DESIGN & IMPLEMENTATION OF FPGA-BASED

MULTI-STANDARD SOFTWARE RADIO RECEIVER

E-STUDYBOARD
AALBORG UNIVERSITY
GROUP ASPI-1044
MASTERS' THESIS, JUNE 2007

Aalborg University
E-Studyboard
Applied Signal Processing and Implementation 10th Semester

TITLE:

Design & Implementation of FPGA-based Multi-standard Software Radio Receiver

ABSTRACT:

The objective of the project was to design and implement FPGA-based Multi-standard Software Radio Receiver. WLAN and UMTS are taken as case study. Xilinx FPGA Virtex-IV is the target platform. Bandpass sampling technique at 840MHz is used to alias the combined band of WLAN and UMTS. The WLAN and UMTS channels are required at baseband with the sampling rate of 20MHz and 61.44MHz respectively. Bandpass filters are used to separate the UMTS and WLAN bands. In the channelization process, in contrast to conventional channelizer, polyphase channelizer is used. In the simulations, optimal-method-based FIR filters are used. In polyphase channelizers, the prototype filter for WLAN has 50 taps, partitioned into 5 polyphase sub-filters whereas the prototype filter for UMTS has 2520 taps, partitioned into 210 polyphase sub-filters. The received channels at baseband has 50dB of dynamic range. In the implementation, different structures for polyphase channelizer are considered (such as) standard structure, symmetric-property based structure, Adder shared structure and serial polyphase structure with serial and parallel MAC. Serial polyphase structure with parallel MAC is selected. In the individual sub-filter implementation, different implementation structures are considered. These being Parallel Multipliers and Accumulate, Bit systolic array, Distributed Arithmetic (DA), Fast FIR, Frequency domain filtering and Multiplier-Less filtering techniques. An analysis based on the approximations for the area requirements for multipliers, adders and registers for these structures is performed. For 16-tap filter, the structures for Parallel-Multiply and accumulate, DA, Fast FIR and Frequency domain filtering require 2896 (without adders), 3072, 4064, and 5572 slices, respectively. The DA is found to be suitable for the implementation due to being resource efficient. Polyphase sub-filter is implemented with Distributed Arithmetic structure or with Xilinx-DSP48 slices for improved performance.

PROJECT PERIOD:
P10
2nd February - 7th June 2007

PROJECT GROUP:
ASPI10-2007 - Gr. 1044

GROUP MEMBERS:
Mehmood-Ur-Rehman Awan
Muhammad Mahtab Alam

PROJECT SUPERVISORS:
Peter Koch
Nastaran Behjou

Publications: 5

Total number of pages: 138

PREFACE

This report is written by group 1044, studying the master specialization Applied Signal Processing and Implementation (ASPI) at Aalborg University. The report serves as documentation of the Thesis work at the 10th semester.

The introduction provides a rationale for the project, and leads to a definition of the problem. The problem is specified in the problem statement, which leads to a definition of the modules necessary for the project. The functionality of the algorithms are examined in the Software Radio System Design, WLAN and UMTS Channelizers, and Simulations chapters, and the mapping of the algorithm to the architecture is conducted in the Algorithm-to-Architecture-Mapping chapter.

The appendices expand on some of the details in the project, where it is not strictly necessary in the main report.

All chapters, sections, figures, and tables have assigned numbers, and the reference will make clear, whether the reference is made for figures or tables. The equations are assigned numbers, and the reference for an equation is shown in parentheses. References to the bibliography are done using Harvard citation style, e.g. [Haykin, 2002, p. 205]

The enclosed CD-ROM contains all the relevant MatLab scripts, and VHDL codes used in the project.

Muhammad Mahtab Alam Mehmood-Ur-Rehman Awan

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Communication is a major part of our everyday life. We communicate through telephones, Emails, Internet chat rooms, in writing and of course face to face. From a fundamental point of view communication can be seen as *transmission of information from one point to another*. The simplified communication system is shown in Figure 1.1. This contains three basic elements namely, *transmitter (information source, modulator), channel* and *receiver (demodulator, destination information)*. The purpose for the transmitter is to convert the message's signal(base-band) into a radio frequency (RF) signal which can be sent through the channel. The task for the receiver is to reconstruct the base-band signal and present it for the user.

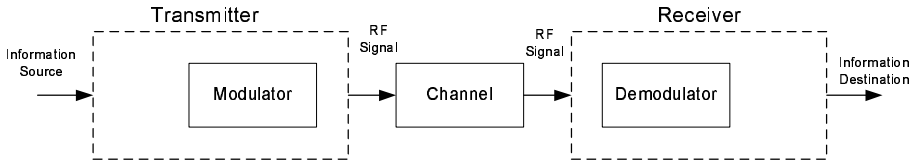


Figure 1.1: The simplified communication system with three basic building blocks i.e transmitter, channel and receiver

1.1 Wireless Radio's

This section explains the evolution in the architectures of transmitters and receivers of the wireless radio's. The wireless radios for the cellular mobile communication system have passed through several generations. The tradional hetrodyne architecture is considered as a first generation where only the base-band processing in done in digital domain. The Figure 1.2 is the first generation of RF architecture of N-channel receiver. The synthesizer consists of dual stage down converters. In the first stage the radio frequency (RF) signal is down converted to band limited intermediate frequency(IF), and in the second stage the output of each IF filter is again down converted to base-band by matched quadrature mixers followed by matched base-band filters that perform the final bandwidth control. Each sub receivers is then converted into digital domain where the output of Analog to Digital Converter(ADC) is processed by DSP engines which perform the required base-band processing i.e. synchronization, equalization, demodulation, detection and decoding.

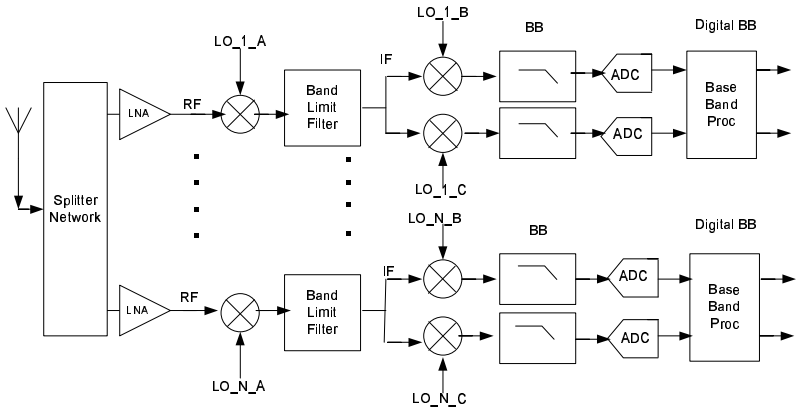


Figure 1.2: A traditional radio receiver with multiple stage down conversion. There could possible be more than one down-conversion in RF with single stage IF down-converter. Only the baseband processing is done in digital domain.

The problem with this type of architecture is that they have amplitude and phase imbalance which results in cross-talk between the narrow band channels due to ageing(time, temperature) of analog components of the quadrature down-converters, so each imbalance related spectral image must be lower then the desired spectral term, which is difficult to sustain over time and temperature. So the need to acheive the extreme levels of I/Q balance brings the second gereration of radio's which is shown in the Figure 1.3. In the second generation, the second stage(IF) down conversion is digitized, so for each sub-channel a digital down converter and a LPF is required. The digital conversion at IF brings more control on the imbalance by manipulating the number of bits involved in the arithematic operation. The precision of coefficients used in the filtering process sets an upper bound to spectral artifacts levels at $-5dB/bit$, so the $12bit$ADC will have an image level below $-60dBs$. Thus DSP based complex down conversion brings two advantages. First, the spectral images are controlled to be below the quantization noise floor of the ADC involved in the conversion process and second, the digital filter following and preceeding the mixers are designed to have linear phase characteristics [Fredric J. Harris and Rice, 2003]. This second geneation of wireless radio is a reliazable verion of software radio and is called 'Software Defined Radio'.

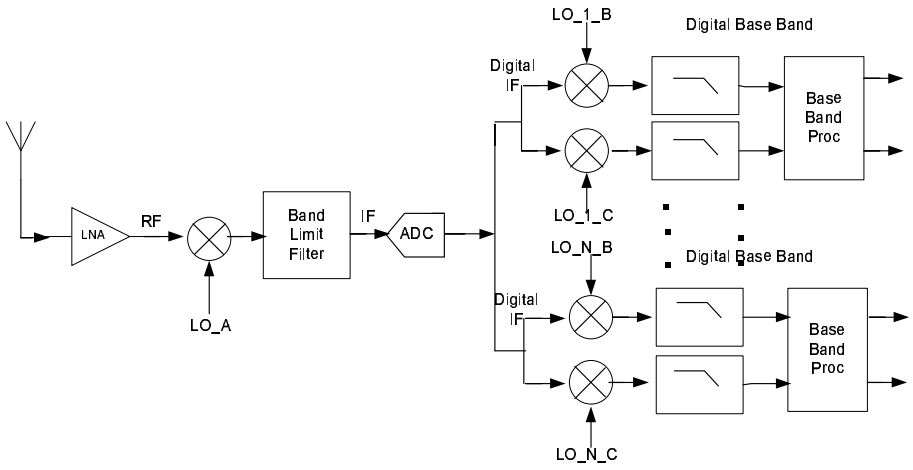


Figure 1.3: The second generation of radio receiver, in which the IF stage becomes completely in digital domain. The second generation bring control on the I/Q imbalance created by hardware oscillators.

1.1.1 Software Defined Radio

A software-defined radio (SDR) system is a radio communication system which can tune to any frequency band and receive any modulation across a large frequency spectrum by means of a programmable hardware which is controlled by software. An ideal software radio(ISR) samples the signal at RF, just after the antenna, whereas the realizable version of the software radio is the one that solve the problem of sampling the RF signal (according to minimum nyquits criteria, i.e. to sample at twice the maximum frequency of the incomming signal), by using a mixer and a reference oscillator to heterodyne the radio signal to a lower frequency (Intermediate frequency), as described in the second generation of cellular radio's, shown in Figure 1.3. In the Section 1.1, the architectural level significance of SDR is described, but there are lots of system level issues in the wireless communication industry which embarks the essence and motivation for the Software Defined Radio's, which are:

- Commercial wireless network standards are continuously evolving from 2G to 2.5G/3G and then further onto 4G. Each generation of networks differ significantly in link-layer protocol standards causing problems to subscribers, wireless network operators and equipment vendors. Subscribers are forced to buy new handsets whenever a new generation of network standards is deployed. Wireless network operators face problems during migration of the network from one generation to next due to presence of large number of subscribers using legacy handsets that may be incompatible with newer generation network.

- The air interface and link-layer protocols differ across various geographies (for e.g., European wireless networks are predominantly GSM/TDMA based while in USA the wireless networks are predominantly IS-95/CDMA2000 CDMA based). This problem has inhibited the deployment of global roaming facilities causing great inconvenience to subscribers who travel frequently from one continent to another. Handset vendors face problems in building viable multi-mode handsets due to high cost and bulky nature of such handsets.

- Wireless network operators face deployment issues while rolling-out new services/features to realize new revenue-streams since this may require large-scale customizations on subscribers' handsets.

SDR technology promises to solve these problems by implementing the radio functionality as software modules running on a generic hardware platform. Further, multiple software modules implementing different standards can be present in the radio system. The software modules that implement new services/features can be downloaded over-the-air onto the handsets. This kind of flexibility offered by SDR systems helps in dealing with problems due to differing standards and issues related to deployment of new services/features. There are lot of advantages of the full-downloadable type software radio, the system can be changed on demand by changing software, there are many gains for not only operators and service providers, but also for government and commercial customers. such as, Global roaming services, bug fixed without the need to recall the product and new services can be added without changing the terminals [Ramjee Prasad, 2002]. The most promising application of SDR is the application of cognitive radio (CR). The radio spectrum becomes more and more sparse, making it an extensive task to allocate a new spectrum for new services. The radio that is aware of its environment, internal state, and its location, then it

make a dicision about its operating behaviour based on that information [Cook, 2006].

1.2 Problem Description

The increasing trend toward a single device integrating several features and capabilities encourage the companies and research centers to develop the multi-standard multi-mode "all-in-one" front-ends. A scenario of multi-standard multi-mode is shown in figure 1.6. High level of integration and small size are precedence objectives in these types of mobile applications. In order to acheive those objectives it is feasible to move most of the data processing to digital domain through shifting the digital to analogue converter (ADC) as close to antenna as possible. Therefore the idea in this project is to use an efficient technique called bandpass sampling which can directly sample the RF signal (after LNA) and all the signal processing to be done in digital domain as shown in Figure 1.4. It will overcome the problems of 2nd generation radios, being sustaining the gain and phase imbalance of analog components. The 3G (UMTS,CDMA2000 etc) wireless systems impose severe requirements on level of I/Q balance. The need to acheive the extreme levels of I/Q balance motivates us to perform the complex conversion process in DSP domain [Fredric J. Harris and Rice, 2003]. Thus by processing the digital data, the unique functionalities of each standard can be set in the digital signal processing programmable parts by employing the concept of software-defined radios (SDR). This enables the front-end to process numerous signals without the traditional hardware limitations.

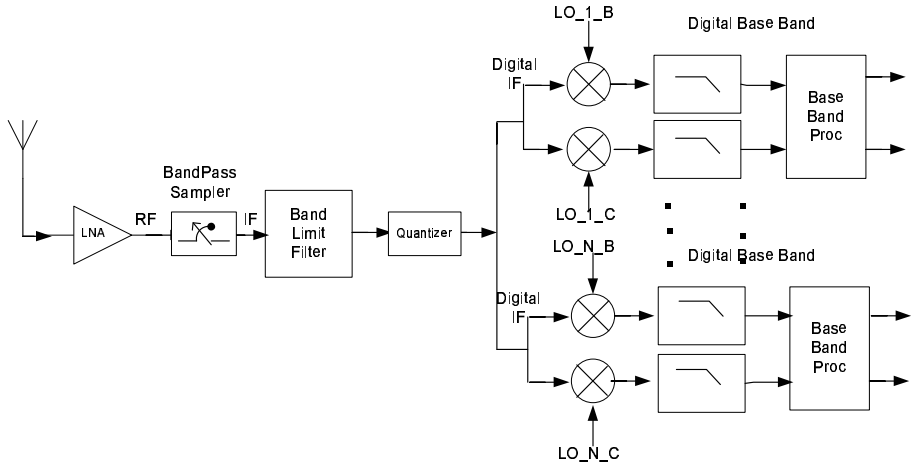


Figure 1.4: This is the proposed architecture of the software radios, where sampling is done at RF just after the LNA which is the only analog component in this architecture.

The scope of this project is to implement an algorithm to perform this multiple reception of standards and process the data in intermediate frequencies and perform all the required reception functionalities such as decimation and downconversion. The development and implementation of the system depend on several things: application requirements, algorithmic capabilities, hardware limitations, etc. In order to describe their dependencies a model named A-cube (A^3) is introduced[1]

[1]This model is used internally at AAU and unfortunately, there exist no literature to document the model

as shown in Figure 1.9. This model deals with two major parts, one being mapping from application to algorithm (algorithm development) and second mapping from algorithm to architecture (implementation). It is an iterative process, which means that we can go back and forth to tune the parameters of application, algorithm and architecture.

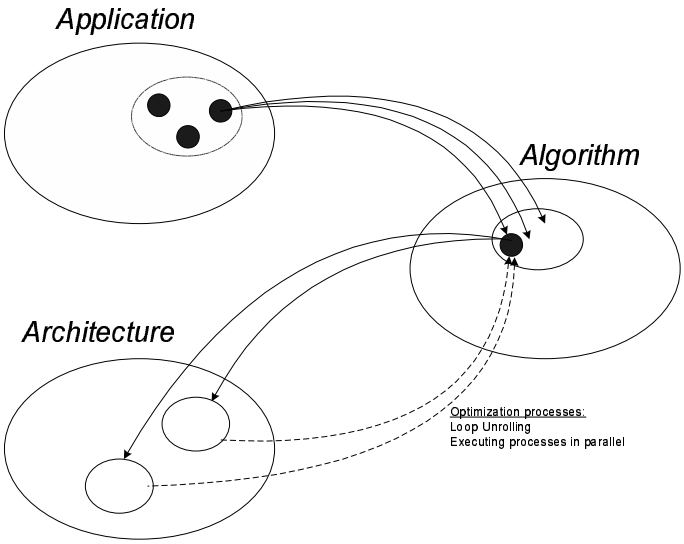


Figure 1.5: The A^3 model, used for illustrating the mapping from the application to the algorithm, and the mapping from algorithm to architecture. It is an iterative process.

1.2.1 Application

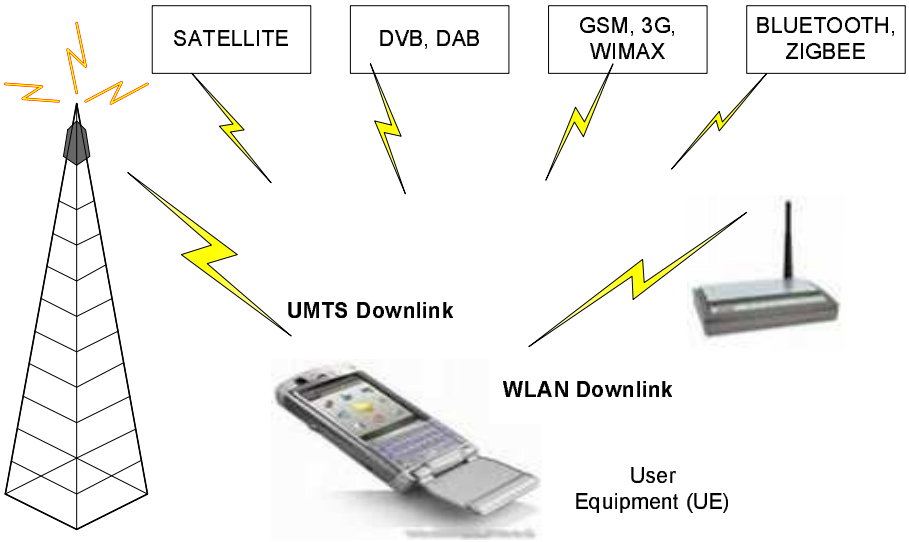


Figure 1.6: A scenario of multi-standard multi-mode "all-in-one" front-ends user equipment. It highlights the user equipment capbale of receiving two standards i.e. UMTS and WLAN.

In the project, a multi-standard software radio reciever is considered. One of the main challanges is the coexistance of several standards in one user equipment(UE), since the chances for channels interferance among the standards is very high [Behjou Nastaran, 2006]. Therefore, out of the multi-standards i.e. GPS, GSM, Bluetooth, zigbee, satellite communication, the application is limited to a case study where two standards being UMTS and WLAN are considered which are shown in Figure 1.6, This is a case study which actually fits to the cellular systems where the possible scenario could be that a doctor is talking with a patient on the mobile phone(UMTS) and at the same time it is down-loading the histroy of that patient(WLAN). Some of the specifications of these standards are shown in Table 1.1 [Behjou Nastaran, 2006].

UMTS and WLAN Specifications for UE		
	UMTS	IEEE 802.11g
Duplexing	FDD	TDD
Frequency Band	1920 - 1980 MHz: UL 2110 - 2170 MHz: DL	2.4 - 2.4835 GHz
Receiver Sensitivity	-117 dBm	-82 to -65 dBm
Transmitter Power Level	24 dBm (Class 3)	20 dBm (Europe)
Channel Bandwidth	3.84 MHz	16.6 MHz
Number of non-overlapping channels	12	3

Table 1.1: Some specifications of UMTS and WLAN standards [Behjou Nastaran, 2006]

In the scenario of the project, the receiver must be able to receive the signals coming in all different channels of these two standards. As mentioned in Table 1.1, the UMTS and WLAN signal bands have 12 and 3 non-overlapping channels respectively. Thus, the target device must be tune-able to serve to all different combinations of the two signals (36 different frequency combinations). we have to recieve only one of the possible combination out of them at a time.

1.2.2 Algorithm

The system level block diagram is shown in figure 4.2. The band-select filter is required to initially select the whole band of information that contains both the standards i.e (UMTS and WLAN) and then seperate them through a channel which have bandpass filters, and the block of channelizer which down convert and down-sample the IF signal to desired baseband along with the required channels for each standard.

The algorithmic development for the design analysis of multiple standards have few phases.

- First, the idea is to use the aliases of the original signal at lower frequency(IF) while sample it at higher frequency(RF). So the technique called 'bandpass sampling' is used to acheive the non-overlapped aliases of the WLAN and UMTS channels.
- Secondly, to design the digital band-select and bandpass filters at intermediate frequency (IF)(within the specification defined by first phase) to extract the relevent band within each standards.
- Finally, the channelizer is required to down-convert the IF signals to baseband and to down-sample to the desired sampling frequency for both the UMTS and WLAN.

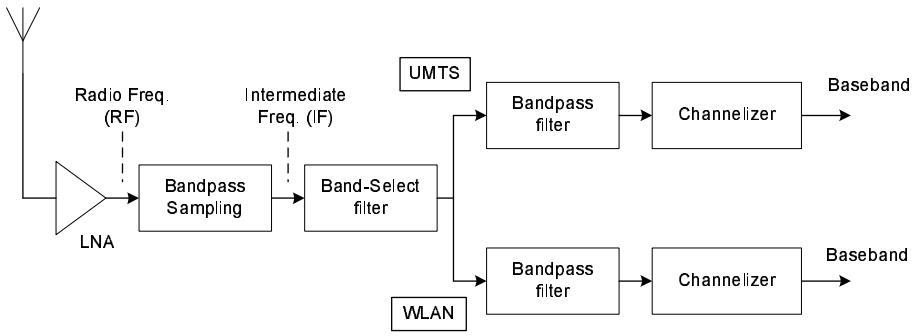


Figure 1.7: This is a complete block diagram of the system. After receiving the signal at antenna it is passed through the low noise amplifier(LNA) which boost-up the signal (adding a low noise). The next block is of bandpass sampling which samples the input at RF and brings the information down to IF. The band-select filter then select the complete band of interest followed by two channels which have individual BPF to seperate the multiple information, in this case it is two i.e. UMTS and WLAN. followed by the block of channelizer which further down convert by down-sampling and filter the IF signal.

1.2.3 Architecture

The RF (Radio Frequency) front-end is supposed to receive two mentioned signals(UMTS,WLAN) at the same time. The signals are adjusted, filtered and consequently downconverted from RF to lower intermediate frequencies in the RF front-end design. After digitization, the signals are passed through the digital signal processing block. The perspective of the project is to employ the state of the art technology such as moderen DSPs or FPGA to process the data in intermediate frequencies and perform all the required reception functionalities such as decimation and down-conversion.

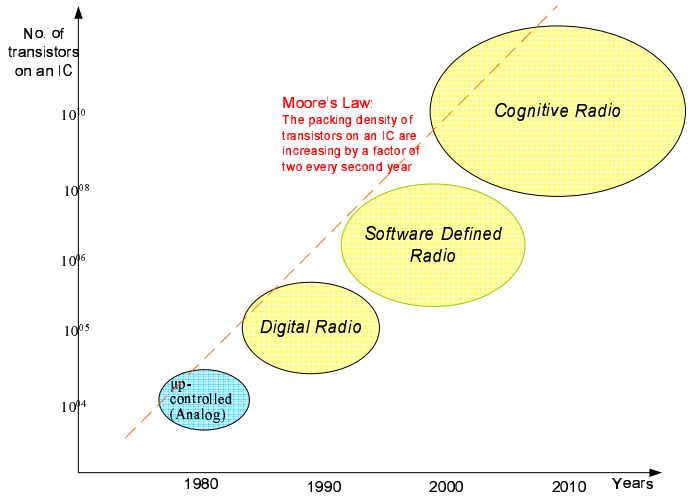


Figure 1.8: Gorden E. Moore Law: The number of transsitors are increasing by a factor of 2 after every 18 to 24 months, due to increasing demand of applications. The complexity of the overall systems are increasing but with the demands of minimum cost, minimum size, faster execution time and least power dissipation.

The selection of the hardware architecture is not easy for SDR based applications in an era where the number of transistors in an integrated circuit are increasing by a factor of two every second year(Moor's Law) as shown in the figure 1.8. As signal processing tasks (the algorithms) are

getting more and more complex which is at the same time putting high requirements on the technologies platform with increasing demand of the MIPS(million of instructions per second). So the software solution for SDR makes it possible to make the transition from dedicated, single-purpose hardware (ASICs, etc.) to highly versatile general-purpose hardware such as FPGAs and DSPs, and even to general-purpose processors whose functionality is defined solely by their software configuration. This in turn paves the way for high-volume/low-cost production, making it financially viable to embed autonomous radio communication devices in a wide range of new kinds of devices and applications [CSD, 2007]. The DSP is a specialized microprocessor optimized for performing multiply and accumulate operations. The DSP has also proven to be inefficient for some tasks, where a customized architecture actually is more suitable. One solution has been the application specific IC (ASIC), which is the least flexible, but most optimal solution with regard to execution time, power dissipation and cost. The cost factor is however, only low for a very large number of units, as the development of an ASIC is very expensive and time consuming. As the applications that request SDR based technology continue to require systems that can be reconfigured fast, as well as provide a massive amount of processing power, the need for powerful reconfigurable architectures emerges. One solution for this is the use of field programmable gate arrays (FPGAs). FPGAs offer the possibility of programming logic to be more suitable for certain algorithms than the general DSP. This goes especially for algorithms where parallelism can be exploited efficiently, but also more special operations like square root, cosine etc. are very suitable for FPGA implementation. The use of FPGAs in handheld devices might be limited, as the cost and power dissipation are still relatively high compared to an ASIC solution.

The generic A3-model as shown in figure is now modified to fit in this project which is shown in figure 1.9. The focus of application to algorithm mapping is to take into account the various multi-rate filtering techniques or the other techniques that can fulfill the required application of multiple standard software radio. Then mapped those formalized algorithm onto the specified architecture i.e. FPGA or ASICs.

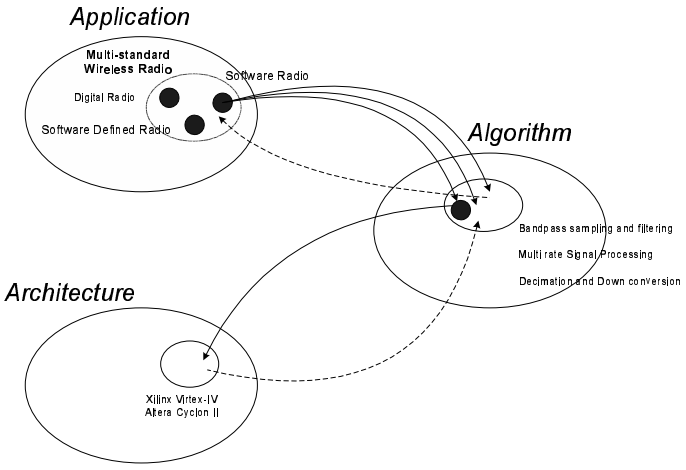


Figure 1.9: The focus of application to algorithm mapping is to take into account the various multi-rate filtering techniques or the other techniques that can fulfill the required application of multiple standard software radio. Then mapped those formalized algorithm onto the specified architecture i.e. FPGA or ASICs.

1.2.4 Problem Definition

Design analysis and implementation of Multi-standard Software Radio Receiver.

CHAPTER 2

SOFTWARE RADIO SYSTEM DESIGN

The principle idea behind the design of a software radio is to place the analog-to-digital and digital-to-analog converters as near the antenna as possible, such that most of the radio functionalities can be implemented on a programmable digital signal processor. One way to achieve this is by direct bandpass sampling of the desired RF signal band to baseband frequency. However, the design of a software radio receiver becomes more complicated when two or more distinct RF signals are to be received [Dennis M. Akos and Caschera, 1999].

In a multi-standard radio receiver design, UMTS and WLAN standards are taken as case study and a receiver is required to receive both these standard simultaneously with same front-end, and downconvert them to baseband separately. The spectral location for UMTS and WLAN standards are shown in Figure 2.1. UMTS has a bandwidth of 60MHz for downlink having 12 channels and WALN has a 84.5MHz of bandwidth having 3 channels. It is required to downsample and to downconvert these channels to baseband.

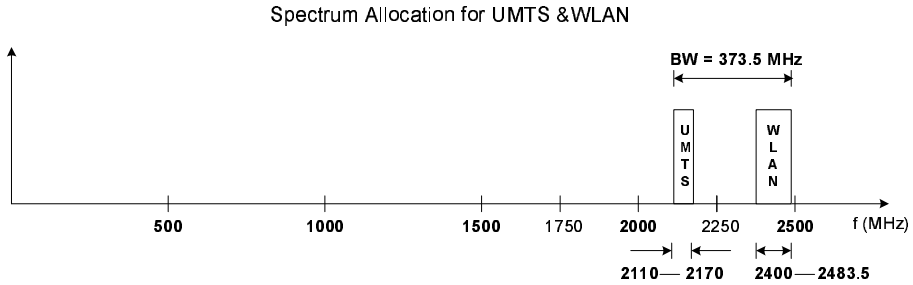


Figure 2.1: Spectrum Allocation for UMTS and WLAN standards. UMTS has a bandwidth of 60MHz for downlink having 12 channels and WALN has a 84.5MHz of bandwidth having 3 channels.

2.1 Downconversion Techniques for Software Radio

Traditionally the superheterodyne architecture has been used extensively for radio systems since it provides a number of advantages such as image rejection and adjacent channel selectivity. Software radio is an enabling technology for future radio transceivers, allowing the realisation of mul-

timode, multiband, and reconfigurable base stations and terminals. Bandpass sampling and direct conversion are two receiver architectures that are suitable for software radios. However, considerable research efforts and breakthroughs in technology are required before the ideal software radio can be realised.

2.1.1 Bandpass sampling Architecture

The sampling of bandpass signals can be carried out at rates lower than conventional lowpass Nyquist sampling, causing intentional aliasing the signal. Bandpass sampling can allow for received signals to be digitized closer to the antenna using manageable sampling rates and hence could be favourable for downconversion in software radios. In this project secnario, the total receiver bandwidth for the UMTS and the WLAN is 373.5MHz, as shown in the Figure 2.1. According to bandpass sampling, the sampling frequency should be twice the signal bandwith rather than twice the maximum frequency component as in the case of Nyquist sampling. So the sampling frequency for the combined band of UMTS and WLAN must be atleast 747MHz to have non-overlap alaises. Today's technology set a limit to achieve such a high sampling rate. Significant improvement in ADC performance is required for sampling at RF.

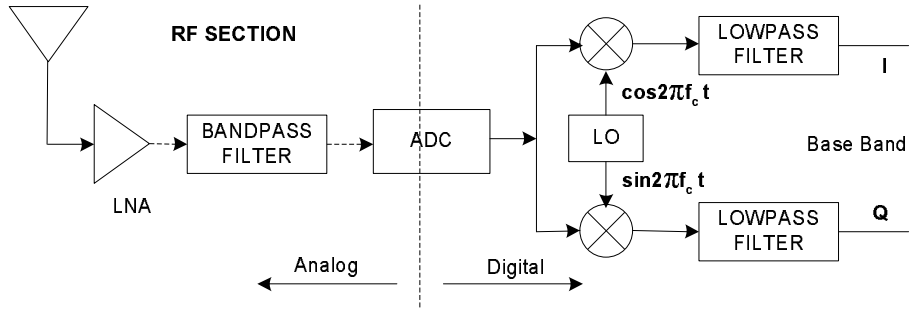


Figure 2.2: Bandpass sampling Architecture of Sofware Defined Radio

As the ADC is moved closer to the antenna, more radio functions can be written in software and embedded on programmable logic. However, ADC performance still is not sufficient enough to perform digitization at RF [Patel and Lane,]. In particular, the input analogue bandwidth, sampling rate, dynamic range and therefore resolution need considerable amounts of improvement if wideband front-ends and sampling at RF are to become in a reality. The performance of DSP must be able to cope with the increased amount of programmable radio functionality as a result of moving the ADC closer to antenna. Schemes using a mixture of DSP and FPGA have been proposed [Patel and Lane,].

2.1.2 Direct Conversion Architecture

Direct conversion, also sometimes called zero-IF, due to the lack of an intermediate frequency, converts the received RF signal direct to baseband. This is particularly attractive for the use in wireless systems, especially in handsets since direct conversion receivers lend themselves more

easily to monolithic integration than heterodyne architectures, since the IF components are replaced by lowpass filters and baseband amplifiers. Direct conversion exhibits immunity to the problem of image since there is no IF [Patel and Lane,]. There are a number of design issues associated with the direct conversion architecture. The most serious problem is DC offset in the baseband, following the mixer. This offset appears in the middle of the downconverted signal spectrum, and may be larger than the signal itself. This phenomenon can be caused by local oscillator leakage and self-mixing [Patel and Lane,].

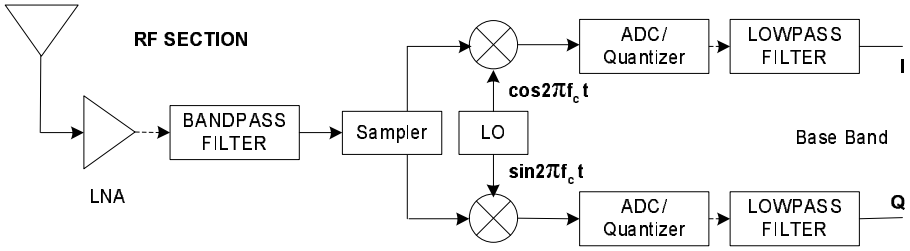


Figure 2.3: Direct conversion Architecture of Sofware Defined Radio

Bandpass sampling allows for the ADC to digitize at RF, providing the ADC is of adequate performance, whereas direct conversion, although consisting of more analogue components, places fewer demands on ADC performance since digitization occurs at baseband.

As mentioned above that by moving the ADC closer to the antenna, more radio functions can be written in software and embedded on programmable logic. Sampling at the antenna is not realistic since some amount of band select and filtering must occur prior to the ADC to minimize adjacent channel issues. However, sampling at the First-IF is practical, yeilding the concept of Direct-IF sampling.

2.1.3 Direct-IF Sampling Architecture

Recent advances in converter technology have allowed data converters to faithfully sample analog signals as high as several hundred MHz. Sample rates need only be as high as twice the signal bandwidth to keep the Nyquist principle. Since most air interface standards are less than a few hundred MHz wide, sample rates in the tens of MHz are required, eliminating the need for extremely fast sample rates in radio design. Thus allowing for low cost digitizers [Brannon,]. A IF-sampling radio receiver is shown in Figure 2.4.

Once digitized, the signal would have to be processed. With a typical sample rate of 20 MHz (for instance), data would stream too fast for even the hottest DSP to do much with in terms of filtering, much less process the data for user information. Therefore, some preprocessing of the data must occur [Brannon,]. With a sample rate of 20 MHz, the data bandwidth would be 10 MHz, much more than is needed for most air interfaces. Therefore, one thing that preprocessing should achieve is to reduce the data bandwidth as well as the data rate. Thus in addition to the ADC (analog-to-digital converter) a DSP preprocessor is required as shown in Figure 2.5.

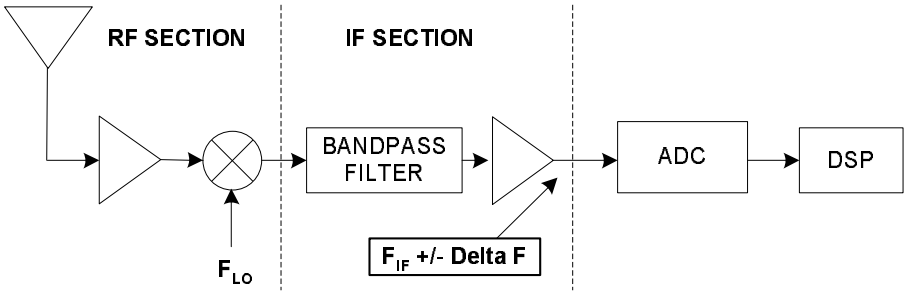


Figure 2.4: Direct IF sampling software Defined Radio

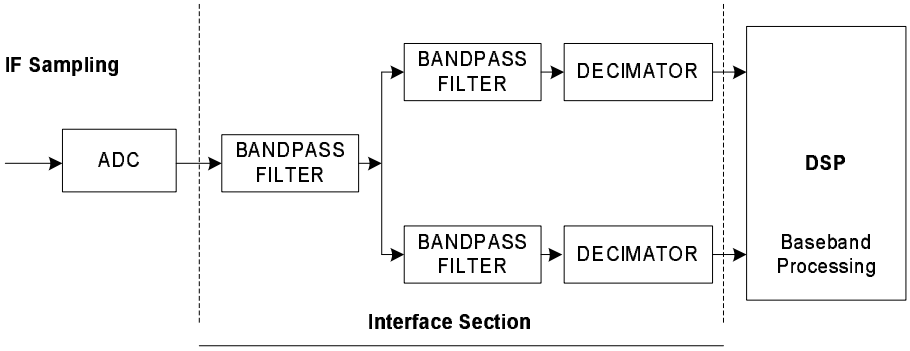


Figure 2.5: Interface circuit required between IF-sampling and baseband processing

2.2 Architecture Selection

In the above section, different architectures have been discussed in terms of their performance and structures. The project aim is to have multi-standard receiver where the ADC is place as close to the antenna as possible. The Direct-IF sampling uses downconversion process prior to ADC conversion. This leaves the other two possible arhitecture i.e bandpass sampling and direct conversion for the consideration. Bandpass sampling architecture does not require additional circuits for downconversion prior to quantization. This leads to all the processing required for bandpass sampling architecture to be implemented on FPGA which can be reconfigured to different radio configuration. Although the choice of ADC becomes more critical, but we will not deal with these issues.

The project focuses on the bandpass sampling architecture. We deal only with the sampled data after the ADC process. It is required to downconvert and downsample the individual channels of UMTS and WLAN standards to baseband.

2.3 Design Process

Bandpass sampling architecture has been selected as discussed in the previous section. This leads to the selection of sampling frequency which is critical. A sampling frequncy of 676MHz is taken as a start [Behjou Nastaran, 2006]. This frequency is below the required sampling of 747MHz, in order to have non-overlap aliases. In the combined spectrum for UMTS and WLAN, there

is an unsed spectrum between them. By having the overlap aliases in this unused spectrum, the sampling frequency can be decreased. This is the case for the sampling frequency at 676MHz as shown in Figure 2.6.

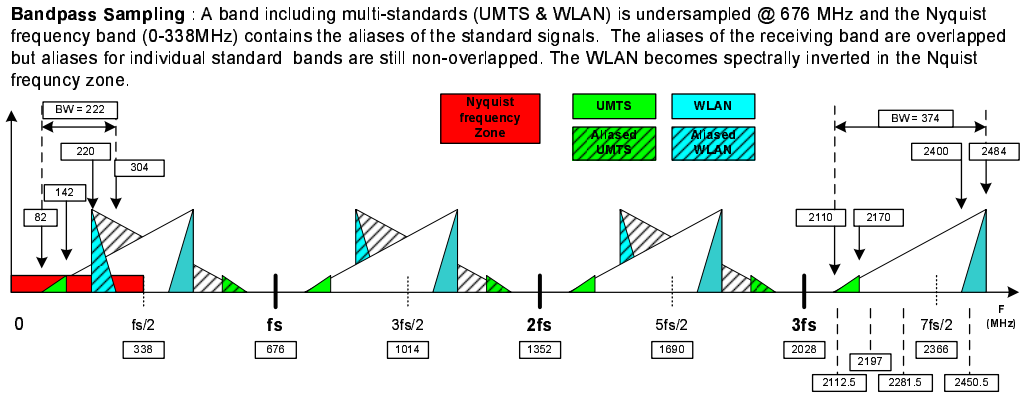


Figure 2.6: Combined spectrum of UMTS and WLAN is bandpass sampled at 676 MHz. 12 channels of UMTS are required to downsample from 676MHz to 61.44MHz, and 3 channels of WLAN are required to downsample from 676MHz to 20MHz, along with downconversion to base band.

The combined spectrum is aliased to Nyquist-zone ($f_s/2$) as overlap aliases but the required UMTS and WLAN bands are still non-overlapped. The zoom of spectrum in the Nyquist-zone is shown in the Figure 2.7.

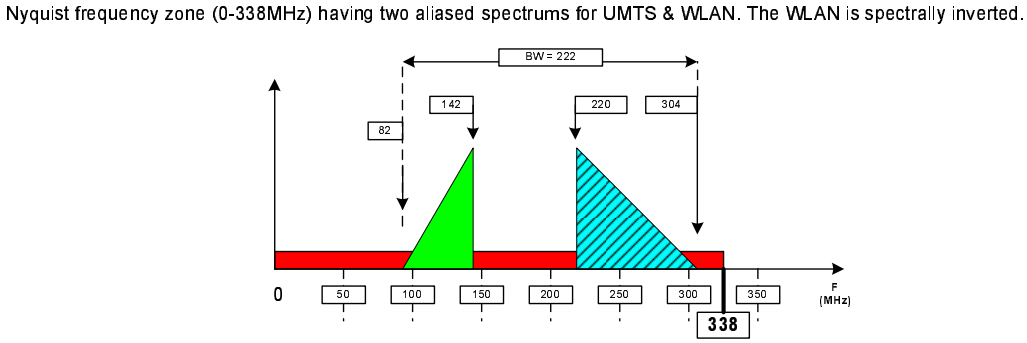


Figure 2.7: The zoom of spectrum in the Nyquist-zone. The resulted aliased signals for UMTS and WLAN lie at (82-142)MHz and (220-304)MHz respectively.

The individual channels (12 UMTS channels and 3 WLAN channels) are shown in Figure 2.8. Each of UMTS channel is 5MHz wide and have 5MHz of spacing between inter-channel carriers, whereas each of the WLAN channel is 24MHz wide and have 30MHz of spacing between inter-channel carriers.

The resulted aliased signals for UMTS and WLAN lie at (82-142)MHz and (220-304)MHz respectively. The goal is to downsample these signals to the desired rate i.e. 20MHz for WLAN and 16×3.84 = 61.44MHz for UMTS. 3.84 is the UMTS bandwidth and the number 16 is the oversampled ratio that can vary as 16, 32, etc. But the number 16 has been taken into account. The required sample rates for UMTS and WLAN are summaried in the Table 2.1.

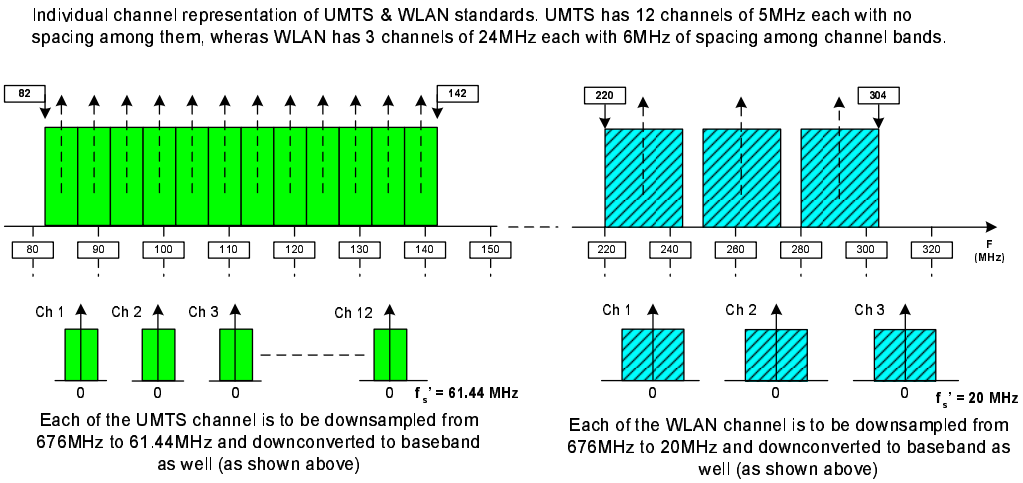


Figure 2.8: Individual channels: (12 UMTS channels and 3 WLAN channels). Each of UMTS channel is 5MHz wide and have 5MHz of spacing between inter-channel carriers, whereas each of the WLAN channel is 24MHz wide and have 30MHz of spacing between inter-channel carriers.

Specifications for UMTS and WLAN sample rates

Standards	Current sampling rate (MHz)	Desired Sampling rate (MHz)
UMTS	676	61.44
WLAN	676	20

Table 2.1: Specifications for UMTS and WLAN sample rates.

In order to extract these channels and downconvert them to baseband at the required rates, channelizers are required, which will be explained in the next section.

