A Survey of Implementation Issues from a FPGA-based Multi-Standard Receiver for SATCOM Handheld Receiver on the TMS DSP Platform Application

Master Thesis

APPLIED SIGNAL PROCESSING
AND IMPLEMENTATION (ASPI)

Group 1044
Jean-Michel LORY

AALBORG UNIVERSITY

Title:
Implementation from a FPGA-based multi-standard receiver on DSP platform

Theme:
Implementation for Polyphase Channelizer

Project Period:
10th Semester
February 2009 to June 2009

Project Group:
ASPI 09gr1044

Participant:
Jean-Michel Lory
jlory@es.aau.dk

Supervisors:
Peter Koch (CSDR)
Mehmood-Ur-Rehman Awan (CSDR)

Publications: 5

Number of pages: 68

Appendices: 1 CD-ROM

Finished: 3rd of June 2009

Abstract

This Master Thesis project for the "Applied Signal Processing and Implementation" specialization at Aalborg University is a study of polyphase channelizer for multi-standard radio receiver on DSP Platform. The project focuses on SATCOM handheld receiver, which require efficient FIR filters utilization to process data received. In our case, WLAN and UMTS applications are chosen. Bandpass sampling techniques at 840MHz are used to alias both combined bands. The output channels are required at baseband and with a sampling rate of 20MHz and 61.44MHz. Simulations are performed on MatLab. The prototype filter for WLAN standard is 150-taps length, partitioned in 5 sub-filters. In UMTS case, the length of the prototype filter is 2520 taps, partitioned in 210 sub-filters. Polyphase filter bank structures are studied. Parallel MAC is selected for the final implementation. The estimation of number of cycles to process data for one WLAN sub-channel is done. This estimation does not respect time constraints (process time through the sub-channel is bigger than 2 data sample intervals). Some optimizations are described to reduce the execution time without improvements to respect constraints. The implementation shows that the execution time is bigger than estimation. Optimizations developed before allow reducing considerably this time but it does not respect time constraints yet. It is concluded that it is not possible to implement this application on DSP TMS320C6713 due to frequency specifications of the application.

Preface

This report is the documentation for the Master Thesis in Applied Signal Processing Implementation (ASPI), and is written by the group 09gr1044 at the Institute of Electronics Systems at Alborg University (AAU). This report, entitled "A Survey of Implementation Issues from a FPGA-based Multi-Standard Receiver for SATCOM Handheld Receiver on the TMS DSP Platform Application", spans from February 1st, 2009 to June 3rd, 2009. Peter Koch, Associate Professor at AAU and Mehmood-Ur-Rehman Awan, PhD fellow at AAU, supervised it during this period.

The introduction provides a general discussion on SDR, as well as the problem definition. Polyphase channelizer, for UMTS and WLAN, is examined in the second chapter. Model Design is applied on the application of this project. Simulations are performed and complexity algorithm is carried out. Then, the algorithm is mapped on the DSP processor and tests are carried out for the final implementation. Finally, the results are concluded. A CD is attached to the report. It contains the code and test material produced during the project as well as an electronic version of this report in pdf.

Jean-Michel Lory

Table of Contents

1 Introduction

1.1 General Discussion on SDR

Software Defined Radio (SDR) system is a radio communication system (composed of filters, amplifiers, modulators, etc) that is defined mainly by software instead of hardware. SDR is achieved on many applications using receivers or transmitters for handheld devices (cellular phone, laptop, Global Positioning Syst, etc). The flexibility of a software radio system is to be able to process on multiple applications, without being blocked by a particular standard. This system has to be compatible with any defined radio mobiles. It is reconfigurable, for example on Digital Signal Processor (DSP), which implement in real time radio interface and upper layer protocols. Two main goals have to be respect to develop a software radio system:

- The Analog-Digital Conversion has to be as close as possible to the Antenna, in the Radio Frequency (RF) domain.
- The Application Specific Integrated Circuits (ASICs) has to be replaced by DSPs for baseband signal processing, in order to define as many radio functionalities as possible in software.

The ideal concept of a SDR system is to go through the digital domain as close as possible to the Antenna. The Figure [1.10] shows the scheme of an ideal software radio receiver. This ideal system is not completely realizable, due to the problem of sampling of the Radio Frequency (RF) signal.

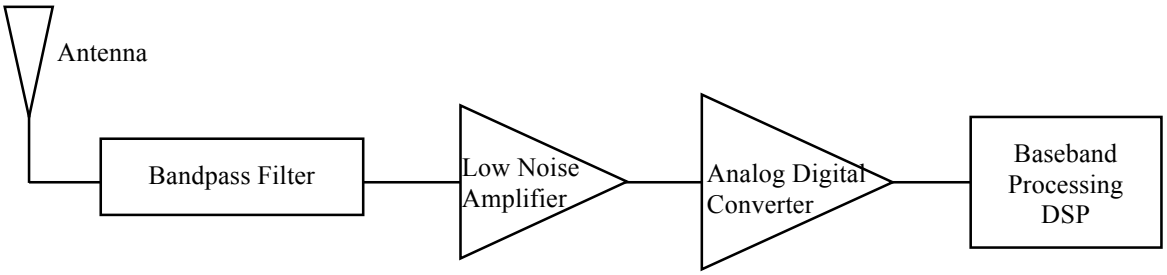


Figure 1.10: *Ideal software radio receiver. The ADC is as close as possible to the Antenna.*

Indeed, it is impossible to build Antennas and LNAs on a bandwidth ranging from hundreds of megahertz to tens of gigahertz. Moreover, problem of time variation make A/D conversion at RF very difficult [2].

The possible solution to realize a software radio receiver is shown in Figure [1.11]. This architecture is composed of three stages: RF stage, Intermediate Frequency (IF) and Baseband (BB) stage. The RF stage is totally analog whereas digital conversion is done in IF one.

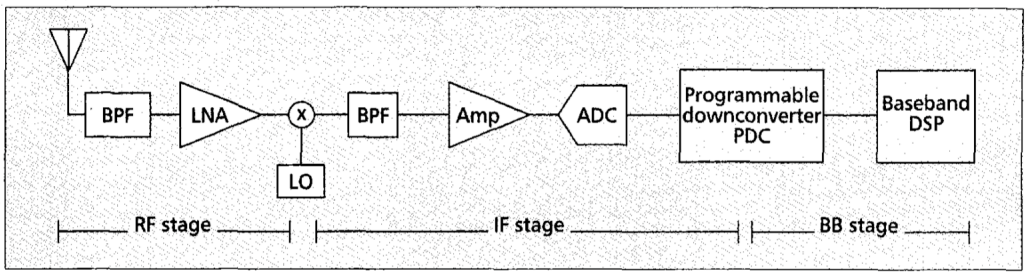


Figure 1.11: *Possible solution for software radio receiver. It composes of three main stages: RF (totally analog), IF (where the digital conversion is done) and BB (where the DSP processes the data) [1].*

The RF stage is composed of Antenna, Bandpass Filter and Low Noise Amplifier. Then, by means of heterodyning, the frequency is reduced and the digital conversion is done in the IF domain. Finally, data are processed in BB domain (use of PDC and DSP).

The motivations for the SDR are multiples. Firstly, commercial wireless communication industry is facing problems due to constant evolution of protocols standards. Indeed, these networks (from 2G to 3G, and 4G now) are different and some problems appear when a new generation of network is created, especially for the migration of the network from one generation to another one.

Another motivation is the incompatibility of wireless network technologies between different countries. For instance, the main wireless network in Europe is GSM whereas in USA, CDMA2000 is used. It is a big problem for people who travel a lot from Europe to USA, and in general from one continent to another one.

Then, problems in rolling out new services due to wide spread presence of legacy subscriber handsets motivate to develop SDR.

1.2 Project subject

The goal of this project is to implement an algorithm capable of receiving multiple standards and processing them (such as down-conversion and filtering operation). These standards can be Bluetooth, ZigBee, DVB, DAB, GSM, WiMAX, etc but it is limited to only two standards: WLAN and UMTS. Some specifications of UMTS and WLAN standards are summarized in Table [1.20].

	UMTS	WLAN
Frequency Band	1.920-1.980 GHz: UL 2.110-2.170 GHz: DL	2.4-2.4835 GHz
Channel Bandwidth	3.84 MHz	16.6 MHz
Receiver Sensitivity	-117dBm	-82 to -65 dBm

Table 1.20: *Specifications of UMTS and WLAN standards*

The block diagram of the system is shown in Figure [1.21]. Firstly, the signal is received on the Antenna and is passed through the LNA. This amplifier allows adding a low noise. Then, the bandpass sampling block samples the signal just after the LNA; therefore, at the output of this block, the signal is in the digital domain and at the intermediate frequency. The analog part is only composed of the Antenna and the LNA. A Band-select filter is used to select the complete band after the sampling (UMTS and WLAN information). There are two separate paths now for UMTS and WLAN. A Bandpass filter and the channelizer compose each path. The Bandpass filter is used to select the appropriate standards (UMTS or WLAN). The IF signal is downconverted, filtered and downsampled through the channelizer. On the output of the channelizer, each channel is at the desired sampling rate. It can be noticed that bandpass filter after the bandpass sampling is removed for the rest of the project. Indeed, bandpass filters in both paths (UMTS and WLAN) are sufficient to select the required passband of the input sampled signal to be processed later in the channelizer.

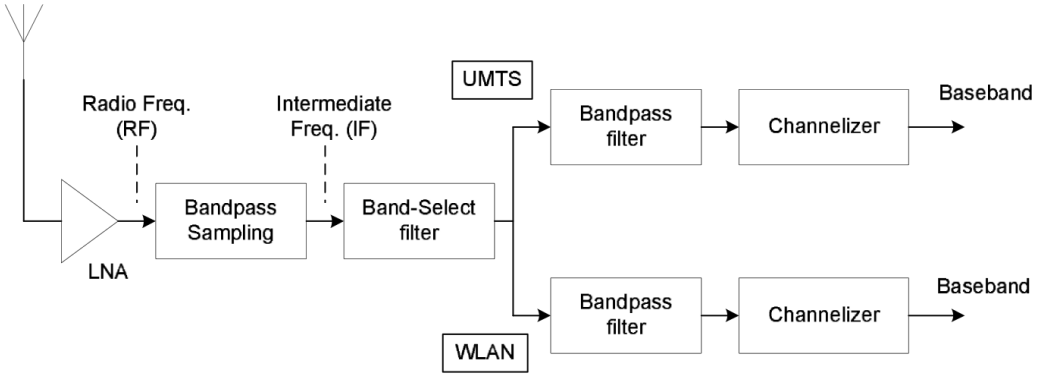


Figure 1.21: *Block diagram of the whole system [12].*

To finish on the description of the project, the architecture where the algorithm is implemented has to be chosen. The purpose of the project is to examine the implementation of polyphase channelizer for UMTS and WLAN application on a DSP platform, i.e. TMS320C6713. The DSP platform is used in this project for:

- Implementation of the algorithm.
- Evaluation of the performance, in particular in terms of execution time.

The DSP uses VLIW architecture [Appendix B], making excellent choice for the multi-channel and multifunction applications. It can execute up to maximum 8 32-bit instructions per cycle. It composes of two data paths. Each data path contains 16 32-bit registers, one multiplier and three ALUs. Both of these data paths can work in parallel, allowing optimized execution for computation. The Figure [1.22] shows the platform used for this project.

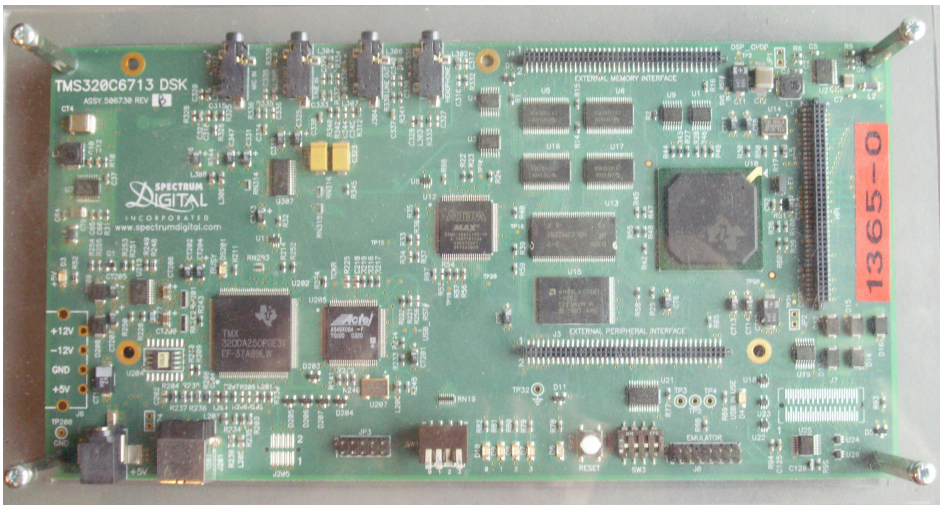


Figure 1.22: *Picture on the DSP platform (TMS320C6713).*

1.5 Problem Definition

The project problem specification is

Performance evaluation of a Digital Signal Processor implementation of a Multi-Standard Digital Radio Receiver

2 Application Description

In this chapter, the application is described. A quick description of multi-standard software radio receiver is done, in particular for WLAN and UMTS standards. Then, WLAN and UMTS polyphase channelizers are designed. Finally, a technical problem is found out and solutions are studied to resolve it.

2.1 Project Specification

The main idea of SDR is to carry out a lot of operations on an input signal in the digital domain, it means, for the study of a receiver (as this is the case of this project), the Analog-Digital Conversion (ADC) must be realize as near the antenna as possible.

For the multi-standard software radio receiver studied in this report, UMTS and WLAN standards are chosen and it is required down-conversion to baseband separately. The spectral representation of these two standards is shown in Figure [2.10]. The UMTS bandwidth is 60 MHz with 12 channels for downlink whereas WLAN bandwidth is 83.5 MHz with 3 non-overlapped channels.

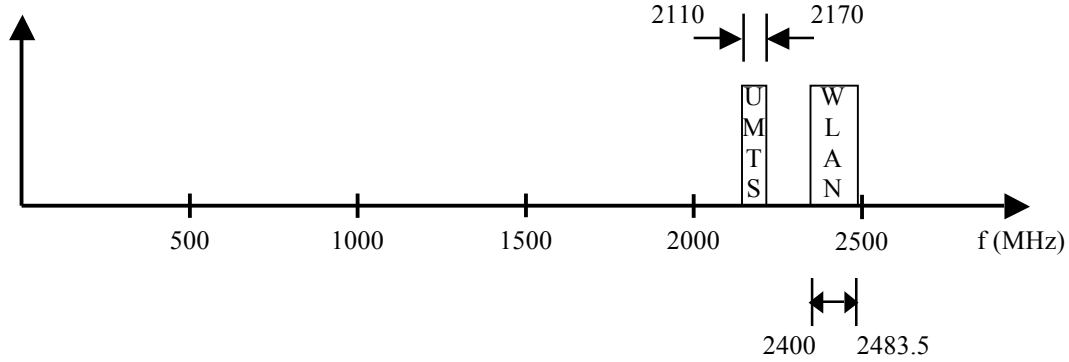


Figure 2.10: *Spectral representation of UMTS and WLAN standards.*

As shown in Figure [2.10], the spectrum combining UMTS and WLAN signals has a bandwidth of 373.5 MHz. According to the Nyquist-Shannon sampling criterion [11], the sampling frequency must be superior or equal to the double bandwidth B (2.11).

$$f_s \geq 2B \tag{2.11}$$

The bandpass sampling is thus 747 MHz. But, in the combined spectrum for UMTS and WLAN shown in Figure [2.12], there is an unused spectrum between them. To overlap this unused spectrum, the sampling frequency can be decreased. It has been hit and tried to reduce it at 676 MHz.

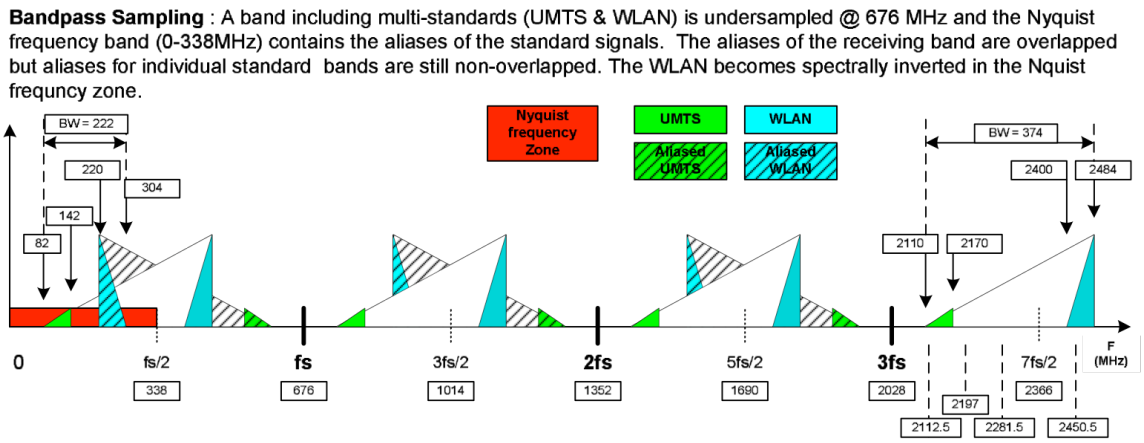


Figure 2.12: *Combined Spectrum of UMTS and WLAN sampled at 676 MHz [12].*

According to the specifications of UMTS and WLAN standards, UMTS bandwidth (WLAN) is 5 MHz (24 MHz) wide. It means the UMTS spectrum contains 12 channels of 5 MHz with 5 MHz of spacing between each inter-channels carriers. For WLAN, 3 non-overlapped channels of 24 MHz with a space of 30 MHz between inter-channels carriers. The representation of all these channels is described in Figure [2.13]. The bandpass corresponds to the aliases of the two combined signals in the Nyquist-zone shown in Figure [2.12]. All the channels are downsampled and downconverted (to baseband). The target output sample rate is 20 MHz (61.44 MHz) for WLAN (UMTS). 61.44 correspond to the product of the UMTS bandwidth (3.84 MHz) and the oversampled ratio of 16, which is taken into account in this case.

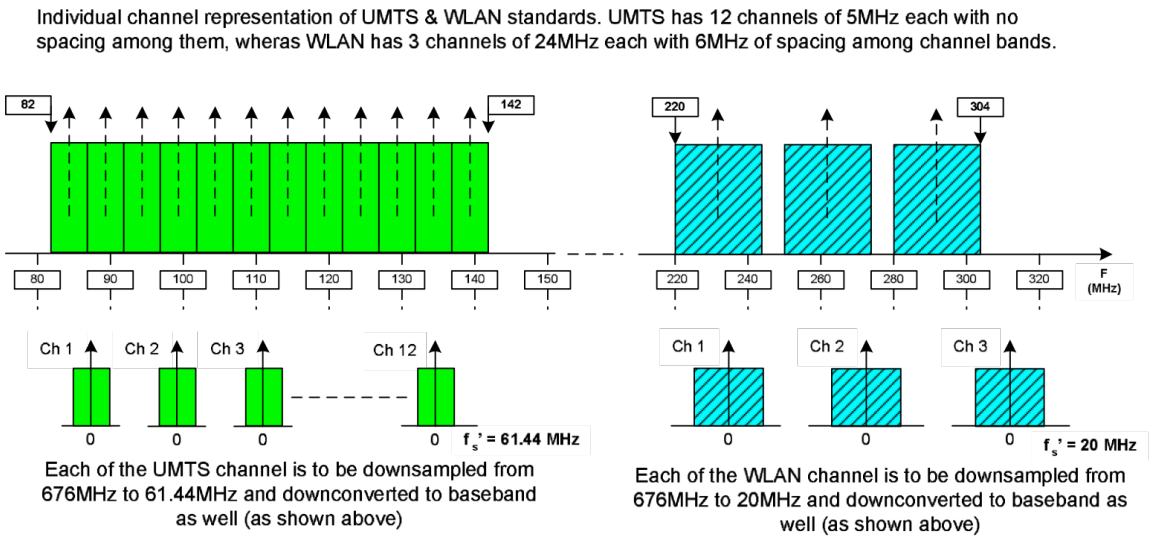


Figure 2.13: *12 UMTS channels of 5 MHz, down sampled at 61.44 MHz and downconverted to baseband. 3 WLAN channels of 24 MHz, down sampled at 20 MHz and downconverted to baseband as well [12].*

A summary of the specifications for the UMTS and WLAN is shown in Table [2.14]:

Standards	UMTS	WLAN
Sampling rate after AD Conversion	676 MHz	676 MHz
Sampling rate desired to separate the channels	61.44 MHz	20 MHz

Table 2.14: *Specifications for UMTS and WLAN sample rates.*

In order to downconvert and downsample the respective channels for UMTS and WLAN, we need to design the channelizer.

2.2 System Design

Now, polyphase channelizers described above are designed for UMTS and WLAN. This design covers the modifications to obtain the desired target-sampling rate respecting the different specifications to design the channelizers.

Until now, Figure [2.13] has shown the UMTS and WLAN channels, as well as the downsampling and down-conversion to baseband inside the polyphase channelizer. To remind, for UMTS, the Nyquist-zone lie at (82-142) MHz, with channels centered at 84.5, 89.5, … and 139.5 MHz whereas, for WLAN, this zone lie at (220-304) MHz, with 3 channels centered at 232, 262 and 292 MHz.

According to [13], the relation between the sampling frequency, the transform size (number of channels) and the channel spacing for the polyphase channelizer is (2.20):

$$f_s = N \times \Delta f \tag{2.20}$$

At 676MHz of sampling frequency, the number of channels for UMTS (WLAN) for a channel spacing of 5 (30) MHz is equal to 135.2 (22.53). But, always according to [13], there are two requirements that have to be met:

- The transform size (number of channels) must be integer.
- The channels that are downsampled and downconverted have to be centered on the multiples of the channel spacing.

For those two constraints, polyphase channelizer doesn't fit the UMTS and WLAN aliases as shown in Figure [2.21].

Channelization in these circumstances involves an offset after the downconversion operation to baseband as shown in Figure [2.22].

