An Efficient Low Cost FPGA MIMO Channel Model

Andrew Slaney, Yichuang Sun and Oluyomi Simpson
School of Physics, Engineering and Computer Science
University of Hertfordshire
Hatfield, UK
andyslaney@bitronix.com.au, y.sun@herts.ac.uk and o.simpson@herts.ac.uk

***Abstract*—In this paper, a mathematical model for generating AWGN and Rayleigh fading is presented and argued based around the generation of Gaussian distributed numbers. This paper is focused on a multiple input multiple output (MIMO) channel model for the design and implementation of space-time coded MIMO modem systems such that the complexity of the design is as much as possible pushed into the digital domain and that the architecture is computationally efficient, driving the emphasis and complexity of implementation into software. We present an FPGA architecture to yield a Rayleigh fading and AWGN model for MIMO systems requiring up to four transmit and two receive antennas while requiring only a slight increase in logic resource over a single input single output model. The design entry was in VHDL and the target FPGA was the Xilinx Spartan 3 XC3S4000.**

Keywords— FPGA, MIMO, Channel, AWGN, Rayleigh

I. INTRODUCTION

There has been much attention paid to the mathematical methods of generating Gaussian distributed numbers, particularly for computer based simulations. Most methods involve initially generating samples of a uniform random variable and then applying a transformation to obtain samples drawn from a unit variance and zero mean Gaussian probability density function. In real hardware systems, the environment will typically supply the noise required and so far less attention has been paid to hardware architectures for generating Gaussian noise, particularly to all digital architectures. With recent advances in Field Programmable Gate Array (FPGA) technology, hardware based simulations are receiving more attention due to their huge performance advantages over software based simulations [1-3][9-11]. It can take many hours to do a computer based simulation to obtain accurate error rate information above 10-6 for a given signal to noise ratio, particularly when simulating a complex Multiple Input Multiple Output (MIMO) communications link [5]. Such error rates can be obtained within minutes for an implemented hardware solution. Hardware based simulations not only offer real-time simulations but enable the designers to effectively and accurately evaluate their hardware architectures [12-13].

The authors of [1-3] have proposed suitable digital hardware architectures for generating Gaussian Noise, but the design requires up to 10% of an expensive Xilinx Virtex-II FPGA. For multiple antenna communication systems many fading parallel coefficients as well as Additive White Gaussian Noise (AWGN) samples are required. The challenge is to design a channel model suitable for MIMO communications that will consume as little logic resource as possible. The principle contribution in this paper is the designs of a simple Rayleigh fading and AWGN model for MIMO systems requiring up to four transmit and two receive antennas while requiring only a slight increase in logic resource to that already proposed. In the rest of the paper, Section II describes the AWGN channel. In Section III a Rayleigh fading channel model is proposed. Section IV presents the FPGA based generations of Gaussian noise, AWGN and Rayleigh fading. The FPGA based 2×2 and 4×2 MIMO channel models are given in Section V. Finally, conclusions are drawn in Section VI.

II. ADDITIVE WHITE GAUSSIAN NOISE

The Additive White Gaussian Noise (AWGN) is commonly used as the system noise model in communications systems. To simulate a communications system and characterise its performance in an AWGN channel, we must add white Gaussian noise to the transmitted signal. The complex noise sample to be added can be defined as follows:

$$n_i = A_t K_i \tag{1}$$

where K_i is a set of complex pseudorandom values of normal distribution with mean of 0 and variance of 1, and A_t is a real gain factor. In the simulations presented throughout this paper the data is expressed in voltage, so the noise signal must be expressed in voltage. Gaussian noise is normally distributed equally in-phase (I) and quadrature-phase (Q) channels, therefore the value A_t is defined as follows [4]:

$$A_t = \sqrt{\frac{1}{2} N_p} \quad , \qquad N_p = \frac{S_p}{F_b} . \frac{B}{(\frac{E_b}{N_o})} \tag{2}$$

where S_P is the average signal power and N_P is the average noise power, B is the Bandwidth (Hz) and F_b is the Channel data rate (Hz), and E_b/N_o is the ratio of the energy per bit to noise power spectral density.

