Hardware Implementation of Overlap-Save based Fading Channel Emulator

M. Najam-Ul-Islam, Muhammad Nauman, Atif Raza Jafri, Jérémy Nadal, Charbel Abdel Nour, Amer Baghdadi

Hardware Implementation of Overlap-Save based Fading Channel Emulator

M. Najam-ul-Islam, Muhammad Nauman, Atif Raza Jafri, Jérémy Nadal, Charbel Abdel Nour, and Amer Baghdadi

***Abstract*— An efficient hardware implementation of correlated Rayleigh fading channel simulator is presented in this paper. It emulates Doppler effects based on the use of Overlap-Save (OLS) method. OLS is used for both, the fading variates generator and the time domain interpolator, leading to a scalable complete solution. Moreover, additional simplifications were introduced to reduce even further algorithmic complexity when compared with the original OLS-based proposal. An efficient hardware implementation is achieved through maximizing the utilization rate of allocated hardware resources. When added to its scalability, this makes the proposal appealing to emulate channels with multipath effects for MIMO systems. Indeed, the proposed parallel architecture enables a throughput of 34 Mega Samples per Second per path at a clock frequency of 275 MHz on Xilinx Virtex-7 FPGA. The achieved throughput allows the support of the most demanding LTE configuration with 20 MHz channel bandwidth. To the best of our knowledge, this is the first ever real-time hardware implementation of OLS-based channel emulator.**

***Index Terms*— Channel Emulator, Overlap-Save Method, High throughput, FPGA**

I. Introduction

SUCCESFUL deployment of a wireless communication system calls for a prior evaluation of its performance under different channel conditions. The use of a channel emulator provides an appealing alternative to tedious and costly real-time field-testing. Generation of Gaussian random variates [1] can be used for emulating a fading channel with Doppler effects. Emulation techniques are categorized into three main methods: Sum of Sinusoids (SOS), Gaussian noise filtering and FFT/IFFT-based. These methods can be extended to support different fading distributions following the works in [2-5].

Sum of Sinusoids (SOS) method [6] is based upon adding a large number of sinusoids to emulate the variation of fading. In real-time scenario having long simulation times, SOS can suffer from a periodicity issue between output samples. The second approach requires filtering the Gaussian noise with a Doppler filter [7-8]. The type of filter, e.g. Finite Impulse Response (FIR) or Infinite Impulse Response (IIR), greatly affects the design in terms of hardware complexity and stability.

The FFT/IFFT-based method [9] incorporates frequency domain filtering to get Rayleigh fading variates. This is achieved by multiplying Fourier transform of Gaussian variables with a frequency domain Doppler filter taps. However, the use of large FFT sizes requires higher hardware cost [10] whereas the use of smaller FFT/IFFT sizes suffers from a discontinuity of generated samples.

Authors in [10] proposed a new technique for generating Rayleigh fading variates by incorporating Overlap-Save (OLS) method. This technique combines a smaller FFT size with OLS to reduce computational complexity and to overcome the discontinuity issue. Moreover, the introduction of an OLS-based interpolator for the support of low Doppler frequencies reduces further computational complexity while achieving a high level of scalability. Therefore, the proposal in [10] achieves a high accuracy close to that of FFT/IFFT and FIR approaches, yet with significantly reduced complexity enabling real-time hardware implementations. Although the complexity of the SOS method remains the lowest, it cannot be used in long real-time emulations due to the periodicity issue that affects the accuracy and reliability of emulated channel effects [10].

Main prior solutions for real-time hardware implementations of different channel emulators can be found in [11-15]. Solutions in [11-14] are SOS-based whereas the solution in [15] represents a novel Sum-of-Frequency-Modulation (SoFM) method used in a Multiple-Input Multiple-Output (MIMO) system. While looking at the OLS-based emulator presented in [10], implementation complexity is provided only in terms of complex arithmetic operations. Taking into account the improved accuracy provided by the OLS-based emulator when compared to the SOS method and the reduced complexity when compared to the FFT/IFFT based filtering method, the work presented in this paper fills the gap of non-existent hardware implementation of OLS-based emulator. Indeed, several architectural choices have been explored to set the course for important choices such as FFT size, number of generated samples, supported sampling frequency, and interpolation size. Hence, this work represents the first attempt to provide a concrete hardware implementation for an OLS-based channel emulator. Moreover, a key feature of our proposed architecture concerns its scalability while keeping a high degree of efficiency and utilization rate. In this regard, multiple instances can be used to emulate multiple channels or to achieve higher throughput or both, adding to the originality of the approach. Many trials and architecture exploration iterations were applied to achieve this goal. This exploration phase involved adjusting operational frequency of functional blocks, pipelining at sub-block and system levels, and using promising methods for area reduction. In particular and differently from the original paper [10], we chose to implement the interpolation block in time domain instead of frequency domain through the management of introduced zeros. This greatly reduces the required memory, even with respect to the reference proposal in [10].

