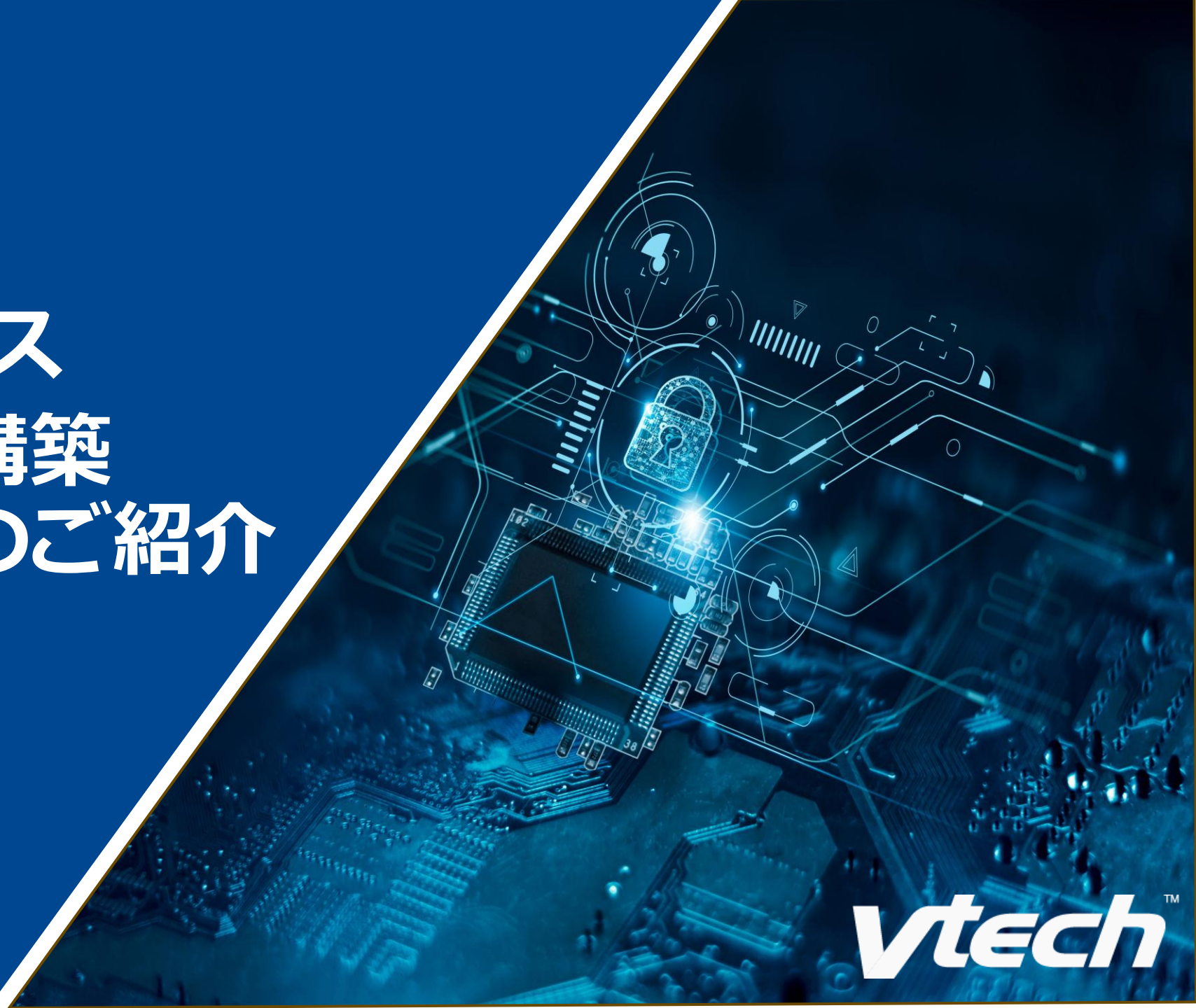
UVMベース
検証環境構築
弊社実績のご紹介



Vtech™

はじめに

- 弊社におけるUVM検証環境構築において得られた知見をベースに、UVMを活用することによる環境構築の効率化と品質改善の考え方・施策についてご紹介いたします。

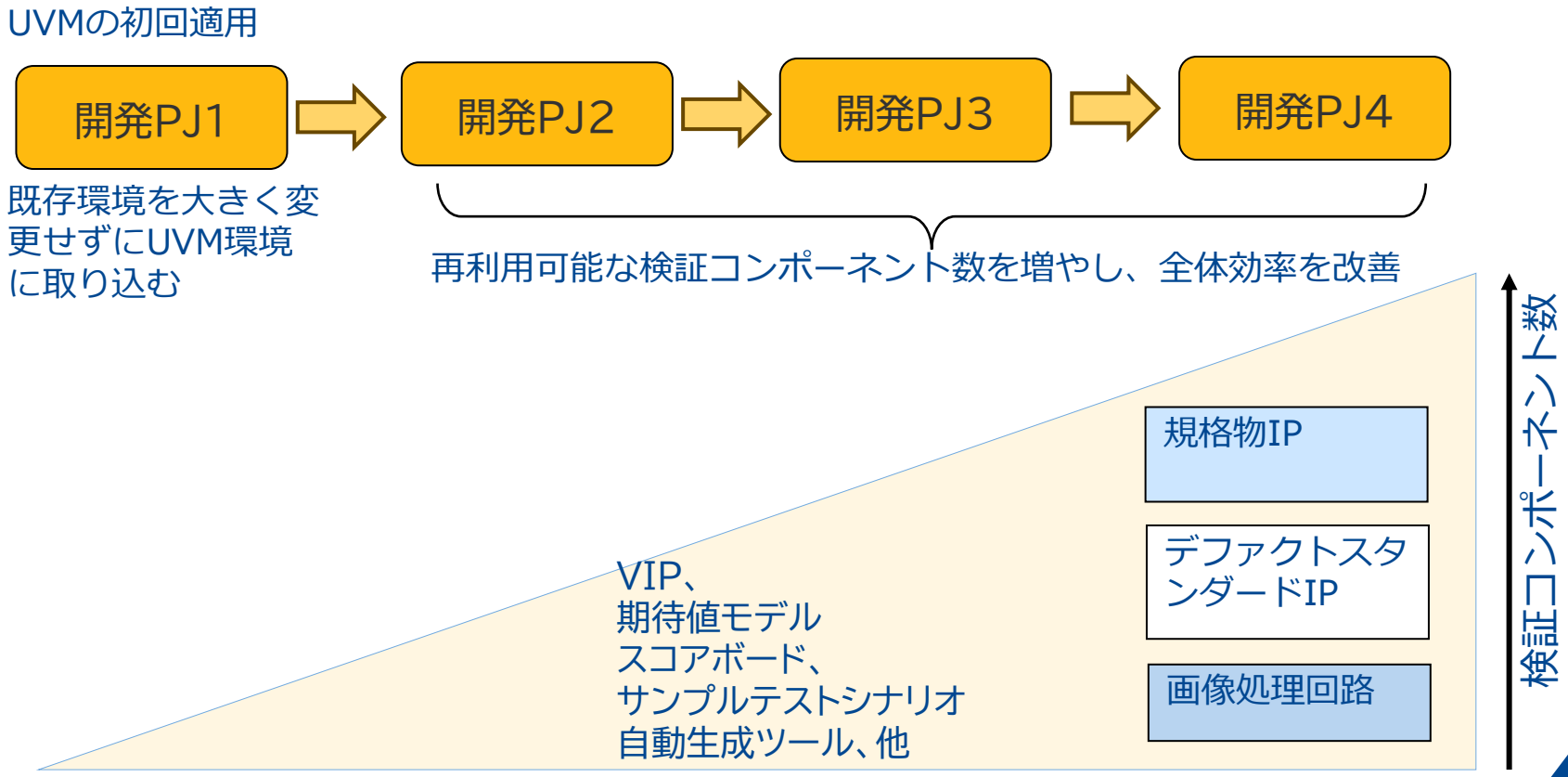
  御社でのUVM環境構築のご参考になれば幸いです。

UVMの特徴

- **UVMは検証の効率化と品質向上を目的としたフレームワーク**
  - SystemVerilogで導入されたオブジェクト指向を使って、検証環境においてソフトウェア開発と同様な開発効率化と品質向上を目指しています。
- **UVMにおける効率化と品質向上の肝は、検証コンポーネントの再利用**
  - 検証コンポーネント = 検証環境やテストシナリオを構成する部品
