XILINX

A dynamic approach towards Register coverage generation and collection to reduce compilation overhead of traditional UVM register layers

Subham Banerjee
DVCon’2020, San Jose

Content

Motivation

> Why Register Coverage ?

- Register Coverage is integral part of any verification sign-off
- It helps to detect coverage holes, which otherwise go undetected
  - Let’s say there’s a register field ‘ADDR’, which is 3 bit wide, and not all values are realized in simulations
    - Two distinct write of 3’b000 & 3’b111 can cover all the toggle coverage
    - Based on the RTL implementation, code coverage can also be 100%
  - Analog registers used inside Behavioral Model (BV) & Real Number Model (RNM), are never analyzed as part code coverage
  - Coverage prior to reset and powerdown may be collected, as part of code coverage
- Register Coverage can un-earth all these holes, and many more

> Problem Statement

- Register Coverage comes with a inherent problem of compilation overhead
- Any typical UVM-RAL based automated-flow, attempts to generate covergroups/coverpoints per register/fields
- The overhead grows with higher number of registers
- Case Study:
  - The Gigabit Transceivers (SERDES) subsystems that we are working on, we have around 5000 odd register fields.
  - In conventional approach, this was resulted in a massive line of code (around 25000), and classes
  - This increased the compilation & elaboration time by ~17mintues comparing to the NO_COVERAGE compilation

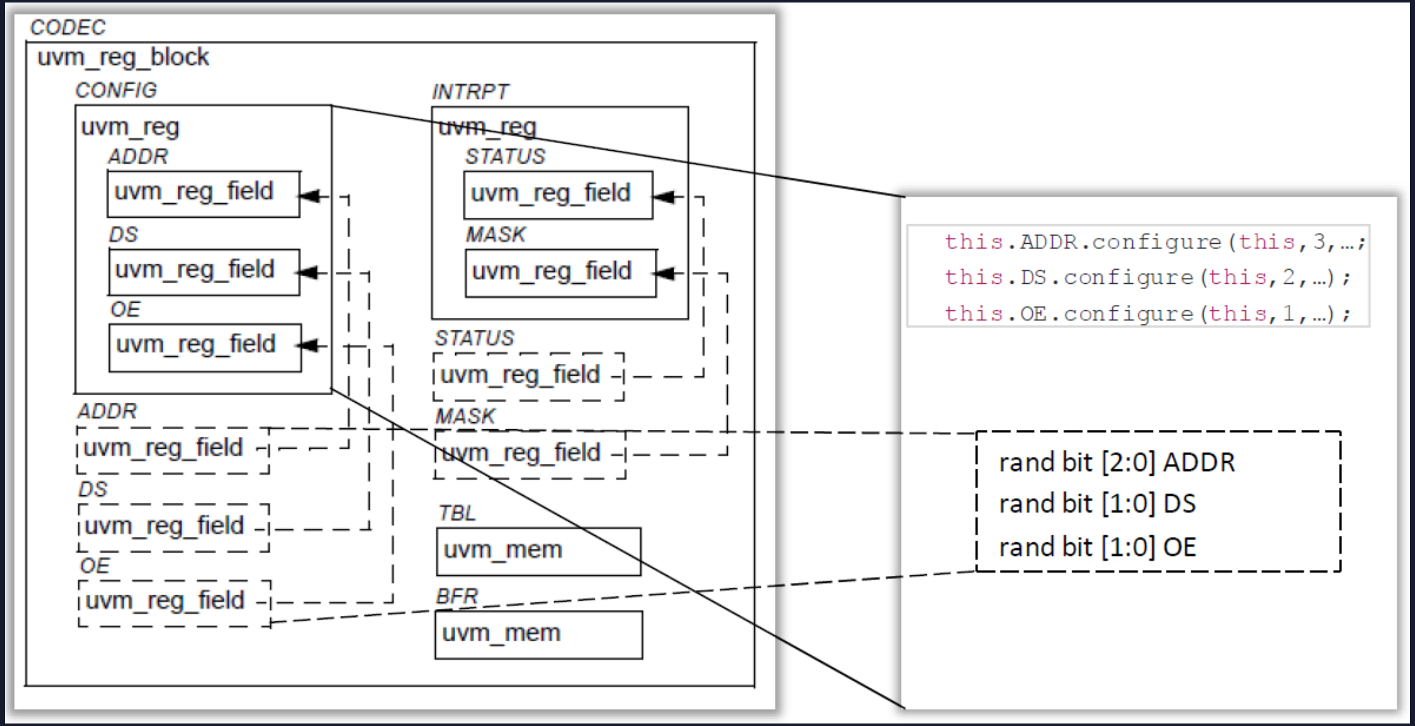
> Proposed Solution

- The proposed approach addresses this issue, by devising a fully reusable methodology which helps dynamic creation of all the covergroups/coverpoints,
- All covergroups are created during run or simulation time, as oppose to compilation time.
- Can be seamlessly extended to generate register cross coverage with minimal user intervention

XILINX

Basic Register Coverage Flow

- A typical UVM-RAL based register model realization
  - Coverage database will be created based on each field of the registers

