2023
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

UVM-SV Feedback Loop – The foundation of self-Improving Testbenches

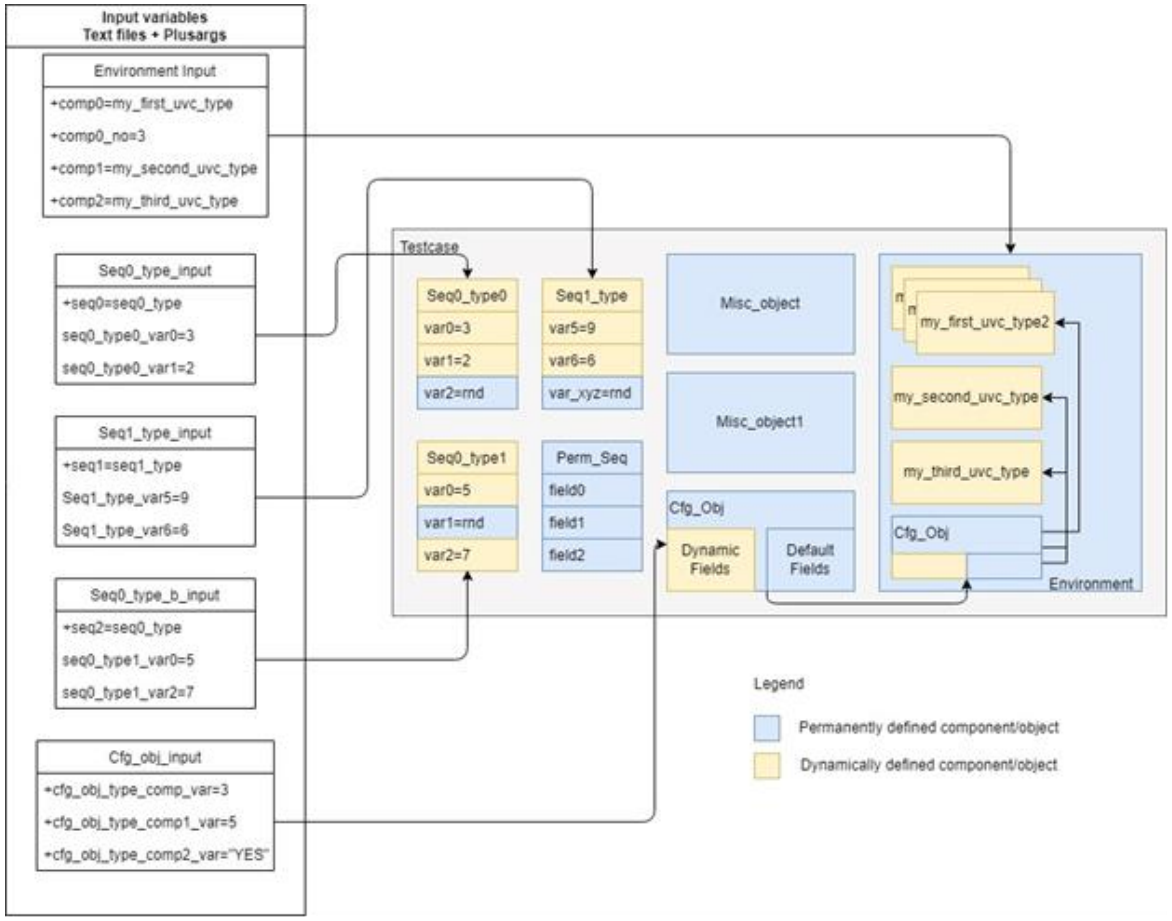
Andrei Vintila, Sergiu Duda

AMIQ Consulting

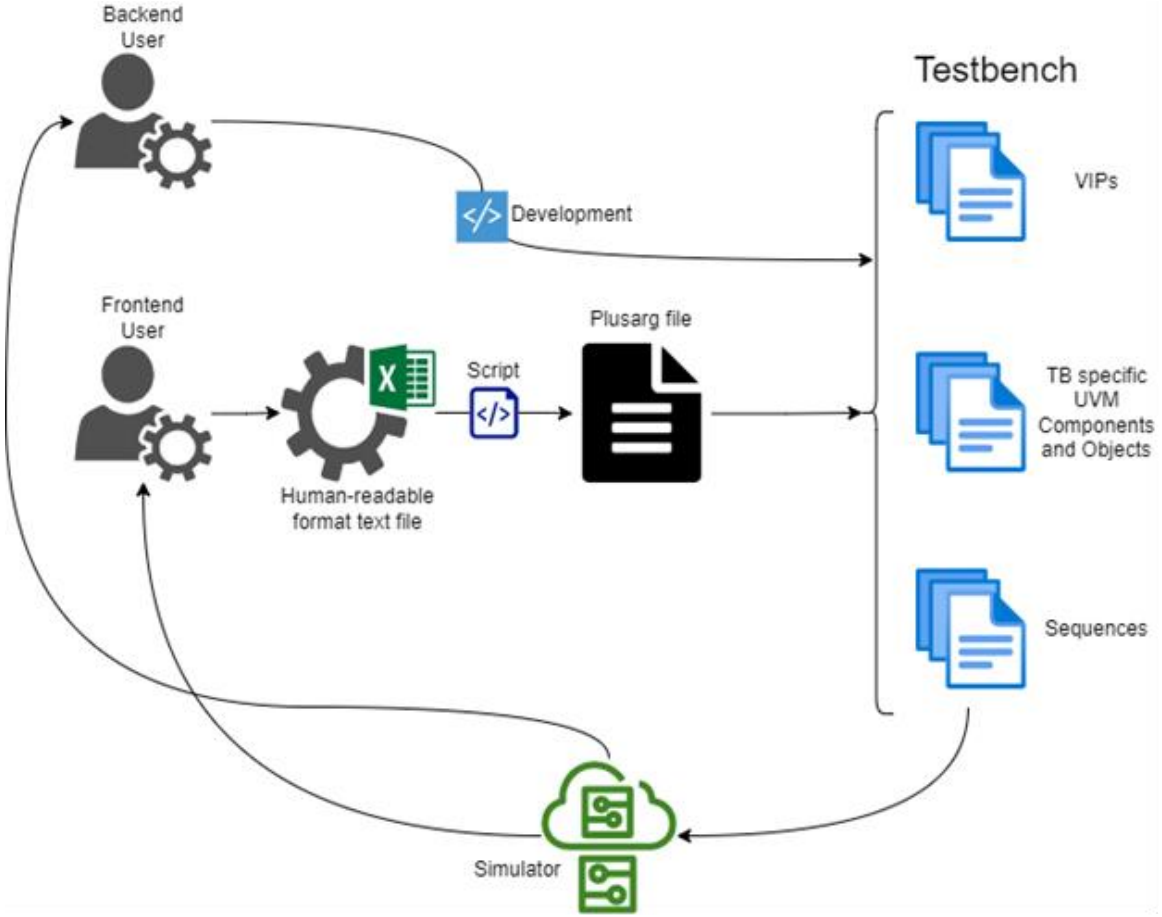
accellera
SYSTEMS INITIATIVE

Contents

Environment Example



TB abstraction layer



Concept and Rules

- Environment should be organized in such a way that all and any component and object can exist without a dependency on the other (Type agnostic / Allows runtime scaling up/down)
- All sequences should be organized as stand-alone entities, that can dynamically form a testcase based on their order and constraints
- All control variables, as well as stimuli relevant variables, should be registered as plusargs
- Creating a testcase means picking an environment and a collection of sequences

Environment setup (1)

```
class amiq_ectb_component extends uvm_component;

    `uvm_component_utils(amiq_ectb_component)

    function new (string name  = "amiq_ectb_component", uvm_component parent);
        super.new(name, parent);
    endfunction : new

    virtual function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        pre_create_objects();
        push_all_objs();
        create_objects();
        post_create_objects();


        pre_create_components();
        push_all_comps();
        create_components();
        post_create_components();

    endfunction : build_phase
```

/amiq_ectb/sv/amiq_ectb_environment.svh

```
+amiq_dvcon_tb_env_comp0=amiq_dvcon_tb_vip_red_agent
+amiq_dvcon_tb_env_comp0_name=red_agent
+amiq_dvcon_tb_env_comp0_no=2

+amiq_dvcon_tb_env_comp1=amiq_dvcon_tb_vip_blue_agent
+amiq_dvcon_tb_env_comp1_name=blue_agent
+amiq_dvcon_tb_env_comp1_no=1

+amiq_dvcon_tb_env_comp2=amiq_dvcon_tb_vip_purple_agent
+amiq_dvcon_tb_env_comp2_name=purple_agent
+amiq_dvcon_tb_env_comp2_no=3
```

