Use Stimulus Domain for Systematic Exploration of Time Dimension and Automatic Testcase Construction

Ning Chen, Infineon Technologies AG, Munich, Germany (*ning.chen@infineon.com*)

Martin Ruhwandl, Infineon Technologies AG, Munich, Germany (*martin.ruhwandl@infineon.com*)

***Abstract*—The state-of-the-art metric-driven constraint-random verification methodology requires achieving full functional coverage as one of its most important metrics. However, defining functional coverage itself is a subjective and dependent process largely based on human intellects. Consequently, given a verification environment and its attached testcases, it is challenging to even answer the simple question objectively whether the stimulated space is random enough. In this paper, we propose a new concept of Stimulus Domain and use it to explore the stimulation space along the time dimension systematically. As a result, the verification methodology is extended and enhanced by a stimuli-driven aspect which in this sense even overweighs the metric-driven aspect, since the former unfolds the primary source of power of randomization and leads to better functional coverage definition. Furthermore, the proposed approach can be used for automatic testcase construction while facilitating the reuse of Stimulus Domains across a large set of IPs shared with common functionalities and interfaces. The proposed approach has been applied to two industrial IPs and its effectiveness has been proved with around 20 issues identified for legacy feature sets in one case and zero bugs ensured in another case.**

***Keywords*—stimulus domain; constraint-random verification; metric-driven; stimulation; functional coverage**

I. Introduction

A digital circuit is a processing machine traversing the state space along a deterministic trajectory of state transitions in discrete steps. During this process, the inputs are transformed into outputs with defined behavior. The ideal target of stimulation is not to traverse the whole state space, but to stimulate the full set of all transition trajectories along the time dimension.

The fundamental underlying principle of simulation-based verification is to capture bugs through redundancy. The defined behavior according to the specification is to be implemented by both the verification engineer and the designer. These two implementations are compared in order to capture eventual mismatches. In many severe cases, where bugs are missed in the pre-silicon phase thus leading to huge cost of redesign, the root cause is not attributed to missing or bad checkers, but rather the fact that the case has never been stimulated. The Universal Verification Methodology (UVM) has laid much value on the reusability and scalability of the verification environment [1]. It provides the basic constructs such as sequence and sequence driver about what can be used to build testcases [2]. It uses functional coverage defined in the verification environment as one of the most important metrics to represent and guide the set of test scenarios to be stimulated and full functional coverage is required for sign-off.

However, there is no systematic framework about how to manipulate the time dimension of stimulation to achieve the maximum extent of randomization in order to reach or come nearer to the ideal target. On the other hand, defining functional coverage is a subjective and dependent process which highly relies on human intellects, since it represents the set of test scenarios that the verification engineer has in mind.

In this paper, we propose to use Stimulus Domain as the core concept to explore the transition trajectories along the time dimension in a systematic way. As a result, the metric-driven verification methodology is extended by a stimuli-driven aspect. The unfolded power of randomization allows covering more corner scenarios deeply hidden and thus leads to a better coverage definition due to its objective nature. Furthermore, the proposed approach allows for automatic testcase construction and reuse of Stimulus Domains across IPs with common functionalities shared [3, 4]. The application results have proved the effectiveness of the proposed approach.

This paper is organized as follows. In Section II, we explain the core concept of Stimulus Domain and its construction. In Section III we present the application areas of the proposed approach. In Section IV, we show the results on verification of two industrial IPs. Section V concludes the paper.

II. Stimulus domain

The basic idea behind Stimulus Domain is to apply a divide-and-conquer approach on the large set of control variables under stimulation. In such a way, the internal structure of the control variables is manifested and its dynamics can be then analyzed thoroughly and completely.

We start with a general definition of Stimulus Domain to reveal its theoretical soundness with the aim to cover all different use categories. Then we dive into some special categories for better and intuitive understanding of the idea.

A. General Definition

We denote the full set of control variables of the design under test as $\boldsymbol{C}$ which includes all inputs and all individual writable register fields. Each control variable is defined on its finest granularity level based on the functionality it controls. Assume that the number of control variables is n.

$$|\boldsymbol{C}| = n \tag{1}$$

The overall functionality of the design under test is controlled by the variables in $\boldsymbol{C}$. The set $\boldsymbol{C}$ must be complete in the sense that any control variable that can be stimulated must be an element in $\boldsymbol{C}$.

