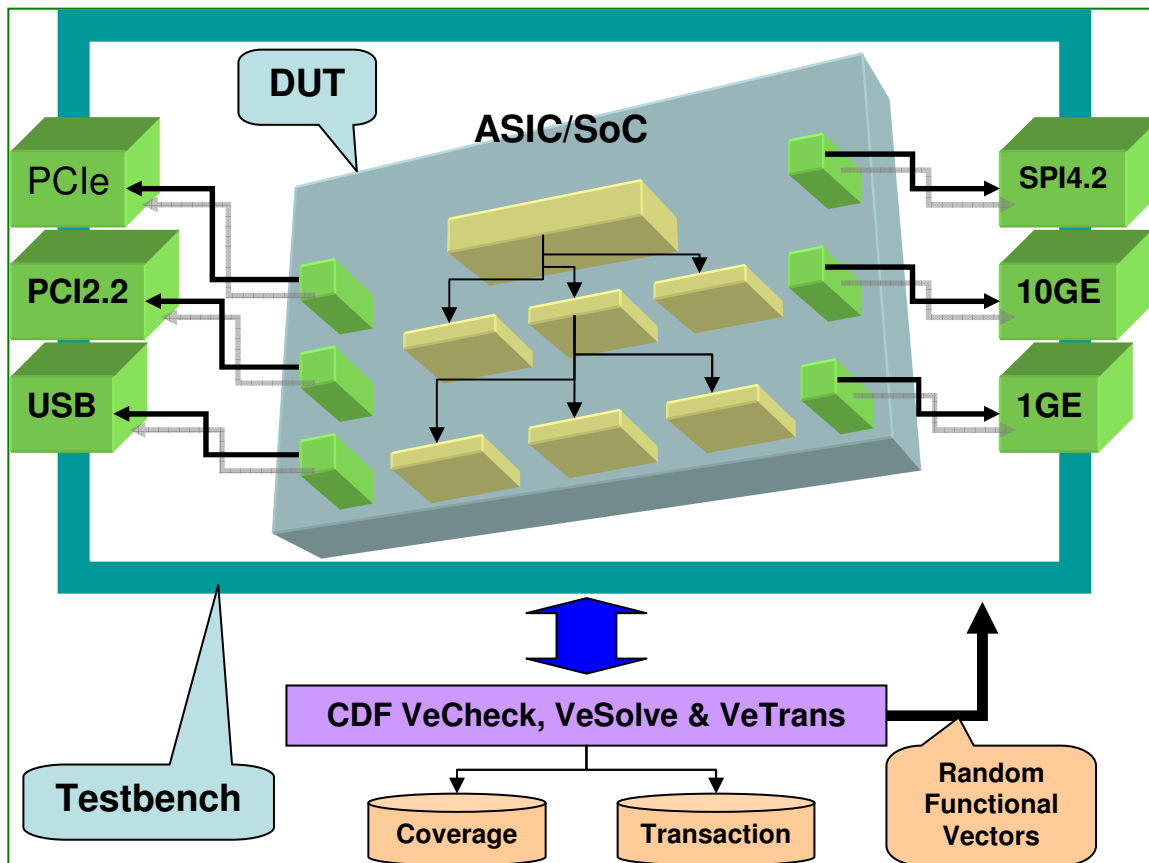


# The Comprehensive Design Framework

## Verification Automation Simplified!

**CDF** – the Comprehensive Design Framework provides true functional verification closure solution to ASIC/SoC design houses. The tool is a complex engineering package with 3 major components and a host of utilities that allow users to specify, measure, manage and converge on functional verification closure. When **CDF** is applied to a verification platform, the user will be able to –

1. Create an '**Executable Verification Plan**' and measure the quality of functional vectors against the derived goals
2. Apply a very efficient '**Constraint Synthesis**' and '**Random Vector Generation**' technique to create the missing vectors and converge on verification
3. Record and playback transactions for post-processing so that designers and verification engineers get to debug simulations at transaction levels



