Università degli Studi di Padova

DEPARTMENT OF INFORMATION ENGINEERING
UNIVERSITY OF PADOVA

Università degli Studi di Padova

Facoltà di Ingegneria

Corso di Laurea Magistrale in Ingegneria Elettronica

Studio ed applicazione delle tecniche di Real Number Modeling e Assertion Based Verification al progetto di circuiti mixed-signal

Study and application of Real Number Modeling and Assertion Based Verification techniques to mixed-signal circuit design

RELATORE

Ch.mo Prof. Andrea Neviani

CORRELATORE AZIENDALE

Ing. Andrea Scenini

TESI DI LAUREA DI

Pietro Vallese

Matr. N. 1036150

Anno Accademico 2014/2015

Indice

Elenco delle figure

Prefazione

L'elaborato che si va ad introdurre, tratta principalmente dello studio delle due tecniche del Real Number Modeling e dell'Assertion Based Verification, che recentemente stanno emergendo nel settore della verifica dei dispositivi Mixed-Signal.

Nel testo, inoltre, si presentano brevemente alcuni aspetti legati alla loro implementazione nell'ambito del progetto di un dispositivo in fase di sviluppo presso Infineon Technologies.

La prima parte della tesi svolge l'analisi della tecnica del Real Number Modeling. In particolare il primo capitolo contiene un approfondimento teorico, che in un primo momento introduce all'argomento dei modelli per le simulazioni in ambito Mixed-Signal e le tipologie di realizzazione attualmente disponibili.

Successivamente si sviluppa il soggetto principale presentando i presupposti di partenza, i linguaggi HDL – che costituiscono i mezzi di sviluppo –, alcuni casi particolari di applicazione ed i limiti della tecnica del Real Number Modeling.

Nel prosieguo si riportano alcuni elementi relativi all'attività di modellizzazione svolta, partendo da una presentazione sintetica, all'interno del Capitolo 2, delle caratteristiche del dispositivo di controllo per convertitori DC/DC scelto per il progetto.

I capitoli successivi illustrano due strumenti alla base dell'attività di modellizzazione svolta. Il primo è il package VHDL che definisce i tipi di dato utilizzati per rappresentare i segnali analogici. Ed il secondo è il netlister VHDL presente nel Cadence VHDL Toolbox, usato per generare in maniera efficiente la gerarchia del dispositivo, estraendo le informazioni dalle librerie già presenti in Cadence Virtuoso.

Infine, si espongono alcuni semplici esempi selezionati dai blocchi che compongono il dispositivo, e si descrive il metodo impiegato per la creazione del modello in VHDL di un convertitore DC/DC BUCK-BOOST con topologia a 4 interruttori, realizzato per ottimizzare la verifica del sistema.

Nella seconda parte l'esposizione si focalizza sull'introduzione dell'Assertion Based Verification nel campo dei circuiti Mixed-Signal.

Si avvia la trattazione descrivendo le attuali esigenze ad organizzare le tecniche di verifica anche nei progetti analogici ed effettuando una breve rassegna del'Universal Verification Methodology, che si sta imponendo nell'ambito digitale come standard per la Metric-Driven Verification (MDV).

Si prosegue la discussione andando a presentare le caratteristiche generali di una componente fondamentale del MDV, ovvero l'Assertion Based Verification

Nel Capitolo 10 si è voluto riportare una breve introduzione degli aspetti più significativi del SystemVerilog Assertion, il linguaggio scelto per la scrittura delle asserzioni.

Infine, si illustrano sinteticamente i risultati della ricerca trattata volta ad implementare lo strumento delle asserzioni per la valutazione dei segnali analogici.

Parte I.

Real Number Modeling

1. Elementi di teoria

1.1. Premessa

Nell'ambito della progettazione di circuiti integrati, l'attività di verifica ha subito una crescita esponenziale parallelamente all'aumento della complessità dei circuiti stessi. Nella verifica dei SoC la maggior parte del tempo impiegato, e del numero di calcoli eseguiti dai computer, sono occupati dalle simulazioni.

Nel settore dei circuiti digitali la verifica funzionale impiega in media il 70% del tempo della progettazione logica, ed anche in questo caso le simulazioni non sono sufficientemente veloci. Eppure le simulazioni RTL digitali sono di molti ordini di grandezza più veloci rispetto alle simulazioni analogiche basate su SPICE.

La maggioranza dei SoC odierni comprendono però sia componenti digitali che analogiche, e per questo si parla di categoria Mixed Signal (MS). Nelle simulazioni di circuiti MS, il tempo impiegato viene influenzato prevalentemente dalle componenti analogiche del progetto.

Ciò è dovuto al fatto che simulazioni analogiche e digitali utilizzano fondamentalmente diversi paradigmi. Mentre i simulatori digitali calcolano espressioni logiche (algebra booleana) basandosi su sequenze di eventi, i simulatori analogici, in ogni istante temporale considerato, risolvono matrici di equazioni differenziali rappresentanti il circuito analogico, in cui ogni componente del sistema può istantaneamente influenzare qualsiasi altro elemento nella matrice.

I test incentrati sui circuiti analogici richiedono pochi stimoli e grande precisione: essi hanno una ridotta velocità di esecuzione a causa appunto delle complesse equazioni richieste per questo livello di accuratezza. D'altro canto, i design digitali necessitano di molte sequenze di ingresso, atte a coprire il maggior numero di possibilità per verificarne il comportamento, ma necessitano di un minor livello di precisione.

Per i motivi appena sottoposti, la verifica di un intero SoC mixed-signal risulta essere un compito impegnativo ed all'aumentare della complessità non è più sufficiente accostare la pre-verifica dei singoli blocchi analogici e l'approccio a "black-box" tipicamente digitale.

Sempre più frequentemente gli errori funzionali sono causati da errori nelle interazioni tra i domini analogico e digitale causando ritardi nei tapeouts e silicon re-spins che possono causare un'ingente perdita economica.

In risposta a tale problematica si sono sviluppate varie tipologie di soluzioni, tra cui una vasta gamma di approcci alla modellizzazione dei circuiti MS. Le simulazioni SPICE sono ancora necessarie per la verifica dei singoli blocchi analogici, mentre nel momento in cui si passa a livello di sottosistema o di chip, si possono utilizzare modelli che ne emulano il comportamento (behavioral) in grado di fornire fino ad un 100x di aumento delle prestazioni.

Per quanto riguarda la verifica top-level di SoC, invece, gli ingegneri possono convertire i modelli analogici in Real Number Model (RNM), ovvero modelli che utilizzano il simulatore ad eventi dei sistemi digitali ma con segnali a valori reali. Questi modelli consentono di rimanere completamente all'interno dell'ambiente di simulazione ad eventi, che permette una velocità quasi alla pari delle simulazioni digitali.

1.2. Introduzione alla Modellizzazione MS

Prima di passare ad analizzare più approfonditamente la tecnica di Real Number Modeling, si vuole presentare brevemente il contesto generale della simulazione AMS. Come già enunciato precedentemente, in questo ambito si creano dei modelli per emulare il comportamento delle componenti analogiche le cui semplificazioni permettono ai simulatori di ridurre il tempo di calcolo.

1.2.1. FastSPICE

La necessità di velocizzare le simulazione analogiche era alla base dello sviluppo dei simulatori FastSPICE. Tuttavia, i risultati ottenuti raggiungevano un miglioramento massimo dell'1 o 2 ordini di grandezza rispetto ai modelli SPICE tradizionali. Questo non è risultato sufficiente per ridurre significativamente il tempo di simulazione della fase di verifica, in quanto si cerca di raggiungere almeno i 4-5 ordini di grandezza al fine di ottenere un livello sufficiente di verifica nei limiti di tempo assegnati.

1.2.2. Behavioral Analog Mixed Signal Modeling

Per raggiungere velocità di simulazione considerate ragionevoli, molti team di progetto MS impiegano il "Behavioral AMS Modeling". Esso è uno stile di modellizzazione che utilizza appositi linguaggi descrittivi, con i quali i simulatori possono risolvere equazioni differenziali algebriche (DAE). Con questo metodo si utilizzano gli stessi principi dei simulatori analogici ma si possono semplificare notevolmente le descrizioni circuitali e, di conseguenza, ridurre i tempi di simulazione.

Tipicamente viene utilizzato uno dei seguenti linguaggi:

- Verilog-AMS - Un linguaggio per modelli MS, basato sullo standard IEEE-1364 Verilog, che definisce comportamenti analogici e digitali, la cui semantica dispone di elementi sia per i modelli a tempo continuo che per simulatori ad eventi
- Verilog-A - È il sottoinsieme del Verilog-AMS esclusivamente per il tempo continuo, teso ai progetti analogici
- VHDL-AMS - Concettualmente simile al Verilog-AMS, questo linguaggio fornisce le estensioni per l'analogico e MS allo standard IEEE-1076 VHDL

Questo approccio può risultare dalle 5 alle 100 volte più veloce rispetto a SPICE. L'effettivo miglioramento dipende fortemente dall'applicazione e dal livello di dettaglio utilizzato nel modello. In quanto gli obiettivi di sviluppo dei modelli potrebbero spaziare dal delineare con precisione i comportamenti critici del circuito al descrivere il comportamento del circuito solo al livello di dettaglio necessario per verificare la corretta funzionalità del progetto.

1.2.3. Real Number Modeling

Il Real Number Modeling (RNM) simula le componenti analogiche esprimendo i valori con dati di tipo reale (numeri a virgola mobile), e considera il tempo come discreto utilizzando i meccanismi tipici dei simulatori ad eventi, impiegati nelle simulazioni digitali. A fronte di quanto appena detto, l'RNM è a tutti gli effetti un modello digitale e con questo metodo si possono guadagnare diversi ordini di grandezza (è realistico parlare di un vantaggio di più di 1000x in termini di velocità).

Per la verifica dei blocchi analogici e MS, il RNM può essere utilizzato per accelerare porzioni del percorso del segnale analogico ad alta frequenza – che può occupare la maggior parte delle simulazioni di verifica – mentre DC bias e porzioni a bassa frequenza rimangono modelli SPICE. Ma il più grande vantaggio del RNM è rappresentato dalla verifica globale del SoC, dove gli ingegneri possono rappresentare tutti i segnali elettrici con equivalenti in RNM e rimanere all'interno dell'ambiente di simulazione digitale. In quest'ultimo caso l'RNM consente di realizzare test di regressione per coprire la funzionalità dell'intero chip, pur mantenendo prestazioni elevate di simulazione.

La figura 1.1 mostra il grado di compromesso che intercorre tra la precisione di simulazione e le prestazioni di SPICE, FastSPICE, Behavioral AMS Models (Verilog-A / AMS e VHDL-AMS), RNM (SV-RNM), e simulazioni puramente digitali. Questi valori sono generici e possono variare in modo significativo in base alle diverse applicazioni.

Si noti la vasta gamma di precisione e le prestazioni che è possibile raggiungere con modelli comportamentali Verilog-AMS e VHDL-AMS. La simulazione puramente digitale può rappresentare un segnale analogico solo come un valore logico, ma potrebbe essere sufficiente per effettuare controlli sulla connettività nei SoC MS.

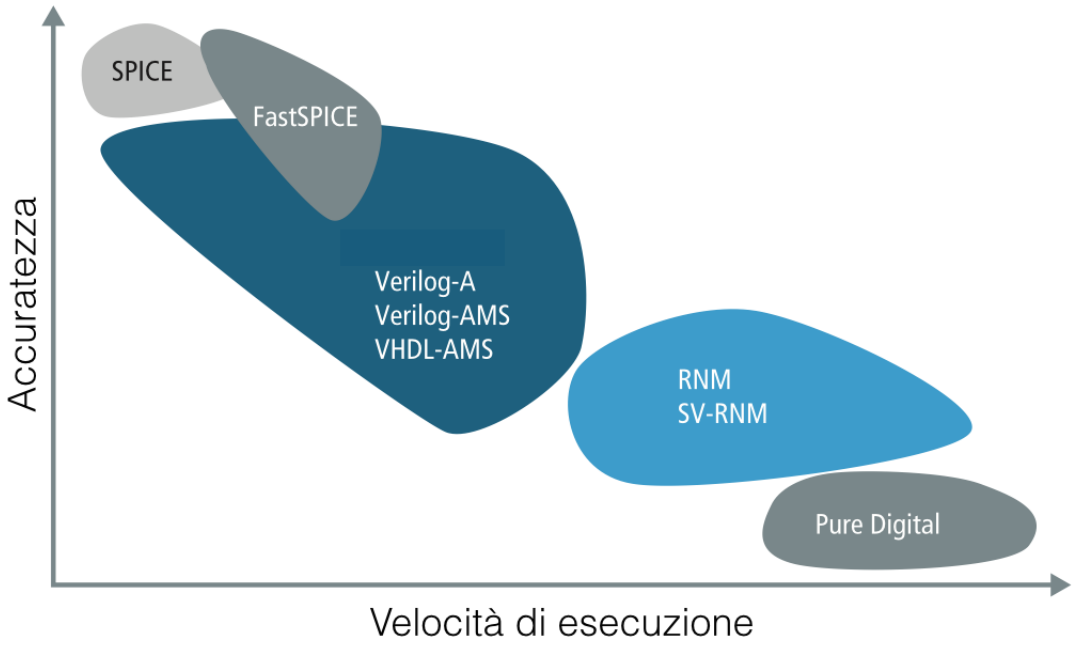

Figura 1.1.: Compromesso performance accuratezza [2]

1.2.4. Considerazioni generali

La creazione dei modelli analogici può risultare onerosa. I progettisti analogici sono avvantaggiati nella creazione di questi modelli, in quanto hanno familiarità con i propri circuiti. Ma a molti designer mancano le competenze di programmazione o le conoscenze richieste per costruire modelli comportamentali, ed un numero limitato di essi ha esperienza con il Verilog o il VHDL. I designer digitali possiedono tali competenze, ma hanno meno padronanza della progettazione dei circuiti analogici.

