Design and Implementation of Polyphase Fast Fourier Transform Channelizer

Ali Ghandour[1*]
[1]National Council for Scientific Research
Beirut, Lebanon
aghandour@cnrs.edu.lb

Ahmad Mansour[2*]
[2]Computer and Communications Engineering Department
Islamic University of Lebanon, Khaldeh, Lebanon
ahmad.mansour.00961@gmail.com

Hussein AlAsadi[3]
[3]Advanced Research, Engineering & Consulting Services
Munich, Germany
alasadi@arecs.eu

Walid Ghandour[4]
[4]Lebanese University
Beirut, Lebanon
ghandour@utexas.edu

***Abstract*—Modern wideband receivers must be able to slice and dice a large band of the spectrum in real time, while available receivers are increasingly faced with the challenges of providing flexible operations on large number of channels simultaneously. We present in this paper a comprehensive review of major channelizer algorithms available in the literature that are compatible with Software Defined Radio platforms. We employ analytical models to evaluate and compare various channelizer algorithms. Moreover, we implement and simulate the Polyphase Fast Fourier Transform (PFFT) channelizer using MATLAB software. We show using the reported results that PFFT computation complexity is about four times lower than any other channelizer. We also contribute the first hardware implementation of PFFT using Vivado High Level Synthesis. In addition, we provide a clear and practical manipulation of the PFFT structure to enable real-time streaming support taking into account system optimization in terms of hardware resources and latency.**

***Index Terms*— Polyphase Fast Fourier Transform Channelizer; Software Defined Radio; High Level Synthesis.**

I. INTRODUCTION

Channelizer is the backbone of wideband receivers that effectively support recent communication systems and real time applications. Several channelizer algorithms are suggested in the literature, such as the traditional approach known as Per Channel (PC) channelizer which down converts and filters each channel independently. PC can be viewed as a stack of traditional single channel channelizer [1] [2]. Pipelined Frequency Transform (PFT) channelizer suggested in [3] [4] is a binary tree of PC channelizer that distributes the computation load into halves while it is growing. Tunable PFT (TPFT) [3] [5] is an extension of PFT with additional interleaver and tuning processes. Frequency Domain Filtering (FDF) channelizer employs large Fast Fourier Transform (FFT) block for down-conversion and decimation operations, followed by frequency domain filtering and then, Inverse Fast Fourier Transform (IFFT) blocks are employed at baseband [6] [1]. Finally, the Polyphase Fast Fourier Transform (PFFT) channelizer [2] [7], which constitutes the main scope of this work, is a combination of Polyphase Filter Bank (PFB) and Fast Fourier Transform (FFT).

The contribution of this paper is three-fold: *(i)* discusses and compares in details the above-mentioned channelizer algorithms. The main metrics considered are computation complexity and flexibility which have not been addressed together in any previous study as far as the authors know. Computation complexity is a relevant metric because it got translated into latency and hardware resources requirements. *(ii)* Moreover, the PFFT channelizer is further investigated through a MATLAB simulation study. *(iii)* Finally, we propose our own PFFT hardware implementation which is suitable for real time applications and very useful for designers using off-the-shelf Fast Fourier Transform (FFT) core. We used Xilinx Vivado High Level Synthesis (HLS) software. HLS approach helps to abstractly and quickly design and optimize the desired digital system under the concept of try and test.

The rest of the paper is organized as follows. In Section 2, we amply elucidate major channelizer algorithms available in the literature and compare them in terms of computation complexity and channelizing flexibility. Section 4 presents the findings of our MATLAB simulation of the PFFT channelizer. In Section 5, we discourse on our PFFT channelizer implementation which is based on a novel manipulation in PFFT structure in order to support real time running feature and benefit from off-the-shelf FFT cores. Section 6 concludes this manuscript.

[*] Both authors contributed equally to this work.

978-1-7281-3129-0/20/$31.00 ©2020 IEEE

Table 1. Formulation of the number of operations required by channelizer algorithms.

