VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV MIKROELEKTRONIKY
DEPARTMENT OF MICROELECTRONICS

MODERNÍ METODY VERIFIKACE SMÍŠENÝCH INTEGROVANÝCH OBVODŮ
MODERN METHODS OF MIXED-SIGNAL INTEGRATED CIRCUIT VERIFICATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Jakub Podzemný**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. Lukáš Fujcik, Ph.D.**
SUPERVISOR

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Mikroelektronika**
Ústav mikroelektroniky

Student: Bc. Jakub Podzemný ***ID:*** 164365
Ročník: 2 ***Akademický rok:*** 2018/19

NÁZEV TÉMATU:

Moderní metody verifikace smíšených integrovaných obvodů

POKYNY PRO VYPRACOVÁNÍ:

Zpracujte přehled moderních metod verifikace integrovaných obvodů pracujících ve smíšeném módu. Porovnejte metody kontroly parametrů a funkčnosti simulovaného obvodu a posuďte vhodnost jednotlivých přístupů pro konkrétní situace. Zaměřte se na „assertion based methodology“. Ve vybraném jazyce vytvořte knihovnu parametrizovaných asertů, které umožní verifikaci obvodu simulovaného na různých úrovních abstrakce a různých reprezentacích jednotlivých bloků. Vytvořenou knihovnu ověřte na vybraném obvodu firmy ON Semiconductor.

DOPORUČENÁ LITERATURA:

[1] Chen J., Henrie M., Mar M. F., Nizic M., „Mixed-Signal Methodology Guide“, Cadence Design Systems, Inc., August 2012, ISBN:978-1-300-03520-6

Termín zadání: 4.2.2019 ***Termín odevzdání:*** 21.5.2019

Vedoucí práce: doc. Ing. Lukáš Fujcik, Ph.D.
Konzultant: Ing. Martin Kejhar, ON Semiconductor, Rožnov pod Radhoštěm

doc. Ing. Lukáš Fujcik, Ph.D.
předseda oborové rady

Abstrakt:

Práce se zabývá metodami, které jsou vhodné pro verifikaci smíšených integrovaných obvodů. Důraz je přitom kladen na tzv. „Assertion-based“ verifikaci. Tato metoda je v praxi aplikovatelná pomocí jazyků PSL a SystemVerilog.

Tyto jazyky jsou mezi sebou porovnány a samostatně otestovány, aby byl následně stanoven jejich potenciál a aby byly nalezeny jejich funkční hranice a omezení. Jeden z těchto jazyků bude následně začleněn do verifikačních postupů společnosti SCG Czech Design Center s. r. o., aby zde mohla být rozvinuta metoda ABV i v analogové a smíšené doméně.

Abstract:

This work aims at methods, which are suitable for mixed-signal integrated circuit verification. The emphasis is on the Assertion-based verification. In practice there are two languages, which can be used for this method - PSL and SystemVerilog.

These languages are compared between each other and individually tested to find their capabilities, functional limits and restrictions. One of them will be integrated into verification flow of SCG Czech Design Center s. r. o. company to develop ABV methodology in analog and mixed-signal domain.

Klíčová slova:

Verifikace, smíšené obvody, ABV, PSL, SVA

Keywords:

Verification, mixed-signal circuits, ABV, PSL, SVA

Bibliografická citace:

PODZEMNÝ, Jakub. *Moderní metody verifikace smíšených integrovaných obvodů.* Brno, 2019. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/115734. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav mikroelektroniky. Vedoucí práce Lukáš Fujcik.

Prohlášení autora o původnosti díla:

Prohlašuji, že svou diplomovou práci na téma Moderní metody verifikace smíšených integrovaných obvodů jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne ....

podpis autora

Poděkování:

Na tomto místě bych rád poděkoval panu Ing. Martinu Kejharovi CSc., za odborné vedení, podnětné konzultace, ochotu a cenné rady a znalosti, které mi během tvorby diplomové práce poskytl. Dále bych rád poděkoval společnosti SCG Czech Design Center s. r. o. za poskytnuté téma diplomové práce a za spolupráci a pomoc při její tvorbě.

SIX research centre
sensor, information and communication systems

Faculty of Electrical Engineering
and Communication

Brno University of Technology
Technicka 12, CZ-61600 Brno, Czechia

http://www.six.feec.vutbr.cz

Experimentální část této diplomové práce byla realizována na výzkumné infrastruktuře vybudované v rámci projektu CZ.1.05/2.1.00/03.0072
Centrum senzorických, informačních a komunikačních systémů (SIX)
operačního programu Výzkum a vývoj pro inovace.

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

OP Výzkum a vývoj pro inovace

Obsah:

Úvod

Verifikace smíšených obvodů a systémů je s postupem času stále náročnější úlohou. S rostoucí komplexností obvodů je vyvíjen větší tlak na verifikační inženýry, kteří už nemohou spoléhat na tzv. „divide and conquer“ (rozděl a panuj) verifikační přístup, kdy jsou digitální a analogové bloky verifikovány zvlášť, aby byly následně sloučeny pro systémovou verifikaci. Místo toho musí provést množství simulací ve smíšené doméně (na systémové i blokové úrovni), aby měli jistotu, že návrh neobsahuje žádné chyby, které by se týkaly mezidoménových interakcí.

Při velikosti dnešních systémů a komplexnosti bloků se ale doba běhu takových simulací pohybuje v řádu hodin i dnů. Tomu následně odpovídá i objem dat s výsledky. Takové simulační výsledky mohou dosahovat velikosti i stovek gigabajtů. Při analýze takového množství výsledků se může snadno, i když neúmyslně, přehlédnout nějaká nesrovnalost, která způsobí nefunkčnost obvodu. Postupem času navíc vývojář ztrácí koncentraci a v odsimulovaných křivkách vidí spíše to, co tam vidět chce. Odstranění lidského faktoru je tak v tomto ohledu velmi žádoucí.

Z těchto důvodů je vhodné zavést do verifikačního procesu metody, které zautomatizují kontrolu simulačních výsledků, a tak zkvalitní hledání chyb. Zde by mohla být vhodnou metodou tzv. „Assertion-based“ verifikace (zkráceně ABV), která podporuje oba tyto aspekty. ABV by měla zajistit automatickou kontrolu chování obvodu v průběhu každé simulace a za všech možných podmínek. Tím by se měl vyloučit lidský faktor při analýze simulačních výsledků – ať už tabulek, nebo grafů.

Metodu ABV lze využít prostřednictvím dvojice jazyků. Těmito jazyky jsou PSL a SystemVerilog. V rámci práce budou tyto jazyky porovnány a otestovány. Jeden z jazyků bude následně zvolen pro rozvinutí ABV metody v analogové a smíšené doméně v rámci verifikačních postupů společnosti SCG Czech Design Center s. r. o.

1 Smíšené integrované obvody

Za pojmem smíšený integrovaný obvod je možné si představit mnoho alternativ. Většina lidí ale bude souhlasit s tím, že se pod tímto pojmem myslí kombinace analogových a digitálních obvodů, které dohromady ve spolupráci zajišťují určitou funkci smíšeného integrovaného systému. Takovou funkcí je obecně myšleno převzetí reálné analogové veličiny (napětí, proud, teplota, atd.), která má proměnnou amplitudu v čase, a následné zpracování této informace v digitálním procesu s diskrétním časovým krokem. [1] [2]

Souhra mezi těmito dvěma doménami je nutným předpokladem z hlediska funkčnosti smíšených integrovaných obvodů. Hned několik zdrojů se pak shoduje na tom, že v současnosti je taková interakce analogových a digitálních obvodů, které společně tvoří smíšenou doménu, daleko komplexnější než v minulosti. [3] [2]

1.1 Analogové bloky

Analogové bloky se skládají ze základních aktivních a pasivních součástek, jako jsou tranzistory, diody, odpory, kondenzátory, cívky, atd. Vývojáři takových bloků postupují většinou podle analogové návrhové metodiky, ve které se běžně používá tzv. „bottom-up“ přístup – tedy odspoda nahoru. U tohoto přístupu je nutná detailní znalost základních součástek, ze kterých vývojář následně sestavuje analogový obvod. [1] [2]

Postup je takový, že vývojář nejprve zjistí nezbytné informace o požadované funkci vyvíjeného bloku, a poté z jednotlivých součástek vytvoří schéma na tranzistorové úrovni. Toto schéma je následně individuálně testováno a ve skupině s dalšími modely (více o modelech níže) pak dále zaintegrován do systému. [1] [2]

Návrhovým prostředím analogového vývojáře obyčejně bývá schematický editor, kterým je vytvořen popis obvodu. Popis obvodu zahrnuje jeho topologii a propojení. V prostředí schematického editoru má vývojář také možnost vytvářet testovací zapojení, spouštět simulaci, optimalizovat obvod pro dosažení požadovaných parametrů a definovat omezení související s implementací. [1]

1.2 Digitální bloky

Digitální bloky jsou běžně sestaveny pomocí předem vytvořených podbloků, které jsou označovány jako logická hradla. Vývojář má tak možnost pracovat na vyšší úrovni abstrakce, kdy digitální obvod sestaví pomocí jednotlivých hradel.

S rostoucí velikostí digitálních obvodů jsou vývojáři nuceni pracovat na vyšší úrovni abstrakce. Metodologie návrhu digitálních obvodů je ale díky tomu automatizovatelná, což se projeví především u automatické syntézy. Navíc se díky vyšší úrovni abstrakce pro návrh digitálních bloků typicky používá tzv. „top-down" přístup – tedy odshora dolů. U této návrhové metody se nejdříve nadefinují požadavky na vyvíjený blok a následně je vytvořen funkční model na vyšší úrovni abstrakce (např. RTL zápis v jazyce HDL). Tento model se v průběhu návrhu zpřesňuje až do finální tranzistorové podoby. Tento proces je dnes automatizován (syntéza a APR). [1]

Ačkoli se způsob návrhu analogových obvodů za poslední léta posunul kupředu, nedosáhl výrazného zlepšení v oblasti analogové syntézy, což je omezení pro přijetí metody „top-down". Některé vývojové týmy ale přesto tuto metodu přijaly, a používají ji v kombinaci s „bottom-up" přístupem i pro návrh analogových bloků. [1]

1.3 Smíšené bloky / systémy

Kombinací a vyvážením výše zmíněných vývojových metod vzniká třetí, která je vhodná pro použití ve smíšené doméně. Tato metoda je tzv. „meet-in-the-middle". V této metodě jsou analogové bloky systému (případně i jednoduché logické funkce) vyvíjeny způsobem „bottom-up", zatímco jsou současně vyvíjeny rozsáhlé logické obvody způsobem „top-down". Ve vhodné fázi návrhu se jednotlivé bloky propojí a vytvoří systém smíšené domény. [2]

Pro návrh obvodů smíšené domény je možné využít způsoby obou návrhových metod. Po vzoru digitální metody je možné využít speciální modelovací jazyky (Verilog-AMS, VHDL-AMS), nebo je, stejně jako u analogové metody, možné nakreslit schéma smíšeného systému. V obou případech je ale nutné znát obvodovou hierarchii a modelovou reprezentaci systémových prvků.

2 Modelování bloků

Klíčovým prvkem pro vývoj a verifikace nejen smíšených integrovaných obvodů je behaviorální modelování dílčích bloků a podbloků. Pro digitální bloky je tato praxe běžná. Ovšem povýšení analogových a smíšených bloků na vyšší úroveň abstrakce umožní efektivnější simulace obvodů smíšené domény. [1]

To ale bohužel není jednoduchá úloha a vývojáři se musí vypořádat s řadou výzev. Především musí perfektně pochopit zaměření a účel modelu, aby byli schopni zvolit správnou modelační techniku. Při použití vývojového postupu „top-down“ jsou modely vytvořeny dříve, než samotný blok. Jednodušší (méně komplexní) modely tak bývají dostatečné pro funkční verifikaci. U přístupu „bottom-up“ musí model odpovídat již implementovanému bloku. Zde jsou proto potřeba detailnější modely. [1] [4]

Dalším aspektem je validace modelů. Modely musí být validovány samostatně, aby se potvrdilo, že reprezentují daný blok s dostatečnou přesností oproti specifikaci. Modely musí být průběžně aktualizovány, aby zůstaly v odpovídající podobě vzhledem k reprezentovaným blokům. Pokud se jedná o analogový nebo smíšený model, je potřeba jej vytvořit tak, aby během simulací nezpůsoboval problémy s konvergencí. [1]

Psaní behaviorálních modelů tak vyžaduje určitou kombinaci dovedností. Je potřeba se dobře orientovat ve vývojových metodách, stejně jako v obvodech samotných. Modely je potřeba vhodným způsobem zapsat a odladit. V neposlední řadě je také potřeba znát simulační algoritmy. [1]

2.1 Výpočetní jádra simulátorů

V současné době je na trhu možné najít množství obvodových simulátorů od různých dodavatelů, které, v závislosti na simulované doméně, dokážou vhodným způsobem odsimulovat chování vyvíjeného obvodu. Přesnost těchto simulátorů bývá ve většině případů dostačující, ale přesto je dobrou praxí při návrhu obvodu využít více než jeden simulátor a výsledky jednotlivých simulací porovnat.

2.1.1 Analogové výpočetní jádro

Jak bylo zmíněno výše, analogové veličiny jsou proměnné v čase s libovolně malou změnou amplitudy. K popisu analogové funkce jsou proto vyžadovány lineární a nelineární rovnice, které popisují vzájemné vztahy napětí a proudů. Analogový simulátor tyto rovnice rozšíří o další, které vycházejí z obvodové topologie v kombinaci s Kirchoffovými zákony. Výsledný systém rovnic je řešen pomocí Newton-Raphson iterační metody, která využívá výpočty s inverzními maticemi, aby bylo dosaženo dostatečně přesných výsledků u všech napětí a proudů v obvodu. Tyto výpočty jsou prováděny souběžně ve všech časových bodech simulace, které jsou definovány podle nastavených simulačních tolerancí. [5] [1] [4]

Analogový simulátor typicky pracuje tak, že rozdělí časovou osu na dostatečně malé intervaly, které aproximují lineární závislosti obvodu v každém kroku simulátoru. To mu dovolí odhadnout řešení rovnic, které popisují, jak by se v těchto malých intervalech měl obvod chovat. Přitom se předpokládá, že je simulátor schopen odhadnout velikosti chyb, které můžou v každém časovém intervalu nastat. Díky tomu simulátor určí velikost časového kroku, o který se může bezpečně posunout, tak aby mohl iterovat a konvergovat k výsledkům napětí a proudů, které dostatečně vyhoví Kirchoffovým zákonům v novém časovém bodě. [1]

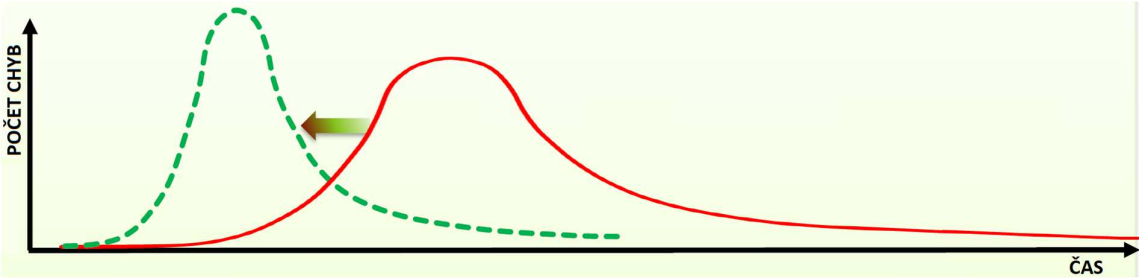
Pokud jsou ale do systému rovnic zařazeny i nelineární rovnice, není možné zaručit, že iterační algoritmus dosáhne výsledku. Každá iterace nového odhadu řešení totiž vyžaduje kompletní přepočet nelineárních vztahů systému. Pokud ale simulátor během určitého počtu iterací nenalezne dostatečně přesné řešení, tak vyhodnotí, že velikost časového kroku je vzhledem k nelinearitě systému nadměrná, výpočet daného bodu zastaví a pokusí se najít řešení systému v jiném, v čase bližším bodě. [1]

Nevýhoda iteračního algoritmu tak spočívá především v jeho složitosti a náročnosti na výpočetní výkon. Běžně také mohou nastat výše popsané problémy s konvergencí. Pokud tak simulátor nenalezne řešení zadaného systému nelineárních rovnic během zadaného počtu pokusů, automaticky to znamená ukončení probíhající simulace. Analogové výpočetní jádro se používá pro simulace bloků na tranzistorové úrovni a pro simulace modelů v jazyce Verilog-A.

2.1.2 Digitální výpočetní jádro

Chování digitálních obvodů je běžně popsáno na vyšší úrovni abstrakce pomocí vztahů Boolovy algebry. Digitální simulátory tak běží mnohem rychleji, protože výpočty jsou prováděny v diskrétní časové doméně a pouze při rekakci na určitou událost (tzv. „event-driven“ simulace) – tedy při změně logických stavů vstupních signálů bez potřeby průběžných výpočtů nelineárních rovnic. Hodnoty výstupů jsou pak určeny přímo podle vnitřních stavů bloku a podle hodnot vstupů v každém časovém bodě simulace. Digitální výpočetní jádro se používá pro simulace modelů reálných čísel a pro simulace čistě logických modelů. [1]

V porovnání s analogovými simulacemi bývají digitální rychlejší v rozsahu čtyř až šesti řádů a díky vyšší úrovni abstrakce existuje možnost zahájit verifikace na systémové úrovni v dřívější fázi návrhu – tzv. „shift left“. Navrhování obvodů na vyšší úrovni abstrakce je tak nezbytné pro vytváření rozsáhlých integrovaný obvodů. [1] [2]



Obrázek 1: Urychlení verifikačního procesu, tzv. „shift left“ [6]

2.1.3 Smíšené (AMS) výpočetní jádro

Simulace smíšených bloků vyžadují použití obou popsaných výpočetních jader, které musí pracovat paralelně a synchronně. Analogové výpočetní jádro je ale v tomto případě omezovačem, který snižuje výkonnost simulací a brání tak v dosažení lepších verifikačních výsledků. K dosažení rozumných simulačních rychlostí se proto mnoho vývojářských týmů, které se zabývají smíšenými obvody, snaží využít analogové behaviorální modelování. Modely je ale stále složitější vytvořit, integrovat a efektivně uplatnit v postupně se zmenšujících technologiích. Komplexnost návrhu, procesní odchylky a fyzikální efekty přidávají do modelu nespočet nových proměnných, které je potřeba brát v potaz. [2] [3]

Pro komplexní analogové obvody se proto stále preferují simulátory typu SPICE (Simulation Program with Integrated Circuit Emphasis), které zajišťují přesnější výpočty. To ale vyvíjí enormní tlak na poskytovatele EDA (Electronic Design Automation) nástrojů, aby byli schopni poskytnout dostatečně výkonné a zároveň přesné řešení, které splní požadavky vývojářů. Vývojové týmy smíšených obvodů tak čelí otázce, kolik času by měly investovat do vytvoření modelů a jestli se návratnost této investice vyplatí z pohledu vynaložených zdrojů. [2]

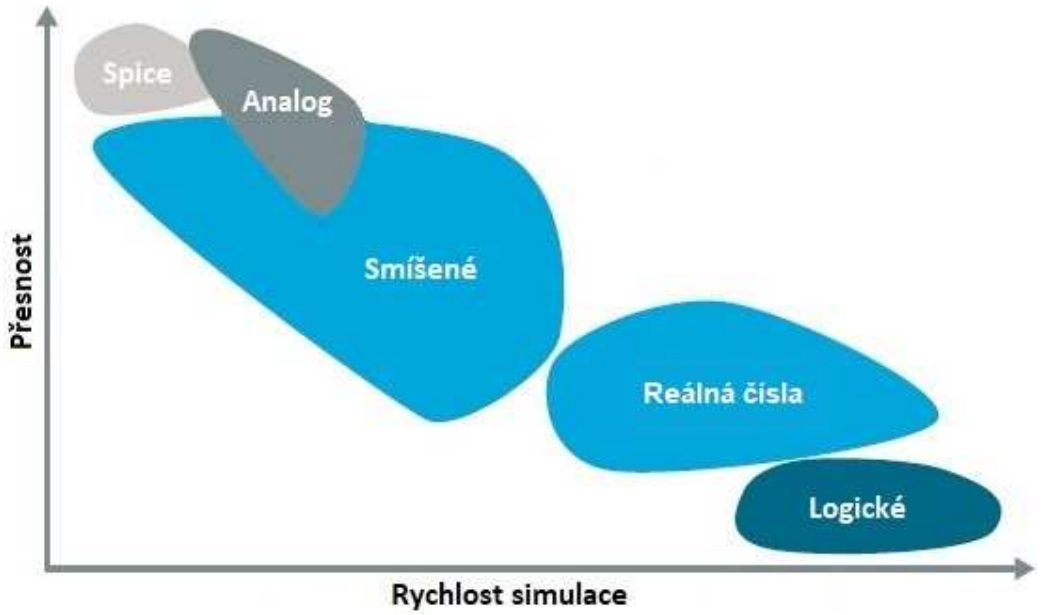
2.2 Typy modelování

Během vývoje a verifikace rozsáhlého integrovaného obvodu je běžné použít různé modelovací formáty, v závislosti na používané verifikační metodě. Mezi takové formáty řadíme:

- model na tranzistorové úrovni (spectre / spice),
- analogové modely (Verilog-A),
- smíšené modely (Verilog-AMS, VHDL-AMS),
- modely s reálnými čísly (Verilog-AMS, VHDL, SystemVerilog),
- logické modely (Verilog, VHDL, SystemVerilog).