The **CDF**'s advanced utilities provide features like -

1. Functional test-grading
2. Coverage report generation from multiple regressions, including discrete runs across unit / chip or system levels
3. Assessment of coverage goals to estimate the quantum of vectors required to achieve desired coverage
4. Transaction viewer and debugger and transaction data mining utility

The **CDF** technology is entirely independent of the implementation language. The Testbench can be C/C++, Verilog/SystemVerilog/OpenVERA, VHDL ... The IP (coverage goals / data and verification plan) created using **CDF** is truly portable and can be easily migrated across various stages of a project (C based architectural exploration, HDL based RTL or even gates and acceleration or emulation) and across many projects. Further, **CDF** can itself fit into a larger framework which helps in verification management for complex projects that execute multi-site.

### 1. What is CDF?

CDF, short for '**Comprehensive Design Framework**' is a new tool suite to achieve functional verification closure for designs at IP, chip and system level design & verification. CDF introduces the concept of '**Verification Plan Driven Closure**'. In this methodology, the user focuses on writing the verification plan and the tool automatically generates code to enable –

- a. functional coverage checks
- b. automated constraint extraction to do constrained-random-vector generation

CDF further provides a rich set of utilities to converge on verification plan in an easy, intuitive manner.

CDF has been defined and created by engineers with design and verification background, who have managed and delivered several complex chip projects themselves. Hence, CDF brings in a very practical and novel approach for functional verification closure.

### 2. What is the difference between CDF based 'Verification Plan Driven Closure' methodology and 'Constraint Random Driven' methodology?

In a constraint random driven verification (supported by HVLs) model, the user writes two important components within his Testbench to manage & generate random vectors –

- a. functional coverage points
- b. constraints for related variables (which need to be randomized)

Typically, we converge on the functional vectors by refinement and elimination process. This involves –

- a. convergence on coverage goals through reviews
- b. debugging the constraints for any illegal vector combination
- c. refinement of constraints with the help of tool-provided utilities

Some of the key issues with this method are –

- a. coverage goals and constraints are written separately
- b. reviewing either of them is a challenge when designers, architects and marketing folks are involved in the process, whose inputs are very important for closure
- c. the flow is tied to the language of Testbench implementation. This defeats flows like C-based architecture evaluation, from where we might want to carry forward verification planning and functional coverage history

As we migrate to 'Verification Plan Driven Closure', we make these two major enhancements to our verification flow –

- a. create a mathematical model for the entire 'Verification Plan'. This can be termed as '**Executable Verification Plan**'
- b. de-couple the twin necessities – functional coverage and constrained vector generation from the Testbench by generating them automatically. This is a logically positive step since 'Verification Plan' is a representation of design specs and is NOT a mere Testbench component

Once the '**Executable Verification Plan**' is ready, CDF can assist the Testbench to automatically generate constrained random vectors (fully solved at word levels) and track functional coverage

### 3. What are the verification plan data structures supported by CDF?

CDF supports verification goal specification at individual fields, transaction or combination (with depth specification) and sequence levels. Using this, users can express all kinds of coverage goals such as –

- a. interface protocol functionality
- b. register space combination coverage
- c. interrupt functionality tracking
- d. power state machine coverage for SoC's & ASICs
- e. performance testing

More advanced structures such as parallel, discontinuous and event sequences are also supported in CDF framework. Using these, it is possible to express required verification points at IP, chip and system levels in a modular and truly re-usable fashion.

#### **4. What is the limitation on my Testbench language in case I want to use CDF?**

CDF does not impose any restriction on the language of Testbench implementation. CDF can work with Testbenches written in Verilog, SystemVerilog, C, SystemC etc. Further, the user is free to use simulator from any vendor.