Un ulteriore fattore di considerevole importanza è l'impegno necessario a predisporre una simulazione e creare il modello. Mentre le simulazioni SPICE sono lente nell'esecuzione, esse sono relativamente facili da configurare. Il tempo necessario per creare un modello di alta qualità, richiede l'impiego di diverse ore di lavoro, fino a raggiungere la durata di alcune settimane nei casi più complessi e/o estesi.

Indipendentemente dal metodo scelto o dal tipo di astrazione, ogni approccio alla modellizzazione comporta una riduzione dell'accuratezza dei risultati che si ottengono dalla simulazione. Inoltre, i modelli scritti a diversi livelli di astrazione non sono equivalenti: più il livello di astrazione aumenta più ci si avvicina ad una descrizione del circuito priva della rappresentazione degli effetti non ideali. È fondamentale per chi si accinge alla

formulazione dei modelli, capire e sapere quali dettagli vengono tralasciati, poiché in fase di utilizzo questi modelli non potranno mettere in evidenza situazioni e/o fallimenti, che non siano stati previsti durante la progettazione del modello.

Quanto appena enunciato implica che più di un livello di astrazione deve essere impiegato nel processo di verifica di un design, a seconda di ciò che deve essere controllato.

1.3. Studio del RNM

L'utilizzo del Real Number Modeling per la verifica di design MS non è un approccio totalmente innovativo. È stato già impiegato intorno al 1988, nello stesso periodo in cui veniva standardizzato il VHDL-AMS. A quel tempo le esigenze erano diverse: non erano disponibili simulatori MS pertanto prima della fabbricazione, l'unico modo per risolvere problemi critici nelle interconnessioni tra le porzioni analogiche e quelle digitali di un design, era quello di creare dei modelli di alcune delle componenti analogiche in un simulatore ad eventi.

Oggi, invece, si possiedono linguaggi e simulatori che simulano accuratamente la parte analogica in ambiente MS. Di conseguenza, l'RNM viene principalmente impiegato per ottenere un aumento significativo di velocità durante la simulazione dei design AMS, anche se implica una perdita di precisione.

Grazie all'elevato livello di precisione ottenibile utilizzando altre tecniche di modellizzazione all'interno dell'intero flusso di verifica, tale perdita di accuratezza dei modelli RN non risulta essere così significativa come lo era un tempo.

I linguaggio attualmente disponibili per l'RNM consentono delle descrizione di tipologia signal dataflow, le quali appartengono ad un concetto della programmazione per cui l'esecuzione delle istruzioni è esclusivamente determinata dalla disponibilità degli argomenti d'ingresso delle istruzioni stesse.

In queste rappresentazioni i segnali rappresentano o un valore di tensione o un valore di corrente, ovvero si può imporre in una net una delle due informazioni ma non entrambe contemporaneamente.

Nei modelli per i simulatori analogici, le equazioni, comunque esse siano formulate, sono risolte solo implicitamente. Mentre il RNM richiede allo sviluppatore di scrivere sempre una soluzione esplicita, questo comporta che solo i sistemi con un'esplicita soluzione –

o almeno una buona approssimazione – possano essere modellati, e di conseguenza si devono formulare delle ipotesi sulle impedenze d'ingresso e/o uscita del blocco.

È possibile, con funzioni di risoluzione definite dall'utente, emulare alcuni effetti di impedenza, ma quando ciò è possibile le alternative tra cui poter scegliere sono le due seguenti: o la tensione è considerata indipendentemente dall'impedenza e la corrente è fissa; o è la corrente ad essere considerata indipendentemente dall'impedenza e la tensione è fissa. In altri termini, l'interdipendenza tra corrente e tensione non può essere modellata, ovvero il comportamento del modello non può essere influenzato dagli elementi a cui è connesso e quindi non è possibile descrivere gli effetti di accoppiamento tra i blocchi.

In compenso, in assenza di simulatori analogici, ci si deve preoccupare molto poco dei problemi legati alla convergenza.

1.3.1. Panoramica sui linguaggi disponibili

Molti linguaggi di descrizione dell'hardware supportano i tipi di dati real e wire-real, o wreal. Di seguito viene proposto per ogni linguaggio considerato una sintetica presentazione ed un breve elenco delle caratteristiche interessanti in relazione al RN modeling:

1.3.1.1. Verilog

Il Verilog fu inventato nel 1985 presso la Automated Integrated Design Systems (più tardi denominata Gateway Design Automation) come linguaggio proprietario. Nel 1990 l'azienda fu acquisita da Cadence Design Systems la quale rese di pubblico dominio il linguaggio, ottenendo uno standard con il nome IEEE 1364, con revisioni nel 1995, 2001 e 2005.

Gli inventori del Verilog volevano un linguaggio con una sintassi simile al C così che fosse familiare agli utilizzatori e facilmente accettato.

Allo stato attuale il Verilog assieme al più moderno ma assai più rigido VHDL possono venire considerati gli unici linguaggi utilizzati nel mondo della progettazione e simulazione digitale, con rispettivamente una quota di mercato pari a circa il 50%.

Vantaggi/svantaggi in relazione col RNM :

- Il tipo di dato real si può utilizzare solo come variabile all'interno dei moduli
- Non permette di definire porte di tipo real (richiede l'utilizzo di funzioni real2bits/bits2real)

- Non supporta gli stati X/Z (indeterminato/alta impedenza)
- Non permette più driver su net di tipo wreal

1.3.1.2. VHDL

VHDL è l'acronimo di VHSIC Hardware Description Language, dove VHSIC è la sigla di Very High Speed Integrated Circuits. Viene rilasciato nel 1987 come standard IEEE 1076; subisce il maggior aggiornamento nel 1993.

Esso è stato sviluppato in un progetto del Dipartimento della Difesa Statunitense con lo scopo di documentare il comportamento delle ASIC fornite da ditte esterne, il VHDL ha preso ispirazione dal linguaggio di programmazione Ada per necessità del dipartimento. Successivamente gli sviluppatori dei simulatori logici e degli strumenti di sintesi hanno fatto in modo che i loro strumenti leggessero ed utilizzassero direttamente le informazioni presenti in questa documentazione.

Vantaggi/svantaggi in relazione col RNM :

- Possibilità di definire porte di tipo real
- L'utente può definire nuovi tipi di dato (User-defined types – UDT)
- L'utente può definire nuove funzioni di risoluzioni (User-defined resolutions – UDR)
- Permette le connessioni a driver multipli
- Ha delle limitazioni sulle connessioni che si interfacciano ai blocchi analogici

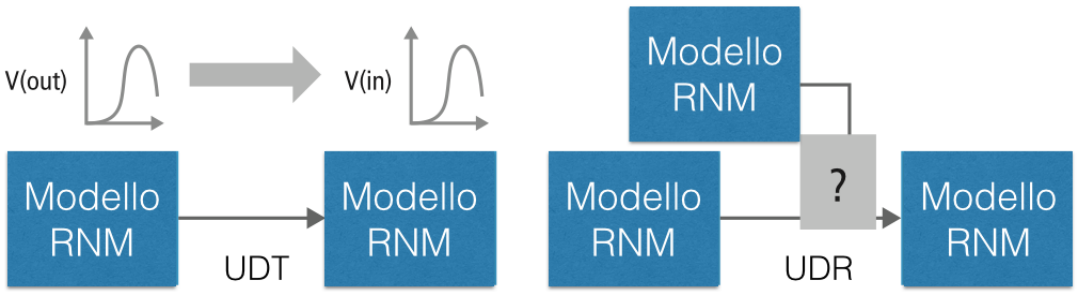

Figura 1.2.: UDT & UDR [2]

1.3.1.3. Specman/*e*

Specman è stato originariamente sviluppato da Verisity, una società con sede in Israele, acquisita da Cadence nel 2005. Ora è parte della suite di verifica funzionale del Cadence, "Incisive Enterprise Simulator", anche se Specman può ancora essere concesso in licenza come prodotto autonomo.

Specman è un tool EDA (Eletronic Design Automation) che fornisce uno strumento automatico per una verifica funzionale avanzata dei progetti hardware. Esso fornisce un ambiente per lavorare, compilare e debuggare testbench scritti nel linguaggio di verifica del'hardware *e*.

Vantaggi/svantaggi in relazione col RNM :

- Principalmente utilizzato per i testbench
- Presenta istruzioni predisposte alla generazione casuale, alla controllo della copertura della verifica e al test automatico
- Accesso diretto ai valori analogici (receive/drive)

1.3.1.4. Verilog-AMS (wreal)

Il Verilog-AMS è un derivato del Verilog, esso include estensioni per l'Analog Mixed Signal per definire il comportamento dei sistemi analogici e/o a segnale misto.

Anche se in questo particolare frangente interessa principalmente il tipo di dato wreal che può essere utilizzato nel simulatore ad eventi, nel linguaggio è disponibile la sintassi per simulatori analogici a tempo continuo.

Vantaggi/svantaggi in relazione col RNM :

- Facili interazioni con gli elementi analogici
- Connessioni dirette alla net di tipo electrical mediante i connect modules E2R and R2E (per l'utilizzo dei modelli in simulazioni con schematici analogici)
- Il compilatore partiziona le net del progetto tra i due risolutori analogico o digitale tramite il processo dell'associazione delle discipline – Disciplines association – che l'utente può eventualmente definire altrimenti risulta completamente automatico
- Supporto per driver multipli su net di tipo wreal

- Possibilità di specificare nuove funzioni di risoluzioni per il tipo di dati wreal
- Identifica gli stati X/Z

1.3.1.5. SystemVerilog

Il SystemVerilog combina in un unico linguaggio la sintassi per la descrizione dell'hardware e per la verifica hardware.

Lo sviluppo del SystemVerilog è iniziato con la donazione del linguaggio Superlog ad Accellera nel 2002. La maggior parte delle funzionalità di verifica si basa sul linguaggio OpenVera fornito da Synopsys.

Nel 2005, SystemVerilog è stato adottato come standard IEEE 1800-2005. Nel 2009, la norma è stata unificata con lo standard Verilog (IEEE 1364-2005), creando lo standard IEEE 1800-2009. La versione attuale è lo standard IEEE 1800-2012.

L'insieme di caratteristiche del SystemVerilog può essere diviso in due ruoli distinti:

- il sottoinsieme per la progettazione RTL è un'estensione del Verilog-2005; tutte le caratteristiche di tale linguaggio sono disponibili in SystemVerilog.
- il sottoinsieme per la verifica utilizza ampiamente tecniche di programmazione object-oriented ed è più legato a Java che al Verilog.

Un serio ostacolo all'adozione dell'utilizzo del RNM per la verifica a livello di SoC, sono le lacune del linguaggio SystemVerilog standard IEEE 1800-2009. Alcune delle limitazioni di questo standard includono:

- Non dispone di generazione di numeri pseudocasuali o strumenti per analizzare la copertura della verifica su segnali del tipo di dato real
- Non consente la definizione di porte bidirezionali di tipo real
- È consentito un solo driver per ogni porta (Non supporta le funzioni di risoluzione RN per più driver)
- Non consente connessioni tra le porte definite come real con altri linguaggi (Verilog-AMS, SPICE, VHDL)

A causa di queste limitazioni, le implementazioni RNM in SystemVerilog richiedevano più codice rispetto alle stesse realizzazioni che utilizzavano il wreal.

Per superare le lacune dello standard IEEE 1800-2009, la commissione ha rilasciato un nuovo manuale di riferimento del linguaggio (Language Reference Manual – LRM) con lo standard IEEE 1800-2012.

Le più interessanti si possono riassumere in:

- User-defined types (UDTs)
  - Permettono la definizione di net di tipo real
  - Permettono la definizione di net con più campi
- User defined resolution (UDRs)
  - Funzioni per risolvere gli UDT
  - Possibilità di associare funzioni particolari alle net di UDT
- Explicit Interconnects
  - nets senza il tipo di dato
  - Utile per connettere solo le porte

Il SystemVerilog (IEEE 1800-2012) risulterebbe uno dei linguaggi più indicati per l'applicazione della tecnica del Real Number Modeling, ma è da notare che attualmente non tutti gli ambienti di simulazione supportano completamente le nuove caratteristiche di questo standard.