Needed Operations	Per-Channel	Pipelined FT	FDF	Polyphase FFT
Complex Multiplications	$\mathrm{fs} * \mathrm{M}$	$2^i * \mathrm{Bw} * \mathrm{i}$	$4\mathrm{fs} * [\log_2(16\,\mathrm{M}) + 4]$	$\frac{\mathrm{fs}}{\mathrm{M}} * M * \log_2 \mathrm{M}$
Real Multiplications	$\mathrm{fs} * \mathrm{M} * 2\mathrm{P}$	$2^i * \mathrm{Bw} * \mathrm{i} * 2\mathrm{P}$	0	$\frac{\mathrm{fs}}{\mathrm{M}} * \mathrm{M} * 2\mathrm{P}$
Real Additions	$\mathrm{fs} * \mathrm{M} * 2(\mathrm{P}-1)$	$2^i * \mathrm{Bw} * \mathrm{i} * 2(\mathrm{P}-1)$	$2(4\mathrm{fs} * [\log_2(16\,\mathrm{M}) + 4])$	$\frac{\mathrm{fs}}{\mathrm{M}} * \begin{bmatrix} 2\mathrm{M}\log_2 \mathrm{M} + \\ \mathrm{M} * 2(\mathrm{P}-1) \end{bmatrix}$

II. DIGITAL CHANNELIZER

Channelization is the digital front end operation that down-converts and filtrates channels of interest of received Frequency Division Multiplexing (FDM) signals to zero frequency for further digital signal processing. Channelizer algorithm is usually a multi-stages process.

Per-Channel (PC) channelizer is the basic and most direct channelizer approach. It is a stack of the conventional single channel channelizer, where θ_i and b_i are determined according to the desired channels' parameters [2]. PC is the most flexible and selective channelizer for any bandwidth at any frequency. In addition, it is easily reconfigurable since the channels are treated independently. However, for a large number of channels, PC channelizer is overloaded by intensive computations which are not feasible to implement especially in real time applications, where each Digital Down Converter (DDC) is flooded at the Analog to Digital Converter (ADC) highest data rate by undesired samples that are discarded later-on by the down-sampler [1].

Pipelined Frequency Transform (PFT) channelizer is a binary tree of DDCs and down-samplers. The root node is fed at the highest sampling frequency (fs). Each node consists of a DDC and a down-sampler that is presented by an input commutator that reduces the input symbol rate to its half. Each parent node is followed by two children nodes operating at half of their parent's rate. The binary tree grows continuously till the rate reaches the defined "bandlimit" according to a desired channels' bandwidth. At leaves' depth, narrowband channels are at zero frequency [4]. PFT channelizer significantly overcomes the computation complexity of PC channelizer by benefiting from the advantage of rate bisections. However, it lacks the needed flexibility since it exactly recovers M *channels,* where $M = 2^i$ and $i \in \mathbb{N}$, under two strict conditions: *(i)* uniform and continuous distribution of channels and *(ii)* bandwidth unification.

Frequency Domain Filtering (FDF) channelizer is an exploitation of FFT in order to simplify the channelizing operation. In this channelizer, the input samples of the wideband signal that were overlapped in the overlapping buffer are converted to frequency domain using FFT. FFT bins of relevant channels are then extracted. This operation effectively down-converts and decimates channels to form the baseband bins that will be multiplied with the frequency response coefficients $H(k)$ of the baseband filters. Filters' outputs are converted back to time domain using IFFT [1].

Polyphase Fast Fourier Transform (PFFT) channelizer [2] is a combination of Polyphase Filter Bank (PFB) [8] and FFT. This combination benefits from the automatic down-sampling of PFB that is enforced through equivalency theorem and noble identity. Also, it is an exploitation of the speed advantage of FFT that rapidly applies intensive mathematical calculations with a complex sinusoid. PFFT has three imposed restrictions: *(i)* the bandwidth must be unified within channels, *(ii)* the carriers have to be uniformly and continually distributed, where all carriers must be multiples of each other ($fc[\![n+1]\!] = \mathbb{Z} * fc[\![n]\!]$) and *(iii)* all channels within the occupied band must be recovered though some of them might be out of interest. In addition, it is evident that FFT requires the number of channels to be a power of two in order to maximally optimize the system.

After the detailed review of the various channelizer algorithms, we are going to study the differences between them to decide about the best channelizer. We focus in our comparison on the two main metrics: *(i)* computation complexity which is defined here as the number of operations required by a channelizer in order to recover *M* channels and *(ii)* channelizing flexibility, namely channel distribution and bandwidth.