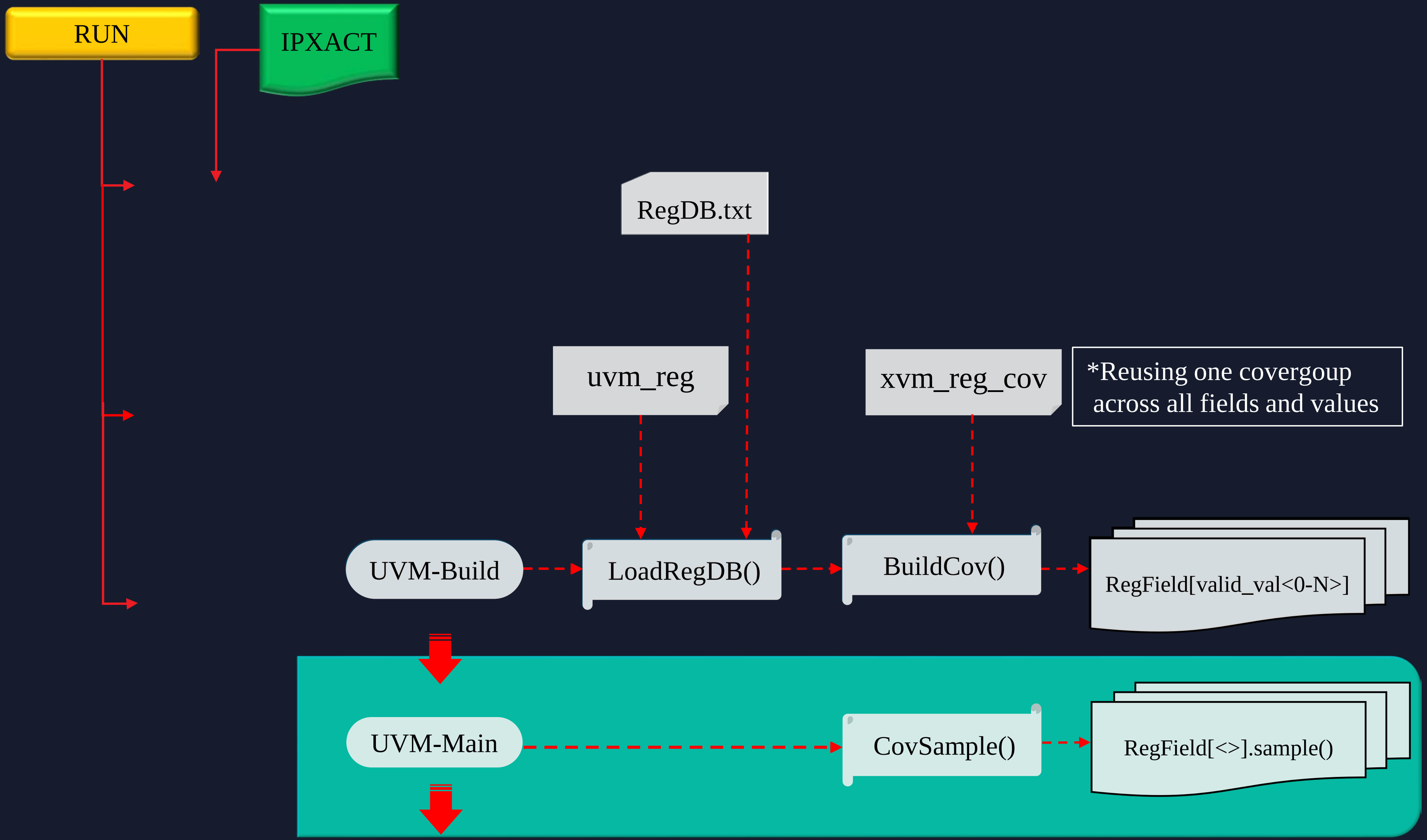


```
this.ADDR.configure(this,3,…;
this.DS.configure(this,2,…);
this.OE.configure(this,1,…);
```

```
rand bit [2:0] ADDR
rand bit [1:0] DS
rand bit [1:0] OE
```

Basic Register Coverage Flow

> Implementation



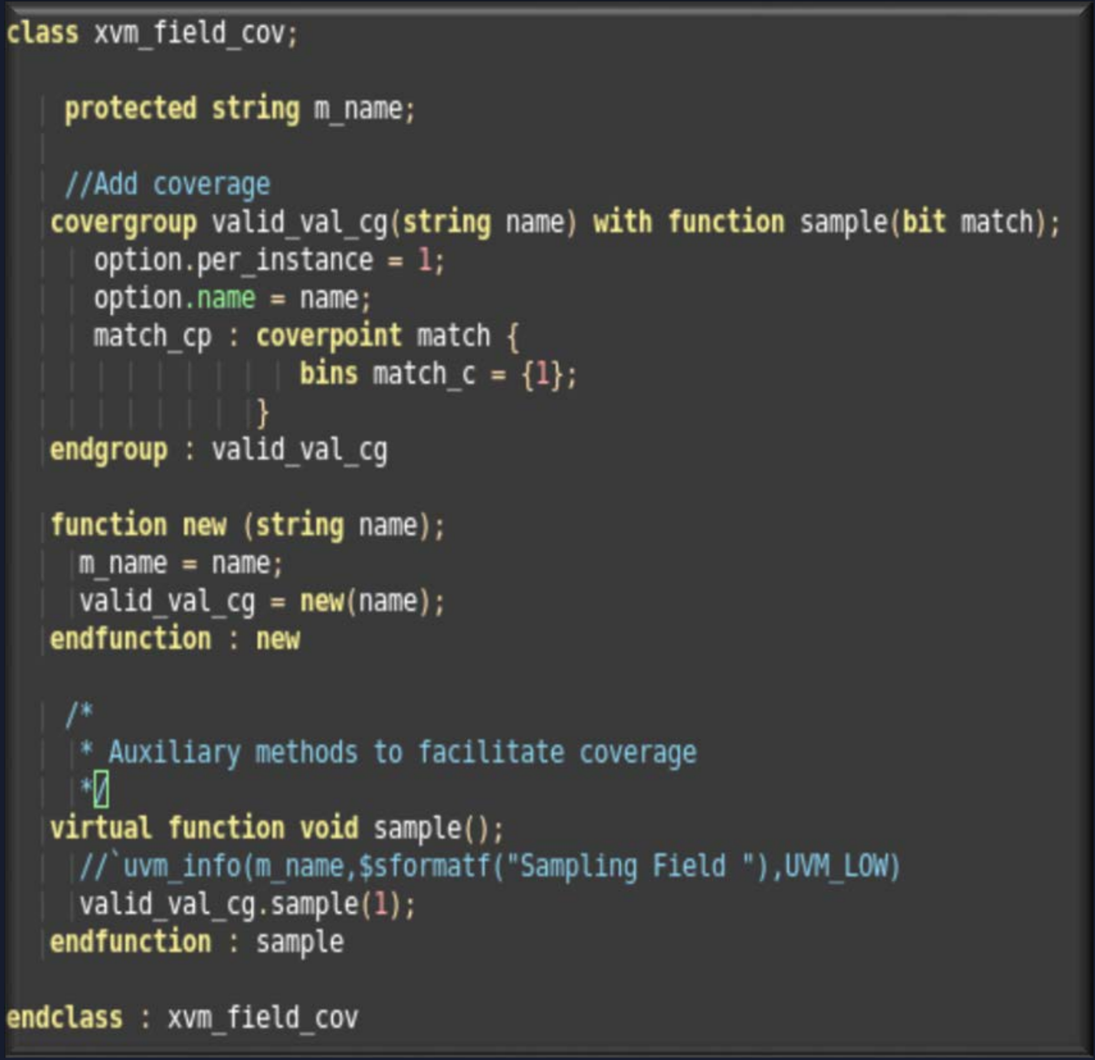
Basic Register Coverage Flow

> Generic Covergroup Creation

- One covergroup for
  - All Registers
  - All Fields within each Registers
  - All values of the field

