2026 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

SANTA CLARA, CA, USA
MARCH 2 - 5, 2026

Streamlining RAL-based Cross-Coverage and Sequence Coverage through Automation

Satyajit Sinari (satyajit.j.sinari@intel.com)
Vijayakrishnan Rousseau (vijayakrishnan.rousseau@intel.com)
Ram Immaneni (ram.n.immaneni@intel.com)
Mangayarkarasi Arumugam (mangayarkarasi.arumugam@intel.com)
Prasad Shah (prasad.s.shah@intel.com)
Raghavendra C K (raghavendra.c.k@intel.com)

Agenda

- Motivation
- Challenges in Coverage Automation
- Proposed Solution – GUI Framework
- Workflow Overview
  - GUI Frontend Features
  - Backend & Integration
- Results & Impact
- Enhancements & Future Scope
- Summary & Conclusions

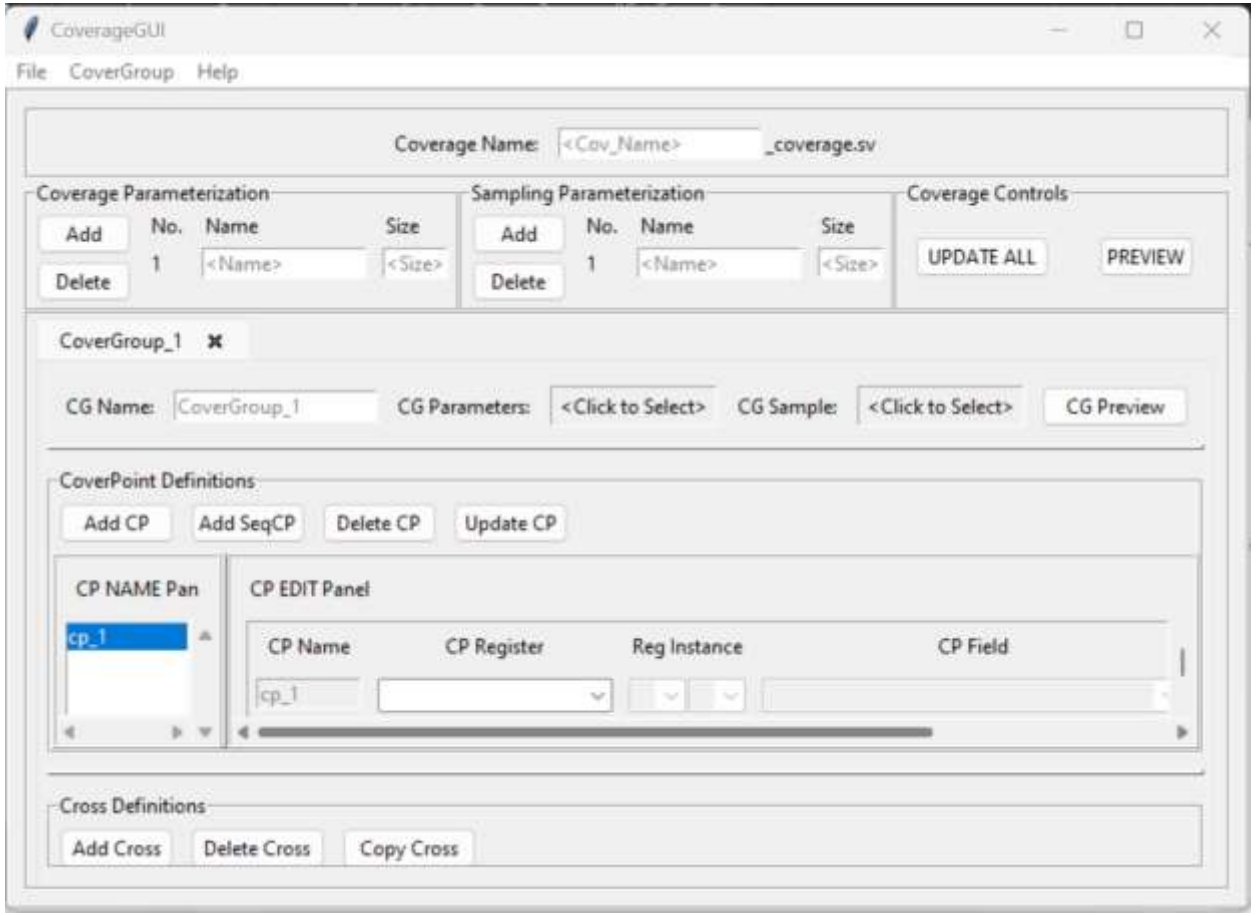
Motivation

- Registers and UVM RAL models are auto-generated; coverage models remain manual.
- Coverage writing is tedious for designs with multiple interoperable features.
- Challenging for complex subsystems, SoCs, and expanding designs.
- Manual coding causes redundancy, inconsistency, and inefficiency.
- Automation boosts efficiency and improves coverage quality.
- Introducing a modular, GUI-based framework for defining, managing, and reusing coverage models.

