2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Graphical Topology Info Structure for Constrained Random Verification in SoC/Subsystem Tests

Evean Qin, Richard Bell, David Chen

Advanced Micro Devices, Inc.

accellera SYSTEMS INITIATIVE

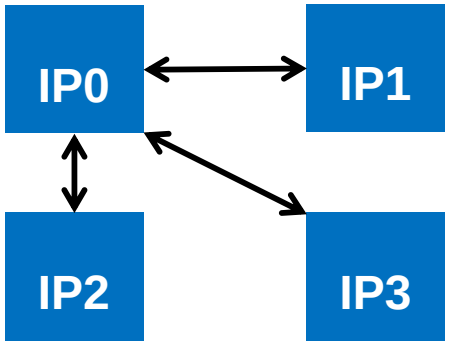
Introduction

Random or not random, that is the question

- At SoC or subsystem level, constrained random approach has been widely adopted in verification.
- Without sufficient system information in the test environment, some test scenarios may not be constrained properly for randomization. So, we are back to the traditional directed tests with manual setup.
- Examples:
  - Power gating (PG) or clock gating (CG) tests where needed to stop traffic going though selected IPs precisely, while other traffic is still running normally.

PG/CG Tests for a Simple System

- In a simple system, manual setup for the directed tests hardly seems to be a problem. For example, to cover all the PG/CG IP combinations for a system with 4 IPs below, 7 scenarios need to be covered.



4 IPs

Scenario	Combinations
Single IP	IP0, IP1, IP2, IP3
2-IP Combo	IP0+IP1, IP0+IP2, IP0+IP3, IP1+IP2, IP1+IP3, IP2+IP3
3-IP Combo	IP0+IP1+IP2, IP0+IP1+IP3, IP0+IP2+IP3, IP1+IP2+IP3
4-IP Combo	IP0+IP1+IP2+IP3

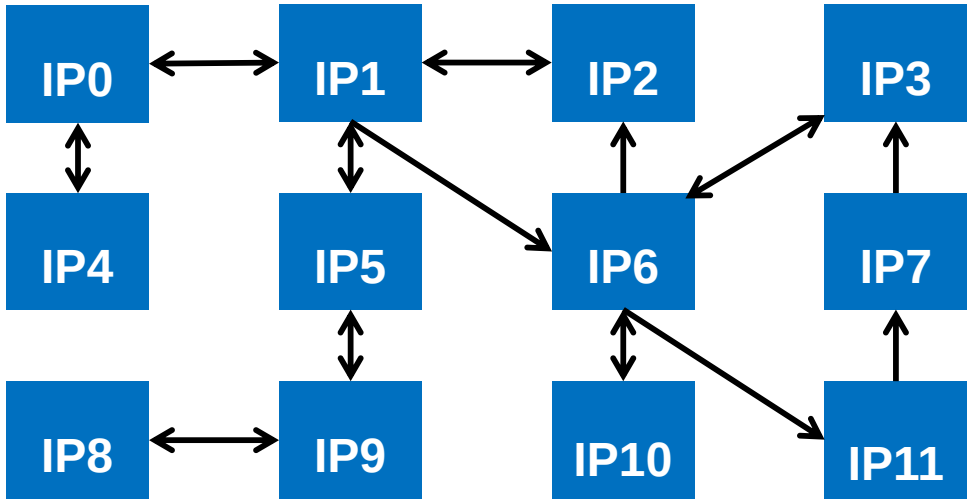
15 Combinations

Traffic to Stop
IP0 <-> IP1 + IP2 + IP3
IP1 <-> IP2 + IP3 + IP0
IP2 <-> IP3 + IP0 + IP1
IP3 <-> IP0 + IP1 + IP2
…

7 scenarios

PG/CG Tests for a Complex System

- When the number of IPs in a system keeps growing, the number of required tests grows exponentially. For example, in a system with 12 IPs, we have more than 4000 combinations and **A LOT OF** scenarios.



12 IPs

$f(n) = 2^n - 1$

4095 Combinations

Traffic to Stop
IP1 <-> IP4
IP1 <-> IP3
IP1 <-> IP11, IP3 <-> IP10
IP4 <-> IP3 + IP1 + IP11
…

Unknown Amount of Scenarios!

