Verification flow

Michael Barskikh

Михаил Барских

Старший инженер по дизайну и верификации,
YADRO Microprocessors

- Закончил кафедру электроники МИФИ
- Кандидат технических наук
- 15 лет был разработчиком процессорных ядер на архитектуре MIPS
- Участвовал в выпуске нескольких поколений микросхем
- Тимлид команды верификации в YADRO Microprocessors

Все сложные дизайны содержат баги

Аксиома разработки

Что не проверено – не работает

Следствие для верификации

Сложность разработки ограничена сложностью верификации

Важное замечание

Вы можете доказать наличие багов, но не можете доказать их отсутствие*

* это пытаются исправить

Некоторые из проблем

- **Максимально быстрое обнаружение всех ошибок в проекте**
- **Возрастающая сложность дизайнов**
- **Ограниченное время на разработку** – «time-to-market»
- **Ограниченные ресурсы** – как человеческие, так и вычислительные
- **Увеличение стоимости ошибки на каждом следующем этапе проектирования SoC**

Немного статистики

Что такое верификация и зачем она нужна

Coverage driven verification

Декомпозиция задач

Шаблоны документов

Планирование верификации

Разработка окружения для повторного использования

Checking techniques

Python: Cocotb and PyUVM

Количество перезапусков проекта

FPGA projects are performing no better than ASIC

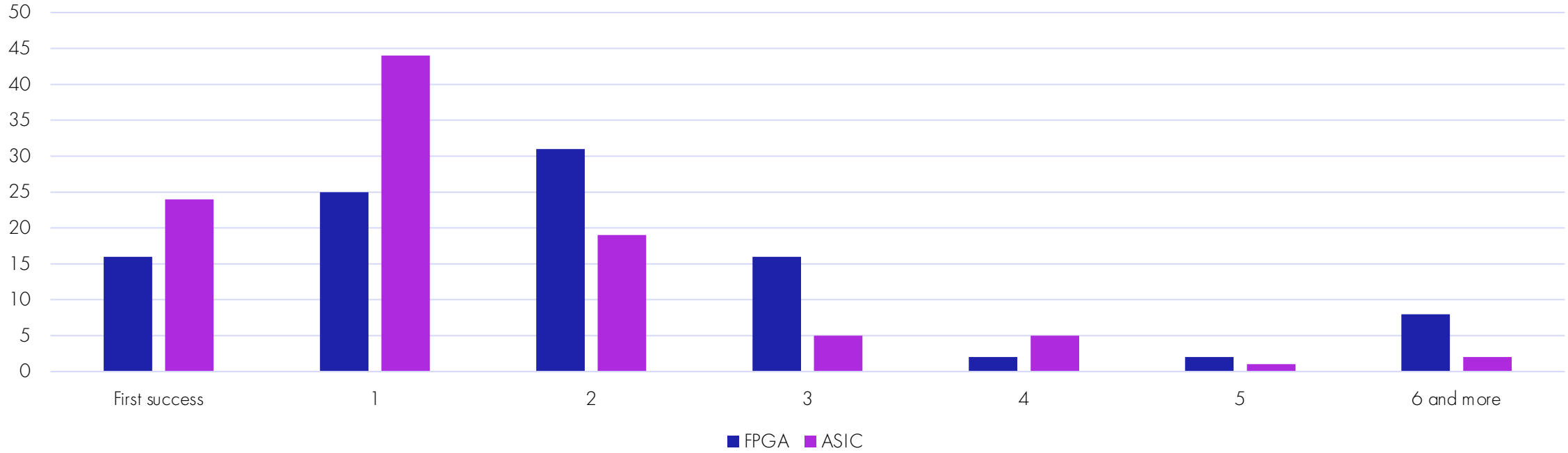
84%

FPGA design projects have non-trivial bug escapes into production

Number of spins before production

76%

ASICs require 2 or more respins



Источник: https://blogs.sw.siemens.com/verificationhorizons/2023/01/23/epilogue-the-2022-wilson-research-group-functional-verification-study/

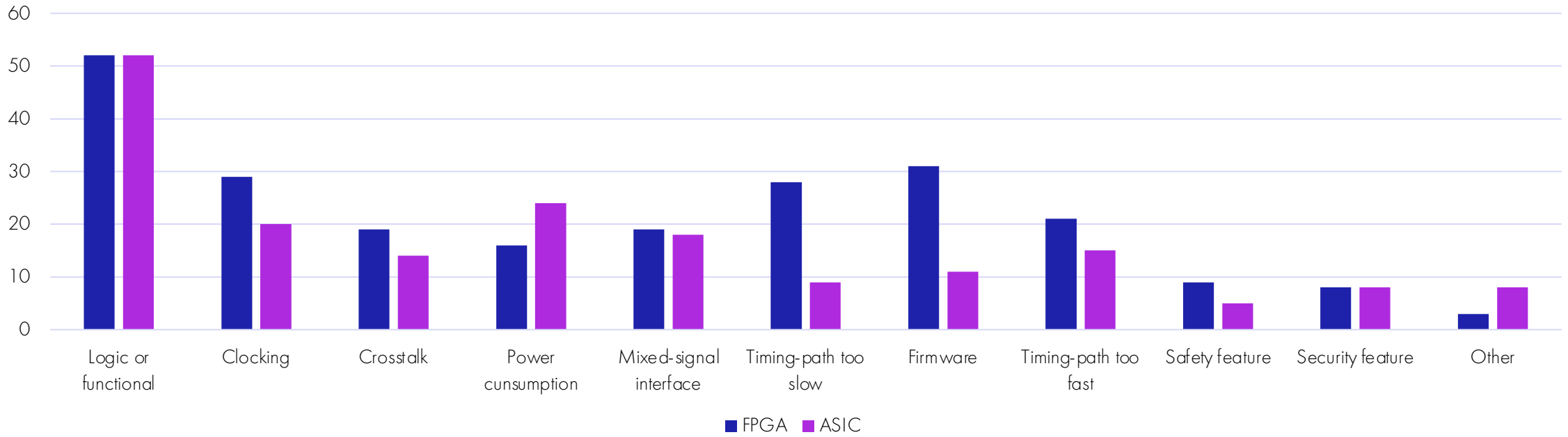
Причины перезапусков

Causes of FPGA non-trivial escapes into production

Logic/functional failures consistently the top cause of FGPA non-trivial bug escapes

Cause of ASIC respins

Logic/functional failures consistently the top cause of respines



Источник: https://blogs.sw.siemens.com/verificationhorizons/2023/01/23/epilogue-the-2022-wilson-research-group-functional-verification-study/

Расписание проекта

Most FPGA projects miss schedule

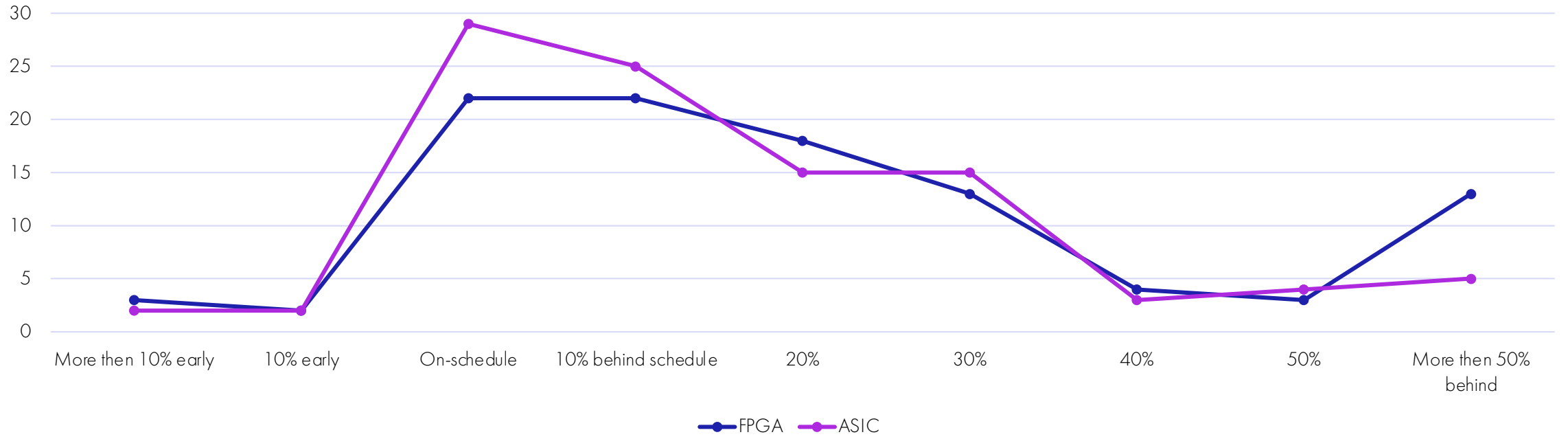
70%
projects behind schedule

12%
projects behind schedule more then 50%

Most ASIC projects miss schedule

66%
projects behind schedule

27%
projects behind schedule by 30% or more



Источник: https://blogs.sw.siemens.com/verificationhorizons/2023/01/23/epilogue-the-2022-wilson-research-group-functional-verification-study/

Время на верификацию

Percentage of FPGA project time spent in verification

40% — 50%

Median project time spent in verification

Percentage of ASIC project time spent in verification

50% — 60%

Median project time spent in verification

Источник: https://blogs.sw.siemens.com/verificationhorizons/2023/01/23/epilogue-the-2022-wilson-research-group-functional-verification-study/

Немного статистики

Что такое верификация и зачем она нужна

Coverage driven verification

Декомпозиция задач

Шаблоны документов

Планирование верификации

Разработка окружения для повторного использования

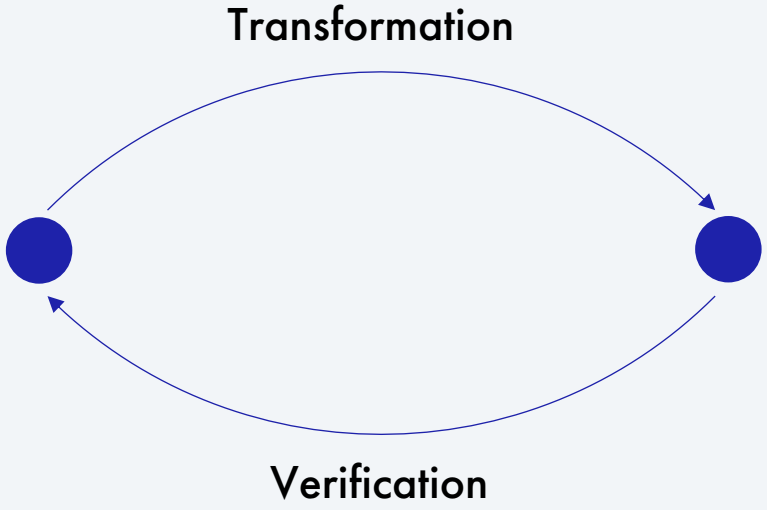
Checking techniques

Python: Cocotb and PyUVM

Реконвергентные пути в верификации

Реконвергентная модель — концептуальное представление верификации

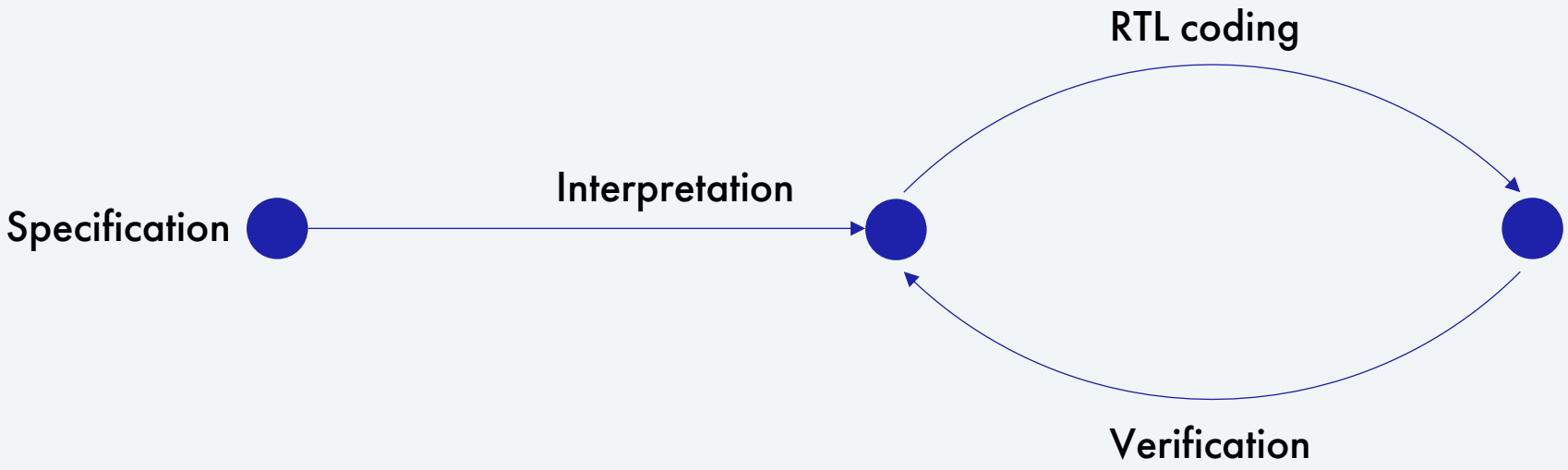
- **Цель верификации —** убедиться, что результат некоторого преобразования является предполагаемым или ожидаемым
- **Верификация —** это согласование с помощью различных средств спецификации и выхода
- **Верификация преобразования** может быть выполнена только через второй реконвергентный путь с общим источником



Человеческий фактор

Реконвергентные пути в неоднозначной ситуации

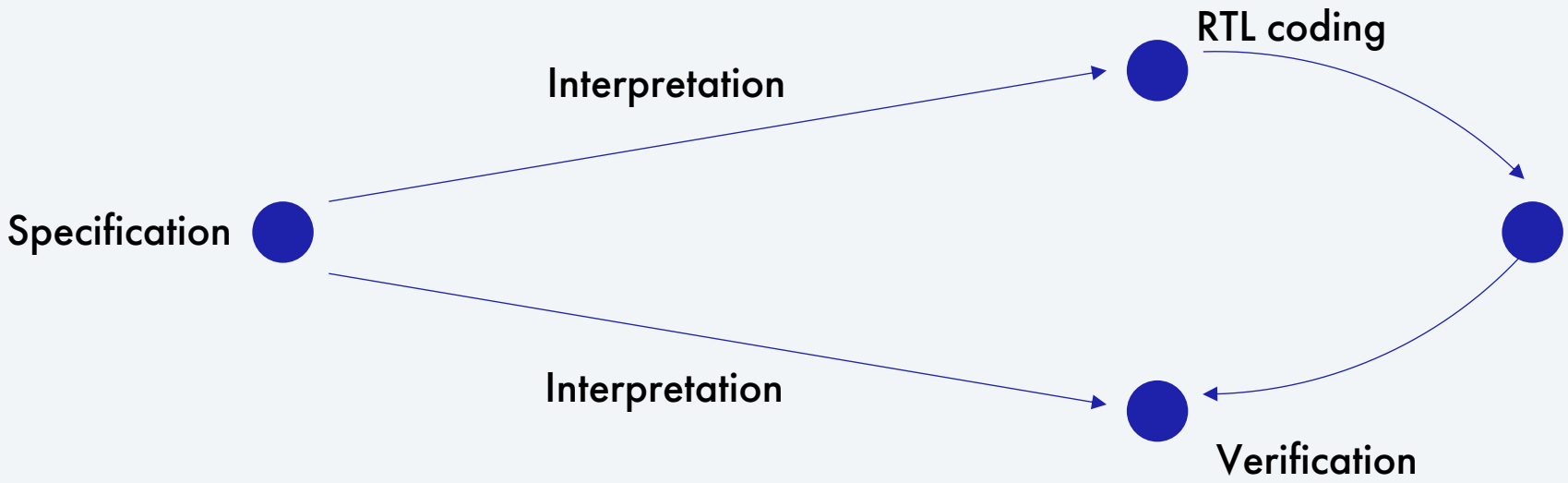
- Верификация собственного дизайна подтверждает его соответствие вашей интерпретации спецификации, а не самой спецификации