Muhammad Nauman, Atif Raza Jafri and M. Najam-ul-Islam are with Electrical Engineering Department of Bahria University Islamabad, Pakistan.

Jérémy Nadal is with Polytechnique Montréal, Canada; Charbel Abdel Nour, and Amer Baghdadi are with IMT Atlantique, Lab-STICC, Brest, France.

II. System Model

The OLS-based channel emulator is shown in Fig. 1 [10]. The generator first provides Rayleigh fading variates at an intermediate sampling frequency dependent on the fading variation or speed of movement. Then additional variates are introduced between two generator variates through interpolation to match the target transmitted signal sampling frequency as seen in Fig. 1 and in compliance with [10].

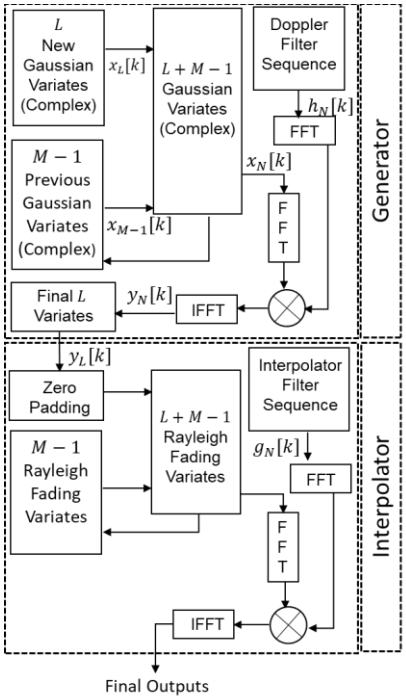

Figure 1: Block diagram of OLS-based channel emulator

A. Generator

The OLS generator resorts to fast circular convolution in frequency domain that can be applied through FFT/IFFT [10]. In each trial, L new samples, corresponding to the useful samples, are introduced along $M-1$ samples from the previous block to provide x_N where N is the FFT size ($N = L + M - 1$). Applying a Doppler filter in frequency domain as proposed in [13] with OLS causes discontinuities between blocks. Therefore, a correction procedure is applied as defined in [10] through shifting filter coefficients and windowing:

$$h_N[k] = \begin{cases} \dfrac{\Gamma\left(\frac{3}{4}\right)}{\Gamma\left(\frac{5}{4}\right)}\sqrt{f_d}, & \text{for } k = \dfrac{M}{2} \\ \Gamma\left(\frac{3}{4}\right)\sqrt[4]{\left(\dfrac{f_d}{\pi T_s|\lambda_k|}\right)}J_{\frac{1}{4}}(2\pi T_s|\lambda_k|), & \text{for } 0 \le k \le \dfrac{N+M}{2} \\ \Gamma\left(\frac{3}{4}\right)\sqrt[4]{\left(\dfrac{f_d}{\pi T_s|\lambda_k - N|}\right)}J_{\frac{1}{4}}(2\pi T_s|\lambda_k - N|), & \text{otherwise} \end{cases} \quad (1)$$

where Γ is the Gamma function, $J_{\frac{1}{4}}$ is the Bessel function (first kind), M is the number of non-zero filter coefficients and $\lambda_k = k - M/2$. Furthermore, the filter $h_N[k]$ is centered at $M/2$ and multiplied with a Kaiser Window [16] to remove discontinuities caused by suppressing filter taps $M \le k \le N-1$) [10]. The size of Kaiser Window is equal to the FFT size whereas filter taps depend on the value of M [10].

