Design and FPGA implementation an accurate real time 3x4 MIMO channel emulator

Omar A. Nasr and Babak Daneshrad

Abstract – **The design and implementation of an accurate and low complexity MIMO channel emulator is presented in this paper. A mathematical analysis is used to verify the accuracy of the emulator over a wide range of SNRs (0 – 35 dBs). The complexity of the emulator is reduced by preprocessing of the channels and hardware/software partitioning. All 802.11n channel models can be emulated on our platform. A 3x4 10MHz version of the emulator is successfully running on a Virtex-II XC2V6000-4 FPGA. A 20MHz version was synthesized and simulated on an XC2V6000-6 FPGA.**

I. Introduction

Testing the hardware implementation of the radio is important to determine the performance of the system. The design cycle of a radio has three phases: floating point simulation, fixed point simulation and hardware implementation. Using fast prototyping languages like MATLAB in the floating point and fixed point simulations has the advantage of flexibility in the design and performance testing, but its major drawback is its speed; the simulation speed is very slow compared to lower level languages like C; which is hard to debug and doesn't have fast prototyping capabilities like MATLAB. As a result, most of the researchers use MATLAB to simulate the performance of the system in floating point and fixed point domains. After hardware implementation of the system on an FPGA, the system needs to be tested and its performance in the real world needs to be quantified. One way to do that is to make field trials, where the Device Under Test (DUT) is taken to the filed and the performance of the system is quantified by sending data from transmitter and receiving them at the receiver and finding the PER. The main problem in this approach is that the field trials are done in a limited number of environments and under uncontrolled conditions. Channel emulation can solve this problem. The Tx DUT and the Rx DUT are connected to the ports of the emulator and the emulator applies a controlled environment to the Tx signal and adds controlled noise to the Rx signal.

In this paper, we show the design and implementation of a 3x4 MIMO channel emulator and integrated it with a full hardware implementation of an 802.11n radio. We have compared the performance of the channel emulator with the performance of software in the loop simulation and we have perfect matching between them. The main contributions of this work are: (1) A mathematical analysis to verify that the implementation of the channels on the hardware will give the same performance as the floating point channels, (2) The addition of AWGN with wide range of SNRs and the special consideration for noise addition to an OFDM signal, and (3) The fast architecture used to implement the emulator on the Hardware. In the next sections, we will discuss the system requirements, and then we will discuss the mathematical analysis used to reduce the number of mathematical operations on hardware. We will then discuss the hardware architecture and the methods used to reduce the number of slices and to increase the emulator speed to meet the requirements of the 802.11n channel models.

II. System requirements

When we designed our channel emulator, our target that it will be used to test transceivers that are 802.11n compliant [1]. The channel emulator should have the capability to apply the 802.11n channel models [2] to the transmitted signal. Channel 'F' is the most challenging channel because it has the largest Maximum Delay Spread. Its RMS delay spread is 150 ns and its maximum delay spread is 1050 ns. The maximum delay spread of channel F exceeds the guard interval size of the transmitted packets, which causes Inter Symbol Interference (ISI). Our design approach was to design the channel emulator in a way that is independent on the receiver design; that means any simplifications of the channel models as in [3] will not be considered. Two testing criteria were used to verify the perfect matching between the emulated channels and the floating point standard channels:

1-A mathematical proof that the emulator will work under the required SNR range.

2-The Packet Error Rate (PER) performance when using the emulator when have 12 bits inputs and outputs versus the PER performance the same input and output but when we apply the standard floating point channel: we need perfect matching in performance.

The main challenges in the FPGA implementation of the channel emulator are:

1-The quantization noise because of the use of fixed point processing inside the FPGA

2-The limited number of multipliers inside the FPGA, and the limited number of BRAMs and slices

3-The implementation should be on Virtex-II FPGA

III. System Design

A. Top level design

The Emulator top modules are shown in Fig. 1 . The channel generation block is responsible for generation of the

instantaneous channel coefficients to be convoluted with the input signal. The channel coefficients should be updated at rate that is greater than $100 f_d$, where f_d is the channel Doppler frequency. The convolution block makes convolution between the instantaneous channel coefficients and the input signal and outputs a signal with rate of 20 MHz. The noise generation block generates AWGN signal to be added to the output of the convolution. The SNR adjustment block scales the convolution output and the AWGN signal and adds them with the correct SNR.