  - 再利用により人が書くコード量が減らせるのと、動作確認済みのコンポーネントの再利用で、効率化と品質向上を実現
  - SoC設計が IPベース設計(=IP再利用)になっているのと似ている
- **UVMでは、流用性を向上するため、検証コンポーネントの作り方に決まり事がある**
  - SoC設計におけるAXI, APBバスと同じような位置づけ

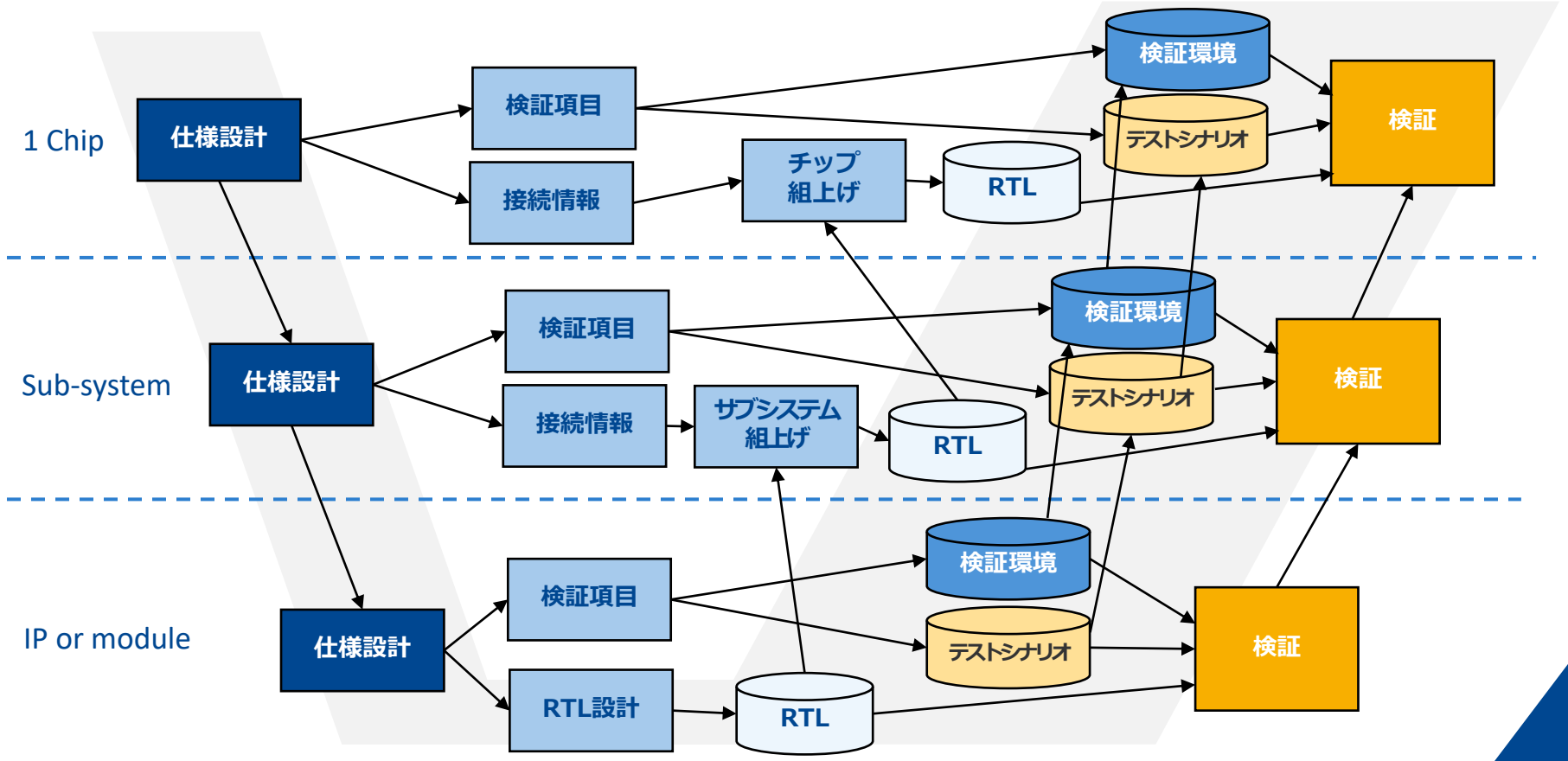
UVM導入による検証環境の品質・効率の改善

- 単発の検証環境構築から脱却し、開発PJにおいて前回の検証環境の流用と改善(検証コンポーネントの追加)を継続していくことが重要と考えております。
- 継続して検証コンポーネントの品揃えを増やし続けることで、検証環境の品質と開発効率を改善できます。
- **再利用可能な検証コンポーネントの数 = 品質と開発効率の改善率**