> Covergroup Instantiation

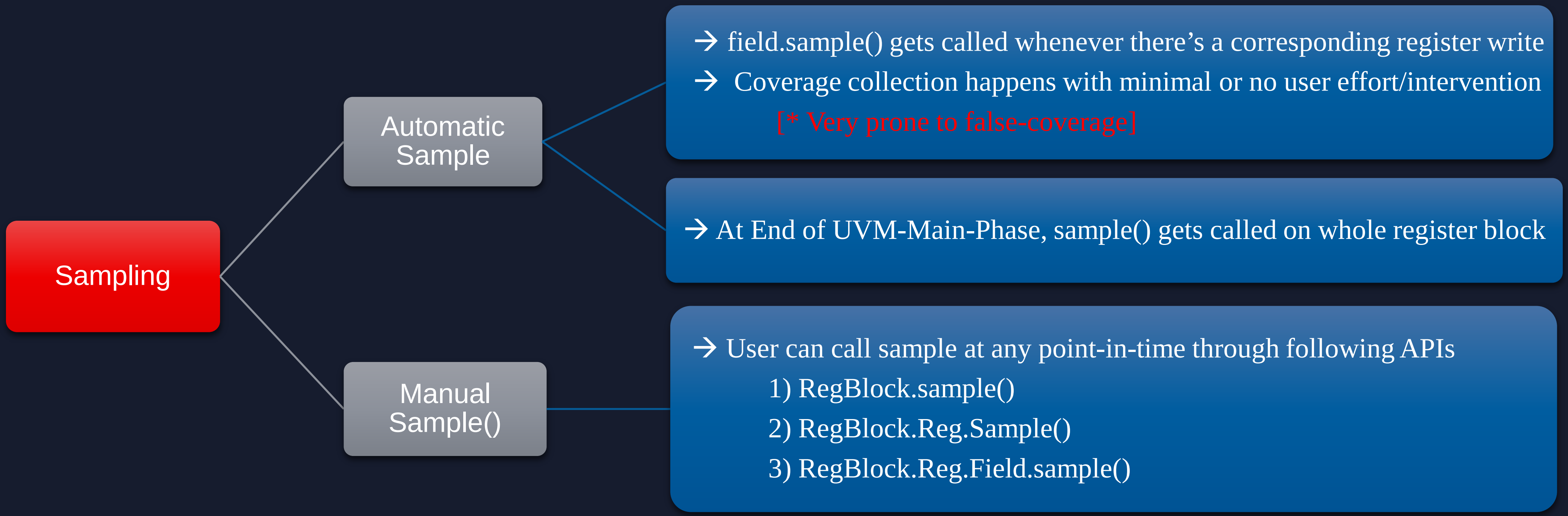
- Creation of covergroup wrapper class
- Instantiate wrapper-class in uvm_build_phase
- Better runtime control through uvm_config_db



```
class xvm_field_cov;

   protected string m_name;

   //Add coverage
  covergroup valid_val_cg(string name) with function sample(bit match);
    option.per_instance = 1;
    option.name = name;
    match_cp : coverpoint match {
                    bins match_c = {1};
                 }
  endgroup : valid_val_cg

  function new (string name);
   m_name = name;
   valid_val_cg = new(name);
  endfunction : new

   /*
    * Auxiliary methods to facilitate coverage
    */
  virtual function void sample();
   //`uvm_info(m_name,$sformatf("Sampling Field "),UVM_LOW)
   valid_val_cg.sample(1);
  endfunction : sample

endclass : xvm_field_cov
```

Basic Register Coverage Flow

> Sampling Techniques



→ field.sample() gets called whenever there’s a corresponding register write

→ Coverage collection happens with minimal or no user effort/intervention

[* Very prone to false-coverage]

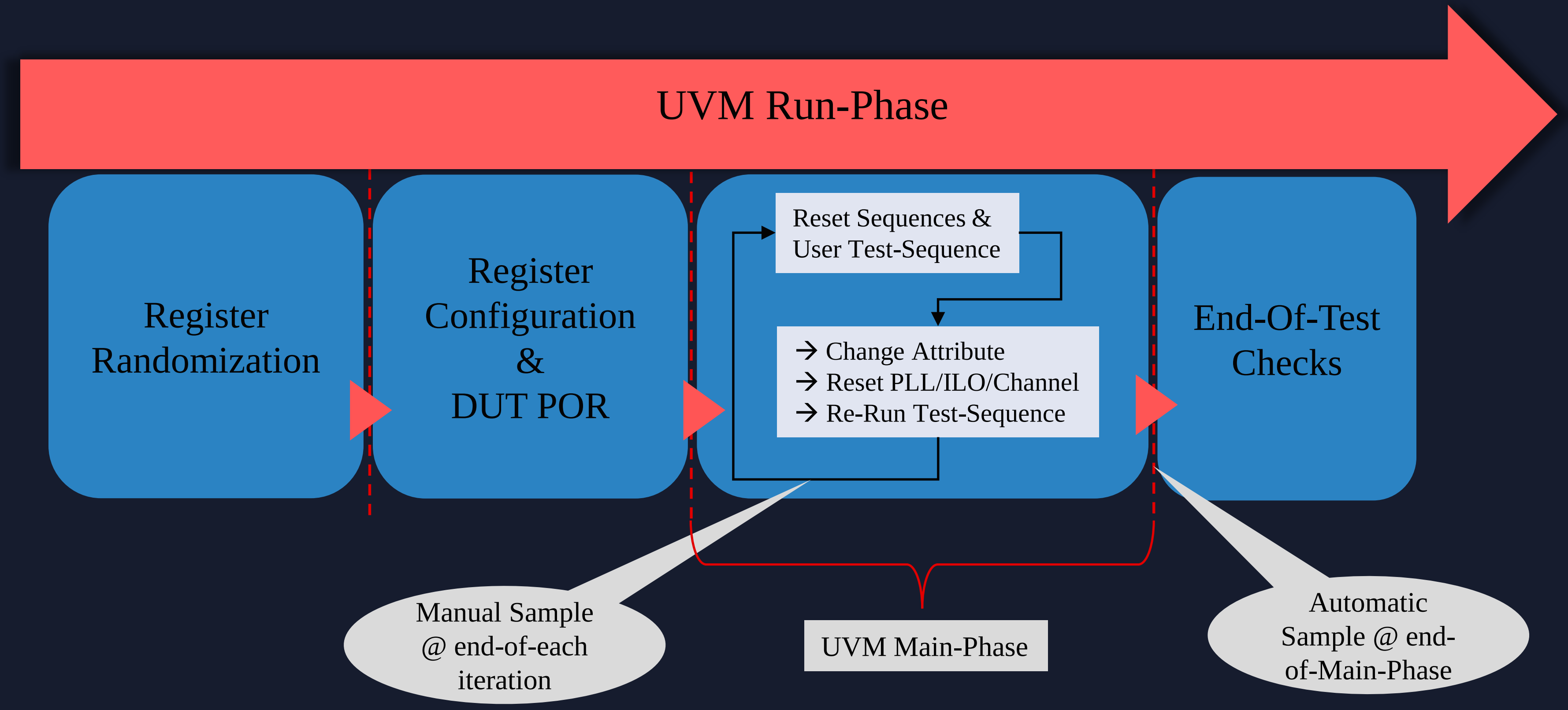
→ At End of UVM-Main-Phase, sample() gets called on whole register block