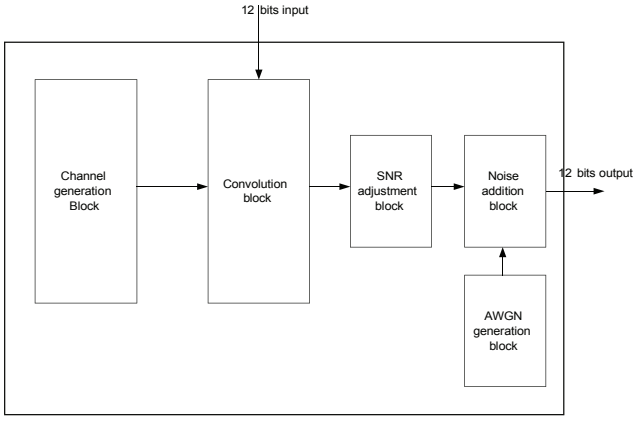

Fig.1. Basic components of the channel emulator

B. Software/Hardware partitioning

The rates of operation of the different modules are shown in Table 1.

Table 1: rates of operation of various blocks

Module	Rate of operation
Channel Generation block	30kHz
Convolution block	20MHz
Noise Generation block	20MHz
SNR adjustment block	20MHz
Noise addition block	20MHz

The channel generation block is the most flexible block in the design. According to [2], the channel generation block should have the following degrees of freedom:

1-It should support a wide range of Doppler frequencies. In our emulator our goal is to support up to 300 Hz

2-It should support multiple Doppler spectra

3-It should support any correlation matrix between the received paths

Fig. 2 shows the components inside the Channel Generation block. The filtering operations in the Doppler spectrum and the correlation matrix application will take a large number of complex multipliers. For example, for a 4x4 system with 18 multipaths between the transmitter and receiver (similar to channel 'F'), the correlation matrix size will be 288 x 288 and will have 2448 non-zero complex numbers. The number of multiplications required to generate one instant of channel coefficients in that example is 2448 complex multiplications. On the other hand, the Doppler filter will also take a lot of resources. Assuming 10 tap FIR filter for Doppler filter, the number of multiplication in a 4x4 system with 18 multipaths requires 2880 complex multiplications. Although the number of multiplications is very big, it is not required to be done with a high rate. So, the Channel Generation block will be implemented on the PC, and the channel coefficients will be transferred through the PCI bus to the FPGA. The rest of the blocks will be implemented on the FPGA.

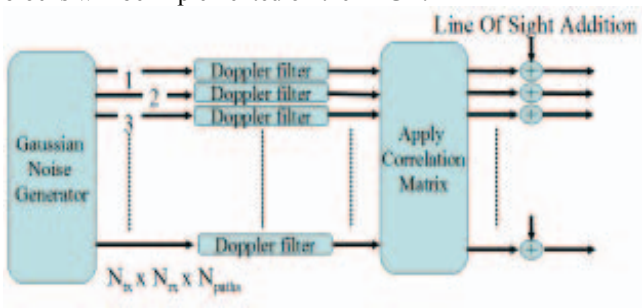

Fig.2 channel generation block

C. Channel generation block

The output of the 802.11n transmitter has a bandwidth of 20 MHz, so; according to the sampling theory; it's sufficient to have a channel model that has paths of a resolution of 50 ns. The conventional way to emulate the channel in the baseband is:

1-Generate a channel instance with a resolution of 10 ns

2-Interpolate the samples output from the transmitter to have a resolution of 10 ns instead of 50 ns using an interpolation filter

3-Convolve the channel instance with the transmitted signal

4-Downsample the result of the convolution back to have a resolution of 50ns

The problem with this is that the interpolation step for the transmitted signal needs a fast processing. To meet the 802.11n system requirements, a 151 FIR interpolation tap is required to interpolate the samples. The filter needs to run at 100MHz rate, which is the rate of the upsampled input signal. For a 4x4 system , this will need 240 multiplication in 100MHz rate. Moreover, the convolution between the paths and the interpolated signal when an 18 paths channel is being emulated requires about 864 multiplications at 100MHz rate.