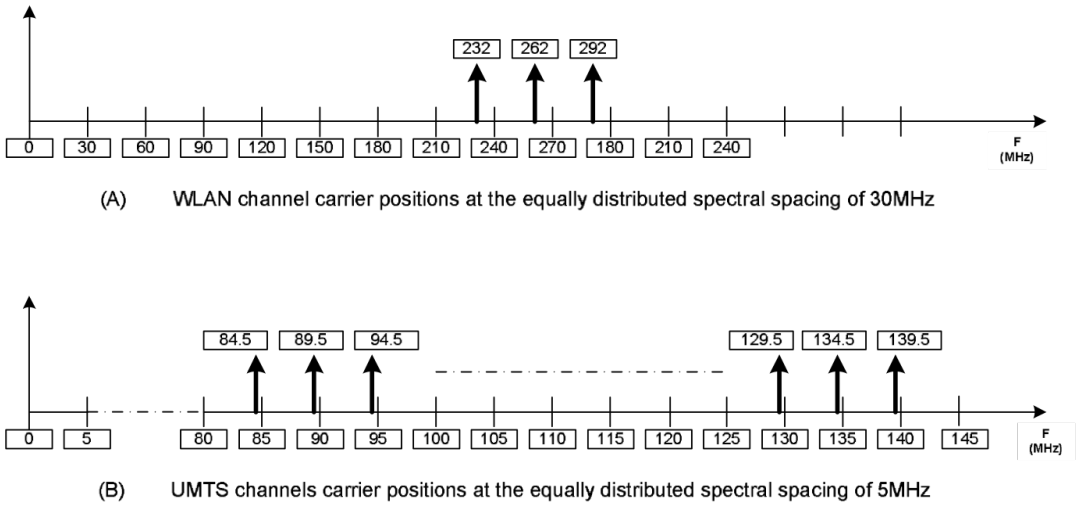


Figure 2.21: *3 WLAN channels carriers (A) with 30 MHz spacing. 12 UMTS channels carriers (B) with 5 MHz spacing. This representation result of the downsampling operation at 676 MHz [12]*

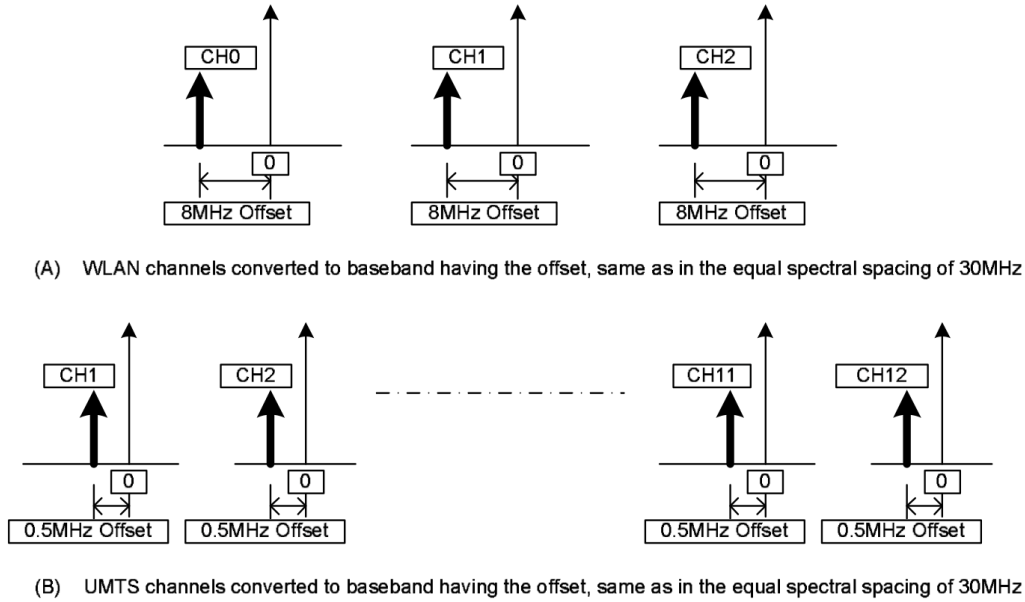


Figure 2.22: *3 WLAN channels downconverted at 20 MHz (A) with 8 MHz offset (fs =20 MHz). 12 UMTS channels down-converted (B) with 0.5 MHz offset (fs = 61.44 MHz)[12].*

There are many possibilities to try to correct this offset. The first possibility is the change of the sampling frequency. The choice of the sampling frequency must be done such that the aliases of UMTS and WLAN channels satisfy the required demands. Furthermore, the frequency must not allow the overlap of the required aliases. Frequencies lower than 676 MHz are chosen but none of them satisfied the specifications described above.

The second possibility is to correct the offset by heterodyning after the polyphase channelizer. It consists of multiplying the outputs of the channelizer by an oscillating and low-pass filtering them. It means this method uses an extra mixer and filter for each channel, requiring more hardware resources.

The best solution is to use the heterodyning method, but inside the polyphase structure. The equation of this structure is seen in (A.11). Then, this equation is developed. The term $2\pi nk$ is congruent to 2π, and the selected frequency k aliased to zero in the polyphase partition. k can be replaced by $k+\frac{s}{d}$ where s=0,1,2…d-1. With d=4, the equation becomes:

$$H(Z)=\sum_{r=0}^{M-1}Z^{-r}e^{j(\frac{2\pi}{M})rk}H_r(Z) \tag{2.24}$$

$$H(Z)=\sum_{r=0}^{M-1}\sum_{n=0}^{(N/M)-1}Z^{-(r+nM)}e^{j(\frac{2\pi}{M})(r+nM)k}h(r+nM)$$

$$H(Z)=\sum_{r=0}^{M-1}e^{j\frac{2\pi}{M}rk}Z^{-r}\sum_{n=0}^{(N/M)-1}Z^{-nM}e^{j(\frac{2\pi}{M})(nM)k}h(r+nM) \tag{2.23}$$

$$H(Z)=\sum_{r=0}^{M-1}e^{j\frac{2\pi}{M}r(k+\frac{s}{4})}Z^{-r}\sum_{n=0}^{(N/M)-1}Z^{-nM}e^{j(\frac{2\pi}{M})(nM)(k+\frac{s}{4})}h(r+nM)$$

$$H(Z)=\sum_{r=0}^{M-1}e^{j\frac{2\pi}{M}r(k+\frac{s}{4})}Z^{-r}\sum_{n=0}^{(N/M)-1}Z^{-nM}e^{j(\frac{2\pi}{4})ns}h(r+nM)$$

The summation representing the polyphase always has a phase shift that varies with time index n. The fact to take d=4 allows to compensate the offset during the channelization when the channels are centered on the multiple to the quarter of the channel spacing. This offset is also embedded in the phase rotators of each polyphase channel (first summation in 2.23). The Figure [2.24] shows the modified structure of the polyphase channelizer.

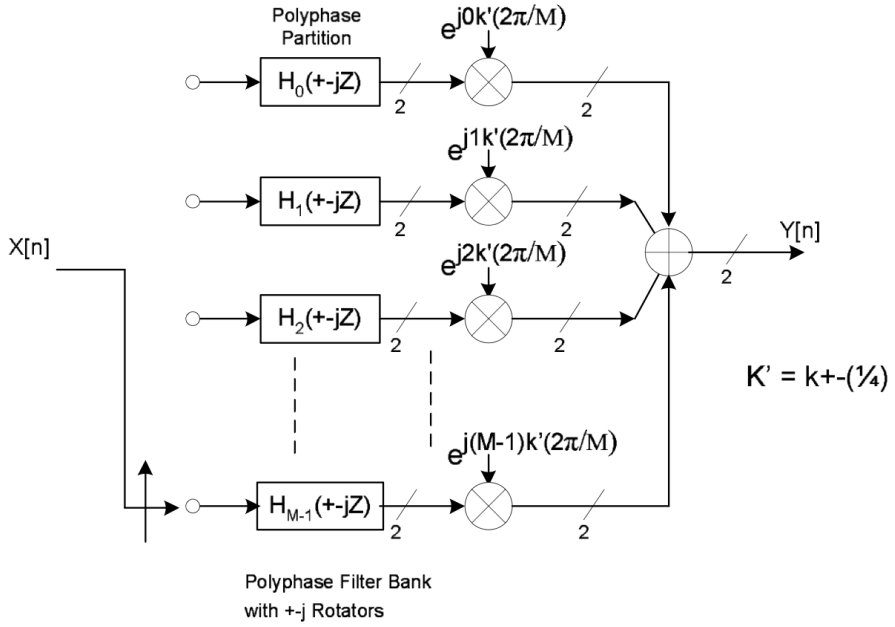


Figure 2.24: *The modified structure of the polyphase channelizer to compensate the offset after the down sampling and down-conversion operation. [12]*

This modified structure is efficient. However, it works only for the offsets of the multiple of quarter of the channel spacing. When the number of channels is not an integer or if the offset is not the quarter sub-multiple of channel spacing, the offset is still present on the output of the channelizer. Another solution has to be found to compensate the offset in these cases.

The solution to compensate this offset is to change the sampling frequency to fit the specifications of the polyphase channelizer. After different tries, a sampling frequency of 840MHz has been chosen. The new spectrum of UMTS and WLAN is shown in Figure [2.25].

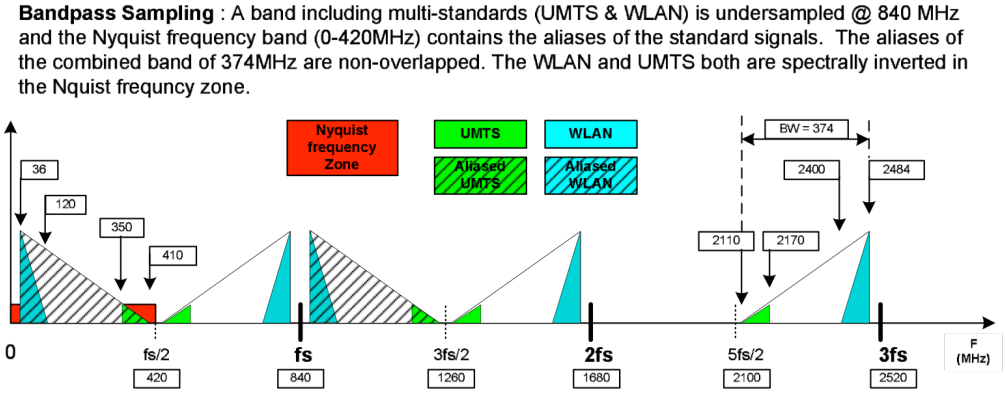


Figure 2.25: *Combined spectrum of UMTS and WLAN sampled at 840MHz. [12]*

For UMTS, the channel spacing still is 5MHz and there are 168 channels at this sampling frequency. For the case of WLAN, the number of channels is 35 for a channel spacing of 24MHz. The Figure [2.26] shows the carriers positions of the channels for UMTS and WLAN.

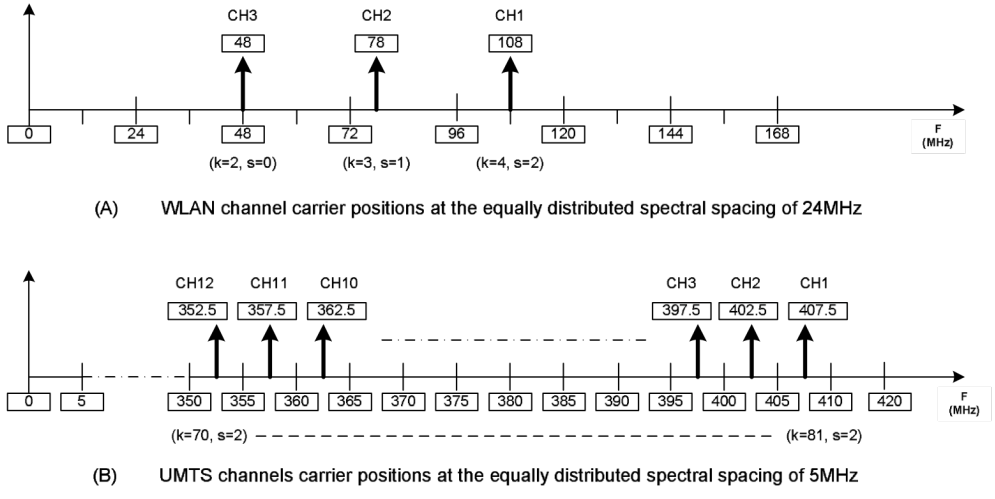


Figure 2.26: *Representation of WLAN and UMTS channel carriers. WLAN channels have 30MHz spacing and are centered on multiple of 24MHZ, where the second and third carriers require an extra offset (one and two quarter of 24MHz) for the position of the carriers. For the UMTS carriers, they are centred on multiple of 5MHz, which is added two quarter of 5MHz[12]*

The first representation shows, for WLAN case, that the carriers are centered at position characterized by (k=2, 3, 4 and s=0, 1, 2). It means that for each channel, the parameter has to change. Thus, only one channel can be extract at one time. For UMTS (second graph), the carriers are centered at k=70, 71, 72, …, 81 and the parameter s doesn't change (s=2). So, the polyphase channelizer can extract all the channels at the same time.

To finish the system design of the polyphase channelizer, the sampling rate has to be changed on the output. The output-sampling rate for WLAN (UMTS) is 20MHz (61.44MHz). But an observation can be done before; the number of channels for WLAN (UMTS) is 35 (168), which require high clock speed and large memory to store all the coefficients during the filtering process. Furthermore, only twelve channels are used and extracted at one time in the case of UMTS, whereas for WLAN, three channels are used and one extracted at one time. The purpose is to reduce the sampling frequency as low as possible on the input of the polyphase channelizer, to not have overlap aliases and still meet the specifications of the channelizer. After different resampling factors have been tried, the choice has been done and WLAN is resampled by a factor 7 whereas UMTS is resampled by a factor 8. It results that the input sampling frequency of the channelizer is 120MHz (105MHz) for WLAN (UMTS). The Figure [2.27] shows this modification. Moreover, the number of channels in the polyphase channelizer goes from 35 to 5 for WLAN and from 168 to 21 for UMTS, which reduce the requirements for the implementation.

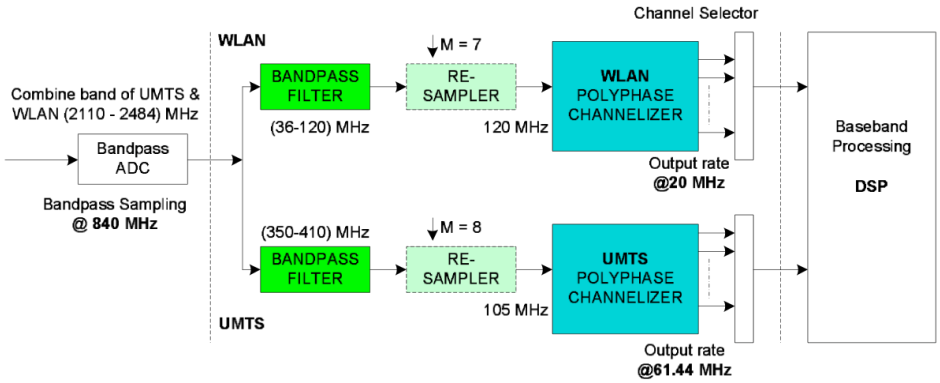


Figure 2.27: *block diagram after modification. WLAN and UMTS path are resampled by M=7 and M=8 before polyphase process. The input sampling frequencies for the polyphase channelizer are now 120MHz and 105MHz for WLAN, respectively UMTS. The number of channels is 5 (21) for WLAN channelizer (UMTS channelizer) [12]*

Now the signal has to be downsampled to the output rate (20MHz for WLAN and 61.44MHz for UMTS). It becomes obvious for WLAN path. Having an input sampling frequency of 120MHz, the signal is downsampled by a factor 6. For the case of UMTS, the factor to downsample from 105MHz to 61.44MHz is 1.7. It means the signal has to be first upsampled by 10 and then downsampled by 17. All these operations are realised by the input commutator.

The system design is now completed. The input signal of the block is sampled at 840MHz. Then, this signal is filtered to separate the WLAN and UMTS spectrum. An operation of resampling is done in order to reduce the number of channels in the polyphase channelizer. The sub-filters inside the channelizer select the good channels, and a down-conversion to baseband is carried out.

2.3 Technical problem

The bandpass sampling after the LNA (Figure [1.21]) allows passing from the RF to the IF domain. The input signal is sampled at 840MHz. Output signal of the bandpass sampling is then process through a bandpass filter to select the band composed of WLAN and UMTS signal. After, the path is divided into two paths (one for WLAN and one for UMTS) and data are processed into the channelizer. Before the channelizer, data are resampled at 120MHz (WLAN) and 105MHz (UMTS). There are resampled in order to reduce the number of channels inside the channelizer. These frequencies have been chosen because there is no overlapping during this operation of resampling as seen in Appendix [A]. The clock frequency of the DSP platform is 225MHz (Appendix [B]). The problem is that the DSP cannot manage to reduce the sampling rate from 840 MHz to 120 or 105MHz due to the low clock frequency. The changing of sample rate has to be done before the tasks perform on the DSP platform. The Figure [2.30] shows the general system.

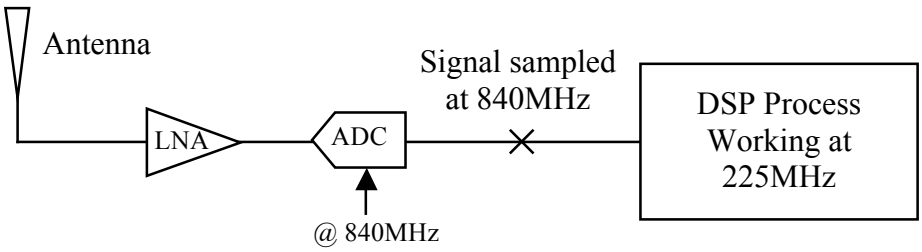
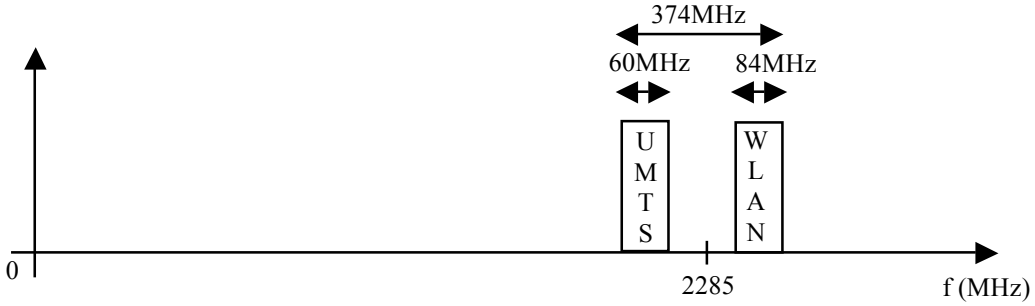


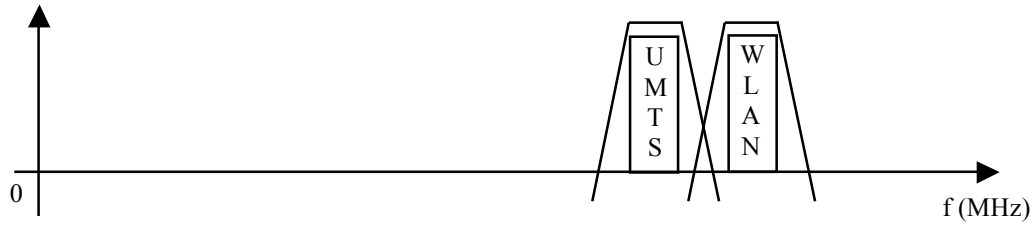
Figure 2.30: *General system of the application. The signal is received at the Antenna, passed through the LNA and bandpass sampled at 840MHz. Then, the signal is filtered and downsampled for the standards (UMTS and WLAN) before the DSP process. In order to downsampling an input frequency at 840MHz, operations have to be done before the platform.*

It has been found several solutions to perform this operation: all the operations are in the analog domain between the LNA and the ADC.

The first solution consists in passing the input signal received at the Antenna through a mixer to bring the signal to a lower possible frequency. The signal, after the LNA, is filtered in two bandpass filters to select the spectrums (UMTS and WLAN). Then, heterodyne multiplies the selected band to bring it to the baseband. Finally, the Nyquist sampling technique is used. The result is sampled at $2*f_{max}$ to respect the Nyquist sampling theorem [11]. Figure [2.31] shows the spectral representation of all these steps while this operation.

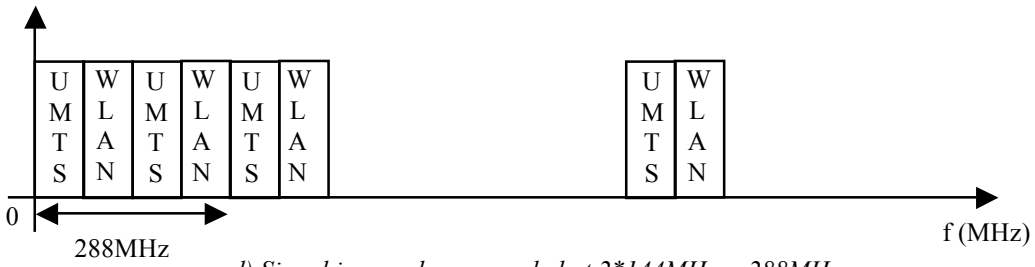


a) Spectral representation of input signal with UMTS and WLAN standards.



b) Input signal is filtered by means of bandpass filter

c) Input signal is brought to baseband by means of heterodyning. The spectrum occupation is now 144MHz.



*d) Signal is now down sampled at 2*144MHz = 288MHz.*

Figure 2.31: *Spectral representation of different steps of the operation to reduce the sampling frequency.*

The Figure [2.32] shows the diagram of this operation. After these operations, the sampling frequency is 288MHz. The problem is therefore not resolved.

Another solution is possible. Instead of filtering all the spectrum of standard, only one channel is selected among 3 channels for WLAN and 12 channels for UMTS. The idea is exactly the same as the previous solution but bandpass filter

specifications change. Indeed, it consists in changing value of passband. In the first solution, the passband for WLAN bandpass filter was 60MHz. WLAN is composed of three channels. The bandwidth of each channel is 24MHz. Therefore, the passband is 24MHz now. In the UMTS case, 12 channels of 5MHz-bandwidth compose the spectrum. Instead of a passband of 84MHz, the new passband is 5MHz. By filtering only one channel for each standard, the bandwidth of combined channels is 29MHz at baseband after the heterodyne operation. The sampling frequency becomes 58MHz. Therefore, operations in the digital domain are performed on DSP.

But there is one problem: the constraints on the analog filters are strong, especially for UMTS case. Indeed, filter has to process around 2GHz with a 5MHz passband and very strong band-edges to select only one channel. This filter is realizable, but the signal is deformed. Thus, this solution is also discarded.

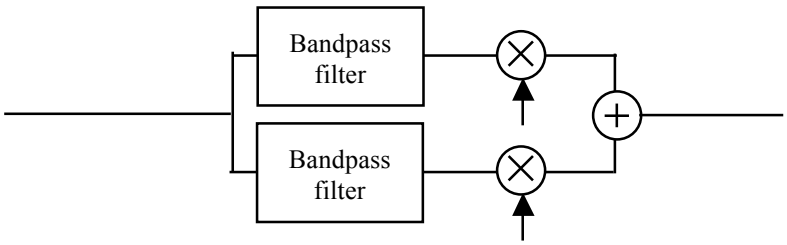


Figure 2.32: *Structure of one solution to reduce the input sampling frequency in order to process all the operations after the ADC on DSP clocked at 225MHz.*

In conclusion, it has been that UMTS and WLAN standards are sampled at 840MHz after reception at the Antenna. WLAN is consisted of 3 channels. The bandwidth of each channel is 24MHz. For UMTS case, it is composed of 12 channels and the bandwidth is 5MHz. They are processed through front-end and polyphase channelizer. The output sample rate is 20MHz (61.44MHz) for WLAN (UMTS). It is decided to perform the front-end on FPGA due to low clock frequency of DSP and therefore, to focus only on the polyphase channelizer for the rest of the project.

3 Design Methodology

3.1 Overview

The purpose of this chapter is to show the different methodologies that are used in the project work. First of all, the A^3 Model is used and specifies the domains and the connections between them. Next, another methodology, the Rugby Meta-Model is presented. A brief conclusion is done to choose the appropriate model.

3.2 The A^3 Model

The design of the A^3-model [6] is divided in three parts: Application, Algorithm and Architecture. First of all, "generic" A^3-model is shown in Figure [3.20]. Then, this model is applied to the project presented in this report, as shown in Figure [3.21].

- Application: is a description of the system with specifications and constraints. It can be time, power, area problems…

- Algorithm: is the mathematical description of the application. It can be existing algorithms or new algorithms. They are optimized on a purely mathematical point of view, i.e. the optimizations are done on the algorithm's parts directly related to the application.

