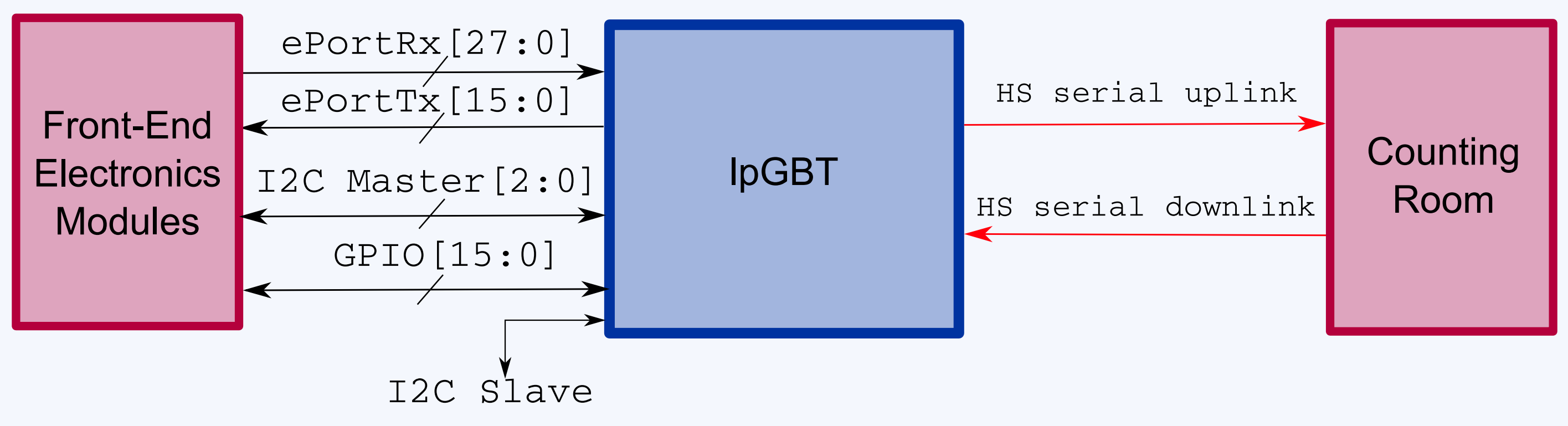
An UVM-based verification environment for the lpGBT 10 Gbps transceiver ASIC

José Fonseca*, Szymon Kulis
CERN, Geneva, Switzerland
(jose.fonseca,szymon.kulis)@cern.ch
*corresponding author

1. Introduction

The lpGBT chip is currently under development to support a common interface for data and clock transmission for future detector systems at the HL-LHC. This ASIC works as transceiver that aggregates many parallel channels (80Mbps to 1.28 Gbps) into a single serial channel, at a much higher data rate (5 or 10 Gbps for uplinks, and 2.56 Gbps for downlinks), as the following figure suggests.