B. Interpolator

The interpolator is introduced to reduce the computational complexity of the complete channel emulator while keeping the required scalability in terms of supported Doppler and signal frequencies for practical scenarios. Its use is particularly critical for the cases where the ratio between the sampling frequency and the Doppler frequency is very large. The interpolator first inserts $N_I - 1$ zero samples between any two generator Rayleigh fading variates to provide L samples. N_I is defined by dividing the sampling frequency f_s by the intermediate sampling frequency. To simplify the interpolator design applying FFT/IFFT, N_I is chosen to be a power of two. Moreover, the choice of the intermediate sampling frequency is constrained by the acceptable level of out-of-band power leakage [10]. Zero packed variates are concatenated with $M-1$ zero packed previous variates and frequency domain filtering is then applied to get L new final samples. To follow the transmitted signal, the interpolator filter applies a sinc function that is sampled and truncated to a FIR filter [10]:

$$g_N[k] = \begin{cases} \dfrac{\sin\left(\pi f_s\left(n - \frac{M}{2}\right)\right)}{\pi f_s\left(n - \frac{M}{2}\right)}, & \text{if } 0 \le k < \dfrac{N+M}{2} \\ \dfrac{\sin\left(\pi f_s\left(n - \frac{M}{2} - N\right)\right)}{\pi f_s\left(n - \frac{M}{2} - N\right)}, & \text{if } \dfrac{N+M}{2} \le k < N \end{cases} \quad (2)$$

Interpolation filter is also multiplied with a Kaiser Window to avoid discontinuities.

C. Multipath Channel and MIMO System

Multipath-MIMO system requires emulation of individual paths of the channels between each transmit and receive antenna. Considering Extended Vehicular A (EVA) and Extended Typical Urban (ETU) channel models of LTE [17], nine statistically independent channel paths are specified with different power-delay profiles in a Single Input Single Output (SISO) configuration. Therefore, 4×9 = 36 paths are required to mimic channel behavior of a 2×2 MIMO system. In order to support the largest bandwidth of 20 MHz, the minimum required throughput should be 30.7 MSPS per channel path. In this regard, we consider a short safe margin and target a slightly larger throughput of 34 MSPS. In addition, to support multiple communications services, the hardware of the emulator should be scalable to support a wide scope of possibilities ranging from single path SISO to multipath MIMO systems while maintaining efficient hardware utilization.

III. Timing Requirements and Architectural Choices

In order to propose an efficient hardware architecture there is a need to assess the timing requirements of each building block. Appropriate hardware implementation techniques such as process level and register level pipelining to improve timing requirements, are applied.

A. Timing Requirements

Timing requirements related to each functional block depend upon the corresponding parameters such as value of L, M and N. In our case $N = 8192$ whereas L can take the value of 4, 8, 16, 32 and 64 to simulate a Doppler frequency f_d ranging from 11 to 344 Hz. Hence, the Generator block of Fig. 1 performs 8K

point FFT/IFFT operations and generates 4, 8, 16, 32 or 64 samples. These outputs of the generator block are up-sampled through zero padding by the interpolator block (Fig. 1) to generate 4K outputs. The other 4K data samples are taken from the previous trial to make a block of 8K data samples. 8K FFT/IFFT operations are then performed and finally 4K new samples are generated. To achieve a throughput of 34 MSpS, the system given in Fig. 1 is applied 34M/4096 i.e. 8301 times/sec. This provides a timing requirement of 120 µsec to complete the processing in all functional blocks of Fig. 1.

B. Architectural Choices

1) Generator Block

In the generator block, the frequency domain Doppler filter coefficients are considered as input (coefficients are generated before the emulation process) due to their slow varying nature. They are computed depending on the speed of mobile device. This avoids the application of the Doppler filter sequence generator and the following FFT in the generator block of Fig. 1. To complete the generator, architectures for Gaussian variate source, 8K-point FFT, complex multiplication and 8K-point IFFT are still required while meeting the requirements of the output sample rate.

2) Interpolator Block

Interpolation process involves up-sampling (zero insertion) followed by filtering. In the original OLS method [10] presented in Fig. 1, frequency domain filtering is used. However, time domain filtering can be also considered due to the presence of a large number of zeros in the up-sampled input data to the filter. This zero insertion provides the opportunity to reduce the hardware cost in FIR filter architecture as proposed in [18].