→ User can call sample at any point-in-time through following APIs

1) RegBlock.sample()
2) RegBlock.Reg.Sample()
3) RegBlock.Reg.Field.sample()

Basic Register Coverage Flow

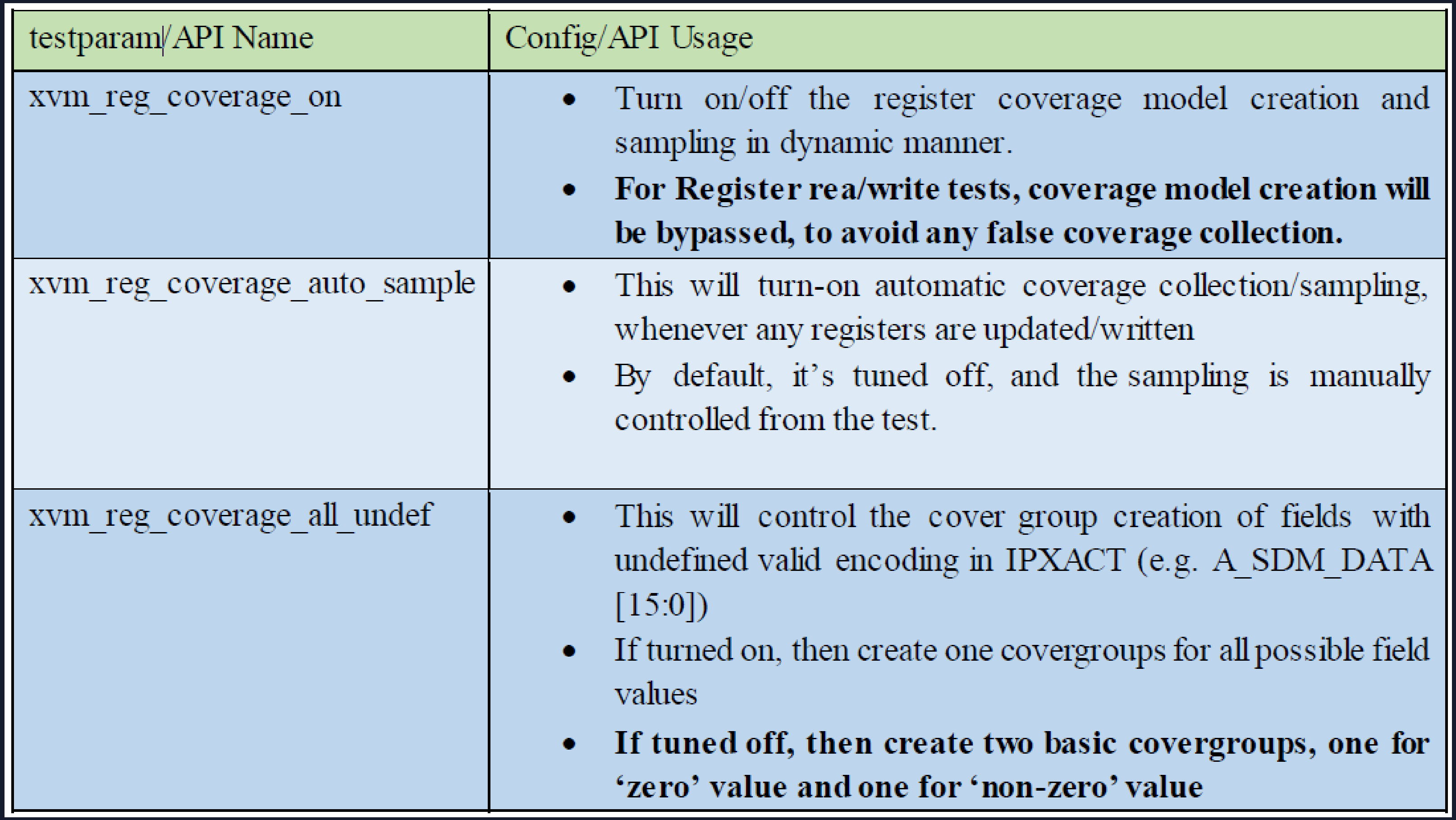
> Sampling Timelines



UVM Run-Phase
Register Randomization
Register Configuration & DUT POR
Reset Sequences & User Test-Sequence
→ Change Attribute
→ Reset PLL/ILO/Channel
→ Re-Run Test-Sequence
End-Of-Test Checks
Manual Sample @ end-of-each iteration
UVM Main-Phase
Automatic Sample @ end-of-Main-Phase

