of $1/2^{11} = 4.88 \times 10^{-4}$ is obtained. This limited resolution, coupled with the need to truncate the results in some operations leads to this error at the end of the processing chain.

A different characteristic that should be analysed is the produced signal's spectral behaviour, as this is one of FBMC's key characteristics. This can be done by looking at the periodogram of a specific symbol. A periodogram is an estimate of the spectral density of a signal. It is expected that observing the periodogram of the obtained signal at the end of the implement system FBMC signal's main characteristics are patent, such as its spectral containment. Using Matlab the periodogram of one of the symbols after the initial transient period was plotted. The result is shown in figure 5.2. From this figure the expected spectral behaviour of an FBMC symbol can be observed as there is a clear containment of this symbol in its frequency band. Some noise can be seen in this figure, which stems from the already mentioned rounding errors introduced from the fixed point calculations. The attainment of this result is important, as this is one of FBMC's fundamental characteristics, because it allows the maximisation of the use of the frequency spectrum, as large guard bands aren't needed. As a comparison, the theoretical result expected for the same symbol can be seen in figure 5.1. The behaviour shown in this figure is similar to the one obtained with the practical results although with much less noise. The frequency containment can be seen more easily, with a clear rolloff from the signal to the guard band.

5.2 Analysis

The results presented in the previous section should be studied and compared with those of other FBMC and OFDM systems, namely those presented in chapter 2, in order to draw important conclusions about this implementation in particular and about FBMC as a waveform in general. When comparing, it should be taken into account both the parameters in use by both implementations but also the FBMC architecture in use. All results shown in table 2.4 correspond to PPN-FBMC architectures, while the implemented in this work consists of an FS-FBMC implementation. Comparing with the implementation presented in [13] (with $K = 4$ and $M = 1024$) the implementation

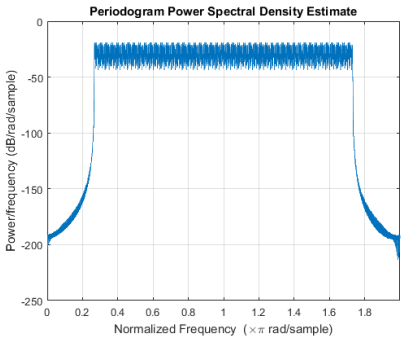

Figure 5.1: Theoretical periodogram of a symbol

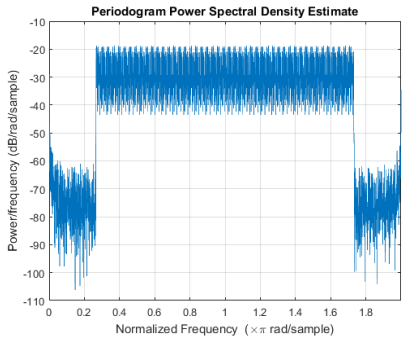

Figure 5.2: Obtained periodogram of a symbol

in this work is considerably less complex: it uses 20% of its Slice Registers, 60% of its LUTs and of its DSPs. The RAM BLK usage is also lower with its use being around two thirds lower. It should be noted, however, that the results from [13] were obtained for a different FPGA family, Xilinx Kintex-7 FPGA, while the implementation in this project is for a Xilinx Zynq-7000.

This is an important result as the IFFT employed by the FS-FBMC is K times larger than the one used in the implementation proposed in [13]. It was expected, then, that the developed implementation was considerably more complex, which isn't the case. The two different architectures implemented in [12] use a configuration of $K = 4$ and $M = 512$. Comparing these to the equivalent configuration for the developed implementation, which can be seen in table 5.2, it can be concluded that the implemented architecture is, once again, less complex, although there is no information regarding the Block RAMs employed in both implementations presented in [12].

From what was seen in chapter 2, it was expected that an FS-FBMC implementation presented higher usage results than a PPN-FBMC one, mostly due to the size of its IFFT block being considerably larger. These results, however, are not observed in this analysis: when compared with other implementations with similar configurations, the implemented architecture presents lower usage results.

