Four Problems with Policy-Based Constraints and How to Fix Them

Dillan Mills
Synopsys, Inc.
Maricopa, AZ
dillan@synopsys.com

Chip Haldane
The Chip Abides, LLC
Gilbert, AZ
chip@thechipabides.com

Abstract

This paper presents solutions to problems encountered in the implementation of policy classes for SystemVerilog constraint layering. Policy classes provide portable and reusable constraints that can be mixed and matched into the object being randomized. There have been many papers and presentations on policy classes since the original presentation by John Dickol at DVCon 2015. The paper addresses three problems shared by all public policy class implementations and presents a solution to a fourth problem. The proposed solutions introduce policy class inheritance, tightly pair policy definitions with the class they constrain, reduce the expense of defining common policies using macros, and demonstrate how to treat policies as disposable and lightweight objects. The paper concludes that the proposed solution improves the usability and efficiency of policy classes for SystemVerilog constraint layering.

I. Constraints and Policy Class Review

When working with randomized objects in SystemVerilog, it is typically desired to control the randomization of the item being randomized. The common method of doing so is by using constrained random value generation. Constraints for the randomization of class properties can be provided within the class definition, which will require randomized properties to always pass the specified constaint check. For example, the following **addr_txn** class contains a constraint on the **addr** property to ensure it is always word aligned.

```
class addr_txn;
    rand bit [31:0] addr;
    rand int        size;

    constraint c_size {size inside {1, 2, 4};}
endclass
```

Constraints can also be provided outside the class definition at the point of randomization by using the **with** construct. This allows variation to the constraints based on the context of the randomization. For example, the same **addr_txn** class can have an instance of it randomized with the following constraint on its **data** property.

```
addr_txn txn = new();
txn.randomize() with {addr == 32'hdeadbeef;};
```

Policy classes are a technique for SystemVerilog constraint layering with comparable performance to traditional constraint methods, while additionally providing portable and reusable constraints that can be mixed and matched into the object being randomized, originally presented by John Dickol [1] [2]. This is possible through SystemVerilog “Global Constraints”, which randomizes all lower-level objects whenever a top-level object is randomized, and solves all rand variables and constraints across the entire object hierarchy simultaneously. A summary of his approach can be found by examining the following code.

```
class policy_base#(type ITEM=uvm_object);
    ITEM item;

    virtual function void set_item(ITEM item);
```

```
        this.item = item;
    endfunction
endclass


class policy_list#(type ITEM=uvm_object) extends policy_base#(ITEM);
    rand policy_base#(ITEM) policy[$];

    function void add(policy_base#(ITEM) pcy);
        policy.push_back(pcy);
    endfunction

    function void set_item(ITEM item);
        foreach(policy[i]) policy[i].set_item(item);
    endfunction
endclass
```

These first two classes provide the base class types for the actual policies. This allows different, related policies to be grouped together in **`policy_list`** and applied to the randomization call concurrently. Both of these classes are parameterized to a specific object type, so separate instances will be required for each class type that needs to be constrained.

```
class addr_policy_base extends policy_base#(addr_txn);
    addr_range ranges[$];

    function add(addr_t min, addr_t max);
        addr_range rng = new(min, max);
        ranges.push_back(rng);
    endfunction
endclass


class addr_permit_policy extends addr_policy_base;
    rand int selection;

    constraint c_addr_permit {
        selection inside {[0:ranges.size()-1]};

        foreach(ranges[i]) {
            if(selection == i) {
                item.addr inside {[ranges[i].min:ranges[i].max - item.size]};
            }
        }
    }
endclass


class addr_prohibit_policy extends addr_policy_base;
    constraint c_addr_prohibit {
        foreach(ranges[i]) {
            !(item.addr inside {[ranges[i].min:ranges[i].max - item.size + 1]});
        }
    }
endclass
```

The next three classes contain implementations of the **policy_base** class for the **addr_txn** class. A list of valid address ranges can be stored in the **ranges** array. The **addr_permit_policy** will choose one of the ranges at random and constrain the address to be within the range, while the **addr_prohibit_policy** will exclude the address from randomizing inside any of the ranges in its list.

```
class addr_txn;
    rand bit [31:0] addr;
    rand int        size;
    rand policy_base#(addr_txn) policy[$];

    constraint c_size {size inside {1, 2, 4};}

    function void pre_randomize;
        foreach(policy[i]) policy[i].set_item(this);
    endfunction
endclass


class addr_constrained_txn extends addr_txn;
    function new;
        addr_permit_policy    permit = new;
        addr_prohibit_policy  prohibit = new;
        policy_list#(addr_txn) pcy = new;

        permit.add('h00000000, 'h0000FFFF);
        permit.add('h10000000, 'h1FFFFFFF);
        pcy.add(permit);

        prohibit.add('h13000000, 'h130FFFFF);
        pcy.add(prohibit);

        this.policy = {pcy};
    endfunction
endclass
```

