2023
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
JAPAN

SoC開発を成功させる検証戦略とは

CMエンジニアリング株式会社
二見　誠一

CM ENGINEERING
CMエンジニアリング株式会社

accellera
SYSTEMS INITIATIVE

概要

近年の大規模SoC開発において検証の比重がますます高くなる中、検証ゴールへの道筋を示す検証戦略を開発目標や条件など多角的に検討して策定すること、そしてプロジェクト全体で共有することは、開発の成功可否を決める検証の第1歩であり重要なポイントです。

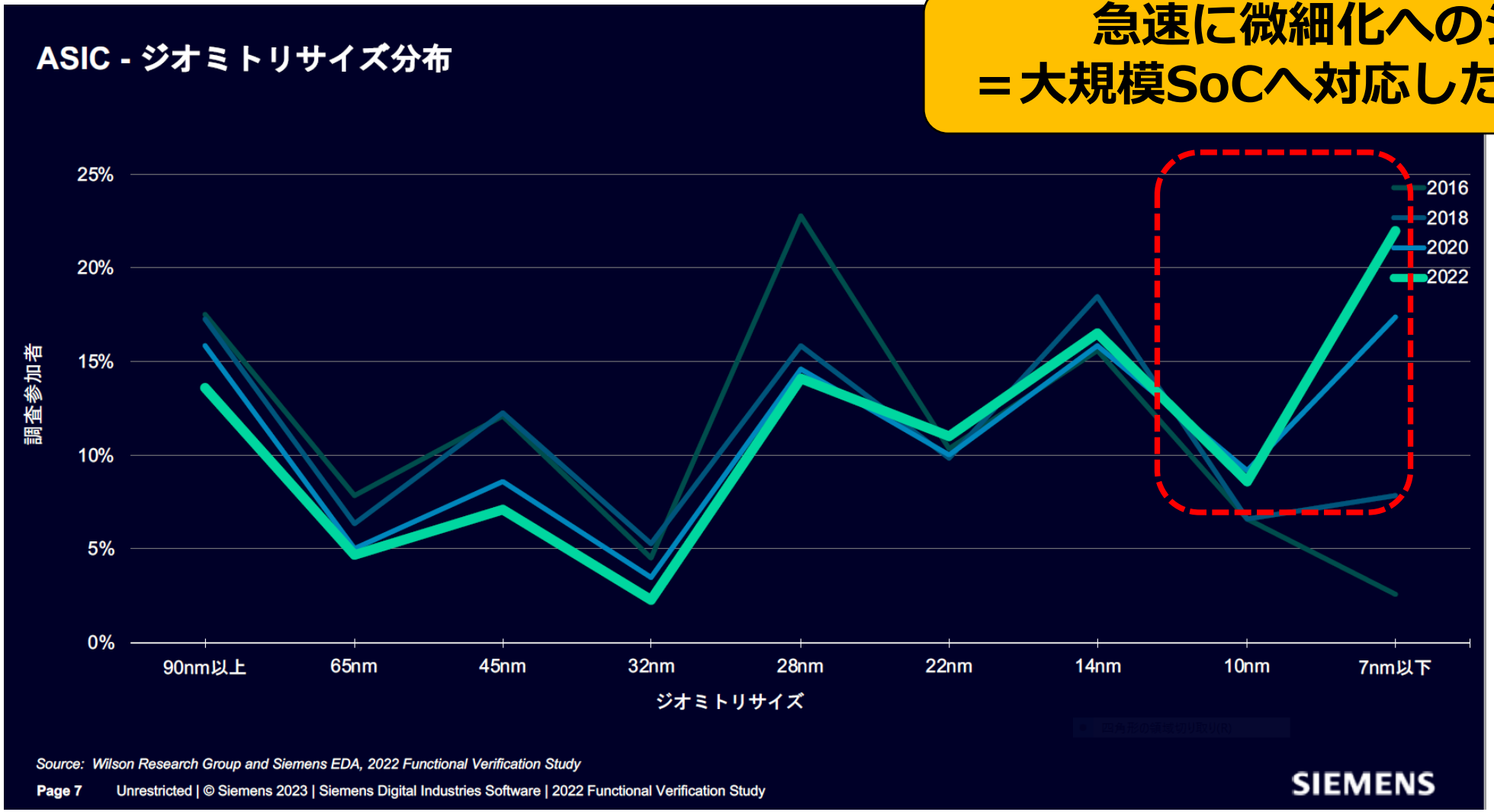
このチュートリアルでは、SoC開発全体の検証戦略を策定する上で検討すべきアイテムと、その中でも重要なキーとなるリスクの洗い出しとそのマネージメントの戦略について、
①検証漏れを防ぐ検証階層、検証範囲、検証手法の設定
②スケジュールとリソース、コストの両立
に着目し、「SoC開発を成功させる検証戦略とは？」としてまとめた内容を説明します。

アジェンダ

1. 検証戦略の必要性
2. SoC検証戦略の策定
3. リスクの洗い出しとリスクマネージメント戦略
   - ◆検証範囲定義ミスによる検証漏れの発生
     検証マップの考え方と検証空間定義の重要性
   - ◆スケジュールとリソースマネージメントの両立
4. SoC開発を成功させる検証戦略とは？
5. まとめ