III. RAYLEIGH FADING

For mobile terrestrial communications systems, the channel path between transmitter and receiver is characterised by various obstacles and signal reflections. The receiver may not have a direct line of sight with the transmitter. In this type of environment, the received signal will be a superposition consisting of several reflected, diffracted and scattered waves all having different phase and times of arrivals. If there are sufficient enough paths in the multi-path environment, then the central limit theorem holds that the channel impulse response will be well-modelled as a Gaussian process irrespective of the distribution of the individual components. The envelope of the channel response is said to be Rayleigh distributed if there is no dominant component to the scatter as such a process will have zero mean and phase evenly distributed between *0* and 2π radians [4]. The Rayleigh fading channel model, $\boldsymbol{H}$ has i.i.d, complex, zero mean, unit variance entries [8]:

$$h_{ij} = B_t K_{ij} \tag{3}$$

where K_{ij} is a set of complex pseudorandom values of normal distribution with mean of 0 and variance of 1 and B_t is a real gain factor defined as follows:

$$B_t = \frac{1}{\sqrt{P}\sqrt{2}} \quad (4)$$

where P is the number of transmit antennas.

IV. FPGA AWGN and Raleigh Fading Generators

A. Hardware Testbed

The modem and channel model have been implemented in a Xilinx Spartan 3 XC3S4000 FPGA. The Testbed consists of a proprietary test card developed at BiTronix Ltd containing a PIC micro-controller with an RS232 control interface, SPI interface for FPGA programming and setup of internal registers, 100MHz crystal oscillator, 100MHz voltage controllers crystal oscillator and two 125MHz digital to analogue converters for I and Q output monitoring. The RS232 control interface should connect to a PC running HyperTerminal, this allows access and control of the internal FPGA registers. There is also a 90-bit output logic bus which can connect to a logic analyser for internal debugging and signal monitoring. The demodulated I and Q outputs were outputted via the on-board DACs to be monitored via an oscilloscope. Fig. 1 shows a block diagram of the setup.

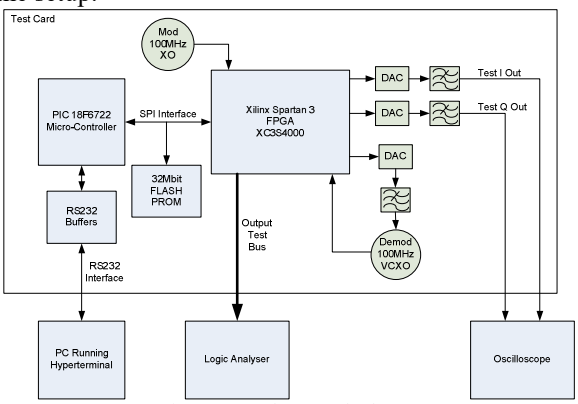

Figure 1 Modem Testbed.

B. The Gaussian Noise Generator

There is little previous work on digital hardware Gaussian noise generators [1-3]. Below we briefly introduce the FPGA based Gaussian noise generator. The choice of algorithm presented here is the Box-Muller method. This method requires the approximation of nonlinear functions which will be shown to be easily achieved within an FPGA.

The Box-Muller algorithm generates two random samples y_1 and y_2 of Gaussian distribution (with zero mean and standard deviation σ=1) from two uniformly distributed variables, u_1 and u_2, with values between the range 0 and 1 using the following set of equations [1]

$$f(u_1) = \sqrt{-\ln(u_1)}\ ,$$
$$g_1(u_2) = \sqrt{2}\sin(2\pi u_2)\ ,\ g_2(u_2) = \sqrt{2}\cos(2\pi u_2) \quad (5)$$
$$y_1 = f(u_1)g_1(u_2)\ ,\ \ y_2 = f(u_1)g_2(u_2)$$

This leads to an implementation architecture shown in Fig. 2 that has a four stage approach [1]:

1. Pseudorandom generation of u_1 and u_2
2. Implementation of the functions f, g_1 , g_2 and the subsequent multiplications.
3. Exploitation of the central limit theorem to overcome quantisation and approximation errors using a sample accumulation stage.
4. A multiplexer-based circuit which multiplexes noise samples y_1 and y_2 to produce a single noise sample (n) per clock cycle.

The generation of uniformly distributed realisations of u_1 and u_2 can be simply achieved by using linear feedback shift registers (LFSRs). An m-bit LFSR with an irreducible polynomial can produce an output with periodicity of 2^{m-1}. The required bit precisions of u_1 and u_2 are related to the maximum σ value that the full system will produce. It is shown in [1] that the maximum output is determined by f which takes on its largest values when u_1 is smallest whereas g_1 and g_2 are bound by $-\sqrt{2}$ and $\sqrt{2}$. A bit precision of 32 for u_1 provides a maximum output of 6.7σ. A bit precision of 18 for u_2 is sufficient without loss of performance [1]. Therefore, the total bit precision required for both u_1 and u_2 is 50 bits. Fifty 60-bit LFSRs with an irreducible polynomial can produce a period of over 10^{18}, which is more than adequate for even the most of ambitious simulations. The polynomial used for each of the LFSRs is $x^{60} + x^{59} + 1$.