The final two classes show how these policies are used. The **addr_txn** class creates a **policy** queue, and during the **pre_randomize** function sets the handle to the **item** object within each policy to itself. The **addr_constrained_txn** subclass adds two policies to a local **pcy** list, one that permits the address to be within one of two ranges, and one that prohibits the address from being within a third range. The **addr_constrained_txn** class can then passes the local **pcy** list to the parent **policy** queue.

At this point, an instance of **addr_constrained_txn** can be created and randomized like normal, and the address will be constrained based on the policies provided.

```
addr_txn txn = new();
txn.randomize();
```

Further work on policy-based constraints has been presented since the original DVCon presentation in 2015. Kevin Vasconcellos and Jeff McNeal applied the concept to test configuration and added many nice utilities to the base policy class [3]. Chuck McClish extended the concept to manage real number values for User Defined Nettypes (UDN) and Unified Power Format (UPF) pins in an analog model [4]. Additionally, he defined a policy builder class that was used to generically build multiple types of policies while reducing repeated code that was shared between each policy class in the original implementation.

Although there has been extensive research on policies, this paper aims to address and provide solutions for three issues that have not been adequately resolved in previous implementations. Furthermore, a fourth problem that arose during testing

of the upgraded policy package implementation will also be discussed and resolved.

II. Problem #1: Parameterized Policies

The first problem with the above policy implementation is that because policies are specialized using parameterization to the class they constrain, different specializations of parameterized classes cannot be grouped and indexed. This is most apparent when creating a class hierarchy. If you extend **`addr_txn`** and add additional fields to be constrained, then you need a new policy type. The new policy type requires an extra policy list to modify any new attributes in the child class and requires the user to know which layer of the class hierarchy defines each attribute they want to constrain. Imagine a complex class hierarchy with many layers of inheritance and extension, and new constrainable attributes on each layer (one common example is a multi-layered sequence API library, such as the example presented by Jeff Vance [5]). Using policies becomes cumbersome in this case because you need to manage a separate policy list for each level of the hierarchy and know what each list can and cannot contain. You could end up with a final class that looks like this:

```
class addr_l2_txn extends addr_l1_txn;
    // ...
endclass

class addr_l3_txn extends addr_l2_txn;
    // ...
endclass

class addr_l4_txn extends addr_l3_txn;
    // ...
endclass

class addr_constrained_txn extends addr_l4_txn;
    // ... new rand fields that need to be constrained
    rand policy_base#{addr_l4_txn} policy_l4[$];

    function new;
        addr_permit_policy        permit = new;
        addr_prohibit_policy      prohibit = new;
        // ... additional policies for the subsequent layers
        policy_list#(addr_l1_txn) pcy_l1 = new;
        policy_list#(addr_l2_txn) pcy_l2 = new;
        policy_list#(addr_l3_txn) pcy_l3 = new;
        policy_list#(addr_l4_txn) pcy_l4 = new;

        // ... add all the policies to the correct list

        this.policy_l1 = {pcy_l1};
        this.policy_l2 = {pcy_l2};
        this.policy_l3 = {pcy_l3};
        this.policy_l4 = {pcy_l4};
    endfunction

    function void pre_randomize;
        super.pre_randomize();
        foreach(policy_l4[i]) policy_l4[i].set_item(this);
    endfunction
endclass
```

The solution to this problem is to introduce class inheritance into the policy classes that mirror the target class hierarchy. So for the four-layered **`addr_txn`** example, four matching policy classes would be created to match.

```
class addr_l1_policy extends policy_base#(addr_l1_txn);
    addr_range ranges[$];

    function add(addr_t min, addr_t max);
        addr_range rng = new(min, max);
        ranges.push_back(rng);
    endfunction
endclass

class addr_l2_policy extends addr_l1_policy;
    // ... additional fields that are used to constrain the object
endclass

class addr_l3_policy extends addr_l2_policy;
    // ... additional fields that are used to constrain the object
endclass

class addr_l4_policy extends addr_l3_policy;
    // ... additional fields that are used to constrain the object
endclass
```

, supplemented by convenient macros to keep the solution simple.