Solution Approach

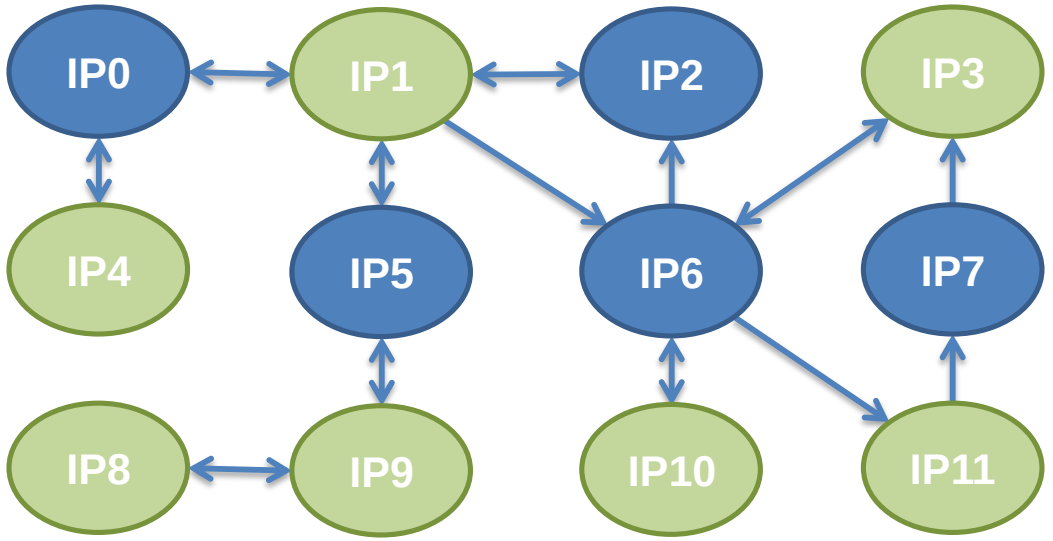
- To achieve the coverage goal in a complex system, manual setup for directed tests seems not feasible, and we need to consider using **random tests**.
- To help deploying CRV to these scenarios, we need a **comprehensive data structure** that contains the connectivity info of the entire system.
- The data structure shall be **simple to build** and **easy to access**.
- The build flow shall be **portable** for different levels and different verification environments.
- Optionally, the data structure shall be **easy to visualize**.

Graphical Topology Info Structure

- This proposed structure consists of nodes representing individual IPs and directed edges representing connection between adjacent IPs. In this way, it is formed as a directed graph.

Nodes	Edges
IP0	IP0_IP1
IP1 (Boundary)	IP0_IP4
IP2	IP1_IP2
…	…
IP11 (Boundary)	IP11_IP7

Graph Representation



Note: Boundary IP means the IP connects to the BFM, which serves as source and/or destination of the end-to-end traffic

Build Flow of the Info Structure

- **Step1: Create Base Classes for Data Structure**
  - Use UVM for example, but it works with other languages such as SystemC or C++.
  - Use uvm_object as the base, so that the data structure can be built anytime.
- Here is a list of UVM classes:

```
class ip_info extends uvm_object;
```

```
class connection_info extends uvm_object;
```

```
class topology_info extends uvm_object;
```

```
class datapath_info extends uvm_object;
```

Build Flow of the Info Structure

- **Step2: Setup Topology Descriptive File**
  - The file can be written by engineer or created by drawing tools (e.g., yEd).
  - The file can be in any format for convenience.
- Here is a Json file example:

```
"ip":{
   "IP0": {
      "is_boundary": 1, …
   }, …
}
"connection": {
   "ip2_ip3": {
      "protocol": "AXI",
      "source": "IP2",
      "destination": "IP3", …
   }, …
}
```

Build Flow of the Info Structure

- **Step3: Create Script to Convert the Descriptive File into Codes**
  - The script parses the descriptive file for the topology info and generates UVM codes to construct the data structure.
  - Plug in the generated codes into the test.
- Here is the code example:

```
class test_base extends uvm_test;
  …
    // Include generated construction codes
    topology_info.add_ip(…);
    …
    topology_info.add_connection(…);
    …
  endfunction
  …
```

Build Flow of the Info Structure

- **Step4: Create/Update Virtual Sequence**
  - The virtual sequence utilizes the excluded datapath when generating transactions.
- Here is the code example: virtual sequence iterates all sources and excludes the destination from each excluded datapath.

```
class virtual_sequence extend uvm_sequence;
  ip_info_array excluded_destination; //destinations from the
excluded datapath
  …
  function void set_excluded_destination(…);
  // Body
  virtual task body();
    …
    // Create traffic with randomization with the refined dest list
    send_transaction(source, dest_list);
    …
```