Избыточность

Верифицировать должен другой

- Верификация возможна при наличии избыточности в сомнительной ситуации



Немного статистики

Что такое верификация и зачем она нужна

Coverage driven verification

Декомпозиция задач

Шаблоны документов

Планирование верификации

Разработка окружения для повторного использования

Checking techniques

Python: Cocotb and PyUVM

Традиционный подход: направленные тесты

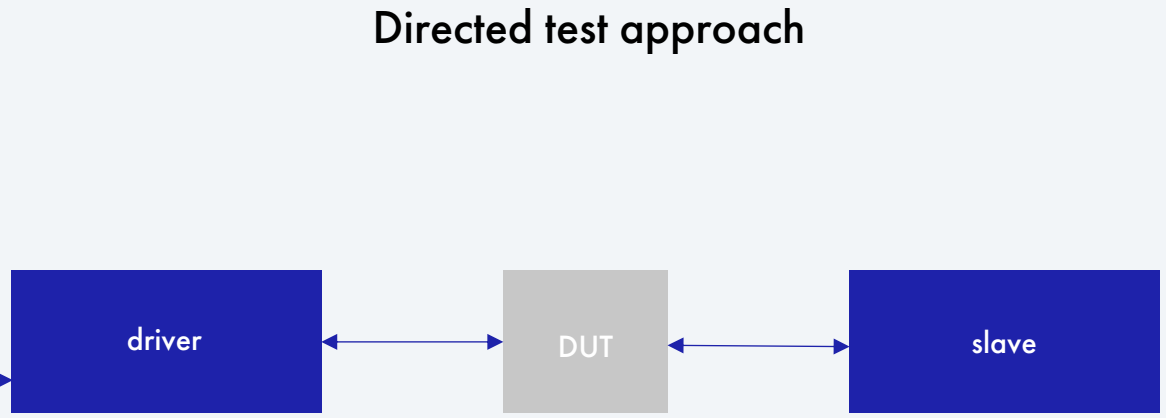
Состав направленного теста

- Каждый из направленных тестов содержит стимул, но требуется что-то большее...
- В тесты обычно встроены проверки для проверки правильности поведения (обычно специфичные для данного сценария тестирования)
- Прохождение каждого теста означает, что функциональность была реализована или «покрыта»

Многократное использование и поддержка

- Тесты могут стать довольно сложными и трудными для понимания цели того, какая функциональность проверяется
- Поскольку проверки распределены по всему набору тестов, их постоянное обновление требует больших усилий
- Обычно сложно или невозможно повторно использовать тесты в разных проектах или от модуля к уровню системы

Directed tests			
1	stimulus	check	cov
2	stimulus	check	cov
3	stimulus	check	cov
4	stimulus	check	cov



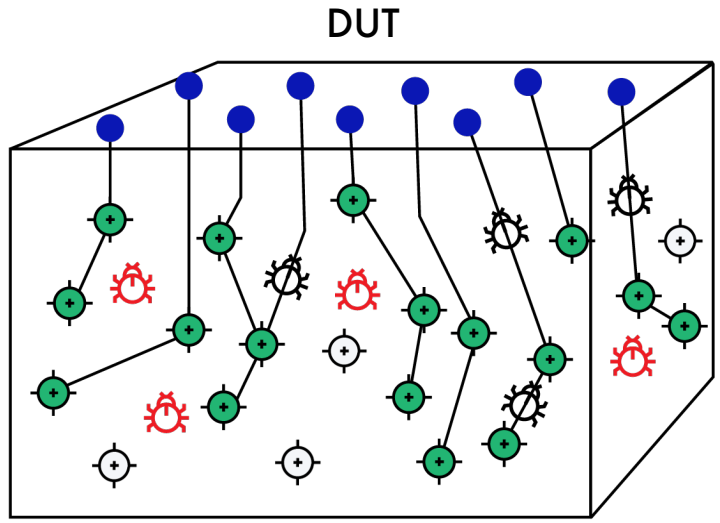
Традиционный подход: направленные тесты

Верификатор:

- Определяет состояния тестируемого устройства на основе поведения спецификации и крайних случаев
- Пишет направленный тест для посещения и проверки каждого элемента плана тестирования

Проблемы:

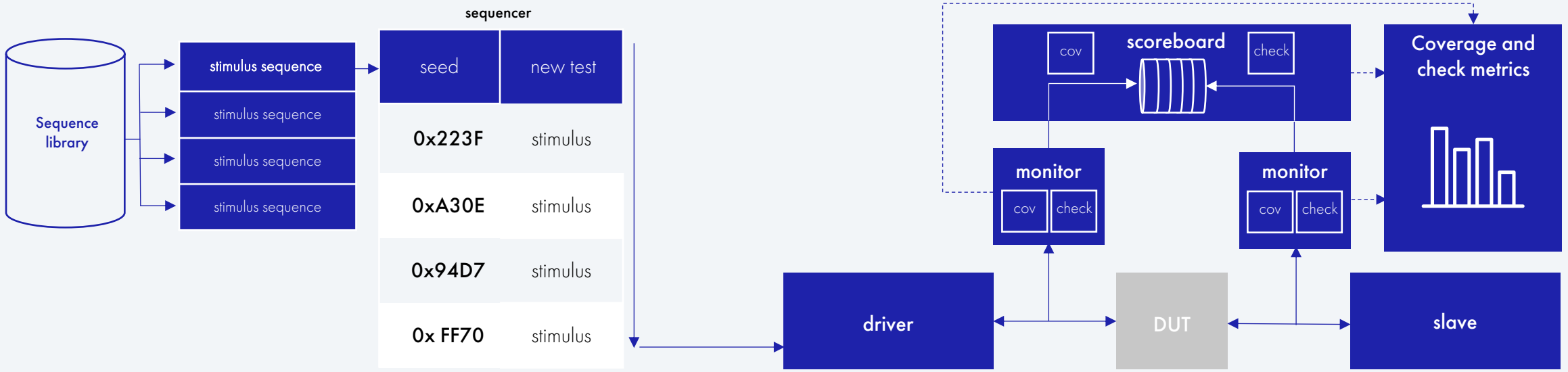
- Значительные ручные усилия для написания всех тестов
- Работа, необходимая для проверки достижения каждой цели
- Плохое освещение нецелевых сценариев... особенно тех случаев, о которых вы «не подумали»



Генерация тестов, управляема покрытием

Состав верификационной среды:

- Многоразовые последовательности стимулов, разработанные с использованием «constrained random generation»
- Запуск уникальных seed'ов позволяет среде реализовывать различные функции
- Мониторы самостоятельно наблюдают за окружением
- Независимые проверки обеспечивают правильное поведение
- Независимые точки покрытия указывают, какие функции были реализованы



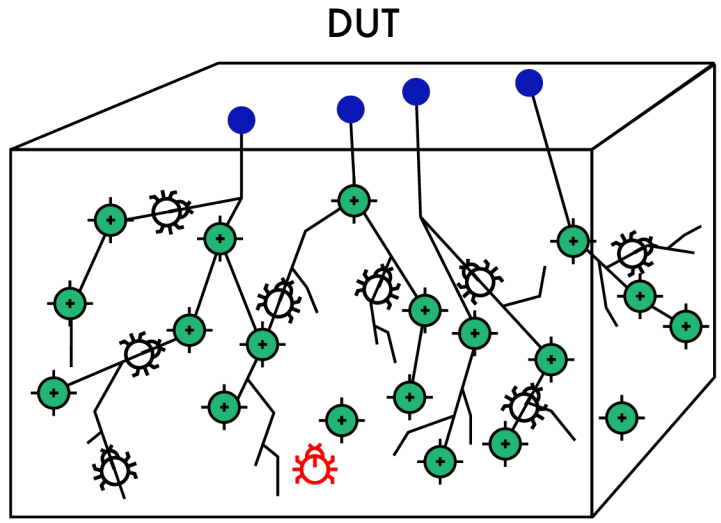
Традиционный подход: направленные тесты

Верификатор:

- Определяет состояния тестируемого устройства на основе поведения спецификации и крайних случаев
- Пишет направленный тест для посещения и проверки каждого элемента плана тестирования

Автоматизация:

- Constrained random generation ускоряет достижение целей охвата и выявление ошибок
- Результаты покрытия и результаты проверок указывают на эффективность каждого моделирования
- Позволяет множеству параллельных прогонов способствовать масштабированию усилий по проверке... Даже для нецелевых состояний



Планирование – самое важное

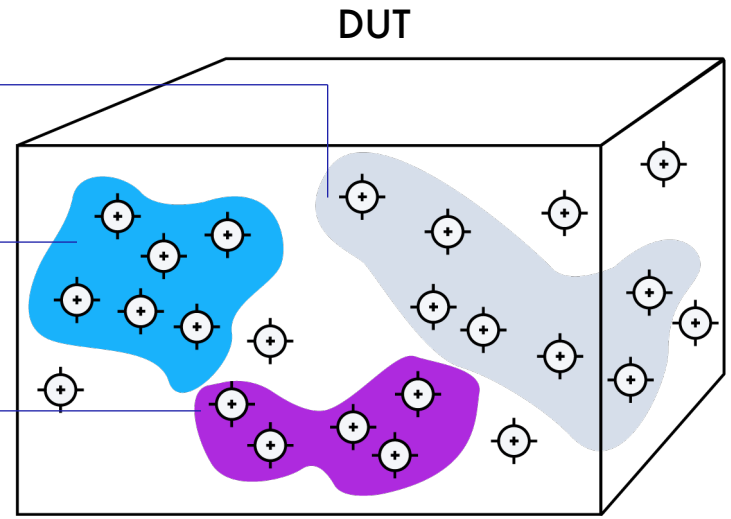
DUT Feature-Based Plan

- Input Interface A
  Coverage & check requirements
- Core Function B
  Coverage & check requirements
- Output Interface C
  Coverage & check requirements

66%*

100%

33%



*** Plan Specifies Coverage and Checks Required for Each DUT Feature:**

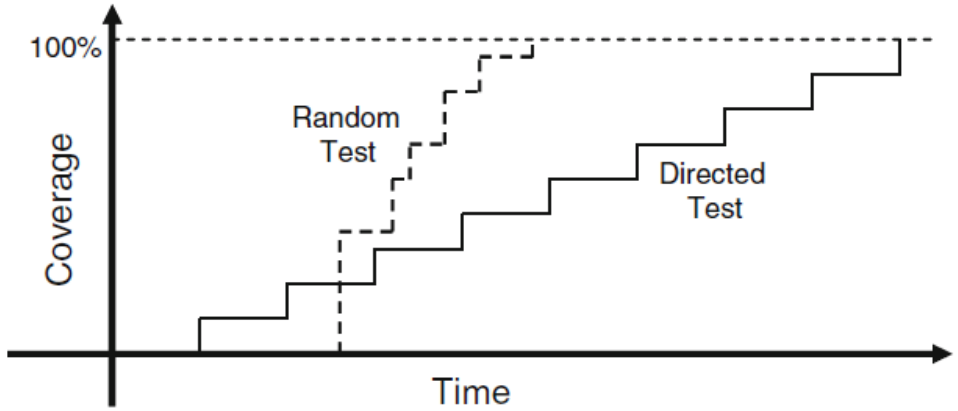
Verification Goals based on:

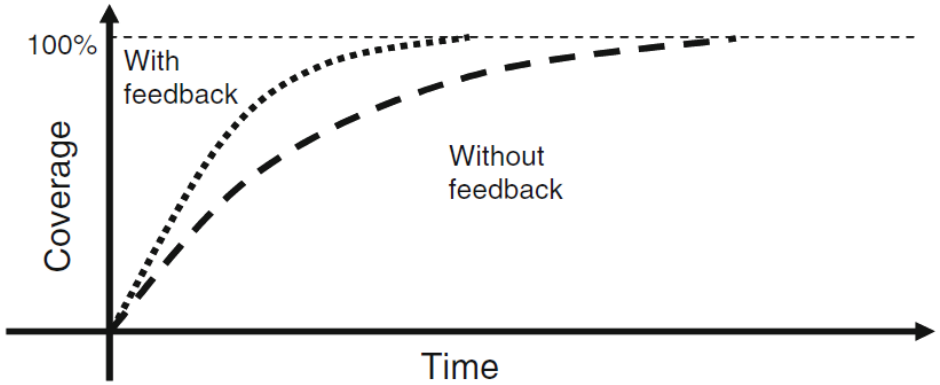
- Analysis of specifications
- Experience of the team

*** Plan Provides Feature Based Analysis and Tracking of Progress**

Coverage & check metrics from regressions annotated back to Features in the Plan

Coverage driver verification





SystemVerilog for Verification. Chris Spear

Writing Testbenches using SystemVerilog. Janick Bergeron

Рандомизация:

- Конфигурация устройства
- Конфигурация окружения
- Входные данные
- Исключения и граничные условия протокола
- Задержки
- Ошибки и нарушения

Немного статистики

Что такое верификация и зачем она нужна

Coverage driven verification

Декомпозиция задач

Шаблоны документов

Планирование верификации

Разработка окружения для повторного использования

Checking techniques

Python: Cocotb and PyUVM

Что нужно для верификации

План тестирования

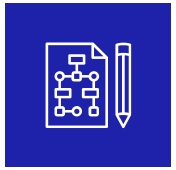
Что и как мы будем проверять,
как мы будем контролировать
прогресс

Система сборки проекта

Надо уметь запустить проект на моделирование,
получить результаты, уметь их воспроизвести

Тестовое окружение проекта

То, что будет подавать тестовые
воздействия и смотреть
за ответами системы



Тесты

Собственно то, что
мы будем запускать

Инструменты отладки

Механизм воспроизведения
ошибок и отладки

Механизм контроля и измерения прогресса

Надо разработать модель покрытия,
уметь собирать и анализировать его

Немного статистики

Что такое верификация и зачем она нужна

Coverage driven verification

Декомпозиция задач

Шаблоны документов

Планирование верификации

Разработка окружения для повторного использования

Checking techniques

Python: Cocotb and PyUVM

Verification Plan

- **Входные данные для верификации** (документация и DUT)
  - Что дано в качестве входа с версиями (git hash)
- **Подход к разработке тестового окружения**
  - Разработка scoreboard или референс-модели
  - Распределение работы между командами
- **Объем верификации**
  - Connectivity
  - Тесты регистров
  - Функциональные тесты
  - Негативное тестирование
  - Тестирование производительности
  - Тестирование потребления
  - GLS
- **Сценарии***
- **Требования к тестовому окружению***
- **Описание модели покрытия***

Verification Scenarios

- **Проверяемые характеристики**
- **Охват сценариев**
  - Точнее, что сценарии НЕ охватывают
- **Сценарии**
  - Сценарий **<N>**
    - Описание
    - Последовательность действий
    - Платформы
    - Условие прохождения
- **Распределение работы между командами**

Environment Description

- **Инструменты и методологии**
- **Структура тестового окружения**
  - Статические интерфейсы
  - Подключение компонент
    - (*agents*) к портам DUT
    - для преобразования транзакций
    - для сбора покрытия
    - checkers
    - внешние источники данных или генераторы воздействий
  - Подключения компонентов окружения через TLM-порты
  - Места вывоза uvm_callbacks
- **Функциональная модель** (*scoreboard*)
- **Структура и иерархия sequence**
  - Структура и иерархия библиотек HAL
- **Модель покрытия**
- **Запуск тестов**
- **Контроль корректности завершения тестов**