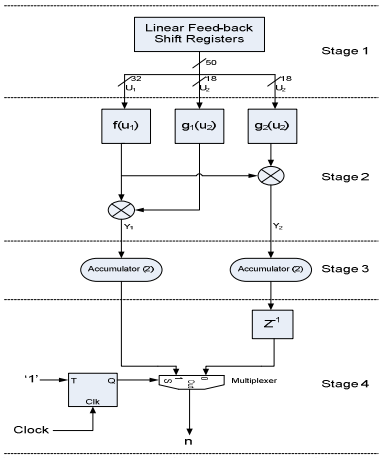

Figure 2 Gaussian noise generator architecture

The function $f(u_1)$ to be produced is a nonlinear function. The non-linearity of f is greater as u_1 approaches the extremities. The function f can be determined by the use of a look up table, but a very large look-up table is required given that u_1 has a bit precision of 32. A more effective way is to break up the function into linear segments, using smaller segments for the more non-linear regions and larger segments for the more linear regions. Segment boundaries at locations 2^{n-32} and $1-2^{-n}$, where $0 \le n < 32$, provides 62 segments. The function f can now be determined by the straight line graph equation:

$$f(u_1) = m_a u_1 + c_a \quad (6)$$

where m_a and c_a are coefficients from a look-up table corresponding to segment a. As the segments boundaries are defined by integer powers of two, very simple logic is required for the address decoding of u_1.

The design presented in this paper is different from that of the one proposed in [1]. The coefficient m_a ranges from -3.1857×108 to -1.1848, the former value requiring at least 28 bits to represent. However, it is also important to accurately represent the lower numbers and their fractional parts thereby requiring even higher bit precisions to cater for all values. Multipliers within an FPGA take up a lot of logic resource, for example a 32-bit input multiplier, yielding a 64-bit output requires 1088 look-up tables (LUTs) in a Spartan 3 XC3S4000 device, whereas a 16-bit input

multiplier requires only 280 LUTs. In this design the coefficients m_a are scaled to maintain a bit precision of 16

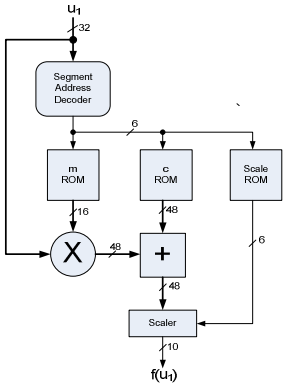

Figure 3 Circuit to calculate the function *f*

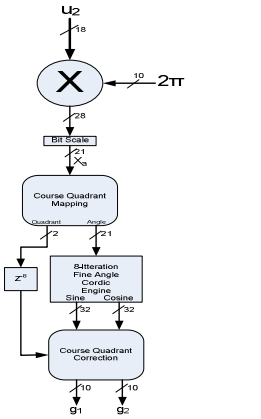

Figure 4 Block diagram for the circuit to calculate functions g_1 and g_2

bits, this leads to the requirement of storing the scaling factor in another look-up-table so that the output can be corrected. The coefficient c_a is scaled by the same amount. By ensuring the scaling factors are integer powers of two, the outputs can be adjusted by simple bit shifts rather than requiring a further multiplier. The block diagram for the circuit to calculate *f* can be seen in Fig. 3. The output $f(u_1)$ from the FPGA is further scaled to 10 bits, this is to provide a 4-bit integer part and 6 bit fractional part. This yields an output resolution of 2^{-6} .

There are many options for computing trigonometric functions, g_1 and g_2, two methods lend themselves well for FPGA implementation. The first is to make use of look-up tables to compute the sine and cosine functions and the second is to use a CORDIC (coordination rotation digital computer) engine. The former will require large amounts of block RAM when compared with the CORDIC method but will utilise less logic area. A memory intensive approach is not a good choice if block memory utilisation is an issue, conversely a computationally rich technique would not be suitable if it is desirable to conserve logic fabric resources [6]. The authors of [1] reject the CORDIC method for evaluating functions g_1 and g_2 on the basis of an execution time that is linearly proportional to the number of bits of the operand, thereby not suitable for applications requiring high accuracy and speed. Instead they propose the use of look-up tables which, as already mentioned, are memory intensive. The CORDIC method is embraced and by pipelining the CORDIC algorithm the speed limitations are overcome and the accuracy determined by the number of pipelined iterations. This eliminates the requirement for extra internal block RAM.