Běžnou praxí pak je verifikovat jednotlivé systémové podbloky na tranzistorové úrovni a pro simulace na vyšší úrovni abstrakce použít jejich behaviorální modely. Dále je běžné pro určitý blok systému použít více modelovacích technik, aby měl vývojář k dispozici více různých reprezentací. Jednotlivé modely pak představují různé kompromisy mezi simulační přesností a rychlostí. [4] [1]

Pokud tyto typy modelování porovnáme z hlediska přesnosti vůči reálnému chování obvodu a z hlediska rychlosti simulace, modely na tranzistorové úrovni jsou jednoznačně nejpřesnější, ale jejich simulace běží nejpomaleji. Opačný konec spektra představují čistě logické modely. Jejich simulace běží nejrychleji, nejsou v nich ale pokryty veškeré jevy, které se v obvodu dějí. V grafické podobě vypadá tento kompromis mezi přesností a rychlostí simulace následovně:



Obrázek 2: Porovnání přesnosti modelu s rychlostí simulace [4] [1]

Modely různých bloků, s výjimkou schémat tranzistorové úrovně, mají textovou podobu a jsou běžně psány v jazycích Verilog, Verilog-A, Verilog-AMS, SystemVerilog, VHDL a VHDL-AMS. Pro podporu modelování analogových obvodů byl k digitálnímu Verilogu vytvořen jazyk Verilog-A. Pro modelování smíšených obvodů následně z kombinace Verilogových jazyků vznikla syntaxe Verilog-AMS. Jazyk VHDL-AMS vznikl rozšířením digitálního VHDL. Jazyk SystemVerilog, původně vytvořený pro modelování a verifikaci digitálních obvodů, je již rozšířen a podporuje modelování analogových a smíšených behaviorálních modelů pomocí reálných čísel. [4]

2.2.1 Modely na tranzistorové úrovni

Modely na tranzistorové úrovni představují nejbližší reprezentaci elektronických obvodů, které se dá dosáhnout vzhledem k jejich fyzické implementaci. Model má podobu schématu nebo textového zápisu, tzv. „netlistu“. Dílčí části modelu reprezentují konkrétní elektronické součástky. Pro vývojáře jsou k dispozici v podobě schématických symbolů nebo různých textových zápisů (spectre, SPICE, atd.), které dohromady tvoří knihovnu použitelných prvků. Tyto knihovny jsou spravovány speciální skupinou inženýrů, kteří zajišťují, aby reprezentace jednotlivých prvků co nejvíce korespondovala s reálnými parametry elektronických součástek. [4] [1]

Modely na tranzistorové úrovni se používají v případech, kdy je do simulace potřeba zahrnout i tolerance různých fyzikálních vlastností, aby se potvrdila správná funkce obvodu i při jejich nepříznivém rozptylu. Výsledky těchto simulací se považují za velmi přesné, a tedy dostatečně reprezentující finální obvod. Na druhou stranu je ale simulace rozsáhlejšího systému na tranzistorové úrovni velmi zdlouhavá. Dále jsou modely na tranzistorové úrovni používány při návrhu analogových obvodů způsobem „bottom-up“. Tranzistorový model je při této metodě po celou dobu návrhu považován za referenční pro další modely na vyšší úrovni abstrakce. [4] [1]

2.2.2 Analogové modely

Pokud simulace systému na tranzistorové úrovni trvá příliš dlouho, je možné vážně uvažovat o vývoji analogových behaviorálních modelů. Tyto behaviorální modely jsou použity pro nahrazení méně důležitých částí systému, takže se simulace celku může vykonat rychleji s relevantními výsledky sledovaných bloků. [1] [2]

Analogové modely mají textovou podobu a jsou psány v jazyce Verilog-A. Model obsahuje rovnice, které reprezentují napěťové a proudové vztahy v obvodu. Tyto vztahy jsou psány na vyšší úrovni abstrakce, než jak jsou definovány u tranzistorových modelů. Simulace analogových modelů tak probíhá zhruba 10x - 50x rychleji, než simulace tranzistorové úrovně stejného bloku. Oba typy simulací přitom zpracovává analogové výpočetní jádro. [1] [4]

2.2.3 Smíšené modely

Smíšené modely umožňují současný popis logických částí, pro které se používají diskrétní konstrukce, společně s analogovými částmi, které jsou modelovány spojitými funkcemi. Modelovacími jazyky jsou Verilog-AMS a VHDL-AMS, které umožňují přirozené modelování smíšených systémů. [1] [4]

Data a události jsou předávány mezi simulačními algoritmy, takže simulátor musí podporovat a koordinovat práci obou typů výpočetních jader, aby bylo dosaženo správných interakcí mezi rozdílnými částmi obvodu. Analogové části modelu v tomto případě zajišťují požadovanou přesnost, zatímco digitální části modelu zvyšují rychlost simulace. [4]

Míra zrychlení simulace pak závisí na množství nahrazených tranzistorových modelů. Použitím smíšených modelovacích technik je možné přesunout logické obvody z analogového výpočetního jádra do digitálního a nahrazení zbylých analogových částí modelovacími funkcemi. Tyto změny oproti tranzistorovým modelům umožní zrychlení simulace 10x až 50x. [1]

2.2.4 Modely s reálnými čísly

Modely s reálnými čísly jsou označovány zkratkou RNM (z anglického Real Number Modeling). Jedná se o speciální techniku používanou pro modelování elektrických signálů pomocí reálných čísel, kde analogovou veličinu (napětí nebo proud) nahradí časově proměnná sekvence reálných čísel - což je princip velice podobný tomu, který používají analogové simulátory. [1]

Rozdíl spočívá v tom, že analogové modely jsou definovány soustavou rovnic, které popisují vzájemné vztahy uzlových napětí a proudů. Analogový simulátor tyto rovnice rozšíří o další, vyplývající z topologických omezení, které určují Kirchoffovy zákony. Poté je celý systém rovnic řešen souběžně v každém časovém bodě, který je volen podle nastavených simulačních kritérií. [1] [5] [4]

Výhoda modelů s reálnými čísly spočívá v tom, že nejsou řešeny analogovým simulátorem, ale v diskrétní doméně. Při jejich simulaci neprobíhají žádné výpočty matic ani spojité časové operace – nejsou zde tedy problémy s konvergencí. Hodnoty výstupů jsou vyčísleny podle navzorkovaných hodnot vstupů a vnitřních stavů bloku. Časové body simulace mají specifický časový krok, nebo jsou řízeny událostmi vstupních signálů. Díky těmto skutečnostem je možné dosáhnout podobné simulační výkonnosti, jaká je u digitálních simulací. [1] [5] [4]

Modelování pomocí reálných čísel je možné pomocí jazyků SystemVerilog, VHDL a Verilog-AMS, a může být použito kdekoli, kde je úroveň abstrakce dostatečně vysoká na to, aby mohly být ignorovány obousměrné a zpětnovazební interakce, které charakterizují analogový obvod. [1] [4]

2.2.5 Logické modely

Logické (digitální) modely definují datový tok a případně jeho zpoždění. Datový tok ze vstupu na výstup je modelován ve formě čistě binárních (logických) hodnot. Analogové operace zde tak vyjadřují pouze přítomnost nebo nepřítomnost daného signálu a často jsou v těchto modelech reprezentovány formou „černých skříněk“. Pro simulaci logických modelů se používá pouze digitální výpočetní jádro a je možné je psát v jazycích Verilog, VHDL a SystemVerilog. [1] [4] [3]

Logické modely smíšených obvodů jsou běžně používány pro kontrolu správného propojení jednotlivých bloků systému. Důsledkem mnoha problémů na systémové úrovni nebývá špatná funkčnost smíšených bloků, ale chyby, jako jsou špatná polarita signálu, zaměněné signální linky, špatně užitá napájecí doména, a podobné. Logické modely dokáží tyto chyby spolehlivě detekovat. [4]

Tento modelovací přístup je velmi výkonný ve zpracování logických a časových vztahů digitálním simulátorem a běžně se využívá pro modelování čistě digitálních obvodů. V takovém případě vývojáři postupují metodou „top-down“ a právě logický model v podobě např. RTL kódu je brán jako referenční pro další modely na nižší úrovni abstrakce. [1]

3 Verifikace

Verifikace, podle amerického slovníku Merriam-Webster ([7]), znamená proces prokázání nebo potvrzení pravdy. V kontextu návrhu integrovaných obvodů, a s tím spojené hardwarové verifikace, by se tato definice dala přeformulovat jako proces, kterým je ověřeno, jestli daný obvodový návrh vyhovuje odpovídající specifikaci. Jinými slovy, hardwarová verifikace je proces, který ověřuje přesnost obvodové implementace vůči specifikaci.

Verifikace obvodů smíšené domény v základu stále začíná sestavením analogových bloků a jejich simulací v analogovém prostředí, zatímco jsou vyvíjeny logické bloky v digitálním prostředí. Jakmile ale dojde ke spojení analogových a digitálních částí, i použití nejrychlejšího analogového simulátoru znamená výkonovou překážku pro verifikační proces. Na systémové úrovni proto bývají analogové bloky integrovány jako „černé skřínky“. Při verifikaci smíšeného systému se tak běžně ignorují analogové funkce a vykonává se pouze malé nutné množství testů, které nezahrnuje vzájemnou interakci mezi analogovými a digitálními bloky. Tento přístup ale vede k chybám a novým iteracím návrhového procesu. [3]

Ačkoli je stále nutné simulovat a verifikovat analogové a digitální obvody samostatně, pro zajištění správné funkce smíšených integrovaných obvodů je to nedostatečné. Chyby na rozhraní analogových a digitálních částí bývají běžným problémem při návrhu smíšených obvodů. Takové obvody už tak není vhodné simulovat odděleně, jako samostatné analogové a digitální subsystémy, ale vývojář musí tyto obvody simulovat dohromady jako smíšenou doménu. Přitom ještě využívá širokou škálu pokročilých verifikačních metod, aby dosáhl požadovaného pokrytí před ukončením návrhu. [1] [2]

Zatímco se digitální verifikační metody v průběhu let rozvinuly, metody zajišťující verifikaci analogových a smíšených obvodů stále zaostávají. Digitální verifikační postup je strukturovaný a vychází z automatizované vývojové metodologie, což při analogové verifikaci nemůže být běžně využito. Je ale žádoucí, aby se uskutečnil přechod na strukturovaný analogový postup, který by byl efektivnější a

vývojářům by umožnil lépe si poradit s analogovými a smíšenými obvody, jejichž velikost stále roste. [2]

3.1 Verifikační vývoj

Mezi základní úlohy většiny verifikačních metod patří vytvoření verifikačního plánu, vytvoření testovacích simulací, simulace samotné a zhodnocení jejich výsledků. To zahrnuje měření a porovnání výsledků oproti specifikaci. [1]

Většina analogových verifikací tradičně závisí na simulacích a následném zhodnocení jejich výsledků. Zhodnocení výsledků simulace znamená v praxi optickou kontrolu výsledných grafů např. tranzientní analýzy. Tímto způsobem však mohou jednoduše, i když neúmyslně, vznikat chyby. Pouhým pohledem, bez dostatečného detailního přiblížení křivek, totiž vývojář nemusí být schopen detekovat některé malé zákmity obvodu, případně časové nesrovnalosti (delay, skew, apod.).

Při velikosti dnešních systémů je pak problémem i samotná doba běhu simulací, která se může pohybovat v řádu hodin i dnů. Tomu pak odpovídá objem dat s výsledky, které mohou mít velikost i stovek gigabajtů. V takovém množství dat a křivek vývojář snadno přehlédne nesrovnalosti, které by mohla odhalit automatická kontrola simulačních výsledků. Postupem času navíc vývojář ztrácí koncentraci a v grafech vidí spíše to, co tam vidět chce. Odstranění lidského faktoru je v tomto ohledu velmi žádoucí.

Potřeba všeobecnějších verifikací smíšených systémů znamená pro verifikační týmy, že musejí kromě zažitých postupů, jako jsou přímé testování, rozmítání veličin a parametrů (sweeps), procesní rozptyly (corners) a analýzy typu Monte Carlo, převzít i další verifikační metody, aby zpřístupnily regresní testování smíšených systémů. Tyto metody zahrnují automatické vytváření testovacích stimulů, sledování pokrytí (coverage), ale především verifikace využívající automatickou kontrolu simulačních výsledků (tzv. „assertion-driven“). Na různých úrovních abstrakce se pak uplatní různé verifikační metody. [2] [8]

Pro úsporu času a zrychlení simulací je pak vhodné využít výše popsané modelovací techniky. Moderní přístupy modelování analogových bloků již existují, ale priorita při verifikování analogových obvodů je stále na straně přesnosti. Navíc, analogová a digitální vývojová prostředí jsou odlišná a je náročné integrovat bloky tak, aby vyhovovaly verifikaci ve smíšené doméně. Aby bylo možné čelit těmto výzvám, je potřeba mít řešení, které bude ve smíšené doméně pracovat rychle a přesně. Navíc by takové řešení mělo být jednoduché na používání a mělo by být jednoduše integrovatelné do současných analogových a digitálních metodologií. [2]

3.2 Verifikační metody

„Efektivní verifikace neznamená seběhnout co nejvíc simulací za co nejkratší čas, ale seběhnout nejefektivnější simulace, co nejvíckrát je to jen možné.“ [9]

Pro verifikace obvodů existují různé metody, které se v současné době i běžně používají – ať už při verifikaci digitálních, analogových nebo smíšených obvodů. Zvolit mezi nimi jednu nejlepší ale není možné.

Pro různé úlohy, které si verifikace vyžaduje, se totiž hodí různé metody a nástroje. Pro některé je vhodná simulace, pro některé emulace, pro některé statická nebo formální verifikace, atd. Cílem by mělo být použít vhodnou metodu a vhodný nástroj pro určitou požadovanou úlohu. To by mělo zajistit bezpečnější nalezení chyb, úsporu času a ve finále i financí. [5]

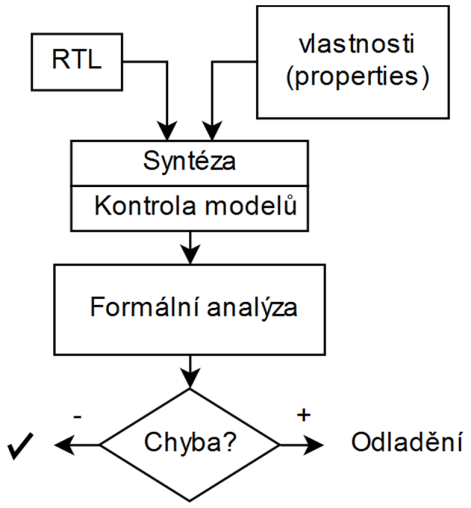
Pokud je například požadavkem funkční verifikace čistě digitálního bloku, vhodnou verifikační metodou by mohla být „pouhá“ simulace na vyšší úrovni abstrakce (např. na úrovni RTL kódu) v kombinaci s „assertion-based“ verifikací. Nástrojem by v tu chvíli byl digitální simulátor. Na druhou stranu, naprosto nevhodným způsobem by bylo zapojení analogového simulátoru, který by celou simulaci neúměrně prodloužil a z funkčního pohledu by nepřidal žádnou hodnotu.

Výběr vhodného verifikačního postupu by se tak dal přirovnat k výběru golfové hole před úderem. Míček jde odpálit každou holí, která je k dispozici. Ale splní odpal

očekávání? A bude odpal dostatečně kvalitní a efektivní na to, aby se míček přiblížil cíli?

3.2.1 Formální verifikace

Formální verifikace využívá matematické formální metody, aby dokázala nebo vyvrátila správnost systémového návrhu s ohledem na formální specifikaci vyjádřenou v podobě vlastností (properties). Jednotlivé vlastnosti (property) jsou, jednoduše řečeno, návrhová pravidla, která by měla být vždy platná. Zastupují v podstatě zadanou specifikaci ve formě kódu. Verifikace tohoto typu je velmi důkladná, ačkoli nevyužívá žádné testovací simulace. [10]



Obrázek 3: Formální verifikace [10]

Tato verifikační metoda zahrnuje více různých úloh a je častěji využívána v souvislosti s digitálními obvody. Zde se nalézá např. v podobě kontroly logické ekvivalence, kde se pomocí formálních matematických algoritmů porovnává shodnost různých reprezentací obvodu. Tento typ kontroly provádějí např. nástroje Conformal (Cadence) nebo Formality (Synopsys). V kontextu analogové domény by se za formální verifikační operace daly považovat kontroly elektrických pravidel a obvodové topologie (např. nástroj společnosti Mentor – PERC).

3.2.2 Prototypová výroba (Prototyping)

Prototypová výroba je samostatným zvláštním odvětvím, které se poměrně liší od ostatních verifikačních metod. I přesto je ale možné ji mezi verifikace zahrnout. V kontextu návrhu integrovaných obvodů je využívána např. pro testování nových obvodových topologií, nebo při testování nových technologických procesů. Prakticky ji využívají např. skupiny vývojářů IP bloků – jinak je tato metoda spíše doplňková.

3.2.3 Emulace

Emulace je druhá verifikační metoda, která využívá fyzickou podobu navrhovaného systému, podobně jako prototypová výroba. A také druhá metoda, která je vhodnější spíše pro digitální doménu. Zde ji zastupují např. emulace pomocí FPGA obvodů, někdy nazývané také jako „FPGA prototyping“. Emulace analogových obvodů a smíšených obvodů možné jsou, ale především emulace analogových obvodů je poměrně složitou činností.

3.2.4 Univerzální Verifikační Metodika (UVM, UVM-MS a A-UVM)

Univerzální verifikační metodika (Universal Verification Methodology – UVM) je první standardizovaná metoda, která je definována organizací IEEE s označením IEEE Std 1800.2. Metodika cílí na současnou verifikační komplexnost a spolupráci mezi společnostmi v rámci celého elektronického průmyslu. UVM vychází z otevřené verifikační metodiky (Open Verification Methodology - OVM), která kombinuje pokročilou verifikační metodiku (Advanced Verification Methodology – AVM) s univerzální metodikou opětovného použití (Universal Reuse Methodology – URM) a koncepty z metodiky ***e*** *(**e*** Reuse Methodology – eRM). Dále UVM naplňuje koncepty a principy psaní kódu verifikačního metodologického manuálu (Verification Methodology Manual – VMM) a shromažďuje znalosti a zkušenosti více než 300 členů UVM pracovní skupiny s názvem Accellera, která pomáhá standardizovat verifikační metody. [11]

V základu je UVM založeno na jazyce SystemVerilog, ve kterém je definována knihovna tříd. Tyto třídy standardizují běžné funkce, jako je komunikace mezi komponentami testu, nebo automatizace práce s daty (kopírování, porovnávání, apod.). [8]

Tato metodika je ale vhodná pouze pro digitální doménu, protože zatím neobsahuje analogové a smíšené rozšíření, o kterých se v posledních letech hovoří.