Challenges in Coverage Automation

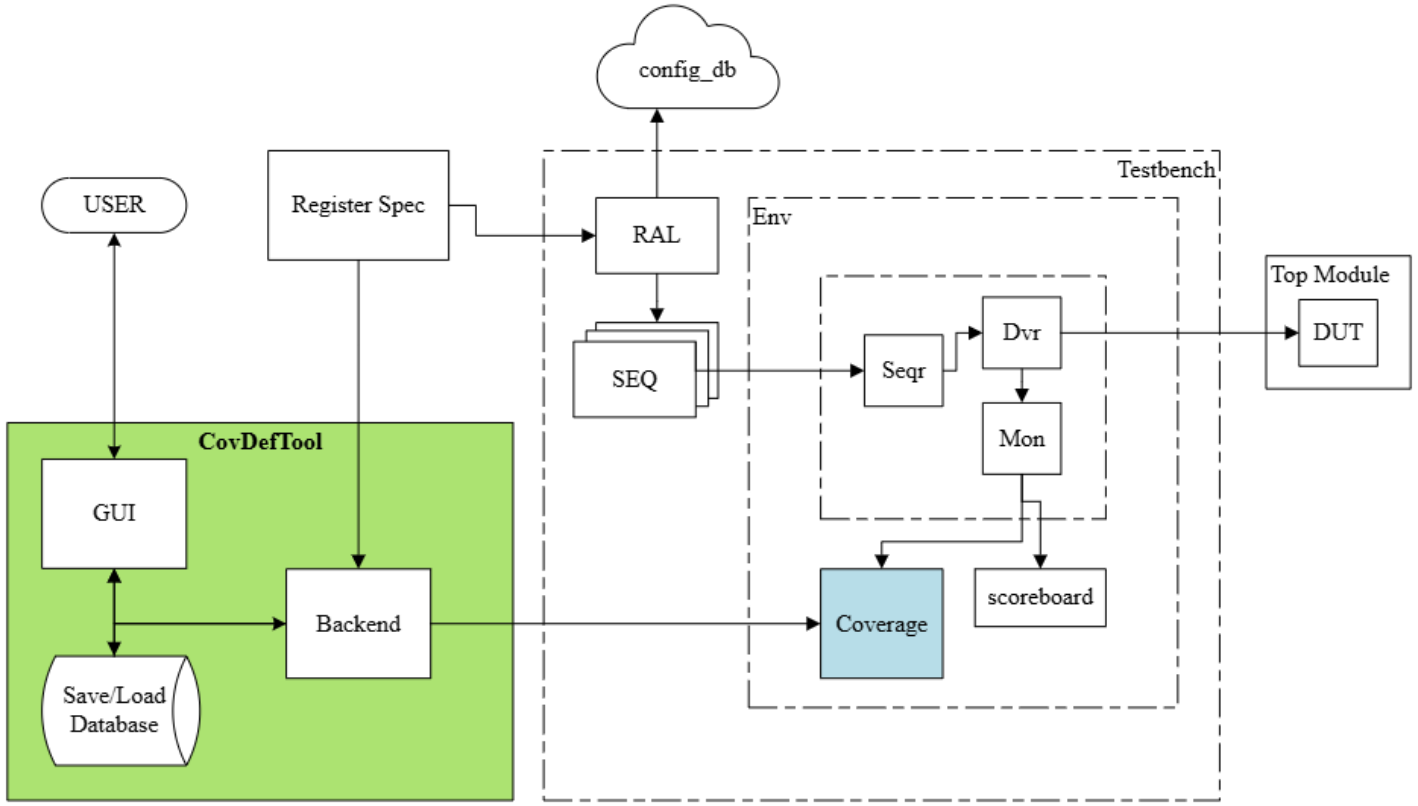
- Automation handles field-level bins, but not complex scenarios
- Manual covergroup development is needed for intricate register interactions and sequences
- Manual process is error-prone and inconsistent, especially with large teams and designs
- Test plan to coverage mapping
- Maintenance over generation of projects

Proposed Solution – Coverage GUI Tool



Proposed Solution – GUI Framework

- Efficiently maps test plan to coverage.
- GUI abstracts manual coding; engineers interact directly with register specs.
- Backend auto-generates RAL and SystemVerilog files from GUI selections.