3) Fixed-Point Representation

In order to scale the data path, the floating point golden reference software model was converted to fixed-point model. Word lengths for all the inputs and outputs are chosen carefully in order to use available resources efficiently while respecting required accuracy. Table 1 illustrates our chosen word lengths. Complex values have two components to represent both real and imaginary parts.

Table 1: Word lengths of the proposed fixed-point representation

Parameter	Complex /Real	No. of Bits
Gaussian Variates $x_N[k]$	Complex	16
Doppler Filter Coefficients $h_N[k]$	Complex	16
FFT Inputs	Complex	16
Interpolator Filter Coefficients $g_N[k]$	Real	25
Final Outputs	Complex	16

IV. Hardware Architecture of Channel Emulator

This section starts by detailing the hardware architecture of individual computational blocks followed by the overall architecture of the channel emulator along with the task mapping on each block.

A. Gaussian Variate Source

For complex Gaussian variate generation, two instances of a Gaussian generator are required. We have used the open source core for Gaussian noise generation (GNG) presented in [19]. It represents a resource-efficient high-throughput implementation while having a long period of 2^{176}.

B. FFT/IFFT

Computing 8K FFT/IFFT requires the major part of the DSP operations carried out in the generator block of Fig. 1. Two approaches are possible: the use of dedicated hardware for each FFT/IFFT block or the re-use of a single FFT/IFFT core to perform all FFT/IFFT blocks needed in the generator block. Considering targeted throughput, we have adopted the second option. This option can simplify achieving the required scalability for emulating multipath channels and MIMO. This also helps to opt for the specialized limited FPGA resources such as DSP blocks and to be able to use them efficiently.

Our FFT/IFFT block is capable of pipelining in frame loading, computing and frame output. One input bit selects FFT or IFFT operation. Each frame-loading and frame-output process takes 8192 Clock Cycles (CC) whereas computations take 8332 CC. This results in 24,716 CC to process one 8K FFT or IFFT. This block can reach 278 MHz on Virtex-7 FPGA.

C. FIR Filter for Interpolator

In order to achieve the required throughput for worst case scenario i.e. $L = 64$ (the case with minimum zero padding), 16 real multipliers are placed to multiply 8 complex inputs with 8 real filtering coefficients in parallel. The coefficients are stored in memory (25 bits for each) whereas 64 registers, storing complex inputs, are used to act as shift registers of the FIR filter. Each input and output of the interpolator are quantized over 16 bits. The architecture of the filter is flexible with regards to specific well-chosen values of L defining required resources through the control path. To generate a set of 4096 channel coefficients, the whole process takes 32778 CC and the highest achievable frequency is 295.8 MHz on Virtex-7 FPGA.

D. Channel Emulator: Complete Architecture

The operating frequency of the FFT/IFFT core being lower than the one of the FIR filter, it is adopted as operating frequency of our channel emulator. It allows 33,379 CC in 120 µs which is the time required to generate one set of 4096 channel coefficients. Since the FFT/IFFT processing in the generator takes 24716 CC and since the FIR filtering in the interpolator is taking 32778 CC, process level pipelining is proposed between these two blocks through a memory able to store L variates created by the generator block.

Since FFT and IFFT processes are performed successively in the generator block, three process level pipelining within our proposed FFT/IFFT block can be optimally utilized if two frames for separate multipath channels are processed in parallel. The frame level processing is shown in Fig. 2. At the start, Frame 1 of complex Gaussian variates of Path 1 (green color) is loaded. After 8192 CC, Frame 1 of path 1 is sent to FFT processing whereas in the meanwhile Frame 1 of Gaussian variates of Path 2 is loaded (orange color). After the processing of Frame 1 of Path 1, the three pipeline lanes are filled and 32778 CC are required to produce successively two sets of L variates for two paths of a Rayleigh fading channel. The interpolation of data for the first path starts once variates of the second path are ready as indicated by the gray colored wait block in Fig. 2. This design choice allows the use of one set of coefficients for both interpolators and hence reduces the size of the corresponding memory storing these coefficients. The variates are given to two interpolator blocks for final outputs generation in parallel while taking 32918 CC to process a block.