#### **5. Once the CDF file is created, how can it be used for functional coverage tracking?**

CDF connects with the Testbench through sample points, which are the variables of interest that make up the verification plan. The sample points are evaluated by CDF at the sample events. CDF then tracks transactions and sequences specified in the '**Executable Verification Plan**'. CDF provides methodologies to either post-process coverage information or perform an on-line coverage query or employ a hybrid approach to achieve closure of functional verification.

#### **6. How can the CDF file be used for generation of constrained random vectors? Does the user have to write constraints as well?**

The user does not have to write any constraints to generate legal random vector combinations. CDF automatically extracts constraints from the '**Executable Verification Plan**'. It constructs the constraint trees within the simulation data structure and employs novel techniques to solve the constraints and expressions at word levels. The auto-generated vectors are returned to the Testbench to drive new transactions and sequences into the 'Design Under Verification'.

#### **7. How can CDF be applied at IP / Block level verification?**

At a block / IP level (take examples of USB, Ethernet Controller, PCIe Controller ...), CDF based '**Executable Verification Plan**' can be written for either communication interfaces (standard and back-end integration interface), the register space and interrupt logic. The Testbench can be setup to first invoke the **ACE** ('**Automatic Constraint Extraction**') engine to program the register space. Once the required configuration settings are done, random transactions (and desired sequence of transactions) can be automatically generated from the interface specific CDF files to be applied to the 'Design Under Verification'. In this way, we can generate several short tests (say, with 100 transactions each) and cover the entire verification plan in a couple of iterations using CDF's '**Hybrid Convergence**' approach.

#### **8. How can CDF be applied at SoC level verification?**

At SoC levels, the application of CDF is an extension of the IP/Block level methodology. Here, we can divide the verification into two phases –

- a. SoC minus CPU verification
- b. Full SoC verification

In the first phase, we replace the CPU with a central bus BFM hooking directly to the system bus of the SoC. Again, the Testbench uses CDF '**Executable Verification Plan**' for the configuration space and brings up the SoC. Using a simple text configuration file, we can then drive random transactions through each interface of the SoC in a controlled manner. This method can cover the entire verification plan for transactions, performance, arbitration and other directed tests. Only a few corner cases have to be coded by hand. The computing farm hence does more work to achieve closure than hand-created tests.

In the second phase, the entire SoC is driven by the Testbench. The transactions at the interface levels now have to be synchronized with the CPU executed code. This may need traditional techniques such as eyeballing to perfect the tests. However, the number of tests in this category is minimal. So, functional verification closure can be achieved quickly by applying CDF enabled '**Executable Verification Plan**' and '**Verification Plan Driven Closure**' techniques.

#### **9. I am a BFM vendor. How can CDF help me?**

As a BFM vendor, you can supply CDF based '**Executable Verification Plan**' along with your BFM to users. When the user compiles his simulation code with CDF library, it is ready to read the '**Executable Verification Plan**' at run time. With a thin, optional wrapper in front

of the BFM APIs (supplied by you, the vendor), the CDF can be applied to do coverage tracking and transaction generation automatically. Since the CDF file is read at run time, the user is free to edit the Testplan (for example : add more transactions / sequences, fine tune address space, prune coverage points etc) as per his specifications without having the need to access the BFM source code. Further, CDF can also be used to record transactions. CDF provides 2 mechanisms to post-process transactions – through a '**Transaction Browser GUI**' or through the '**Transaction Mining Utility**', which is a command line executable. The transaction recorders can help tremendously in debugging, where we can even let the user select the transaction and ask the tool to bring up your favorite Verilog / SV debugger into which the simulation signals associated with the transaction are automatically loaded and cursors are placed at the concerned transaction timestamp.