III. Problem #2: Definition Location

The second problem is "where do I define my policy classes?" This is not a complicated problem to solve, and most users likely place their policy classes in a file close to the class they are constraining. However, this paper will present a solution that explicitly pairs policy definitions with the class they constrain, so users never need to hunt for the definitions or guess where they are located, and so the definitions have an appropriate and consistent home in the environment. This is done by embedding a **`POLICIES`** class inside the class being constrained. This class acts as a container for all relevant policies and allows policies to be created by simply referencing the embedded class through the **`::|`** operator.

IV. Problem #3: Boilerplate Overload

The third problem is that policies are relatively expensive to define since you need at a minimum, a class definition, a constructor, and a constraint. This will be relatively unavoidable for complex policies, such as those defining a relationship between multiple specific class attributes. For generic policies, such as equality constraints, range constraints, or set membership (keyword **`inside`**) constraints, a strategy using macros is presented that drastically reduces the expense and risk of defining common policies. The macros are responsible for creating the specialized policy class for the required constraint within the target class, as well as a static constructor function that is used to create new policy instances of the class. Additional macros are utilized for setting up the embedded **`POLICIES`** class within the target class. These macros can be used hand in hand with the non-macro policy classes needed for the complex constraints, if necessary.

V. Problem #4: Unexpected Policy Reuse Behavior

A fourth problem we noticed in our usage of policy classes was occasional unexpected behavior when attempting to reuse a policy. It appeared as though the policies "remembered" being randomized and wouldn't reapply their constraints during subsequent **`randomize`** calls. An exact diagnosis of the issue was not discovered because we transitioned our policy approach to a "use once and discard" approach, where we would treat a policy as disposable and model them as extremely lightweight classes. We implemented practices such as static functions to create new instances of the policies and policy methods to return fresh copies of themselves.

VI. Conclusion

Functional example code will be provided to demonstrate the solutions to these problems and provide other minor improvements to the original policy class implementation.

APPENDIX A
POLICY PACKAGE

A. *policy_pkg.sv*

```
`ifndef __POLICY_PKG__
`define __POLICY_PKG__

`include "uvm_macros.svh"

package policy_pkg;

    import uvm_pkg::*;

    `include "policy.svh"

    typedef policy POLICY_QUEUE[$];

    `include "policy_container.svh"

    `include "policy_imp.svh"
    `include "policy_list.svh"

endpackage: policy_pkg

`endif
```

B. *policy.svh*

```
`ifndef __POLICY__
`define __POLICY__

interface class policy;

    pure virtual function string name();

    pure virtual function string type_name();

    pure virtual function string description();

    pure virtual function bit item_is_compatible(uvm_object item);

    pure virtual function void set_item(uvm_object item);

    pure virtual function policy copy();

endclass: policy

`endif
```

C. *policy_container.svh*

```
`ifndef __POLICY_CONTAINER__
`define __POLICY_CONTAINER__

interface class policy_container;

    // Queries
    pure virtual function bit has_policies();

    // Assignments
    pure virtual function void set_policies(policy_queue policies);

    pure virtual function void add_policies(policy_queue policies);

    pure virtual function void clear_policies();

    // Access
    pure virtual function policy_queue get_policies();

    // Copy
    pure virtual function policy_queue copy_policies();


endclass: policy_container

`endif
```

D. policy_imp.svh

```
`ifndef __POLICY_IMP__
`define __POLICY_IMP__

class policy_imp #(type ITEM=uvm_object) implements policy;

    ITEM item;

    virtual function string name();
        return ($typename(this));
    endfunction: name

    virtual function string type_name();
    endfunction: type_name

    virtual function string description();
    endfunction: description

    virtual function bit item_is_compatible(uvm_object item);
    endfunction: item_is_compatible

    virtual function void set_item(uvm_object item);

        if (item == null)
            `uvm_fatal("policy::set_item()", "NULL item passed")
```

```
        else if (!$cast(this.item, item))
            `uvm_warning("policy::set_item()", $sformatf("Item type <%s> is not compatible with policy type <%s>", item.get_type_name(), $typename(this.item)))

    endfunction: set_item

    virtual function policy copy();
    endfunction: copy

endclass: policy_imp

`endif
```