In the scope of this project it is also useful to compare this implementation with ones of OFDM systems, not only other FBMC systems. In that sense the results were compared with an existing OFDM system[2], whose results are shown in table 5.9.

Several configurations for the OFDM system are studied in this project; The resource usage of the ones with comparable parameters to those of the implemented FBMC system are presented in

[2] Values from an unpublished report communicated by Mário Ferreira

Table 5.9: OFDM system's usage

	IFFT size	**Slice Regs**	**LUTs**	**DSP48E1**	**RAM BLKs**
OFDM 1	256	2356 (2.21%)	2748 (5.17%)	11 (5.00%)	5.5 (3.93%)
OFDM 2	512	2523 (2.37%)	2975 (5.59%)	14 (6.36%)	8.5 (6.07%)
OFDM 3	1024	2588 (2.43%)	3497 (6.57%)	14 (6.36%)	12 (8.57%)
OFDM 4	2048	3004 (2.82%)	3948 (7.42%)	17 (7.73%)	19.5 (13.93%)

Table 5.10: OFDM system's power consumption

	IFFT size	**Dynamic (W)**	**Device Static (W)**
OFDM 1	256	0.132	0.256
OFDM 2	512	0.146	0.257
OFDM 3	1024	0.174	0.259
OFDM 4	2048	0.219	0.262

this table. It should also be noted that the OFDM system portrayed by these results is a reconfigurable system. This reconfiguration inserts some overhead in the system that must be accounted for when comparing the values. The overhead introduced corresponds to 1792 Slice Regs, 1354 LUTs and 1.5 RAM blocks.

Comparing these results with the ones obtained for the implemented configuration it can be concluded that the OFDM implementations are considerably less expensive than FBMC ones. From what was seen in chapter 2 this result was to be expected, as FBMC's advantages when compared with OFDM (its increased spectral efficiency, for example) come at the cost of a higher resource usage. One of the characteristics of FBMC is also the increased size of its IFFT block, which also contributes to this result. This is the reason why the FBMC waveform has not been considered a viable option until now.

The system's power usage can also be compared to that of the already mentioned OFDM project, although it is not possible to compare it to other FBMC implementations, as this data is not available. This OFDM system's power usage for the same configurations shown in table 5.9 can be seen in table 5.10.

These values are obtained in the same conditions as the ones from the developed FBMC system, listed in table 5.6. Comparing these results it can be seen that the dynamic power consumption of the implemented system is lower than the one for the equivalent OFDM configurations, although this FBMC system presents a higher resource usage. In terms of static power consumption the FBMC presents a considerable decrease in this result, around 50%.

The system's maximum frequency can also be compared to other systems. Both implementations mentioned in [12] present a maximum clock frequency of 220 MHz which is considerably higher than the one achieved in this FBMC implementation.

The system's throughput can also be compared to that of other implementations. As mentioned in the previous section this value corresponds to 100 MSamples/s at a frequency of 100 MHz or 113.96 MSamples/s at the maximum frequency of the system, which consists of the system's maximum throughput. In the aforementioned OFDM system the output throughput achieved corresponds to 94.5 MSamples/s, which exceeds LTE's most demanding requirements of 30.72 MSamples/s [20]. It can, then, be concluded that the implemented FBMC system has a similar performance to that of the mentioned OFDM system, but lower than that of other existing FBMC implementations.

Chapter 6

Conclusions and future work

In this chapter the most important considerations about this project's development are presented. Some key ideas about the initial OFDM vs FBMC problem are discussed based in the results obtained from this project. Some key tasks that can be made to improve and cap off this project are also presented and discussed.

6.1 Conclusions

Considering the initial objectives of implementing and validating a baseband processor for FBMC transmission it can be said that these objectives were fulfilled as a functional version of a system of this kind was developed and studied, with its results being presented and compared to those of other implementations. In order to produce this implementation a thorough study had to be performed regarding the FBMC waveform and its existing architecture proposals. Through this study it was possible to compare this waveform with OFDM, the other main 5G candidate waveform.

