DSP CHANNELIZER IMPLEMENTAION ON FPGA

BY

Dasanayaka C.D.B. E/13/050

Dayananda R.M.N E/13/053

Project G13

SUPERVISED BY

Dr. A.U.A.W Gunawardena

Mr. Sanka Piyarathne

Department of Electrical and Electronic Engineering

Faculty of Engineering

University of Peradeniya

Peradeniya, Sri Lanka.

February, 2019

ABSTRACT

Channelizers are widely used in modern digital communication systems. Advanced uniform multirate channelization has been theoretically proved to be capable of reducing the computational load, with better performance. In this project a chanelizer with polyphase architecture is designed and Register Ransfer Level(RTL) level simulation is done using Vivado HLS(High Level Synthesis) tool for comprehensive evaluation of resource usage, performance ,and frequency response. A Quadrature Phase Shift Key(QPSK) modulated and sampled data sequence was used as the input signal .Outputs of the channels and performances of the chanelizer were observed.It was observed that the latency is lesser when using fixed point representation for numerical values than when using floating point representation. Pipelining was also used in order to increase the performance of the chanelizer.Chanelizer architecture was also implemented using C language to compare the outputs of RTL simulation and software simulation and to verify the results.

Keywords- Polyphase ,FPGA ,High Level Synthesis, Filter bank, Digital channelizer

ACKNOWLEDGEMENT

First of all, We would like to thank our supervisor Dr. Aruna Gunawardena and Mr. Sanka Piyarathne. Thank you for your supervision, kindness and patience. We are deeply grateful for all the help we received during this project.

TABLE OF CONTENTS

List Of figures

List of tables

CHAPTER 1: INTRODUCTION

In Modern communication systems, flexibility, reconfigurability, and support for dynamic spectrum allocation techniques are becoming more important .Support for these can be facilitated by reconfigurable radio and software defined radio(SDR) systems. A Key element of these systems is the receiver channelizer which is responsible for extracting independent channels from the received signal through band pass filtering and down conversion.

Poly phase filters and filter-banks are one of the outstanding channelization examples to represent multi rate digital filters They offer a significant reduction in processing complexity by way of separating the input signals into several channels. Poly phase filter banks are widely used in industry, such as in the MP3 audio format The theory of poly phase filter banks was formed in the 80s and has been further developed since then.

The reason of implementing the theoretic poly phase filter-bank on an FPGA is that an FPGA has sufficient resources and a high performance that can allow the implementation of a large number of DSP algorithms very efficiently compared to a single chip processor.

1.1 literature review

Several poly phase DFT-FB (Discrete Fourier Transform-Filter Bank) FPGA Implementations have already been realized[6]. Generalized DFT-FB implementation has been done using IP cores.[4] Coefficient decimated poly phase filter bank has been implemented[5].There are no much work has done using FPGA

In this project VHDL is going to be used for the RTL(Register Transfer Level) design and Xilinx FPGA boards will be used. To save time VHDL code is going to be created using Vivado HLS (High level synthesis) software.

1.2 Methodology

First filter architecture is designed .Then it is coded in C language to test the performances. Then a proper C code is developed for real time processing considering samples. Then it is converted to a VHDL code using Vivado HLS software. Then FPGA board is programmed .after that verification is done as in figure 1.1

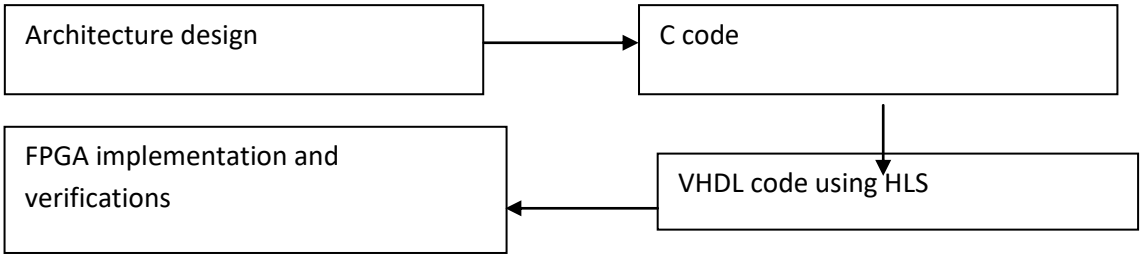


Figure 1.1 Methodology

CHAPTER 2: BACKGROUND

2.1 DSP theories

2.1.1 Channelization Introduction

Channelization is part of a digital signal processing that divides the wideband into separate channels, and down converts them to baseband, extracting one or more desired channels. The channels may have uniform or non-uniform allocation. Normally channelization is implemented by a down-converter and a low-pass filter Figure_2.1illustrates a 4 channels channelizer. This wideband input signal has 4 interested channels, each of them is being filtered and down-converted to DC (baseband frequency), and is ready for following processing.

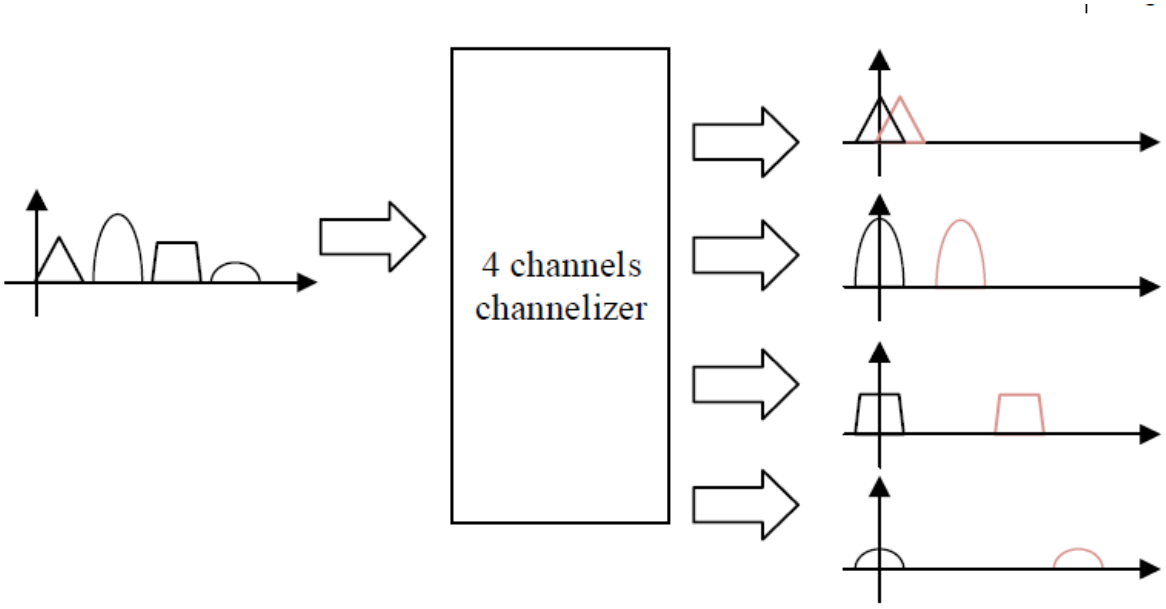


Figure 2.1:four channel channelizer

2.1.2.Channelization approaches

2.1.2.1 Per channel approche

This approach operates K independent one-channel channelizers in parallel, where K is the number of channels. Each sub channel extracts one channel of interest in the wideband input The 'per-channel approach' provides a high level of flexibility in the choice of channels' centre frequencies and bandwidths. Channels do not have the constraints of equal bandwidth or that of a uniform allocation. However, this kind of design would need many more hardware resources and power than other efficient designs As the down-converter requires quite a lot of complex multiplications and other operations, and as every channel requires its own down-converter, then as the number of channel K increases, the system complexity greatly increases. In higher sample rate applications, the 'per-channel approach' is not a wise option to

implement channelization, as the current digital signal processer and FPGA cannot provide enough performance for this computational load

