# Bit-Accurate RTL Simulation of an NQR Spectrometer's SoC-FPGA

Noreddine Kachkachi, Hassan Rabah

# Bit-accurate RTL simulation of an NQR spectrometer's SoC-FPGA

Noreddine Kachkachi
*CNRS*
*CRM2*
F-54000 Nancy, France
noreddine.kachkachi@univ-lorraine.fr

Hassan Rabah
*Université de Lorraine*
*IJL*
F-54000 Nancy, France
hassan.rabah@univ-lorraine.fr

***Abstract*—This paper presents a comprehensive verification environment and methodology for fully verifying a System on Chip-Field-Programmable Gate Array (SoC-FPGA) used in building an NQR spectrometer through RTL simulation. The verification environment includes an RTL testbench, a database of tests, design modules simulation models, and simulation automation scripts. Initially, the system is simulated in an RTL testbench written in System Verilog, with the SoC-FPGA as the Device Under Test (DUT). To simulate real NMR/NQR experiments, various stimuli are created, each corresponding to a specific experiment, comprising current test configuration files and reference files. For each test, the simulation output files of the DUT are compared bit-wise to the reference files. Simulation and check scripts provide a snapshot of the design's non-regression status. This methodology and platform enable in-depth verification of the SoC-FPGA's functionality, significantly reducing the experimental validation time of the NQR spectrometer on real samples. Future enhancements could include verifying the signal denoising algorithms implemented on the SoC-FPGA.**

***Index Terms*—Verification, SoC-FPGA, RTL, Simulation, Reference, Automation, Scripting, Test**

## I. Introduction

The design and verification of complex hardware-software systems using SoC-FPGA platforms present significant challenges due to the complex integration of hardware and software components. SoC-FPGAs are widely used in various applications, from telecommunications to automotive systems, due to their flexibility and performance advantages. The ability to reconfigure hardware allows for rapid prototyping and adaptation to evolving requirements, making SoC-FPGAs a preferred choice for complex system designs. However, the complexity of these systems necessitates a robust verification methodology to ensure functionality, reliability, and performance. Simulation, often performed at the Register Transfer Level (RTL) using HDL languages, allows for detailed examination of the design's behavior under various conditions, thereby reducing the risk of costly errors post-deployment [1].

Hardware-software co-verification techniques are employed to ensure that both components work seamlessly together, including verifying communication protocols, timing constraints, and functional correctness across the hardware-software boundary [1]. Future work is likely to focus on further enhancing verification techniques, including the use of machine learning to predict potential issues and the integration of advanced signal processing algorithms [2].

Verification and simulation are critical in developing NQR/NMR spectrometers, especially for SoC-FPGA platforms, ensuring both hardware and software efficiency under all conditions. These spectrometers integrate analog, digital, and software components, requiring advanced mixed-signal simulation tools. Precise timing is essential; verification ensures pulse sequences are executed correctly and data systems handle throughput without latency. Comprehensive testing is needed to ensure multiple subsystems work correctly together. Verification must consider different configurations/scenarios, ensuring the system can be easily reprogrammed or reconfigured for various experiments without major modifications. Moreover, SoC-FPGA platforms have limited resources; verification ensures efficient use of logic elements, memory, and I/O bandwidth while meeting performance needs.

To address this, we propose a functional verification methodology based on RTL bit accurate simulation, similar to ASIC methodologies, focusing on exhaustive simulation early in the design cycle to anticipate potential issues. The remainder of the paper is structured as follows: Section II provides an overview of the system architecture and verification methodology environment. Section III presents the simulation results for the various modules. Finally, the verification platform is discussed, followed by concluding remarks in Section IV.