Our technique to do it is to use the fact that the transmitted signal has a bandwidth of 20MHz, so we don't need to resolve the paths in a resolution of 10 ns.

The new technique is:

1-Generate a channel instance with a resolution of 10 ns

2-Filter the channel with the FIR Decimation filter

3-Downsample the channel to have a resolution of 50 ns

4-Send the coefficients to the FPGA

5-Convolve the channel with the input signal

The main problem in this technique is that the Decimation filter should be long enough in the time domain to be flat in

the passband of the transmitted signal in the frequency domain. The bandwidth of the transmitted 802.11n signal is 18MHz, so the filter should be flat in the range from -9MHz to 9MHz. To have a sharp filter, we used a 151 taps decimation filter. The problem is that after filtering the channel and downsampling, the number of taps of the result of convolution will be bigger than the original number of taps. For example, when we use the 151 taps decimation filter with channel 'F' and then downsampling will lead to a channel of size = (150+105)/5+1 = 52 coefficients. But the advantage is that the convolution rate will be at 20MHz, not 100MHz.

Another advantage of the pervious technique is that we can reduce the number of coefficients more by considering only the large energy paths. The Power Delay Profile (PDP) of channel 'F' after the passing by the Downsampling filter is shown in Fig. 3. The power of the paths at the beginning and the end of the PDP are much lower than the rest of the paths. But to decide if the paths that will be removed, we need to be sure that it will not affect the performance of the system.

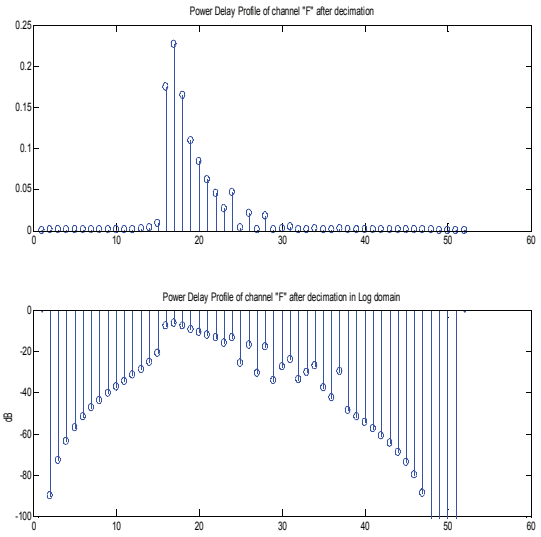

Fig.3 : power delay profile of channel F

We have developed a mathematical model (shown in Fig. 4) that will be used to determine the paths that can be neglected without affecting the performance of the system and will be used to determine the quantization used for the channel coefficients.

Where $x(n)$ is the input signal to the channel emulator, $h(n)$ is the correct channel impulse response, $h_{app}(n)$ is the channel that will be used in the emulation process, which is after rejecting some paths of $h(n)$ and quantization of the rest of the coefficients. We will call it the approximated channel. $h_e(n)$ is the error between the correct channel and the approximated channel. $y_c(n)$ is the ideal correct output after the convolution between the input and the channel. $y_e(n)$ is the output after the convolution between the input and the error channel. $n(n)$ noise added to the output of the convolution block with certain signal to noise ratio and $n_t(n)$ is the total actual noise as a result of adding the correct noise to the error signal. To add the correct value of noise, we need the error output to be significantly smaller than the AWGN added noise. A factor of 10 is acceptable between the added noise and the error signal so that the final noise is dominant by the added noise, not the error due to path removal and quantization. Using our mathematical model, it can be easily shown that the condition required to satisfy a factor of 10 between the added noise and the error signal is:

$$\frac{E_{h_e}}{E_h} < \frac{10}{SNR}$$

Where E_{h_e} is the total energy of the paths of the error channel due to quantization and path removal, E_h is the energy of the correct channel and *SNR* is maximum target signal to noise ratio that can be applied by the channel emulator.