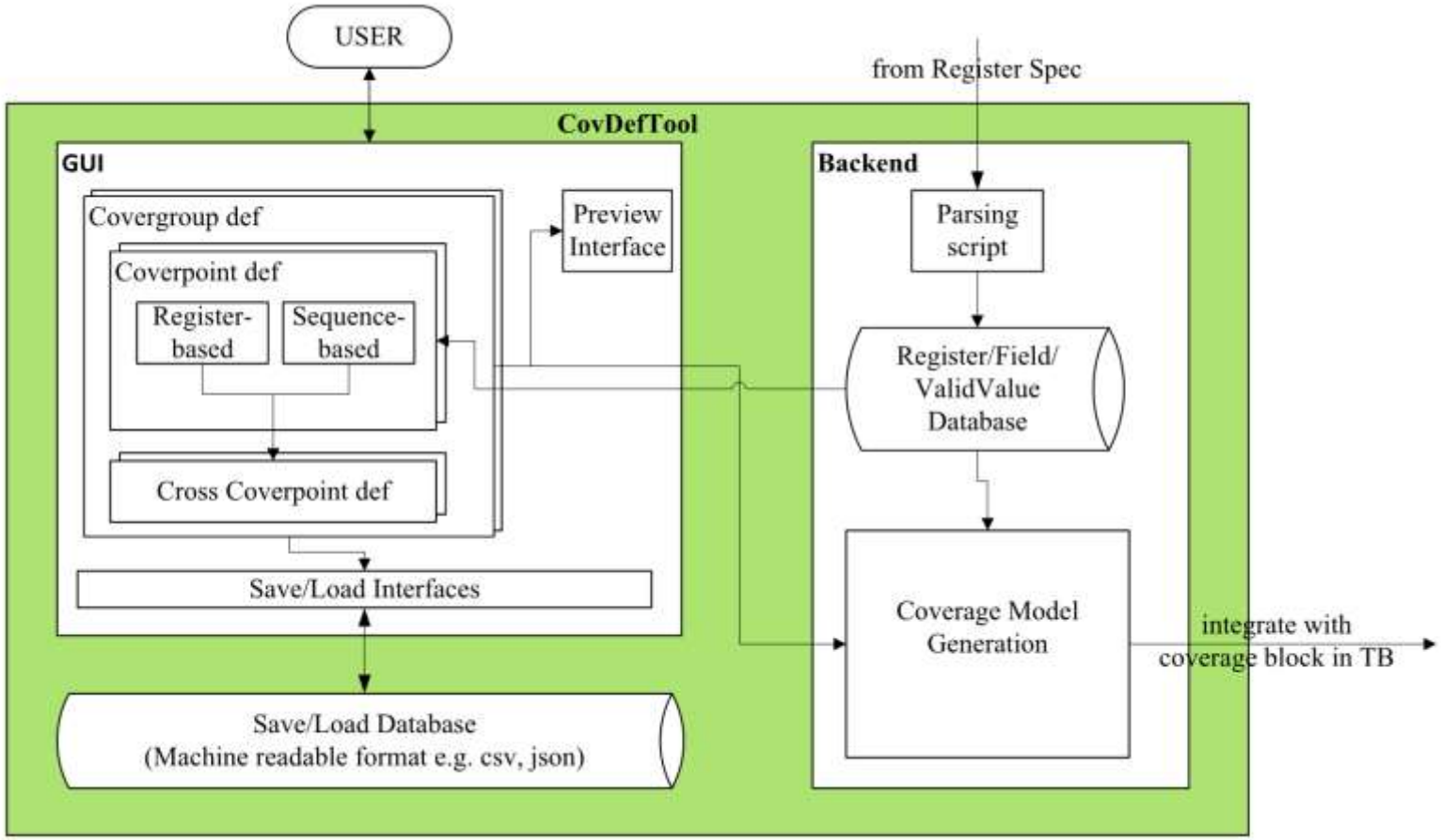


Uniqueness of this Framework

- Coverage models stored as machine-readable artifacts, not ad-hoc SystemVerilog.
- Unified handling of cover points, cross, and sequence coverage.
- Decouples coverage intent from UVM/RAL syntax.
- Enables easy coverage management across multiple project generations.