The amount of logic resource required by the CORDIC implementation is largely determined by the number of iterations required. A block diagram for the circuit to calculate the sine and cosine functions can be seen in Fig. 4. Eight pipe-lined iterations of the CORDIC engine are sufficient to keep the required logic resource to a minimum. The pipelining takes place within each of the adder/subtracter components shown in the 8-iteration CORDIC engine. Each of these components has a latency of one clock sample. Therefore, each iteration has a latency of one clock sample. The CORDIC engine is only capable of calculating the sine and cosine of angles (X_a) between 0 and π/2 radians. For values outside of this range the input angle needs to be modified to fall within this range and then the output values need to be corrected accordingly. The input angle will range between 0 and 2π and can be split into one of four quadrants:

Quadrant 1: $0 \le X_a < \pi/2$

Quadrant 2: $\pi/2 \le X_a < \pi$

Quadrant 3: $\pi \le X_a < 3\pi/4$

Quadrant 4: $3\pi/4 \le X_a < 2\pi$

The purpose of the course quadrant mapping circuit is to produce an output angle X_b that falls within the range $0 \le X_b < \pi/2$ using the following rules:

1) If X_a is in Quadrant 1 then $X_b = X_a$
2) If X_a is in Quadrant 2 then $X_b = \pi - X_a$
3) If X_a is in Quadrant 3 then $X_b = X_a - \pi$
4) If X_a is in Quadrant 4 then $X_b = 2\pi - X_a$

The course quadrant correction circuit produces outputs g_1 and g_2 based on the following rules:

1) If X_a is in Quadrant 1 then $g_1 = sin(X_b), g_2 = cos(X_b)$
2) If X_a is in Quadrant 2 then $g_1 = sin(X_b), g_2 = -cos(X_b)$
3) If X_a is in Quadrant 3 then $g_1 = -sin(X_b), g_2 = -cos(X_b)$
4) If X_a is in Quadrant 4 then $g_1 = -sin(X_b), g_2 = cos(X_b)$

Since further implementation, which exploits the central limit theorem, requires a division of √2 [1] while the computation of g_1 and g_2 requires the multiplication of √2, this term can be dispensed with, as shown above. The outputs of g_1 and g_2 have been scaled to 10 bits that have 4-bit integer value and 6-bit fractional representation, the same as for function f.

The output *n* has been scaled to 10 bits, which have 4-bit integer value and a 6-bit fractional representation. A histogram plot of 2 million samples of noise data with a 1000 bins is shown in Fig. 5. It is clear to see that the output values have a Gaussian distribution.

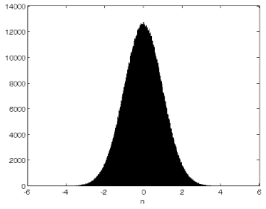

Figure 5 Histogram plot of output *n*

C. The AWGN Generator

The design is adapted further by the addition of a multiplier at the output n to adjust the amplitude of the noise samples for different *Eb/No* values required. The design is further changed to produce four Gaussian noise outputs by careful de-multiplexing of the output of the gain stage multiplier into four paths. However, the drawback of doing this is that four clock samples are required per output of noise sample on each port. Within Modem designs symbol rate interpolation filters are typically clocked at a frequency of four times the symbol rate. Therefore, by using the same interpolation clock frequency to supply the AWGN generator, four output noise samples are available for one modulation symbol period. As will be seen, this is enough to simulate up to two received antenna paths for complex modulated signals (such as QPSK). A block diagram for the modified design can be seen in Fig. 6.

The AWGN gain factor A_t is determined by (2). The gain value is a fixed point representation with 6-bit integer and 6-bit fraction. Histogram plots of over 500000 samples of all four outputs from the AWGN generator are shown in Fig. 7. It is clear to see that the output values have a Gaussian distribution. The design occupies approximately 11% of a Xilinx Spartan-3 XC3S4000-4 FPGA, requiring 3204 logic slices and can run at a clock speed of 100MHz, thus producing 4 noise samples every 25MHz.