The error channel has two components: the quantization noise component and the small energy paths removal component. For the quantization noise, the instantaneous channel coefficients are having a Gaussian distribution, and we want to determine the number of bits required to encode these coefficients. Given certain number of bits, we can determine the energy of the error channel using the rate distortion theory [4]. The rate distortion theory states that for a Gaussian Random Variable, we can encode the random variables in R number of bits, and that will give the following distortion:

$$D = \sigma_x^2 4^{-R}$$

where σ_x is the power of the random variable and D is the power of the error signal.

In case of the most challenging channel in the 802.11n channel models (channel 'F'), we can keep only 33 paths of the downsampled channel in Fig. 3 and quantize the channel paths using 15 bits to secure that we can simulate the system to a signal to noise ratio of up to 35 dBs.

D. Convolution and noise generation blocks

We have designed the convolution block on the FPGA using dedicated multipliers on the FPGA. It makes convolution between the input signal and a 33 tap channel passed from the PC to the FPGA. The convolution block applies the following equation:

$$y(n) = \sum_{m=0}^{m=32} x(m)\,h(n-m)$$

The input and the channel coefficients are complex numbers. We use only three real multipliers to multiply any two complex numbers.

$$y = (x_r + i\,x_i)(h_r + i\,h_i)$$

$$y = (x_r(h_r + h_i) - (x_r + x_i)h_i) + i\,(x_r(h_r + h_i) + (x_i - x_r)h_r)$$

For the most efficient use of the dedicated multipliers on the

FPGA, we run them with their full speed on the FPGA and they can be timeshared with a factor of 11. The input signal has a 20MHz bandwidth and we can run the multipliers at 220MHz. This way, the total number of multipliers to implement the convolution between a single Tx-Rx branch will be only 9 multipliers.

We can only achieve the maximum speed of operations of multipliers by using the technique used in [5]. We have constrained the synthesis of the design to put the channel coefficients in the closest BRAM to the multiplier that will be used in multiplication; moreover, we have constrained the accumulation block to be close to the BRAM and the multiplier. The layout of the basic cell that will be used in the multiplication is shown in Fig.4 . For our design; the input signal is quantized to 12 bits, the channel is encoded into 16 bits, so the output of the multiplication will be 28 bits. The accumulation of 28 bits will be slower than the required 220MHz of the multipliers. To solve that problem, we are using two accumulators : Acc_{low} and Acc_{high}

The Acc_{low} accumulate the output of the least significant bits [b0:b13], and the Acc_{high} accumulates the rest of the bits. After accumulating the 11 results of multiplication, the accumulated values are latched to latches $Acc_{lowLatch}$ and $Acc_{highLatch}$ and then added together after shifting the $Acc_{highLatch}$ by 14 bits to be aligned with $Acc_{lowLatch}$

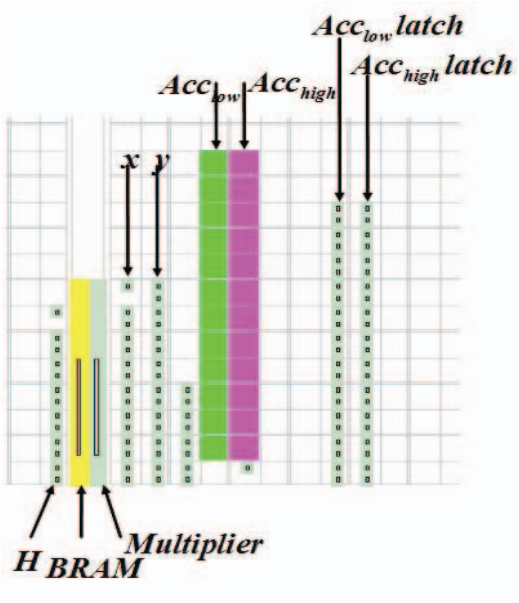

Fig.4 Basic cell

Wallace method was used to generate the AWGN, and the implementation is similar to the implementation in [6]. The main advantage of the Wallace method compared to Box-Miller method [7] is in the implementation complexity. The Wallace method uses no multiplication at all as described in [6]. For an implementation of an *N*x4 system on a Virtex-II XC2V6000-6 FPGA, 4 samples for the real part and 4 samples of the imaginary part are needed at a rate of 20MHz. The overall design needed 2 instants of the AWGN generation block; one for the real part and one for the imaginary part of the complex AWGN, each running at speed of 80MHz and generating 4 samples at a rate of 20MHz.