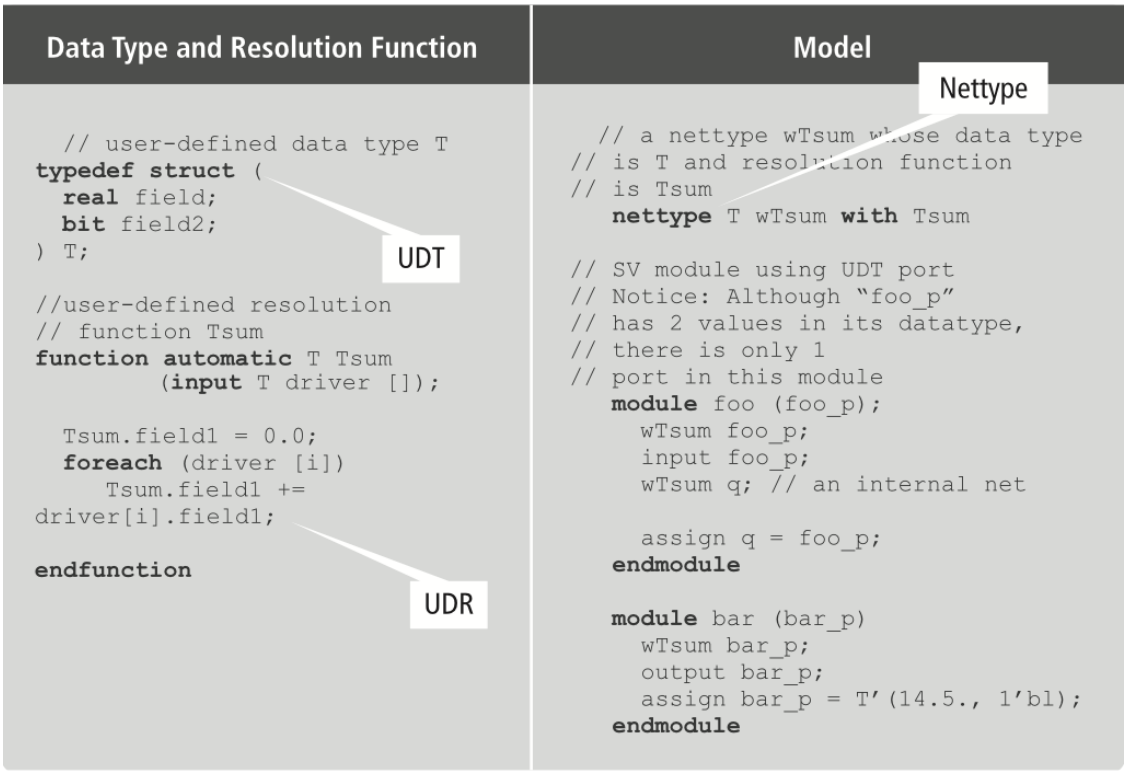

Figura 1.3.: Sintassi SystemVerilog per la definizione di UDT ed UDR [2]

1.3.2. Casi particolari

Quando si utilizzano tecniche di RNM, per default i segnali non sono continui, anche se i segnali sono di tipo a virgola mobile, così vengono utilizzati dei regimi di campionamento per rappresentare un segnale continuo nel dominio discreto.

Questo richiede una conoscenza a priori della massima frequenza di interesse poiché una rappresentazione a dati campionati contiene tutte le informazioni fino ad una data frequenza.

Inoltre la scelta del periodo di campionamento è una questione di compromesso, esso deve essere sufficientemente alto per prevenire effetti di aliasing e per produrre una buona approssimazione, ma non troppo elevato per evitare di gravare sul tempo di simulazione

I modelli AMS sono idonei a formulare il comportamento di un sistema in modo tale che il simulatore possa rappresentarli in termini di un insieme di equazioni algebriche differenziali (DAE) formulati sulla base delle leggi di Kirchhoff, e che presentano tempi e valori continui.

Questi paradigmi di modellizzazione non sono direttamente disponibili in RN modeling. Alcuni di essi possono essere approssimati, ma richiedono un lavoro notevole da parte di coloro ai quali è affidata la scrittura del modello.

Ad esempio per realizzare un valore continuo, è necessario approssimarlo con una funzione polinomiale. Tali approssimazioni sono definite dal numero di coefficienti costanti. Una rappresentazione chiamata Piece Wise Constant (PWC) richiede un valore a virgola mobile per ogni net o porta. Questi modelli sono semplici da scrivere e usare, al contrario di una rappresentazione Piece Wise Linear (PWL) che richiede due valori in virgola mobile per ogni net o porta: uno per il valore e uno per la sua derivata.

Questo non può essere fatto in tutti i linguaggi HDL, perché richiede che esso permetta dichiarazioni di record o strutture. Esistono delle alternative con altri coefficienti, ma sono troppo complessi da impiegare in applicazioni pratiche.

Nella simulazione analogica è possibile calcolare la risposta in frequenza di un circuito. Se la risposta in frequenza è nota, tale informazione può essere adoperata per formulare un approssimazione nel modello RN. Quando risulta possibile scrivere la funzione analogica come trasformata di Laplace (solo per alcuni sistemi lineari), si può ottenere una versione nel dominio della trasformata Z, campionando i segnali di ingresso e di uscita ad una certa frequenza. A questo scopo viene utilizzata la trasformazione bilineare:

$$s = \frac{2}{T}\frac{z-1}{z+1}$$

dove s è l'operatore complesso in frequenza e T il periodo di campionamento.

La limitazione intrinseca legata alla formulazione di un modello RN in termini della trasformata Z derivata da parametri misurati s, è l'assunzione di un sistema lineare: questo richiede la conoscenza a priori delle linearizzazioni che il progettista intende utilizzare. Se il sistema viola l'ipotesi di linearità, produrrà un comportamento errato, che non corrisponde affatto al circuito originale.

1.3.3. Limiti della tecnica

Come è già stato riportato, il Real Number modeling è limitato a rappresentazioni signal-flow, in cui deve essere presente una formulazione esplicita della funzione di trasferimento e si assumono l'impedenza di ingresso come infinita e l'impedenza di uscita pari a zero.

Il RNM non è tuttavia un sostituto per la simulazione analogica: non è adatto per le interazioni di basso livello che coinvolgono feedback in tempo continuo.

Non è previsto per i sistemi che sono molto sensibili alle interazioni non lineari delle impedenze d'ingresso/uscita.

In aggiunta, le conversioni real-to-electrical richiedono un'attenta considerazione: se una conversione è troppo conservativa, si riscontrerà un gran numero di punti temporali; se una è troppo lasca, si potrebbe verificare una perdita di precisione del segnale.

Questi casi appena enunciati costituiscono il motivo per cui risulta essere così importante capire quando l'RNM deve o non deve essere utilizzato in termini di limiti tecnici e di tipologie di informazioni (ad esempio, la precisione) che si perdono rispetto ad altre tecniche di modellizzazione.

Questo può essere fatto utilizzando un modello di riferimento che non perde alcun dettaglio, come ad esempio SPICE. Le descrizioni SPICE rappresentano il modello indipendentemente da come verrà impiegato, e forniscono un riferimento con il quale queste diverse astrazioni possono essere verificate.

È anche altamente raccomandato l'utilizzo delle asserzioni all'interno dei modelli al fine di evitare un uso improprio durante la verifica come, ad esempio, un controllo sulla massima frequenza in cui può essere utilizzato il modello.

1.3.4. Ricapitolazione

L'RN modeling produce enormi miglioramenti in termini di prestazioni, nei casi in cui funzioni correttamente. Questo è un avvertimento critico, poiché l'RN modeling è limitato nella sua applicabilità ed è difficile da eseguire con precisione.

Andando a riassumere:

- Il primo passo è quello di definire gli effetti da modellare e garantire che vi sia un modo esplicito per descriverli, dato che gli ingressi e le uscite saranno segnali digitali.
- Quando si calcola la soluzione del sistema, si è a conoscenza solamente di ciò che sta all'interno del modello, non di quello che sarà ad esso collegato.
- Tranne utilizzando qualche formalismo VHDL o SV-2012 con rapporti tensione/-corrente, gli ingressi dei modelli sono considerati come alte impedenze e le uscite basse impedenze.
- Gli ingressi e le uscite dei modelli devono essere univocamente distinguibili (niente porte bidirezionali).
- Spesso, il progettista non può scegliere il linguaggio durante la modellizzazione. Ma, come si è potuto osservare in precedenza, tutti i linguaggi disponibili non sono applicabili a tutte le caratteristiche che devono essere descritte. Quindi se il progettista ha la facoltà di scegliere il linguaggio, non dovrebbe poterlo fare fino a che non sia certo di quali effetti sono in procinto di essere modellati.
- Poiché i simulatori per l'RN Modeling eseguono solo al variare degli ingressi del modello o a seconda di passi temporali predefiniti, a volte risulta necessario inserire nelle descrizioni delle istruzioni per l'aggiornamento dei valori.

Si sono identificate tre diverse circostanze in cui le descrizioni in RN modeling possono essere utilizzate senza correre rischi:

- Modelli signal-flow che non richiedono un sistema conservativo.
- Modelli conservativi dove è sufficiente la modellizzazione solo della tensione o della corrente.
- Modelli per controllare la corretta propagazione dei dati, segnali di controllo e segnali di potenza durante la simulazione senza alcun dettaglio implementativo.

2. Caso di applicazione

Si è scelto di applicare la tecnica di Real Number Modeling ad un dispositivo in fase di progetto presso Infineon Technologies, realizzando la modellizzazione in linguaggio VHDL di ogni blocco analogico legato alla funzionalità del dispositivo. Di seguito si procede con una breve sintesi delle caratteristiche del prodotto

2.1. Descrizione del prodotto

Il dispositivo è un circuito integrato progettato per il controllo di un convertitore DC/-DC Buck-Boost non invertente realizzato con MOSFET ed un singolo induttore. Esso incorpora varie funzioni di diagnostica e protezione ed un'interfaccia SPI. Questo sistema è particolarmente indicato per pilotare LED ad alta potenza ($V_{LED} = 2 \div 52\,[V]$) con la massima efficienza ed il minimo numero di componenti esterni.

Il circuito offre più modalità di dimming sia analogiche che digitali. Esso presenta un generatore di clock interno configurabile o sincronizzabile ad una sorgente esterna e la frequenza di switching può essere impostata in un range compreso tra i 200 [kHz] ed i 700 [kHz].

Inoltre, il sistema nella modalità di regolazione di corrente dispone di un anello stabile gestito grazie a semplici componenti esterni di compensazione. Il dispositivo possiede anche una funzione di Soft Start per limitare i picchi di corrente ed overshoot di tensione all'avvio del sistema.

