DVCon 2013
Design & Verification Conference & Exhibition

February 25-28, 2013
DoubleTree, San Jose

accellera SYSTEMS INITIATIVE

UCIS Applications: Improving Verification Productivity, Simulation Throughput, and Coverage Closure Process

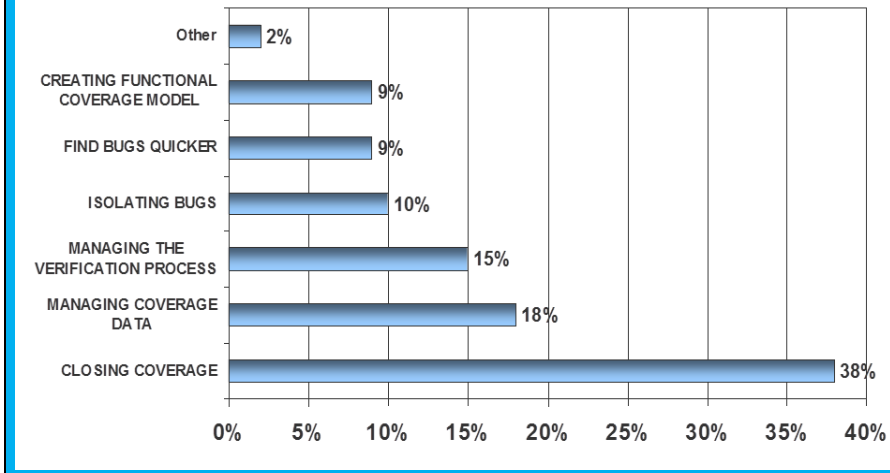
by
Ahmed Yehia
Verification Technologist
Mentor Graphics Corp.

Mentor Graphics®

Questa Verification Platform

Motivation

- Designs are getting more complex
  - Effort spent in verification is increasing
  - Larger designs require more tests to verify them
  - Time spent in simulation regression is significant for larger designs
  - Verification Challenges
    - Managing the verification process, closing coverage, finding/isolating bugs, and debugging



Verification Challenges
Wilson Research Group Functional Verification Study 2010

UCIS, how can it help?

Idea at a Glance…

- Make use of UCIS developing useful applications
  - Can be easily developed and deployed in any project
  - Serve post-run needs, improving:
    - Analysis, verification management, coverage closure process, and project tracking
  - Serve runtime needs, improving:
    - Simulation throughput, tests quality, verification productivity, and coverage closure

Outline

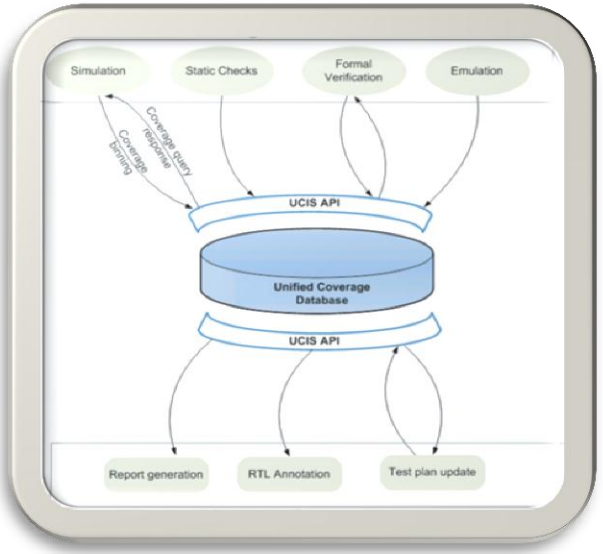
- Introduction to UCIS
- UCIS Runtime Applications
  - Test preparation for runtime applications
  - Guiding test behavior at runtime
  - Save test runtime and functional metrics
- UCIS Post-run Applications
  - Regression Analysis and Reporting
  - Merging
  - Ranking
- Conclusion

Introduction to UCIS

- Unified Coverage Interoperability Standard
  - Open and industry-standard API to improve verification productivity
  - Allow interoperability of verification metrics across different tools from multiple vendors
  - UCIS 1.0 released by Accellera UCIS committee in June 2012

Introduction to UCIS

- UCIS Database (UCISDB)
  - Single repository of all coverage data from all verification processes
  - Main structures
    - Scope: A hierarchical node that stores hierarchical design structure
    - Coveritem: Holds actual counts of recorded events
    - Historical nodes: Track historical construction processes and record environment attributes
- APIs and data structures
  - C language based
  - Enables access and manipulation of data inside UCISDB



