2023
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
JAPAN

効果的なフォーマルテストベンチの構築

Synopsys, Inc.

SYNOPSYS®

accellera
SYSTEMS INITIATIVE

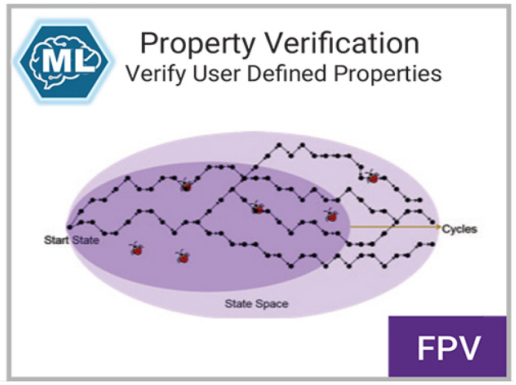
内容

- フォーマルサインオフのフロー
- フォーマルテストベンチの重要性
- フォーマルテストベンチ構築の注意点
  - テストプラン
  - プロパティのタイプ
- 未収束プロパティへの対応

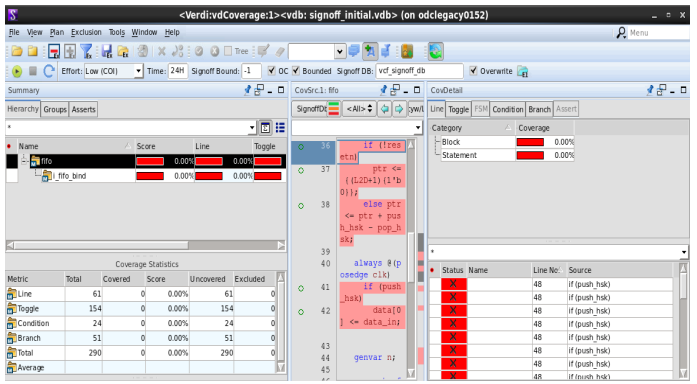
フォーマルサインオフのフロー

VC Formal ＆Verdi

アサーション足りてる？	COI Analysis
過制約による見逃しはない？	Overconstraint Analysis
サイクル数は妥当？	Bounded Coverage
実際に検証された部分は？	Formal Core Coverage
潜在バグを見つけられる環境？	Fault Injection Analysis



Signoff Dashboard



全てのカバレッジ結果を俯瞰的に確認

Coverage Database

COI: Cone Of Influence ロジックコーン

フォーマルサインオフ

- 検証対象のブロック全体をフォーマルで検証可能な場合
  - フォーマルサインオフのフローは効果的
  - クライテリアの例
    - No falsified, No vacuous
    - 100% COI Coverage, 100% Core Coverage
    - 0 non-detected faults
  - 全てのカバレッジ結果を俯瞰的、一元的に確認する機能が必要
- フォーマルとシミュレーションで補完させて検証を完了させる場合
  - カバレッジ結果のマージが必要になる
    - コア解析のレベル、若しくはFault Injection解析のレベル
- シミュレーションがメインの検証、フォーマルはバグハンティング目的で適用する場合
  - フォーマルサインオフは限定的

フォーマルテストベンチの重要性

- ブロックによってはフォーマル検証のみで完了させることが現実的に
  - 付け焼刃的にアサーション記述をしていては、全体を見渡せずフォーマルでサインオフすることが出来ない
  - 最初から全体を網羅するにはどのような検証が良いかの戦略が必要
- 機能仕様書に基づいて、何が要求されるのかを把握して検証戦略を立てる
  - デザインの動作を基にアサーション記述しない　→　検証にならない
- 複雑なシーケンス演算子などは避けて、モデリングロジックを活用することが必要
  - シンプルな記述は収束性やデバッグ容易性の面でも有効
  - シミュレーション同様にテストベンチの構成が重要

フォーマルテストベンチ構築の注意点

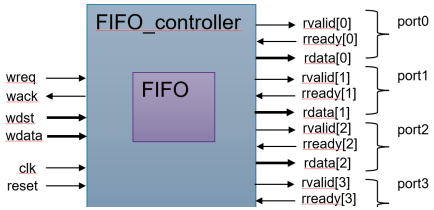
- テストプラン
  - 機能仕様に基づいて作成
  - 機能を網羅しているか
  - フォーマルサインオフを意識したプランになっているか

- プロパティ記述
  - 極力シンプルな記述
    - 複雑なシーケンス演算子を避け、モデリングロジックを活用
  - フォーマル検証におけるアサーション記述や制約記述
    - 状態数を極力増やさない
  - シミュレーションのアサーションとフォーマル検証の制約で共用可能な記述

テストプラン

- 機能仕様書から検証項目を抽出して、仕様と項目の対応付け

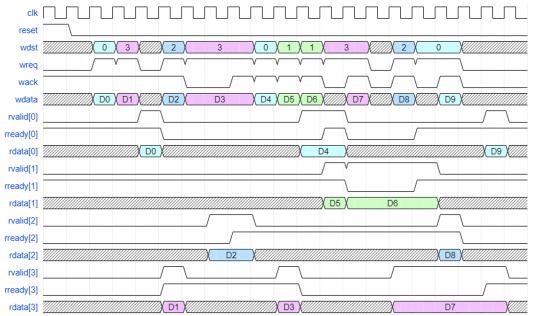
- Parameter and port declaration
- Functional specification
  - FIFO_controller is 1-in-N-out data router (N=PORTS)
  - Data on **wdata** port is stored and transferred to **rdata**[destination] port
  - Destination is specified by **wdst** port
  - Next page for the detail



Parameters		
Name	Description	Default value
WIDTH	Data bit width	16
PORTS	Number of ports	4
SELWD	wdst bit width	$clog2(PORTS)
DEPTH	FIFO depth	16

Ports			
Name	Dir.	Bit width	Description
clk	In	1	clock
reset	In	1	Reset - active High
wdst	In	SELWD	Destination port
wreq	In	1	Write request
wack	Out	1	Write acknowledge
wdata	In	WIDTH	Write data to be stored
rvalid	Out	PORTS	Read valid
rready	In	PORTS	Read ready