#### **10. What are the functional verification closure utilities provided by CDF?**

CDF is a very practical framework for functional coverage closure. It provides utilities like

- a. Verification Plan Qualifier
- b. Verification Plan Pre-processor and Parser
- c. XML to CDF convertor
- d. Functional Coverage Test Grader
- e. Coverage Database Upgrader
- f. Transaction Browser GUI
- g. Transaction Database Mining Utility
- h. CDF File Creator GUI

#### **11. How does CDF compare to assertions?**

Assertions and CDF need to co-exist. Assertions can be used to check corner cases and exception conditions at micro-architectural levels and can exist in design or Testbench code. CDF applies more at functional feature levels which are the only access points of the manufactured chip by the system and software. A robust verification closure methodology will employ all these techniques for a more confident RTL freeze.

#### **12. Is this a new language? What is the ramp-up time?**

Yes, CDF is a new language, rather like a meta-language. It is similar to XML, but is a reduced to ease on the verbosity. The entire language has very few keywords and the full LRM is just over 20 pages only, inclusive of examples. Compare this with other complex languages and you will appreciate the ease of learning. We have been able to train customers to create and review CDF in a few hours!

#### **13. I am already using HVLs. How will I benefit from CDF?**

HVL and SystemVerilog are great engineering solutions that are designed to create complex Testbenches. CDF does not provide any Testbench creation help. The complex actions of threading, synchronization, user defined data structures, score-boarding etc are still implemented using HVL and SystemVerilog. CDF enhances the setup by providing for '**Executable Verification Plan**' to achieve function verification closure.

#### **14. Once CDF functional coverage tracker is enabled, does it slow down my simulation?**

No. Since CDF is very light weight and uses several new generation techniques for coverage tracking and constraint solving, we will never slow down the simulation by more than 1-2%, which is negligible. Even the memory overhead is minimal and the '**Verification Plan Qualifier**' can upfront predict the memory loading in a couple of seconds for you.

#### **15. Can Axiom supply CDF files for standard protocols?**

Yes. We have CDF based '**Executable Verification Plan**' for several protocols.

#### **16. What is the roadmap for CDF in terms of enhancing the framework?**

We have plans to provide framework for '**Automatic Testbench Synthesis**' for SoCs, following which we can do '**Automatic Scenario Creation**' as well.

**17. How does CDF compare with any other commercially available verification planning tools?**

CDF has the core technology to develop a graphical user interface based verification planning tool with the following features –

- a. a visual interface to enter and review test item descriptions
- b. a visual interface to create and refine CDF verification items for each test description, with active annotation of CDF equation with each test item
- c. a mechanism to register a regression script
- d. a visual interface to assign and edit each test to an engineer, with all important attributes like priority, timeline etc
- e. a report generator which can provide a host of statistical reports (post-regression, inclusive of historical data management) with specialized views to high level executives (GM etc), D&V managers, verification engineers etc.

Axiom is well equipped to create and deploy this solution as an intranet-accessible utility to make it seamlessly available for complex, multi-site project requirements. The enabling core technology is already available with us. The new implementation is limited to database interface and GUI only.

**18. What are the other areas into which CDF can be applied?**

CDF can be very easily applied to software verification domain as well. CDF can prove to be a very productive utility in the verification of complex software like protocol stacks. Once the stack '**Verification Plan**' is created in CDF format, automatic vector generation can be employed to thoroughly regress the software stack.

**19. Is CDF available in production now?**

Yes. CDF is available as a production release on Linux platform interfaced with all popular Verilog simulators. Axiom will fully support its customers to jumpstart customers by providing training, '**Executable Verification Plan**' creation services and CDF integration services.

For more information on CDF, please get in touch with Axiom Sales at –

[sales@axiom-da.com](mailto:sales@axiom-da.com)

Axiom Design Automation, Inc.,  
1900 McCarthy Blvd,  
Suite 207, Milpitas,  
CA, 95035  
Tel: +1-408-433-9997  
Fax: 408-433-9998

Axiom EDA Products (India) Pvt. Ltd.,  
II Floor, # 406, 7th Main Road,  
Jayanagar II Block,  
Bangalore – 560 011  
Tel: +91-80-4061-7777  
Fax: +91-80-4061-7787