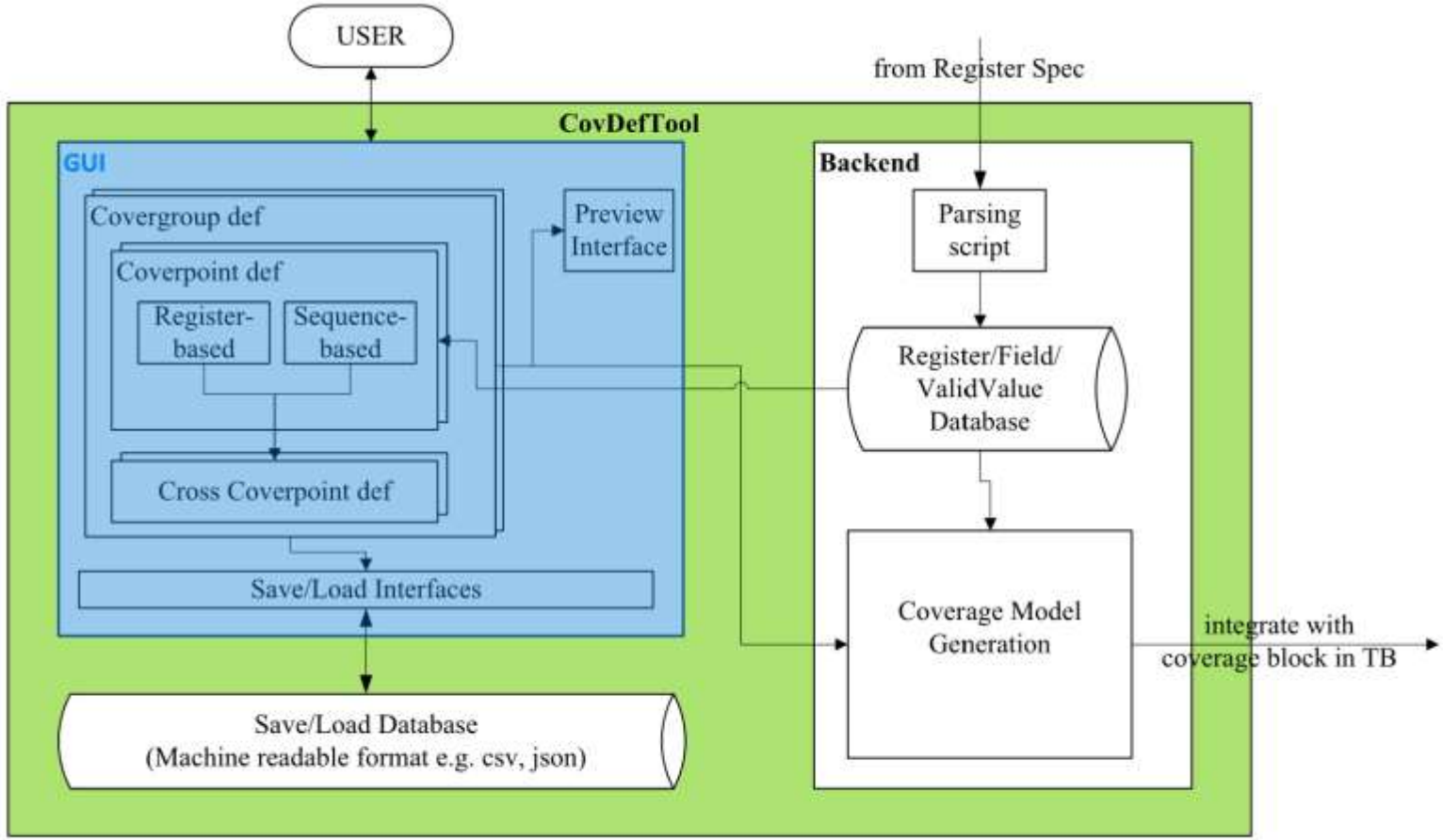
Workflow Overview

- GUI Frontend
- Backend
- Coverage Model Management



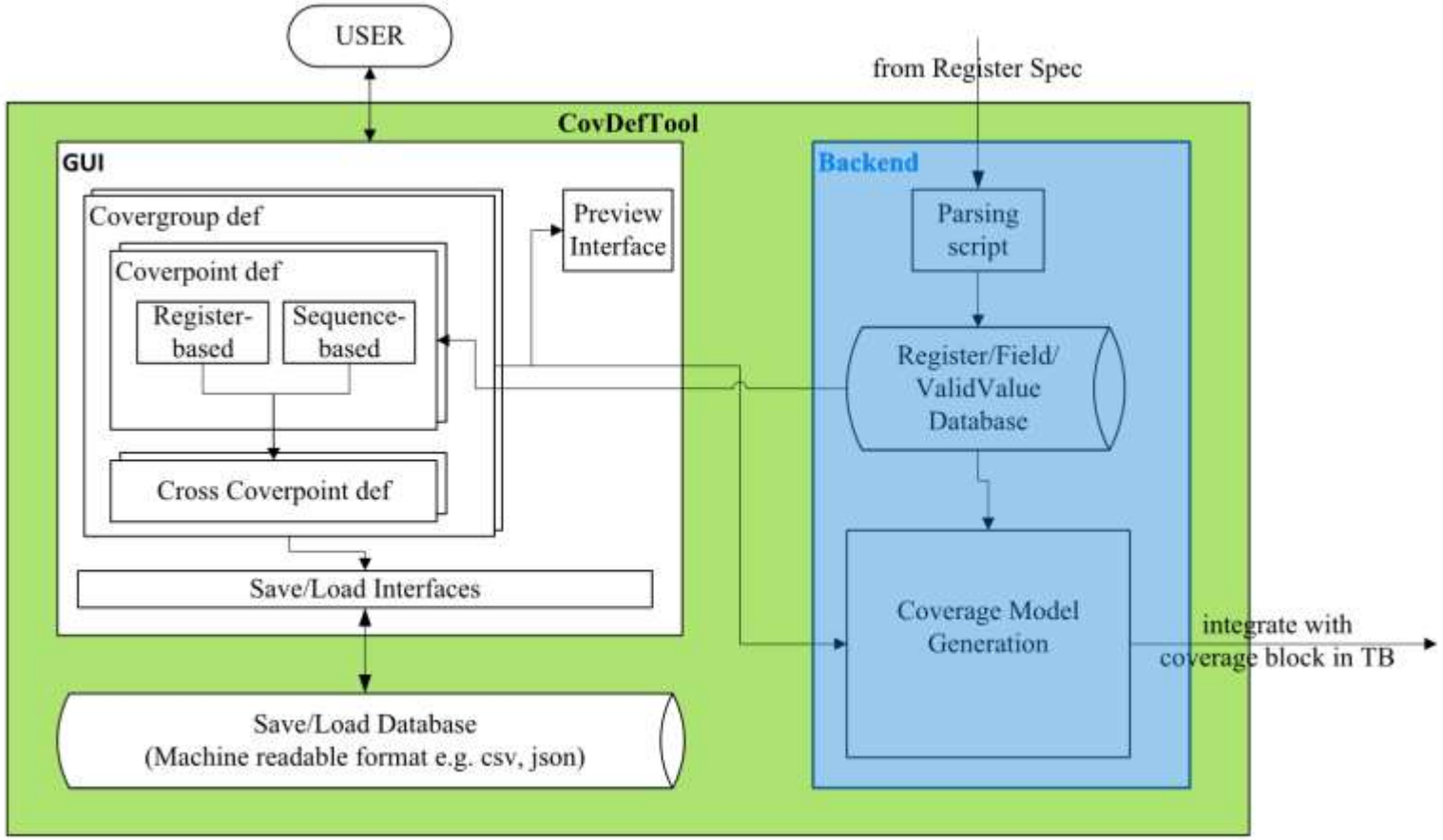
Workflow Overview

- GUI Frontend
  - User interface to query registers
  - add different cover groups, cover points, cross and sequence coverage.



