2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

SVRAND – Random Configuration – One class to resolve all parts

Kaushal Vala, Krunal Kapadiya, Joseph Bauer, Shyam Sharma
Cadence Design Systems (India, USA)
(kaushalm@cadence.com, krunalk@cadence.com, jbauer@cadence.com, ssharma@cadence.com)

***Abstract*- Cadence's Memory Models use configuration files to uniquely describe the characterization attributes of every legal and real memory device described in protocol standards and vendor data sheet. The Memory Model performs timing checks and protocol responses according to the combination of attribute settings in each device file. As the number of these memory device configurations increases per memory type, a significant challenge arises for the EDA provider and for the design verification engineers using them. Memory part offerings even within one protocol have increased into the tens of thousands while the number of characterization attributes describing a memory and the dependencies between them increases with each protocol generation. Moreover, the set of memory part offerings and attributes evolves over many version updates of the protocol standard and vendor data sheets during user's product development. The EDA provider is challenged to provide and maintain accurate updates for all the evolving real part configurations. In conjunction, the user's challenge is to validate that their SoC's (System on Chip) memory sub-system is compatible across the applicable memory part variations their SoC can externally connect to. To ensure compatibility as evolving updates are taken at regular intervals, the user has had to repeat their process steps to manage their repository snapshot of configurations while incorporating them into their test environment's part selection and coverage tracking.**

A new widely adopted solution is available that represents all parts as one class file while simplifying the selection and coverage scoping to the right level of memory differentiating attributes aligned to the specification, vendor, and application. This "SystemVerilog constraint random configuration (SVRAND)" flexible solution represents all valid parts in a single SystemVerilog class of constraints which resolves evenly across user's application scope of required configurations, while simultaneously providing compatibility coverage for closure over that same scoped set of parts. This constraints class, representing the relational intersection of all configuration settings relative to these memory differentiating attributes, is auto generated from a human friendly form for ease maintaining specification alignment.

***Keywords*- functional verification; verification; memory model; systemverilog constraint; randomization;**

I. Related work

While the "SystemVerilog constraint random configuration (SVRAND)" has been applied to various protocols, we can use as example the DDR5 DIMM memory protocol to illustrate the challenge faced in the prior art. The variety of memory parts offered for a given protocol is determined by certain part differentiating configurations, and this combinational set can be huge. For example, considering the part differentiating configurations below, DDR5 DIMM has 23 thousand parts of JEDEC and other vendors.

15 Operating Speeds x 5 Densities x 5 Stack-heights x 3 Data-widths x 2 DIMM-widths x 4 DIMM Types x 2 Rank

Memory differentiating attributes determines the settings on the many, often hundreds, of functional and physical characteristics that accurately represents a legal real each part in digital simulations. Directly maintaining hundreds of attribute key-value sets across this high number of configuration files is time-consuming and error prone, but necessary for qualifying the memory sub-systems in System-on-Chips (SoCs) for functional compatibility to these parts requires compliance testing across these many configuration combinations. The real and relevant space of part configurations is determined by these memory differentiating attributes since not all combinations of valid attribute values are legal or applicable to an application's target space, and ignoring legal applicable configurations is not good practice.

A challenge for users is the effort, time, and resources to:

- Maintain the iterative download of EDA's provided files of "memory configurations SOMA files (Specification Of Memory Architecture)."
- Integrate SOMA files into their environment and validation flow.
- Target applicable configurations scoped within the set of all valid configurations.

This effort would include:

- Convert all "SOMA" files to native format such as SystemVerilog (SV) device classes.
- Compile all the SV device classes.
- Form an SV object queue of the many "SOMA" (for iteration and/or query).
- Form valid queries to meet some selection criteria (valid within available parameters and values and inter-dependencies).
- Query search the queue for the applicable matches which is still iterative.
- Evenly select one amongst the matched queue items.
- Provide and enable coverage tracking to refine targeted scope.

For an emerging memory protocols like DDR5, specification and ballot updates come frequently from the standards body which include timing and feature changes affecting these configurations. These repeated updates again require tedious efforts as just mentioned.

II. APPLICATION

The "SystemVerilog constraint random configuration" solution automates the generation of these many part-configurations. Instead of maintaining the many thousands of configuration files, one metadata file is now maintained. The metadata file contains the same parameter set as listed in the many "SOMAS," but instead of a single value associated to each parameter, the set of all possible values are listed, AND for each value any dependency the noted memory differentiating attributes. With every new revision of the protocol specification, it becomes much easier to maintain the one metadata file and with less opportunity for flaws. The full set of parts ("SOMA") and part coverage can be generated from this metadata by unfolding these dependency connections, i.e., every value combination on the small set of memory differentiating attributes provides the 'key' to context align the intersect of a value across the many, often hundreds, of functional and physical memory characterization attributes.

This metadata representation of parameters values and interdependences to the small set of memory differentiating attributes is also auto converted into an equivalent SystemVerilog representative set of valid value constraints and legal implication constraints we call SVRAND, the "SystemVerilog constraint random memory configurator class" (with examples shared shortly).