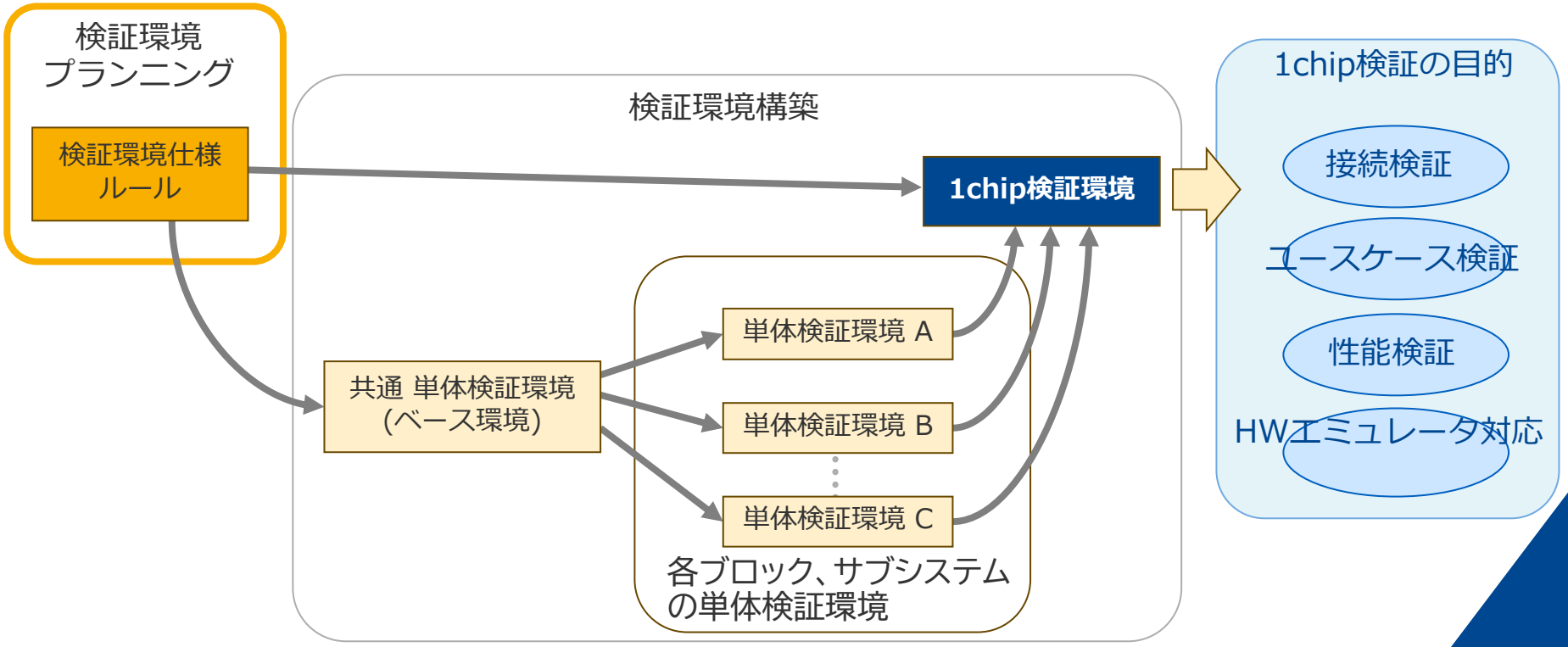
UVMと1 Chip階層設計

- UVMは、異なるSoC間だけではなく、1 Chip設計において下位階層→上位階層への検証環境の流用性の向上にも寄与します。
- 1 Chip RTL設計においては、IP/モジュール→サブシステム→1Chipと階層設計を行います。
- 同様に、検証環境、テストシナリオも下位階層のアウトプットを上位階層で利用することにより、トータルの検証効率を向上することができます。