2.1 Computation Complexity

In Table 1, we model the number of operations required by a channelizer per 1 second. For this model, we assume an input wideband signal that consists of M narrowband FDM channels of bandwidth $Bw = 20kHz$ each, sampled at rate $fs = Bw * M$. The following assumptions were made too: *(i)* $i = ceil(\log_2 M / \log_2 2)$ and *(ii)* P is the number of filter taps per one channel. FFTs were substituted by $(n \log_2 n)$ complex multiplications and $(n \log_2 n)$ complex additions corresponding to the overall improvement of radix-2 FFT [9]. In FDF, 16 FFT bins were dedicated per one channel that correspond to a resolution of 1.25 kHz and 50% overlapping was assumed [6].

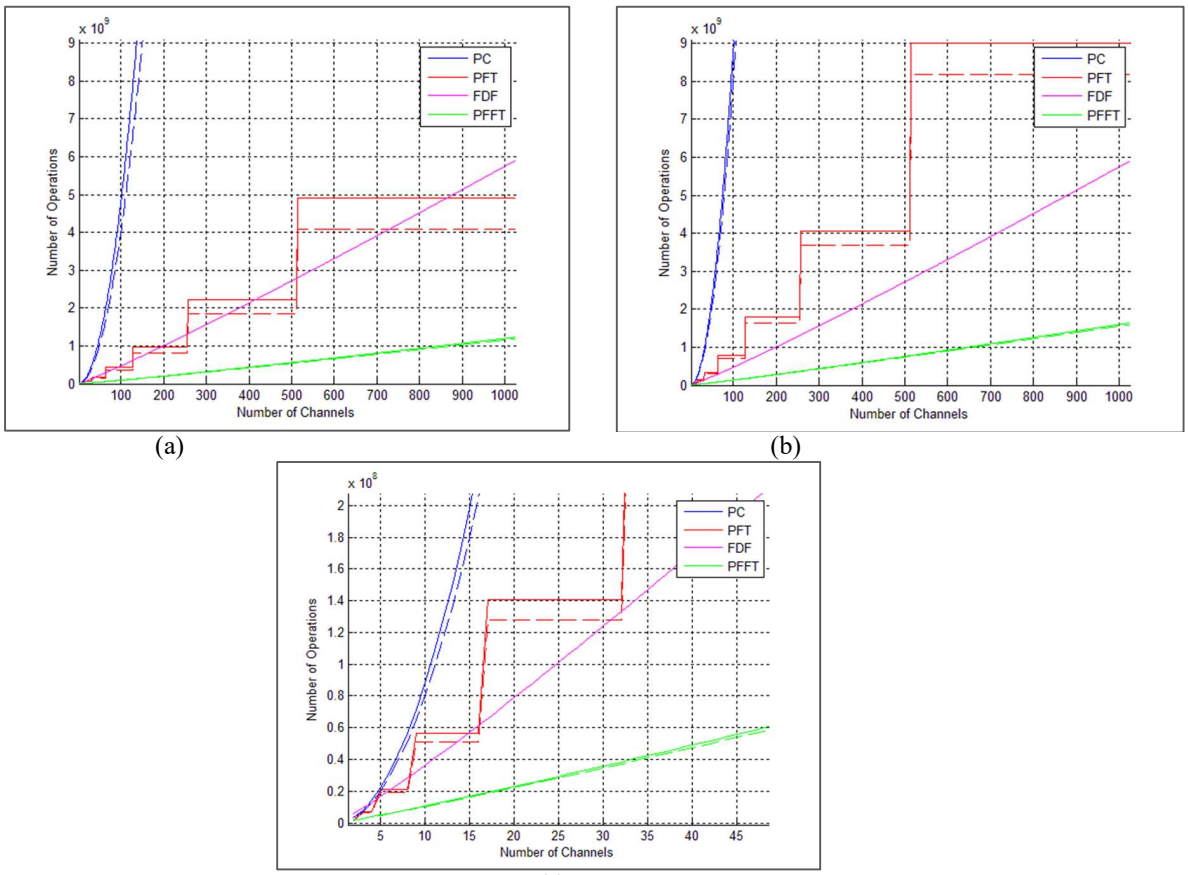

Figure 1. Number of operations using continuous and discontinuous curves required by PC, PFT, FDF, and PFFT channelizers using $\boldsymbol{P = 10}, \boldsymbol{P = 20}$ and $\boldsymbol{P = 20}$ filter taps are show in $\boldsymbol{Figure\ 11(a)}, \boldsymbol{Figure\ 11(b)\ and\ Figure\ 11(c)}$ respectively.

Table 2. ChansRatio metric for four different combinations of channelizers, categorized in terms of: Real Additions and Real Multiplications using *P = 10* and *P = 20*.