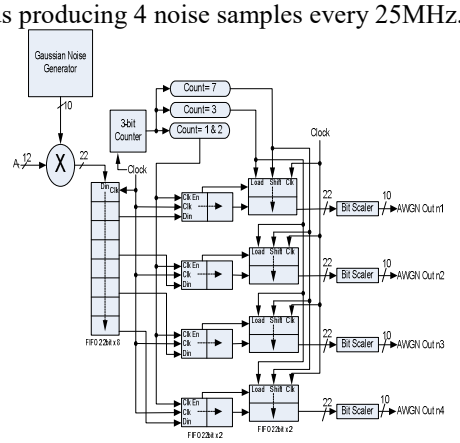

Figure 6 Modified Gaussian Noise Generator design to produce a 4 output AWGN generator

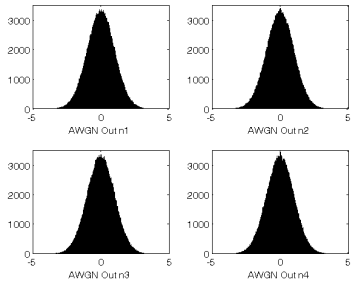

Figure 7 Histogram plots of over 500000 samples of all four outputs from the AWGN generator

D. The Raleigh Fading Generator

The Raleigh fading generator design differs slightly from the AWGN generator and requires very little more logic resource. There are two primary differences; the first is that the gain factor resolution B_t in (4) is only 6 bits, as $B_t < 1$, the second difference is that there are up to 16 fading coefficients available at the output for MIMO designs requiring up to four transmit antennas (P=4). The design for the 16 coefficient output is shown in Fig. 8.

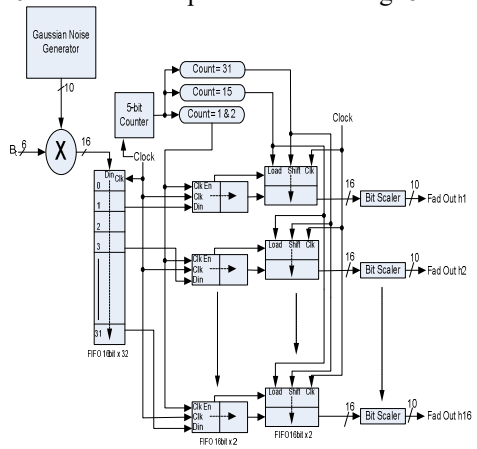

Figure 8 Rayleigh Fading Generator with 16 outputs

V. THE FPGA MIMO CHANNEL MODEL

For a system with P transmit and M receive antennas as, the receive signal can be defined as:

$$y = Hx + n \tag{7}$$

where $\boldsymbol{y}$ is the M x 1 received signal vector, $\boldsymbol{x}$ is the complex P x 1 transmit signal vector and $\boldsymbol{H}$ is the M x P complex channel gain matrix. The signal vector $\boldsymbol{n}$ consists of an M x 1 independent and identically distributed complex Gaussian noise components of modulus variance normalised to one.

A MIMO channel model design for a 2 × 2 and a 4 × 2 system can be seen in Fig. 9 and Fig. 10, respectively. The output latches following the fading generator are there to allow for the synchronisation of fading coefficients to the start of a space-time block code. This is important in evaluating the optimum performance of the design of any space-time block coding system as often it is assumed that the fading coefficients will remain fixed for the length of a block code. For example, the Alamouti transmit diversity code [7] achieves maximum performance when the channel fading coefficients are assumed to be constant across two consecutive symbols. The channel simulations are performed on the I and Q symbols of a QAM system.

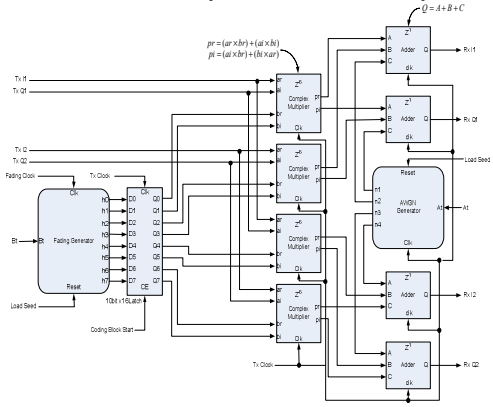

Figure 9 FPGA MIMO channel model for two transmit and two receive antennas.