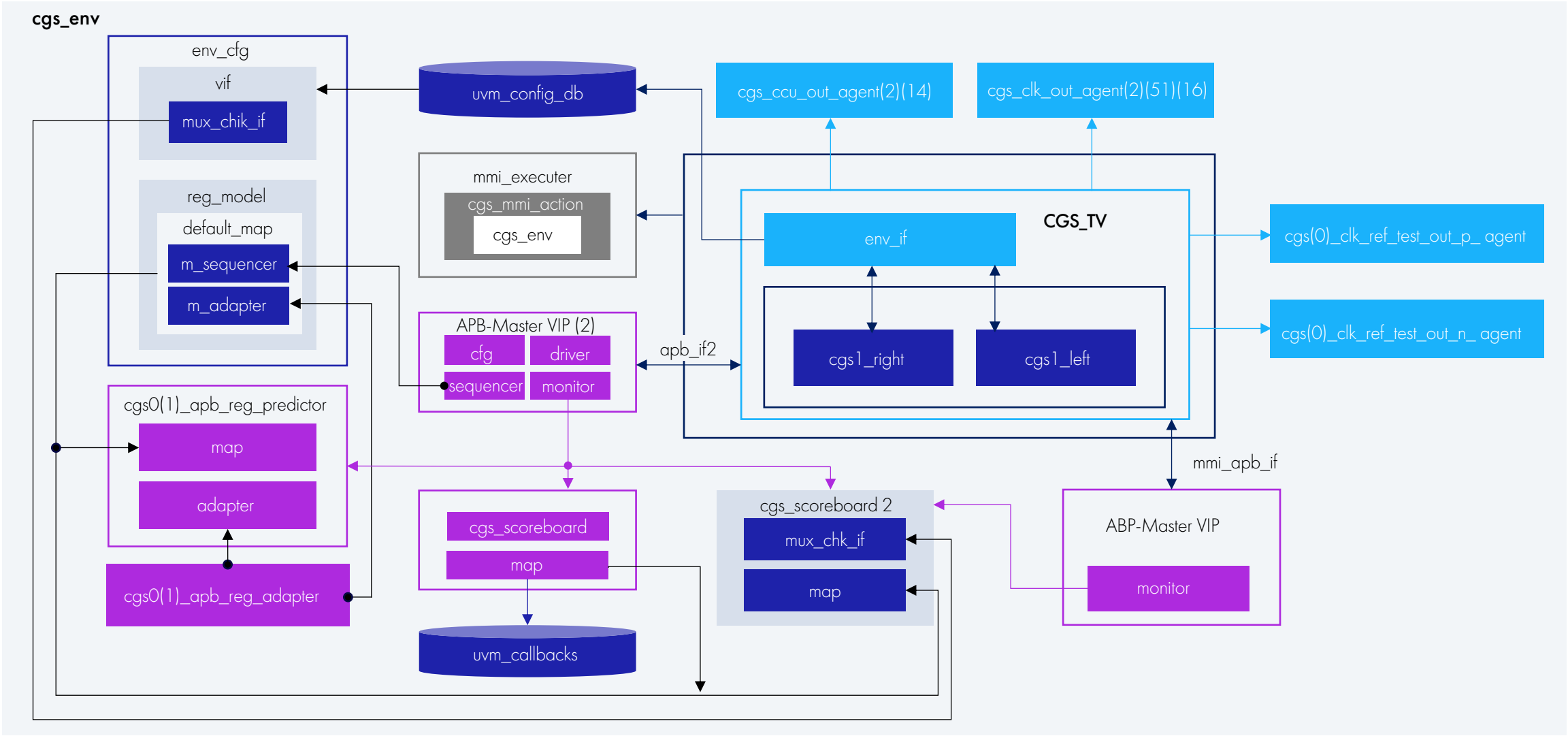
Functional Coverage Description

- **Кодовое покрытие**
  - Блок **<N>**
    - Метрика
  - Интерфейсы
- **Функциональное покрытие**
  - Метрика **<N>**
    - Атрибут
    - Собираемые значения
    - Место сбора
    - Момент сбора
- **Формальные утверждения (assertions)**
  - Assertion **<N>**
    - Описание утверждения со ссылкой на документацию
- **Направленные тесты**
  - Тест **<N>**

Verification Reports

- **Входные данные для верификации** (документация и DUT)
  - Что дано в качестве входа **с версиями** (*git hash*)
    - Архитектурная спецификация
    - DUT
    - Описания тестового окружения и сценариев
- **Выходные данные**
  - Что получено в качестве результата **с версиями** (git hash)
- **Реализованные тесты**
  - Список, параметры, строки запуска
- **Отчет о покрытии**
- **Статистика по результатам проведенной работы**
  - количество найденный ошибок дизайне
  - время, потраченное на разработку
- **Изменения относительно исходных планов**

Пример структуры тестового окружения



Немного статистики

Что такое верификация и зачем она нужна

Coverage driven verification

Декомпозиция задач

Шаблоны документов

Планирование верификации

Разработка окружения для повторного использования

Checking techniques

Python: Cocotb and PyUVM

Проблемы разработки верификационного плана

- **Как узнать, что мой план полный?**
  - Нет недостающих частей
  - План точный
  - Отражает функциональные и проектные характеристики, включая изменения спецификаций

- **Как написать план, который можно будет использовать позже?**
  - От модульного уровня к уровню системе
  - Между проектами
  - Среди разных разработчиков в команде

- **Как мы отслеживаем наш прогресс в выполнении этого плана?**
  - Динамическое отслеживание полноты каждой запланированной характеристики с использованием измеряемых показателей (на основе покрытия и утверждений)
  - Точно прогнозируйте график верификации и отслеживайте прогресс в соответствии с целями, установленными для каждой контрольной точки проекта

Составление плана

- План определяет конкретные измеримые цели для того, что должно быть проверено

  Т.е. их можно измерить функциональным покрытием

- План выражает точное намерение «что» будет проверяться (а не «как»)

  Может быть понят всеми заинтересованными сторонами

- План обеспечивает отслеживание именно того, что было проверено

  Последний статус покрытия на основе плана показывает, что вы еще не протестировали

- System Bus Subsystem Features to be verified:
  - **Transfer Types:** The bus supports both single transfers and burst transfers
  - **Transfer Directions:** The bus supports both read and write transfers
  - **Transfer Targets:** The bus supports transfers from the master to all slaves
- Packet Interface
- Graphics Engine

Источники для составления плана верификации

brainstorming

Directed tests

100% uart_tests
- 100% uart_tests.apb_to_uart_1stopbit_test
- 100% uart_tests.u2a_a2u_full_rand_test
- 100% uart_tests.apb_uart_rx_tx_data_aa
- 100% uart_tests.cdn_uart_scoreboard_traffic
- 100% uart_tests.uart_bad_parity_test
- 100% uart_tests.uart_incr_payload_test
- 100% uart_tests.uart_bad_driver_factory

Code coverage and other metrics

- uart_dut (uart)
  - ua_apb_if1 (uart_apb_if)
  - ua_brg1 (uart_baud_rate)
  - ua_ctrl1 (uart_control)
  - ua_int_ctrl1 (uart_int_ctrl)
  - ua_mod_ctrl1 (uart_modem_ctrl)
  - ua_mode_sw1 (uart_mode_switch)
  - ua_rcvr1 (uart_receiver)

Test pass/ fail metrics

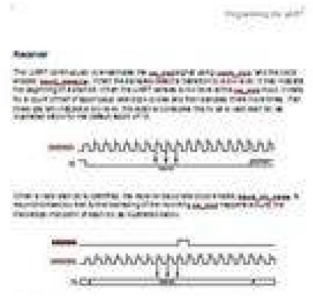
Coverage metrics

Verification plan

Specification outline of features

Other contributing vPlans

Outline from a functional spec



VIP Compliance vPlan and module level vPlans

Какие метрики мы включаем в план?

Различные метрики дополняют друг друга:

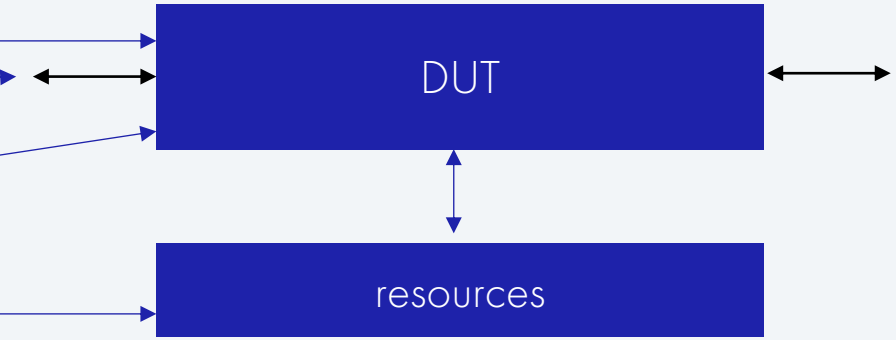
- На основе спецификации
  - Функциональное покрытие
  - Формальные утверждения (assertions)
- На основе реализации
  - Покрытие кода (например, покрытие FSM, блоков, выражений)
- На основе сценариев использования
  - Сложные случаи использования (например, загрузка прошивки)

Ни один показатель сам по себе не является полным.

План и анализ «общего покрытия» имеют важное значение

Некоторые промеры:

- Режимы устройства и опции конфигурации
- Протокол и трафик
- Обработка ошибок
- Внутренние свойства устройства
- Арбитрация внешних ресурсов
- Типичные и критические пользовательские сценации



Составление модели покрытия

Функции плана верификации преобразуются в набор поддающихся количественному измерению показателей, отвечая на вопрос КАК они будут измеряться

- **Какие** атрибуты и значения важны для каждой функции?
- **Где** следует наблюдать каждое значение?
- **Когда** допустимо брать значения для выборки?

  (Когда они дают результаты, которые можно измерить?)

- System Bus Subsystem Features to be verified:
  - **Transfer Types:** The bus supports both single transfers and burst transfers
  - **Transfer Directions:** The bus supports both read and write transfers
  - **Transfer Targets:** The bus supports transfers from the master to all slaves
- Packet Interface
- Graphics Engine

Составление модели покрытия

Функции плана верификации преобразуются в набор поддающихся количественному измерению показателей, отвечая на вопрос КАК они будут измеряться

- **Какие** атрибуты и значения важны для каждой функции?
- **Где** следует наблюдать каждое значение?
- **Когда** допустимо брать значения для выборки?

  (Когда они дают результаты, которые можно измерить?)

Coverage Items

Transfer_types: single, burst
Transfer_direction: read, write
Transfer_targets: (all slaves) 0xa000, 0xa004, 0xa008, 0xa00C
cross: transfer_types, transfer_direction, transfer_targets

- System Bus Subsystem Features to be verified:
  - **Transfer Types:** The bus supports both single transfers and burst transfers
  - **Transfer Directions:** The bus supports both read and write transfers
  - **Transfer Targets:** The bus supports transfers from the master to all slaves
- Packet Interface
- Graphics Engine

Составление модели покрытия

Функции плана верификации преобразуются в набор поддающихся количественному измерению показателей, отвечая на вопрос КАК они будут измеряться

- **Какие** атрибуты и значения важны для каждой функции?
- **Где** следует наблюдать каждое значение?
- **Когда** допустимо брать значения для выборки?

  (Когда они дают результаты, которые можно измерить?)

Coverage Items

Transfer_types: single, burst
Transfer_direction: read, write
Transfer_targets: (all slaves) 0xa000, 0xa004, 0xa008, 0xa00C
cross: transfer_types, transfer_direction, transfer_targets

Observed by the system bus monitor

- System Bus Subsystem Features to be verified:
  - **Transfer Types:** The bus supports both single transfers and burst transfers
  - **Transfer Directions:** The bus supports both read and write transfers
  - **Transfer Targets:** The bus supports transfers from the master to all slaves
- Packet Interface
- Graphics Engine

Составление модели покрытия

Функции плана верификации преобразуются в набор поддающихся количественному измерению показателей, отвечая на вопрос КАК они будут измеряться

- **Какие** атрибуты и значения важны для каждой функции?
- **Где** следует наблюдать каждое значение?
- **Когда** допустимо брать значения для выборки?

  (Когда они дают результаты, которые можно измерить?)

Coverage Items

Transfer_types: single, burst
Transfer_direction: read, write
Transfer_targets: (all slaves) 0xa000, 0xa004, 0xa008, 0xa00C
cross: transfer_types, transfer_direction, transfer_targets