Např. v roce 2018 v rámci evropské návrhové a verifikační konference (Design and Verification Conference and Exhibition – DVCON) proběhla diskuze, jejímž cílem bylo zjistit velikost zájmu a podpory při případném dalším vývoji rozšíření UVM metodiky o analogovou a smíšenou část (UVM AMS a A-UVM). Tématu rozšíření UVM se ještě předtím věnovala také kniha „Advanced Verification Topics“ (ISBN 9781105113758), která byla publikována v roce 2012, tedy krátce po vydání první verze UVM (r. 2011). Poslední verze standardu IEEE Std 1800.2-2017 z roku 2017 ale zatím žádné takové rozšíření neobsahuje. [8]

3.2.5 Simulace

Simulace jsou základní a v podstatě neoddělitelnou součástí vývoje jakéhokoli typu obvodu a jsou elementárním prvkem verifikací v jakékoli doméně. V současné době se používají u naprosté většiny návrhů. Pouhé užití simulací, jako prostředku pro verifikování obvodu, je ale naprosto nedostatečné. Simulace se tak běžně kombinují s dalšími verifikačními metodami, aby bylo dosaženo jejich větší efektivity.

Po odsimulování návrhu jsou k dispozici výsledky, které odpovídají použitým modelům. Otázkou ale je, jak prokázat, že jsou tyto výsledky správné – jak prokázat, že se obvod chová podle specifikace za všech okolností – případně jak odhalit chování obvodu, které specifikováno není, aby mohla být iniciována diskuze o správnosti takového chování.

S odhalením nespecifikovaného chování obvodu souvisí způsob jeho buzení během simulace. Metody podporující náhodné a pseudonáhodné buzení dokáží vhodným způsobem zvýšit verifikační pokrytí. S větším pokrytím se zvětšuje i množství dat, které je potřeba následně zanalyzovat. Pouhé prohlížení křivek a simulačních výsledků ale není efektivní, a navíc se tak jednoduše generují chyby. Analýza simulačních výsledků by tak měla být strojová a automatická. Pro tyto účely slouží verifikační metody popsané dále v textu.

Pro základní strojovou kontrolu simulačních výsledků je ale možné využít i výrazy a rovnice, které se dají definovat v rámci simulačního prostředí. Výrazy a rovnice zpracovávají hodnoty odsimulovaných veličin a výsledky těchto rovnic je možné porovnat proti nastaveným specifikacím. Díky tomu je tak možné získat zpětnou vazbu o tom, které parametry v simulaci vyhověly, a které ne. Taková metoda post-processingu je ale individuální pro každý obvod a testovací schéma a není možné ji nijak reprodukovat nebo přenášet mezi návrhy.

3.2.6 Kontrola bezpečné pracovní oblasti (SOA)

Kontrola bezpečné pracovní oblasti, nebo také „Save Operating Area checks“ (SOA), představuje základní způsob automatické kontroly simulačních výsledků a v současné době je součástí většiny simulátorů. Jedná se o kontrolu obvodových parametrů pomocí tvrzení (asserts), které mají textovou podobu. Syntaxi těchto tvrzení popisuje např. uživatelská příručka simulátoru Spectre ([12]) a v podstatě se jedná o velmi zjednodušenou ABV metodu (viz. níže).

Pomocí SOA tvrzení je možné kontrolovat pouze překročení limitních hodnot některého parametru, nebo obvodové veličiny (napětí, proud, apod.). Limitní hodnoty lze definovat pro minimální i maximální mez, přičemž pokud hodnota kontrolované veličiny opustí definovanou bezpečnou pracovní oblast na delší dobu, než je stanoveno v tvrzení, vyhlásí simulátor chybovou hlášku podle nastavené úrovně kritičnosti.

Obecná syntaxe a příklad části kódu se zápisem SOA tvrzení jsou uvedeny na následujících řádcích:

```
simulator lang=spectre
library <lib_name>
section <sect_name>
<assert_name> assert sub=<subckt_name> expr=.. min=.. max=.. level=.. duration =..
…
…
endsection <sect_name>
endlibrary <lib_name>
```

```
simulator lang=spectre
library my_lib
section asserts
soa_Iin assert sub=iso_q expr="i(n)" min=-20e-6 max=20e-6 level=warning
      duration =500e-9
endsection asserts
endlibrary my_lib
```

Základní využití metody SOA je běžné v rámci jednotlivých součástek a obvykle se tato tvrzení nacházejí přímo v modelech. Návrhář nebo verifikační inženýr má pak možnost definovat vlastní tvrzení na libovolné hierarchické úrovni návrhu. Nevýhodou SOA je, že neumožňuje detailní specifikaci podmínek, a tak často hlásí falešné chyby. Tyto pak musí návrhář individuálně procházet a rozhodovat, zda-li se opravdu o falešnou chybu jedná, či nikoli. Zde je tak potřeba ještě nástroj pro prohlížení a klasifikaci výsledků.

3.2.7 Verifikace pomocí tvrzení (ABV)

ABV je výkonný verifikační přístup, který je dlouhodobě prověřen verifikačními inženýry a návrháři digitálních obvodů, a kterým stále pomáhá zlepšit kvalitu návrhu. Potenciál ABV začal být pro smíšenou doménu dostupný díky rozšíření jazyků SystemVerilog a PSL (Property Specification Language). [1]

Tato verifikační metoda je založena na tvrzeních (z angl. assert), které zachycují požadované chování a vlastnosti (property) obvodu, za určitých podmínek, ve formě kódu. Jednotlivá tvrzení jsou psána během vývoje obvodu („low-level assertion“) i jeho verifikačního prostředí („high-level assertion“). První typ tvrzení píší obvodoví návrháři a jedná se o implementačně orientovaná tvrzení, která souvisejí s funkcí konkrétního bloku. Druhý typ tvrzení píší verifikační inženýři a jsou to tvrzení, která souvisí s funkcí systému jako celku. Návrháři i verifikační inženýři přitom vycházejí ze stanovené specifikace. [1] [13]

Vlastnosti (properties) zachycené v tvrzeních (asserts) mohou mít širokou škálu podob, od jednoduchých až po velmi propracované a komplexní, kterými je možné detekovat nejrůznější chyby v návrhu. Při simulaci jsou pak všechna tato tvrzení současně kontrolována, zda-li nedošlo k jejich porušení. Jedná se tak o další vysoce účinnou metodu, která automaticky zpracovává simulační výsledky. [1]

Účel této metody pramení ze skutečnosti, že pouhé optické zkoumání výsledků časové analýzy je pracný proces, kterým jsou jednoduše generovány chyby. Některé obvodové poruchy totiž nemusí být ihned viditelné pouhým pohledem do výsledných grafů. Doplnění tohoto procesu o ABV metodu znamená rychlejší a pečlivější identifikaci chyb. [1]

Výhoda metody ABV oproti předchozím automatickým simulačním kontrolám spočívá v tom, že je nezávislá na jednotlivých simulacích a do jisté míry je tak znovupoužitelná pro více testů i návrhů. V porovnání s metodou SOA pak nabízí mnohem širší portfolio kontrol, včetně možnosti kontrolovat nejrůznější sekvence a signálové průběhy, ať už v digitální nebo analogové doméně. Jednotlivé kontroly navíc probíhají v průběhu simulace, a ne jako součást posimulačního zpracování.

Oproti standardu UVM je mnohem jednodušší ji použít. Pokud se vývojář orientuje v syntaxi jazyku SystemVerilog, tak pro něj nebude složité sepsat i obvodová tvrzení. A i několik jednoduchých tvrzení dokáže okamžitě zkvalitnit verifikaci obvodu. [14]

3.2.8 Verifikace pokrytím (CBV)

Verifikační inženýři analogových a smíšených obvodů tradičně spoléhají na přímé testování. To ale neposkytuje žádnou zpětnou vazbu o tom, které části systému již byly otestovány, v jakých stavech a při jakých podmínkách. Nelze tak nijak vyčíslit nebo určit pokrytí verifikace (coverage), tedy jaká část návrhu je již zverifikována, nebo které části obvodu nebyly dostatečně otestovány. Další a další testy navíc nemusí poskytnout žádnou přidanou hodnotu. [1]

Těmto výzvám je možné čelit pomocí metody sledující verifikační pokrytí CBV, tedy „Coverage-based Verification“, která je již zavedená a běžná při verifikaci digitálních obvodů. Cílem této metody je pomocí definovaných skupin (coverage groups) a bodů (coverpoints) kvantifikovat verifikační postup. Skupiny pokrytí a jejich body reprezentují různé obvodové vlastnosti nebo funkce, a v širším slova smyslu je lze rozdělit na statické (např. kontrola hodnot) nebo dynamické (např. kontrola sekvencí). [1]

Jako výhodné body pokrytí, které lze jednoduše měřit, se osvědčily jednotlivé vlastnosti (properties) z předchozí ABV metody. Obě tyto metody (ABV i CBV) jsou navíc nedílnou součástí metricky-řízené verifikace (MDV, viz. níže). Se zavedením MDV je pak možné pokrytí i měřit, a tak určit stav verifikace – tedy je možné odpovědět na otázky: „Je verifikace dokončena? Jak velkého pokrytí je dosaženo?“. Dále je díky tomuto přístupu možné identifikovat verifikační nedokonalosti a mezery v pokrytí, které napoví, jakým způsobem dále rozvíjet verifikační plán a jeho konkrétní testy. [1]

Vzhledem k původnímu využití CBV metody, jsou základní oblasti pokrytí orientovány do digitální domény. Mezi typické oblasti a jejich podmnožiny, které podléhají měření pokrytí, patří:

- pokrytí kódu (code coverage),
  - větvení kódu (branch coverage),
  - řádky kódu (line / statement coverage),
  - pokrytí výrazů (expression coverage),
  - pokrytí cest / sekvencí (path coverage),
- pokrytí stavových automatů (FSM coverage),
  - pokrytí stavů (state coverage),
  - pokrytí přechodů (transition coverage),
- strukturní pokrytí (structual coverage),
  - stavové změny (toggle coverage),
  - kombinační pokrytí (combinational coverage),
- funkční pokrytí (functional coverage).

Pokrytí kódu závisí jednoduše na zdrojovém kódu modelu. U této oblasti je kontrolováno, zda-li simulace prověřila veškeré větvení modelu (tedy všechny části bloků if-else a case) a všechny kombinace jejich posloupností (kombinace různých sekvencí), jestli byly v akci všechny řádky kódu a jestli se všechny logické výrazy vyhodnotily do obou logických úrovní. [1]

Pokrytí stavových automatů kontroluje průchod všemi stavy pomocí všech definovaných přechodů. Strukturní pokrytí pak kontroluje počet stavových změn jednotlivých proměnných a signálů v modelu (tedy kolikrát u určité proměnné nastala změna log. 0 -> log. 1, apod.) a pokrytí vstupních kombinací jednotlivých výrazů. [1]

Poslední oblast, funkční pokrytí, kontroluje, kolik funkcí obvodu bylo otestováno. Jednotlivé body tohoto pokrytí definuje vývojář a měly by korespondovat s funkcemi definovanými ve specifikaci. Jednotlivé funkce bývají vyjádřeny v podobě popsaných vlastností (properties) a celkově je tato oblast pouze doplňková pro oblast pokrytí kódu.[1]

Využití metody CBV ve smíšené doméně pak skrývá několik problémů. Například při náhodně generovaných simulacích je potřeba vygenerovat dostatečné množství analogových hodnot, kterými je následně testovaný obvod stimulován. Touto množinou hodnot je pak potřeba verifikovat všechny režimy, ve kterých je daný obvod schopen pracovat. Kolik analogových hodnot ale zajistí dostatečné pokrytí? [1]

Kromě toho, automaticky generované přístupy pokrytí nepracují velmi dobře s analogovými modely. Především pak s modely tranzistorové úrovně, které jsou reprezentovány schématem. Zde je prakticky nemožné využít kontrolu pokrytí kódu a stavových automatů. Někteří dodavatelé nástrojů se ale v oblasti smíšených obvodů snaží podporovat alespoň funkční pokrytí a strukturní pokrytí. [1]

3.2.9 Metricky-řízená verifikace (MDV)

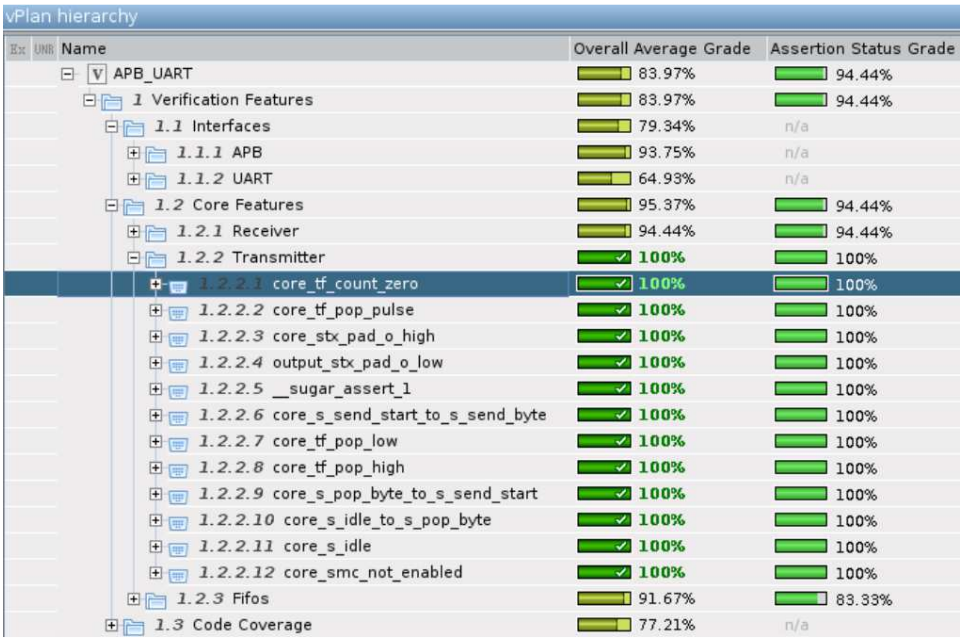
Jakousi nástavbou nad CBV je metoda MDV, tedy „Metric-Driven Verification“. Jedná se původně o digitální metodiku, která v podstatě slučuje výše zmíněné přístupy a v digitální doméně pomáhá vývojářům dosáhnout požadovaného pokrytí.

Základ tvoří simulace, které využívají náhodně generovaných stimulů. Tyto stimuly zajistí zvýšení počtu kombinací vstupních hodnot, kterými je buzen testovaný obvod. Pro nezávislou kontrolu simulačních výsledků je zde zahrnuta metoda ABV. ABV navíc zajišťuje informace o částech obvodu, které byly testovány, a tím zpřístupní potřebné informace pro vyhodnocení pokrytí. Díky těmto informacím je pak možné využít výhody metody CBV a vyvarovat se tak zbytečným testům, které míru pokrytí nezvyšují. [1] [5] [2]

Výhoda MDV spočívá v tom, že dokáže měřit jednotlivé „metriky“ – tedy nejrůznější kontrolní podmínky (checks), tvrzení (assertions) a další oblasti a body, které podporuje metoda CBV. Hodnoty jednotlivých oblastí jsou pomocí MDV nástroje

zpracovány a následně je možné určit míru verifikačního pokrytí a kvalitu testů. Metoda tak ve zkratce určuje, zda-li bylo dosaženo dostatečného pokrytí, aby se testovaný obvod prohlásil za zverifikovaný.

Pomocí MDV nástroje je také možné, kromě zhodnocení výsledků, přehledně a hierarchicky sestavit verifikační plán. Takovým nástrojem je v digitální doméně např. vManager (Cadence) a ačkoli je na oficiální webové stránce tohoto nástroje uvedena podpora i smíšené domény (v podobě RNM modelů), zástupci společnosti tuto skutečnost prozatím vyvracejí (informace z verifikační konference, Mnichov, Q4-2018). Otázkou tak je, co zástupci společnosti považují za smíšenou doménu. RNM modely totiž mnoho lidí řadí do digitální domény, z čeho může vyplývat tento rozpor.



Obrázek 4: Prostředí nástroje vManager [9]

Metoda MDV je hojně používaná pro verifikace RTL bloků a softwaru. Tato skutečnost vychází především z principu CBV, kde se hlavní oblasti pokrytí věnují vykonání různých částí kódu. Otázkou zůstává, jestli je tato metoda využitelná pro verifikaci smíšených obvodů, potažmo analogových bloků.

Pro využití MDV metody v analogové a smíšené doméně je potřeba vkládat tvrzení (assert) do SPICE a analogových behaviorálních modelů, poskytnout podporu pro pseudonáhodné generování vstupních analogových stimulů a definovat jednotlivé

„metriky“ a oblasti pokrytí, které by vyhovovaly těmto doménám. To jsou nejzásadnější výzvy, kterým je potřeba čelit, než bude možné bezpečně aplikovat MDV metodu v analogové a smíšené doméně.

3.2.10 Verifikace požadavků (RDV)

Metoda RDV, tedy „Requirements-Driven Verification“, je založena na principu verifikace jednotlivých obvodových požadavků (requirements), které jsou definovány seznamem požadavků, tzv. „Requirement Tracking System“. Tento seznam definuje funkce, které musí navrhovaný obvod vykonávat a je potřeba ho definovat pro každý obvod, blok nebo systém.

Na základě seznamu požadavků jsou následně určeny aspekty, které mají být verifikovány, a podmínky, při jakých mají být verifikovány. Metoda RDV tedy určuje, co má být verifikováno. Jednotlivé požadavky jsou následně i verifikovány, přičemž množství verifikovaných požadavků může být jednou z metrik metody MDV. Kombinace těchto dvou metod, tedy RDV a MDV, se pak přímo nabízí. Metodou RDV je definováno, co se má verifikovat, a metodou MDV, jestli je to dostatečně verifikováno.

Metoda RDV je především kvůli přípravě seznamu požadavků velmi časově náročná. Pokud je ale k danému obvodu nebo bloku dostupná specifikace, znatelně zvyšuje kvalitu verifikace. Na druhou stranu, při absenci specifikace je tato metoda nepoužitelná. Na následujících řádcích je uveden vhodný formát pro záznam požadavků a jednoduchý příklad: [14] [15]

```
The {output or verifiable aspect}
 shall
  {always, unconditionally, only}
   {assert, deassert, set to value}
    {before, after, when, during, within}
     {x nsec, the next rising edge of a clock}
      when
       {inputs are set to a combination of high/low,
        a sequence of events has occurred or
        a timed period elapses}
```

```
IP_CORE_FR_001:
The {dscrt_out}
 shall
  {always}
   {assert to logic HIGH}
    {within}
     {40 nanoseconds}
      when
       {dscrt_in is asserted to logic HIGH}
```

4 ABV metodologie pro smíšenou doménu

Cílem, nebo spíše současným směrem, kterým se chce vývojové oddělení společnosti SCG Czech Design Center v otázce verifikací vydat, je začít využívat původně digitální verifikační metody RDV a MDV i v analogové a smíšené doméně.

K zavedení těchto metod, které se příznivým způsobem doplňují, je ale potřeba přistoupit po částech a k žádanému výsledku se propracovat postupnými kroky. V rámci metody RDV je tak potřeba najít optimální způsob, jak definovat seznamy požadavků, jakou formou a v jakém formátu. Souběžně s tím pak postupně sestavovat metodiku MDV – tedy definovat jednotlivé metriky a oblasti pokrytí, s čímž taky souvisí popisování obvodových vlastností (properties) ve formě kódu a tvrzení (assert), které se vážou na seznamy požadavků metody RDV a specifikace.