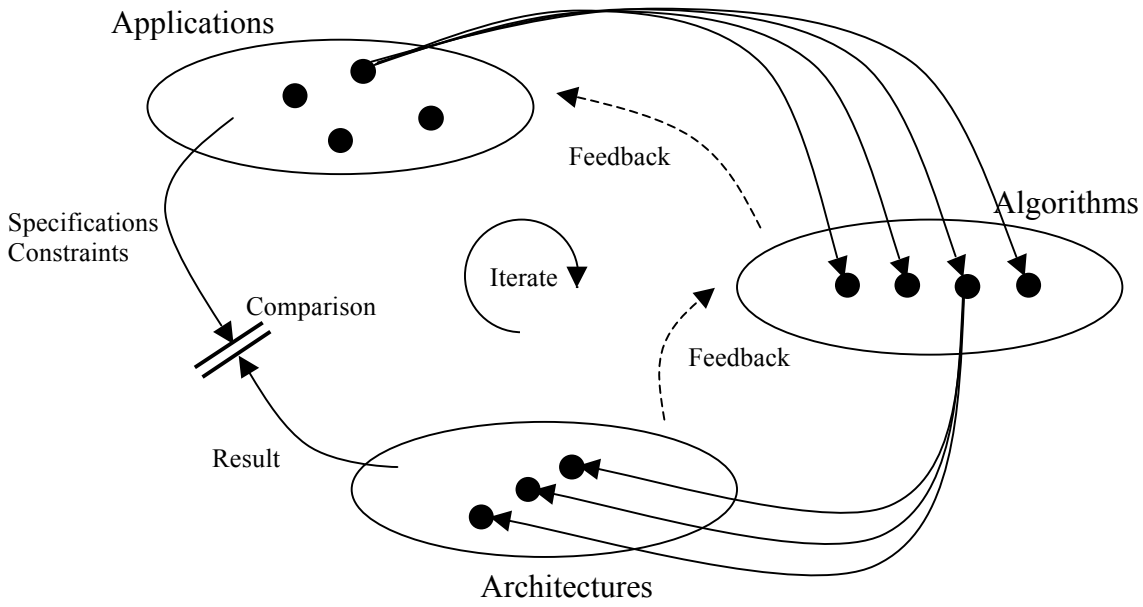


Figure 3.20: *The"generic" A^3 design methodology*

- Architecture: is the platform where the algorithms are mapped (DSP, FPGA, Cell-BE…). The results and the specifications/constraints of the application are compared and modifications are done in case of incompatibility.

In the application domain, a presentation of SDR front-end for SATCOM handheld Receiver in section [2] is done.

One algorithm is developed. First, an algorithmic survey is done based on Multi-rate DSP methodology. Then, an algorithm exploration is analysed by means of MatLab. Finally, polyphase channelizer algorithm is coded in C language.

In the architecture domain, the platform used to implement the algorithms is analysed. Available hardware and system limitations are studied. Then, measurements in terms of resource utilisation, execution speed, etc are realized.

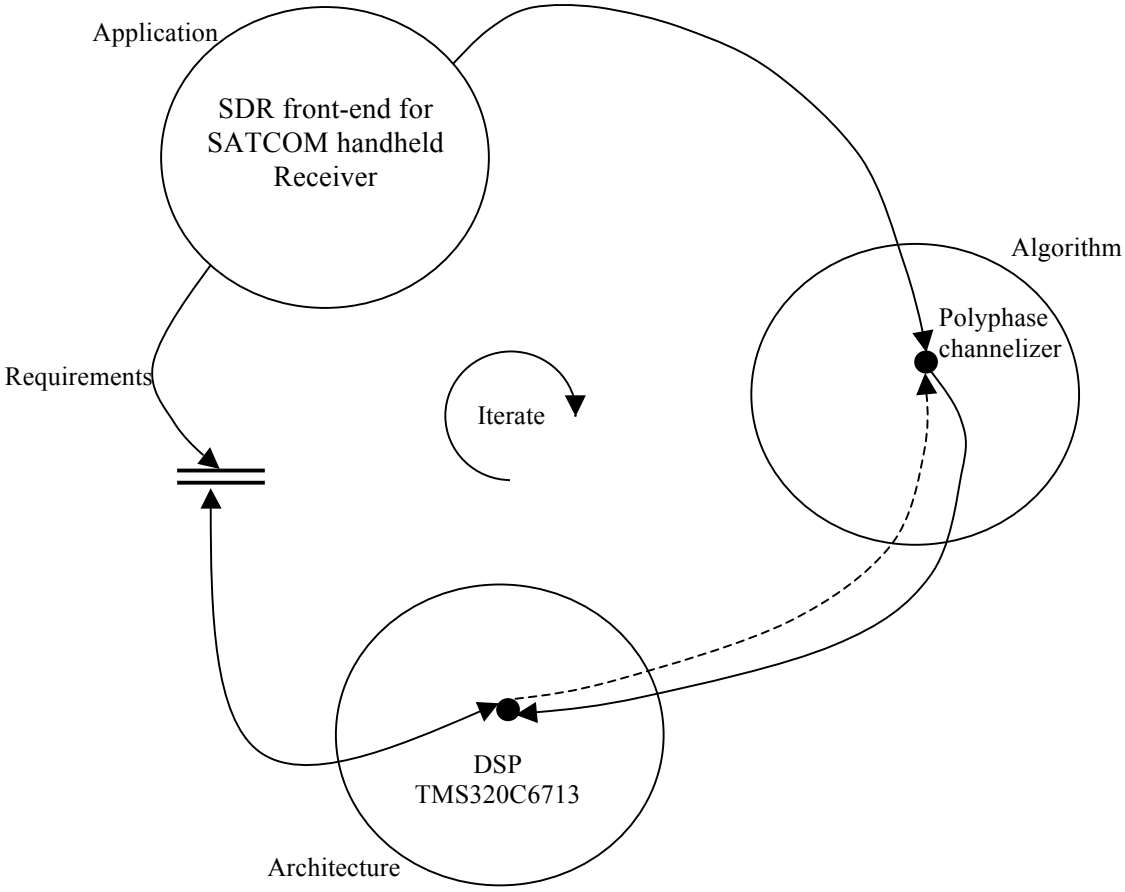


Figure 3.21: *A^3 model for project*

3.3 The Rugby Meta-Model

The Rugby Meta-Model Methodology [7] is explained in this section. This model is based on the Y Chart [7], introduced in 1983 by Gajski and Kuhn but with some

modifications due to the increase of the complex systems requiring concurrent processes (many activities on the same device). The rugby Meta-Model is composed of four domains: Time, Computation, Communication and Data with different abstraction levels for each domain as shown in Figure [3.30].

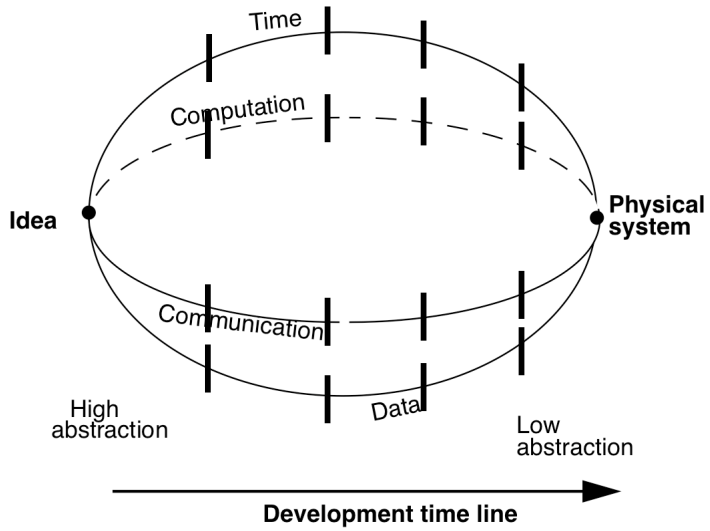


Figure 3.30: *The general Rugby Meta-Model, composed by the four domains Time, Computation, Communication and Data. The development time line proceeds from the left to the right whereas the abstraction levels (represented by the vertical lines) go from a high to low. The designer starts from an idea to arrive to a physical system.*

- Time: is the domain concerned with the time relations between activities. At each levels of abstraction, the timing in the architecture is specified. For example, at the highest level, it is sufficient to consider the causality, whereas at the low level, delays appear on every port and signals and are essential.

- Computation: it concerns the relation between input and output values. This domain describes the behaviour of the components at different levels of abstraction. For example, at high level, these components could be transmitter or receiver; on the other hand, at low level, they could be logic blocks or instruction set.

- Communication: treats the connections between design elements. For instance, during the first steps of the development, it describes the protocols of communication between the functional blocks. At the low level software, it deals with the interaction between the storage and the computation part of a processor. For the hardware branch, the communication is concerned by the connections between logical ports.

- Data: is the domain that informs the data types at every abstraction level. That could be real numbers like voltage for low level, Boolean or logic before this one or integer, real for high one.

Now, the Rugby Meta-Model is applied on the project, as presented in Figure [3.31]. Different abstraction levels are added according to the requirements of the project.

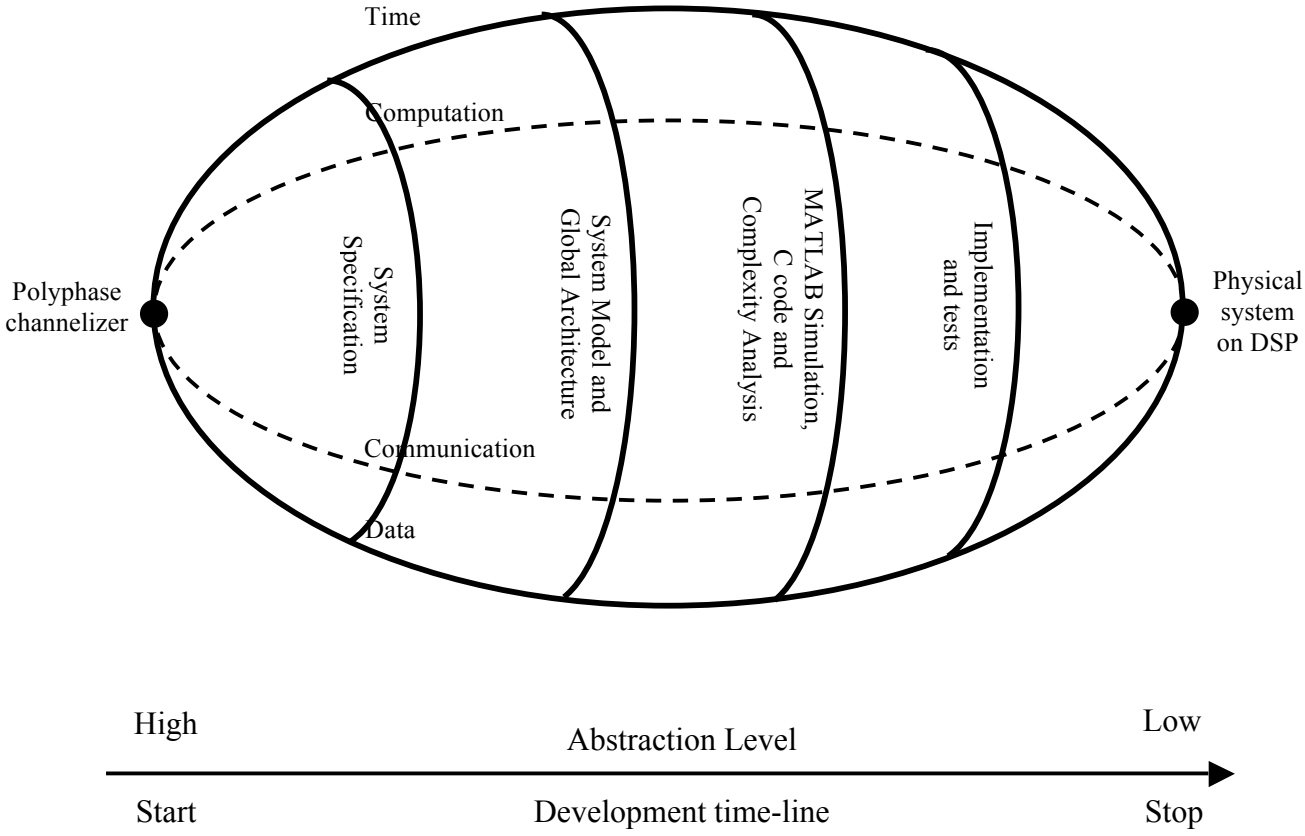


Figure 3.31: *The Rugby Meta-Model applied on this project, with domains and abstraction levels*

In the meta-model of this SatCom Handheld Receiver, the different abstraction levels are defined as following:

- The *System Specification* corresponds to the constraints and requirements for a multi standard receiver, which both UMTS and WLAN standards receive simultaneously on the same front-end.
- The *System Model and Global Architecture*, where a survey of the given platform (DSP) is done as well as the application of the project (polyphase channelizer)
- The *Algorithm analysis*, with a mathematical description and simulations on MatLab (floating point), is analysed and developed. Developing and debugging of the C code to prepare the implementation.
- The *Implementation and tests* consist to download the program on the platform and carry out tests to satisfy the system specification.

In conclusion, the *A^3 model* is chosen for this project. Indeed, this methodology is more general than Rugby Meta-Model that is applied on one specific application. Moreover, *A^3 model* corresponds to our project. The project flow development follows this model. Application is studied, algorithm is simulated and implemented on the architecture. After comparison with the constraints, optimizations are developed in order to satisfy the specifications of the application.

4 Algorithm Analysis

4.1 Overview

In this chapter, the analysis of filtering calculations is discussed to implement them later on the platform. First of all, different design filter methods are presented. Then, the chosen filter is applied to the polyphase channelizer and simulations are carried out. Finally, complexity analysis is studied. A short conclusion is done about the results of the different simulation on MatLab. According to the A^3 design model, this section belongs to the algorithm domain, as illustrated in Figure [4.10].

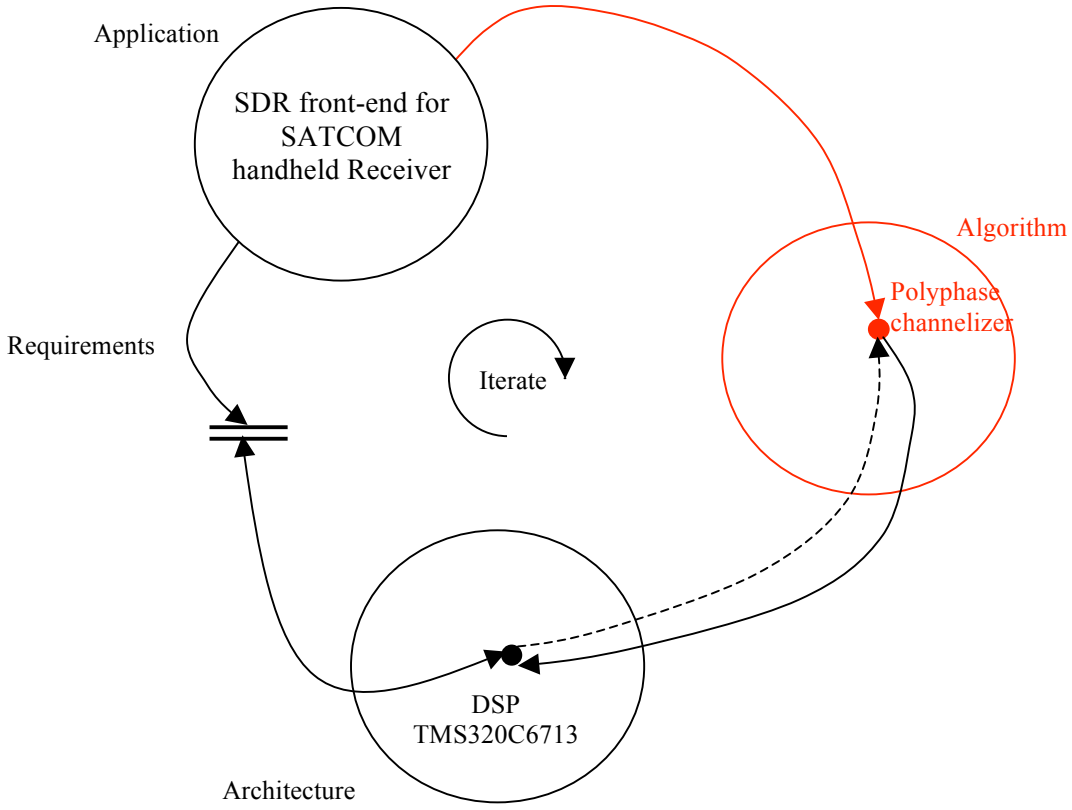


Figure 4.10: *A^3 model for project. Highlighted in red, the analysis of the algorithm*

4.2 Algorithms

4.2.1 Signal Theory

A digital filter is a system that performs operations on sampled discrete signal to modify its characteristics. There exist two classes of digital filters: Infinite Impulse Response (IIR) and Finite Impulse Response (FIR) filter. The type of filter is chosen according to several criteria: complexity, computational speed, and required

resources. The IIR filter is supposed to be unstable and is difficult to control in terms of phase [8]. But as mentioned in Appendix [A], the application requires least phase distortion; therefore FIR filter is the best choice, especially by means of its linear phase and its stability. Moreover, this type of filter is supported by purpose DSP that have Multiplier and Accumulator (MAC) that reduce the computational speed of higher order FIR filters.

The filter has to be design, i.e. finding the coefficients from frequency specifications. There are three methods (the most commonly used) to design it:

- Window design method
- Frequency sampling method
- Equiripple design

In this section, the study is carried out only on the window and the equiripple design method. The "window design method" is the simplest method to design a FIR filter because it's very easy to use and understand. The process to use it is described below:

1) Specify the desired frequency response
2) Calculate the IFFT which give lots of coefficients
3) Truncate the filter coefficients
4) Apply a window function to sharpen up the filter's frequency response

But this method has some limitations. Indeed, the truncation of filter coefficients introduces some ripples and overshoots called Gibb's phenomenon. Another problem is that the stopband attenuation is fixed for a given window. Thus, for a given attenuation specification, the window has to fit perfectly the filter's specifications. Moreover, this method is not flexible. This method is not optimal, hence another method called Equiripple.

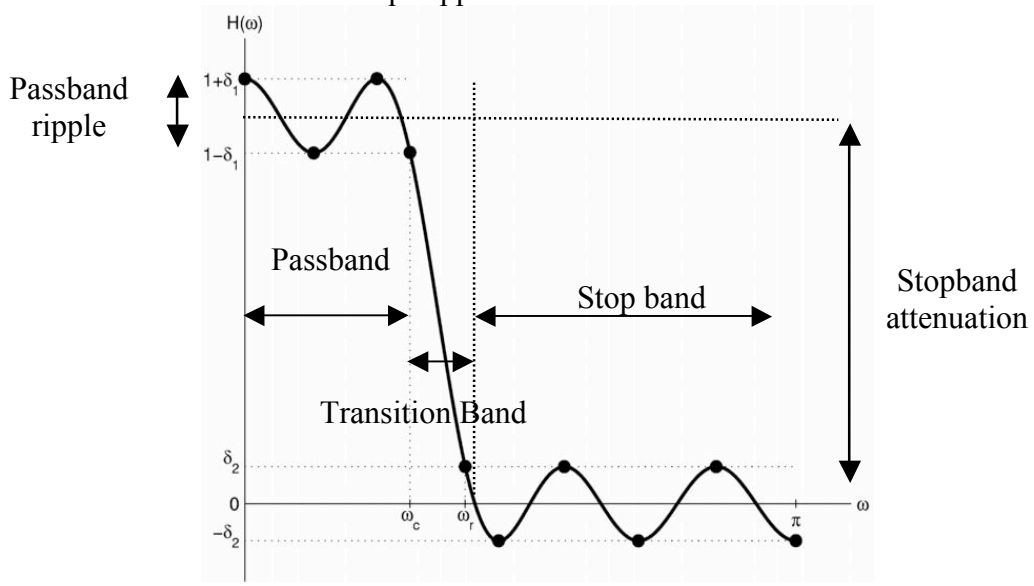


Figure 4.2.1: *Frequency response of a low pass filter used the equiripple method [9]*

This method, created by Remez/Parks-Mclellan, uses an algorithm that iterates between the filter coefficients and the frequency response until it finds the filter that fits with the given specifications and with the lowest number of coefficients. This method just meets the specifications without over performing. On the other hand, many window methods design filter better than the specifications, hence wasting the performance. An example of the frequency response of a low-pass filter using the equiripple method is shown in Figure [4.2.1]

By means of this method, the optimal filter is defined by the following specifications; sampling frequency, transition width (ω_c,ω_r), passband ripples (δ_1) and stopband attenuation (δ_2). Then, the order of the filter is computed with the formula (4.2.2) and presented in [10].

$$N \approx \frac{C_\infty(\delta_1,\delta_2)}{\Delta F} + g(\delta_1,\delta_2)\Delta F + 1 \qquad (4.2.2)$$

Where

$$C_\infty(\delta_1,\delta_2) = \log_{10}\delta_2\left[b_1\left(\log_{10}\delta_1\right)^2 + b_2\log_{10}\delta_1 + b_3\right] + \left[b_4\left(\log_{10}\delta_1\right)^2 + b_5\log_{10}\delta_1 + b_6\right]$$

$$g(\delta_1,\delta_2) = -14.6\log_{10}\left(\frac{\delta_1}{\delta_2}\right) - 16.9$$

$$b_1 = 0.01201; \quad b_2 = 0.09664 \quad b_3 = -0.51325$$
$$b_4 = 0.00203 \quad b_5 = -0.5705 \quad b_6 = -0.44314$$

And ΔF is the transition width normalized to the sampling frequency. To finish, the ratio from the passband ripples to stopband ripples gives the weight for each band.

To summarize, the choice of the design method must be done carefully. The different specifications of the filter impose us to design it with the optimal method, to obtain the minimum numbers of coefficients (therefore less resources to implement) and to fit perfectly to the requirements.

4.2.2 Simulation

In this section, the equiripple design method has been chosen for the polyphase channelizer simulations. These simulations are carried out by means of MatLab. For each standard (WLAN and UMTS), the prototype filter is shown as well as one channel on the output of the polyphase channelizer

The filter specifications (developed in the previous chapter) have shown that the channel spectral distribution for WLAN and UMTS are 24 MHz, respectively 5 MHz. For WLAN, the passband bandwidth is 16.6 MHz and the transition bandwidth is 7.4 MHz. For the case of UMTS, the passband bandwidth is 3.84 MHz whereas the

transition bandwidth is 1.16 MHz. The Figure [4.2.3] and [4.2.4] show these filter specifications.

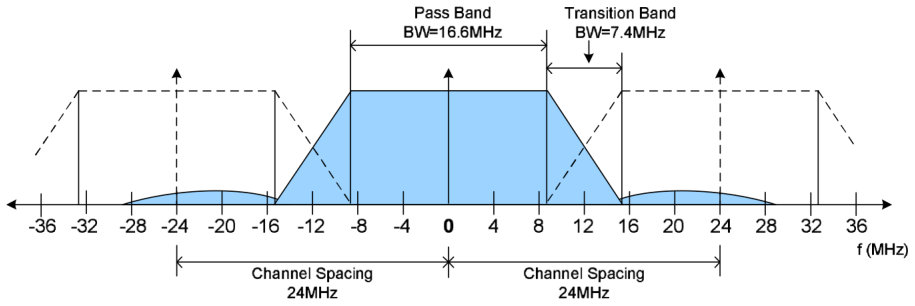


Figure 4.2.3: *Filter specifications for WLAN channelizer. The passband is 16.6 MHz and the transition band is 7.4 MHz. The channel spacing is 24 MHz [12]*

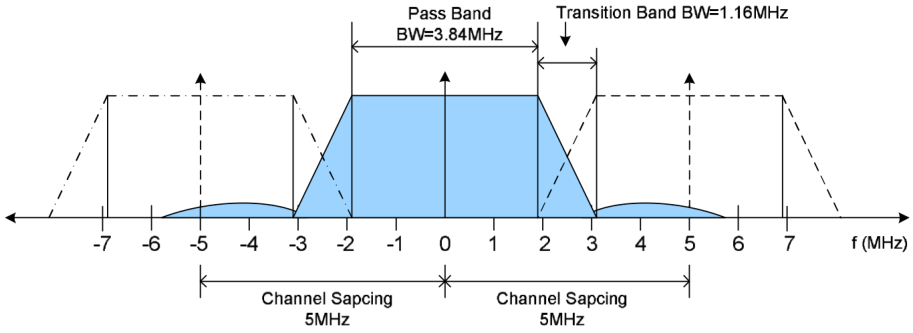


Figure 4.2.4: *Filter specifications for UMTS channelizer. The passband is 3.84 MHz and the transition band is 1.16 MHz. The channel spacing is 5 MHz [12]*

First, the order of the filter has to be defined according to the specifications (cutoff frequencies, attenuation in the stopband). The MatLab function *firpmord* allows obtaining the approximate order of the filter. This function needs four parameters to determine the value; the vector of the frequencies band edges between 0 and Fs/2 (Fs is the sampling frequency), the vector that defines the amplitude in the different bands, the vector specifying the desired attenuation in the stopband and finally the sampling frequency.