- Functional specification
  1. Data on **wdata** port is stored when both **wreq** and **wack** are High
  2. Port **wdst** indicates destination port of data
     - For example, If **wdst**[1:0]==2'b00, **wdata** transferred to **rdata**[0], if **wdst**[1:0]==2'b01, to **rdata**[1]
  3. Write data **wdata** should be routed to correct port based on **wdst**
  4. The order of data transfer should be guaranteed for same read port but no ordering rule between different read ports
  5. Once **wreq** is asserted, it should be asserted until **wack** is asserted
  6. Ports **wdst** and **wdata** should be stable while **wreq** is asserted until **wack** is asserted
  7. When valid read data is available for the port n, **rvalid[n]** is asserted along with **rdata[n]**
  8. Once **rvalid**[n] is asserted, it should be asserted until **rready**[n] is asserted
  9. Once **rvalid**[n] is asserted, **rdata[n]** should be stable until **rready**[n] is asserted
  10. Port **rvalid[n]** should be asserted only when valid data is available for the port n
  11. Once data is written, the data should be driven on the destined read port and **rvalid[n]** should be eventually asserted



Implementation notes:
12. Design has FIFO per read port
13. When FIFO is empty, pop data from FIFO should not occur
14. When FIFO is full, push data to FIFO should not occur unless pop occurs at the same cycle

- Functional specification
  1. Data on **wdata** port is stored when both **wreq** and **wack** are High
  2. Port **wdst** indicates destination port of data
     - For example, If **wdst**[1:0]==2'b00, **wdata** transferred to **rdata**[0], if **wdst**[1:0]==2'b01, to **rdata**[1]
  3. Write data **wdata** should be routed to correct port based on **wdst**
  4. The order of data transfer should be guaranteed for same read port but no ordering rule between different read port
  5. Once **wreq** is asserted, it should be asserted until **wack** is asserted
  6. Ports **wdst** and **wdata** should be stable while **wreq** is asserted until **wack** is asserted
  7. When valid read data is available, **rvalid[n]** is asserted along with **rdata[n]**
  8. Once **rvalid**[n] is asserted, it should be asserted until **rready**[n] is asserted
  9. Once **rvalid**[n] is asserted, **rdata[n]** should be stable until **rready**[n] is asserted
  10. Port **rvalid[n]** should be asserted only when valid data is written
  11. Once data is written, data should be driven on read port, **rvalid[n]** should be eventually asserted
- Implementation notes:
  12. Design has FIFO per read port
  13. When FIFO is empty, pop data from FIFO should not occur
  14. When FIFO is full, push data to FIFO should not occur unless pop occurs at the same cycle

Category	Description	Type	Spec Reference
Data transfer (Data Integrity, Latency, In to Out)	**wdata** should be routed to correct port based on **wdst** value	assert	#3
	The order of data transfer should be guaranteed on same read port	assert	#4
	rvalid[n] should be asserted only when there's outstanding transaction for this port	assert	#11
	wdata is written, it should be eventually driven on **rdata** port	assert	#12
	Indicate **wdata** is written for all read ports or not	cover	#1, #2
	Indicate **rdata** is driven in all read ports or not	cover	#7, #11
	How many transactions are outstanding in each port	cover	
Interface handshake	Once **wreq** is asserted, it should be asserted until handshake completes (**wack** is returned)	assume	#5
	Once **wreq** is asserted, **wdst** should be stable until handshake completes (**wack** is returned)	assume	#6
	Once **wreq** is asserted, **wdata** should be stable until handshake completes (**wack** is returned)	assume	#6
	Once **rvalid[n]** is asserted, it should be asserted until handshake completes (**rready[n]** is returned)	assert	#8
	Once **rvalid[n]** is asserted, **rdata[n]** should be stable until handshake completes (**rready[n]** is returned)	assert	#9
	Combination of handshake signals such as ~wreq & ~wack, ~wreq & wack, wreq, ~wack and wreq & wack, ~rvalid[n] & ~rready[n], rvalid[n] & ~rready[n], ~rvalid[n] & rready[n] and rvalid[n] & rready[n]	cover	#1, #8
FIFO Overflow	If Full signal is asserted, push should not be issued unless pop is asserted	assert	#14
FIFO Full condition	Indicate FIFO Full case occurs or not (# of outstanding data reaches to FIFO depth)	cover	#14
FIFO Underflow	If Empty signal is asserted, pop should not be issued	assert	#13

フォーマル検証におけるプロパティ記述

- フォーマル検証向きのアサーション
  - 任意に選択される特定の転送に着目したアサーション記述

- フォーマル検証における制約記述
  - フォーマル検証では、制約記述の方がアサーション記述よりも難しい
  - 複雑な制約は収束性に悪影響を及ぼす
  - 過制約はバグを見逃すことにつながる
  - できれば同一の検証項目に対して複数のプロパティを記述したくない
    - シミュレーション用アサーション
    - フォーマル検証用アサーション
    - フォーマル検証用制約