		ChansRatio(PC, PFFT)	*ChansRatio(PFT, PFFT)*	*ChansRatio(FDF, PFFT)*	*ChansRatio(PFT, FDF)*
P = 10	Real Additions	186.7282	3.1746	5.0871	0.6203
	Real Multiplications	216.1780	3.6628	4.8964	0.7443
P = 20	Real Additions	273.6124	4.5384	3.6656	1.2405
	Real Multiplications	293.1427	4.8542	3.5664	1.3646

In Figure 1, we plot the formulations of Table 1 using continuous and discontinuous curves representing real multiplications and real additions respectively, where each complex multiplication was substituted by four real multiplications and two real additions. In Table 2, we summarize the findings of Figure 1 where *ChansRatio(X,Y),* $\sum_M \frac{\text{X operations}}{\text{Y operations}}$ is defined as the summation of the ratio of the number of operations required by a channelizer X and the one required by a channelizer Y, over 1023 distinct values of M (varying from 2 to 1024). The resultant summation metric is normalized by the number of samples. For PFT channelizer, only values of $M = 2^i$ were considered and the ratio at each iteration was multiplied by the value of M as a weighting index. It is worthy to note that PFT curves in Figure 1 appear much higher than expected, due to the limitation discussed earlier about PFT always recovering 2^i channels. Hence, for a fair computational complexity study, we recommend to study PFT at $M = 2^i$.

Table 3. PFFTSim parameters of Scenario 1.

PFFT structure	Standard	Number of channels	8	Filter order	160
Channels' support	Parallel	Bitrate	90kHz	filter taps per channel	20
Channel up convertor	complex	Channel bandwidth	100kHz	Channel of carriers -300 & 200 kHz	Unused
Digital Modulator	BPSK	Sampling frequency	800kHz	Shift the channel of carrier 100kHz by	4kHz
Generate testbench files	No	Simulation time	100ms	Recover the shifted channel	No

Results show that PFFT channelizer ranks first (least complex) in terms of computation complexity since it offers about $4x$ reduction in computation complexity compared to PFT and FDF and more than $200x$ with respect to PC. The second channelizer in terms of computation complexity is dependent upon the filter order where in the cases of $P = 10$ and $P = 20$, PFT seems to require about seven tenths (0.7) and thirteen tenths (0.13) the number of operations needed by FDF respectively, as shown in Table 2. Note that, as shown in Figure 1.a, PFT reports higher efficiency as the number of channels decreases. In Figure 1.c, we focus on small values of *M* where $P = 20$. We notice that $M = 32$ is approximately the break-even point for FDF and PFT. Thus, FDF seems to be more efficient with large number of channels, especially if a filter of high order is used in PFT. Finally, PC is by far the most computationally complex channelizer.

Despite this accurate evaluation, it is worth to recall that computation complexity translates into both latency and hardware resources requirements. This means that an algorithm may be more computationally intensive but faster, while another requires less computation but slower. For instance, PFT will be fast if a hardware block is dedicated for each stage, FDF is usually limited by a large FFT block, and PFB can be made faster utilizing more hardware resources if all banks are simultaneously operated as shown hereafter.

2.2 Channelizing Flexibility

PC is the most flexible channelizer in terms of the number, the bandwidth and the distribution plan of channels. It is particularly constructed for the desired plan while other flexible algorithms, made of rigid structures, use tuners and additional processing to offer flexibility. PC supports channelizing operation for frequency hopping system, as shown in [10] where an FPGA implementation is presented. TPFT channelizer follows it by the intermediate channels that translates into flexibility in the number of channels and the bandwidth and by the fine and course tuning which is high flexibility in the distribution plan. Though, the additional intermediate channels support selectivity, TPFT at each stage should operate on 2^i uniformly and continually distributed channels. FDF channelizer allows flexibility in terms of channels' distribution applying the additional processes where they are tuned to zero frequency in a post process. So, this will not totally eliminate the limitation of FFT bins spacing over channel distribution. FDF is also a selective channelizer where channels of interest are extracted out of the frequency components of FFT bins. Regarding PFT and PFFT, they are efficient with uniformly and continually distributed channels. Two main differences can be noted in terms of flexibility: *(i)* Channels' number have to be power of 2 in PFT, while this constraint is relaxed in PFFT and *(ii)* intermediate outputs extraction of wider bandwidth is possible in PFT while PFFT is limited by the filter structure.