2.4 Channelization

A conventional way to do channelization is presented in the Figure 2.9, where each channel is first downconverted to baseband and then downsampled after passing through a lowpass filter.

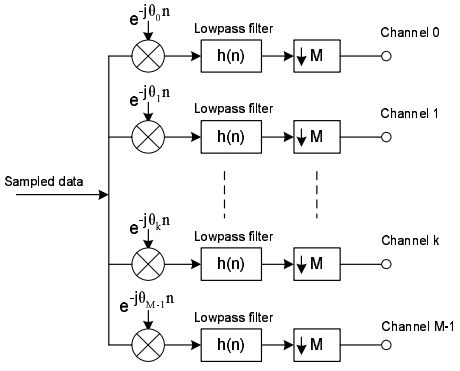


Figure 2.9: Conventional channelization: where each channel is first downconverted to baseband and then downsampled after passing through a lowpass filter.

The digital sample rate conversion uses the techniques such as decimation, interpolation and combination of them to have rate conversion by rational numbers. In a process of decimation by M, $M-1$ samples are discarded and every Mth sample is taken in to account. This results in the spectral expansion, so the bandwith of the signal is first reduced before the decimation process to compensate this expansion. The general system for decimation is shown in Figure 2.10.

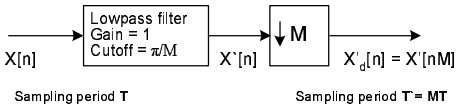


Figure 2.10: General system for sampling rate reduction by facor M [Alan V. Oppenheim, 1999].

On the other hand, in a process of interpolation by M, $M-1$ zeros are inserted between the samples. It results in the aliases at the multiple of the output frequency which are removed by using a lowpass filter after the interpolation process. The zero insertion also results in decrease in the average signal energy, which is compensated by the gain of the filter. The general system for interpolation is shown in Figure 2.11. These processes are explained in detail in Appendix A.

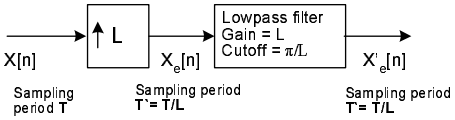


Figure 2.11: General system for sampling rate increase by facor L [Alan V. Oppenheim, 1999]

There are some of the observation in the conventional channelizer, which are listed below:

- The rate conversion process is carried out after downconversion and passing through the filter, which simply discards the samples processed by the downconverter and filter(in the case of decimator). There is no need to process the samples which are eventually discarded by the down sample operation. This will result in the significant computational savings.
- Rate changes by large factor, requires a long filter which results in an increase in computational complexity. One of the solutions is to have multi-stage operations.
- The downconversion and the filter operate at the same rate as the input sampling frequency.

Based on the above observations, there should be some efficient channelizer structure. An efficient structure performs the channelization as a single merged process called a polyphase-path filter bank, which is shown in Figure 2.12. The polyphase filter bank partition offers a number of significant advantages relative to the set of individual down-conversion receivers. The primary advantage is reduced cost due to major reduction in system resources required to perform the multichannel processing [Fredric J. Harris and Rice, 2003]. The next section describes the polyphase channelization in detail.

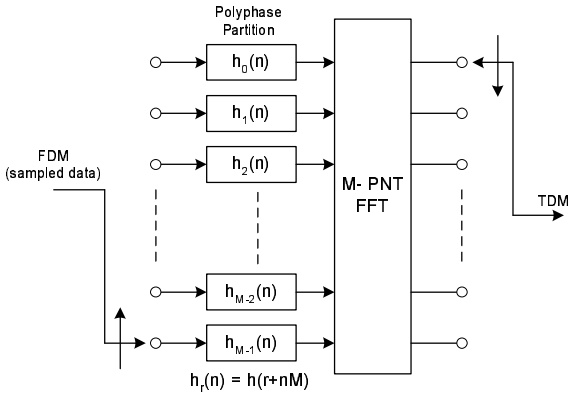


Figure 2.12: Polyphase channelizer: resampler, all-pas partition, and FFT phase shifters

2.5 Polyphase Channelization

In conventional channelizer as shown in Figure 2.9, individual channelizer for each channel are required. One can form only one channel and by having reconfigurability of that, can be used for other channels. On the other hand, the channelizer proposed by 'Fredric J. Harris' as shown in Figure 2.12 is capable of delivering all the required channels just by using one channelizer. Besides that it is more efficient when large sampling rate changes are required.

In the understanding of polyphase channelizer, a stepwise process is explained now, starting from the conventional channelizer and transforming it to the polyphase channelizer [Fredric J. Harris and Rice, 2003]. The block diagram of a single channel of a conventional channelizer is shown in Figure2.13. This structure performs the standard operations of down conversion of the selected channel with a complex heterodyne, low-pass filtering to reduce bandwidth to the channel bandwidth, and down sampling to a reduced rate commensurate with the reduced bandwidth.

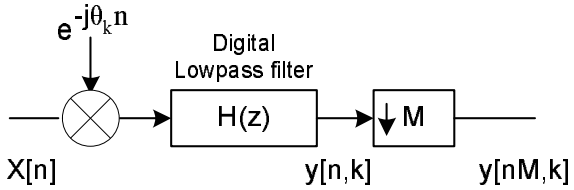


Figure 2.13: Kth channel of conventional channelizer

The expression for $y(n,k)$, the time series output from the kth channel, prior to resampling, is a simple convolution, as shown in the following:

$$y(n,k) = [x[n]e^{-j\theta_k n}] * h[n] \quad (2.1)$$

$$= \sum_{r=0}^{N-1} x[n-r]e^{-j\theta_k(n-r)}h[r] \quad (2.2)$$

The summation of Equation 2.2 can be rearranged to obtain a related summation reflecting the equivalency theorem. The equivalency theorem states that the operations of down conversion

followed by a low-pass filter are totally equivalent to the operations of a bandpass filter followed by a down conversion.

$$y(n,k) = \sum_{r=0}^{N-1} x[n-r]e^{-j\theta_k(n-r)}h[r] \quad (2.3)$$

$$= \sum_{r=0}^{N-1} x[n-r]e^{-jn\theta_k}h[r]e^{jr\theta_k} \quad (2.4)$$

$$= e^{-jn\theta_k}\sum_{r=0}^{N-1} x[n-r]h[r]e^{jr\theta_k} \quad (2.5)$$

The block diagram demonstrating this relationship is shown in Figure 2.14, while the rearranged version of Equation 2.2 is shown in Equation 2.5.

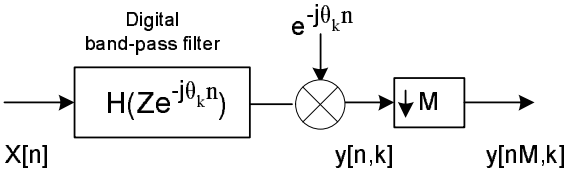


Figure 2.14: Bandpass filter, Kth channel of channelizer

Applying the transformation suggested by the equivalency theorem to an analog prototype system does not make sense since it doubles the required hardware. It would have to replace a complex scalar heterodyne (two mixers) and a pair of low-pass filters with a pair of bandpass filters, containing twice the number of reactive components, and a full complex heterodyne (four mixers), whereas digital filters which are defined as a set of weights stored in coefficient memory. So, in the digital world, no cost is incurred in replacing the low-pass filter required in the first option with bandpass filter required for the second option. This is accomplished by a simple download to the coefficient memory.

It is noted that following the output down conversion, a sample rate reduction is performed by retaining only one sample in every M samples. Recognizing that there is no need to down convert the samples that are discarded in the down sample operation, so only the retained samples are to be down sampled. This is shown in Figure 2.15.

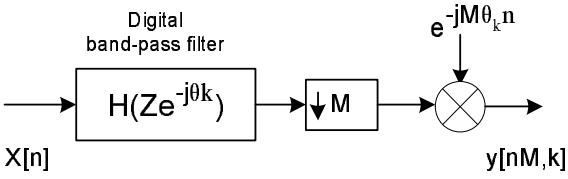


Figure 2.15: Down-sampled down-converted bandpass kth channel of channelizer

The down converter is shifted to the low data-rate side of the resampler, it is, in fact, also down sampling the time series of the complex sinusoid. The rotation rate of the sampled complex sinusoid is Θ_k and $M\Theta_k$ radians per sample at the input and output, respectively, of the M-to-1

resampler.

This change in rotation rate produce an aliasing affect, a sinusoid at one frequency or phase slope, appears at another phase slope when resampled. A constraint is invoked on the sampled data center frequency of the down-converted channel, by choosing center frequencies, which will alias to DC (zero frequency) as a result of the down sampling to $M\Theta_k$. This condition is assured if $M\Theta_k$ is congruent to 2π, which occurs when $M\Theta_k = k2\pi$ or, more specifically, when $\Theta_k = k2\pi/M$.

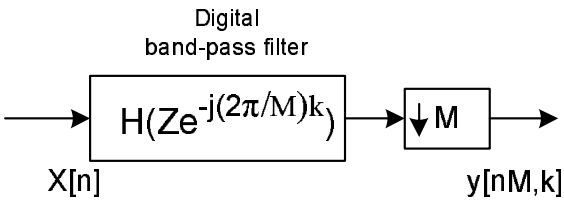


Figure 2.16: Alias to baseband down-sampled down-converted bandpass kth channel of channelizer

The modification to Figure 2.15 to reflect this provision i.e. $\Theta_k = k2\pi/M$is seen in Figure 2.16. The constraint that the center frequencies be integer multiples of the output sample rate assures aliasing to baseband by the sample rate change. When a channel aliases to baseband by the resampling operation, the resampled related heterodyne defaults to a unity-valued scalar, which consequently is removed from the signal-processing path.

The operations invoked by applying the equivalency theorem to the down-conversion process has following sequence of maneuvers:

- slide the input heterodyne through the low-pass filters to their outputs;
- doing so converts the low-pass filters to a complex bandpass filter;
- slide the output heterodyne to the downside of the down sampler;
- doing so aliases the center frequency of the oscillator;
- restrict the center frequency of the bandpass to be a multiple of the output sample rate;
- doing so assures alias of the selected passband to baseband by the resampling operation;
- discard the now unnecessary heterodyne.

The savings realized by this form of the down conversion is due to the fact that it no longer requires an oscillator, nor the input mixer to effect the frequency translation.

2.5.1 Transforming the channelizer

The current configuration of the single-channel down converter involves a bandpass filtering operation followed by a down sampling of the filtered data to alias the output spectrum to baseband. There is no need to compute the output samples from the passband filter that will be discarded by

the down sampler. Now interchange the operations of filter and down sample with the operations of down sample and filter. The process that accomplishes this interchange is known as the *noble identity* which states that 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 Z^M in the filter impulse response shows that the coefficients in the filter are separated M-samples rather than the more conventional one sample delay between coefficients in the filter H(z).

In order to apply the noble identity, some rearrangement has to be done and it starts with an initial partition of the filter into M-parallel filter paths. The Z-transform description of this partition is presented in Equation 2.8, which is interpreted in Figures 2.17, 2.18, 2.19. For ease of notation, first the baseband version of the noble identity is examined and then trivially extend it to the passband version.

$$H(Z) = \sum_{n=0}^{N-1} h[n]Z^{-n} \tag{2.6}$$

$$= \sum_{r=0}^{N-1} Z^{-r} H_r(Z^M) \tag{2.7}$$

$$= \sum_{r=0}^{N-1} Z^{-r} \sum_{n=0}^{(N/M)-1} h(r+nM)Z^{-Mn} \tag{2.8}$$

The block diagram reflecting this M-path partition of a resampled digital filter is shown in figure 2.17.

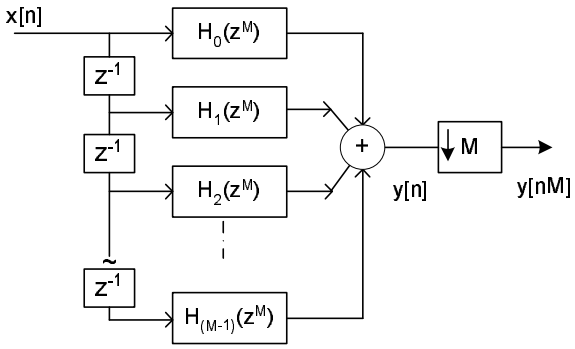


Figure 2.17: M-path partition of a prototype low-pass filter with output resampler

The output of the filter is the resampled sum of the output of the separate filter stages along the M-paths. The resampler is pulled through the output summation element and down sample the separate outputs, only performing the output sum for the retained output sample points. With the resamplers at the output of each filter, which operates on every Mth input sample, which is prepared to invoke the noble identity and pull the resampler to the input side of each filter stage. This is shown in Figure 2.18.

The input resamplers operate synchronously, all closing at the same clock cycle. The signal delivered to the filter's path are one-stage delay line, which is the previous input sample. The

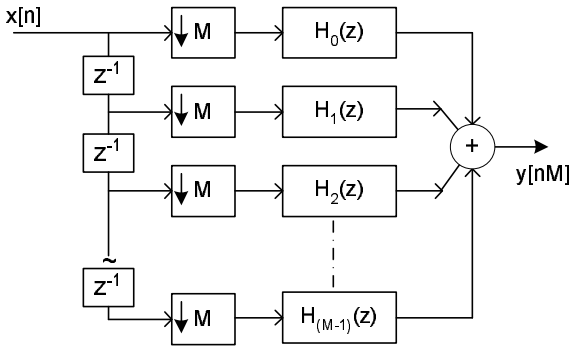


Figure 2.18: M-path partition of a prototype low-pass filter with input resampler (Noble Identity)

interaction of the delay lines in each path with the set of synchronous switches(M-1 converters) can be likened to an input commutator that delivers successive samples to successive legs of the M-path filter. This interpretation is shown in Figure 2.19.

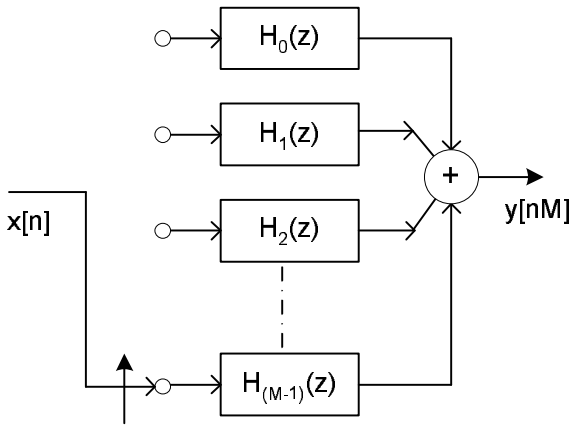


Figure 2.19: M-path partition of a prototype low-pass filter with input path delays and M-1 resamplers replaced by input commutator.

Now the final steps of the transform is carried out that changes a standard mixer down converter to a resampling M-path down converter. By applying the frequency translation property of the Z-transform, a low-pass filter can be converted to a bandpass filter by associating the complex heterodyne terms of the modulation process either with the filter weights or with the delay elements storing the filter weights.

$$H(Z) = \sum_{n=0}^{N-1} h[n]Z^{-n} \tag{2.9}$$

$$G(Z) = H(Z)|_{z=e^{j\theta}Z} = H(e^{-j\theta}Z) \tag{2.10}$$

Now applying this relationship to Equation 2.5 or, equivalently, to Figure 2.19 by replacing each Z with $Ze^{-j\theta}$, or more clearly, replacing each Z^{-1} with $Z^{-1}e^{j\theta}$, with the phase term satisfying the congruency constraint that $\theta = k(2\pi/M)$. Thus,Z^{-1} is replaced with $Z^{-1}e^{jk(2\pi/M)}$, and $Z^{-M}$is replaced with $Z^{-M}e^{jkM(2\pi/M)}$. By design, the kMth multiple of $2\pi/M$is a multiple

of 2π for which the complex phase rotator term defaults to unity, or in this interpretation, aliases to baseband (dc). The default to unity of the complex phase rotator occurs in each path of the M-path filter shown in Figure 2.20. The nondefault complex phase angles are attached to the delay elements on each of the M paths. For these delays, the terms $Z^{-r}$are replaced by the terms $Z^{-r}e^{jkr(2\pi/M)}$. The complex scalar attached to each path of the M-path filter can be placed anywhere along the path and, in anticipation of the next step, the complex scalar are placed after the down-sampled path filter segments $H_r(Z)$. This is shown in Figure 2.20.

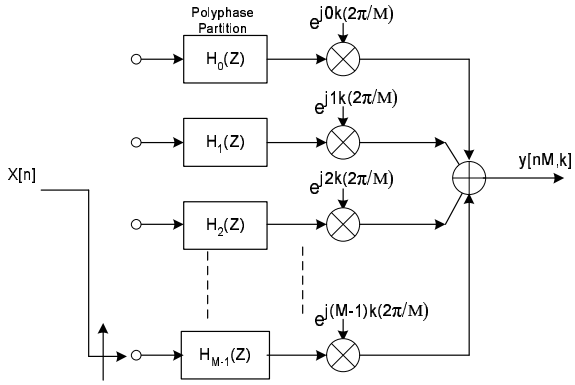


Figure 2.20: Re-sampling M-path down converter

The modification to the original partitioned Z-transform of Equation 2.8 to reflect the added phase rotators of Figure 2.20 is shown in the following:

$$H(Ze{-}j(2\pi/M)k) = \sum_{r=0}^{M-1} Z^{-r}e^{j(2\pi/M)rk}H_r(Z) \tag{2.11}$$

The computation of the time series obtained from the output summation in Figure 2.20 is shown in Equation 2.12.

$$y(nM,k) = \sum_{r=0}^{M-1} y_r(nM)e^{j(2\pi/M)rk} \tag{2.12}$$

Here, the argument nM reflects the down-sampling operation, which increments through the time index in stride of length M, delivering every M^{th} sample of the original output series. The variable $y_r(nM)$ is the nM^{th} sample from the filter segment in the r^{th} path, and $y(nM,k)$ is the nM^{th} time sample of the time series from the k^{th} center frequency. The down-converted center frequencies located at integer multiples of the output sample frequency are the frequencies that alias to zero frequency under the resampling operation. Note the output $y(nM,k)$ is computed as a phase coherent summation of the M output series $y_r(nM)$. This phase coherent sum is, in fact, a discrete Fourier transform (DFT) of the M-path outputs, which can be likened to beam forming the output of the path filters.

The beam-forming perspective offers an interesting insight to the operation of the resampled down-converter system. The reasoning proceeds as follows: the commutator delivering consecutive samples to the M input ports of the M-path filter performs a down-sampling operation. Each

port of the M-path filter receives data at one M^{th} of the input rate. The down sampling causes the M-to-1 spectral folding, effectively translating the M-multiples of the output sample rate to baseband. The alias terms in each path of the M-path filter exhibit unique phase profiles due to their distinct center frequencies and the time offsets of the different down-sampled time series delivered to each port. These time offset are, in fact, the input delays shown in Figure 2.18 and in Equation 2.13. Each of the aliased center frequency experiences a phase shift shown in Equation 2.13 equal to the product of its center frequency and the path time delay.

$$\phi(r,k) = \omega_k \Delta T_r = 2\pi(fs/M)krT_s = 2\pi(fs/M)kr(1/f_s) = (2\pi/M)kr \tag{2.13}$$

The phase shifters of the DFT perform phase coherent summation, very much like that performed in narrow-band beam forming, extracting from the myriad of aliased time series, the alias with the particular matching phase profile. This phase-sensitive summation aligns contributions from the desired alias to realize the processing gain of the coherent sum while the remaining alias terms, which exhibit rotation rates corresponding to the M roots of unity, are destructively canceled in the summation.

The inputs to the M-path filter are not narrow-band, and phase shift alone is insufficient to effect the destructive cancellation over the full bandwidth of the undesired spectral contributions. To successfully separate wide-band signals with unique phase profiles due to the input commutator delays, the operation equivalent of time-delay beam forming must be performed. The M-path filters, obtained by M-to-1 down sampling of the prototype low-pass filter supply the required time delays. The M-path filters are approximations to all-pass filters, exhibiting, over the channel bandwidth, equal ripple approximation to unity gain and the set of linear phase shifts that provide the time delays required for the time-delay beam-forming task.

A useful 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 used to down convert and down sample a single channel, the phase rotators are implemented as external complex products following each path filter. When a small number of channels are being down converted and down sampled, appropriate sets of phase rotators can be applied to the filter stage outputs and summed to form each channel output. When the number of channels becomes sufficiently large in the order of $log_2(N)$, the DFT operation can be used to simultaneously apply the phase shifters for all of the channels required to extract from the aliased signal set. For computational efficiency, the FFT algorithm is used to implement the DFT.

2.5.2 Summary

The commutator performs an input sample rate reduction by commutating successive input samples to selected paths of the M-path filter. Sample rate reduction occurring prior to any signal processing causes spectral regions residing at multiples of the output sample rate to alias to baseband. This desired result allows to replace the many down converters of a standard channelizer, implemented with dual mixers, quadrature oscillators, and bandwidth reducing filters, with a col-

lection of trivial aliasing operations performed in a single partitioned and resampled filter.

The partitioned M-path filter performs the task of aligning the time origins of the offset sampled data sequences delivered by the input commutator to a single common output time origin. This is accomplished by the all-pass characteristics of the M-path filter sections that apply the required differential time delay to the individual input time series. The DFT performs the equivalent of a beam-forming operation; the coherent summation of the time-aligned signals at each output port with selected phase profiles. The phase coherent summation of the outputs of the M-path filters separate the various aliases residing in each path by constructively summing the selected aliased frequency components located in each path, while simultaneously destructively canceling the remaining aliased spectral components.

2.6 Polyphase filter bank parameters

Channel bandwidth, **spectral spacing** and the **output sampling rates** are the parameters, required to be adjusted for the polyphase channelizer. The DFT performs the task of separating the channels after the polyphase filter so it is natural to conclude that the transform size is locked to the number of channels [Fredric J. Harris and Rice, 2003]. Filter bandwidth is determined by the weights of the low-pass prototype and that this bandwidth and spectral shape is common to all the channels. In standard channelizer designs, the bandwidth of the prototype is specified in accord with the end use of the channelizer outputs. when a channelizer is used to separate adjacent communication channels, which are characterized by known center frequencies and known controlled nonoverlapping bandwidths, the channelizer must preserve separation of the channel outputs. Inadequate adjacent channel separation results in adjacent channel interference [Fredric J. Harris and Rice, 2003].

The polyphase filter channelizer uses the input M-to-1 resampling to alias the spectral terms residing at multiples of the output sample rate to baseband. This means that, for the standard polyphase channelizer, the output sample rate is the same as the channel spacing. When operated in this mode, the system is called a *maximally decimated filter bank*.

2.7 Maximally decimated filter bank

The general blocks of this efficient channelizer are shown in Figure 2.21. When sample rate matches with the spectral spacing, the filter bank is said to be maximally decimated.

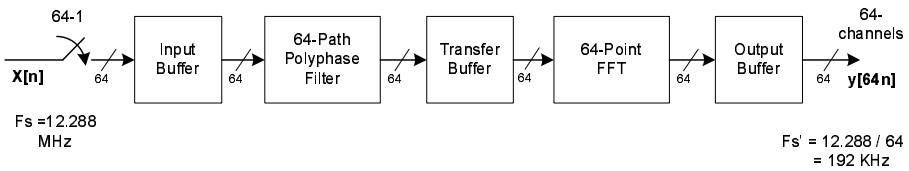


Figure 2.21: Maximally decimated filter

Figure 2.21 shows a system in which a sequence at f_s is downsampled by a factor of $M = 64$ and fed to a 64-path polyphase filter. The commutator performs an input sample rate reduction by commutating successive input samples to selected paths of the M-path filter. The down-sampler- a commutator operating at a rate of M (64), is an efficient implementation of down-sampler, instead of using delay elements and then 64-1 (M-1) down-samplers for each polyphase path.

In order to change the desired sampling rate along with number of channels (which are same for maximally decimated filter bank), the filter structure is modified as shown in Figure 2.21. Let the desired downsampling factor be 48 instead of 64 and number of cannels be 50 instead of 64.

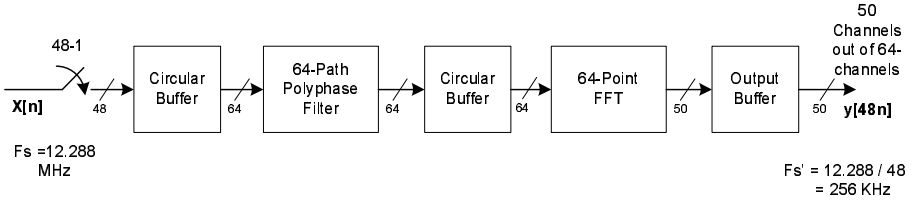


Figure 2.22: Modified Maximally decimated filter to have different decimated factor along with different number of channels

Thus, the task is to use the 64-point DFT to separate and deliver 50 of the possible 64 channels spanned by the sample rate, but to deliver one output sample for every 48 input samples. Figure 2.22 is a block diagram of the modified form of original maximally decimated version of the 64-stage polyphase channelizer. The difference in the two systems resides in the block inserted between the 64-stage polyphase filter and the 64-point FFT. Remarkably, the inserted block performs no computation, but rather only performs a set of scheduled circular buffers shifts [Fredric J. Harris and Rice, 2003]. The details of polyphase channelizer for non-maximally decimated mode will be explained in the next sections.

2.8 Polyphase Computational Complexity

This section compares the computational workload required to implement a channelizer as a bank of conventional down converters with that required to implement the polyphase resampling approach.

Taking an example of the 50-channel channelizer to supply actual numbers. First the length of the finite impulse response (FIR) prototype filter s required to satisfy the filter specifications. The filter designed to operate at its input rate (12.288 MHz) has its specifications controlled by its output rate (256 kHz). This is because the filter must satisfy the Nyquist sampling criterion after spectral folding as a result of the down-sample operation. The length of any FIR filter is controlled by the ratio of input sample rate to filter transition bandwidth and the required out-of band attenuation, as well as level of in-band ripple. Standard design rules determine the filter length from the filter specification, and the filter length was found to be 512 Taps.

An important consideration and perspective for filters that have different input and output sample rates is the ratio of filter length (with units of operations/output) to resample ratio (with units of inputs/output) to obtain the filter workload (with units of operations/input) [Fredric J. Harris and Rice, 2003]. A useful comparison of two processes is the number of multiplies and adds per input point. A multiply and add with their requisite data and coefficient fetch cycles is counted as a single processor operation and uses the shorthand notation of "ops" per input.

A single channel of a standard down-converter channelizer requires one complex multiply per input point for the input heterodyne and computes one complex output from the pair of 512 tap filters after collecting 48 inputs from the heterodyne. The four real ops per input for the mixer and the two (512/48) ops per input for the filter result in a per channel workload of 26 ops per input, which occur at the input sample rate [Fredric J. Harris and Rice, 2003].

The polyphase version of the down converter collects 48 input samples from the input commutator, performs 1024 ops in the pair of 512 tap filters, and then performs a 64-point FFT with its upper bound workload of real ops. The total workload of 1024 ops for the filter and 768 ops for the FFT results in 1792 ops performed once per 48 inputs for an input workload of 38 real ops/input. The higher workload per input is the consequence of forming 64 output channels in the FFT, but preserving only 50 of them [Fredric J. Harris and Rice, 2003].

The workload per input sample for the standard channelizer was found to be 26 ops, and for the polyphase channelizer was found to be 38 ops. The advantage is that the polyphase 38 ops per input built all 50 channels, and the standard down converter's 26 ops per input built only one channel and has to be repeated 50 times. Thats impressive!. By comparing numbers, it can be concluded that the polyphase form should be used even if just a few output channels are required, because the polyphase down converter requires less computations than even two standard down converters.

While comparing hardware resources, the standard channelizer must build and apply 50 complex sinusoids as input heterodynes to the input data at the high input sample rate and further must store the 50 sets of down converted data for the filtering operations. On the other hand, the polyphase filter bank only stores one set of input data because the complex phase rotators are applied after the filter rather than before and the phase rotators are applied at the filter output rate, as opposed to the filter input rate.

CHAPTER 3

WLAN AND UMTS CHANNELIZERS

In this chapter, polyphase channelizers for WLAN and UMTS are designed based on the analysis carried out in the Chapter 2. It cover the basic channelizers, system level modifications, and techniques to obtain the desired output sampling rate. Based on the observations, polyphase channelizers are reconstructed (after resampling the data), in order to reduce the processing load on the sub-filters. Refering back to A^3-Model, we are now in the Algorithm domain, performing the Application to Algorithm mapping, as shown in the Figure 3.1.

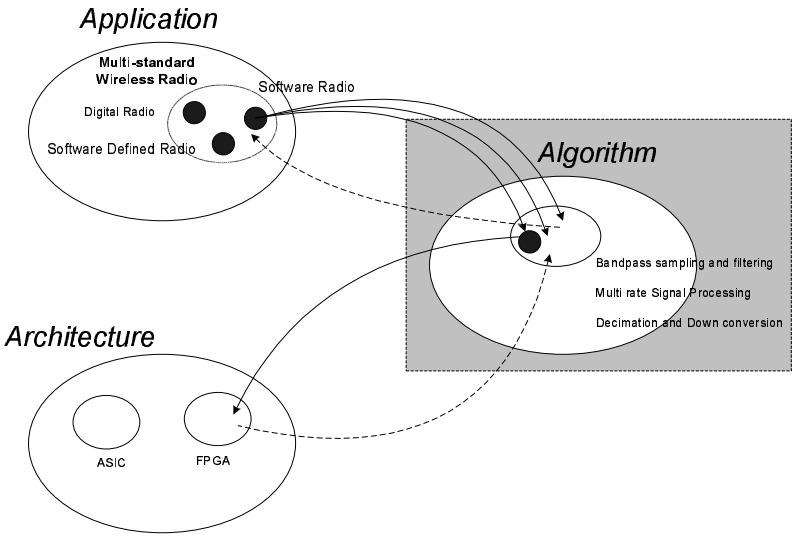


Figure 3.1: A^3-Model: Emphasising the Algorithm domain, where the mapping from the Application to Algorithm is performed.

3.1 WLAN and UMTS Channelizers

Polyphase channelizer are most efficient in term of computations and required hardware resources as compared to standard channelizer. Based on the uniques features of the polyphase channelizer,

we have choosen it, to implement the project scenario.

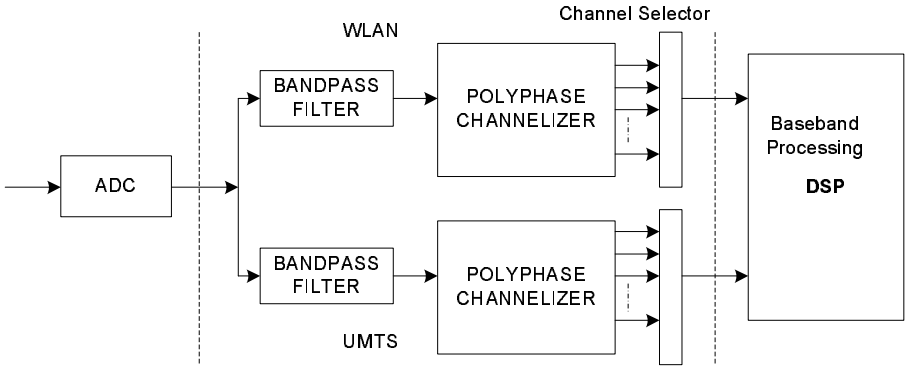


Figure 3.2: The modified bandpass receiver structure: The bandpass filters are required to seperate the bands of UMTS and WLAN standards, and their channels are further down-sampled and down converted to base band by using channelizers.

In order to use the polyphase channelizer, the bandpass receiver structure has to be modified as shown in the Figure 3.2. The bandpass filters are required to seperate the bands of UMTS and WLAN standards, and their channels are further down sampled and down converted to base band by using channelizers. Focusing to the channelizer block, we have to calculate the channel spacing, number of channels/transform size in accordance with the input sampling rate, along with the downconversion factor. In WLAN, the Nyquist zone alias reside at (220-304)MHz, with individuals channels centered at 232, 262 and 292 MHz, whereas in UMTS, the Nyquist zone alias reside at (82-142)MHz, with individuals channels centered at 84.5, 89.5, ... and 139.5 MHz.

The relation between the sampling frequency, channel spacing and number of channels for the polyphase channelizer is [Fredric J. Harris and Rice, 2003]:

$$f_s = N \times \Delta f \tag{3.1}$$

where f_s is the input sampling frequency, N is number of channels/transform size and Δf is the inter channel spacing.

The number of channels/transform size for WLAN and UMTs comes out to be 22.53 and 135.2 for the channel spacings of 30 and 5 MHz respectively at 676MHz of sampling fequency. These are the numbers for which the input spectrum is divided into equal spectral bins. There are two requirements here, one is that the (N) number of channels should be an integer, and second that the channels to be down-sampled and down-converted to baseband should be centered to the multiples of the channel spacing. For both of these two conditions, polyphase channelizer do not fit over the given scenario of UMTS and WLAN aliases as shown in Figure 3.3.

Channelization in this case will results in the corresponding offset from the baseband for each of the downconverted channel as shown in the Figure 3.4.

The first condition i.e. channel numbers to be an integer number, can be meet by changing either the sampling frequency (f_s) or the channel spacing (Δf). The baseband offsets of the down-

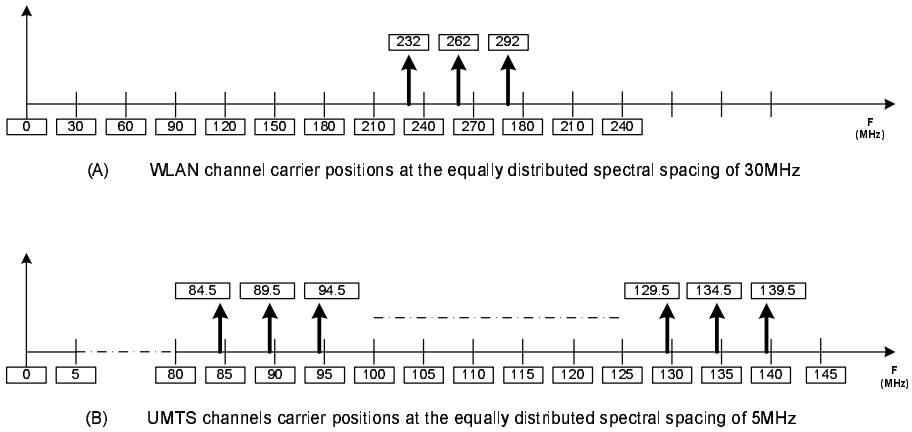


Figure 3.3: Equal spectral distribution of WLAN and UMTS spectrum to 30 and 5MHz channel spacing respectively. Non of the WLAN and UMTS channels become centered on the multiples of 30 and 5 MHz placing respectively.

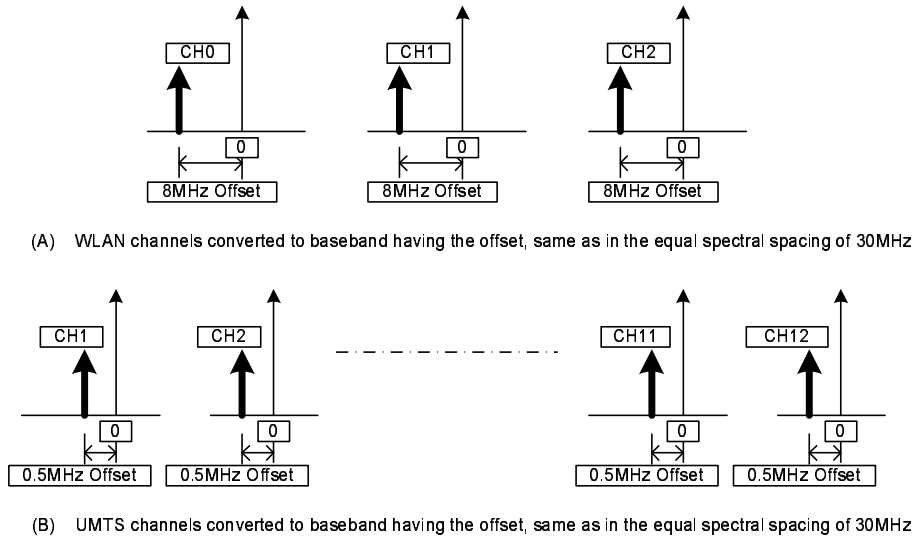


Figure 3.4: Channelization results in the corresponding offset from the baseband for each of the downconverted channel, due to non-centered channels on the equally distribted spectral placing.

converted channels due to voilation of the second requiremnt, can be treated by having follwing three ways:

- One is to change the sampling frequency and chosen such that the aliases of UMTS and WLAN channels satisfy the required demands of being at the equal spectral intervals. Different sampling frequencies instead of 676MHz are tried (frequencies lower than 676MHz), keeping the required aliases of UMTS and WLAN non-overlapped, in order to meet the integer number of channels and equal spectral spacing, but non of them satisfied both of these conditions.

- Second way is to use the channelizer as it is, with unequal channel placements but having the first condition true. In order to compensate the resulted frequency offset from the base band, the signal can be further hyterodyned and lowpass filtered, as shown in the Figure 3.5.

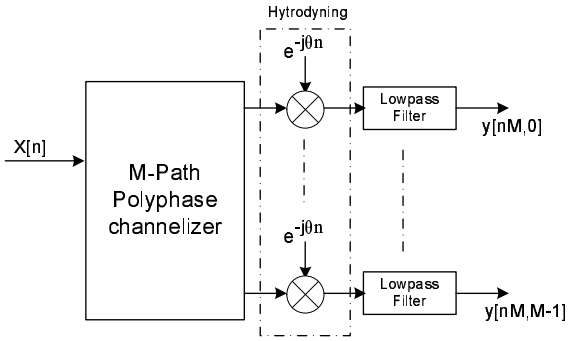


Figure 3.5: Frequency offset from the base band is compensated by further hyterodyned and lowpass filtering the signal.

But this is not an efficient structure, since it uses an extra filter and a mixer for each of the channel, resulting in the requirement of more hardware resources. The required mixer can simply be restricted to ± 1 or 0, if the required hytrodyne is a simple translation from the quarter sampling rate to the base band, thus avoiding the use of actual multiplication.

- The third way is to made some changes in the polyphase structure so that this hetrodyning gets embedded in it. Now a variant of the polyphase structure having the required funtionality is described [Harris, 2006]. The Z-transform of the frequency translated version of the prototype filter impluse response is:

$$H(Z) = \sum_{n=0}^{N-1} h(n) e^{j(2\pi/M)nk} Z^{-n} \tag{3.2}$$

and the 1-to-M polyphase partition of it is:

$$H(Z) = \sum_{r=0}^{M-1} \sum_{n=0}^{(N/M)-1} h(r+nM) e^{j(2\pi/M)(r+nM)k} Z^{-(r+nM)} \tag{3.3}$$

$$= \sum_{r=0}^{M-1} e^{j(2\pi/M)rk} Z^{-r} \sum_{n=0}^{(N/M)-1} h(r+nM) e^{j(2\pi/M)nMk} Z^{-nM} \tag{3.4}$$

When the frequency index k is an integer. $2\pi nk$ is congruent to 2π, and the selected frequency bin, bin k, aliases to zero in the polyphase partition. A variant of this relation ship is obtained by replacing k with $k + s/d$, where s= 0,1,2...d-1. Lets take $d = 4$ and the s becomes 0,1,2,3. and the resulted equation is:

$$H(Z) = \sum_{r=0}^{M-1} e^{j(2\pi/M)r(k+s/4)} Z^{-r} \sum_{n=0}^{(N/M)-1} h(r+nM) e^{j(2\pi/M)nM(k+s/4)} Z^{-nM} \quad (3.5)$$

$$= \sum_{r=0}^{M-1} e^{j(2\pi/M)r(k+s/4)} Z^{-r} \sum_{n=0}^{(N/M)-1} h(r+nM) e^{j(2\pi/4)ns} Z^{-nM} \quad (3.6)$$

which shows the inner sum representing the operation of polyphase stage still has a phase shift that varies with time index n. The residual phase term for the case $d = 4$ is simple power of j. In the operation, when path cofficints are loaded into the path filters, the coefficents are rotated by the path rotation $exp(j0.5\pi n)$ for s=1 or $exp(-j0.5\pi n)$ for s=3. This pre-rotation of the weights results in successful conversion, by the sampling operation, of the frequency component of the channels offset by the quarter of the channel spacing. This offset is also embedded in the phase rotators on each polyphase arm that are applied in the outer summation of Equation 3.6, embedding the j phase rotator in the path weights has a slight impact on the structure of the plyphase filter arms and the subsequent phase rotator. While no actual complex products are involved in the polyphase arm, the data formed by the polyphase arm are now complex rather than real. This means that formely complex scalar phase rotator applied at the stage output now requires a full complex product. The structure of the modified polyphase filter is shown in the Figure 3.6.

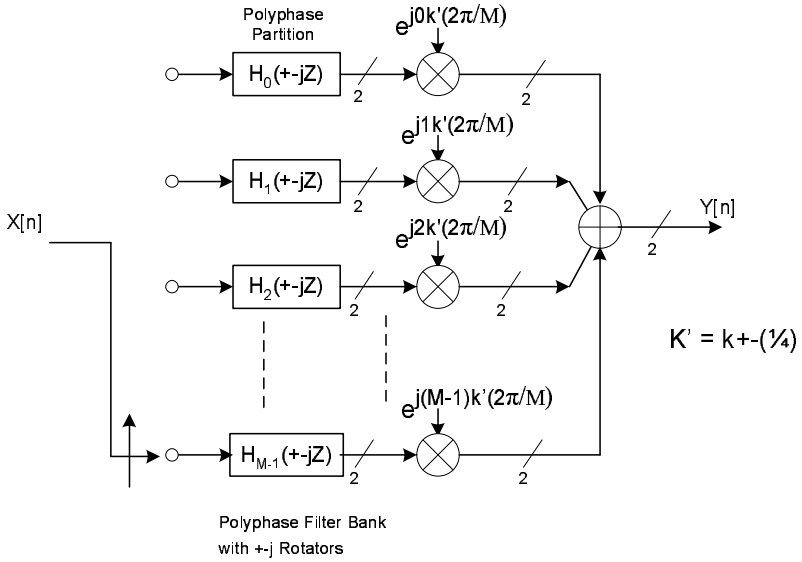


Figure 3.6: Modified form of the polyphase channelizer to compensate the baseband offset to zero. The struture is efficient for the offsets of quarter multiples of the channel spacing.

The structure shown above describe the senario of offsets in quarter steps of the channel spacing. The significance of this quarter offset intervals is that no actual complex products

are involved in the polyphase arms. The offsets other than the quarter one can be achieved in the polyphase structure but doing so results in complex products in the polyphase arms, requiring more hardware resources.

3.1.1 Conclusions

The methods for compensating the offsets of the downconverted signals to baseband have been discussed above and it is seen that the modified polyphase structure is more efficient. Furthermore the modified structure is best for the offsets of quarter multiples of the channels spacing.

3.2 Modified System Design

Based on the conclusions above, it is seen that the offset of the quarters of the channel spacing can be efficiently downconverted to the baseband. But even this modification does not help in transforming the UMTS and WLAN channels to baseband, because in WLAN the offsets of 8MHz is not the quarter sub-multiple of channel spacing of 30MHz and in UMTS the offsets of 0.5MHz is not the quarter sub-multiple of channel spacing of 5MHz. In order to fit the polyphase channelizers, there can be two options:

- The UMTS and WLAN channels are resampled in such a way that the resulted signals fullful the requirements of the polyphase channelizers. It is shown in Figure 3.7.

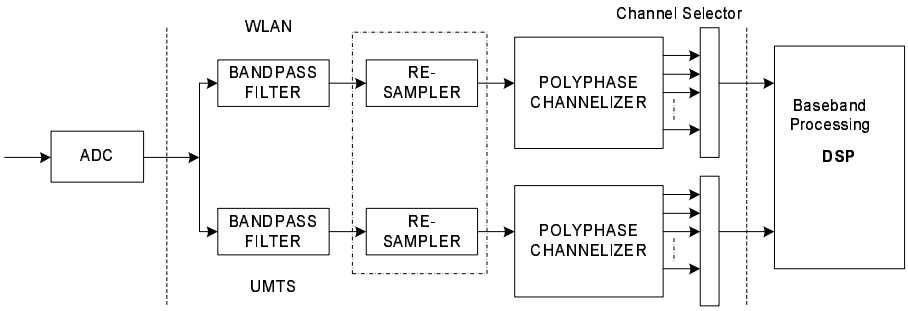


Figure 3.7: Re-sampler blocks inserted prior to the UMTS and WLAN channelizers to fullfill the polyphase channelizer requirements, but it do not work out.