フォーマル検証向きのアサーション

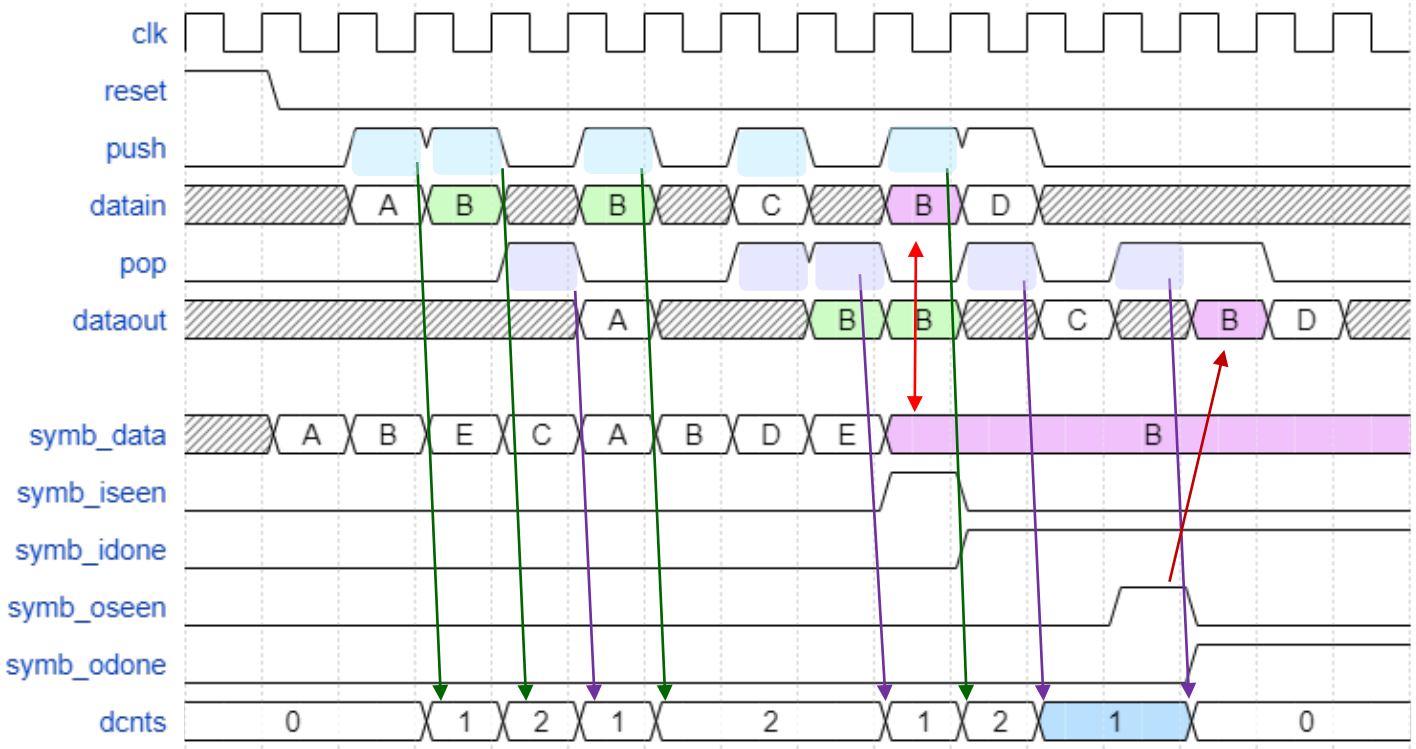
- フォーマル検証では収束性の問題がつきまとう
- 効率的に検証を行う記述が望ましい

- 任意に選択される特定の転送に着目してチェック
  - ある一つの転送のみをチェック
  - 着目した転送は全ての転送を含む（表すことが出来る）

- 検証対象デザインの状態は網羅的に検証
- プロパティ記述により生じる状態数の増加を最小限に

FIFO検証における記述例

- 全ての値をとり得る変数を使用してある特定のデータに着目
  - 着目したデータが入力された時に、期待される出力タイミングを追跡



symb_iseen:
push & ~symb_idone & datain==symb_data
symb_idone:
symb_iseenでHigh
symb_oseen:
pop & symb_idone & ~symb_odone & dcnts==1
symb_odone:
symb_oseenでHigh
dcnts:
(push & ~symb_idone) で +1
(pop & ~symb_odone) で -1

Assume: symb_idone |-> $stable(symb_data)
Assert: symb_oseen |=> dataout==symb_data

フォーマル検証におけるプロパティ記述

- フォーマル検証では、制約記述の方がアサーション記述よりも難しい
  - 制約記述の難しさ
    - 複雑な制約は収束性に悪影響を与える
    - 過制約はバグを見逃す
  - プロパティ記述の仕方は様々
    - 記述によって制約として使えるかどうかが決まる