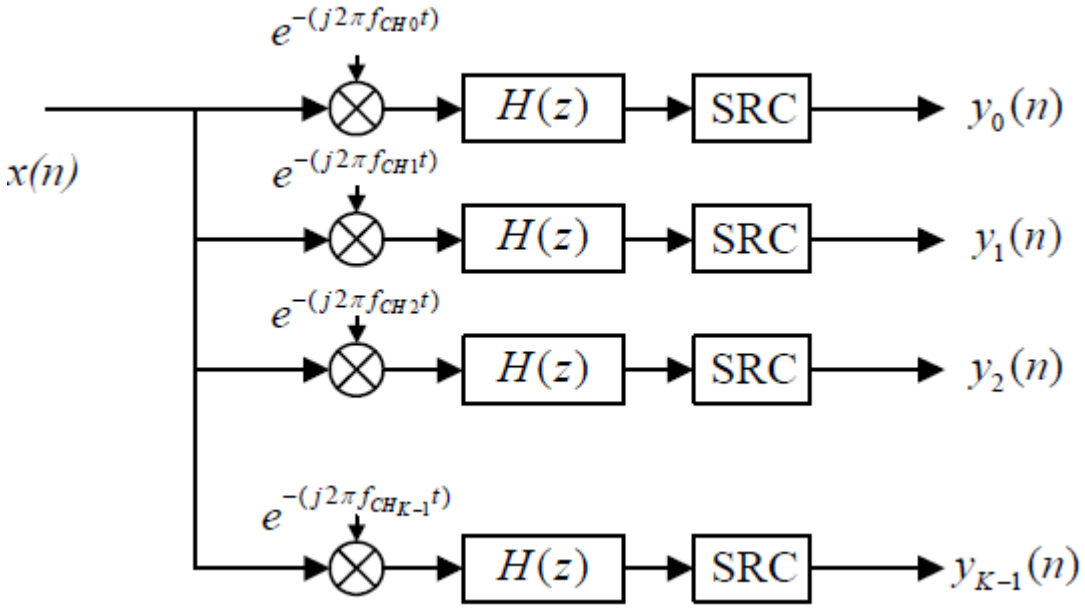


Figure 2.2 ： Channelizer using per channel approach

2.1.2.2 Pipelined frequency transform

Another channelization technology is called pipelined frequency transform. This technology occupies a structure, which contains a binary tree of DDCs (Digital down Converters) followed by a number of SRCs (Sample Rate Converters). Every level of the tree divides the incoming wideband signal into a low frequency half and a high frequency half, and the next level divides these half bands again, until the tree's last level separates out the channels of interest. This structure is also called QMF (Quadrature Mirror Filter) tree . As a result, the system complexity is greatly reduced compared to the per-channel approach, because of the utilization of the half band symmetry and sample rate reduction at each level. The pipeline frequency transform offers a more efficient option in terms of hardware usage and power consumption compared to per-channel approach. This is especially in applications where a large number of channels are needed to be separated from the wideband signal. However, it has weaknesses in terms of flexibility, as all the channels are required to have equal bandwidths and to be uniformly allocated. The diagram of pipeline frequency transform structure is shown in Figure 2.3

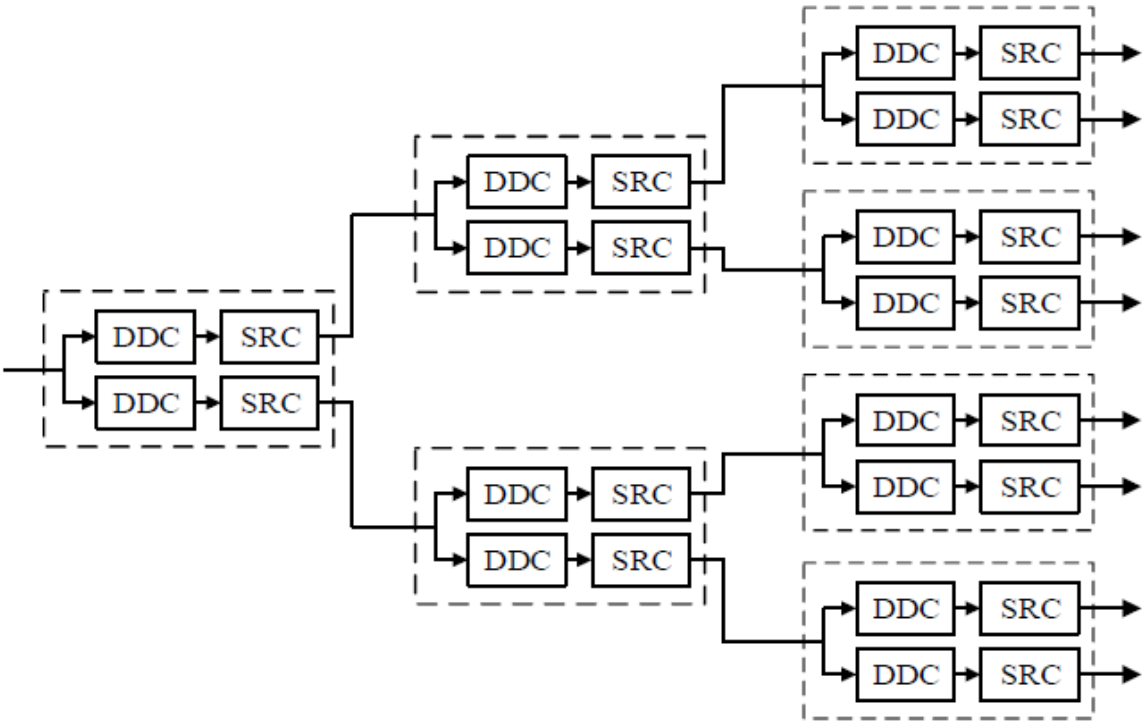


Figure 2.3 :Pipeline frequency transform structure

2.1.2.3 Poly phase filter-bank

Computational efficient channelizers have been designed by using fast Fourier Transform (FFT) The poly phase filter offers further improvement in terms of efficiency. The operational efficiency and design simplicity is obtained from the fact that only one low-pass filter-bank is needed to be designed, and that the remaining band-pass filters will get their properties automatically after the modulation of the prototype filter. An analysis filter-bank will divide the wide-band signal uniformly (in the even stacked allocation), such that every sub-band would have the same space from its adjacent channels in the receiver side. (Even stacked means that there is one sub-band centered at DC .The structure of a poly phase filter-bank is shown in Figure 2.4. The input wideband signal samples will first be down-sampled and sent to poly phase decomposed filters. The filtered samples are then extracted using DFT. This type of channelizer is also referred to as DFT-FB (Discrete Fourier Transform Filter-Bank). DFT-FBs require that extracted channels in the wideband signal have to be uniformly allocated, and the signal sample rate has to be an integer multiple of the sub-channel's bandwidth.

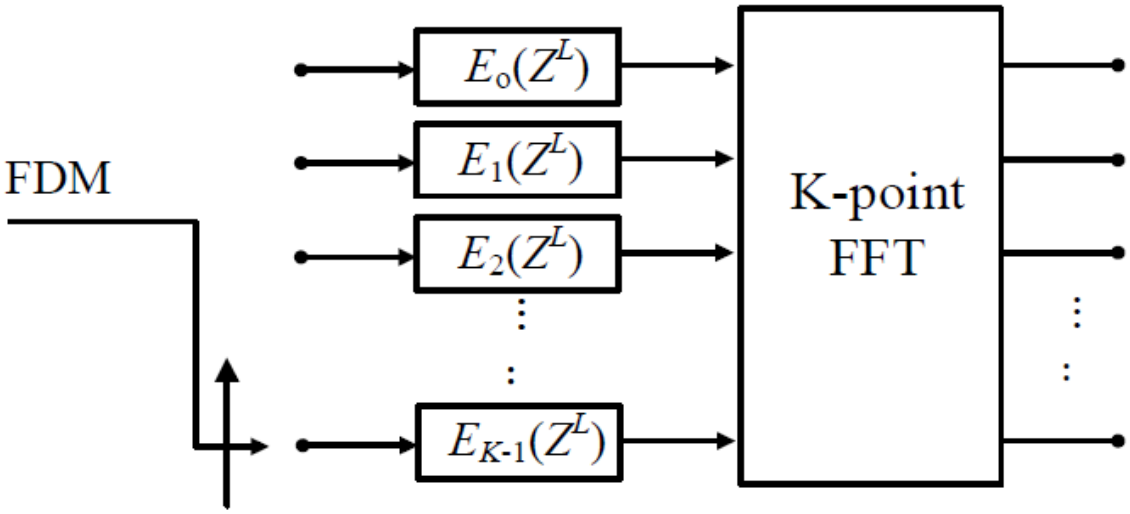
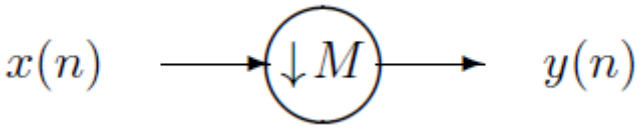


Figure 2.4 : The structure of poly phase filtering

2.1.3 Mathematical background

2.1.3.1 Down sampling (Decimation)

The down sampler is represented by following symbol



It is defined as

y(n) =x(Mn)

in frequency domain ,

$$Y^f(\omega) = \mathrm{DTFT}\left\{[\downarrow M]\, x(n)\right\} = \frac{1}{M}\sum_{k=0}^{M-1} X^f\left(\frac{\omega - 2\pi\, k}{M}\right).$$

If the input signal is a multiple of 2π/M, where M is the decimation rate .This would create an image at 0 Hz (Baseband).There is no need of a Band pass filter to shift the signal to baseband.