III. PFFT CHANNELIZER SIMULATION

We develop a MATLAB tool that we call PFFT Simulator (PFFTSim) to simulate two PFFT channelizer structures: *(i)* the standard and straight structure, and *(ii)* the straight-less structure which is based on our own practical manipulation that will be discussed in the coming sections. In order to prepare the received complex wideband signal, PFFTSim generates for each narrow band channel a random binary sequence at bitrate 90kHz for 100ms simulation time and modulates them using BPSK modulator. BPSK symbols are up-sampled in two stages: *(i)* we reach in the first stage the rate of the narrowband channel bandwidth which is 100 kHz and *(ii)* in the second stage, we reach the sampling frequency defined according to the number of channels and channel's bandwidth, by applying a shaping filter that limits each channel bandwidth to its bitrate. PFFTSim allows the user to control the following six parameters:

- The desired PFFT structure.
- The number of channels
- How many kHz the channel of carrier 100kHz would be shifted.
- The choice or option of later-on recovering the shifted channel.
- The need for the testbench files.
- If operations would be sequentially done using MATLAB *conv* function or in parallel where all channels are operated with each other.

Parallelization is useful to clarify how to type PFFT channelizer code for real implementation. Then, PFFTSim upconverts the upsampled symbols using complex heterodyne

sinusoid over carriers uniformly and continually distributed by 100kHz spacing except the channels of carriers -300 and 200 are left unsed while that of 100kHz is undergone user's control. Therefore, the complex wideband signal will be recovered and analyzed using a PFFT channelizer, one downsampling stage from the narrow bandwidth to the bitrate where other stage was done by PFFT, a BPSK demodulator, descriptive figures and a bit error rate calculator.

We simulate Scenario 1 using PFFTSim with various parameters depicted in Table 3, using complex wideband spectrum with LPF and BPFs that are uniformly and continually shifted using complex sinusoid with spacing 100kHz. Filter's taps were delayed by the number of channels M(e.g. for channel 2 located at 100kHz, the filter taps' order is $h(M+1), \dots, h(end), h(1), \dots, h(M))$ and presented in green, to highlight the fact that this specific delay exactly follows the BPFs. Figure 2 illustrates the spectrum of the output channels of PFFT channelizer at baseband surrounded by their filter banks (20 taps out of 160 taps each), to highlight the fact that as the whole filter cover each channel at bandpass, the decimated filter does at baseband where the decimation factor is unified between the filter and the wideband signal. The five major realizations in the recovered channels are: *(i)* the channels' power spreads along 90 kHz(bitrate) because of the shaping filter, *(ii)* the received power ($\cong -50db$) of the active carriers is conserved by PFFT, *(iii)* the power at inactive carriers is negligible, *(iv)* PFFT successfully detects power along non-uniformly distributed channel, namely channel 8 and *(v)* channel 8 which is shifted by 4 kHz in bandpass, is exactly 4kHz away from zero frequency.

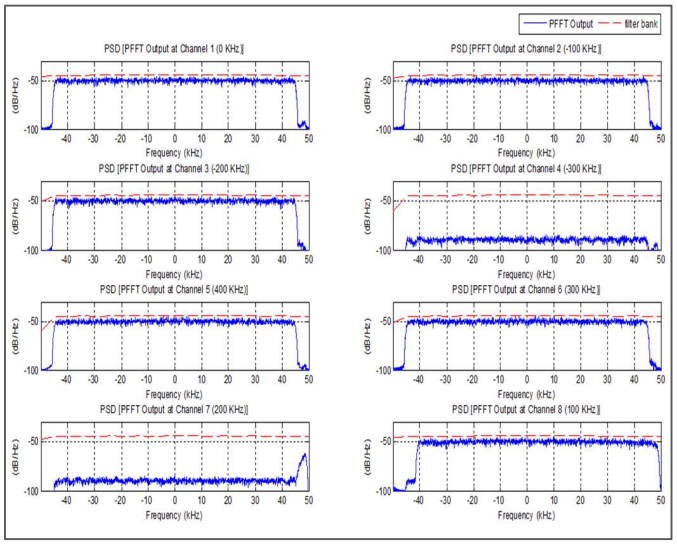

Figure 2. PFFT outputs for Scenario 1.

Then, PFFTSim down samples the baseband channels to their bitrates and recovers the binary sequences using BPSK demodulator. Finally, the binary sequences are compared to the transmitted data using the bit error rate calculator, and only the data that is held on active and uniformly distributed channels is successfully recovered (BER=0).