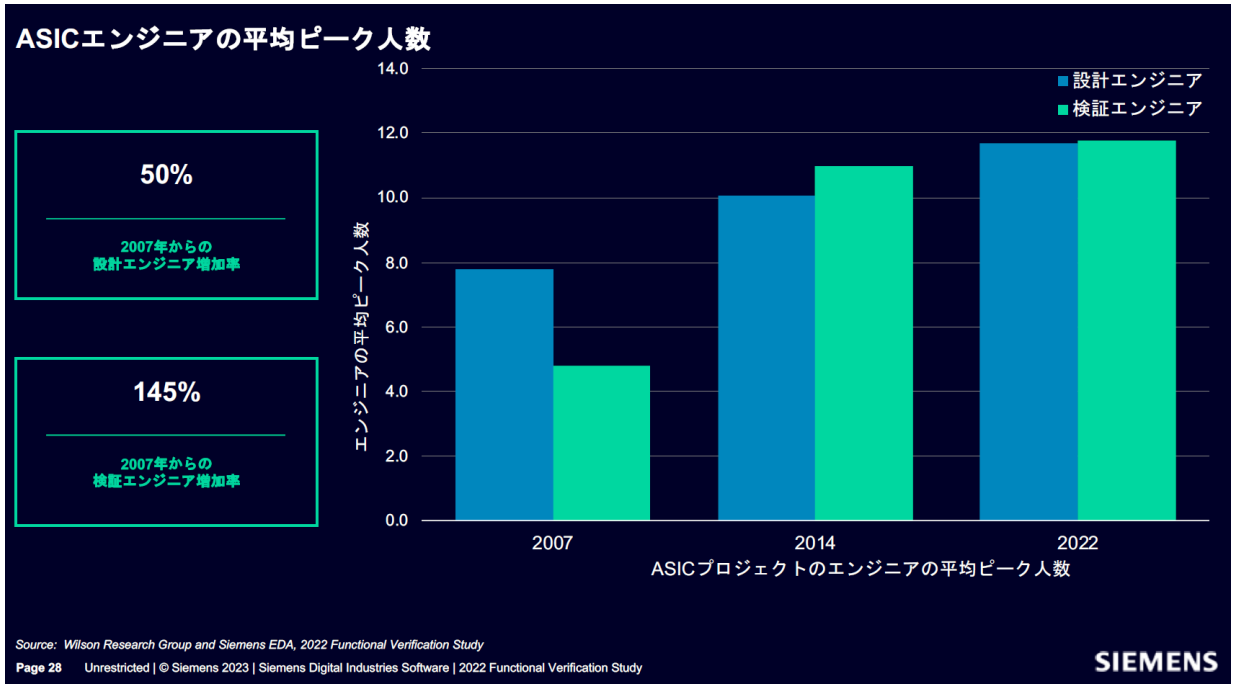
1．検証戦略の必要性（1/4）

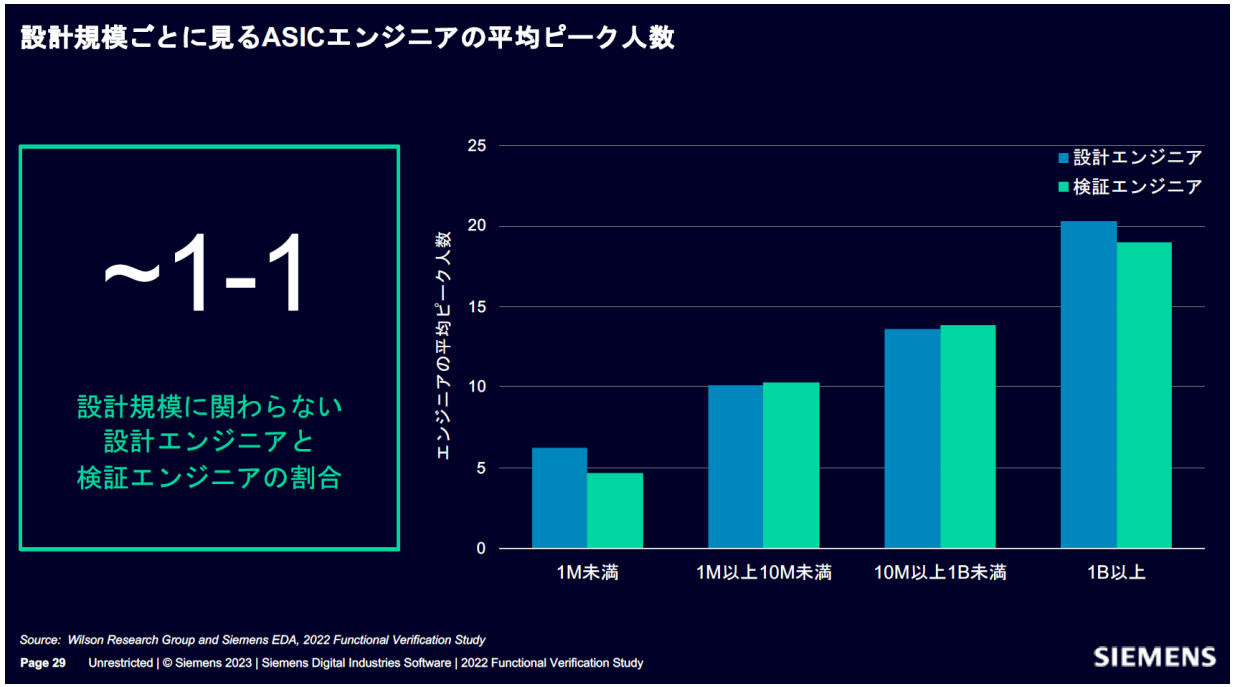
急速に微細化へのシフト
＝大規模SoCへ対応した検証が必要



SIEMENS様
「機能検証の最新市場調査とその分析」より

1．検証戦略の必要性（2/4）



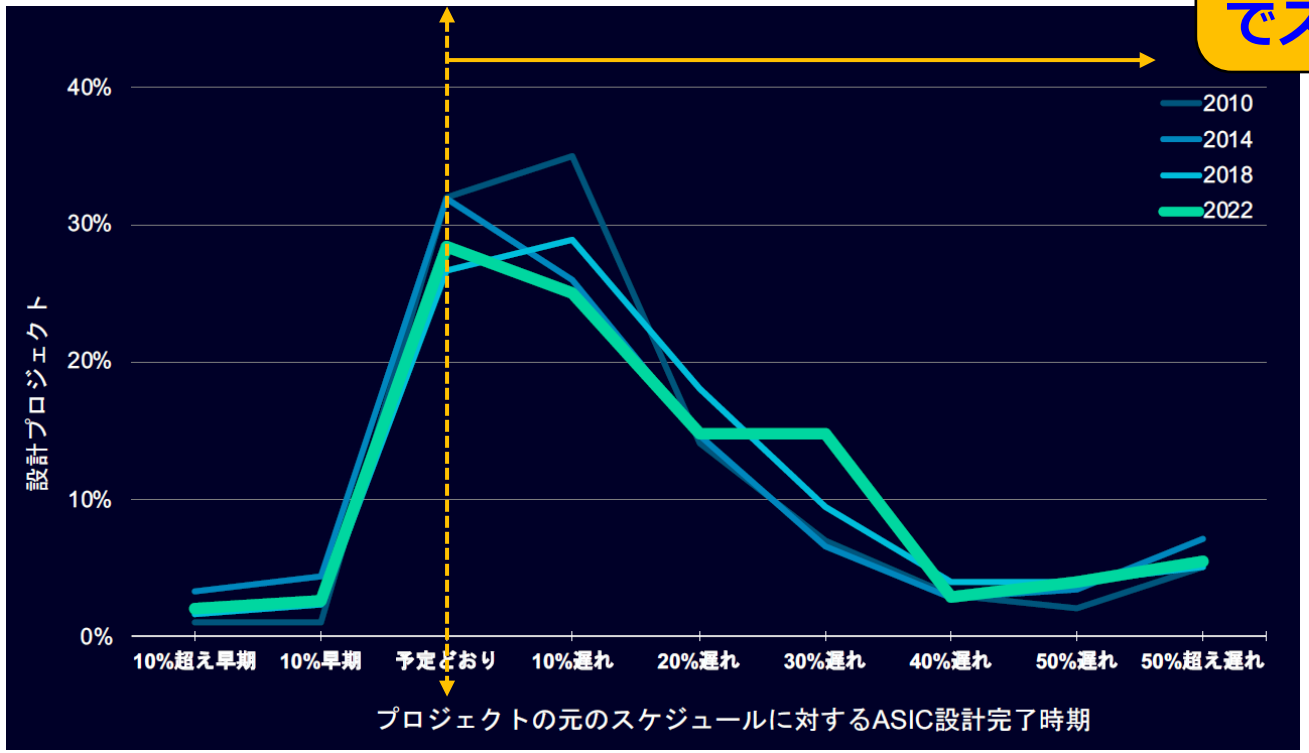


現状：設計エンジニア＝検証エンジニア
大規模SoC開発：設計エンジニア ＜ 検証エンジニア

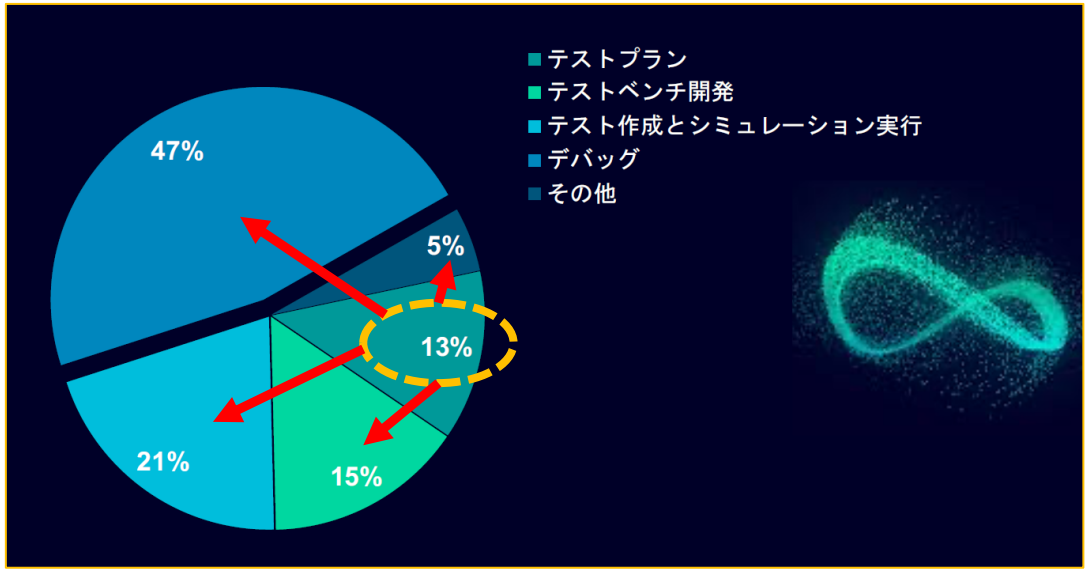
SIEMENS様
「機能検証の最新市場調査とその分析」より

1．検証戦略の必要性（3/4）

予定より遅延したプロジェクト：66%



検証工程の約半分はデバッグで消費
効果的な検証手法を導入し生産性を向上させる事でスケジュール遵守、且つ検証品質確保が可能



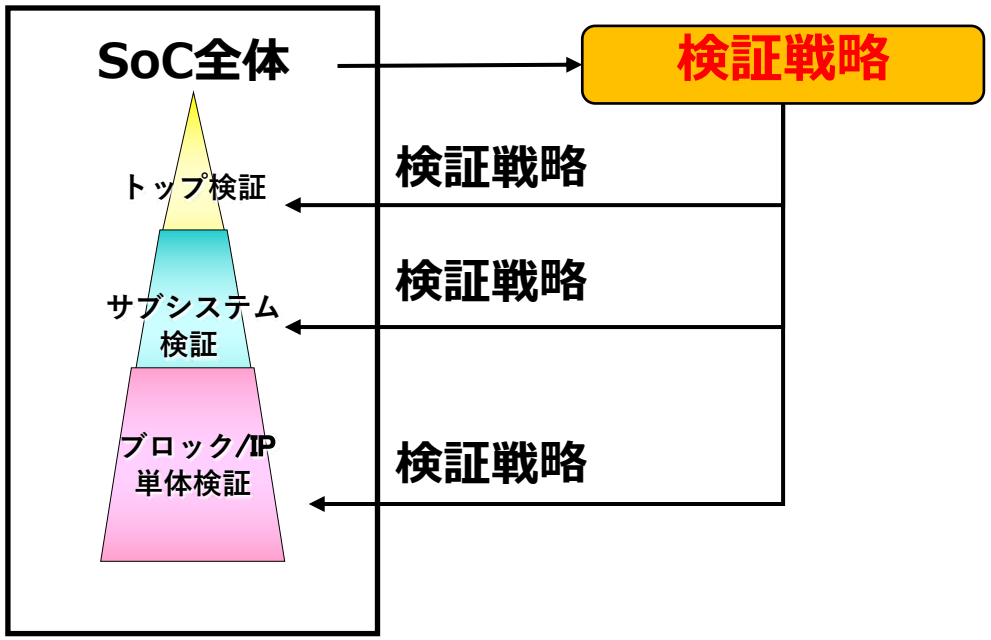
SIEMENS様
「機能検証の最新市場調査とその分析」より

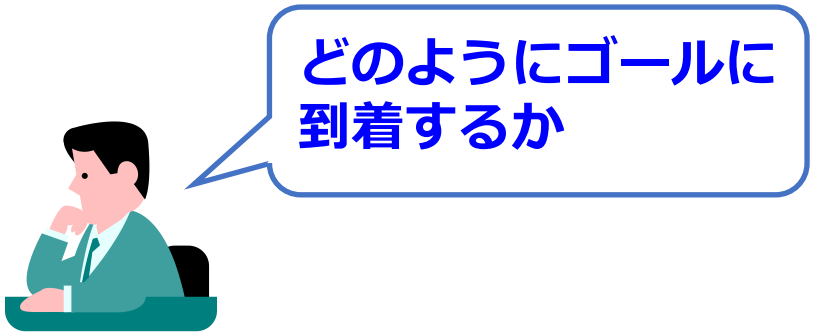
テストプラン（検証戦略）は検証すべての工程に影響し、
プロジェクト成功可否に大きく影響を与える重要なポイント

1．検証戦略の必要性（4/4）