The re-sampler block is in fact a sample rate converter. Different integer factors have been tried but even doing so does not solve the required channel spacing and integer number of channel requirement. Rational number factor have not been tried since it would require an extra filter, which will require extra hardware.

- The second option is to change the sampling frequency of the system and select the one that results in equal channel spacing and integer number of channels, as required by the polyphase channelizer. Different sampling frequencies have been tried to bind all the required parameters (frequencies higher than 676MHz) and finally a sampling frequency of 840MHz become the one that meets these requirements. So the sampling frequeny of the

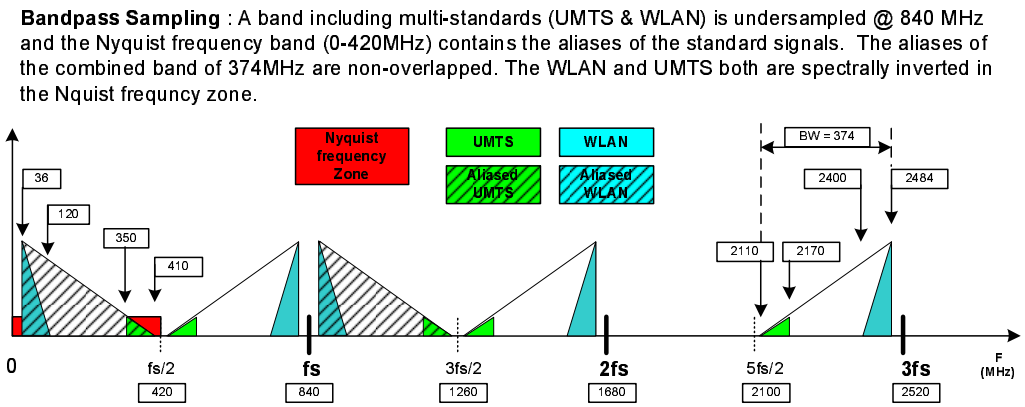


Figure 3.8: The combined spectrum of UMTS and WLAN is bandpass sampled at 840MHz, and the resulted aliases in the Nyquist zone are spectrally inverted.

system has been changed to 840MHz. The corrsponding sampling aliases are shown in Figure 3.8.

The corresponding channels of UMTS and WLAN are shown in Figure 3.9.

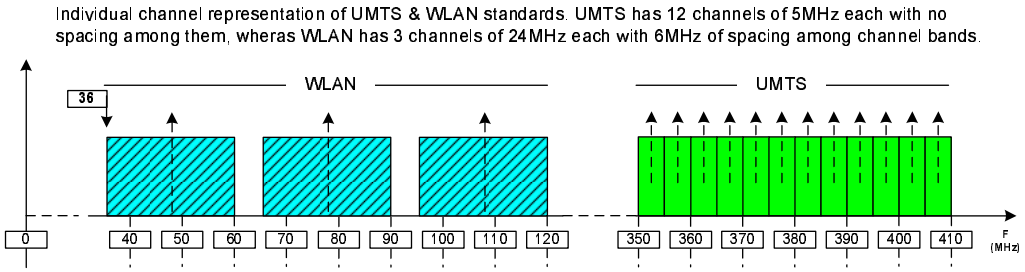


Figure 3.9: Individual channels: (12 UMTS channels and 3 WLAN channels). Each of UMTS channel is 5MHz wide and have 5MHz of spacing between inter-channel carriers, whereas each of the WLAN channel is 24MHz wide and have 30MHz of spacing between inter-channel carriers.

The channel spacing for UMTS is set to be 5MHz that corresponds to 168 channels for the input frequency of 840MHz, whereas for WLAN, channel spacing is set to be 24MHz that corresponds to 35 channels. The channel positions of UMTS and WLAN on the spectral distribution of 5MHz and 24MHz are shown in Figure 3.9.

It is seen from the Figure 3.9 that in the case of UMTS, all the channels centered at the offset of half of the channel spacing (2nd multiple of the quarter of channel spacing), so the corresponding value of s becomes 2 and channel number k varies from 70 to 81. So one polyphase structure can extract all the channels at the same time. In the case of WLAN, the arrangement is somehow different. Channels are centered at positions characterized by (k=2,s=0),(k=3,s=1),(k=4,s=2). This means that for each of the channel, one has to change the s parameter. This will limit the extraction of WLAN channel to one at a time, and depends on the value of s along with value of k. Since we deal with only one of the channels from UMTS and WLAN at a time, so it does not make a difference. The channel spacing and corresponding number of channels are listed in Table 3.1.

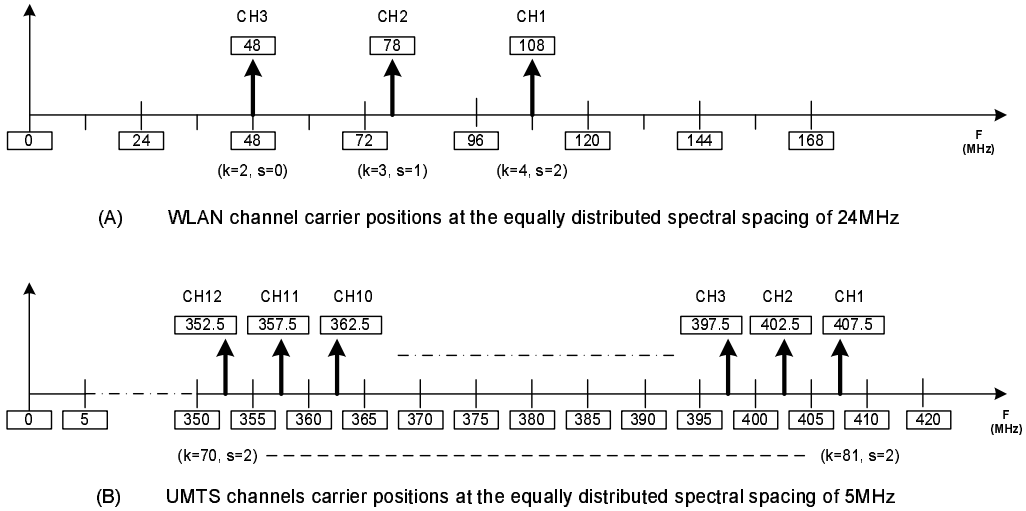


Figure 3.10: Equal spectral distribution of WLAN and UMTS spectrum to 30 and 5MHz channel spacing respectively. UMTS channels become centered on the multiples of 5MHz plus two quarters of 5MHz, whereas in WLAN channels, one become centered on the multiples of 24MHz, another requires an extra offset of one quarter of 24MHz and the other requires an extra offset of two quarters of 24MHz.

Specifications for UMTS and WLAN channelizers

Cases	Sampling rate (MHz)	Channel Spacing (MHz)	No. of Channels
UMTS	840	5	168
WLAN	840	24	35

Table 3.1: Specifications for the channelizer for UMTS and WLAN

After selecting the channel spacing and corresponding number of channels, the corrsponding structures for polyphase channelizer for WLAN and UMTS are shown in Figures 3.11 and 3.12 respectively.

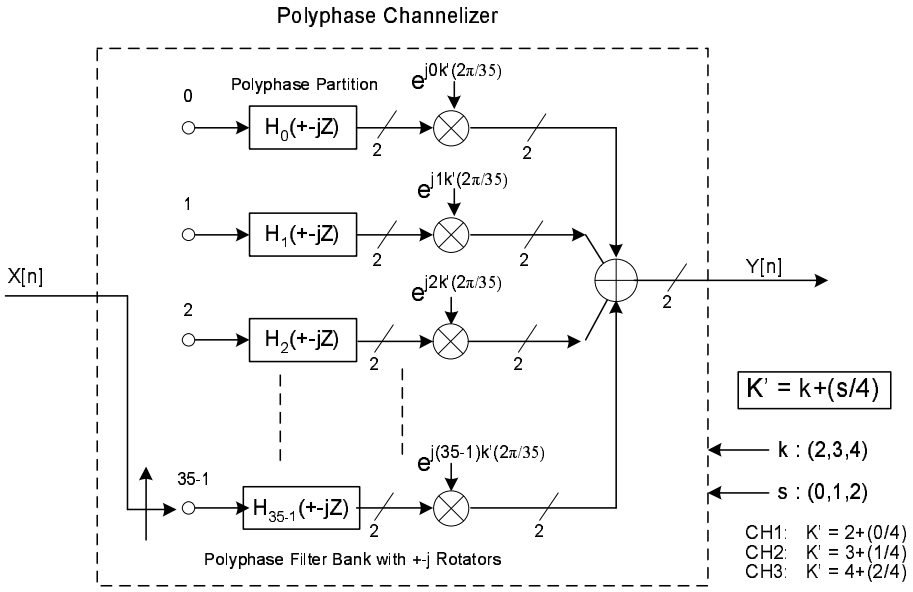


Figure 3.11: 35-Path Polyphase channelizer for WLAN

After specifying the polyphase channelizer structure, i.e. channel spacing and number of channels, now the next task is to obtain the desired output sampling frequency, which is controlled by the commutation operation at the input to the channelizer. The next section describe the operations to change the output sampling rate.

3.3 Sampling Rate Changes

In order to change the sampling rate in a polyphase channelizer, one of the straight forward approach is to change the sampling rate after the polyphase and FFT operation [Chris Dick,], as shown in the Figure 3.13. In this case P/Q resampler blocks are used at the output of each channel. By changing the values of P and Q, required sampling rate can be achieved.

An alternate option is to embed the resampling in the polyphase commutator, in the interaction between input data registers and the polyphase coefficients, and in the interaction between the polyphase outputs and the FFT input [Chris Dick,]. This option has no computational cost, requiring only a state machine to schedule the interactions, which will be explained in the next sections.

3.3.1 WLAN Target Sampling Rate of 20MHz

In the case of WLAN, the number of channels i.e. 35 corresponds to the maximum achievable factor in the maximally decimated mode, but the required decimation factor is $840/20 = 42$. As

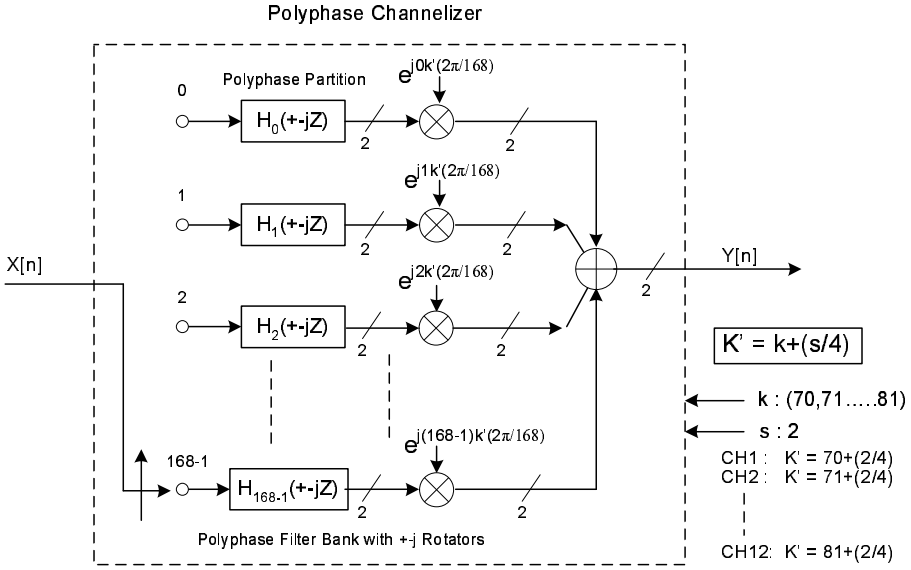


Figure 3.12: 168-Path Polyphase channelizer for UMTS

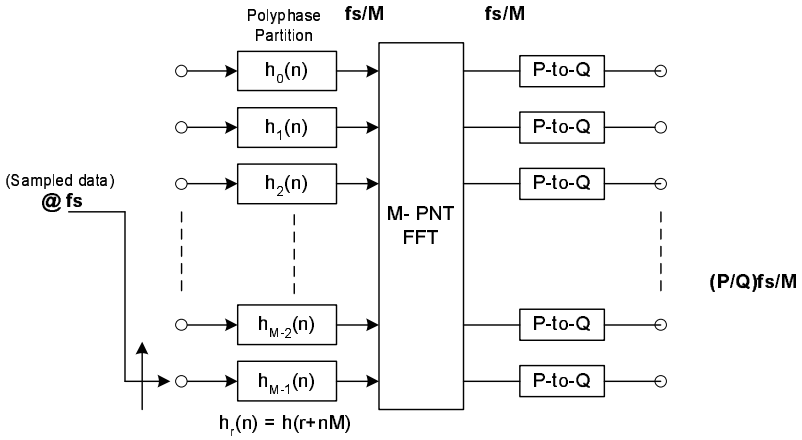


Figure 3.13: Straight forward approach is to change the sampling rate after the polyphase and FFT operation [Chris Dick,]. In this case P/Q resampler blocks are used at the output of each channel. By changing the values of P and Q, required sampling rate can be achieved.

mentioned eariler, that any arbitrary sampling rate can be achieved by simply modifying the commutation operation.

The task is to modify the input commutator to support the 42-to-1 down sample rather than the standard 35-to-1 down sample. This is an almost trivial task. The modified resampling is arranged by keeping the 35-path filter, but feeding 42 ports from the commutator. The commutator for the standard 35-point polyphase filter starts at port 34 and delivers 35 successive inputs to ports 34, 33, 32, and so on through 0, the modified commutator starts at port 6 and delivers 7 successive inputs to ports 6, 5, 4, and so on through 0, and again starting from the bottom i.e. 34, 33 and so on through 0 thus completing 42 inputs. The previous data at the port is shifted before accomodataing next data. Input memory for the 35-path filter must be modified to support this long commutator input schedule. The mapping structure of the reindexing scheme is best seen in the original one-dimensional prototype filter shown in figure and then transferred to the two-dimensional polyphase partition [Harris, 2006].

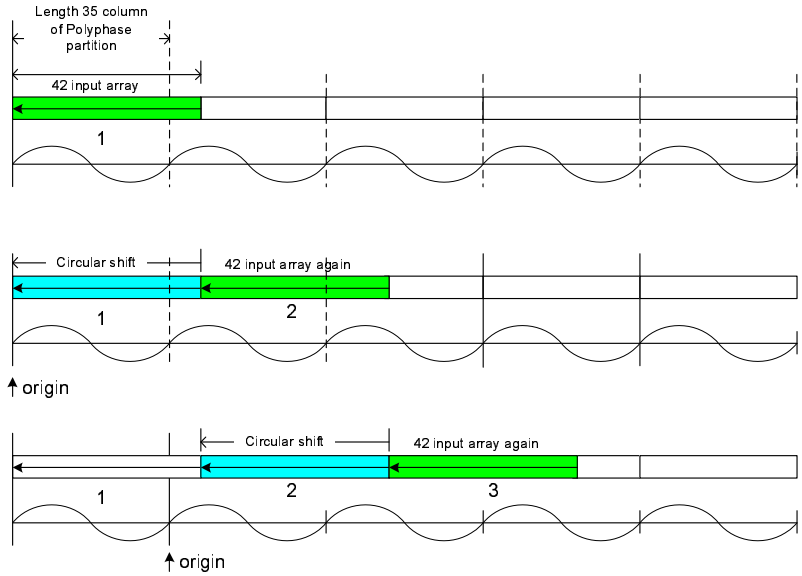


Figure 3.14: Memory contents for successive 42-point input data blocks into a 35-point prototype pre-polyphase partitioned filter and FFT.

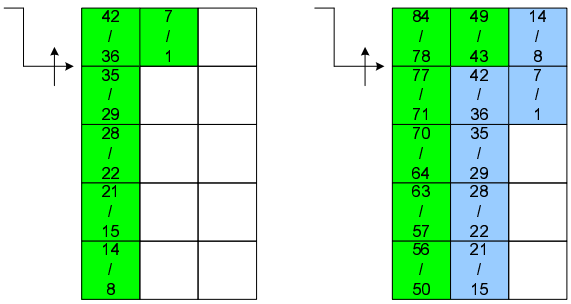


Figure 3.15: Memory contents for successive 42-point input data blocks into a 35-point polyphase filter.

Figure 3.14 presents the memory content for a sequence of successive 42-point input data blocks presented to the 35-point partitioned prototype filter. This figure indicates the interval of 35-tap boundaries that become the columns of the two-dimensional array, as well as the boundaries of successive 42-point input blocks that are presented to the input array. Successive input blocks start loading at address 6 and work up to address 0, and another from 34 up to address 0 again by shifting the previous data. The beginning and end of this interval are denoted by the tail and arrow, respectively, of the left-most input interval in the filter array. As each new 42-point input array is delivered, the earlier arrays must shift to the right-hand side. These shifting array blocks, and the 42 input as well, cross the 35-point column boundaries and, hence, move to adjacent columns in the equivalent two-dimensional partition. This crossing can be visualized as a serpentine shift of data in the two-dimensional array or, equivalently, as a circular row buffer down shift of 42 rows in the polyphase memory with a simultaneous column buffer right shift of the input data column. The operation of this circular buffer is illustrated in Figure 3.15, which indicates the indexes of input data for two input cycles. It is seen that, between two successive input cycles, the rows in the bottom one-fifth of memory translates to the top one-fifth, while the top four-fifths of rows translates down one-fifth of the memory. The columns in the bottom four-fifths shift to the right-hand side on column once, and top one-fifths shift to the right-hand side on column twice during the circular row translations. The next input array is loaded in the left-most columns of this group of addresses.

Returning to Figure 3.14, the one-dimensional prototype, it is noticed that every new data block shifts the input data origin to the right by 42 samples. The vector $\hat{y}(r, 42n)$ formed as the polyphase filter output from all 35 path filters is processed by the FFT to form the vector $\hat{Y}(k, 42n)$ of channelized (index k) output time series (index 42). On each successive call to the FFT, the origin of the sinusoids in the FFT is reset to the beginning of the input array. Since the origin of the input array shifts to the right on successive inputs while the origin of the FFT simultaneously resets to the beginning of the input array, a precessing offset exists between the origins of the polyphase filter and FFT. Origins are aligned, removing the offsets, by performing a circular shift of the vector $\hat{y}(r, 42n)$ prior to passing it to the FFT. Since the offset is periodic and is a known function of the input array index, the circular offset of the vector can be scheduled and controlled by a simple state machine. Figure 3.14 shows the location of the two origins for two successive 42-point input arrays and the amount of circular offset required to align the two prior to the FFT. Note that the offset schedule repeats in 5 cycles, 5 being the number of input intervals of length 42 that is a multiple of 35. The cyclic shift for schedule for the array prior to the FFT is shown in Figure 3.16.

In this description of data memory management the filter coefficients are not moved from their original polyphase partition.

Notice in Figure 3.17 that the cyclic upshift of the seven bottom rows to the top of the stack is undone by the cyclic down-shift at the output of the sub filters. Rather than cycle the data registers, they are kept anchored and the coefficient sets are rotated. This equivalent mapping scheme converts the cyclic shift of the input array and the output buffer to a sliding cyclic load by the input commutator to a fixed set of registers and a cyclic shift of the coefficient memory [Chris Dick,]. This is the process described in the state machine listed in Table 3.2. It is noted that the Load Sequence is always to the next 42- registers where the indexing is performed modulo-35. Thus the next register to accept data as we leave state-0 and move to state-1 is R-1, which is actually R34.

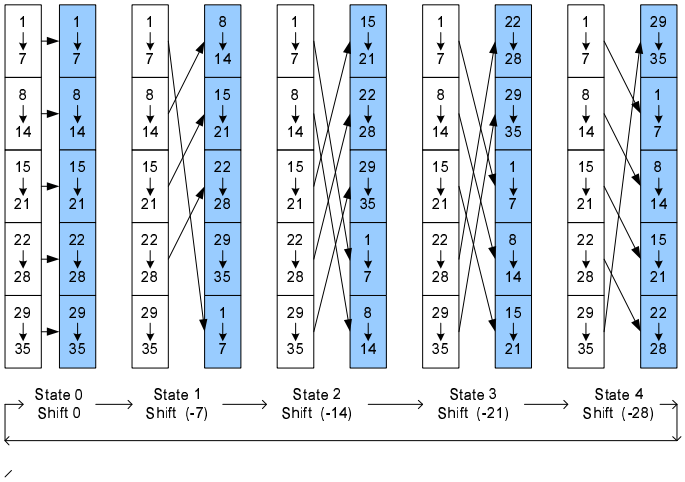


Figure 3.16: Cyclic shift schedule for input array to FFT operation.

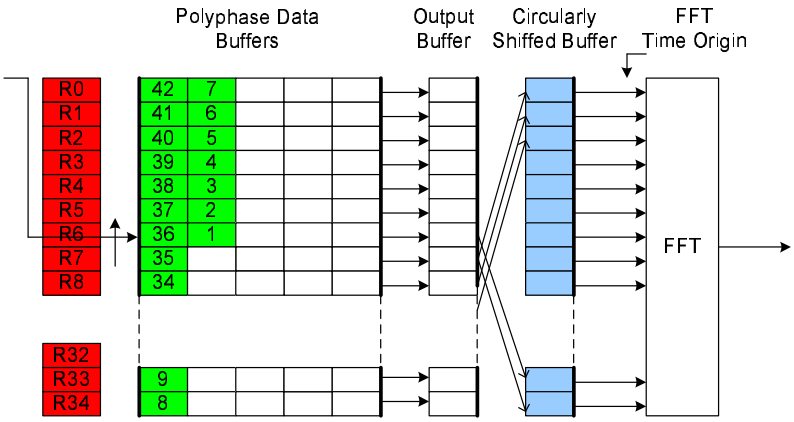


Figure 3.17: Cyclic shift of polyphase output buffer to align time origin of cyclically shifted input buffer with FFT's reset time origin.

In a similar fashion the filter weights assigned to perform the inner products with the registers are always offset -7 modulo 35 relative to the previous filter set.

State Machines for LOAD and FILTER coefficients

STATE	LOAD	FILTERS
0	R6, R5 ,...,R0,R34,R33,...,R0	C0, C1 ,.....,C34
1	R34,R33,...,R0,R34,R33,...,R28	C7, C6 ,...,C34,C0,C1,...,C6
2	R27,R26,...,R0,R34,R33,...,R21	C14,C13,...,C34,C0,C1,...,C13
3	R20,R19,...,R0,R34,R33,...,R14	C21,C22,...,C34,C0,C1,...,C20
4	R13,R12,...,R0,R34,R33,...,R7	C28,C29,...,C34,C0,C1,...,C27

Table 3.2: State Machines for register loads and filter coefficients

3.3.2 UMTS Target Sampling Rate of 61.44MHz

In the case of UMTS, the number of channels i.e. 168 corresponds to the maximum achievable factor in the maximally decimated mode, but the required decimation factor is $840/61.44 = 13.67$.

The required resampling ratio the process is then 13.6 or 68/5. This ratio can be realized by first up sampling the input stream by 5 and then down sampling by 68. The up sampling is performed by zero packing the input data and the down sampling by serpentine shifting data through the filter in stride of length 68. This process is illustrated for two data load iterations in Figure 3.18.

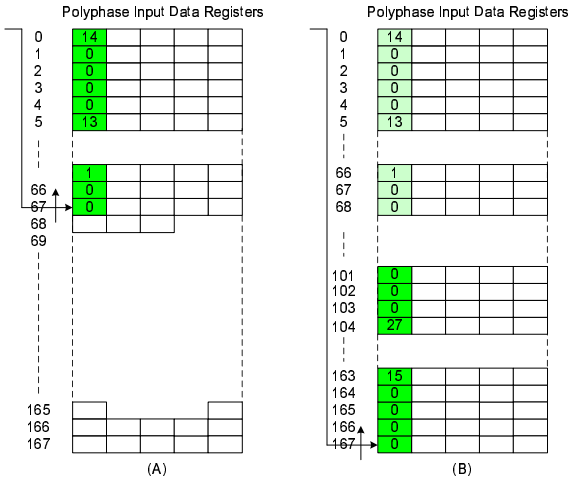


Figure 3.18: Successive serpentine data shifts in polyphase memory and data load for a 68/5 re-sampling in a 168-stage polyphase filter. It shows just two data load operations.

There is no actual zero packing in the final configuration, This is just to illustrate, how the (non-zero) data memory interacts with coefficient memory. It is worth noting that in the first data load, the left Figure, 14-actual data samples are delivered to the 68 register addresses, while in the second load 13-actual data samples are delivered to the 68 register addresses. The data loading procedure is found to be periodic in 210-load cycles for which it will require 210-states to control the process. (The least common multiple of 68 and 168 is 2856, and since 68 zeropacked inputs

are delivered at a time, results in 42 states. For upsampling factor of 5, the LCM of 42 and 5 becomes 210, which is the periodic interval). Table 3.3 lists the memory loading instructions for the process that anchors the data registers and cycles the data load and coefficient sets. Note that in the 210-states, a total of 2856 inputs are delivered and take from the polyphase engine 210 outputs to realize the desired embedded 68/5 resampling. The loading scheme is seen to be a constant offset of -5 modulo 168 within a sequence as well as in the transition between sequences. The -5 offset is a consequence of the 1-to-5 up sampling represented by the zero packing but not actually implemented in the process.

State Machines for LOAD Sequence

STATE	**No. of Inputs**	**LOADING SEQUENCE**
0	14	R67, R62, R57, R52, R47, R42, R37, R32, R27, R22, R17, R12, R7, R2
1	13	R165, R160, R155, R150, R145, R140, R135, R130, R125, R120, R115, R110, R105
2	14	R100, R95, R90, R85, R80, R75, R70, R65, R60, R55, R50, R45, R40, R35
⋮	⋮	⋮
208	13	R34, R29, R24, R19, R14, R9, R4, R167, R162, R157, R152, R147, R142
209	14	R137, R132, R127, R122, R117, R112, R107, R102, R97, R92, R87, R82, R77, R72

Table 3.3: Polyphase filter's Data loading sequence with the state machine

Because of the 1-to-5 up sampling implemented by the zero packing, only one fifth of the weights in each stage actually contributes to the subfilter output. Thus each stage is further partitioned into 5 sub sets of weights, which results in a total of $168 \times 5 = 840$ filter weight sets. These sets are denoted by C0, C1,..., C839 where the integer is the starting index from the original non-partitioned prototype filter. Each filter starts with its index and increments in stride of length 840. Table 3.4 lists the filter assignment to the 168-successive data registers for 210-states of the process.

Table 3.4 shows that in a given state the successive filter index increments by 169 modulo-840 and between states, the filter index increments by 68 modulo-840. The integer 169 is the offset between two data samples in the zero-packed load in two adjacent rows. The 68 index is the number of zero-packed data points introduced per data load cycle.

State Machines for FILTER Co-efficients

STATE	**FILTER Co-efficients sets**
0	C0, C169, C338, C507, C676,.....,C836, C165, C334, C503
1	C68, C237, C406, C575, C744,.....,C64, C223,C402, 571
⋮	⋮
209	C772, C101, C270, C439, C608,.....,C599, C768, C97, C266, C435

Table 3.4: Filter Co-efficients loading sequence with the state machine

In the design process of up sample the data by a factor of five on the way into the filter requires that the prototype filter has to be designed to operate at 5 times f_s or 4200 MHz. Consequently, the filter becomes five times longer than the standard design but since only one-fifth of it is used per processing cycle so no processing penalty is paid.

The 1-to-5 zero-packed signal presents 5 spectral copies to the processing stream of filter and FFT. Figure 3.19 indicates the frequency indices of the spectra prior to up sampling and after the 1-to-5 up sampling. By examining the spectra of the up sampled data at the frequencies that correspond to the pre-up sampled locations, i.e. multiples of $2\pi/168$, different spectral centers are obsevered. For instance, the spectra at $2\pi/168$ at the input rate is frequency "1", but at the output rate there is frequency "5". The spectral locations are re-ordered as a result of processing the up-sampled data in the polyphase filter. This re-ordering is an expected result and is seen in the Good-Thomas (or Prime Factor) algorithm as the result of reside addressing of a-two dimensional array [Chris Dick,]. Thus the FFT processing the polyphase data outputs frequencies in the order [0, 5, 10, 15, ...,158,163,168], which is seen to be indexing stride of 5 modulo-168. The response to the observed rearranged indices is to re-order them back to their natural order.

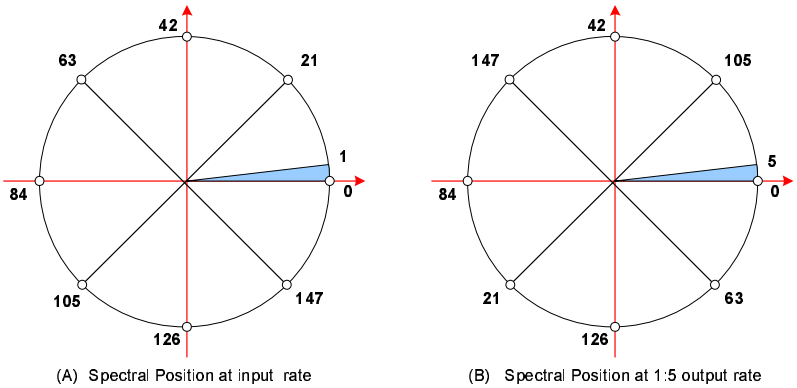


Figure 3.19: Re-ordering of the spectral locations as a result of processing the up-sampled data in the polyphase filter

3.4 Observations

In the polyphase channelizer for WLAN, which has 35 channels of 24 MHz at the input sampling rate of 840 MHz and UMTS, which has 168 channels of 5MHz at the input sampling frequency of 840MHz, there are some observations which are:

- In UMTS, the filter designed for polyphase channelizer has to operate on 5 times the input sampling frequency of 840MHz, results in a very long-length filter. The filter is decomposed to have 840 sets of 168 sub-filter coefficients. Although only one-fifth of the cofficients are used at a time, but it requires large memory to store all the filter coefficients.
- There are total of 168 channels, from which the desired are only 12 of them and only one is used at a time.

- Similarly in WLAN, there are total of 35 channels, but the desired are 3 of them and only one is used at a time.

This puts the extra load on the filtering process in terms of high clock speed requirement and large memory storage for filter coefficients.

In order to lower down these burdens, one of the ways is to decrease the sampling frequency of the signal, that can be done by resampling the signal before the polyphase channelizer input. The idea is that the sampling frequency should be kept as low as possible (not to have overlap aliases). The resampling factor should be chosen such that the resampled signal for WLAN and UMTS have integer number of channels of 5MHz and 30 MHz (or 24MHz as in previous case) respectively based on the new sampling frequency. The second thing is that the correponding aliased channels of WLAN and UMTS must be centered on their channel placing or on the multiples of the quarter of their channel spacing respectively.

Based on the above criteria, different resampling factors for WLAN and UMTS have been tried, which are listed in Tables 3.5 and 3.6.

WLAN Resampling process

Downsampling factor	New Sampling Freq. (MHz)	Channel Status	Channel Integer/non-Integer
2	420	non-overlapped	non-integer
3	280	non-overlapped	non-integer
4	210	overlapped	non-integer
5	168	overlapped	integer
6	140	overlapped	non-integer

Table 3.5: Resampling factors for WLAN, showing the channel status as overlapped/non-overlapped and resulting number of channels as integer/non-integer.

UMTS Resampling process

Downsampling factor	New Sampling Freq. (MHz)	Channel Status	Channel Integer/non-Integer
4	**210**	**non-overlapped**	**integer**
5	168	non-overlapped	non-integer
6	140	overlapped	integer
7	120	overlapped	integer
8	105	overlapped	integer

Table 3.6: Resampling factors for UMTS, showing the channel status as overlapped/non-overlapped and resulting number of channels as integer/non-integer.

It is seen from the Table 3.5 that non of the resampling factors satisfy the two mentioned conditions. Similarly Table 3.6 shows that only one resampling factor i.e. 4 satisfy the two above mentioned conditions.

The bottleneck in the resampling process is the image signal that do not allow the resampling by large factors. This restriction can be omitted if the bandpass filters for WLAN and UMTS

operates on complex signal. This will discard the image and then the signal can be resampled by large factors such that the resultant sampling frequency is above the total signal bandwidth. Resampling process in this case is simply the spectrum translation. Based on this technique, the WLAN and UMTS bandpass filters are made complex and the resultant image free signals for WLAN and UMTS are again tried by different resampling factors to have the mimimum possible sampling freqencies, which are listed in Tables 3.7 and 3.8.

WLAN Resampling process (at Complex signal)

Downsampling factor	New Sampling Freq. (MHz)	Channel Status	Channel Integer/non-Integer
5	168	non-overlapped	non-integer
6	140	non-overlapped	non-integer
7	**120**	**non-overlapped**	**integer**
8	105	non-overlapped	non-integer
10& above	84 & below	< 84.5 MHz WLAN bandwidth	

Table 3.7: Resampling factors for WLAN with complex signal, showing the channel status as overlapped/non-overlapped and resulting number of channels as integer/non-integer.

UMTS Resampling process (at Complex signal)

Downsampling factor	New Sampling Freq. (MHz)	Channel Status	Channel Integer/non-Integer
4	**210**	**non-overlapped**	**integer**
7	120	non-overlapped	integer
8	**105**	**non-overlapped**	**integer**
10	84	non-overlapped	non-integer
12	70	non-overlapped	integer
14	60	non-overlapped	integer
15& above	below 60	< 60 MHz UMTS bandwidth	

Table 3.8: Resampling factors for UMTS with complex signal, showing the channel status as overlapped/non-overlapped and resulting number of channels as integer/non-integer.

Table 3.7 shows that a maximum resampling factor of 7 is possible that results in a new sampling frequency of 120 MHz, with 5 channels of 24 MHz. Table 3.8 shows the maximimum possible resampling factor of 14, that results in 60MHz of new sampling frequency. In order to have desired UMTS rate of 61.44 MHz, an embedded resampling factor of 125/128 is required. Similarly with other two resampling factors of 12 and 8, embedded resampling factors of 875/768 and 875/512 are required. In this rational number embedded resampling factors, we have to design the prototype filter at upsampled frequency. To have the minimum upsampled factor, embedded resampling factor of 875/512 is selected and is rounded to 17/10.

Finally re-sampling factors of 7 and 8 are selected for WLAN and UMTS respectively, resulting in new sampling frequencies of 120 MHz and 105 MHz respectively. This is illustrated in system level block diagram as shown in Figure 3.20.

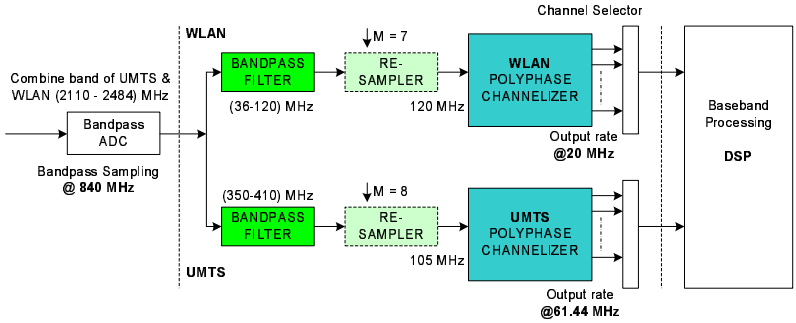


Figure 3.20: Modified system block diagram having re-samplers prior to UMTS and WLAN channelizers.

Based on the above results, a re-sampling factor of 7 is selected for WLAN and the sampling frequency is reduced to 105MHz. The corresponding band of WLAN channels translate to (-42, -12 and 48)MHz. Based on the change in the input sampling frequency, the polyphase channelizer has to be restructured. The modified channelizer for WALN is shown in Figure 3.21 with reduced numbers of polyphase sub-filters.

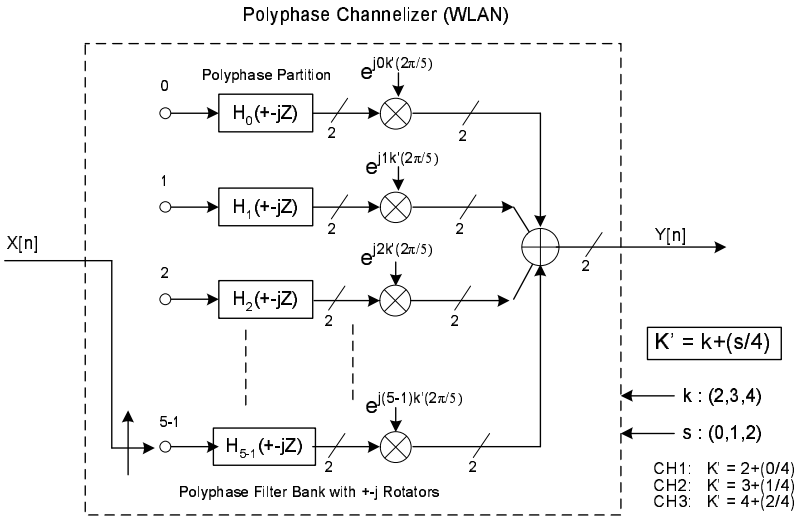


Figure 3.21: Modified WLAN channelizer.

With the sampling frequency of 120MHz and channel spacing of 24MHz, the number of channels become 5 which is the number of the polyphase decomposition. Now the new downfactor to have 20MHz required rate at 120MHz sampling frequency, is 6. This is realized by down sampling by serpentine shifting data through the filter in stride of length 6. This overall process is same as in the previous case with sampling frequency of 840MHz. The only difference is the shifting data through the filter of 5 stages in stride of length 6, instead of shifting data through the filter of 35 stages in stride of length 42. The modified process is illustrated for two data load iterations in Figure 3.22.

A resampling factor of 8 is selected for UMTS and the sampling frequency is reduced to 105MHz. The corresponding band of UMTS channels translate to (35.5 to -12.25)MHz. Based on the change in the input sampling frequency, the polyphase channelizer has to be restructured.

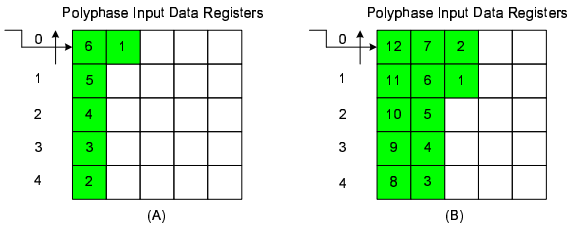


Figure 3.22: Successive serpentine data shifts in polyphase memory and data load for 6:1 re-sampling in a 5-stage polyphase filter. It shows just two data load operations.

The modified channelizer for UMTS is shown in Figure 3.23 with reduced numbers of polyphase sub-filters.

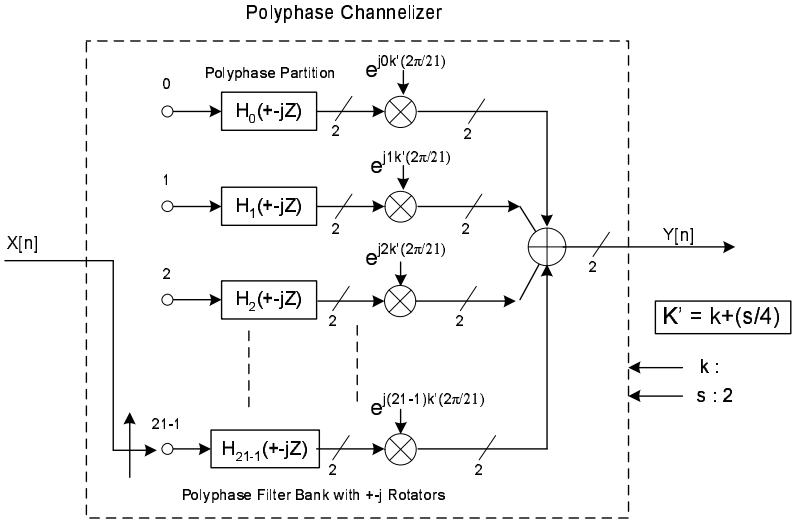


Figure 3.23: Modified UMTS cahnnelizer

With the sampling frequency of 105MHz and channel spacing of 5MHz, the number of channels become 21 which is the number of the polyphase decomposition. Now the new downfactor to have 61.44MHz required rate at 105MHz sampling frequency, is 1.7 or 17/10. This ratio can be realized by first up sampling the input stream by 10 and then down sampling it by 17. The up sampling is performed by zero packing the input data and the down sampling by serpentine shifting data through the filter in stride of length 17. This overall process is same as in the previous case with sampling frequency of 840MHz. The only difference is the shifting data through the filter of 21 stages in stride of length 17, instead of shifting data through the filter of 168 stages in stride of length 68. The modified process is illustrated for two data load iterations in Figure 3.24.

There is no actual zero packing in the final configuration. In the first data load, the Figure 3.24a, 2-actual data samples are delivered to the 17 register addresses, while in the second load 2-actual data samples are delivered to the 17 register addresses. The data loading procedure is found to be periodic in 210-load cycles for which it will require 210-states to control the process. (The least common multiple of 21 and 17 is 357, and since 17 zeropacked inputs are delivered at a time, results in 21 states. For upsampling factor of 10, the LCM of 21 and 10 becomes 210, which is the periodic interval). The periodic factor is same as in the previous case. Table 3.9 lists the

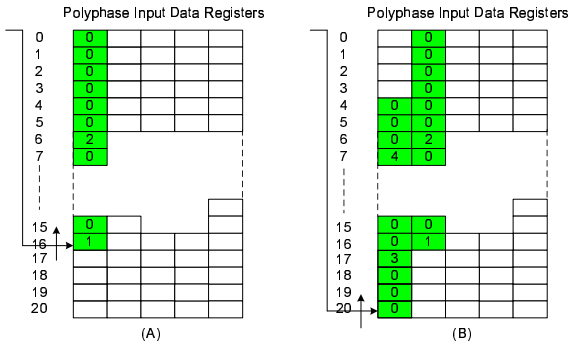


Figure 3.24: Successive serpentine data shifts in polyphase memory and data load for a 17/10 re-sampling in a 21-stage polyphase filter. It shows just two data load operations.

memory loading instructions for the process that anchors the data registers and cycles the data load and coefficient sets. Note that in the 210-states, a total of 357 inputs are delivered and take from the polyphase engine 210 outputs to realize the desired embedded 17/10 resampling. The loading scheme is seen to be a constant offset of -10 modulo 21 within a sequence as well as in the transition between sequences. The -10 offset is a consequence of the 1-to-10 up sampling represented by the zero packing but not actually implemented in the process.

State Machines for Register Load Sequence

STATE	No. of Inputs	LOADING SEQUENCE
0	2	R16, R6
1	2	R17, R7
2	2	R18, R8,
3	1	R19
4	1	R9, R20
⋮	⋮	⋮
208	2	R4, R15
209	1	R5

Table 3.9: Polyphase filter's Data loading sequence with the state machine

Because of the 1-to-10 up sampling implemented by the zero packing, only on tenth of the weights in each stage actually contributes to the subfilter output. Thus each stage is further partitioned into 10 sub sets of weights, which results in a total of $21 \times 10 = 210$ filter weight sets. These sets are denoted by C0, C1,...., C209 where the integer is the starting index from the original non-partitioned prototype filter. Each filter starts with its index and increments in stride of length 210. Table 3.10 lists the filter assignment to the 42-successive data registers for 210-states of the process.

Table 3.10 shows that in a given state the successive filter index increments by 22 modulo-210 and between states, the filter index increments by 21 modulo-210. The integer 22 is the offset between two data samples in the zero-packed load in two adjacent rows. The 21 index is the number

of zero-packed data points introduced per data load cycle.

State Machines for FILTER Co-efficients

STATE	**FILTER Co-efficients sets**
0	C0, C22, C44, C66, C88,.....,C164, C186, C208, C20
1	C17, C39, C61, C83, C105,.....,C181, C203, C15, C37
2	C34, C56, C78, C100, C122,.....,C198, C10, C32, C54
|	|
|	|
|	|
209	C193, C5, C27, C49, C71,.....,C147, C169, C191, C3

Table 3.10: Filter Co-efficients loading sequence with the state machine

The prototype filter has to be designed to operate at 10 times f_s or 1050 MHz due to up sample the data by a factor of ten on the way into the filter. Consequently, the filter becomes ten times longer than the standard design but since only one-tenth of it is used per processing cycle so no processing penalty is paid.

CHAPTER 4

SIMULATIONS

A filter is essentially a system or a network that selectively change the wave shape in the form of magnitude-frequency and phase-frequency characteristics of the signal. A digitial filter is a set of mathematical equations which forms an algorithm and that can be implemented in hardware or software to produce a desired output against the specific input [Emmanuel C. Ifeachor, 2002]. The nature of modification of the signal is dependent on the phase and amplitude characteristics of the filter. Some useful definitions are explained below, which will guide us to better understanding of the filters design flow.