Digital frequency is defined as following equation

Øk= 2π(f/Fs)

Where Øk is the digital frequency is the analog frequency ,Fs is the sampling rate which is 1/T .T is the period between two samples.

Then for frequency shifting operation in digital domain variables can be modified like this.

$$g(t) = \exp(j2\pi f_k t)$$
$$g(n) = g(t)_{t=nT} = \exp(j2\pi f_k nT)$$
$$g(n) = \exp\left(j2\pi \frac{f_k}{f_S} n\right) = \exp(j\theta_k n).$$

2.1.3.2 Conventional channelizer

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 .It is shown in Figure 2.5

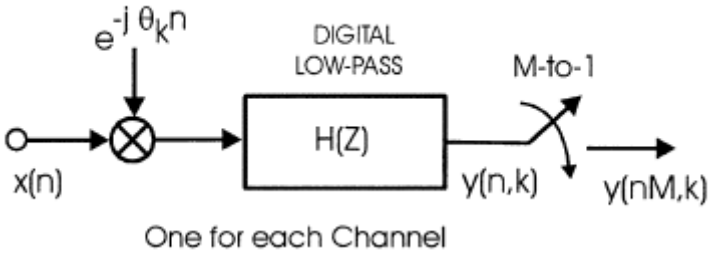


Figure 2.5 : k th channel of conventional channelizer

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

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

We can rearrange the summation 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

band pass filter followed by a down conversion. The block diagram demonstrating this relationship is shown in Figure 2.6

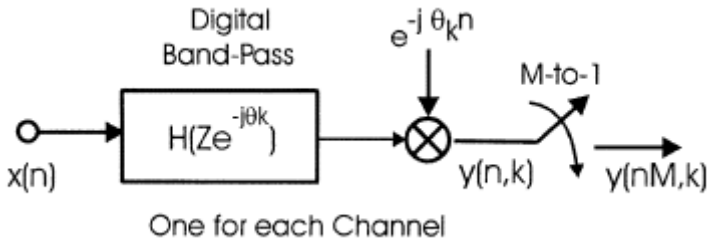


Figure 2.6 : Band pass filter followed by a down converter

when M Øk = k2π ,

Øk (center frequency) will alias to dc as a result of the down sampling to M Øk. Therefore figure 2.6 can be modified as figure 2.7

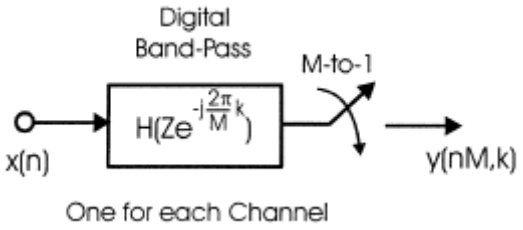


Figure 2.7 :Modified filter

2.1.3.3.The noble identity

The noble identity is compactly presented in figure 2.8, which we describe with similar conciseness by “The output from a filter $H(Z^M)$ followed an M to 1 down sampler is identical to an M-to-1 down sampler followed by the filter .” H(Z)

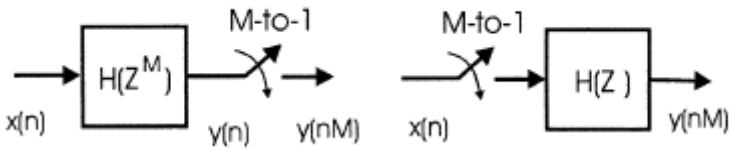


Figure 2.8 Noble identify

2.1.3.4. Poly phase decomposition

The process can be re arranged as shown in following equations

$$H(Z) = \sum_{n=0}^{N-1} h(n)Z^{-n}$$
$$= h(0) + h(1)Z^{-1} + h(2)Z^{-2} + h(3)Z^{-3} + \cdots + h(N-1)Z^{-(N-1)}.$$

Separating the terms as below

$$H(Z) = \begin{array}{ccccccc} h(0) & + & h(M+0)Z^{-M} & + & h(2M+0)Z^{-(2M+0)} & + & \cdots \\ h(1)Z^{-1} & + & h(M+1)Z^{-(M+1)} & + & h(2M+1)Z^{-(2M+1)} & + & \cdots \\ h(2)Z^{-2} & + & h(M+2)Z^{-(M+2)} & + & h(2M+2)Z^{-(2M+2)} & + & \cdots \\ h(3)Z^{-3} & + & h(M+3)Z^{-(M+3)} & + & h(2M+3)Z^{-(2M+3)} & + & \cdots \\ \vdots & & \vdots & & \vdots & & \\ h(M-1)Z^{-(M-1)} & + & h(2M-1)Z^{-(2M-1)} & + & h(3M-1)Z^{-(3M-1)} & + & \cdots \end{array}$$

Each raw of above equation can be described as $Z^{-r}H_r(Z^M)$ so that above equation can be written in a compact form as shown in the following equation .

$$H(Z) = H_0(Z^M) + Z^{-1}H_1(Z^M) + Z^{-2}H_2(Z^M) + \cdots + Z^{-(M-1)}H_{(M-1)}(Z^M).$$

Can be written in traditional summation form as following equation

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

Frequency translation property of Z transforms

If

$$H(Z) = h(0) + h(1)Z^{-1} + h(2)Z^{-2} + \cdots + n(N-1)Z^{-(N-1)} = \sum_{n=0}^{N-1} h(n)Z^{-n}$$

and

$$\begin{aligned} G(Z) &= h(0) + h(1)e^{j\theta}Z^{-1} + h(2)e^{j2\theta}Z^{-2} + \cdots \\ &\quad + h(N-1)e^{j(N-1)\theta}Z^{-(N-1)} \\ &= h(0) + h(1)\left[e^{-j\theta}Z\right]^{-1} + h(2)\left[e^{-j\theta}Z\right]^{-2} + \cdots \\ &\quad + h(N-1)\left[e^{-j\theta}Z\right]^{-(N-1)} \\ &= \sum_{n=0}^{N-1} h(n)\left[e^{-j\theta}Z\right]^{-n} \end{aligned}$$

then

$$G(Z) = H(Z)|_{Z=e^{-j\theta}Z} = H\left(e^{-j\theta}Z\right).$$

By applying above relationship and øk = 2π/M, following equation can be obtained.Block diagram is shown in figure 2.9

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

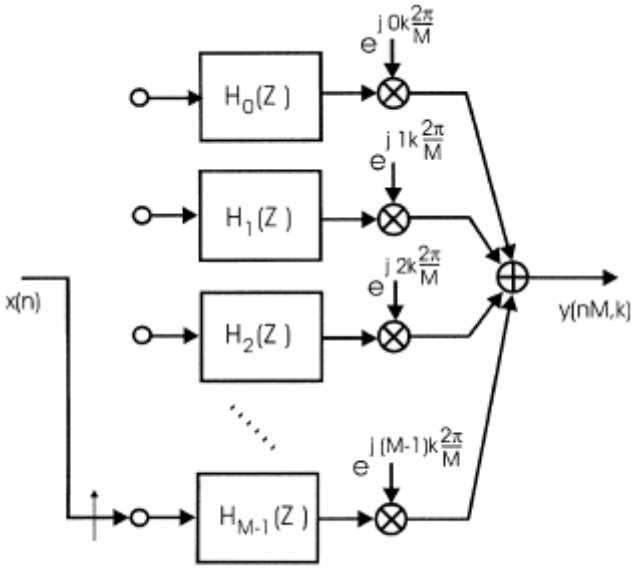


Figure 2.9 :Resampling M-path down converter

The computation of the time series obtained from the output summation

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

By adding a M point FFT to the end all the outputs can be extracted as shown n figure 2.10

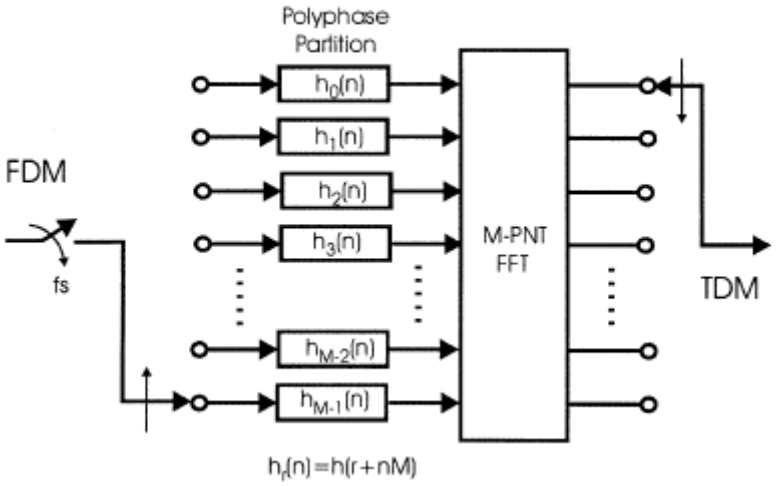


Figure 2.10:Final structure of poly phase channelizer

2.2 FPGA basics

2.2.1 Introduction

FPGA (Field programmable Gate Array) Technology continues to advance since its invention by Xilinx 1984.From basic point of view FPGA is a reprogrammable silicon chip. It is a physical architecture to implement digital logics by using prebuilt logic blocks. The design should be developed in software and then it can be complied to configuration file which says how the components are wired together. FPGAs are programmed and configured using HDL (hardware description language) such as Verilog,VHDL. Unlike Processors, FPGAs have a purely parallel processing architecture which can provide high speed. Because of that FPGAs are widely used in audio processing, medical electronics, Digital signal processing etc.