With the order of the filter, the coefficients can be computed now. The function *Firls* is used. This function returns a row vector of N+1 coefficients of the n FIR filter whose approximately match the characteristics of the filter according to frequency and amplitude vector specified above in the description of *firpmord* function. The output coefficients are real and symmetric.

The frequency response of the filter prototype is shown in Figure [4.2.5] for the WLAN channelizer and in Figure [4.2.6] for the UMTS channelizer. Frequency response of WLAN in Figure [4.2.5] and UMTS in Figure [4.2.6] respect specifications of the filters described previously (Figure [4.2.3] for WLAN and Figure [4.2.4] for UMTS). Filters are designed at baseband.

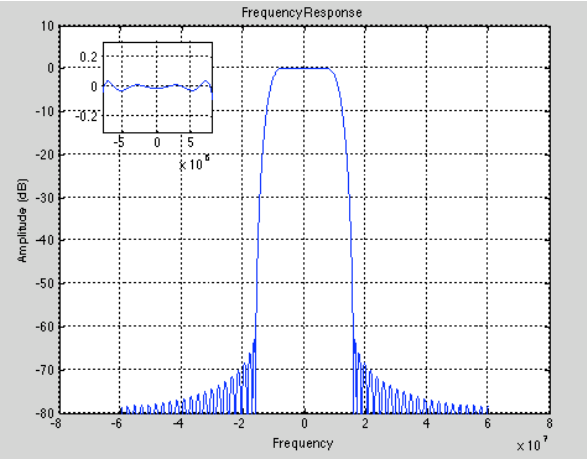


Figure 4.2.5: *Frequency response of the prototype filter for WLAN channelizer. The passband is 16.6 MHz with ripple less than 0.5 dB whereas the transition band is 7.4 MHz. There is an attenuation of 60 dB in the stopband.*

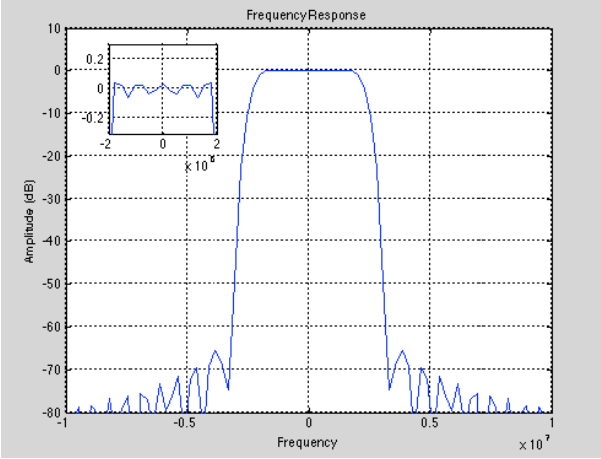


Figure 4.2.6: *Frequency response of the prototype filter for UMTS channelizer. The passband is 3.84 MHz with ripple less than 0.5 dB whereas the transition band is 1.16 MHz. There is an attenuation of 60 dB in the stopband.*

For the WLAN filter, the passband is 16.6MHz with a ripple less than 0.5dB. The transition-band is 7.4MHz and the stopband attenuation is 60dB. The MatLab function *firpmod* gives a length of 51 for non-partitioned WLAN filter. For the polyphase decomposition, the number of channels is 5. In order to have an integer number of coefficients for the sub filters inside the polyphase channelizer, the order of the filter is decreased to 50. Therefore, the length of sub filter is 10 taps.

In the case of UMTS, the passband is 3.84MHz with a ripple less than 0.5dB. The transition-band is 1.16MHz and the stopband attenuation is 60dB. The filter is designed at 1050MHz. The length of the non-partitioned filter is 2521. The number of channels for the polyphase decomposition is 21. Like the WLAN applications before, the order of the non-partitioned is modified in order to have an integer number of coefficients. The new order is 2520. The length of each sub filters in the filter bank is 120 taps. But in the UMTS polyphase channelizer, a down-conversion of 1.7 or 17/10 is required (seen in section [2])(upsampling by 10 and downsampling by 17). This upsampling creates 10 copies of the signal. Thus, the number of taps is divided by 10. The length of the sub filters is 12 taps.

Figure [4.2.7] and [4.2.8] show the test-signal generated for WLAN and UMTS standards. Indeed, signals from the bandpass filters are composed of only one standard. For this simulation, it was necessary to show all the channels of both standards. Therefore, these signals have been created to represent the channel carriers at the desired sampling frequency. They have been built by adding several exponentials together.

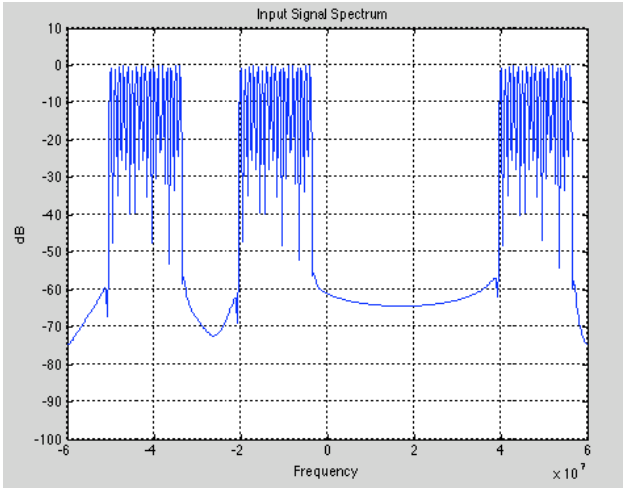


Figure 4.2.7: *Input signal for WLAN channelizer sampled at 120MHz. 3 channels are centered at -42, -12 and 48MHz occupying a bandwidth of 16.6MHz.*

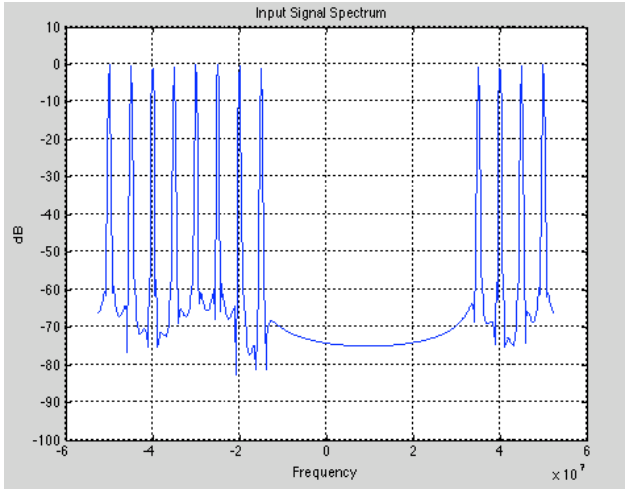


Figure 4.2.8: *Input signal for UMTS channelizer sampled at 105MHz. 12 channels are centered at 15, 20, 25...70MHz occupying a bandwidth of 3.84MHz.*

Data are fed into the polyphase channelizer by means of commutator. In the case of WLAN, data are dealt and shifted inside a two dimensional array (5 channels with 10 data samples each). Then, 6 data are fed at a time. The coefficient sets are stored in a two dimensional register (5 channels with 10 coefficients each). There are 5 different states to feed the data inside the input register. This channelizer is non-maximally decimated, i.e. output sampling rate and channel spacing are different. An offset appeared by feeding 6 data in 5 channels at a time. Therefore, the coefficients sets have to be rotated each time data are fed. The convolution is performed between data and coefficient sets. Finally, signal is downconverted to baseband and downsampled to the required sampling rate of 20MHz. The Figure [4.2.9] represents the output spectrum WLAN channel centered at -12MHz corresponding to variable $k = 4$ and $s = 2$.

The result is unexpected. Indeed, the output signal does not correspond of the input channel in Figure [4.2.7]. It is supposed that the downsampling operation into the polyphase channelizer causes data loss in the initial signal, hence this kind of result.

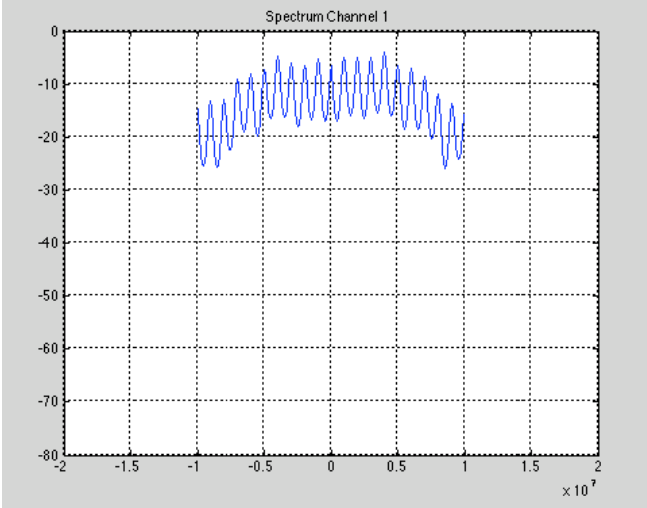


Figure 4.2.9: *output spectrum of WLAN channelizer. The channel is centered at -12MHz, corresponding to k=4 and s=2. The signal is downconverted to baseband and downsampled at 20MHz.*

For UMTS, the algorithm is a little bit different. Indeed, there is an operation of upsampling by 10 and an operation of down sampling by 17 to reduce the input sampling frequency at 105MHz to the output sampling frequency at 61.44MHz. Data are shifted and fed in a two dimensional register. Due to upsampling and downsampling, 9 zero packing are inserted between two data through the register in stride of length 17. Only 1 or 2 data are fed at a time. The same sequence of feeding repeats every 10 cycles; data are fed according to 10 states. Moreover, the filter has been designed at 1050MHz therefore one tenth of coefficients are used at a time. Then, convolution operation performs the filtering. The Signal is downconverted to baseband and downsampled to the output-sampling rate of 61.44MHz. In Figure [4.2.10], the output of one channel of UMTS channelizer is drawn. The channel centered at 35MHz, corresponds to k=7 and s=2.

It appears that the attenuation is not 60 dB in the stopband. It is supposed that the successive up/downsampling operations cause this kind of problem. Indeed, these operations result in data loss; therefore the signal energy decrease and that the reason why the attenuation is only 40 dB.

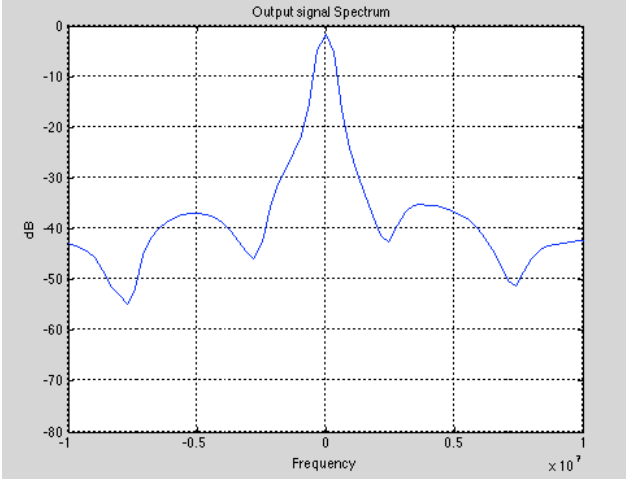


Figure 4.2.10: *the channel centered at 35MHz, which variable k=7 and s=2, corresponds to UMTS standard. The signal is downconverted to baseband and downsampled at 61.44MHz.*

To summarize, the order of the WLAN filter is 50. It is divided in 5 sub-filters. Therefore, the polyphase channelizer is divided in 5 channels. 6 data are fed at a time and the signal is downconverted to baseband by means of DFT. The output-sampling rate is 20MHz.

In the case of UMTS, the filter is composed of 2520 taps. It is divided in 210 sub-filters. 1 or 2 data are fed at a time. The channelizer is divided in 21 channels and the signal is downconverted to baseband. The output-sampling rate is 61.44MHz.

4.2.3 Complexity Analysis

In this section, a particular attention is done on the design of polyphase filter bank. Indeed, the analysis of the complexity must allow knowing how many times the functional units in DSP architecture are used in order to reduce the execution time for the final implementation. First of all, structures of FIR filters, used in the polyphase filter bank, are studied. Then, the structures of polyphase filter is described and optimized. Finally, the best design solution is selected for the implementation.

FIR Filters

FIR Filters use discrete convolution of the input and the frequency response of the filter. The formula [4.2.11] shows the discrete convolution with the input *x* and the filter coefficients *h*:

$$y(n) = \sum_{k=0}^{M-1} h(k)x(n-k) \qquad (4.2.11)$$

According to (4.2.11), the filter structure is designed in Figure [4.2.12]. This structure is most commonly called direct-form. It consists of parallel multipliers and accumulators (MACs). In this structure, each MAC computes the delayed input and the corresponding coefficient of the filter. All the results accumulated produce the output y(n).

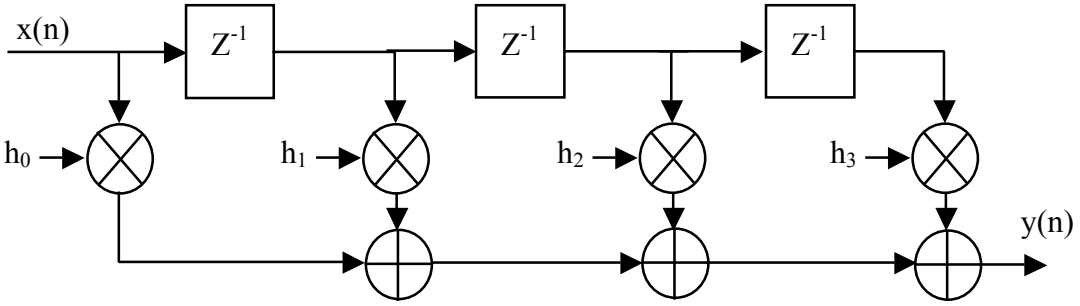


Figure 4.2.12: *Block diagram for direct-form FIR filter, with 3 taps length.*

However, this structure produces long delays through the accumulation way. It appears to implement another structure of FIR that correspond better to computational hardware. That is why the transpose-form FIR filter structure is chosen. The representation of this structure is drawn in Figure [4.2.13]. It consists to change the direction of the arrows of the direct form (Figure [4.2.12]), and exchange the input with the output.

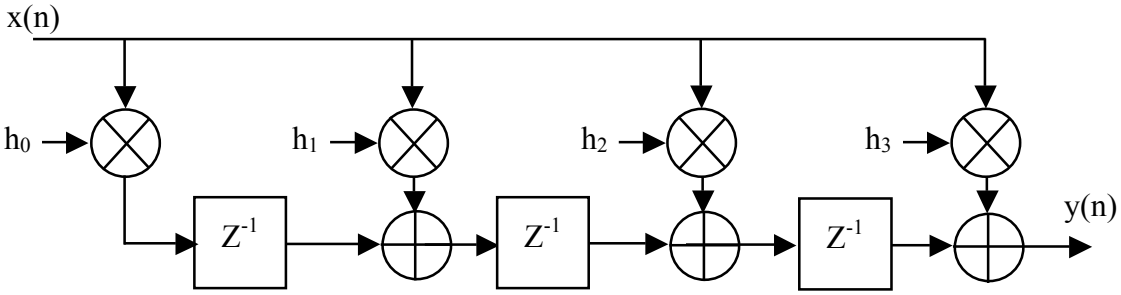


Figure 4.2.13: *Block diagram for transpose-form FIR filter, with 3 taps length.*

The most advantage of this structure is that it can have the accumulation way pipelined to increase the performance, especially in terms of execution time.

Polyphase Filter

The structure of polyphase channelizer used in this project is shown in Figure [4.2.14]. The 1-to-M commutator deals the sample data to M-Path Polyphase Filter Bank. Then, FFT block processes the down sampled data from the polyphase filters to construct the individual channels before the last summation.

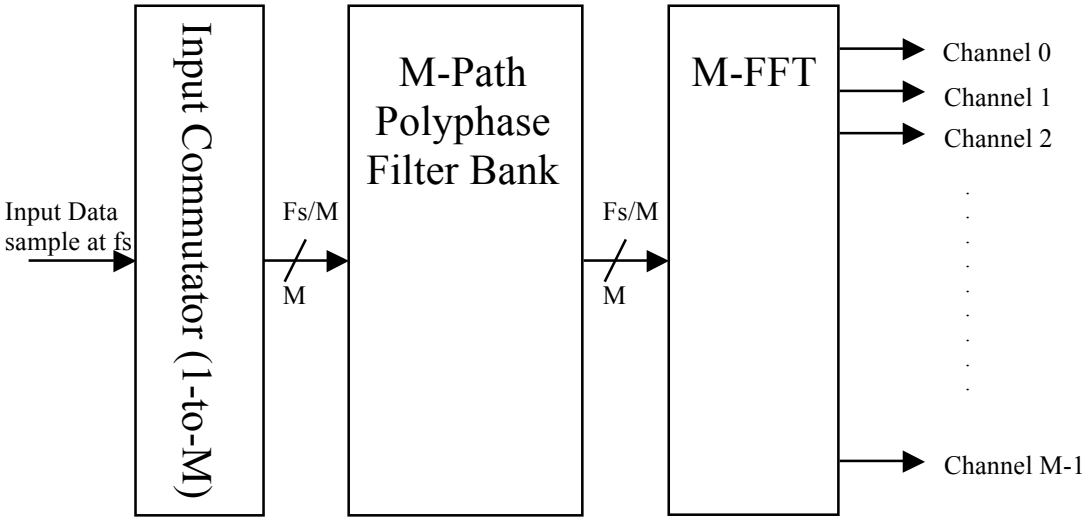


Figure 4.2.14: *The input commutator feeds the M-path Polyphase filter bank, operating at M times the reduced time than the input sample frequency*

The commutator deals the data to each sub-filter in the polyphase filter bank. Therefore, it loads data from the memory to registers. For the WLAN, according to the simulation in the previous section, the order of the non-partitioned filter is 50. The polyphase filter bank is composed of 5 channels (sub-filters). Each sub-filter has 10 coefficients. In the case of UMTS channelizer, the order of the non-partitioned filter is 2520. There are 210 sub-filters with a length of 12 coefficients. But an operation of upsampling (by 10) is performed inside the channelizer. It means that only $1/10^{th}$ of the coefficients are used at time. The UMTS channelizer is composed of 21 channels (sub-filters).

The complexity carried out on polyphase filters gives the number of multiplication, addition and register access for the final implementation. For the WLAN channelizer, $\frac{50}{5}$ multiplications, $\frac{50}{5}-1$ additions and $\frac{50\times 2+50}{5}$ register accesses are necessary for each sub-filter. For the case of UMTS, $\frac{252}{21}$multiplications, $\frac{252}{21}-1$ additions and $\frac{252\times 2+252}{21}$ register accesses are necessary for each sub-filter. By multiplying all these operations by the number of sub-filters present in the polyphase filter bank (UMTS and WLAN), the following complexity is obtained and tabled in Table [4.2.15].

	Multiplication	Addition	Register access	
			Load	Store
WLAN (5 sub-filters)	50	45	100	50
UMTS (21 sub-filters)	252	231	504	252

Table 4.2.15: *Complexity analysis for the UMTS and WLAN polyphase filter banks for the general form of polyphase filter.*

However, some improvements are possible to obtain less operation for the final implementation and therefore, reduce the execution time.

First of all, the structure of polyphase filter is symmetric. It means that for N filter coefficients, the N/2 first coefficients are the same as the last N/2 coefficients but in the reverse order. The formula (4.2.16) shows the symmetry of a filter. The Figure [4.2.17] gives a preview of the polyphase filter bank structure considering this symmetry.

$$h(i) = h(n-1-i) \qquad 4.2.16$$

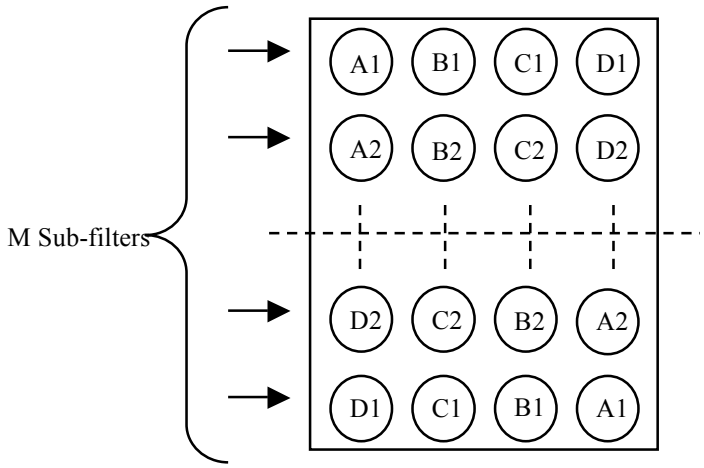


Figure 4.2.17: *Symmetric polyphase filter bank. The first N/2 coefficients are the same as the last N/2 coefficients but in the reverse order*

This specification allows reducing the number of coefficient multiplication, thus sharing the multiplication. The first sub-filters and the last sub-filters share the same coefficient multiplication (A1, B1, C1, D1), the second and the one next to last share the same coefficient multiplication (A2, B2, C2, D2) and so on. The input commutator decimates now by M/2 instead of M. It is half the size for this type of architecture. After dealing the first M/2 sub-filters from the bottom to the top sub-filter, it changes the direction and deals the last M/2 sub-filters from the top to the bottom. The Figure [4.2.18] explains the movements of it.

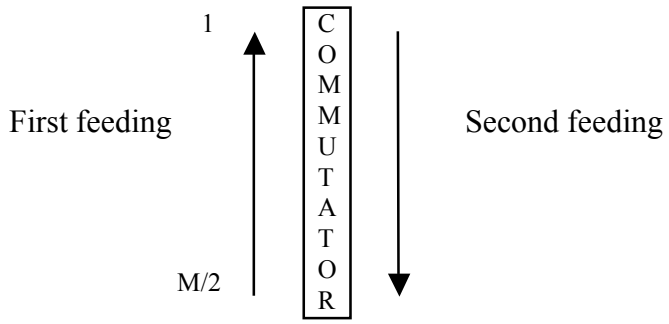


Figure 4.2.18: *Input commutator (1-to-M/2) for the new structure of polyphase filter bank. The commutator feeds the sample data from the bottom sub-filter and move up until the top sub-filter. Then, it keeps feeding but in the other way (move down)*

The Figure [4.2.19] shows the shared multiplication optimizations apply to the first and the last sub-filters. A second optimization, which shares multiplications and additions, is presented in Figure [4.2.20], always for the first and the last sub-filters.