Naturally each control variable is responsible for only part of the overall functionality. We can divide and organize the control variables into different groups according to the functionality under control. Based on these groups we formulate the concept of Stimulus Domain.

A Stimulus Domain, denoted by $\boldsymbol{D}_i$ with i as index, consists of following elements:

- $\boldsymbol{C}_i$: a set of control variables, i.e., a subset of $\boldsymbol{C}$, bundled densely by the functionality under its control from the stimulation perspective.

$$\boldsymbol{C}_i \subset \boldsymbol{C} \tag{2}$$

- $\boldsymbol{S}_i$: a set of states defined on $\boldsymbol{C}_i$. Assume the number of states is n_i.

$$|\boldsymbol{S}_i| = n_i \tag{3}$$

- $\boldsymbol{P}_i(t)$: a conditional and time-variant state transition probability matrix based on $\boldsymbol{S}_i$, with each transition attached with a random integer number of clock cycles, denoted by $r_i(t)$**,** to represent the delay of state transition. The conditional and time-variant nature stems from the fact that the state transition may be affected by events during the runtime such as those of the readable register fields and outputs of the design. Note that the time-invariance can be viewed as one special case of the time-variance property.

$$\boldsymbol{P}_i(t) = \begin{bmatrix} p_{i,11}(t) & \cdots & p_{i,1n_i}(t) \\ \vdots & \ddots & \vdots \\ p_{i,n_i1}(t) & \cdots & p_{i,n_in_i}(t) \end{bmatrix} \tag{4}$$

  Each matrix element represents the transition probability from one state to another as in a random process and the probability can change along the time dimension due to its time-variant nature. A generalized state transition diagram from state s_j to state s_k is shown in Fig. 1 where the reference to the index of Stimulus Domain is omitted.

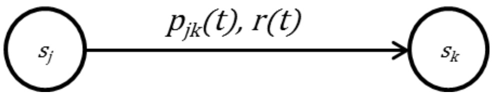

Figure 1. State Transition Diagram

As can be seen in Fig. 1, upon a certain choice of next state for stimulation based on the probability distribution, each transition is also attached with a random delay to represent the time shift between two states introduced in the stimulation. The combination of transition probability and random delay manifests the prime power of randomization.

As a summary, a Stimulus Domain can be written as a bundle of all the elements formulated above.

$$\boldsymbol{D}_i \equiv \{\boldsymbol{C}_i, \boldsymbol{S}_i, \boldsymbol{P}_i(t), r_i(t)\} \tag{5}$$

The Stimuli as a whole, denoted by $\boldsymbol{D}$, can be formally defined as the full set of all defined Stimulus Domains. Assume that the number of Stimulus Domains defined is m.

$$\boldsymbol{D} \equiv \{\boldsymbol{D}_i\}, i = 1..m \tag{6}$$

The Stimuli is constructed with following three principles enforced.

- Completeness: It should cover the full set of control variables. No control variable should be neglected unless being excluded explicitly and upon agreement.

$$\boldsymbol{C} = \bigcup_{i=1}^{m} \boldsymbol{C}_i \tag{7}$$

- Singularity: Each control variable should belong to only one Stimulus Domain.

$$\boldsymbol{C}_i \bigcap \boldsymbol{C}_j = \emptyset, \forall i, j = 1..\mathrm{m}, i \neq j \tag{8}$$

- Orthogonality: This is to ensure the independency of control variables with respect to stimulation between Stimulus Domains. The simple criteria to judge the orthogonality are to ask the question whether the subset of control variables under focus can be stimulated independently on others. If a strong dependency exists, it is to consider putting all dependent ones into one Stimulus Domain.

$$\boldsymbol{D}_i \perp \boldsymbol{D}_j, \forall i, j = 1..\mathrm{m}, i \neq j \tag{9}$$

The set of Stimulus Domains can be organized in a tree structure on top of it with respect to the functional partitioning, with the defined Stimulus Domains as leaf nodes fulfilling the three principles above.