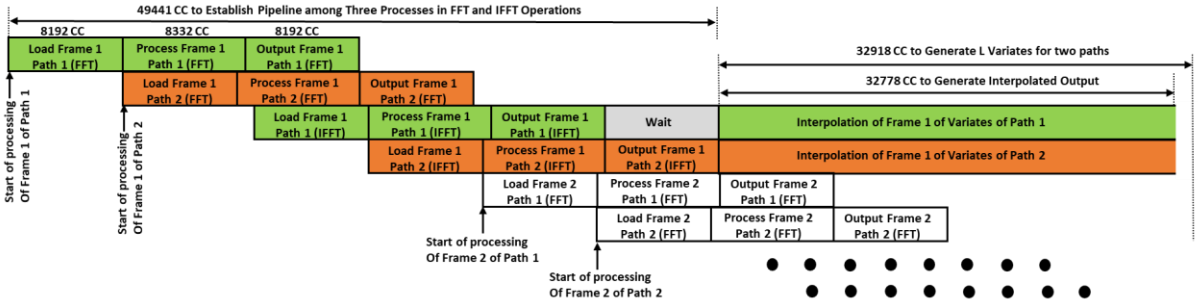

Figure 2: Process execution sequence in the generator and interpolator blocks

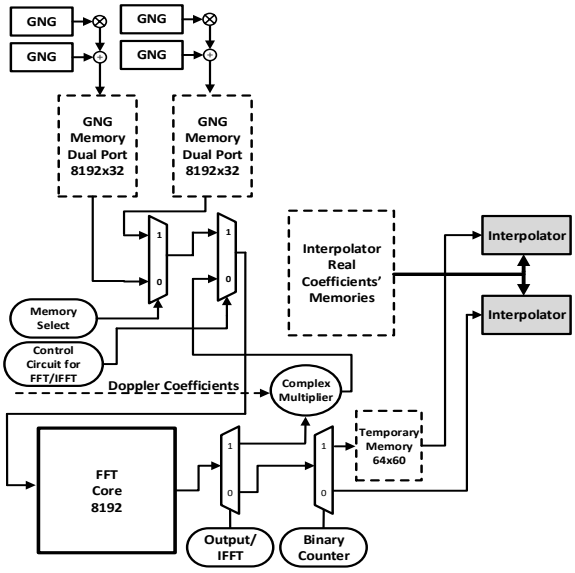

Figure 3: Hardware architecture of channel emulator

Since both processes, the generation and the interpolation, require a number of CC less than the allowed 33,379 CC, a throughput of 34 MSpS per path for two paths can be achieved with our proposed pipelining and task scheduling on used resources. The complete architecture of channel emulator is shown in Fig. 3 where two GNG memories hold Gaussian variates for two paths generated through four instances of GNG IP core. A single FFT/IFFT block is used along with multiplexers/de-multiplexers and control circuitry to perform both FFT and IFFT operations following the schedule illustrated in Fig. 2. One Doppler coefficients memory and one complex multiplier are used to multiply the output samples of the FFT by these coefficients. A 64-word (for highest value of $L = 64$) temporary memory holds the result of the generator of the first path to start interpolation of both paths in parallel. As explained earlier, this single memory can be re-used for two interpolators. Finally, two instances of the interpolator, detailed in Section III. C, are placed to generate final outputs.

V. Implementation & Performance Comparison

A. *Implementation Results*

A Virtex-7 XC7VX550T FPGA from Xilinx was selected as implementation platform. Table 2 illustrates the post place and route results of our architecture. Fewer resources are required in the interpolator block compared to the generator block especially in terms of memory: almost 5 times less 18Kbits BRAM and 27 times less 36Kbits BRAM. This fully justifies our proposal of implementing the interpolation in time domain.

Table 2: Post Place & Route results of the proposed channel emulator

Process	Slices	BRAM (18K)	BRAM (36K)	DSP Slices	Max. Freq. (MHz)
Generator	2,402	52	27	37	278.164
Interpolator	2,352	10	1	32	295.770
2 × 2 MIMO	85,572	1,116	504	1,242	278.164

Moreover, thanks to improved hardware efficiency through pipelining at process and hardware architecture levels, two streams of channel coefficients, each achieving a throughput of 34 MSpS, were provided. The resulting architecture is inherently scalable in support of multiple paths and throughput levels. As a path scalability example, consider a 2×2 MIMO system where each of the 4 channels have 9 paths. 18 instances of the proposed architecture can be easily instantiated to emulate the corresponding channel. The post Place & Route results are shown in Table 2.