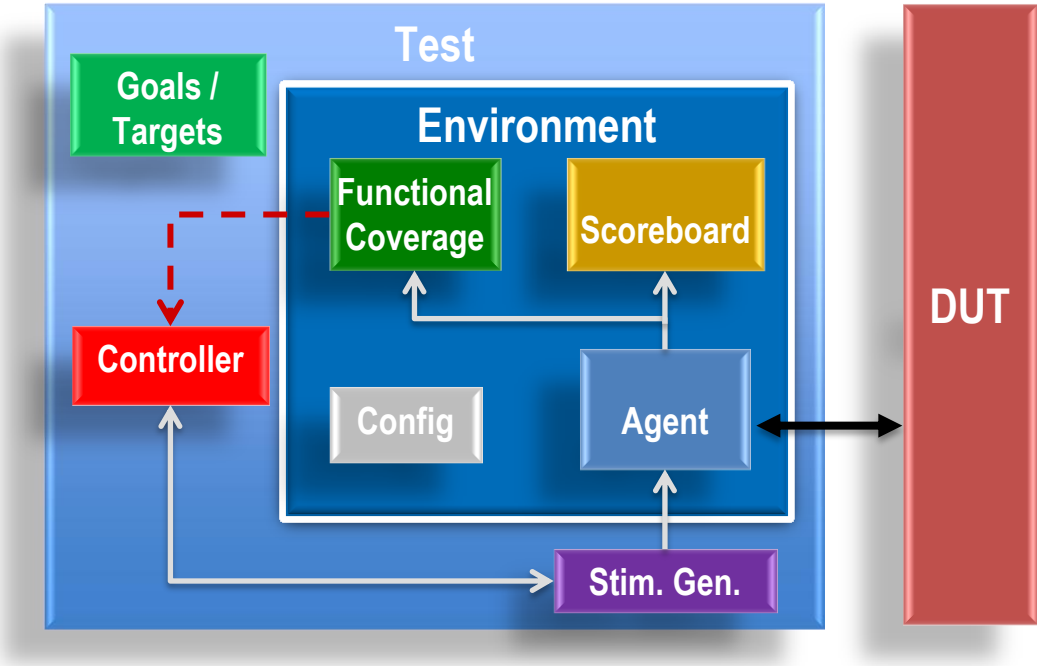
A. UCIS Runtime Applications

- Apps loaded as shared objects during simulation
- Connected to running test
  - Feedback information of interest to the test controller at runtime
- Guide test behavior at runtime
  - Maximize throughput and minimize resources
  - Tracking and improving test quality
  - Faster coverage closure
- Save test specific data of interest in a UCISDB for further post-run analysis
  - Tracking and trending project momentum

Test Preparation for Runtime Apps

1. Linking test to its coverage goals and targets

- Objective
  - Track/Measure test quality
  - Improve test efficiency
- How?
  - Test normally written to fulfill specific objectives
  - Embed info in dynamically executable documents
    - E.g. Verification plan
  - Pass info to test at beginning of simulation

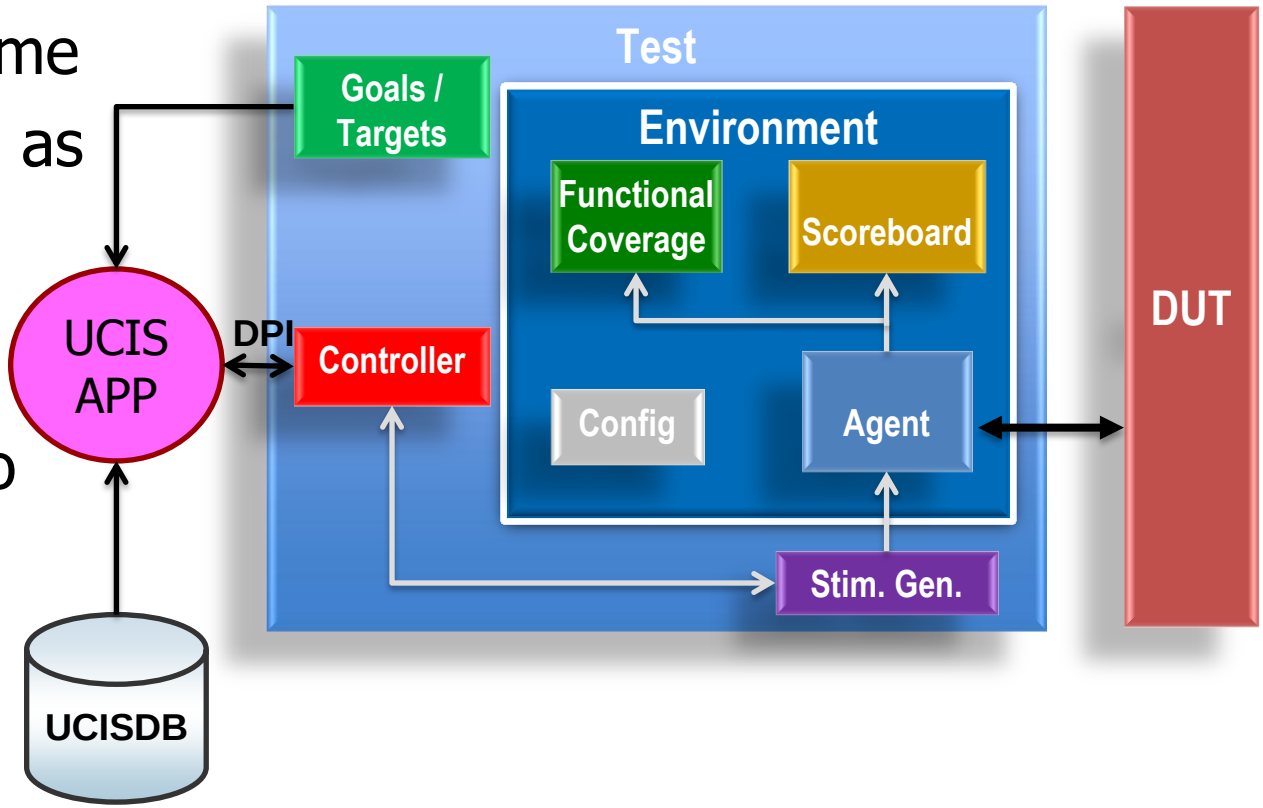



```
vsim fpu_tb_top +UVM_TESTNAME=test1
+COVERSCOPES=/19:fpu_agent_pkg/11:fcoverage
```