IV. PFFT FPGA Implementation

We use Vivado HLS to design a PFFT channelizer that supports complex channels. Vivado HLS transforms C specifications into a Register Transfer Level (RTL) implementation that synthesizes into a Xilinx Field Programmable Gate Array (FPGA). For validation, we create MATLAB test bench files using PFFTSim to generate the inputs and evaluate the outputs of the HLS block, where 32-bit wide PC_in and PC_out are the IQ channels of the input and the output respectively. $Ch\_identifier$ is a 16-bit output indicates the id of the channel of the current channelized sample where the carrier of the channel is $id \times spacing$.

Regarding FFT, we decide to use the Xilinx FFT IP core This IP core takes the input and directs the output in arrays of length equal to FFT bins and operates in several architectures. Most designers currently use of-the-shelf FFT core. In order to achieve real time performance using of-the-shelf FFT core, we suggest the new architecture of PFFT namely PFFT straight-less structure that was discussed and validated below in terms of formulations, Bit Error Rate (BER) in presence of White Gaussian Noise (WGN), and real time running support using of-the-shelf FFT core.

While filling the input samples, we delayed samples' "pass" by $M-1$. This necessitates the input commutator to rotate clock-wise starting from the topmost as we altered $\mathrm{FDM_{Ci}}$ while the filter banks are flipped topsy-turvy. We refer to this structure as the "PFFT channelizer in the straight-less structure". Note that the output in the new structure is rotate counter clock-wise starting from the topmost as we altered $\mathrm{TDM_{Ci}}$ due to the change in the direction of the input commutator.

Figure 3 presents the straight-less structure of PFFT where the three simple changes are: *(i)* the input commutator switches in clock-wise direction *(ii)* the banks of the PFB are flipped topsy-turvy which is a simple change in addressing and *(iii)* the output channels are ordered as follows: *(iii.1)* the baseband frequency at the upper output (y_0), *(iii.2)* followed by the positive carriers in increasing order, *(iii.3)* and finally the negative carriers also in increasing order as PC_out_TID is numbered in Figure 5 while the output channels' order of the straight structure can be achieved by following $\mathrm{TDM_{C1}}$.

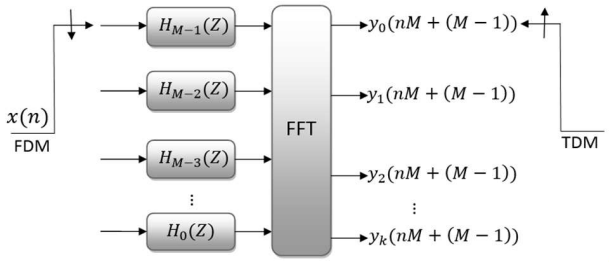

Figure 3. Straight-less structure of PFFT Channelizer.

This manipulation is validated using PFFTSim where it passes with $BER = 0$. We also validated it using BER metric in comparison with Per channel channelizer and PFFT straight structure where White Gaussian Noise (WGN) was added and SNR was increased from -40 dB to -1 dB. As shown in Figure 4, PC channelizer, PFFT straight structure and PFFT straight-less structure are similar in terms of BER in presence of WGN.

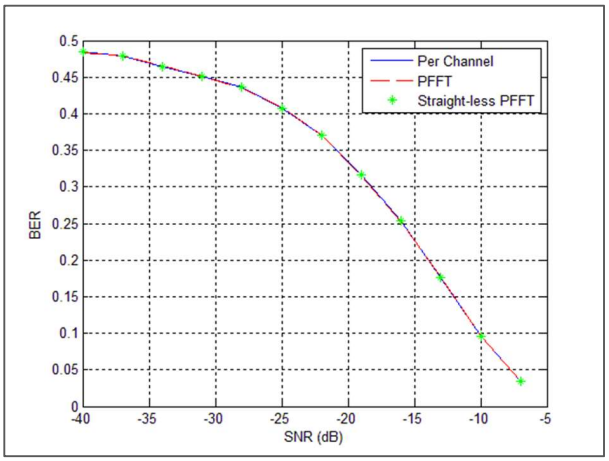

Figure 4. BER versus SNR in presence of WGN.