Riassunto delle principali funzioni:

- di protezione
  - dai sovraccarichi dei MOSFET dei ponti
  - dalla condizione di cortocircuiti del carico per scariche elettrostatiche (ESD)
  - dal sovrariscaldamento con spegnimento del dispositivo e riavvio automatico
- di diagnostica
  - informazioni di diagnostica attraverso le lettura delle “Error Flags” del SPI
  - rilevamento della condizione di assenza del carico (circuito aperto) durante il funzionamento
  - preavviso sulle condizioni di temperature del dispositivo e dello spegnimento per sovrariscaldamento
  - monitoraggio intelligente e funzioni avanzate fornite dalle informazioni sulla corrente che attraversa i LED e la corrente dal generatore d’ingresso

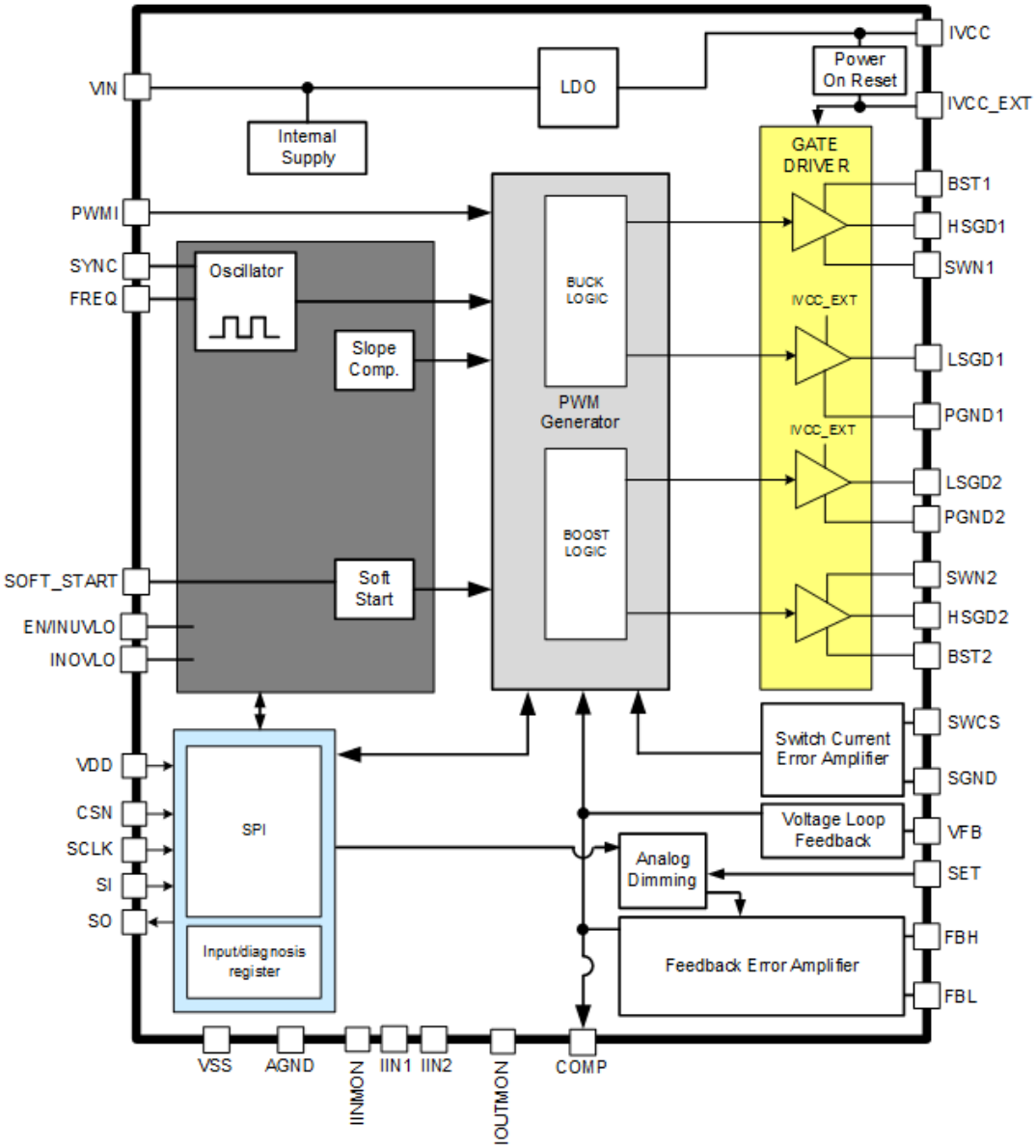

Figura 2.1.: Diagramma a blocchi semplificato del dispositivo [4]

2.1.1. Power States

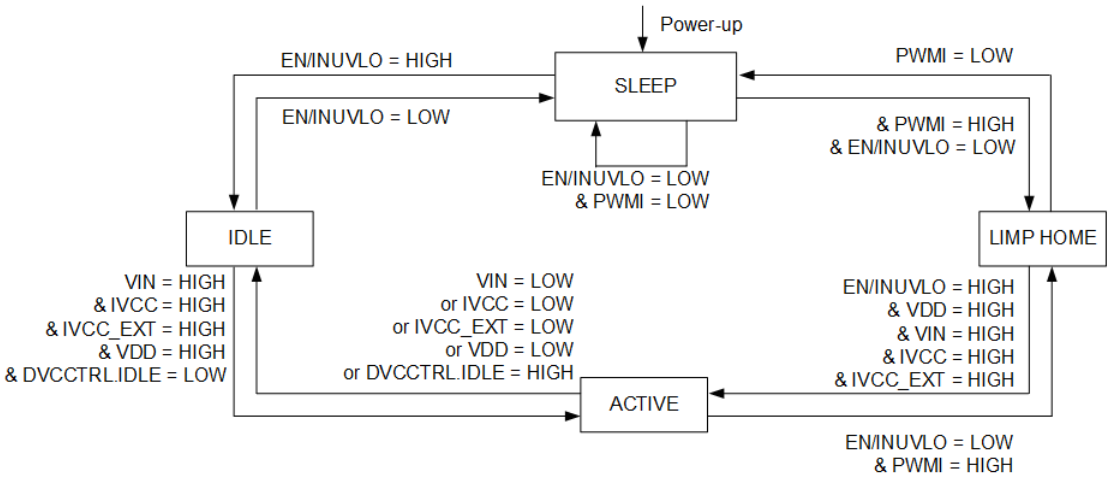

Figura 2.2.: Macchina a stati delle macro condizioni di funzionamento[4]

La macchina a stati presente in figura 2.2 riassume i quattro macro-stati di funzionamento del circuito, dove si possono vedere anche le possibili transizioni in funzione dei segnali interessati.

Quando il dispositivo è in stato di **SLEEP**, tutte le uscite sono spente ed i registri del SPI sono in stato di reset, indipendentemente dalle tensioni VIN, VDD IVCC e IVCC_EXT. Per la condizione di 'Power-up' è sufficiente che una delle tensioni di alimentazione VIN e VDD superino la propria soglia minima.

Nello stato di **IDLE** i regolatori delle alimentazioni interne stanno lavorando, ma non c'è attività di switching a prescindere dai valori delle tensioni di VIN, VDD IVCC e IVCC_EXT. In questo stato le funzioni di diagnostica non sono disponibili e se in presenza di VDD con un valore sufficiente la comunicazione SPI è attiva.

Lo stato di **LIMP HOME** permette di soddisfare determinati requisiti di sicurezza del sistema e fornisce la possibilità di mantenere un livello di corrente/tensione in uscita definita tramite un circuito di controllo di riserva. Il sistema di riserva permette di pilotare il carico durante un malfunzionamento dell'eventuale microcontrollore collegato al dispositivo.

Nello stato **ACTIVE** il sistema è pronto a far partire l'attività di switching, non appena il segnale PWMI va alto.

2.1.2. Funzioni di regolazione

Il controllore integra tutte le funzioni necessarie per regolare una corrente costante al carico – il caso ideale per applicazioni LED –, opzionalmente è possibile utilizzarlo per regolare una tensione costante.

Il sistema dispone di due ingressi analogici (IIN1, IIN2) per il controllo sulla corrente di ingresso per limitarne il valore ed un secondo loop con rilevamento di corrente, connesso ai pin FBL e FBH, per regolare la corrente erogata al carico.

La logica del dispositivo regola la corrente erogata fornendo un segnale PWM (Pulse Width Modulated) ai 'gate driver', figura 2.1 a pagina 29. I pin (HSGD1,2 e LSGD1,2) connessi ai gate driver vengono utilizzato per pilotare i MOSFET esterni. È anche possibile fornire il segnale PWM esternamente tramite il pin PWMI.

2.1.3. Soft Start

Il comportamento della Soft Start limita le correnti che attraversano l'induttore e i MOSFET esterni durante l'inizializzazione (alla prima accensione e dopo aver rivelato le condizioni di cortocircuito o di circuito aperto del carico). La funzione di Soft Start incrementa gradualmente la corrente nell'induttore e nei MOSFET durante $t_{SOFT\text{–}START}$ per evitare di fornire una tensione troppo elevata al carico. La rampa di Soft Start è definita da un capacitore ($C_{SOFT\text{–}START}$) connesso al pin SOFT_START. La scelta del valore di capacità per $C_{SOFT\text{–}START}$ dipende dalla seguente equazione:

$$t_{SOFT\text{–}START} = \frac{1.4\,[V]}{I_{SOFT\text{–}START\text{–PU}}} C_{SOFT\text{–}START}$$

Il pin SOFT_START è anche utilizzato per definire un filtro temporale per le condizioni di guasto. Quando viene rilevato sull'output un circuito aperto o un cortocircuito, viene attivato un generatore di corrente di pull-down $I_{SOFT\text{–}START\text{–PD}}$. Questa corrente abbassa il valore di $V_{SOFT\text{–}START}$ fino a che non viene raggiunta una soglia di reset che riattiva la corrente $I_{SOFT\text{–}START\text{–PU}}$ la quale innesca un nuovo ciclo fino all'uscita della condizione di guasto.

2.2. Topologia Buck-Boost sincrono non invertente

Il dispositivo è progettato per controllare i segnali che pilotano i terminali di gate di 4 MOSFET esterni disposti in una topologia Buck-Boost sincrono non invertente a singolo induttore. Questa topologia consente di operare in modalità BOOST, BUCK-BOOST o BUCK in applicazioni ad alta potenza con la massima efficienza.

I passaggi tra le differenti modalità di regolazione sono eseguiti automaticamente dal dispositivo, in funzione delle condizioni al contorno dell'applicazione.

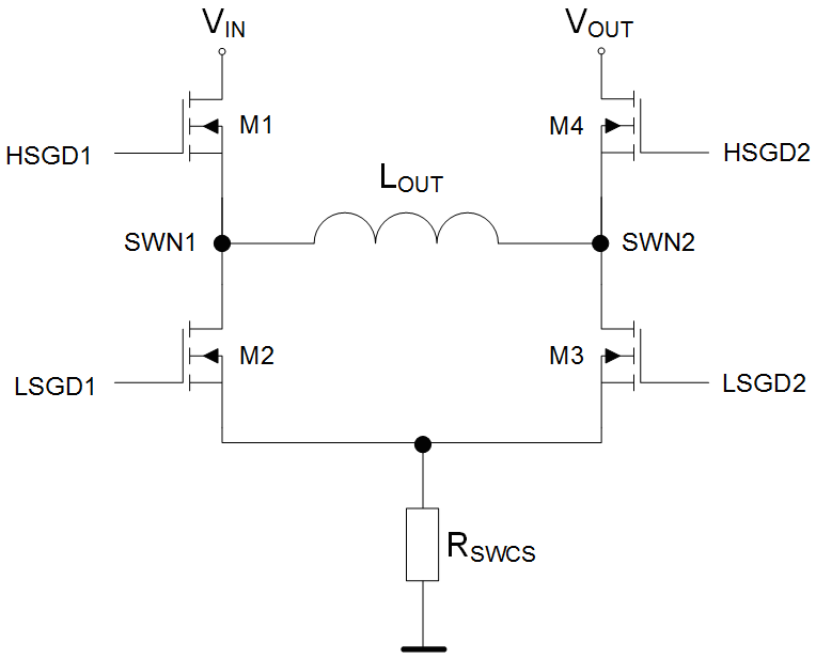

Figura 2.3.: Architettura a 4 interruttori Buck-Boost sincrono non invertente [4]

La figura 2.3 presenta lo schema della configurazione con l'induttore L_{OUT} e i 4 MOSFET a canale n (M1, M2, M3 e M4).

I transistor M1 e M3 quando sono accesi formano un percorso tra V_{IN} e massa che attraversa L_{OUT} in una direzione (pilotati dai due PIN HSGD1 e LSGD2). Mentre l'accensione di M2 e M4 forma un percorso tra V_{OUT} e massa che attraversa L_{OUT} in direzione opposta (pilotati dai due PIN HSGD2 e LSGD1) .

Il dispositivo monitora i nodi SWN1, SWN2, la tensione su R_{SWCS} e le correnti d'ingresso e di uscita.

2.2.1. Modalità di funzionamento BOOST

$V_{OUT} > V_{IN}$

Per lavorare in questa modalità i transistor che compongo il ponte 1, M1 e M2 vengono rispettivamente tenuti sempre acceso e spento, e l'attività di switching avviene sul ponte 2. In ogni ciclo si avvia prima M3 – in questa fase il dispositivo di controllo valuta la corrente che attraversa l'induttore (controllo dei picchi di corrente) – e rimane acceso finchè la tensione su R_{SWCS} non raggiunge la soglia massima di riferimento. Allo spegnimento di M3 si accende M4 per il tempo del ciclo residuo.

Nella figura 2.4 è rappresentato graficamente quanto appena esposto sinteticamente ed in figura 2.5 nella pagina successiva si mostra un confronto semplificato con l'approccio asincrono del Boost tradizionale.