Jedním ze základních kroků tak musí být osvojení metody ABV, která pracuje s obvodovými vlastnostmi a tvrzeními. V současné době je tato metoda ve společnosti používána pouze pro verifikaci digitálních obvodů. V analogové a smíšené doméně se zde tato metoda začíná teprve rozvíjet, ačkoli již byly v minulosti iniciovány aktivity, které se tímto tématem zabývaly.

Metoda ABV je v analogové a smíšené doméně podporována dvěma jazyky. Jsou to jazyky PSL a SystemVerilog (SystemVerilog Assertion – SVA). Tyto jazyky jsou tzv. „Assertion-based“, tedy jejich hlavním účelem je stručný popis chování obvodu (neměl by „kopírovat“ kontrolovaný HDL kód), které může zasahovat i do více hodinových cyklů.

Pro popis takového chování existuje v obou jazycích množství sekvenčních operátorů. Díky těmto operátorům má vývojář možnost definovat požadované nebo nechtěné průběhy kontrolovaných singnálů v přehledném časovém sledu. Definice těchto sekvencí je jádrem obou jazyků a je to zároveň hlavní síla metody ABV.

Oba jazyky jsou si velmi podobné a oba vyhovují chystanému verifikačnímu konceptu společnosti. Tyto jazyky je tak nutné vzájemně porovnat a zhodnotit, který z nich má větší budoucí potenciál, který z nich poskytuje více výhod, a který má méně

omezení. V návaznosti na toto porovnání pak bude možné určit, který z nich se bude nadále využívat, případně, zda-li nebude vhodnou alternativou podpora obou těchto jazyků.

4.1 Problémy smíšených tvrzení

Zavedení ABV do analogové a smíšené domény s sebou přináší i řadu problémů, kterým je potřeba čelit. Mezi nejzásadnější patří způsob aplikace jednotlivých tvrzení na různé obvodové úrovně abstrakce.

4.1.1 Doménová univerzálnost

Tradiční využití metody ABV v digitální doméně předpokládá práci na vyšší úrovni abstrakce, kdy jsou jednotlivé bloky reprezentovány např. HDL kódem. Jednotlivá tvrzení mají také podobu kódu, který může být vložen přímo do jednotlivých modulů („low-level assertion“), nebo je součástí simulačního prostředí („high-level assertion“).

V analogové a smíšené doméně je ale běžné, že jsou některé bloky reprezentovány na tranzistorové úrovni – tedy buď schématickým nákresem, nebo popisem, např. netlistem v jazyce SPICE. I na tyto bloky a jejich signály ale musí být možné aplikovat verifikační tvrzení.

Při simulacích obvodů smíšené domény je pak běžné, že je jeden konkrétní blok v různých testech reprezentován různými modely. Může tak nastat situace, kdy některý konkrétní signál bude v jednom testu součástí digitální domény, ale v jiném testu bude analogového typu. Tuto možnost je potřeba zohlednit a tvrzení konstruovat tak, aby byla připravena na obě varianty.

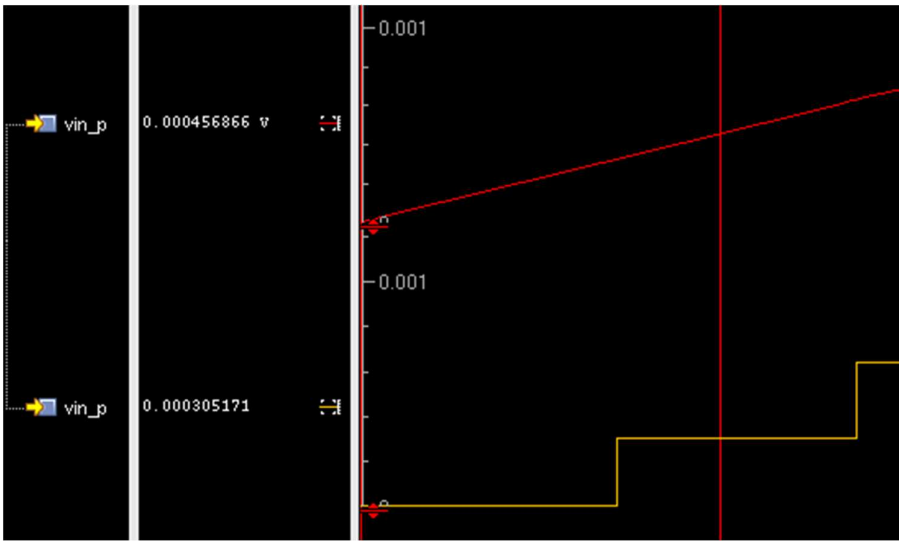
4.1.2 Přístup k obvodovým signálům

S předchozí kapitolou souvisí způsob přístupu k jednotlivým signálům. Z funkčního pohledu by v případě zpracování digitálních signálů neměl ani jeden

z výše uvedených jazyků narážet na nějaká omezení, protože svým původem do této domény patří.

Naopak, při zpracování analogových signálů jsou jejich přístupy naprosto odlišné. Jazyk PSL poskytuje možnost přímého vyčítání analogových hodnot. Dokáže tedy přistoupit k signálům, které jsou analogového typu (“electrical“). SVA namísto toho pracuje s reálnými čísly a objekty analogového typu vůbec nepodporuje. V tomto případě je tak nutné převézt analogové hodnoty na reálná čísla, což vyžaduje použití speciálních propojovacích modulů (u Cadence tzv. „connect module“, u Metor tzv. „boundary element“), které tento převod zajišťují.

Průběh převedeného signálu je ale nespojitý a vznikají při něm drobné odchylky, se kterými je potřeba počítat. Přesnost propojovacích modulů sice je do jisté míry nastavitelná, ale větší přesnost se negativně projeví na rychlosti simulace. Následující obrázek zobrazuje srovnání signálu analogové domény (červená křivka) a převedeného reálného čísla (žlutá křivka) po zpracování simulátorem. Z obrázku je patrné, že nastavený krok propojovacího modulu je v tomto případě 305,171 µV.

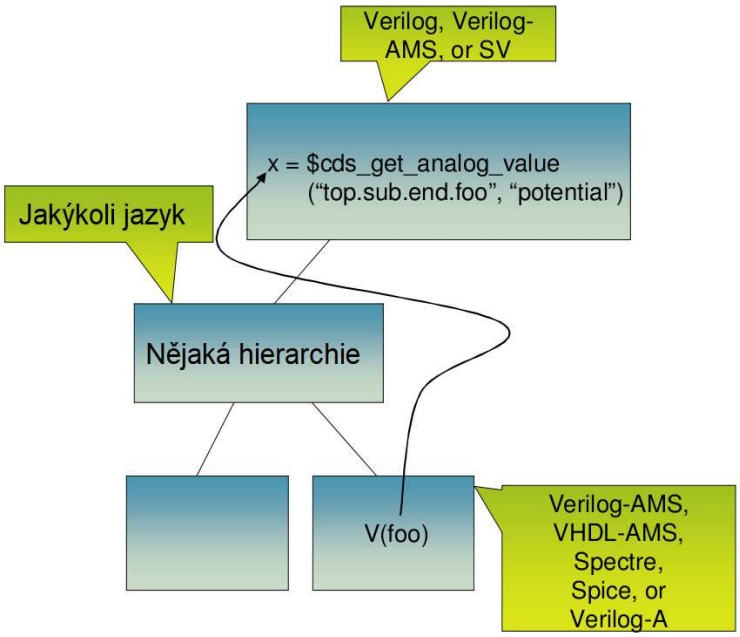


Obrázek 5: Přesnost mezidoménového převodu hodnot

Dalším způsobem, kterým lze přistoupit k analogovým hodnotám, je použití systémové funkce “$cds_get_analog_value“ (zkrácenou formou “$cgav“). Tato funkce slouží k vyzvednutí hodnoty napětí, proudu, výkonu nebo parametru součástky

z analogového objektu (tzv. „value fetch“) a oficiálně je uvolněna od verze simulátoru INCISIV SimVision 9.20.018.

Parametrem této funkce je hierarchická cesta k analogovému objektu a typ veličiny (nebo název parametru), který má být vyčten. Návratovou hodnotou této funkce je reálné číslo. Opět se tak jedná o převod hodnoty z analogové domény na reálné číslo, tentokrát ale bez použití propojovacích modulů. Ukázka užití této funkce je na následujícím obrázku. [1] [16] [17]



Obrázek 6:Princip funkce $cgav [16]

Nevýhodou této funkce je skutečnost, že se jedná o systémovou funkci nástrojů společnosti Cadence. Tvrzení, která tuto funkci budou využívat, tak zřejmě nebude možné použít v jiných nástrojích.

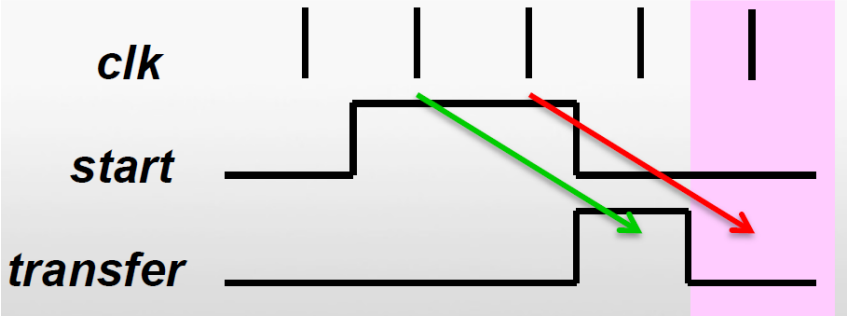
4.1.3 Preciznost při psaní tvrzení

Dalším aspektem, který je potřeba zohlednit, je preciznost při sestavování tvrzení. Vývojář totiž může při složitosti systému přehlédnout nějakou skutečnost, která způsobí nesprávnou kontrolu požadovaného chování, případně hlášení falešných chyb.

K demonstraci nejlépe poslouží příklad (syntaxe SVA). Pokud je tedy např. žádané chování obvodu specifikováno tak, že po detekci signálu “start“ se má se zpožděním dvou hodinových cyklů nastavit signál “transfer“ na hodnotu log. 1, mohlo by tvrzení takové sekvence vypadat následovně:

```
assert property (@(clk) start |-> ##2 transfer) ;
```

Toto tvrzení ale nezohledňuje situaci, kdy signál “start“ zůstane ve stavu log. 1 po dobu delší, něž jeden hodinový cyklus – což je chování, které v popisu výše není ničím zakázáno. V takovém případě by mohlo dojít k nesprávnému vyhodnocení tvrzení a potencionálně k vyhlášení falešné chyby. Taková situace je znázorněna na následujícím obrázku.



Obrázek 7: Chybně vyhodnocené tvrzení [18]

Vhodná úprava tvrzení by měla být taková, že se s hodinovým signálem “clk“ bude detekovat náběžná hrana signálu “start“:

```
assert property (@(clk) $rose(start) |-> ##2 transfer) ;
//  nebo
assert property (@(clk) !start ##1 start |-> ##2 transfer) ;
```

Stejná situace by ovšem mohla nastat i u signálu “transfer“. V takovém případě by původní tvrzení ani nezachytilo chybu, protože by se při obou evaluacích vyhodnotilo jako platné.

S tím souvisí, a dalším důležitým prvkem je, kontrola funkce samotných tvrzení. Součástí testovacích simulací by tak měla být část, která otestuje reakci tvrzení. Tu je možné zkontrolovat např. tak, že se testovaný obvod záměrně uvede do chybných

stavů. Odladit správnou funkci jednotlivých tvrzení je totiž zhruba stejně náročná činnost, jako odladit funkci samotného obvodu. Vývojář musí pomocí množství různých simulací otestovat správnou reakci tvrzení při nejrůznějších podmínkách a signálových průbězích, aby zajistil, že kontrola obvodu bude probíhat korektně, a že tvrzení nebude hlásit zbytečné falešné chyby.

4.2 Testovací prostředí smíšených tvrzení

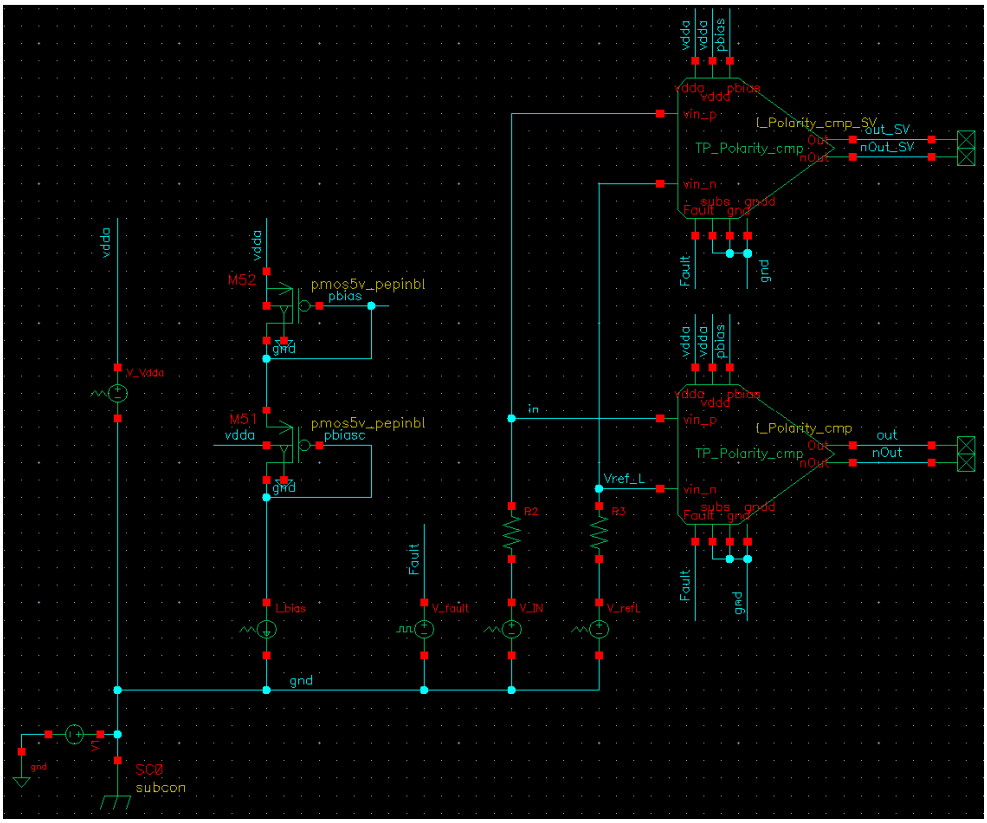
Kromě porovnání obou jazyků je potřeba i jejich praktické otestování a ověření jejich potenciálu při verifikaci analogových a smíšených obvodů. Testování by mělo dokázat, že zvolené jazyky jsou vhodné pro použití při metodě ABV bez omezení, nebo alespoň nalézt limity jejich použitelnosti.

K testování a simulacím byla využita výpočetní technika a softwarové vybavení společnosti SCG CDC. Konkrétně se jednalo o vývojové prostředí Virtuoso – Custom IC Design Environment od společnosti Cadence ve verzi IC6.1.7. Toto prostředí bylo spouštěno pod operačním systémem Linux od společnosti Red Hat Enterprise ve verzi 2.6.32 s nástavbou GNOME v.2.28.2.

4.2.1 Testovací schéma

Pro potřeby testování bylo vytvořeno jednoduché testovací schéma v podobě, která je běžná pro testování individuálních bloků systému. Toto schéma obsahuje nezbytné součástky pro nastavení pracovního bodu obvodu, zdroje signálu pro generování vstupních průběhů a několik modelově různě reprezentovaných instancí testovaného bloku. Schéma je zobrazeno na předchozím obrázku.

Jako testovaný blok byl zvolen komparátor, který ve zvoleném provedení sdružuje analogové i digitální funkce a je dostupný v různých modelových reprezentacích. Tvrzení by se ale měla více zaměřovat na jeho analogové chování, které může být reprezentováno analogovými veličinami, nebo reálnými čísly. Digitální funkce je zde zastoupena pouze vstupem “Fault“, který při aktivaci zajistí nastavení pevné hodnoty na výstup obvodu. V závislosti na zvolené modelové reprezentaci pak tato hodnota odpovídá buď napájecímu napětí, nebo logické úrovni ‘1‘.

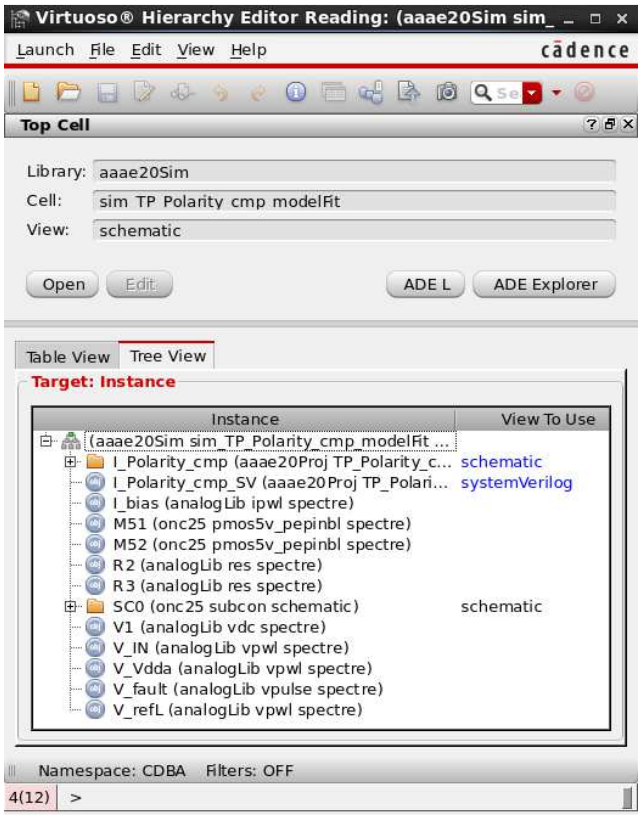


Obrázek 8: Schéma pro testování smíšených tvrzení

4.2.2 Konfigurace základní hierarchie

Pro spuštění simulace smíšeného obvodu je potřeba definovat jeho hierarchii a modelovou skladbu. K definici hierarchie obvodů slouží hierarchický editor, který je součástí vývojového prostředí Virtuoso. V hierarchickém editoru je především potřeba definovat nejvyšší simulační instanci (tzv. „Top Cell“), kterou je v tomto případě testovací schéma z minulé kapitoly. Poté je graficky a přehledně zobrazena hierarchie obvodových prvků a jejich modelových reprezentací, které budou při simulaci použity.

Modelové reprezentace lze pomocí editoru u jednotlivých instancí také volit. Pro potřeby testování smíšených tvrzení tak bude jeden z dvojice komparátorů simulován se schématickou reprezentací, zatímco druhý bude simulován jako HDL modul popsaný v jazyce SystemVerilog. Výslednou hierarchii zobrazuje následující obrázek.



Obrázek 9: Základní hierarchie testovacího schématu

4.2.3 Nastavení simulace

Dalším krokem po definici hierarchie je nastavení simulace pomocí analogového simulačního prostředí ADE (Analog Design Environment), které je opět součástí prostředí Virtuoso. Okno simulačního prostředí po nastavení simulace je zobrazeno na dalším obrázku.