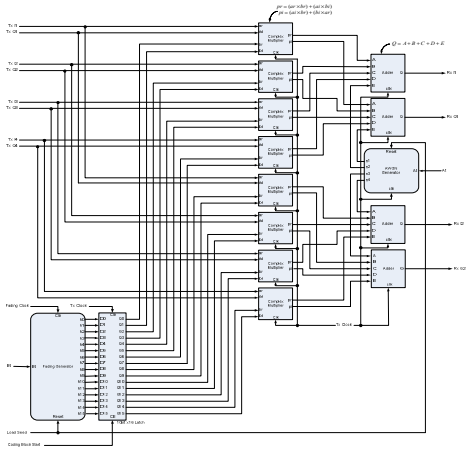

Figure 10 FPGA MIMO channel model for four transmit and two receive antennas.

VI. CONCLUSION

A simple mathematical model for generating AWGN and Rayleigh fading was presented and argued based around the generation of Gaussian distributed numbers. Using the Box-Muller method and with consideration to the logic requirements of a modem design, a computationally efficient circuit for generating numbers with a Gaussian distribution was presented. A bit-true analysis of the design was carried out and shown at various stages together with

the final implementation results. The design was modified further in an area-efficient manor, to include a gain factor and to produce non-correlated multiple outputs that all have a Gaussian distribution. A full channel model for a 2 × 2 and a 4 × 2 MIMO system was shown. We have demonstrated that it is possible to produce a Rayleigh fading MIMO channel model within a cheap Xilinx Spartan 3 FPGA. These models can be scaled up for more antenna MIMO channels.

REFERENCES

[1] D. –U. Lee, W. Luk, J. D. Villasenor and P. Y. K. Cheung, "A Gaussian Noise Generator for Hardware-Based Simulations", IEEE Transactions on Computers, Vol.53, No.12, pp 1523-1534, December 2004.

[2] J. –L. Danger, A. Ghazel, E. Boutillon, and H. Laamari, "Efficient Implementation of Gaussian Noise Generator for Communication Channel Emulation", The 7th IEEE International Conference on Electronics Circuits & Systems, Lebanon, pp. 366-369, Dec 2000.

[3] D. –U. Lee, W. Luk, J. D. Villasenor and P. Y. K. Cheung, "A Hardware Gaussian Noise Generator for Channel Code Evaluation", Proceedings of the 11th Annual IEEE Symposium on Field Programmable Custom Computing Machines, pp. 69-78, 2003.

[4] H. Harada and R. Prasad, "Simulation and Software Radio for mobile communications", p51, Artech House Publishers, 2002.

[5] A. Slaney and Y. Sun, "Space-time coding for wireless communications: an overview", IEE Proceedings, Communications, vol. 153, no. 4, pp. 509-518.

[6] G. J. Foschini and M. J. Gans, "On Limits of Wireless Communications in a Fading Environment when Using Multiple Antennas", Wireless Personal Communications, Vol. 6, pp. 311-335, 1998.

[7] C. Dick, F. Harris, and M. Rice, "FPGA Implementation of Carrier Synchronisation for QAM Receivers", Journal of VLSI Signal Processing, 36, pp. 57-71, 2004.

[8] S. M. Alamouti, "A Simple Transmit Diversity Technique for Wireless Communications", IEEE Journal on Selected Areas in Communications, Vol. 16, No. 8, pp.1451-1458, October 1998.

[9] O. Hiari, R. Mesleh and A. Alkhatib, "A Physical Transmitter Implementation of a Quadrature Space Shift Keying MIMO System," in IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 68, no. 1, pp. 251-255, Jan. 2021, doi: 10.1109/TCSII.2020.3006434.

[10] P. Zhang et al., "Design of Reconfigurable SDR Platform for Antenna Selection Aided MIMO Communication System," in IEEE Access, vol. 7, pp. 169267-169280, 2019, doi: 10.1109/ACCESS.2019.2946720.

[11] P. Huang, M. J. Tonnemacher, Y. Du, D. Rajan and J. Camp, "Towards Massive MIMO Channel Emulation: Channel Accuracy Versus Implementation Resources," in IEEE Transactions on Vehicular Technology, vol. 69, no. 5, pp. 4635-4651, May 2020, doi: 10.1109/TVT.2020.2980583.

[12] S. F. Fard, A. Alimohammad and B. F. Cockburn, "An FPGA-Based Simulator for High Path Count Rayleigh and Rician Fading," in IEEE Transactions on Vehicular Technology, vol. 59, no. 6, pp. 2725-2734, July 2010, doi: 10.1109/TVT.2010.2046660.

[13] A. Alimohammad and S. F. Fard, "A Compact Architecture for Simulation of Spatio-Temporally Correlated MIMO Fading Channels," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 61, no. 4, pp. 1280-1288, April 2014, doi: 10.1109/TCSI.2013.2285892.