Test Preparation for Runtime Apps

2. Connecting UCIS apps to HVL testbench at runtime

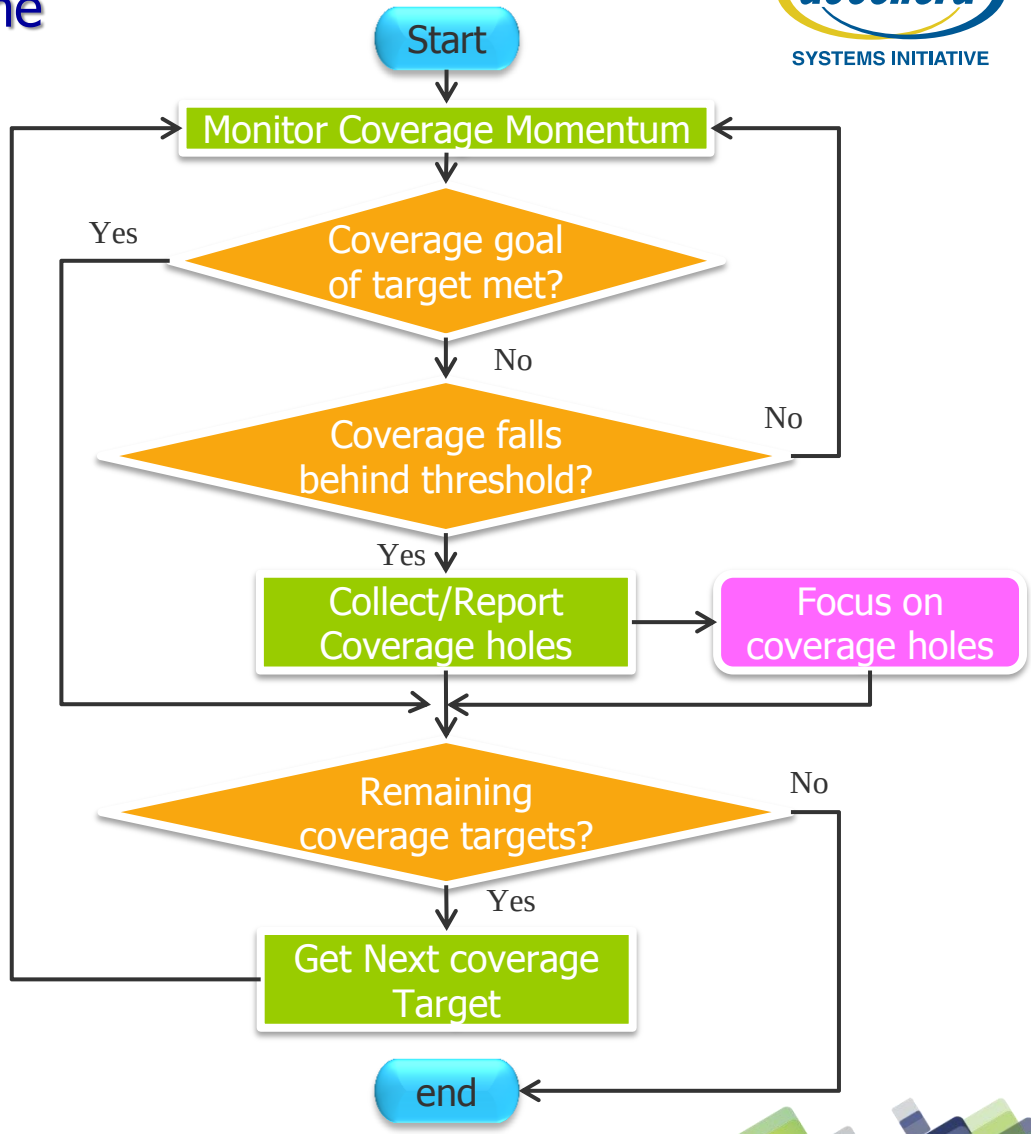
- Test saves coverage metrics/model in a UCISDB at run time
- UCIS App loaded as shared object in simulation
- Using DPI test controller talks to C application