## II. Verification methodology and environment

### A. Overview of the NQR system architecture

NQR/NMR spectrometers are highly complex systems (Fig.1) [3]. They rely on RF transmitters and receivers to emit and capture signals interacting with sample nuclei. Data acquisition involves high-speed ADCs and DACs converting analog signals to digital data, facilitating processing and analysis of spectral information (Fig.1.c). Control electronics, powered by FPGAs, manage timing and operations, including pulse sequence generation, data synchronization, signal processing, and real-time parameter adjustments, ensuring precise experimental control and analysis. The embedded software is highly advanced, managing pulse sequence programming for precise RF pulse control and timing (Fig.1.a). The spectrometer also

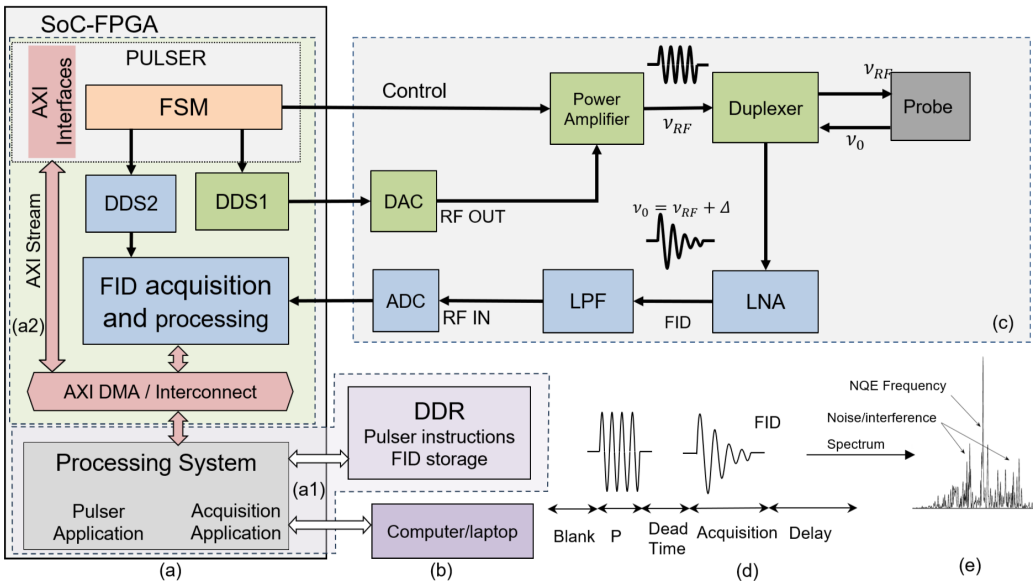


Fig. 1. System architecture of the NMR/NQR Spectrometer, illustrating (a) the SoC-FPGA subsystem with its software (a1) and hardware (a2) components, which represent the design under test (DUT). The rest of the system includes (b) the computer/laptop running the GUI and performing post-processing tasks, (c) the RF front end and analog/digital interface. (d) gives example of an NMR/NQR pulse sequence, and (e) the result of spectrum analysis of the FID.

incorporates complex algorithms for signal processing crucial for enhancing data quality (1.b).

The verification of such highly complex system is not straightforward. In this work, we focus on the verification of the hardware and software embedded on the SoC-FPGA. The other parts of the system are used as elements of test bench. In order to verify and validate in depth the SoC-FPGA during functional RTL simulation a complete verification environment was built. The adopted approach is justified in this context since the targeted application, which is a compact SoC-FPGA based NQR spectrometer, constitutes a complex system that requires a perfect synchronization between several hardware and software modules. In order to guarantee that the system will behave as expected in the real NQR experiment context, a thorough and exhaustive bit accurate simulation of the SoC-FPGA should be performed in an early stage of the development cycle.

### *B. Verification environment description*

The verification platform is composed of an RTL testbench and several elements which are, references generators (C or Matlab executable models) for the modules, tests database containing the tests of different modules and simulation automation scripts. The hierarchical view of this verification environment is depicted in Fig.2.