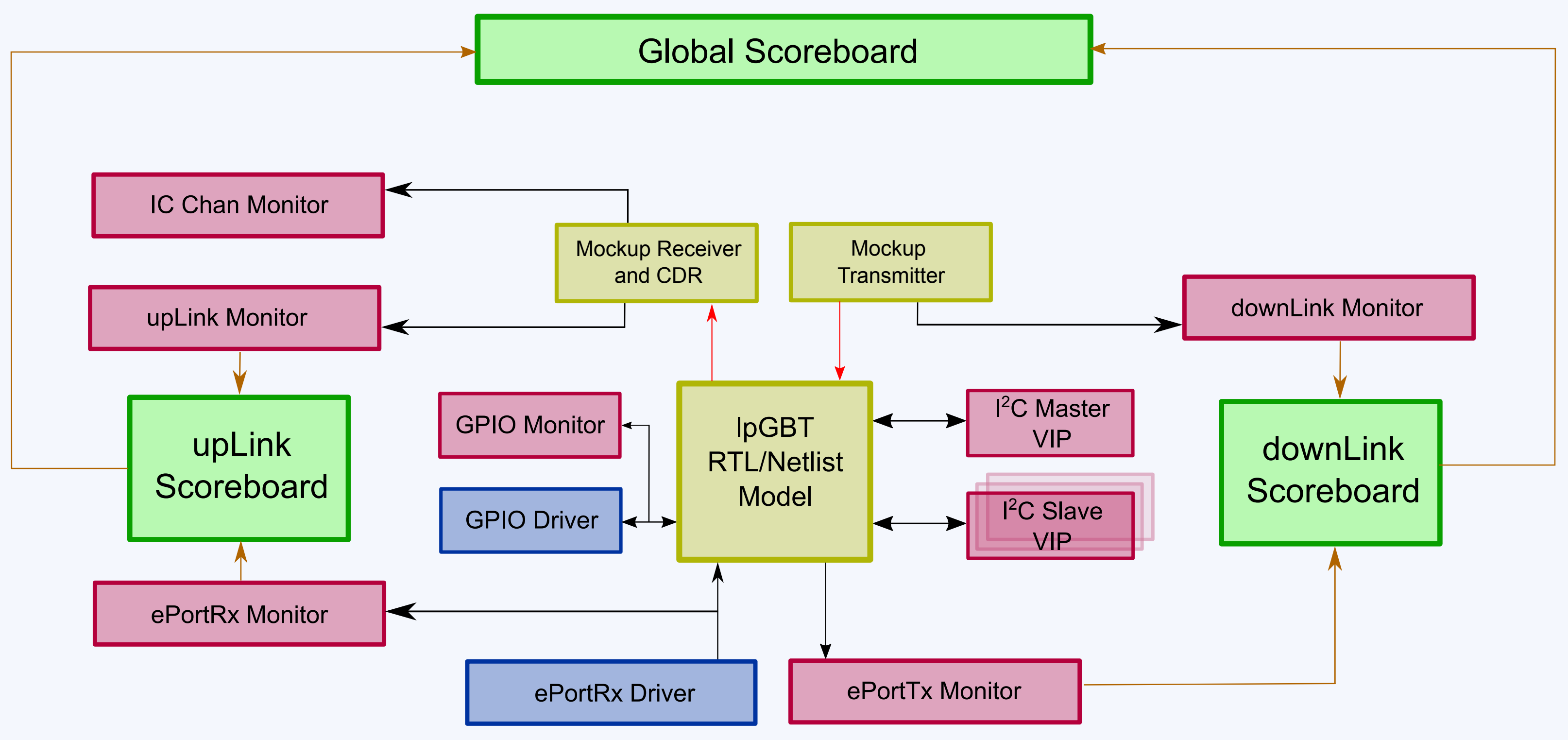


The lpGBT can operate in several modes (Simplex Rx/Tx, Transceiver) at various data rates, with two FEC encoding schemes (enabling the end users to optimize their bandwidth and immunity to Single Event Effects), slow control features (like I2C master/slave interfaces, parallel IO, ADC, DAC) are also included.

Given the nature of the chip, its many interfaces and high level of configurability, the Universal Verification Methodology (UVM) was selected to streamline the verification process of all its features. It allowed to ease the verification of standard interfaces, such as I2C, for which commercially available verification IP could be used. However, given that the lpGBT high-speed interfaces are data and protocol agnostic, with no clear transaction model, some concepts of the UVM methodology had to be adapted. Moreover, given the complexity of clock and data recovery circuits, as well as the complex and multi-stage initialization sequence, the simulation time was very long, which also presented an additional challenge.

In this poster, we start by presenting the overall structure of the testbench, followed by a description of the methodology adopted for each test and the results achieved. The final section presents the verification flow adopted by the team, in terms of how tests are

2. Testbench Structure, Methodology and Results



The figure above shows a simplified view of the testbench structure. A SystemVerilog package holds all the chip configuration values/parameters, which are **randomized for each run (constrained to sensible values using SystemVerilog constraints)**.