A. UCIS Runtime Applications

1. Guiding test behavior at runtime

- On the fly change tests' runtime behavior upon collected coverage analysis
- Maximize simulation throughput
- Track tests quality



A. UCIS Runtime Applications

1. Guiding test behavior at runtime (cont.)

- Monitor achieved coverage
  - Notify testbench components when a target is achieved
  - Exit simulation when all targets/objectives are met

```
int checkCoverGoalMet(const char* dbname,
const char* scopename, double* scopecoverscore){
  ucisT db; /*UCIS in-memory DB handle*/
  ucisScopeT scope = NULL;/*UCIS scope handle*/
  int scopecovergoal;
  /*Populate in-memory DB from physical UCISDB*/
  db = ucis_Open(dbname);
 /*Find matching scope by name, get its handle*/
  scope = ucis_MatchScopeByUniqueID(db, NULL,
                                    scopename);
  *scopecoverscore = coverageScore(db, scope);
  scopecovergoal= ucis_GetIntProperty(db,scope,
                      -1, UCIS_INT_SCOPE_GOAL);
  /*Close DB and return list of holes string*/
  ucis_Close(db);
  return(*scopecoverscore*100 >=scopecovergoal);
}
```

```
import "DPI-C" function int checkCoverGoalMet
(string dbname, string scopename,
 output real scopecoverscore);

task test1::run_phase(uvm_phase phase);
  uvm_event target_met_e;
  phase.raise_objection (this, "test1");
  fork
    //Coverage score monitor loop
    while (1) begin
      //When no more targets to cover break
      if (i >= coverscopes.size())  break;
      wait_for_next_transaction();
      assert(!$coverage_save_mti("test1.ucisdb",
                  "/fpu_agent_pkg/fcoverage"));
      if (checkCoverGoalMet("test1.ucisdb",
          coverscopes[i].scope,
          coverscopes[i].coverage_score) > 0)
      begin
        //Notify testbench components when
        //specific coverage target is met
        uvm_config_db # (uvm_event)::get (agent,
       "", {coverscopes[i++].scope, "_goalmet"},
         target_met_e);
        target_met_e.trigger();
      end
    end
    //Start main test virtual sequence
    test1_v_seq.start(m_env.m_v_sqr);
  join_any
  phase.drop_objection (this, "test1");
endtask
```

A. UCIS Runtime Applications

1. Guiding test behavior at runtime (cont.)

- Monitor coverage momentum
  - Collect coverage holes when momentum falls behind threshold
- Feedback coverage holes to testbench components

```
char* getCoverHoles(const char* dbname,
                    const char* scopename){
  db = ucis_Open(dbname);
  scope = ucis_MatchScopeByUniqueID(db, NULL,
                                    scopename);
  scopetype = ucis_GetScopeType(db, scope);
  if((scopetype == UCIS_COVERPOINT)||
     (scopetype == UCIS_CROSS)){
    /*Coverpoint or a cross scope*/
    populateHolesList (db, scope, &holeslist);

  } else if(scopetype == UCIS_COVERGROUP)  {
    /*Covergroup scope: Loop on all sub-scopes*/
    ucisScopeT subscope = NULL;
    ucisIteratorT iterator = ucis_ScopeIterate(
                             db, scope, -1);
    while(subscope = ucis_ScopeScan(db, iterator)){
      populateHolesList(db,subscope,&holeslist);
    }
  } else {/*Handle other FCOV scopetypes...*/}
  ...
  return holeslist;
}
```

```
import "DPI-C" function string getCoverHoles
                      (string dbname,
                       string scopename);

task test1::run_phase(uvm_phase phase);
  //Coverage monitor loop
  while (1) begin
  ...
  coverage_momentum =
                coverscopes[i].coverage_score /
                num_of_trans;
      if(coverage_momentum < threshold_momentum)
      begin
        coverscopes[i].coverholes =
           getCoverHoles("test1.ucisdb",
                         coverscopes[i].scope);
        uvm_config_db # (uvm_event)::get (agent,
         "", {coverscopes[i].scope, "_holes"},
         coverholes_update_e);
        coverholes_update_e.trigger();
      end
    end
  end
endtask
```