E. policy_list.svh

```
`ifndef __POLICE_LIST__
`define __POLICE_LIST__

class policy_list implements policy;

    rand POLICY_QUEUE list;

    function new(POLICY_QUEUE policy_queue={});
        list = policy_queue;
    endfunction: new

    virtual function string name();
        int unsigned list_size;
        int unsigned last_idx;
        string       s;

        list_size = list.size();
        last_idx  = ((list_size) ? (list_size - 1) : 0);

        foreach(list[idx]) s = {s, list[idx].name(), ((idx == last_idx) ? "" : ", ")};

        return ({"{", s, "}"});
    endfunction: name

    virtual function void set_item(uvm_object item);
        foreach(list[idx]) list[idx].set_item(item);
    endfunction: set_item

    virtual function void add(policy pcy);
        list.push_back(pcy);
    endfunction: add

    virtual function void clear();
        list.delete();
    endfunction: clear

    virtual function int unsigned size();
```

```
        return (list.size());
    endfunction: size

endclass: policy_list

`endif
```

APPENDIX B
POLICY MACROS

A. *policy_macros.svh*

```
`ifndef __POLICY_MACROS__
`define __POLICY_MACROS__

//================================================
// Embedded POLICIES class macros
//================================================

`define start_policies(cls)                     \
class POLICIES;                                 \
`LOCAL_POLICY_IMP(cls)

`define start_extended_policies(cls, parent)    \
class POLICIES extends parent::POLICIES;        \
`LOCAL_POLICY_IMP(cls)

`define end_policies                            \
endclass: POLICIES

`define LOCAL_POLICY_IMP(cls)                   \
    typedef policy_imp#(cls) LOCAL_POLICY_T;


//================================================
// Policy template macros
//================================================

`include "constant_policy.svh"
`include "fixed_policy.svh"
`include "ranged_policy.svh"
`include "set_policy.svh"

`endif
```

B. *constant_policy.svh*

```
`ifndef __CONSTANT_POLICY__
`define __CONSTANT_POLICY__

// Full policy definition
`define constant_policy(POLICY, TYPE, field, constant)                          \
`CONST_POLICY_CLASS(POLICY, TYPE, field, constant)                              \
```

```
`CONST_POLICY_CONSTRUCTOR(POLICY)

// Policy class definition
`define CONST_POLICY_CLASS(POLICY, TYPE, field, constant)                    \
    class POLICY``_policy extends LOCAL_POLICY_T                             \
        typedef TYPE l_field_t;                                              \
                                                                             \
        constraint c_const_value {                                           \
            (item != null) -> (item.field == l_field_t'(constant));          \
        }                                                                    \
                                                                             \
        function new();                                                      \
        endfunction: new                                                     \
                                                                             \
        virtual function string name();                                      \
            return (`"POLICY(field==constant)`");                            \
        endfunction: name                                                    \
    endclass: POLICY``_policy

// Policy constructor definition
`define CONST_POLICY_CONSTRUCTOR(POLICY)                                     \
    static function policy POLICY();                                        \
        POLICY``_policy new_pcy = new();                                     \
        return (new_pcy);                                                    \
    endfunction: POLICY

`endif
```

C. fixed_policy.svh

```
`ifndef __FIXED_POLICY__
`define __FIXED_POLICY__

// Full policy definition
`define fixed_policy(POLICY, TYPE, field, RADIX="%0p")                       \
`FIXED_POLICY_CLASS(POLICY, TYPE, field, RADIX)                              \
`FIXED_POLICY_CONSTRUCTOR(POLICY, TYPE, RADIX)

// Policy class definition
`define FIXED_POLICY_CLASS(POLICY, TYPE, field, RADIX="%0p")                 \
    class POLICY``_policy extends LOCAL_POLICY_T                             \
        typedef TYPE        l_field_t;                                       \
                                                                             \
        protected TYPE      m_fixed_value;                                   \
        protected string    m_radix=RADIX;                                   \
                                                                             \
        constraint c_fixed_value {                                           \
            (item != null) -> (item.field == l_field_t'(m_fixed_value));     \
        }                                                                    \
                                                                             \
        function new(TYPE value, string radix=RADIX);                        \
            this.set_value(value);                                           \
            this.set_radix(radix);                                           \
        endfunction: new                                                     \
```

```
                                                                                \
        virtual function string name();                                         \
            return ({                                                           \
                `"POLICY(field==",                                              \
                $sformatf(m_radix, m_fixed_value),                              \
                `")`"                                                           \
            });                                                                 \
        endfunction: name                                                       \
                                                                                \
        virtual function void set_value(TYPE value);                           \
            this.m_fixed_value = value;                                         \
        endfunction: set_value                                                  \
                                                                                \
        virtual function TYPE get_value();                                      \
            return (this.m_fixed_value);                                        \
        endfunction: get_value                                                  \

        virtual function void set_radix(string radix);                          \
            this.m_radix = radix;                                               \
        endfunction: set_radix                                                  \
                                                                                \
        virtual function string get_radix();                                    \
            return (this.m_radix);                                              \
        endfunction: get_radix                                                  \
    endclass: POLICY``_policy

// Policy constructor definition
`define FIXED_POLICY_CONSTRUCTOR(POLICY, TYPE, RADIX="%0p")                     \
    static function policy POLICY(TYPE value, string radix=RADIX);             \
        POLICY``_policy new_pcy = new(value, radix);                            \
        return (new_pcy);                                                       \
    endfunction: POLICY

`endif
```