Basic Register Coverage Flow

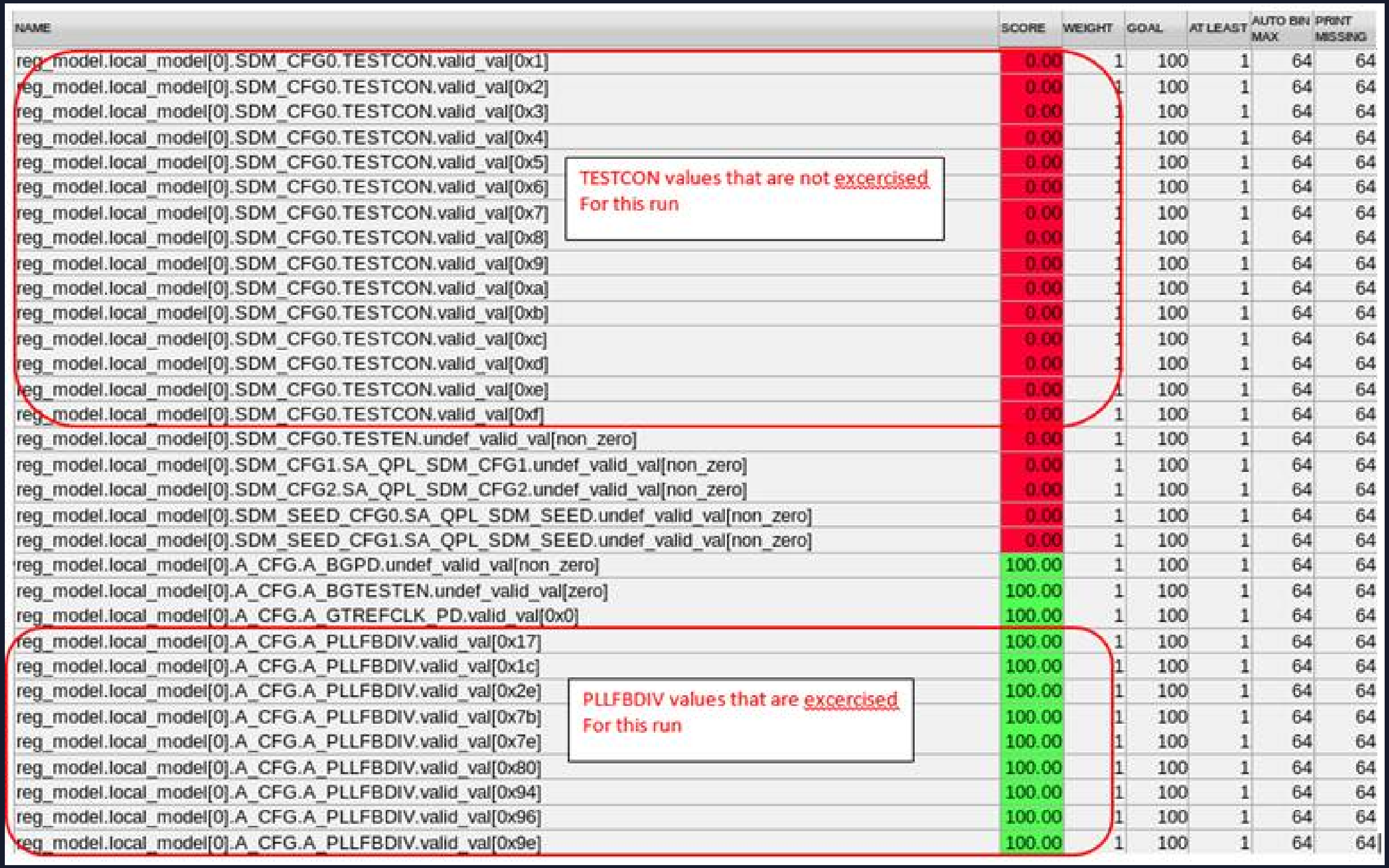
> Use Control for flow integration



testparam/API Name	Config/API Usage
xvm_reg_coverage_on	• Turn on/off the register coverage model creation and sampling in dynamic manner. • **For Register rea/write tests, coverage model creation will be bypassed, to avoid any false coverage collection.**
xvm_reg_coverage_auto_sample	• This will turn-on automatic coverage collection/sampling, whenever any registers are updated/written • By default, it's tuned off, and the sampling is manually controlled from the test.
xvm_reg_coverage_all_undef	• This will control the cover group creation of fields with undefined valid encoding in IPXACT (e.g. A_SDM_DATA [15:0]) • If turned on, then create one covergroups for all possible field values • **If tuned off, then create two basic covergroups, one for 'zero' value and one for 'non-zero' value**