The produced implementation was analysed and its performance was measured and compared with that of other implementations. Analysing these results in the context of the 5G waveform alternatives, having in mind the previous study it is possible to draw comparisons between the different implementations of both FBMC and OFDM. It was also possible to compare the obtained results with those expected from an FBMC baseband processor. From these comparisons it was seen that this implementation presents similar results to those of existing implementations, with better characteristics in terms of resource usage, although with a lower throughput than some implementations. The results expected from the characteristics of each waveform were also verified, such as the resource usage of FBMC being higher than that of OFDM systems.

These results were discussed in chapter 5, where it was concluded that this implementation presented good results regarding its resource and also presented a good result regarding the system's throughput, although it is still lower than some of the existing implementation. This value is similar to that in existing OFDM implementations and is suitable for 4G standards, however new 5G systems will require higher values in terms of the system's throughput.

Some improvements can be done in order to better the system's performance, some of which were already briefly mentioned in chapter 5, namely regarding the error introduced by the fixed point representation in use. This would allow the system to present a lower error in its output values.

In chapter 2 it was already seen that the FBMC waveform presents several benefits when compared to OFDM, especially its higher spectral efficiency, as this is an important result when in the context of 5G communications. Its main downside corresponds to its higher complexity. The analysis from chapter 2, coupled with the results from this project show that it is possible to produce FBMC implementations with similar complexity results as OFDM. The spectral behaviour obtained both in other implementations and in the present follow the expected results, where a sharp roll-off is observed in the frequency spectrum.

The results of this implementation and also of different existing ones, coupled with the theoretical knowledge about FBMC's main characteristics show that FBMC is a valid candidate waveform for 5G systems.

6.2 Future work

To improve this project some additional tasks can be performed, such as the implementation of an FPGA prototype or studying how to exploit dynamic reconfiguration for different parameter configurations or for switching between OFDM and FBMC. Some improvements to the system's architecture can also be done, namely in the implementation of fixed point arithmetic, as this would allow the achievement of lower rounding errors. It is also relevant to develop an FBMC receptor to be able to implement a testing environment complete with both transmission and reception parts.

The implementation of a prototype using the Zynq-7000 FPGA would provide a different level of validation of the developed system. Besides validating the system's function this would also provide more accurate results for the system's performance and would help compare its performance to that of other implementations where the same prototyping has been done.

The addition of reconfigurability to this system would increase the system's flexibility and adaptability, by allowing the configuration of the system to change in run-time.

Finally, by developing the receiving end of the FBMC transmission chain it would be possible to analyse the performance of an FBMC receptor. This is also relevant when comparing this waveform to the existing ones as it would provide information on whether the results seen in the transmission end are also seen in the receiving end.

References

[1] F. Schaich. Filterbank based multi carrier transmission (FBMC); evolving OFDM: FBMC in the context of WiMAX. In *Wireless Conference (EW), 2010 European*, pages 1051–1058, April 2010.

[2] M. Bellanger. Phydyas project, "Documents D3.1 and 5.1" Specification. Technical report, 2008.

[3] M. Bellanger. FBMC physical layer : a primer. Technical report, 2010.

[4] Wikiwand. Finite impulse response, February 2017.

[5] Uwe Meyer-Baese. *Digital Signal Processing with Field Programmable Gate Arrays*. Springer, November 2016.

[6] Greg Jue and Sangkyo Shin. Implementing a Flexible Testbed for 5G Waveform Generation and Analysis, June 2016.

[7] T. Ihalainen, A. Viholainen, T. H. Stitz, and M. Renfors. Generation of Filter Bank-Based Multicarrier Waveform Using Partial Synthesis and Time Domain Interpolation. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 57(7):1767–1778, July 2010.

[8] Mário Ferreira. Reconfigurable FPGA-Based FFT Processor for OFDM Cognitive Radio Applications. Technical report, March 2015.

[9] Maurice Bellanger. FS-FBMC: a flexible robust scheme for efficient multicarrier broadband wireless access. 2012.

