Accelerating the Verification of Forward Error Correction Decoders by PCIe FPGA Cards

Daniel Suárez, Víctor Fernández, Héctor Posadas, Pablo Sánchez

***Abstract*—Pre-silicon forward error correction (FEC) decoding hardware is typically designed using hardware description languages (HDL). Its verification is a hard task due to its intrinsic tendency to correct errors. The generation and injection of millions of random inputs as well as the cross-checking of the corresponding outputs is highly recommended. Using HDL simulations for such work leads to prohibitive execution times. This letter proposes a verification strategy in which the software testbed is executed on a multi-core host and the hardware under verification is prototyped on a PCIe accelerator card. Data are transferred in big blocks of codewords over a high-bandwidth PCIe channel and applied to the decoder using a pipeline management to maximize the use of computational resources and to minimize the verification time. The decoder is replicated with parallel access to DDRs. OpenMP is used to leverage the parallel capabilities of the host and OpenCL, together with Xilinx Runtime Library (XRT), to manage the PCIe FPGA card execution. The results show an important speed-up with respect to HDL simulation and to other prototyping approaches.**

***Index Terms*—Verification, Data Center Alveo Cards, FPGA Acceleration, Prototyping, Emulation, BER/CER testing.**

I. Introduction

FORWARD error correction (FEC) components are widely used in modern communications distributed systems. The Viterbi, Reed-Solomon, Bose–Chaudhuri–Hocquenghem (BCH), low density parity check (LPDC) and Polar codes, among others, are essential components in charge of reducing error rates (or the power required to get the same error rate) caused by channel noise. New standards promote the use of these modules with new higher requirements, which leads to novel hardware implementations.

The verification of pre-silicon FEC decoders is a challenging task. In the initial stages of the design/correction iteration process, HDL simulations are used due to their wide debugging capabilities. However, decoders have, inherently, a tendency to hide errors. When the design evolves and becomes more stable, the detection of last errors is very difficult. A validation test with a massive number of random inputs is highly recommended blueat this point. To accomplish this, two complementary methods are proposed in this letter: to compare the decoder outputs to the ones produced by a golden model (a mismatch is detected when outputs are different), if it is available, and to obtain the bit or codeword error rate (BER/CER) performance metric, by comparing the outputs with the ones generated by the coder (an error is computed when the output of the decoder is different than the one generated by the coder). Proposed computations demand a big number of inputs [1], making HDL simulation extremely slow.

In pre-silicon phases, emulation and FPGA prototyping are the two main verification alternatives to HDL simulation. In recent years, emulation tools have evolved into very powerful programmable computing infrastructures [2], [3]. They are able to emulate very complex hardware/software (HW/SW) SoC systems with high degree of debugging capabilities. Due to those characteristics, they are, typically, slower than ad-hoc FPGA prototypes [4], like the one presented in this letter. In addition, they are much more expensive.

Concerning FPGA prototyping [5] and focusing on FEC decoders, here are some previous relevant contributions. In [6], predefined library components are used to verify the performance of a communication system. However, this is not useful for new designs that cannot be modeled based on those components. Some approaches use simple, poorly accurate, hardware random generators [7] or develop complex ad-hoc hardware implementations [8], [9]. One of the most critical aspects is that random values generated as coder inputs or as channel noise have to meet stringent distributions, which implementation in hardware can be tricky. In [10], a hardware/sofware approach is presented with a Xilinx microblaze device integrated into an FPGA with the task of debugging an HDL version of a FEC decoder. The approach is focused on improving the debugging capabilities. Speed is not a target, and the generation of random data is performed in hardware with a simple linear feedback shift register (LFSR).

In [11], Xilinx SoC FPGAs were used in an integrated hardware/software solution for FEC decoding verification. The decoder was integrated in the Programmable Logic (PL) part of the FPGA. The rest of the testbed (pattern generation and output checking) was programmed on the Processing System (PS) part as software. The verification process is accelerated from tens of days (with HDL simulation) to tens of minutes. Processing is done on a codeword by codeword basis, and the execution of software and hardware is sequential.

This letter presents a deep improvement of the verification strategy shown in [11]. The presented approach uses procedures, equipment and tools not used by other authors. It proposes an execution flow in which data is transferred in big blocks of codewords over a high bandwidth channel combined with HW/SW pipeline management to maximize the use of computational resources and minimize the verification time. A PCIe interface and several DDR blocks with large storage capabilities are used to optimize HW/SW communications. A Xilinx Alveo U200 card is used since it provides all these

Manuscript received XX; revised XX...This work has been supported by Project PID2020-116417RB-C43, funded by Spanish MCIN/AEI/10.13039/501100011033 and by Project No 101007273 ECSEL DAIS, funded by EU H2020 and by Spanish pci2021-121988.