Basic Register Coverage Flow

> Reporting



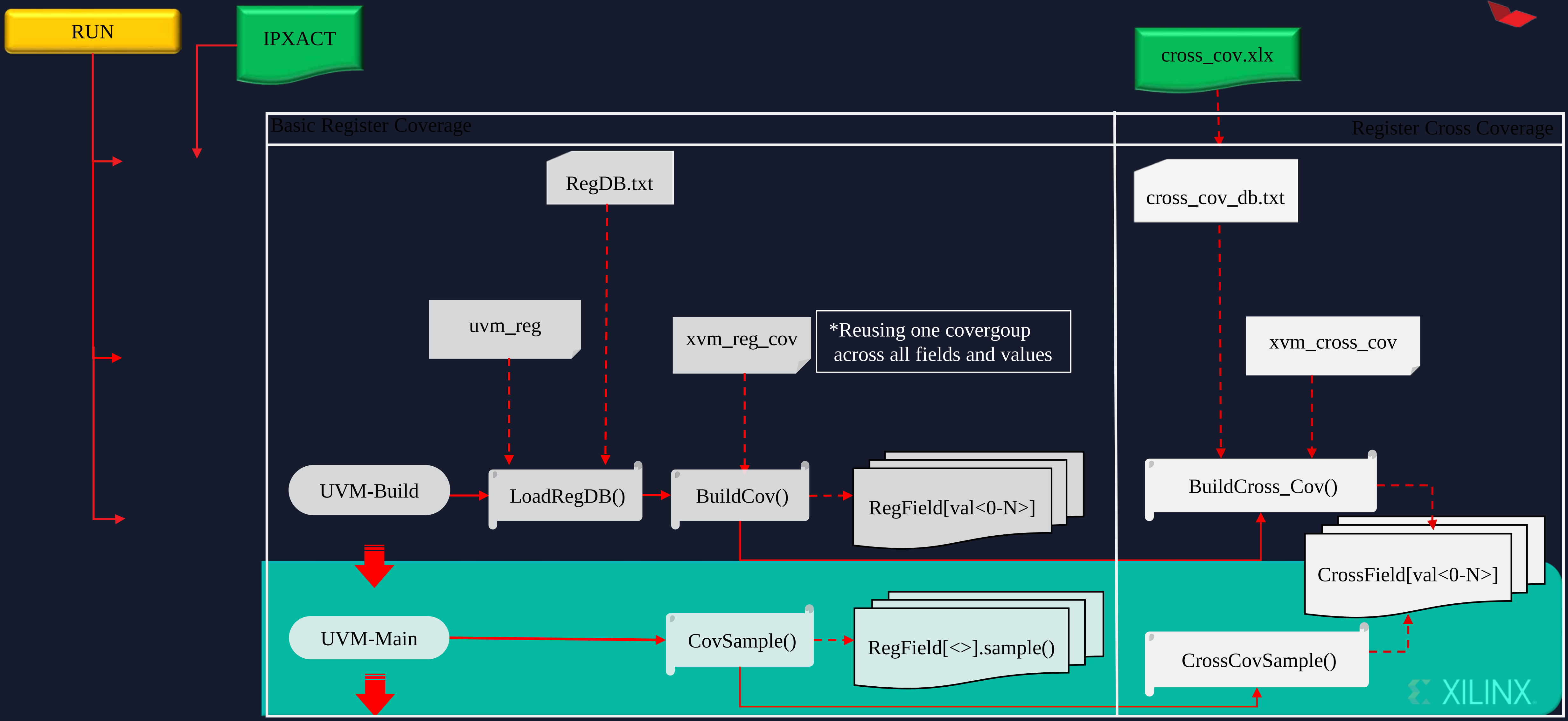
NAME	SCORE	WEIGHT	GOAL	AT LEAST	AUTO BIN MAX	PRINT MISSING
reg_model.local_model[0].SDM_CFG0.TESTCON.valid_val[0x1]	0.00	1	100	1	64	64
reg_model.local_model[0].SDM_CFG0.TESTCON.valid_val[0x2]	0.00	1	100	1	64	64
reg_model.local_model[0].SDM_CFG0.TESTCON.valid_val[0x3]	0.00	1	100	1	64	64
reg_model.local_model[0].SDM_CFG0.TESTCON.valid_val[0x4]	0.00	1	100	1	64	64
reg_model.local_model[0].SDM_CFG0.TESTCON.valid_val[0x5]	0.00	1	100	1	64	64
reg_model.local_model[0].SDM_CFG0.TESTCON.valid_val[0x6]	0.00	1	100	1	64	64
reg_model.local_model[0].SDM_CFG0.TESTCON.valid_val[0x7]	0.00	1	100	1	64	64
reg_model.local_model[0].SDM_CFG0.TESTCON.valid_val[0x8]	0.00	1	100	1	64	64
reg_model.local_model[0].SDM_CFG0.TESTCON.valid_val[0x9]	0.00	1	100	1	64	64
reg_model.local_model[0].SDM_CFG0.TESTCON.valid_val[0xa]	0.00	1	100	1	64	64
reg_model.local_model[0].SDM_CFG0.TESTCON.valid_val[0xb]	0.00	1	100	1	64	64
reg_model.local_model[0].SDM_CFG0.TESTCON.valid_val[0xc]	0.00	1	100	1	64	64
reg_model.local_model[0].SDM_CFG0.TESTCON.valid_val[0xd]	0.00	1	100	1	64	64
reg_model.local_model[0].SDM_CFG0.TESTCON.valid_val[0xe]	0.00	1	100	1	64	64
reg_model.local_model[0].SDM_CFG0.TESTCON.valid_val[0xf]	0.00	1	100	1	64	64
reg_model.local_model[0].SDM_CFG0.TESTEN.undef_valid_val[non_zero]	0.00	1	100	1	64	64
reg_model.local_model[0].SDM_CFG1.SA_QPL_SDM_CFG1.undef_valid_val[non_zero]	0.00	1	100	1	64	64
reg_model.local_model[0].SDM_CFG2.SA_QPL_SDM_CFG2.undef_valid_val[non_zero]	0.00	1	100	1	64	64
reg_model.local_model[0].SDM_SEED_CFG0.SA_QPL_SDM_SEED.undef_valid_val[non_zero]	0.00	1	100	1	64	64
reg_model.local_model[0].SDM_SEED_CFG1.SA_QPL_SDM_SEED.undef_valid_val[non_zero]	0.00	1	100	1	64	64
reg_model.local_model[0].A_CFG.A_BGPD.undef_valid_val[non_zero]	100.00	1	100	1	64	64
reg_model.local_model[0].A_CFG.A_BGTESTEN.undef_valid_val[zero]	100.00	1	100	1	64	64
reg_model.local_model[0].A_CFG.A_GTREFCLK_PD.valid_val[0x0]	100.00	1	100	1	64	64
reg_model.local_model[0].A_CFG.A_PLLFBDIV.valid_val[0x17]	100.00	1	100	1	64	64
reg_model.local_model[0].A_CFG.A_PLLFBDIV.valid_val[0x1c]	100.00	1	100	1	64	64
reg_model.local_model[0].A_CFG.A_PLLFBDIV.valid_val[0x2e]	100.00	1	100	1	64	64
reg_model.local_model[0].A_CFG.A_PLLFBDIV.valid_val[0x7b]	100.00	1	100	1	64	64
reg_model.local_model[0].A_CFG.A_PLLFBDIV.valid_val[0x7e]	100.00	1	100	1	64	64
reg_model.local_model[0].A_CFG.A_PLLFBDIV.valid_val[0x80]	100.00	1	100	1	64	64
reg_model.local_model[0].A_CFG.A_PLLFBDIV.valid_val[0x94]	100.00	1	100	1	64	64
reg_model.local_model[0].A_CFG.A_PLLFBDIV.valid_val[0x96]	100.00	1	100	1	64	64
reg_model.local_model[0].A_CFG.A_PLLFBDIV.valid_val[0x9e]	100.00	1	100	1	64	64