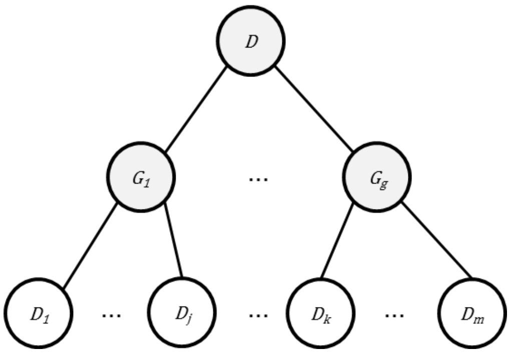

Figure 2. Tree Structure of Stimuli

As shown in Fig. 2, the Stimulus Domains have been organized into g groups denoted by G_1 to G_g to represent the whole Stimuli D in a two-level tree structure, with each group being assigned to a certain part of the functionality under the control of all its consisting Stimulus Domains. Naturally in a more general sense the depth of the tree can be further extended as necessary.

An interaction matrix, denoted by $\boldsymbol{I(t)}$ as a time-variant global data structure, is defined on $\boldsymbol{D}$ with each entry $I_{ij}(t)$ representing the encoded message initiated by the domain i for the domain j at time t. Note that this interaction matrix doesn't have to be part of the definition of the Stimuli, but rather serves as a combination of static information and runtime data structure to provide a communication mechanism among the Stimulus Domains.

$$\boldsymbol{I}(t) = \begin{bmatrix} I_{11}(t) & \cdots & I_{1m}(t) \\ \vdots & \ddots & \vdots \\ I_{m1}(t) & \cdots & I_{mm}(t) \end{bmatrix} \tag{10}$$

In the simple Boolean case where $I_{ij}(t)$ only takes the value 0 or 1, each matrix element can be the on/off state enforced by domain i on domain j. With each Stimulus Domain being able to be on and off, the state transition diagram of each Stimulus Domain can be simplified to a diagram of three super-states as shown in Fig. 3 which allows more insight into the structure of the general state transition defined in Fig. 1.

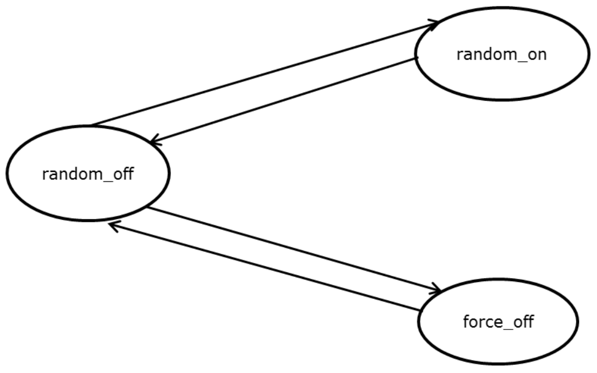

Figure 3. Simplified State Transition Diagram under Boolean Interaction

The three super-states in Fig. 3 are explained as follows.

- force_off: No stimulation initiated by the current Stimulus Domain, i.e., for a certain period of time, the Stimulus domain is switched off;
- random_off: Randomization is allowed, but currently in the waiting state until the random delay expires so that e.g. it can be switched on again;
- random_on: Randomization is being initiated under this state which acts as a super-state covering a detailed state transition diagram defined by the state transition as in Fig. 1.

Note that the Boolean case represents only a simple special case but a commonly used one for the interaction matrix which allows simplification and insight into the structure of the generalized state transition diagram with transition probability and random delay as shown in Fig. 3.

B. Special Categories

The general definition above covers a broad spectrum of Stimulus Domains which includes among others the following two special categories.

- Autonomous: The state transition of an autonomous Stimulus Domain is not affected by any event along the time dimension. Furthermore, there is no interaction from other Stimulus Domains as formulated in the interaction matrix. For such a Stimulus Domain, the stimuli can be constructed in a very simple way based on its own defined dynamics without taking other Stimulus Domains into account.
- Reactive: A reactive Stimulus Domain becomes active only when a certain event happens. During its inactive phase, no stimuli are constructed. A good example would be a sequence acting as interrupt service routine to clear the interrupt status. It will be only driven when the interrupt status bit has been set.