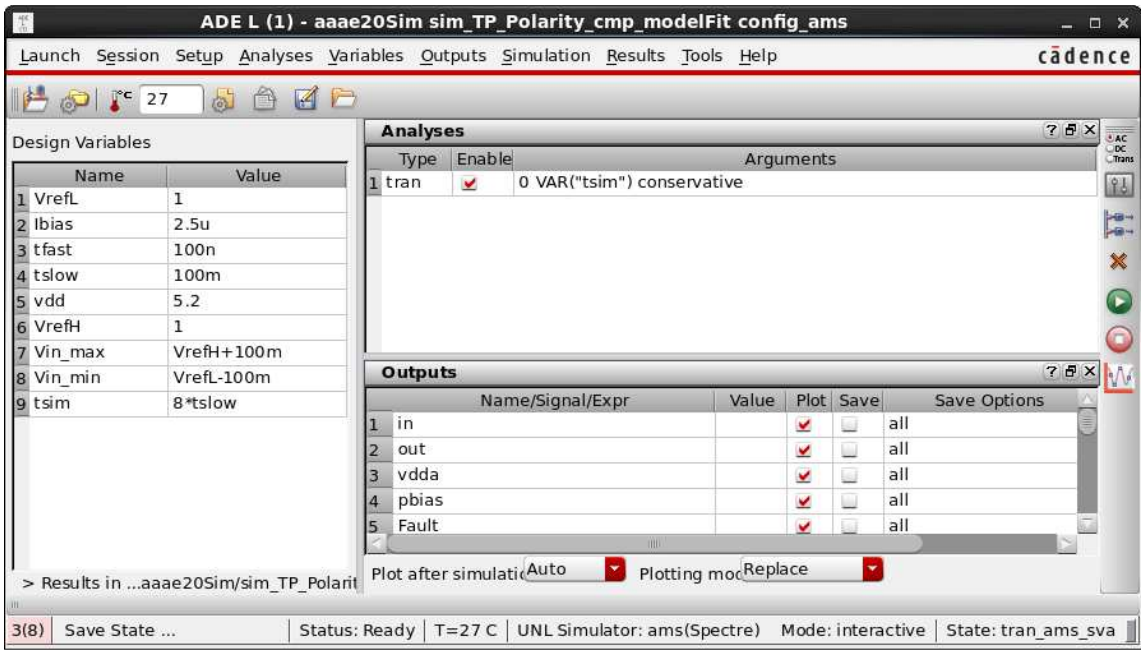
V případě simulace obvodu smíšené domény je v tomto prostředí potřeba nastavit typ simulátoru, pravidla pro mezidoménové propojovací moduly („connect rules“), způsob generování „netlistu“, simulační režim, ale především simulaci samotnou a její výstupy.

Pro potřeby testování ABV je zvoleno následující nastavení:

- simulátor – ams,
- mezidoménové propojovací pravidla – 5V se zvýšenou přesností,
- způsob generování „netlistu“ – AMS.

Jako simulační režim je volen režim “Interactive“. Tento režim způsobí, že po spuštění simulace dojde k otevření simulačního prostředí SimVision. Toto prostředí je také produktem společnosti Cadence a pro testování smíšených tvrzení je zvolena verze 18.03-s012.

Pro samotnou simulaci je zvolen transientní typ analýzy a pro podrobné vyhodnocení simulačních výsledků jsou ve formátu ‘psfxl‘ ukládána všechna napětí i proudy obvodu.



Obrázek 10: Analogové simulační prostředí ADE L

4.2.4 Průběhy testovacích stimulů a výsledek simulace bez ABV

Testovací stimuly simulovaného obvodu tvoří trojice analogových signálů “Vref_L“, “in“ a “Fault“. Signál “Vref_L“ tvoří referenční napětí komparátoru a je po celou dobu simulace držen na hodnotě 1 V. Signál “in“, který je připojen ke vstupu komparátoru, je v průběhu simulace několikrát rozmítán od 0,9 V až po 1,1 V a zpět. Při přechodu mezi těmito úrovněmi jsou navíc použity různé strmosti hran.

Poslední vstupní signál, signál “Fault“, je první polovinu simulace držen na hodnotě 5 V, čímž blokuje funkci komparátoru. Druhou polovinu simulace je nastaven na hodnotu 0 V. V této části simulace je možné pozorovat reakci výstupů komparátoru

v místech, kde vstupní signál protíná referenční hodnotu. Výsledný graf testovací transientní analýzy je zobrazen v příloze č. 1.

Z grafu je patrné, že výstupy komparátoru reprezentovaného schématickým modelem reagují s různým zpožděním v závislosti na strmosti vstupního signálu. Dále se jejich zpoždění liší od zpoždění pozorovaného u komparátoru modelovaného pomocí jazyku SystemVerilog. To jsou aspekty, které je nutné při psaní a testování smíšených tvrzení zohlednit, a které tvorbu takových tvrzení i zásadně ztěžují – tedy rozdílné chování různých modelů a charakter analogových obvodů.

U tvrzení cílících na analogové obvody je tak především potřeba odladit jejich funkci při vystavení různým signálovým průběhům. Průběh reakce analogového obvodu je totiž různý při aplikaci strmých nebo pozvolných hran.

4.3 Jazyk PSL

Jazyk PSL (Property Specification Language) vznikl z jazyku Sugar v roce 2003, díky specifikaci, kterou vytvořila pracovní skupina Accellera. Tato skupina tuto specifikaci také udržovala a až do roku 2005 aktualizovala. V roce 2005 byl jazyk PSL standardizován a převeden do normy IEEE Std.1850, jejíž poslední verze byla vydána v roce 2010.

Smyslem PSL je spíše rozvinutí současných zažitých postupů, než vytvoření nové syntaxe. PSL tak využívá mnoho základních operátorů a HDL syntaxí k sestavení logických výrazů. K tomu ale přidává množství sekvenčních operátorů a funkcí, díky kterým lze sestavit komplexní časové vztahy mezi jednotlivými logickými výrazy. To tvoří hlavní přidanou hodnotu jazyka PSL.

4.3.1 Topologie jazyka PSL

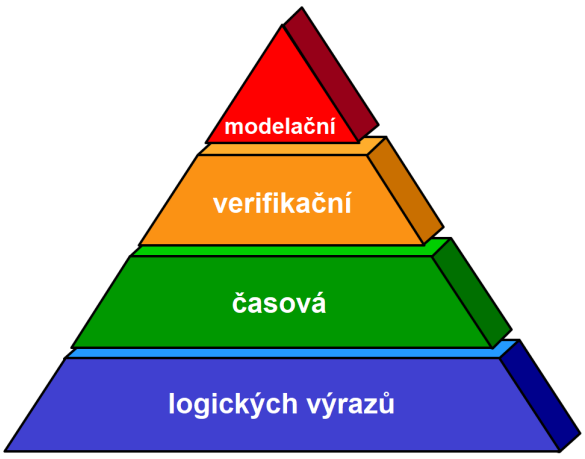
Syntaxe jazyka PSL má pět „mutací“. Tato vlastnost je zde proto, aby bylo dosaženo kompatibility s HDL jazyky SystemVerilog, Verilog, VHDL, SystemC a GDL. Podle zvolené „mutace“ se pak liší zápis některých částí tvrzení. Pro demonstraci této vlastnosti jsou na následujícím příkladu použity dva způsoby zápisu jednoduchého

tvrzení, přičemž první způsob používá syntaxi kompatibilní s jazykem Verilog a druhý způsob syntaxi kompatibilní s jazykem VHDL.

```
1) chk_mutex:  assert  always  !(rd && wr)      @(posedge clk);
2) chk_mutex:  assert  always  not(rd and wr)   @(rising_edge(clk));
```

Dále se PSL skládá ze čtyř vrstev. Nejnižší a základní vrstvu tvoří logické výrazy (Boolean layer), které používají standardní syntaxi HDL jazyků. V předchozím příkladu je tato vrstva reprezentována popisem vztahu signálů „rd“ a „wr“.

Další nadřazenou vrstvou je časová vrstva. Tato vrstva představuje skutečnou sílu jazyku PSL a ve zkratce určuje, kdy musí být podmínka nižší vrstvy platná. V PSL se do této vrstvy řadí ještě množství operátorů, které slouží pro popis komplexnějších sekvencí (tzv. „SERE - Sequential Extended Regular Expressions“ operátory). Jedním z hlavních požadavků na jazyky popisující tvrzení je totiž schopnost stručně popsat chování obvodu, které trvá i více hodinových cyklů. V příkladu je tato vrstva zastoupena klíčovým slovem „always“.



Obrázek 11: Vrstvy jazyka PSL

Nejvyšší vrstvou, která se ještě týká jednotlivých tvrzení, je vrstva verifikační. Tato vrstva využívá verifikační direktivy, které předávají nástroji (např. simulátoru) informaci o tom, co má být s tvrzením provedeno. V příkladu je použita verifikační direktiva „assert“.

Poslední, tedy čtvrtou vrstvou, je modelační vrstva. Tato vrstva zastupuje doplňkový kód, kterým je možné modelovat dodatečné funkce např. pro zjednodušení zápisu jednotlivých tvrzení. Typickým zástupcem modelační vrstvy je vytvoření virtuálního hodinového signálu, kterým je řízena evaluační událost tvrzení.

V příkladu výše jsou navíc uvedeny další dva prvky jazyka PSL. Jedná se o identifikátor tvrzení (chk_mutex:) a evaluační událost (část zápisu za znakem "@"). Identifikátor slouží k jednoznačné identifikaci jednotlivých tvrzení, a ačkoli není jeho uvedení povinné, je doporučeno jej zapisovat. Evaluační událost je pak podmínka, při jejímž splnění se dané tvrzení zkontroluje. Kontrola tvrzení je při splnění této podmínky provedena během jednoho výpočetního cyklu simulátoru.

4.3.2 Definice sekvencí a vlastností

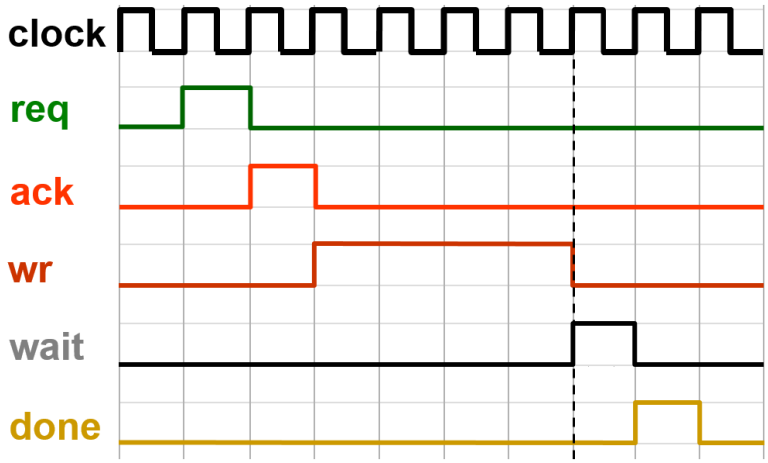
Díky tomu, že je jazyk PSL složen z jednotlivých vrstev, je možné jeho zápis dobře strukturovat a jednotlivé vrstvy popisovat individuálně samostatnými konstrukcemi. Následné sloučení jednotlivých konstrukcí pak tvoří požadované tvrzení. Nejpoužívanějšími konstrukcemi jsou definice sekvencí a vlastností, které obě spadají do časové vrstvy jazyka PSL. Následující příklad zobrazuje složení jednoduchého tvrzení pomocí různých vrstvových konstrukcí:

```
// Logická vrstva
// (wait && !req)

// Časová vrstva
sequence s1 is {req ; ack ; wr[*4]} ;
sequence s2 is {(wait && !req) ; done} ;
property p1 is s1 |=> s2 ;

// Verifikační vrstva
ckh_a1 : assert always p1 @(posedge clock) ;
```

Přiklad nejprve zobrazuje výraz logické vrstvy. Dále jsou popsány prvky časové vrstvy. Do této vrstvy řadíme sekvence a vlastnosti (property), které lze popisovat samostatně. Vlastnost p1 tak definuje, že po detekci sekvence s1 musí proběhnout sekvence s2. Toto chování obvodu je kontrolováno díky použití verifikační direktivy, která patří do verifikační vrstvy. Popsané chování obvodu by mělo mít následující signálové průběhy:



Obrázek 12: Příklad signálové sekvence

Díky tomuto strukturovanému zápisu a množství sekvenčních operátorů je tak možné vytvořit komplexní znovupoužitelné konstrukce, které naleznou své uplatnění na více místech návrhu, nebo i napříč různými nezávislými projekty. Sekvence a vlastnosti lze navíc popisovat i parametricky, což ještě umocňuje univerzálnost zápisu.

4.3.3 Aplikace jazyka PSL

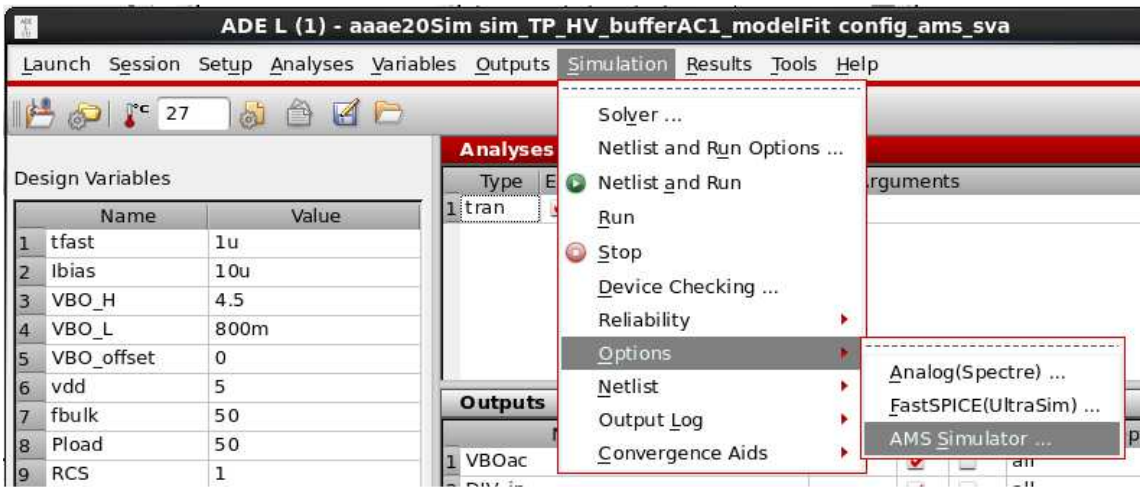
Jednotlivé sekvence, vlastnosti, tvrzení a verifikační direktivy mohou být vnořeny přímo do HDL kódu příslušného bloku jako jednořádkové, nebo i víceřádkové komentáře. V takovém případě předchází PSL zápisu znaky označující jednořádkový komentář (v syntaxi Verilog to je "//") a klíčové slovo „psl". V případě víceřádkového zápisu pak mezi tímto klíčovým slovem a ukončovacím znakem (středník – ";") nesmí být žádné další komentáře. Příklad z předchozí kapitoly, vnořený do HDL kódu se syntaxí jazyku Verilog, by vypadal následovně:

```
// Jednořádkový zápis
// psl sequence s1 is {req ; ack ; wr[*4]} ;
// psl sequence s2 is {(wait && !req) ; done} ;
// psl property p1 is s1 |=> s2 ;

// Víceřádkový zápis
// psl ckh_a1 :
//    assert always p1 ;
```

Další možností, jak aplikovat PSL kód, jsou verifikační jednotky (vunit). Verifikační jednotka je speciální typ modulu, který je většinou zapsán v samostatném souboru a tedy mimo cílový HDL kód. Verifikační jednotky neobsahují vstupní ani výstupní porty a ke kontrolovanému bloku jsou přilinkovány až při simulaci. Tyto jednotky se používají pro seskupení nejrůznějších PSL deklarací, direktiv i dalšího modelačního kódu a v praxi jsou upřednostňovány před zápisem vnořeného kódu.

Základní podmínkou pro aplikaci PSL při simulaci je ale dodatečné nastavení AMS simulátoru, které zahrnuje použití dodatečných přepínačů. Toto nastavení je možné nalézt v okně ADE v nabídce Simulation – Options – AMS Simulator. Cesta k požadované nabídce je zobrazena na následujícím obrázku.



Obrázek 13: Cesta k nastavení dodatečných simulačních argumentů

Po otevření okna nastavení simulátoru je možné do pole „Additional arguments“ dopsat přepínač “-assert“. Tento přepínač zajistí, že se v průběhu simulace budou vyhodnocovat tvrzení. Pokud jsou tvrzení součástí verifikační jednotky, která je v samostatném souboru, je navíc potřeba použít ještě přepínač “-propfile_vlog“, kterým je definována cesta k tomuto souboru (např. -propfile_vlog proj_path/psl.pslvlog).

4.3.4 PSL v analogové a smíšené doméně

Silnou stránkou PSL v oblasti analogových tvrzení je schopnost přímého přístupu k analogovým objektům, tedy k signálům typu “electrical“. PSL při přístupu

k takovému objektu prostřednictvím tvrzení dokáže vyčíst analogovou hodnotu, kterou ihned použije při vyhodnocení tvrzení.

Analogové objekty mohou být součástí logické vrstvy, nebo evaluační podmínky. V takovém případě ale není možné, aby bylo tvrzení citlivé pouze na úroveň analogového signálu. Je nutné použít analogový výraz, který nějakou událost definuje. Takovým výrazem je například detekce protnutí úrovně.

```
// nepodporovaný zápis
@(V(in))

// možný zápis
@(cross(V(in)))
```

Jinak je ale možné pomocí PSL vytvořit řadu různorodých konstrukcí. Ty mohou a nemusejí využívat sekvenční operátory, mohou být citlivé na analogovou nebo digitální událost, nebo nemusejí evaluační podmínku obsahovat vůbec (funkční pouze při kontrole digitálních signálů). Vyhodnocovat přitom lze digitální signály, reálná čísla (typ "real" nebo "wreal") nebo hodnoty analogových objektů, které souvisejí s napětím. Tvrzení pro kontrolu proudů vyžadují odlišný přístup, ve kterém je potřeba navíc integrovat funkci $cgav.

Následující příklady popisují základní funkci testovaného komparátoru. První tvrzení kontroluje nepřítomnost chybového signálu nepřetržitě po celou dobu simulace. V případě detekce nenulové hodnoty tohoto signálu bude simulátorem okamžitě vyhlášena chyba. Další dvě tvrzení se věnují překlápění komparátoru.

```
// tvrzení bez evaluační podmínky – musí být platné po celou dobu simulace
// (použitelné pouze pro kontrolu digitálních signálů,
// které lze vhodně generovat v rámci modelační vrstvy)
assert always (!Fault) ;

// tvrzení s evaluační podmínkou
// kontrola překlopení komparátoru (každý směr jinou metodou)
assert always (( ( V(vin_p) > V(vin_n)) -> next_e [0:100] (V(Out) < (V(gndd) + 0.5))) @(posedge(clk)) ;
assert always (( {(V(vin_p) < V(vin_n))[*100]} |=> (V(Out) > (V(vddd) - 0.5))) @(posedge(clk)) ;
```

V prvním případě je použit logický implikační operátor '->' v kombinaci se základním operátorem ze skupiny 'next*'. Implikační operátor v podstatě zastupuje

standardní logickou podmínku typu IF-THEN. Tedy pokud je výraz na jeho levé straně vyhodnocen jako TRUE, musí být následně splněn i výraz na jeho pravé straně. Operátorem ‘next_e‘ (zkratka pro „eventualy“) je následně definováno, že během následujících 100 evaluačních cyklů bude alespoň jednou splněna zadaná podmínka. Tvrzení tak ve zkratce definuje, že pokud je napětí uzlu ‘vin_p‘ vyšší než napětí uzlu ‘vin_n‘, do určité doby musí výstupní napětí poklesnout pod úroveň 0,5 V.

Ve druhém případě je pro popis opačného směru překlopení použit lehce odlišný zápis, ale především zápis pomocí sekvencí (složené závorky {} a sekvenční operátor ‘|=>‘). Tvrzení ve zkratce definuje, že výstupní napětí musí být blízké napájecímu napětí, pokud je nepřetržitě po dobu 100 evaluačních cyklů napětí uzlu ‘vin_p‘ nižší než napětí uzlu ‘vin_n‘,

4.3.1 Omezení PSL

Při psaní tvrzení, ať už přímo nebo parametricky, jsou používány názvy signálů, které mají být kontrolovány (v předchozím zápisu např. signál ‘vin_p‘ – odpovídá vstupnímu signálu komparátoru z testovacího schématu). Definice názvů a typů těchto signálů jsou buď součástí bloku, kam je PSL zápis vnořen, nebo jsou definovány v rámci verifikační jednotky. U obou způsobů je ale nutné, aby se názvy kontrolovaných signálů shodovaly se jmény použitými v tvrzeních. Tato skutečnost snižuje univerzálnost zápisu a schopnost znovupoužitelnosti u jiných, podobných bloků.