TESTCON values that are not excercised
For this run

PLLFBDIV values that are excercised
For this run

Register Cross Coverage Extension

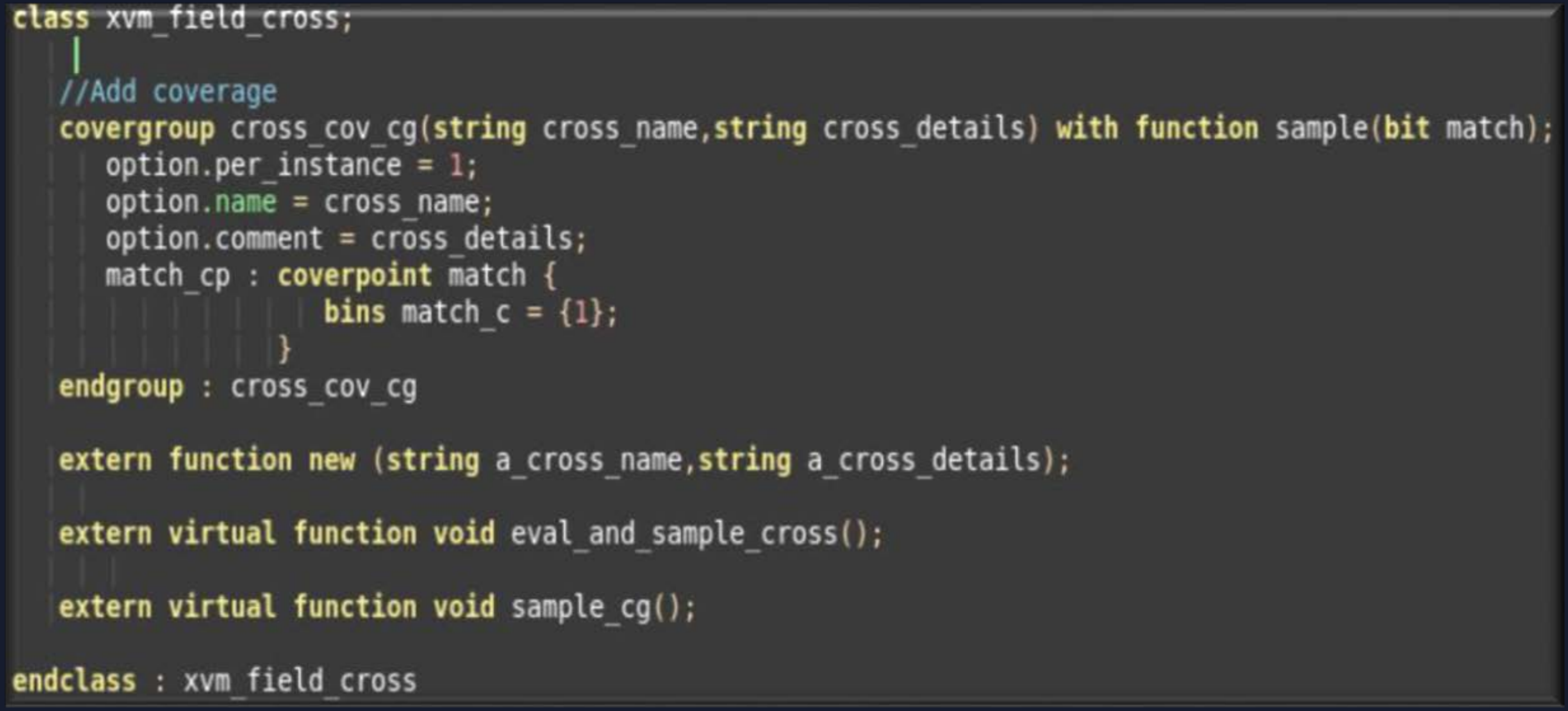
> Flow Implementation



Register Cross Coverage Extension

> Generic Cross Coverage Class

- One covergroup for any cross
- Each cross has multiple contributing fields
- Any of these fields changes, cross is sampled
- 'cross_details' field in the covergroup will have all the individual field info

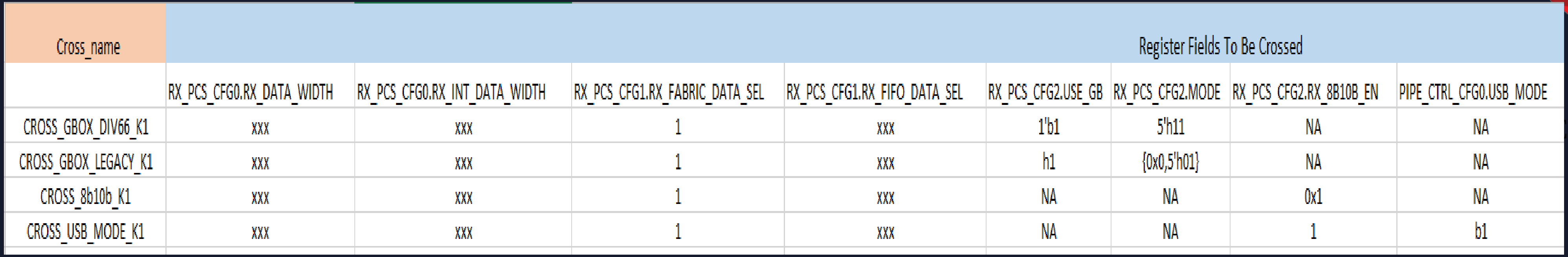


```
class xvm_field_cross;

    //Add coverage
    covergroup cross_cov_cg(string cross_name,string cross_details) with function sample(bit match);
        option.per_instance = 1;
        option.name = cross_name;
        option.comment = cross_details;
        match_cp : coverpoint match {
                        bins match_c = {1};
                    }
    endgroup : cross_cov_cg

    extern function new (string a_cross_name,string a_cross_details);

    extern virtual function void eval_and_sample_cross();

    extern virtual function void sample_cg();

endclass : xvm_field_cross
```