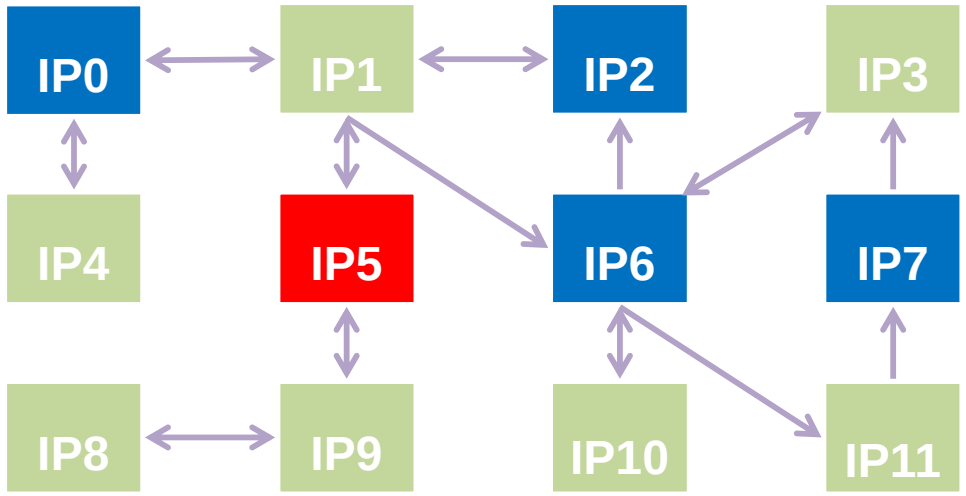
Build Flow of the Info Structure

- **Step5: Create PG/CG Tests**
  - Create PG/CG tests extending from the test base.
  - Setup traffic according to the needs.
- Here is the code example:

```
class pg_test extend test_base;
  …
  pg_ip = topology_info.get_ip_combinations();
  excluded_list = topology_info.get_datapath(pg_ip);
  …
  v_sequence.set_excluded_destination(excluded_list)
  …
  v_sequence.randomize();
  v_sequence.start();
…
```

Example of a PG Test

- When investigating IP5, helper function just needs to search and return all the datapaths with IP5 in the topology info structure.



Excluded Datapath

- IP1 -> IP8
- IP1 -> IP9
- IP3 -> IP8
- IP3 -> IP9
- IP4 -> IP8
- IP4 -> IP9
- IP9 -> IP11
- IP10 -> IP8
- IP10 -> IP9
- IP11 -> IP8
- ...

```
class pg_test extend test_base;
  datapath_info exlist [$];
  …
  exlist = t_info.get_excld(…);
  v_seq.set_source(all_src);
  v_seq.set_dest(all_dest);
  v_seq.set_excluded(exlist);
  v_seq.randomize();
  v_seq.start();
   …
```

Other Applications

- The topology info helps setting up tests to avoid traffic through the missing IP or broken IP.
- The topology info can carry information/pointer to the IPs' module UVC or interface UVC, so that the tests can easily access them to configure or control the checkers/scoreboards/monitors.
- This graphical topology info can be extended to embrace other functionality to help testing in more areas such as QoS or performance.

Limitations and Future Work

- End-to-End Transactions ONLY!
  - SW programming, RAS or interrupts can also trigger transactions through IPs. In this case, the source IP may not be at the boundary of the system.
- One Datapath ONLY from source to destination!
  - Some systems may have multiple datapaths between two IPs depending on the address ranges. To improve, the topology info needs to contain more routing attributes in the IP and the connection.
- No Loop in Datapath
  - Though the case with a loop in the datapath is very rare in most SoCs, the path searching function in graphical topology info can also be improved to handle this.

Summary

- SoC/subsystem tests can now easily leverage the constrained random approach when generating traffic for the desired datapath.
- One constrained random test replaces hundreds or thousands of directed tests or pre-generated tests. With a coverage-driven testing method, regression can achieve the coverage convergence quickly with multiple runs.
- The topology info is reusable in different SoCs and scalable with different number of IPs.
- The build flow can be deployed into different testing environment (i.e., emulation) or with different methodologies (i.e., SystemC, C++).
- **Engineering effort can be saved from the manual work in creating directed tests or configuring the verification environment for special needs and for better coverage**.

2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

THANK YOU

Q&A

accellera SYSTEMS INITIATIVE

Backup: Datapath Search Function

- The topology info is actually a graph, which can leverage different available algorithms to search a datapath. Without any concern on the runtime performance, the simplest way is to exhaustively traverse all sources towards the possible destination and build a lookup table.
- For example, lookup table for IP1, IP3, IP4, and IP8.

Destination / Source	IP1	IP3	IP4	IP8
IP1	1	1, 6, 3	1, 0, 4	1, 5, 9, 8
IP3	3, 6, 2, 1	3	3, 6, 2, 1, 0, 4	3, 6, 2, 1, 5, 9, 8
IP4	4, 0, 1	4, 0, 1, 6, 3	4	4, 0, 1, 5, 9, 8
IP8	8, 5, 9, 1	8, 9, 5, 1, 2, 6, 3	8, 9, 5, 1, 0, 4	8

- IP1 -> IP8
- IP3 -> IP8
- IP4 -> IP8
- IP8 -> IP1
- IP8 -> IP3
- IP8 -> IP4

Disclaimer & Attribution

DISCLAIMER

The information contained herein is for informational purposes only, and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale. GD-18

ATTRIBUTION