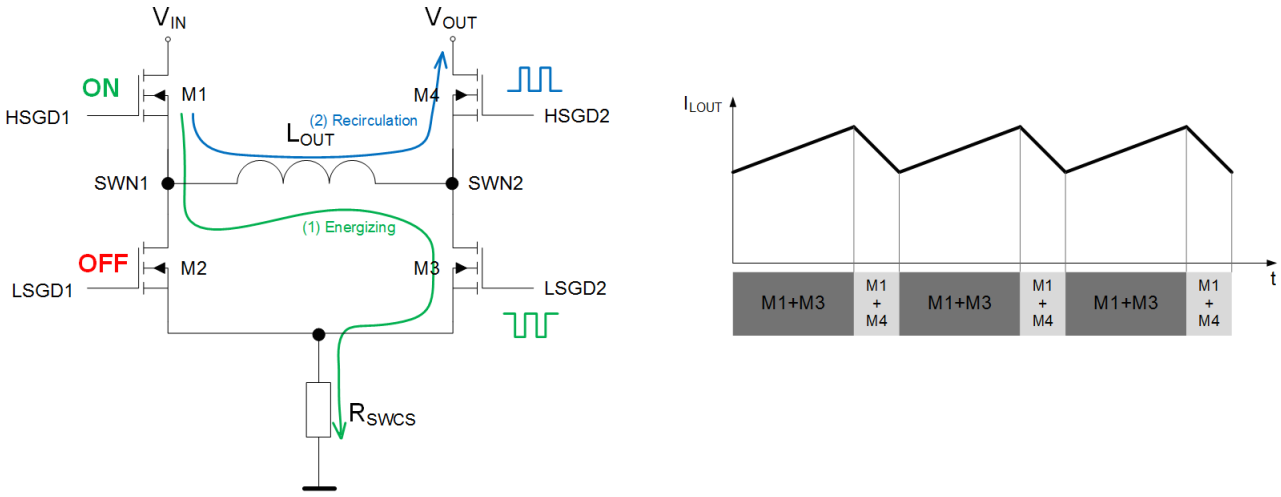

Figura 2.4.: Modalità di regolazione BOOST[4]

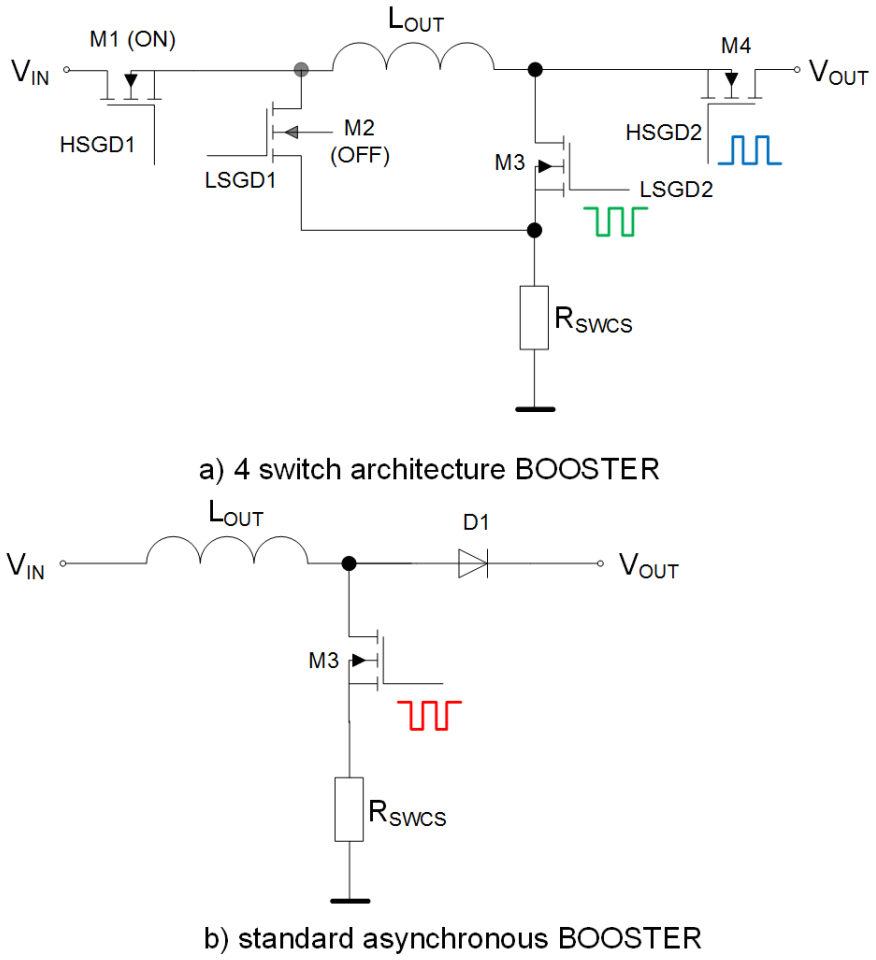

Figura 2.5.: Confronto con il circuito in configurazione BOOST asincrono standard [4]

2.2.2. Modalità di funzionamento BUCK

$V_{IN} > V_{OUT}$

Per lavorare in questa modalità i transistor che compongo il ponte 2, M4 e M3 vengono rispettivamente tenuti sempre acceso e spento, e l'attività di switching avviene sul ponte 1. In ogni ciclo si avvia prima M2 – in questa fase il dispositivo di controllo valuta la corrente che attraversa l'induttore – e rimane acceso finchè la tensione su R_{SWCS} non raggiunge la soglia minima di riferimento. Allo spegnimento di M2 si accende M1 per il tempo del ciclo residuo.

Nella figura 2.6 è rappresentato graficamente quanto appena esposto sinteticamente ed in figura 2.7 nella pagina successiva si mostra un confronto semplificato con l'approccio asincrono del Buck tradizionale

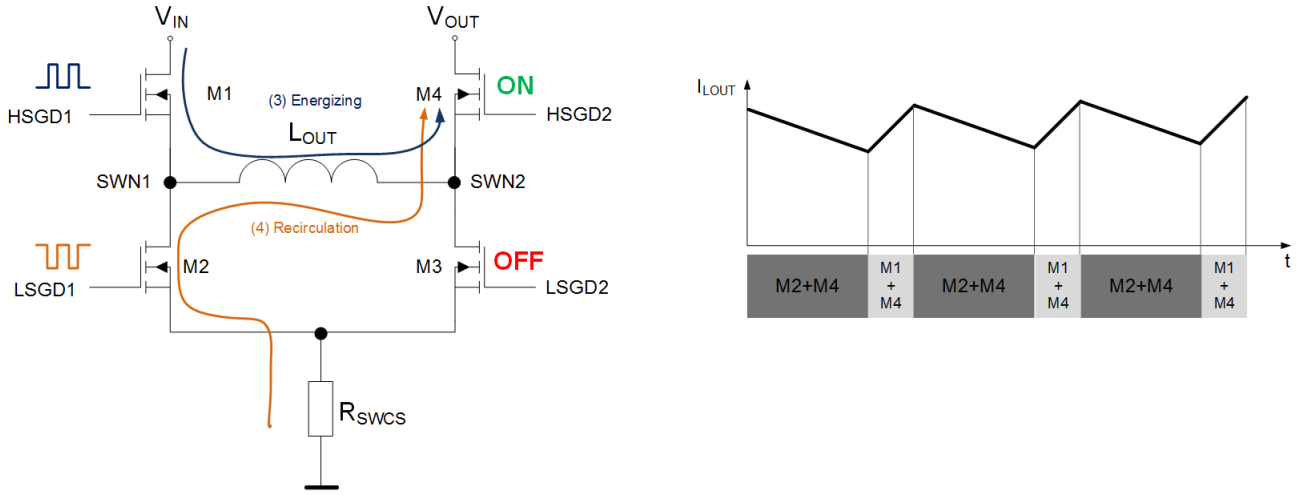

Figura 2.6.: Modalità di regolazione BUCK [4]

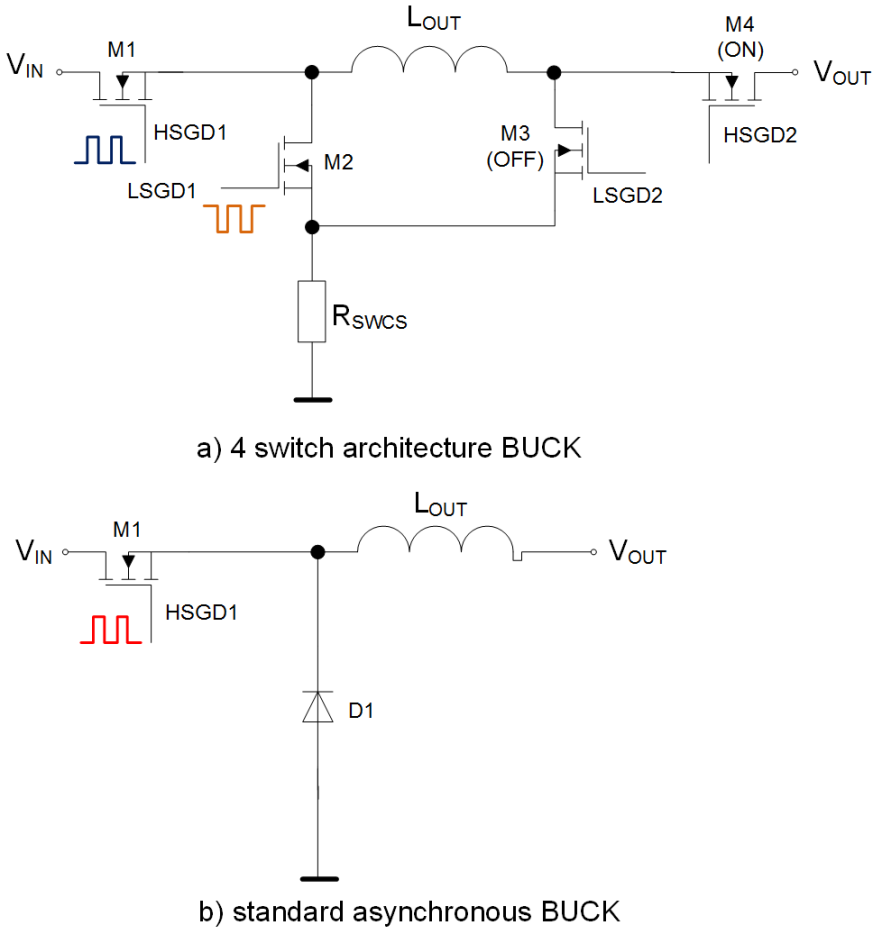

Figura 2.7.: Confronto con il circuito in configurazione BUCK asincrono standard [4]

2.2.3. Modalità di funzionamento BUCK-BOOST

Il controllore è in modalità Buck-Boost quando la V_{IN} è vicina alla V_{OUT} .

In questa modalità tutti e quattro i MOSFET vengono accessi alternativamente combinando le due modalità descritte precedentemente e come mostrato in figura 2.8

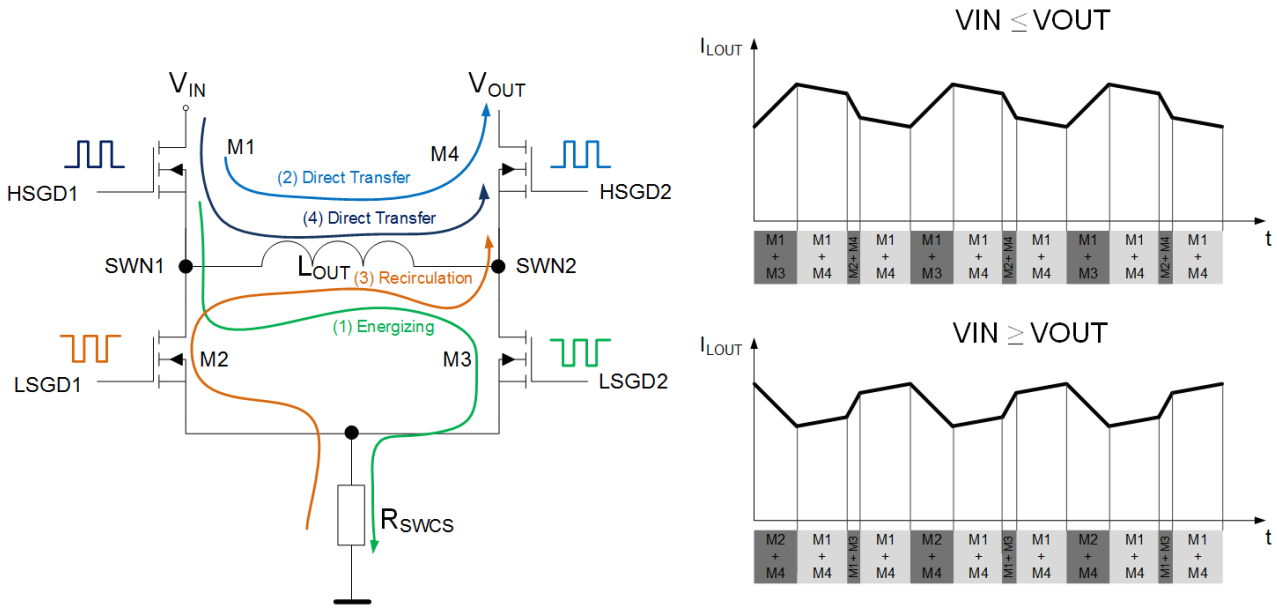

Figura 2.8.: Modalità di regolazione BUCK-BOOST [4]

	BOOST MODE	**BUCK-BOOST** MODE	**BUCK** MODE
M1	ON	PWM	PWM
M2	OFF	PWM	PWM
M3	PWM	PWM	OFF
M4	PWM	PWM	ON

Figura 2.9.: Tabella di riassunto dello stato dei transistor nelle configurazioni di lavoro [4]

3. VHDL – ANALOG PACKAGE

Il punto di partenza per l'attività di modelizzazione è definire adeguatamente i tipi di dato da utilizzare.