There are three main components in a FPGA

- Programmable logic blocks
  Provide basic functions and storage resources to the digital system.containsb basic logic gates like AND or OR ,multiplexers,look-up-tables(LUT) and wide fanin AND-OR strucrure

- Programmable interconnections

  The purpose of the programmable interconnection of a FPGA is to make connections among the logic blocks and I/O blocks to match the user defined in the design.

- I/O blocks

  interaction with external components off the FPGA chip through the interface called I/O blocks. The I/O blocks are located around the boundary of the FPGA architecture. They play important roles, and occupy about 40% of the FPGA area

Figure 2.11 shows typical FPGA architecture

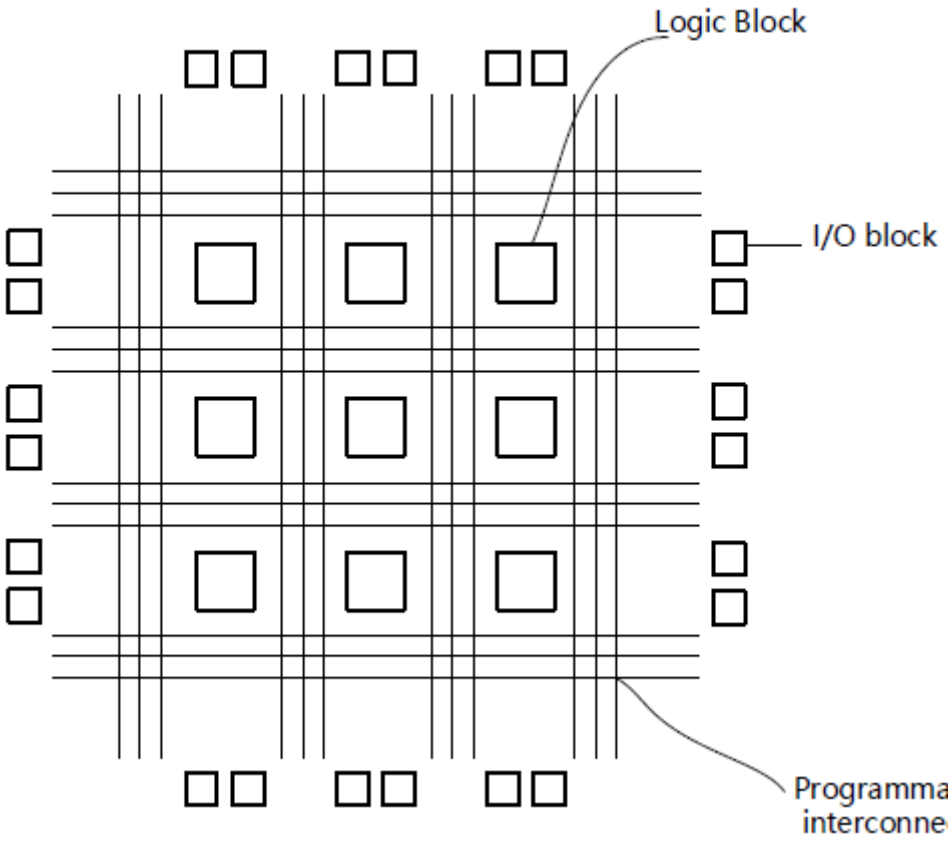


Figure 2.11 : FPGA architecture

2.2.2.VHDL

VHDl is VHSIC Hardware Description Language .VHSIC stands for Very High Speed Integrated Circuits. VHDL can be used to program FPGA. It can also be used as a general purpose parallel programming language.HLS can be used to convert a C code to a VHDL code.Vivado HLS is one software which is used in this project.

2.2.3 Fixed point DSP

Digital signal processing can be separated into two categories – fixed point and floating point These refer to the format used to store the data in the devices. For a common 16 bits fixed point application, there are up to 65,536 possible bit patterns (2^{16}). Signed fixed point value can use two's compliment to make the value include negative numbers

Normally fixed point arithmetic is much faster than floating point in general purpose computers. The internal architecture of the floating point hardware is more complex than the fixed point hardware.

We use fixed point arithmetic in this project.

CHAPTER 3 :HIGH LEVEL DESIGN

3.1 Introdudction

Before going to design the channelizer in hardware description language(HDL).Basic theories were tested using C language and MATLAB.This is not a proper C code which can be directly converted to VHDL using High Level Synthesys (HLS).We generated a QPSK modulated signal in MTLAB and wrote it into a text file to be used as the input signal .It has only one channel in 200 Hz.as shown in figure 3.1

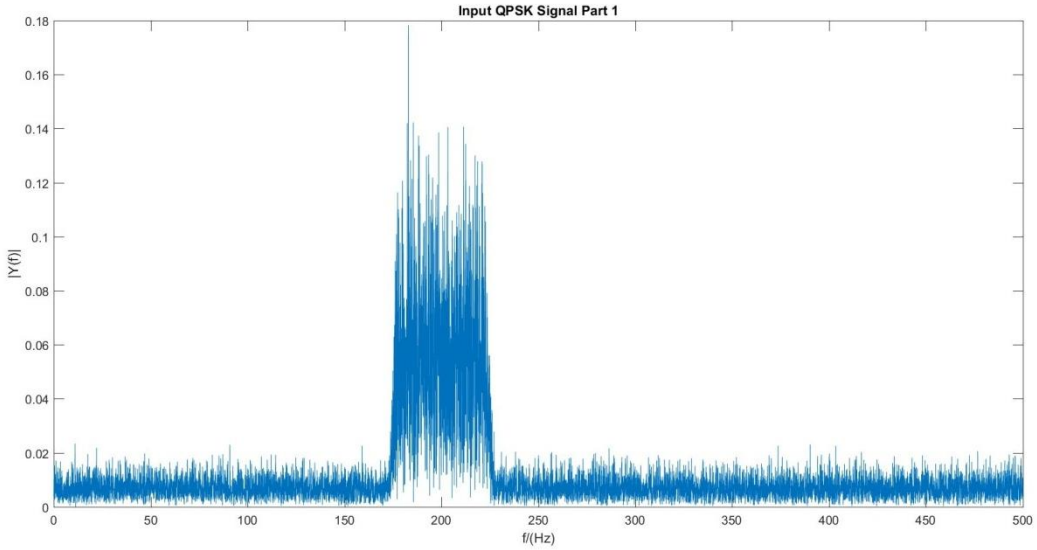


Figure 3.1 :: input signal in frequency domain

3.2 Per channel approach

3.2.1 Frequency shifting and low pass filtering

First the signal was shifted to base band and then filtered and decimated. Filter specification was as follows

Low pass, Equiripple

f_{pass} - 4 Hz

f_{stop} - 5 Hz

Figure 3.2 shows signal after frequency shifting and figure 3.3 shows signal after low pass filtering