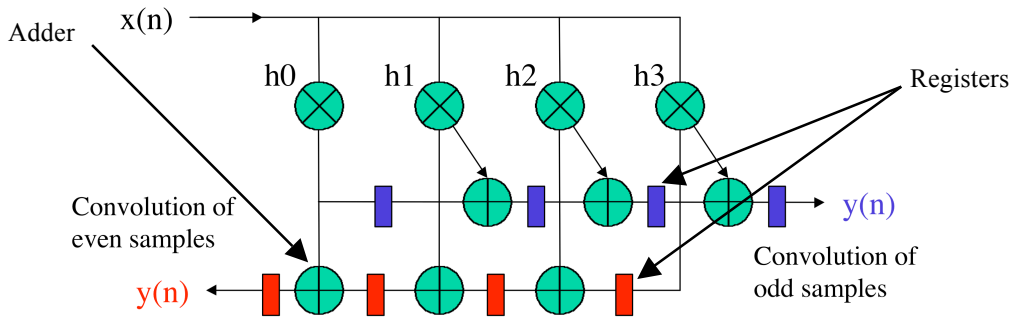


Figure 4.2.19: *Optimization of the polyphase filter bank structure using sharing multiplication method. The coefficient multiplications (h0, h1, h2, h3) are shared for the convolution inside the first sub-filter and the last one. The results are on red output y(n) (respectively blue output y(n)) [18]*

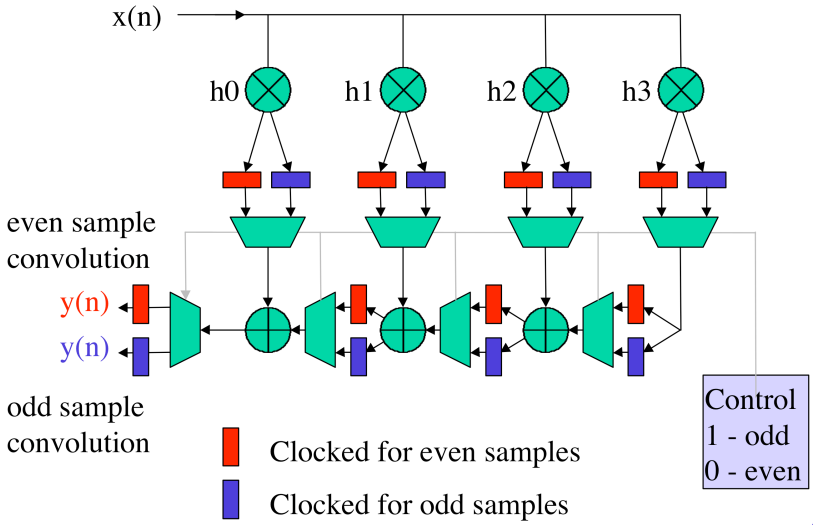


Figure 4.2.20: *Optimization of the polyphase filter bank structure using sharing multiplication and addition method. Coefficient multiplications (h0, h1, h2, h3) are shared for the convolution inside the first sub-filter and the last one. Then, the required result is loaded from register, additions are shared and results are on red output y(n) (respectively blue output y(n)) [18]*

The complexity of these two optimizations is shown in Table [4.2.21] for WLAN and UMTS channelizer. Multiplexers and demultiplexer in Figure [4.2.20] correspond to register accesses. Moreover, clock speed is specifies. Indeed, the clock speed is doubled because the size of the input commutator is divided by 2.

Channelizer	Optimizations	Multiplication	Addition	Register access		Clock
				Load	Store	
WLAN (5 sub-filters)	Opt. 1 (Shared Multiplication)	50/2 25	45	100	50	2*fs/5
	Opt. 2 (Shared Multiplication and addition)	50/2 25	45/2 23	39*5 195	20*5 100	2*fs/5
UMTS (21 sub-filters)	Opt. 1 (Shared Multiplication)	252/2 126	231	504	252	2*fs/21
	Opt. 2 (Shared Multiplication and addition)	252/2 126	231/2 116	47*21 947	24*21 504	2*fs/21

Table 4.2.21: *Complexity of optimized polyphase filter bank structure. The number of multiplications and additions is strongly reduced by means of optimizations in opposition to the number of register accesses (loading and storing). Furthermore, the clock speed doubles due to the change of the commutator length.*

These optimizations allow reducing time because multiply and logical unit are used less time than the “basis” implementation (Table [4.2.15]). On the opposite, a lot of accesses to register are done. For the optimization 1, 100 loads and 50 stores are necessary: 50 coefficients and 50 data are loaded from memory for multiplication and 50 results of the accumulation are stored in memory.

According to the description of the architecture of the DSP in Appendix [B], it is composed of two data paths. Each data path has 1 multiply, 1 logical (addition for example) and 1 shift unit. Moreover, there are 16 32-bits registers. Although the structure with optimization 2 (Shared Multiplication and addition) reduces the number of computation, the DSP architecture does not allow implementing this structure. Indeed, the DSP architecture is fixed and it appears complicated to implement it.

However, filters inside the polyphase filter bank are not all used at the same time. On M sub-filters in the channelizer, M-1 are not used at all the time. Thus, the polyphase filter bank can be implemented as a serial form. To realize this implementation, Multiply and Accumulate (MAC) are used. They can be implemented as two methods: serial form or parallel form. Firstly, the serial form is studied for the serial implementation of the polyphase filter bank. The Figure [4.2.22] shows a serial MAC. The set of data corresponding to each-filter in polyphase filter bank is stored in memory. The position of the input commutator allows choosing the good data at the good address as well as the corresponding coefficient multiplying this data. The result of the multiplication is accumulated to have the final result at the good time for the sub-filter. This process is performed to all the channels (sub-filters) according to the input commutator.

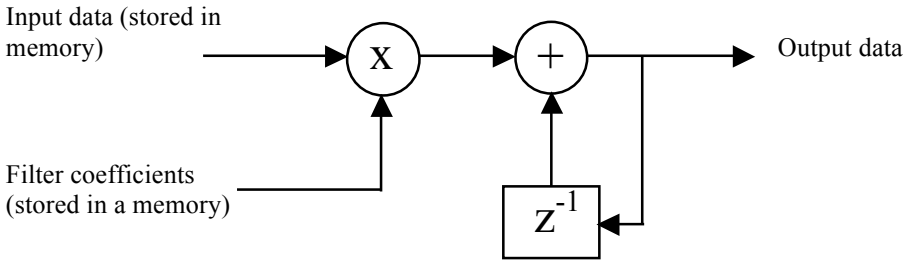


Figure 4.2.22: *Serial polyphase filter bank implemented with serial MAC. The input data (set of data corresponding to each sub-filters) are stored in memory. These data are multiplied by the corresponding coefficients, added and stored in the corresponding registers.*

The complexity for polyphase filter bank using the serial MAC is tabulated in Table [4.2.23]. N corresponds to the number of coefficients.

	Multiplication	Addition	Register access		Clock Speed
			Load	Store	
Serial MAC Implementation	N	N-1	2*N	N	N*fs/M

Table 4.2.23: *Complexity of serial MAC for a serial polyphase filter bank. The clock of the system is the number of coefficients of each sub-filter (N) by the number of channels (sub-filters)(M) at the sampling frequency.*

For the parallel MAC, the input data and the coefficients are multiplied in parallel, at the same time. The clock is the same for the MAC operation as the clock for the input data deliverance. DSP platform can processed 2 MAC in parallel. The Figure [4.2.24] shows the parallel MAC implementation.

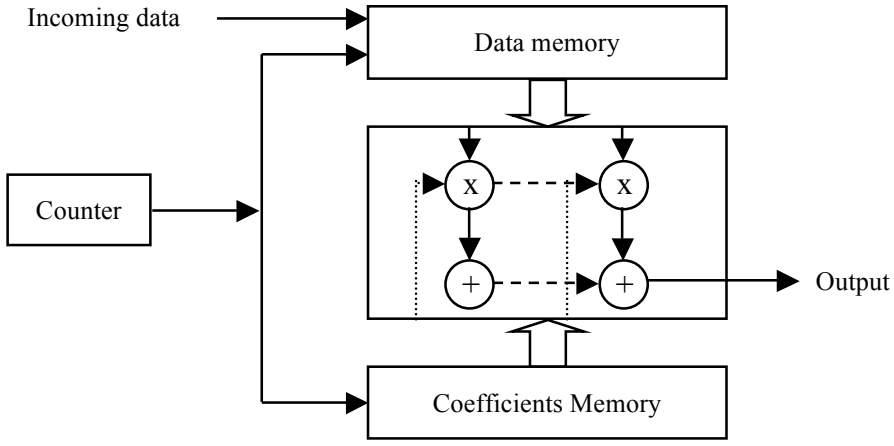


Figure 4.2.24: *Serial polyphase filter bank implemented with parallel MAC. The input data (set of data corresponding to each sub-filters) are stored in memory. These data are multiplied by the corresponding coefficients, added in parallel. Then, the output result is stored in registers to be further processed by the DFT [12].*

The Table [4.2.25] gives the complexity of the parallel MAC implementation for polyphase filter bank. According to [4.2.23] and [4.2.25], the serial MAC implementation uses less resources in terms of hardware than the parallel one, but needs a high clock speed, whereas the parallel MAC is clocked at f_s.

	Multiplication	Addition	Register access		Clock Speed
			Load	Store	
Parallel MAC Implementation	N/(2*M)	(N/(2*M)) -1	2*N	N	fs

Table 4.2.25: *Complexity of serial MAC for a serial polyphase filter bank. The clock of the system is fs in this case. N is the numbers of taps of the non-partitioned filter whereas M is the polyphase sub-filters. The parallel MAC uses more resources in terms of hardware than the serial MAC, but the clock is lower than the other implementation.*

To conclude, simulations have been performed. Results respect almost the specifications presented in chapter [2]. But some problems have to be resolved like downsampling for WLAN channelizer and attenuation in the stopband for UMTS application. The analysis complexity has been carried out on the polyphase filter bank structure. Different solutions for the FIR implemetation have been tried as the general form, the shared multipliers or shared multipliers and adders' methods, the implementation of serial or parallel MAC on a serial polyphase filter bank. According to the Table [4.2.21], [4.2.23] and [4.2.25] giving the complexity of these different structures and the architecture of the DSP, it results that the parallel MAC implementation for a serial polyphase filter bank is the best choice in terms of time constraints.

5 Algorithm to Architecture Mapping

5.1 Overview

In this chapter, the Algorithm to Architecture Mapping is presented. The three parts of the polyphase channelizer (Commutator, FIR filters, DFT) are developed separately for the architecture mapping. WLAN application is chosen (1 sub-channel) and algorithms are mapped on the platform (DSP TMS320C6713). Then, optimizations are carried out to improve the implementation. Functional Units of the DSP are analysed, in terms of execution time (number of cycles). According to the A^3 design model, this section belongs to both algorithm and architecture domain, as illustrated in Figure [5.10].

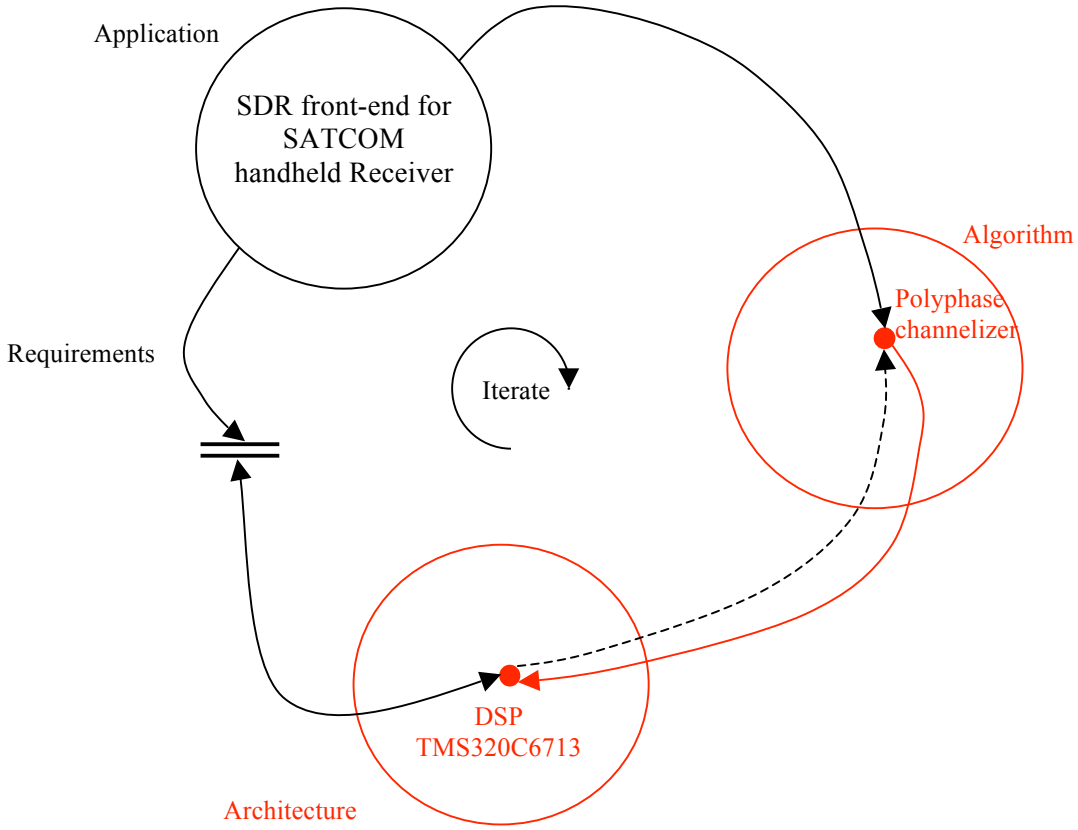


Figure 5.10: *A^3 model for project. Highlighted in red, the mapping on the platform*

In the polyphase channelizer, WLAN polyphase filter bank has 5 sub-filters each of length 10 taps [chapter 4]. Input Sample rate for the WLAN channelizer is 120MHz. 10 data samples are taken as a test signal and acquired in real time. Data are real. Every time a data is fed in the input register, it is processed through the filter and DFT operations before another one is acquired.

The Figure [5.11] shows the part of the polyphase channelizer on which it has been decided to focus.

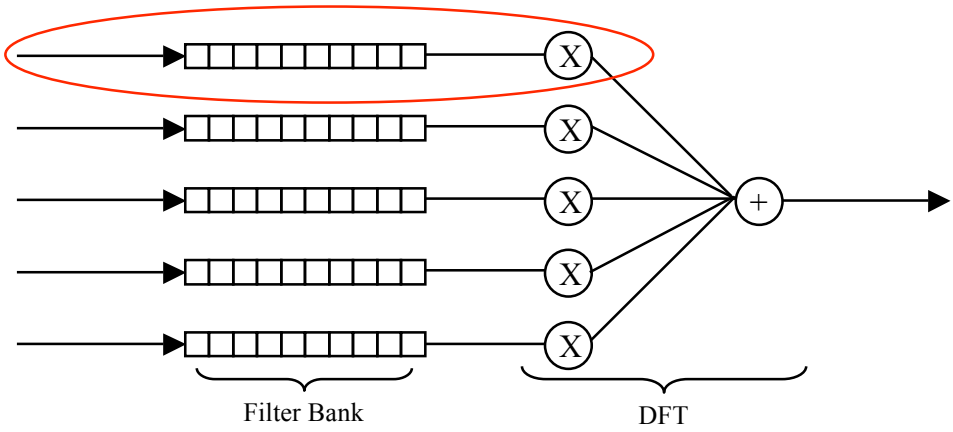


Figure 5.11: *Structure of one channel of WLAN polyphase channelizer. Highlighted in red, one sub-channel. Algorithm corresponding of this part is mapped on DSP processor.*

5.2 Commutator

In this part of the channelizer, two different operations are done: feeding the input array to compute the convolution later and shifting the data inside it for the next operation. Input data are stored in the data memory due to limited number of CPU registers. Operation consists in shifting the data inside the input array to allow the convolution computation in second part of the channelizer. Instruction used is *MVC*. Functional units L1 or L2 perform this task. By examining the architecture in appendix [B], two arguments are passed to this instruction: source register (*src2*) and destination register (*dst*) where data is moved. This operation is carried out by instruction *ADD* where data that must been moved is added by value zero and stored in destination register.

Example

MV .L1 A0, A6

The L1 functional unit moved data from register A0 to register A6. By using instruction *ADD*, data in register A0 is added by zero and stored in register A6. This instruction requires one single cycle of the pipeline for the execution [19].

The instruction *MV* loads data from memory to register. As seen in appendix [B], data are loaded through LD1a and LD1b (for data path A) while data path B uses LD2a and LD2b. The instruction used to load data is *LDW*. This instruction allows loading a word from memory to register. It receives two arguments: *src* that indicates the source (memory) and *dst* which is the destination (register). D1 and D2 are functional units used for this loading.

Example: *LDW* .D1 *A10, B1

In this case, the D1 unit loads data at address pointed by A10 in the internal memory in the register B1. This operation requires 5 cycles of the pipeline for the execution [19]. Indeed, in the first stage (E1), the register file modified the pointer of data address. The stage E2 sends the data address to the memory. Then, the memory reads the address (E3). During the next stage, the data arrives at the CPU Core and during E5, the data is stored in the corresponding register. The Figure [5.20] summarizes these operations inside the pipeline.

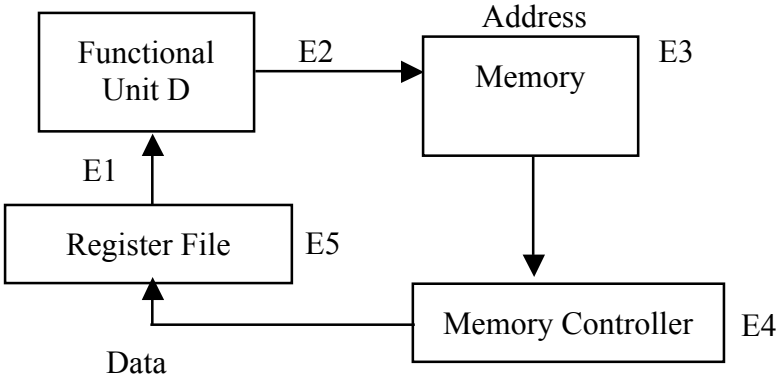


Figure 5.20: *Execution of Load instruction inside the pipeline.*

Data are stored in memory after the shifting. The instruction *STW* is used to carry out this operation. It stores a word to memory from register. Functional unit D1 and D2 are used to do this operation (appendix [B]).

Example: *STW* .D2 B4, *+ B6 [1]

In the example, data in the register B4 is stored at the address pointed by register B6 with an offset of 1. The execution of this instruction requires 3 clock cycles [19]. The Figure [5.21] shows the operation in the pipeline to perform it. While the stage E1, the address where the data will be stored is computed. This address and the data are sent to the memory during E2. Finally, the data is written (E3).

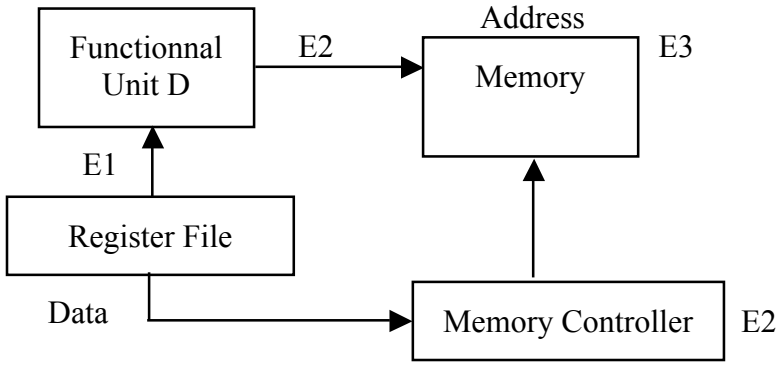


Figure 5.21: *Execution of Store instruction inside the pipeline.*

The Figure [5.22] shows the movement of data between memory and registers for the shifting and feeding operation. The Table [5.23] summarizes the number of clock cycle for the Commutator execution.

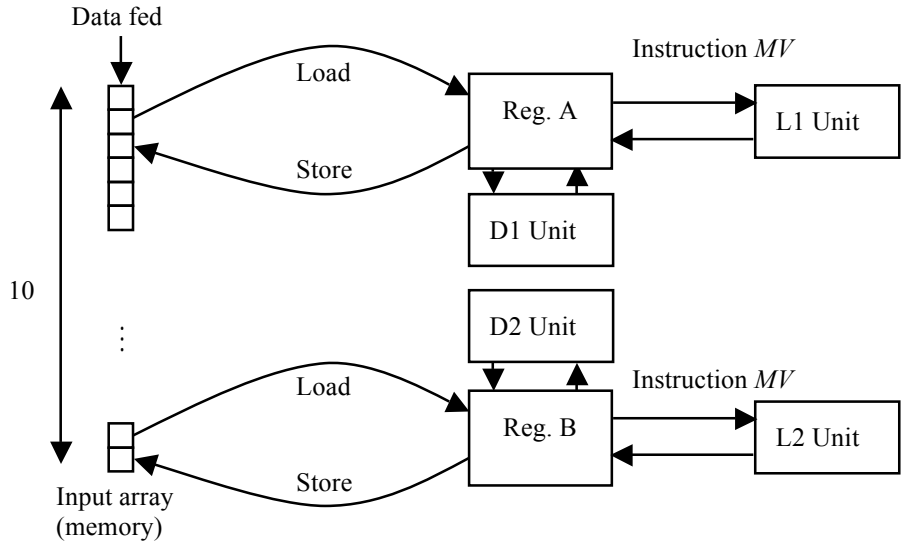


Figure 5.22: *Data flow for the commutator. Firstly, data are shifted and then, new data is fed.*

Data are shifted and fed in an input array. The length of the input array in the external memory is 10. During the shifting, 10 data are loaded from memory to registers to perform the shifting operation (instruction *MV*). Then, shifted data are stored in the input array. These operations are executed in parallel (use of both data paths). Finally, new input data is fed at the appropriate address.

Instruction	Load	Shift	Store
Number of cycles	10*5/(2*2)	10/2	(10*3)/2

Table 5.23: *Number of cycles for one input data fed in the commutator.*

10 loads are carried out to shift the entire input array. Five clock cycles are necessary to perform this instruction. Both data paths work in parallel and there are 2 load inputs for each. 10 shifts are performed in total and 3 cycles are necessary to store data shifted into memory.

5.3 FIR Filter

After having fed and shifted data in input register, the second step of channelizer is filtering the data. A FIR filter carries out this operation. It consists of a simple convolution. All data inside input register are multiplied by filter coefficients and accumulated. This operation, called *MAC* (multiply and accumulate) requires only one single cycle. Architecture, described in appendix [B], is composed of two multipliers (one into each data path). It means that two *MACs* are performed per cycle. Both are

executed in parallel. Therefore, 5 cycles are necessary to convolve 10 input data. Figure [4.2.24], shown in section [4.2.3], describes the structure of parallel MAC implementation. Figure [5.30] represents data path for a *MAC* operation through multiplier and accumulator.

