Towards FPGA Emulation of Fiber-Optic Channels for Deep-BER Evaluation of DSP Implementations

Erik Börjeson, Christoffer Fougstedt, and Per Larsson-Edefors
Dept. of Computer Science and Engineering,
Chalmers University of Technology, SE-412 96 Göteborg, Sweden
erikbor@chalmers.se

Abstract: We introduce an FPGA-based fiber-optic channel emulator, including both AWGN and carrier phase noise, which can be used to perform deep-BER simulations of DSP implementations and accurately evaluate DSP implementation penalties.
OCIS codes: (060.0060) Fiber optics and optical communication; (060.1660) Coherent communications

1. Introduction

Efficient implementation of digital signal processing (DSP) algorithms is critical to the advancement of high-speed fiber-optic communication systems. However, as these systems become more complex, the effort spent on test and characterization of the implementation can become prohibitively large [1]. For designs in which deep bit-error rate (BER) results are needed, e.g., forward error correction (FEC), the time spent on testing will increase further, since simulations of up to 10^{16} transmitted bits are not uncommon [2]. An additional methodological challenge is that algorithms implemented in hardware description languages (HDLs) likely differ from ideal floating-point MATLAB or C implementations, since the mapping of algorithms to fixed-point, sequential hardware creates implementation artifacts that are not only hard to anticipate, but also hard to model in hardware-agnostic environments like MATLAB [3].

A method commonly used to test hardware implementations is MATLAB-HDL co-simulation, where input data are generated in MATLAB, simulated using a software model of the hardware, and imported back into MATLAB for analysis. Although accurate, this method is very slow. To speed up simulation and enable deep-BER analysis, field-programmable gate arrays (FPGAs) have become indispensable, especially for FECs [2]. The additive white Gaussian noise (AWGN) source traditionally in focus is clearly necessary when modelling a realistic channel [4]. However, for rigorous evaluations of DSP implementations, we need to digitally emulate other end-to-end fiber channel impairments. We here present an FPGA-based emulator for AWGN and carrier phase noise (CPN), which enables practical deep-BER simulations of carrier phase estimation, such as our previous blind phase search (BPS) implementation [5].

2. FPGA-Based Emulation System

Since FPGAs have a limited amount of hardware resources, it is imperative to limit the complexity of the AWGN+CPN emulator, both to allow for additional DSP functionality and to later be able to add other impairments to the emulation system. A block diagram of the FPGA implementation is shown in Fig. 1, where the first block is used to generate the binary data to be transmitted through the emulated channel. The data generation uses an xorshift random number generator (RNG), composed of linear-feedback shift registers, to generate a pseudo-random bit sequence (PRBS) [6]. The RNG has an adjustable internal word length of k bits, resulting in a periodicity of $2^k - 1$ bits. (For our simulations we chose $k = 64$.) To increase throughput, the components following the RNG are parallelized in P lanes, and the PRBS is split to these lanes and modulated using a look-up table (LUT) based approach. The modulated signal is fed to a block, which is emulating an AWGN channel by using a Gaussian noise generator (GNG) IP Core, based on [7], with tails extending to $\pm 9.1\sigma$.

The phase noise present in a fiber-optic channel can be modelled as a Wiener process, $\theta_i = \theta_{i-1} + \Delta_i$, where θ_i is the phase of the ith sample and Δ_i is a normally distributed random variable. The distribution of Δ_i has a zero mean and a variance, $\sigma^2_{\Delta_i} = 2\pi\Delta f T_s$, where Δf is the combined phase noise of the carrier and local oscillator lasers and T_s is the symbol period. Our CPN generator implementation, shown in Fig. 2, uses the same GNG as the AWGN generator described above, to generate Δ_i. The output from the GNG is scaled by $2\pi\Delta f T_s$, before being added to the θ_{i-1} signal. A LUT is used to convert Δ_i to a complex value, which is multiplied with the IQ input. The DSP block under test is

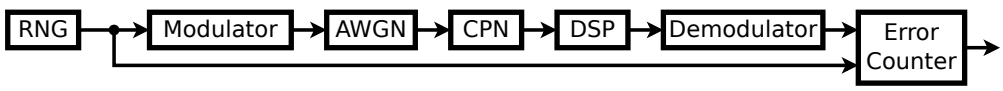

Fig. 1: Block diagram of the implemented system.

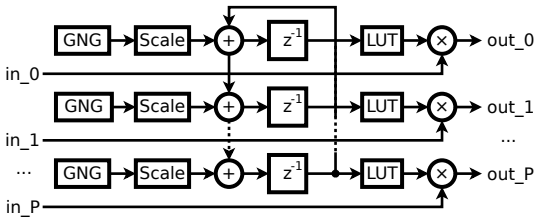

Fig. 2: Block diagram of the CPN generator.