The 'key' with "SystemVerilog constraint random configuration" and "SOMA" automation is deriving real and legal parameter value combinations from dependencies to a small set of memory differentiating attributes. We refer to these selection attributes as 'High-Level Parameters,' (HLPs) which for DDR5 include [Speed, density, width, etc.] i.e., the list mentioned earlier. Users can scope their criteria for part selection on these 'High-Level Parameters' and the constraints will resolve dependencies evenly to only 'real' legal parameter value combinations.

III. RESULTS

The existing "SOMA" flow has a tedious pre-simulation task to collect all the parts available either through batch or manual configuration download from eMemory website. These downloaded parts must all be converted into

SystemVerilog device class files, compiled, queued, and queried during simulation to select an applicable configuration.

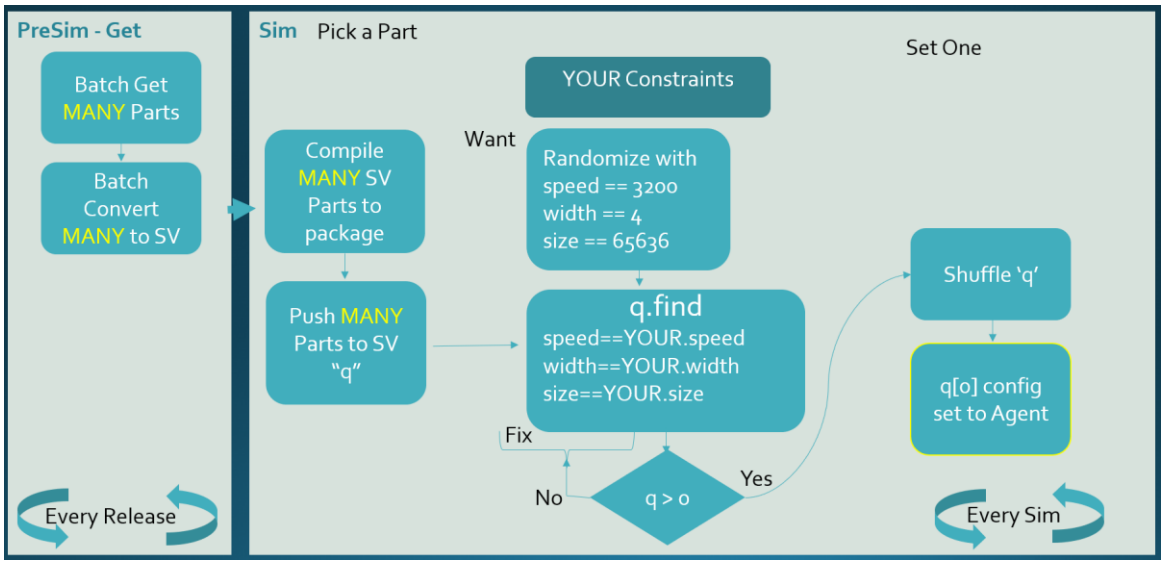

Figure 1. Comparison of existing "SOMA" flow versus new "SV-Rand" solution

For the new SVRAND solution, only one SystemVerilog constraint class file is required. Subsets or specific parts within the space of all real and legal configurations can be targeted in a 'randomize' method call having in-line constraints on these 'High-Level Parameters.'

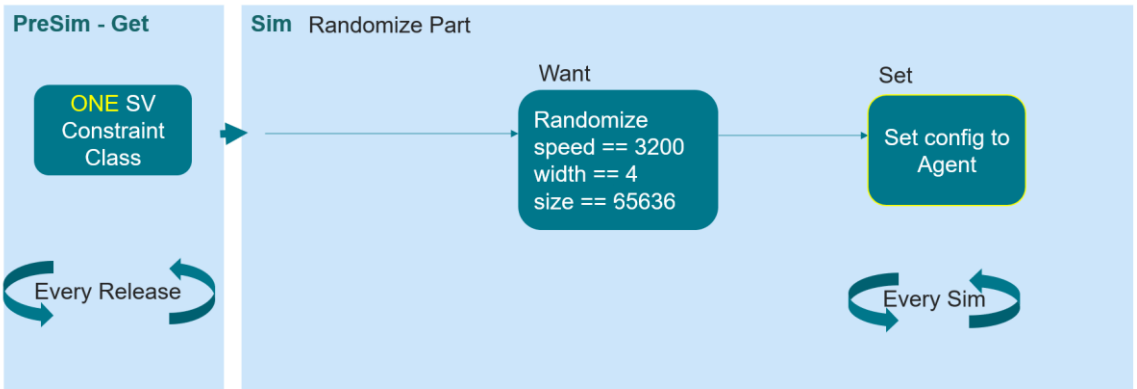

Figure 2. Randomization of SV-Rand to derive a part.