The authors are with the Microelectronics Engineering Group, Universidad de Cantabria, 39005 Santander, Spain (e-mail: suarezd,victor,posadash,sanchez@teisa.unican.es)

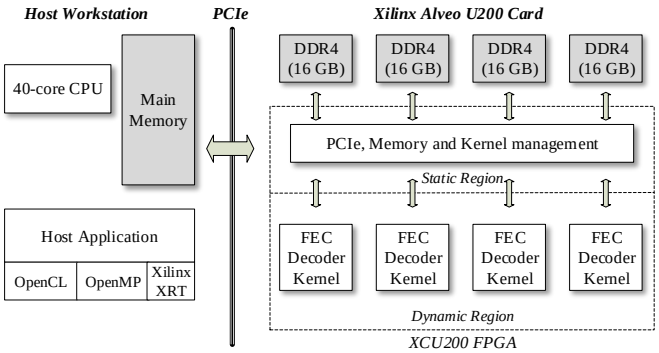

Fig. 1. Executing platform for the verification approach.

competences, but other similar options are applicable. By using an Alveo card and Xilinx Tools, we avoid the need for other specific verification equipment and tools, minimizing the cost and knowledge required to implement the proposed task. Decoding hardware under verification is replicated in the Alveo card, maximizing the parallel use of its DDR memories. Processing in the host is also parallelized by using available cores with OpenMP. Moreover, OpenCL and Xilinx Runtime are used to control the host-Alveo communication, execution and synchronization.

The next section shows all the details of the proposed verification procedure. Section III reports the results of applying the methodology to an LDPC decoder. The conclusions are wrapped up in Section IV.

II. Verification Procedure

The executing platform on which the verification proposal is deployed can be seen in Fig. 1. Xilinx Vivado and Vitis tools [12] make up the design environment for the generation, integration and execution of hardware and software parts.

The verification infrastructure is split into two parts: the testbed software, in charge of decoder input generation and output verification, is implemented as software on the host computer, and the component under verification (the FEC decoder) is implemented in HW.

C code is used for input and expected output generation. The decoder outputs can be tested with two possibilities: they can be compared to expected decoder outputs and/or they can be compared to the outputs generated by the coder in order to compute the CER curve, for a range of E_b/N_0 values in both cases. In the first case, the expected decoder outputs are obtained by executing a software golden model of the decoder. In the second case, the model of the decoder is not necessary (or, maybe, it is not available).

To speed-up software execution, using the multi-core host parallelization capabilities, OpenMP is used. It is a user-friendly language (few modifications are needed in the input generation and output checking code) and is optimum for long-running multi-thread execution on a CPU. In addition, OpenCL and Xilinx XRT facilities are used in order to manage the transfer of inputs and outputs between the main memory and the Alveo Card, as well as to control the execution of the

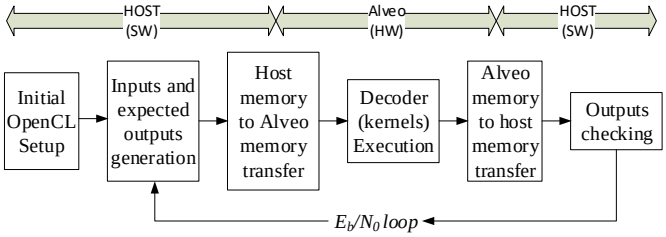

Fig. 2. Global sequential execution flow.

decoder in the hardware part. More details can be found in section II.A.

FEC decoders under consideration decode each input codeword independently. Thus, the total number of input codewords can be split into several blocks to be processed in parallel by several decoders. The Alveo U200 card comprises four independent 16GB DDR memory banks. In order to accelerate the decoding speed, four instances of the decoder under verification are integrated, with parallel access to input codewords (See Fig. 1). The integration of four hardware decoders (which are called kernels in OpenCL terminology) and their control is performed with OpenCL and Xilinx tools as it will be detailed in section II.B. If more speed is needed, more decoders can be integrated in parallel and, in addition, several codewords per memory address can be arranged.

Regarding the debugging capabilities of the presented approach, two features are addressed. In the comparison with the golden model, apart from the primary outputs, other internal values can be considered for comparison. In addition, if any mismatch is produced, the software testbed records the working status to enable its replication by HDL simulation for a more detailed analysis.