E. MIMO consideration

The basic cell in Fig.4 is used to generate the terms for our system. The layout of the basic cell that is used for a 1x1 with 33 paths is shown in Fig. 4. It consists of 9 instances of the basic cell in Fig.3 . For an *N*x*M* system, we repeat the same basic cell *NM* times.

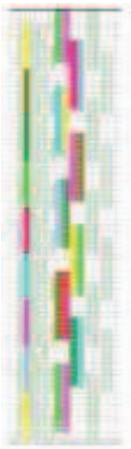

Fig. 5. Basic 33 paths cell

IV. Implementation Results And Verification

The channel emulator has been implemented on hardware on a Virtex-II xc2v6000-4, which is a low speed grade Xilinx FPGA and has been synthesized and simulated on xc2v6000-6 and xc2v8000-5. The resources used to implement the system in the different FPGAs is shown in Table 2

Table 2. Resources used in the hardware implementation of the channel emulator

xc2v	6000-4	6000-6	8000-5
Configuration	**3x4**	**3x4**	**4x4**
Multipliers	**108/144**	**108/144**	**144/168**
Block RAMs	**126/144**	**126/144**	**162/168**
Slices	**21k/33k**	**21k/33k**	**26k/46k**
Max In. Speed	**10MHz**	**20MHz**	**15MHz**

For each basic cell in Fig. 5 we are using 9 BRAMs and 9 Multipliers. Each instance of the AWGN uses 2 BRAMs and no multipliers. The rest of BRAMs are used for the communication between the PC and the FPGA. The bandwidth of the input signal is determined by the speed grade of the FPGA. The maximum number of TxRx branches is limited by the BRAM count of the system.

To test and verify our design, we have first verified the AWGN block. We have transferred 5e5 samples of the AWGN noise output to the PC and used them as the source of noise in a BPSK AWGN simulation and compared it with the results when using floating point AWGN using Box-Miller

method. Fig.6 shows exact matching between both BER curves. We have also applied the Chi-square test for the samples and passed the test in [6].

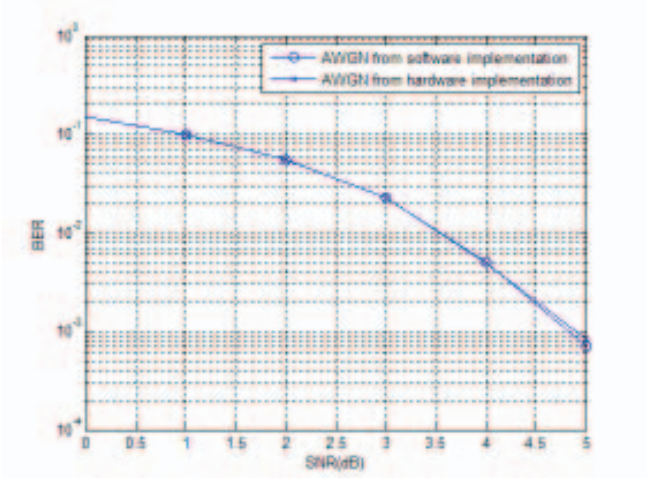

Fig.6. BER for AWGN generated using software and hardware

To test the convolution block, we have used sample by sample matching test. We have sent an OFDM signal on the PC, and convolve it with the quantized channel on the PC and store the results, and then we sent the coefficients to the FPGA and made the convolution there and send the result of the convolution to the PC. We then compared the results of the convolution inside the FPGA and on the PC. We have verified that we got exact matching between the two outputs.

To test the overall system, we have tested the emulator as hardware in the loop; so the hardware emulator uses quantized samples of an 802.11n software transmitter as its input, and sends its output back to PC as an input to the software receiver. We have tested our in a lot of channels and we got exact matching each time. For example, in the case of Rayleigh flat fading channel with MCS6, we got the results in Fig. 7.