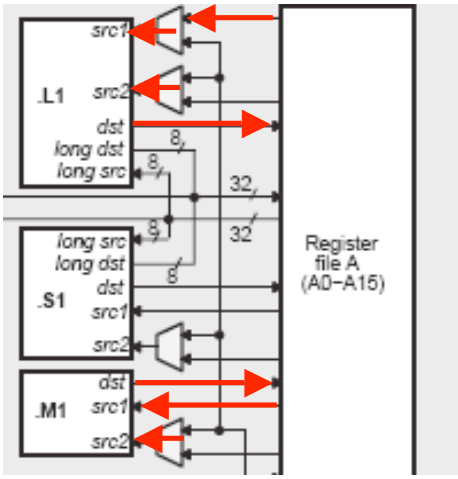


Figure 5.30: *Data path (red line) to execute MAC instruction in one single cycle.*

Phase	Functional Unit
1	Multiplication (M1)
2	Addition (L1)
3	Store (register file)

Table 5.31: *Phase of data movement inside data path (functional unit) for MAC instruction.*

Two operands are loaded from register A to M unit with input *src1* and *src2*. Result of multiplication is on the output *dst* of the multiplier and is accumulated by means of L unit. Table [5.31] shows the different phases of data movement through the functional units.

For this application (WLAN), the length of one sub filter is 10 coefficients (section [4.2.2]. Thus, 10 *MAC*s operations are performed per data fed, or 5 clock cycles are executed for this computation. Moreover, 10 data and 10 coefficients are loaded from memory to registers to perform this convolution (in parallel). Then, the two results of both data paths are added together to obtain the final result of the convolution. This result is stored to register for further computation. The Table [5.33] summarizes the number of clock cycles to execute convolution operation.

Instruction	Load	MAC	Store
Number of cycles	10*5*2/(2*2)	10/2 + 1	-

Table 5.33: *Number of cycles for convolution computation.*

The Figure [5.32] shows data movement between memory and data path. 10 data and 10 coefficients are loaded (5 cycles) in parallel through to 2 load inputs into each data path. 2 *MAC*s are performed in parallel. One last *MAC* is carried out to obtain the final result of the convolution. Finally, result is stored into register instead of memory to not use one store instruction.

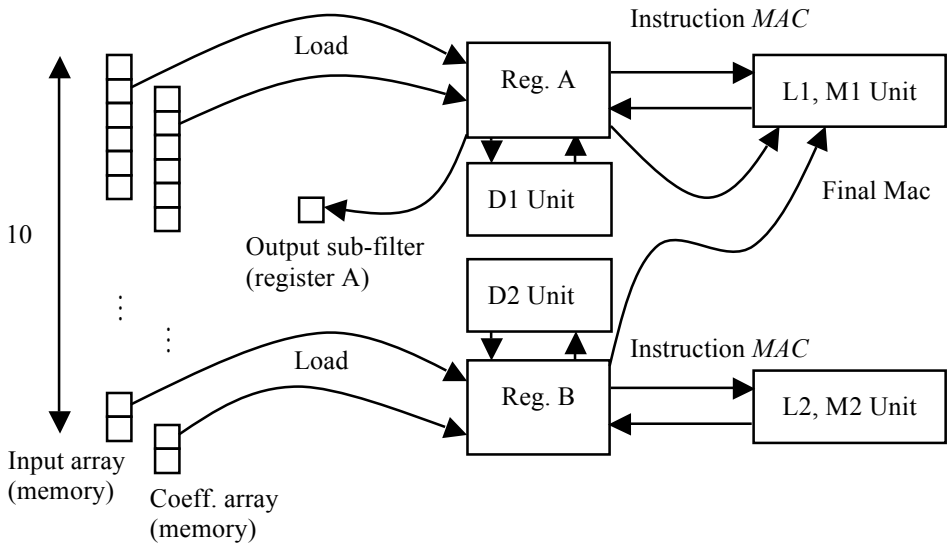


Figure 5.32: *Data flow for the FIR filter. Firstly, data and coefficients are loaded from memory, MACs instructions are performed. The final MAC is carried out into data path A. Final result is stored into register.*

5.4 Discrete Fourier Transform

The last step of the channelizer process is coherent phase summation to extract the down sampled data at baseband. To perform this operation, Discrete Fourier Transform computation is done. The DFT operation is defined by formula [5.40]:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-2i\pi k\frac{n}{M}} \tag{5.40}$$

DFT computation requires N^2 complex multiplications and $N^2 - N$ complex additions. *MAC* instruction is used to compute DFT like the convolution developed above. The difference is these operations are complex, thus the result is separate in two parts: one for the real and one for the imaginary. Indeed, this exponential term can be developed in a sum of cosine and sine shown in formula [5.41]:

$$e^{-2i\pi k\frac{n}{M}} = \cos(2\pi k\frac{n}{M}) - i\sin(2\pi k\frac{n}{M}) \tag{5.41}$$

Therefore, for one DFT, two output data are produced (real part and imaginary part). The computation of DFT requires $4N^2$ multiplications and $6N^2 - 3N$ additions. In terms of execution, two *MAC*s work in parallel to compute DFT (real and imaginary part). One clock cycle is necessary to compute the multiplication of the data by the exponential term and the accumulation. In the case of one sub-channel, N = 1 thus there is no accumulation. Results of data paths are stored in 2-length array (real and imaginary part) into memory. The Figure [5.42] shows the data movement

between memory and data paths. The Table [5.43] summarizes the number of clock cycles to perform DFT computation.

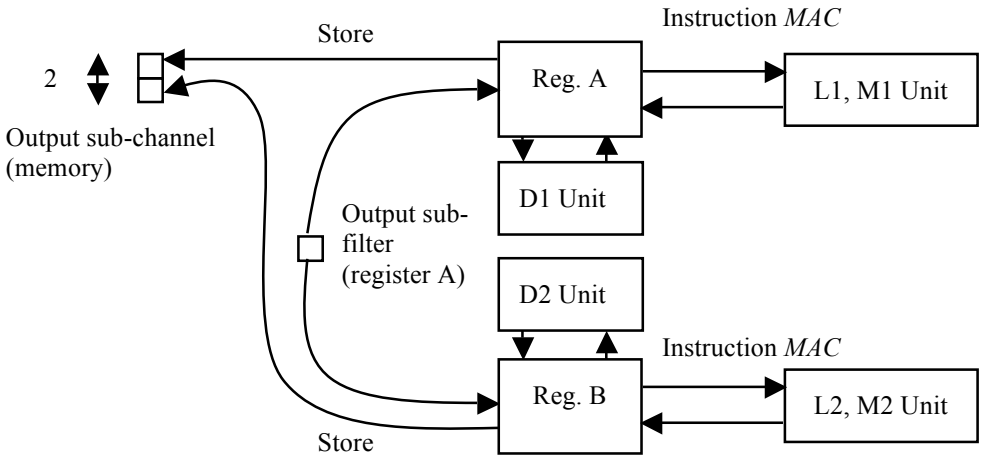


Figure 5.42: *Data flow for DFT computation. Firstly, output sub-filter is moved from register A to register B, MAC instruction is performed for real part (imaginary). Two results are stored in memory.*

Instruction	Shift	MAC	Store
Number of cycles	1	2/2	2*3

Table 5.43: *Number of cycles for DFT computation.*

Two loads are carried out in parallel (one for the real and one for the imaginary part). 2 *MAC*s instructions are also performed in parallel and 1 complex result is stored into memory. The estimation of the computation of exponential term has not been taken into account (computation of cosine and sine).

To summarize this analysis, Table [5.44] shows the number of clock cycles for one sub-channel into the channelizer. One data is fed at a time; the sub-filter inside the channelizer is 10 taps.

Part of sub-channel	Commutator	FIR filter	DFT	Total
Number of cycles	33	31	8	72

Table 5.44: *Number of total clock cycles for one WLAN sub-channel. One date is fed at the input sample rate (120MHz).*

The total number of clock cycles to process data in one WLAN sub-channel is 72. The clock frequency of the DSP is 225MHz. It means that the execution time is $\frac{72}{225} = 320ns$. The input sampling rate is 120MHz, so every $\frac{1}{120} = 8.333ns$, there is a new data fed. Time constraints are not respected. The Figure [5.45] shows timing constraints as well as the execution time of one data process through the channelizer.

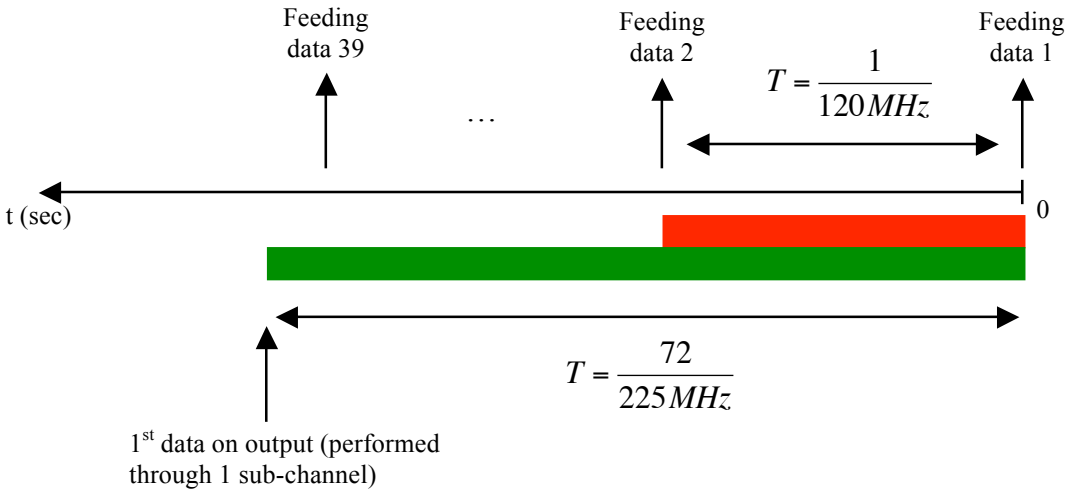


Figure 5.45: *Time processing of one data through one sub-channel (green). One data is fed in the sub-channel every 1/120MHz (red).*

It appears that first data is totally processed in a time when 39 data samples have already been fed. It is necessary to optimize the code in order to reduce the number of instructions, therefore reducing the execution time. The goal is to obtain an execution time less than two input sample intervals (i.e. $\frac{1}{120MHz}$).

5.5 Optimizations

The previous sections ([5.2], [5.3], [5.4]) presented the implementation of the algorithm without optimizations. This part develops implementation modifications in order to reduce the number of cycles. Firstly, circular buffer is presented. Then, deterministic complex terms optimizations is developed.

5.5.1 Circular Buffer

It has been seen in Commutator part of the channelizer that data have to be shifted before feeding new input data. This shifting takes a lot of clock cycles. Indeed, 1 clock cycle is taken to shift one data from it actual place in memory to the next one. 10 cycles are necessary to carry out this operation into functional units for 10 data inputs. The TMS320C6713 can perform linear or circular addressing. Circular addressing is very interesting for the FIR computation because input data have to be shifted before every data feed. This addressing mode is only possible with registers A4-A7 and B4-B7 [19]. The Figure [5.50a-b] illustrates the behaviour of a circular buffer in five consecutive memory locations.

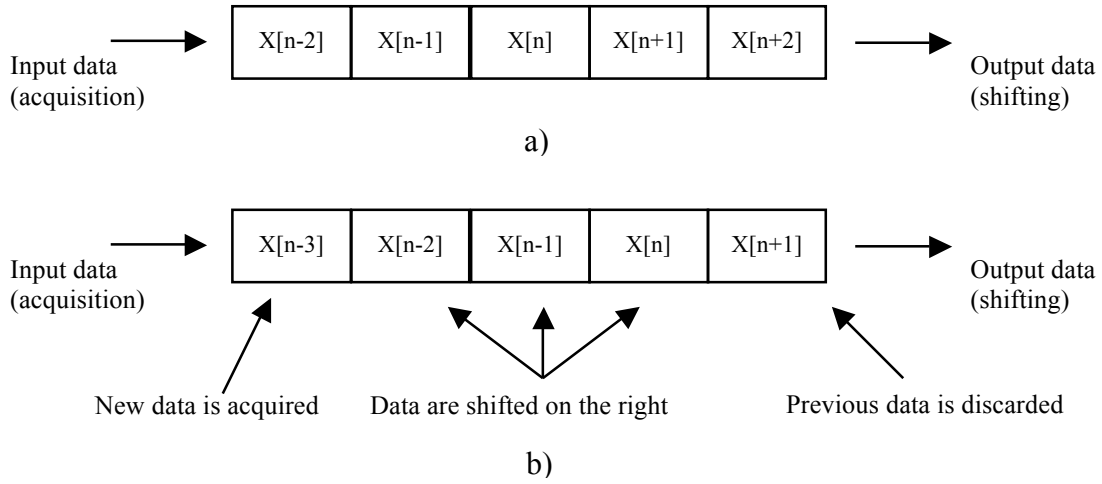


Figure 5.50: *Circular buffer with 5 samples. State of the buffer at one particular instant (a). A new sample is acquired (b) and all the sample are shifted to the right (one step)*

In Figure [5.50a], there are five samples stored at one particular instant. The Figure [5.50b] shows the changes when a new sample is acquired. All the data in the buffer are shifted (one step) to the right. The last previous data of the buffer is discarded (X[n+2] in our example).

Circular buffer is very efficient on the platform used on this project. Indeed, all the data are shifted in only one single clock cycle. In comparison with the first implementation, which used the instruction *MV* and that required one cycle to move one sample from a register to another (10 clock cycles in total), the number of cycles is reduced for each data acquired.

As seen before, this addressing mode is only possible on 8 registers (A4-A7 and B4-B7). It means that only 8 data are shifted per cycle. Input array length in memory is 10. Therefore, 2 cycles are necessary to shift all data before feeding a new input sample. The new results for the commutator execution, in terms of number of cycles, are tabled in Table [5.51].

Instruction	Load	Shift	Store
Number of cycles	10*5/(2*2)	10/(2*2)	(10*3)/2

Table 5.51: *Number of cycles for one data feed in the commutator, with circular buffer optimization.*

The difference between the first (in section [5.2]) and the optimized implementation is the reduction of clock cycles for the shift instruction. The number of cycles is reduced by 2.

5.5.2 Deterministic Complex Terms

In the previous implementation of the DFT, the output of sub-filter (stored in register) is multiplied by the exponential term. The exponential term $e^{-2\pi ik\frac{n}{M}}$ is computed inside functional units before the multiplication with data. It takes lot of instructions to compute this term (cosine and sine). In order to reduce the number of computation, deterministic complex terms optimization is applied. The DFT length is fixed. Therefore, the exponential term is known before the DFT computation. It is stored in memory. The number of instruction to compute all these exponential terms is reduced. Only 1 load from the memory to registers is necessary to carry out *MAC* instructions for DFT.

5.5.3 Building Optimizations

It has been found out some builds options on CCS to compile code in an optimized way. These optimizations, presented in Appendix [B], allow reducing code size, using pipeline and parallel architecture in an efficient way in order to reduce the number of clock cycles. A general description of these optimizations is done in Appendix [B]. These optimizations are also implemented. They should allow reducing the execution time in order to respect time constraints.

In conclusion on this chapter, the algorithm has been mapped on the architecture. One date has been fed in the sub-channel and the number of instruction cycles has been estimated. It has been seen that time constraints are not respected. Indeed, execution time is higher than the input sample rate. In order to reduce this time, some optimizations have been studied. It is difficult to know if these optimizations can bring some improvements, due to the impossibility to give an estimation (especially for building options and deterministic terms). The next chapter, which treats the implementation, goes to allow knowing if these improvements can reduce time in order to respect time constraints.

6 Implementation

6.1 Overview

This chapter puts in practice the theoretical analysis developed in chapter [4] and [5]. It contains the results of the first implementation of one sub-channel of WLAN polyphase channelizer as well as the results of various optimizations applied to this implementation. All these results are evaluated, compared and discussed. According to the A^3 design model, this section belongs to the architecture domain, as illustrated in Figure [6.10].

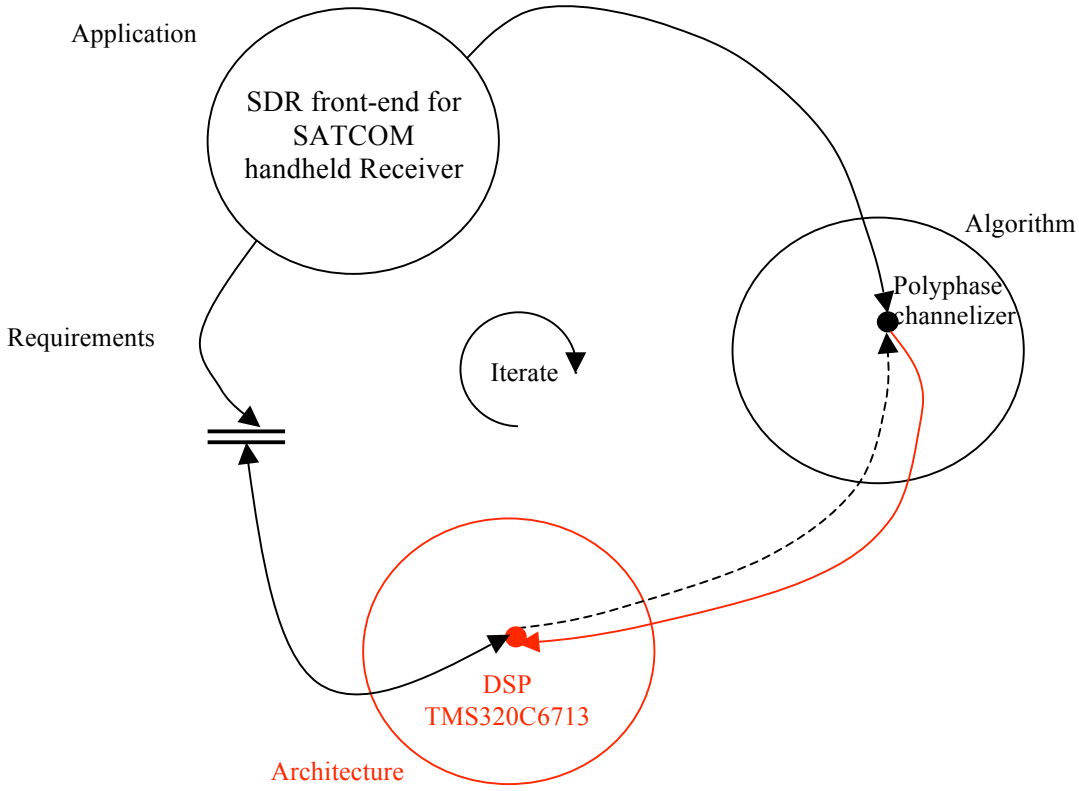


Figure 6.10: *A^3 model for project. Highlighted in red, the implementation on the platform*

6.2 Test Definition

The tests are carried out on the algorithm implemented on DSP platform. Firstly, results are compared with MatLab ones to have a reference. Each result of parts of the sub-channel (commutator, convolution and DFT) are analysed and confronted to MatLab to check if the code is working well.

Then, various tests are performed. Profiler on CCS is used. By adding some functions or piece of code, profiler gives some measures of the selected functions or pieces of code while the execution of code. The first test is carried out on one sub-

channel of polyphase channelizer for WLAN application. Then, the different optimizations developed in chapter [5] are applied to this sub-channel, to see the potential improvements.

Two parameters are evaluated during these tests: execution time and number of clock cycles. Measurement of the time and number of cycles is realized by the profiler and is carried out on the function *Commutator*, *FIR filter* and *DFT*. The hardware utilization in data paths is examined by using the option "mixed view C/ASM" on CCS. ASM view allows seeing what are the instructions used in various functions.

6.3 Tests Results

6.3.1 One WLAN Sub-Channel

The Table [6.30] gives the results of the execution of one sub-channel of the polyphase channelizer. The time to perform feeding, convolution and DFT are shown. For each part of the sub-channel, execution time, when one data is fed, is measured. Moreover, the measurement of the sub-channel execution is done. Measurements are done in terms of number of instruction cycles and of time (nsec).

	Commutator	Convolution (FIR filtering)	DFT	Total sub-channel
Number of instruction cycles for one data fed	236	11000	458	11600
Time for one data fed (nsec)	1180	55162	2290	58000

Table 6.30: *Execution time for each sub part of one WLAN sub-channel (commutator, FIR and DFT). Results are displayed in number of instruction cycles and time (nsec). Furthermore, a measurement of the execution time for the whole sub-channel is realized.*

The results in the Table [6.30] are unexpected. Indeed, the results between the theoretical analysis in chapter [5] (Table [5.44]) and the implementation (Table [6.30]) are different. For instance, for the input commutator, expected result was 33 cycles and the practical result is 236 cycles. The estimation and the practical results are tabled in Table [6.31].

	Commutator	Convolution (FIR filtering)	DFT	Total sub-channel
Estimation	33	31	8	72
Practical results	236	11000	458	11600

Table 6.31: *Comparison of estimation and the practical results of the different parts of the sub-channel. Results are in number of clock cycles. One data is fed and processed through the sub-channel. Furthermore, the number of clock cycles for the whole sub-channel is presented.*

There are differences between the estimation and practical results. For example, concerning DFT computation, there are two explanations to justify this difference:

- First of all, the estimation of DFT computation has not taken into account of exponential term computation. Indeed, this estimation is consisted of:

    1) Loading of data
    2) Multiplication of data by exponential term

    In this estimation, the computation of exponential term should have added. Indeed, the computation of cosine and sine used lot of instructions: number of clock cycles is 304 for cosine function whereas the sine function takes 260 clock cycles.

- Secondly, it has been said that the DFT was computed by using *MAC* instructions. But for this case of implementation (N=1), there is no accumulation. Only one multiplication is performed. Data is just multiplied by the exponential term. The instruction used is not a *MAC* but *MPYI*. This instruction required 9 clock cycles to be performed [19].

Moreover, it appears, on the sight of the assembly code, that the compiler, while the declaration of a variable (int i = 2 for instance), affects the value 2 to a register and then, stores the value of this register in the stack. As seen previously, the instruction *STW* requires 3 clock cycles.

Time constraints are not respected. According to the theoretical analysis developed in section [5], execution time to process 1 data was bigger than two input sample intervals. Therefore, it is obvious to say that this implementation does not respect time constraints because practical results are bigger than estimation.

The different optimizations developed in chapter [5] are necessary to try to reduce the number of clock cycles in order to respect the timing. The next section presents results of optimized code.

6.3.2 Results After Optimizations

Firstly, build options are changed in order to reduce and reorganize the source code. In this way, the optimizer tries to fit the code well on the platform in order to use efficiently the architecture of the DSP (parallelism, pipeline). The first optimizations (–O0) enables register optimizations, –O1 concerns local optimizations, –O2 treats functions whereas the last one (–O3) performs files modifications. The Table [6.30] summarizes results of these optimizations. It is noticed that these optimizations reduce the number of clock cycles in comparison with the first implementation in section [6.3.1]. The more the level of optimization is, the more is the reduction in execution time decreased. By analysing the assembly code, with optimization, the code size is reduced and the shift operation inside the commutator uses the parallel architecture

well to shift all the data inside the input array. However, results are still bigger than estimation, and thus also constraints. Table [6.3.2] regroups all these results.