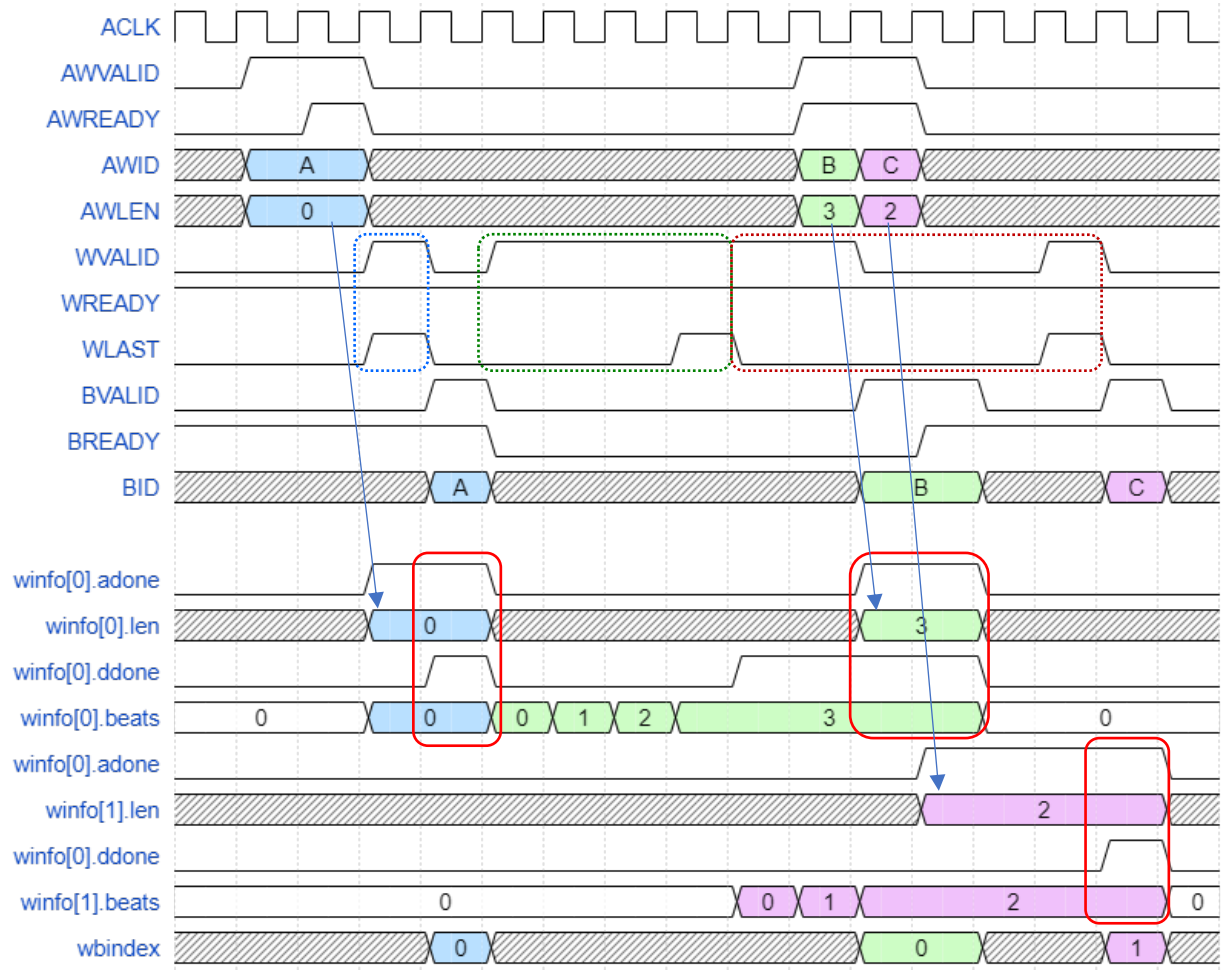
記述		フォーマル検証		シミュレーション（アサーション）
		制約	アサーション	
1	リアルタイムのチェックでない場合や長いサイクル数を要するチェック	×	△	○
2	特定の転送に着目（フォーマル検証向き記述）	×	○	×
3	リアルタイムに全ての転送をチェック	○	○	○
4	仮定に基づく動作記述	○	×	×

AXIのライト転送に関する記述

- 難しさ:
  - ライトデータがライトアドレスよりも先に転送されることがある

  - 記述例１:チェックするタイミングを遅らせてライトレスポンスの時にチェック
    - シミュレーションで使用
    - フォーマル検証でもアサーションとしては使用可能だが余計なサイクル数を要する
    - フォーマル検証の制約としては使えない
  - 記述例２:特定の転送に着目してチェック
    - フォーマル検証のアサーションとしてのみ使用可能
  - 記述例３:転送進行時にリアルタイムでチェック
    - フォーマル検証の制約とアサーションの両方で使用可能
    - シミュレーションでも使用可能
  - 記述例４:データ転送開始時に仮想的なアドレスチャネル情報を使用
    - フォーマル検証の制約としてのみ使用可能

ライトレスポンスの時にチェックする記述



アドレスチャネル終了時にAWLENをwinfor[n].lenに保存
データチャネルでのデータ転送数をwinfo[n].beatsに保存
ライトレスポンス時にwinfo[n].lenとwinfo[n].beatsを比較

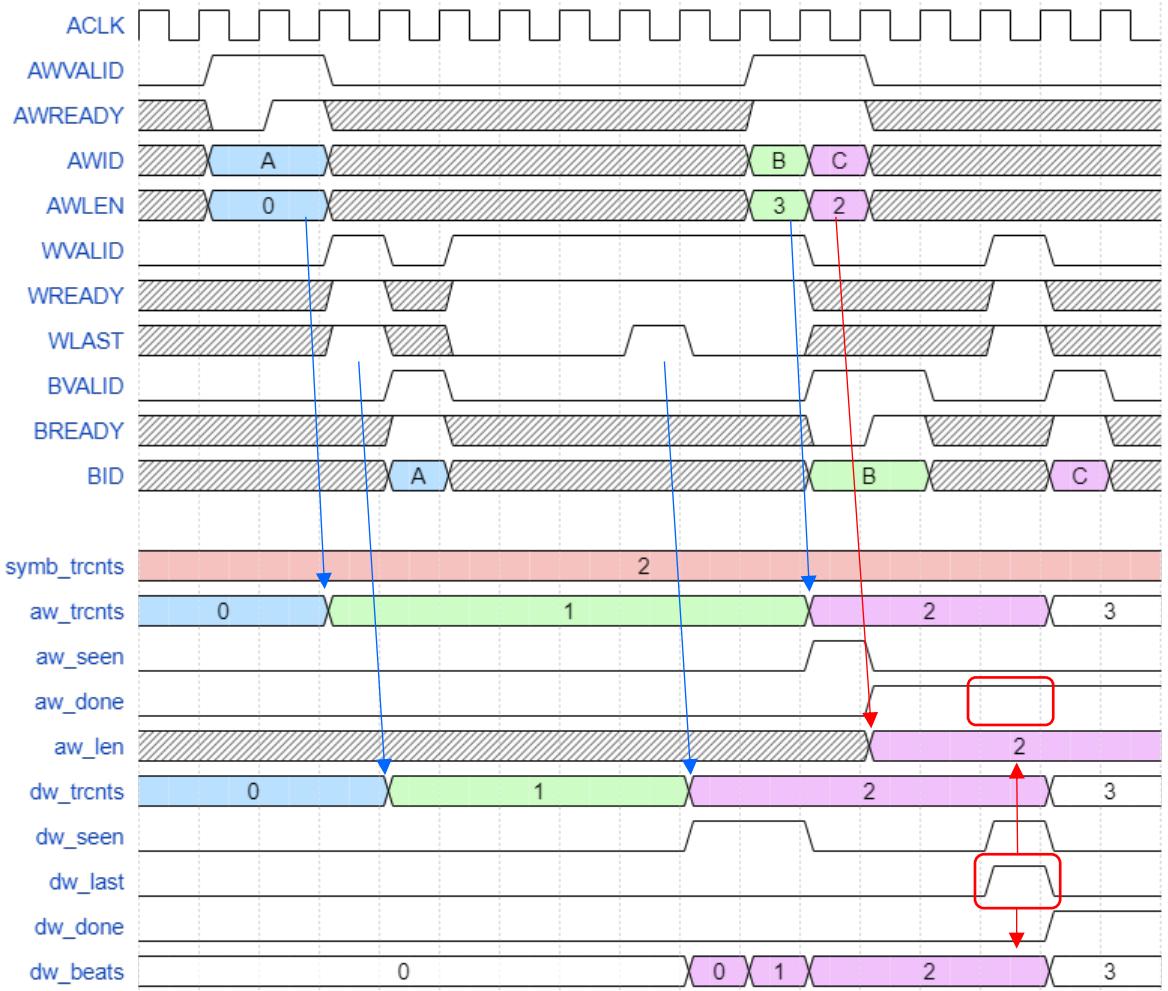
winfo[n].len:
AWVALID && AWREADYでAWLENを保存
winfo[n].beats:
WVALID && WREADY && !WLASTで+1
BVALID && BREADYで0

```
BVALID |->
 (winfo[wbindex].adone &&
  winfo[wbindex].ddone &&
  winfo[wbindex].len == winfo[wbindex].beats)
```