- **検証戦略はトップダウンで定義**

それを基に各階層で検証戦略を定義





方針：

① **ボトムアップからのアプローチはもう古い！ボトムとトップで検証範囲をバランス良く設定する事が重要**

② **ブロック単体の段階で網羅的な検証を実施**

③ **上位階層は、複数ブロックが関係する複合的な動作の確認**

④ **階層毎に検証の着目点、検証の役割を明確化**

2. SoC検証戦略の策定

検証戦略における各アイテム

① 検証方針（検証への取り組み方）/重点項目（特に重視する事柄）

② 検証ゴール（検証の終了判断）/検証プロセス（進捗・課題管理）

③ 検証手法（採用する検証手法）/EDAツール

④ 再利用性に関する方針（採用する再利用資産/今後の再利用へ向けた対応）

⑤ 機能検証項目/検証の進め方
- 検証項目の把握： 検証すべき項目の抽出
- 検証の進め方： 対象の決定、実施順序等の決定

⑥ スケジュール/分担
- 検証ボリューム
- スケジュール：実施計画、担当者：担当者のアサイン

3. リスクの洗い出しとリスクマネージメント戦略

リスクの種類

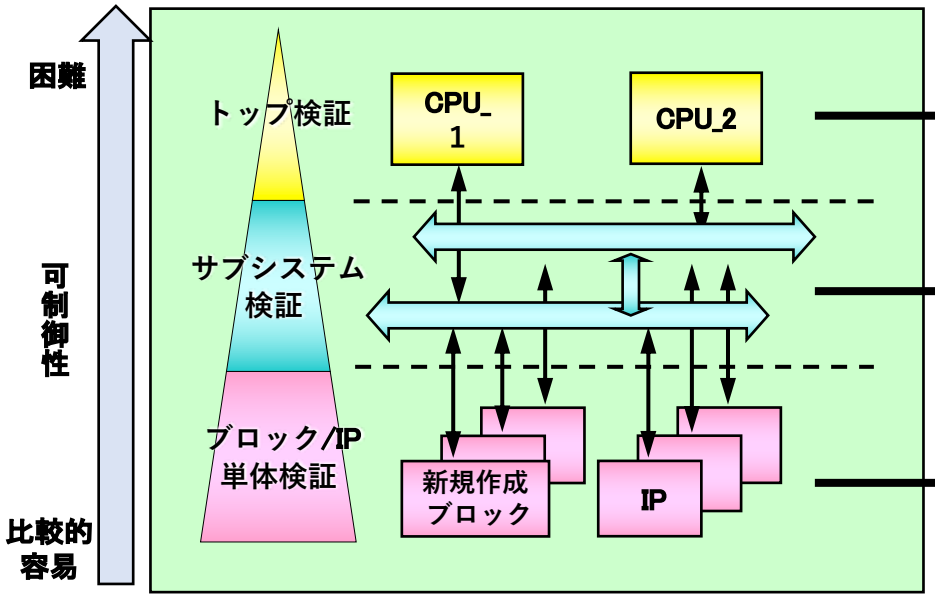
◆検証範囲定義ミスによる検証漏れの発生
検証単位：TOP、ユニット、ブロック
検証手法：ダイナミック（シミュレーション、エミュレーション、FPGA、実機）
フォーマル