/amiq_ectb/tb/tc/amiq_dvcon_tb_comp_args

Environment Setup (2)

```
class amiq_dvcon_tb_env extends amiq_ectb_environment;

    // Components of the environment
    amiq_dvcon_tb_env_cfg env_cfg;

    // Automatic and scaling env example
    amiq_dvcon_tb_vip_red_agent my_red_agents[$];
    amiq_dvcon_tb_vip_red_cfg_obj my_red_agents_cfg[$];

    amiq_dvcon_tb_vip_blue_agent my_blue_agents[$];
    amiq_dvcon_tb_vip_red_cfg_obj my_blue_agents_cfg[$];

    amiq_dvcon_tb_vip_purple_agent my_purple_agents[$];
    amiq_dvcon_tb_vip_purple_cfg_obj my_purple_agents_cfg[$];

    amiq_dvcon_tb_coverage_collector cov_collector;

    amiq_dvcon_tb_sqr virtual_sequencer;

    `uvm_component_utils(amiq_dvcon_tb_env)

    function new(string name = "amiq_dvcon_tb_env", uvm_component parent);
        super.new(name, parent);
    endfunction : new

    /**
     * @see amiq_dvcon_pkg::amiq_dvcon_environment.post_create_components
     */
    virtual function void post_create_components();
        super.post_create_components();
        cast_agents();
        configure_agents();

        // We cannot have a working TB without a sequencer, so we create it outside
        // of the plusarg dynamic scheme
        virtual_sequencer = amiq_dvcon_tb_sqr::type_id::create("virtual_sequencer", this);
    endfunction : post_create_components
```

```
function void cast_agents();
    foreach(components[i]) begin
        case(components[i].get_type_name())
            "*red_agent*": begin
                amiq_dvcon_tb_vip_red_agent proxy_agent;
                $cast(proxy_agent, components[i]);
                my_red_agents.push_back(proxy_agent);
            end
            "*blue_agent*": begin
                amiq_dvcon_tb_vip_blue_agent proxy_agent;
                $cast(proxy_agent, components[i]);
                my_blue_agents.push_back(proxy_agent);
            end
            "*purple_agent*": begin
                amiq_dvcon_tb_vip_purple_agent proxy_agent;
                $cast(proxy_agent, components[i]);
                my_purple_agents.push_back(proxy_agent);
            end
            // Keep in mind, that components that have to always exist, can be created separate
            // This serves as an example of an env that can be created without a coverage collector
            "*coverage_collector*": begin
                $cast(cov_collector, components[i]);
            end
        endcase
    end
endfunction


function void configure_agents();
    if(my_red_agents.size()>1)
        foreach(my_red_agents[i]) begin
            amiq_dvcon_tb_vip_red_cfg_obj proxy_red_agent_cfg;
            proxy_red_agent_cfg = red_cfg(i);
            uvm_config_db#(amiq_dvcon_tb_vip_red_cfg_obj)::set(this, $sformatf("*red_agent*%0d", i)
        end
```