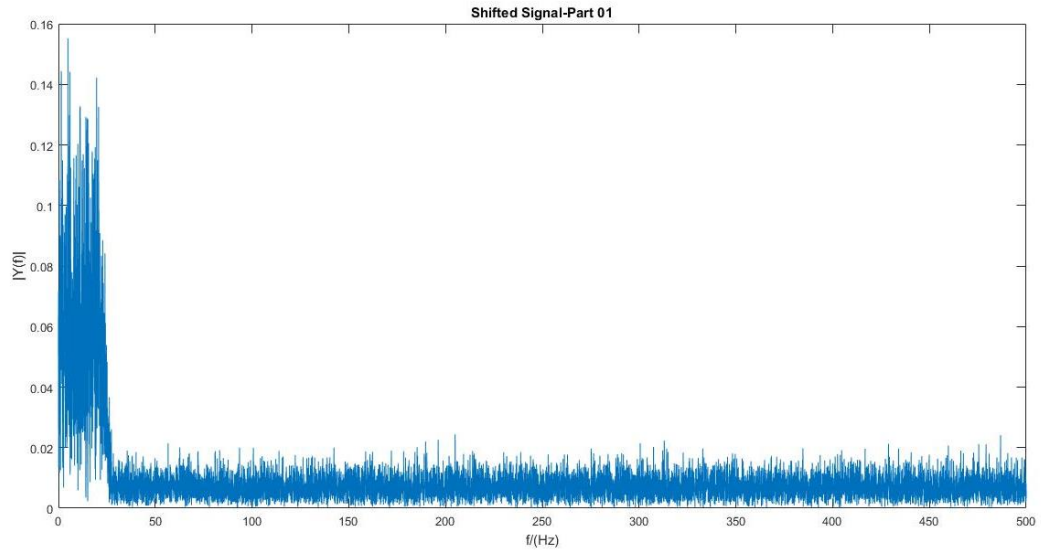


Figure 3.2 : frequency shifted signal

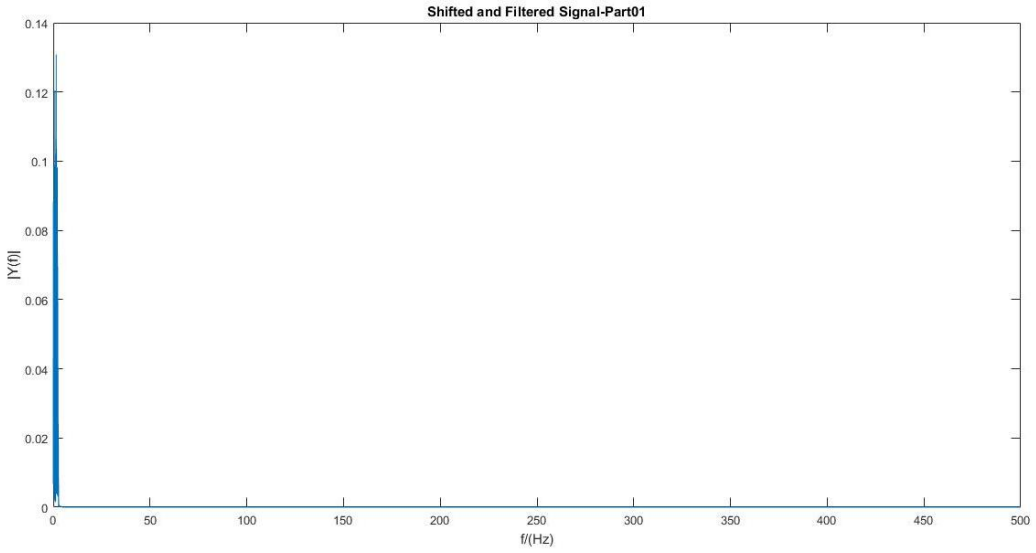


Figure 3.3 : frequency shifted and low pass filtered signal in frequency domain

3.2.2 Band pass filtering and frequency shifting

This part is the same as the previous part but the signal is filtered before it is frequency shifted. Down sampling is done last as the original signal is up-sampled. This part is done to prove the Equivalency theorem. Input signal is same as the previous one .filter specifications is as follows

Band pass, Equiripple

fpass 1 -190Hz

fstop 1 - 195Hz

fpass 2 - 205Hz

fstop 2 – 210Hz

figure 3.4 shows band pass filtered signal and figure 3.5 shows band pass filtered and frequency shifted signal in frequency domain

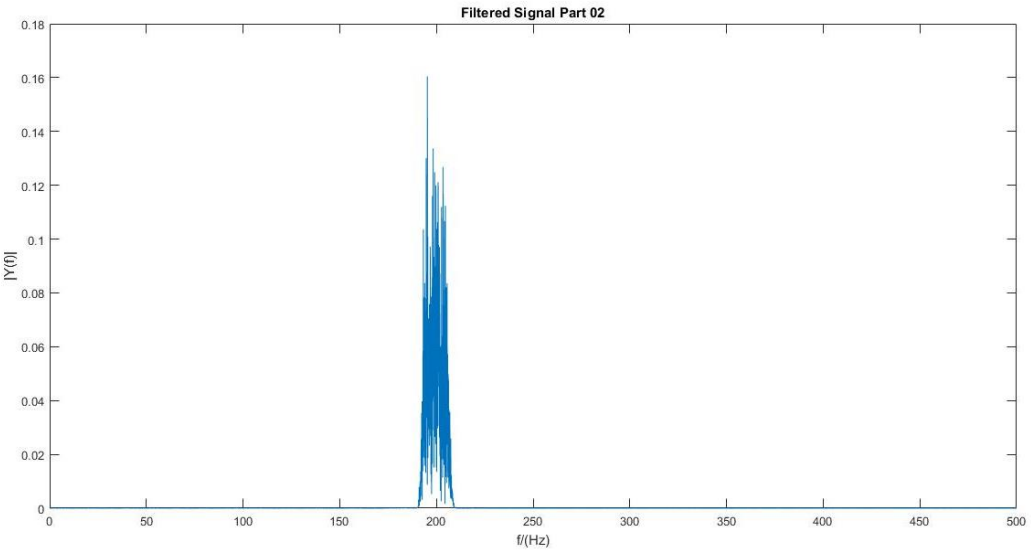


Figure 3.4 : band pass filtered signal

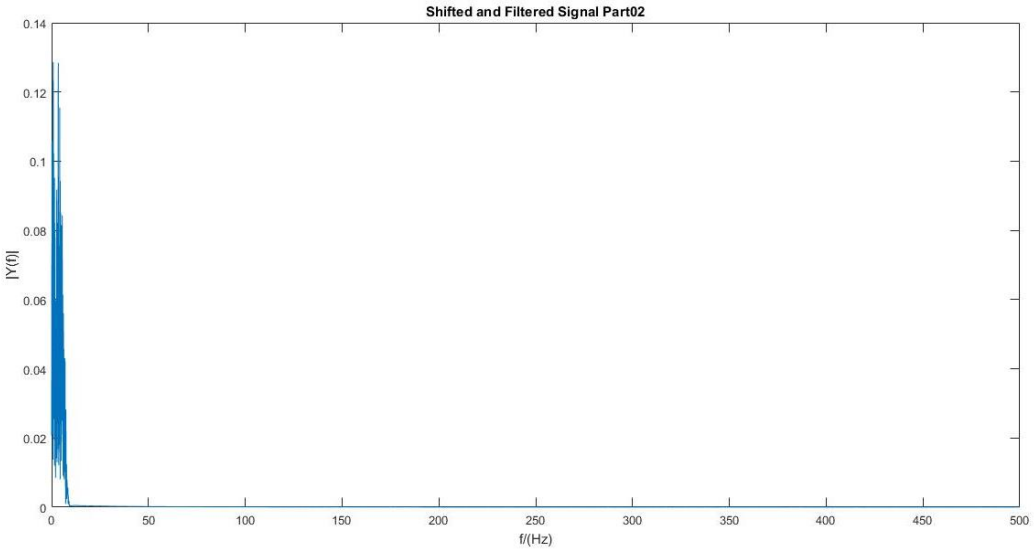


Figure 3.5 band pass filtered and shifted signal

3.3 poly phase filtering

Then poly phase architecture was implemented in C language .Here also the C version cannot be directly used in HLS because we assumed we had the full input signal .No real time operation was assumed. Our decimation rate is 128 therefore there are 128 channels extracted .Then All the output channels were ploted in frequency domain using MATLAB as shown in figure 3.6.But there are only one channel with data.it can be seen in the figure in blue color.

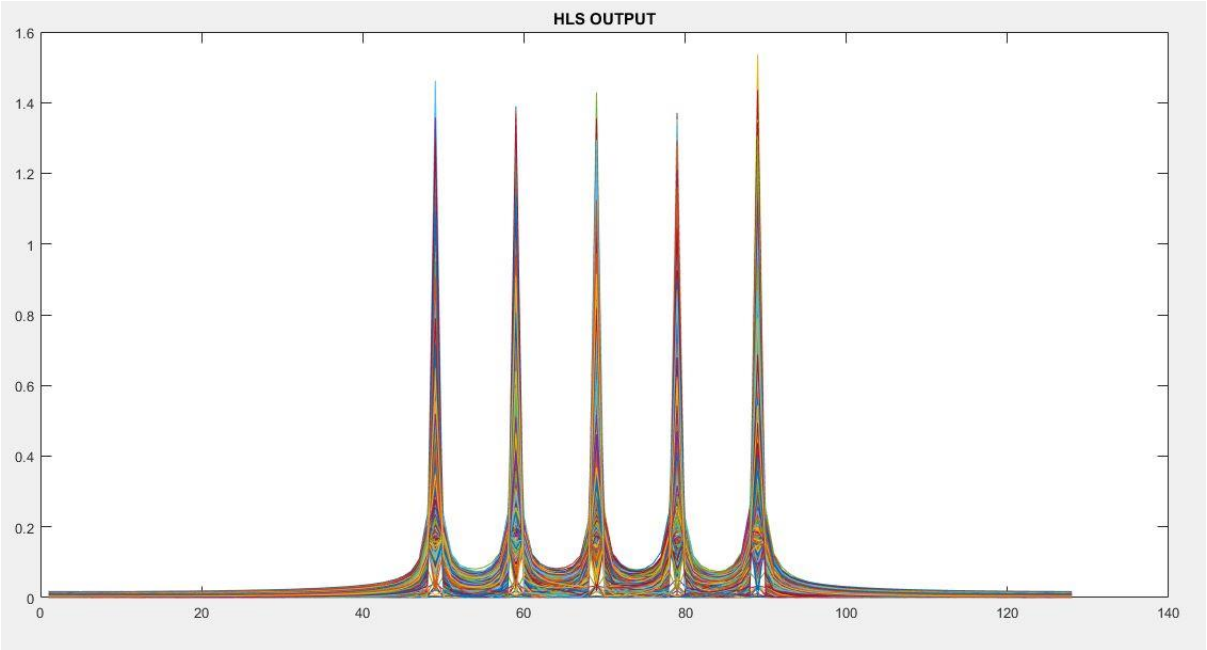


Figure 3.6 output channels in frequency domain separating

CHAPTER 4 : VIVADO HIGH LEVEL SYNTHESIS

4.1 Why High Level Synthesis

- Algorithmic based approaches are getting popular due to accelerate design time and time to market