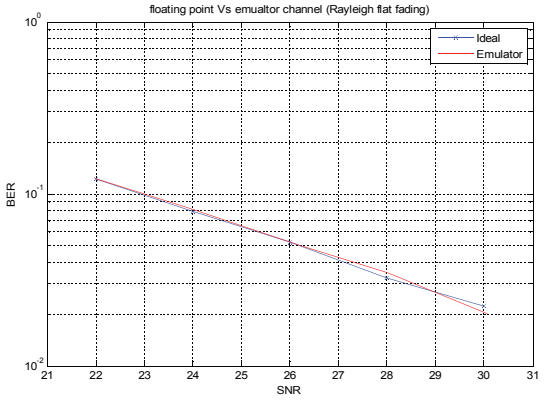

Fig.7 BER using the channel emulator versus using floating point channel model and noise

Another example is testing the channel in a 3 ray channel model with gains {1,1/2 and 1/4}. We have got the results in Fig. 7. From the results above, we have an exact matching between the performance of the channel emulator and the floating point channel in the SNR under consideration.

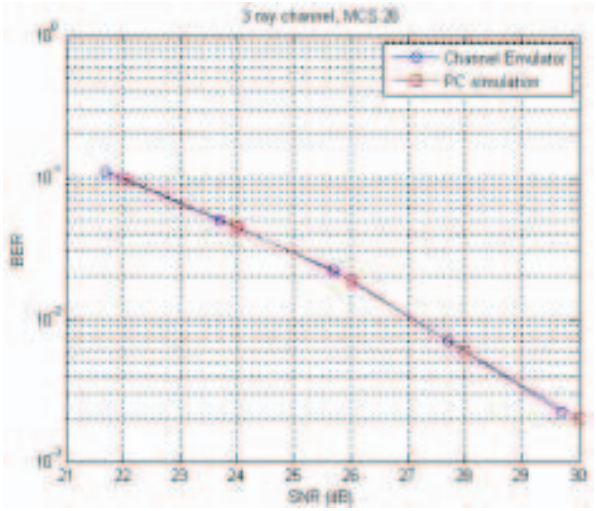

Fig.7 BER curves in a 3 tap channel

V. Conclusion

A MIMO channel emulator was designed and implemented on a Virtex-II FPGA and integrated with a hardware 802.11n transmitter and receiver. A mathematical model was used to verify the matching between the emulated channel models and the 802.11n channel models. A fast implementation on the FPGA was used to meet the speed requirements of the 802.11n standard. A noise generator was used to add AWGN with different SNRs. We have tested our emulator in a hardware in the loop setup and we got exact matching between the emulated system and the floating point system.

VI. References

[1] **Enhanced Wireless Consortium (EWC), "HT PHY Specification," Interoperability PHY Specification v1.01, Oct. 2005**

[2]**V. Erceg et al., "TGn channel models," IEEE P802.11, 802.11-03/940r2, January 2004.**

[3] **Mehlfuhrer, C.; Rupp, M.; Kaltenberger, F.; Humer, G., "A scalable rapid prototyping system for real-time MIMO OFDM transmissions," DSPenabledRadio, 2005. The 2nd IEE/EURASIP Conference on (Ref. No. 2005/11086) , vol., no., pp. 7 pp.-, 19-20 Sept. 2005**

[4]**Cover, T.M. and Thomas, J.A. 1991. Elements of Information Theory. John Wiley and Sons**

[5] **XAPP636 - Optimal Pipelining of the I/O Ports of the Virtex-II Multiplier application note**

[6] **Dong-U Lee; Luk, W.; Villasenor, J.D.; Guanglie Zhang; Leong, P.H.W., "A hardware Gaussian noise generator using the Wallace method," Very Large Scale Integration (VLSI) Systems, IEEE Transactions on , vol.13, no.8, pp. 911-920, Aug. 2005**

[7]Lee, D.-U.; Luk, W.; Villasenor, J.D.; Cheung, P.Y.K., "A Gaussian noise generator for hardware-based simulations," Computers, IEEE Transactions on , vol.53, no.12, pp. 1523-1534, Dec. 2004