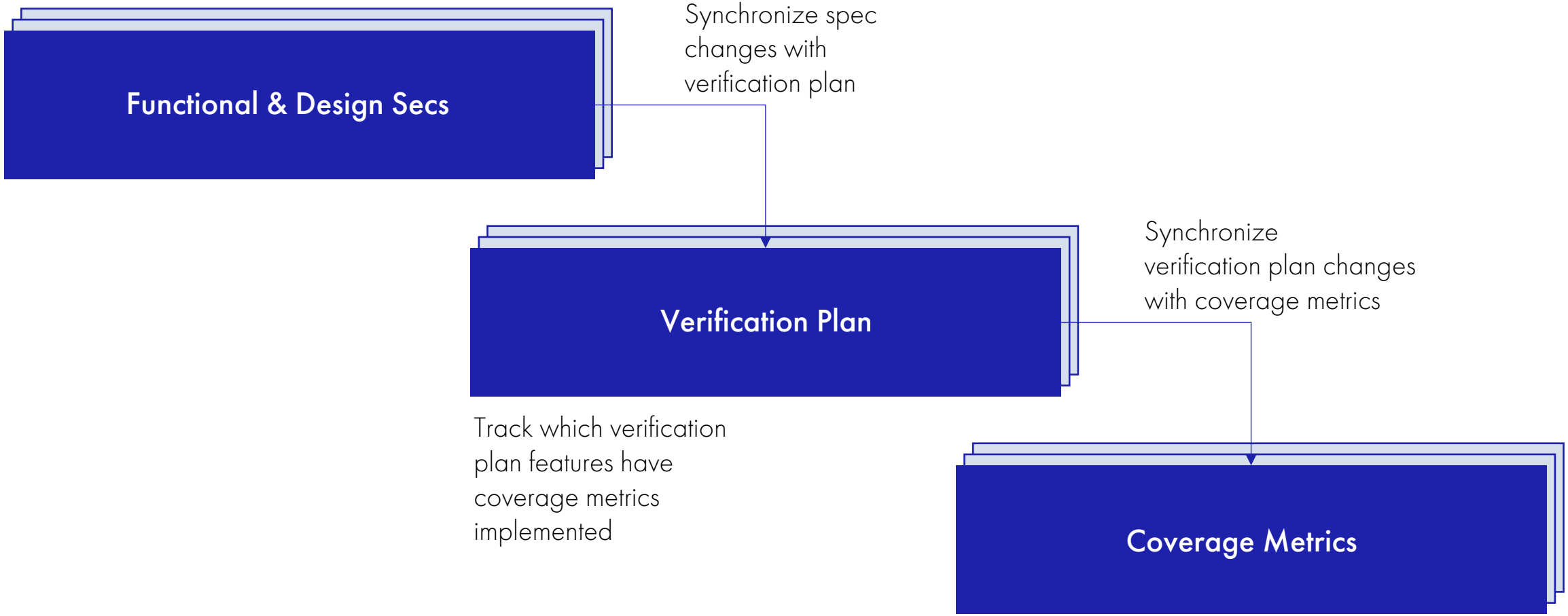
Observed by the system bus monitor

Sampled when transfer_completes

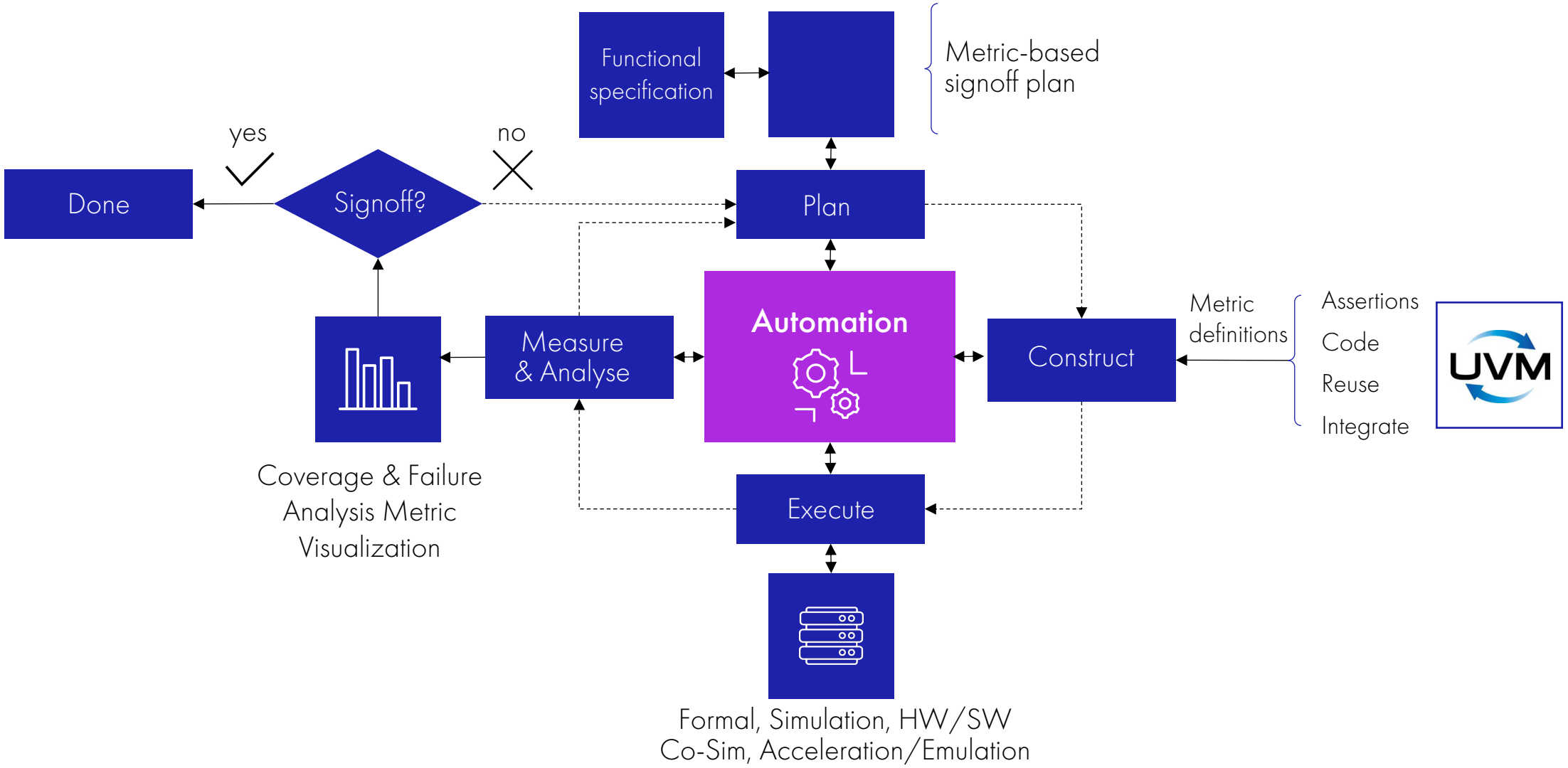
- System Bus Subsystem Features to be verified:
  - **Transfer Types:** The bus supports both single transfers and burst transfers
  - **Transfer Directions:** The bus supports both read and write transfers
  - **Transfer Targets:** The bus supports transfers from the master to all slaves
- Packet Interface
- Graphics Engine

Связь спецификации с моделью покрытия

Specification → Verification Plan → Coverage Metrics



The Coverage Driven Verification Cycle



Code vs. Function coverage

> 97%

Code coverage

= 100%

Function coverage

* возможно, с исключениями

В качестве небольшого итога

Планирование верификации дает:

- Понимание объема работ
- Разделение работ по командам
- Понимание узких мест
- График работ
- Понимание прогресса

Выполнение плана смотрится по точкам функционального покрытия.

Их разработка - задача команды верификации.

Немного статистики

Что такое верификация и зачем она нужна

Coverage driven verification

Декомпозиция задач

Шаблоны документов

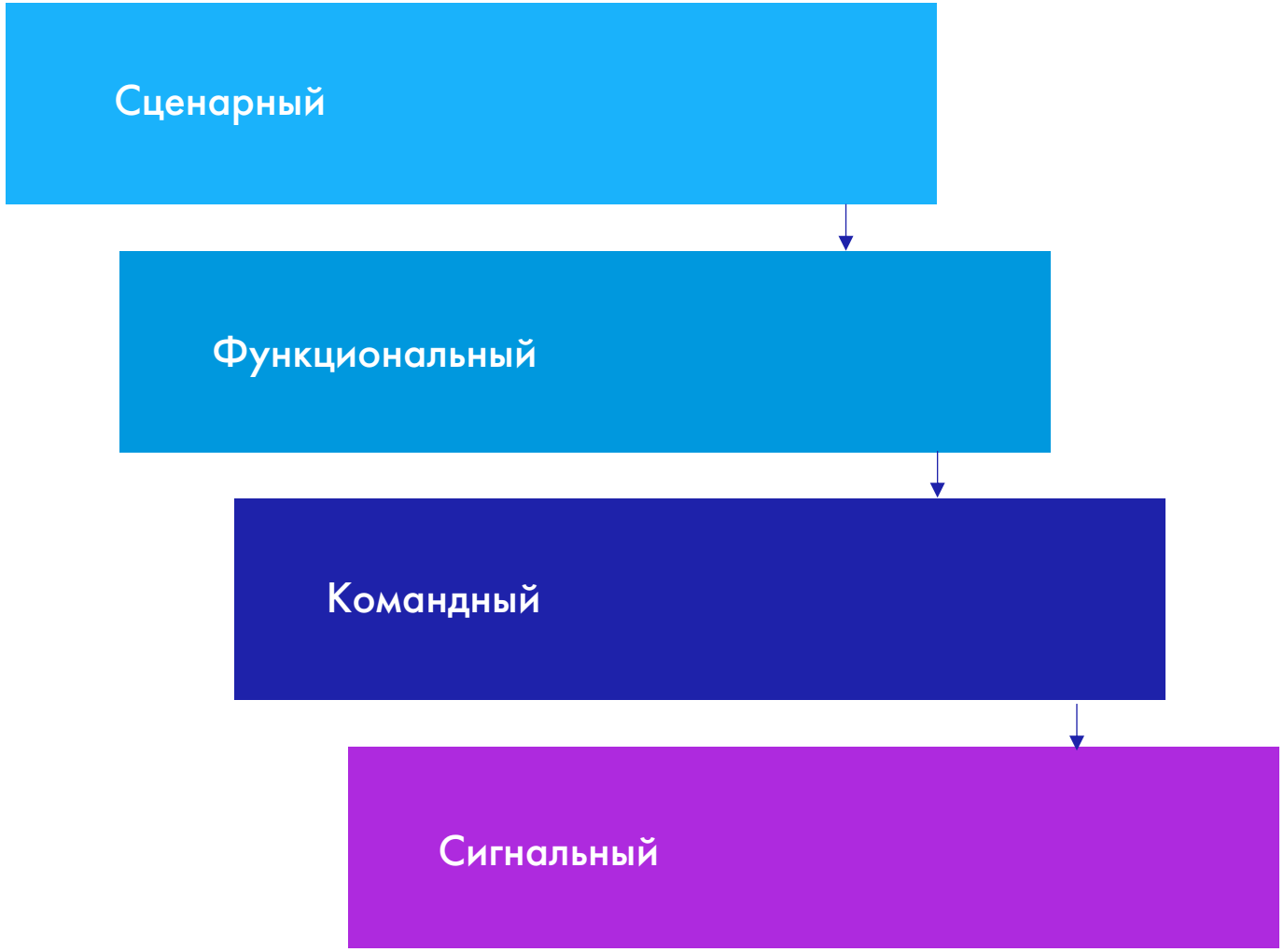
Планирование верификации

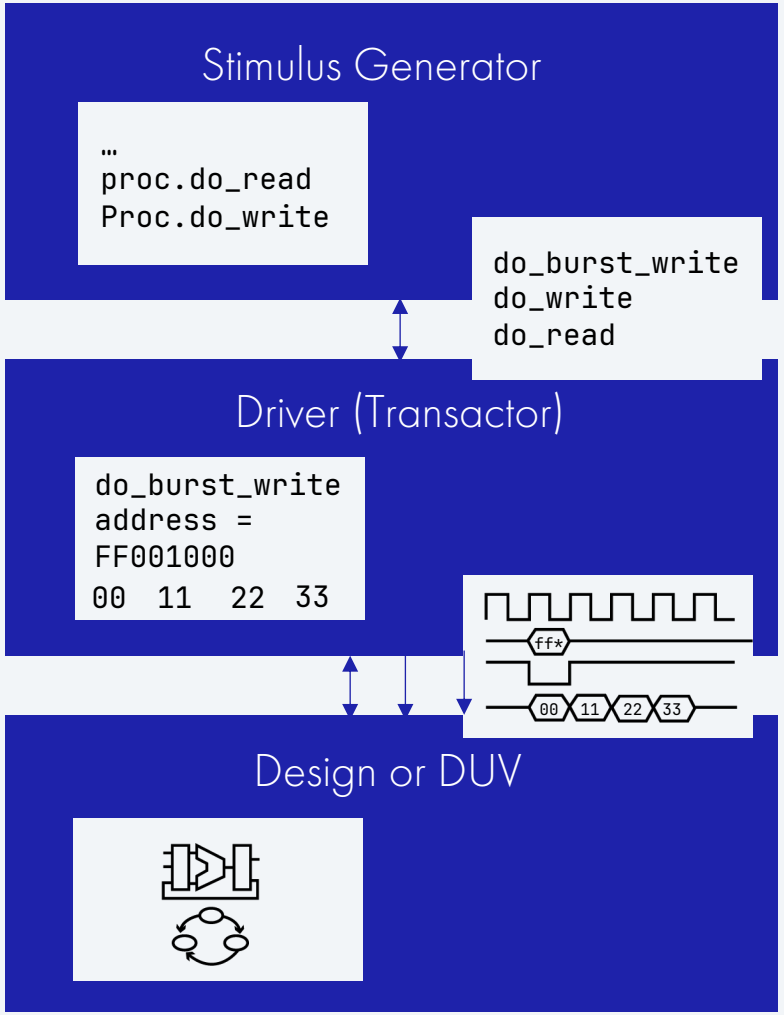
Разработка окружения для повторного использования