記述例1

フォーマル検証の制約としては使えない

特定の転送に着目してチェック



aw_trcnts	何番目のライトアドレスかを示す
aw_seen	検証対象のライトアドレス開始したことを示す
aw_done	検証対象のライトアドレスを観測終了したことを示す
aw_len	検証対象のライトアドレスのAWLENを保存
dw_trcnts	何番目のライトデータかを示す
dw_seen	検証対象のライトデータが転送中であることを示す
dw_last	検証対象のライトデータの最後のビートであることを示す
dw_done	検証対象のライトデータが終了したことを示す
dw_beats	検証対象のライトデータの何ビート目を転送しているかを示す

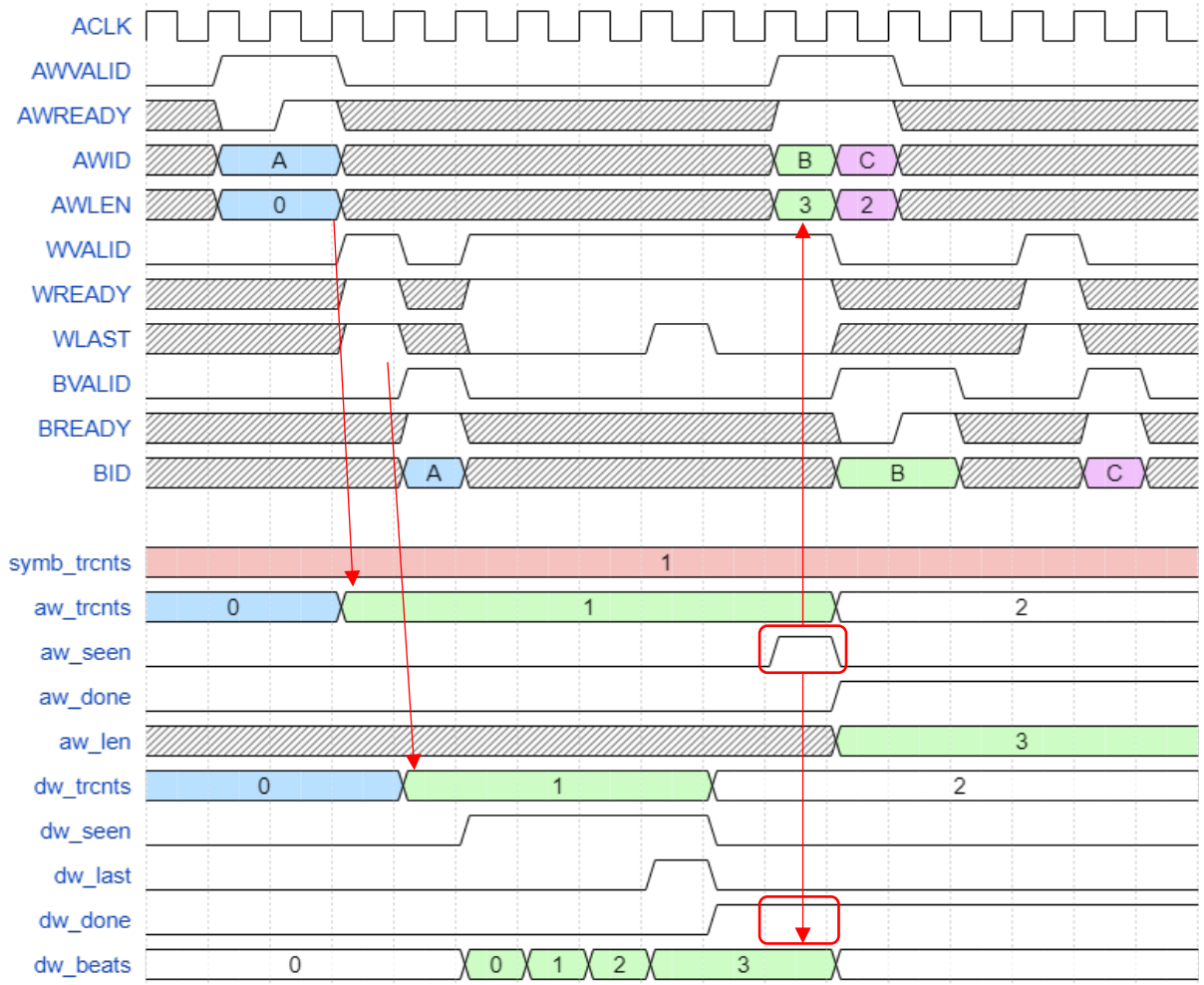
ライトデータの最後のビート転送時に既にアドレスチャネル完了:（左記A, C）
ライトデータの最後の転送時（dw_last==1）に保存したAWLENの値とデータ転送数を比較
(aw_done && dw_last) -> (aw_len==dw_beats)