Tipicamente le porte dei blocchi sono di tre tipologie differenti: pin digitali (segnali di abilitazione, clock, etc), pin analogici che rappresentano tensioni (alimentazione, tensione di riferimento) e pin analogici che rappresentano correnti (correnti di bias, sensed currents).

Ad ognuna di queste tipologie si assegna un tipo di dato diverso, andando a scegliere tra STD_ULOGIC, ANALOG_UT e CURRENT_UT.

Il primo è definito nel package 'std_logic_1164' della 'libreria' IEEE mentre gli altri due si sono definiti nel package 'analog_pack' della libreria 'analog_hw'. Pertanto è importante dichiarare nell'intestazione del modello VHDL le due librerie e l'uso del package come in figura 3.1:

```
LIBRARY IEEE;
USE ieee.std_logic_1164.ALL
LIBRARY analog_hw;
USE analog_hw.analog_pack.ALL
```

Listing 3.1: Intestazione dei file VHDL

```
type std_ulogic is ( 'U',  -- bit non inizializzato/indefinito
                     'X',  -- valore sconosciuto (forte)
                     '0',  -- valore logico basso (forte)
                     '1',  -- valore logico alto (forte)
                     'Z',  -- stato di alta impedenza
                     'W',  -- valore sconosciuto (debole)
                     'L',  -- valore logico basso (debole)
                     'H',  -- valore logico alto (debole)
                     '-'); -- don't care
```

Listing 3.2: Dichiarazione del tipo di dato std_ulogic

```
type analog_ut is range real'low to real'high;
subtype current_ut is analog_ut;
```

Listing 3.3: Dichiarazione del tipo di dato analog_ut e current_ut

In figura 3.2 nella pagina precedente è possibile vedere come viene dichiarato il tipo di dati STD_ULOGIC, mentre si sono definiti ANALOG_UT e CURRENT_UT fondamentalmente come tipo di dato REAL – figura 3.3 – con la particolarità di aver assegnato uno specifico significato ad alcuni valori.

Questi sono:

'ANALOG_U'/'CURRENT_U' (valore di tensione/corrente indefinito)

'ANALOG_X'/'CURRENT_X' (valore di tensione/corrente sconosciuto)

'ANALOG_Z' (valore di alta impedenza)

```
constant ANALOG_U : analog_ut := analog_ut'left;
constant ANALOG_X : analog_ut :=
                    analog_ut(real(analog_ut'left) / 2.0);
constant ANALOG_Z : analog_ut :=
                    analog_ut(real(analog_ut'left) / 4.0);
```

Listing 3.4: Dichiarazione di ANALOG_U, ANALOG_X e ANALOG_Z

```
constant CURRENT_U : current_ut := current_ut'left;
constant CURRENT_X : current_ut :=
                     current_ut(real(current_ut'left) / 2.0);
```

Listing 3.5: Dichiarazione di CURRENT_U e CURRENT_X

Come mostrato in figura 3.4 ANALOG_U è definito come il minimo valore che può assumere il tipo di dati REAL, si noti che nel VHDL il valore iniziale – se non specificato nella dichiarazione – dell'oggetto, variabile o signal, di tipo T è T'left, quindi ANALOG_U/CURRENT_U è il valore iniziale al momento della dichiarazione del segnale.

ANALOG_X e ANALOG_Z sono definiti rispettivamente la metà e un quarto di ANALOG_U.

Nella tabella 3.2 sono mostrati i valori di ANALOG_U, ANALOG_X, ANALOG_Z che si sono ottenuti nelle nostre applicazioni, ma per come sono stati definiti essi possono cambiare al variare del simulatore.

Per questo motivo sono state implementate due funzioni di nome 'analog_known' e 'current_known', le quali restituiscono il valore booleano TRUE quando il segnale passato come argomento assume uno dei valori tra ANALOG_U, ANALOG_X, ANALOG_Z o CURRENT_U e CURRENT_X.

ANALOG_U	-1e+308
ANALOG_X	-5e+307
ANALOG_Z	-2,5e+307

Tabella 3.2.: Valore di default di ANALOG_U, ANALOG_X e ANALOG_Z

Tra gli altri tipi dati definiti all'interno del package sono da citare le versioni vettoriali:

```
type analog_vector_ut  is array (natural range <>) of analog_ut;
type current_vector_ut is array (natural range <>) of current_ut;
```

Listing 3.6: Dichiarazione di anaog_vector_ut e current_ut_vector

e le versioni 'resolved' ANALOG_T e CURRENT_T.

Quando viene usato un tipo di dati 'resolved' viene richiamata un funzione di risoluzione per decidere quale tra i segnali driver ha la priorità nell'assegnazione al segnale ricevente.

La figura 3.7 illustra la funzione di risoluzione per il tipo di dato std_logic resolved dello std_ulogic. Per il tipo di dati ANALOG_T, viene implementata una tabella simile.

	Value_2	ANALOG_Z	ANALOG_X	ANALOG_U
Value_1	'ANALOG_X'	'Value_1'	'ANALOG_X'	'ANALOG_X'
ANALOG_Z	'Value_2'	'ANALOG_Z'	'ANALOG_X'	'ANALOG_X'
ANALOG_X	'ANALOG_X'	'ANALOG_X'	'ANALOG_X'	'ANALOG_X'
ANALOG_U	'ANALOG_X'	'ANALOG_X'	'ANALOG_X'	'ANALOG_X'

Tabella 3.4.: Tabella di risoluzione di ANALOG_T

```
type stdlogic_table is array(std_ulogic, std_ulogic) of std_ulogic;
constant resolution_table : stdlogic_table := (
-- ---------------------------------------------------------------------
--| U    X    0    1    Z    W    L    H    -         |    |
-- ---------------------------------------------------------------------
  ( 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U' ), -- | U |
  ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ), -- | X |
  ( 'U', 'X', '0', 'X', '0', '0', '0', '0', 'X' ), -- | 0 |
  ( 'U', 'X', 'X', '1', '1', '1', '1', '1', 'X' ), -- | 1 |
  ( 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', 'X' ), -- | Z |
  ( 'U', 'X', '0', '1', 'W', 'W', 'W', 'W', 'X' ), -- | W |
  ( 'U', 'X', '0', '1', 'L', 'W', 'L', 'W', 'X' ), -- | L |
  ( 'U', 'X', '0', '1', 'H', 'W', 'W', 'H', 'X' ), -- | H |
  ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' )  -- | - |
);
```

Listing 3.7: Funzione di resoluzione del std_logic (look-up table)

Essa può essere interpretata molto semplicemente considerando ANALOG_Z come la condizione più debole e se un altro driver sta imponendo una tensione quest'ultimo avrà priorità maggiore. Altrimenti verrà assegnato il valore ANALOG_X in quanto non è possibile avere un nodo con più livelli di tensione.

La tabella di risoluzione del tipo di dati CURRENT_T è molto simile a quell'ANALOG_T, dove la principale differenza sta nell'implementazione della legge di Kirchhoff delle correnti, quindi un segnale connesso a più driver risulterà la somma algebrica dei valori imposti dai driver.

	Value_2	CURRENT_X	CURRENT_U
Value_1	'Value_1 + Value_2'	'CURRENT_X'	'CURRENT_X'
CURRENT_X	'CURRENT_X'	'CURRENT_X'	'CURRENT_X'
CURRENT_U	'CURRENT_X'	'CURRENT_X'	'CURRENT_X'

Tabella 3.6.: Tabella di risoluzione di CURRENT_T

Ad eccezione di alcuni casi in cui avere più driver è imposto dal circuito da modellizzare, si consiglia l'utilizzo di tipi di dati unresolved. Questo permette di evitare eventuali cortocircuiti indesiderati tra 2 segnali, poichè in fase di compilazione vengono segnalate le net pilotate da più di un driver.

4. Netlister VHDL

4.1. Cadence VHDL Toolbox

Come esposto precedentemente si intende creare il modello Real Number di un prodotto in fase di progetto, i cui schematici analogici sono stati in precedenza predisposti in librerie di Virtuoso® Cadence.

Questo permette l'utilizzo del Netlister disponibile nella VHDL Toolbox di Virtuoso®.

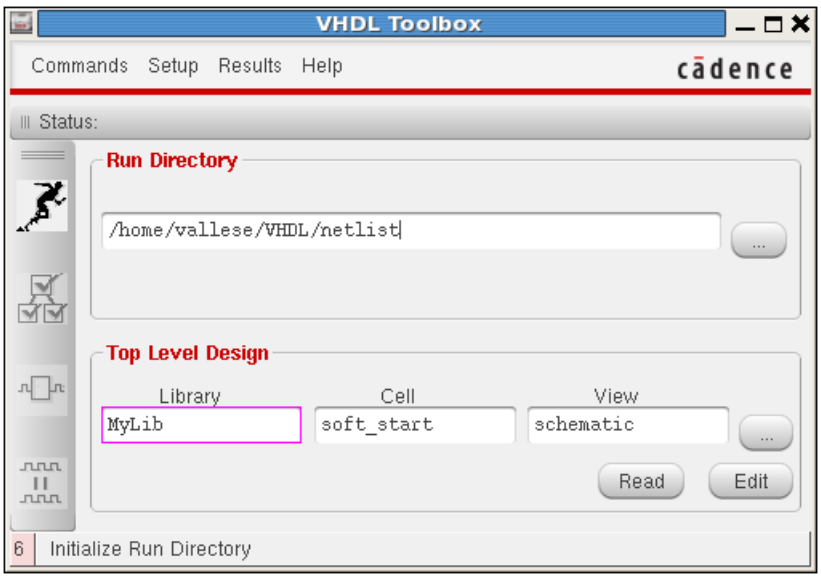

Figura 4.1.: Schermata iniziale del Cadence VHDL Toolbox

Nella schermata iniziale – figura 4.1 – nel campo 'Run Directory' è neccessario introdurre il percorso di una cartella con permesso di scrittura, mentre nel campo 'Top Level Design' si inserisce la view dell'elemento di gerarchia superiore di cui si intende creare il modello.

Dopo aver impostato alcune opzioni, di cui alcune si parlerà in seguito, il Netlister andrà a generare due file contenenti entity ed architecture per ogni blocco di ciascun livello gerarchico inferiore, ed un unico file di configurazione per il blocco superiore.

Il Netlister utilizza il metodo in tre fasi per istanziare i componenti del VHDL'87:

- dichiarazione:
  - serve a specificare le caratteristiche dell'interfaccia del componente
  - la sintassi compare nella parte dichiarativa (prima di BEGIN) delle entity dell'architecture del blocco che istanzia il componente
- specificazione (binding)
  - per ogni componente istanziato, definisce la entity e l'architecture da utilizzare
  - anch'esso nella parte dichiarativa o, come in questo caso, in un file di configurazione
- istanza
  - è l'istruzione concorrente con cui si inserisce il componente figlio, all'interno del modello VHDL del genitore
  - elenca implicitamente o esplicitamente le connessioni dei terminali dell'istanza ai segnali interni del genitore
  - nella parte assertiva dell'architecture (tra BEGIN ed END)

4.2. Vantaggi e setup del Netlister

Il primo vantaggio che il netlister offre è la capacità di recuperare automaticamente i blocchi delle porte logiche ed altri elementi puramente digitali dalle librerie presenti nel progetto. In questo passaggio si può intuire l'uso del file di configurazione infatti, nell'eventualità che le stesse celle venissero compilate in librerie con nome diverso, è sufficiente modificare le istruzione di binding presenti in un unico file. Questo fatto favorisce la portabilità e la riusabilità del lavoro.

Inoltre nel Setup – figura 4.3 – è possibile inserire il percorso delle cartelle utilizzate per salvare i modelli già realizzati, ed allo stesso modo delle porte logiche il programma li andrà automaticamente ad istanziare all'interno del blocco gerarchico superiore.

```
ARCHITECTURE <architecture_father_name> OF <entity_father_name> IS
-- [...]
  component <entity_son_name>
    [generic(<generic-list>); ]
    port (A, B, CI: in std_logic;
    port(<port-list>);
  end component;
-- [...]
BEGIN
  <label>: <component_name>
     PORT MAP (<port-association-list>);
END

CONFIGURATION <configuration_name> OF <entity_father_name> IS
  FOR <architecture_son_name>
    FOR <label>: <component_name>
      USE ENTITY <library_name>.<entity_son_name>(architecture_son_name);
      FOR <architecture_son_name>
        --  [...]
      END FOR;
    END FOR;
  END FOR;
END <configuration_name>;
```

Listing 4.1: Sintassi per l'istanza di un componente e di una configurazione in VHDL'87

In definitiva il Netlister permette di ridurre in due modi le cause di errore. Primo, creando la possibilità di atomizzare il lavoro di modellizzazione concentrandosi solo su piccoli/medi blocchi analogici che compongono i livelli gerarchici inferiori e semplificando quindi le funzioni da realizzare coi modelli. In secondo luogo, creando automaticamente le connessioni nelle gerarchie superiori.