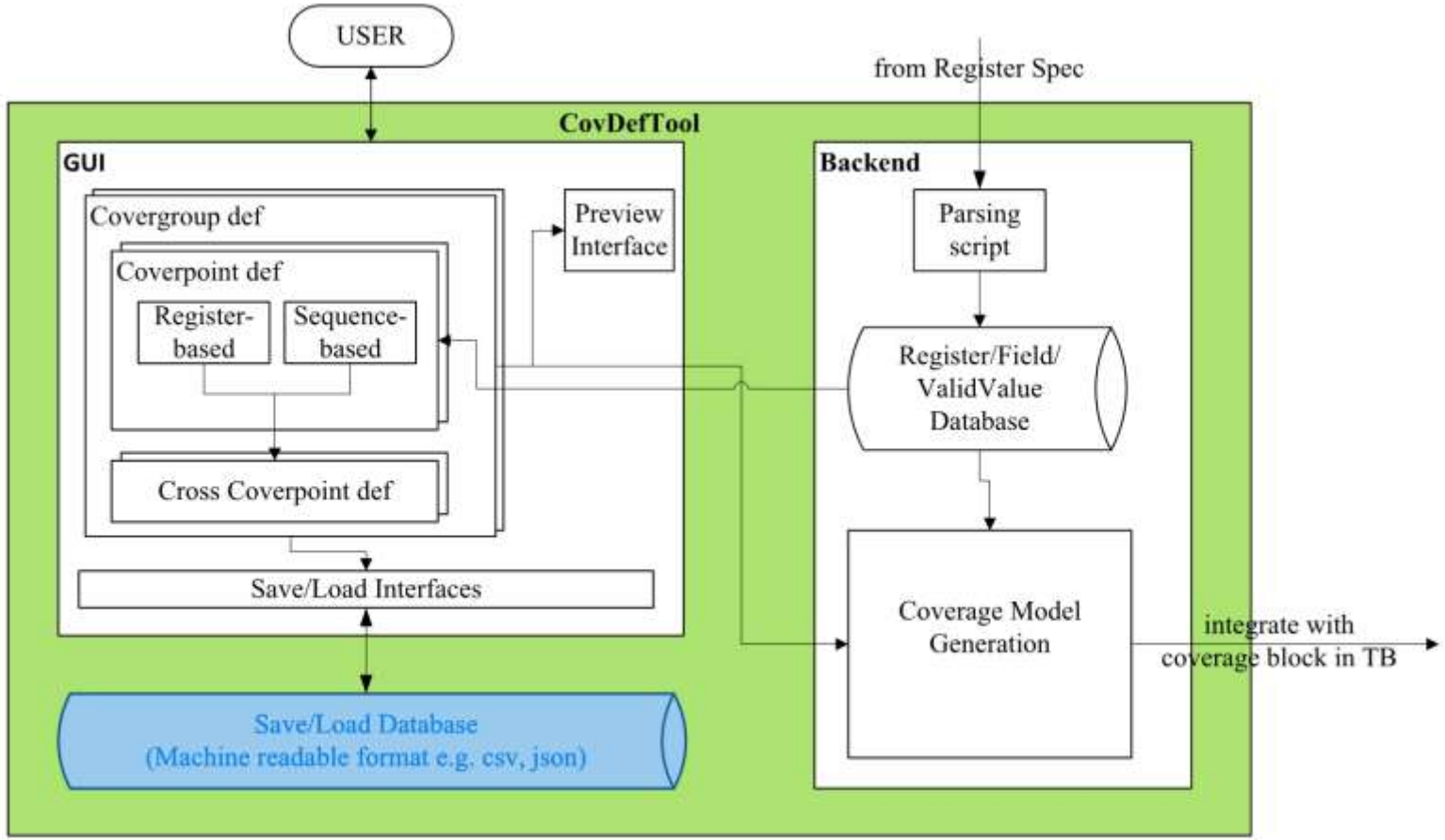
Workflow Overview

- Backend
  - Parses register specs and stores them in an indexed database
  - Generates SystemVerilog coverage classes
  - Integrates coverage model into UVM testbench, editing files as needed



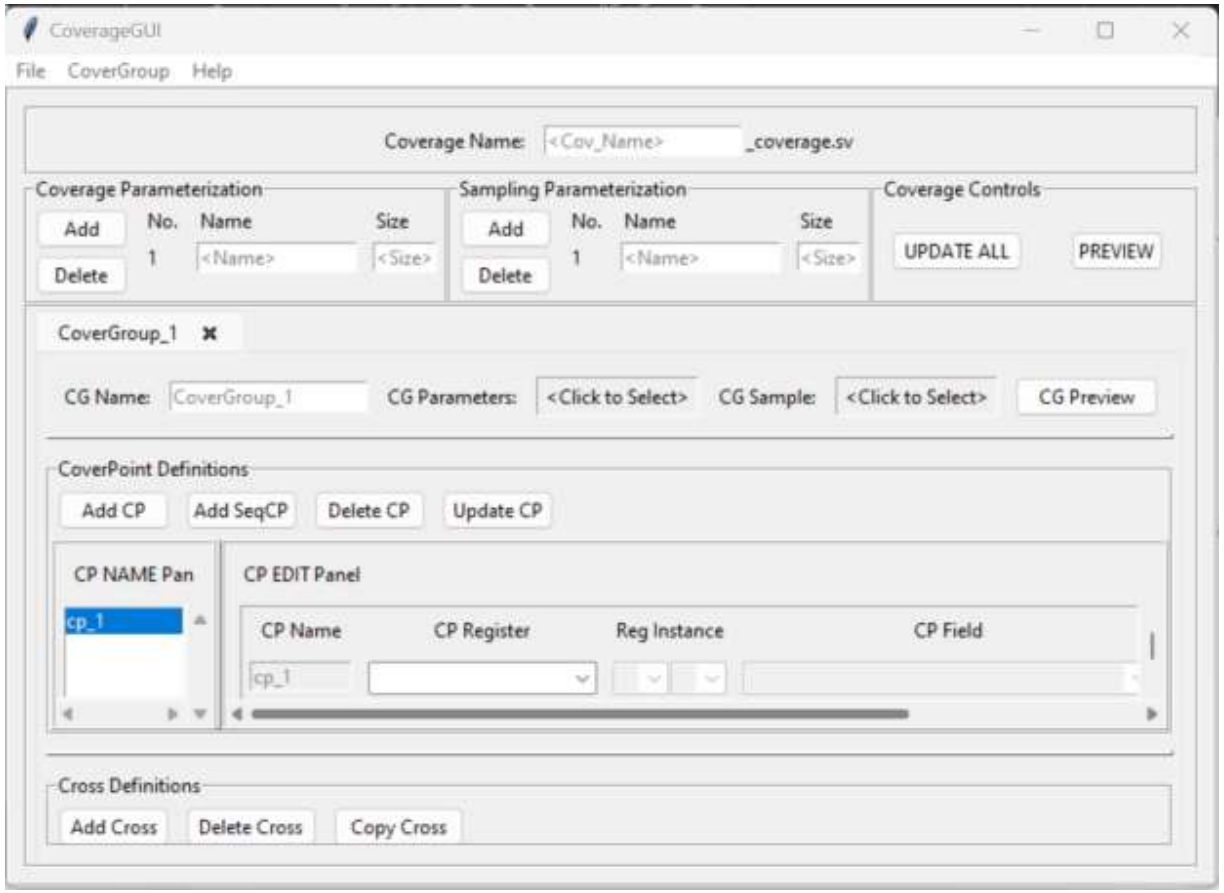
Workflow Overview

- Coverage Model Management
  - Save coverage models in a machine-readable format
  - Load, reuse, extend, or modify preexisting models