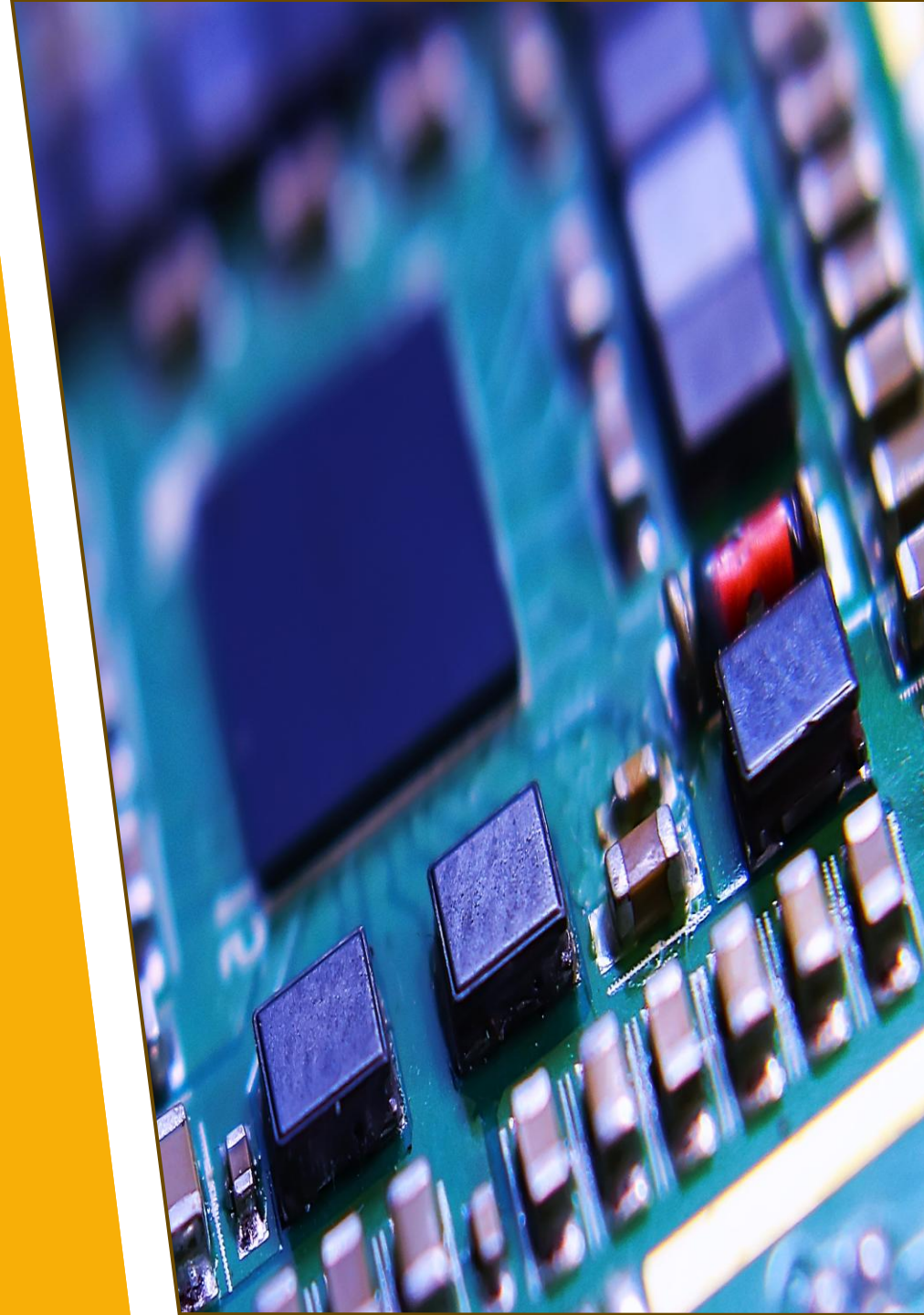


検証環境プランニングの重要性

- 検証環境プランニングにおいて、構築方針、検証環境仕様、実装ルールなどを策定することで、流用性を向上させます。
  - 1Chip開発Prj内の流用、後続Prjでの流用
  - 既存の各ブロック・サブシステム単体検証環境の流用
  - 1Chip検証の目的
- UVMをベースにすると検証コンポーネントの流用が実現可能になりますが、実際の実装では留意すべき点がいくつかあり、それらを検証環境仕様、環境構築のルールとしてまとめる必要があります。