For the throughput level scalability, the outputs of multiple architecture instances can be collected in a buffer and can be sent at a rate of one coefficient per clock cycle. Hence, the output can be raised beyond 278 MSpS per path. Finally, in order to validate the accuracy of output results, a comparison in terms of achieved Power Spectral Density (PSD) was performed as shown in Fig. 4. The PSD drawn from the floating point OLS C-model was taken as reference. It is clear that the PSD results of fixed-point model and actual FPGA implementation closely match the reference C-model.

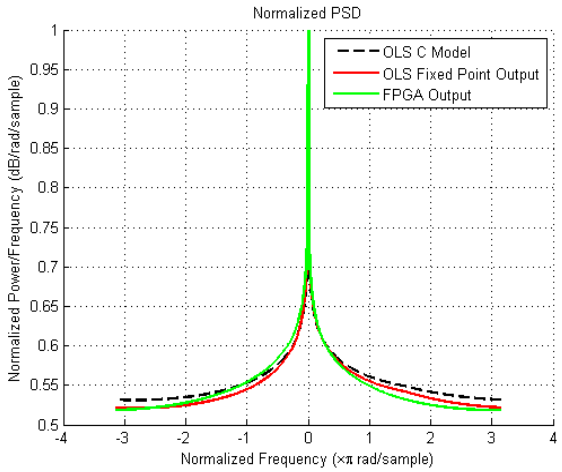

Figure 4: Normalized PSD comparison of output samples taken from floating point reference model, fixed-point model and FPGA for f_d = 17.22 Hz and f_s = 30.72 MHz

Table 3: Comparison with state-of-the-art in terms of features, scalability and hardware implementation

Ref.	FPGA	Application Domain	Scalability		Slices/LUTs	DSP48	BRAMs	MSpS	Throughput/Area (Samples/Slice)
			Throughput	Path					
[11]	Virtex-2	SISO	No	Yes	8814/-	-	-	210	23825
[12]	Virtex-4	SISO	No	Yes	1164 /-	-	-	0.585	502
[13]	Virtex-4	SISO	No	No	725/922	3	1	no info.	N/A
[14]	Virtex-4	SISO	No	Yes	731/1220	-	-	0.1	136
[15]	Kintex-7	MIMO	N/A	N/A	381/1524	16	17	0.0625	164
Our Solution	Virtex-7	SISO/MIMO	**Yes**	Yes	4754/13029	69	28	68	14303

B. Comparison with state-of-the-art

Keeping in mind the limited relevant published work, comparison with state-of-the-art is challenging from a hardware implementation perspective. In this regard, data taken from cited work [11-15] is tabulated in Table 2. This table contains information regarding key aspects of a solution such as domain of application, scalability towards increasing throughput and/or emulating more paths/channels, FPGA and its resource consumption, throughput and throughput/area ratio. As far as application domain is concerned, our solution can be used for SISO and MIMO systems in contrast to previous works that are limited to only one particular system. Scalability is another figure of merit since instantiating multiple copies of the proposed architecture can be simply used to improve throughput and/or to emulate multiple paths. This is difficult to achieve with previous methods such as SOS while still providing accurate simulations. The highest throughput is achieved by the solution in [11]. Achieved at a rate of one sample per clock cycle, this throughput cannot be increased further through scalability without re-thinking the architecture. However, our solution can provide a throughput more than 278 MSpS using multiple blocks of the proposed architecture.

Hardware utilization is another key outcome of our work. It can be observed that since the works in [11-14] are SOS based, no or very few BRAM or DSP Blocks are used due to their simplified architecture. Our solution requires additional hardware in terms BRAM and DSP due to the entirely different nature of involved computations. Nonetheless, this provides the price to pay in hardware resources by the use of the OLS method to overcome the drawbacks in terms of accuracy associated with SOS method.

Due to the different nature of the computations, any fair comparison of the throughput/area ratios across the methods should be considered jointly with the achieved level of accuracy. In the work presented in [15], the throughput is far less when compared to our solution and hence providing much lower throughput over area ratio.