  -large design pose challenges in design and verification of hardware at HDL level

- Industry trend is moving towards hardware acceleration to enhance performance and productivity

  -CPU-intensive task can be offloaded to hardware accelerator in FPGA hardware accelerator require a lot of time to understand and design

- Vivado HLS tool converts algorithmic description written in C-based design flow into hardware description (RTL)

  -Elevates the abstraction level from RTL to algorithms

- High level synthesis is essential for maintaining design productivity for large designs

4.2 WHAT IS HIGH LEVEL SYNTHESIS

High level synthesis is used create an RTL implementation from C level source code. It extracts control and data flow from the source code and implements the design based on defaults and user applied directives.

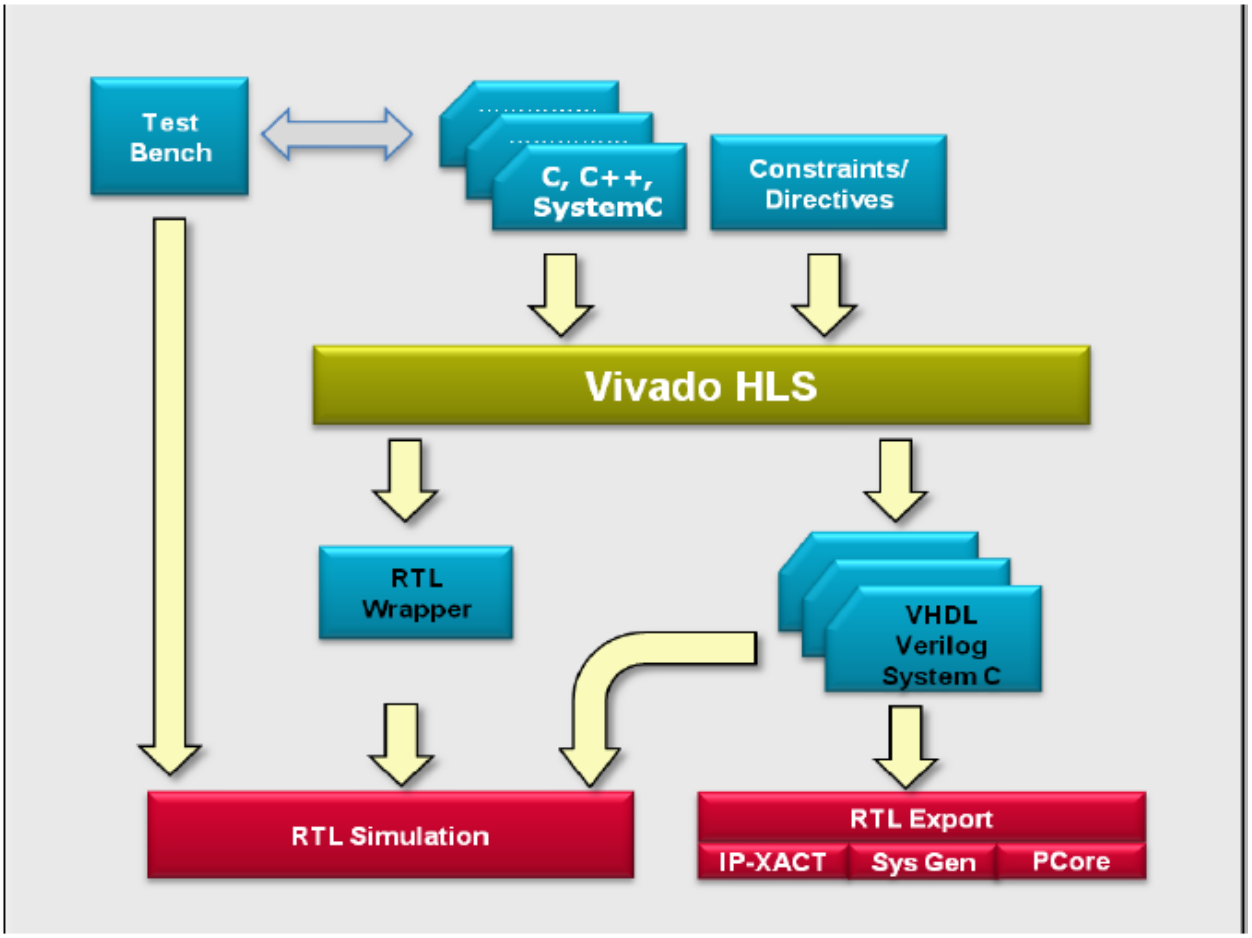


Figure 4.1 High Level Synthesis Use Model

4.3 DESIGN FLOW OF HIGH LEVEL SYNTHESIS

- Step 1 Validate the C code

  The first step in an HLS project is to confirm that the C code is correct. This process is called C Validation or C Simulation

- Step 2 High Level Synthesis

  In this step the C design will be synthesized in to an RTL design

- Step 3 RTL Verification

  RTL design will be verified using C test bench

- Step 4 IP Creation

  The final step in the High Level Synthesis design flow is to package the design as an IP block for use with other tools in vivado design suite

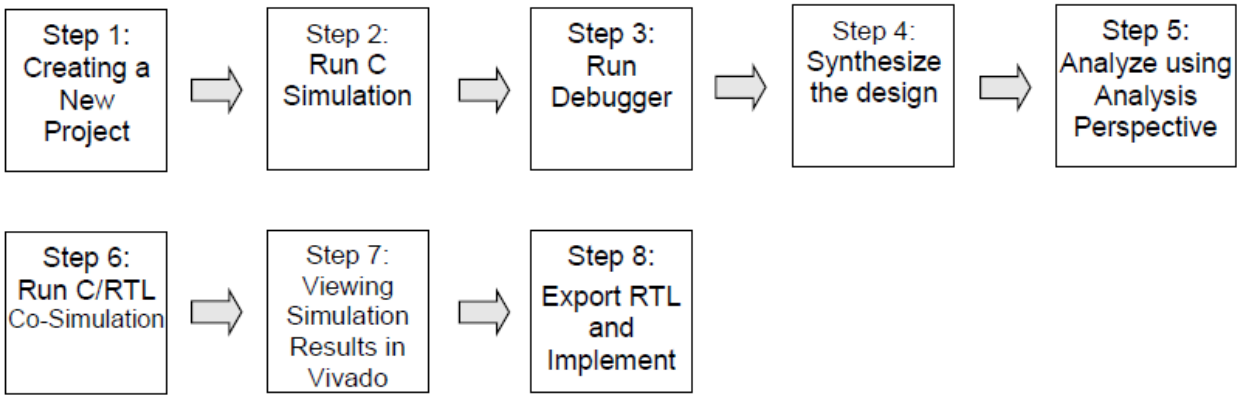


Figure 4.2 Steps of High Level Synthesis

4.4 THE KEY ATTRIBUTES OF C CODE

- Functions

  All code is made up of functions which represent the design hierarchy the same in hardware

- Top level IO

  The arguments of the top level function determine the hardware RTL interface ports.

- Types

  All variables are of a defined type. The can influence the area and performance

- Loops

  Functions typically contain loops. How these are handled can have a major impact on area and performance

- Arrays

  Arrays are used often in C code. They can influence the device IO and become performance bottlenecks

- Operators

  Operators in the C code may require sharing to control area or specific hardware implementations to meet performance

4.5 BENEFITS OF USING FIXED POINT NUMBERS

With almost all designs today, minimizing power consumption is a high priority. Most applications must meet aggressive power and thermal envelopes before they can be deployed to production. It is widely accepted that designing in floating point leads to higher power usage for the design compared to lower precisions. This remains true for FPGAs where floating-point DSP blocks have been hardened in the FPGA, and also where customers must implement a soft solution using provided DSP resources and additional FPGA resources. Floating-point implementations require larger amounts of FPGA resources than an equivalent fixed-point solution. With this higher resource usage comes higher power consumption and ultimately increased overall cost of implementing a design.