S tímto faktem souvisí i další otázka, a to jakým způsobem vytvářet doménově nezávislá tvrzení, pokud musí být použito určité jméno signálu, který ale při různých testech může měnit svůj typ? Pokud je PSL tvrzení připraveno pro vyhodnocení analogové hodnoty, nelze v něm vyhodnocovat signál logického typu. Pro pokrytí různých variant je tak potřeba více tvrzení – tvrzení pro vyhodnocení analogové hodnoty, tvrzení pro logickou hodnotu a případně tvrzení pro zpracování reálného čísla. Sepisovat pro každou vlastnost několik typů tvrzení ale není efektivní.

Jiným možným přístupem je použití systémové funkce $cgav. Ani ta ale není řešením na výše uvedený problém, protože parametrem této funkce může být pouze

analogový objekt ([17], [19], [16]). V opačném případě bude návratová hodnota funkce nulová. Je to tedy stejná situace, jako při přímém přístupu k analogovému objektu.

Při použití funkce $cgav by sice do jisté míry mohly být prospěšné její pomocné funkce $cds_analog_is_valid a $cds_analog_exists (kontrolují, jestli cílový objekt vlastní analogové výpočetní jádro, [17]), nevyřeší se tím ale nutnost soupisu více druhů tvrzení. Tuto skutečnost zřejmě při použití PSL nelze nijak odstranit.

Uživatelská příručka prostředí Spectre AMS Designer ([19]) pak popisuje řadu dalších omezení v souvislosti s použitím PSL v tomto simulátoru.

PSL tvrzení, které ve své logické vrstvě obsahuje analogový objekt, musí mít jednu evaluační událost. Tvrzení, které tuto událost neobsahují, které ji obsahují jen částečně, nebo které mají více těchto událostí, nejsou podporovány.

```
electrical sig1, sig2 ;
reg a, clk, clk1, clk2 ;

// bez evaluační události
assert always (V(sig1) > V(sig2)) ;

// částečně evaluované tvrzení (evaluační událost se týká pouze signálu “a“)
assert always {V(sig1 ; V(sig2)} |=> {a} @(cross(v(clk))) ;

// více evaluačních událostí
assert always {V(sig1) ; V(sig2)} @(cross(v(clk1))) |=> {a} @(cross(V(clk2))) ;
```

Pokud je evaluační událost tvrzení citlivá na úroveň digitálního signálu, není možné v logickém výrazu tohoto tvrzení použít analogový objekt.

```
electrical sig1, sig2 ;
reg a, b, clk ;

assert always ({V(sig1) ; a} |=> {V(sig2) ; b}) @(clk) ; // nesmí obsahovat analogové objekty
```

Zřejmě nejkritičtější omezení se ale týká použití verifikačních jednotek. Přestože verifikační jednotky vázané na moduly ve svých výrazech mohou obsahovat analogové objekty, jednotky vázané na konkrétní instance nikoli. Toto omezení v podstatě znemožňuje selektivní výběr kontrolovaných instancí smíšené domény. Nelze tak kontrolovat pouze části obvodu, které jsou zájmem určitého testu.

```
// modulová vazba – projeví se u všech instancí modulu “test“
vunit myvunit(test) {
// může obsahovat analogové objekty
}

// vazba na konkrétní instanci
vunit myvunit(top.mytest) {
// nesmí obsahovat analogové objekty
}
```

4.4 Metoda SVA

SVA, tedy „SystemVerilog Assertion“, je zažitý název pro využití metody ABV prostřednictvím standardizovaných konstrukcí jazyku SystemVerilog. Samotný jazyk SystemVerilog je definován normou IEEE Std.1800, jejíž celý název zní „IEEE Standard for SystemVerilog - Unified Hardware Design, Specification, and Verification Language“. Už z názvu je tak patrné, že se jedná o ucelenou normu, která se nezabývá pouze modelováním obvodů, ale i verifikačními záležitostmi. Poslední vydaná verze této normy se pak datuje na prosinec roku 2017.

Zápis tvrzení pomocí SVA se od PSL v zásadě příliš neliší. Drobné rozdíly v syntaxi a způsobu zápisu jednotlivých tvrzení jsou samozřejmé, nejsou ale nijak razantní a celková podoba mezi oběma přístupy je evidentní. V obou případech se jedná o strukturovaný zápis dělený do stejných vrstev – tedy logické, časové, verifikační a modelační. Následující příklad zobrazuje tvrzení popsané jazykem PSL se syntaxí Verilog a metodou SVA:

```
PSL:
   chk_mutex: assert always !(rd && wr) @(posedge clk);
SVA:
   chk_mutex: assert property (@(posedge clk) !(rd && wr)) ;
```

V příkladu je opět zobrazena základní konstrukce tvrzení, kterou zpravidla tvoří identifikátor, verifikační direktiva, evaluační událost a samotná kontrolovaná vlastnost. Vlastnost může být vyjádřena, stejně jako u PSL, pouze jednoduchým logickým výrazem, nebo komplexní sekvencí signálových průběhů. Sekvence a vlastnosti lze opět definovat individuálně a parametricky.

4.4.1 Akční bloky

Co má metoda SVA oproti jazyku PSL navíc, jsou akční bloky. Tyto bloky specifikují akce, které se vykonají bezprostředně po vyhodnocení tvrzení. Akční bloky jsou nepovinou součástí tvrzení a jsou častěji využívány při negativním vyhodnocení tvrzení. Na následujícím příkladu je zobrazeno tvrzení s akčním blokem, který v případě porušení tvrzení zastaví simulaci a vypíše chybovou hlášku s 2. nejvyšší úrovní kritičnosti:

```
chk_mutex: assert property (@(posedge clk)
    !(rd && wr))
else begin
    $error("Výpis chybové hlášky."); // další jsou $fatal, $warning, $info a $display
end
```

4.4.2 Aplikace SVA

Díky tomu, že jsou verifikační direktivy plnohodnotnou součástí syntaxe jazyka SystemVerilog, je možné zapisovat tvrzení přímo do zdrojového HDL kódu kontrolovaného bloku. Tato tvrzení tak není nutné „skrývat" v komentářích kódu. Na druhou stranu, při použití takového zápisu budou tvrzení vyhodnocována při každé simulaci. S tím souvisí také fakt, že při tomto způsobu aplikace není potřeba nijak donastavovat analogové simulační prostředí ADE. Absence dodatečného simulátorového přepínače ale eliminuje možnost jednoduché volby, zda-li mají být tvrzení vyhodnocována, nebo ne.

Dalším způsobem, jak zapisovat SVA tvzení, jsou samostatné moduly. Jedná se o klasickou SystemVerilog konstrukci modulu, který obsahuje pouze vstupní porty a v porovnání s PSL jde o analogii verifikační jednotky. Tyto moduly tak obsahují pouze doplňkový modelační kód a jednotlivá tvrzení. V takovém případě už ale je nutné simulaci doplnit o další prvky.

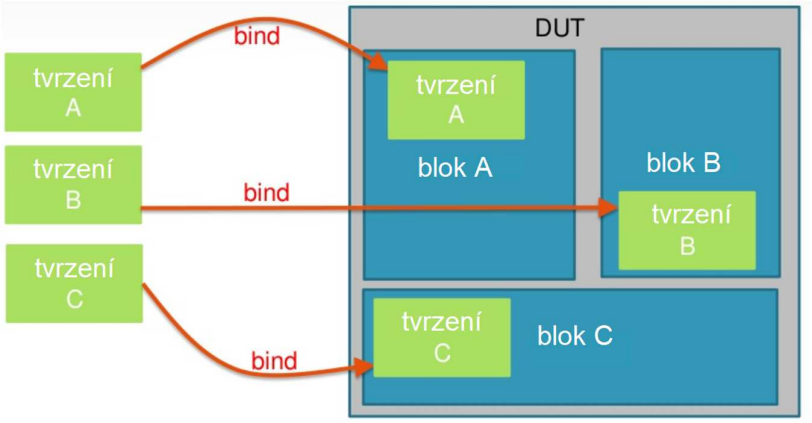
Možnosti, jak s těmito moduly pracovat, jsou dvě. V prvním případě je možné modul využít jako simulační monitor. To znamená, že instance modulu bude přímo součástí simulačního schématu i hierarchie a na její vstupní porty budou připojeny kontrolované signály. Kontrolovat se přitom dají signály ze všech úrovní hierarchie

obvodu, které mají schématickou reprezentaci. Vnitřní signály behaviorálních modelů jsou při tomto přístupu nedostupné, protože modely jsou v rámci hierarchie zpracovávány jako „černé skříňky“ a jejich vnitřní signály tak nemají zastoupení ve vygenerovaném „netlistu“.

Druhým způsobem je externí instancování do kontrolovaných modulů, nebo do jejich konkrétních hierarchických instancí. Pro tuto operaci se v jazyce SystemVerilog nachází klíčové slovo “bind“ a zápis takového příkazu vypadá následovně:

bind BlockA AssertionsA # (<mapování parametrů>) I_assertionsA (<mapování portů>) ;

Tento zápis vytvoří v cílovém modulu ‘BlockA‘ instanci modulu ‘AssertionsA‘, která bude mít název ‘I_assertionsA‘. Výhoda tohoto přístupu spočívá v tom, že je instancování provedeno externě a není nutné nijak upravovat cílový modul. Monitorovací moduly je pomocí této techniky možné zavádět přímo do behaviorálních modelů v jakékoli úrovni hierarchie, což umožní kontrolu jejich vnitřních signálů.

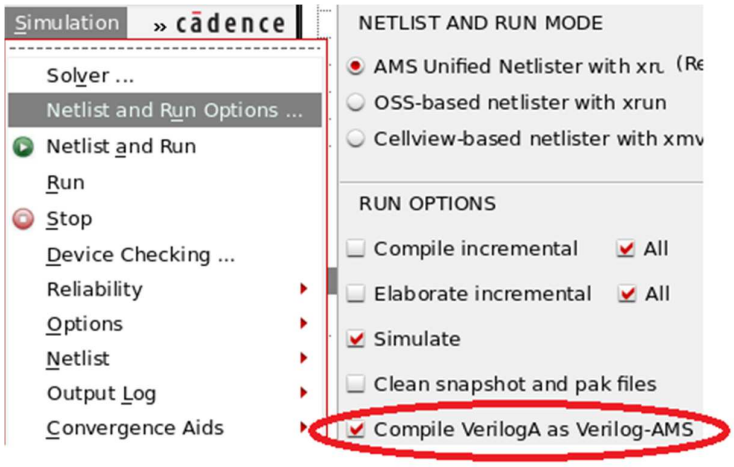


Obrázek 14: Znázornění techniky externího instancování [20]

Samotný zápis příkazu “bind“ je pak umístěn v samostatném modulu, který je hierarchicky nadřazený cílovému modulu. V rámci simulačního schématu a hierarchie je tak původní simulační monitor nahrazen modulem, který zajišťuje externí instancování. Je přitom důležité, aby tento modul byl součástí hierarchie a následného vygenerovaného „netlistu“. Pouhé zavedení tohoto modulu do simulace jako externího souboru nezpůsobí externí instancování.

4.4.3 Omezení spojená s externím instancováním

S touto technikou ale souvisejí i jistá omezení, která je potřeba zohlednit. První z nich je nefunkčnost v případě užití na analogové modely psané v jazyce Verilog-A. Tuto skutečnost lze jednoduše obejít tak, že se v analogovém simulačním prostředí ADE v nastavení spouštěcích parametrů povolí kompilace Verilog-A modelů jako Verilog-AMS modelů (viz. obrázek níže).



Obrázek 15: Změna způsobu kompilace Verilog-A modelů

Další omezení souvisí se schématickými reprezentacemi bloků. Pokud je totiž v simulační hierarchii více instancí nějakého konkrétního bloku, které jsou reprezentovány schématem, aplikovaná tvrzení zůstanou nefunkční. V takovém případě, a zřejmě i obecně, je lepší jako cílový modul externího instancování zvolit hierarchicky nadřazený blok a k testovaným signálům v rámci mapování přistoupit pomocí hierarchického zápisu.

```
// externí instancování s přímým mapováním
bind TOP.BlockA AssertionsA I_assertionsA ( .sigA (sigA) ) ;

// externí instancování s hierarchickým mapováním
bind TOP AssertionsA I_assertionsA ( .sigA (BlockA.sigA)) ;
```

S tím také souvisí další omezení. Pokud je cílový modul externího instancování reprezentován modelem v jazyce Verilog-AMS nebo SPICE, nelze pro mapování

signálů použít tzv. „dot-star“ (.*) zápis známý z jazyků Verilogu. Pokud bude ale obecně zažitým postupem hierarchické mapování, tato vlastnost pozbývá platnosti.

4.4.4 SVA v analogové a smíšené doméně

Využitelnost SVA je z funkčního pohledu velice podobná PSL. V tomhle ohledu jsou si oba přístupy rovny. Oba obsahují možnosti, jak definovat parametrické vlastnosti (property) a sekvence s využitím nejrůznějších sekvenčních operátorů, včetně dalších vnitřních funkcí ($rose, $fell, $stable, atd.).

V kontextu analogové a smíšené domény pak SVA využívá odlišný přístup pro zpracování obvodových veličin. V rámci SVA totiž není možné zpracovávat analogové hodnoty reprezentované typem “electrical“, ale pouze jejich obrazy v podobě reálných čísel. K mezidoménovému převodu je přitom možné využít propojovací moduly, které se běžně využívají při simulacích obvodů smíšené domény, nebo systémovou funkci $cgav.

Mezi výhody SVA pak patří možnost vytvářet doménově univerzální tvrzení. Jazyk SystemVerilog totiž popisuje způsob využití přímého programovacího rozhraní, tzv. „DPI“ (Direct Programming Interface), kterým je možné v zápisu SystemVerilog použít knihovny a funkce jiných programovacích jazyků, např. C.

Vhodnou kombinací DPI, $cgav a pomocné funkce $cds_analog_is_valid je možné vytvořit konstrukci, která nejprve zkontroluje, do které domény spadá kontrolovaný signál, aby mohla následně zvolit správnou metodu extrakce jeho hodnoty. Pokud je kontrolovaný signál součástí analogové domény, je možné využít funkci $cgav. V opačném případě je možné využít přímé vyčtení hodnoty, protože se nejedná o mezidoménový převod. [20]

Touto technikou by s největší pravděpodobností mělo být možné kontrolovat i obvodové proudy. Tato funkce ale v době psaní práce nebyla ještě otestována.

4.5 Porovnání PSL a SVA

Z pohledu ABV jsou oba jazyky v podstatě totožné. Ve prospěch PSL je možnost přístupu přímo k analogovým objektům kontrolovaného obvodu. To vylučuje nutnost použití propojovacích modulů a mezidoménového přečvodu hodnot. Na druhou stranu ale PSL skrývá určitá omezení. Mezi nejzásadnější pak patří:

- nutnost respektovat názvy signálů jednotlivých bloků,
- nemožnost vytvoření doménově nezávislých tvrzení,
- nemožnost selektivní volby kontrolovaných instancí.

PSL by se ale přesto mohlo uplatnit při psaní „low-level“ tvrzení, které se týkají vnitřního chování bloků. V takovém případě by byl PSL zápis vnořen přímo do kódu daného bloku, což by zneutralizovalo zjištěná omezení. Modelový i verifikační zápis by tak byly spolu na jednom místě a při aktualizaci modelu by mohla být i adekvátně upravena tvrzení. Jednotlivá tvrzení by se pak buď aktivovala přepínačem “-assert“, nebo by zůstala neaktivní. Jednalo by se tak o jednodušší přístup z pohledu správy a údržby, který by ale vyloučil použití PSL na schématické reprezentace obvodů.

V případě SVA je situace jiná. Vše, co lze vytvořit pomocí PSL, je možné realizovat i s SVA, přičemž SVA netrpí zjištěnými omezeními PSL. A ačkoli SVA nedokáže pracovat s analogovými objekty, je možné toto omezení obejít za pomoci mezidoménových propojovacích modulů. Případnou další možností je použití systémové funkce $cgav.

Dalším aspektem, který je ve prospěch SVA, je aktuálnost specifikace. Specifikace SVA, tedy IEEE Std.1800, se od roku 2012 dočkala své pravidelné pětileté aktualizace a její poslední vydání se datuje na konec roku 2017. Na druhou stranu, specifikace PSL (IEEE Std.1850) má své poslední vydání z roku 2010.

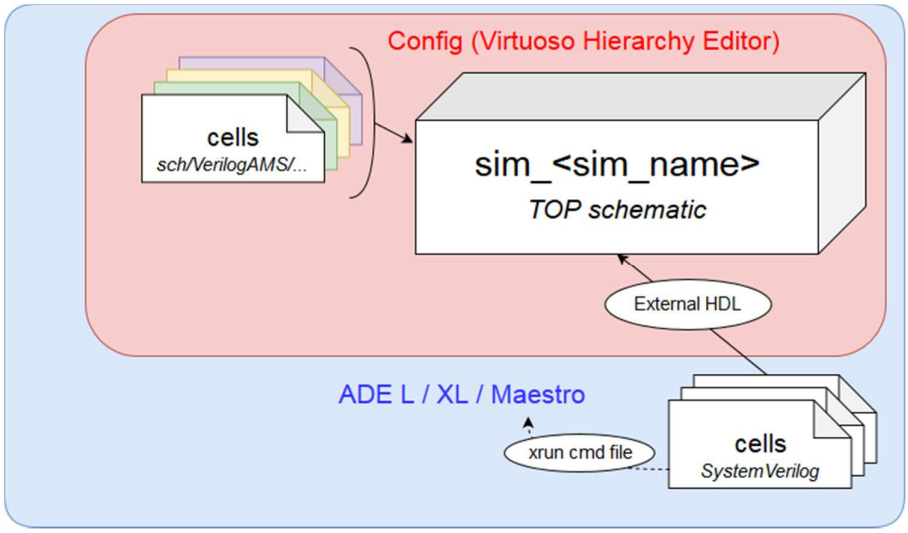
Díky těmto faktům je tak možné považovat SVA za metodu, která má v rámci ABV větší potenciál. Hlavně pak díky eliminaci omezení PSL a větší flexibilitě při použití. A i když má SVA také svá omezení, nejedná se o nic kritického, co by bylo v zásadním rozporu s plánovaným použitím v rámci společnosti SCG CDC.

5 Unifikované SVA prostředí

Dalším krokem při zavádění SVA v rámci ABV metody, je vytvoření vhodného strukturovaného simulačního prostředí, které zajistí znovupoužitelnost, přenositelnost a jednoduchou práci s tvrzeními. Takové prostředí musí vhodným způsobem sdružovat všechny simulační soubory, které jsou pro „Assertion-based“ verifikaci ve smíšené doméně potřeba. Konkrétně se jedná o zdrojové soubory modelů simulovaného systému, soubor popisující smíšenou hierarchii, nastavení simulačního prostředí a soubory související s SVA.

5.1 Souborová hierarchie a simulační prostředí pro SVA

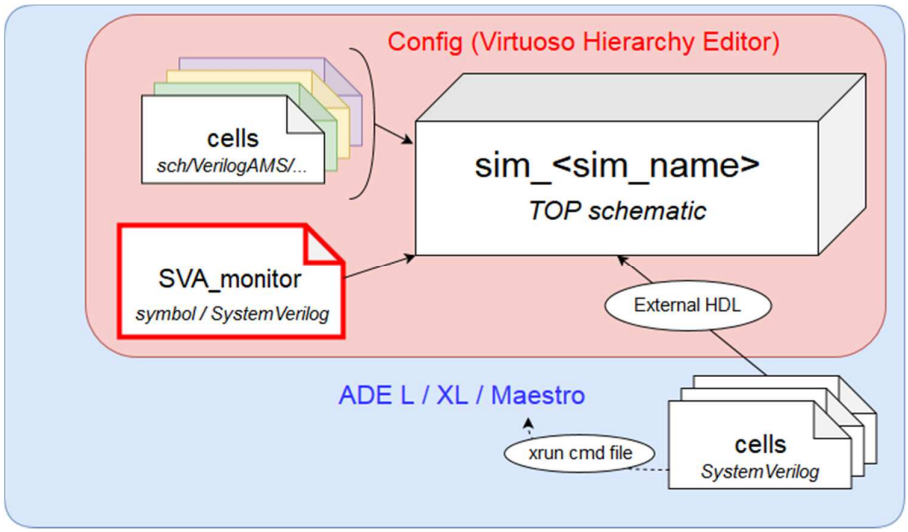
Pokud se SVA do testovací simulace aplikuje ve formě modulově vepsaných tvrzení, není z pohledu simulačního prostředí potřeba nic měnit. Nastavení simulace i obvodová hierarchie zůstávají stejné a tvrzení se budou vyhodnocovat při každém simulačním běhu. Znázornění takového prostředí je na následujícím obrázku.