2.1 - Many Small Tests vs One Big Test

When a given test is run, the **appropriate set** of Drivers/Monitors are instantiated, so each test focuses on a restricted set of features: **this is preferrable to a single test that tests all functionalities at the same time, allowing parallel execution of tests in a regression pipeline. It proved to be a clearer way of pinpointing of bugs.**

2.2 - Bitstream verification of the Simplex/Transceiver operation

The **scoreboards reconstruct the channel streams** in both directions and perform a **bit-accurate checking**, after automatically locking both streams. Since the chip is agnostic to the data contents, there is no clear transaction model which could be accomodated in the typical UVM methodology. Instead, the monitors send **clusters of bits recovered from the streams**, which are then **reconstructed in the Scoreboards**, and aligned with each other. The **checking is then performed on strips of 5000 bits**, instead of bit-by-bit, which simplifies the coding and improves the efficiency of the testbench execution.

The verification procedure does not rely only on stream checking, but **lower level checks and assertions are included** in the code, which automatically trigger errors and potential hazards that otherwise wouldn't be caught by the Scoreboards. Furthermore, **each run always requires a full chip initialization**.

2.3 - Description of some other tests, and main bugs caught by them

I2C Slave/Master - the Mentor Graphics **Verification IP (VIP)** was used to **formally verify**, and generate stimuli, for the I2C slave port, and the three I2C master ports of the lpGBT. The Master ports were controlled via registers written through the Slave port, where all possible operations were used. Furthermore, a **full register write/read** was performed using the Slave interface. Other than the SDA line in the Slave interface being actively driven (instead of open-drain), no issues were found.

Control Channels (IC/EC) - the register access of the lpGBT is multiplexed between the I2C Slave and a serial protocol (80 Mbps) embedded in the high-speed down/up links. Apart from randomized read/write transactions, full register readout was performed. An issue was found related to bitstuffing problems (which was not found in previous verification approaches).

Built-In Self Test (BIST) features - the lpGBT posseses many PRBS checkers and generators, as well as clock and pattern generators for testing purposes. These can be output/input either through the high-speed serial links or the ePorts. All these functionalities were tested, again in a bit-accurate way. Due to parameter randomization, we were able to isolate a bug in the PRBS generator initialization that would cause it not to work under certain corner cases. Furthermore, it was found to be periodically inserting a wrong bit in its streams in the PRBS31 operation mode.

Power-up State Machine - although each run requires a full PUSM initialization sequence, many of its fail-safe features (like the watchdog and BOD operation) were not exercised, since no fail conditions were reproduced. Hence, we heavily stripped the UVM infrastructue (since no stream checking was performed) to the bear minimum, and **reproduced all failure scenarios**. We discovered some problems in the logic coding that would cause a deadlock in particular corner cases.

GPIOs - the direction of the 16 GPIOs was randomized, as well as the values they should drive/read. No issues were found

2.4 - Runtime metrics and results

Our pipeline consisted of a total of **34 directed tests**, and the batch server allowed 4 parallel thread executions. The tests included a mix of RTL Block tests and Full-chip UVM testbenches, including the **triplicated version** of it. The longest test was the Transceiver +ICChannel test, taking about 21 minutes. The **total runtime for this pipeline was around 33 minutes** (varying from run to run). The average runtime per test was around one minute.

3. Verification Flow and Strategy

Separate tests were created for **separate functionalities** (although some of them were clustered, like Transceiver stream check + IC register access).

Chip configuration (High speed link data rate, ePort data rates, chip address, FEC node...) was randomized for each simulation run, regardless of the functionality under test, thus **increasing the coverage of the tested configuration**.

Tests are added to the **automatic batch testing environment**. This test environment is built on top of the Continued Integration (CI) features provided by the GitLab application, where the **Git repository** that contains our RTL code is hosted. so that the test batch is **run everytime someone pushes changes** to the RTL code. In our flow, all tests are run, **regardless the change is unrelated to the functionality** a given test verifies, everytime someone pushes changes to the RTL code This enabled us to:

Catch new bugs at the source: when the designer commits "poisonous" changes (which sometimes are apparently not directly related with the bug!)

Improves the verification coverage by randomizing the chip's configuration and applied stimulus.

After all tests pass in RTL, **simulations with timing backannotation** are run instead, in an almost **seamless way**.

Each test is then **executed a number of additional times, and some further tests are also added**, (I2C and IC/EC full register writing and readout, for instance) which were omitted from the pipeline to reduce its runtime. This is repeated for as long as possible unitl the submission