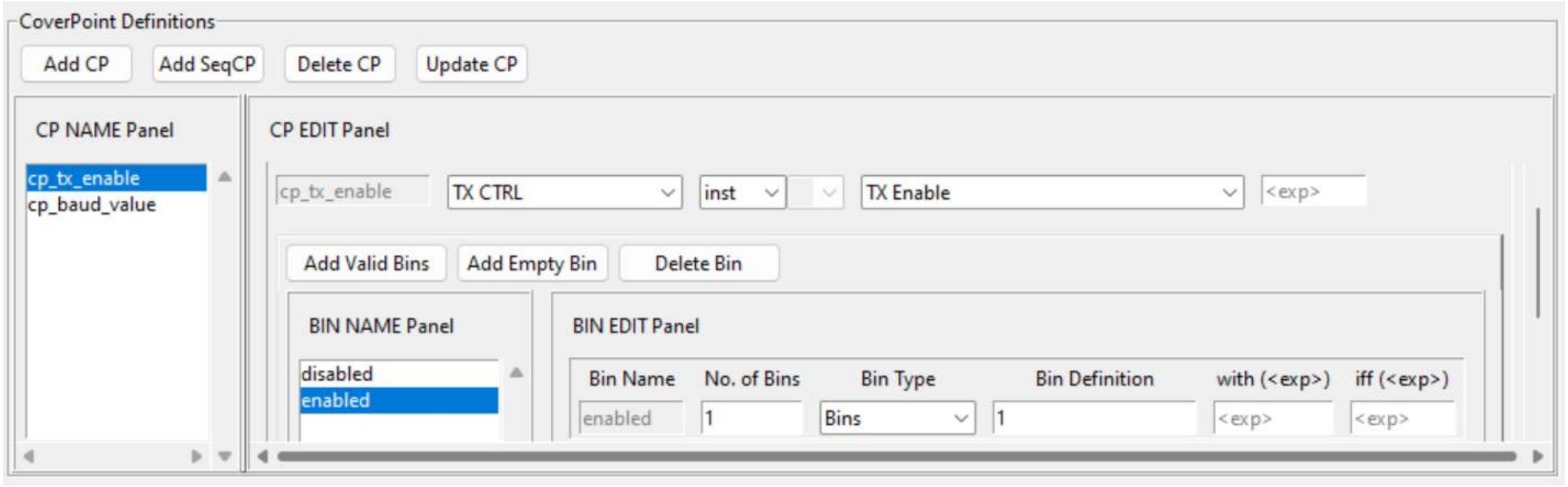
GUI Frontend Features

- Rapid coverpoint definition from register fields
- Register search with wildcard support
- Customizable bins: valid, ignore, illegal
- Cross coverpoint setup via dropdowns
- Sequence coverpoint definition for transaction order
- Preview of generated SystemVerilog code