Phase/Group Delay. The phase delay or group delay provides a useful measure of how the filter modifies the phase characteristics of the signal. Phase delay is the amount of delay that each frequency component of the composite signal suffers, as it passes through the filter. The group delay on the other hand is an *average* time delay the composite signal suffers at each frequency. Mathematically the phase delay i.e.T_p is the negative of the phase angle divided by frequency, whereas the group delay i.e.T_g is the negative of the derivitive of the phase with respect to frequency:

$$T_p = -\theta(w)/w \tag{4.1}$$

$$T_g = -d\theta(w)/dw \tag{4.2}$$

A filter with non-linear phase characteristics (i.e the delay not proportional with frequency) causes the phase distortion in the signal that passes through it.

Symmetric Impulse Response. A filter is said to have linear phase response if its phase response satisfies one of the following condition.

$$\theta(w) = -\alpha w \tag{4.3}$$

$$\theta(w) = \beta - \alpha w \tag{4.4}$$

where α and β are constant. If a filter satisfies the equation 4.3 it will have both constant phase/group delay response and it must have positive symmetry, and is described mathematically as:

$$h(n) = h(N-n-1) \quad \begin{cases} n = 0, 1, \ldots, (N-1)/2 \;\; (N : Odd) \\ n = 0, 1, \ldots, (N/2) - 1 \;\; (N : Even) \end{cases} \tag{4.5}$$

(4.6)

$$\alpha = (N-1)/2 \tag{4.7}$$

Whereas, if the equation 4.4 of linear phase of the filter is satisfied then it will have only constant group delay. In this case the impulse response have negative or anti symmetry, which is expressed as:

$$h(n) = -h(N-n-1) \tag{4.8}$$

$$\alpha = (N-1)/2 \tag{4.9}$$

$$\beta = pi/2 \tag{4.10}$$

4.1 Digital Filter

There are two classes of digital filters i.e. finite impulse response (FIR) and an infinite impulse response (IIR) filter. The question of economics also arise in implementation of digital filter. The economic concerns are mainly measure interms of hardware complexity, chip area, and computational speed, If we put aside the linear consideration in IIR filters then it can be the best choice, but since many application does require least phase distortion in passband and also the fact that FIR filters have been supported by special purpose DSP which have multiplier and accumulator that help to reduce the computational speed of higher order FIR filters. The FIR filters have mainly been analyzed for algorithmic devolopment and for simulations due to the reason that the channels of the WLAN and UMTS are compactly placed in the spectrum and the phase distortion in the passband can cause severe interference.

4.1.1 FIR Filters

The brief summary of the methods for finding the FIR filter coefficients is explined below. Generally there are three methods for finding the FIR filter coefficients

1. Window Method
2. Optimal Method
3. Frequency Selective Method

The most common choice among the methods for finding the FIR filter coefficients is a *window method*. The flow of this method for calculating the filter coefficients is:

- Specify the desired frequency responsse of the filter $H_d(w)$.
- Obtain the impulse response $h_d(n)$ of the desired filter by taking the inverse Fourier transform. This impulse response tends towards the infinity.
- To truncate the infinite impulse response, an appropriate window function that satisfies the passband and stopband attenuation specifications, is multiplied with the desired impulse response to determine the number of filter coefficients.

The window method is a most common choice, as it requires the least computational requirements, simple, and is easy to understand, but it has some limitations:

Lack Of Flexibility. Both the peaks of passband and stopband are equal, so the designer may end up with either too small passband ripples or too larger stopband attenuation. Therefore the designer does not have a freedom to select the parameters according to its requirement.

Approximation Issues. The window method does not provide individual control over the approximation error in different bands. The approximation of the desired frequency response have peaks in the passband ripples near the band edges and decreases away from the band edges.

The better approximation of the desired frequency response can be achieved by evenly distributed ripples in the passband or stopband that oscillates between $1 + \delta_p$ and $1 - \delta_p$, in the passband and between 0 to δ_s,in the stopband. where δ_p and δ_s are the passband and the stopband ripples respectively. The *Optimal filter* is the one which provides such an approximation with equal ripple design and is known as *equiripple design*. The above discussion is shown in Figure 4.1.

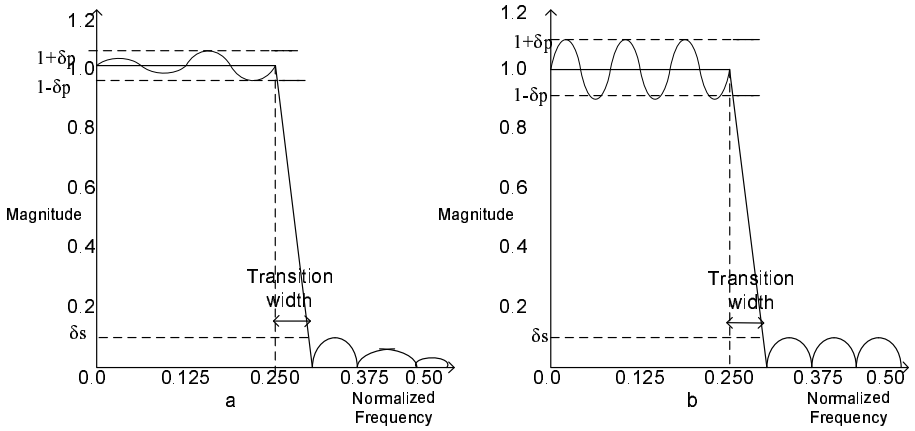


Figure 4.1: Comparision of the frequency response of a low pass filter(a) the window filter and (b) the optimal filter. In (a) the ripples are larger near the band edge; and in (b) the ripples have same peaks(equiripple) in passband or stopband

The difference between the ideal and practical response can be viewed as error function:

$$E(w) = W(w)[H_d(w) - H(w)] \tag{4.11}$$

where, $w \ \epsilon \Omega$
$H_d(w)$: desired response

$H_{(w)}$: practical response

$W(w)$ is a weighting function which provides the relative error control over the different bands, for the low pass filter. It is defined as:

$$W(w) = \begin{cases} 1/K & 0 \leq w \geq w_p, \\ 1 & w_p \leq w \geq pi, \end{cases} \tag{4.12}$$

(4.13)

The objective in the optimal filter design is to determine the filter coefficients $h(n)$ such that the value of the maximum weighted approximation error $|E(w)|$ is minimized in the passband and stopband, this particular criterion used in the design procedure of Park-MaClellan algorithm is called *minimax* or *chebyshev* [J. H. McClellan and Rabiner, 1973]. Mathematically it can be expressed as:

$$\|E(w)\| = min[max\,|E(w)|] \tag{4.14}$$

The second issue of flexibility have been overcome in the optimal design by two approaches [**?**]. In the first approach, developed by Hermann and schussler N, δ_p and δ_s parametrs are fixed and w_p, w_s are variables, whereas in the second procedure from Park MaClellan N, w_p, w_s and the ratio of δ_p/δ_s parametrs are fixed and δ_p, δ_s are variables. The algorithmic details for these approaches can be found from [J. H. McClellan and Rabiner, 1973].

4.1.2 Summary of Equiripple Optimal Filter

- choose the required design specifications i.e sampling frequency, band edge and transition width (normalized form), passband ripples and stopband attenuation.
- Then calculate the order of the filter (there is different formula for the bandpass and low pass filter). The following formula calculates the order of the bandpass filter [Emmanuel C. Ifeachor, 2002].

$$N \approx \frac{D_\infty(\delta_p, \delta_s)}{\Delta F} \; - \; f(\delta_p, \delta_s)\Delta F \; + \; 1 \tag{4.15}$$

  ΔF: is a normalized transition width.

$$D_\infty(\delta_p, \delta_s) = \log(\delta_s)[b_1(\log(\delta_p))^2 + b_2(\log(\delta_p)) + b_3] + [b_4(\log(\delta_p))^2 + b_5(\log(\delta_p)) + b_6] \tag{4.16}$$

$$f(\delta_p, \delta_s) = -14.6\log(\delta_p/\delta_s) - 16.9 \tag{4.17}$$

  All the b's coeffients are approximated and are listed in [Emmanuel C. Ifeachor, 2002].

- After finding the filter taps then the proper weights for the passband and stopband can be find by the ratio of passband to stopband ripples.

4.1.3 Conclusion

A design process that permits different levels of passband and stopband ripple are required. Filters with relaxed passband ripple requirements, will require fewer coefficents, hence require fewer resources to implement. Therefore the above discussion guids to choose optimal filters for designing the band-select and bandpass filtering.

4.1.4 Design Specifications for Band-Select and Bandpass Filter

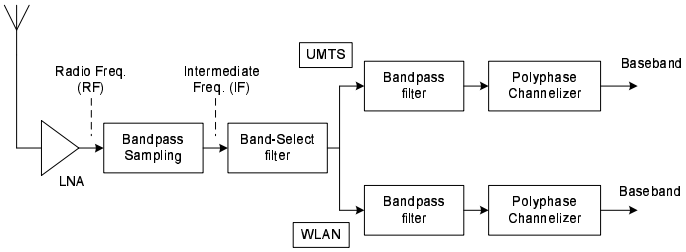


Figure 4.2: This is a complete block diagram of the system. After receiving the signal at antenna it is passed through the low noise amplifier(LNA) which reduce the noise and bost-up the signal, the next block is of bandpass sampling which samples the input at RF and bring the information down to IF. The band-select filter then select the complete band of interest followed by two channels which have individual BPF to seperate the multiple information, in this case it is two i.e. UMTS and WLAN. followed by the block of channelizer which further down convert by down-sampling and filter the IF signal.

According to the scenario shown in Figure 4.2, the band-pass sampling at RF brings the information band in IF. A band-select filter is required to select the complete spectrum of WLAN and UMTS. Now for the case of WLAN the information band is between $36MHz$ to $120MHz$ and for the case of UMTS it is between $350MHz$ to $410MHz$ with the sampling frequency of 840MHz. The frequency response along with the design specifications of the band-select filter is shown in Figure 4.3. The design specifications for the first band-select filter must have to cover the complete spectrum of the two standards, and are given by:

- Sampling freq (fs = 840MHz)
- Transition Width ($\Delta F = TW_1 + TW_2$ = 10MHz)

  TW_1 : transition width of the right half of the band.
  TW_2 : transition width of the left half of the band
- Passband edge frequencies (36 - 410)MHz
- The passband ripples (0.5 dB)
- Stopband attenuation (>60 dB)

After having the above specifications which fulfill the requirements, next step is to find out the weights for the bands. The weights are dependent on the passband and stopband deviation, the deviations in the ordinary units can be find out from passband ripples and stopband attenuation. The maximum passband ripple values for many system designs are on the order of 1% to 5%.

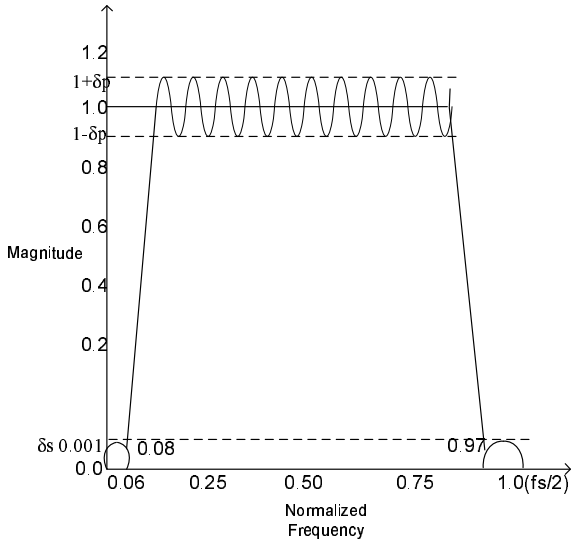


Figure 4.3: The frequency response of the band select filter. The sampling frequency is 840 MHz, the normalized edge frequencies are $PB_1/fs/2 = 36/420$ and $PB_2/fs/2 = 410/420$ with 10MHz of transition band, and the $\delta_p = 0.059$ results in filter order of 177(apprrox).

These valuse are significantly larger then the stop band ripple values which are usually in the order of 0.1% to 0.01% [Harris, 2006]. The suitable specs within this are found to be

$$0.50db \ \ ripple : 20log(1+\delta_p) \ \ = \ \ \delta_p = 0.05925 \tag{4.18}$$

$$60db \ \ attenuation : -20log(\delta_s) \ \ = \ \ \delta_s = 0.0016 \tag{4.19}$$

The ratio of δ_p to δ_s is 33.33 i.e. $105/3$. Thus we could use the weights 3 and 105 for the passband and stopband respectively. The order of this band-select filter is found by using the Equation 4.15, and is equal to 177. The design specifications for the individual bandpass filters are almost the same except that the filter order is twice for the case of WLAN as its band edges are different. The UMTS band egdes are $(350-410)$MHz, the WLAN its $(36-120)$MHz.

Cases	Passband (MHz)	Transition band (MHz)	Ripples in Passband	Stopband Attenuation
UMTS	350-410	10	1%	-60dB
WLAN	36-120	20	1%	-60dB
WLAN+UMTS	36-410	10	1%	-60dB

Table 4.1: The design specifications of bandpass and band-select filter

4.1.5 Results

The simulation results of the band-select filter followed by bandpass filters are explained in this section. The two approaches i.e. equiripple and least square for mimimizing the weighted error $E(w)$ are used in the simulations. The FIR filter design by the MatLab function 'firpm/remez' uses an equiripple Park-Maclellan algorithm which has the best approximation to the desired frequency

response in the minimax sense. However, it may not be desirable if we want to minimize the energy of the error signal, Concequently if we want to reduce the energy of the error signal $E(w)$ as much as possible in certain frequency bands, least square design is preferrable. The MatLab function 'firls' provides a measure that minimizes the error in the least square sense that is 2-norm i.e. ($\|E(w)\|_2$).

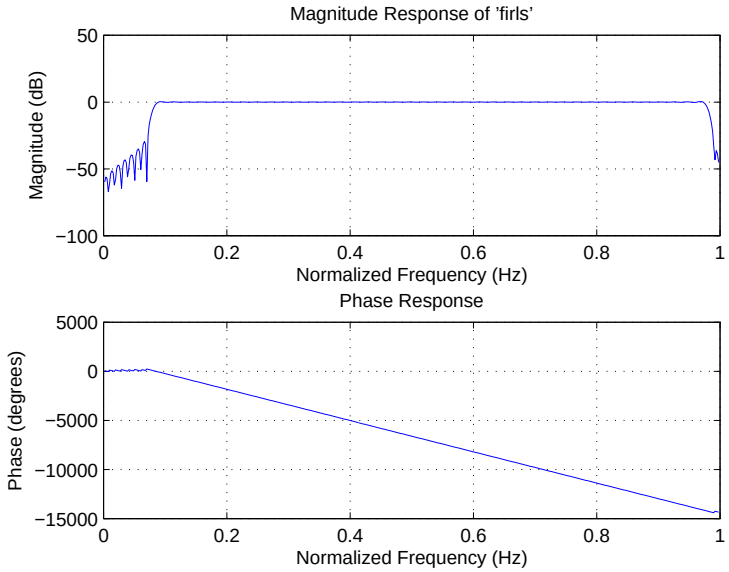


Figure 4.4: The magnitude and phase response of the 'firls'. The ripples in the passband and in the stopband are minimum in the magnitude response, with the constant group delay in the passband.

The simulation results of the band-select filter with both equiripple and with the least square design are carried out. The Figure 4.4 shows the phase and the magnitude response of the filter with 'firls'. The ripples in the passband and in the stopband are minimum which is visible from the magnitude response in Figure 4.6 and the stopband attenuation is around $50dB$ on the average. The phase reponse has a constant group delay meaning that every frequency component is facing the same delay. The result from the 'firpm/remez' equiripple design is very much similar except the fact that there are equal ripple in the pass/stop band in the magnitude response as shown in Figure 4.6. These ripples are not minimize to zero as it is in the case of 'firls'. The stop band attenuation is much better for the case of equiripple design with the same specs i.e order,transition width.

The impulse response for both the functions are shown in Figures 4.7 and 4.8. The filter coefficients obtained from the function 'firls' have an *Odd Symmetry* as $h(n) = -h(N-n-1)$, whereas the function 'firpm' have an *Even Symmetry* with $h(n) = h(N-n-1)$. So the linear phase characteristics of these FIR filters help to reduce the computation by exploiting this symmetry. Therefore the effective order of the FIR filter is $N/2$, which means that the computational resourses would also be reduced by factor of 2, and the overall computational speed would become double.

The magnitude and phase response of the individual bandpass filters that seperate the multiple-band are shown in the Figures 4.9 and 4.10. The least square algorithm provides the best fit as it

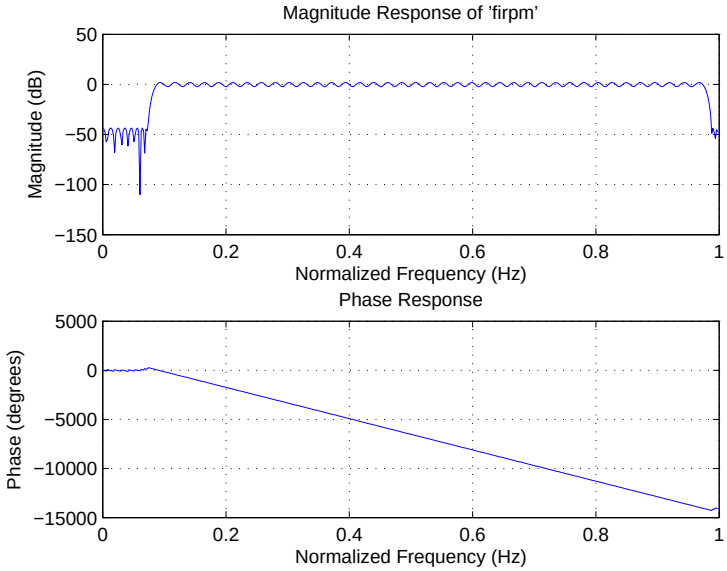


Figure 4.5: Magnitude and phase response of the 'firpm'. The equiripple design does not have a constant magnitude in the passband, and therefore it is well suited for the application where certain amount of tolerance has to met, but the phase response is linear in the passband.

.

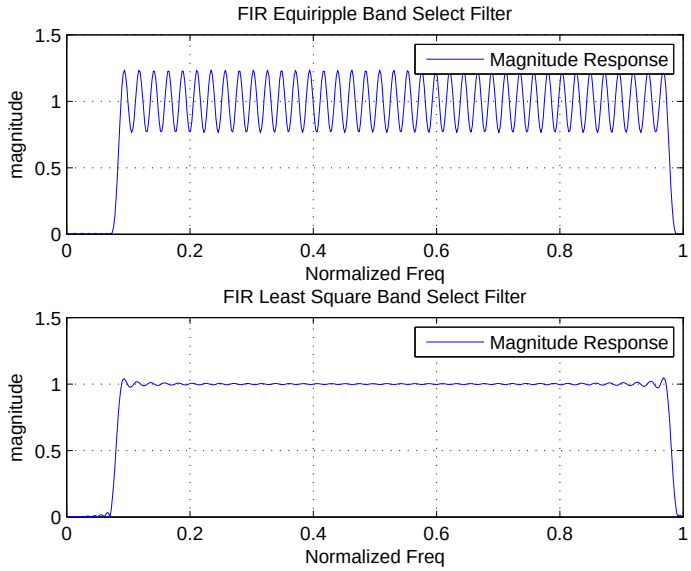


Figure 4.6: This is a magnitude response of the band-select filter from the 'firpm' and 'firls'. The equiripple design have an equal ripples in the passband whereas the least square design have minimized these ripples in the passband

.

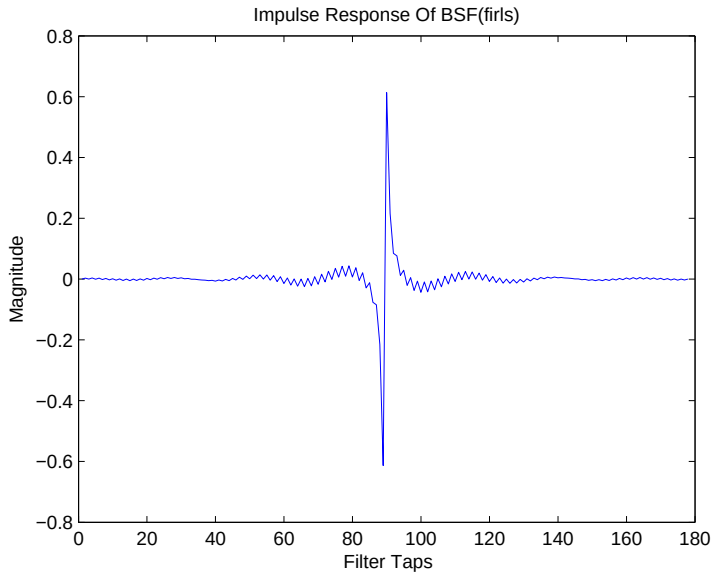


Figure 4.7: The impulse response of the band-select filter by 'firls'. It has an odd symmetry with half of the filter coefficients are same i.e h(n) = -h(N-n-1)

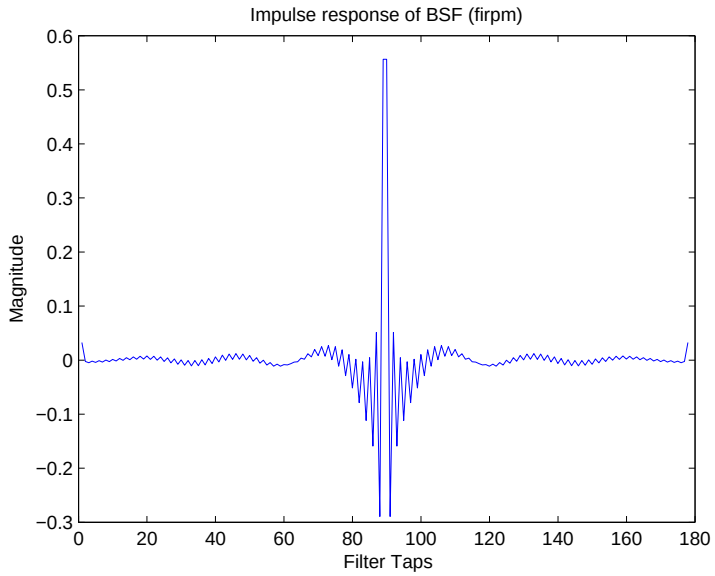


Figure 4.8: The impulse response of the band-select filter by 'firpm'. It has an even symmetry with half of the filter coefficients are same i.e h(n) = h(N-n-1).

.

minimizes the ripples in the pass/stop band, therefore the FIR bandpass filters and lowpass prototype filters for polyphase channelizers are design by using the function 'firls'. The bandpass filter for the WLAN can have a relaxed transition width as we can reduce the order, but in the broader scenario where there could be more than two standards then we may not have that relaxation. Nevertheless the transition width for the WLAN is now double i.e. 20MHz, whereas the width for the UMTS is still 10MHz as it is very close to the edge of the stectrum (fs/2).

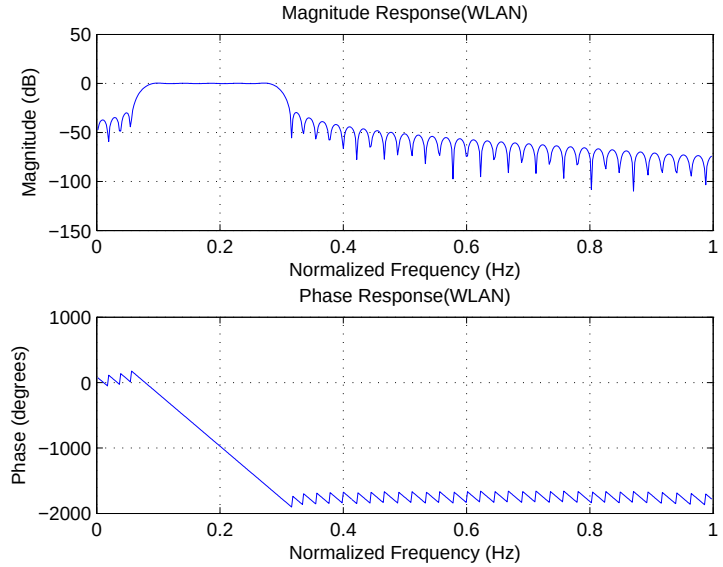


Figure 4.9: FIR Bandpass filter response for the WLAN. The normalized band edge frequencies are 0.0857 and 0.2857 with transition band of 20MHz(0.0476)

.

The output spectrum of the two standards are shown in Figure 4.11. It can be seen that the magnitude of the input signal remains the same after passing through the band-select and pandpass filters, which is a great advantage of the linear phase filter as it has a constant group delay. So the specifications for bandpass filters are justifiable since we have the desired output spectrum at the output of bandpass filters.

The ouput spectrum of the UMTS and WLAN signals shown in Figure 4.12 is an output without applying the initial filtering i.e. the band-select filter. It is evident from the output spectrum that there is **no need** of selecting the band of interest(UMTS,WLAN) at the first. So the modified system level block diagram is now shown in Figure 4.13, does not include band-select filter block.

4.2 Polyphase Channelizers

This section explains the MatLab simulations carried out on the channelizers designed in the previous chapter for WLAN and UMTS. It starts with the required filter specifications, calculation of the filter length and finally breakup in to their polyphase decomposition as in the polyphase channelizer. At the end, results are concluded.

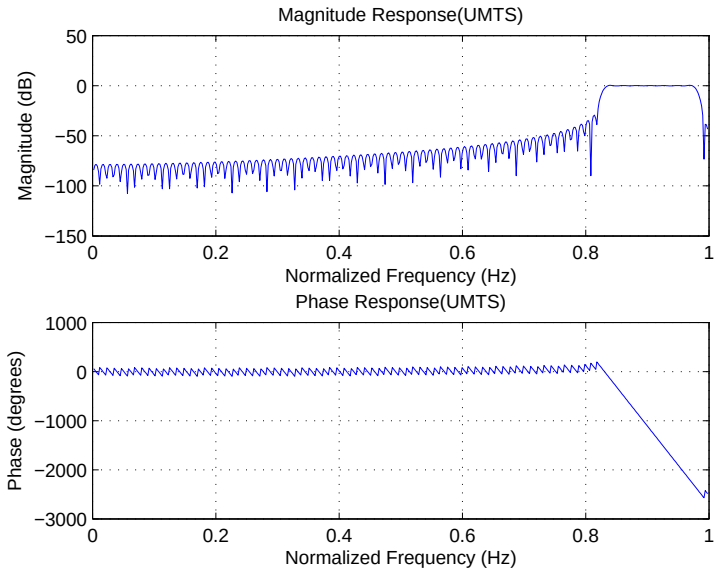


Figure 4.10: FIR Bandpass filter response for the UMTS. The normalized band edge frequencies are 0.833 and 0.976 with transition band of 10MHz(0.03)

.

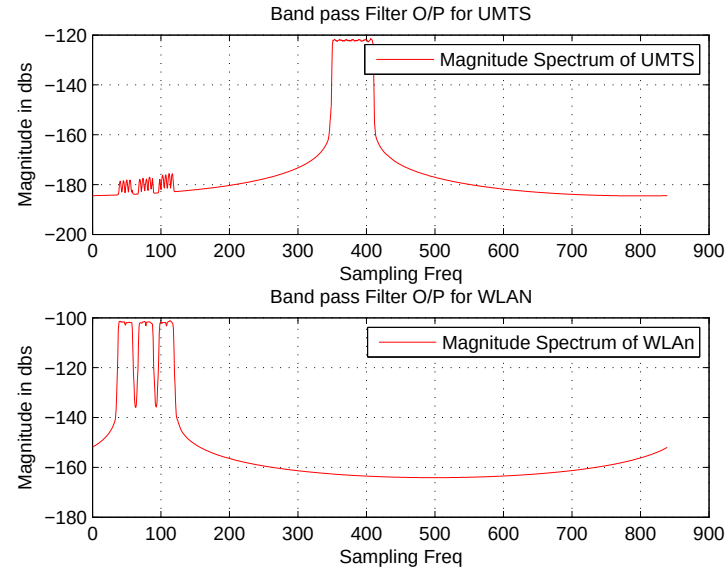


Figure 4.11: The output spectrum of the UMTS and WLAN signals after passing through the band-select and bandpass filters. Interesting point to note is that after passing through both the filters the power level of the two signals are still the same. Which shows that the designed filters have enough stopband attenuation, hence the filters can caters all the unwanted signals in the relevent spectrum

.

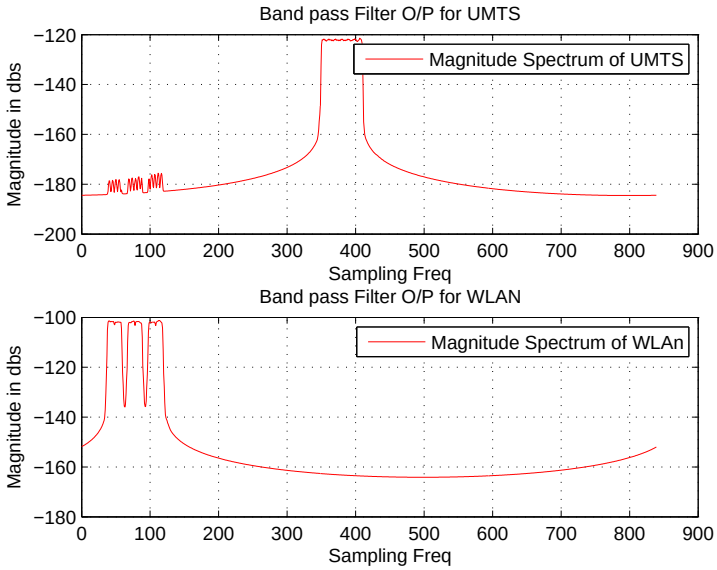


Figure 4.12: The output spectrum of the UMTS and WLAN signals after passing through the band-select and bandpass filters. Interesting point to note is that after passing through both the filters the power level of the two signals are still the same. Which shows that the designed filters have enough stopband attenuation, hence the filters can caters all the unwanted signals in the relevent spectrum

.

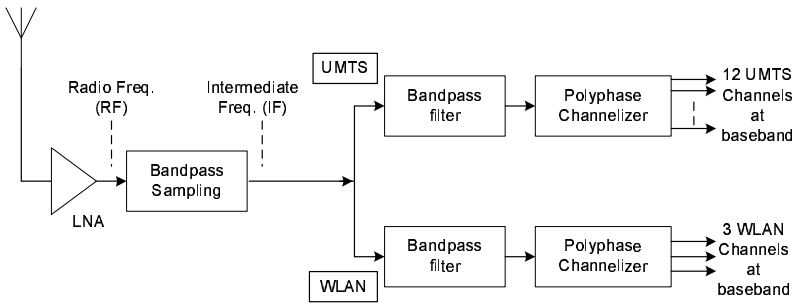


Figure 4.13: The modified system level block diagram. The band-select filter is not required at the up-front for selecting the band of information as the required performance can be achieved from the individual bandpass filters.

4.2.1 Filter Specifications

In the previous chapter, it is discussed that the polyphase channelizer requires equal spectral distribution of the input sampling frequency to form channels. The resulted channel spectral distribution for UMTS and WLAN are 5MHz and 24MHz respectively. In polyphase channelizer, the filter is designed at the baseband. The corresponding desired filters for WLAN and UMTS are shown in Figures 4.14 and 4.15. In WLAN filter specifications, the passband bandwidth is 16.6MHz and the transition bandwidth is 7.4MHz. The transition band is taken as the bandwidth between edges of the passband of the adjacent channels. Similarly in the case of UMTS, the passband bandwidth is 3.84MHz and the transition bandwidth is 1.16MHz. The filter specifications are summarized in the Table 4.2.

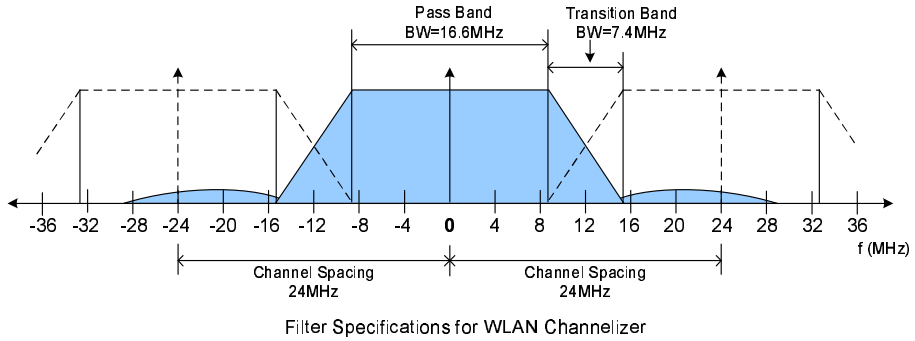


Figure 4.14: Filter specifications for WLAN channelizer. It has passband bandwidth of 16.6MHz and the transition bandwidth is 7.4MHz.

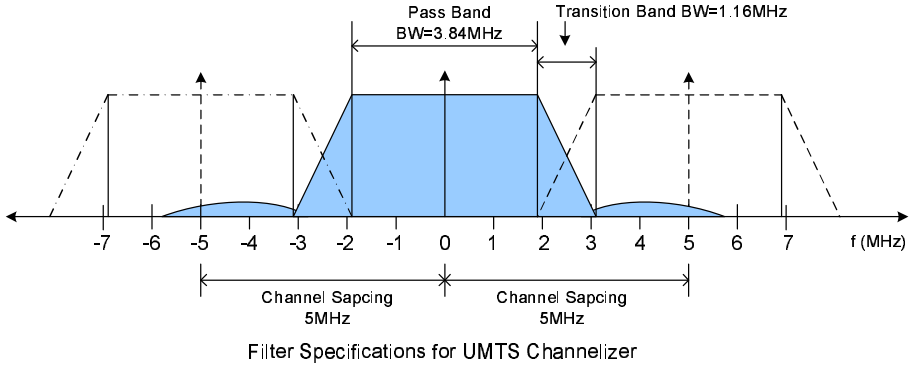


Figure 4.15: Filter specifications for UMTS channelizer. It has passband bandwidth of 3.84MHz and the transition bandwidth is 1.16MHz.

Filter specification for UMTS and WLAN

Cases	**PassBand Bandwidth** (MHz)	**Transition Bandwidth** (MHz)	**PassBand Ripples**	**Stop band Attenuation**
WLAN	16.6	7.4	1%	-60dB
UMTS	3.92	1.16	1%	-60dB

Table 4.2: Filter specifications for UMTS and WLAN

In the design process of the filters, the first task is to determine the filter length. The design is based on equirripple optimal method. The filter length can be approximated by the following empirical formula [Harris, 2006]:

$$N \approx \frac{f_s}{\Delta f} \frac{Atten(dB)}{22} \tag{4.20}$$

where f_s is the sampling frequency and Δf is the transition bandwidth. The MatLab expression for calculating the order of equiripple filter is:

$$FilterLength = firpmord(F, A, DEV, F_s) \tag{4.21}$$

where $'F'$ is a vector of cutoff frequencies in Hz, in ascending order between 0 and half the sampling frequency $'F_s'$ (Nyquist frequency). $'A'$ is a vector specifying the desired function's amplitude on the bands defined by $'F'$. $'DEV'$ is a vector of maximum deviations or ripples (in linear units) allowable for each band.

The next task is to calculate the filter coefficients for the calculated filter length and specifications as given in the Table 4.2. MatLab function *firls* is used for calculating filter coefficients

$$B = firls(N, F, A) \tag{4.22}$$

which returns a length N+1 linear phase (real, symmetric coefficients) FIR filter which has the best approximation to the desired frequency response described by F and A in the least square sense. F is a vector of frequency band edges in pairs, in ascending order between 0 and 1. 1 corresponds to the Nyquist frequency or half the sampling frequency. A is a real vector the same size as F which specifies the desired amplitude of the frequency response of the resultant filter B.

By using the MatLab expression, the corresponding filter length for WLAN comes out 41. For polyphase decomposition as in the case of polyphase channelizer for WLAN, the number of channels is 5, which is also polyphase decomposition number. In order to have a integer number of co-efficients in each of the decomposed sub-filters, the order is increased to 50, which corresponds to 10 coefficients in each of 5 sub-filters. The impluse response and frequency response of the prototype filter is shown in Figure 4.16.

The composite complex signal consisting of three WLAN channels is generated by adding the exponentials together, is not the original WLAN signal that has been used in the bandpass filtering. The reason is that we need to have all of the WLAN channels and in the original signal we have only one of them. The purpose is to illustrate the functionality of Polyphase channelizer with multiple channels. This is also the case for composite complex signal consisting of the twelve UMTS channels. The generated complex signal consisting of the three WLAN channels shown in Figure 4.17. It shows three channels centered at frequencies of -42, -12 and 48 MHz occupying a bandwidth of 16.6MHz (useful bandwidth). The spectral placing is same as it would be in the original WLAN signal.

The composite signal is fed to the polyphase channelizer for WLAN, along with the parameters of channel number k and channel offset s to extract the desired channel. The channel centered

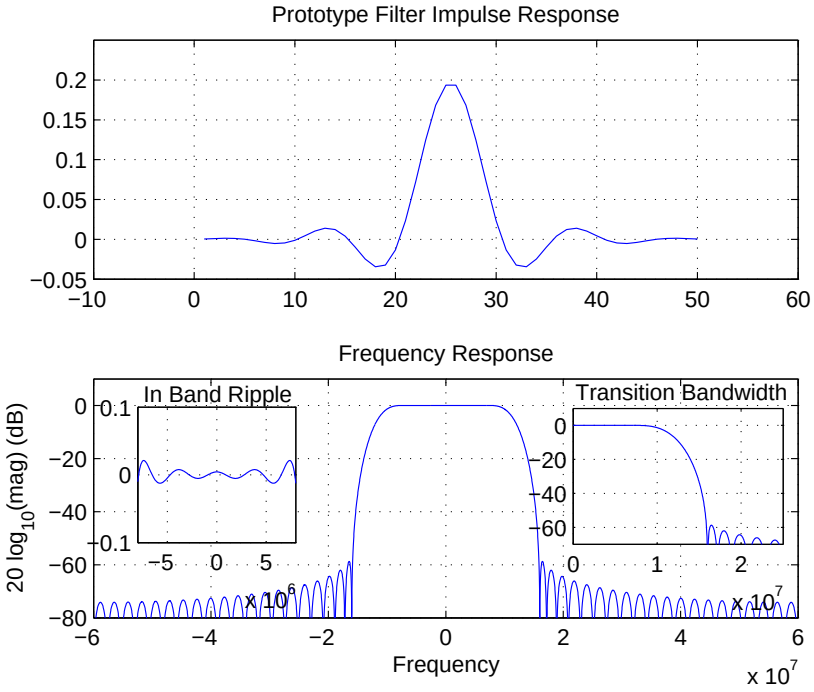


Figure 4.16: Impluse response and frequency of the prototype filter for WLAN Channelizer

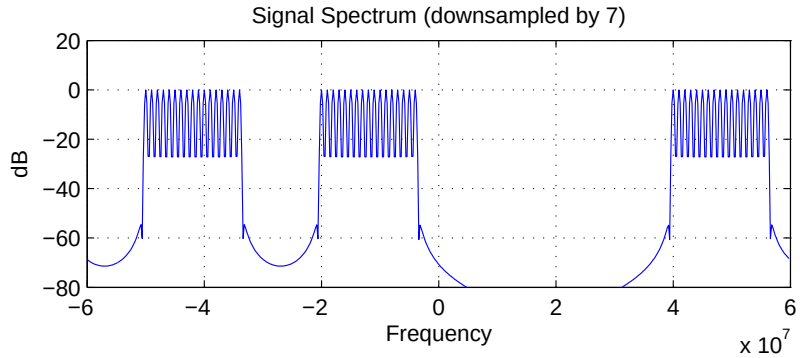


Figure 4.17: Generated complex signal of the WLAN channels. It shows three channels centered at frequencies of -42, -12 and 48 MHz, occupying a bandwidth of 16.6MHz. The spectral placing is same as it would be in the original WLAN signal.

at 48MHz corresponds to k=2, and s=0, and its output which is downconverted to baseband and downsampled to the required sampling rate of 20MHz is shown in Figure 4.21.

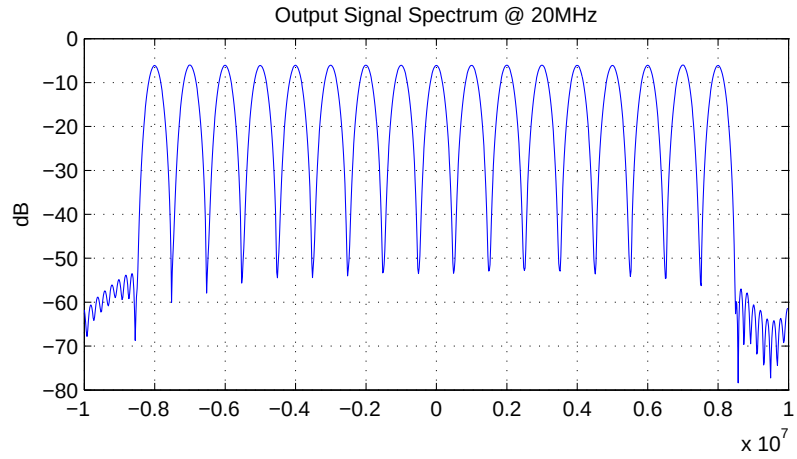


Figure 4.18: The channel centered at 48MHz corresponds to k=2, and s=0, is downconverted to baseband and downsampled to the required sampling rate of 20MHz.

In the polyphase channelizer for UMTS, a downsampling factor of 17/10 is required, which is achieved by simultaneously upsampling by 10 and downsampling by 17 in the polyphase commutator. This upsampling of factor 10 creates 10 copies of the signal. Thus, the filter as shown in Figure 4.15 has to be designed on the new sampling frequency, which is $10 \times 105 = 1050MHz$.

The corresponding filter length at frequency 1050MHz comes out 2301. For polyphase decomposition as in the case of polyphase channelizer for UMTS, the number of channels is 21, which is also the polyphase decomposition number. In order to have a integer number of co-efficients in each of the decomposed sub-filters, the order is increased to 2520, which corresponds to 12 coefficients in each of 21 sub-filters. The impluse response and frequency response of the prototype filter is shown in Figure 4.19.

The generated composite signal consisting of the UMTS channels and the signal downsampled by factor 8, is shown in Figure 4.20. It shows 12 channels centered at frequencies of (15,20,25....70)MHz occupying a bandwidth of 3.84MHz. The spectral placing is same as it would be in the original UMTS signal.

The composite signal is fed to the polyphase channelizer for UMTS, along with the parameters of channel number k and channel offset s to extract the desired channel. The channel centered at 35 MHz corresponding to k=7 and s=2, is downconverted to base band and downsampled to the required sampling rate of 61.77MHz ($\tilde{6}$1.44MHz) as shown in figure 4.21.