Since the HLP combinational value sets map 1:1 to legal real parts, the relational metadata also unfolds to provide comprehensive memory part coverage in SVRAND for compatibility closure, e.g., DDR5 SDRAM:

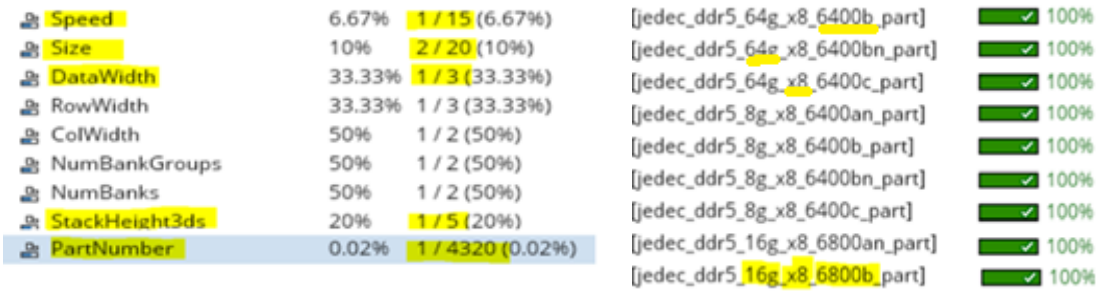

Figure 3. DDR5 SDRAM part configuration coverage

A. Query Part Select Example:

The traditional SOMA approach requires maintaining and compiling many part classes and then creating an object queue of all these valid parts to verify all possible valid configurations, in the case of DDR5-DIMM more than thousands of valid configurations needs to be verified. And still a unique HLP value set per part must be correctly added into each part class as selection criteria for user to query on.

Input: DIMM-settings is randomized with parts which only pushed to queue.

```
dimmSettings settings;
assert( settings.randomize() with {
                                    speed == spd ;
                                    numRanks == 2;
                                    dramDataWidth == 4;
                                    size == 524288;
                                    subClass == UDIMM;
                                    compsIn3dsStack == 1;
                                    }
        );
return settings.get_matching_part(settings);
```

Output: DIMM part is selected as it only existed in queue when queue size is more than zero.

```
Query part selected = jedec_ddr5udimm_64gbyte_2r_x4_3200bn, udimm, 2r, 3200, x64, x4, size=524288
```

B. SVRAND Solution Example:

Above traditional SOMA approach of putting all valid configurations into a queue for part selection means maintaining, compiling, and queuing many configurations, e.g., DDR5 DIMM has tens of thousands. Whereas the new solution is one class object for a user to randomize and scope applicable configurations as needed to High-Level Parameters.

For example:

1. To limit selection to a single valid configuration - in-line value constrain all High-Level Parameters:

Input: DIMM selection is randomized on high level parameters

```
JEDEC_RandSettings = new();
assert( JEDEC_RandSettings.randomize() with {
                                        kind == lrdimm;
                                        density == 128;
                                        prank == 2;
                                        width == 4;
                                        lrank == 1; // non-3ds
                                        rate == an3200;
                                        eccType == withEcc;
                                    } );
$cast( randSettings, JEDEC_RandSettings);
```

Output: Resolves DIMM part

```
jedec_ddr5lrdimm_128gbyte_2r_x4_3200c
```

2. To wildcard or widen the selection criteria such that 'randomize' resolves – Constrain some High-Level Parameters or none.

Input: DIMM selection randomized On High Level Parameters

```
JEDEC_RandSettings = new();
assert( JEDEC_RandSettings.randomize() with {
                                        kind == lrdimm;
                                        density == 128;
                                        prank == 2;
                                        width == 4;
                                        lrank == 1; // non-3ds
                                        rate inside {an3200,b3200,bn3200,c3200} ;
                                        eccType == withEcc;
                                    } );
$cast( randSettings, JEDEC_RandSettings);
```

Output: Resolves DIMM part within a range of devices

```
jedec_ddr5lrdimm_128gbyte_2r_x4_3200an
jedec_ddr5lrdimm_128gbyte_2r_x4_3200b
jedec_ddr5lrdimm_128gbyte_2r_x4_3200bn
jedec_ddr5lrdimm_128gbyte_2r_x4_3200c
```

IV. CONCLUSION

The traditional approach of "SOMA" flow demands significant efforts, time, and compute resources.

The new "SVRAND flow provides:

- Effective, optimized, and user-friendly way to verify thousands of valid configurations.
- Widely adopted in industry.
- Overall configuration tasks cut by 30%.
- Simplified Maintenance from both sides, EDA, and SOC.
- Intuitive, comprehensive, and fast part selection.
- Part compatibility coverage closure measured.

REFERENCES

[1] JEDEC. (2024). *DDR5 SDRAM* (JESD79-5C ed.). https://www.jedec.org/standards-documents/docs/jesd79-5c

[2] JEDEC. (2023b). *LOW POWER DOUBLE DATA RATE (LPDDR) 5/5X* (JESD209-5C ed.). https://www.jedec.org/document_search?search_api_views_fulltext=lpddr5

[3] Cadence Xcelium Logic simulator Xcelium Logic Simulator | Cadence

[4] Cadence memory model VIPs Memory Models | Cadence