D. ranged_policy.svh

```
`ifndef __RANGED_POLICY__
`define __RANGED_POLICY__

// Full policy definition
`define ranged_policy(POLICY, TYPE, field, RADIX="%0p")                         \
`RANGED_POLICY_CLASS(POLICY, TYPE, field, RADIX)                                \
`RANGED_POLICY_CONSTRUCTOR(POLICY, TYPE, RADIX)

// Policy class definition
`define RANGED_POLICY_CLASS(POLICY, TYPE, field, RADIX="%0p")                   \
    class POLICY``_policy extends LOCAL_POLICY_T                                \
        typedef TYPE        l_field_t;                                          \
                                                                                \
        protected TYPE      m_low;                                              \
        protected TYPE      m_high;                                             \
        protected bit       m_inside;                                           \
        protected string    m_radix=RADIX;                                      \
```

```
                                                                                \
        constrant c_ranged_value {                                              \
            (item != null) ->                                                   \
                ((!m_inside) ^ (item >= m_low && item <= m_high));               \
        }                                                                       \
                                                                                \
        function new(                                                           \
            TYPE   low,                                                         \
            TYPE   high=low,                                                    \
            bit    inside=1'b1,                                                 \
            string radix=RADIX                                                  \
        );                                                                      \
            this.set_range(low, high);                                          \
            this.set_inside(inside);                                            \
            this.set_radix(radix);                                              \
        endfunction: new                                                        \
                                                                                \
        virtual function string name();                                         \
            return ({                                                           \
                `"POLICY(field ",                                               \
                m_inside ? "inside [" : "outside [",                            \
                $sformatf(m_radix, m_low),                                      \
                ", ",                                                           \
                $sformatf(m_radix, m_high),                                     \
                `"])`"                                                          \
            });                                                                 \
        endfunction: name                                                       \
                                                                                \
        virtual function void set_range(TYPE low, TYPE high);                   \
            if (low <= high) begin                                              \
                this.m_low = low;                                               \
                this.m_high = high;                                             \
            end else begin                                                      \
                this.m_low = high;                                              \
                this.m_high = low;                                              \
            end                                                                 \
        endfunction: set_range                                                  \
                                                                                \
        virtual function TYPE get_low();                                        \
            return (this.m_low);                                                \
        endfunction: get_low                                                    \
                                                                                \
        virtual function TYPE get_high();                                       \
            return (this.m_high);                                               \
        endfunction: get_high                                                   \
                                                                                \
        virtual function void set_inside(bit inside);                           \
            this.m_inside = inside;                                             \
        endfunction: set_inside                                                 \
                                                                                \
        virtual function bit get_inside();                                      \
            return (this.m_inside);                                             \
        endfunction: get_inside                                                 \
                                                                                \
```

```
        virtual function void set_radix(string radix);                          \
            this.m_radix = radix;                                               \
        endfunction: set_radix                                                  \
                                                                                \
        virtual function string get_radix();                                    \
            return (this.m_radix);                                              \
        endfunction: get_radix                                                  \
    endclass: POLICY``_policy

// Policy constructor definition
`define RANGED_POLICY_CONSTRUCTOR(POLICY, TYPE, RADIX="%0p")                    \
    static function policy POLICY(                                              \
        TYPE   low,                                                             \
        TYPE   high=low,                                                        \
        bit    inside=1'b1,                                                     \
        string radix=RADIX                                                      \
    );                                                                          \
        POLICY``_policy new_pcy = new(low, high, inside, radix);                \
        return (new_pcy);                                                       \
    endfunction: POLICY

`endif
```