Optimizations	Measurements	Commutator	Convolution (FIR Filtering)	DFT	Total sub-channel
Opt. -O0	Number of cycles	188	11034	452	11528
	Time (nsec)	(940)	(55170)	(2260)	(57640)
Opt. -O1	Number of cycles	161	10996	457	11419
	Time (nsec)	(805)	(54981)	(2285)	(57095)
Opt. -O2	Number of cycles	32	474	804	11207
	Time (nsec)	(160)	(2370)	(4020)	(56036)
Opt. -O3	Number of cycles	32	474	804	11207
	Time (nsec)	(160)	(2370)	(4020)	(56036)

Table 6.3.1: *Execution time for each part of one WLAN sub-channel (commutator, FIR and DFT) with various compiling optimizations. Results are displayed in number of instruction cycles and time (nsec).*

Execution	Measurements	Commutator	Convolution (FIR Filtering)	DFT	Total sub-channel
Estimation	Number of cycles	33	31	8	72
	Time (nsec)	(146)	(137)	(35)	(318)
Execution without Opt.	Number of cycles	236	11000	458	11600
	Time (nsec)	(1180)	(55162)	(2290)	(58000)
Execution with Opt. –O3	Number of cycles	32	474	804	11207
	Time (nsec)	(160)	(2370)	(4020)	(56036)

Table 6.3.2: *Comparison of the estimation of execution time and various implementations (implementation without optimizations, with compiling optimization). Execution time for each part of one WLAN sub-channel (commutator, FIR and DFT). Results are displayed in number of instruction cycles and time (nsec).*

Table [6.3.2] shows that compiling optimization –O3 reduced strongly the number of clock cycles for commutator and FIR parts of the sub-channel in comparison with the implementation. However, number of cycles for DFT computation is always high with or without optimizations. Furthermore, timing constraints are not respected yet. The total execution time for the optimized implementation (level –O3) is 11207ns whereas one data is fed every 8.33ns (120MHz).

Now, implementation with the deterministic complex term optimizations is carried out. Exponential terms of the DFT are stored in memory. Instead of compute these term in data path, there are just loaded from memory to register to be multiplied by data in the inner product of DFT. This optimization is performed in order to reduce

number of clock cycles for DFT part of the sub-channel. Table [6.3.3] shows the result of this optimization as well as the results of previous optimizations for comparison. The number of cycles is reduced for DFT part when the implementation is performed with the deterministic term optimization. As it has been said above, one load is necessary to compute the inner product in DFT operation. The result of the combination of compiling option –O3 and deterministic term optimization is also shown in Table [6.3.3].

Execution	Measurements	Commutator	Convolution (FIR Filtering)	DFT	Total sub-channel
Estimation	Number of cycles	33	31	8	8
	Time (nsec)	(146)	(137)	(35)	(35)
Execution without Opt.	Number of cycles	236	11000	458	11600
	Time (nsec)	(1180)	(55162)	(2290)	(58000)
Execution with Opt. –O3	Number of cycles	32	474	804	11207
	Time (nsec)	(160)	(2370)	(4020)	(56036)
Execution with Deter. Terms	Number of cycles	236	11000	55	11351
	Time (nsec)	(1180)	(55162)	(256)	(56783)
Execution with Opt. –O3 & Deter. Terms	Number of cycles	32	474	55	10654
	Time (nsec)	(160)	(2370)	(256)	(47351)

Table 6.3.3: *Comparison of the estimation of execution time and various implementations (implementation without optimizations, with compiling optimization –O3, deterministic terms and combination of the last both). Execution time for each part of one WLAN sub-channel (commutator, FIR and DFT). Results are displayed in number of instruction cycles and time (nsec).*

The combination of the compiling option and deterministic terms gives satisfied results in terms of execution time in comparison with the first implementation without optimization. By comparing with the estimation, optimizations are better for the commutator part. Only FIR and DFT is still higher. The Circular Buffer optimization has not been implemented. Indeed, this kind of addressing mode requires coding in Assembly code, especially to select the appropriate registers to perform it. But it might guess that this optimization reduces the execution time for the Commutator part.

Although improvements are performed in order to reduce the execution time, time constraints are not respected yet. After having reflexion on this problem, it appears that implementation of this algorithm is impossible at this sampling frequency (120MHz). The interval between two data samples is too small to perform all the process. This interval corresponds of 2 clock cycles of the DSP. It is not possible to obtain data on the output of the sub-channel in only 2 clock cycles.

7 Conclusion & Perspectives

7.1 Conclusions

The goal of Software Defined Radio system is to be reconfigurable without changing DSP processor, i.e. it must serve a wide variety of radio protocols in real time. In our case, this is realized for multi standards receivers such as mobile phones, Global Positioning System, etc. These receivers work for applications as satellite communication, Bluetooth, ZigBee, WiMAX, etc. For this project, two standards have been selected: UMTS and WLAN. To speed up the data process on these receivers, special algorithm can be implemented. With processor platform especially designed for signal processing, the speed can be improved as soon as the architecture is well managed.

The goal of this Master Thesis ASPI project is to answer the problem defined in section [1.3] as follow:

"Performance evaluation of a Digital Signal Processor implementation of a Multi-Standard Digital Radio Receiver?"

First of all, an analysis of the design has been done to determine how the system can process data received at the antenna. It appears that the use of polyphase channelizer is the best way to process this application. Polyphase channelizer is composed of three parts: Commutator, FIR filter and DFT. It has been discovered that digital front-end (part before the polyphase channelizer) could not be performed on DSP platform. The sampling frequency is higher than DSP platform. Some techniques (filtering + heterodyning) in the analog domain have been tried in order to resolve this problem, but without success. Therefore, it has been decided to focus only on the polyphase channelizer survey.

By means of this analysis, the algorithm is simulated on MatLab to fit the specifications of the application. The results of simulation allowed confirming that the algorithm was well coded. A complexity analysis of the algorithm has been carried out to determine what kind of implementation and which resources (in terms of mathematical functions) are used on the architecture. Parallel MAC implementation has been chosen for filter bank inside the channelizer.

Then, the architecture of the board has been studied as well as tool used for the final implementation. The platform is a DSP from Texas Instruments. The target platform is TMS320C6713. It is clocked at 225MHz and designed especially to process floating point data. The algorithm is coded in C language and Code Composer Studio is used to compile and load program on the board.

Before the final implementation, the mapping of the algorithm has been done to see how the algorithm fit on the architecture. A theoretical analysis of the instructions is realized to have an idea of results we should except. This analysis consists of the estimation of number of clock cycles of instruction used to process the channelizer (for instance, load filter coefficients from memory to registers). This theoretical analysis does not respect time constraints. Indeed, theoretical execution time is bigger than specifications time (time to process data through the channelizer is bigger than

time between 2 data samples). Therefore, some optimizations have been developed to reduce the execution time. It has been discovered that the required specifications of the application could not be obtained with the implementation on this processor. Indeed, the clock frequency of the DSP is lower than the frequency specifications of both standards (UMTS and WLAN).

However, the implementation on the platform has been performed and tests have been done. Indeed, although this application cannot be performed on this DSP platform, it allows seeing how this algorithm fit on this DSP architecture. The first implementation has been carried out on only one WLAN sub-channel of the polyphase channelizer. The fact to execute one sub-channel provides a preview of the execution on the platform. The results, in terms of computation are the same as MatLab. The measures have shown that the compiler used more instructions than the ones developed in the theoretical analysis. The estimation for the execution of the channel was 72 clock cycles whereas the practical result is 11600. Some optimizations, like deterministic terms and compiling options, have been developed and results have been improved, in terms of execution time. The combination of deterministic term and compiling –O3 optimizations gives almost the same result as the estimation. Although these optimizations, time constraints are not respected.

7.2 Perspectives

Firstly, some perspectives are possible in architecture point of view.

It has been seen that the clock frequency of the DSP was too low for this application. The main problem is that the bandwidth of input signal is too wide. Aliasing and overlapping problems appear after the sampling operation. The implementation of standards is possible on DSP platform with a higher clock frequency. Nowadays, DSPs typically run at 1 GHz. This type of platform may suit for this application that requires an 840MHz-sampling frequency.

By keeping TMS320C6713 DSP, only one standard can be processed on it. For a multi standard receiver, the use of several DSPs has to be done. The standards are still received at 1 antenna, but filtered in analog domain before the conversion to digital domain. There is thus only one DSP per standard.

Secondly, improvement can be done in terms of code. In this project, algorithm has been coded in C-language but it has been seen in Appendix [B] that the Assembly code is the last step of code flow development on CCS, especially for optimizations. The fact that to code in Assembly allows directly manipulating registers, instruction, etc. It will allow, for instance, developing Circular Buffer optimization. It is a low level language that is the closest to the architecture. The problem is that for this kind of application that requires a lot of code, the ASM is complicated and requires lot of time to code.

Bibliography

[1] Enrico Buracchini, "The Software Radio Concept", CSELT, in IEEE communications Magazine, September 2000

[2] Fredric J. Harris, Chris Dick and Michael Rice, "Digital Receivers and Transmitters Using Polyphase Filter Banks for Wireless Communications" in IEEE transactions on microwave theory and techniques, Vol. 51, NO.4, April 2003

[3] S. K. Mitra, "Digital Signal Processing: A Computer Based Approach", 2nd Ed. New York: McGraw-Hill, 2001

[4] J. Wozencraft and I. M. Jacobs, "Principles of Communication Engineering", New York: Wiley, 1967, sec.7.2

[5] P. P. Vaidyanathan, "Multirate Systems and Filter Banks", Englewood Cliffs, NJ: Prentice-Hall, 1993

[6] Yannick Le Moullec, DSP Design Methodology, AAU 2007. Lectures notes for mm1 of course in DSP Design Methodology, ASPI8-4 http://kom.aau.dk/ylm/aspi8-4-part1-2007.pdf

[7] Axel Jantsch, Shashi Kumar, Ahmed Hemani, "The Rugby Meta-Model", school of engineering, Jönköping University, Jönköping, Sweden, March 21 2000

[8] Schafer & Buck, "Discrete Time Signal Processing", OpenCourseWare 2006, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science

[9] Roger S. Meier, "Digital Decimating Filter For a Monolithic Sonar Receiver", http://www.the-meiers.org/professionalinfo/publications/msthesis/thesis/thesis.html, visited the 21st of April 2009

[10] Emmanuel C. Ifeachor and Barrie W. Jervis, "Digital Signal Processing – A Practical Approach", Addison Wesley Publishers Ltd, 1993

[11] Nyquist-Shannon sampling theorem, http://en.wikipedia.org/wiki/Nyquist–Shannon_sampling_theorem, visited the 3rd of March 2009

[12] Mehmood-Ur-Rehman Awan, Muhammad Mahtab Alam, "Design & Implementation of FPGA-based Multi-standard Software Radio Receiver", AAU 2007, Master Thesis Project, ASPI10-2007 – Gr. 1044, http://projekter.aau.dk/projekter/retrieve/9933823?format=application/pdf.

[13] Fredric J. Harris, "Multirate Signal Processing for Communication Systems", 2004 Pearson Education, Inc.

[14] “TMS C6000 Technical Brief”, Litterature Number: SPRU197D, February 1999 http://www.ee.ic.ac.uk/pcheung/teaching/ee3_Study_Project/C67x%20Technical%20Brief(197d).pdf

[15] Andreas Popp, TMS320C6713 Workshop I, AAU 2007, Crash course, http://kom.auc.dk/~anp/teaching/tms6713workshop2007/

[16] “TMS320C6713 DSP Description”, http://www.entegra.co.uk/c6713.htm, visited the 25th of April 2009.

[17] “TMS320C6713 Floating Point Digital Processor”, SPRS 186L, November 2005 http://focus.ti.com/lit/ds/symlink/tms320c6713.pdf, visited the 27th of April 2009.

[18] Raghu Rao, Matthieu Tisserand, Mike Severa, Prof. John Villasenor, “FPGA Polyphase Filter Bank Study and Implementation” http://slaac.east.isi.edu/presentations/retreat_9909/polyphase.pdf

[19] “TMS320C67xx/C67x+ DSP, CPU and Instruction Set Reference Guide”, http://www.diegm.uniud.it/~bernardi/Didattica/DSP-TI/spru733.pdf

Appendix A: Polyphase Channelizer

The FDM signal is downconverted to baseband, filtered and subjected to a sample rate reduction. A conventional channelizer, in Figure [A.1], can perform this task. It is composed of down-converters, baseband filters and resamplers.

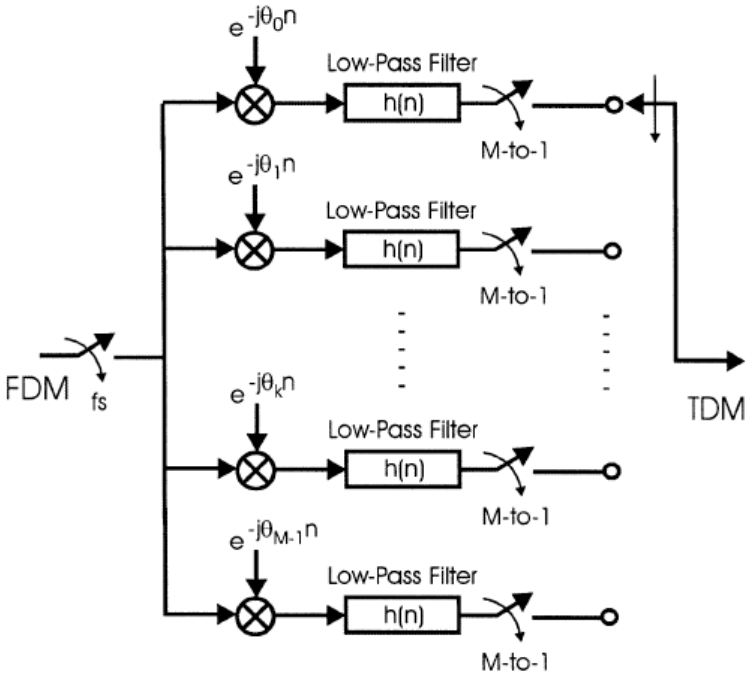


Figure A.1: *Conventional channelizer [2]*

In the conventional channelizer, each channel needs individual channelizer and these channelizers can perform only one channel. Another implementation that performs the channelization is called Polyphase *N*-path filter bank, as shown in Figure [A.2].

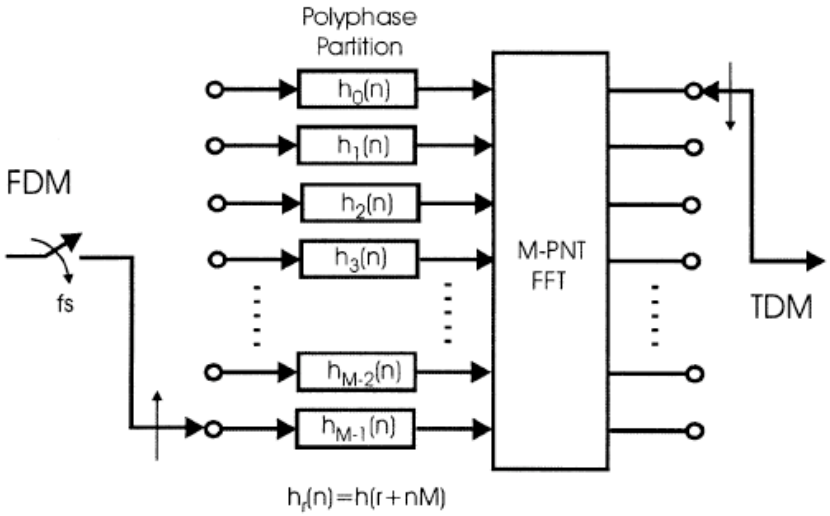


Figure A.2: *Polyphase channelizer [2]*

It is capable of delivering all the required channels with only one channelizer. The new channelizer offers more advantage than the previous channelizer shown in [A.1] in terms of cost, due to reduction in system resources required to perform the multichannel processing, and is more efficient when large sampling rate changes are required.

Polyphase channelizer uses resampler, all-pass partition and FFT phase shifters. The path between the conventional and polyphase channelizer is described below now. First of all, the block diagram of one channel of the conventional channelizer is shown in Figure. [A.3]. The output $y(n,k)$ of the digital low-pass filter is a simple convolution operation, as described in equation (A.4):

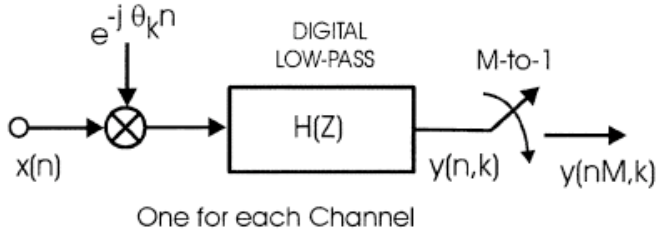


Figure A.3: *k^{th} channel for conventional channelizer [2]*

$$y(n,k) = \left[x(n)e^{-j\vartheta_k n}\right] * h(n)$$
$$y(n,k) = \sum_{r=0}^{N-1} x(n-r)e^{-j\vartheta_k (n-r)}h(r) \qquad \text{(A.4)}$$

The summation in equation (A.4) is rearranged in order to the equivalency theorem [4] which says that the operation of down-conversion followed by a low pass filter is equivalent to an operation of a bandpass filter followed by a down-conversion. This rearrangement is shown in (A.5) and the new version of the kth channel is shown in Figure [A.6].

$$y(n,k) = \sum_{r=0}^{N-1} x(n-r)e^{-j\vartheta_k (n-r)}h(r)$$
$$y(n,k) = \sum_{r=0}^{N-1} x(n-r)e^{-jn\vartheta_k}h(r)e^{jr\vartheta_k}$$
$$y(n,k) = e^{-jn\vartheta_k}\sum_{r=0}^{N-1} x(n-r)h(r)e^{jr\vartheta_k} \qquad \text{(A.5)}$$

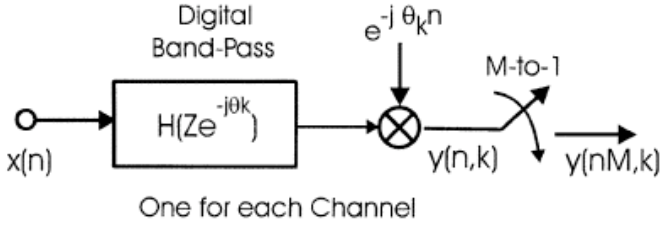


Figure A.6: *Bandpass filter, k^{th} channel for channelizer [2]*

Moreover, during the sample rate conversion, there only is one retained sample in every M samples. Therefore, there is no interest to downconvert all the output samples from the filter. The next operation consists to interchange the down-converter with the down sampler. In this case, only the retained samples are downconverted. The following Figure [A.7] shows this transformation. The time series of the complex sinusoid is also downsampled, hence the rotation rate $M\vartheta_k$ now.

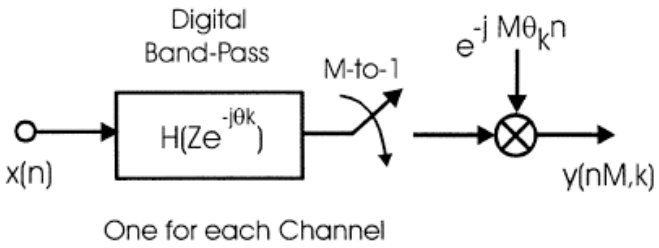


Figure A.7: *Down sampled and down-converter bandpass, k^{th} channel [2]*

However, a problem of aliasing appears because of this change. Indeed, the fact to bring the down-converter after the resampler downsamples the time series of the complex sinusoid. The rotation rate of the sampled complex sinusoid is $M\vartheta_k$ radians per sample at the output of the resampler. A sinusoid at one frequency or phase slope could be at another phase slope after have been resampled. A constraint is applied on the sampled data center frequency. The center frequencies ϑ_k alias to zero conversion (dc) as the result of the down sampling $M\vartheta_k$. It involves $M\vartheta_k = k2\pi$ or $\vartheta_k = \frac{k2\pi}{M}$. This modification is seen in Figure. [A.8].

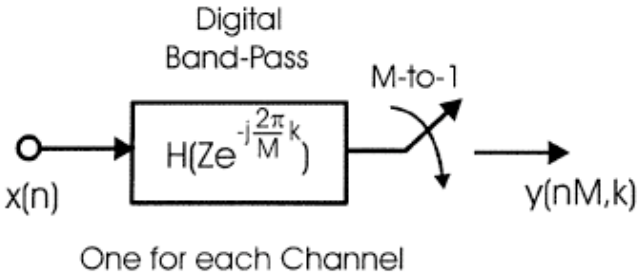


Figure A.8: *Alias to baseband down sampled down-converter bandpass, k^{th} channel [2]*

As the idea developed above (equivalency theorem) and seeing the Figure [A.8], it appears useless to compute the output discarded samples from the passband filter. Following the theorem of noble identity [4], the operations of down sampling are done before the computation in the bandpass filter. According to the noble identity, "The output from a filter $H(Z^M)$ followed by an M-to-1 down sampler is identical to an M-to-1 down sampler followed by the filter $H(Z)$." The noble identity works in our case but a rearrangement of the filter is necessary. For the moment, the Figure [A.9] shows the M-path partition of a resampled digital filter.

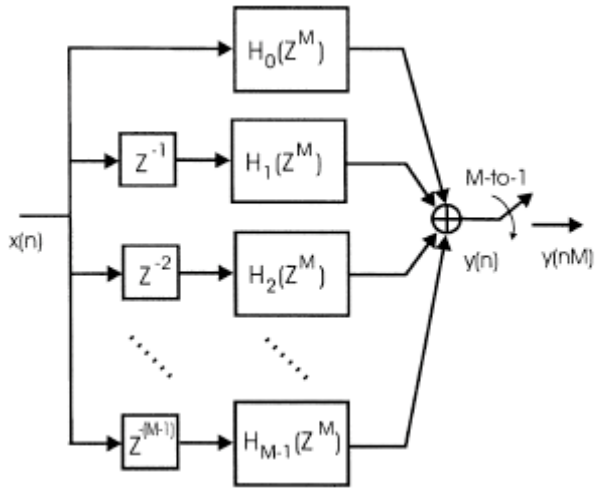


Figure A.9: *M-path filter with output resampler [2]*

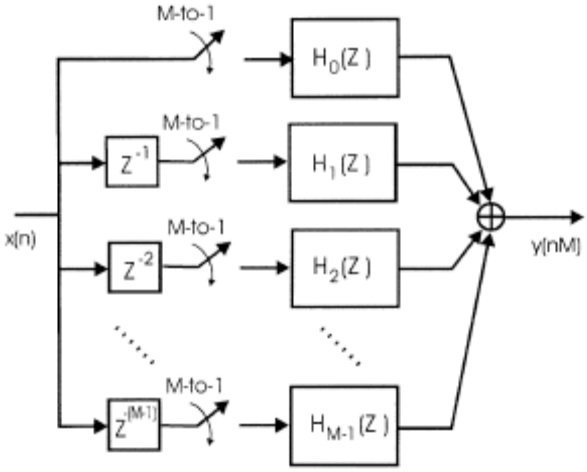


Figure A.10: *M-path filter with input resamplers [2]*

The representation is explained below, with a Z-transform description of the partition:

$$H(Z) = \sum_{n=0}^{N-1} h(n)Z^{-n}$$
$$H(Z) = h(0) + h(1)Z^{-1} + h(2)Z^{-2} + ... + h(N-1)Z^{N-1} \quad \text{(A.11)}$$

Anticipating the *M*-to-1 resampling, the summation in (A.11) is partitioned in a summation of summation, as shown in (A.12)

$$\begin{aligned} H(Z) &= h(0) + h(M+0)Z^{-M} + h(2M+0)Z^{-(2M+0)} + ... \\ &..... + h(1)Z^{-1} + h(M+1)Z^{-(M+1)} + h(2M+1)Z^{-(2M+1)} + ... \\ &..... + h(2)Z^{-2} + h(M+2)Z^{-(M+2)} + h(2M+2)Z^{-(2M+2)} + ... \\ &..... + h(3)Z^{-3} + h(M+3)Z^{-(M+3)} + h(2M+3)Z^{-(2M+3)} + ... \end{aligned} \quad \text{(A.12)}$$

$$..... + h(M-1)Z^{-(M-1)} + h(2M-1)Z^{-(2M-1)} + h(3M-1)Z^{-(3M-1)} + ...$$

This equation is easily rewritten in a compact form described after (A.13):

$$H(Z) = H_0(Z^M) + H_1(Z^M)Z^{-1} + H_2(Z^M)Z^{-2} + ... + H_{N-1}(Z^M)Z^{N-1}$$
$$H(Z) = \sum_{r=0}^{M-1} Z^{-r} H_r(Z^M) = \sum_{r=0}^{M-1} Z^{-r} \sum_{n=0}^{N-1} h(r+nM)Z^{-nM} \quad \text{(A.13)}$$

By means of the noble identity, the down sampling operation is executed before the filter operation, as shown in Figure [A.10]. The effect of this change is that the filter only operates on the retained output samples from the resampler. Moreover, all the switches are closed at the same clock cycle. Therefore, when they close, the input

signal delivered to the filter on the top path is the current sample. And for the following path, this signal corresponds to the previous sample. The combination of the delays and the resamplers is replaced by a commutator that delivers successive samples to the successive *M*-path filter, as seen in Figure [A.14].