A. UCIS Runtime Applications

2. Save test functional and runtime attributes

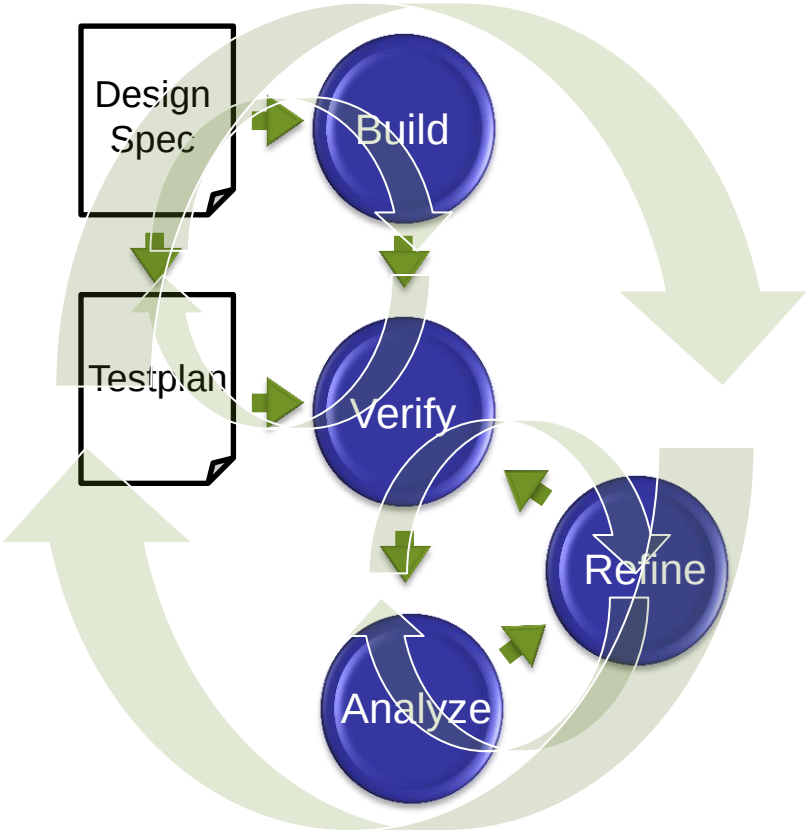
- Test may save specific functional and/or runtime metrics for post-run analysis
  - Help determine quality, cost-benefit, and means for improvement
  - Runtime metrics
    - Test name, test status, simtime, cputime, mem footprint, sim CLI, seed, date, username, etc.
  - Functional metrics
    - Test objective(s), targets met/missed, execution paths, sequences exercised, user-defined metrics

```
function void test1::extract_phase(uvm_phase phase);
  ...
  value.svalue = "PASSED";
  saveAttr("test1.ucisdb", "TESTSTATUS", value);

  value.svalue = "YES";
  saveAttr("test1.ucisdb", "COVER_TARGETS_MET",
                           value);
  value.svalue = coverholes;
  saveAttr("test1.ucisdb", "COVER_HOLES", value);
  ...
endfunction
```

```
void saveAttr (const char* dbname, const char * key,
               AttrValueT * value){
  ...
  db = ucis_Open(dbname);
  /*Get handle to the test record inside UCISDB*/
  iterator = ucis_HistoryIterate
             (db, NULL, UCIS_HISTORYNODE_TEST);
  while (test = ucis_HistoryScan (db, iterator))
  {
   value_.type = (ucisAttrTypeT) value->type_;

   if (value_.type == UCIS_ATTR_STRING) {
     value_.u.svalue = value->svalue;
   } else if (...) {...}
   assert(!ucis_AttrAdd(db,test,-1, key, &value_));
  }
  ucis_Write(db, dbname, NULL, 1, -1);
  ucis_Close(db);
}
```

B. UCIS Post-Run Applications

- Require UCISDBs generated prior to simulation run termination
  - A test should save a UCISDB holding its run metrics
- Measure/Analyze
  - Tests and regression runs status
  - Individual tests and aggregated regression coverage results
- React
  - Improve tests quality, efficiency and regression throughput
  - Leads to achieve coverage closure



B. UCIS Post-Run Applications

1. Regression analysis and reporting

- Reports general test info
- Pass/Fail status
  - Failing reasons facilitates debugging and results analysis
- Coverage targets status
  - Coverage holes identification helps improving tests quality
- Performance profiling
  - Spots bottlenecks and provides means to regression throughput improvement
- Regression summary statistics

TEST	Coverage Targets	CLI Args	Pass/Fail Status	Fail Reason	Fail Time	Coverage Targets Met	Coverage Holes	CPUTime (Sec)	Peak Memory Consumed(MB)
Wishbone_if_compliance	wishbone_cvg	+UVM_TESTNAME=Wishbone_if_compliance	PASS	-	-	YES	-	423.67	550.4
Wishbone_tx_rx_fifo	fifo_statistic, fifo_cornercases	+UVM_TESTNAME=Wishbone_tx_rx_fifo	PASS	-	-	NO	fifo_cornercases:C1, fifo_cornercases:C3,	1890.3	450.5
buffer_desc_1	BdRam_Cvg	+UVM_TESTNAME=buffer_desc_1	FAIL	** FATAL: randomization() of primitive class failure	18349 ns	NO	BdRam_Cvg:wrXadd, BdRam_Cvg:wrXdin, num_bd_cvg:TXBDs, txbd_frm_cvg:under_run, txbd_frm_cvg:retry_cnt, txbd_frm_cvg:retran_limit, txbd_frm_cvg:cs_lost,	120.6	234.4
ethmac_rand_rxtx_test	rx_tx_cvg	+UVM_TESTNAME=ethmac_rand_rxtx_test	PASS	-	-	NO	rxtx_seq_item_cg::rxtx_size_cross::<scenario_type[RXTX],min_sz, min_sz>, <scenario_type[RXTX],	845.5	332.9