In order to validate the real time support of PFFT straight-less structure using of-the-shelf FFT core, we made **PFFTChannelizer** in this new structure using Xilinx FFT IP core, set streaming constraint on the input/output interfaces, created and simulated the block. Simulation's result is shown in Figure 5 where PC_out_TVALID is always 1 and PC_out_TID always indicates the next channel at each period. Note that, where the failure is expected at $PC\_out\_TID = 0$, PC_out_TVALID goes down for less than a clock cycle. Thus, **PFFTChannelizer** continuously receives and transmits samples without freezing the input commutator that operates at maximum applicable ADC's rate because of the channelizing process delay. Then, **PFFTChannelizer** allows real time feature support at the widest applicable band.

V. CONCLUSION

In this paper, we describe and analyze major channelizer algorithms. The requirements of each application including speed, hardware resources requirements and flexibility overbalance one over another. We also realize that speed is often the datum point in applications that require real time support and/or large multi-channels system support. Hence, we focus on PFFT channelizer and simulate it using MATLAB tool. We also present a PFFT channelizer hardware implementation applicable for real time applications based on a newly suggested architecture (referred to as straight-less architecture) that benefits from of-the-shelf FFT cores.

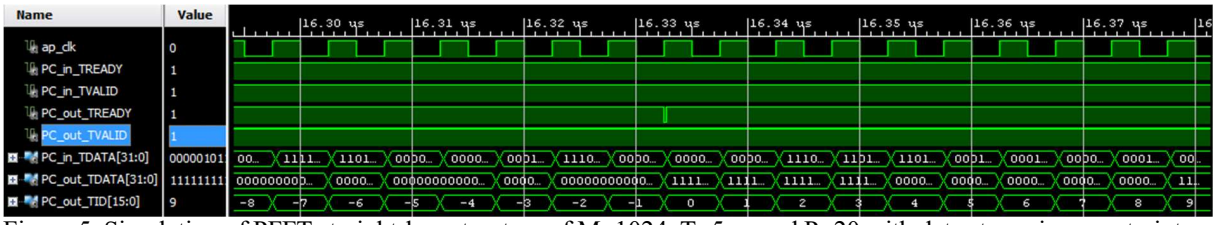

Figure 5. Simulation of PFFT straight-less structure of M=1024, T=5ns, and P=20 with data streaming constraint.

REFERENCES

[1] L. Pucker, "Channelization Techniques For Software Defined Radio," in Proceedings of SDR Forum Conference, 2003.

[2] F. Harris, C. Dick and M. Rice, "Digital Receivers and Transmitters Using Polyphase Filter Banks," IEEE Transactions on Microwave Theory and Techniques, vol. 51, no. 4, pp. 1395 - 1412, April 2003.

[3] J. Lillington, "Flexible Channelisation Architectures for Software Defined Radio Front Ends using the Tuneable Pipelined Frequency Transform," in DSP enabled Radio, 2003 IEE Colloquium on, Scotland, 2003.

[4] R. Mahesh, A. P. Vinod, E. M.-K. Lai and A. Omondi, "Filter Bank Channelizers for Multi-Standard Software Defined Radio Receivers," Journal of Signal Processing Systems, vol. 62, no. 2, pp. 157-171, Feb 2011.

[5] J. Lillington , "electronicdesign," 1 Dec 2003. [Online]. Available: http://electronicdesign.com/analog/slice-and-dice-chunks-radio-spectrum.

[6] R. Kumar, T. Nguyen, C. Wang and G. Goo, "Signal Processing Techniques For Wideband Communications Systems," in Military Communications Conference Proceedings, 1999. MILCOM 1999. IEEE, Atlantic City, NJ, 1999.

[7] F. Harris and C. Dick, "Performing Simultaneous Arbitrary Spectral Translation and Sample Rate Change, in Polyphase Interpolating or Decimating Filters in Transmitters and Receivers," in Software Defined Radio Conference, San Diego, California, USA, 2002.

[8] D. Zhou, "A Review Of Polyphase Filter Banks And Their Application," Air Force Research Laboratory, Rome, NY, 2006.

[9] M. Frigo and S. Johnson, "The Design and Implementation of FFTW3," Proceedings of the IEEE , vol. 93, no. 2, pp. 216-231, 2005.

[10] M. Kumar, D. Digdrasim and T. Ram, "Design and FPGA Implementation of Channelizer & Frequency Hopping for Advanced SATCOM System," International Journal on Recent and Innovation Trends in Computing and Communication, vol. 1, no. 1, pp. 1 - 6, Jan 2013.