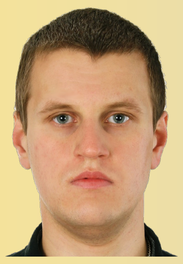
Simulation Environment Based on the Universal Verification Methodology (UVM)



Adrian Fiergolski[1)]
on behalf of the CLIC detector and physics (CLICdp) collaboration
[1)]The European Organization for Nuclear Research
Geneva, Switzerland

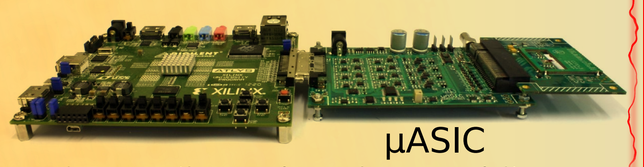
The Universal Verification Methodology (UVM) is a standardized approach of verifying integrated circuit designs, targeting a Coverage-Driven Verification (CDV). It combines automatic test generation, self-checking testbenches, and coverage metrics to indicate progress in the design verification. The flow of the CDV differs from the traditional directed-testing approach. With the CDV, a testbench developer, by setting the verification goals, starts with a structured plan. Those goals are targeted further by a developed testbench, which generates valid stimuli and sends them to a device under test (DUT). The progress is measured by coverage monitors added to the simulation environment. This way, the non-exercised functionality can be identified. Moreover, the additional scoreboards indicate undesired DUT behaviour.

Such verification environments were developed for three recent ASIC and FPGA projects which have successfully implemented the new work-flow:

- the **CLICpix2** 65 nm CMOS hybrid pixel readout ASIC design;
- the **C3PD** 180 nm HV-CMOS active sensor ASIC design;
- the **FPGA-based DAQ** (µASIC) system of the CLICpix chip.

CLICpix2

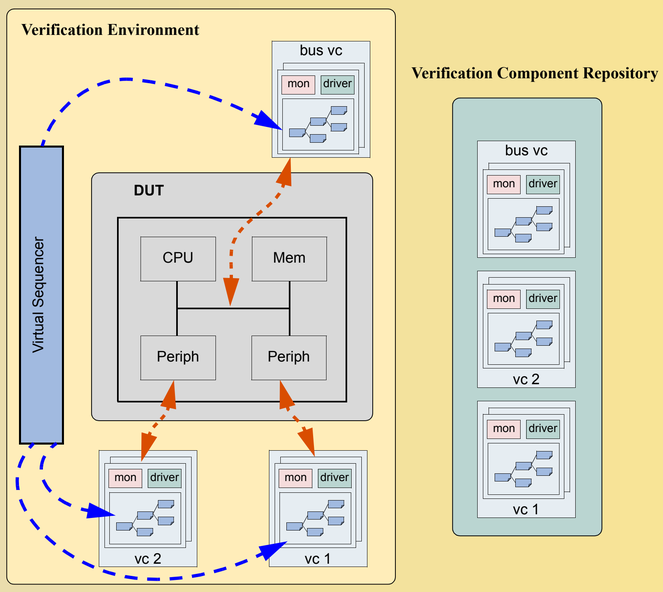
C3PD



µASIC

Different interfaces (Ethernet, trigger and timing interface, I2C, SPI) which stimulate the DUTs are handled by complex and versatile testbenches enabling an exhaustive system verification and identification of difficult-to-track design flaws.

UVM concept



- Coverage-Driven simulation (CDV)
- random generation of test vectors constrained by a user
- high level of abstraction (Transaction Level Modelling)
- EDA vendors provide libraries to verify popular interfaces (Ethernet, SPI, I2C, etc.)

System Verilog

- extremely scalable
  - extension of Verilog
  - Verification (and RTL)
- C++ like syntax
  - easy data modelling
  - object oriented
- interfaces
- assertion language (SVA)
- randomization and coverage support
- Direct Programming Interface (DPI)