B. UCIS Post-Run Applications

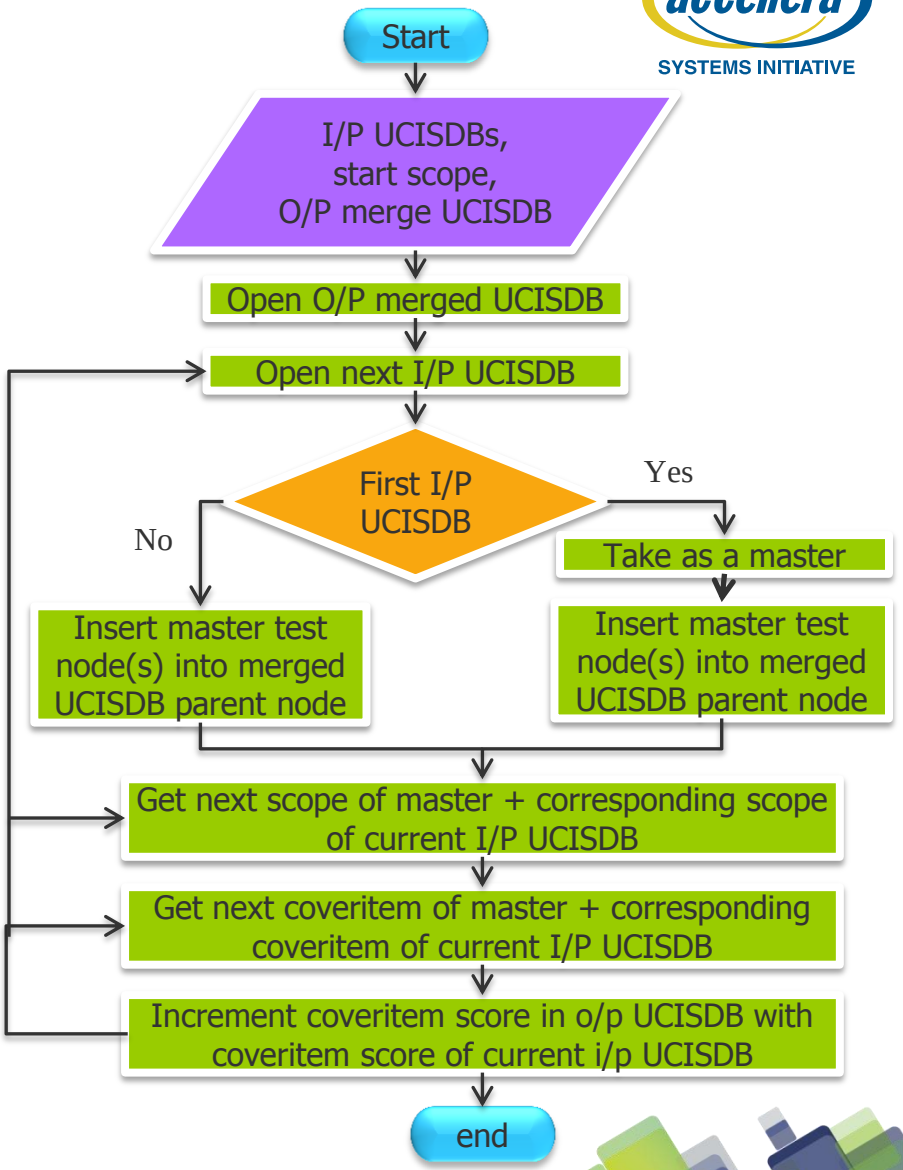
2. Merging UCISDBs

- Merges tests individual coverage results altogether in a single UCISDB
- Gets insight about overall regression coverage score w.r.t. verification plan target and objectives
- Helps answering questions "Are we done?", "What's missing to get things done?"

B. UCIS Post-Run Applications

2. Merging UCISDBs (cont.)

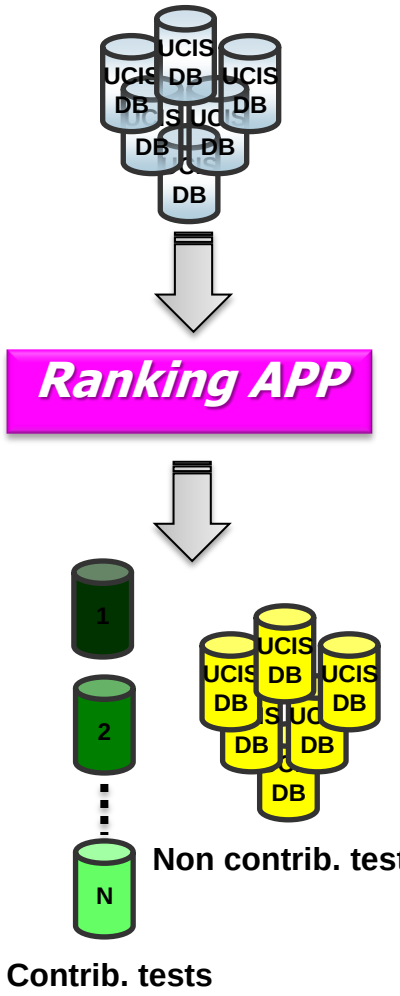
- Many merge modes do exist
- Totals Merge
  - All coveritems coverage scores in all UCISDBs are aggregated and written in the final merged UCISDB
  - Pros: simple, fast, compact merged UCISDB
  - Cons: no information about which test hit which coveritem is retained
- Test-Association Merge
  - Retains information about which test hit which coveritem
  - Pros: verbose, better analysis
  - Cons: Complex, additional coding, relatively larger merged UCISDB
  - Less overhead when solution is reused from supporting tools