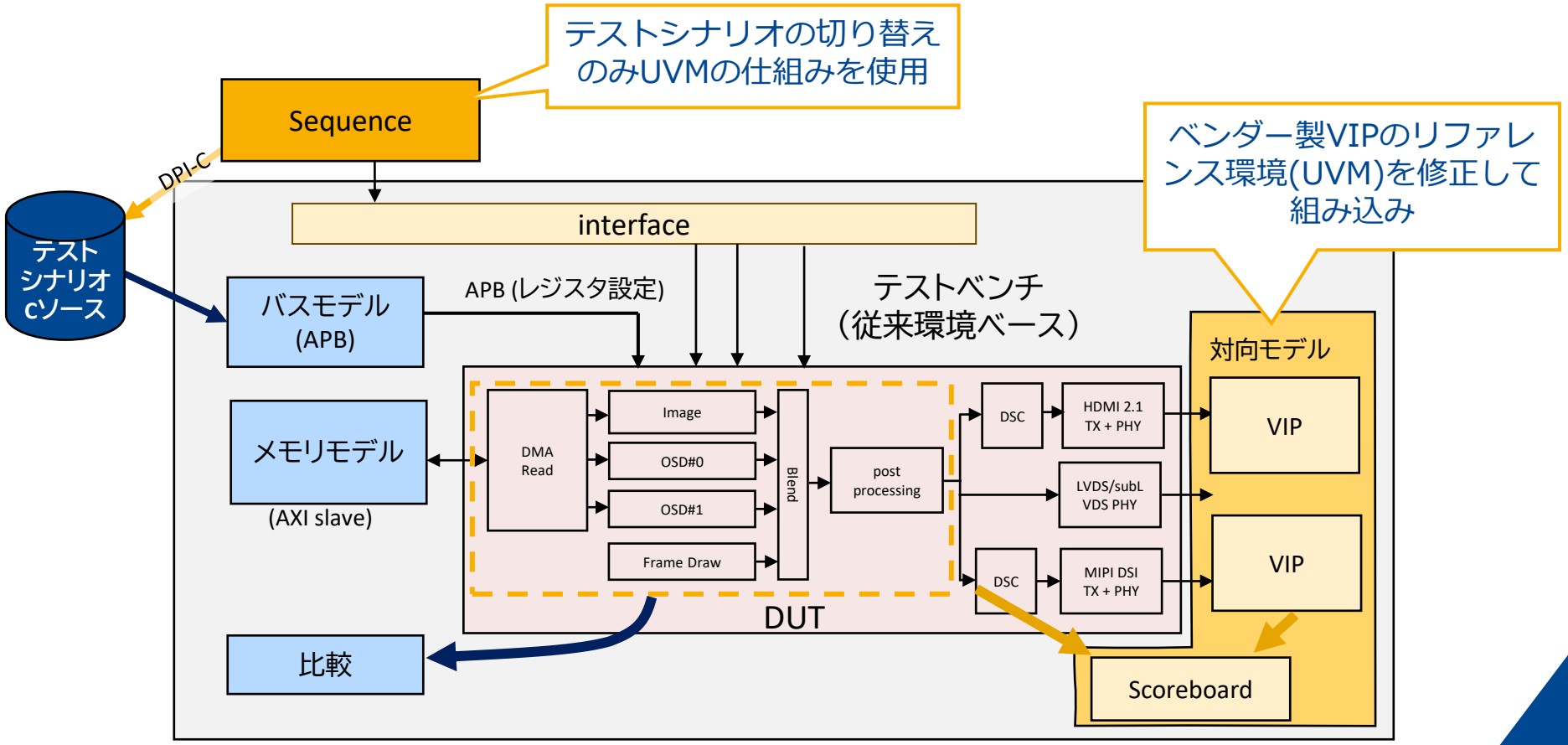


検証環境のUVM対応例
流用性向上の
施策例



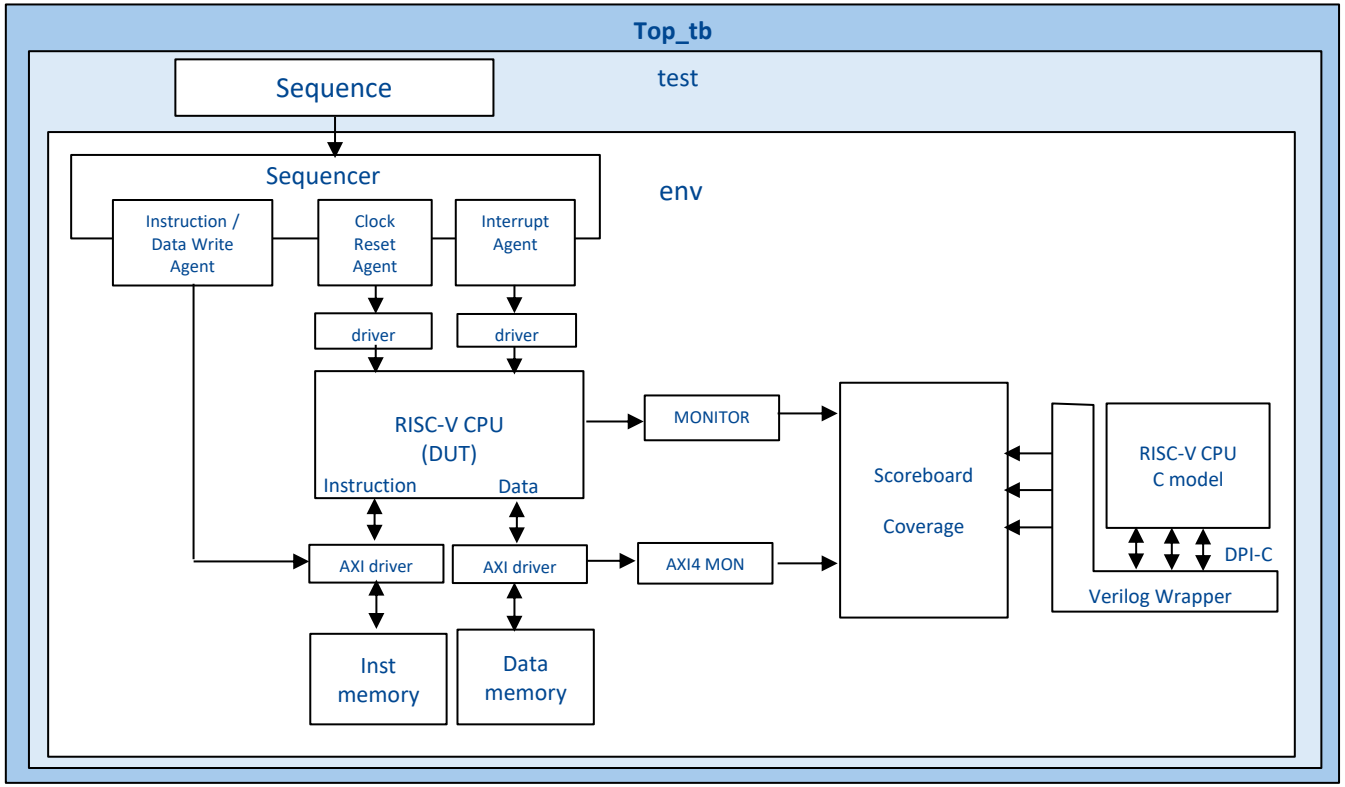
最小限のUVM対応の例

- 従来の検証環境をベースに最小限のUVM対応を実施した例です。
- 再利用性は全く考慮されていません。



RISC-VコアのUVM検証環境例

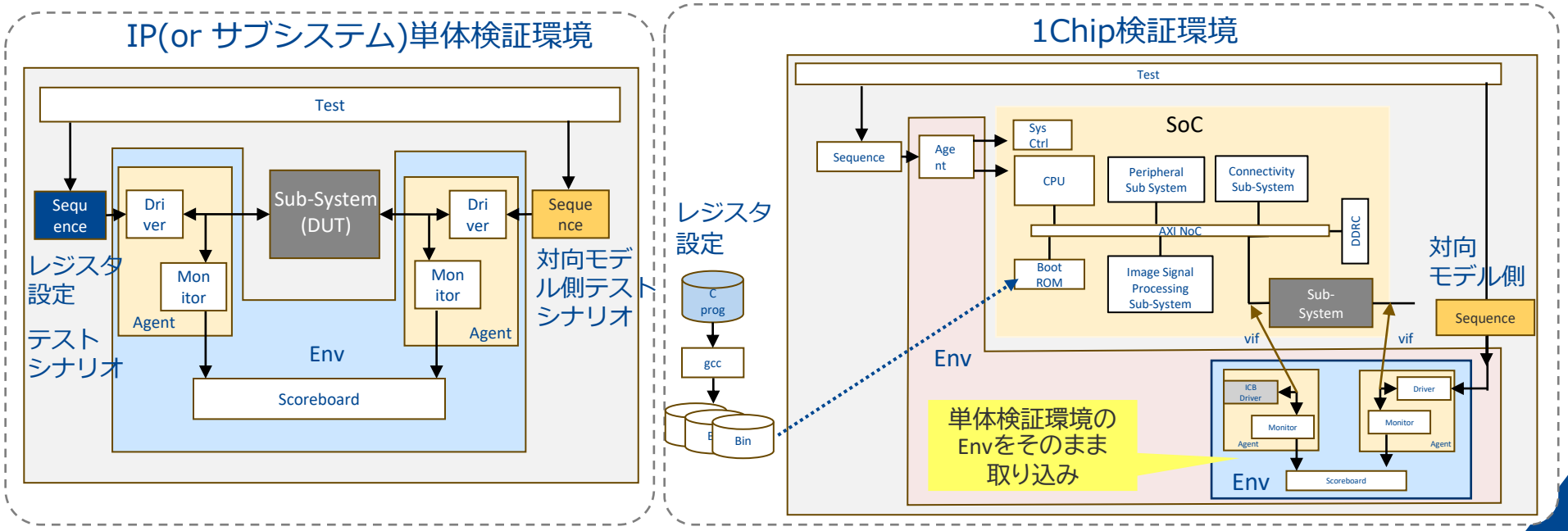
- 顧客設計のRISC-VコアをVtechで検証した時の検証環境の例です。
- これもRISC-Vコア単体の検証環境で、1chip検証環境への流用は考慮していないです。
- RV32I, M-Ext, C-Ext命令を対象に、命令単体の検証、連続する2つの命令の組み合わせ、割り込みとの組み
- 合わせの検証を実施しました。
- シミュレーション環境に組み込んでいるCモデルは、弊社にて開発したものです。



検証環境の上位階層での流用

UVMの特徴を生かし、IP単体やサブシステム検証環境を上位階層や1chipの検証環境へ変更なしで取り込めるようにします。

検証コンポーネント	単体検証環境のEnvを、そのまま1Chip検証環境のEnvに組み込む。 レジスタ設定バス側(入力側)のAgentは、単体検証ではACTIVEモード、1Chip検証ではPASSIVEモードで使用する。
DUTのレジスタ設定側のテストシナリオ	DUTのレジスタ設定等は、単体検証ではSequenceで記述、CPUを含む1chip検証ではCプログラムで記述し、CPUが実行する。
対向モデル側テストシナリオ	対向モデル側のテストシナリオ(Sequence)は、そのまま1chip検証環境に組み込む。



検証環境の流用性向上のための施策(1)

UVMは、本来、検証コンポーネントが流用できる仕組みになってはいますが、単体検証環境を上位階層の検証環境にそのまま組み込めるようにするためには、次の点について対応が必要となります。