[10] L. Varga and Z. Kollár. Low complexity FBMC transceiver for FPGA implementation. In *Radioelektronika (RADIOELEKTRONIKA), 2013 23rd International Conference*, pages 219–223, April 2013.

[11] James Cooley, Peter Lewis, and Peter Welch. The Fast Fourier Transform and Its Applications. *IEEE Transactions on Education*, 12(1):27–34, March 1969.

[12] J. Nadal, C. Abdel Nour, and A. Baghdadi. Low-Complexity Pipelined Architecture for FBMC/OQAM Transmitter. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 63(1):19–23, January 2016.

[13] Robin Gerzaguet, Nikolaos Bartzoudis, Leonardo Gomes Baltar, Vincent Berg, Jean-Baptiste Doré, Dimitri Kténas, Oriol Font-Bach, Xavier Mestre, Miquel Payaró, Michael Färber, and Kilian Roth. The 5G candidate waveform race: a comparison of complexity and performance. *EURASIP Journal on Wireless Communications and Networking*, 2017(1):13, January 2017.

[14] Y. Qi and M. Al-Imari. An enabling waveform for 5G; QAM-FBMC: Initial analysis. In *2016 IEEE Conference on Standards for Communications and Networking (CSCN)*, pages 1–6, October 2016.

[15] YongJu Won, JongGyu Oh, JinSeop Lee, and JoonTae Kim. A Study of an Iterative Channel Estimation Scheme of FS-FBMC System. *Wireless Communications and Mobile Computing*, 2017:e6784142, 2017.

[16] Mathworks. FBMC vs. OFDM Modulation - MATLAB & Simulink Example, June 2017.

[17] P. Weitkemper, J. Koppenborg, J. Bazzi, R. Rheinschmitt, K. Kusume, D. Samardzija, R. Fuchs, and A. Benjebbour. Hardware experiments on multi-carrier waveforms for 5G. In *2016 IEEE Wireless Communications and Networking Conference*, pages 1–6, April 2016.

[18] V. Berg, J. B. Doré, and D. Noguet. A flexible FS-FBMC receiver for dynamic access in the TVWS. In *2014 9th International Conference on Cognitive Radio Oriented Wireless Networks and Communications (CROWNCOM)*, pages 285–290, June 2014.

[19] ARM. AMBA 4 AXI4-Stream Protocol.

[20] M. L. Ferreira, A. Barahimi, and J. C. Ferreira. Dynamically reconfigurable LTE-compliant OFDM modulator for downlink transmission. In *2016 Conference on Design of Circuits and Integrated Systems (DCIS)*, pages 1–6, November 2016.

[21] Xilinx. Polyphase Filter Bank Channelizer. Application note 1161, March 2013.

[22] Paul D Fiore. Low-Complexity Implementation of a Polyphase Filter Bank. *Digital Signal Processing*, 8(2):126–135, April 1998.

[23] Avnet. ZedBoard (Zynq Evaluation and Development) Hardware User's Guide, January 2014.

[24] B. Farhang-Boroujeny. OFDM Versus Filter Bank Multicarrier. *IEEE Signal Processing Magazine*, 28(3):92–112, May 2011.

[25] M. Bellanger. FS-FBMC: An alternative scheme for filter bank based multicarrier transmission. In *2012 5th International Symposium on Communications, Control and Signal Processing*, pages 1–4, May 2012.

[26] Amir Aminjavaheri, Arman Farhang, Nicola Marchetti, Linda E. Doyle, and Behrouz Farhang-Boroujeny. Frequency spreading equalization in multicarrier massive MIMO. Institute of Electrical and Electronics Engineers Inc., September 2015.

[27] Fa-Long Luo and Jianzhong Zhang. *Signal Processing for 5G: Algorithms and Implementations*. Wiley, October 2016.

[28] Randy Yates. *Fixed-Point Arithmetic: An Introduction*, January 2013.

[29] Jeremy Nadal, C. Abdel Nour, A. Baghdadi, and H. Lin. Hardware prototyping of FBMC/O-QAM baseband for 5G mobile communication.