A. Execution control

The global flow of execution can be depicted in Fig. 2. It shows the main blocks executed in a, at first, sequential way. Such blocks are detailed hereafter.

An initial stage ("Initial OpenCL Setup" in Fig. 2) is necessary in order to setup the OpenCL configuration. Involved tasks are related to platform, device, context, program, kernels and memory configurations. Moreover, the memory needed for storing the inputs and outputs in the DDR memories of the Alveo card is allocated.

The block in charge of input and expected output generation comprises the communication chain components: random input generation, FEC coder, modulation, noise addition, demodulation and FEC decoder (only for golden model matching).

Transfers between host and accelerator card memories are implemented using proper OpenCL instructions. After the data transfer and before the kernels execution, the number and location of input codewords are transmitted. Kernels are, then, executed. After that, outputs located on Alveo DDR memories are read back from the host. To handle objects like kernels or buffers located on Alveo, it is necessary to use a command queue. In sequential execution flow, the host enqueues commands and waits for the event that indicates the completion of tasks.

Finally, the outputs of the decoders are cross-checked ("Outputs checking" in Fig. 2) with the ones from the coder for CER computation, and/or with the ones from the golden model, software decoder.

The memory for storing inputs and outputs in the host is arranged as follows. Three buffers are allocated (by the *aligned_alloc* instruction): one for the inputs, one for the expected outputs and the last one for the decoder outputs generated on the Alveo card. The three buffers are split into four sections to transfer data from/to the four DDR memories of the Alveo.

In order to speed up the execution, the global flow depicted in Fig. 2, which is sequential, has been modified according to two strategies: software tasks have been parallelized by using OpenMP and hardware/software execution has been pipelined by OpenCL synchronization mechanisms.

For the pipelined flow, the three buffers of inputs and outputs are duplicated in *A* and *B* sets. Buffers are always accessed in ping-pong *ABAB...* order. With this solution, the software part can execute in parallel with the hardware decoding without stopping. At the beginning of the output checking, there is a synchronization point, waiting for hardware outputs. Just after the wait point, the kernel decoding is fired (including memory transfers from host to Alveo and vice versa). The fired hardware execution can even end before the outputs checking is concluded because both processes use different output buffers.

The exposed synchronization mechanism guarantees a correct execution, regardless of the hardware and software delays. For the example used in the results section, the four hardware kernels are fast enough to avoid the waiting for its termination.

Memory buffers are sized at 8 GB (four sections of 2 GB, each). This is compatible with PCIe bandwidth, with DDR sizes and with the host memory size (48GB in total, for the six buffers). Sections of 2GB are transferred from the host to Alveo and vice-versa. When the number of codewords needed implies a greater value than that limit, the E_b/N_0 iteration is split into permissible sizes.

B. Decoder Integration as a kernel

The design under verification is an HDL FEC decoder.The AXI-Stream protocol is used for input reading and output writing as data are used/generated in a sequential fixed order. For this streaming, the decoder needs to access the DDR memories and also obtain an indication of the data location and size. To accomplish this, some wrapping components, provided by Xilinx, have to be included in the kernel architecture.

A control and several data registers are added. The control register is accessed in order to start and finish the decoding process, and the user registers are accessed to indicate the location and number of inputs and outputs. They are reachable via an AXI-Lite interface. Data located in DDR memories is reachable via an AXI4 memory mapped interface. Data transfer between the kernel AXI4 and the decoder AXI-Stream interfaces is accomplished via a FIFO memory.

The kernel structure is compiled with the Xilinx IP Packager and *package_xo* tools to generate a kernel object (*.xo*). The *v++ -l* linking command uses the platform info contained in the XSA Xilinx file and links the kernel objects in order to get the *.xclbin* file which is the binary requested by *clCreateProgramWithBinary* openCL instruction. The linking command is configured by *connectivity.nk* (to use 4 kernels), *connectivity.sp* (to associate the kernels with DDRs) and *connectivity.slr* (to associate the kernels with super logic regions) options.