流用部分と、変更が必要な部分の分離	レジスタ設定側のAgent, Interface (通常、APBやAXI等のレジスタ設定バス) と、 対向モデル側のAgent, Interfaceの分離
環境構成をパラメータで変更可能にする	Env以下の各種設定を上位階層(Testクラス)からuvm_config_db経由で設定できるようにする。 • Virtual interfaceのインスタンス名 • AgentのPASSIVE/ACTIVEモード切替 • その他の設定
レジスタ設定のSequenceをC言語で記述	サブシステム検証においては、DUTのレジスタ設定部分をC言語で記述し、1chip検証でCPUがそのまま実行できるようにする。 単体検証において、CプログラムをDPI-Cで実行するケース、単体検証環境にCPUを追加して対応するケースがある。
UVMとして一般的な作法	期待値とmonitorで取得したトランザクションをスコアボードで比較する仕組みの実装
	AgentのPASSIVEモードへの対応
	Testクラス、Sequenceクラスの明確な使い分け。Testクラスはテスト全体の設定や制御、SequenceはDriverの動作制御を行う。 • Testクラスは上位階層の検証には基本的には流用できないが、Sequenceクラスは上位階層の検証に流用する。 • 1chip環境でCPUが実行するプログラムはtestクラスから指定する。(Sequenceクラスでは指定しない)

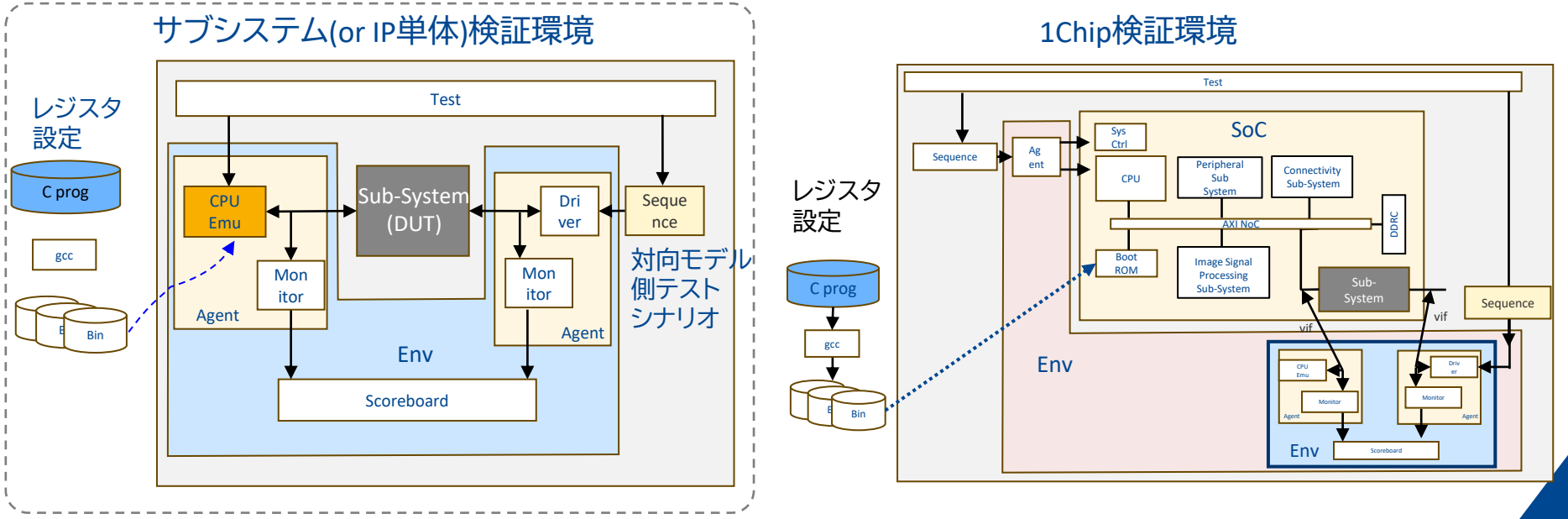
単体検証環境でのCプログラム実行高速化

単体検証環境にCPUコアを組み込むことで、レジスタ設定のテストシナリオをCプログラムで記述し、1Chip検証にもCプログラムを流用する手法が使われる場合があります。

→ 単体検証環境にCPUコアを組み込むため、シミュレーション実行時間が長くなる欠点がある

対策案

CPUコア RTLの代わりに、オープンソースソフトのCPUエミュレータを検証環境に組み込むことで、シミュレーションの高速化を実現します (ARMおよびRISC-Vなどに対応可能)

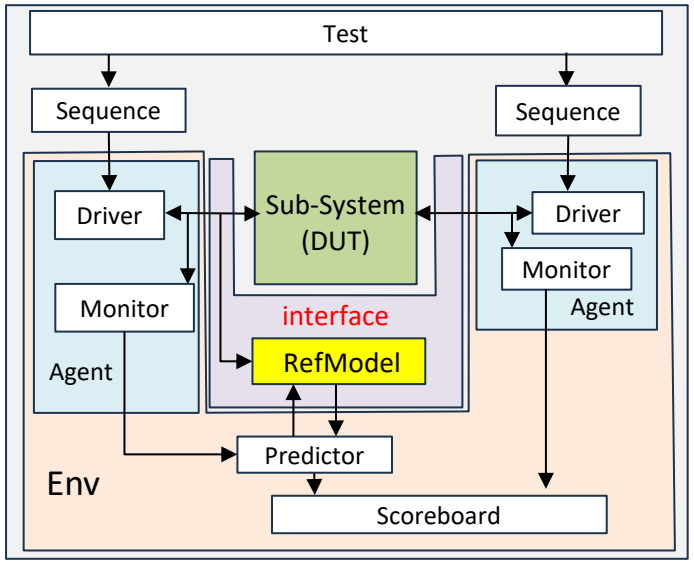


検証環境の流用性向上のための施策(2)

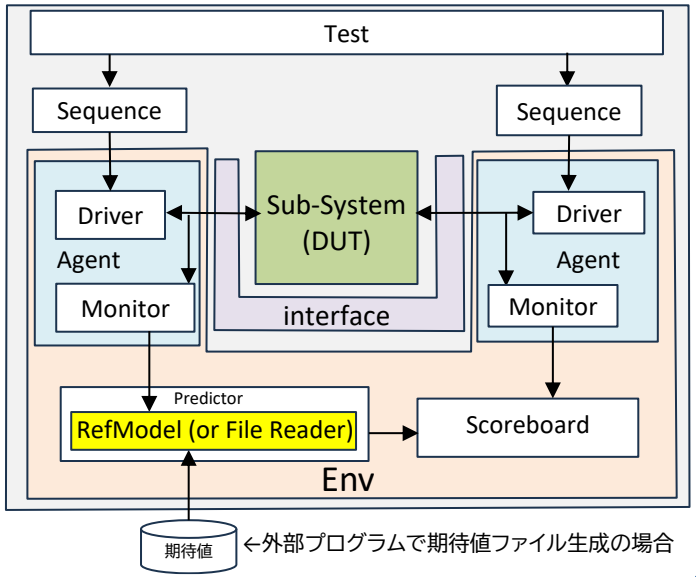
検証環境の流用性を上げるため、期待値を生成するReference Modelの種類により、検証環境への組み込み方法が異なります。