B. UCIS Post-Run Applications

3. Ranking UCISDBs

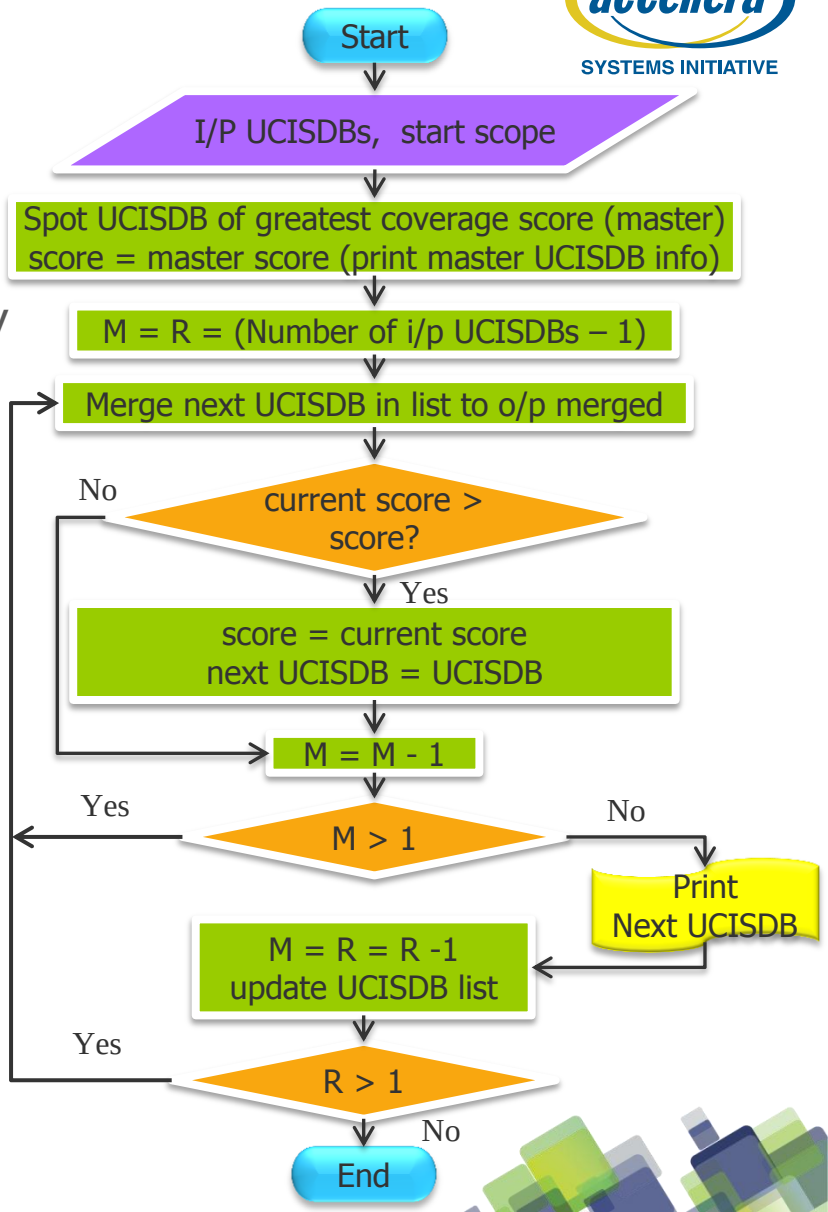
- Value tests individually according to their contribution to overall regression coverage
- Abandon redundant tests from regression
  - Save resources and boosts regression throughput
  - Hints for improving redundant tests
- Identify highly contributing tests in the regression
  - Acceptance sanity checking regression subset
  - Verify recent code changes with high TAT (Turn Around Time)
- Taking runtime metrics into consideration can also help in boosting overall regression performance