TABLE I
EXECUTION TIMES FOR [11] AND PROPOSED APPROACHES

Verification Approach	Executing platform	Verification Time
Golden Model Matching (10^5 Matches at Each E_b/N_0)	HDL Simulation	≈ 7 days
	ZCU 102 SoC FPGA Board [11]	20 min
	Host+PCIe Alveo Card	**1.2 min**
CER Computation (E_b/N_0 in [0,6] dB, 1dB step)	HDL Simulation	≈ 80 days
	ZCU 102 SoC FPGA Board [11]	67 min
	Host+PCIe Alveo Card	**3.1 min**

III. RESULTS

The proposed verification method has been applied to the same (128,64) binary LDPC decoder used in [11] in order to compare verification speeds (the same VHDL has been used). The decoder is specified in [13] by the Consultative Committee for Space Data Systems (CCSDS) for Tele-Command (TC) link. The algorithm used for decoding was the Normalized Min Sum and the architecture follows a common partial parallel structure [14]. The proposed methodology can be applied to other types of decoders (Reed Solomon, BCH, Polar, etc.) in the same way.

In terms of random values generated in the host, input coder values (information words) are generated using the SFMT method [15] with a period of $2^{19937} - 1$ values [16]. In addition, the noise model applied to the channel is the classic Additive White Gaussian Noise (AWGN). To include it, random normalized values are generated based on the Ziggurat method, following [17]. Values from both random sources are generated by several parallel threads created with OpenMP. In order to avoid correlated sequences, seeds are generated by using the *shr3_seeded* function. Other generation methods or noise models can be supported without any modification of the presented methodology.

The execution times needed to complete the two proposed verification procedures are reported in Table I. For the golden model matching approach, the checking is accomplished in around one minute. In addition, the computation of the CER curve requires nearly 3 minutes. Both timing results are obtained with the parallel and pipeline flow strategy. For the software tasks executed on the host, 40 threads (for the 40-core host computer) were managed with OpenMP.

Execution profiling is shown in Fig. 3 and Fig. 4. Execution times are computed in the host by the use of *clock_gettime* function. Four main software blocks are considered: OpenCL

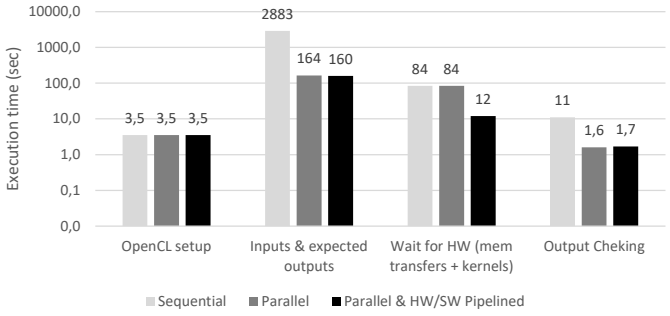

Fig. 3. Execution profiling for CER computation

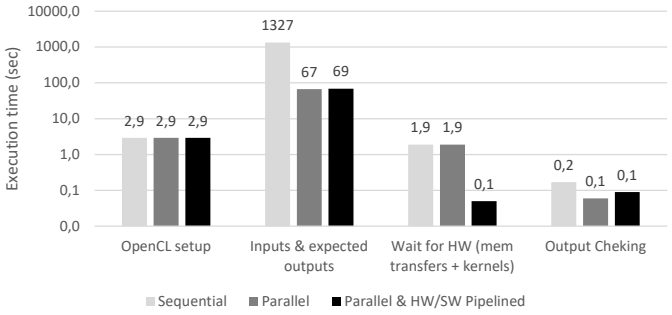

Fig. 4. Execution profiling for output matching with golden model

setup, in/out generation, the wait for hardware and, finally, the output checking. The wait for HW step covers the waiting from the host perspective, including both the time corresponding to the host/Alveo card data transfers and the decoder operation. Purely sequential and parallel & HW/SW pipelined options are compared. An intermediate step, labeled as "Parallel", in which the software is parallelized but the hardware is not pipelined with the software, is also included.

A speed-up is observed in two actions. The software parallelization with OpenMP produces accelerations of more than one order of magnitude (note the logarithmic scale) for in/out generation and output checking. The HW/SW pipelining also decreases the wait for hardware decoding by one order of magnitude. Some residual time is still needed for the decoding of the last set of codewords, which cannot be used for computing the next inputs and outputs.

For the used decoder and the two applied verification approaches, the software waits for zero time (for hardware completion). In CER computation, the software needs a long execution time to generate a massive number of codewords. In golden model comparison, the number of codewords is rather smaller, but the software has to execute the golden decoder.

IV. Conclusion

This letter proposes a verification methodology for FEC decoding modules at the pre-silicon stage, capable of dealing with large amounts of verification data-sets, where HDL simulation is not applicable. The proposed platform and toolset are comprised of a multi-core workstation, a PCIe Xilinx Alveo accelerator card (including an Ultrascale+ XCU200 FPGA and 4 banks of DDR memory) and Xilinx XRT and design tools.