◆スケジュールとリソース、コストの両立
スケジュール：検証ボリュームとスケジュールのアンバランス
リソース：担当者のスキル問題

◆HW/SWの役割分担

◆新規設計とIP、既存回路の修正箇所における検証粒度

検証範囲定義ミスによる検証漏れの発生（1/8）



確認事項：接続、連携動作、割り込み
検証手法：ダイナミック検証（シミュレーション、エミュレーション、FPGA、実機）

確認事項：バス性能
検証手法：ダイナミック検証（シミュレーション）

確認事項：機能
検証手法：ダイナミック検証（シミュレーション）
フォーマル検証

検証項目			確認内容	備考
大項目	中項目	小項目		
機能1	機能1-1	機能1-1①		
		機能1-1②		
	機能1-2	機能1-2①		
		機能1-2②		
		機能1-2③		
機能2	機能2-1	機能2-1①		
		機能2-1②		
	機能2-2	機能2-2①		

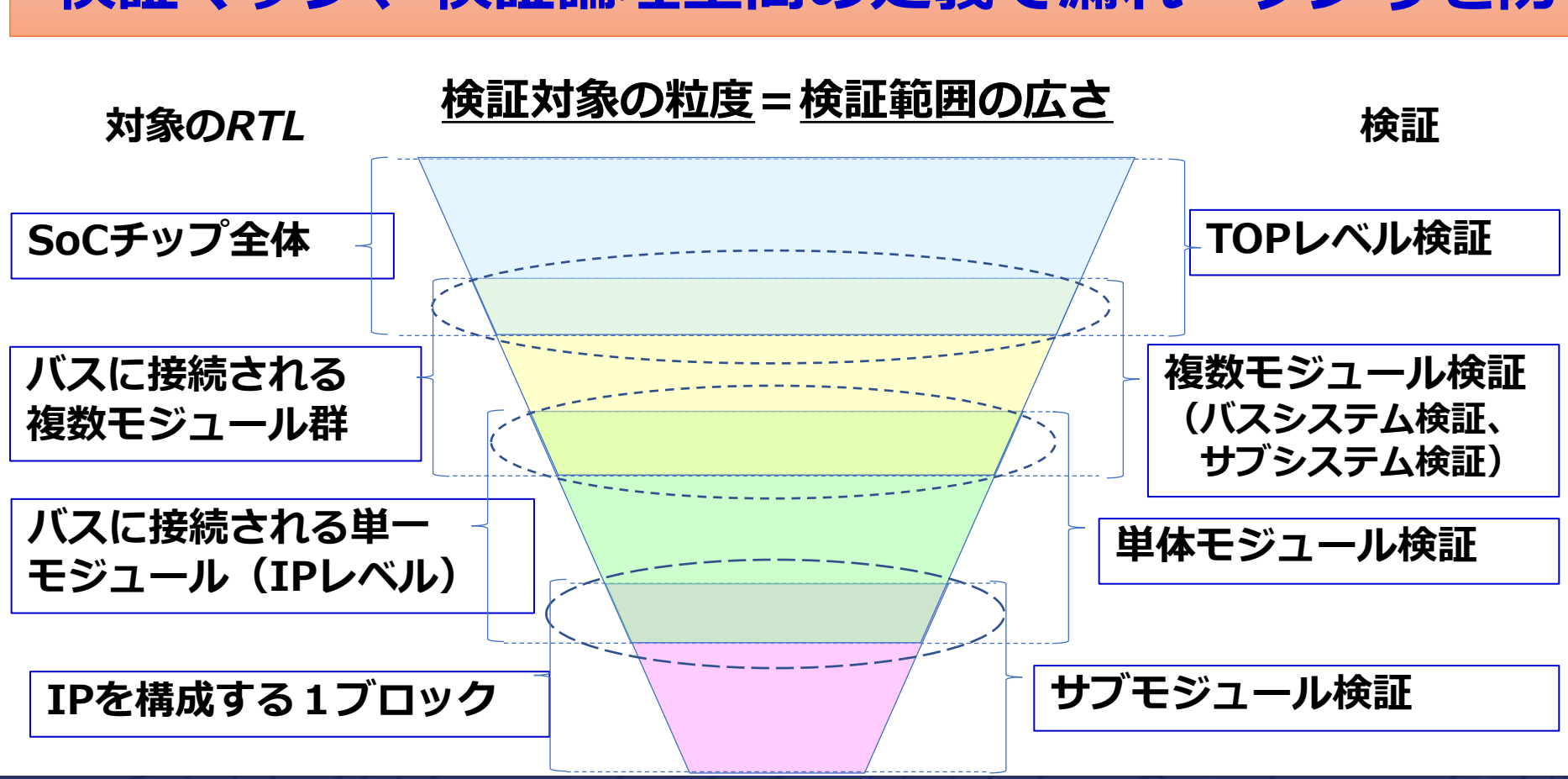
検証範囲は明確？
検証手法有りきになってない？

×検証手法　⇒　検証項目
○検証項目　⇒　検証手法

検証範囲定義ミスによる検証漏れの発生（2/8）

■ 検証の階層分け

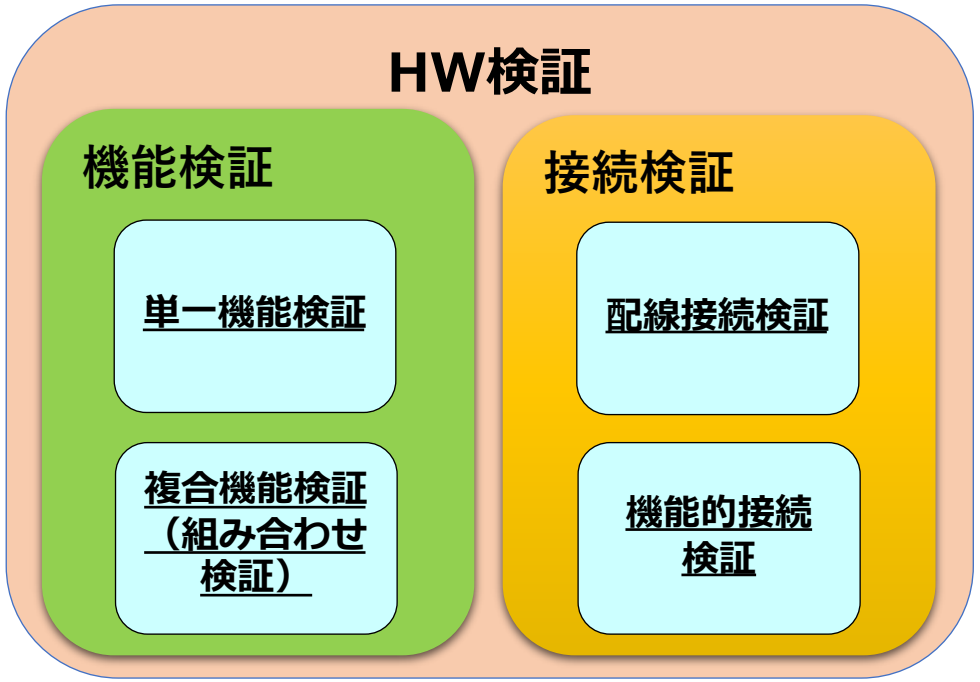
検証マップ、検証論理空間の定義で漏れ・ダブりを防ぐ



検証範囲定義ミスによる検証漏れの発生（3/8）

検証マップの考え方

検証マップの定義は二つに分けられる
①機能を階層に分け、検証単位で紐付け
②全体ブロック図から検証範囲を定義、要求仕様/機能仕様/検証単位で紐付け



検証分類		検証観点
LV1	LV2	
機能検証		対象ブロックの機能が仕様通り、正しく動作
	単一機能検証	機能ブロック内の全機能の確認 ありえるタイミング、取りえるパラメータ確認
	複合機能検証	モジュール間の連携動作、仕様の齟齬
接続検証		対象ブロックの接続が正しい
	配線接続検証	アナログ、電源、特殊信号等の配線が正しい
	機能的接続検証	モジュール間信号が正しく接続されている 極性、タイミング、プロトコルが正しい

検証範囲定義ミスによる検証漏れの発生（4/8）

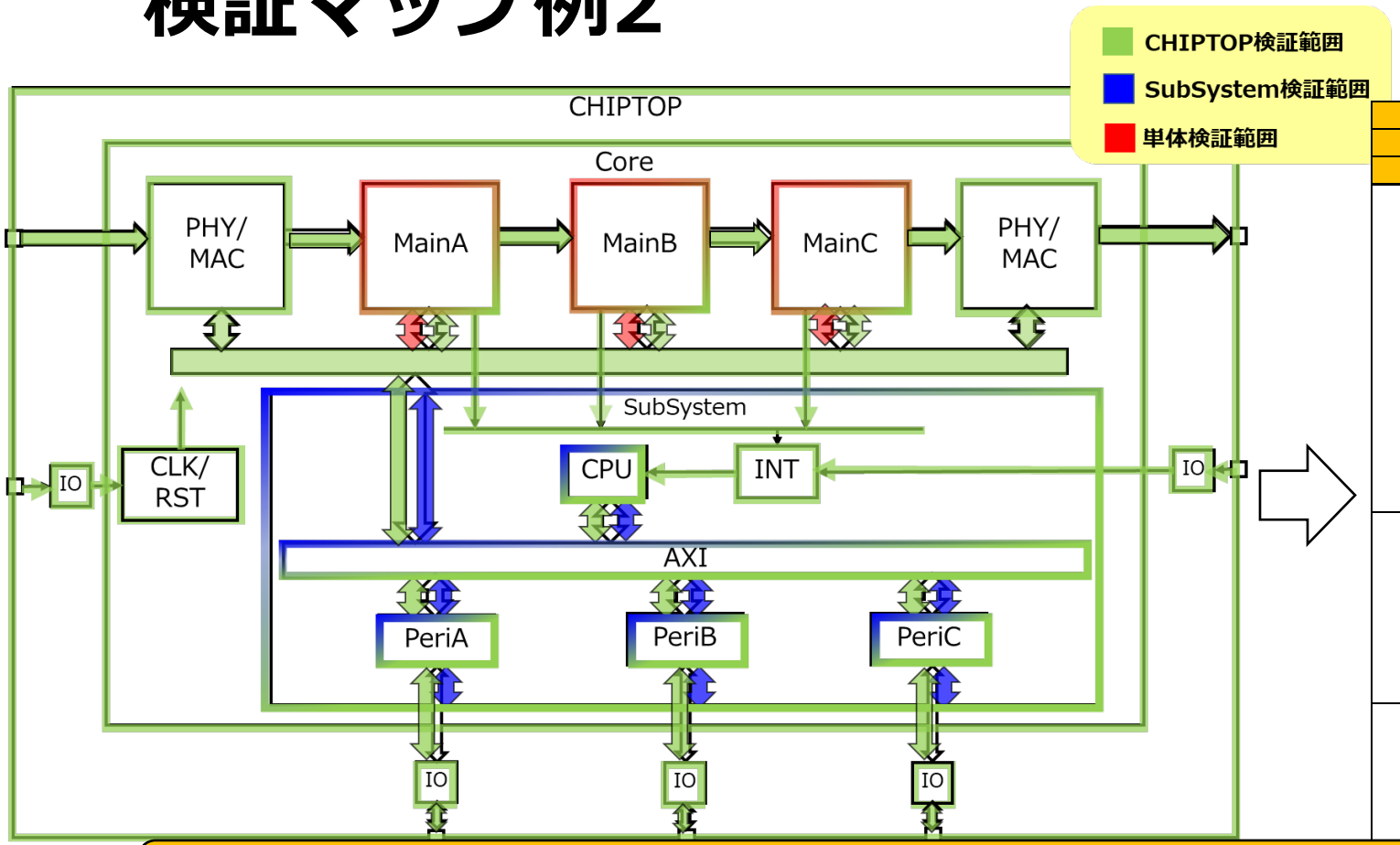
検証マップ例1

機能毎に階層分け⇒検証単位に分解

No	分類の階層 Lv1	Lv2	Lv3	Lv4	Lv5	Lv6	Lv7	Lv8	Lv9	XXXプロジェクト内で実施する検証 TOP検証（●）	サブシステム検証	モジュール単体検証	XXXプロジェクト以外でIPとして検証済（▼）	検証管理担当 チームA（◆）	チームB（☆/★）
1	ハードウェア全体機能														
1-1		外部インタフェース													
1-2		クロック、リセット													
1-3		システム制御													
1-4		CPU													
1-4-1			CPU												
1-4-1-1				コンフィグレーション											
1-4-1-1-1					コンフィグレーション・パラメータ					-	-	CPU_Top検証	-	◆	-
1-4-1-2				機能											