Checking techniques

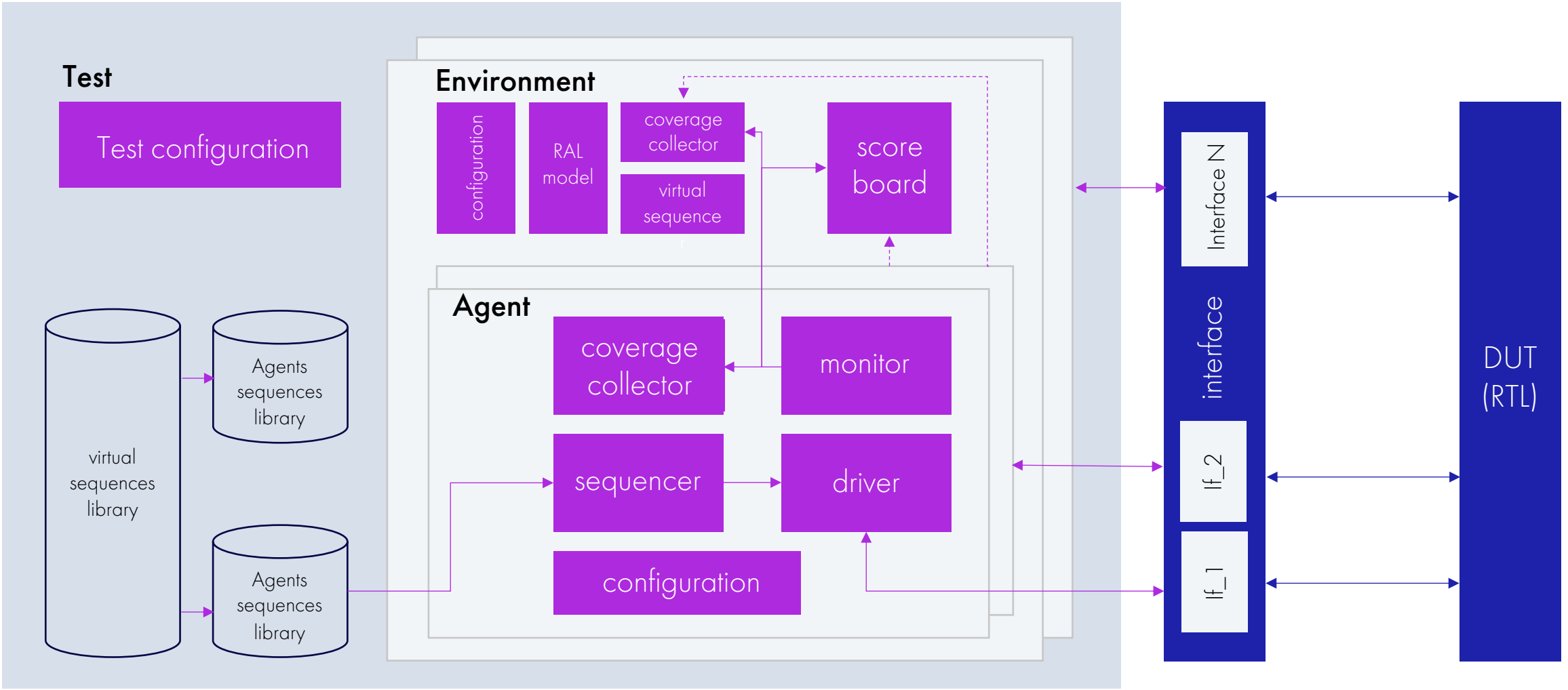
Python: Cocotb and PyUVM

Многоуровневая модель окружения





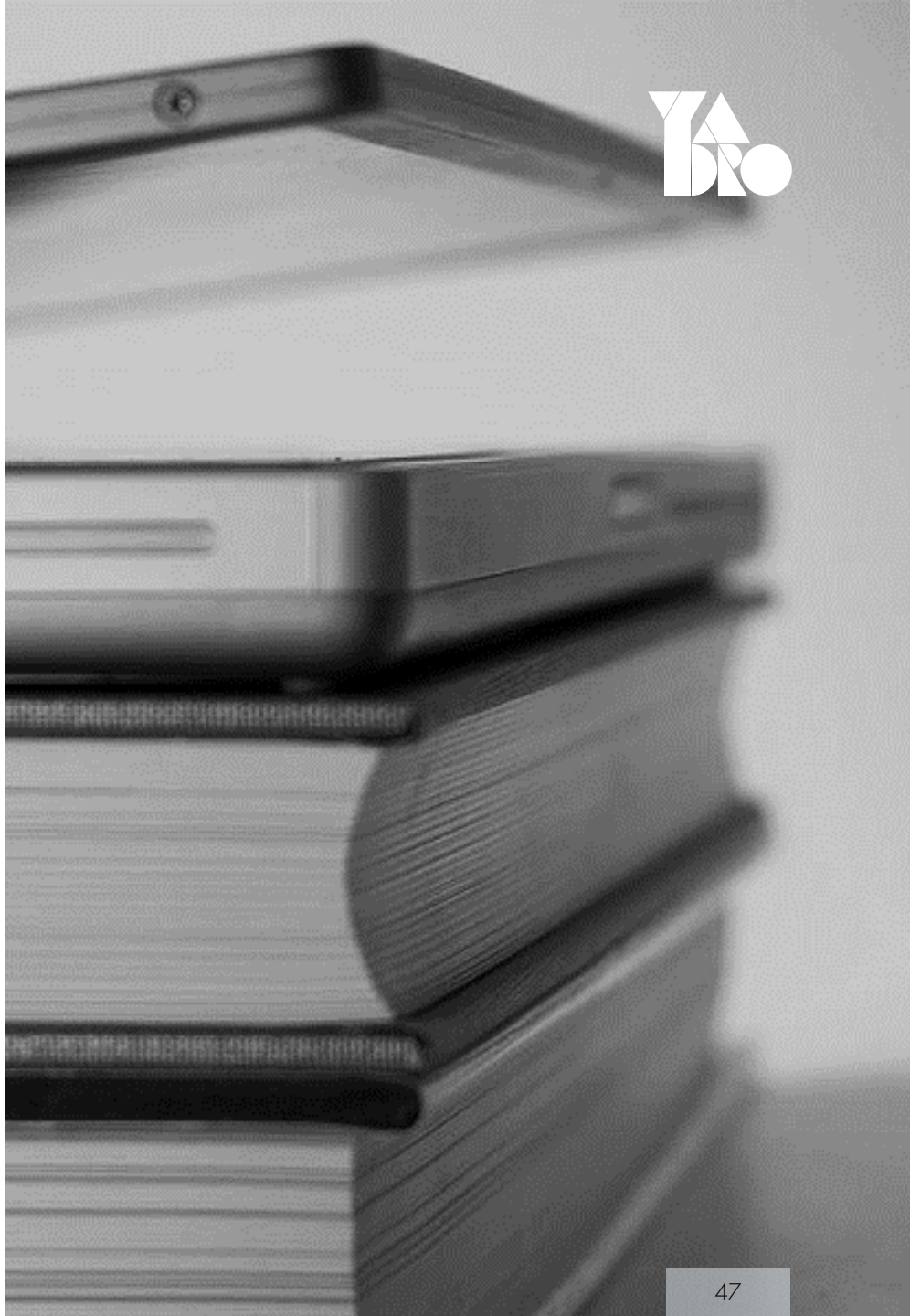
UVM Testbench Architecture



UVM Cookbooks & Tutorial

- IEEE Std 1800-2012
- Universal Verification Methodology (UVM) 1.2 User's Guide
- UVM Cookbook, Verification Academy
- UVM Primer, Ray Salemi
- Practical UVM. Step by Step Examples, Srivatsa Vasudevan

- www.chipverify.com/tutorials/uvm
- verificationguide.com/uvm/uvm-tutorial/
- www.youtube.com/@DoulosTraining



Active/Passive Component Tips

- Do not connect scoreboards to active components
- Perform functional checks in passive components
- Promote warnings only from passive components
- Do not control end-of test from passive components
- Update configuration only from passive components
- Allow disable checks

- Drivers post items to scoreboard
- Coverage defined in active components
- Important messages from drivers
- Passive components control end-of-test schedule
- Configuration updates from active components
- Drivers do functional timeout checks
- Uncontrollable end-of-test scoreboard checks

Рекомендации по переиспользованию компонент

- Пассивный и активный режим для всех компонентов
- Унификация окружений: структура, сообщения, конфигурация и т.д.
- Общая кодовая база: базовые классы, оболочки, библиотеки утилит и т. д.
- Рекомендации по стилю кодирования
- Автогенерация окружения на основе единого шаблона
- Собственная библиотека UVC (Universal Verification Component)
- НЕТ чекеров в активных компонентах: драйверах, секвенсорах и т.п.
- Scoreboards подключаются только к пассивным компонентам
- Функциональное покрытие собирается в пассивных компонентах

Цена универсальности

- **Дополнительные усилия по проверке компонентов**
  - архитектурные концепции представлены методологией
  - ярлыки и быстрые исправления экономят время в первоначальном проекте
  - усилия по повторному использованию выгоды как для разработчика, так и для пользователя компонента

- **Дополнительные усилия в верификационном окружении**
  - Использование **packag**, конфигураций, интерфейсов стоят усилий
  - усилия здесь только принесут пользу повторному пользователю

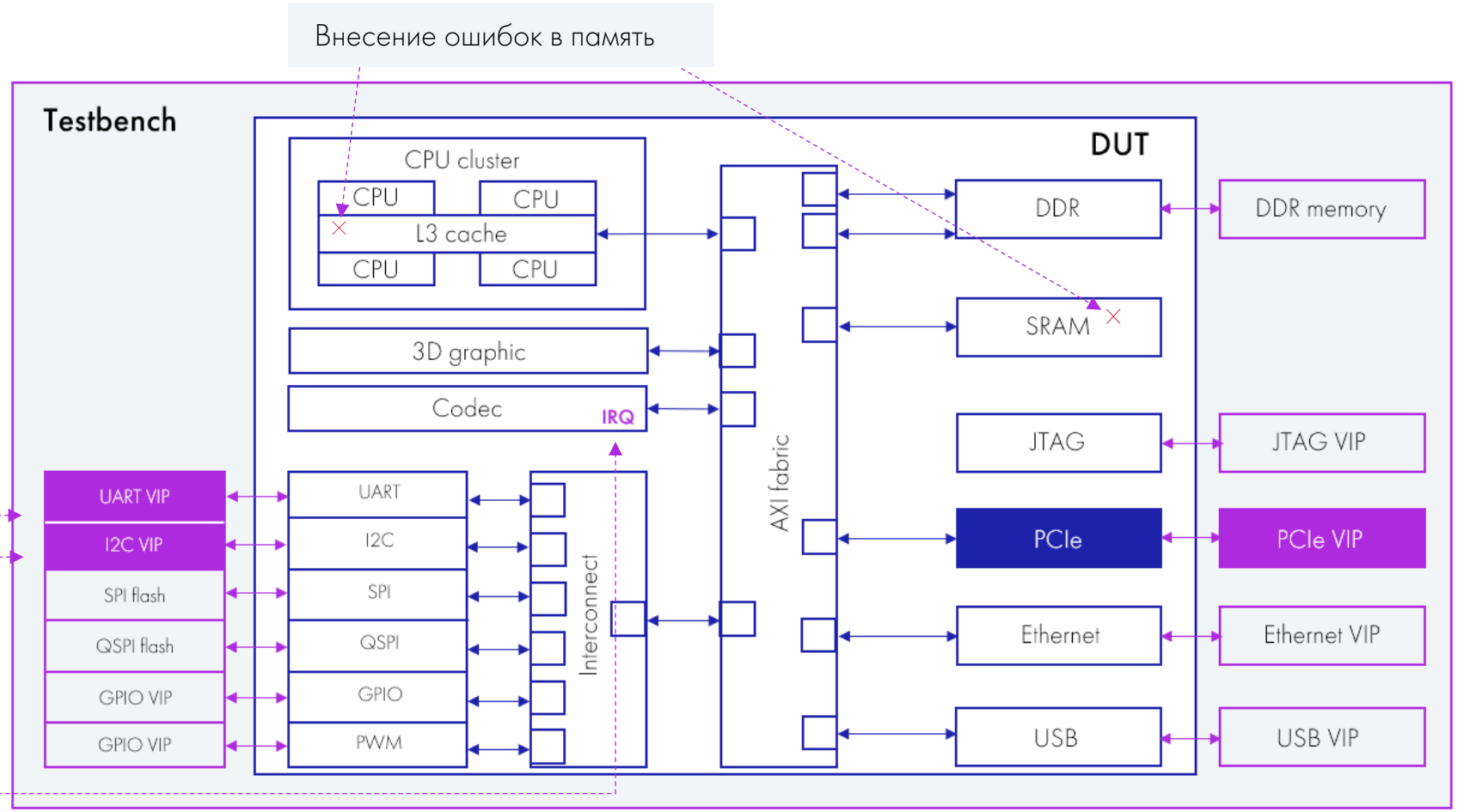
- **Архитектурные исправления для повторного использования должны быть внесены в исходный код.**

Управление тестовым окружением из SW-тестов

- Тестовые сценарии – программы на языке C
- Исполняются на реальной модели процессора из памяти
- Управляют тестовым окружением

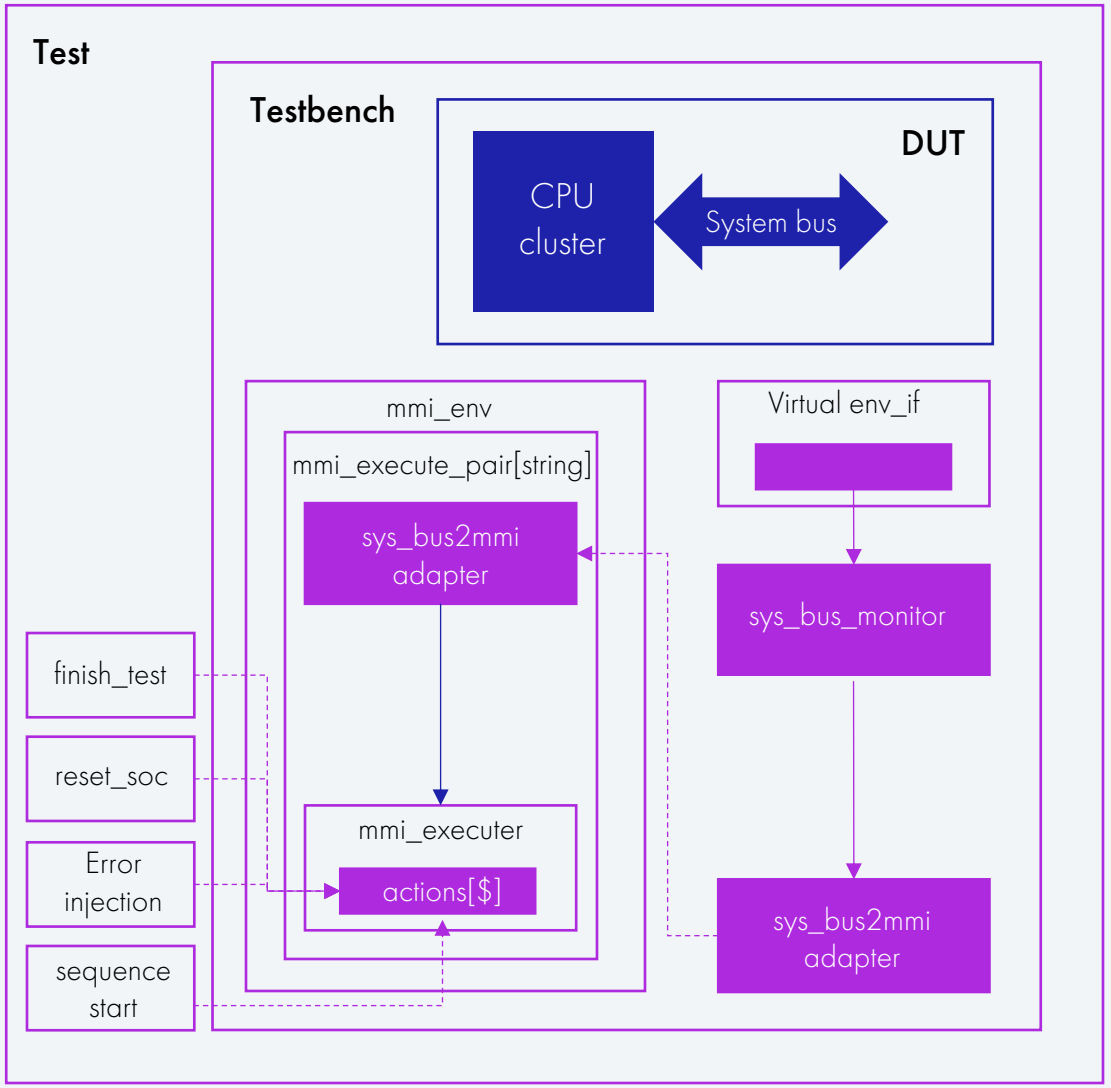
Подача внешних воздействий

Задание значений внутренним сигналам



MMI environment

- Выделить в памяти специальное адресное пространство, через доступ к которому программа будет передавать команды для тестового окружения
- Тестовое окружение будет захватывать такие обращения на системной шине и передавать в специальный обработчик.
  - Монитор **sys_bus_monitor** собирает транзакции с системной шины и отправляет их в **sys_bus2mmi_adapter.**
  - Данный класс проверяет, принадлежит ли транзакция MMI пространству и, если да, то преобразует транзакцию шины в MMI транзакцию (**MMI Item**).
  - mmi_executer, получив **MMI Item**, пробегает по хранилищу исполнителей (**actions[$]**), в котором лежат объекты с реализацией команды, и каждому объекту передаёт принятую MMI-транзакцию
  - Объект исполняет команду, если она ему предназначена
- Описание регистров данного интерфейса и структуры команд называется **Memory-mapped interface (MMI).**



MMI окружение: программная модель

Offset	Name	Description	Parameter
0x00	STOP	Остановка моделирования с кодом выхода	Код выхода (0 - без ошибки)
0x08	CHAR	Печать символа	ASCII код символа
0x10	STRING	Печать строки	Адрес строки в памяти
0x18	HEX	Печать числа в шестнадцатеричном формате	
0x20	CMD	Регистр команд для TB	Код команды
0x28	OPT	Регистр опций для TB	Код опции для команды
0x30	reserved	Зарезервировано	
0x38	reserved	Зарезервировано	

MMI окружение: примеры команд

Код	Описание	Функция	Описание	immOPT	OPT[0:3]
0x01	Работа с TB DDR	0x05	Backdoor MR Write	[4] Channel [3:2] Chip	OPT[0][15:8] -- Code OPT[0][7:0] -- MR address
0x05	Управление линиями прерываний	0x00	IRQ_FORCE: перевести линию в необх. состояние	[7:0] - индекс прерывания [8] - логическое состояние	
		0x01	IRQ_RELEASE: вернуть линию в исходное состояние		
0x08	Работа с блоком USB	0x00	Запуск на VIP USB секвенции для режима VIP_DEV_20	[1:0] USB VIP selection	
		0x01	Запуск на VIP USB секвенции для режима VIP_DEV_SS		
		0x02	Запуск на VIP USB секвенции для режима VIP_HOST_20		
		0x03	Запуск на VIP USB секвенции для режима VIP_HOST_SS		
		0x04	Запуск на VIP USB секвенции для проверки отрицательных веток для режима VIP_DEV_SS		
		0x05	Запуск на VIP USB секвенции для теста PING		
		0x06	Сброс VIP USB после снятия питания (VIP -- device)		
		0x07	Переключить линии для режима SS	[0] Line selection	
		0x08	Запуск на VIP USB секвенции для режима VIP_DEV_20_LOW_POWER		
		0x09	Запуск на VIP USB секвенции для режима VIP_DEV_SS_LOW_POWER		
0x0A	Управление сбросом системы	0x00	PWRGD - установить активный уровень сигнала через 1000 пс после приёма команды		
		0x01	RSTN - установить активный уровень сигнала через 1000 пс после приёма команды		
0x0B	Работа с памятью	0x00	Memory Set	[1:0] Code [9:2] Seed	opt_buf[2] - addr1 opt_buf[1] - addr2 opt_buf[0] - range
		0x01	Memory Compare		
		0x02	Memory Copy		
0x0F	Работа с блоком PMIC	0x00	Значение напряжения на порту VDD_SCU	[15:8] bits – voltage integer; [7:0] bits – voltage fraction.	
		0x01	Значение напряжения на порту VDD		
		0x02	Значение напряжения на порту VDD_CPU_TOP		
		0x03	Значение напряжения на порту VDD_MM		
		0x04	Значение напряжения на порту VDD_VIDEO		
		0x05	Выключение всех напряжений		