Converting a floating-point design to fixed point can help meet these challenging specifications in the following ways:

- Reduction in FPGA resources

  Fewer DSP48E2s, look-up tables (LUTs), and flip-flops are needed when working with fixed-point data types.

  Smaller amounts of memory are required to store fixed-point numbers.

- Lower power consumption

  Reduction in FPGA resource usage inherently leads to lower power consumption.

- Reduced BOM cost

  Designers can utilize the additional available resources for extra features in their application for the same cost.

  The resource savings enable massively increased compute capabilities within the FPGA. This increased compute power benefits many applications, e.g., Machine Learning DNNs.

  It is possible the resource savings can reduce the size of the device needed for the design.

- Latency improvements

Again, reducing the resources, in particular the DSP48E2 slices, when implementing the FIR brings a latency improvement in the fixed-point design.

- Comparable performance and accuracy

  For designs and applications that do not require the dynamic range achievable with floating point, fixed-point implementations can provide comparable results and accuracy. In some cases, the results can even be improved.

Table 4.1 Comparison between fixed point and floating point

Fixed point	Floating Point
Very fast when based 2	Slower
No complicated logic	Accuracy varies
Radix point not encoded	Represent very large number set
Fixed accuracy	Radix point encoded
Can only represent small numbers	Complex logic required

4.6 HARDWARE ALLOCATION COMPARISON

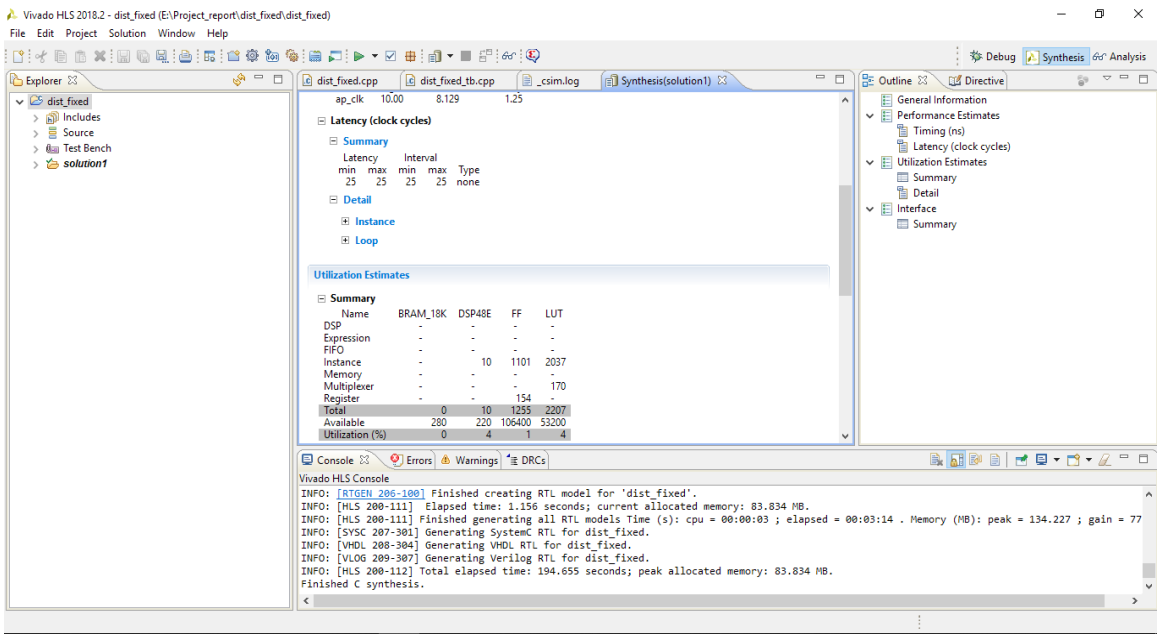


Figure 4.3 Hardware utilization estimation of Floating point design

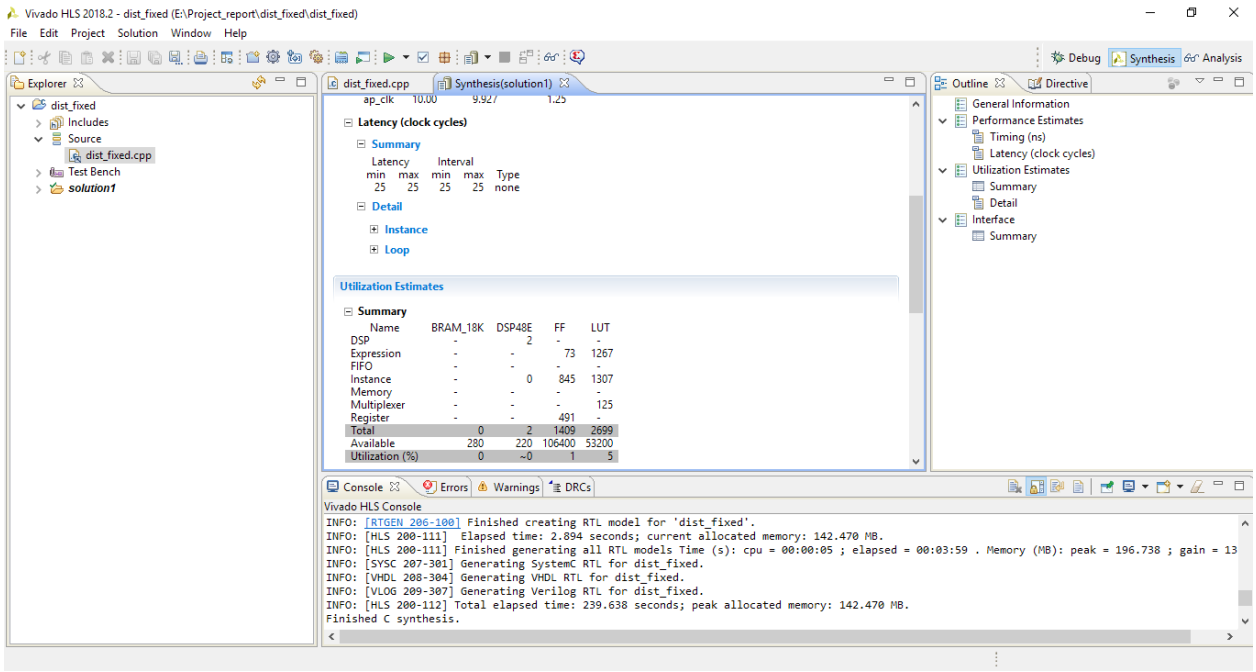


Figure 4.5 Hardware utilization estimation of Fixed point design

Table 4.2 Hardware Utilization Comparison

	BRAM_18K	DSP48E	FF	LUT
Fixed point	0	2	1409	2699
Floating point	0	10	1255	2207

4.7 Design analysis

The general design methodology for creating an RTL implementation from C includes the following tasks

- Synthesizing the design.
- Reviewing the results of the initial implementation.
- Applying optimization directives to improve performance.

A key part of this process is the analysis of the results

Vivado HLS generates the Synthesis report which can be used to observe the performance of the design. Following are the parameters found in the Synthesis report.

- The timing summary shows the target and estimated clock frequency. If the estimated clock frequency is greater than the target, the hardware will not function at this clock frequency. The clock frequency should be reduced by using the Data motion Network Clock Frequency option in the Project settings.Alternatively because this is only an estimate at this point in the flow .it might be possible to proceed through the remainder of the flow if the estimate only exceeds the target by 20%. Further optimizations are applied when the bitstream is generated, and it might still be possible to satisfy the timing requirements. However, this is an indication that the hardware function is not guaranteed to meet

- The initiation interval (II) is the number of clock cycles before the function can accept new inputs and is generally the most critical performance metric in any

system. In an ideal hardware function,the hardware processes data at the rate of one sample per clock cycle. If the largest data set passed into the hardware is size N (e.g., my_array[N]), the most optimal is N + 1. This means the hardware function processes N data samples in N clock cycles and can accept new data one clock cycle after N samples are processed. It is possible to create a hardware function with an II < N, however, this requires greater resources in the PL with typically little benefit.The hardware function will often be ideal as it consumes and produces data at a rate faster than the rest of the system.

- The initiation interval is the number of clock cycles before the next iterations of a loop starts to process data. This metric becomes important as you delve deeper into the analysis to locate and remove performance bottlenecks.