B. UCIS Post-Run Applications

3. Ranking UCISDBs (cont.)

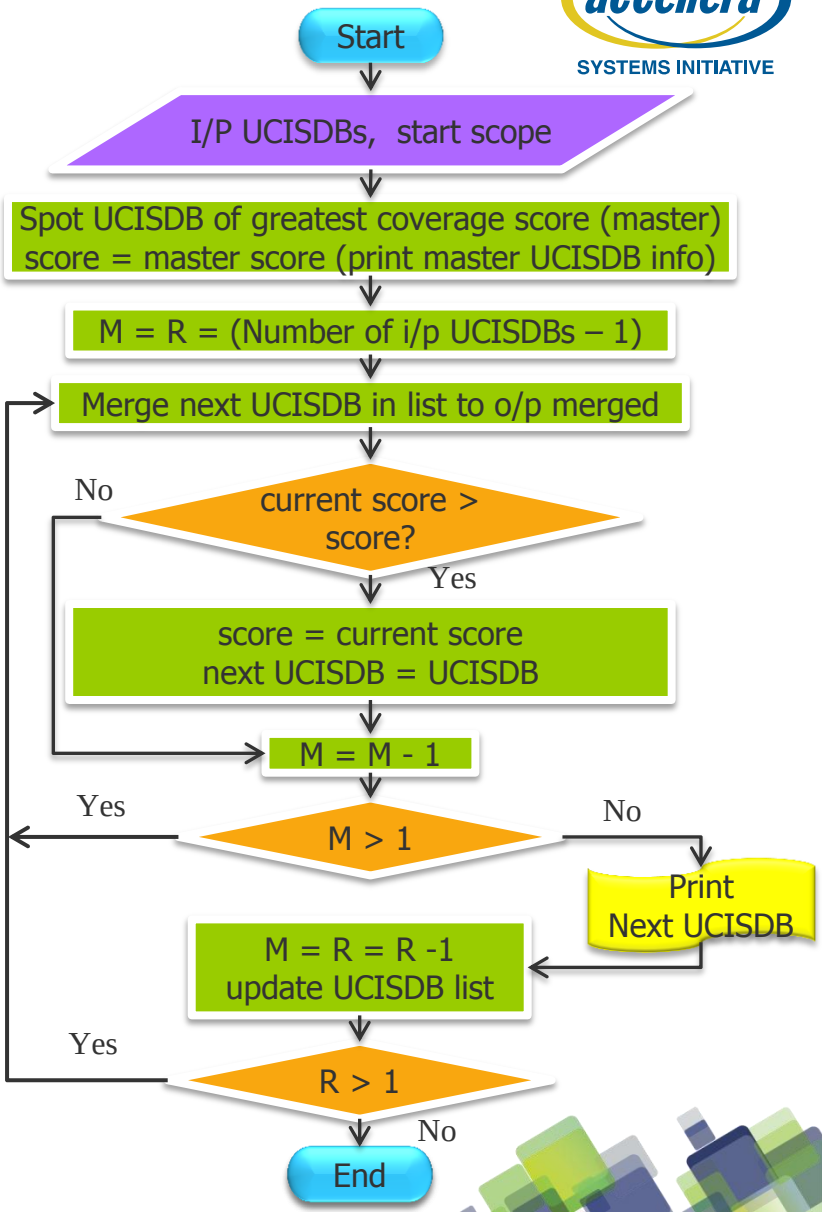
- Iterative Ranking
  - Ranks i/p UCISDBs by iterative merges seeking greatest coverage score
  - Although simple, it is a slow and greedy algorithm
    - Requires $\left(\frac{N^2+N}{2}\right)$ merges
  - Requires identical i/p UCISDBs
- Test-associated Ranking
  - Single merge
    - Rank based upon a test-associated merged results held in memory
  - Complex to implement
  - Much faster than iterative ranking
  - Less overhead when solution is reused from supporting tools rather than being built from scratch



B. UCIS Post-Run Applications

3. Ranking UCISDBs (cont.)

Rank	TEST	Seed	Coverage Score	Contributing
1	ethmac_rand_rxtx_test	17	56.8	YES
2	Wishbone_tx_rx_fifo	20301	72.3	YES
3	buffer_desc_1	1	79.7	YES
4	wishbone_if_compliance	0	85.4	YES
5	reg_read_writes	35	89.3	YES
6	reg_resets	2298773	89.5	Yes
7	ethmac_rand_simple_sanity	1529813	89.6	NO
8	patternset_ip	1856469	89.6	NO



Conclusion

- The UCIS contribution in the verification process can be significant.
- UCIS does not only help in post-run analysis and project tracking, however can be extended to runtime as well.
  - Runtime Apps: can maximize simulation throughput, on the fly change tests' behaviors and track tests' quality.
  - Post-run Apps: Verification Metrics analysis, improve coverage closure process and project tracking.
- Market expects growth and activities in UCIS to address many of its challenges.

References

[1] Accellera *Unified Coverage Interoperability Standard (UCIS)* Version 1.0, June 2, 2012.

[2] Wilson Research Group, *2010 Functional Verification Study*.

[3] IEEE Standard for SystemVerilog, *Unified Hardware Design, Specification, and Verification Language*, IEEE Std 1800-2012, 2012.

[4] UVM User Manual, uvmworld.org.