4.3 Conclusion

The composite complex signal consisting of three WLAN channels is generated by adding the exponentials together, is not the original WLAN signal that has been used in the bandpass filtering. The reason is that we need to have all of the WLAN channels and in the original signal we have

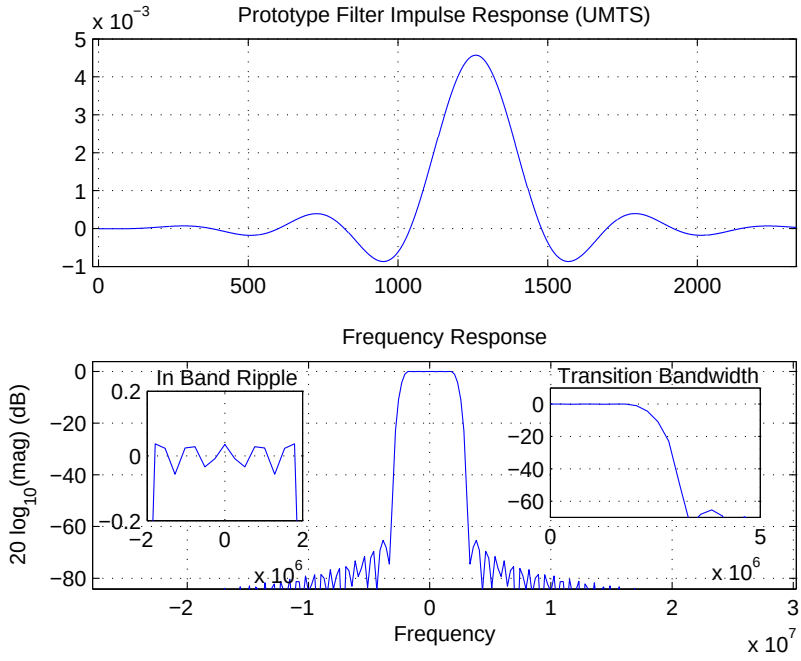


Figure 4.19: Impluse response and frequency of the prototype filter for UMTS Channelizer

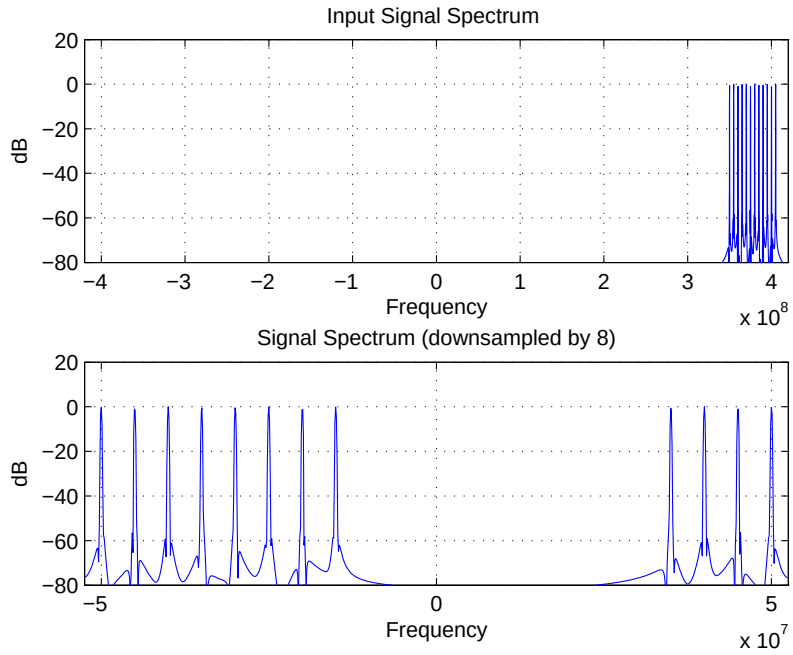


Figure 4.20: Generated composite signal of the UMTS channels (Top Figure). The bottom figure shows the composite signal downsamped by factor 8. 12 channels are translated to center frequencies of (15,20,25....70)MHz occupying a bandwidth of 3.84MHz. The spectral placing is same as it would be in the original UMTS signal.

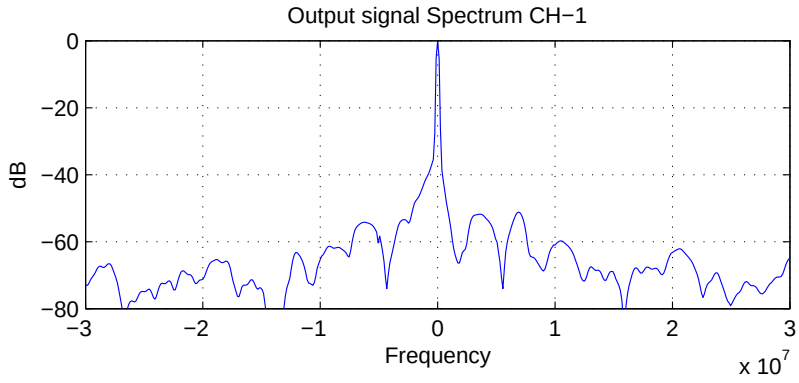


Figure 4.21: The channel centered at 35 MHz corresponding to k=7 and s=2, is downconverted to base band and downsampled to the required sampling rate of 61.77MHz ($\tilde{6}$1.44MHz).

only one of them. The purpose is to illustrate the functionality of Polyphase channelizer with multiple channels. This is also the case for composite complex signal consisting of the twelve UMTS channels.

The prototype filter for WLAN has 50 taps which are partitioned into 5 polyphase sub-filters, so that each sub-filter has 10 coefficients, whereas the prototype filter for UMTS has 2520 taps which are partitioned into 210 polyphase sub-filters, so that each sub-filter has 12 coefficients. Only one tenth of the sub-filters i.e. 21 out of 210 are used at a time. This saves the processing panelty due to embedded upsampling process. The recovered signals at the baseband has 50dB of dynamic range.

CHAPTER 5

IMPLEMENTATION ANALYSIS

This chapter illustrates the hardware design of the polyphase filters. It starts with their theoretical complexity analysis, followed by the filter structures and some of their optimizations. Then, it focuses on the basic FIR filter struture, illustrating some of the designs. Finally the design based on the defined cost-function is selected for the final implementation to the target platform.

5.1 Polyphase Filter Structure

A M-path polyphase filter consists of M parallel sub-filters. The data is fed by a commutator and the ouput is taken after the DFT (FFT) operation as shown in figure 5.1.

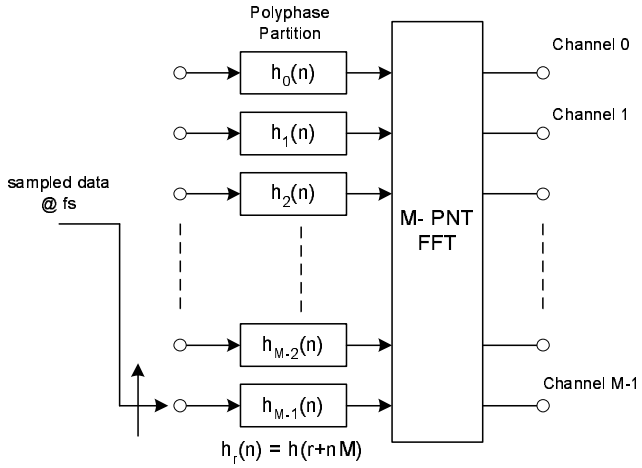


Figure 5.1: A M-path polyphase filter consists of M parallel sub-filters, which are the partitioning of the prototype filter. The data is fed by a commutator and the ouput is taken after the DFT (FFT) operation.

The commutator is a M-1 sampler, feeding data to each of sub-filters that operate at M-times the reduced rate than the incoming sampling rate, as shown in the Figure 5.2. The DFT block construct the individual channels from the downsampled data.

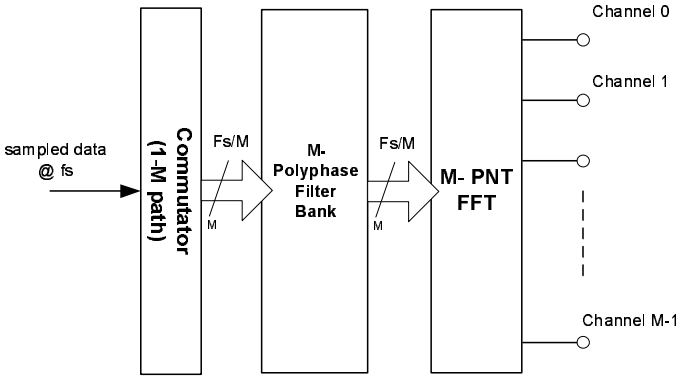


Figure 5.2: The commutator is a M-1 sampler, feeding data to each of sub-filters that operate at M-times the reduced rate than the incoming sampling rate.

The commutator is a demultiplexer that splits the incoming data to M paths. The demultiplexer is clocked at the same rate as the incoming sampling rate. Each demultiplexed output has M times reduced data rate. DFT(FFT) block is used for constructing multiple channel, but in the system design, we focused one one of the channels only. The polyphase channelizers for the UMTS and WLAN are shown in Figures 5.3 and 5.4.

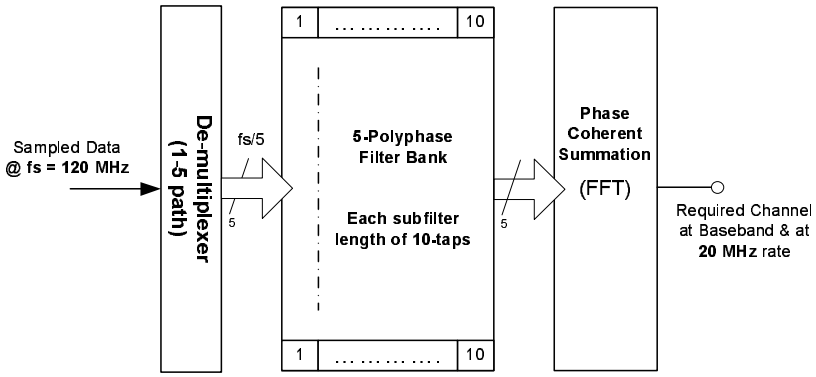


Figure 5.3: WLAN channelizer: Input sampling rate is 120MHz. The commutator is of length 5, which is same as the number of the channels. A down-sampling rate of 6 is embedded in the polyphase commutator structure to have an output rate of 20MHz from 120MHz of input.

In case of WLAN, the filter order for non-partioned filter comes out to be 50 [from Simulation chapter]. So partitioning polyphase filter into 5, each of the sub-filters has 10 coefficients. Whereas in the case of UMTS channelizer, the filter order for non-partitioned filter comes out to be 2520 [from Simulation chapter], so partitioning polyphase filter into 210, each of the sub-filters has 12 coefficients. UMTS channelizer has an upsampling factor of 10, which increases the filter length by 10 times, by the way only one tenth of the coefficents ($1/10^{th}$ of 2520) are used at a time, which saves the processing palenty.

A complexity analysis is carried out to have the required numbers of multipliers, adder/subtractors, and registers for the polyphase filters. Let f_s be the input sampling frequency, N be the length of the non-partitioned filter, and M be the number of polyphase sub-filters (same as number

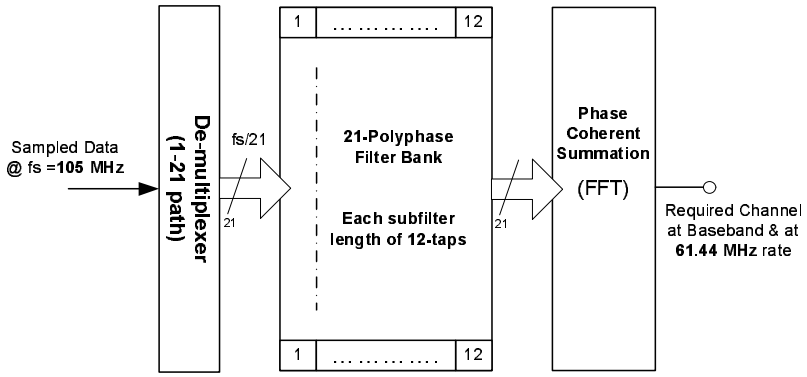


Figure 5.4: UMTS channelizer: Input sampling rate is 105MHz. The commutator is of length 21, which is same as the number of the channels. A sampling-conversion rate of 17/10 is embedded in the polyphase commutator structure to have an output rate of 61.44MHz from 105MHz of input.

of channels), then the length of each sub-filter becomes N/M. For WLAN channelizer, each of the sub-filters of length (N/M) 10-tapes require (N/M) 10 multipliers, $((N/M)-1)$ 9 adders and $((L/M)-1)$ 9 registers that results in overall requirement of (10×5) multipliers, (9×5) adders and (9×5) registers for (M) 5 polyphase sub-filters. Whereas in the case of UMTS, each of the sub-filters of length (N/M) 12-tapes require (N/M) 12 multipliers, $((N/M)-1)$ 11 adders and $((N/M)-1)$ 11 registers that results in overall requirement of (12×21) multipliers, (11×21) adders and (11×21) registers for (M) 21 polyphase sub-filters. These results are tabulated in the table 5.1.

Complexity Analysis for WLAN and UMTS polyphase filter banks

Cases	**Multipliers**	**Adders**	**Registers**
WLAN (N=50, M=5)	(N/M)x M	((N/M)-1)x M	((N/M)-1)x M
(Direct form)	50	45	45
UMTS (N=252, M=21)	(N/M)x M	((N/M)-1)x M	((N/M)-1)x M
(Direct form)	252	231	231

Table 5.1: Complexity Analysis for WLAN and UMTS polyphase filter banks, in terms of multipliers, adders, and registers.

5.2 Symmetric Structure

The filters used for polyphase filter bank are symmetric. This means that the first $N/2$ and the last $N/2$ coefficients are the same, but in reverse order. By exploiting this symmetry of the filter bank, the number of coefficient multipliers can be reduced [Raghu Rao,].

This symmetry can be used in polyphase filter bank by re-structuring the filter as illustrated in Figure 5.5. By examining the filter structures for the first and the last sub-filters, it is concluded that the coefficients multiplied are same but used in the reverse order.

The two filters - the first and the last sub-filters, shown in figure 5.6 have their coefficients multiplied in the reverse order. These two sub-filters can be combined to have a single sub-filter

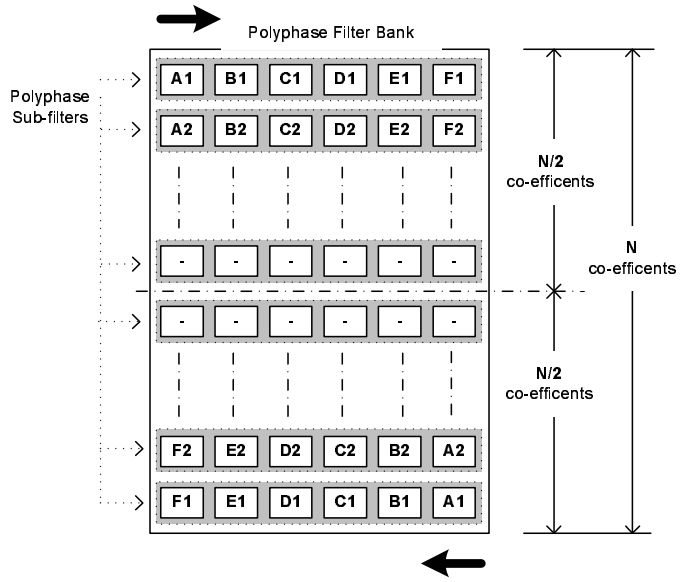


Figure 5.5: Polyphase symmetric structure: First $N/2$ and the last $N/2$ coefficients are the same, but in reverse order. By using this symmetry, the number of coefficient multipliers can be reduced [Raghu Rao,].

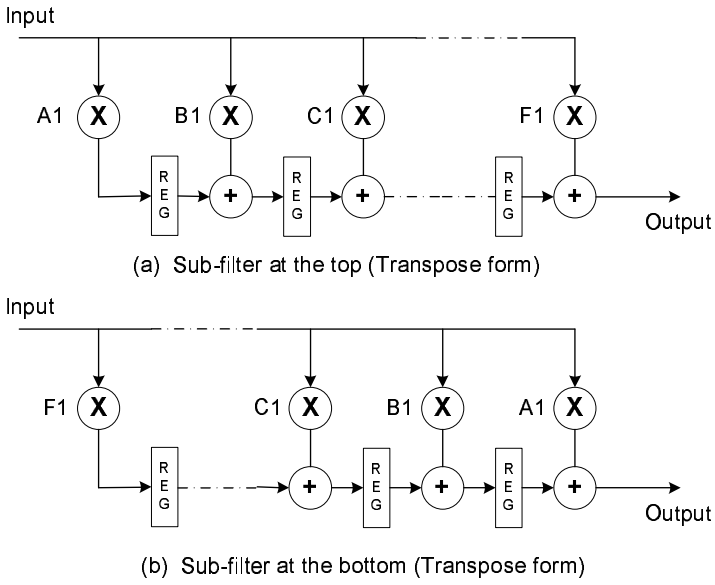


Figure 5.6: Filter structures for the first and the last sub-filters. The coefficients multiplied are same but used in the reverse order.

as shown in figure 5.7, that uses the same coefficient multipliers (multipliers are shared).

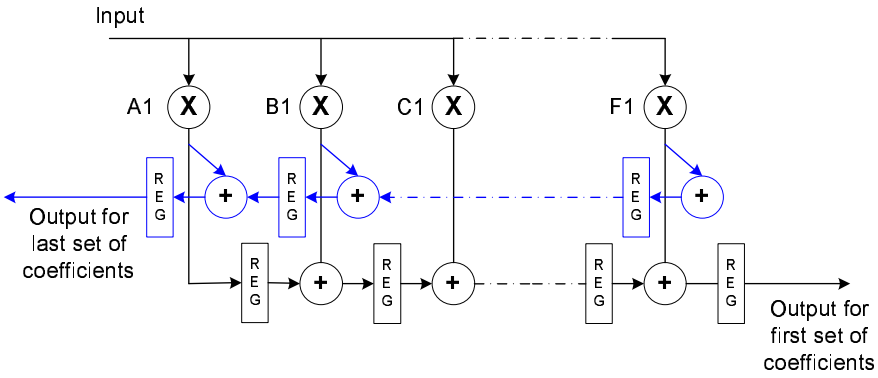


Figure 5.7: Combined sub-filter for the top and the bottom sub-filters of the polyphase filter bank. The multipliers become shared due to the filter symmetry.

In the normal way, the commutator starts feeding from the bottom sub-filter and move up to the first sub-filter. But in structure exploiting the symmetry, the commutator starts feeding from the bottom sub-filter and move up to the first sub-filter and then again start from the first sub-filter to the bottom sub-filter. Since each filter convolves alternate samples, giving two outputs, one a convolution of even samples and the other a convolution of odd samples, so it also performing decimation by 2. So, the initial decimator needs to decimate only by $M/2$ instead of M. The commutator becomes half the size for this new structure. After feeding $M/2$ filters, it reverses direction as shown in the figure 5.8.

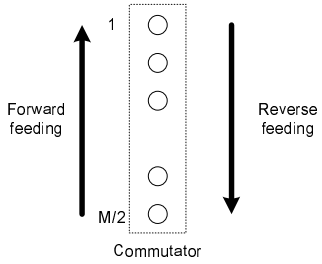


Figure 5.8: Commutator Sequence: The commutator starts feeding from the bottom sub-filter and move up to the first sub-filter and then again start from the first sub-filter to the bottom sub-filter. After feeding $M/2$ sub-filters, it reverses direction [Raghu Rao,].

The filter structure shown in figure 5.7 which has shared multipliers can further be optimized by sharing the adders as well. The new optimized filter structure having shared multipliers and adders, is shown in figure 5.9.

A complexity analysis is carried out for these three types of filter structures, both for WLAN and UMTS polyphase filter banks, which is tabulated in Tables 5.2 and 5.3.

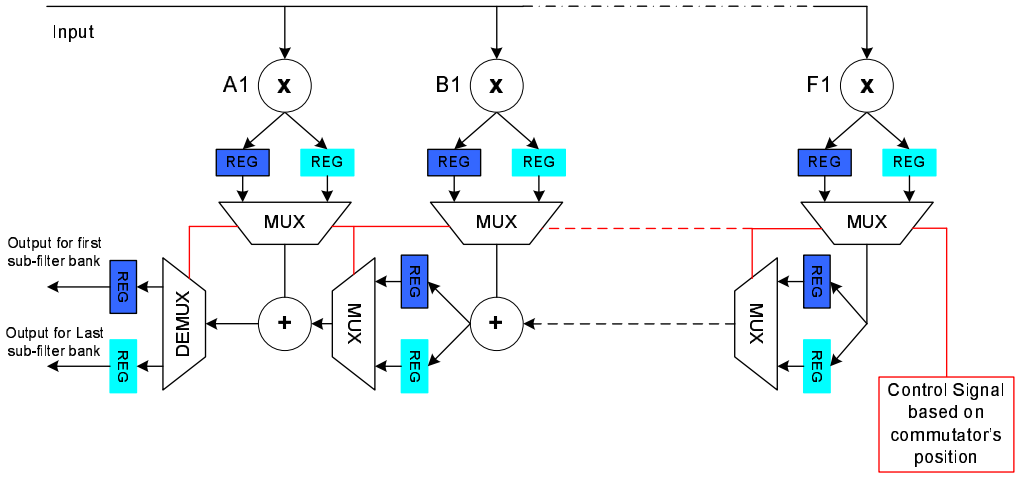


Figure 5.9: Optimized Structure: It has shared multipliers and adders. The sharing is achieved by using multiplexers and demultiplexers in the data path of the coefficient-multipliers and accumulators [Raghu Rao,].

Complexity Analysis for WLAN polyphase filter bank

Cases	**Multipliers**	**Adders**	**Registers**	**MUX**	**DEMUX**	**Clock speed**
Polyphase General (Transpose form)	(N/M)x M 50	((N/M)-1)x M 45	(N/M)x M 50	-	-	f_s/M
Optimization-I (Shared Multipliers)	((N/M)x M)/2 25	((N/M)-1)x M 45	(N/M)x M 50	-	-	$2f_s$/M
Optimization-II (Shared Multipliers & Adders)	((N/M)x M)/2 25	((N/M)x M)/2 $\approx$ 23	((N/M)x M)x2 100	((N/M)-1)x (N/M)x(M/2) $\approx$ 48	M/2 $\approx$3	$2f_s$/M

Table 5.2: Complexity Analysis for WLAN polyphase filter bank, in terms of multipliers, adders, registers, multiplexers and demultiplexers. It shows the result of basic-form, symmetric-form, and the optimized-symmetric-form. The clock requirements for symmetric-form and the optimized-symmetric-form are doubled because of the reduced commutator length.

Complexity Analysis for UMTS polyphase filter bank

Cases	**Multipliers**	**Adders**	**Registers**	**MUX**	**DEMUX**	**Clock speed**
Polyphase General (Transpose form)	(N/M)x M 252	((N/M)-1)x M 231	(N/M)x M 252	-	-	f_s/M
Optimization-I (Shared Multipliers)	((N/M)x M)/2 126	((N/M)-1)x M 231	(N/M)x M 252	-	-	$2f_s$/M
Optimization-II (Shared Multipliers & Adders)	((N/M)x M)/2 126	((N/M)x M)/2 $\approx$ 116	((N/M)x M)x2 504	((N/M)-1)x (N/M)x(M/2) $\approx$ 242	M/2 $\approx$11	$2f_s$/M

Table 5.3: Complexity Analysis for UMTS polyphase filter bank, in terms of multipliers, adders, registers, multiplexers and demultiplexers. It shows the result of basic-form, symmetric-form, and the optimized-symmetric-form. The clock requirements for symmetric-form and the optimized-symmetric-form are doubled because of the reduced commutator length.

Tables 5.2 and 5.3 shows the resource and clock speed requirements for WLAN and UMTS filter banks. It shows the result of basic-form, symmetric-form, and the optimized-symmetric-form. The clock requirements for symmetric-form and the optimized-symmetric-form are doubled because of the reduced commutator length.

5.3 Serial Polyphase Filter Bank

In M-path polyphase filter bank there are M sub-filters, from which M-1 sub-filters are unused at all the time. A more efficient implementation can be achieved by having a serial implementation of polyphase filter bank [Murphy,]. By doing so, we can get rid of unnecessary sub-filters. MAC (Multiply-Accumulate) can be implementated both in parallel and serial form. In the case of serial MAC, the system have to be clocked at N/M rate, where M is the polyphase sub-filters and N is the number of coefficeints in each of the subfilter. A serial MAC implementation structure is shown in figure 5.10 [Murphy,].

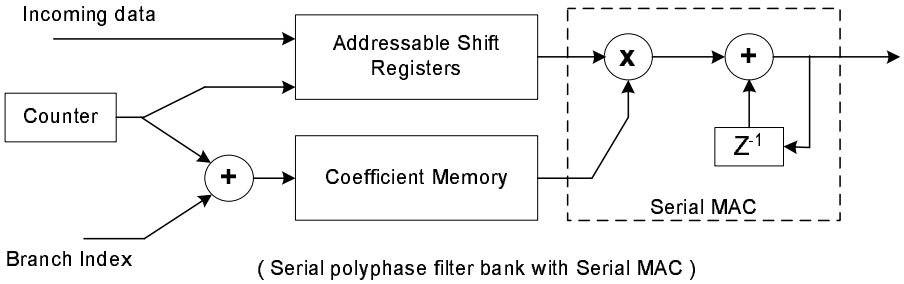


Figure 5.10: Serial polyphase filter bank structure: The incoming data to each of the sub-filters is fed to a combined block of **Addressable-Shift-Registers**. The set of data that corresponds to the individual sub-filter is accessed by the pointer addressed by the **counter**, which updates at the rate of incoming data. At any **counter** value (same as commutator position), the **Branch Index** provide the offsets for accessing filter coefficients which are multiplied to the data-set corresponding to the sub-filter at that time. The data from the **Addressable-Shift-Registers** and the coefficient from the **Coefficient Memory** are multiplied and accumulated in a serial fasion to have the final accumulated results for sub-filter at that time. This process continues for other sub-filters operation as directed by **counter** value that provides the commutator position.

The incoming data to each of the sub-filters is fed to a combined block of **Addressable-Shift-Registers**. The set of data that corresponds to the individual sub-filter is accessed by the pointer addressed by the **counter**, which updates at the rate of incoming data. At any **counter** value (same as commutator position), the **Branch Index** provide the offsets for accessing filter coefficients which are multiplied to the data-set corresponding to the sub-filter at that time. The data from the **Addressable-Shift-Registers** and the coefficient from the **Coefficient Memory** are multiplied and accumulated in a serial fasion to have the final accumulated results for sub-filter at that time. This process continues for other sub-filters operation as directed by **counter** value that provides the commutator position.

In the case of parallel MAC structure, the data and the coefficients are fed parallel for the MAC operation, which run at the same clock as the incoming data. In this case there is no need to have

offset from the **Branch Index** block to access individual coefficients, as all the coefficients are accessed and multiplied in parallel. A parallel MAC implementation structure is shown in figure 5.10.

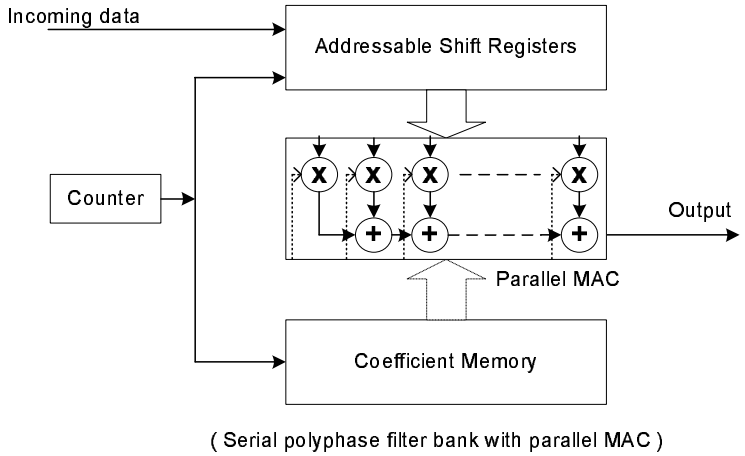


Figure 5.11: Parallel MAC structure: The data and the coefficents are fed parallel to the MAC operation, which run at the same clock as the incoming data. In this case there is no need to have offset from the **Branch Index** block to access individual coefficients as all the coefficients are accessed and multiplied in parallel.

In both of these cases, the output of the filter has to be stored in the individual registers (equal to polyphase sub-filters), which is further processed by the DFT block.

The complexity analysis for Serial Polyphase filter bank implementation with serial and parallel MAC structures is tabulated in Table 5.4.

Complexity Analysis for Serial Polyphase filter bank

Cases	Multipliers	Adders	Registers	Clock Requirement
Serial Polyphase (Serial MAC)	1	1	N	f_s x (N/M)
Serial Polyphase (Parallel MAC)	N/M	(M/N)-1	N	f_s

Table 5.4: Complexity Analysis for Serial Polyphase filter bank, interms of multipliers, adders, registers and clock speed requiremnets.

The table 5.4 shows Area-Speed trade-off between two structures. Serial polyphase structure with serial MAC requires less resources but demands high clock speed, whereas structure with parallel MAC requires large resources but low clock speed.

5.4 Conclusion

we have analyzed the structure of polyphase channelizer and presented different structural techniques to carry out the implementation. In this regard, general polyphase structure, optimized

structures - symmetric property based structure, adder shared structure, serial polyphase structures with serial and parallel MAC are considered. Complexity Analysis is carried out to choose the least expensive solution amoung them. Based on Tables 5.2, 5.3 and 5.4, serial polyphase structure with parallel MAC is selected for the final implementation. In an M channels filterbank, each sub-filter operates at 1/M of the input sample-rate (f_s), we take advantage of this property and share the multipliers and the multi-operand adder of one sub-filter among all the sub-filters in the filter bank. This results in a clock requirement to be same as input sampling rate. Thus we come up with **time-constraint** for implementation, which is equal to input sampling rate (f_s).

We have discussed the system level modifications and optimizations of the polyphase channelizer having focus on polyphase filter bank. As the basic block of the polyphase filter bank is a sub-filter, which is a FIR filter as well, so in the next sections, we look further into different implementation structures for the basic FIR filter, which later on, is used in the polyphase filter bank.

5.5 FIR Filtering

A FIR filter computes the discrete convolution of an input and a finite length filter response. This convolution can be written as

$$y[n] = \sum_{k=0}^{N-1} h[k]x[n-k] \tag{5.1}$$

where N is the length of the filter response and is referred to as the number of taps in the filter.

Several different algorithms or filtering techniques are investigated for implementing a FIR filter including:

- FIR filtering using parallel multipliers and accumulators
- A bit-level systolic array
- Distributed Arithmetic
- Fast FIR algorithms
- Frequency domain filtering
- Multiplier-less FIR filter (SOPOT)

In the next sections, each of these different types is briefly explained, followed by their structures and optimizations.

5.5.1 Parallel Multipliers and Accumulators

The most direct realization of a FIR filter is to calculate the output using parallel multipliers and accumulators (MACs). The parallel MAC structure is illustrated in Figure 5.12, and is derived directly from the FIR convolution in equation 5.1 [Alan V. Oppenheim, 1999]. In this structure,

each MAC computes the product of the delayed input and the tap's active coefficient. The outputs from each multiplier are then accumulated together to produce the filter's output.

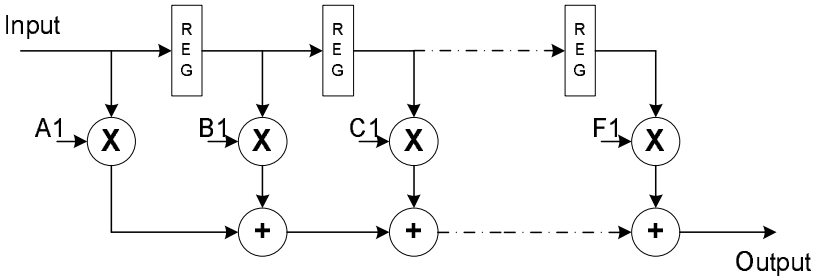


Figure 5.12: Direct realization of a FIR filter to calculate the output using parallel multipliers and accumulators (MACs). Each MAC computes the product of the delayed input and the tap's active coefficient. The outputs from each multiplier are then accumulated together to produce the filter's output.

The structure in Figure 5.12 has long combinatorial delays through the accumulation chain, so the summer tree network shown in Figure 5.13 or the transposed form shown in Figure 5.14 are often used in actual FIR computational hardware.

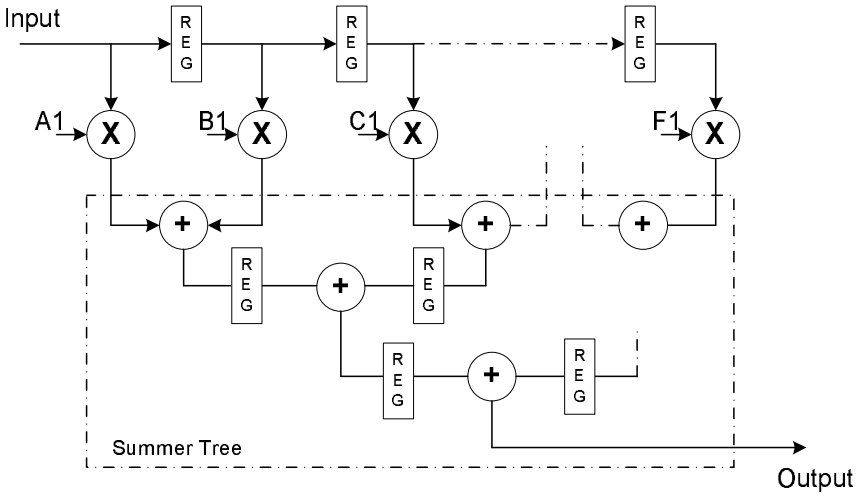


Figure 5.13: FIR filter structure using summer tree network. The structure is used to avoid long combinatorial delays through the accumulation chain.

Both forms produce the same output, but can have their accumulation chains pipelined to increase performance. The benefit of the transposed form is that each MAC communicates only with adjacent MACs, as Figure 5.14 shows. This allows the MACs to be placed in a linear systolic fashion, where adjacent MACs are placed next to each other in a line so that each MAC only has routes to and from its nearest neighbors. This maximizes the performance of the design while minimizing its area.

The tree network requires long, complicated route lengths to the inputs of each stage of adders. As the number of MACs that have been summed together for a given adder increases, the MACs

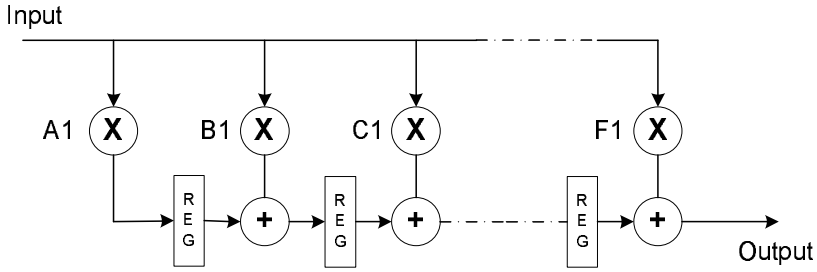


Figure 5.14: Transpose-FIR filter structure. The structure is used to avoid long combinatorial delays through the accumulation chain.

grow farther apart. This prohibits a simple linear distribution of MAC cells and slows the design's performance due to the long routes.

One problem with the transposed form of the parallel MAC filter is that it requires a large fan-out on the **input** signals, as they must connect to every MAC. To reduce this fan-out while maintaining pipelining in the accumulation chain and allowing the MACs to be placed in a linear systolic fashion, an additional stage of pipelining in both the inputs and outputs of each MAC can be introduced into the direct filter structure as shown in Figure 5.15.

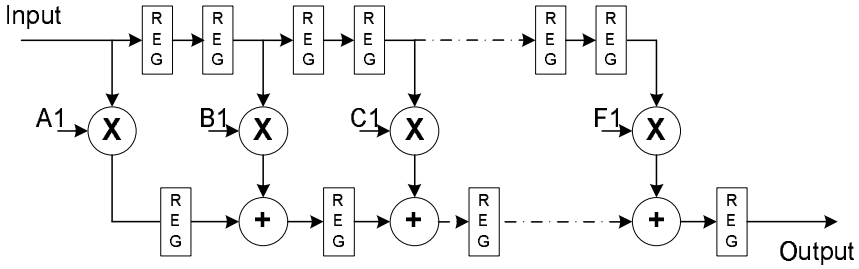


Figure 5.15: Direct filter structure having additional stage of pipelining in both the inputs and outputs of each MAC, inorder to reduce large fan-out on the **input** signal while maintaining pipelining in the accumulation chain and allowing the MACs to be placed in a linear systolic fashion.

A frequent method used to decrease the area of a parallel MAC approach to FIR filtering is to increase the number of taps computed per MAC. This is the technique used in the custom VLSI chip [Moeller and Martinez, 1999] [David R. Martinez and Teitelbaum, 2000].

A block diagram of the custom VLSI chip's architecture is shown in Figure 5.16 [Moeller and Martinez, 1999] [David R. Martinez and Teitelbaum, 2000]. It consists of 64 MAC units. Each MAC unit contains a multiplier, accumulator and intermediate storage memory, and two banks of coefficient memory. The two banks of coefficient memory allows one set of coefficients to be active and is used by the multipliers while a new set can be loaded into the other coefficient bank. Once the new set has been loaded, it can now become active, allowing the chip to instantly change from one set of coefficients to another. Each MAC, by using the accumulator and intermediate storage memory, is capable of forming the products of the current chip input and up to eight filter taps (i.e. eight coefficients). These products are accumulated together as required within the MAC, and then added

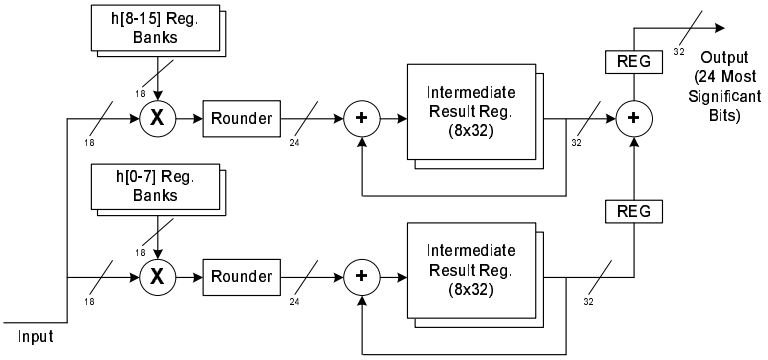


Figure 5.16: Block diagram of the custom VLSI chip's architecture. It consists of 64 MAC units. If the MACs are operating in their eight tap mode, they must run at a clock rate eight times the input sample rate so that all eight taps' products are computed each time a new input arrives. In this mode, the 64 MACs can compute a 512-tap real filter [Moeller and Martinez, 1999] [David R. Martinez and Teitelbaum, 2000].

to the other MACs' results when a new input is present. If the MACs are operating in their eight tap mode, they must run at a clock rate eight times the input sample rate so that all eight taps' products are computed each time a new input arrives. In this mode, the 64 MACs can compute a 512-tap real filter [Moeller and Martinez, 1999] [David R. Martinez and Teitelbaum, 2000].

One variation [Moeller, 1999] of this technique is shown in Figure 5.17. In this structure, a single multiplier is re-used eight times to compute the product of eight **input** values multiplied by eight coefficients for each input into the filter. An eight-word deep RAM stores the eight coefficients for the tap, and a seven-word long shift register stores the **input** values. This architecture is similar to the one used in the custom VLSI chip (Figure 5.16), except that shift registers have been used to store multiple input values for each MAC instead of RAM storing the multiplier's outputs. The result is the same except that storing multiple inputs per tap requires less memory since the inputs are only 16-bits long versus the 24-bit multiplier outputs.

The shift registers are loaded with a new value at the beginning of each eight-clock cycle. The shift registers are then fed their outputs back into their inputs for the next seven clock cycles. This moves the input that was shifted into the register at the beginning of the eight clock cycles to the second shift register position at the beginning of the next eight clock cycles so that the next new value is loaded into the first position. After eight clock cycles, this input becomes the new value to the next MAC's shift registers. At the same time, the shift registers are arranged so that eight consecutive input values are supplied to the multiplier to be multiplied by eight consecutive coefficient values. These will be added together by the final accumulator shown in Figure 5.17.

The output of the first two MACs' shift registers and coefficient registers (i.e. the multiplier inputs) are shown in Table 5.5, for two-tap MACs. With eight-tap MACs, the movement of inputs through the shift registers produces the same effect with 64 multipliers as if 512 multipliers had been used with a clock rate equal to the sample rate.

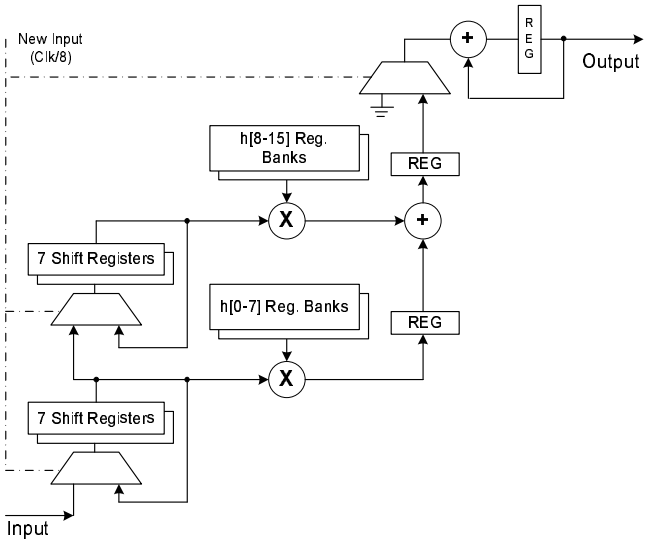


Figure 5.17: A variation [Moeller, 1999] of custom VLSI technique. A single multiplier is re-used eight times to compute the product of eight **input** values multiplied by eight coefficients for each input into the filter. An eight-word deep RAM stores the eight coefficients for the tap, and a seven-word long shift register stores the **input** values.

	MAC 0				MAC 1				
Clock	MAC	Shift Register		Coeff	MAC	Shift Register		Coeff	Final
Cycle	Input	In	Out	Output	Input	In	Out	Output	Output
0	x[0]	x[0]		h[1]				h[3]	
1		x[0]	x[0]	h[0]				h[2]	x[0]*h[0]
2	x[1]	x[1]	x[0]	h[1]				h[3]	x[0]*h[1] +
3		x[1]	x[1]	h[0]				h[2]	x[1]*h[0]
4	x[2]	x[2]	x[1]	h[1]	x[0]	x[0]		h[3]	x[0]*h[2] +
5		x[2]	x[2]	h[0]		x[0]	x[0]	h[2]	x[1]*h[1] + x[2]*h[0]
6	x[3]	x[3]	x[2]	h[1]	x[1]	x[1]	x[0]	h[3]	x[0]*h[3] + x[1]*h[2]
7		x[3]	x[3]	h[0]		x[1]	x[1]	h[2]	x[2]*h[1] + x[3]*h[0]

Table 5.5: The output of the first two MACs' shift registers and coefficient registers (i.e. the multiplier inputs) for two-tap MACs.

5.5.2 Bit-level Systolic Array

It is a fully-efficient bit-level systolic structure by [Chin-Liang Wang and Chen, 1988]. With this technique, single-bit processors compute each tap's multiplication partial products and accumulate tap outputs together in a systolic array. As inputs propagate through the array, filtered outputs are produced. The systolic nature of this approach lends itself well to VLSI, and would be ideal for a FPGA if it was area-efficient, as it would limit the routing requirements in the FPGA to local connections between CLBs. The details of the mathematical derivations can be found in [Chin-Liang Wang and Chen, 1988]

This technique did not turn out to be efficient in a FPGA architecture (detials are in the next chapter). It is presented to show the differences between architectures optimized for a FPGA's coarse-grained structure versus architectures optimized at the transistor level for a VLSI approach [Moeller, 1999].

5.5.3 Distributed Arithmetic (DA)

This section describes Distributed Arithmetic (DA), and is based on information presented in [Xilinx,] [Moeller and Martinez, 1999].

Distributed Arithmetic works by distributing the bit arithmetic of the sum-of-products (also called the vector dot product) used to calculate the FIR filter output given in Equation5.1. This equation will be re-written as

$$y[n] = \sum_{k=0}^{N-1} A_k X_k(n) \tag{5.2}$$

where $A_k = h[k]$.

A FIR filter is typically implemented with some variation of Figure 5.12 or Figure 5.14, where a summation of the results of N multipliers each calculating an $A_k X_k(n)$ product produce the output for a given n input.

The number format used in the custom VLSI chip and in the FPGA design is 2's complement fractional fixed-point [Moeller, 1999]. In this format, the binary point is to the right of the most significant bit so that the most significant bit of a number represents -1, and each subsequent bit represents a power of $1/2$. Using this format, the variable may be written as

$$X_k = -X_{k0} + \sum_{b=1}^{B-1} X_{kb} 2^{-b} \tag{5.3}$$

where x_{kb} is the bth bit of x_k , and B is the number of bits in the input variable.

Substituting Equation 5.3 into Equation 5.2 gives (n has been dropped, as we are only concerned with a single given output sample)

$$y = \sum_{k=0}^{N-1} A_k \left(-X_{k0} + \sum_{b=1}^{B-1} X_{kb} 2^{-b} \right) \tag{5.4}$$

which when rewritten, gives

$$y = -\sum_{k=0}^{N-1} X_{k0} A_k + \sum_{b=1}^{B-1} 2^{-b} \left(\sum_{k=0}^{N-1} X_{kb} A_k \right) \tag{5.5}$$

Explicitly writing the summation results in the following DA equation:

$$\begin{aligned} y &= -\left[X_{00}A_0 + X_{10}A_1 + ... + X_{(N-1)0}A_{N-1}\right] \\ &= +\left[X_{01}A_0 + X_{11}A_1 + ... + X_{(N-1)1}A_{N-1}\right] 2^{-1} \\ &\quad .... \\ &= +\left[X_{0(B-1)}A_0 + X_{1(B-1)}A_1 + ... + X_{(N-1)(B-1)}A_{N-1}\right] 2^{B-1} \end{aligned} \tag{5.6}$$

Each multiplication of a X_{kb} term and an A_k term is the product of a coefficient word with an input bit. This can be implemented by using an AND gate between each bit of the coefficient word and the input bit. Each scaling factor 2^{-i} can be implemented by shifting the data to be scaled right i bits. Equation 5.6, therefore becomes the summation of the scaled summation of a series of AND gates. This operation could be performed in parallel or bit-serially, where on each clock cycle a single bit from every X_k is multiplied by the corresponding A_k, forming one bracketed term in Equation 5.6. These partial products are then accumulated together with the appropriate scaling to produce a final multiplier output. An example of bit-serial multiplication for a single coefficient and input is shown in figure 5.18 [Moeller and Martinez, 1999].