The proposed methodology can be easily applied in similar PCIe acceleration environments. The highly integrated hardware/software infrastructure comprises four copies of the decoder under test, each accessing a dedicated DDR memory, running in the FPGA while all the testbed is executing, with a high level of parallelism, in the host. In addition, the testbed software (in the host) and the hardware design under verification (in the Alveo card) run in a pipelined way for extra acceleration.

Results show that a verification based on millions of random inputs is feasible in just 1-3 minutes, giving the design engineers quick confidence in the designed or acquired decoder IP.

References

[1] M. C. Jeruchim, P. Balaban and K. S. Shanmugan, *Simulation of Communication Systems: Modeling Methodology and Techniques*, Boston, MA, USA:Kluwer, 2000.

[2] Siemens Veloce Strato, Siemens Corporation. Available: https://eda.sw.siemens.com/en-US/ic/veloce/strato-hardware/

[3] Palladium Emulation Platform, Cadence Corporation. Available: https://www.cadence.com/en_US/home/tools/system-design-and-verification/emulationandprototyping/palladium.html

[4] F. Schirrmeister, M. Bershteyn, and R. Turner, "Hardware-Assisted Verification and Software Development," in *Electronic Design Automation for IC System Design, Verification, and Testing*, Boca Raton, FL, USA, CRC Press 2018, ch. 19, pp. 461-489.

[5] Wilson Research Group functional verification study. IC/ASIC functional verification trend report. Available: https://resources.sw.siemens.com/en-US/white-paper-2020-wilson-research-group-functional-verification-study-ic-asic-fucntional-verification-trend-report

[6] V. Singh, A. Root, E. Hemphill, N. Shirazi and J. Hwang, "Accelerating bit error rate testing using a system level design tool," *11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2003. FCCM 2003.*, Napa, CA, USA, 2003, pp. 62-68.

[7] B. Unal, M. S. Hassan, J. Mack, N. Kumbhare and A. Akoglu, "Design of High Throughput FPGA-Based Testbed for Accelerating Error Characterization of LDPC Codes," *2019 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, Cancun, Mexico, 2019, pp. 1-8.

[8] A. Alimohammad and S. F. Fard, "FPGA-based bit error rate performance measurement of wireless systems," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 7, Jul. 2014, pp. 1583-1592.

[9] F. Angarita, V. Torres, A. Pérez-Pascual and J. Valls, "High-throughput FPGA-based emulator for structured LDPC codes," *2012 19th IEEE International Conference on Electronics, Circuits, and Systems (ICECS 2012)*, Seville, 2012, pp. 404-407.

[10] P. Sakellariou, I. Tsatsaragkos, N. Kanistras, A. Mahdi and V. Paliouras, "An FPGA-based prototyping method for verification, characterization and optimization of LDPC error correction systems," *International Conference on Embedded Computer Systems (SAMOS)*, 2012, pp. 286-293.

[11] V. Fernández, C. Abad, Á. Álvarez, Í. Ugarte and P. Sánchez, "Pre-Silicon FEC Decoding Verification on SoC FPGAs," in *IEEE Communications Letters*, vol. 25, no. 1, Jan. 2021, pp. 127-131.

[12] Xilinx Vitis, Xilinx Corporation. Available: https://www.xilinx.com/products/design-tools/vitis/vitis-platform.html

[13] CCSDS 231.0-B-3, "TC Synchronization and Channel Coding", Blue Book, Sept. 2017.

[14] F. Demangel, N. Fau, N. Drabik, F. Charot and C. Wolinski, "A generic architecture of CCSDS Low Density Parity Check decoder for near-earth applications," *2009 Design, Automation & Test in Europe Conference & Exhibition*, Nice, 2009, pp. 1242-1245.

[15] M. Saito and M. Matsumoto, "SIMD-oriented fast Mersenne Twister: a 128-bit pseudorandom number generator," in *Monte Carlo and Quasi-Monte Carlo Methods*, Springer, 2006, pp. 607-622.

[16] M. Matsumoto and T. Nishimura, "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Transactions on Modeling and Computer Simulation*, vol. 8, no. 1, 1998, pp. 3-30.

[17] W. Tsang and G. Marsaglia, "The Ziggurat Method for Generating Random Variables," *Journal of Statistical Software*, vol. 5, no. 8, Oct. 2000.