VI. Conclusion

We have proposed the first real-time FPGA implementation of an OLS-based channel emulator. It achieves a throughput of 34 MSpS per path for the 20MHz channel of the LTE standard. It presents several advantages: support of SISO and MIMO, scalability in terms of throughput and multiple paths, improved hardware efficiency through pipelining and sample distribution accuracy. The proposed time-domain interpolation largely reduces the hardware complexity thanks to the devised simplifications made possible through zero padding. Comparison with existing prior art was done including the features of the emulator. This comparison clearly identifies the cost in terms of hardware resources to overcome the spectral property issues related to the SOS method through adopting the OLS method. Hence, a new accuracy/hardware complexity tradeoff becomes a possible choice for channel emulation.

VII. References

[1] J. S. Malik and A. Hemani, "Gaussian random number generation: a survey on hardware architectures," *ACM Computing Surveys*, vol. 49, no. 3, pp. 53:1-53:37, Oct. 2016.

[2] Ma, Y., and Zhang, D.: 'A method for simulating complex Nakagami-m fading time series with non-uniform phase and prescribed autocorrelation characteristics', IEEE Trans. Veh. Technol., 2010, 59, (1), pp. 29–35.

[3] L. Martino, D. Luengo, "Extremely efficient acceptance-rejection method for simulating uncorrelated Nakagami fading channels", Communications in Statistics - Simulation and Computation, Volume 48, Number 6, Pages: 1798-1814, 2019.

[4] Cao, L., and Beaulieu, N.C.: 'Simple efficient methods for generating independent and bivariate Nakagami-m fading envelope samples', IEEE Trans. Veh. Technol., 2007, 56, (4), pp. 1573 – 1579

[5] Zhu, Q.M., Dang, X.Y., Xu, D.Z., and Chen, X.M.: 'Highly efficient rejection method for generating Nakagami-m sequences', Electron. Lett., 2011, 47, (19), pp. 1100–1101

[6] M. Patzold, C.-X. Wang and B. O. Hogstad, "Two new sum-of-sinusoids-based methods for the efficient generation of multiple uncorrelated rayleigh fading waveforms," IEEE Trans. Wireless Commun , vol. 8, no. 6, pp. 3122-3131, Jun. 2009.

[7] C. Iskander, "A matlab-based object-oriented approach to multipath fading channel simulation," Feb. 2018. [Online].

[8] C. Komninakis and J. F. Kirshman, "Fast Rayleigh fading simulation with an IIR filter," RF Design, pp. 24-34, Jul. 2004.

[9] D. Young and N. Beaulieu, "The generation of correlated Rayleigh random variates by inverse discrete Fourier transform", vol. 48, no. 7, pp. 1114-1127, Jul. 2000.

[10] J. Yang, C. A. Nour and C. Langlais, "Correlated fading channel simulator based on the overlap-save method," IEEE Trans. Wireless Commun , vol. 12, no. 6, pp. 3060 - 3071, Jun. 2013.

[11] A. Ali Mohammad and B. F. Cockburn, "Modeling and hardware implementation aspects of fading channel simulators," in IEEE Tran. Veh. Technol., vol. 57, no. 4, pp. 2055-2069, Jul. 2008.

[12] A. Alimohammad and S. F. Fard, "FPGA Implementation of isotropic and nonisotropic fading channels," IEEE Trans. Circuits Syst., II, Exp. Briefs, vol. 60, no. 11, pp. 796 - 800, 26 Sep. 2013.

[13] P. Huang, "A novel structure for Rayleigh channel generation with consideration of the implementation in FPGA," IEEE Trans. Circuits Syst., II, Exp. Briefs, vol. 63, no. 2, pp. 216-220, Feb. 2016.

[14] P. Huang, Y. Du and Y. Li, "Stability analysis and hardware resource optimization in channel emulator design," IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 63, no. 7, pp. 1089 - 1100, 27 Jun. 2016.

[15] Q. Zhu et. al "A novel 3D nonstationary wireless MIMO channel simulator and hardware emulator," IEEE Trans. Commun., vol. 66, no. 9, pp. 3865 - 3878, 09 Apr. 2018.

[16] A. V. Oppenheim, R. W. Schafer and J. R. Buck, "Discrete-Time Signal Processing" in , Prentice-Hall, 1999.

[17] ETSI, "ETSI TS 136 101 V10.3.0," June 2011. [Online].

[18] A. R. Jafri et. al "Hardware complexity reduction in universal filtered multicarrier transmitter implementation," IEEE Access, vol. 5, pp. 13401-13408, 2017.

[19] G. Liu, Gaussian noise generator core specification, OpenCores, Jan. 2015.