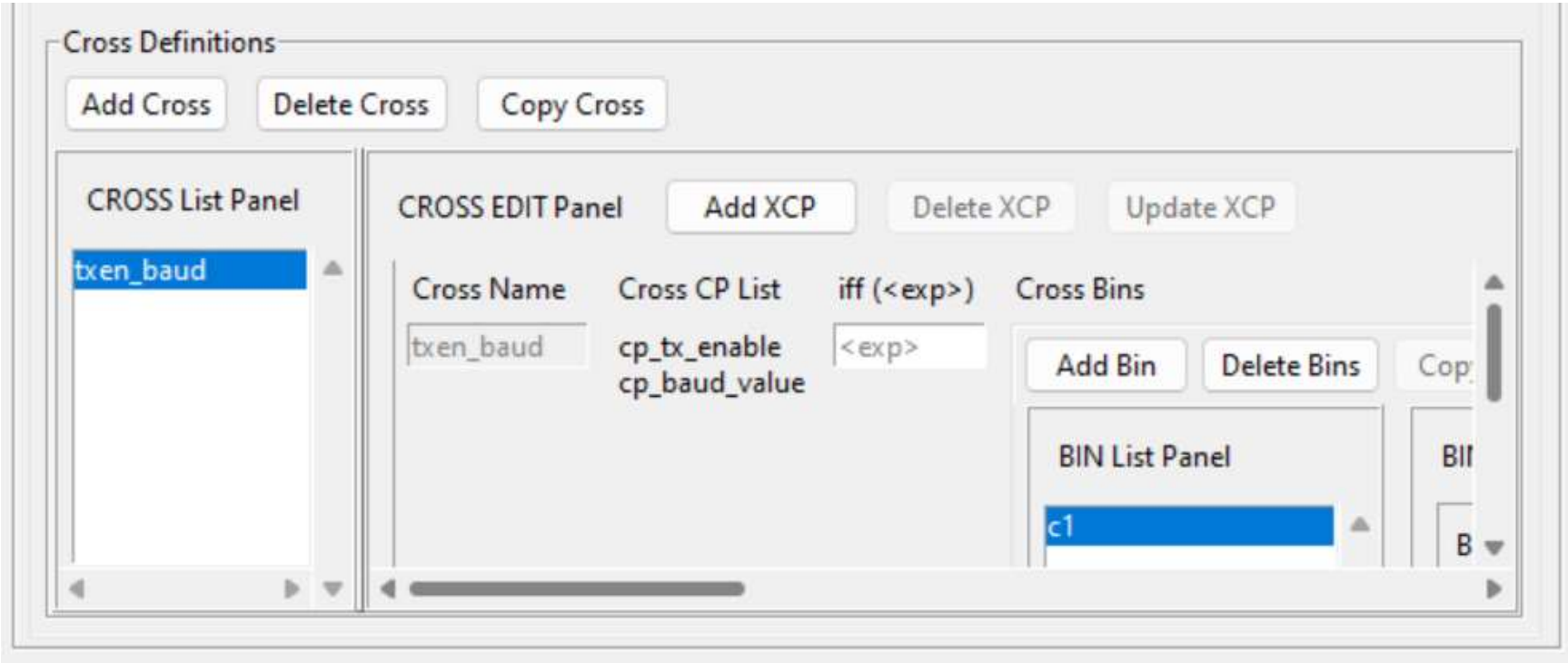
Coverpoint definition

- Select registers from specification and create bins
- Add, delete, or update cover points as needed



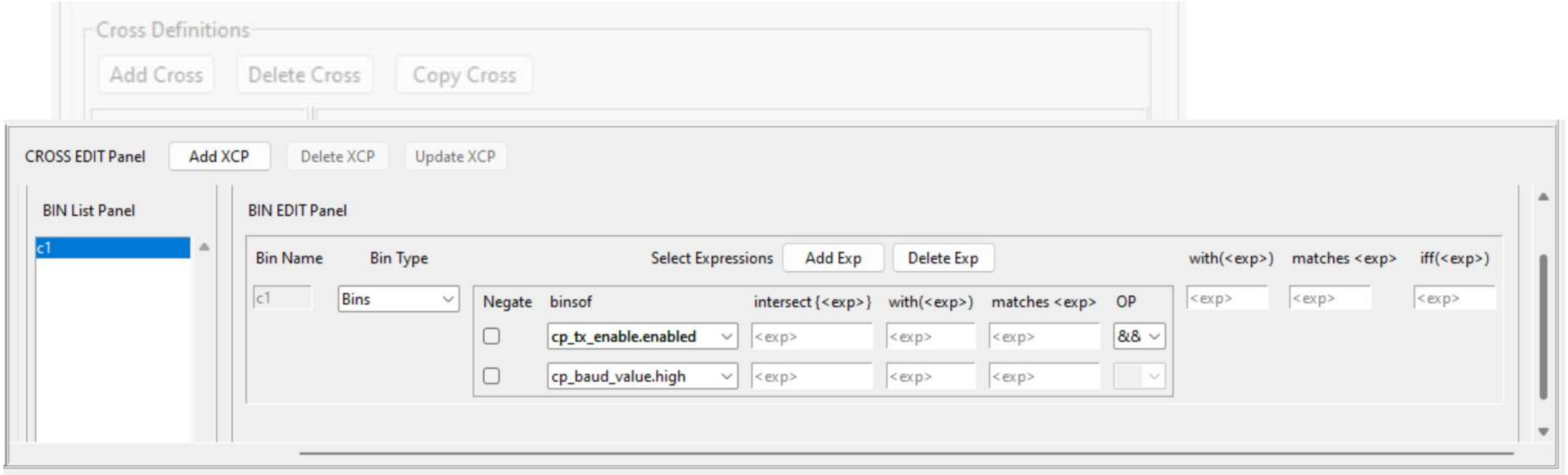
Cross coverpoint definition

- Supports SV LRM syntax keywords: negate, intersect, with, matches, iff
- Enables AND/OR operations between expressions when creating bins



Cross coverpoint definition

- Supports SV LRM syntax keywords: negate, intersect, with, matches, iff
- Enables AND/OR operations between expressions when creating bins