This platform is composed of the following elements:

*1) Testbench:* the testbench components are:

- DUT: In this environment, the DUT is the SoC-FPGA.
- CPU and AXI Bus Models: The transactions between the different blocks of the SoC-FPGA (CPU, AXI bus, DDR model) are modeled by a verification IP from Xilinx [4].
- AXI Bus traffic generator: This module, obtained from the Redpitaya project [5], generates the transactions and serves to transfer the instructions from the pulser's instructions file to the DDR models.
- DAC IF outputs generator: It is instantiated to verify the acquisition direct detection mode.

*2) Tests database:* In order to simulate a real NMR/NQR experience, several tests were created. Each test is associated to a directory that contains several input files or stimulii that permit the current test configuration and give the input data for the simulation, like the pulser instructions or the input FID. The test directory contains the relevant instructions file adapted to each test scenario that reproduces the tested NQR experiment.

In addition to the instructions file, the test directory contains a model of the input FID to test the acquisition module behavior. For the testbench, the origin of this data can be either a file previously filled with data from a theoretical or even real FID, or the output of the DDS which was recorded during the excitation simulation phase and re-injected as input to the testbench. The file containing the FID model is organized as a text file where each line contains an FID point.

In addition to the stimulii files, that are the instructions and the FID files, the test directory contains a certain number of configuration files, which content may change according to the test scenario. These files are the registers configuration, the mode configuration file and the FIR coefficient file.

Finally, at the output side, each test directory contains the different expected outputs reference files that will be compared at the end of simulation to the outputs of the pulser, the acquisition, the DDS, etc.

*3) Simulation automation scripts:* In order to automate the launch of simulations and the archiving of results, a non-regression verification script was developed to automatically launch the simulation of all tests or a given test and collect the

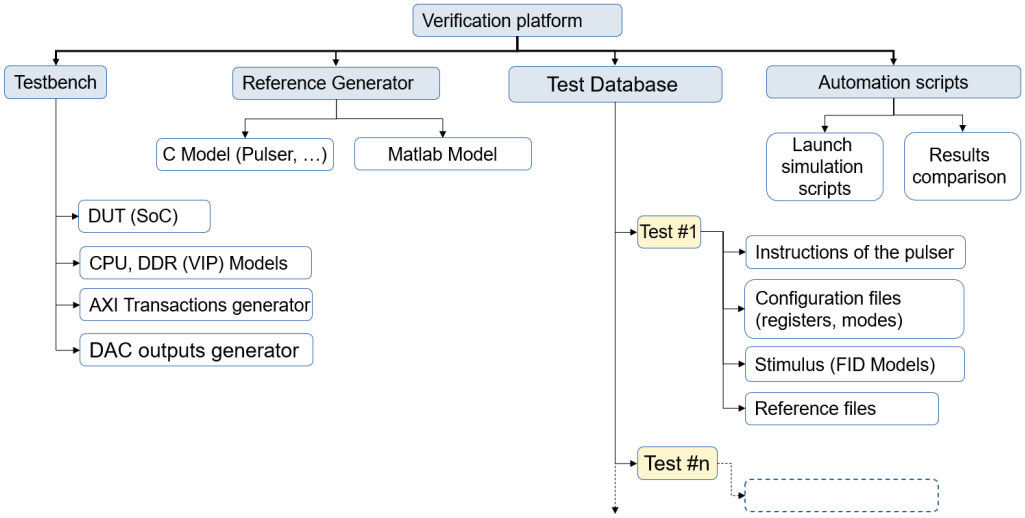


Fig. 2. Hierarchical view of the verification platform

results (non-regression). The role of this non-regression script is to perform the following operations for each test: copy, from the test database, the different files corresponding to the current test, then launch simulation and finally compare, after simulation ends, simulation outputs against reference ones and update the non-regression status file.

### C. Simulation progress

The simulation begins by configuring SoC modules using a register file read by the testbench. Instructions from the pulse generator are copied to DDR via the traffic generator. The DMA write channel of the pulse generator transfers outputs back to DDR. The pulse generator DMA loads 128-bit instructions from DDR. DDS and excitation phases are programmed before pulse generation starts. Outputs are saved for later analysis.