A common Stimulus Domain will be lying between an Autonomous one and a reactive one. The choice of its next state to stimulate is constrained by some event while certain interaction patterns between individual Stimulus Domains are applied as defined by the interaction matrix.

III. APPLICATIONS

We have identified the following main application areas for the concept of Stimulus Domain.

A. *Mindset for Stimulation*

The concept of Stimulus Domain as proposed in Section II describes a rigorous framework about how to construct stimuli in a more objective way which exhausts the power of randomization and allows exposing corner cases normally difficult to touch. It builds the basis of a mindset for a verification engineer to divide and conquer the whole stimulation space in order to come nearer to the ideal target. Naturally, the definition of Stimulus Domains is still relying on the human intellects of the verification engineer.

B. *Automatic Testbench Construction*

Due to the formal definition of Stimulus Domain and its constituent elements, we can use it for automatic testbench construction. A tool/script can take all the necessary input information of the Stimuli formulated in Eq. (6) and traverse the state transition probability matrix of each Stimulus Domain to construct any random testcases either online or offline. If certain format of the input information is defined, one can think of using some generic core to construct the stimuli assembled from all the constituent Stimulus Domains independently on the design under test.

- The most generic testcase can be constructed by stimulating all Stimulus Domains in parallel based on their own dynamics defined in Section II. Each Stimulus Domain acts similar as a random process with its behavior defined by the state transition probability matrix. The difference is that a random delay is attached to each state transition. For each Stimulus Domain the verification engineer can think of e.g. writing a generic sequence with the state as one enumeration type to decide what kind of detailed stimuli is to be constructed, or using one specific sequence for each individual state attached with a few random variables and assembling all in the end.

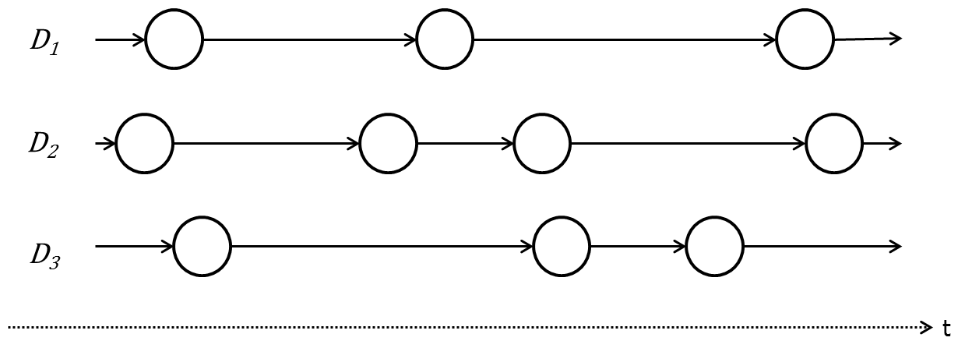

Figure 4. General Stimuli

The general stimuli are shown in Fig. 4, where as an example a snapshot of three Stimulus Domains along the time dimension is drawn. Each circle represents one state for the respective Stimulus Domain and the distance between two states is used to show the delay of two consecutive states. All Stimulus Domains are run in parallel with the interaction already encoded in the states selected, i.e., upon certain event the time-variant state transition probability matrix will exclude certain states and allow only a constrained subset of states to be selected as next ones. Note that the stimulation of each state may also consume time, e.g., by using a series of register writes to configure a specific functionality.

- By manipulating the interaction matrix ***I(t)***, certain Stimulus Domains can be switched on/off. Testcases corresponding to specific test scenarios can be constructed. It is actually the normal case that the verification engineer starts with basic Stimulus Domains related with the elementary feature set of the design under test. As more Stimulus Domains are added, it is time to start to think of the interactions between them and put them successively into parallel stimulation for maximum randomization.

C. *Reuse across IPs*

For IPs with common functionalities shared, such as common configuration and common debug features, the defined Stimulus Domains can be reused across IP boundaries. That means part of the mature stimuli that have already been developed can be simply reused so that further development cost can be saved. As shown in Fig. 5, two IPs share the same group, denoted by G_c, of Stimulus Domains.