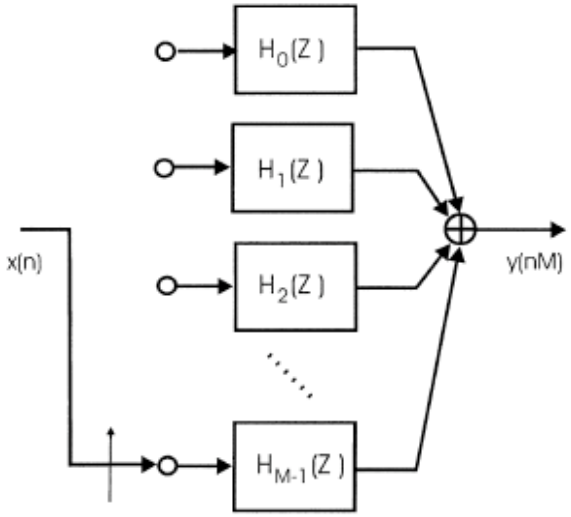


Figure A.14: *M-path filter with delays and input resamplers replaced by the input commutator [2]*

The last step of this change is the replacement of Z^{-1} by $Z^{-1}e^{j\frac{2\pi k}{M}}$ or Z^{-M} by $Z^{-M}e^{j\frac{2\pi k}{M}}$ to satisfy the constraint developed in Figure [A.8]. The complex scalar $e^{jr\frac{2\pi k}{M}}$ attached to each path of the *M*-path filter is placed after the down sampled path filter segments $H_r(Z)$, as shown in Figure [A.15]. In the formula (A.13), the phase rotators are inserted and the new result is shown in (A.16).

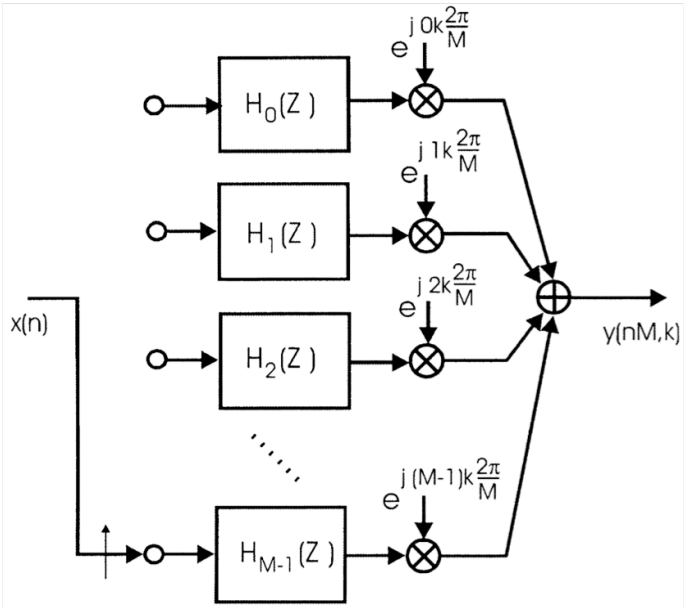


Figure A.15: *M-path filter with commutator, down-converter [2]*

$$H(Ze^{-j(\frac{2\pi}{M})k}) = \sum_{r=0}^{M-1} Z^{-r} e^{j(\frac{2\pi}{M})rk} H_r(Z) \tag{A.16}$$

The computation of the sum $y(nM,k)$ in Figure [A.15] is presented in (A.17). The argument nM represents the down sampling operation. This argument increments through the time index, delivering every M^{th} sample of the original series. The variable $y_r(nM)$ is the nM^{th} sample from the filter and $y(nM,k)$ is the nM^{th} time sample of the time series from the k^{th} center frequency. Another notification on this formula is that the sum is a Discrete Fourier Transform (DFT) of the M^{th} path outputs.

$$y(nM,k) = \sum_{r=0}^{M-1} y_r(nM) e^{j(\frac{2\pi}{M})rk} \tag{A.17}$$

However, seeing deeply the operation of down sampling, it causes the M-to-1 spectral folding, translating the M-multiples of the output sample to baseband. The alias terms in each M-path filters have a unique phase profiles due to their distinct center frequencies and the time offsets that are the input delays in Figure [A.10]. Each of the aliased center frequency has a phase shift shown in (A.18).

$$\phi(r,k) = \omega_k \Delta T_r = 2\pi \frac{f_s}{M} kr T_s = 2\pi \frac{f_s}{M} kr \frac{1}{f_s} = \frac{2\pi}{M} kr \tag{A.18}$$

Examining (A.17), the phase shifters of the DFT perform phase coherent summation and the alias with the particular matching phase profile as shown above in (A.18).

The inputs of the M-path filter are not narrow band enough to delete the undesired spectral contributions. To separate wide-bands signal with the unique phase profiles (described before), an operation of time delay must be performed. The M-path filters supply the required time delays. The M-path filters behave like all pass filters, with in the channel bandwidth, equal ripple approximation to unity gain and the set of linear phase shifts, providing the wanted time delays.

Another perspective is that the phase rotators following the filters perform phase alignment of the band center for each aliased spectral band while the polyphase filters perform the required differential phase shift across these same channel bandwidths. When the polyphase filter is performing operations (down-conversion, down sampling) on a single channel, the phase rotators are implemented as external complex products. After these filter operations, a set of phase rotators is applied to the filter outputs and summed to form each channel output. On the other hand, if the number of channels is large (on the order of $\log_2(N)$), and as the phase rotators following the polyphase filter stages are the same as the phase rotators of a DFT, the DFT can be performed instead of applied a set of phase rotators. Furthermore, the Fast Fourier Transform (FFT) can compute DFT efficiency.

To summarize this section, the channelizer using polyphase filter banks is composed of:

- The commutator performs an input sample rate reduction by commutating successive input samples to selected paths of the *M*-path filter. However, it causes some spectral regions residing at multiples of the output sample rate to alias to baseband. Therefore, polyphase filter bank is implemented to obtain the desired result.
- Polyphase filters perform down sampling and down-conversion's operations.

Appendix B: DSP Architecture

B.1 Overview

The purpose of this appendix is to describe the architecture of the DSP platform used to implement the algorithm describing the application of this project. First of all, a general description is done about the TMS320C6713 platform. Then, the software used for the project is presented. Description of data flow development and possible optimizations are described.

B.2 The TMS320C6713

The TMS320C6713 has been chosen for our application because it has a high performance architecture using a VLIW (Very Long Instruction Word) CPU (Central Processing Unit). Furthermore, the TMS320C67x can process on floating point whereas the previous generation (TMS320C62x) processed only on fixed point. Theses specifications make that this platform is a good choice for multichannel and multifunction applications [14].

The architecture of the TMS320C6713 is composed of three main parts: CPU, memory and peripherals. The different parts are linked together by buses (data bus and address bus). The block diagram of this architecture is presented in Figure [B.1].

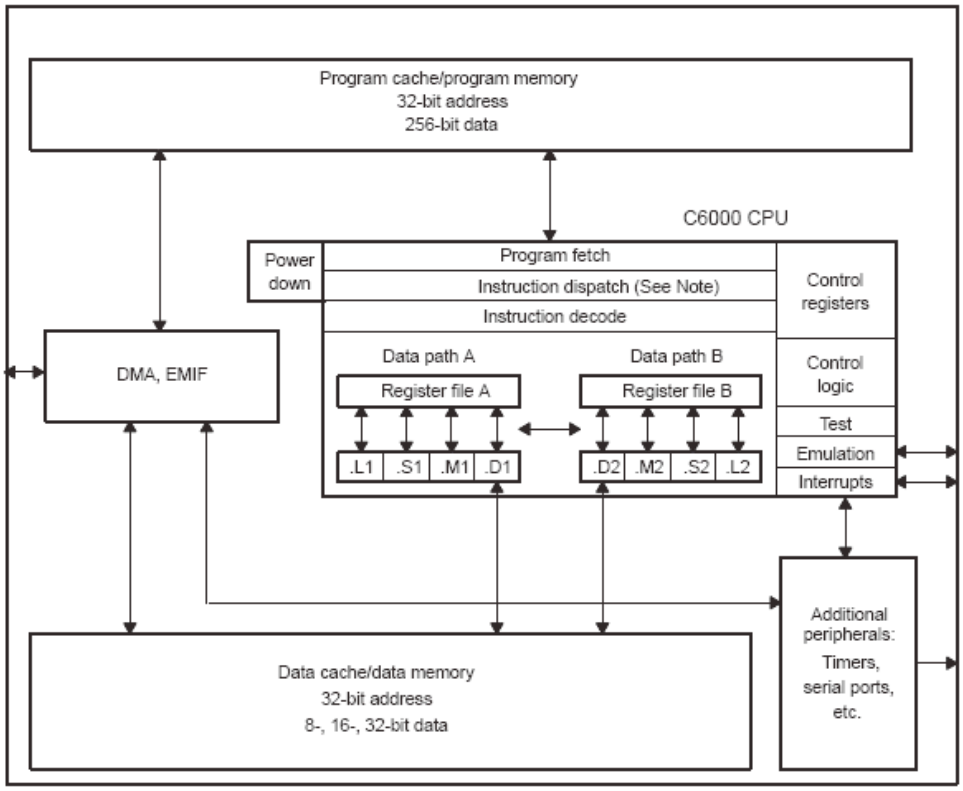


Figure B.1: *Block diagram of the TMS320C6713 [17]*

CPU Description:

The platform is operating at 225 MHz. It can be deliver up to 1350 million of operations per second (MFLOPS) and 1800 million instructions per second (MIPS). The CPU core is composed of two data path with four functional units each (Logic, Shift, Multiply and Data) and a register file. The representation of both of the data paths is shown in Figure [B.2].

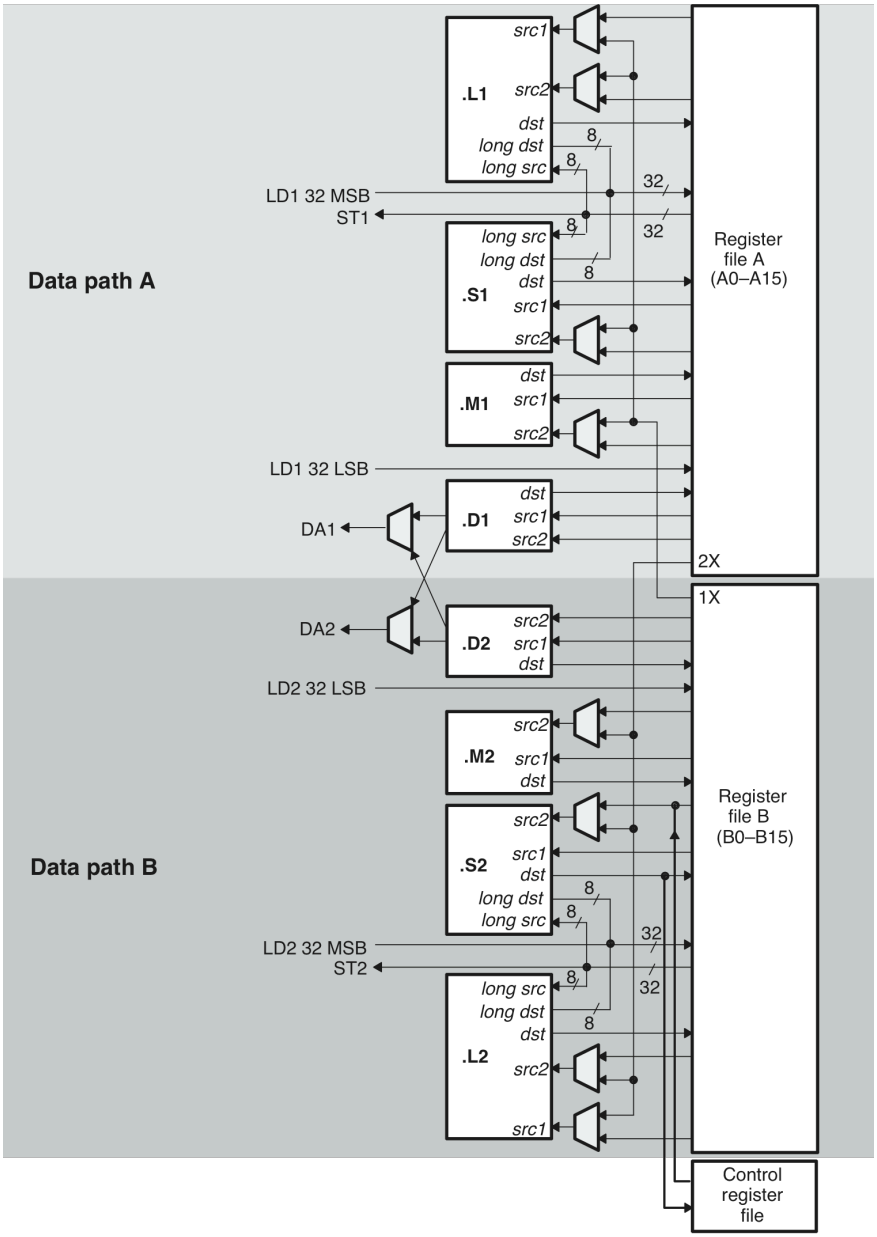


Figure B.2: *Representation of the data path inside the TMS320C67x [14]*

There are 32 32-bits wide registers. These registers can support 32 or 40 bits wide data. For the 40 bits, 2 registers are used: in the first one are stored the 32 LSBs (even register) whereas the 8 MSBs bits remaining are stored in the LSBs of the following register (always odd register). This association also is very useful to store

floating values (64 bits wide). Among the eight functional units, six of them (L1, S1, M1, L2, S2, M2) execute floating points instructions.

All the units have a single data bus connected to registers. Each data path contains one multiplier, three ALUs (Arithmetic and logical Unit) and one register file mentioned above. Four 32 bits paths (LD1 and LD2) allow loading data from the memory to the registers simultaneously. The data-address paths (DA1 and DA2) allow data addresses from the registers to store data to the memory (ST1 and ST2).

The Program Fetch, Instruction Dispatch and Instruction Decode, also present in the CPU core, deliver up to eight 32 bits instruction (256 bits wide) from the memory to the functional units per clock cycle.

The Control Registers perform linear or circular addressing (Addressing Mode Register), control status bits (Control Status Register), command interrupts thanks to Interrupt Clear Register...

Memory Description:

It exists two-level cache inside the platform: the first level is divided into 2 parts: the L1P for the program is a 4 K-Byte direct-mapped cache whereas the L1D for the data is a 2-way set-associative cache with the same memory space (4 K-Byte).

The second level is a 256 K-Byte shared in 2 parts; 64 K-Bytes can be configured as mapped memory, cache or unified cache /mapped RAM. The remaining free space serves as mapped SRAM.

Peripherals Description:

The EDMA (Enhanced Direct Access Memory) allows movements from or to memory, peripherals or external devices without the intervention of the CPU. These movements can be read or write transfer data, frame or block transfer…the EDMA has 16 independent channels, allowing 16 different contexts for operation.

The Host-Port Interface (HPI) is a 16 bit wide parallel port where a host processor can be plug. The HPI can access to the memory or the peripherals and functions as a master to the interface. A control register is used to configure the host as an interface.

The EMIF (External Memory Interface) can be connected to memory like asynchronous (SRAM, EPROM, flash) or synchronous (SBSRAM, SDRAM) devices. The EMIF allows addressing 512 M-Bytes external memory space [16].

B.3 Code Composer Studio

Code Compose Studio is the software used to code, load and run the program on DSP applications. It delivers all of the hosts tools and runtime software support for TMS320 DSP and multimedia applications on mobiles phones based real-time embedded applications. It includes C/C++ compiler, debugger and optimizations tools

(developed in chapter [5]), linker, real-time analysis… the version uses for this project is 2.20.05.

Development

Figure [B.3] shows the code development flow on CCS. This flow consists of four phases. The first three phases focus on the optimizations whereas the fourth one includes linear assembly code.

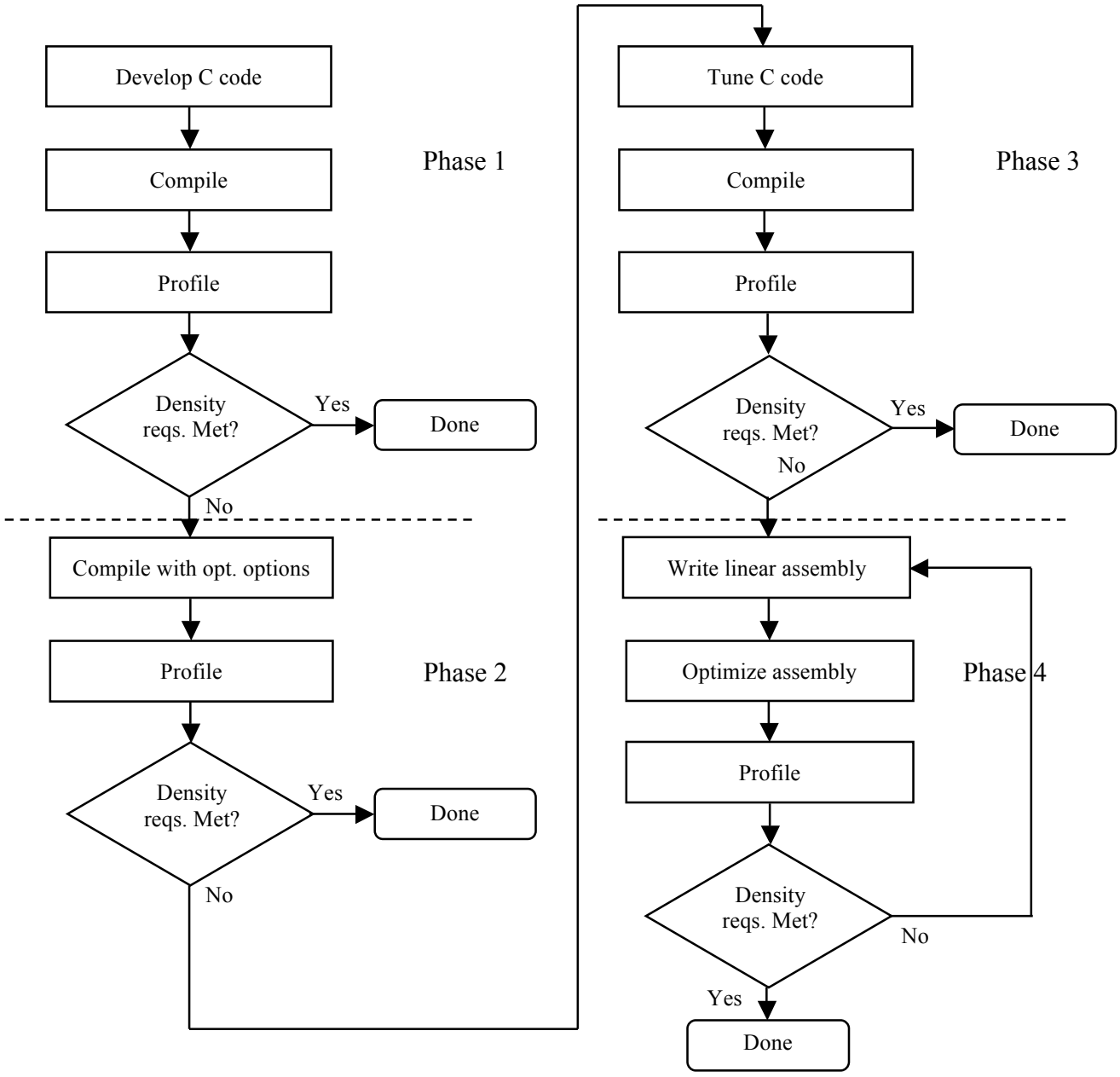


Figure B.3: *code development flow on CCS composed of 4 phases.*

Phase 1 compile and profile the baseline C. The C source file describes the application of the project.

Then, phase 2 involves some optimizations options for the compilation. Usually, the code is improved in terms of code size and execution time. Four optimizations, developed in following chapters, can be performed.

During the phase 3, different techniques are used to tune the C code for better performance. The goal is to allow the compiler to schedule some instructions in parallel.

Finally, the last phase is needed if the performance requirements are not met yet, especially after the tuning phase.

Compiling optimizations

The C compiler on Code Composer Studio can perform different optimizations. The compiler, usually composed of parser (to check correct syntax of source file) and code generator (to generate code in assembly), integrates an optimizer to run faster the code on the platform. The Figure [B.4] illustrates the execution flow inside the compiler.

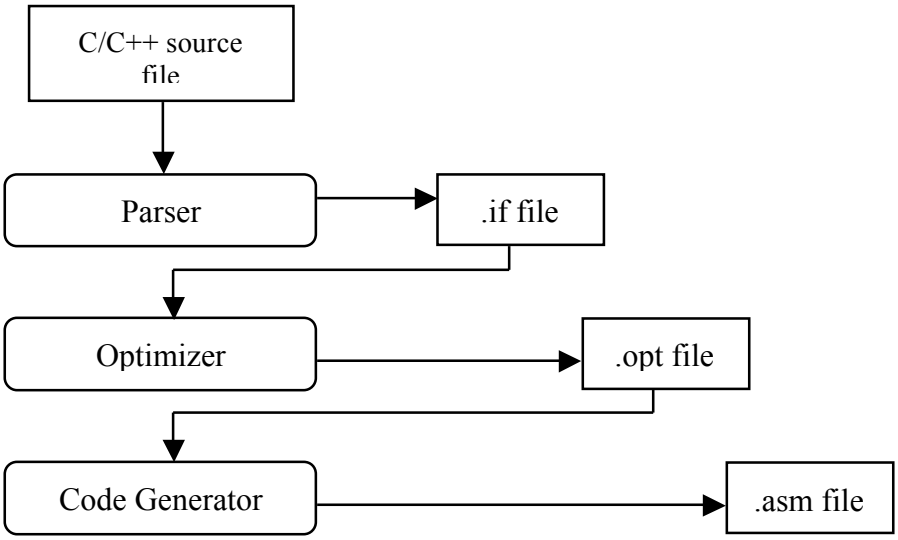


Figure B.4: *execution flow of the compiler. Optimizer block is inserted between parser and code generator to improve the execution of the code on the DSP.*

It exists four various optimizations, which do not optimize the code in the same way. These optimizations are specified with –O*n* where n can take the values 0, 1, 2 and 3. *n* values represent the level of optimization. Here it is a list of some optimizations applied according to the level of optimization:

Optimization –O0 (register):
Elimination of unused code
Simplification of expressions and statements
Allocation of variables to registers…

Optimization –O1 (local):
Performs all –O0 optimizations

Eliminate local common expressions…

Optimization –O2 (function):
Performs all –O1 optimizations
Performs software pipelining, loop optimizations
Converts array references to incremented pointer form…

Optimization –O3 (file):
Performs all –O2 optimizations
Reorders function declarations…

These various compiling optimizations are applied during the implementation. Results are in Chapter [6].