During simulation, the pulser module is verified in real-time by comparing its output to a reference at each clock cycle. FID read and write DMAs are set up before acquisition to prepare for on-the-fly mode. The FID undergoes signal processing stages before being sent to DDR as packets. The testbench determines the size of data to transfer via DMA.

Throughout acquisition, processed FID data and pulse generator outputs are stored in files updated after each scan. At the simulation's end, all scan data and pulser debug information are saved. Debug output data from the pulser is read after the last FID.

### D. Monitoring and output files

At each clock cycle, the pulse programmer output signals are read and concatenated to form an output vector and stored in an output text file. To facilitate organization, the output file has a .dump extension.

Other output files are generated according to the same principle, where there is a storage of digital outputs at each rising clock edge of the different modules of the SOC of the spectrometer (acquisition, excitation, DDS, etc.).

The behavior of the pulse programmer, acquisition and excitation is verified by automatically comparing to the bit level the content of the files obtained at the output of the simulation of each module (with .dump extension) with the content of the reference file (with .exp extension) for the corresponding module.

The reference files are generated before performing the simulation and stored in the test database; they contain the expected data for each module (pulse, acquisition, etc.).

## III. Verification Results

### A. Pulser module verification

In the case of the pulse generator, the exact value as well as the duration or temporal length of each signal at the output of the pulse generator (such as the width of the RF pulse, the duration of the delays, etc.) is verified by automatically comparing all the values taken by this signal throughout the simulation with the expected values. The pulse generator stimulus is the instruction text file. Each line of the FID corresponds to a 128-bit instruction encoded in hexadecimal format. During the simulation and at each clock cycle, the testbench records the 8-bit bus values of the pulse generator outputs in files.

Monitoring and recording of the pulse generator outputs is carried out during all simulations. All of the 8 tests exposed in the following subsection (III-B) also include the verification of the pulse generator, since each of these tests contains pulser output reference files.

### B. Acquisition module verification

The functionality of the acquisition is verified using two methods. The first method, which is rather visual, consists of post-processing the FID detected through a program that was written in Matlab (which reproduces the main NMR/NQR post-processing of the interfacing software) to verify that the spectrum behaves as desired (for example, that it has

the correct $\Delta$ offset). The other method consists of directly comparing, to the bit, the detected FID with the reference FID already calculated before the simulation.

The FID model stimulus is detected and processed by the acquisition pipe. It is then stored in the DDR zone dedicated to writing the FID via the write channel, "Stream to memory", of DMA 1. The testbench reads this memory zone through the read channel, "Memory to stream" , of DMA 2. The read FID is stored in a text file, where each line corresponds to a point of the FID, so that it is compared with the file which contains the reference values.

In order to generate the references, and given that the complete executable or Matlab model for acquisition are not yet available for all tests, we adopted an approach based on visual check of the simulation waveforms, combined with Matlab processing of the resultant FID, and reusing the generated simulation outputs as references for the future simulations.

This method still brings some advantages as it enables time saving since no extensive waveform check is needed after each design change, and it is sufficient enough in terms of precision for verifying the acquisition module.

### C. Transmitter module verification

The testbench programs the frequencies of the two DDS by configuring the SoC DDS dedicated registers, and programs the excitation phase through the pulser outputs. Configuring the amplitude divider register sets excitation pulse amplitude.

Monitoring, as well as recording of DDS outputs, is carried out during all simulations. The 14-bit results of the simulation, which correspond to the RF output of the SoC (just before the DAC) during the simulation, are stored in a text file. Thus, all of the 8 tests (described in III-D2) already include the verification of the DDS and the excitation module, since each of these tests contains reference files specific to the DDS which are compared to the files obtained at the output of the DDS. For the generation of references, the same method is used as explained in the previous subsection (III-B).

### D. Test scenarios and non-regression status

*1) Simulation modes configuration:* To test the SoC-FPGA in the different possible modes, three configuration parameters have been defined which are a parameter to choose the source of the FID model (DDS output or input file), the chosen acquisition mode (signal processing or bypass), and finally the type of accumulation carried out (OTF or SW).