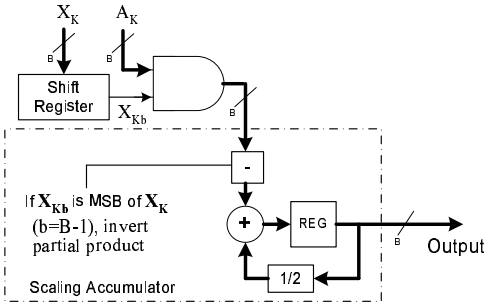


Figure 5.18: Bit-serial multiplication: The partial products obtained by multiplying each bit of input with the filter coefficient through AND gate, are accumulated together with the appropriate scaling to produce a final multiplier output.

Figure 5.18 illustrates, on each clock cycle a single partial product consisting of one bit of the input multiplied by the coefficient is produced. This partial product is then added to an accumulating sum of partial products, which has been shifted right one bit (multiplied by 1/2). This

operation produces the following result for a four-bit input (with each term in parenthesis being computed each clock cycle):

$$y_k = ((\ (X_{k3}A_k)2^{-1} + X_{k2}A_k)2^{-1} + X_{k1}A_k)2^{-1} - X_{k0}A_k \tag{5.7}$$

which when simplified, gives

$$y_k = X_{k3}A_k2^{-3} + X_{k2}A_k2^{-2} + X_{k1}A_k2^{-1} - X_{k0}A_k \tag{5.8}$$

and finally, results in the product of the input and the coeffcient after four clock cycles:

$$y_k = X_kA_k \tag{5.9}$$

To maintain full-precision, the accumulator must be able to hold the entire multiplied result. The number of bits required is the number of bits in the input data plus the number of bits in the coefficients.

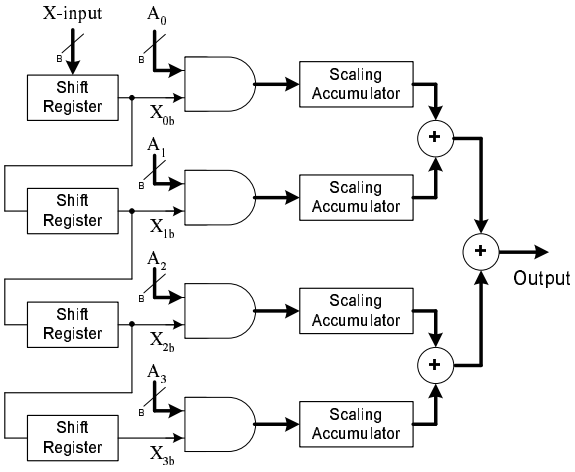


Figure 5.19: FIR filter with Bit-serial multiplier structure where a parallel input to the FIR is converted into a serial stream of bits. Data is loaded to first Bit-serial multiplier, and on every clock cycle it serial shift through the next next tap. The outputs from each of the scaling accumulators are added together to have the final output.

The MAC structure in Figure 5.12 (direct realization of FIR filter) can be implemented with the bit-serial multiplier in Figure 5.18 as shown in Figure 5.19, where a parallel input to the FIR is converted into a serial stream of bits [Moeller and Martinez, 1999]. On each clock cycle, one bit of the input is presented to the first scaling accumulator, and placed into a serial shift register for the next tap, so that each tap's input sample is presented to each scaling accumulator in a serial fashion. Each tap takes B (no. of bits for input data) clock cycle to produce a product, which are then summed together to produce an output sample. However, as the bracketed terms in Equation 5.6 shows, the partial products computed by each AND gate can be summed together first, then accumulated with scaling. In this method, one bracketed term in Equation 5.6 is computed each clock cycle, so B clock cycles are still required, yet each tap requires less hardware, since

only one master scaling accumulator is now necessary. The new FIR structure is shown in Figure 5.20 [Moeller and Martinez, 1999].

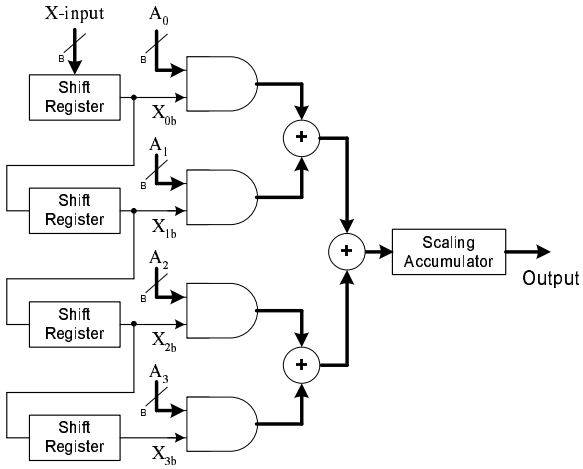


Figure 5.20: FIR filter with Bit-serial multiplier structure where a parallel input to the FIR is converted into a serial stream of bits. In this structure, only one scaling accumulator is used after adding partial products from all the taps.

To maintain full precision in this case, the scaling accumulator is now required to hold the number of bits in the input plus the number of bits in the coefficients plus the number of bits added due to word growth through the adder stages (1 bit per stage).

If the coefficients for the filter are constant, then the output of the summer tree depends solely on the single-bit inputs to each tap. With this being the case, the storage registers for the coefficients, the AND gates, and the summer tree can all be replaced by a single look-up-table addressed by the single-bit shift register outputs as shown in Figure 5.21.

With four taps as shown in Figure 5.21, a LUT with 16 entries is required. Each 4-bit address into the LUT can be thought of as being a sum of coefficients: if a particular address bit is high, then that address' sum should include the corresponding coefficient. To keep the output of the LUT at full precision, the LUT should be two bits larger than the size of the coefficients to accommodate for word growth through the additions.

Implementation considerations

The 16x1 RAM units within the Xilinx CLBs are ideal candidates for this sort of DA scheme. One bit of a single 4-input LUT can fit into one of these units with no unused logic [Moeller and Martinez, 1999]. For FIR filters larger than 4-taps, the filter can be broken into four tap groups, each constructed as shown in Figure 5.21. For example, a 8-tap FIR is shown in figure 5.22. To eliminate overflow, each adder stage must grow by one bit, and the scaling accumulator must also grow accordingly in size (the scaling accumulator could drop the lower bits in its accumulation if less precision is required).

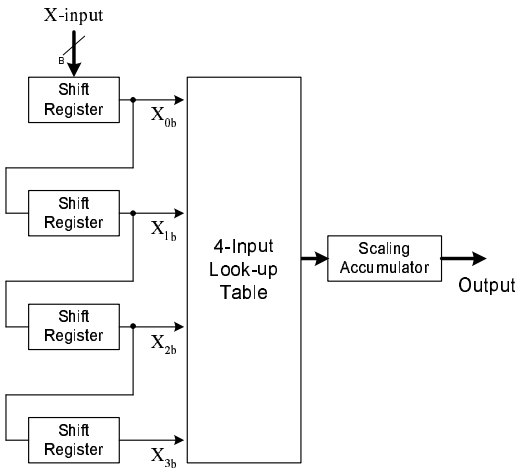


Figure 5.21: Look-up Table based Serial distributed FIR structure.

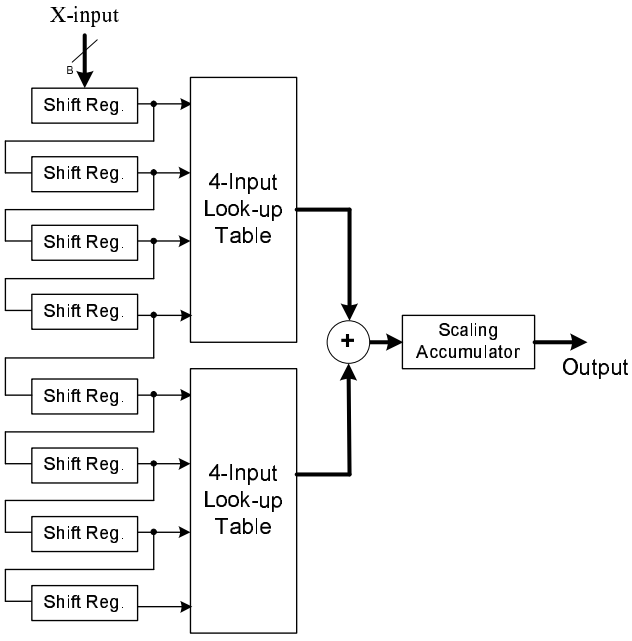


Figure 5.22: 8-Tap Serial Distributed FIR filter structure. Two 4-Tap Serial Distributed FIR filters are used to have a 8-Tap filter.

Although larger LUTs could be used with less adders, LUTs larger than four inputs do not save space. For example, a five-input LUT would require 32- entries and take up two 16x1 RAM units (an entire slice). However, if these two 16x1 RAM units were used separately, they could each be addressed by four taps, allowing an entire slice to handle eight taps. The extra adder needed to sum the two four-input LUTs together would not significantly increase the area enough to justify a five-input LUT.

Parallel Distributed Arithmetic

A benefit of distributed arithmetic is that it easily allows a trade-off to be made between the filter's area and performance. By doubling the filter's area, the filter's throughput or sample rate can be doubled without changing the clock rate that the individual filter components operate at. In the serial distributed arithmetic (SDA) designs discussed before, a clock rate B times the sample rate is required, as one clock cycle is needed to look up a partial product for each bit of x. However, by taking advantage of a feature inherent in the DA equation, Equation 5.6, fewer clock cycles can be required per input sample. Presently, one term in the equation has been computed per clock cycle. However, any number of terms can be computed per clock cycle (referred to as parallel distributed arithmetic, or PDA). For example, if two terms are computed per clock cycle, then B/2 clock cycles are required to compute an output.

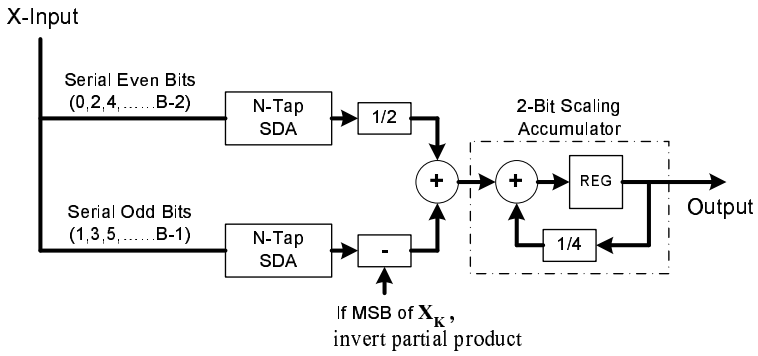


Figure 5.23: 2-Bit Parallel Distributed Arithmetic FIR. By computing 2 terms per clock cycle, then B/2 clock cycles are required to compute an output.

To compute two terms per clock cycle, two identical SDA FIR filters as described above must be constructed. Each filter will compute one term in Equation 5.6 so that two terms are computed per clock cycle. One filter will compute outputs for even input sample bits, and the other filter will compute outputs for odd input sample bits. For example, on the first clock cycle, the first filter will compute the output term associated with X_{k0} while the other filter computes the output term associated with X_{k1}. These outputs are then added together, being the first filter's output (the bit 0 term) scaled by 1/2, and then sent to the scaling accumulator. On each clock cycle, the scaling accumulator scales its registered accumulation by 1/4 to accommodate for the fact that it is handling two partial products per clock cycle instead of one. The 2-bit PDA approach requires twice as much area as the serial approach, but has twice the performance, and is illustrated in Figure 5.23.

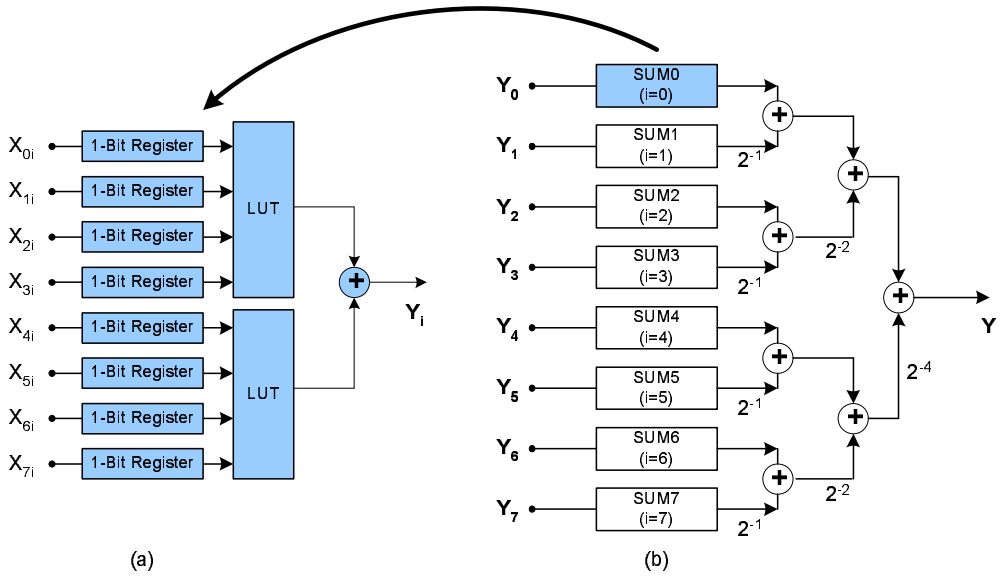


Figure 5.24: (a). A single-bit PDA and (b). A 8-bit fully PDA FIR filter.

For the fully parallel 8-bit PDA FIR filter implementation, the 8-bit input sample is partitioned into eight 1-bit sub-samples so as to achieve maximum speed. Figure 5.24 [Al-Haj, 2004] shows the ultimate fully parallel PDA FIR filter, where all 8 input bits are computed in parallel and then summed by a binary-tree like adder network. The lower input to each adder is scaled down by a factor of 2. No scaling accumulator is needed in this case, since the output from the adder tree is the entire sum of products.

5.5.4 Fast FIR Algorithm

The class of fast FIR algorithms (FFA) attempt to increase the parallelism of the FIR structure without a linear increase in area [Parker and Parhi, 1997] [Jin-Cyun Chung and Wang., 1998]. Traditionally, to double the throughput of a FIR filter without increasing the clock rate of the filter itself, the filter area would have to be doubled.

Doubling the throughput of a FIR filter without changing its internal clock rate means that two outputs are to be calculated each clock cycle. These two outputs will be referred to as $y[2j]$ and $y[2j+1]$. Producing two outputs per clock cycle would require two inputs per clock cycle as well, $x[2j]$ and $x[2j+1]$. This leads to the following set of equations:

$$\begin{aligned} x_0[j] &= x[2j] \\ x_1[j] &= x[2j+1] \\ y_0[j] &= y[2j] \\ y_1[j] &= y[2j+1] \end{aligned}$$

where x_0 and y_0 represent the even inputs and outputs, and x_1 and y_1 represent the odd inputs and outputs.

Two polyphase decompositions of the filter will be required, one containing the even samples of the original filter, the other the odd:

$$\begin{aligned} h_0[k] &= h[2k] \\ h_1[k] &= h[2k+1] \end{aligned}$$

where $h[n]$ is the original filter, and N is the length of the original filter. The above equations give the following z-transforms:

$$\begin{aligned} X &= X_0 + X_1 z^{-1} \\ H &= H_0 + H_1 z^{-1} \\ Y &= Y_0 + Y_1 z^{-1} \end{aligned}$$

which leads to the following two-parallel polyphase representation of the FIR filter:

$$\begin{aligned} Y &= X.H \\ &= (X_0 + X_1 z^{-1})(H_0 + H_1 z^{-1}) \\ &= X_0 H_0 + (X_0 H_1 + X_1 H_0) z^{-1} + X_1 H_1 z^{-2} \\ Y_0 &= X_0 H_0 + X_1 H_1 z^{-2} \\ Y_1 &= X_0 H_1 + X_1 H_0 \end{aligned} \tag{5.10}$$

Equation 5.10 indicates that to double the throughput of the overall FIR filter two of each of the length N/2 polyphase filters would be required as shown in Figure 5.25, resulting in an overall filter with twice as many taps as the original filter (four N/2 length filters).

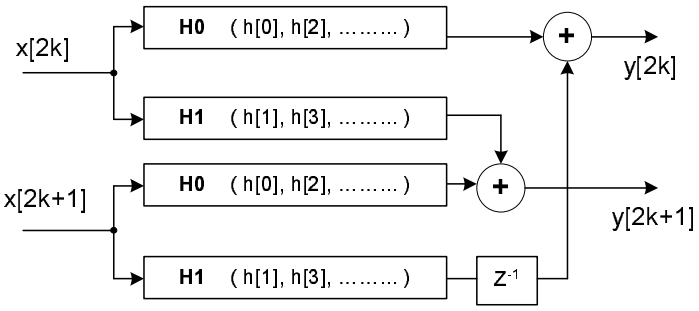


Figure 5.25: Traditional Two-parallel FIR Filter Implementation. To double the throughput of the overall FIR filter, two of each of the length N/2 polyphase filters are required.

Two input samples are collected at a time and passed into the filter structure as illustrated in Figure 5.25, which produces two output samples. Each filter block shown in the Figure 5.25 is running as fast as the original filter, however, the throughput has been doubled. The FFA approach takes advantage of a rewriting of the polyphase equations derived from Equation 5.10:

$$\begin{aligned} Y &= X_0H_0 + (X_0H_1 + X_1H_0)z^{-1} + X_1H_1z^{-2} \\ &= X_0H_0 + [(X_0 + X_1)(H_0 + H_1) - X_0H_0 - X_1H_1]z^{-1} + X_1H_1z^{-2} \end{aligned} \quad (5.11)$$

which implies that

$$\begin{aligned} Y_0 &= X_0H_0 + X_1H_1z^{-2} \\ Y_1 &= (X_0 + X_1)(H_0 + H_1) - X_0H_0 - X_1H_1 \end{aligned}$$

The structure that implements Equation 5.12 is shown in Figure 5.26 for the same overall filter inputs and outputs. This filter only requires 1.5 times as many taps as the original, non-parallel, filter, although the coefficients for the middle FIR element in this case must be pre-computed before being loaded into the FIR element. This is not an issue for most applications, as such a computation can be performed external to the filter.

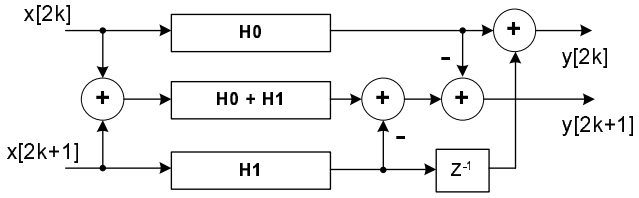


Figure 5.26: Two-Parallel FFA Implementation

5.5.5 Frequency Domain Filtering

Instead of using convolution to calculate the output response for a FIR filter, the filtering can be performed in the frequency domain. Convolution in the time domain is simply a multiplication operation in the frequency domain, so such an operation requires a transformation from the time domain to the frequency domain by a fast Fourier transform (FFT), a point multiplication of the input signal's spectrum by the filter's spectrum, and a transformation back to the time domain by an inverse fast Fourier transform (IFFT). The benefit of this technique is that it requires much less computational hardware than any of the approaches discussed so far using convolution. A FFT's computational requirements scales on the order of log_2N versus N for convolution approaches.

The FFT is derived from the discrete Fourier transform (DFT), which is used to transform discrete time waveforms into discrete frequency spectrums [Alan V. Oppenheim, 1999] [Groginsky and Works.,]. The DFT is defined by

$$x[k] = \sum_{n=0}^{N-1} x[n]W_N^{kn} \quad (5.12)$$

where

$$W_N^{kn} = e^{-j(2\pi/N)kn} \quad (5.13)$$

$x[n]$ is a complex data sample at time n, $x[k]$ is a complex frequency sample at frequency k/N, and N is the number of frequency samples to calculate. W_N^k is sometimes referred to as a "twiddle factor". The DFT requires on the order of N^2 computational requirements, so a more efficient method of computing the DFT is required. If N is an integer power of r, i.e. N=r^v, then an especially easy representation of the DFT appears, the radix-r FFT [Groginsky and Works.,].

For r=2, the algorithm is especially simple. At each stage, the algorithm passes through the entire array of N complex numbers, two at a time, generating a new array of N numbers. The basic numerical computation operates on a pair of numbers at a time, and is referred to as a "butterfly". The decimation in frequency FFT structure is shown in Figure 5.27 for and a butterfly is shown in Figure 5.28. The twiddle factors for butterfly are also shown. A radix-4 FFT also exists, where four outputs are computed per butterfly for four inputs [Alan V. Oppenheim, 1999] [Groginsky and Works.,].

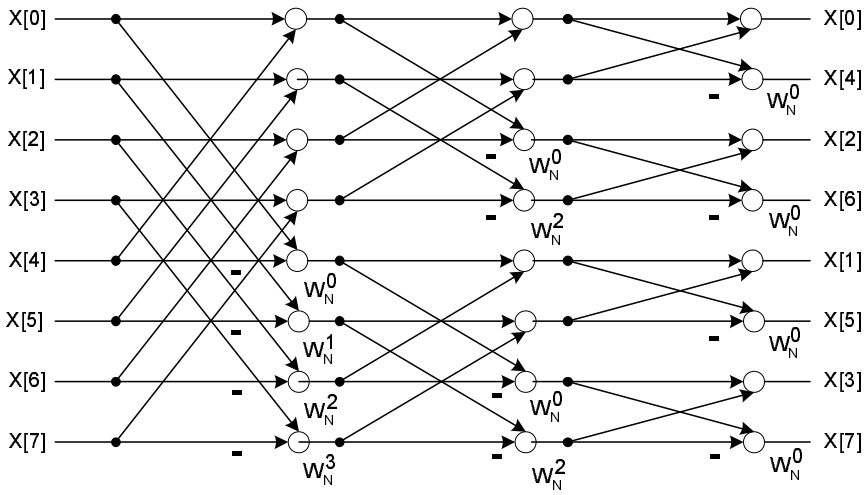


Figure 5.27: Eight-point Decimation-In-Frequency FFT

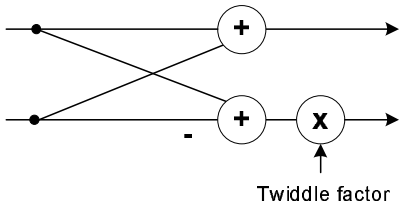


Figure 5.28: Decimation-In-Frequency Butterfly

Pipelined FFT

A nice feature of the FFT is that it can be easily pipelined by stage, as each stage needs only data from the proceeding stage. Each vertical grouping of butterflies in Figure 5.27 is referred to as a stage. Only one butterfly needs to be calculated in each stage at a time, although (to maximize the

sample rate), each stage must have a butterfly calculated each clock cycle. Therefore, the FFT can be built in a pipelined fashion, with each stage handled by a single pipelined butterfly.

Each pipelined butterfly needs shift register storage words long (where m is the stage number, with 0 being the right-most stage) to align its inputs and outputs correctly. For example, the first stage has butterflies that process inputs four samples apart in time. A shift register four words long is required to store the first four inputs, then output those four inputs to the top of that stage's butterfly as the next four inputs arrive at the bottom of the butterfly to compute the correct butterfly outputs. The top output of the butterfly are sent to the next stage while the bottom outputs are put into the shift register, which are then shifted out after the four butterflies have been computed. An example of a pipeline module is shown in Figure 5.29 [Groginsky and Works.,], and an eight-point pipelined FFT architecture is shown in Figure 5.30 [Groginsky and Works.,].

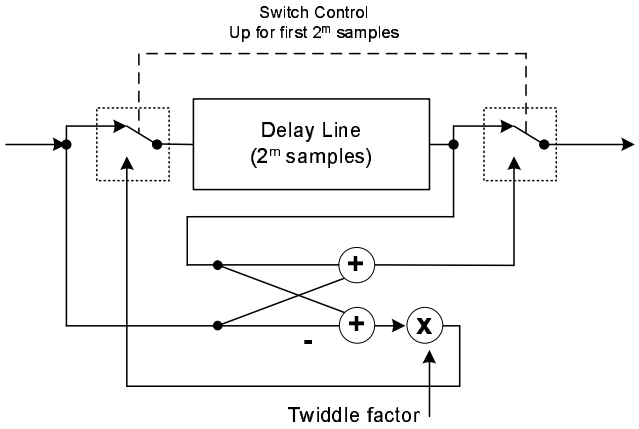


Figure 5.29: Pipelined FFT Butterfly Module

The twiddle factors can be arranged so that they may be sent to all of the modules from a common memory if they are retrieved at the correct time, reducing memory requirements [Groginsky and Works.,].

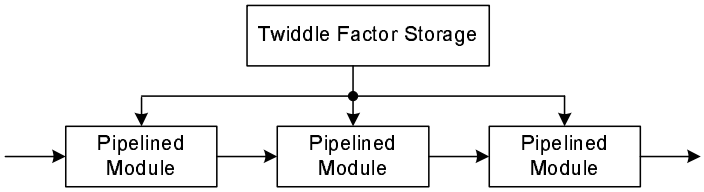


Figure 5.30: 8-Point Pipelined FFT Architecture Block Diagram

FFT Covolution

Performing convolution with a FFT (i.e. transforming to the frequency domain, multiplying, and transforming back to the time domain) requires a FFT at least twice as large as the length of the filter to avoid time-aliasing in computing the DFT of the filter coefficients [Alan V. Oppenheim, 1999].

If a filter is N taps long, the FFT of the filter will have to be 2N points (with zero padding used to extend the N taps to 2N inputs for a 2N-point FFT). The convolution is performed by retrieving a block of N inputs, performing a 2N-point FFT (with zero-padding filling out the inputs) on them, multiplying them by the 2N filter frequency components previously transformed to the frequency domain by FFT, and performing an IFFT on the multiplication outputs. However, since only N inputs were taken and N outputs should be produced from the filter for a given block, and the 2N-point IFFT produces 2N outputs, only the last N IFFT outputs should be used as filter outputs, as the first N IFFT outputs do not represent correct values of the convolution of the filter and the block of inputs [Alan V. Oppenheim, 1999]. This process is termed overlap-save.

A fixed-point FFT requires a considerable amount of rounding, as each stage has a multiplier that increases the stage's bit-length drastically. Rounding is required to reduce the stage's output to a manageable size. The noise analysis for a fixed-point FFT is complex, and is described in detail in [Alan V. Oppenheim, 1999] and [Liu, 1975]. The important result is that each stage's butterfly's outputs require a scaling factor of 1/2 to keep their adders from overflowing, which also reduces the final total amount of noise at the output.

5.5.6 Multiplier-less FIR filter

There are many structures for implementing FIR filters in the literature. Figure 5.31 shows the direct form implementation of the FIR filter. The filter coefficients and the registers (denoted by R) form the tapped-delay line of the FIR filter. For low data throughput rate, the coefficient multiplications can be implemented using the multipliers in a digital signal processor (DSP).

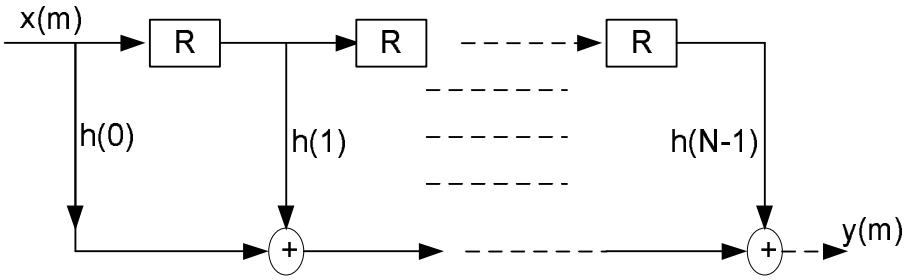


Figure 5.31: Direct Form implmentation of general FIR filter

For high data throughput, usually in traditional VLSI design, hardware multipliers are used which are expensive in terms of hardware resources and power consumption, So in order to solve this problem Sum Of Power Of Two (SOPOT) coeffient representation is used. Where SOPOT can be implemented by shift and add operations. The hardware complexities of these multiplier-less FIR filter is thus very low. Another efficient method for reducing hardware complexity is to employ the hardware multiplier block technique explained in [A.G. Dempster, 1995].

The Z-transform of a general FIR filter with $h(n)$ being the impulse response and its representation in SOPOT coefficients are given by:

$$H(z) = \sum_{n=0}^{N-1} h(n) Z^{-n} \tag{5.14}$$

$$h(n) = \sum_{k=0}^{L-1} a_{k,n} 2^{b_{k,n}} \tag{5.15}$$

where $a_{k,n} \in -1, 0, 1$
and $b_{k,n} \in -l_{b,n}, \ldots, -1, 0, 1, \ldots, \mu_{b,n}$

$l_{b,n}$ and $\mu_{b,n}$ determine the wordlength dynamic range of each filter coefficient. The larger the numbers $l_{b,n}$,$\mu_{b,n}$ and L, closer the SOPOT approximation will be to the original real numbers. In order to approximate the filter coefficeints in power of two terms random search algorithm, trellis search algorithm etc are used. In the random search algorithm, the real-valued coefficients using the least squares approach are obtained as explained in simulation. Let b be the vector containing the real-valued coefficients, then the algorithm repetitively calculates a candidate SOPOT vector b_c given by,

$$b_c = \lfloor b + \lambda b_p \rfloor_{SOPOT} \tag{5.16}$$

λ : is a user defined controlling parameter,
b_p : random vector between ± 1.
$\lfloor \quad \rfloor_{SOPOT}$: is a rounding operator.

Higher the searching time, higher the chance of finding the optimal solution. Following are the steps used in random search algorithm for finding the approximation of filter coefficients in SOPOT form.

- select the real valued coefficients
- represent them in SOPOT expression as explained in Equation 5.15
- $min(T_{SOPOT})$ subject to $\begin{cases} \delta_p < \delta_{p-max} \\ \delta_s < \delta_{s-max} \end{cases}$
  $min(T_{SOPOT})$: total minimum terms of SOPOT.

Programmable SOPOT Unit

The basic building block of programmable SOPOT unit is shown in Figure 5.32. It implements a filter coefficients $h(n)$ with two SOPOT terms.

$$h(n) = a_{0,n}.2^{b_{0,n}} + a_{1,n}.2^{b_{1,n}} \tag{5.17}$$

Let $x(m)$ be the input signal to the FIR filter. The input to this programmable SOPOT unit, which is the delayed input signal $x(m-n)$ from the previous registers, is shifted by $b_{0,n}$ and $b_{1,n}$ positions (shift1 and shift2) using the two programmable shifters. The shifted signals are then

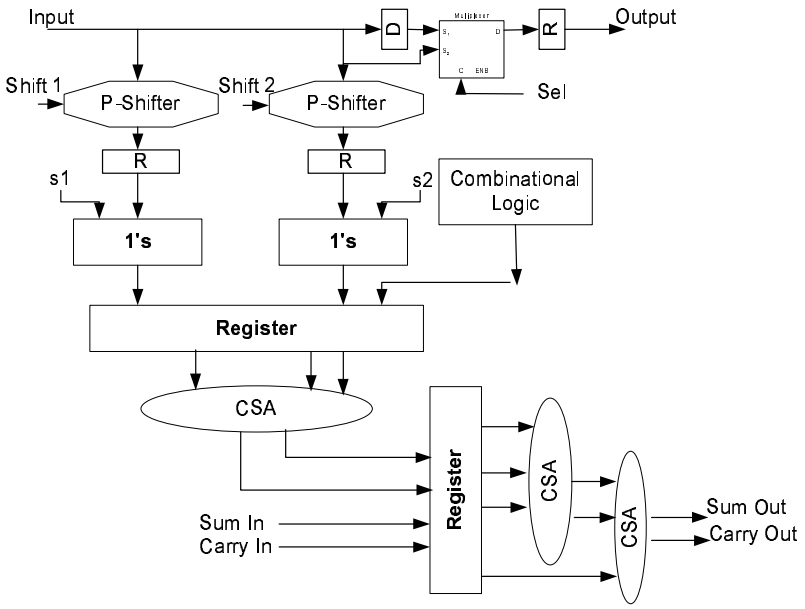


Figure 5.32: The internal structure of the programmable SOPOT unit. The input is first shifted right by the amount specified by the SOPOT term, The shifted signals are then multiplied by the binary numbers and by passing them through the $1's$ complement circuits (controlled by the signals $s1$ and $s2$) before inputting to the carry-save adder(CSA). In order to speed up the data throughput rate and reduce the hardware resources, the carry save adders are employed throughout the multiplier-less FIR filter to carry out the addition of the intermediate signals after each multiplication with the filter coefficients in direct form structure. At the final stage of the direct form FIR filter, the intermediate signals will he accumulated and fed to a summing adder to produce the filter output

multiplied by the binary numbers and by passing them through the $1's$ complement circuits (controlled by the signals $s1$ and $s2$) before inputting to the carry-save adder (CSA). In order to speed up the data throughput rate and reduce the hardware resources, the carry save adders are employed throughout the multiplier-less FIR filter to carry out the addition of the intermediate signals after each multiplication with the filter coefficients in direct form structure. At the final stage of the direct form FIR filter, the intermediate signals will he accumulated and fed to a summing adder to produce the filter output. The registers denoted by R are inserted to fully pipeline the entire operation and the latency of this programmable SOPOT unit is three clock cycles.

The Multiplier-Less FIR Filter Architecture

The Figure 5.33 shows an example structure of the multiplier-less FIR filter using programmable SOPOT coefficients. It consists of sixteen programmable SOPOT units (P-Units), two (16-to-1) multiplexers, N full-adder and some appropriate registers. The programmable SOPOT units as well as the entire multiplier-less FIR filter structure are pipelined. As mentioned earlier, the programmable SOPOT units implement the SOPOT filter coefficients and the delay line of the direct form FIR filter. Each programmable SOPOT unit can realize up to two SOPOT terms. In other words, each unit can implement one filter coefficient if it consists of two or less SOPOT terms, or part of the filter coefficient if it needs more than two SOPOT terms. It can be seen that the outputs of the programmable SOPOT units are serially connected together so that a multiplier-less FIR filter with a certain maximum number of total SOPOT terms can be implemented. The output of the each P-Unit can be carried through tri-state bus, so that a global bus can be used

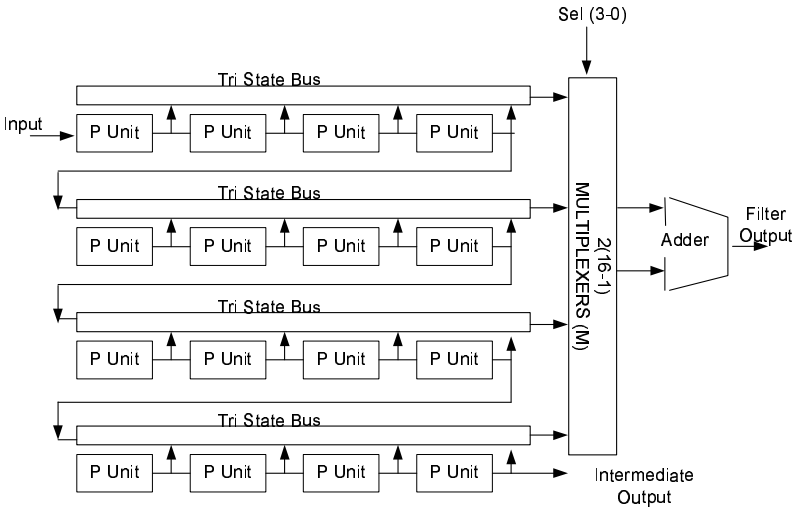


Figure 5.33: The multiplier-less FIR architecture which consists of sixteen programmable SOPOT units (P-Units), two $(16 - to - 1)$ multiplexers, N full-adder and some appropriate registers. The outputs of the programmable SOPOT units are serially connected together so that a multiplier-less FIR filter with a certain maximum number of total SOPOT terms can be implemented.

to reduce the routing area. (16-to-1) multiplexers are used for each sum and carry, which can be configured by using *Sel(3:0)* as control signal.

Complexity Analysis of Multiplier-less FIR filter

The P-unit consists primarily of shifters and adders, so the complexity of each P-unit depends on these processing elements. The Programmable shifter is shown in Figure 5.34. It is used to shift the input signal from (0-31) bits to the right. It consists of 5, (2-to-1) mux arranged in 5 different stages and some registers for pipelining purpose. The position of the shift is first decomposed into weighted binary representation. Since the amount of shift is either 0 or 2^n, they can be implemented by (2-to-1) multiplexer with appropriate hardwiring of its input. The carry save adder is an effective replacement of the normal ripple-carry adders, as it just saves the carry and finally the Pipelined ripple-carry adder is used to calculate the accumulated and intermediate outputs of the last P-unit as shown in Figure 5.33. These adders needs $(N + 1)$ clock cycles, whereas the carry save adder can add N bits in 1 cycle. There are 2, (16-to-1) multiplexers require to select the sum and carry of each P-unit. The internal structure of $(N - 1)$ MUX needs to have $log_2(N)$ stages, so (16-to-1) Mux needs to have 4 stages of (2-to-1) multiplexers.

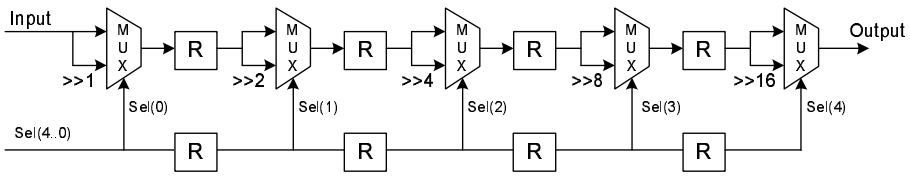


Figure 5.34: The programmable shifter, it consists of $log_2(N)$ stages with either 0 or 2^n shifts and it can be implemented with $(2 - to - 1)$ multiplexer by appropriate hardwiring of its input.

Each P-unit is representing the single filter coefficent with two SOPOT terms. The Table 5.6 is showing the complexity analysis in terms of hardware resourses, and the computational clock cycles for each operation required for filtering, The multiplier-less filtering is compared with the normal FIR filtering.

Hardware Resourses	**Normal FIR Filter**	**Multiplier-less FIR Filter**
$Multiplexers$	0	$2(N-1)+2N(log_2(N)+N$
$Multipliers$	$N*B$	0
$Adders$	$(N-1)*B$	$3N+B$
Total Cycles	2NB -B	4N-1+2Nlog_2(N)+B

Table 5.6: The complexity of multiplier less FIR filter is mainly dominated by multiplexers and adders, whereas the normal FIR filter require adders, multipliers and shifters. N is a filter length and B corresponds to Word-length of the input

The 18bit multiplier-less FIR filter using programmable SOPOT coefficients requires 6769 logic cells and its maximum clock frequency is 54.94MHz, In the contrast, a single 18bit multiplier in the Altera FLEX 10K FPGA requires 973 logic cells and its maximum clock frequency is 12.67MHz. The number of logic cells in general FIR structure with 8 filter taps require $8*973 = 7784$ logic cells just for the multiplications which is much more than even complete FIR implementation [K.S Yeung, 2002]. Based on these results it is clear that multiplier-less FIR filtering using Programmable units is much faster with fewer resources in comparision to normal FIR filter.

5.5.7 Results

Filter Coeff	**SOPOT Coeff**	**Filter Coeff**	**SOPOT Coeff**
$h(0)=h(49)$	$2^{-9}-2^{-10}$	$h(12)=h(37)$	$2^{-6}-2^{-9}$
$h(1)=h(48)$	$2^{-9}-2^{-10}$	$h(13)=h(36)$	$2^{-6}-2^{-8}$
$h(2)=h(47)$	$2^{-9}-2^{-10}$	$h(14)=h(35)$	$2^{-8}-2^{-10}$
$h(3)=h(46)$	$2^{-9}-2^{-10}$	$h(15)=h(34)$	$-2^{-7}+2^{-9}$
$h(4)=h(45)$	$2^{-9}-2^{-10}$	$h(16)=h(33)$	$-2^{-5}+2^{-7}$
$h(5)=h(44)$	$-2^{-8}+2^{-9}$	$h(17)=h(32)$	$-2^{-5}+2^{-8}$
$h(6)=h(43)$	$-2^{-7}+2^{-8}$	$h(18)=h(31)$	$-2^{-5}+2^{-8}$
$h(7)=h(42)$	$-2^{-8}+2^{-10}$	$h(19)=h(30)$	$-2^{-6}+2^{-8}$
$h(8)=h(41)$	$-2^{-8}+2^{-10}$	$h(20)=h(29)$	$2^{-5}-2^{-7}$
$h(9)=h(40)$	$-2^{-9}+2^{-10}$	$h(21)=h(28)$	$2^{-4}-2^{-7}$
$h(10)=h(39)$	$2^{-8}-2^{-10}$	$h(22)=h(27)$	$2^{-3}-2^{-10}$
$h(11)=h(38)$	$2^{-6}-2^{-8}$	$h(23)=h(26)$	$2^{-3}-2^{-5}$
$h(24)=h(25)$	$2^{-3}-2^{-4}$		

Table 5.7: For the WLAN, there are 50 filter coefficents, which are decomposed in the polyphase structure with 10 sub-filter each having 5 coefficients. Due to positive symmetry of the filter, the filter coefficients $h(0) = h(49)$, $h(1) = h(48)$ etc. The sopot approximation of the original coefficients are represented. Only two sopot terms are required for each coefficients.

Table 5.7 shows the approximation of the original filter coefficients of the WLAN, for the UMTS similar approximations can be achieved. Only two SOPOT terms are required for repre-

senting each filter coefficient. Maximum power of 2 is 10 which means that we only require 10 right shifts in order to achieve the FIR filtering, whereas the normal filtering requires shifts equal to the word-length. The Figure 5.35 shows the impulse reponse of the approximated coefficients in order to represent it in power of two. The approximated impulse response is very similar to the original impulse response which is shown in the simulation chapter (WLAN) and therefore the polyphase channelizer output is also acceptable with required accuracy as shown in Figure 5.36.

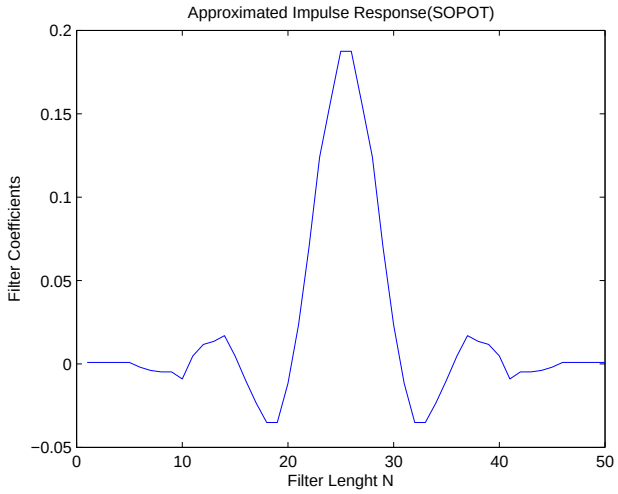


Figure 5.35: The approximated impulse response of the polyphase filter of WLAN. The approximation in SOPOT form is shown in 5.7.

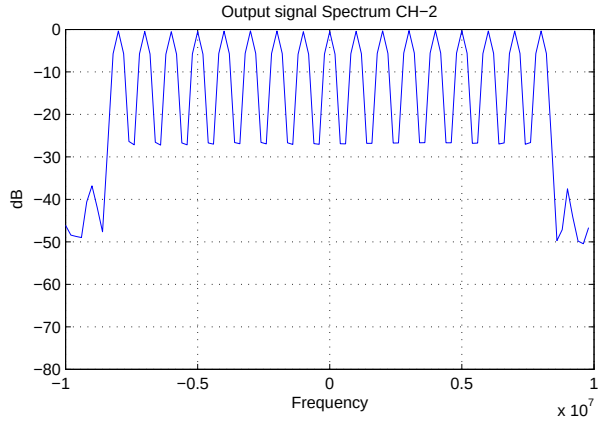


Figure 5.36: The output of the polyphase channelizer for the second channel of the WLAN. The results shows that the approximations are acceptable

There are 25 P-Units require for implementing the WLAN polyphase filter, which can be obtained by cascading the two structures shown in Figure 5.33 as each structure implments 16 coefficients.