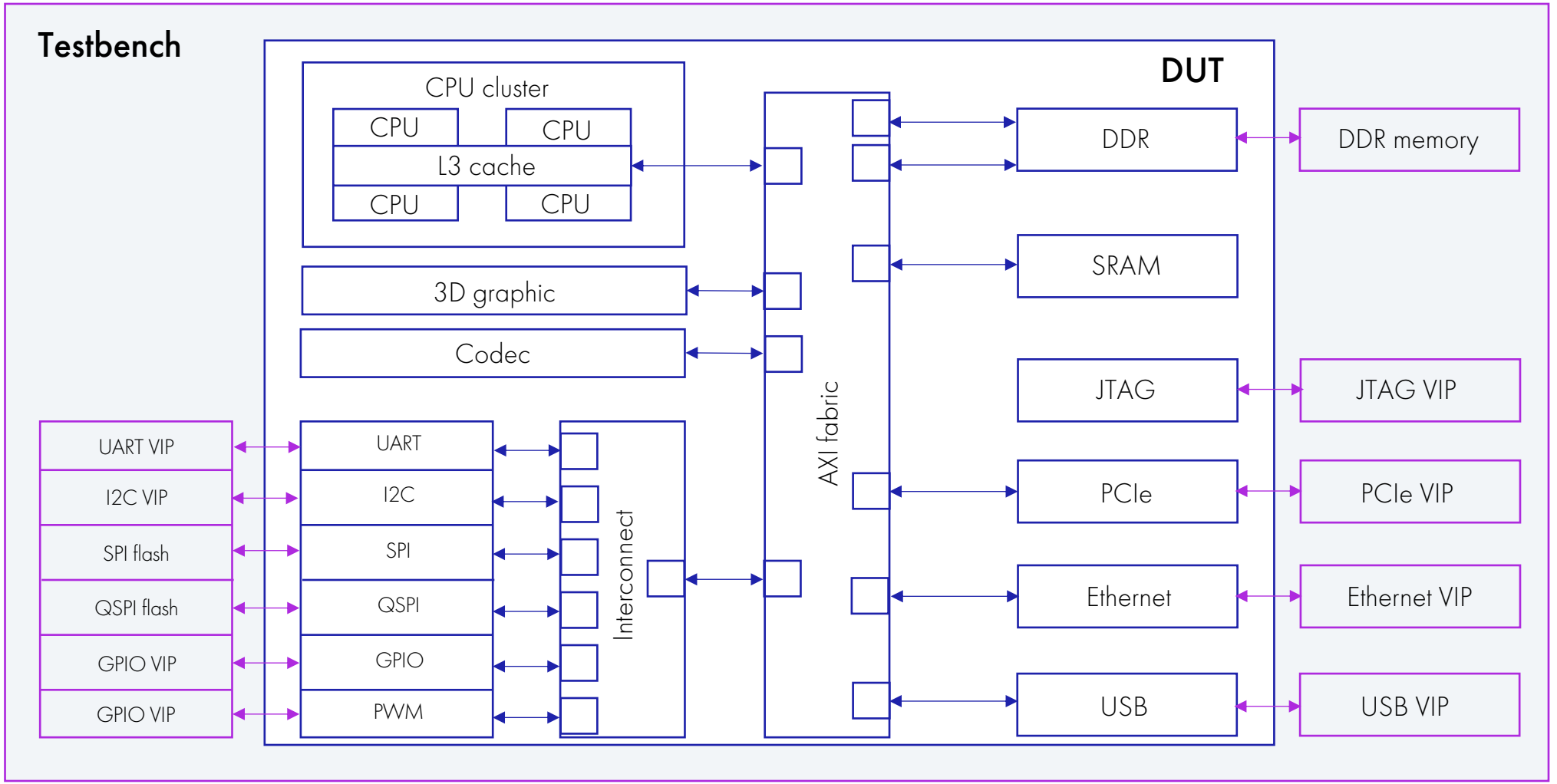
DPI: Универсальный SystemVerilog API

- Модификатор import «DPI-C» означает, что метод вызывается на стороне «SV», а его реализация находится на стороне «C»
- Модификатор export «DPI-C» означает, что метод вызывается на стороне «C», а его реализация находится на стороне «SV»
- Методы делятся на два типа: functions и tasks. Отличия в том, что функция выполняется строго мгновенно (в одном time slot), в то время как исполнение task'a может занять какое то время (несколько time slot)

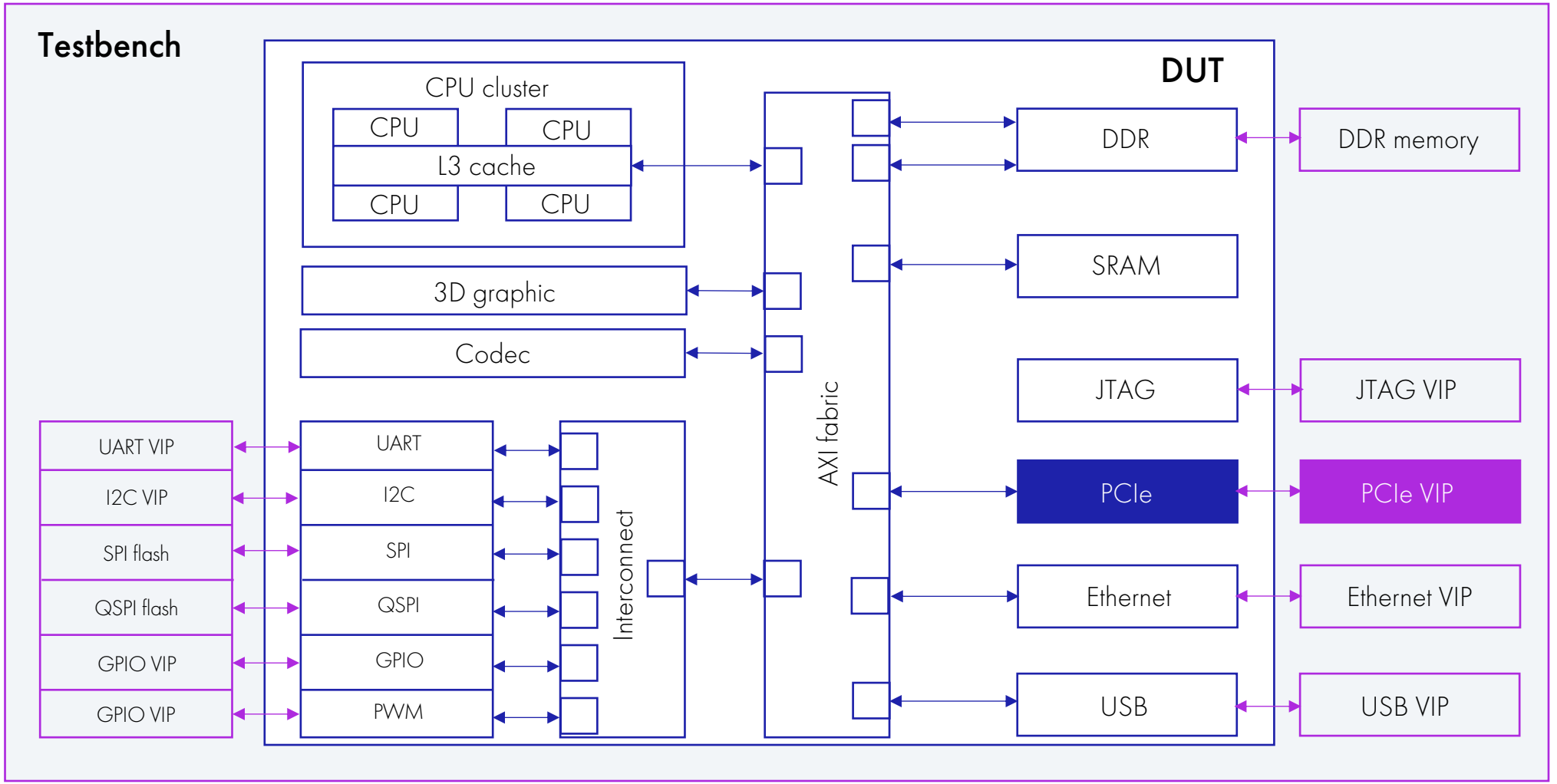
	Global functions
1	// Позволяет «SV» узнать у «C» текущую версию реализацию DPI функции `import «DPI-C» context task version (output int ver);`
2	// Старт работы одноименной функции «C», аналог main функции в классическом «C» `import «DPI-C» context task main_sim (input int argc, input string argv);`
3	// Запрос со стороны «C» на передачу набора параметров (текущий набор параметров описан ниже) `export «DPI-C» task request_setting_list();`

4	// Вызов функции вложен в реализацию метода request_setting_list() и передает на сторону «C» набор параметров
	// в виде открытого массива
	`import «DPI-C» context task put_settings_list (input longint setting_list[]);`