*2) Test scenarios:* Based on the 8 possible values of the three parameters described above, there are at least 8 tests for the simulation of the acquisition (table I) )

*3) non regressions status:* As explained in the section II-B in sub section II-B3, a script for automating the simulation of all the tests and collecting the results (non-regression) was developed. The role of this script is to automate the simulations and the comparison of the results and their archiving in a "status" file (status.log). The status.log file thus constitutes a dashboard which gives, at a given moment, a snapshot (overview) of the status of the design to be checked ( table I).

| Test name | FID mode | Acq. mode | Accu. mode | Status |
|---|---|---|---|---|
| dds_bypass_sw | 0 | 0 | 0 | PASS |
| dds_bypass_otf | 0 | 0 | 1 | PASS |
| dds_ts_sw | 0 | 1 | 0 | PASS |
| dd_ts_otf | 0 | 1 | 1 | PASS |
| fid_bypass_sw | 1 | 0 | 0 | PASS |
| fid_bypass_otf | 1 | 0 | 1 | PASS |
| fid_ts_sw | 1 | 1 | 0 | PASS |
| fid_ts_otf | 1 | 1 | 1 | PASS |

TABLE I
NON REGRESSION STATUS : TEST NAMES CONTAINING "DDS" USE THE DDS OUTPUT AS SOURCE OF THE FID, THE OTHERS USE A THEORETICAL OR REAL FID. TEST NAMES WITH "TS" CORRESPOND TO SIGNAL TREATMENT MODES, THE ONES WITH "BYPASS" CORRESPOND TO BYPASS MODES. TEST NAMES CONTAINING "SW" CORRESPOND TO SOFTWARE ACCUMULATION MODES, THOSE WITH "OTF" CORRESPOND TO FPGA ACCUMULATION MODES.

The need for this automation arises after each modification of the design by adding new options. Indeed, a re-launch of all the tests (non-regression tests) is necessary after each modification in order to ensure in an automatic, exhaustive and rigorous manner that there has been no regression ( failed tests that previously gave PASS) after the design change.

## IV. CONCLUSION AND PERSPECTIVES

In this paper we presented the methodology and the functional verification environment that was adopted and designed, as well as the different components that were developed to build this environment in order to properly verify and validate the architecture and the design of the SoC-FPGA. This methodology is based on rigorous and exhaustive verification through the simulation of different possible scenarios and the bit-by-bit comparison of the results obtained with the references generated beforehand. This methodology and this development, even if they are particularly time-consuming, have proven to be very profitable and effective in terms of detecting failures (bugs) and in particular by helping to shorten the time required to carry out tests on real samples. Furthermore, this environment remains very scalable in order to verify in depth and in a rigorous manner the signal processing and denoising algorithms which can be implemented on FPGA. As a result of having thoroughly verified the SoC by simulation, the experimental validation of the device on real samples was easier and faster.

## REFERENCES


[1] S. Yang and Z. Yu, "A highly integrated hardware/software co-design and co-verification platform," *IEEE Design & Test*, vol. 36, no. 1, pp. 23–30, 2019.

[2] G. Huang, J. Hu, Y. He, J. Liu, M. Ma, Z. Shen, J. Wu, Y. Xu, H. Zhang, K. Zhong, *et al.*, "Machine learning for electronic design automation: A survey," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 26, no. 5, pp. 1–46, 2021.

[3] N. Kachkachi, A. Gansmuller, and H. Rabah, "Design and experimental validation of a SoC-FPGA based compact NQR spectrometer," *IEEE Transactions on Instrumentation and Measurement*, vol. 73, pp. 1–12, 2024.

[4] Xilinx, "Axi verification ip (vip)." https://www.xilinx.com/products/intellectual-property/axi-vip.html.

[5] Redpitaya, "Redpitaya fpga." https://www.redpitaya.com/.