AWVALIDがアサートされた時に既にデータチャネル終了:（左記B）
ライトアドレスの転送時（aw_seen==1）にAWLENの値とデータ転送数を比較
(aw_seen && dw_done) -> (AWLEN==dw_beats)

アドレスチャネルと該当するデータチャネルが同時に進行:
AWLENの値とデータ転送数が同じ場合はWLASTがHigh、異なる場合はLow
(aw_seen && dw_seen) -> ((AWLEN==dw_beats)==WLAST)

記述例2　フォーマル検証のアサーションとしてのみ使用可能

特定の転送に着目してチェック



aw_trcnts	何番目のライトアドレスかを示す
aw_seen	検証対象のライトアドレス開始したことを示す
aw_done	検証対象のライトアドレスを観測終了したことを示す
aw_len	検証対象のライトアドレスのAWLENを保存
dw_trcnts	何番目のライトデータかを示す
dw_seen	検証対象のライトデータが転送中であることを示す
dw_last	検証対象のライトデータの最後のビートであることを示す
dw_done	検証対象のライトデータが終了したことを示す
dw_beats	検証対象のライトデータの何ビート目を転送しているかを示す

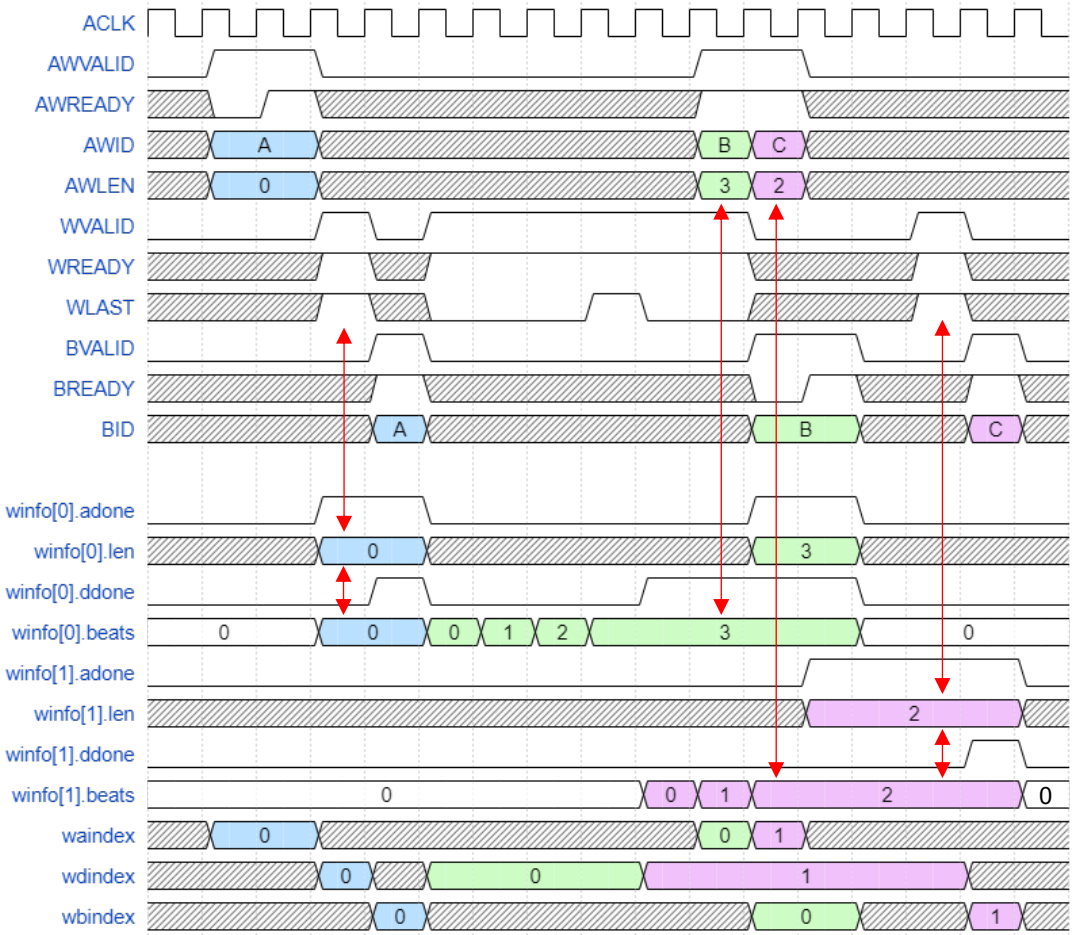
ライトデータの最後のビート転送時に既にアドレスチャネル完了:（左記A, C）
ライトデータの最後の転送時（dw_last==1）に保存したAWLENの値とデータ転送数を比較
(aw_done && dw_last) -> (aw_len==dw_beats)

AWVALIDがアサートされた時に既にデータチャネル終了:（左記B）
ライトアドレスの転送時（aw_seen==1）にAWLENの値とデータ転送数を比較
(aw_seen && dw_done) -> (AWLEN==dw_beats)

アドレスチャネルと該当するデータチャネルが同時に進行:
AWLENの値とデータ転送数が同じ場合はWLASTがHigh、異なる場合はLow
(aw_seen && dw_seen) -> ((AWLEN==dw_beats)==WLAST)

記述例2　フォーマル検証のアサーションとしてのみ使用可能

転送進行時にリアルタイムでチェックする記述



WVALIDがアサートされた時にアドレスチャネル終了:（左記A）
その時のビート数と保存したAWLENの値が等しい場合はWLASTをアサート
AELENの値より小さい場合はWLASTをアサートしない
(WVALID && winfo[wdindex].adone) ->
((winfo[wdindex].len==winfo[wdindex].beats)==WLAST)