Environment Setup (3)

```
class amiq_dvcon_tb_env_cfg extends amiq_ectb_object;

    // Note: We created it with the maximum number of agents we can use.
    // This can be made dynamic as well, but you'll need to know the number
    // of agents during register_all_vars() call (additional variable interrogated)
    // Minimizing the number of variables results in too small of a gain

    // Active/passive for all vips
    uvm_active_passive_enum red_vip_is_active[5];
    uvm_active_passive_enum blue_vip_is_active[5];
    uvm_active_passive_enum purple_vip_is_active[5];

    // Enable/disable checks for all vips
    bit red_vip_has_checks[5];
    bit red_vip_has_coverage[5];
    bit blue_vip_has_checks[5];
    bit blue_vip_has_coverage[5];
    bit purple_vip_has_checks[5];
    bit purple_vip_has_coverage[5];

    `uvm_object_utils(amiq_dvcon_tb_env_cfg)

    function new (string name = "amiq_dvcon_tb_env_cfg");
        super.new(name);
    endfunction : new

    virtual function void register_all_vars();

        // Active/Passive
        red_vip_is_active[0] = uvm_active_passive_enum'(bit_reg("red_vip0_is_active", UVM_ACTIVE));
        red_vip_is_active[1] = uvm_active_passive_enum'(bit_reg("red_vip1_is_active", UVM_ACTIVE));
        red_vip_is_active[2] = uvm_active_passive_enum'(bit_reg("red_vip2_is_active", UVM_ACTIVE));
        red_vip_is_active[3] = uvm_active_passive_enum'(bit_reg("red_vip3_is_active", UVM_ACTIVE));
        red_vip_is_active[4] = uvm_active_passive_enum'(bit_reg("red_vip4_is_active", UVM_ACTIVE));
        red_vip_is_active[5] = uvm_active_passive_enum'(bit_reg("red_vip5_is_active", UVM_ACTIVE));
```

```
class amiq_dvcon_tb_tc extends amiq_ectb_test;

    `uvm_component_utils(amiq_dvcon_tb_tc)

    amiq_dvcon_tb_env my_env;
    amiq_dvcon_tb_env_cfg env_cfg;

    realtime system_time_history[$];

    function new(string name = "amiq_dvcon_tb_tc", uvm_component parent=null);
        super.new(name,parent);
    endfunction : new

    virtual function void build_phase(uvm_phase phase);
        set_type_override_by_type(amiq_ectb_environment::get_type(), amiq_dvcon
        super.build_phase(phase);

        env_cfg = new("env_cfg");
        my_env = amiq_dvcon_tb_env::type_id::create("amiq_dvcon_tb_env", this);
        my_env.env_cfg = env_cfg;
```

```
+env_cfg_red_vip0_is_active=1
+env_cfg_red_vip1_is_active=0
+env_cfg_blue_vip0_is_active=1
+env_cfg_purple_vip1_is_active=0
```

Text inputs vs tests (1)

```
task run_phase(uvm_phase phase);
    super.run_phase(phase);

    if(virtual_sequencer==null) `uvm_fatal(get_name()

    // Retrieve the factory globally so it is availab
    factory = uvm_factory::get();

    // Read the plusargs for all the defined sequence
    retrieve_all_seq_type();

    // Based on the previous returned types, names ar
    for(int i=0; i<sequence_types.size(); i++) begin
        schedule_sequence(i);
        wait_threads(i);
    end
endtask : run_phase
```

/amiq_ectb/sv/amiq_ectb_test.svh

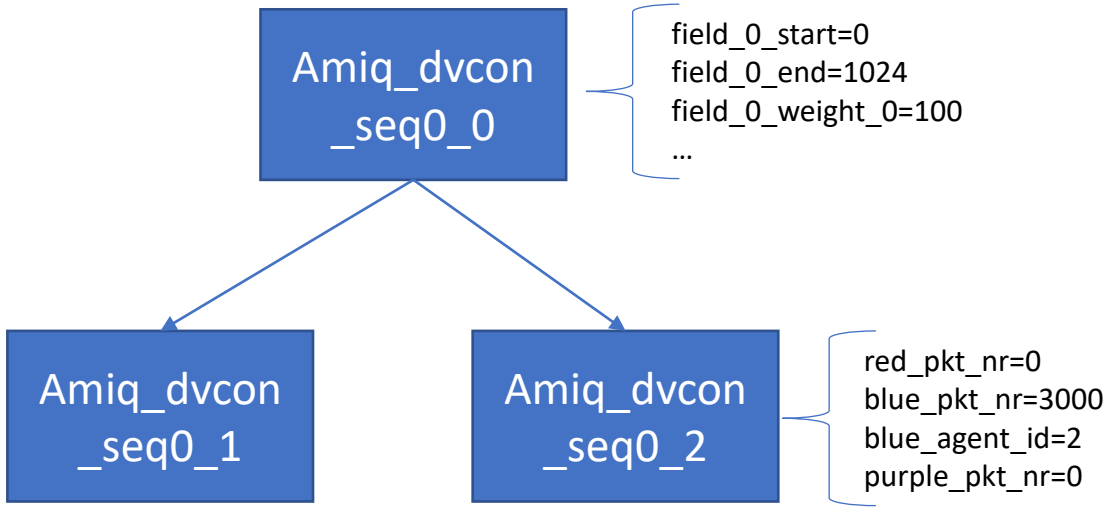
```
task run_phase(uvm_phase phase);
    phase.raise_objection(this);
    super.run_phase(phase);
    phase.drop_objection(this);
    collect_current_system_time_in_s();
endtask : run_phase
```