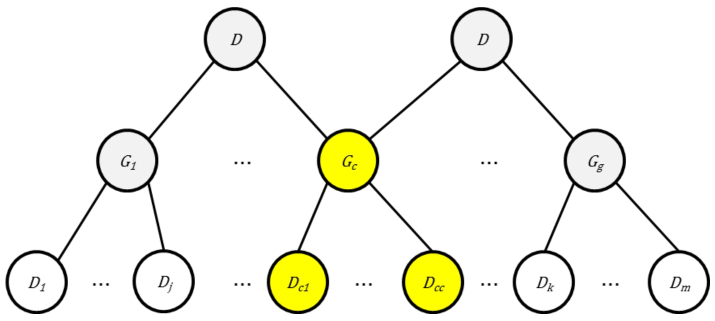

Figure 5. Reuse of Stimulus Domain

Other application areas can be conceived such as in functional safety where faults are injected and the interplay between failures modes and safety mechanism is to be verified. This is basically another Stimulus Domain with the set of control variables extended to internal signals in the design.

IV. RESULTS

The proposed approach has been applied to two industrial IPs. For the first IP it is related with the legacy feature set where the concept of Stimulus Domain has been used to improve the testbench quality and thus identify potential issues. For the second IP it is related with the new feature set newly defined which has required building the testbench from scratch. The concept of Stimulus Domain has helped to ensure the zero-bug quality as already proven by silicon.

A. *First IP*

The design under test is an IP for the Local Interconnect Network (LIN), a serial communication protocol widely used to connect components in vehicles [5]. The IP can work in either the master mode or the slave mode. For the purpose of this paper, we limit the description to the slave mode.

As discussed in Fig. 2, the Stimulus Domains can be organized into a tree structure for further clustering. For this IP we have used a tree of depth 2, i.e., the Stimulus Domains are organized in groups which are located at the same level directly under the root node. We have given a name to each Stimulus Domain and each group as follows.

- Functional
    - Configuration: configure the mode and rx/tx related register fields
    - RX: receive line for LIN header and LIN response
    - TX: transmit line for LIN response
- Interrupt
    - ISR: interrupt service routine for clearing interrupt status bits

- Inactive
    - Sleep: put the design under test into sleep
    - Disable: disable the design under test
- Reset
    - Kernel Reset: do reset on the kernel
    - FPI Reset: do reset on the bus
    - Debug Reset: do reset related with debug
- Debug
    - Suspend: trigger different suspend modes

We take the Stimulus Domain RX as an example to discuss the definition of states and probability transition matrix. The responsibility of this domain is to stimulate the receive line according to the protocol. One state can also be viewed as one basic sequence that can be stimulated with a few un-timed random variables attached. The full set of states can be also built into tree structures. For the RX domain, two main super-states Wake-Up and Frame are defined.

- Wake-Up: with random wake-up pulse length as attached random variables.
- Frame
    - Full Frame: with random break lengths, random PIDs, random parity bit errors and random stop bits.
    - Broken Frame: with random variable to signify the position where the frame is broken.

The state transition diagram is constructed on different levels. On the top level, the system switches among the super-states. Behind each super-state, a detailed transition of its consisting states is constructed as above, e.g., between the different kinds of broken frames.

After the states have been defined, it is time to think about the state transition probability matrix and the distribution of random time interval between two consecutive states. In the simplest case, equal transition probability can be allocated to all the individual states defined. However, since we want to test more on the super-state Frame, we have allocated a much higher probability of super-state Frame (90%) against of super-state Wake-Up (10%). It has followed an equal transition probability between the Broken Frame and the Full Frame, since on one hand we want to see how the slave behaves under a broken frame while on the other hand we want to test the correct functionality of the design. For the random time interval, an equal distribution is assumed with the range up to a certain limit.

Next, we take a look at the temporal relationship between the two Stimulus Domains TX and Suspend. We show one snapshot of the interleaving of the frame and the suspend request in Fig. 6 with the variables having following meanings.