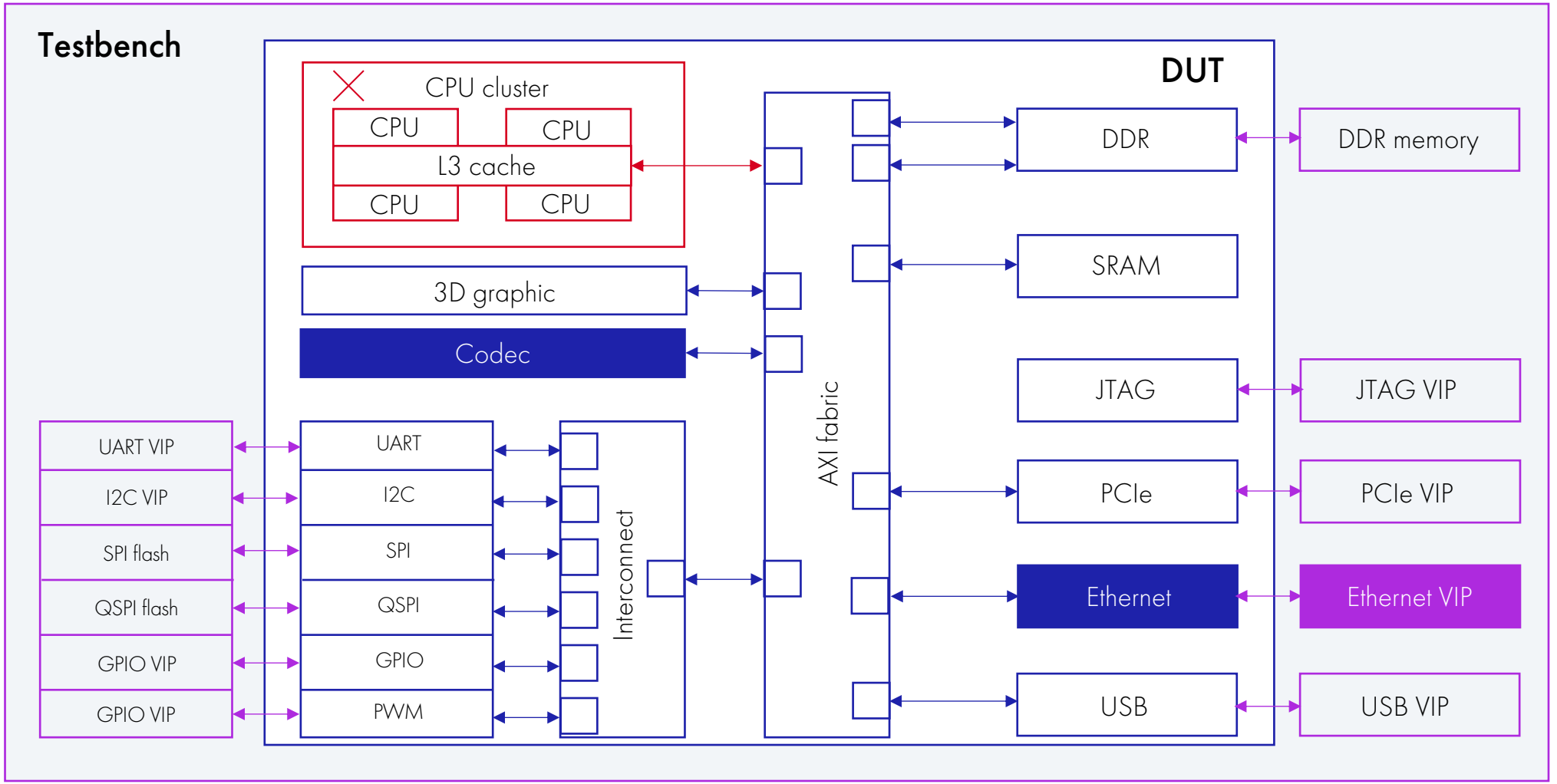
Применение DPI



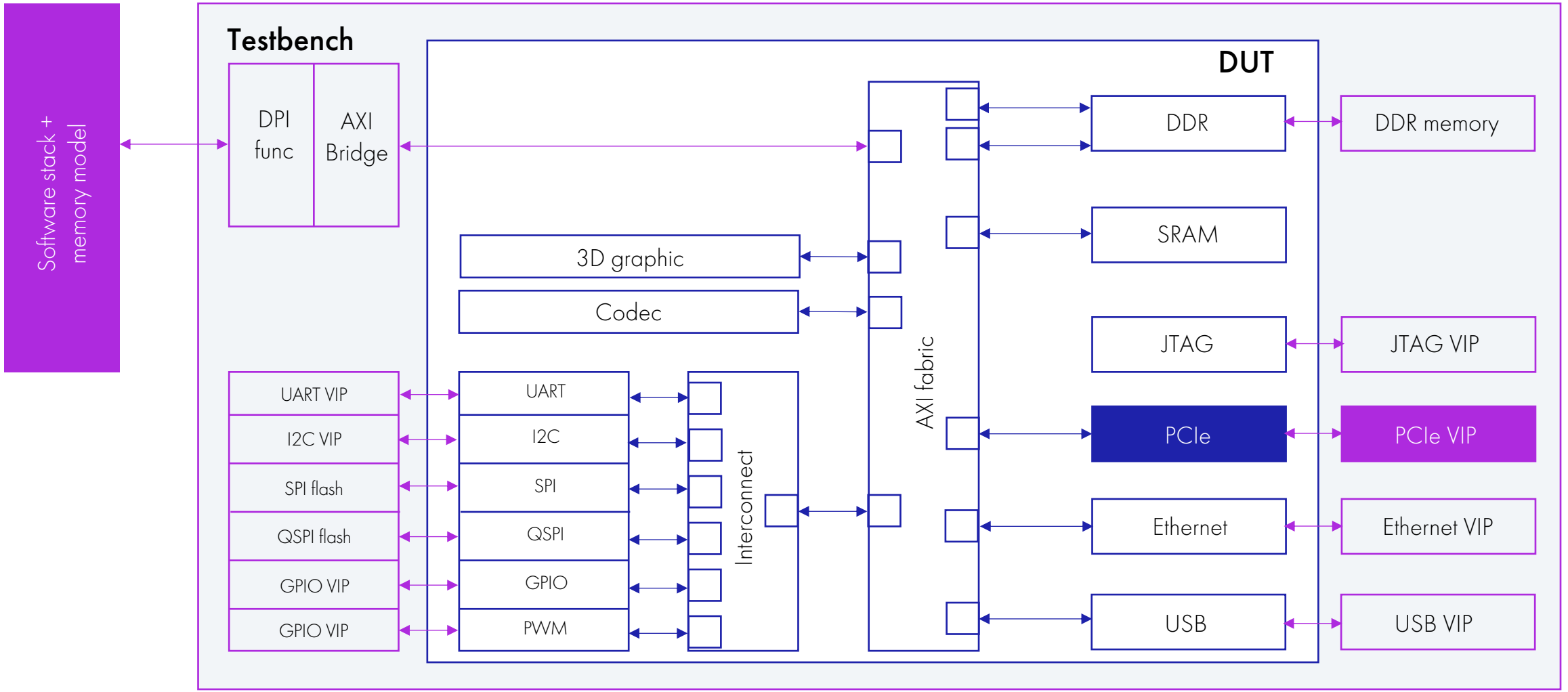
Применение DPI



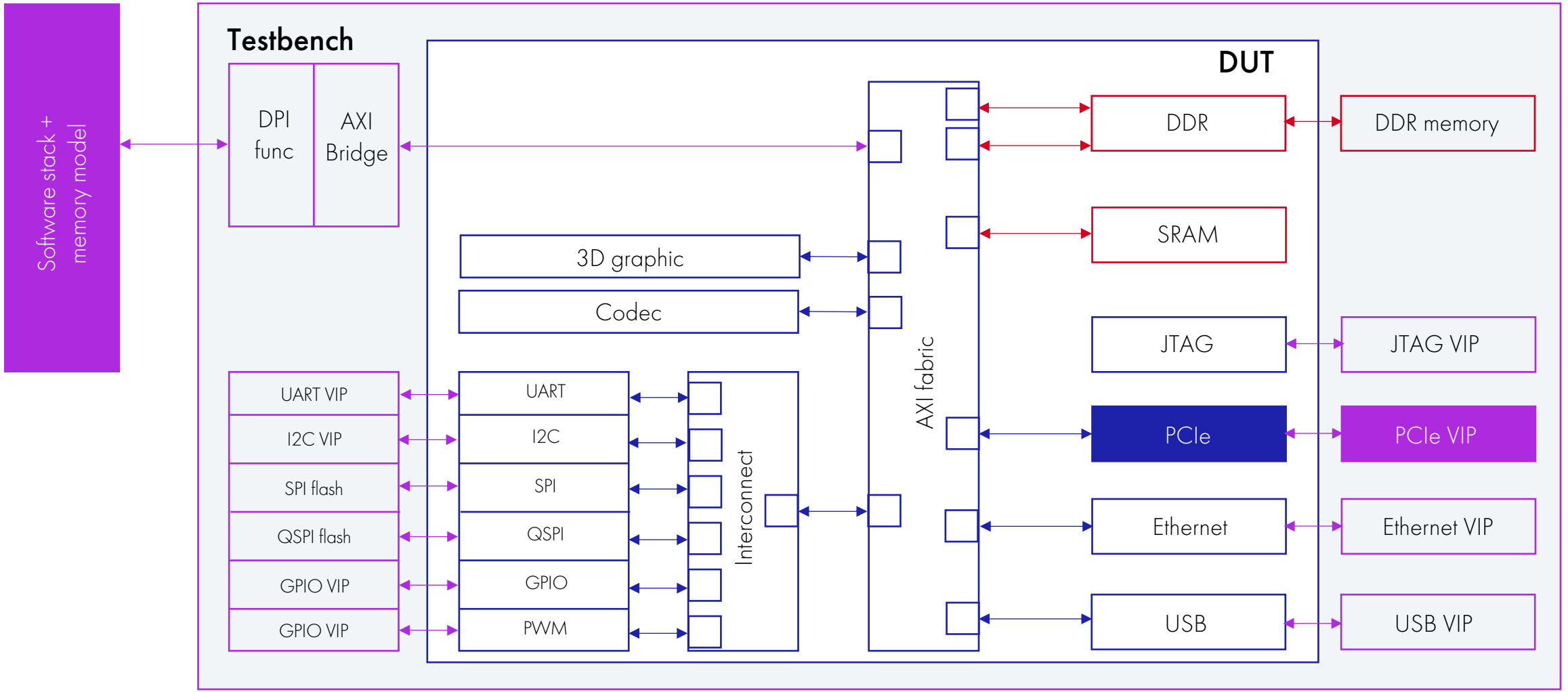
Применение DPI



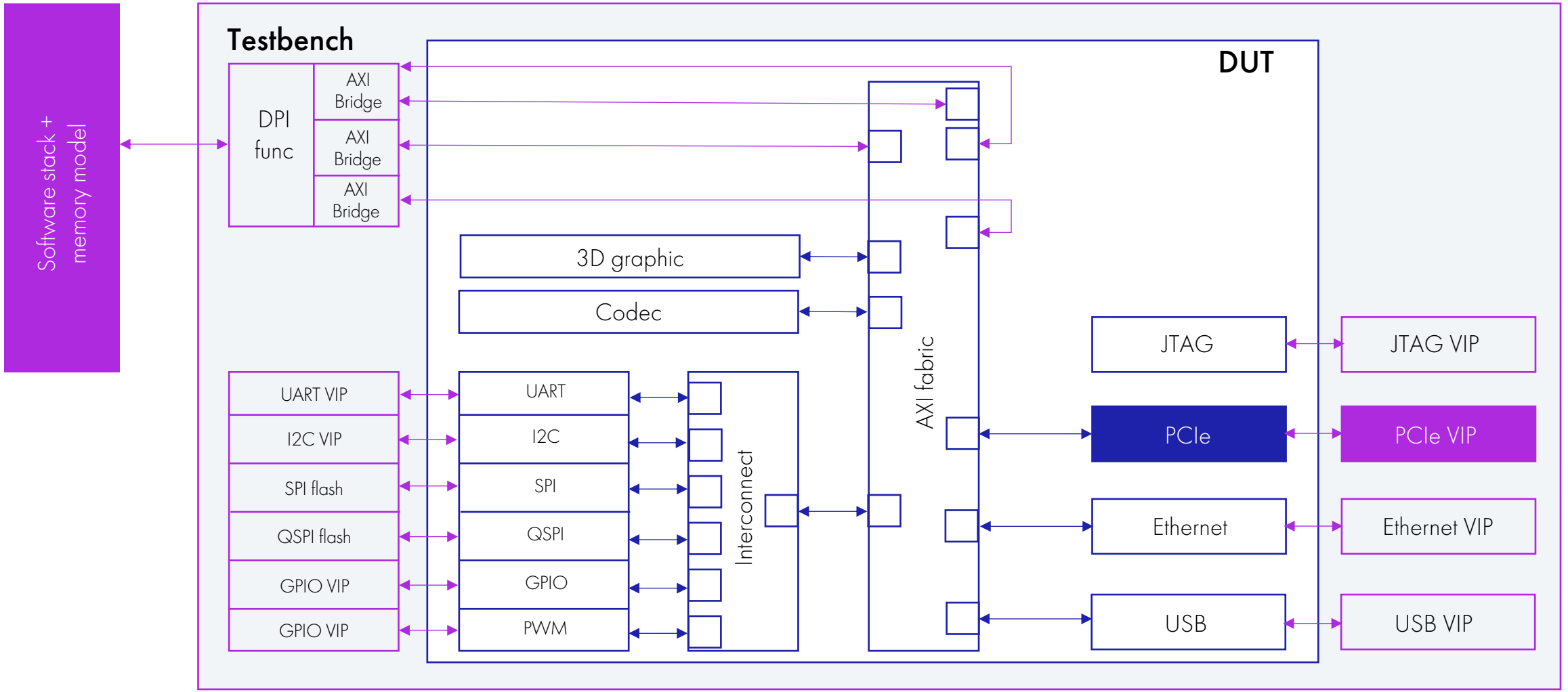
Применение DPI



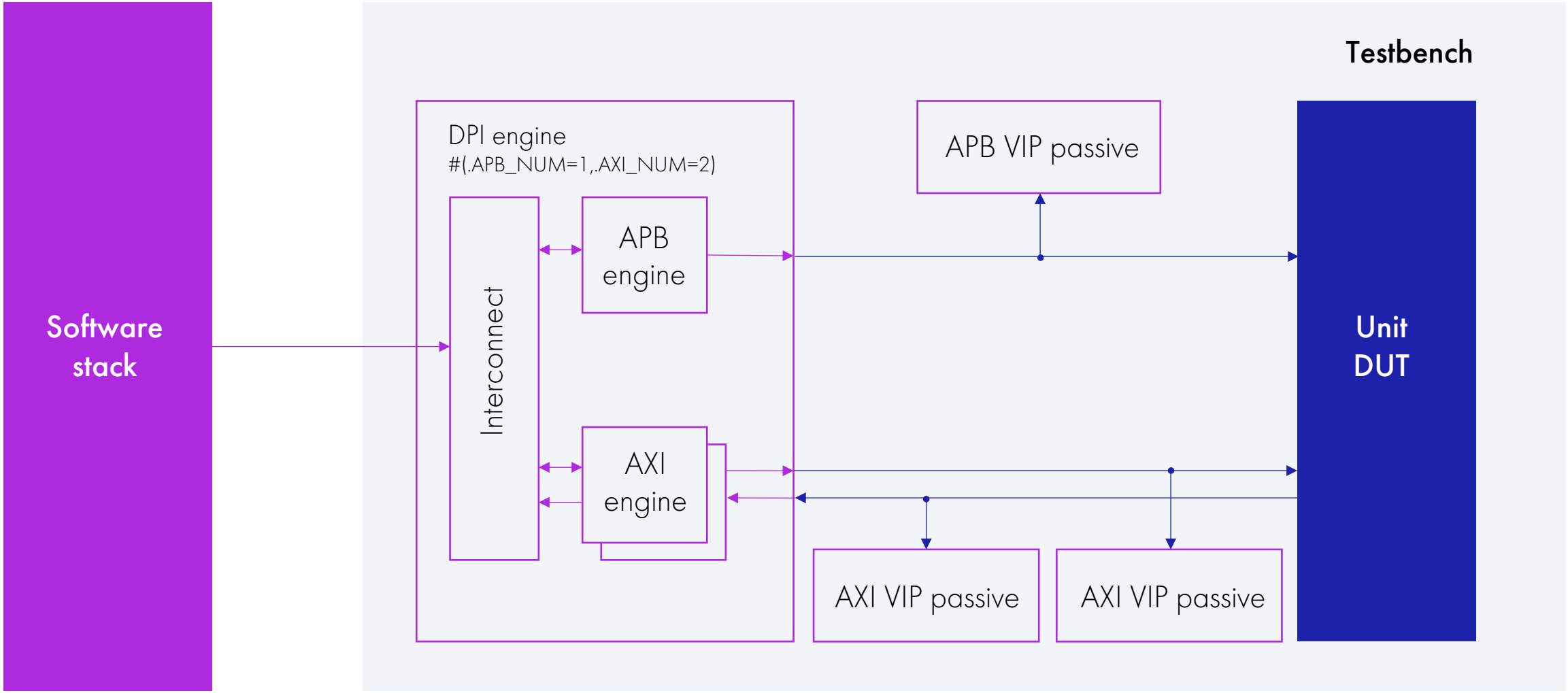
Применение DPI



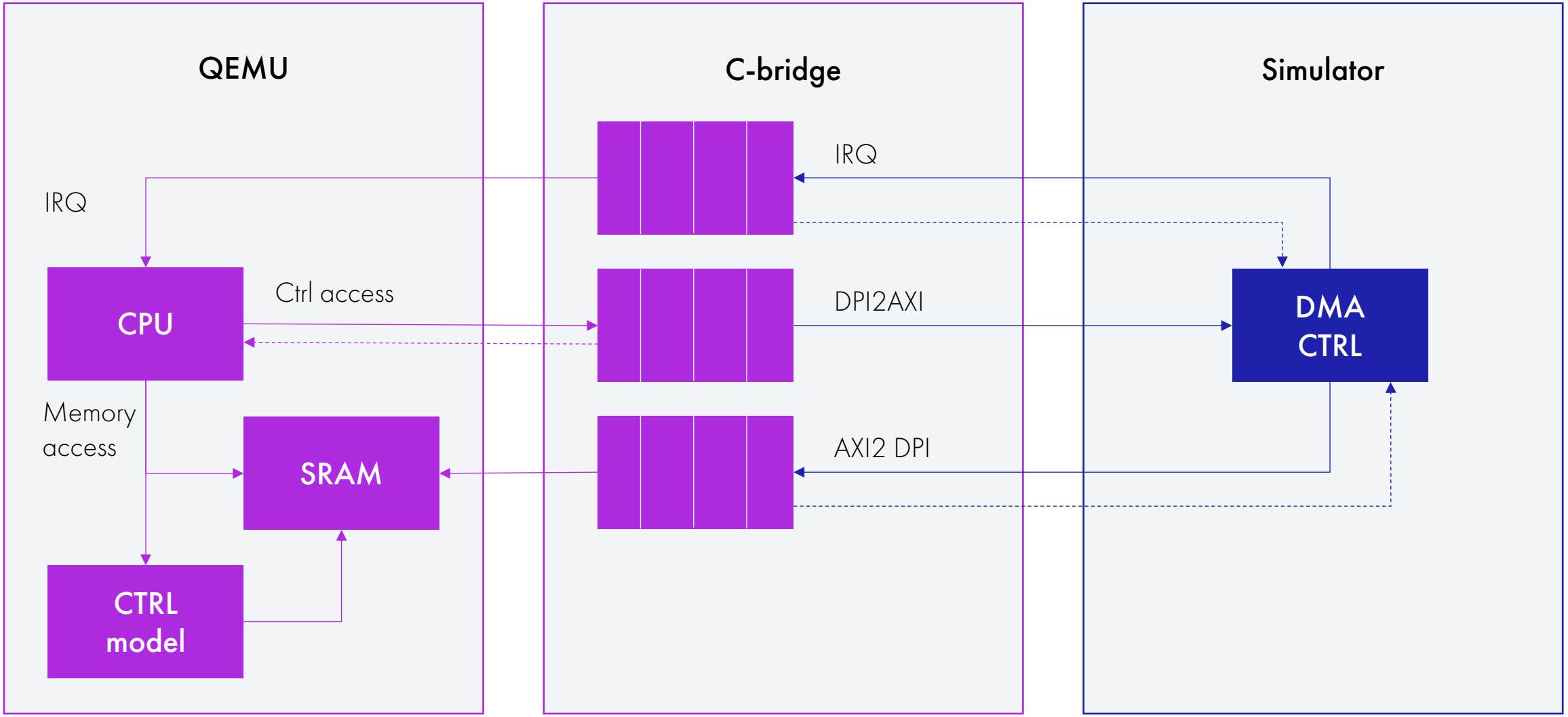
Применение DPI



Универсальный компонент DPI



Подключение QEMU



Немного статистики

Что такое верификация и зачем она нужна

Coverage driven verification

Декомпозиция задач

Шаблоны документов

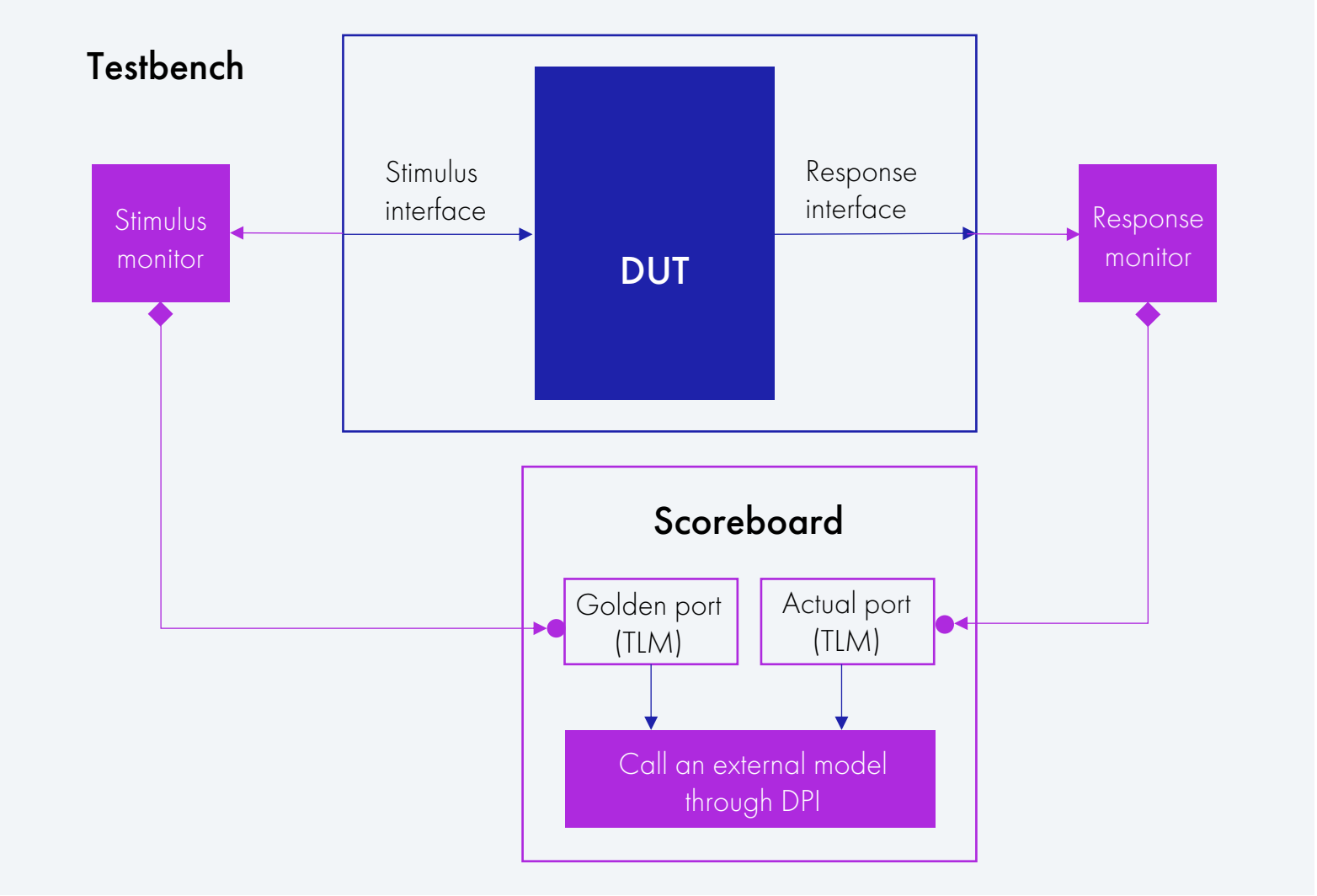
Планирование верификации

Разработка окружения для повторного использования

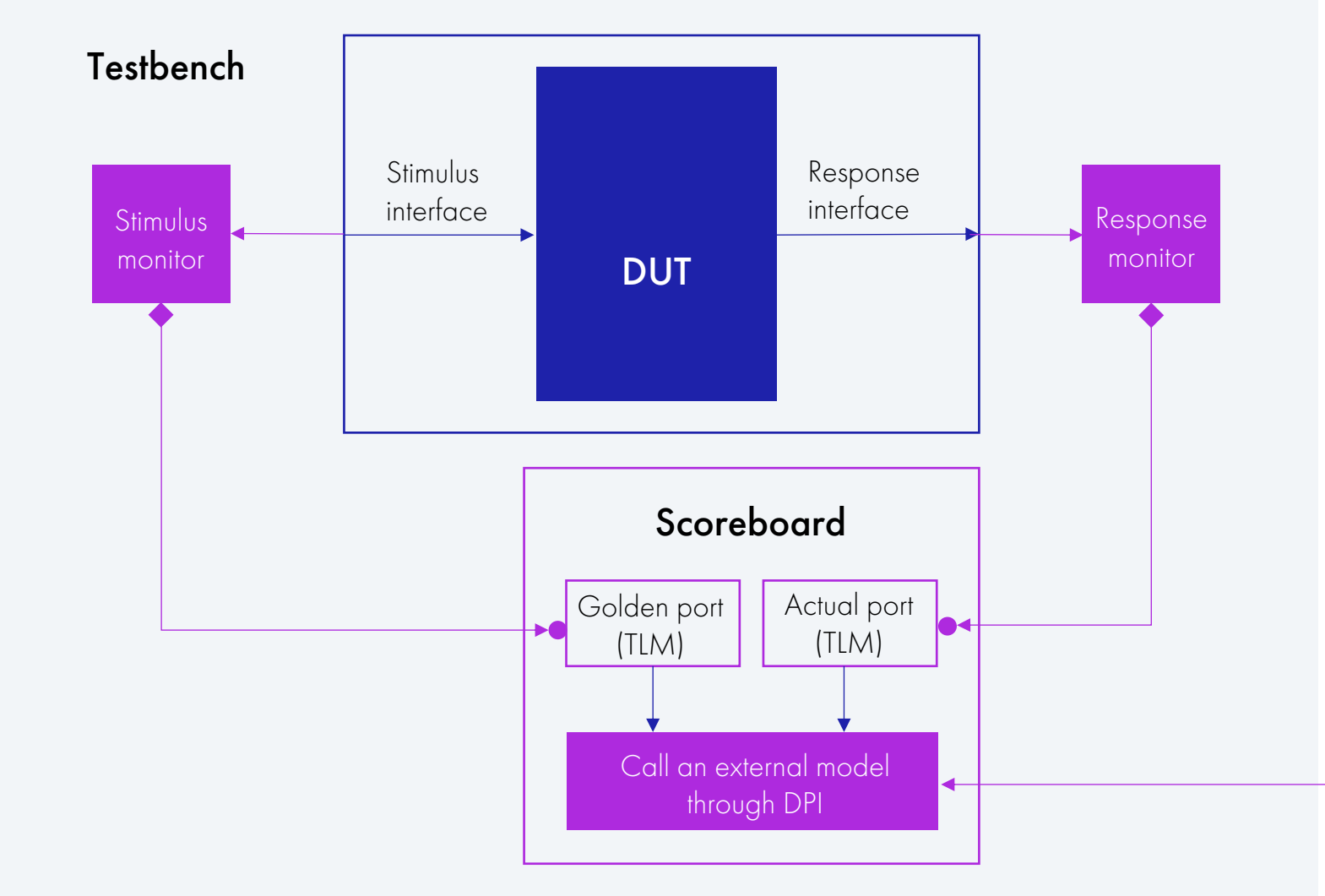
Checking techniques

Python: Cocotb and PyUVM

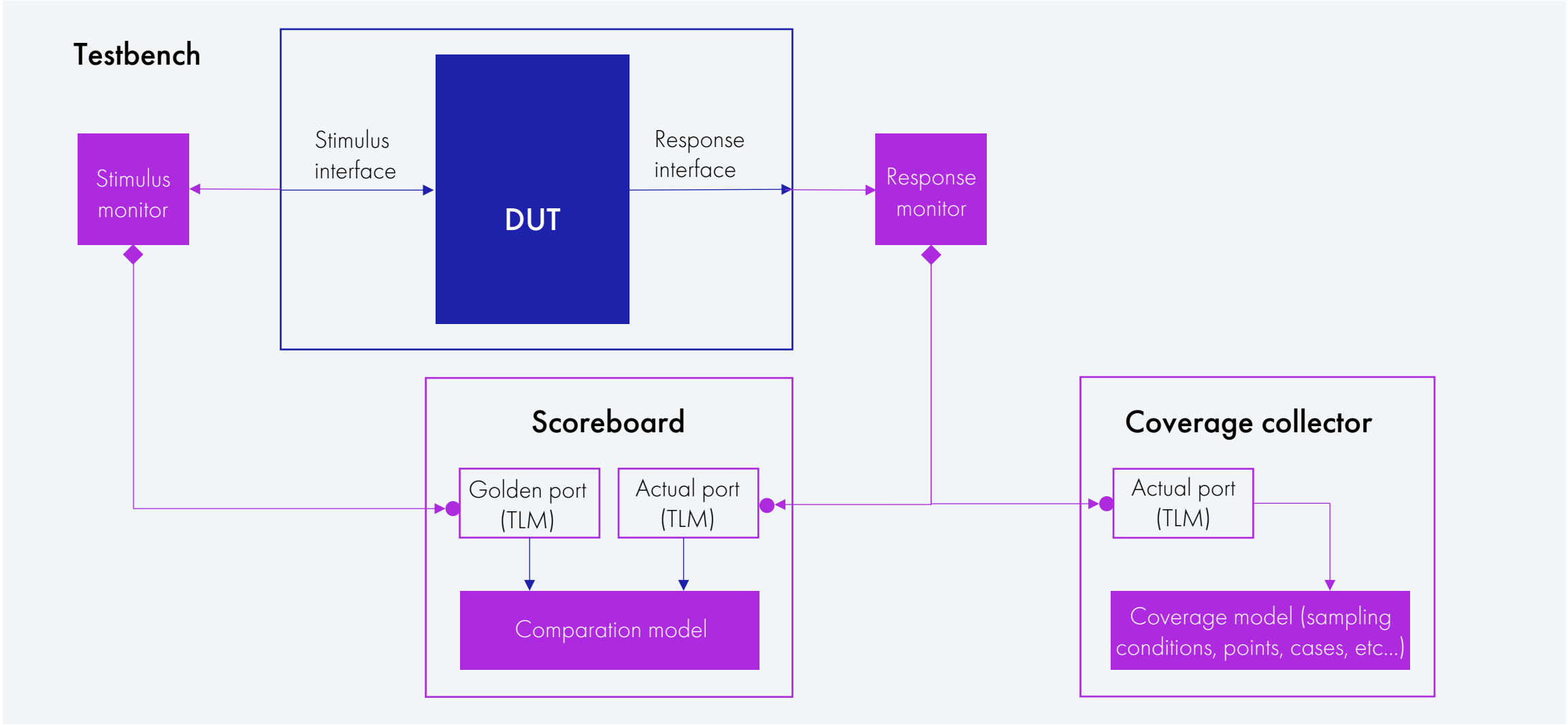
Подключение scoreboard



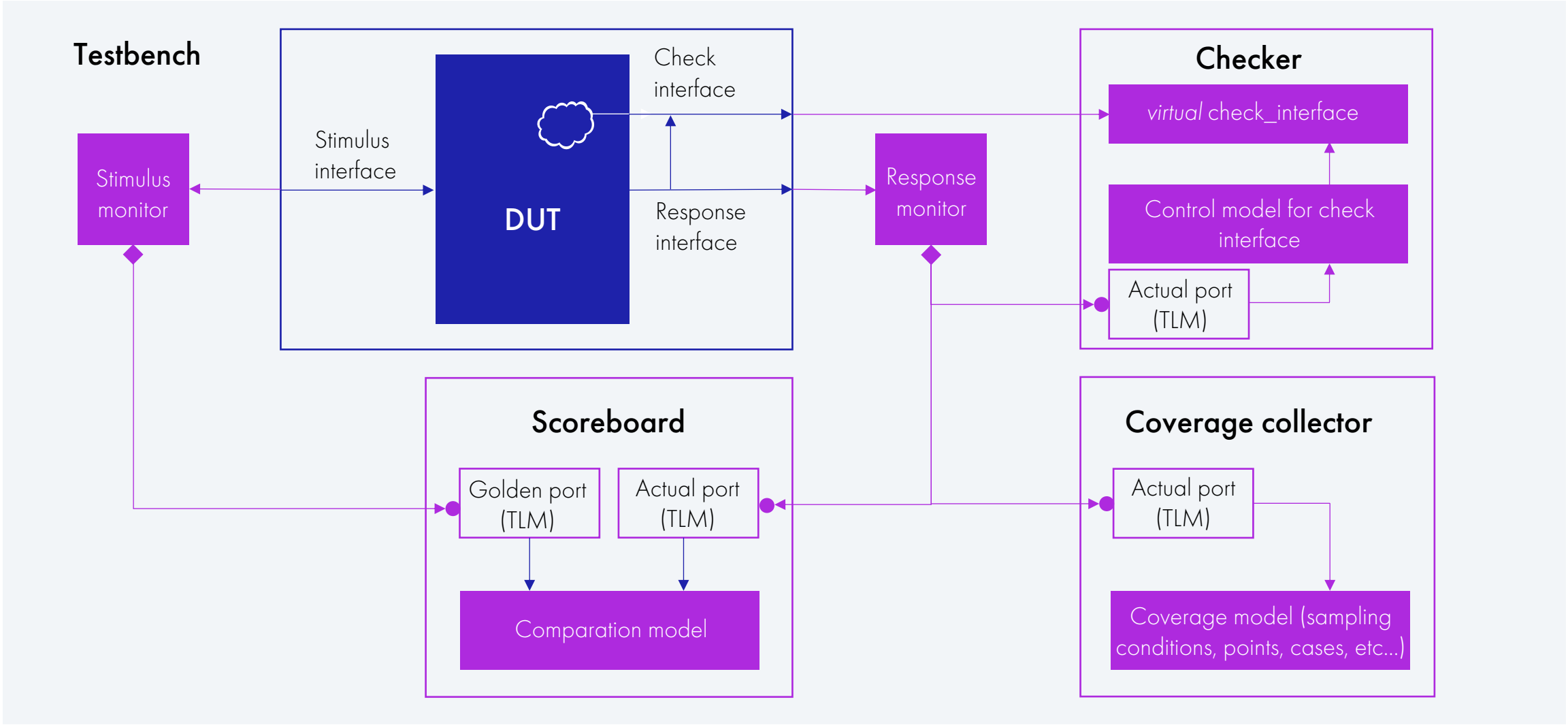
Подключение внешней модели через DPI



Coverage collector



Checker & SVA


Testbench
Stimulus monitor
Stimulus interface
DUT
Check interface
Response interface
Response monitor
Checker
virtual check_interface
Control model for check interface
Actual port (TLM)
Scoreboard
Golden port (TLM)
Actual port (TLM)
Comparation model
Coverage collector
Actual port (TLM)
Coverage model (sampling conditions, points, cases, etc...)

Немного статистики

Что такое верификация и зачем она нужна

Coverage driven verification

Декомпозиция задач

Шаблоны документов