アドレスチャネルと該当するデータチャネルが同時進行中:
その時点での転送数はAWLEN以下
(WVALID && AWVALID && waindex==wdindex) && ((AWLEN==winfo[wdindex].beats)==WLAST)

WVALIDがアサートされた時にアドレスチャネル未開始:（左記B,C）
その時のビート数は最大転送数以下
(WVALID && !winfo[wdindex].adone) -> (winfo[wdindex].beats <= MAX_BURST_LENGTH)

AWVALIDがアサートされた時にデータチャネル終了:（左記B）
AWLENの値は転送数に一致
(AWVALID && winfo[waindex].ddone) -> (AWLEN==winfo[waindex].beats)

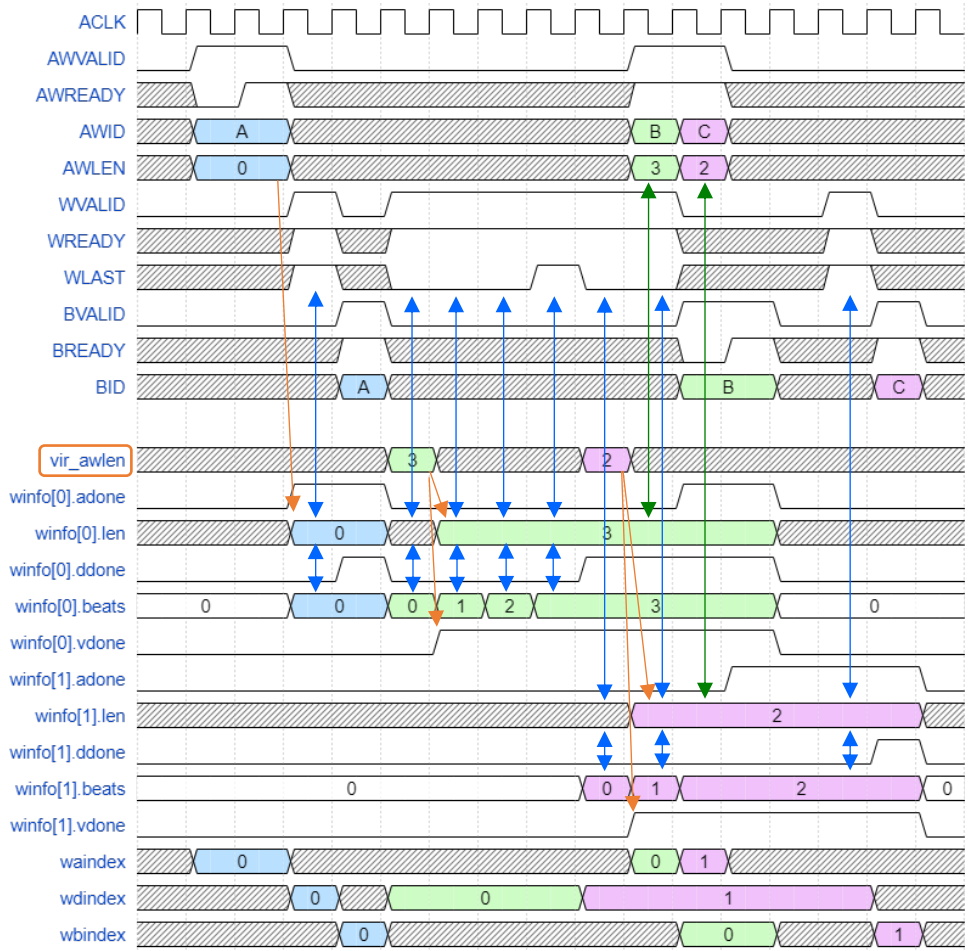
AWVALIDがアサートされた時にデータチャネル進行中:（左記C）
AWLENの値はその時点での転送数以上
(AWVALID && !winfo[waindex].ddone && winfo[waindex].beats>0) ->
(AWLEN>=winfo[waindex].beats)

winfo[n].len:
AWVALID && AWREADYでAWLENを保存
winfo[n].beats:
WVALID && WREADY && !WLASTで+1
BVALID && BREADYで0

記述例3

フォーマル検証の制約としてもシミュレーションと
フォーマルのアサーションとしても使用可能

データ先行時に仮定情報を基にチェックする記述



フォーマル検証の制約としてのみ使用可能

記述例4

ライトアドレスが先行またはアドレスとデータが同時の場合、実際のアドレス情報
ライトデータが先行した場合、１ビート目でアドレスチャネルの情報を仮定

最初のWVALIDがアサートされた時にアドレスチャネル終了:（左記A）
その時のビート数と保存したAWLENの値が等しい場合はWLASTをアサート
AWLENの値より小さい場合はWLASTをアサートしない
(WVALID && winfo[wdindex].adone) ->
((winfo[wdindex].len==winfo[wdindex].beats)==WLAST)

1ビート目のWVALIDがアサートされた時にAWVALIDもアサート開始:
AWLEN==0ならWLASTをアサート、AWLEN>0ならWLASTはアサートしない
(AWVALID && WVALID && winfo[wdindex].beats==0 &&waindex==wdindex) ->
((AWLEN==0)==WLAST)

2ビート目以降のWVALIDがアサートされた時:
その時のビート数と保存したAWLENの値を比較
(WVALID && winfo[wdindex].beats>0) ->
((winfo[wdindex].len==winfo[wdindex].beats)==WLAST)

1ビート目のWVALIDがアサートされた時にアドレスチャネル未開始:（左記B,C）
仮想のAWLENを使用し、WLASTがアサートされている場合は、仮想AWLEN==0に制約
WLASTがアサートされてない場合は、仮想AWLEN>0に制約
(WVALID && winfo[wdindex].beats==0 && !winfo[wdindex].adone) -> ((vir_awlen==0)==WLAST)
winfo[wdindex].lenに仮想AWLENの値を保存