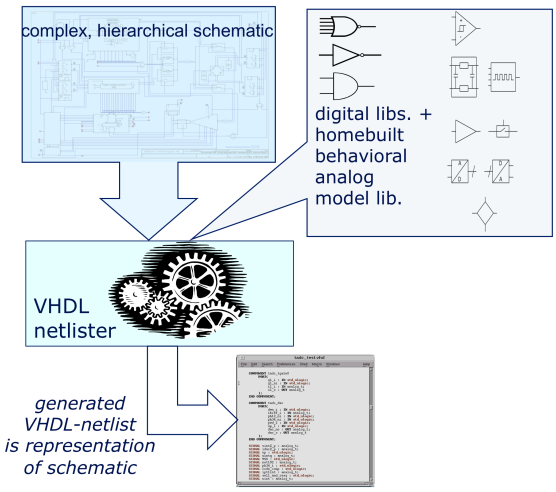

Figura 4.2.: Schema di funzionamento del Netlister [7]

Un'altra funzionalità del programma, è la possibilità di personalizzare l'intestazione dei file VHDL, permettendo così di linkare i package delle librerie IEEE e analog_hw come visto in figura 3.1 a pagina 39.

Questo consente di definire le porte dei modelli con i tipi di dato che si desiderano, ed esse verranno riconosciute dal blocco che li istanzia. Inoltre il netlister propaga il tipo di dato utilizzato per una porta in tutte le net e porte dei blocchi gerarchici superiori, ad eccezione del caso in cui un segnale risulti connesso a due porte di tipo diverso. In questa occasione lancia un warning sulla console, indicando il pin e le net responsabili e richiama una funzione di conversione predefinita, che nell'eventualità, si può modificare. Data la difficoltà di generalizzare quale sia la corretta procedura di conversione, si consiglia di provvedere manualmente alla risoluzione di tali problemi.

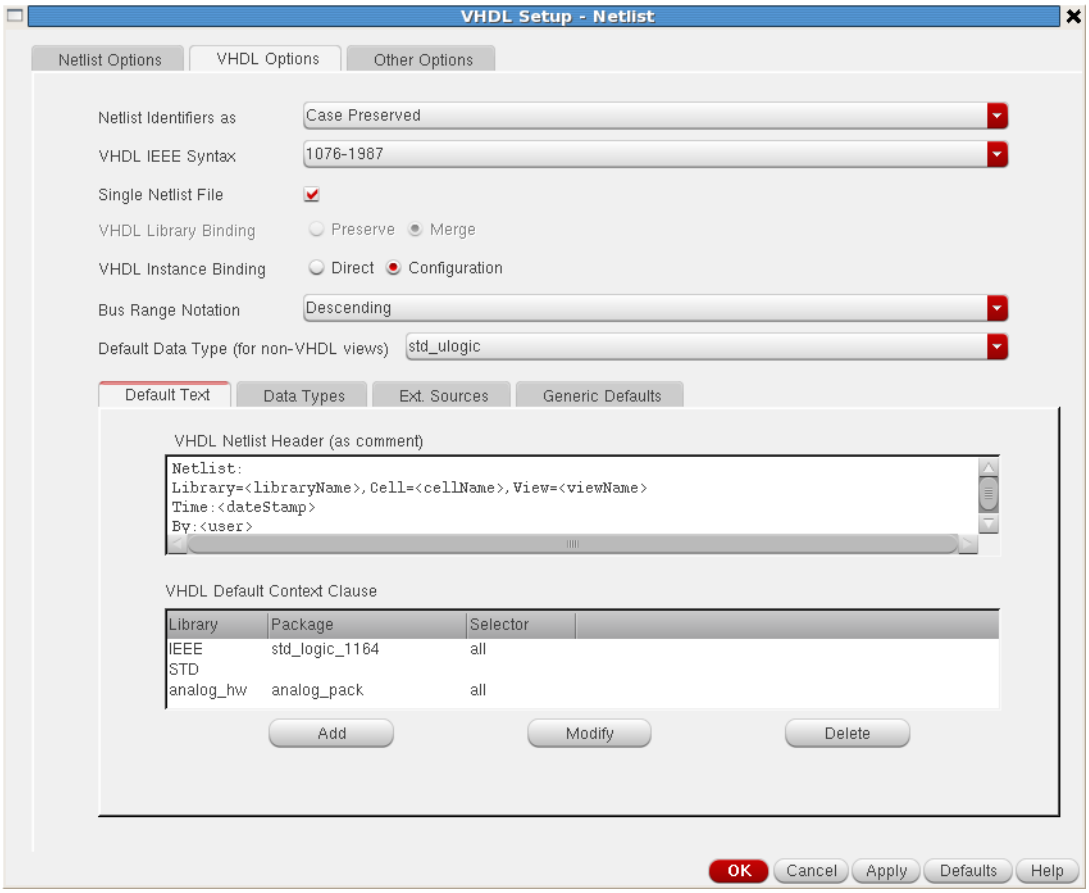

Figura 4.3.: Setup del netlister 1 – Intestazione del file VHDL

Figura 4.4.: Setup del netlister 2 – Percorso dei modelli

Come ultima nota sull'utilizzo del netlister si vuole indicare la possibilità di indicare al programma gli "Skip Design Unit". Questo elenco permette di far sostituire automaticamente gli elementi indicati – anche intere librerie – con dei circuiti aperti come potrebbe avvenire per alcuni capacitori. È anche possibile indicare in un modo diverso gli elementi che si vogliono sostituire con un cortocircuito, come può succedere per i resistori.

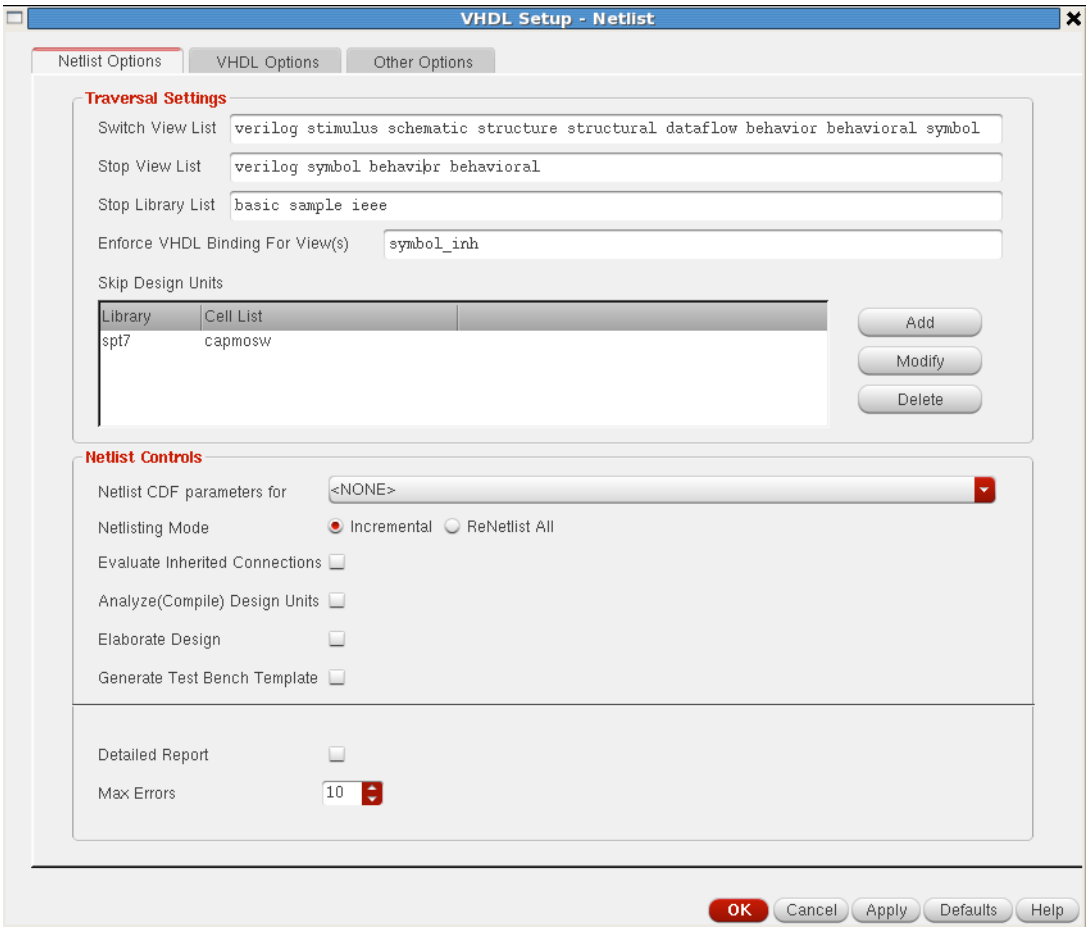

Figura 4.5.: Setup del netlister 3 – Skip Design Unit

5. Esempi di modelli VHDL

5.1. Regolatore ideale di tensione

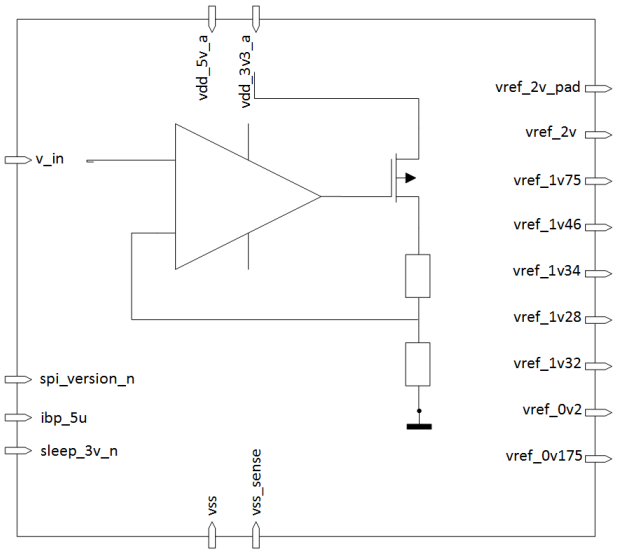

Figura 5.1.: Schema del Regolatore di tensione

Lo schema in figura 5.1 si riferisce ad un regolatore di tensione presente nella Power Management Unit (PMU) del dispositivo.

Lo scopo della modellizzazione di questo elemento è quello di velocizzare la simulazione e quindi ridurre al minimo il numero di calcoli che il simulatore deve eseguire. Per questo il modello assegna semplicemente alle uscite del blocco il valore di tensione che idealmente dovrebbero avere, non appena si verificano le condizioni di abilitazione.

```
1  LIBRARY IEEE,STD;
2  USE IEEE.std_logic_1164.all;
3  LIBRARY analog_hw;
4  USE analog_hw.analog_pack.all;
5
6  ENTITY pmu_vref_2v IS
7  PORT(
8         vdd_5v_a       : IN  analog_ut;
9         vdd_3v3_a      : IN  analog_ut;
10        vss            : IN  analog_ut;
11        vss_sense      : IN  analog_ut;
12        vin            : IN  analog_ut;
13        ibp_5u         : IN  current_ut;
14        sleep_3v_n     : IN  std_ulogic;
15        spi_version_n  : IN  std_ulogic;
16        vref_2v        : OUT analog_ut;
17        vref_2v_pad    : OUT analog_ut;
18        vref_1v75      : OUT analog_ut;
19        vref_1v46      : OUT analog_ut;
20        vref_1v34      : OUT analog_ut;
21        vref_1v28      : OUT analog_ut;
22        vref_0v32      : OUT analog_ut;
23        vref_0v2       : OUT analog_ut;
24        vref_0v175     : OUT analog_ut
25       );
26 END pmu_vref_2v;
27
28 ARCHITECTURE schematic of pmu_vref_2v IS
29 -- Local SIGNALS and CONSTANTS ---
30   constant vin_th             : analog_ut  := 1.0;
31   constant vref_2v_value      : analog_ut  := 2.0;
32   constant vref_2v_pad_value  : analog_ut  := 2.0;
33   constant vref_1v75_value    : analog_ut  := 1.75;
34   constant vref_1v46_value    : analog_ut  := 1.46;
35   constant vref_1v34_value    : analog_ut  := 1.34;
36   constant vref_1v28_value    : analog_ut  := 1.28;
37   constant vref_0v32_value    : analog_ut  := 0.32;
38   constant vref_0v2_value     : analog_ut  := 0.2;
39   constant vref_0v175_value   : analog_ut  := 0.175;
40   signal   en_internal        : std_ulogic := '0';
41 BEGIN
```

```
42 -- Concurrent Signal Assignments ---
43   en_internal <= '1' when (      analog_known(vdd_5v_a)
44                             and analog_known(vdd_3v3_a)
45                             and analog_known(vin)
46                                                  and (vin > vin_th)
47                             and current_known(ibp_5u)
48                             and (sleep_3v_n = '1') )
49                       else '0';
50   vref_2v     <= vref_2v_value     when en_internal = '1' else analog_ut(0.0);
51   vref_2v_pad <= vref_2v_pad_value when en_internal = '1' else analog_ut(0.0);
52   vref_1v75   <= vref_1v75_value   when en_internal = '1' else analog_ut(0.0);
53   vref_1v46   <= vref_1v46_value   when en_internal = '1' else analog_ut(0.0);
54   vref_1v34   <= vref_1v34_value   when en_internal = '1' else analog_ut(0.0);
55   vref_1v28   <= vref_1v28_value   when en_internal = '1' else analog_ut(0.0);
56   vref_0v32   <= vref_0v32_value   when en_internal = '1' else analog_ut(0.0);
57   vref_0v2    <= vref_0v2_value    when en_internal = '1' else analog_ut(0.0);
58   vref_0v175  <= vref_0v175_value  when en_internal = '1' else analog_ut(0.0);
59
60 END schematic;
```