/amiq_ectb/tb/tc/amiq_dvcon_tb_tc.svh

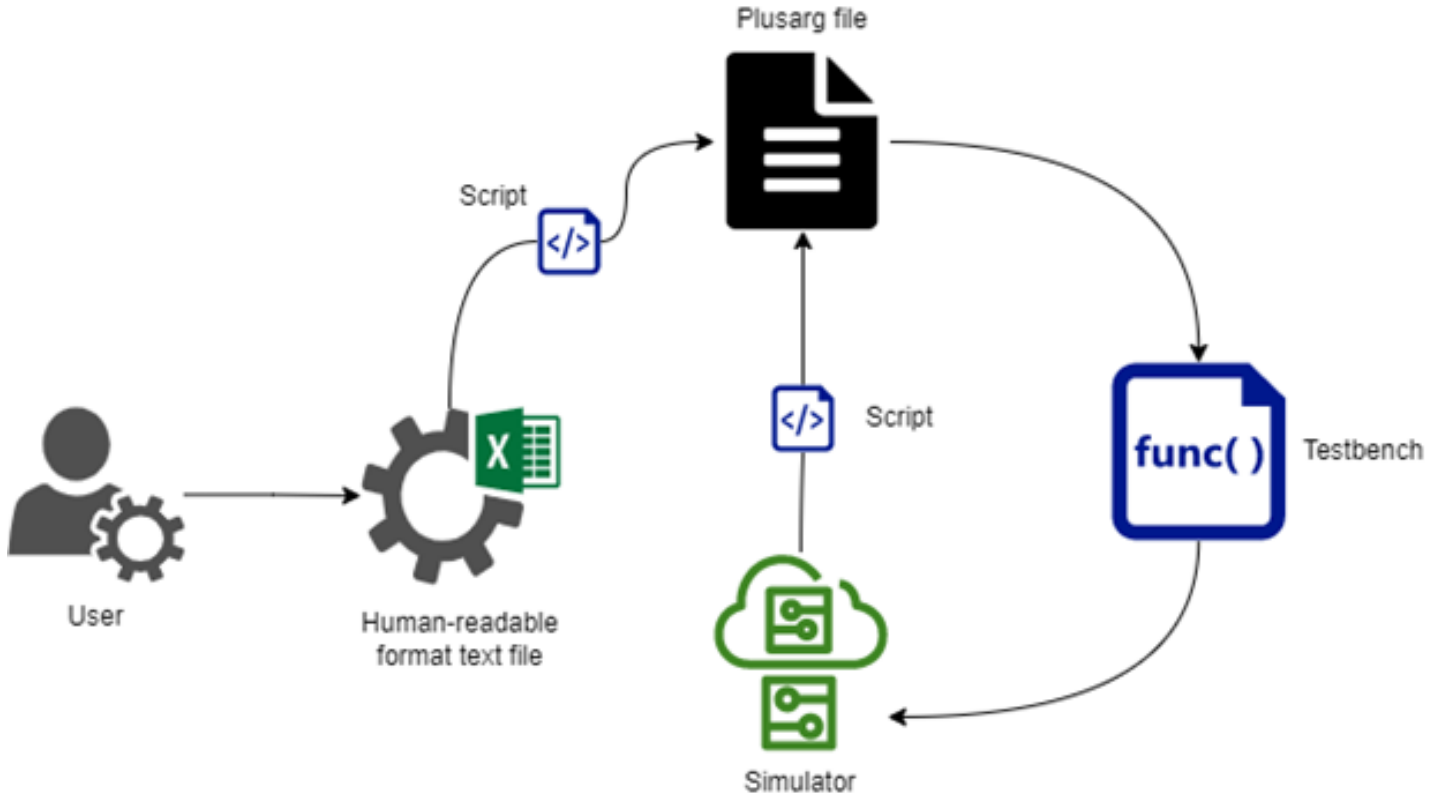
```
+seq0=amiq_dvcon_tb_seq0
+seq0_name=amiq_dvcon_tb_seq0_0
+amiq_dvcon_tb_seq0_0_red_field0_end_0=1024
+amiq_dvcon_tb_seq0_0_red_field0_weight_0=100

+seq1=amiq_dvcon_tb_seq0
+seq1_name=amiq_dvcon_tb_seq0_1
+seq1_p=1
+amiq_dvcon_tb_seq0_1_blue_pkt_nr=3000
+amiq_dvcon_tb_seq0_1_blue_agent_id=0

+seq2=amiq_dvcon_tb_seq0
+seq2_name=amiq_dvcon_tb_seq0_2
+seq2_p=1
+amiq_dvcon_tb_seq0_2_purple_pkt_nr=1500
+amiq_dvcon_tb_seq0_2_purple_field0_end_0=127
+amiq_dvcon_tb_seq0_2_purple_field0_weight_0=50
+amiq_dvcon_tb_seq0_2_purple_field0_start_1=128
+amiq_dvcon_tb_seq0_2_purple_field0_end_1=512
+amiq_dvcon_tb_seq0_2_purple_field0_weight_1=50
```