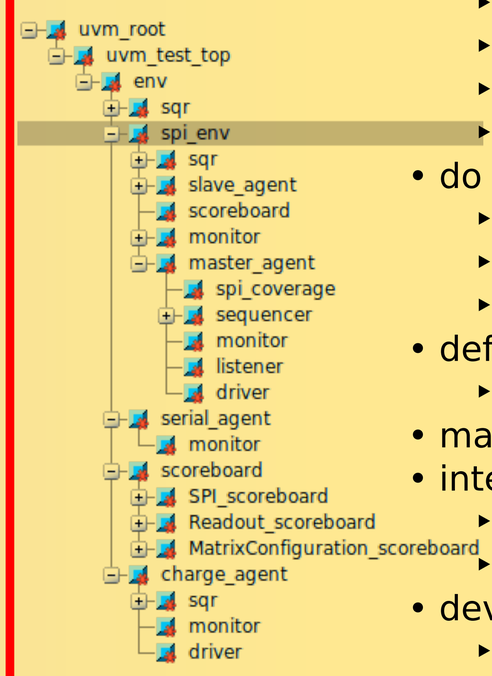
Step 1: Verification plan

Section	Title	Description	Link	Type	Weight	Goal
2	**SPI interface**				**1**	**100**
2.1	Single or back to back	This selects whether the transaction is single transfer or a back to back tr	spi_cvg,*:single_or_continuous	CoverPoint	0	100
2.2	SPI_read_corresponds_to_register_model	Read data corresponds the behavioral model.	SPI_read_corresponds_to_register_model	Assertion	1	100
2.3	write_read_all_addresses	Write and read all valid addresses.	SPI_access_cg:second_readout SPI_read_corresponds_to_register_model SPI_access_cg:first_readout	Coverpoint Assertion Coverpoint	1	100
2.4	check_default_values	Check default values of the registers after reset.	SPI_read_corresponds_to_register_model	Assertion	1	100
2.5	SPI_matrix_configuration_model_vs_registers	Checks whether values of the pixel matrix model correspond with the RTL	SPI_matrix_configuration_model_vs_registers_mask SPI_matrix_configuration_model_vs_registers_th_adj SPI_matrix_configuration_model_vs_registers_count_mode SPI_matrix_configuration_model_vs_registers_TP_en SPI_matrix_configuration_model_vs_registers_long_counter	Assertion	1	100
2.6	readback_matrix_configuration	Read back matrix configuration.	Readout_cg:readout_kind MatrixConfiguration_cg:matrix_configuration_scenario	Coverpoint Coverpoint	1	100
2.7	different_matrix_configuration_scenario	Check different matrix configurations	MatrixConfiguration_cg:matrix_configuration_scenario	CoverPoint	1	100
2.8	dummy_address	Access dummy addresses	SPI_access_cg:dummy_address	Coverpoint	1	100
3	**Reset signal**				**0**	**0**
4	**Readout interface**				**1**	**100**
4.1	UNDEFINED_X_STATE_ON_THE_LINK	No X or Z state on the bus.	UNDEFINED_X_STATE_ON_THE_LINK	Assertion	1	100
4.2	Number_of_columns	Readout with different number of parallel columns being read	Readout_cg:parallel_collumns	Coverpoint	1	100
4.3	header_and_carrier_extend_check	Check header and carrier extend	PACKET_HEADER_SHOULD_BE_A_REGULAR_DATA_WORD RCR_IN_PACKET_HEADER_DIFFERRS_FROM_REGISTER UNSUPPORTED_RCR_IN_PACKET_HEADER FOUND_UNEXPECTED_CONTROL_WORD_IN_PACKET	Assertion	1	100

- extract verification points from the chip specification
- choose the most appropriate implementation of the coverage model for each point of the plan:
  - Covergroup Modelling : checking permutations of condition and state when a known result is achieved (System Verilog Covergroups)
  - Cover Property Modelling: checking that a set of state transitions has been observed (System Verilog Assertions)

Tip: Questa provides add-on for MS Excel and Word and supports Open Office

Step 2: Verification platform



- build a **generic** environment eventually customized by the specific tests
  - use EDA libraries for standard interfaces (*Tip: Questa Verification IP*)
  - create a library and reuse your agents for your custom interfaces
  - take advantage of UVM configuration, logging and factory classes
  - follow UVM patterns (environment, item execution, etc.)
- do not use hierarchical paths to DUT inside the verification environment
  - create dedicated interfaces
  - use assertions in a top module or interfaces
  - use System Verilog directives (`*define*)
- define basic transactions
  - recording makes a waveform more readable
- make extensive use of randomization (use constraints)
- integrate the simulation with software over **DPI**
  - facilitates debugging (eg. Ethernet packets view in Wireshark)
  - allows debugging of the DUT interfaced with its slow control/DAQ software
- develop tests
  - generic test
  - inherit from the above specialized tests with dedicated sequences, eg. spiTest, chargeInjectionTest...

Tip: Look for help on https://verificationacademy.com/
Tip: Check https://gitlab.cern.ch/CLICdp/HDLVerificationLibrary

Step 3: Actual verification

- use issue ticketing system to share with other developers spotted bugs and ambiguity of the DUT specification (*Tip: Jira*)
- management tools
  - reduce maintenance
  - improve maintenance
  - instead of one long, launch few shorter simulations in **parallel**
  - change randomization seed

Tip: Check Questa Verification Management

- control progress of the verification process
  - coverage metrics
  - corner cases can be reached by specialized sequences
- swap the RTL description of the DUT with a back annotated netlist and rerun the tests
  - more realistic DUT description including internal delays