1-4-1-2-1					CPU機能					-	-	-	▼	-	-
1-4-1-3				接続											
1-4-1-3-1					CPU内部接続					-	-	-	▼	-	-
1-4-1-3-2					CPU外部接続										
1-4-1-3-2-1						ATCM				-	-	CPU_Top検証	-	◆	-
1-4-1-3-2-2						BOTCM				-	-	CPU_Top検証	-	◆	-
1-4-1-3-2-3						AXIマスタ				-	バスシステム検証	-	-	-	☆
1-4-1-3-2-4						AXIスレーブ				-	バスシステム検証	-	-	-	☆
1-4-1-3-2-5						DAP				●	-	-	-	-	★
1-4-1-3-2-6						端子の固定				-	-	CPU_Top検証	-	◆	-
1-4-1-3-2-7						その他				●	-	-	-	-	☆
1-5		バス													
1-5-1			APBバス（UART、GPIO、I2C）												
1-5-1-1				機能											
1-5-1-1-1					AHB⇒APBブリッジ					-	バスシステム検証	-	-	-	☆
1-5-1-2				接続											
1-5-1-2-1					APBマスタ										
1-5-1-2-1-1						AHB⇒APBブリッジ				-	バスシステム検証	-	-	-	☆
1-5-1-2-2					APBスレーブ										
1-5-1-2-2-1						UART				-	バスシステム検証	-	-	-	☆
1-5-1-2-2-2						GPIO				-	バスシステム検証	-	-	-	☆
1-5-1-2-2-3						I2C				-	バスシステム検証	-	-	-	☆
1-5-1-2-3						その他				●	-	-	-	-	☆
1-6		通常動作機能ブロック													
1-7		テスト動作機能ブロック													
1-7-1			BIST(SRAM)												
1-7-1-1				機能											
1-7-1-1-1					BISTコントローラ					-	-	-	▼	-	-
1-7-1-1-2					BISTコントローラ以外					●	-	-	-	◆	-
1-7-1-2				接続											
1-7-1-2-1					BISTコントローラ内部					-	-	-	▼	-	-
1-7-1-2-2					BISTコントローラ外					●	-	-	-	◆	-
1-8		組み合わせ拡張													
1-8-1			機能の組み合わせ												
1-8-1-1				AA+BB+CC							XYZサブシステム検証				
1-8-1-3				上記の組み合わせ以外						●					
1-8-2			タイミングの組み合わせ												
1-8-2-1				AA+BB+CC							XYZサブシステム検証				
1-8-2-3				上記の組み合わせ以外						●					
2	システム動作									●					