Text inputs vs tests (2)

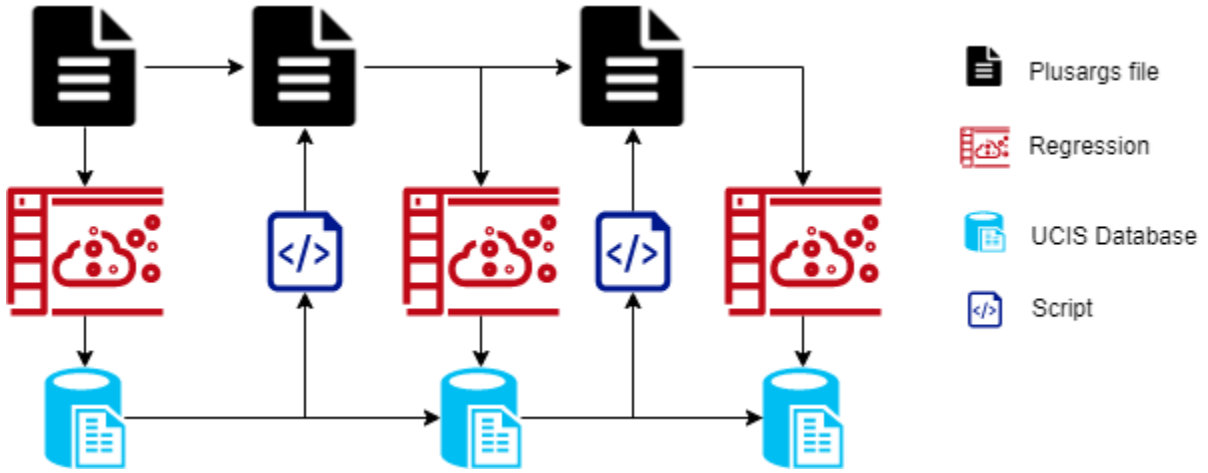
```
+seq0=amiq_dvcon_tb_seq0
+seq0_name=amiq_dvcon_tb_seq0_0
+amiq_dvcon_tb_seq0_0_red_field0_end_0=1024
+amiq_dvcon_tb_seq0_0_red_field0_weight_0=100

+seq1=amiq_dvcon_tb_seq0
+seq1_name=amiq_dvcon_tb_seq0_1
+seq1_p=1
+amiq_dvcon_tb_seq0_1_blue_pkt_nr=3000
+amiq_dvcon_tb_seq0_1_blue_agent_id=0

+seq2=amiq_dvcon_tb_seq0
+seq2_name=amiq_dvcon_tb_seq0_2
+seq2_p=1
+amiq_dvcon_tb_seq0_2_purple_pkt_nr=1500
+amiq_dvcon_tb_seq0_2_purple_field0_end_0=127
+amiq_dvcon_tb_seq0_2_purple_field0_weight_0=50
+amiq_dvcon_tb_seq0_2_purple_field0_start_1=128
+amiq_dvcon_tb_seq0_2_purple_field0_end_1=512
+amiq_dvcon_tb_seq0_2_purple_field0_weight_1=50
```