Appendix A

Code

A.1 Matlab Script

The software model used to validate the implementation is available in [16]. It consists of a Matlab model of a frequency spreading FBMC transmitter.

```
close all; clear; clc;

%%Read results file
file4 = fopen( 'Chain_Results.txt', 'r' );
file5 = fopen( 'Sim_Input.txt', 'wb' );
Chain =  fscanf(file4,'%d %d');

j=1;
for k=1:2:size(Chain)
    Chain_re(j)=Chain(k);
    Chain_im(j)=Chain(k+1);
    j=j+1;
end

Chain_re = Chain_re';
Chain_im = Chain_im';
Chain_re_FLOAT = Chain_re./(2^11);
Chain_im_FLOAT = Chain_im./(2^11);
Chain_FLOAT = Chain_re_FLOAT + Chain_im_FLOAT.*i;

%%System Parameters
numFFT = 1024;           % Number of FFT points
numGuards = 137;         % Guard bands on both sides
K = 4;                   % Overlapping symbols, one of 2, 3, or 4
numSymbols = 5;        % Simulation length in symbols
bitsPerSubCarrier = 4;   % 2: 4QAM, 4: 16QAM, 6: 64QAM, 8: 256QAM
```

```
%%Filter Bank Multi Carrier Modulation
% Prototype filter
switch K
    case 2
        HkOneSided = sqrt(2)/2;
    case 3
        HkOneSided = [0.911438 0.411438];
    case 4
        HkOneSided = [0.971960 sqrt(2)/2 0.235147];
    otherwise
        return
end

% Build symmetric filter
Hk = [fliplr(HkOneSided) 1 HkOneSided];

% QAM symbol mapper
 qamMapper = comm.RectangularQAMModulator(...
     'ModulationOrder', 2^bitsPerSubCarrier, ...
     'BitInput', true, ...
     'NormalizationMethod', 'Average power');


% Transmit-end processing
%   Initialize arrays
L = numFFT-2*numGuards;  % Number of complex symbols per OFDM symbol
KF = K*numFFT;
KL = K*L;
dataSubCar = zeros(L, 1);
dataSubCarUp = zeros(KL, 1);

sumFBMCSpec = zeros(KF*2, 1);
sumOFDMSpec = zeros(numFFT*2, 1);

numBits = bitsPerSubCarrier*L/2;    % account for oversampling by 2
inpData = zeros(numBits, numSymbols);
rxBits = zeros(numBits, numSymbols);
txSigAll = complex(zeros(KF, numSymbols));
symBuf = zeros(2*KF, 1);

% Loop over symbols
for symIdx = 1:numSymbols
    inpData(:, symIdx) = randi([0 1], numBits, 1);

    %QAM modulation
```

```
        modData = step(qamMapper,inpData(:, symIdx));

    % OQAM Modulator: alternate real and imaginary parts
    if rem(symIdx,2)==1     % Odd symbols
        dataSubCar(1:2:L) = real(modData);
        dataSubCar(2:2:L) = 1i*imag(modData);
    else                    % Even symbols
        dataSubCar(1:2:L) = 1i*imag(modData);
        dataSubCar(2:2:L) = real(modData);
    end

    % Upsample by K, pad with guards, and filter with the prototype filter
    dataSubCarUp(1:K:end) = dataSubCar;

    dataBitsUpPad = [zeros(numGuards*K,1); dataSubCarUp; zeros(numGuards*K,1)];

    MyX = filter(Hk,1,dataBitsUpPad);
    X = [MyX(K:end); zeros(K-1,1)];


    % Compute IFFT of length KF for the transmitted symbol
    txSymb = fftshift(ifft(X));


    %Overlap and Add
    symBuf = [symBuf(numFFT/2+1:end); zeros(numFFT/2,1)];
    symBuf(KF+(1:KF)) = symBuf(KF+(1:KF)) + txSymb;

    currSym = symBuf(1:KF);

    txSigAll(:,symIdx) = currSym;
end
```