検証環境内にReference Modelを組み込む方法	module記述 (Verilog ビヘイビア記述)	Interface部に組み込む
	Cモデル記述(DPI-C経由)	Env内に組み込む
	System-C記述	
	class記述 (System Verilog)	
外部プログラム(Reference Model)による期待値生成	外部ファイルからの期待値読み込み	期待値ファイル名は、testクラスで指定できるようにする

Reference Modelのinterface部への組み込み



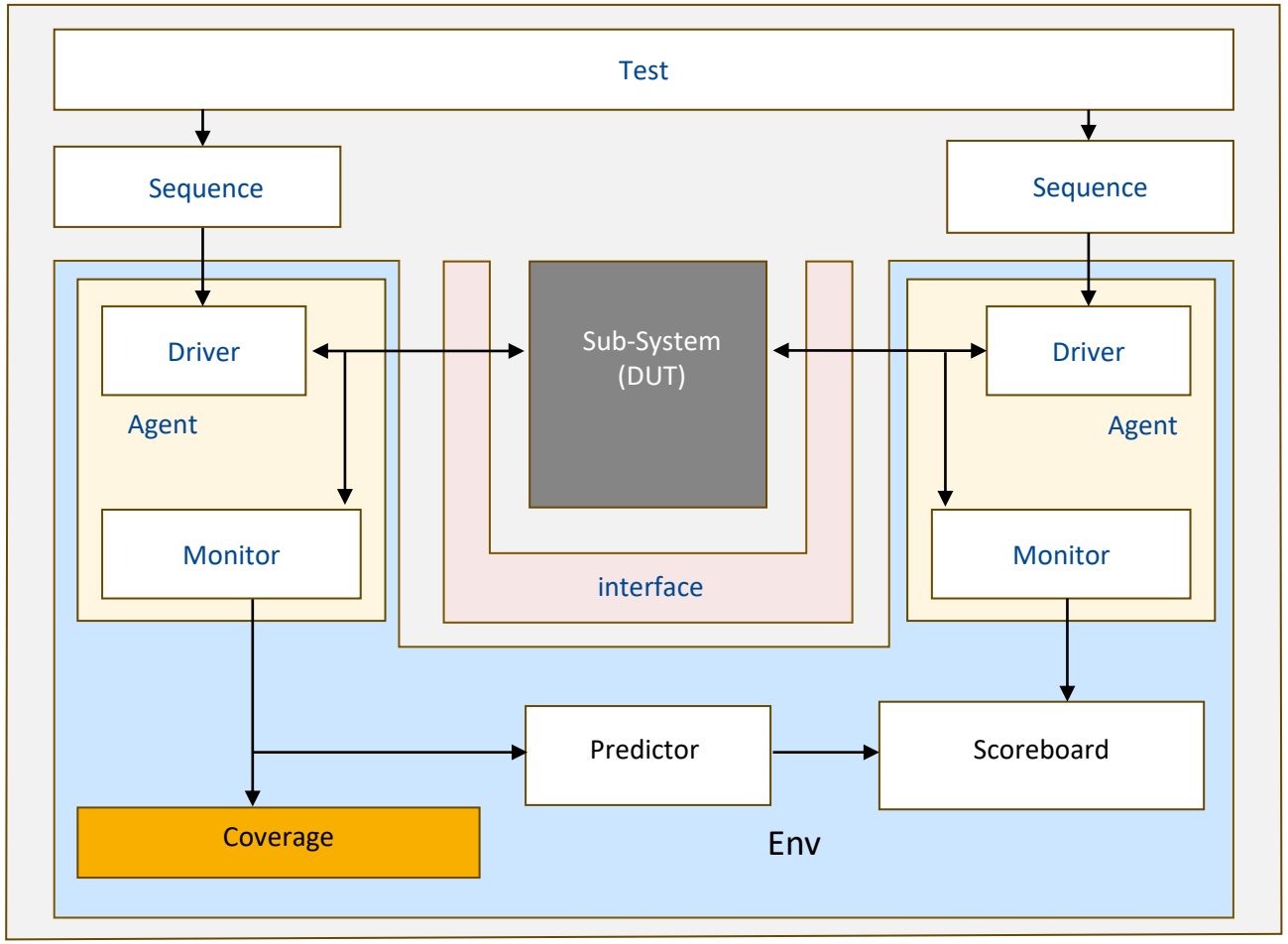
Reference ModelのEnv内への組み込み



検証環境の流用性向上のための施策(3)

検証環境の流用性を上げるため、カバレッジ記述は次のように組み込みます。

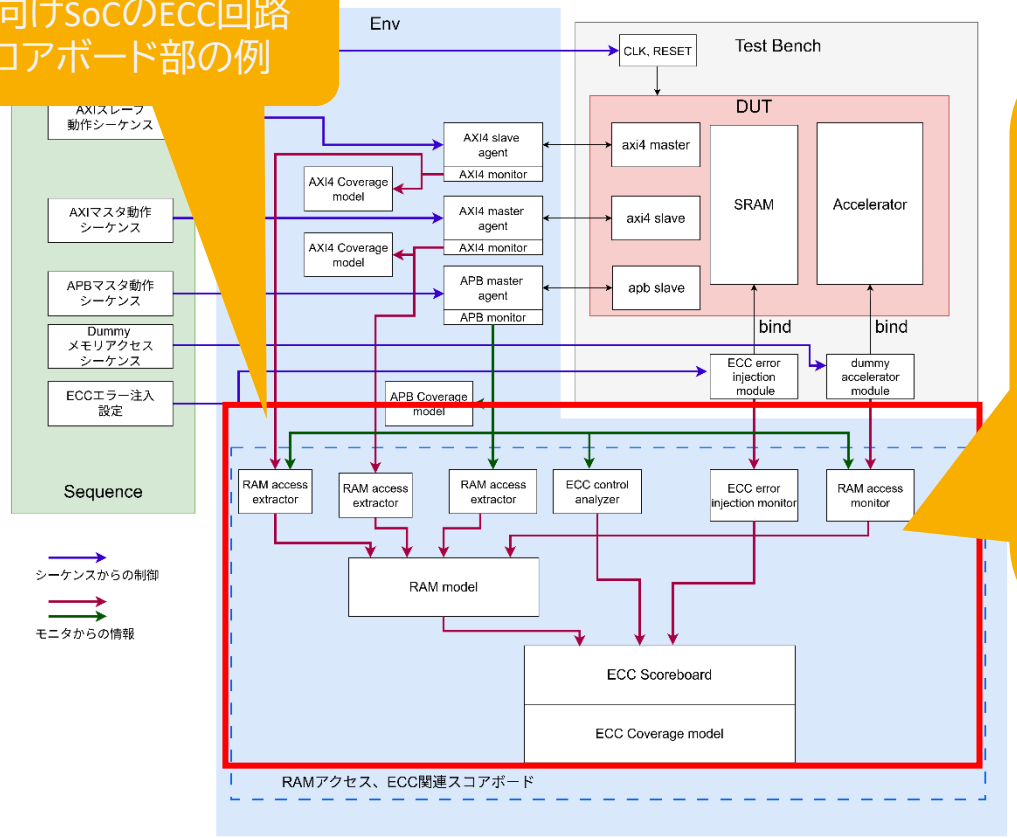
- UVMコンポーネントとして記述し、Env内に組み込みます。
- Monitor情報をカバレッジの入力とします。(複数のMonitor情報を入力可)