	KC705 [8]	VC709 [8]	VCU110 [8]
LUTs	22,043 (10.8%)	22,043 (5.0%)	21,697 (2.0%)
BRAMs	12 (2.7%)	12 (0.8%)	12 (0.3%)
DSPs	48 (5.7%)	48 (1.3%)	48 (2.7%)

Table 1: FPGA resources used for the fiber-channel emulator; given as *total number of blocks used (proportion of available blocks used)*.

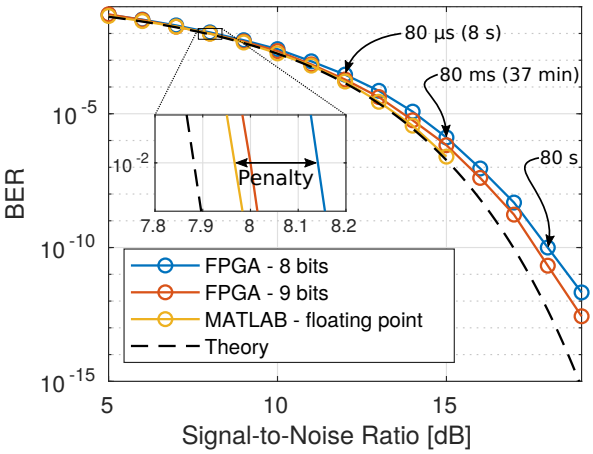

Fig. 3: BER for 16QAM BPS, showing implementation penalty and simulation runtime as *FPGA (MATLAB-HDL)*.

connected after the CPN generator, and the output from this block is fed to a demodulator. The resulting bit sequence is compared to the output of the RNG and the number of errors is calculated.

3. Simulation Results and FPGA Utilization

First the AWGN+CPN emulator, without any DSP blocks, was synthesized from an HDL implementation using Xilinx Vivado for three different development boards, featuring three Xilinx FPGAs representing very different capability/cost tradeoffs. The resource usage and utilization for a 16QAM AWGN+CPN emulator, parallelized in four lanes, with a signal word length of 8 bits, are shown in Table 1. We use a clock rate of 80 MHz for all three FPGAs, which corresponds to a bit rate of 1.28 Gbps. The bottleneck of the design is the parallelization of the CPN generator, which currently does not allow for more than four lanes due to the feedback loop. The utilization is, however, low for all of the boards, which opens up for the possibility to run multiple instances of the emulation system simultaneously. Such an approach could increase the bit rate by a factor of 10, depending on the size of the DSP block under test and the type of FPGA used.

Next, an instance of our BPS implementation [5] was inserted as the DSP block shown in Fig.1 and a new synthesis was carried out. The results for a 16QAM implementation with 8-bit word length and 16 test phases are shown in Fig. 3. The penalty of the HDL implementation compared to a floating point one is less than 0.2 dB at BER $= 10^{-2}$, for a linewidth of 20 kHz and a simulated symbol rate of 20 GBd, which is consistent with our previous findings. The speed up of using FPGA emulation, compared to MATLAB-HDL co-simulation, is at least 5 orders of magnitude.

4. Conclusion

We have demonstrated FPGA-based emulation of two important channel properties, viz. AWGN and CPN, to reduce the runtime of HDL simulations of DSP implementations. The method takes the effect of algorithm mapping to fixed-point, sequential hardware into account, creating opportunities to test and characterize hardware implementations of DSP algorithms at deep BERs and to identify implementation penalties.

Acknowledgement: This work was financially supported by the Knut and Alice Wallenberg Foundation and Vinnova.

References

[1] W. Haas, "Comparison of current FPGA technology: Case study implementing FEC for the 100G optical transport network," in "ECOC," (2009), p. 4.4.4.

[2] Y. Cai *et al.*, "FPGA investigation on error-floor performance of a concatenated staircase and Hamming code for 400G-ZR forward error correction," in "OFC," (2018), p. Th4C.2.

[3] D. Cardenas *et al.*, "Fixed point and power consumption analysis of a coherent receiver for optical access networks implemented in FPGA," in "ECOC," (2013), p. Mo.3.C.4.

[4] Z. Zhang *et al.*, "Investigation of error floors of structured low-density parity-check codes by hardware emulation," in "IEEE Globecom 2006," (2006), p. GEN03-6.

[5] E. Börjeson *et al.*, "ASIC design exploration of phase recovery algorithms for M-QAM fiber-optic systems," in "OFC," (2019), p. W3H.7.

[6] G. Marsaglia, "Xorshift RNGs," J. Stat. Softw. **8**, 1–6 (2003).

[7] R. Gutierrez *et al.*, "Hardware architecture of a Gaussian noise generator based on the inversion method," TCAS II **59**, 501–505 (2012).

[8] Xilinx Inc., *Boards, Kits and Modules* (2019). https://www.xilinx.com/products/boards-and-kits.html.