検証範囲定義ミスによる検証漏れの発生（5/8）

検証マップ例2



7点P些乱		ﾙﾋﾞ1加些乱		μs9日		
7点P些乱竕刃		ﾎﾟﾝ俑㋮㋇1月㊮	ﾙﾋﾞ1加些乱竕刃	μs9日个粟/μs9日竕ﾎﾟﾝ竕刃		
㋹竕ﾎﾟﾝ	ﾏｯﾊﾟ竕ﾎﾟﾝ			CHIPTOPμs9日	SubSystem μs9日	佾ﾀﾞｰμs9日
A	A-1	㋮㋇1月㊮獣	A-1-1	TOP-A-1		獣-A-1-1-1
				-		獣-A-1-1-2
		㋮㋇1月㊮獣	A-1-2	TOP-A-2		獣-A-1-2-1
				-		獣-A-1-2-2
	A-2	㋮㋇1月㊮獲	A-2-1	TOP-A-3		獲-A-2-1-1
	A-3	㋮㋇1月㊮獣	A-3-1	-	Sub-A-3	獣-A-3-1-1
						獣-A-3-1-2
						獣-A-3-1-3
						獣-A-3-1-4
			A-3-2	-		獣-A-3-2
	A-4	㋮㋇1月㊮獣	A-4-1	-		獣-A-4-1
	A-5	㋮㋇1月㊮獣	A-5-1	-		獣-A-5-1
B	B-1	㋮㋇1月㊮獲	B-1-1	TOP-B-1		獲-B-1-1
		㋮㋇1月㊮獲	B-1-2	TOP-B-2		獲-B-1-2-1
						獲-B-1-2-2
						獲-B-1-2-3
		㋮㋇1月㊮獲	B-1-3	-		獲-B-1-3
	B-2	㋮㋇1月㊮獲	B-2-1	TOP-B-3		獲-B-2-1
	B-3	㋮㋇1月㊮獲	B-3-1	TOP-B-4		
C	C-1	㋮㋇1月㊮獷	C-1-1	TOP-C-1		獷-C-1-1
		㋮㋇1月㊮獷	C-1-2	-	Sub-C-1	獷-C-1-2
	C-2	㋮㋇1月㊮獷	C-2-1	-		獷-C-2-1
		㋮㋇1月㊮獷	C-2-2	-		獷-C-2-2
	C-3	㋮㋇1月㊮獷	C-3-1	-		獷-C-3-1
		1月㊮獷	C-3-2	-		獷-C-3-2

全体ブロック図から検証範囲を定義⇒検証単位に分解

検証範囲定義ミスによる検証漏れの発生（6/8）

検証論理空間定義の重要性

検証論理空間X：考えられる組合せ
検証論理空間A：仕様を網羅する空間
検証論理空間B：検証者が機能仕様の理解をもとに定義する空間
検証論理空間C：本開発で検証すべき空間
（ユースケース、システムに影響を与える箇所、設計実績等）