The multiplier-less FIR filtering technique described in this section could possible be optimized in the VLSI design. Most of the operations can be hardwired and highly optimized based on the given cost function (Area, Time, Power, etc.)

5.6 Cost Function for the Implementation

When making a decision, leading to an outcome, there are some parameters involved in the design. The suitable architecture can be analysed and decided based on the parameters, such as algorithm, constraint and platform in the following manner:

$$Architecture(i) = f\{Algorithm(i), Constraints(i), Platform(i)\}$$

It means that the intrinsic property of the algorithm, the constraints and the final target platform can have a strong influence on final architecture decision. This means that the final architecture should inherit the intrinsic properties of the algorithm. The parameters for choosing the final architecture can be described by the cost function based on the common design metrics in the following way:

$$C = f\{TE, A, N, TD\}$$

- Execution time (TE): The execution time is the time needed to execute the algorithm, which corresponds to the performance in the design metrics. Mainly the data length determine the execution time, i.e., longer data length results in longer execution time, as longer arithmetic operations are to be performed.
- Area (A): Area is defined as the amount of hardware used in the system. It relates to the physical size of the product and has the influence on the power consumption and the financial cost. Applied technology and the data length determines the area.
- Numerical properties (N): Rounding noise is produced when data are rounded to finite data length. Applied data length determines the amount of the rounding noise. i.e., longer data length causes less noise. Rounding noise can cause degradation in algorithm performance in comparison. Algorithms sensitive to rounding noise may become unstable in the worst case.
- Development time (TD): Development time is defined as the time needed to design and implement the algorithm on to the simulation or hardware platform.

The purpose of cost function is to optimize the required variable/variables with respect to some constraints. In this project, the focus is on the area optimization of the algorithm with respect to the time constraints (105MHz for UMTS and 120MHz for WALN). The area is bounded by the area parameter of Xilinx Virtex-IV XC4VSX35 chip.

5.7 Design Space Exploration

A design can be implemented in a number of ways on a number of architectures. These number of solutions form a huge solution space. Design space exploration provides area-time trade off curves of the implementation of a design. This is due to inherent parallelism that can be deployed

for the design. Figure 5.37 shows the area-time curve of a design, giving a number of solutions in a design space.

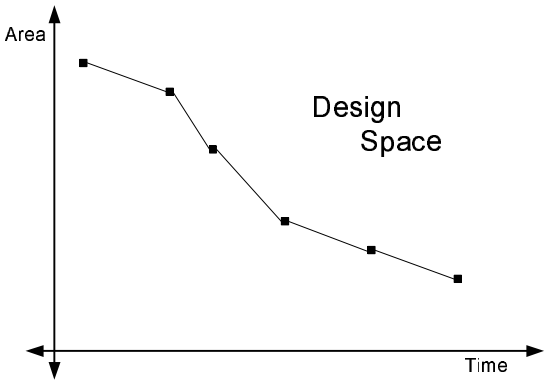


Figure 5.37: Design Space Exploration: Its shows that there are numerous way of implementing a design, but it will be trade-off between Area and the Time reqiured to execute the process in the design. Black spots shows some of the solutions

CHAPTER 6

ALGORITHM-TO-ARCHITECTURE MAPPING

In this chapter, the Algorithm-to-Architecture mapping is presented. A part of Polyphase channelizer, which is a sub-filter (same as FIR filter) is selected for architecture mapping. In this regard, the structures based on the different methods described in previous chapter, are mapped to the platform (Xilinx Virtex-IV FPGA). Different resources of the FPGA are explored. The analysis is based on the approximation of the hardware resouces required by each of the filter structure. Refering back to A^3-Model, we are now in the Architecture domain, performing the algorithm to architecture mapping, as shown in the Figure 6.1.

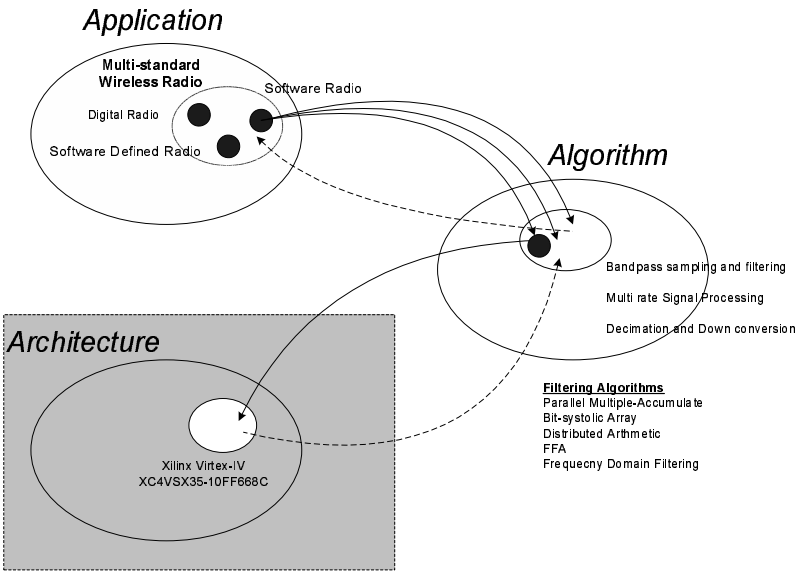


Figure 6.1: A^3-Model: Emphasising the Architecture domain, where the mapping from the Algorithm to Architecture is performed.

In the polyphase channelizers, UMTS polyphase filter bank has 21 sub-filters each of length 12 taps, whereas WLAN polyphase filter bank has 5 sub-filters each of length 10 taps. So we have to design filters of length 12 and 10. The basic design parameters are:

- Input Data-width: 16 Bits
- Filter's Coefficient Data-width: 18 Bits
- Input data and Filter Coefficent both are taken as Real data (for simplicity)

A filter length of 16 is selected to make the flow simpler. The design is based on following requirements:

- Clock frequency for WLAN sub-filters is 120MHz and for UMTS sub-filters is 105MHz
- In the serial implementation of the Polyphase filter bank (as discussed in the previous chapter), it is required to have a swapable coefficient memory bank, so that one subfilter's coefficients are used in multiply-accumulate process while the next coefficients are being loaded into the other memory bank for the next process.

6.1 Parallel Multipliers and Accumulators

In the custom VLSI chip, each multiplier was re-used eight times per input sample. This meant that each tap would compute eight products and accumulate them together every input sample for eight separate coefficients. In this manner, 2 MACs were all that were needed to compute a 16-tap filter. However, each MAC needed to operate at a clock rate eight times that of the input data rate, so for a 105MHz input sample rate for UMTS, a 840 MHz clock was required, and for a 120MHz input sample rate for WLAN, a 960MHz clock was required. These clock specifications are quite high for Vixtex-FPGA (Maximum clock freq. of 500MHz). The clock speed can be decreased by decreasing the number of taps per MAC to be processed. By processing 4 taps per MAC, the required clock speed becomes 420MHz for UMTS and 480MHz for WLAN, which are within the specifications of Vixtex-FPGA, but it will increase the number of MACs.

The Xilinx Virtex series FPGA has dedicated multiplication resources so that two multiplication bits can fit into a single slice. An **a**-bit by **b**-bit parallel multiplier requires approximately

$$\frac{\mathbf{b}\, log_2\mathbf{b} + (\mathbf{b}-1)\mathbf{a}}{2} \tag{6.1}$$

CLB slices [Xilinx_Multiplier, 2000] [Moeller, 1999]. A 16-bit by 18-bit parallel multiplier would require 163 slices. Therefore, the multipliers alone in a parallel MAC structure of 4 taps per MAC would require 652 slices. The adders, additional registers and control logic required for this design would push this number higher. A 16-Tap parallel MAC filter is shown in Figure 6.2 [Moeller, 1999].

If each MAC was responsible for 4 taps, the coefficient storage for a single bank for a single MAC would require 9 slices since a slice contains two 16x1 RAM blocks, and one RAM block could hold a single bit for all 4 taps (maximum 16 taps). 18-bit coefficients would therefore require

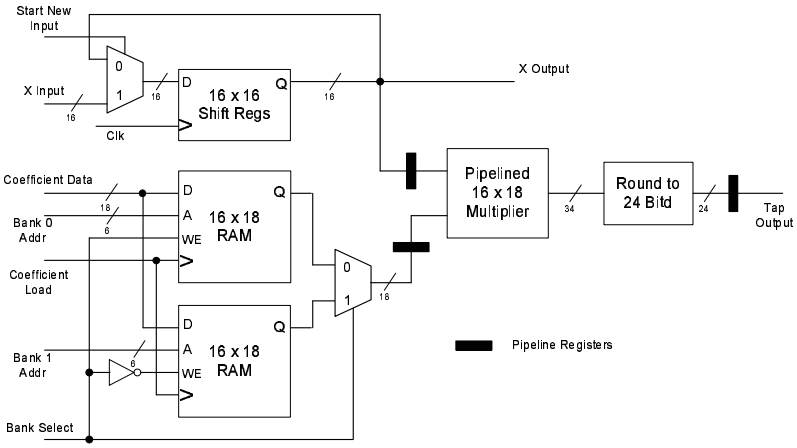


Figure 6.2: 16-Tap parallel MAC filter [Moeller, 1999]

18 RAM blocks which can be contained within 9 slices. Two coefficient banks are required by the design specifications, so 18 slices are needed per MAC for coefficient storage. With 1 MACs, this means that 18 slices will be required for coefficient storage. With 4 MACs, this means that 72 slices will be required for coefficient storage. So a total of 652+72 slices will be required for a 16-tap filter implemented as parallel 4 MAC processing 4 taps each.

The resource ultilization for Parallel MAC structure for having different numbers of MAC units, is described in the Table 6.1.

Resource Ultilization for Parallel MAC structure

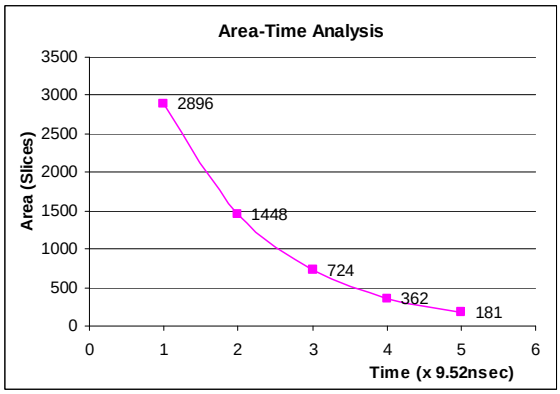
No. of Taps/MAC	No. of Required MACs	Clock Requirement (MHz)	Slices Required (Approx.)		
			MAC	Coeff. Memory	**Total**
16	1	1680	163	18	**181**
8	2	840	326	36	**362**
4	4	420	652	72	**724**
2	8	210	1304	144	**1448**
1	16	105(same as input clock rate)	2608	288	**2896**

Table 6.1: Resource Ultilization for Parallel MAC structures: for 16,8,4,2,and 1 tap per MAC configuration. The input sampling rate is 105MHz. A 16-bit by 18-bit parallel multiplier would require 163 slices.

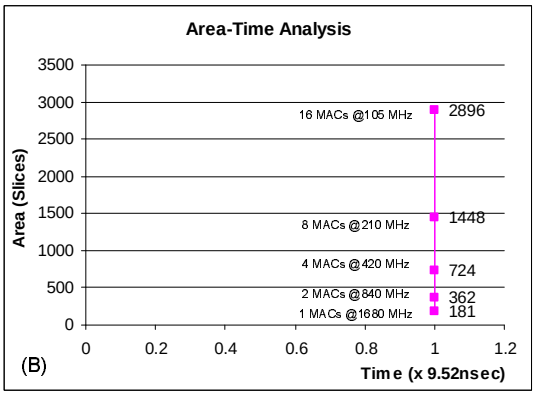
Area-Time Analysis of the parallel MAC structure filter is shown in Figure 6.3. Figure 6.3a shows the scenario where the operating clock for all the cases is same. So each case will give its final output at different time. Whereas Figure 6.3b shows the scenario where the operating clock for all the cases (except one) is boosted up, so that each case will have its final output at the same time, which is desired.

6.2 Bit Systolic Array Architecture

In the Bit systolic Architecture, each cell's output is required to be registered in order to pipeline the array. The three outputs from each main cell each require a register, so the cell requires at



Area-Time Analysis for parallel MAC structure Filter. The operating clock rate is same for all the cases (i.e. 105MHz).



Area-Time Analysis for parallel MAC structure Filter. The operating clock rate is increased for each case (which is number of taps/MAC multipled with 105MHz), so that the filter works at the input clock rate of 105MHz

Figure 6.3: Area-Time Analysis of the parallel MAC structure filter. Figure 6.3a shows the scenario where the operating clock for all the cases is same. So each case will give its final output at different time. Whereas Figure 6.3b shows the scenario where the operating clock for all the cases (except one) is boosted up, so that each case will have its final output at the same time, which is desired.

least one and a half slices. Each main cell also must compute two functions of four inputs (each of which can fit into a Virtex LUT) and needs to store one bit of a coefficient (without dualbanking as required by the design specifications). A CLB LUT may be used to store this bit. Therefore, each main cell requires three registers and three LUTs. Two main cells may be contained within three slices, which would contain six LUTs and six registers, without dual-banking.

According to [Chin-Liang Wang and Chen, 1988], the number of main cells required for the systolic array is $(B + L).N$, where B = the number of bits in the input, N = the number of taps, and $L = log_2 N$. For a 16-tap, 18-bit filter, the number of main cells required for the systolic array is 320. This would require 480 slices to implement. Since this structure is bit-level pipelined, one output is produced every B clock cycles, or every 16 clock cycles for the 16-bit design. Therefore, for a 105 MHz input and output sample rate, the array will have to operate at 1680MHz, which is too high for Vixtex-FPGA (Maximum clock freq. of 500MHz). In addition to the main array, there are other cells required for the design that would also increase the area.

This architecture was used for a full-custom, transistor mask-level design. The reason that this design was so efficient for a full-custom chip versus a FPGA is that it requires a very fine-grained architecture. The basic cell in the bit-level systolic array is only three registers and a few logic gates, which takes up very little area on a custom chip. However, for a coarse-grained FPGA such as the Virtex, the simplicity of a single cell is actually a drawback, area-wise. A better approach for a FPGA design is to use cell sizes that more appropriately map into the FPGA's architecture, such as those used in Distributed Arithmetic.

6.2.1 Distributed Arithmetic

The distributed arithmetic approach can easily be expanded to 16-taps, but two problems remain. First of all, the DA coefficients are constant, but the design requirements demand two swapable,

loadable coefficient banks. Second, a SDA filter requires B clock cycles to process a single input sample. For a 105MHz input, this means a 16-bit input filter must run at 1680MHz, which is too high for virtex chip.

To solve the first problem, two banks of LUTs are used for each four-tap group. One bank is used as the active LUT - this is the LUT that is addressed by the shift-register outputs and one bank is loadable by the user. Therefore, the user can load all of the LUT values into one bank of the FIR filter in the background while the old LUT values stored in the other bank are active. By toggling a bank select line, the two banks are switched so that the previously loadable bank is now active, and the previously active bank is now loadable.

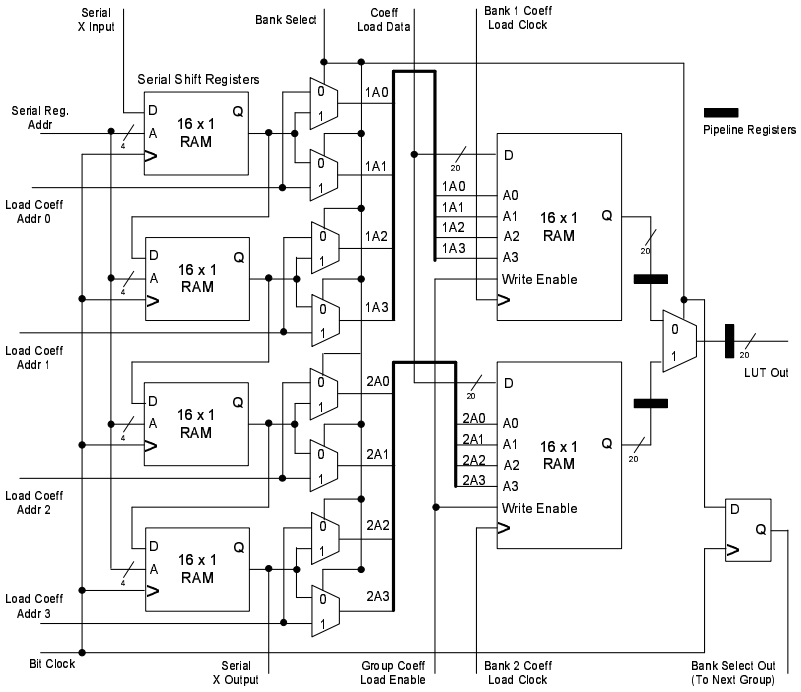


Figure 6.4: Four Tap Group including the shift registers and the double-banking LUTs [Moeller, 1999]

Figure 6.4 [Moeller, 1999] shows the complete block diagram for a single four-tap group including the shift registers and the double-banking LUTs as described above. The four-tap group accepts a serial data input (from the last tap's shift register of the previous group), and produces a serial data output for the next group's first tap's shift register. A bank selection line selects (through multiplexers) which bank of LUTs are active, and which are used for loading new coefficients. Each bank is 20-bits long due to two-bit word growth in computing the LUT contents. The active bank is addressed by the four shift register outputs. The bank's output is the group's output. The loadable bank is addressed by an external set of coefficient address lines that select which of the 16 bank addresses is being written. A group coefficient load enable line selects whether this group is to have its coefficients updated versus another group, and a bank write clock line writes the data into the correct bank (the bank being loaded). A separate clock was used for each bank's

LUT write clock to minimize the amount of logic local to a group.

Pipelining was inserted in the four-tap group so that the combinational delay between pipeline registers has been kept to a minimum to increase performance. In addition, as shown, the bank selection line has been pipelined between four-tap groups. This prevents a single bank selection line from having to drive all the multiplexers in every four-tap group, which would lead to a very high fan-out and a slow signal, decreasing the overall system performance.

One drawback is that changing coefficient banks will take 4 clock cycles during which the new bank selection signal is propagated through its pipelining registers. Any outputs produced during that time will consist of outputs from both coefficient banks, and will be incorrect responses from either bank's filter. This drawback is addressed below with the linear-network summer tree. In addition, coefficients in a given four-tap's stand-by registers cannot be altered after a bank selection switch until the new bank selection signal has propagated to that four-tap, or else the wrong bank would be updated.

In a single four-tap group, each of the 16-bit shift register can be implemented by using a 16x1 RAM. So four 16-bit shift registers will require 2 slices. A filter coefficent bank of 4x20 will require 10 slices. So for two memory banks, 20 slices will be required. 9 multiplexer, 1 latch and 4 registers (pipeline registes) will require 14 slices (one slice for each). So a total of 36 slices are approximated for 4-tap SDA group.

Resource Ultilization for 4-taps SDA structure

	Hardware Resource	No.of Resources	Slices Required (Approx.)
Shift Register	16x1(RAM)	4	2
Coefficient Memory Bank	16x1(RAM)	40	20
Multiplexer	LUT	9	9
Registers	16x1(RAM)/LUT	5	5
Total			36

Table 6.2: Resource Ultilization for 4-taps SDA structure, including Shift-Registers, Coefficient Memory Bank, Multiplexer and Registers.

The resource utilization for 16-Tap filter designed using four 4-tap group Distributed Arthmetic module is given in the Table 6.3. 20-bit Adder, 21-bit Adder and Scaling Accumulator are coded in VHDL (hardware desciptive language) to have their slice count.

6.2.2 Linear Summer Network

The summer tree used to create the full 16-tap partial product requires routing lengths. The large summer tree requires long routing lengths to provide the inputs for the last few adders, and is not geometrically easy to fit into a FPGA without wasting area or introducing even longer wire lengths.

A linear design was created, where each four-tap group has a summer associated with it that adds the previous four-tap group's output to its own output. This sum is pipelined, and send to the next tap. Due to the added stage of pipelining between the summers at each four-tap group, a

Resource Ultilization for 16-taps SDA structure (Summer Tree)

	Hardware Resource	No.of Resources	Slices/Module (Approx.)	Total Slices (Approx.)
4-taps SDA	decribed above	4	36	144
20-bit Adder	LUT	2	10	20
21-bit Adder	LUT	1	11	11
Scaling Accumulator (29-Bit)	Slice	1	15	15
Total				190

Table 6.3: Resource Ultilization for 16-taps SDA structure (Summer Tree), including 4-taps SDAs, adders and scaling accumulator.

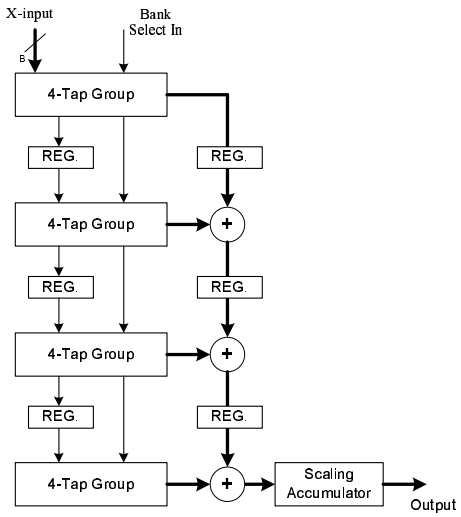


Figure 6.5: Linear Summer Network SDA

stage of pipelining must be inserted between each group's serially cascaded input value. This will keep the outputs and inputs correctly synchronized. The linear summer technique is illustrated in Figure 6.5.

With the linear technique applied, the design is more efficient as each four-tap group has one summer attached to it that needs to communicate with only adjacent groups, so all of the groups may be stacked together. There are no long routing lengths required in this design like there are in the summer tree technique. To minimize wasted area, each summer is only as many bits long as required to protect against overflow. For example, the summer for the second four-tap group need only be 21 bits long because it is adding the 20-bit result of the first group to the 20-bit result of the second group. The third group's summer needs to be 21-bits long, as it is adding three 20-bit results. The number of extra bits per summer can be found by taking the integer portion of $log_2 i$, where i is the four-tap group's number [Moeller, 1999].

The resource utilization for 16-Tap filter designed using four 4-tap group Distributed Arithmetic module having linear summer network is given in the Table 6.4.

Resource Ultilization for 16-taps SDA structure (Linear Summer Network)

	Hardware Resource	No.of Resources	Slices/Module (Approx.)	Total Slices (Approx.)
4-taps SDA	decribed above	4	36	144
20-bit Adder	LUT	2	10	20
21-bit Adder	LUT	1	11	11
Pipeline Registers	Slice FlipFlop	6	1/2	3
Scaling Accumulator (29-Bit)	Slice	1	15	15
Total				193

Table 6.4: Resource Ultilization for 16-taps SDA structure (Linear Summer Network), including 4-taps SDAs, adders and scaling accumulator.

A benefit of the combination of the linear network and pipelining the bank selection line is that, upon the execution of a bank switch (inverting the bank selection signal), the four-tap groups sequentially switch their coefficient banks from stand-by to active each clock cycle. Outputs being formed by the four-tap groups and being passed along through the linear summer network before the banks were switched will continue to have partial products generated using the old coefficients added to them as they move down the summer network's pipelining chain. Since the outputs and the bank selection signal propagate through the network at the same rate, the first output after the bank selection switch will only have partial products generated with the new coefficients added to it. The coefficient banks in a given four-tap group will swap at the same time this output enters the group, resulting in the correct partial product being summed to the output by the group. This means that no incorrect data will be generated during a bank switch.

This technique has two small drawbacks. First of all, it is slightly larger than the summer tree technique. Although the number of adders is the same for both techniques, but the adder tree

requires less area as the adder bit size grows, whereas the linear network requires more. However, as long as the design still fits within a Virtex device, this is acceptable. The second drawback is that the output has a latency of 4 clock cycles due to the pipelining of each four-tap group's output. In most signal processing applications, small latencies such as this are not detrimental.

6.2.3 Achieving Low clock rate performance for 105 MHz Sample-Rate

The serial distributed arithmetic design as described above requires a clock rate 16 times faster than the input data rate. For a 105 MHz data rate, this means the serial filters must run at 1680 MHz.

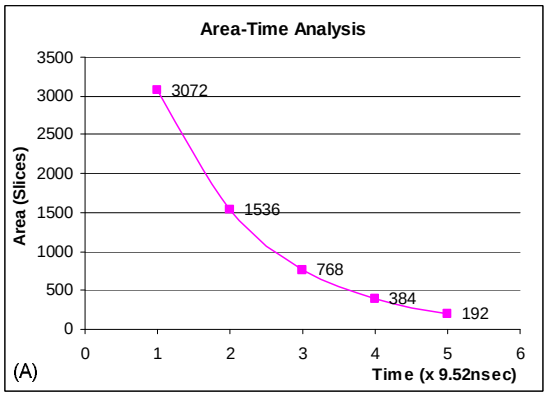
Two solutions exist to solve this problem. The first is to operate the SDA filter at 1680 MHz, using the Virtex DLL to multiply the external 105 MHz clock up to a 1680 MHz internal clock rate. But the DLL can work up to 500 MHz. The second is to use 2-bit PDA, with two 16-tap SDA filters. A clock rate 8 times faster than the sample rate would be required for this design, and the internal filters would have to operate at 840 MHz, yet the design would require twice as much area as a SDA design. Even this solution donot fit for the clock specification of the Virtex Chip. To use 4-bit PDA, with four 16-tap SDA filters, would requires a clock rate 4 times faster than the sample rate. The internal filters would have to operate at 420 MHz, but the design would require four times the area as a SDA design. The trade-off between the these techniques is speed versus area. The 4-bit PDA design requires 768 slices for the 16-tap linear-network filter.

The Area-Time analysis for Distributed Arithmetic technique, having different numbers of bits processed at a time is shown in Figure 6.6. Figure 6.6a shows the scenario where the operating clock for all the cases is same. So each case will give its final output at different time. Whereas Figure 6.3b shows the scenario where the operating clock for all the cases (except one at 105MHz) is boosted up, so that each case will have its final output at the same time, which is desired.

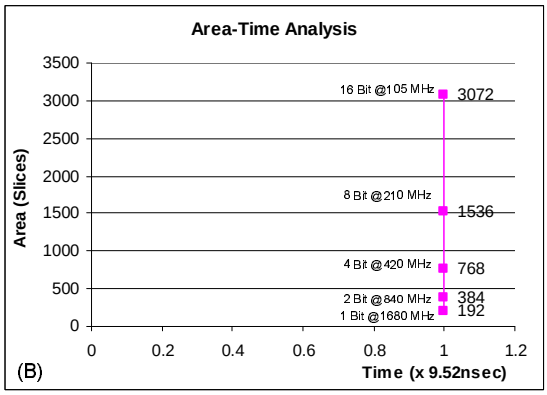
6.3 Fast FIR Algorithm

The parallel MAC approach to the filtering problem can be combined with the FFA algorithm to derive small filtering structures [Moeller, 1999]. In the parallel MAC approach implementation, 4 MACs used to calculate a 16-tap FIR response by having each MAC performing four multiplications per input word, so that each MAC handle four filter taps. This could be decreased to 2 taps per MAC, so that 8 MACs would be needed with an 210 MHz clock rate.

If each MAC is responsible for 2 taps, 8 MACs would be required, but a 210MHz clock rate is required. The new FFA approach would take the polyphase decomposition of the filter. Each polyphase filter (i.e. H_0 or H_1) would be half the size of the original 16-tap filter, or 8-taps. Applying the FFA algorithm would require three 8-tap filters. Each of these smaller filters would now have to run at only half the original filter's sample rate, or 52.5 MHz to maintain an overall sample rate of 105 MHz (the FFA approach allows an overall throughput twice that of the individual filters' sample rates). Each polyphase filter could be implemented with 4 MACs, where each MAC handled 2 taps. The benefit is that the sample rate of each filter is now only 52.5 MHz, meaning that each MAC would only have to run at 105 MHz to compute the results from 2 taps per input.



Area-Time Analysis of DA structure Filter with different numbers of Bits processed at a time. The operating clock rate is same for all the cases (i.e. 105MHz).



Area-Time Analysis of DA structure Filter with different numbers of Bits processed at a time. The operating clock rate is increased for each case (which is (Total No. of Bits / No. of Bits processed at a time) multiplied with 105MHz), so that the filter works at the input data rate of 105MHz

Figure 6.6: The Area-Time analysis for Distributed Arithmetic technique, having different numbers of bits processed at a time. Figure 6.6a shows the scenario where the operating clock for all the cases is same. So each case will give its final output at different time. Whereas Figure 6.3b shows the scenario where the operating clock for all the cases (except one at 105MHz) is boosted up, so that each case will have its final output at the same time, which is desired.

With no change in clock rate (i.e. at 210MHz), the total number of MACs would be decreased to 6 (Polyphase structure) from the original 8.

One major change is that the middle filter would be multiplying 17-bit inputs by 19-bit coefficients because of the one-bit word growth through the addition of before the filter and the one bit word growth in the addition of to compute the filter's coefficients. Therefore, the middle polyphase filter would be slightly larger than the other two filters. The overall FFA filter will also be slightly larger due to the five extra summers and the delay element.

The middle filter requires MACs of size 17x19 bits, whereas the other two filters requires MACs of size 16x18 bits. The area required by multipliers of size 17x19 bits and 16x18 bits is 187 and 173 slices respectively. So 16 multipliers of 16x18 bits (both top and bottom filters) requires 2768 slices and 8 multipliers of 17x19 bits (middle filter) requires 1496 slices, which results in overall slices of **4264**. The five extra adders will push this value little high.

6.4 Frequency Domain Filtering

For the frequency domain filtering implementation, a 32-point FFT and IFFT are required (previous chapter). However, since the input data is real, both the real and imaginary parts of the FFT may be used to hold the real input data [Alan V. Oppenheim, 1999] [Groginsky and Works.,]. Therefore, a 32-point FFT can be calculated with a 16-point FFT structure.

The next step for the frequency domain filtering implementation was to determine the bit-size required for the FFTs in order to meet the custom VLSI chip's output precision. The minimum bit-width to maintain approximately 18 bits of output precision was determined for each variable

separately, and is shown in Table 6.5 [Moeller, 1999].

Frequency Domain Filtering Output Precision

Parameter	Bit-Width	Output Precision
FFT Bit-width	30	17.704
	31	18.802
Filter Spectrum Bit-width	28	17.973
	29	19.589
Point multiplication round output width	34	17.487
	35	18.465
IFFT Bit-width	35	17.536
	36	18.512

Table 6.5: Bit-width for Frequency Domain Filtering parameters to have Output Precision equal to VLSI design (18-bits)

The bold bit-widths in Table 6.5 are used together in the frequency domain filtering technique. The output precision is 17.881 bits, which is determined to be closed enough to the custom VLSI output precision, as adding a bit to any of the parameters above would drastically increase the technique's area.

With the bit-widths above, a preliminary area calculation was made for multipliers and memory. Using the Equation 6.1, the 31-bit x 31-bit multipliers in the FFT would require about 542 slices. The point multiplication would be multiplying the FFT's 31-bit result by a 29-bit filter spectrum value, which would require about 512 slices. The IFFT multipliers would be multiplying 36-bit numbers by 36-bit numbers, and would require about 723 slices. This slice utilization is shown in Table 6.6.

Area Requirements for different Multipliers

Parameter	Multiplier Size	Slices Required (Approx.)
FFT	31x31 Multiplier	542
Point Multiplier	31x29 Multiplier	512
IFFT	36x36 Multiplier	723

Table 6.6: Area Requirements for different Multipliers

Assuming a 16-point FFT was being used as described above, the FFT and IFFT would each have 4 stages, so 8 butterfly pipeline stages would be required. With four multipliers per stage to compute each stage's complex twiddle factor multiplication and four multipliers for the complex point multiply, the multipliers would require 22288 slices. Since the multipliers are fully-parallel, the clock rate for this implementation would be equal to the sample rate.

If the clock rate was quadrupled and each multiplier was reused four times per input sample (i.e. one multiplier per complex multiply), the multipliers would require 5572 slices. Reusing the multipliers any more times would be difficult, as a single multiplier would have to perform the multiplications for multiple pipeline stages of the FFT or IFFT.

Each pipeline stage requires words of storage for its delay line (previous chapter), where m is the stage number. This means that 15 complex words of storage are required for the FFT and 15 complex words are required for the IFFT. 8 complex twiddle factors are required for each FFT, but may be shared between the FFT and IFFT. Two banks of 32 complex words are required for the frequency spectrum storage so that one bank may be loaded while the other is active. Each complex word requires two RAM words of storage. With 31-bit words for the FFT delay storage, 29-bit words for the frequency spectrum storage, 36-bit words for the IFFT delay storage, and 36-bit words for the twiddle factor memory (to accommodate the 36-bit IFFT requirement), the total filter requires 6298 bits of memory storage.

One advantage of this implementation is that much of the memory storage consists of large blocks of RAM where only a single location needs to be accessed at a time. This means that the Virtex block RAM could be used for this application. The Virtex *XC4VSX35* has 192 blocks of 18kbit Block RAM, giving a total of 3456 Kbit of memory. The required memory of 6298 bits can easily be accomodated into this block RAM.

Therefore, the frequency multiplication technique for a 16-tap filter would require 5572 slices for multipliers and the Virtex BlockRAM, used for data storage. The control logic for this implementation is complex and would be area-intensive, especially with a single multiplier being re-used 4 times per input sample. In addition, adders, pipeline registers, twiddle factor distribution logic, and the control logic to implement the overlap-save method necessary to filter the continuous input stream would increase the area dramatically.

Although the pipelined FFT algorithm is (hence the name) highly pipelinable, a design reusing a single multiplier four times would not be regular and would require long route lengths due to the complex nature of its control, reducing its performance compared to the other, more regular designs. Therefore, the FFT algorithm was ruled as being larger and slower than the DA or FFA designs.

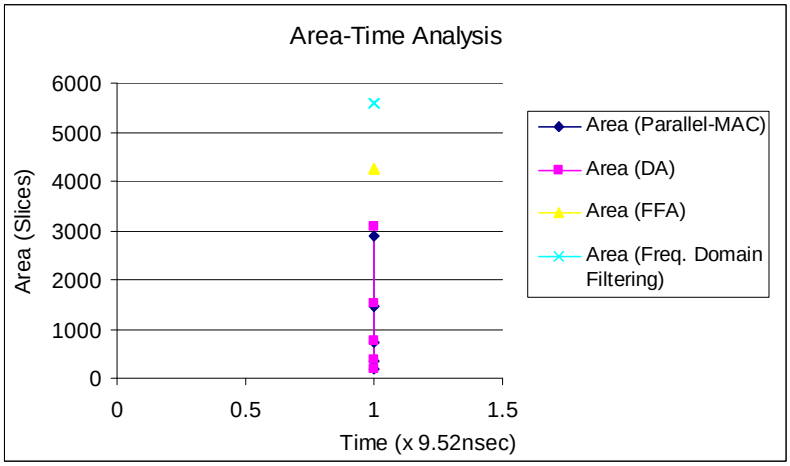
One point to note with the FFT algorithm is that moving from a 16-tap filter to a 32-tap filter would require a smaller area increase than moving from a 16-tap filter to a 32-tap filter using a FIR approach, as such a move requires the addition of one stage to the FFT and IFFT and twice as much twiddle factor and spectrum storage memory, whereas the FIR techniques would require a doubling of area. The FFT algorithm's area benefits would become even more obvious as the filter's size increased further, as each doubling of taps requires a doubling of area but a small increase in frequency spectrum filtering area.

6.5 Conclusion

In this chapter, we have started with straight forward approach of FIR filter implementation i.e. parallel-Multiple-Accumulate structure. We have explored different techniques in order to lower down the requirements for multiplier. Bit Systolic Array architecture is not suited for FPGA because of simplicity of its single cell which is actually a drawback, area-wise. Distributed Arithmetic structure is best suited for FPGA because of its efficently usage of slices. Frequency domain

filtering is the most resource consuming. The FFT algorithm's area benefits would become obvious as the filter's size increased, as each doubling of taps requires a doubling of area but a small increase in frequency spectrum filtering area.

So finally Distributed Arithmetic structure is delected for the final implementation to the FPGA because of being resource efficent. Area-time analysis of these different structured filters is shown in Figure 6.7.



Area-Time Analysis of Different filtering techniques. These are Parallel multiple and accumulate, Distributed Arithmetic, FFA and Frequency domain filtering. The curves show their performance having same clock rate but with different required areas.

Figure 6.7: Area-Time Analysis of different structured filters

In this all analysis, we have not consider the parameter of maximum operating speed for the designs, which will definately restrict some of the design for the application. This is due to the fact that combinational logic lower down the maximum operating speed for a designed system.

The DSP48 slice is a new element in the Xilinx development model referred to as Application Specific Modular Blocks (ASMBL) architecture. The purpose of this model is to deliver off-the-shelf programmable devices with the best mix of logic, memory, I/O, processors, clock management, and digital signal processing. ASMBL is an efficient FPGA development model for delivering off-the-shelf, flexible solutions ideally suited to different application domains.

Each XtremeDSP tile contains two DSP48 slices to form the basis of a versatile coarse-grain DSP architecture. Many DSP designs follow a multiply with addition. In Virtex-4 devices, these elements are supported in dedicated circuits. The DSP48 slices support many independent functions, including multiplier, multiplier-accumulator (MACC), multiplier followed by an adder, three-input adder, barrel shifter, wide bus multiplexers, magnitude comparator, or wide counter. The architecture also supports connecting multiple DSP48 slices to form wide math functions, DSP filters, and complex arithmetic without the use of general FPGA fabric.

The math portion of the DSP48 slice consists of an 18-bit x 18-bit, two's complement multiplier followed by three 48-bit datapath multiplexers (with outputs X, Y, and Z) followed by a

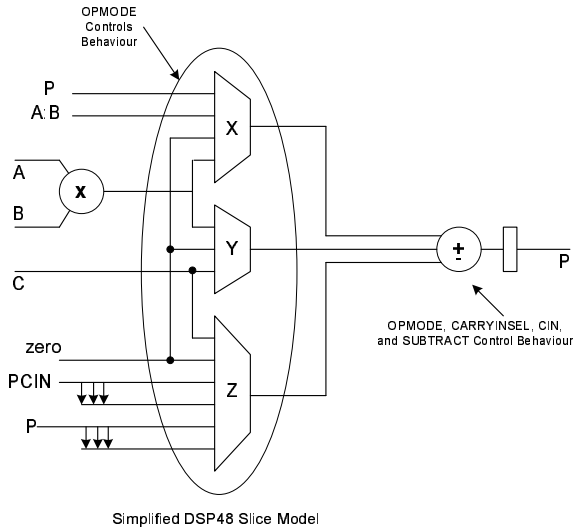


Figure 6.8: Simplified DSP48 Slice: The math portion of the DSP48 slice consists of an 18-bit x 18-bit, two's complement multiplier followed by three 48-bit datapath multiplexers (with outputs X, Y, and Z) followed by a three-input, 48-bit adder/subtracter.

three-input, 48-bit adder/subtracter as shown in Figure 6.8. The data and control inputs to the DSP48 slice feed the arithmetic portions directly or are optionally registered one or two times to assist the construction of different, highly pipelined, DSP application solutions. The data inputs A and B can be registered once or twice. The other data inputs and the control inputs can be registered once. Full speed operation is 500 MHz when using the pipeline registers.

The DSP48 slice is ideally suited to implement multirate sampling because of its high speed and filter-like structure. The cascaded data input and output paths, pipeline registers, high precision two's complement multiplier followed by an adder/subtracter and accumulation capability provide needed elements for multirate filtering.

The target platform Xilinx Virtex-IV XC4VSX35-10FF668C FPGA has 192 DSP48 slices. It is wise to use these resources instead of creating our own resources based on FPGA's slices/CLBs. 16-Taps filter can be efficiently implemented by using 16 DSP48 slices.

CHAPTER 7

CONCLUSION

7.1 Conclusions

The overall goal of this project is to design and implement a multi-standard software radio receiver. In a multi-standard secnario, WLAN and UMTS are taken as case study. The goal was stated in the form of an initial problem:

How can an efficient multi-standard (WLAN & UMTS) software radio receiver be designed and implemented.

In order to meet the goal, two project objectives were stated: 1) Select an algorithm / technique to obtain the desired functionality and 2) design and implement a system based on the selected algorithm. In the following, conclusions are made for these objectives.

In the selection of the technique for multi-standard software radio, bandpass sampling technique has been selected. The UMTS and WLAN standards have 12 and 3 channels respectively, but only one channel from each standard is required at a time. It is required to down convert these channels to baseband at the required sampling rate of 20MHz for WLAN and 61.44MHz for UMTS. Initially, the combined band of UMTS and WLAN is undersampled at 676MHz. The bandpass filters are requried to seperate these two bands of WLAN and UMTS. The application requirement demands a linear phased characteristic of the bandpass filter, as the channels of each standard are very compactly spaced, and therefore the FIR filters are desirable choice. Further, a brief overview of the different methods which are used to calculate the filter coefficients are compared. It is concluded that an **Optimal Filter** is required for the filtering purpose based on the application requirements. It is required to have that the minimum ripples in the passband which is provided by the least square criterion in the optimal filter design. Finally, a channelizer was required to obtain the desired functionality of downconversion and downsampling.

Two types of channelizer were considered in this case. The first one is the conventional channelizer which has a downconverter followed by a lowpass filter, and rate converters. In this approach, each channelizer works only for one of the channels. Thus to have multiple channels at the output, multiple channelizers are required. The second type is the polyphase channelizer which

has commutator, polyphase partitioned filter and beam former by coherent phase summation. It has unique features of having all the channels at baseband with required sampling rate just by using one filter (polyphase filter). It also has the features of efficient utilization of the resource with reduced clock-rate requirements. All the process to have a polyphase channelizer from the conventional channelizer is investigated. It uses the **Equivalence Theorem** and **Nobel Identity**. The polyphase channelizer has been selected based on these unique features.

In order to meet the requirements of polyphase channelizer i.e. equal channel spacing and integer number of channels over the span of sampling frequency, we have gone through different iterations. Different sampling frequencies are tried to have non-overlap alaised channels along with polyphase channelizer requirements of equal channel spacing and integer number of channels. The sampling frequency is iteratively changed to 840 MHz, but still not perfectly matched to one of the polyphase requirement of equal channel spacing. Unequal channel spacing results in baseband offset for the downconverted signals.

In order to compensate for the baseband offset, we use a variant of polyphase channelizer that has hyterdyning embedded in it. This variant polyphase channelizer is best for the offsets of multiples of quarter of channel spacing. Because of different channel bandwidth and channel spacing, both WLAN and UMTS require seperate polyphase channelizer. The resultant channelizers for WLAN and UMTS have 35 and 168 sub-filters respectively with the input sampling rate of 840MHz. The processing load on sub-filters in the polyphase channelizer is reduced by lowering the input sampling frequency (as low as possible) by resampling the signal. In this case, the input to bandpass filters in WLAN and UMTS path is considered as complex while making the filter coefficients as real, which means that the output of the bandpass filter is complex with less complexity i.e. multiplications and additions. This allow the ssampling frequency for WLAN and UMTS to be lowered to 120 and 105 MHz respectively, due to having no interference with image signals. The resultant channelizers for WLAN and UMTS now have 5 and 21 sub-filters, with the input sampling rate of 120 and 105MHz respectively.

In order to achieve the desired sampling rate at the output, different techniques are considered (such as) P/Q resampling and resampling embedded in the polyphase channelizer. Out of them, resampling embedded in the polyphase structure is selected. This is based on serpentine shift at the input data and circular shifting the date before the coherent phase summation. The technique is further modified to have sliding movement of commutator and circulating the filter coefficient.

In the simulations, the results are the same having a band-select filter (complete spectrum) followed by bandpass filter UMTS and WLAN) as only having bandpass filters which removes the need to select the complete band at the start. The signal power spectrum after passing through the bandpasss filter is still the same. The polyphase channelizers for WLAN and UMTS are simulated to shows the desired signal at baseband with the required sampling rates of 20MHz and approx. 61.44MHz respectively. The received signals have 50dB of dynamic range. The prototype filter for WLAN has 50 taps which are partitioned into 5 polyphase branches, so that each sub-filter has 10 coefficients. In UMTS channelizer, based on the required output sampling rate, a downfactor of approximately 17/10 is required, which has an upsampling of 10. This upsampling factor of 10 requires a 10 times longer prototype filter than the normal filter because of 10 times increase in

sampling frequency. The resultant prototype filter for UMTS has 2520 taps which are partitioned into 210 polyphase branches, so that each sub-filter has 12 coefficients. Only one tenth (1/10) of the sub-filters i.e. 21 out of 210 are used at a time. This saves the processing panelty due to increased filter length.