スコアボード部の品質/流用性向上

- UVM環境において、スコアボード部は期待値とDUTの出力を比較する機構ですが、期待値生成や比較のための記述が複雑になりがちです。(スコアボードだけで数千行以上のケースも多いです)
- 複雑化するとメンテナンス性や流用性が低下するため、それを防ぐことが重要です。
- 対策としては、スコアボード部の機能をさらに小さなコンポーネントに分割して、個々のコンポーネントの複雑さを抑えます。

車載向けSoCのECC回路のスコアボード部の例



スコアボードと期待値モデル生成(Reference Model)の機能を単純機能の小さなコンポーネントに分割

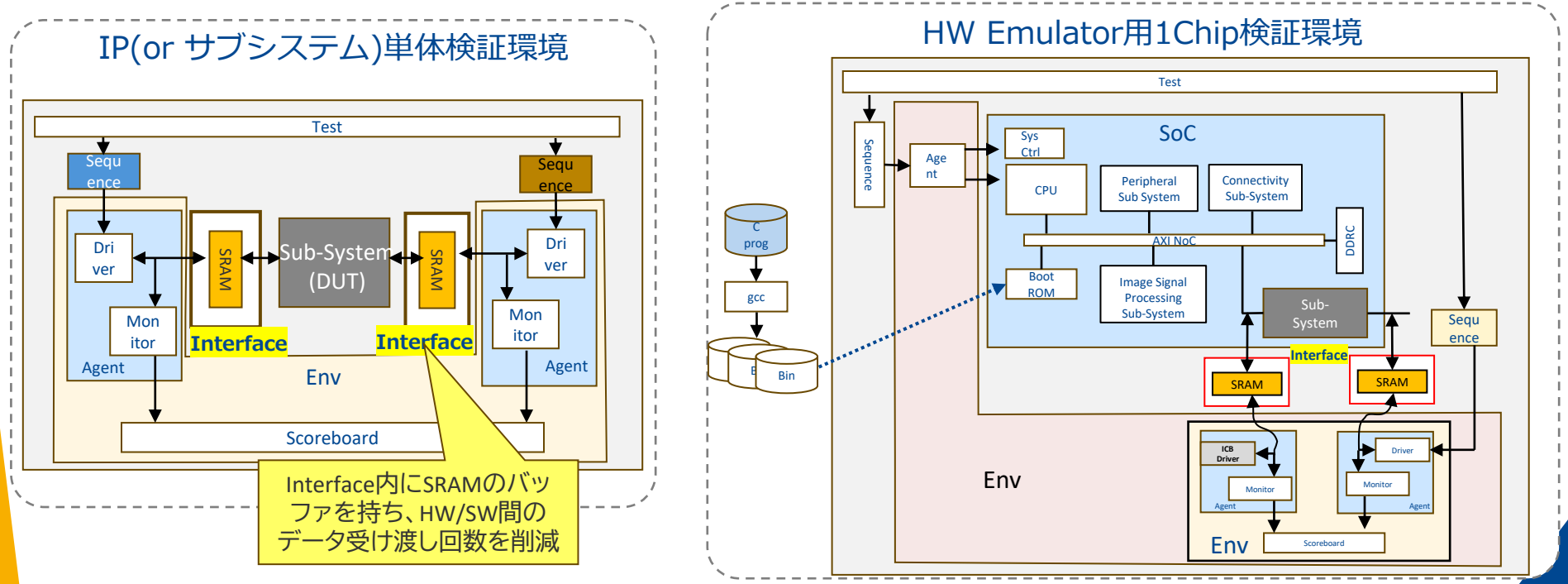
- AXIバスのモニタ情報からRAMアクセス情報のみを抽出する部分
- APBバスのモニタ情報からECC制御レジスタ情報のみを抽出する部分
- RAMモデルの部分
- 比較機能の部分(狭義のスコアボード)
- エラー注入箇所のカバレッジ

各コンポーネント間の接続を標準化することで、流用性と拡張性をさらに改善することも可能です。

HW Emulator対応の検証環境構築

IP/サブシステムの単体検証環境構築の時点からHW部とSW部の間のデータ受け渡しの回数を削減する対策を行い、HW Emulator対応による検証環境の修正(後戻り)を抑えます。

- HW Emulatorで1chip検証を実行する場合、検証環境のSW処理部分はアクセラレートできない問題があるため、 HW部(DUT他)とSW部(UVM検証環境)間のデータ受け渡しの回数を減らし、より高速に実行できるようにする必要があります。
- 単体検証環境構築時に次の対策を行います。
  - 毎クロック動作する記述はInterface部分のみに限定し、Interface内にSRAMで構成されるバッファやパターンジェネレータを持ちます。
  - 1個のSequence Item(トランザクション)で数百～数千回程度のクロックサイクル(例：画像処理の1ライン or 1フレーム分)のデータを一括で生成/チェックします。



弊社における再利用・効率化の取り組み

- 弊社では、独自VIPを開発し、再利用による検証環境の品質・開発効率の改善に取り組んでいます。
- 各VIPに対し、サンプル環境、サンプルテストシナリオを整備し、開発効率を改善しています。
- 各VIPは、前述の施策で流用性を高めています。

種類	VIP (開発中を含む)
バス	AXI, AHB, APB
CPU	ARM Cortex-A, ARM Cortex-M
	RISC-V
シリアル	I2C, CXPI
高速IF	MIPI
	Ether
ECC RAM	エラー注入モジュール
期待値モデル	検討中 (画像処理など)
スコアボード	検討中
自動生成ツール	検証環境自動生成ツール

Thank you!