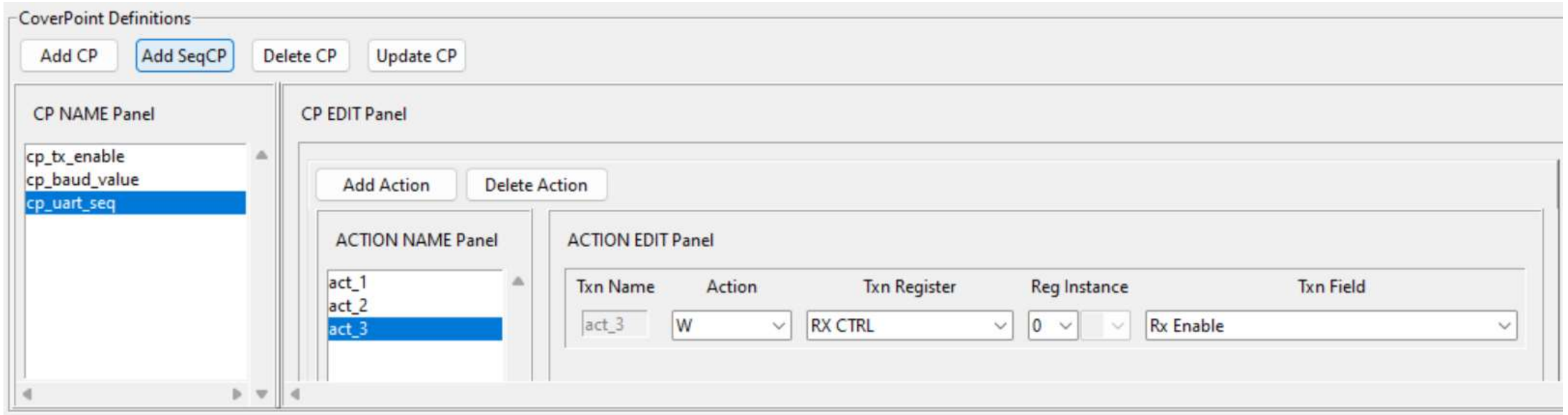


Covergroup code preview

```
covergroup CoverGroup_1 (int inst) ;
    //Enabled option to track coverage for each instance of covergroup
    option.per_instance = 1;

    // Coverpoint for TX_CTRL Register/TX Enable Field
    cp_tx_enable: coverpoint ral_reg.tx_ctrl_reg[inst].tx_enable.value {
        bins disabled = {0};
        bins enabled  = {1};
    }

    // Coverpoint for BAUD Register/Baud Value Field
    cp baud value: coverpoint ral_reg.baud_reg[inst] .baud_value. value {
        bins low    = {[0:9600]};
        bins mid    = {[9601:115200]};
        bins high   = {[115201:$]};
    }

    //=================CROSS Definitions========= //
    txen baud: cross cp_tx_enable, cp_baud value {
        bins c1 = binsof (cp_tx_enable.enabled) && binsof (cp_baud_value.high);
    }

endgroup
```

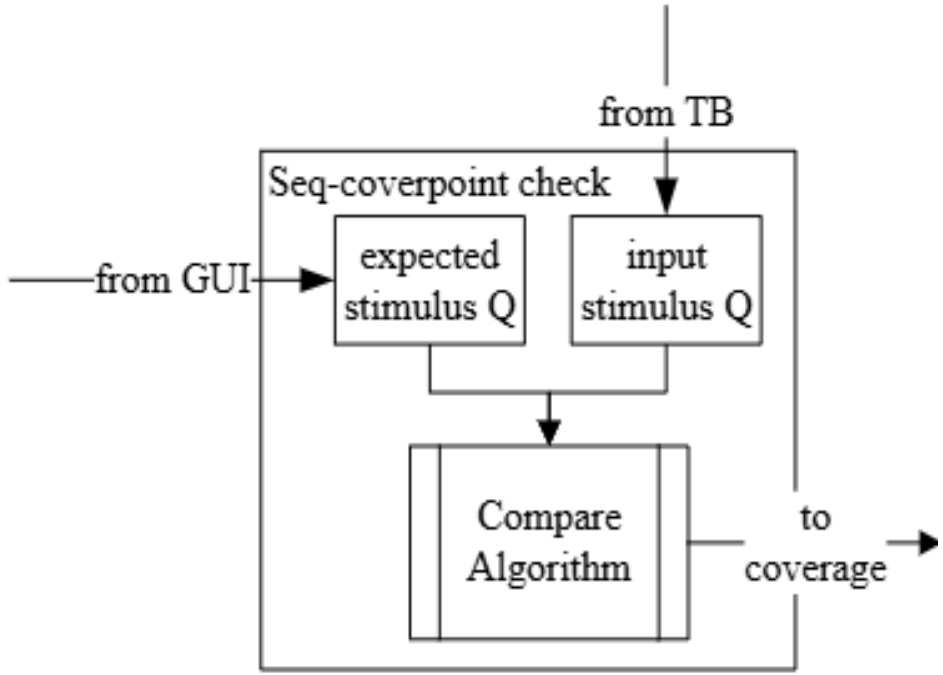
Sequence coverpoint definition

- Add multiple register actions in order
- Supports writes, reads, polls



Sequence-based Coverage Flow

- Track specific register transaction patterns
- GUI enables users to define expected sequences
- Backend compares expected vs. actual stimulus queues
- Supports unordered sequences using guarc transactions
- Enables coverage for complex scenarios beyond simple transitions



Sequence coverpoint code preview

```
//Generated using GUI backend
covergroup sample1_sequence(int inst);
    // Track individual enable bits for each instance
    cp_uart_seq: coverpoint check_uart_sequence(inst) {  bins enabled = {1'b1}; }
endgroup

//Generated using GUI backend
function bit check_uart_sequence(int inst);
    reg_action_def exp_txn_q[$];

    exp_txn_q.push_back('{Action:WRITE_REG, Reg: ral_reg.baud_reg[inst],    Field: ral_reg.baud_reg[inst].baud_rate,    Value:16'h1c200});
    exp_txn_q.push_back('{Action:WRITE_REG, Reg: ral_reg.tx_ctrl_reg[inst], Field: null,                                Value:32'h3});
    exp_txn_q.push_back('{Action:WRITE_REG, Reg: ral_reg.rx_ctrl_reg[inst], Field: null,                                Value:32'h3});
    exp_txn_q.push_back('{Action:POLL_REG,  Reg: ral_reg.status_reg[inst],  Field: ral_reg.status_reg[inst].error,      Value:1'h0});
    exp_txn_q.push_back('{Action:WRITE_REG, Reg: ral_reg.tx_ctrl_reg[inst], Field: ral_reg.tx_ctrl_reg[inst].tx_enable, Value:1'h0});

    // Check if expected sequence is subset of input_stimulus
    return is_sequence_subset(exp_txn_q);
endfunction
```

Results & Impact

- GUI tool deployed across multiple engineers.
- 71% reduction in development time (9 man-days vs. 32).
- 83% reduction in code defects.
- Discovery of previously undetected corner case bug.
- Improved coverage definition, accuracy, and verification thoroughness.

Enhancements & Future Scope

- Scaling to SoC-level environments
  - Abstraction
  - configurable interfaces.
- Register loading for huge designs
  - On demand
  - advanced search/tagging/filtering
  - Logical grouping
  - bin explosion management
- Assertion/monitor integration for correct sampling.
- Potential extension to non-UVM environments.

Summary & Conclusions

- GUI-based framework for robust coverage modelling and validation.
- Significant gains in efficiency, accuracy, and bug detection.
- Supports robust validation for complex hardware designs.

“Coverage intent should be authored once, reused everywhere, and survive design evolution”

Q&A

Thanks