- $t_{start\_of\_frame}$, $t_{end\_of\_frame}$: time point of frame start and frame end
- τ_{frame}: duration of frame, covered by the state defined in TX domain
- $t_{start\_of\_suspend}$, $t_{end\_of\_suspend}$: time point of suspend request entry and exit
- $\tau_{suspend}$: duration of suspend request, covered by the state defined in Suspend domain

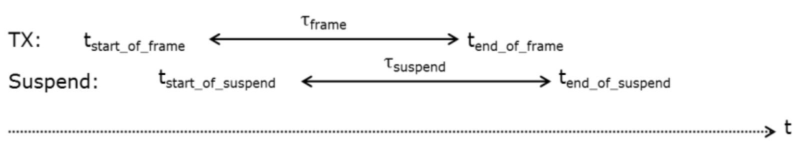

Figure 6. Temporal Relationship between TX and Suspend

Further, multiple frame and multiple suspend requests shall be stimulated in random ways in order to generate all constellations between frame and suspend requests. Consequently, two other variables come into place as shown in Fig. 7.

- τ_{frame_wait}: wait time between two consecutive frames, covered by the random delay in TX domain
- $\tau_{suspend\_wait}$: wait time between two suspend requests, covered by the random delay in Suspend domain

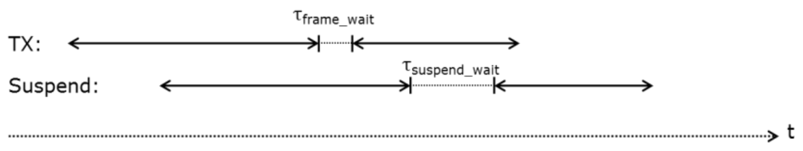

Figure 7. Random Time Interval

For the verification of this first IP, as a summary, we have focused on the definition of states and its probability transition probability. Totally we have defined 10 Stimulus Domains to reach the maximum extent of randomization. The result has been very positive and around 20 issues have been identified. Most of the issues are related with corner cases that have never been stimulated before.

Due to better steering into corner cases based on the proposed method, it creates more intellectual challenges on the verification environment which needs to handle these corner cases correspondingly. However, the objective nature of the corner cases has forced the verification engineer to think of their validity and, if valid, extending the verification environment. In the end, the functional coverage still has been used for the final sign-off.

B. Second IP

The feature set to be verified under the second IP is related with the bus control unit in the system which offers among others the arbitration, debug and alarm functionalities to the whole bus system. The focus of the verification based on the concept of Stimulus Domain is to identify individual control variables in a first step which fulfills the property of orthogonality as in Eq. (9). Each writable register field has been analyzed carefully to answer the question whether it can be stimulated rather independently on the others. The clustering of individual control variables into Stimulus Domains has happened only when it is necessary, although in the general sense the clustering creates an order from an unordered mess of control variables. If no clustering is meaningful, then one control variable together with its dynamics has been viewed as one standalone Stimulus Domain. In total we have identified 14 Stimulus Domains. The parallel running of all domains pushes the stimulation into its edge by giving access to the deep stimulation space. As a result, zero bugs have been ensured based on the validation of silicon already available today.

V. Conclusions

In this paper, we have developed a systematic framework of using Stimulus Domain to guide testcase construction. We have discussed its elements, properties and application areas. Its effectiveness has been proved with two industrial IPs.

References

[1] Accellera Systems Initiative, "Universal Verification Methodology (UVM) Working Group," http://www.accellera.org/activities/working-groups/uvm.

[2] K. Schwartz, T. Corcoran, "Error Injection: When Good Input Goes Bad," DVCON U.S., 2017

[3] A. Hamid, D. Koogler, and T. Anderson, "Using Portable Stimulus to Verify Cache Cohenrency in a Many-Core SoC," DVCON U.S., 2016

[4] Accellera Systems Initiative, "Portable Stimulus Specification Working Group," http://www.accellera.org/activities/working-groups/portable-stimulus.

[5] LIN protocol, https://www.cs-group.de/wp-content/uploads/2016/11/LIN_Specification_Package_2.2A.pdf