AWVALIDがアサートされた時にデータチャネル進行中か終了:（左記B,C）
AWLENの値を仮想AWLENに一致するよう制約
(AWVALID && winfo[waindex].vdone) -> (AWLEN==winfo[waindex].len)

フォーマル検証における各記述例の考察

記述例		コメント	ライン数
1	ライトレスポンス時にチェック	フォーマル検証での使用はあまり推奨されない ・チェックするまで余分なサイクル数が必要	104
2	特定の転送に着目	フォーマル検証でよく使われる記述 意図せず検証範囲を狭めてないか注意が必要 ・例えば、最初に合致するIDだけを検証することが無いように	78
3	リアルタイムチェック	検証手法を問わずに使用可能 WLASTのチェックでは問題ないが、WSTRBのチェックでは全てのビートの値を保存する必要があり、FF数が増大し収束性に影響を与える	153
4	仮想情報で制約	WSTRBのチェックでも値を保存する必要がないのでFF数を増やさない	177

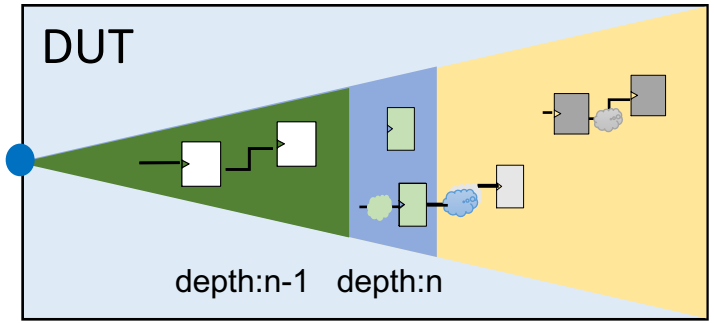
未収束プロパティへの対応

- 未収束を発生させている要素の特定
  - 検証コアの進捗状況を解析
  - どの要素がどの程度の割合で影響を与えているかをレポート
  - レポートからユーザーが判断
- 未収束を発生させている要素への対応
  - 進化するフォーマルエンジンやオーケストレーションの活用
  - 抽象化
    - カットポイントやブラックボックス
    - デザイン中の複雑な論理の抽象化
    - 抽象化モデルによる置換
  - プロパティ記述を変更
  - ヘルパーアサーションの追加
  - 検証戦略の見直し

未収束を発生させている要素の特定

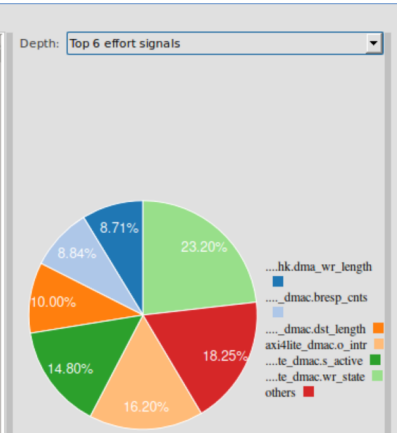
- 従来はCOIの構造的解析やコア解析によるレポートがメイン
- シーケンシャルな要素の段数の深さ毎のコア解析から、どの部分が進捗に影響を与えているかをレポート

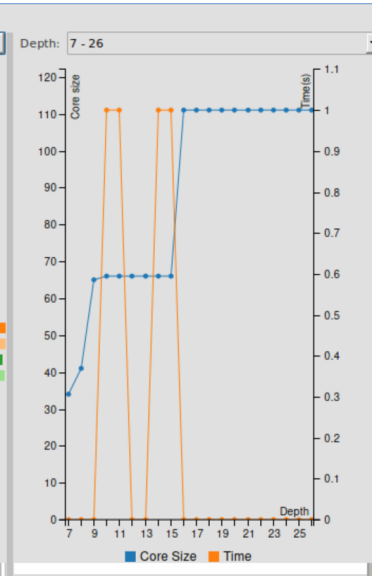
  ⇒収束を困難にさせているブロックや信号線の特定



Property: (Verbose) {axi4lite_dmac.axi4lite_dmac_chk.ast_interrupt_rose_only_dma_dor Engine: deepest

Scope	Effort	Type	Signal	Notes
axi4lite_dmac.axi4lite_dmac_chk.ast_interrupt_rose_only_dma_done				[1001] Confirm co...
Top 6 effort signals				
	23.2%		axi4lite_dmac.wr_state	
	16.2%		axi4lite_dmac.o_intr	
	14.8%		axi4lite_dmac.s_active	
	10%		axi4lite_dmac.dst_length	
	8.84%		axi4lite_dmac.bresp_cnts	
	8.71%		axi4lite_dmac.axi4lite_dmac_chk.dma_wr_length	
Depth: 326 (46s)				
axi4lite_dmac	84%			
	23.2%	Register	wr_state	
	16.2%	Register	o_intr	
	14.8%	Register	s_active	
	10%	Register	dst_length	
	8.84%	Register	bresp_cnts	
	4.75%	Register	set_length	
	2.77%	Register	o_pslverr	
	1.98%	Register	intr_en	
	1.45%	Register	o_pready	
		Gate	BREADY	
		Gate	PREADY	
		Gate	PSLVERR	
		Gate	addr_decerr	
		Gate	inp_slverr	
		Gate	fifo_empty	
		Gate	apb_write	
		Gate	dma_done	
		Gate	intr_clr	
		Gate	kick_dma	
		Gate	CLK	
		Gate	INTR	
		Gate	RSTN	
		Input	AWREADY	





まとめ

- フォーマル検証のみでブロック検証を完了させる現実性
  - フォーマルサインオフのフローでカバレッジ解析が必須

- フォーマルテストベンチの構築の仕方で検証の成否が決まる
  - 検証戦略と効率的なプロパティ記述が必要
  - 収束性を意識した記述を心がけることが重要

- 未収束プロパティの対応は難しい
  - 新たなフォーマル技術の活用が必須
    - 進化するフォーマルエンジンやオーケストレーション
    - 収束を困難にさせているブロックや信号線を特定