In the implementation phase, polyphase channelizers are analyzed in terms of the required components, consisting of demultiplexer as commutator, a filter bank having polyphase filters, and finally the coherent phase summation (multiply and accumulate). Different structural techniques to carry out the implementation were presented. In this regard, general polyphase structure, optimized structures - symmetric property based structure, adder shared structure, serial polyphase structures with serial and parallel MAC are considered. Based on the complexity analysis, serial polyphase structure with parallel MAC is selected for the final implementation.

In the individual sub-filter implemetation, different implementaion structures are considered. These being Parallel multipliers and accumulate, Bit systolic array, Distributed Arithmetic, Fast FIR, frequency domain filtering and Multiplier less filtering techniques. Each structure and its variants are analyzed in terms of hardware resources. The analysis is based on the approximations for the area requirements for multipliers, adders and registers etc. For 16-tap filter Parallel-Multiply and accumulate, Distributed Arithmetic, Fast FIR and Frequency domain filtering structures require 2896(without adders), 3072, 4064, and 5572 slices, respectively. The Distributed arithmetic is found to be suitable for the implementation due to being resource efficient.

The focus of the above techniques is to use multipliers as less as possible, to save the area. But due to technology advancement, now the FPGAs have dedicated multiplier blocks which are more efficient than the CLB-slices based multipliers in terms of operating speed and reduced power requirements. Xilinx FPGA, Virtex-IV has XtremeDSP blocks that can perform multiplication upto 500MHz rate. The system performance is increased by using these blocks. Each XtremeDSP block has two DSP48 slices. So the polyphase filter bank implemented as serial-polyphase-filter structure with parallel MAC for WLAN and UMTS channelizer can be built by using 10 and 12 DSP48 slices respectively.

We have choosen the target architecture from the start, and we have said that it can only be FPGA or ASIC that can meet the required clock frequency for this project. We know from the start that moderen FPGA's like Virtex-IV have dedicated multipliers which are higly optimised as explined above and at the end we are using them. But it was necessary to understand the knowledge of all the methods and techniques for the basic DSP operations and thats what we did. Now at this stage we can say that the VLSI design is possible to have optimized architecture for DSP operations based on the techniques that we have gone through. We can also guess that the technique with Programmable SOPOT have an architecture which is very similar to the architecture of the dedicated multipliers, (This is an assumption since the internal architectures of the dedicated multipliers are confidentials of Xilinx, and they are not available).

The A3 model was used to describe the problems of mapping the algorithm to a system architecture. Many solutions of an architecture existed for the algorithm and the architecture also had to comply with the constraints of the application. The many different solutions formed a

huge design space. Furthermore, the constraints on execution time and area consumption had to be maintained making the mapping between the algorithm and the architecture an iterative process.

Finally, the answer to the initial problem is given:

Bandpass sampling technique has been selected for the WLAN and UMTS standard receiver, which is followed by polyphase channelizers to have the channels at the baseband with required sampling rates. Serial polyphase-channelizer implementation structure with parallel MAC is used for implementation. Polyphase sub-filter is implemented with Distributed Arithmetic structure or with Xilinx-DSP48 slices for improved performance.

In the following section, the future perspectives of the project are given. These perspectives are based on the obtained results and considerations concerning the assumptions that have been made.

7.2 Future prospective

There is always room for improvement, and we also have some suggestions for the future work.

- In the project, we have considered a scenario by taking two standards (UMTS and WLAN), which can be expanded to include more standards.
- Polyphase channelizer is not used to its level best advantages of extracting all the channels at the same time. This is due to the fact the different standards have different channel bandwidth and inter-carrier spacing. Even for one of the standards, all of its sub-channels are not converted at the same time. This is due to the unequal channel spacing (from the DC). The polyphase channelizer can be used to its level best features that is extracting all of the channels for any standard, by having a heterodyning at the input of the polyphase channelizer, and heterodyning-carrier is selected such that the translated channels have equal channel spacing. This case will result in extracting all the channels of a standard, just by using standard polyphase channelizer, not by its variant to compensate the offsets of multiples of quarter of channel spacing.
- In the polyphase channelizer for UMTS, the required downfactor of 875/512 is rounded to 17/10, which results in the output sampling rate of 61.76 MHz instead of 61.44 MHz. Arbitrary sampling rate techique as mention in [Harris, 2006] can be used along with polyphase channelizer to have the exact required sampling rate of 61.44 MHz.
- The implementation part is based on the analysis of the required hardware resources by having estimates and writing simple VHDL programs. The full implementation of the system is a proposed future work.

BIBLIOGRAPHY

[CSD, 2007] (2007).
Stream radio goes digital.
http://www.csdr.dk.

[A.G. Dempster, 1995] A.G. Dempster, M. M. (1995).
Use of minimum adder and multiplier blocks in fir digital filter.
IEEE Trans. circuit system II.

[Al-Haj, 2004] Al-Haj, A. M. (2004).
An FPGA-Based Parallel Distributed Arithmetic Implementation of the 1-D Discrete Wavelet Transform.
Department of Computer Engineering, Princess Sumaya University for Technology, Al-Jubeiha P.O.Box 1438, Amman 11941, Jordan. http://ai.ijs.si/informatica/PDF/29-2/13_Al-Haj-An%20FPGA-Based%20Parallel...pdf.

[Alan V. Oppenheim, 1999] Alan V. Oppenheim, R. W. S. (1999).
Discrete-Time Signal Processing.
Prentice Hall, second edition.

[Behjou Nastaran, 2006] Behjou Nastaran, Priyanto, B. E. J. O. K. L. T. (2006).
Interference issues between umts & wlan in a multi-standard rf receiver.
IST Mobile Wireless Comms Summit.

[Brannon,] Brannon, B.
Designing a Superheterodyne Receiver Using an IF Sampling Diversity Chipset,AN-502 APPLICATION NOTE.
http://www.analog.com.

[Chin-Liang Wang and Chen, 1988] Chin-Liang Wang, C.-H. W. and Chen, S.-H. (1988).
Efficient bit-level systolic array implementation of fir and iir digital filters.
IEEE Journal on Selected Areas in Communications, 6(3):484–493.

[Chris Dick,] Chris Dick, f. h.
Performing Simultaneous Arbitrary Spectral Translation and Sample Rate Change, in Polyphase Interpolating or Decimating Filters in Transmitters and Receivers.
www.xilinx.com/products/logicore/dsp/sdr_paper_1.pdf.

[Cook, 2006] Cook, P. G. (2006).

Sdrf cognitive definition.
http://www.sdrforum.org.

[Crochiere and Rabiner, 1983] Crochiere, R. and Rabiner, L. (1983).
Multirate Digital Signal Processing.
Prentice Hall, Englewood cliff, NJ.

[David R. Martinez and Teitelbaum, 2000] David R. Martinez, T. J. M. and Teitelbaum, K. (2000).
Application of reconfigurable computing to a high performance front-end radar signal processor.
Journal of VLSI Signal. Kluwer Academic Publishers Processing Systems.

[Dennis M. Akos and Caschera, 1999] Dennis M. Akos, Michael Stockmaster, J. B. Y. T. and Caschera, J. (1999).
Direct bandpass sampling of multiple distinct rf signals.
IEEE TRANSACTIONS ON COMMUNICATIONS, 47(7).

[Emmanuel C. Ifeachor, 2002] Emmanuel C. Ifeachor, B. W. j. (2002).
Digital Signal Processing (A Practical Approach).
Pearson Education Ltd., second edition.

[Fredric J. Harris and Rice, 2003] Fredric J. Harris, C. D. and Rice, M. (2003).
Digital receivers and transmitters using polyphase filter banks for wireless communications.
IEEE Transactios on microwave theory and techniques, 51(4).

[Groginsky and Works.,] Groginsky, H. L. and Works., G. A.
A pipeline fast fourier transform [Liu, 1975].
pages 369–373.

[Harris, 2006] Harris, F. J. (2006).
Multirate Signal Processing for Communication Systems.
Prentice Hall.

[Haykin, 2002] Haykin, S. (2002).
Adaptive Filter Theory.
Prentice Hall, 4. edition.

[J. H. McClellan and Rabiner, 1973] J. H. McClellan, T. W. P. and Rabiner, L. R. (1973).
Computer program for designing optimum fir linear phase digital filters.
IEEE Trans. Audio Electroacoust.

[Jin-Cyun Chung and Wang., 1998] Jin-Cyun Chung, Yong-Bae Kim, H.-G. J. K. K. P. and Wang., Z. (1998).
Efficient parallel fir filter implementations using frequency spectrum characteristics.
Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS '98), pages 354–358.

[K.S Yeung, 2002] K.S Yeung, S. C. (2002).
Multiplier-less fir digital filters using programmable sopot coefficients.
Field-Programmable Technology, 2002. (FPT). Proceedings. 2002 IEEE International Conference.

[Liu, 1975] Liu, E. (1975).
Digital Filters and the Fast Fourier Transform (Benchmark papers in electrical engineering and computer science ; v. 12).
John Wiley & Sons Inc.

[Moeller, 1999] Moeller, T. J. (1999).
Field Programmable Gate Arrays for Radar Front-End Digital Signal Processing.
Master Thesis at MIT. http://cag.csail.mit.edu/šaman/student_thesis/Tyler-Moeller-99.pdf.

[Moeller and Martinez, 1999] Moeller, T. J. and Martinez, D. R. (1999).
Field programmable gate array based radar front-end digital signal processing.
In IEEE Symposium on Field-Programmable Custom Computing Machines, IEEE Computer Society, pages 21–23.

[Murphy,] Murphy, P.
Digital Communications using FPGAs & Xilinx System Generator.
October 5-6, 2006,IIT Delhi, India. http://cmclab.rice.edu/workshops/materials/2006-10-05_DigitalComm/Rice_Comm_Workshop_Slides.pdf.

[Parker and Parhi, 1997] Parker, D. A. and Parhi, K. K. (1997).
Low-area/power parallel fir digital filter implementation.
Journal of VLSI Signal Processing, 17:75–92.

[Patel and Lane,] Patel, M. and Lane, P.
Comparison of downconversion techniques for software radio.
Department of Electronics and Electrical Engineering, University College London.
www.ee.ucl.ac.uk/lcs/papers2000/lcs050.pdf.

[Raghu Rao,] Raghu Rao, Matthieu Tisserand, M. S. P. J. V.
FPGA Polyphase Filter Bank Study & Implementation.
Image Communications/Reconfigurable Computing Lab. Electrical Engineering Dept.UCLA http://slaac.east.isi.edu/presentations/retreat_9909/polyphase.pdf.

[Ramjee Prasad, 2002] Ramjee Prasad, H. H. (2002).
Simulation and software radio for mobile comm.
Artech House.

[Xilinx,] Xilinx, P.
The role of distributed arithmetic in FPGA based signal processing.
http://www.xilinx.com/appnotes/theory1.pdf.

[Xilinx_Multiplier, 2000] Xilinx_Multiplier (2000).
Variable Parallel Virtex Multiplier V2.0.
http://www.xilinx.com/ipcenter/catalog/logicore/docs/mult_vgen_v2_0.pdf.

APPENDIX A

MULTIRATE SIGNAL PROCESSING

A.1 Introduction

The increasing need in modern digital systems to process data at more than one sampling rate has led to the development of a new sub-area in DSP known as multirate processing. A straight forward approach is to convert the digital signal back to analog and the resampled at the desired rate. However, this is not a desireable approach, because of the non-ideal analog reconstruction filter, D/A converter, and A/D converter that would be used in a practical implementation. Thus it is of interest to consider methods of changing the sampling rate that involve only discrete-time operations [Alan V. Oppenheim, 1999].

The two primary operations in multirate processing are decimation and interpotaion and they enable the data rate to be altered in an efficeient manner [Emmanuel C. Ifeachor, 2002]. Decimation reduces the sampling rate (sampling frequency), effictively compressing the data and retaining only the desired information. Interpolation, on the otherhand increases the sampling rate. Often the purpose of converting the data to a new rate is to make it easier (e.g. computationally more efficient) to process or to achieve compatibility with another system.

There are many advantages of multirate processing which have been exploited in many and increasing number of modern systems. Some of them are [Emmanuel C. Ifeachor, 2002]:

- High quality data acquisition and storage systems uses multirate techniques. Analog signal is sampled at much higher frequency than specified by the sampling theorem, which requires a much simpler anti-aliasing filter to bandlimit it before digitizing. Once in the digital form the signal can be readily reduced to desired rate using the multirate approach.
- In speech processing, multirate techniques are used to reduce the storage space or the transmission rate of speech data. Estimates of speech parameters are computed at a very low sampling rate for storage or transmission. When required, the original speech is reconstructed from the low bit-rate represenation at much higher rates using multirate approach.
- Multirate processing has found important application in the efficient implementaion of DSP functions. For example, the implementation of narrow band digital FIR (Finite Impulse

response) filters using conventional DSP poses a serious problem because such filters require a very large number of coefficients to meet their tight frequency response specifications. The use of multirate techniques leads to very efficient implemetation by allowing filtering to be performed at a much lower rate, which greatly reduces the filter order.

Multirate processing allows the strength of the conventional DSP to be exploited. Anti-aliasing and anti-imaging filtering in real time DSP systems can be performed in the digital domain, enabling both sharp magnitude frequency as well as linear phase responses.

A.2 Sampling rate reduction (Decimation)

The sampling rate of a sequence can be reduced by sampling it i.e. by defining a new sequence.

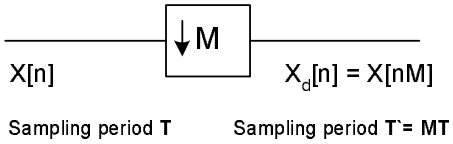


Figure A.1: Sampler rate compressor or decimator

Figure A.1 shows the M-fold decimator, which takes an input sequence $x[n]$ and produces the output sequence $x_d[n]$

$$x_d[n] = x[nM] \tag{A.1}$$

where M is an integer. Only those samples of $x[n]$ which occur at time equal to multiples of M are retained by the decimator. More precisely, sampling rate reduction is achieved by discarding $M-1$ samples for every M samples of the signal. Equation A.1 defines the system shown in figure A.1 and called the sampling rate compressor or simply compressor.

Figures A.2 illustrate the decimation process of a signal. Figure A.2a is the Fourier transform of the impulse train of samples with sampling period of T, ($\Omega_N T = \pi/2$) i.e. sampling rate is twice the minimum rate to avoid aliasing. Figure A.2b is the Fourier transform of the downsampled sequence when M=2. As the original sampling rate is twice the minimum sampling rate and the downsampled factor M is also 2, so no aliasing occur. If the downsampled facor M is more than 2 then alaising will result, as shown in figure A.2c.

Decimation results in aliasing unless $x[n]$ is bandlimited. Sampling rate can be reduced by a factor M without aliasing if the original sampling rate was at least M times the Nyquist rate or if the bandwidth of the sequence is first reduced by a factor of M by discrete-time filtering. The downsampling process accompanied by discrete-time filtering is shown in figure A.3. Figure A.3a is the Fourier transform of the impulse train of samples with sampling period of T ($\Omega_N T = \pi/2$) and is same as shown in figure A.3a. In general, to avoid alaising in downsampling by a factor of M requires $\Omega_N < \pi/M$. In order to downsampled by a factor M=3 without aliasing, signal has to be bandlimited before downsampling. Thus if the signal $x[n]$ is filtered by an ideal lowpass filter with cutoff frequency $\Omega_c = \pi/M = \pi/3$, then the signal can be downsampled ($M = 3$) without aliasing, as shown in figure A.3d.

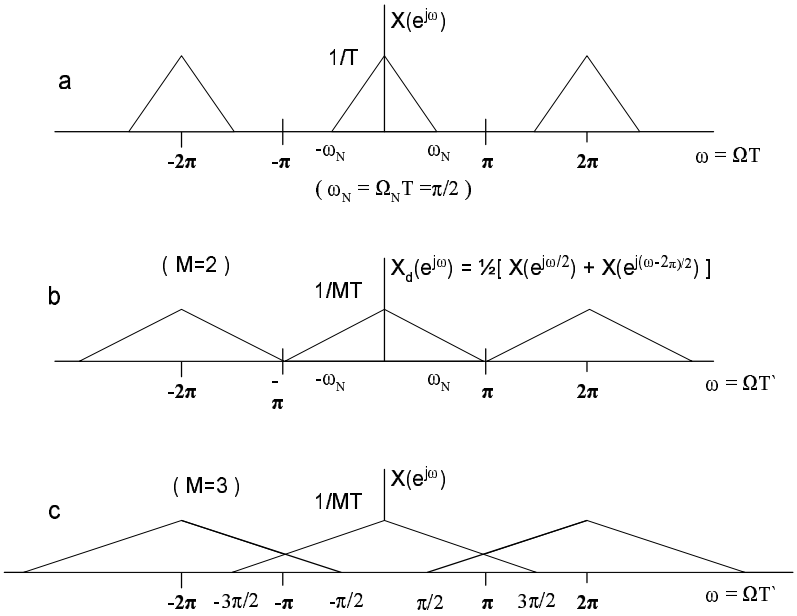


Figure A.2: frequency-domain representaion of downsampling

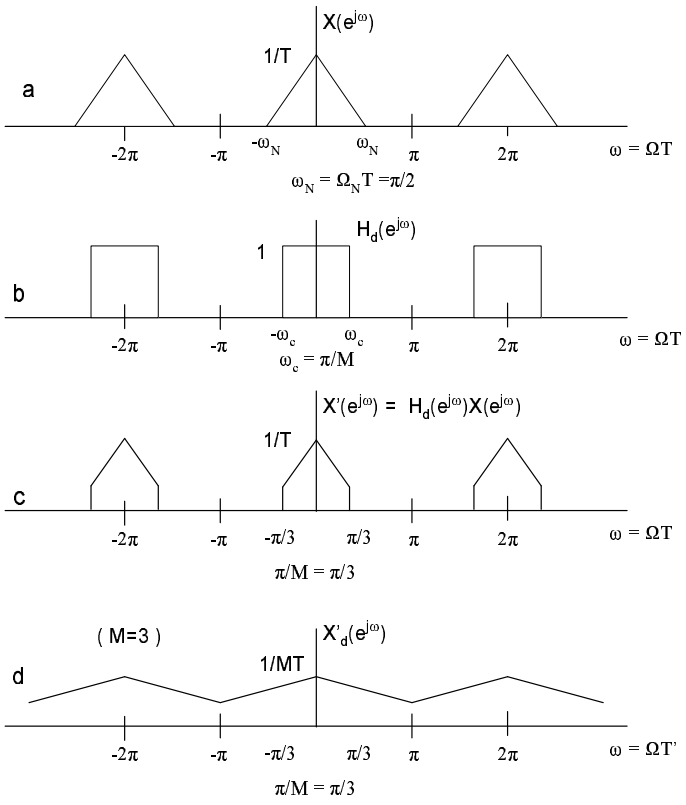


Figure A.3:

A general system for downsampling by a factor of M is shown in figure A.4. Such a system is called decimator, and downsampling by lowpass filtering followed by compression has been termed decimation [Crochiere and Rabiner, 1983].

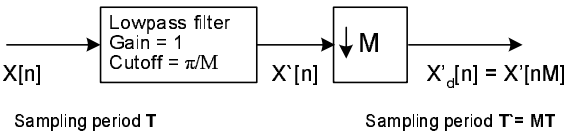


Figure A.4: General system for sampling rate reduction by facor M [Alan V. Oppenheim, 1999]

A.3 Sampling rate expansion (expander)

The sampling rate of a sequence can be increased by operation analoqous to D/C conversion.

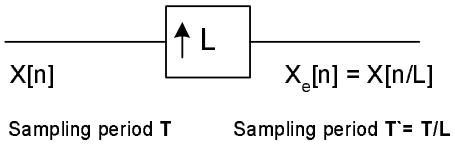


Figure A.5: Sampler rate expansion or expander

Figure A.5 shows the L-fold expander, which takes an input sequence $x[n]$ and produces the output sequence $x_e[n]$

$$x_e[n] = \begin{cases} x[n/L] & \text{if n is integer multiple of L} \\ 0 & \text{otherwise} \end{cases} \tag{A.2}$$

where L is an integer. For each sample of $x[n]$ the expander inserts $L-1$ zero-valued samples to form the new signal at rate LF_s (where F_s is the original sampling rate). The signal is then lowpass filtered to remove image frequencies created by rate increase. The insertion of $L-1$ zero spread the energy of each signal sample over L output samples, effectively attenuating each sample by a factor of L. Thus it is necessary to compensate for this, for example by multiplying each sample of $x_e[n]$ by L. Equation A.2 defines the system shown in figure A.5 and called the sampling rate expander or simply expander.

Figures A.6 illustrate the upsampling process of a signal. Figure A.6a is the Fourier transform of the impulse train of samples with sampling period of T, ($\Omega_N T = \pi$) i.e. sampling rate is equal to the minimum rate to avoid aliasing. Figure A.6b is the Fourier transform of the upsampled sequence when L=2. The new upsampled sequence can be obtained from expended sequence as shown in figure A.6b by correcting the amplitude scale from 1/T to 1/T' and by removing all the image signals except at integer multiples of 2π. So it requires alowpass filter with a gain of 2 and a cuttoff frequency $\pi/2$ as shown in figure A.6c. In general, required gain would be L and the cuttoff frequency would be π/L. The final upsampled signal without aliases is shown in figure A.6d.

A general system for upsampling by a factor of L is shown in figure A.7. Such a system is called expander or interpolator as it fills in the missing samples.

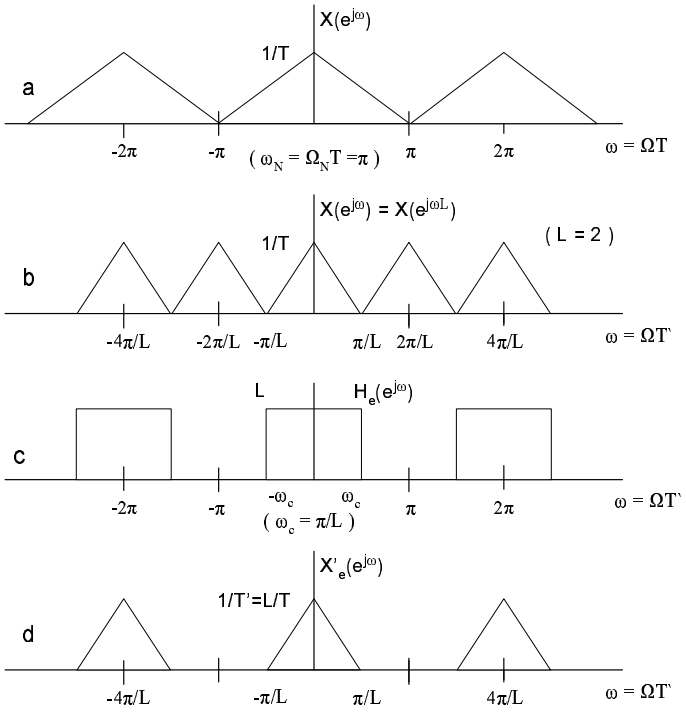


Figure A.6: frequency-domain representaion of upsampling

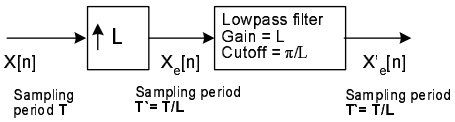


Figure A.7: General system for sampling rate increase by facor L [Alan V. Oppenheim, 1999]

A.4 Changing the sample rate by non-integer factor

In the previous sections, methods for sample rate increase and decrease by a integer factor has been presented. By combining these operation i.e. decimation and interpolation, sampling rate can be changed to a non-integer factor.

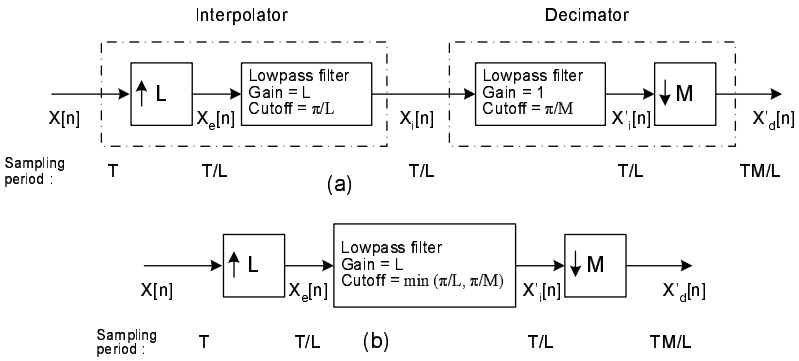


Figure A.8: [Alan V. Oppenheim, 1999]

Figure A.8a shows an interpolator that decreases the sampling period from T to T/L, followed by a decimator that increases the sampling period by M, producing an output sequence $x'_d[n]$ that has an effective sampling period of $T' = TM/L$. By choosing L and M appropriately, any desired ratio of sampling period can be achieved.
If M>L, there is a net increase in the sampling period (a decrease in the sampling rate) and if M<L, then there is net decrease in sampling period (a increase in the sampling rate). As the interpolation and decimation filters in figure A.8a are cascaded, they can be combined to form a single lowpass filter with gain L and cutoff frequency equal to the minimum of π/L and π/M as shown in figure A.8b.

Table A.1 shows the characteristics of the combined lowpass filter.

Combined filter behavior for non-interger sample rate conversion		
Cases	**Dominant cutoff frequency**	**Effect**
M > L	π/M	Reduction in sampling rate (Increase in sampling period)
M < L	π/L	Increase in sampling rate (Reduction in sampling period)

Table A.1:

In the previous section, methods for changing sample rate by combination of decimation and interpolation has been presented. If a new sampling period of T'=1.01T is required, the input sequence is first interpolated by L=100 using a lowpass filter that cuts off at $W_c = \pi/101$, and then decimate by M=101. These large intermediate changes in the sampling rate would require large amount of computation for each output sample if filteration is implemented in a straight forward manner at the high intermediate sampling rate that is required. It is possible to greatly reduce

the amount of computation required by taking the advantages of some basic multirate processing techniques. These techniques include identities

A.5 Polyphase Decomposition

The polyphase decomposition of a sequence is obtained by representing it as a superposition of M subsequnces, each consisting of every mth value of successfully delayed version of the sequence. By appling this decomposition to filter impusle respose, efficient implementation structure for linear filters can be obtained. Conside an impulse response $H[n]$ that has to be decompose into M sebsequnces $h_k[n]$ as follows:

$$h_k[n] = \begin{cases} h[n+k] & \text{if n is integer multiple of M} \\ 0 & \text{otherwise} \end{cases} \tag{A.3}$$

By successfully delaying these subsequences, the original impluse response $h[n]$ can be constructed; i.e.

$$h[n] = \sum_{k=0}^{M-1} h_k[n-k] \tag{A.4}$$

This decomposition can be represented by block diagram as shown in figure A.9. The sequences $e_k[n]$ are

$$e_k[n] = h[nM+k] = h_k[nM] \tag{A.5}$$

and are reffered as polyphase components of h[n]. Figure A.10 shows a chain of advance elements at the input and a chain of delay elements at the output and is equivalent to figure A.9.

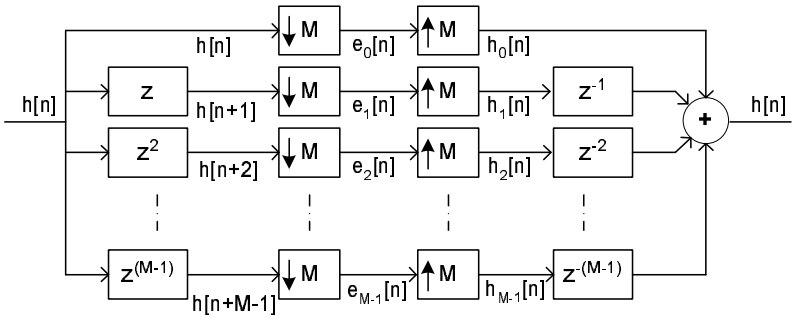


Figure A.9: Polyphase decomposition of filter h[n] using components $e_k[n]$ [Alan V. Oppenheim, 1999]

Figures A.9 and A.10 are not realization of the filter, but they show the decomposition of filter into M parallel filters. In frequency or z-domain, the polyphase components are represented as:

$$H(z) = \sum_{k=0}^{M-1} E_k(z^M) z^{-k} \tag{A.6}$$

The equation A.6 expresses the system function H(z) as a sum of delayed polyphase components filters, as shown in figure A.11.

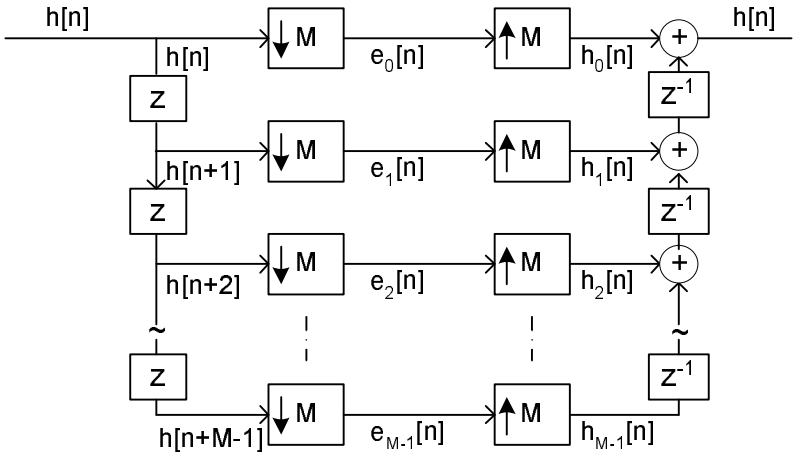


Figure A.10: Polyphase decomposition of filter h[n] using components $e_k[n]$ with chained delays [Alan V. Oppenheim, 1999]

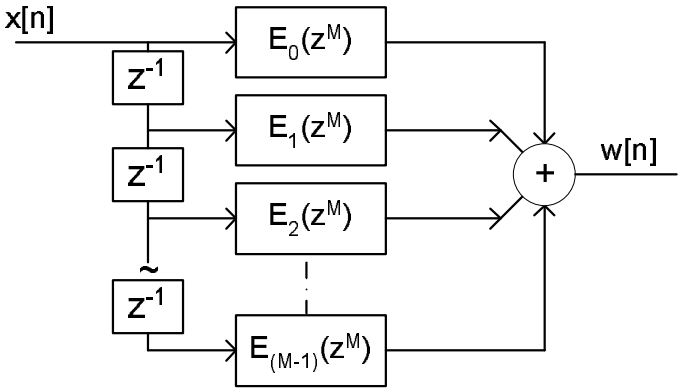


Figure A.11: Realization structure based on polyphase decomposition of h[n] [Alan V. Oppenheim, 1999]

Important applications of polyphase decomposition is in the filters whose output is then downsampled or upsampled.

A.6 Polyphase Implementation of Decimation filters

In the general system for downsampling by a factor of M as describe in section A.2 in figure A.4, the filter computes an output sample at each value of n, but then only one of every M output samples are retained. There should be a more efficient implementation which does not compute the samples that are just thrown away.

To obtain a efficient implementation, polyphase implemetation of filter can be exploited. The general system for downsampling as shown in figure A.12 can be implemented by its polyphase components, shown in figure A.13.

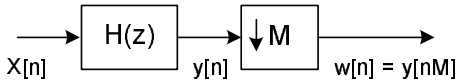


Figure A.12: General system for sampling rate reduction by facor M

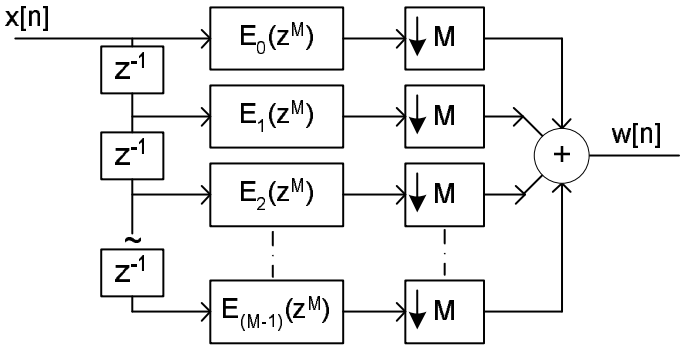


Figure A.13:

A more efficient implementation can be achieved by exploiting the noble identity as shown in figure A.14, which shows that a filter processing every Mth input sample followed by an output downsampler M is same as an input M down sampler followed by a filter processing every Mth input sample.

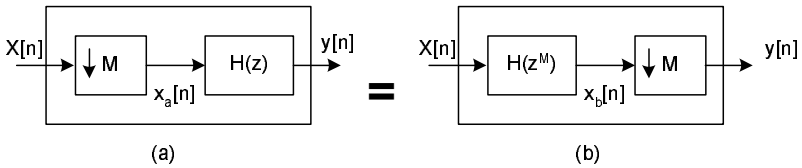


Figure A.14: Noble Identity which shows that a filter processing every Mth input sample followed by an output downsampler M (A.14b), is same as an input M down sampler followed by a filter processing every Mth input sample (A.14a)

Applying the nobel identity to decimator structure shown in figure A.13, the resulted efficient structure is shown in figure A.15. For computational efficiency of the structure shown in figure A.15, consider an input x[n] clocked at a rate of 1 sample per unit time and H(z) an N-point FIR filter. Straight forward implementation of decimator shown in figure A.12 would require N multiplications and (N-1) additions per unit time. In the structure shown in figure A.15, each of filter $E_z(k)$ is of length N/M, and their inputs are clocks at the rate of 1 per M units of time. So each filter requires $1/M(N/M)$ multiplications per time and $(N/M-1)+(M-1)$ additions per unit time, resulting in significant saving for values of M and N.

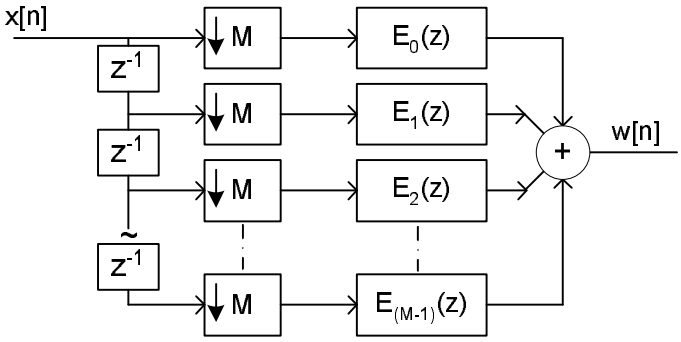


Figure A.15:

A.7 Polyphase Implementation of Interpolation filters

An efficient implementation as presented for decimation filters in previous section, can also be achieved for interpolation filters shown in figure. Since only every Lth sample of w[n] is nonzero, the most stsright forward implementation of figure would require applying filter cofficient to sequence that are known to be zero. So there should be some efficient implementation.

To obtain a efficient implementation, polyphase implemetation of filter can be exploited. The general system for upsampling as shown in figure can be implemented by its polyphase components, shown in figure A.13.

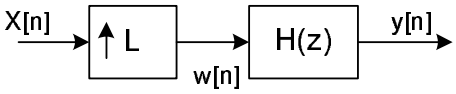


Figure A.16: General system for sampling rate increase by factor L

A more efficient implementation can be achieved by exploiting the noble identity for upsampling shown in figure , which shows that a filter processing every input sample followed by an output upsampler L (A.18b), is same as an input L up sampler followed by a filter processing every Lth input sample. The resulted rearranged upsampler system is shown in figure A.19.

For computational efficiency of the structure shown in figure A.19, consider an input x[n] clocked at a rate of 1 sample per unit time and H(z) an N-point FIR filter. Straight forward

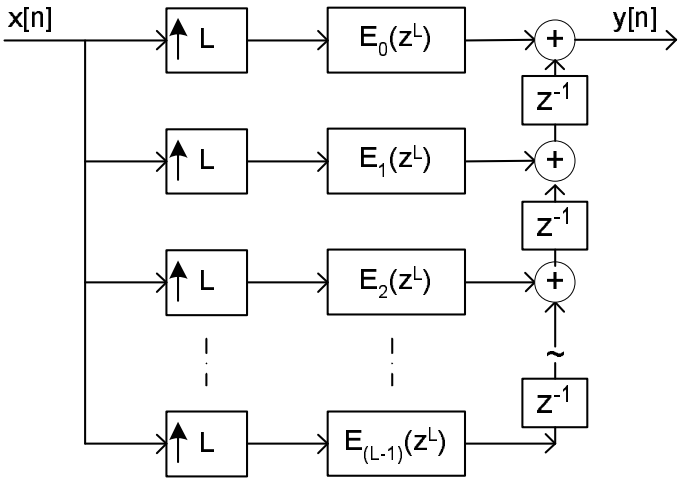


Figure A.17:

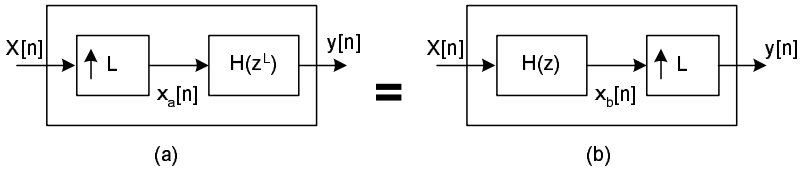


Figure A.18: Noble Identity for upsamping which shows that a filter processing every input sample followed by an output upsampler L (A.18b), is same as an input L up sampler followed by a filter processing every Lth input sample (A.18a).

implementation of upsampler shown in figure A.16 would require NL multiplications and (NL-1) additions per unit time. In the structure shown in figure A.19, each of filter $E_z(k)$ requires $L(N/L)$ multiplications and $L(N/L-1)$ additions per unit time, plus (L-1) additions to obtain output $y[n]$. It results in a significant saving for values of L and N.

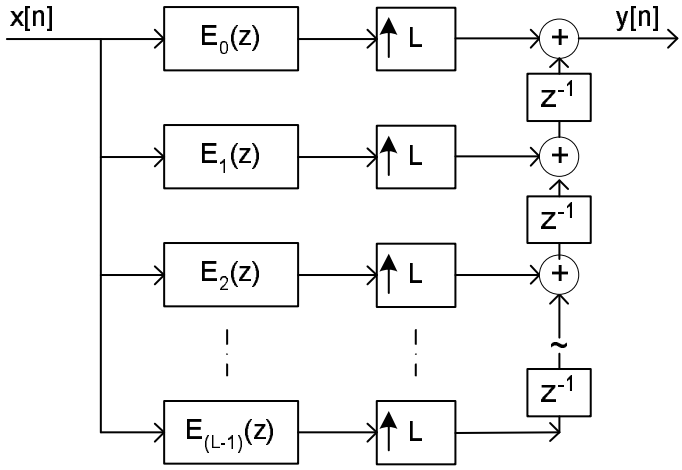


Figure A.19:

Gain in computational efficieny for both the decimation and interpolation results from the re-arranging the operations so that the filtering is done at the low sampling rate.

A.8 Multi-stage filters

Changes in sampling rate can be achieved in single stage using one decimator or interpolator. When large changes in sampling rate is required, it is more efficient to change the rate in two or more than two stages [Emmanuel C. Ifeachor, 2002]. Most practical multirate systems employ the multistage approach because it allows a gradual change in the sampling rate, leading to a significant relaxtion in the requirements of anti-aliasing or anti-imaging filter at each stage. Figure shows an I-stage decimator. The overall decimation factor M, is expressed as the product of the smaller factors:

$$M = M_1\ M_2\ M_3\ .....\ M_I\ =\ N \tag{A.7}$$

Each stage is independent as shown in dashed boxes. If M»1 the multistage approach leads to much reduced computational and storage requirements, a relaxation in the characteristics of the filters used in the decimators, and consequently to filters that are less sensitive to finite wordlength effects. But these advantages are achieved at the expense of increased difficulty in the design and implemetation of the system [Emmanuel C. Ifeachor, 2002].

The design of a practical multiple stage sample rate converter can be broken down into following four steps [Emmanuel C. Ifeachor, 2002]:

- Specify the overal anti-aliasing or anti-imaging filters and those for individual stages;
- Determine the optimum number of stages of decimation or interpolation that yield the most efficient implementation;

- Determine the decimation or interpolation factors for each stage;
- Design an appropriate filter for each stage.

The performance of multirate system depends critically on the type and quality of the filter used. Either FIR or IIR filters can be used for decimation or interpolation, but the FIR is the more popular. FIR filters have desirable attributes like linear phase response and low sensitivity to finite wordlength effect, as well as being simple to effect. In particular, the optimal and half-band filters are widely used [Emmanuel C. Ifeachor, 2002].

APPENDIX B

VIRTEX-FPGA

FPGAs are similar to custom designed chips in that they implement specific circuitry for a particular function. The major difference is that a FPGA is configured by a bitstream instead of by being hardwired through fabrication at a factory. This means that a FPGA's internal circuitry may be altered an unlimited number of times.

FPGAs may be classified as "coarse-grained" or "fine-grained", referring to the number and complexity of each basic logic element in the FPGA. Xilinx Virtex series chips are coarse-grained, and have logic units based on look-up tables (LUTs) and registers. The basic Virtex logic element is a Configurable Logic Block (CLB) slice. Two slices are present in each CLB. Each slice contains two 4-input, 1-output LUTs and two registers. The interconnections between these elements are configured by multiplexers controlled by SRAM cells programmed by a user's bitstream. The LUTs allow any function of five inputs, any two functions of four inputs, or some functions of up to nine inputs to be created within a CLB slice. The outputs of these functions may be registered, or the registers may be used independently of the LUTs. This structure allows a very powerful method of implementing arbitrary, complex digital logic.

The Xilinx slices also have the ability to implement distributed memory instead of logic. Each 4-input LUT in a slice may be used to implement a 16x1 ROM or RAM, or the two LUTs may be combined together to create a 32x1 ROM or RAM or a 16x1 dual-port RAM. This allows each slice to trade logic resources for memory in order to maximize the resources available for a particular application.

The CLBs in a Virtex FPGA are connected via programmable interconnect called the general purpose routing. This interconnect consists of differing length lines, some connecting adjacent CLBs together, while some span the entire length of the chip and others are designed for high fanout signals such as clocks. The connections between the interconnect and the CLBS are controlled by switch matrices called general routing matrices (GRMs). The programmable interconnect allows mappings that require local communication to be handled efficiently along with requirements for arbitrary, longer-distance, routing demands. In addition to the programmable interconnect, there are a few dedicated routing resources. One example is the carry-chains between CLBs that

allow high-speed carry propagation through a series of slices, enabling high-speed adders and other arithmetic units to be designed in a chain of CLBs. Connections between the internal routing and the external world are made through Input/Output Blocks, or IOBs, which contain input/output registers and connect directly to a package pin.

The Virtex FPGAs also include Block RAM units on the edges of the FPGA. These resources are ideal when large amounts of memory are required that would not use the small, distributed CLB-based memory efficiently. Finally, the Virtex also has advanced clock management resources built in, including a delay locked loop (DLL) that reduces clock skew and can divide (by up to 16) or multiply (by 2) external clocks for slower or faster internal clocking.

The highly replicated, register rich architecture of the Virtex makes it suitable for custom computing applications. Each slice can perform a two-bit computation or look-up, allowing a systolic structure of processors to be built out of the regular array of CLBs in a Virtex. There are cases when a finer grain structure may be more efficient, as the CLB structure may not be the most efficient medium for very small systolic-cell based arithmetic.

The VirtexIV-XC4VSX35, has 15,360 CLB slices and 3,456Kb Block RAM. More details can be seen on Virtex datasheets.

B.1 Computational Unit Implementation

To illustrate the capacity of a slice, two commonly used DSP computational units, an adder and a multiplier, are presented with their area in terms of CLB slices. The Virtex has dedicated fast-carry chain resources built into each CLB. Two adder bits can fit into a single slice so that a b-bit adder consumes b/2 CLB slices. A 16-bit adder would require 8 slices. [Xil98a] The Virtex also has dedicated multiplication resources so that two multiplication bits can fit into a single slice. An a-bit by b-bit multiplier requires approximately

$$\frac{\mathbf{b}\, log_2\mathbf{b} + (\mathbf{b}-1)\mathbf{a}}{2} \tag{B.1}$$

CLB slices. A 16x16-bit multiplier would require about 152 slices.