Планирование верификации

Разработка окружения для повторного использования

Checking techniques

Python: Cocotb and PyUVM

Использование Python в верификации

Основная проблема отрасли - нехватка инженеров в области проектирования и верификации цифрового дизайна

Причины:

- Требуется знание языков описания аппаратуры (SV/VHDL)
- Знание методологии UVM
- Проприетарные инструменты (САПР)

Попытка решения - Python-фреймворки:

- Легкий старт с языком Python, множество открытых библиотек
- Кроссплатформенность (Windows, Linux, MacOS)
- Open-source friendly – поддержка бесплатных симуляторов (Icarus Verilog, Verilator) и проприетарных (Synopsys, Cadence)
- Возможность развития Python-разработчика до SV-UVM верификатора

Python разработчик

Cocotb

PyUVM

SV-UVM Верификатор

Использование Python в верификации

Используемые Python-фреймворки:

- **cocotb** - an open source coroutine-based cosimulation testbench environment for verifying VHDL and SystemVerilog RTL using Python — github.com/cocotb/cocotb
- **pyvsc** – types, constraints and randomize object — github.com/fvutils/pyvsc
- **cocotb-coverage** - collecting functional coverage — github.com/mciepluc/cocotb-coverage
- **PyUVM** - the Universal Verification Methodology implemented in Python instead of SystemVerilog, this uses cocotb to interact with the simulator and schedule simulation events — github.com/pyuvm/pyuvm


БУДУЩЕЕ
В НАШИХ
РУКАХ