- The latency is the number of clock cycles required for the function to compute all output values. This is simply the lag from when data is applied until when it is ready. For most applications this is of little concern,

- The iteration latency is the number of clock cycles it takes to complete one iteration of a loop, and the loop latency is the number of cycles to execute all iterations of the loop

Resource Utilization

The Area estimation section of the report details how many resources are required in the PL to implement the hardware function and how many are available on the device. The key metric here is the utilization (%). The utilization (%) should not exceed 100% for any of the resources. A figure greater than 100% means there are not enough resources to implement the hardware function and a larger FPGA device might be required. As with the timing at this point in the flow this is an estimate.If the numbers are only slightly over 100%, it might be possible for the hardware to be or๏mbz;7 during bitstream creation.

4.8 OPTIMIZATIONS IN VIVADO HLS

In order to generate optimized RTL code, Vivado HLS uses a number of optimizations such as dataflow pipelining, function pipelining, resource limitation, loop unrolling, memory partition etc. These optimizations are specified as directives using tcl scripts, or can be embedded in the source code.

4.8.1 Pipelining in Vivado HLS

Pipelining in Vivado HLS can be applied as an optimization between functions or the operations within a function. It can also be applied to the operations inside a loop or between loops. In each case, this optimization increases throughput, by ensuring that the function, loop or operation is not required to wait until a previous function, loop or operation has completed all its operations before it can begin. When the pipelining is applied as an optimization between functions, it's referred to as dataflow pipelining. Dataflow pipelining transforms a sequential implementation of functions into a parallel architecture as shown in the figures 4.1 and 4.2.

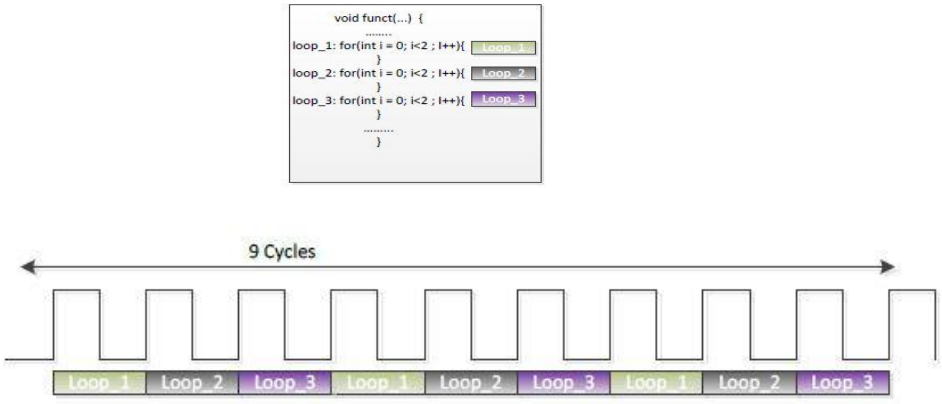


Figure 4.1 Loop without Pipeline

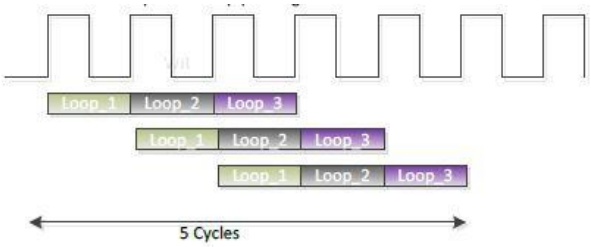


Figure 4.2 With Loop without Pipeline

4.8.2 Array partitioning in Vivado HLS

This pragma control how large arrays are partitioned into multiple smaller arrays, to reduce RAM access bottleneck. Also used to ensure arrays are implemented as registers and not RAMS. .

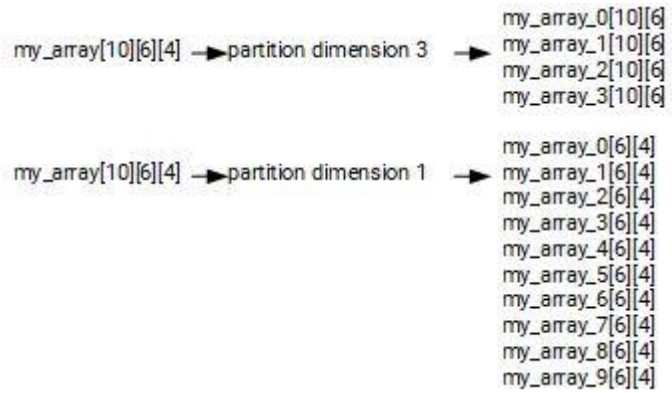


Figure 4.3 Array Partitioning in different dimensions

4.3.3 Optimization results of final design

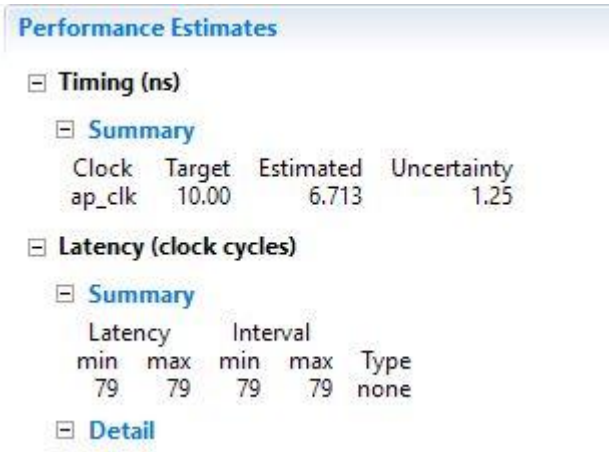


Figure 4.4 Performance estimation Without pipeline

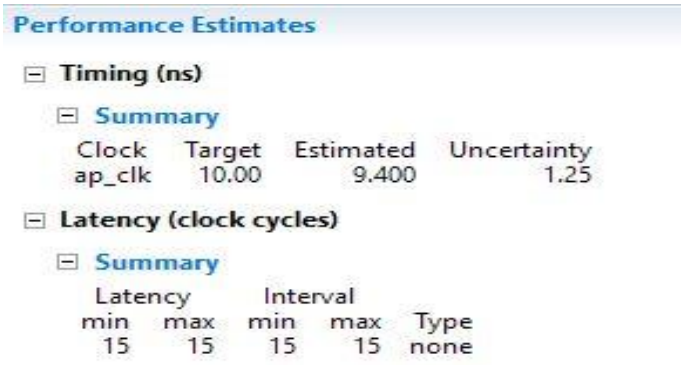


Figure 4.5 Performance estimation With pipeline

CHAPTER 5 : CONCLUTION AND FUTURE WORK

5.1 Future Work

The focus of this project was to implement efficient and uniform channelizers based on FPGAs. However, there is still room for improvements. Some possible options of the future work are:

- Further efficiency of the FPGA uniform filter bank can be obtained from replacing FIR filter by IIR filter with an approximately linear phase response.

- Use multiple FIR compiler core combination to achieve 256 channel or even higher channel number of uniform channelizer, because FIR compiler has a limit of 128 channels in maximum for polyphase filter-bank implementation.

- Design a complex FIR core that can process input samples with complex coefficients. It could greatly simplify the design process for odd stacked polyphase filters.

5.2 Conclution

Polyphase filter-bank plays a signification role in the DSP system. It has been widely studied during last 20 years. At first we introduce the background of DSP theories required for channelization and basics of FPGA after that we have presented the results that we have obtain from the C language code and matlab code. Then we moved into RTL design and for that we started to work on VIVADO HLS and the obtained result are presented. Our next step will be developing the complete C code and test bench which can be directly converted in to VHDL code and then it will be implemented in Xillinx FPGA plat form and It will be verified in hardware level.

REFERENCE

[1] F. J. Harris, C. Dick, and M. Rice, "Digital receivers and transmitters using polyphase filter banks for wireless communications," Microwave Theory and Techniques, IEEE Transactions on, vol. 51, pp. 1395-1412, 2003.

[2] Awan M, Yannic Le M, Peter K & Fred H, “Polyphase Filter Banks For Embedded Sample Rate Changes In Digital Radio Front-Ends”, *ZTE Communications* Vol 9, No 4 (2011): pp. 3 - 9.

[3] Lianping G, Shuling T & Zhigang W, “Research on broadband signal spectrum measurement technique based on channelization”, Journal of Scientific & Industrial Research Vol 72, November 2013, pp. 673 - 680.

[4] F. Wu, N. Palomo, and R. Villing, "FPGA realization of GDFT-FB based channelizers," in Signals and Systems Conference (ISSC), 2015 26th Irish, 2015, pp. 1-6.

[5] P. K. Devi and R. S. Bhuvaneshwaran, "FPGA implementation of coefficient decimated polyphase filter bank structure for multistandard communication receiver," Journal of Theoretical and Applied Information Technology, vol. 64, 2014.

[6] J. Lillington, "Comparison of Wideband Channelization Architectures," in International signal processing conference, Dallas, 2003.