検証論理空間の決定⇒検証項目抽出⇒検証ランク付け

検証範囲定義ミスによる検証漏れの発生（7/8）

検証項目抽出方針、チェック方針を検証戦略で策定

① 検証項目表の標準カテゴリの決定（クロック/リセット、インタフェース、単一機能、組合せ機能、性能、・・・）

② 仕様書と検証項目表の対応（紐付け）

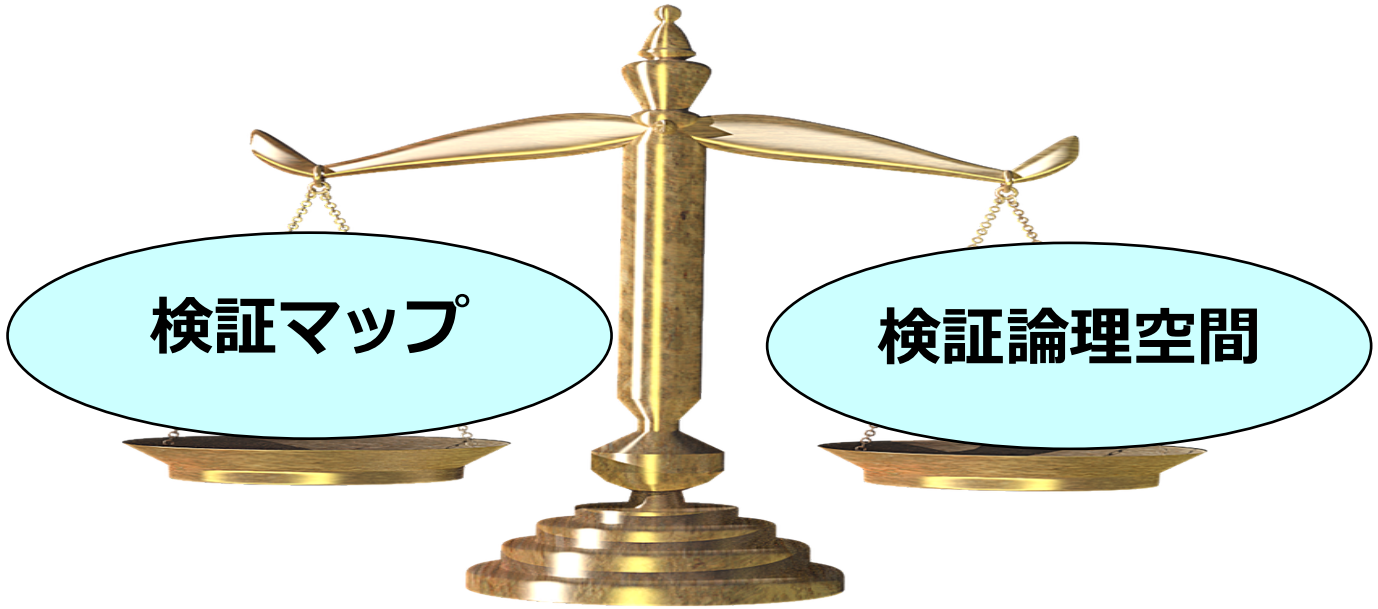
◆仕様書側の粒度(章レベル、節レベル、文章レベル、その他)を明確にする事が重要

③ 設計者からの設計情報Input

◆設計者が気になるポイント

④ レビューでのチェックポイント抽出

検証範囲定義ミスによる検証漏れの発生（8/8）



検証範囲を定義→検証項目抽出方針確定

検証漏れを撲滅

スケジュールとリソース、コストの両立（1/6）

検証ボリュームが膨大

開発スケジュールが厳しい

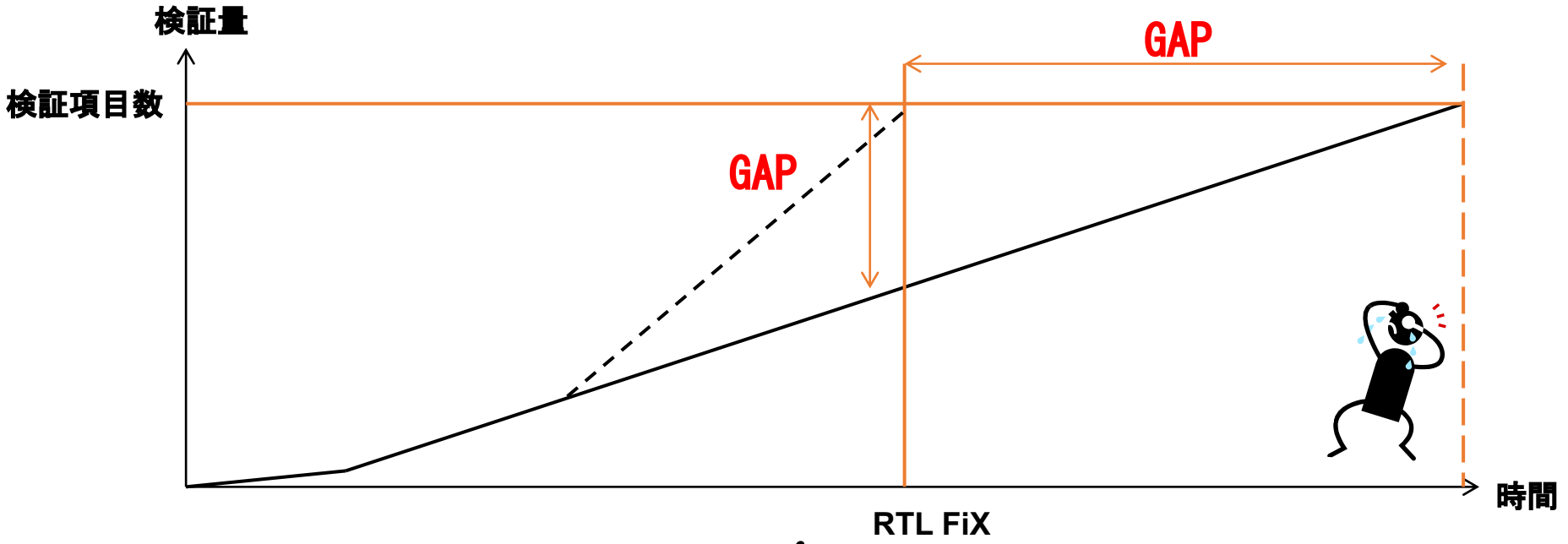
どう両立してプロジェクトを
成功に導けばよいの？？？

リソースは無限ではなく有限

コスト要求が厳しい

スケジュールとリソース、コストの両立（2/6）

1. 検証ボリュームとスケジュールのアンバランス

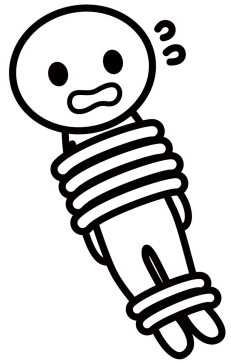


どう解決していけば良いの？

スケジュールとリソース、コストの両立（3/6）

SoC開発において再利用が多ければ多いほど呪縛にとらわれやすい・・・

- ダイナミック検証でやり切れた（成功体験の呪縛）
- 新しい手法を取り入れる時間がない（不安の呪縛）



⇒もう一度、検証対象に対して必要な検証手法を考えよう

①Human（人）vs ツール（手法）、②ツール vs コスト 、③ 開発期間
①、②、③のパラメータを比較し、 最適な検証手法を選択

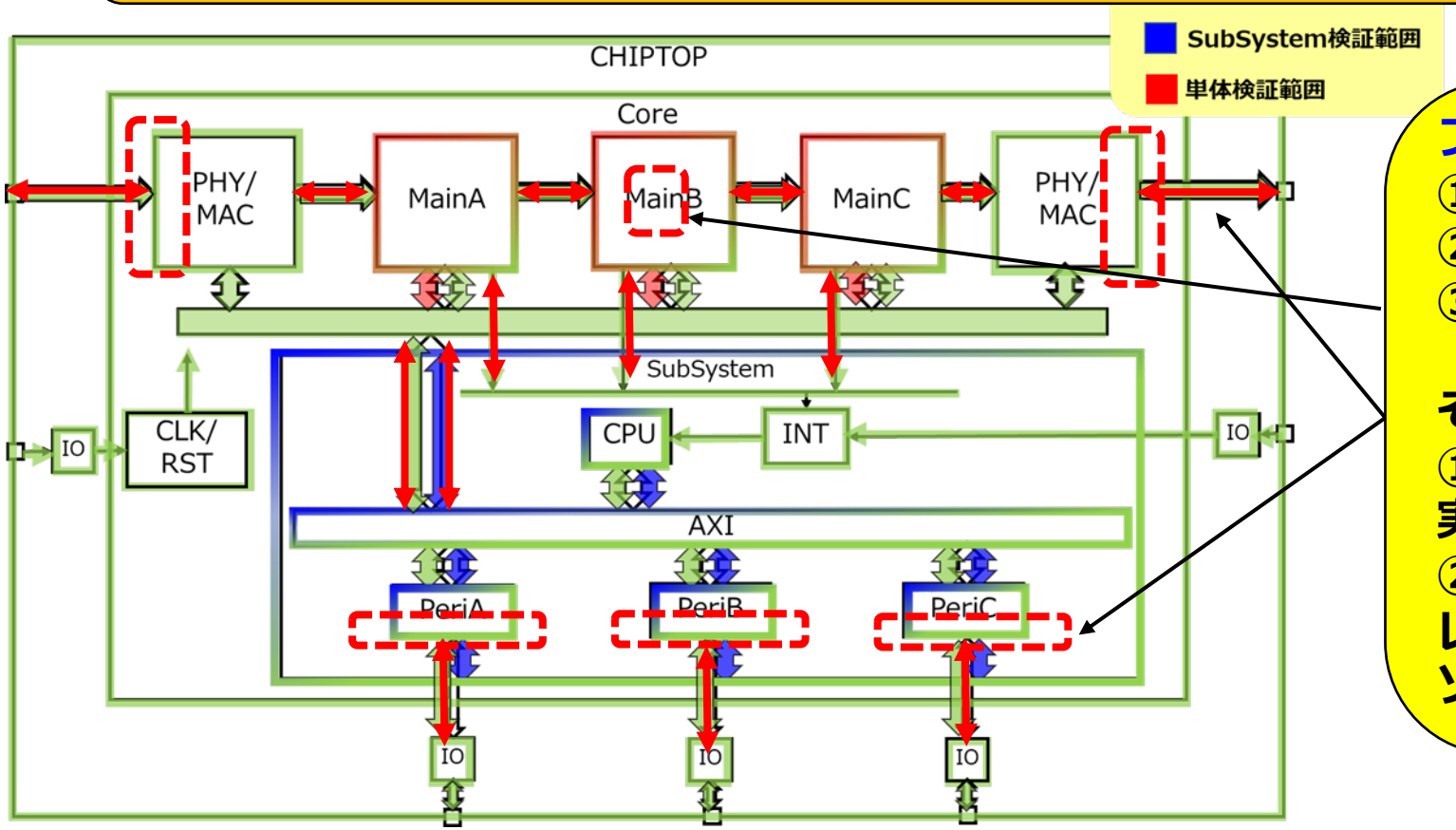
スケジュールとリソース、コストの両立（4/6）

ダイナミック（シミュレーション以外）とフォーマルの特長

手法	実行速度	メリット	デメリット
ダイナミック（エミューレーション）	○	テストベンチの一部をエミュレータに取り込み数百倍の速度で実行可能 ⇒①大量のデータを扱う検証 ②確認対象の時間が長い検証	テストベンチの移植に少し手間がかかる コンパイルに時間を要する