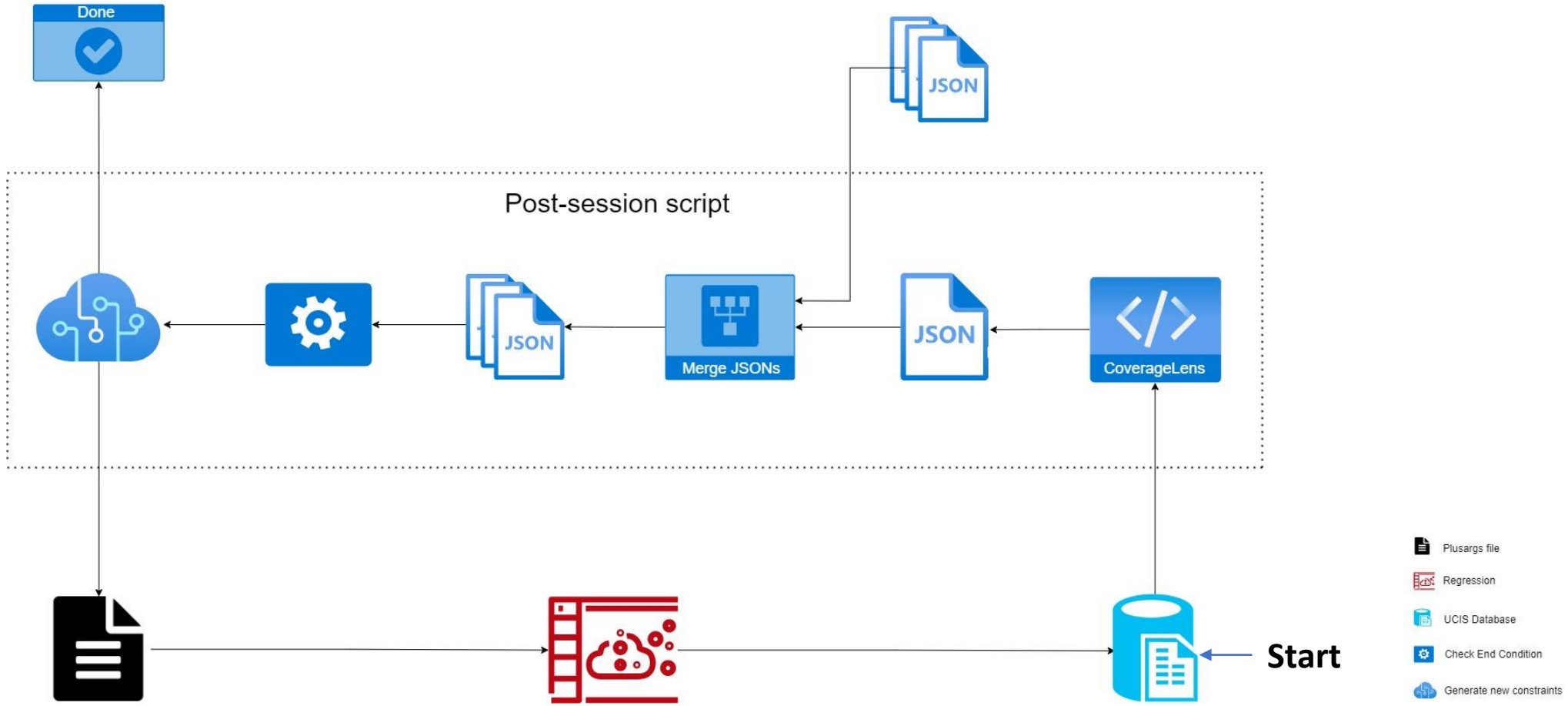
Feedback Loop



Subsequent regressions - Automation



Automation – In-depth

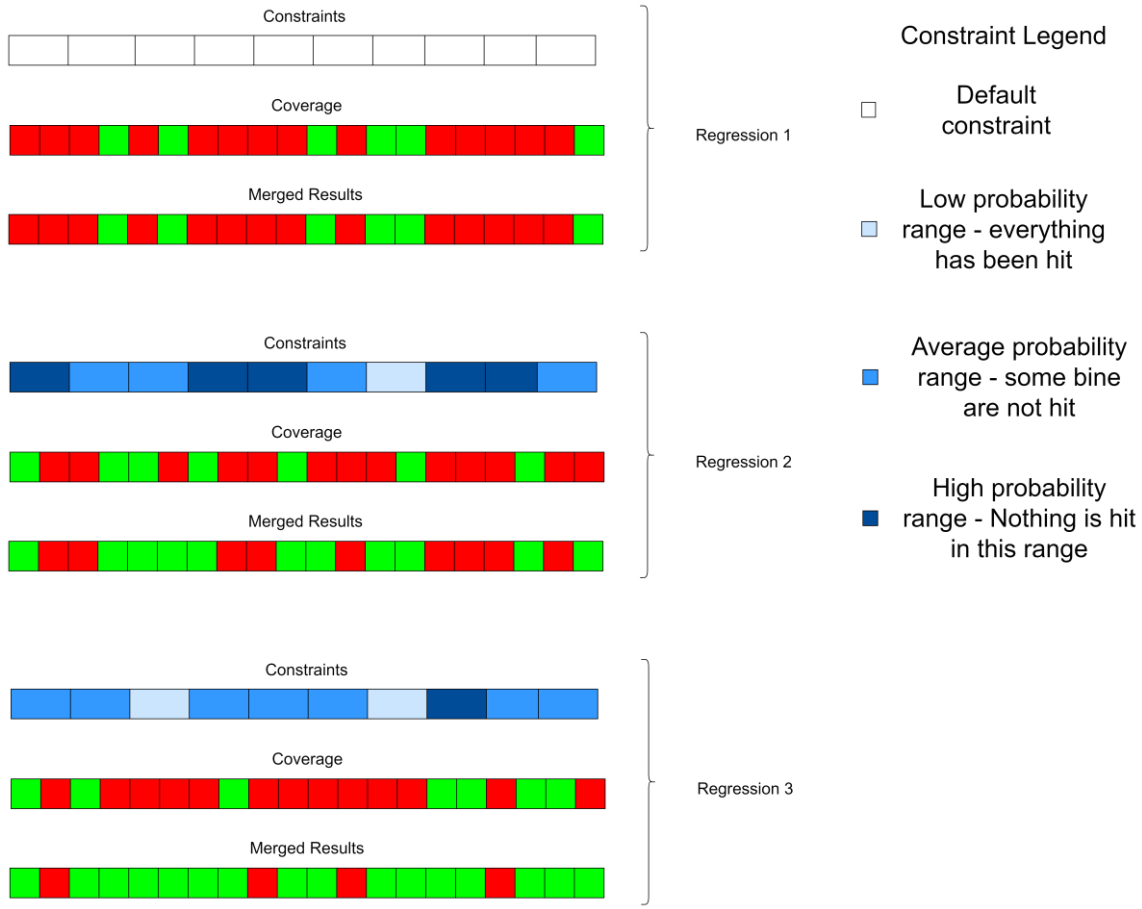


Coverage Lens

- https://www.amiq.com/consulting/2017/07/21/how-to-automate-code-coverage-analysis-with-coverage-lens/
- Open-source UCIS DB parser

← Start

Coverage Closure



Coverage Closure - Scenario

- Case 1
- Int x Int x Int (2**96 rand space)
- Minimum singural values
- Mid equal intervals
- Maximum singural values

- Case 2
- Int x Int x Int
- Scarce values

```
covergroup red0_cg with function sample(amiq_dvcon_tb_vip_red_item red_item);
    option.auto_bin_max=2048;
    option.per_instance = 1;

    red_field0 : coverpoint red_item.field0
    {
        bins low[5] = {0, 1, 2, 3, 4};
        bins med[10] = {[5:max_int-6]};
        bins high[5] = {max_int-5, max_int-4, max_int-3, max_int-2, max_int-1};
    }

    red_field1 : coverpoint red_item.field1
    {
        bins low[5] = {0, 1, 2, 3, 4};
        bins med[10] = {[5:max_int-6]};
        bins high[5] = {max_int-5, max_int-4, max_int-3, max_int-2, max_int-1};
    }

    red_field2 : coverpoint red_item.field2
    {
        bins low[5] = {0, 1, 2, 3, 4};
        bins med[10] = {[5:max_int-6]};
        bins high[5] = {max_int-5, max_int-4, max_int-3, max_int-2, max_int-1};
    }

    red_cross : cross red_field0, red_field1, red_field2;

endgroup : red0_cg
```

Coverage Closure - Results

- Case 1
- No speed-up necessary for 90%
  - Completely random
- Can track changes of coverage and improve chances through constraint optimization
- Closure is guaranteed and can be forced

- Case 2
- Alternate between complete random and directed testcases
- Challenge: Figure out the algorithm

Roadmap

- More options and better IO possibilities
- Testing with other open source coverage parsers
- Improvements to algorithm inclusion process

Conclusions

- Simplicity and ease of use are key to scalabity
- A higher abstraction layer is necessary for automation – Moving out of the simulator space
- PSS has the right idea and the worst delivery
- SV/UVM is still viable, but requires exposure to new tools/algorithms

Resources

- Blog: https://www.amiq.com/consulting/blog/
- AMIQ_ECTB: https://github.com/amiq-consulting/amiq_ectb.git