E. set_policy.svh

```
`ifndef __SET_POLICY__
`define __SET_POLICY__

// Full policy definition
`define set_policy(POLICY, TYPE, field, RADIX="%0p")                            \
`SET_POLICY_CLASS(POLICY, TYPE, field, RADIX)                                   \
`SET_POLICY_CONSTRUCTOR(POLICY, TYPE, RADIX)

// Policy class definition
`define SET_POLICY_CLASS(POLICY, TYPE, field, RADIX="%0p")                      \
    class POLICY``_policy extends LOCAL_POLICY_T                                \
        typedef TYPE        l_field_array_t[];                                  \
                                                                                \
        protected l_field_array_t   m_values;                                   \
        protected bit               m_inside;                                   \
        protected string            m_radix=RADIX;                              \
                                                                                \
        constrant c_set_value {                                                 \
            (item != null) ->                                                   \
                ((m_inside) ^ (item.field inside {m_values}));                  \
        }                                                                       \
                                                                                \
        function new(                                                           \
            l_field_array_t values,                                             \
            bit             inside=1'b1,                                        \
            string          radix=RADIX                                         \
        );                                                                      \
            this.set_values(values);                                            \
            this.set_inside(inside);                                            \
```

```
            this.set_radix(radix);                                          \
        endfunction: new                                                    \
                                                                            \
        virtual function string name();                                     \
            string values_str = "";                                         \
                                                                            \
            foreach(m_values[i])                                            \
                values_str = {                                              \
                    values_str,                                             \
                    $sformatf(m_radix, m_values[i]),                        \
                    i == m_values.size()-1 ? "" : ", "};                    \
                                                                            \
            return ({                                                       \
                `"POLICY(field ",                                           \
                m_inside ? "inside {" : "outside {",                        \
                values_str,                                                 \
                `"})`"                                                      \
            });                                                             \
        endfunction: name                                                   \
                                                                            \
        virtual function void set_values(l_field_array_t values);           \
            this.m_values = values;                                         \
        endfunction: set_values                                             \
                                                                            \
        virtual function l_field_array_t get_values();                      \
            l_field_array_t l_array;                                        \
            foreach (this.m_values[i])                                      \
                l_array[i] = this.m_values[i];                              \
            return (l_array);                                               \
        endfunction: get_values                                             \
                                                                            \
        virtual function void set_inside(bit inside);                       \
            this.m_inside = inside;                                         \
        endfunction: set_inside                                             \
                                                                            \
        virtual function bit get_inside();                                  \
            return (this.m_inside);                                         \
        endfunction: get_inside                                             \
                                                                            \
        virtual function void set_radix(string radix);                      \
            this.m_radix = radix;                                           \
        endfunction: set_radix                                              \
                                                                            \
        virtual function string get_radix();                                \
            return (this.m_radix);                                          \
        endfunction: get_radix                                              \
    endclass: POLICY``_policy

// Policy constructor definition
`define SET_POLICY_CONSTRUCTOR(POLICY, TYPE, RADIX="%0p")                   \
    typedef TYPE POLICY``_array_t[];                                        \
    static function policy POLICY(                                         \
        POLICY``_array_t values,                                            \
        bit              inside=1'b1,                                       \
```

```
        string          radix=RADIX                                          \
    );                                                                       \
        POLICY``_policy new_pcy = new(values, inside, radix);                \
        return (new_pcy);                                                    \
    endfunction: POLICY

`endif
```

REFERENCES

[1] J. Dickol, "SystemVerilog Constraint Layering via Reusable Randomization Policy Classes," DVCon, 2015.
[2] J. Dickol, "Complex Constraints: Unleashing the Power of the VCS Constraint Solver," SNUG Austin, September 29, 2016.
[3] K. Vasconcellos, J. McNeal, "Configuration Conundrum: Managing Test Configuration with a Bite-Sized Solution," DVCon, 2021.
[4] C. McClish, "Bi-Directional UVM Agents and Complex Stimulus Generation for UDN and UPF Pins," DVCon, 2021.
[5] J. Vance, J. Montesano, M. Litterick, J. Sprott, "Be a Sequence Pro to Avoid Bad Con Sequences," DVCon, 2019.