ダイナミック（FPGA）	◎	実時間に近い環境でテストが可能 実際の規格の周波数で動作確認が可能 ⇒①ドライバ、アプリソフトの開発 ②実デバイスを接続しての動作確認	環境整備（ボード、ドライバ、ソフト等）の準備に時間を掛かる コンパイルに時間を要する
フォーマル	◎	検証環境が不要（①、③、④） 全検証（網羅検証）が可能 ⇒①コネクティビティ（接続） ②プロパティ（インタフェース周辺） ③等価検証 ④オートチェック	収束のための抽象化、モデル置き換え等の負荷次第ではダイナミック検証よりハードルが高くなる可能性あり

スケジュールとリソース、コストの両立（5/6）

フォーマル検証をどう活用できるかがポイント
（ダイナミック検証とフォーマル検証のすみ分け）



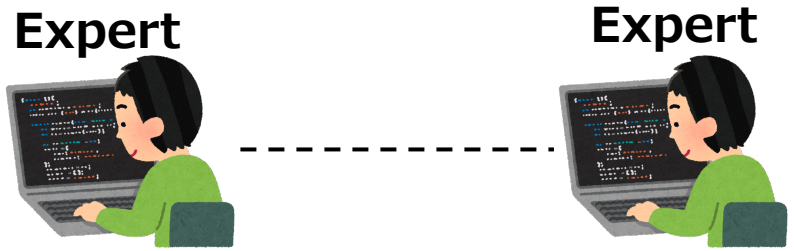
フォーマル検証対象
①ブロック間結線
②外部インタフェースブロック
③演算処理部

その他をダイナミック検証対象
①単体、SubSystem、Top単位で実施
②Topはシミュレーション、エミュレーションを活用
ソフトウェア開発をFPGAで実施

スケジュールとリソース、コストの両立（6/6）

2. 担当者のスキル問題

理想（全ての工程でスペシャリスト）



現実（皆がスペシャリストではない）

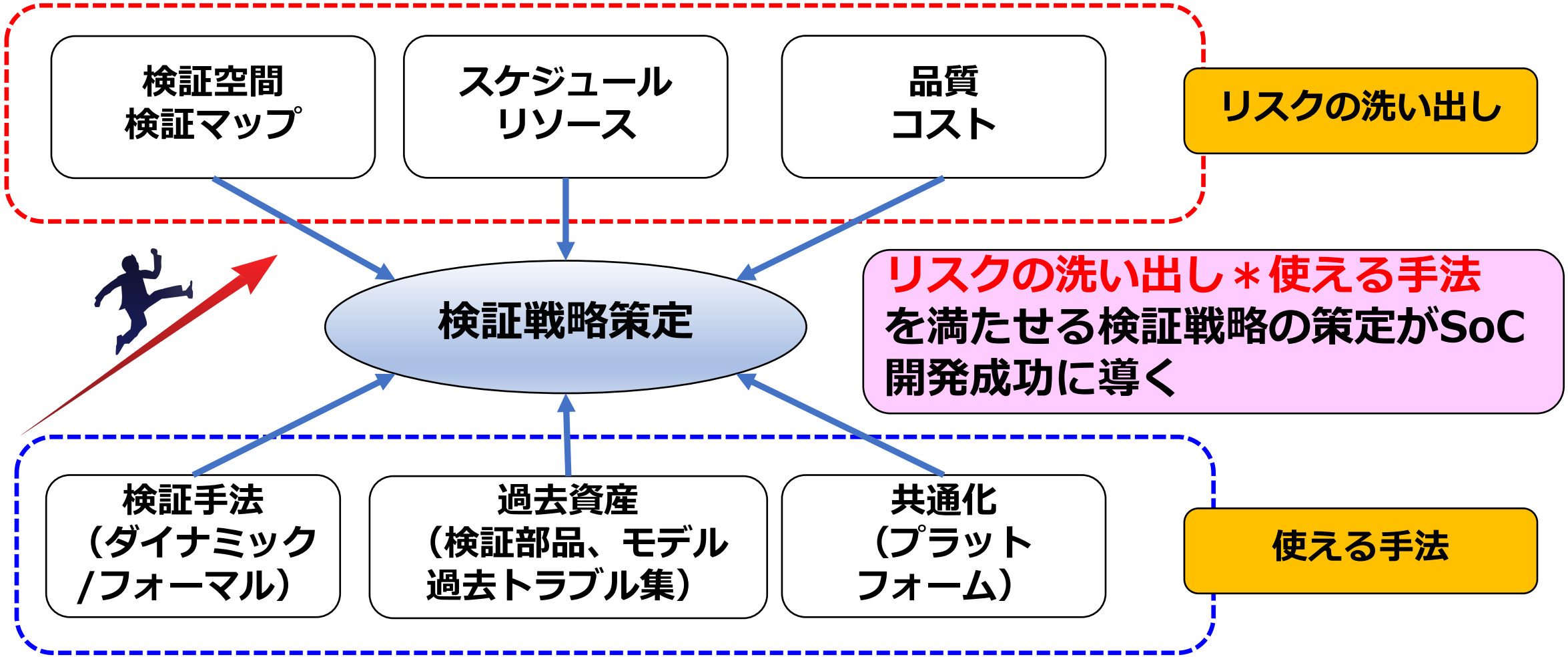
設計チーム	仕様定義	アーキテクチャ設計、論理実装	製造へ
検証チーム		ダイナミック検証 フォーマル検証	
実装チーム		合成、ビルド環境	実装設計、タイミング検証

XXX部は
外部委託

自社だけで拘らない⇒最適なSoC開発チームを構築

TEAMWORK

4. SoC開発を成功させる検証戦略とは？



5. まとめ

ポイントは以下の3点

1. **全体検証戦略をトップダウンで定義し、各階層の検証戦略へ展開**
2. **検証マップ X 検証空間で検証範囲を定義し、検証漏れを撲滅**
3. **検証手法とリソース配置の最適化**

SoC一発完動が当たり前に・・・

リスクの洗い出し、解決策を検証戦略に折り込み、リスクマネージメントを徹底し、SoC開発を成功させましょう！

質疑応答

ご清聴ありがとうございました。
ご質問があればお願いします。

後日、本プレゼンテーションの内容についてご質問があれば
下記まで問合せ願います。

CMエンジニアリング株式会社　二見　誠一
futami.seiichi@cmengineering.co.jp

CMエンジニアリング株式会社