Obrázek 16: Základní SVA simulační prostředí

Takový přístup ale není příliš efektivní. Navíc tímto způsobem nelze aplikovat tvrzení na bloky, které jsou při simulaci reprezentovány schématickým modelem. Jako lepší způsob užití SVA se tak jeví zápis tvrzení do samostatného modulu.

Jak je popsáno výše, zde jsou tedy k dispozici dva přístupy. První z nich využívá SVA modul jako simulační monitor. Tento modul je vložen do simulačního schématu, aby se stal součástí obvodové hierarchie. Na jeho vstupní porty se následně připojí kontrolované signály.



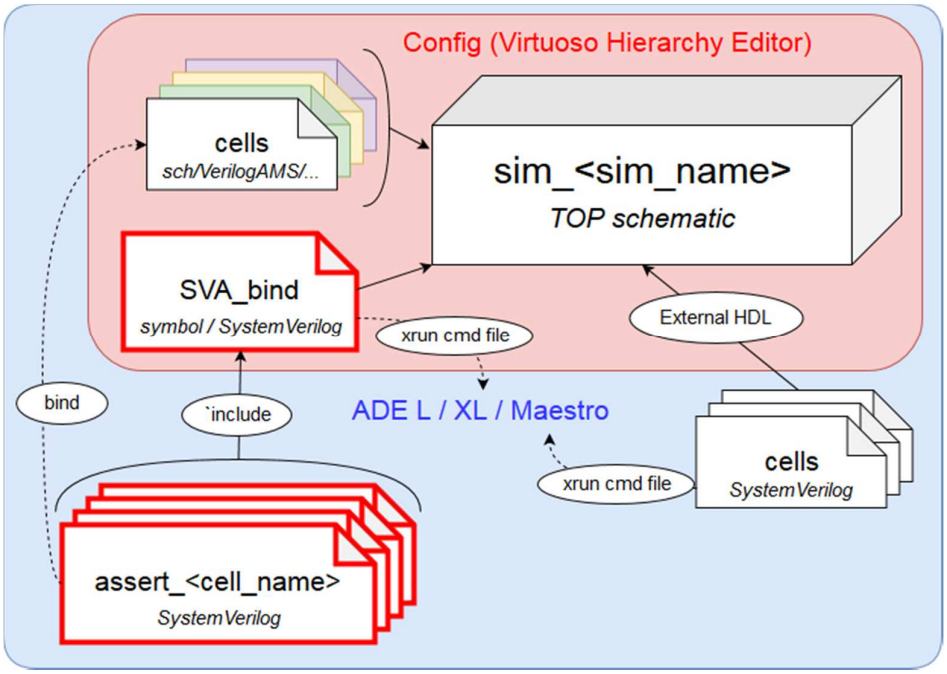
Obrázek 17: Aplikace SVA jako simulačního monitoru

Tento přístup přináší výhodu v tom, že jsou všechna tvrzení, která se týkají obvodu, na jednom místě. Nevýhodou ale je, že je tento modul použitelný pouze individuálně na konkrétní testovací schéma. Navíc připojení kontrolovaných signálů může být zmatečné a připojení vnitřních signálů behaviorálních modelů ani nelze realizovat.

Druhým způsobem je externí instancování. U tohoto přístupu je potřeba nadřazený modul, který bude do jednotlivých bloků externě zavádět SVA moduly. Tento nadřazený modul má podobu bloku nebo instance, která nemá vstupní ani výstupní porty, ale pouze sdružuje zápisy externího instancování. V rámci simulační hierarchie tento nadřazený modul zaujme místo SVA monitoru.

Samotný SVA monitor se pak rozdělí na množství menších modulů, které odpovídají jednotlivým blokům. Tvrzení v těchto modulech jsou tak uzpůsobena pro kontrolu jednotlivých bloků. Vhodným způsobem, jak tyto bloky navázat na nadřazený modul, je pak pomocí direktivy „`include“. Tedy pokud v rámci testovací simulace někde v obvodové hierarchii existuje blok, který má být kontrolován, je potřeba nalézt

nebo sepsat jemu odpovídající SVA modul. Tento SVA modul dále začlenit do nadřazeného modulu, který jej do kontrolovaného bloku externě nainstancuje.

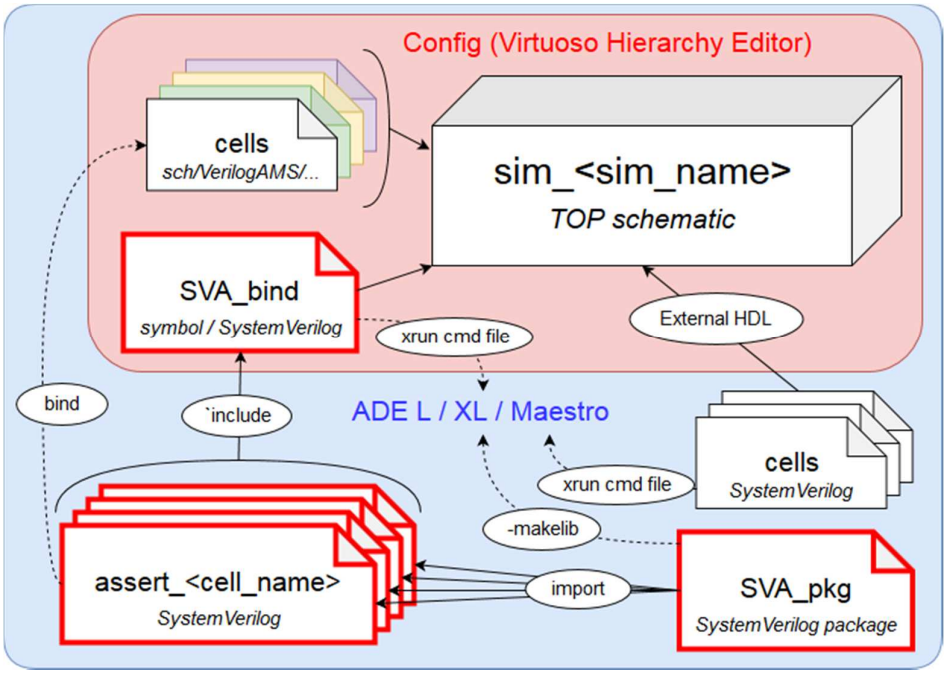


Obrázek 18: Externí instancování SVA

Výhodou této metody je to, že SVA moduly mohou být sepsány v předstihu a cíleně pro určitý blok. V rámci simulace se pak vývojář může rozhodnout, pro které bloky vyžaduje kontrolu a pro které ne. SVA moduly jsou navíc znovupoužitelné a přenositelné mezi simulacemi a projekty.

Pro zobecnění zápisu tvrzení pak vhodně poslouží parametrický zápis jednotlivých vlastností a sekvencí. Vlastnosti a sekvence se již dají do určité míry definovat univerzálně, což u přímo psaných tvrzení není úplně možné. Pro podporu univerzálnosti celé ABV je tak vhodné sestavovat tvrzení z předem definovaných vlastností a sekvencí. Tyto je pak už žádoucí uchovávat na jednom místě a nejlépe pod správou zodpovědné osoby a verzovacího systému. Formátem takového souboru může být například SystemVerilog package.

Výsledkem tohoto rozdělování a zobecňování jednotlivých prvků SVA je rozšířené simulační prostředí, které zobrazuje následující obrázek.



Obrázek 19: Rozšířené SVA simulační prostředí

5.2 SVA tvrzení

Vytvořené prostředí konečně dává prostor a jasné meze pro to, jak zpracovat množství různých tvrzení, která jsou na míru přizpůsobena jednotlivým blokům. Záměrně se jedná o systémové bloky, protože kontroly typu proud kanálem tranzistoru nebo maximální výkonové zatížení diody jsou vhodnější spíše pro metodu SOA.

Tvrzení, a obecně ABV v analogové a smíšené doméně, by se měly spíše soustředit na funkční vlastnosti jednotlivých bloků. Tvrzení by tak mohla kontrolovat např. rozdíl napětí na vstupech komparátoru, saturaci operačního zesilovače nebo stabilitu referencí. Tato tvrzení by se pak řadila do skupiny jakýchsi statických kontrol, kdy je hlídáno překročení mezních hodnot nebo rozptyl.

Do skupiny jakýchsi dynamických (sekvenčních) kontrol by se pak řadily tvrzení, které by byly závislé na průběhu určité sekvence. Zde by se tak řadily kontroly typu: včasná reakce výstupů při chybovém stavu, chování bloku při typických sekvencích (reset, start, apod.) nebo kontroly časových závislostí obvodu (latency, skew, slew rate, apod.).

Obecně ale mohou být tvrzení využity k definici správné funkce obvodu (podmínky a sekvence, které by měly nastat) nebo k popisu chybné funkce obvodu (podmínky a sekvence, které nesmí, nebo by neměly nastat – předpokládané chyby).

Další dělení by mohlo být na nepřetržitá tvrzení (Immediate Assertions), která ve svém zápise neobsahují evaluační událost a jsou tak vyhodnocována po celou dobu simulace. Tato tvrzení ale musí být umístěna uvnitř nějakého procedurálního bloku (např. always_comb). Jejich protikladem jsou tvrzení, která reagují na nějakou událost (Concurrent Assertions). Tato tvrzení obsahují evaluační podmínku.

5.2.1 Příklady tvrzení

Pro zlepšení čitelnosti kódu nebudou následující příklady tvrzení psány parametricky, ale tzv. řádkovým zápisem jejich finální podoby. V praxi by jednotlivé podmínky, vlastnosti a sekvence byly psány parametricky, a byly by součástí souboru, který tyto prvky slučuje. V rozšířeném simulačním prostředí je tento soubor nazván jako SVA_pkg.

V příkladech tvrzení jsou všechny použité signály typu ‘real‘ a ke kontrolovaným signálům stejného typu jsou připojeny přímo. Ke kontrolovaným signálům typu ‘electrical‘ jsou připojeny pomocí mezidoménových propojovacích modulů. U těchto příkladů tak není využita systémová funkce $cgav.

Křivky s výsledky jednotlivých simulací jsou uvedeny jako přílohy této práce.

5.2.2 Statická tvrzení

Uvedené příklady statických tvrzení jsou napsány jako nepřetržitá tvrzení. Tato tvrzení tak neobsahují evaluační podmínku a jejich vyhodnocení probíhá při každé změně některého z kontrolovaných signálů. Taková tvrzení ale musí být součástí procedurálního bloku. Stejná tvrzení by bylo možné zapsat i v podobě s evaluační podmínkou. První z trojice tvrzení navíc obsahuje funkci pro výpočet absolutní hodnoty výrazu, která je definovaná v rámci kódu jako textové makro.

```
`define abs(x) (((x)>0)?(x):-(x))
...
```

```
always_comb begin
   assert ( `abs(vin_p - vin_n) < 0.07) ; // napěťový rozdíl vstupů komparátoru menší než 70 mV
   assert ( (vdda > 4.8) && (vdda < 5.3) ) ; // kontrola stability napěťové reference
   assert ( !((Out_r >= 4.8) || (Out_r <= 0.4)) ) // kontrola saturace výstupu
end
```

První z tvrzení kontroluje napěťový rozdíl vstupů komparátorů. Pokud tento rozdíl přesáhne hodnotu 70 mV, simulátor vyhlásí chybu. Z výsledného detailního grafu simulace (příloha č. 2) je také možné pozorovat vliv mezidoménových propojovacích modulů. Tento vliv se projeví jako nespojitý průběh reálných hodnot. Tvrzení je pak vyhodnocováno při každé změně této hodnoty. Díky tomu je možné pozorovat drobné zpoždění při detekci chyby – oproti hodnotám analogových signálů ‘in‘ a ‘Vref_L‘ je chyba detekována se zpožděním zhruba 113 µs.

Druhé tvrzení hlídá rozptyl napájecího napětí komparátoru. Pokud toto napětí opustí definované meze, simulátor vyhlásí chybu. Podobné je i třetí tvzení, které kontroluje úroveň saturace výstupu komparátor. Toto tvrzení by se lépe uplatnilo u operačního zesilovače. Zde však slouží jako ukázka zápisu předpokládaného nevyžádaného chování obvodu. Takové chování je možné zapsat pomocí negace celého logického výroku.

5.2.3 Sekvenční tvrzerní

Jako příklad sekvenčního tvrzení je uvedena kontrola překlopení komparátoru. Toto tvrzení ve svém zápisu obsahuje evaluační podmínku. Proto je v příkladu uvedena i část, která generuje událost ‘clk‘. Samotné tvrzení pak ve své logické vrstvě slučuje analogové (reálné) i logické výrazy a množství sekvenčních operátorů a funkcí.

```
...
event clk ;

initial
   forever
      #10000 -> clk ;

assert property (@(clk) $past(vin_p < vin_n) ##0 ( vin_p > vin_n )[*30] |=> ##[0 : 620] !Out ) ;
...
```

U části věnující se generování události ‘clk‘ je potřeba znát globální simulační krok AMS simulátoru. Tento krok je definován v nastavení prostředí ADE a pro tuto simulace je to 1 ns. Událost ‘clk‘ tak spouští kontrolu tvrzení každých 10 µs.

Logická vrstva tvrzení se dělí na dvě poloviny, které jsou od sebe odděleny sekvenčním operátorem “|=>“. V případě platnosti první poloviny se následně očekává splnění i druhé poloviny tvrzení. Dokud však nedojde ke splnění první poloviny tvrzení, vyhodnotí se jeho stav jako neaktivní.

Celé tvrzení pak ve zkratce definuje následující chování – pokud bylo vstupní napětí v minulém evaluačním cyklu ($past) nižší než referenční a v současném evaluačním (##0) cyklu je vyšší než referenční, jedná se o protnutí signálů, přičemž vstupní napětí roste. Pokud tento stav setrvá nepřetržitě po dobu třiceti evaluačních cyklů (300 µs), je první polovina sekvence vyhodnocena kladně. Od tohoto okamžíku (|=>) se očekává, že během následujících maximálně 620 evaluačních cyklů (6200 µs) přejde výstupní signál ‘Out‘ do hodnoty log. 0.

Grafy s výsledky simulací jsou k nahlédnutí v přílohách č 5, 6 a 7. Na prvním grafu je zobrazena reakce komparátoru při pozvolném náběhu vstupního napětí. Z grafu je patrné, že překlopení výstupního sihnálu ‘Out‘ trvalo déle než 6200 µs, a proto bylo tvrzení vyhodnoceno jako chybné.

Na druhém grafu je zobrazena reakce komparátoru na prudkou změnu vstupního napětí. Zde je výstupní signál překlopen ještě dříve, než dojde k aktivaci tvrzení. To ale neznamená porušení žádné podmínky a tvrzení je okamžitě po splnění první poloviny tvrzení prohlášeno za platné. Třetí graf zobrazuje detail tohoto průběhu. Z grafu je patrné, že k překlopení výstupního signálu dojde během 150 ns.

Závěr

Při komplexnosti dnešních smíšených integrovaných obvodů, je pouhá analýza simulačních výsledků vývojářem, jako verifikační metoda, nedostatečná. Optická kontrola je zdlouhavá, vyčerpávající a vnáší do procesu verifikace lidský faktor, jehož vliv by měl být minimalizován. V rámci práce tak byly popsány různé metody, které se svou povahou hodí pro verifikaci smíšených integrovaných obvodů.

Z výčtu současných verifikačních metod pro smíšenou doménu lze za potenciálně dobře využitelné označit metody MDV a RDV. Vzhledem k jejich povaze se komplementární využití těchto přístupů přímo nabízí. Metodou RDV je možné určit, co se má verifikovat a metodou MDV je možné kvantifikovat verifikační postup. Lze tedy odpovědět na otázku, zda-li je verifikace obvodu dostatečná.

Velmi účinným přístupem pro kontrolu simulačních výsledků je tzv. „Assertion-base“ verifikační metoda, která je elementární součástí metody MDV. Její podstatou je automatizovat kontrolu simulačních průběhů a tím zkvalitnit hledání chyb. Hlavní silou této metody je schopnost stručného popisu chování obvodu pomocí sekvenčních operátorů. Popis takového chování umožňují jazyky PSL a SystemVerilog.

Tyto jazyky byly v rámci práce porovnány a prakticky otestovány, aby byly nalezeny meze jejich použitelnosti. Oba jazyky jsou z pohledu ABV téměř totožné. Výhodou PLS je schopnost zpracovávat analogové objekty. Na druhou stranu má PSL jistá funkční omezení, která jazyk SystemVerilog překonává. Proto se metoda SVA (SystemVerilog Assertion) jeví jako vhodnější přístup využití ABV.

Pomocí metody SVA bylo následně sestaveno simulační prostředí, které je možné pomocí definovaného postupu užít k verifikaci analogových a smíšených obvodů. I přes to všechno se ale výsledek verifikace odvíjí od kvality simulace, která je pouze tak dobrá, jak dobré jsou její vstupní stimuly.

Seznam obrázků

Seznam použitých zkratek

ABV	- Assertion Based Verification
AMS	- Analog-Mixed Signal
APR	- Automatic Place&Route
EDA	- Electrionic Design Automation
FPGA	- Field Programmable Gate Array
GDL	- General Description Language
HDL	- Hardware Description Language
IEEE	- Institute of Electrical and Electronics Engineers
IP	- Intelectual Property
MDV	- Metric Driven Verification
PSL	- Property Specification Language
RTL	- Register-Transfer Level
SERE	- Sequential Extended Regular Expression
SOA	- Save Operating Area
SPICE	- Simulation Program with Integrated Circuit Emphasis
SVA	- SystemVerilog Assertion
VHDL	- VHSIC Hardware Description Language
VHSIC	- Very High Speed Integrated Circuit

Použité zdroje literatury

[1] CHEN, Jess, Michael HENRIE, Monte F. MAR a Mladen NIZIC, BAILEY, Brian, ed. CADENCE DESIGN SYSTEMS, INC. *Mixed-Signal Methodology Guide: Advanced Methodology for AMS IP and SoC Design, Verification, and Implementation*. USA: Lulu, 2012, 408 s. ISBN 978-1-300-03520-6.