XILINX

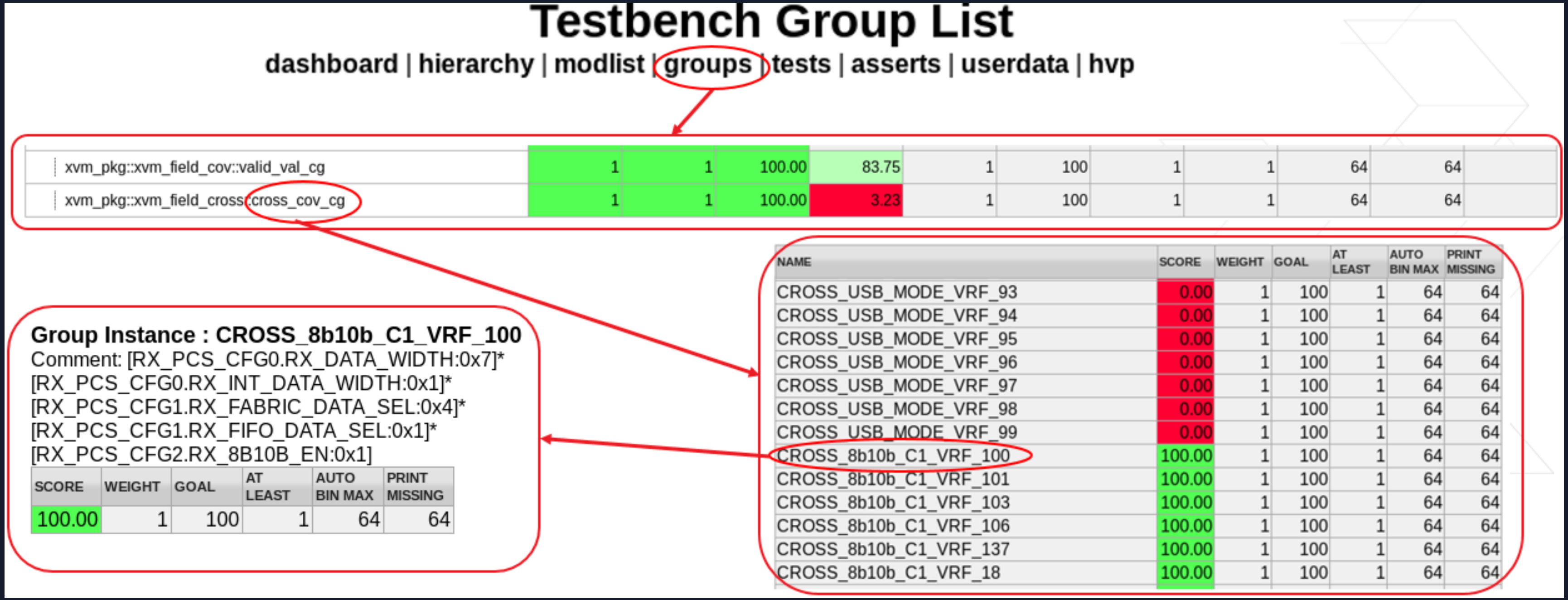
Register Cross Coverage Extension

> Cross Coverage Spreadsheet Format



Cross_name	Register Fields To Be Crossed							
	RX_PCS_CFG0.RX_DATA_WIDTH	RX_PCS_CFG0.RX_INT_DATA_WIDTH	RX_PCS_CFG1.RX_FABRIC_DATA_SEL	RX_PCS_CFG1.RX_FIFO_DATA_SEL	RX_PCS_CFG2.USE_GB	RX_PCS_CFG2.MODE	RX_PCS_CFG2.RX_8B10B_EN	PIPE_CTRL_CFG0.USB_MODE
CROSS_GBOX_DIV66_K1	xxx	xxx	1	xxx	1'b1	5'h11	NA	NA
CROSS_GBOX_LEGACY_K1	xxx	xxx	1	xxx	h1	{0x0,5'h01}	NA	NA
CROSS_8b10b_K1	xxx	xxx	1	xxx	NA	NA	0x1	NA
CROSS_USB_MODE_K1	xxx	xxx	1	xxx	NA	NA	1	b1

> Reporting



Testbench Group List

dashboard | hierarchy | modlist | groups | tests | asserts | userdata | hvp

xvm_pkg::xvm_field_cov::valid_val_cg	1	1	100.00	83.75	1	100	1	1	64	64
xvm_pkg::xvm_field_cross::cross_cov_cg	1	1	100.00	3.23	1	100	1	1	64	64

NAME	SCORE	WEIGHT	GOAL	AT LEAST	AUTO BIN MAX	PRINT MISSING
CROSS_USB_MODE_VRF_93	0.00	1	100	1	64	64
CROSS_USB_MODE_VRF_94	0.00	1	100	1	64	64
CROSS_USB_MODE_VRF_95	0.00	1	100	1	64	64
CROSS_USB_MODE_VRF_96	0.00	1	100	1	64	64
CROSS_USB_MODE_VRF_97	0.00	1	100	1	64	64
CROSS_USB_MODE_VRF_98	0.00	1	100	1	64	64
CROSS_USB_MODE_VRF_99	0.00	1	100	1	64	64
CROSS_8b10b_C1_VRF_100	100.00	1	100	1	64	64
CROSS_8b10b_C1_VRF_101	100.00	1	100	1	64	64
CROSS_8b10b_C1_VRF_103	100.00	1	100	1	64	64
CROSS_8b10b_C1_VRF_106	100.00	1	100	1	64	64
CROSS_8b10b_C1_VRF_137	100.00	1	100	1	64	64
CROSS_8b10b_C1_VRF_18	100.00	1	100	1	64	64

Group Instance : CROSS_8b10b_C1_VRF_100
Comment: [RX_PCS_CFG0.RX_DATA_WIDTH:0x7]*
[RX_PCS_CFG0.RX_INT_DATA_WIDTH:0x1]*
[RX_PCS_CFG1.RX_FABRIC_DATA_SEL:0x4]*
[RX_PCS_CFG1.RX_FIFO_DATA_SEL:0x1]*
[RX_PCS_CFG2.RX_8B10B_EN:0x1]

SCORE	WEIGHT	GOAL	AT LEAST	AUTO BIN MAX	PRINT MISSING
100.00	1	100	1	64	64

XILINX

Results & Analysis

> Comparison Between Predefined vs Dynamic Flow

Coverage Methodology	Data Metrics	Performance/Results
Pre-Defined Coverage Database (Old Flow)	• 5000 covergroups created • ~15000 coverage bins & no cross-coverage support • 25000 lines of extra code got added for each SERDES block • Total 12800 tests are running as part of the whole regression suit	• Total Compilation time including parsing & elaboration was at 25 minutes for one SERDES block/QUAD • For a subsystem with 2 SERDES-QUAD, the compilation time was around 35~40 minutes • Simulation time for 1 test with 1024KB data transfer was 25 minutes • Merging time for all 12800-coverage database from individual tests, is ~3 to 3.5 hours

Coverage Methodology	Data Metrics	Performance/Results
Dynamic Coverage Database (New Flow)	• 5000 covergroups created • ~ 15000 coverage bins & 126800 cross coverage bins were created • <100 lines of code are added, **this is constant for block level & subsystem level** • Total 12800 tests are running as part of the whole regression suit	• On a VCS based simulation platform, the time consumed to create the whole **coverage database was between 12~15 seconds** • **Compilation time reduced to 10~12minutes** • For a subsystem with 2 SERDES-QUAD, the compilation time is around 18 minutes • Simulation time for 1 test with 1024KB data transfer is **same as before, no significant change** • Merging time for all 12800-coverage database from individual tests, is ~3 to 3.5 hours, **which is comparable with previous flow**

Results & Analysis

> Conclusion

- This approach fits well into any chip or IP tapeout execution
- Makes verification engineer’s life a bit easier through a push button methodology for register coverage creation & collection
- The actual effort can be quickly put into analysis, as oppose to spending time in coverage creation and dealing with higher compilation overhead

XILINX®

Thank You