[2] VISHWAKARMA, Sumit. VERIFICATION HORIZONS: Simplifying Mixed-Signal Verification. *Mentor, a Siemens Business, leads in electronic design automation software - Mentor Graphics* [online]. Wilsonville: Mentor, A Siemens Busines, c2018, Nov. 2018 [cit. 2019-05-19]. Dostupné z: http://s3.mentor.com/fv/volume14-issue3.pdf

[3] Mixed-Signal Verification: Delivering accuracy and throughput for your unique designs. *EDA Tools and IP for System Design Enablement | Cadence* [online]. c2019, c2019 [cit. 2019-05-19]. Dostupné z: https://www.cadence.com/content/cadence-www/global/en_US/home/solutions/mixed-signal-solutions/mixed-signal-verification.html

[4] IKODINOVIC, Igor. Modeling for Analog and Mixed-Signal Verification. *ChipEstimate.com: Semiconductor IP Core Portal & Chip Design Resource* [online]. c2019, December 13, 2016 [cit. 2019-05-19]. Dostupné z: https://www.chipestimate.com/Modeling-for-Analog-and-Mixed-Signal-Verification/HDL-Design-House/Technical-Article/2016/12/13

[5] BRENNAN, John, Thomas ZILLER, Kawe FOTOUHI a Ahmed OSMAN. The How To'sof Advanced Mixed-Signal Verification. *DVCon Europe* [online]. Munich, c2019, 2015 [cit. 2019-05-19]. Dostupné z: https://dvcon-europe.org/sites/dvcon-europe.org/files/archive/2015/proceedings/DVCon_Europe_2015_T13_Presentation.pdf

[6] SAFARPOUR, Sean, Iain SINGLETON, Shaun FENG, Syed SUHAIB a Mandar MUNISHWAR. Formal Verification Tutorial: Breaking Through the Knowledge Barrier. *DVCON* [online]. San Jose, c2018, March 01, 2018 [cit. 2019-05-19]. Dostupné z: http://events.dvcon.org/2018/proceedings/slides/05T.pdf

[7] *Dictionary by Merriam-Webster: America's most-trusted online dictionary* [online]. Springfield, c2019 [cit. 2019-05-19]. Dostupné z: https://www.merriam-webster.com/

[8] SIMON, Sebastian, Joen WESTENDORP a Patrick LYNCH. UVM Mixed Signal Extensions: Sharing Best Practice and Standardization Ideas. *DVCon Europe* [online]. Munich, c2019, October 24, 2018 [cit. 2019-05-19]. Dostupné z: http://events.dvcon.org/events/proceedings.aspx?id=260--6-T

[9] HEATON, Nick. Maximizing Verification Effectiveness Using MDV. *Cadence: EDA Tools and IP for System Design Enablement* [online]. c2019, c2014 [cit. 2019-05-19]. Dostupné z: https://www.cadence.com/content/dam/cadence-www/global/en_US/documents/tools/system-design-verification/maximizing-metric-driven-ver-wp.pdf

[10] AYARI, Abdelouahab. What is Formal, Anyway?. *Mentor Graphics: Mentor, a Siemens Business, leads in electronic design automation software* [online]. November 2018 [cit. 2019-05-19]. Dostupné z: https://drive.google.com/drive/folders/1yK7TGnSmxsK3vEJ51FpVSKYFTCH8Ppsb

[11] IEEE STD 1800.2. *IEEE Standard for Universal Verification Methodology Language Reference Manual*. 2017. New York: The Institute of Electrical and Electronics Engineers, 2017. ISBN 978-1-5044-4000-4.

[12] Spectre Circuit Simulator User Guide: Product Version 5.0. *Electrical Engineering* [online]. New York: Columbia University, c2019, January 2004 [cit. 2019-05-19]. Dostupné z: https://www.ee.columbia.edu/~harish/uploads/2/6/9/2/26925901/spectre_reference.pdf

[13] FOSTER, Harry. MENTOR GRAPHICS CORPORATION. *Assertion-Based Verification: Introduction to ABV*. c2014, 42 s.

[14] HSIAO, Charlie a Paul WILLIAMS. Selecting the most productive SoC Design Verification Techniques. *Mentor Graphics: Mentor, a Siemens Business, leads in electronic design automation software* [online]. Setember 2018 [cit. 2019-05-19]. Dostupné z: https://drive.google.com/drive/folders/1yK7TGnSmxsK3vEJ51FpVSKYFTCH8Ppsb

[15] BERNA, Ateş a Ahmet JORGANXHI. Requirements Driven Design Verification Flow Tutorial. *DVCon Europe* [online]. Munich, c2019, October 24, 2018 [cit. 2019-05-20]. Dostupné z: http://events.dvcon.org/events/proceedings.aspx?id=260--12-T

[16] - CADENCE DESIGN SYSTEMS. *Tutorial on analog assertions and analog value fetch.* Version 0.8. 2010.

[17] CADENCE DESIGN SYSTEMS. *AMS Designer: Fetching value from an Analog Object using cds_get_analog_value or cgav*. c2013.

[18] FOSTER, Harry. MENTOR GRAPHICS CORPORATION. *Assertion-Based Verification: Introduction to SVA*. c2014, 43 s.

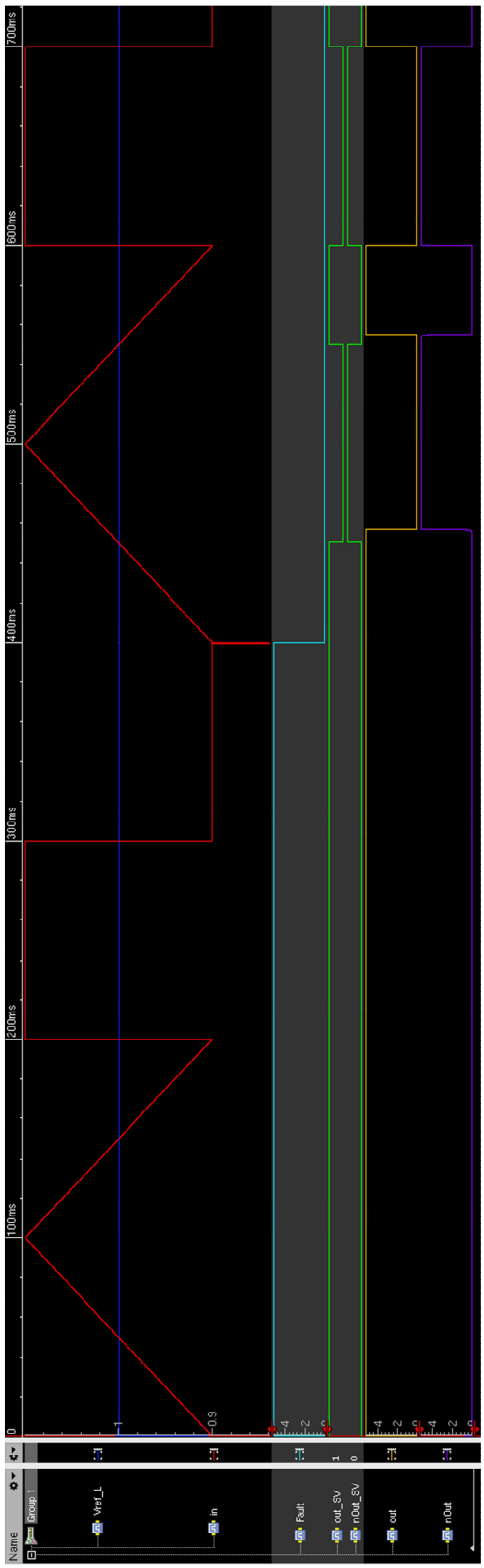
[19] CADENCE DESIGN SYSTEMS. *Spectre AMS Designer and Xcelium Simulator Mixe-Signal User Guide: Product Version 17.10*. October 2017. San Jose, c2000-2017.

[20] SANTONJA, Regis. Re-usable continuous-time analog SVA assertions. *Share and Discover Knowledge on LinkedIn SlideShare* [online]. c2019, May 2013 [cit. 2019-05-20]. Dostupné z: https://www.slideshare.net/regis_santonja/re-usable-continuoustime-analog-sva-assertions-slides

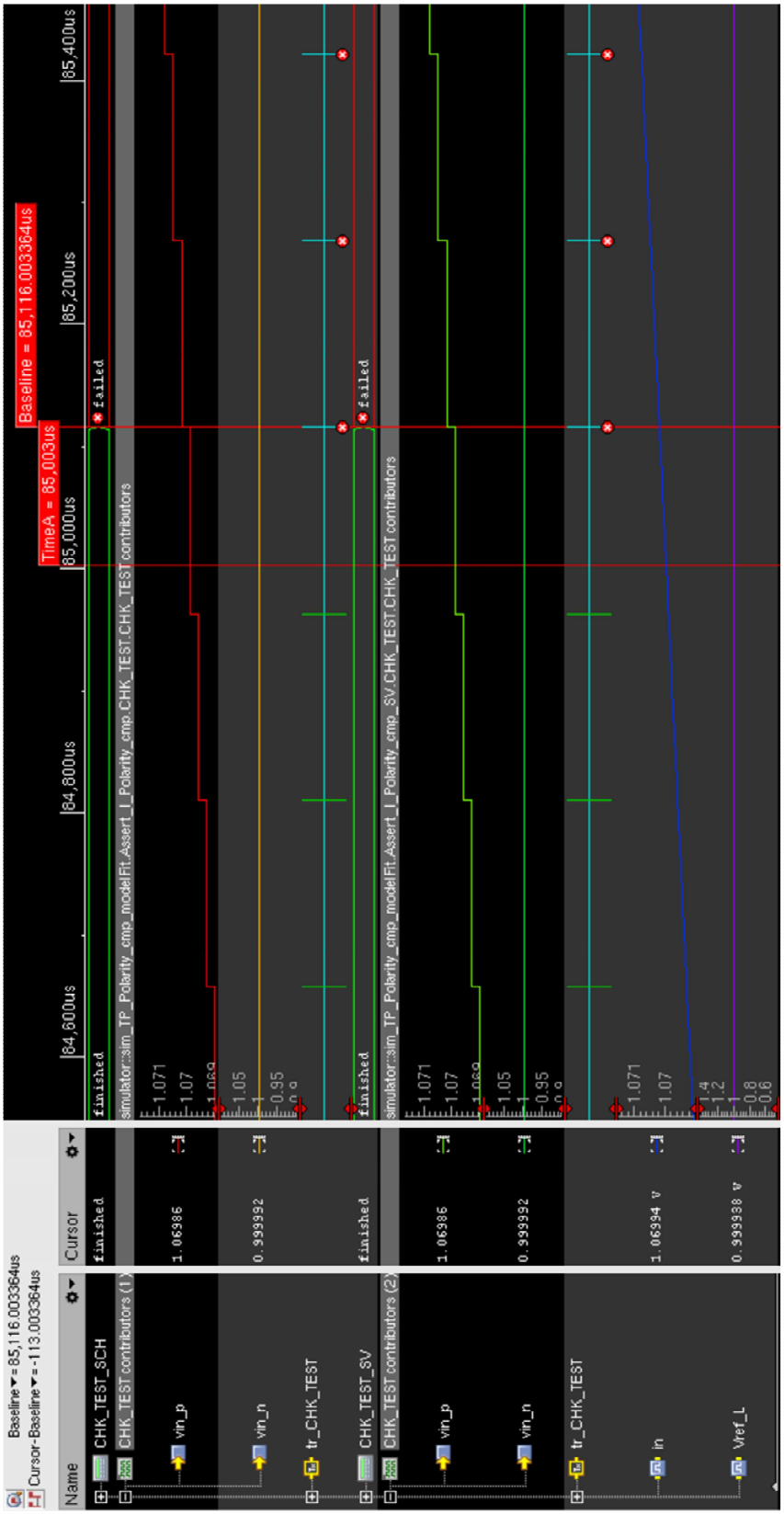
[21] IEEE STD 1800. *IEEE Standard for SystemVerilog— Unified Hardware Design, Specification, and Verification Language*. 2017. New York: The Institute of Electrical and Electronics Engineers, 2017, 1315 s. ISBN 978-1-5044-4509-2.

[22] IEEE STD 1850. *IEEE Standard for Property Specification Language (PSL)*. 2010. New York: The Institute of Electrical and Electronics Engineers, 2010, 188 s. ISBN 978-0-7381-6255-3.

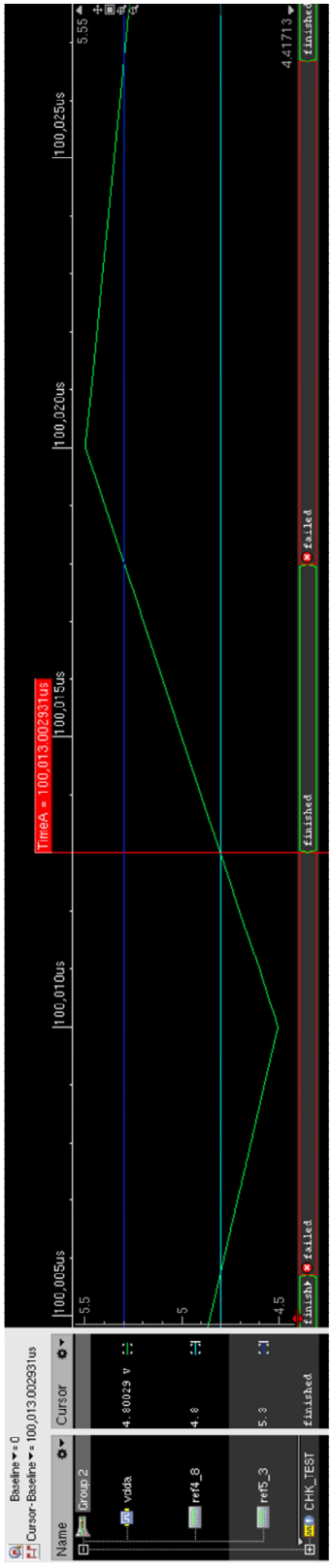
Příloha č. 1: Výsledek testovací simulace bez použití ABV



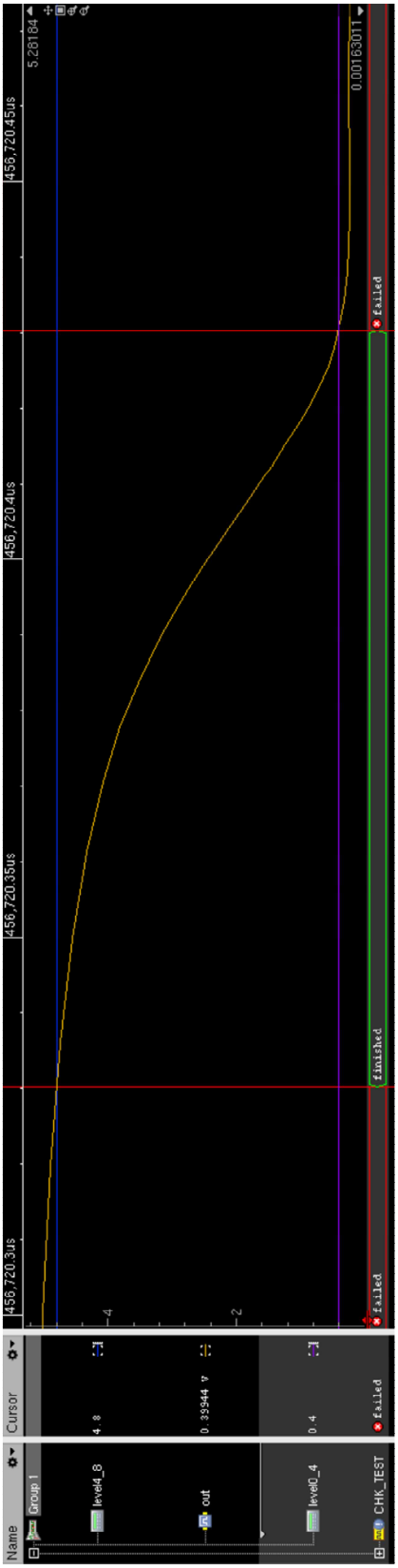
Příloha č. 2: Napěťový rozdíl vstupů komparátorů



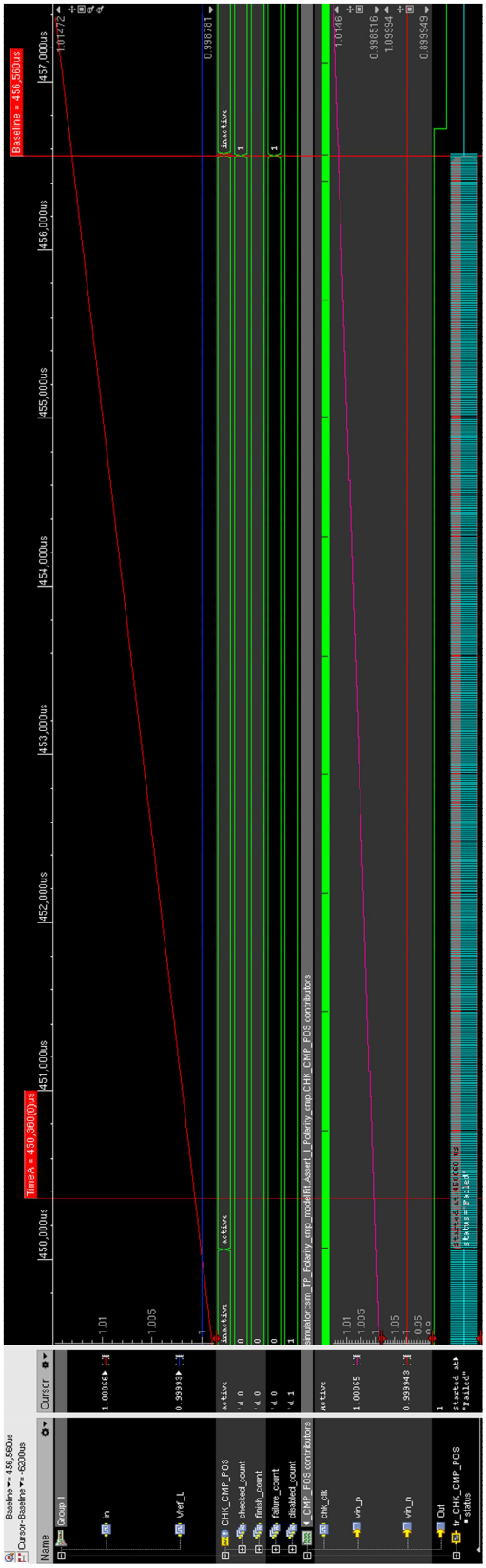
Příloha č. 3: Kontrola stability reference



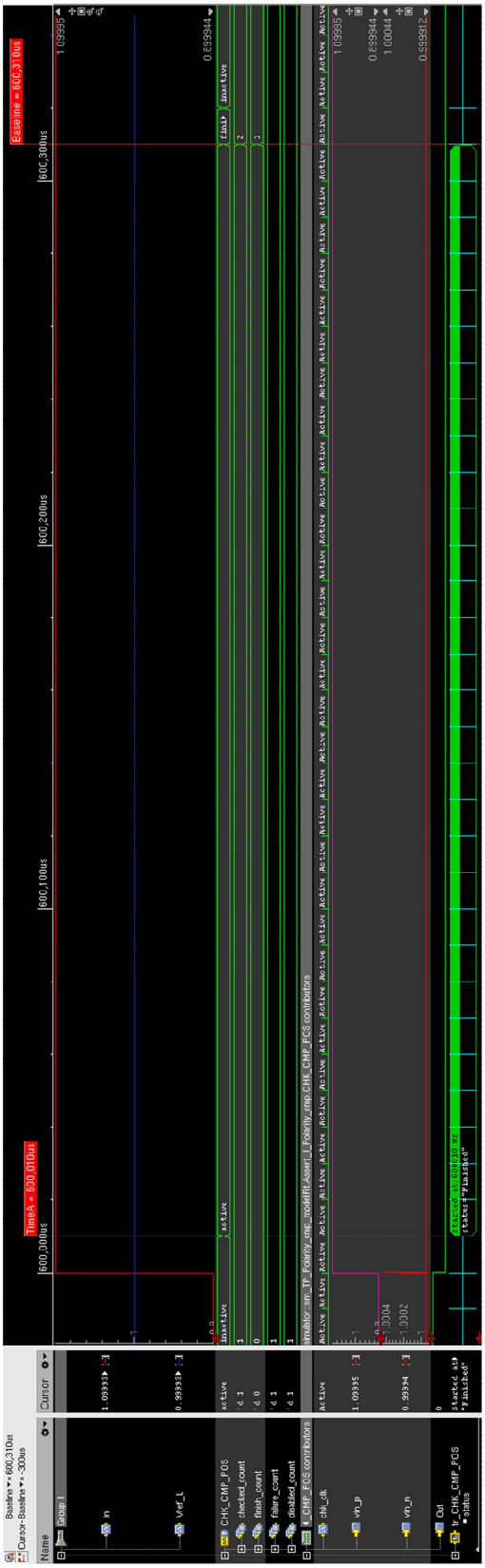
Příloha č. 4: Kontrola saturace výstupu



Příloha č. 5: Kontrola reakce komparátoru č. 1



Příloha č. 6: Kontrola reakce komparátoru č. 2



Příloha č. 7: Kontrola reakce komparátoru č. 3