5.2. Comparatore

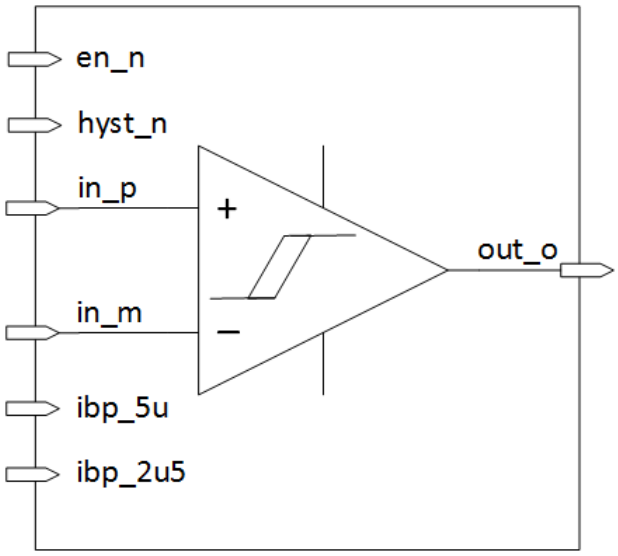

Figura 5.2.: Schema del comparatore

Quest'esempio è un comparatore di due livelli di tensione ed uscita digitale. In esso è possibile osservare che si è riportato un'isteresi simmetrica sulla soglia di transizione ed il tempo di propagazione dagli ingressi all'uscita.

Come si può vedere il modello ignora gli effetti delle impedenze connesse ai suoi PIN.

Inoltre può essere interessante notare che l'istruzione VHDL *'after'* fornisce un filtro temporale sulle transizioni degli ingressi, questo evita che il comparatore riporti in uscita variazioni con durata inferiore al tempo di propagazione.

```
1  LIBRARY IEEE,STD;
2  USE IEEE.std_logic_1164.all;
3  LIBRARY analog_hw;
4  USE analog_hw.analog_pack.all;
5
6  ENTITY comparator_pmos_medium IS
7    PORT(
8          in_m     : IN  analog_ut;
9          in_p     : IN  analog_ut;
10         ibp_5u   : IN  current_ut;
11         ibp_2u5  : IN  current_ut;
12         en_n     : IN  std_ulogic;
13         hyst_n   : IN  std_ulogic;
14         out_o    : OUT std_ulogic
15        );
16 END comparator_pmos_medium;
17
18 ARCHITECTURE beh of comparator_pmos_medium IS
19
20 -- Local SIGNALS and CONSTANTS ---
21 constant ibp_5u_treshold  : current_ut := 4.5e-6;
22 constant ibp_2u5_treshold : current_ut := 2.0e-6;
23 signal   enable_cond      : boolean;
24 signal   v_a              : analog_ut;
25 signal   d_int            : std_logic  :='0';
26 signal   out_o_temp       : std_logic;
27 -- hysteresis, threshold margin
28 constant v_hys            : analog_ut  := 0.04;
29 signal   threshold        : analog_ut  := v_hys;
30 -- propagation delay
31 constant t_delay          : time       := 100 ns;
32
33 BEGIN
34 -- Concurrent Signal Assignments ---
35   enable_cond <= TRUE when ( analog_known(ibp_5u)           and
36                              ( ibp_5u > ibp_5u_treshold )   and
37                              analog_known(ibp_2u5)          and
38                              ( ibp_2u5 > ibp_2u5_treshold ) and
39                              ( en_n = '0' ) )
40                         else FALSE;
41   -- Differential signal input
```

```
42    v_a          <= (in_p - in_m) when enable_cond
43                                  else analog_ut(0.0);
44    -- Treshold valutation
45    threshold    <= -v_hys when ( out_o_temp = '1' )
46                           else v_hys;
47  -- Processes ----
48  comp:
49    process( v_a, enable_cond )
50    begin
51      if enable_cond then
52        if (v_a > threshold) then        -- input above reference
53          d_int <= '1';
54        else                             -- input below reference
55          d_int <= '0';
56        end if;
57      else
58        d_int <= '0' ;
59      end if; -- enable_cond
60    end process; -- comp
61
62    out_o_temp <= d_int after t_delay;
63
64    out_o       <= out_o_temp;
65
66  END beh;
```

5.3. Blocco di Soft Start

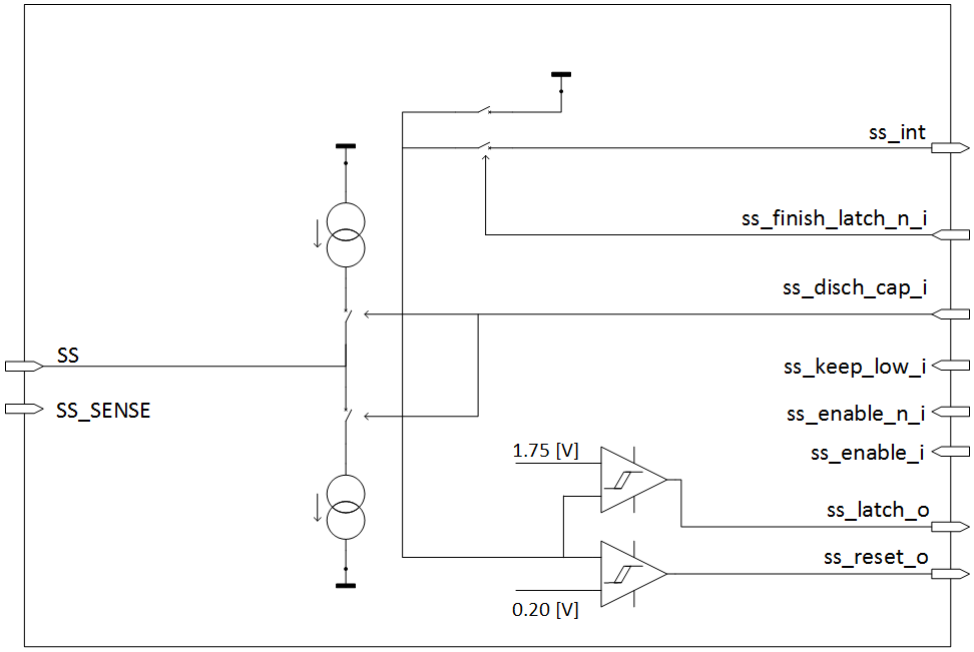

Figura 5.3.: Schema del blocco di Soft Start

In figura 5.3 viene riportato uno schema semplificato del blocco di Soft Start, il cui funzionamento è sinteticamente descritto nel paragrafo 2.1.3 a pagina 31.

Nel modello è stata riportata la generazione della rampa di tensione in funzione delle correnti che caricano o scaricano il capacitore, il cui valore di capacità è stato posto come un segnale all'interno dell'architecture.

Il blocco inoltre presenta l'istanza di due comparatori per segnalare alla logica che gestisce i segnali di controllo, l'attraversamento della tensione sul capacitore delle due soglie di riferimento.

```
1  LIBRARY IEEE,STD;
2  USE IEEE.std_logic_1164.all;
3  LIBRARY analog_hw;
4  USE analog_hw.analog_pack.all;
5  ENTITY soft_start_block IS
6  PORT(
7        vdd_3v3_a          : IN  analog_ut;
8        vss_a              : IN  analog_ut;
9        ibp_5u_rpp_ss      : IN  current_ut;
10       vref               : IN  analog_ut;
11       ss_enable          : IN  std_ulogic;
12       ss_finish_latch_n  : IN  std_ulogic;
13       ss_disch_cap       : IN  std_ulogic;
14       vref_1v75          : IN  analog_ut;
15       ss_pad             : IN  std_ulogic;
16       ss_pad_sense       : IN  analog_ut;
17       ss_enable_n        : IN  std_ulogic;
18       ss_keep_low        : IN  std_ulogic;
19       vref_0v2           : IN  analog_ut;
20       pok                : IN  std_ulogic;
21       ss_latch           : OUT std_ulogic;
22       ss_reset           : OUT std_ulogic;
23       ss_int             : OUT analog_ut
24       );
25 END soft_start_block;
26
27 ARCHITECTURE beh of soft_start_block IS
28         COMPONENT comparator_pmos_medium
29         PORT(
30                 out_o    : OUT std_ulogic;
31                 in_m     : IN  analog_ut;
32                 hyst_n   : IN  std_ulogic;
33                 ibp_2u5  : IN  current_ut;
34                 en_n     : IN  std_ulogic;
35                 ibp_5u   : IN  current_ut;
36                 in_p     : IN  analog_ut
37               );
38         END COMPONENT;
39 -- Local SIGNALS and CONSTANTS ---
40   constant vdd_3v3_a_threshold      : analog_ut  := 3.0;
41   constant ibp_5u_rpp_ss_threshold : current_ut := 4.5e-6;
```

```
42    signal    net33                    : current_ut;
43    signal    net35                    : current_ut;
44    signal    net36                    : current_ut;
45    signal    net37                    : current_ut;
46    signal    hyst_n                   : std_ulogic := '0';
47    signal    enable_cond              : boolean;
48    -- Passo temporale fisso per la generazione della rampa
49    signal    ss_time_step             : time       := 5 ns;
50    constant  one_psec                 : time       := 1 ps;
51    signal    charge_ampl_step         : analog_ut;
52    signal    discharge_ampl_step      : analog_ut;
53    signal    vcap_temp                : analog_ut  := 0.0;
54    signal    v_cap                    : analog_ut  := 0.0;
55    signal    comp_reset_in_m          : analog_ut;
56    signal    comp_latch_in_p          : analog_ut;
57    -- Soft Start pull up current
58    constant  i_ss_PU                  : analog_ut  := 20.0e-6;
59    -- Soft Start pull down current
60    constant  i_ss_PD                  : analog_ut  :=  2.0e-6;
61    signal    cap_ss_value             : analog_ut  := 32.0e-9;
62    signal    cap_s_step               : analog_ut;
63  BEGIN
64  -- Concurrent Signal Assignments ---
65    enable_cond          <= TRUE when ( analog_known(vdd_3v3_a)
66                                        and ( vdd_3v3_a > vdd_3v3_a_threshold)
67                                        and current_known(ibp_5u_rpp_ss)
68                                        and ( ibp_5u_rpp_ss > ibp_5u_rpp_ss_threshold)
69                                        and analog_known(vref)
70                                        and ( vref > analog_ut(0.0) )
71                                        and ( pok = '1' ) )
72                                       else FALSE;
73    ss_int               <= v_cap when (ss_finish_latch_n = '1')
74                                  else vdd_3v3_a;
75    cap_step             <= cap_ss_value /
76                            (analog_ut(ss_time_step / one_psec)/1.0e+12);
77    charge_ampl_step     <= i_ss_PU / cap_step;
78    discharge_ampl_step  <= i_ss_PD / cap_step;
79  -- Processes ---
80  current_mirror:
81    process(vdd_3v3_a, ibp_5u_rpp_ss, ss_enable)
82    begin
```

```
83      if ( analog_known(vdd_3v3_a)
84           and ( vdd_3v3_a > vdd_3v3_a_threshold)
85           and current_known(ibp_5u_rpp_ss)
86           and ( ibp_5u_rpp_ss > ibp_5u_rpp_ss_threshold)
87           and ( ss_enable = '1' ) )   then
88        net33 <= current_ut(5.0e-6);            -- i_comp_reset
89        net35 <= current_ut(2.5e-6);
90        net36 <= current_ut(5.0e-6);            -- i_comp_latch
91        net37 <= current_ut(2.5e-6);
92     else
93        net33 <= current_ut(0.0);
94        net35 <= current_ut(0.0);
95        net36 <= current_ut(0.0);
96        net37 <= current_ut(0.0);
97       end if;
98    end process; -- current_mirror
99  cap_reg:
100   process(enable_cond, ss_keep_low, ss_disch_cap, v_cap, vcap_temp, vref, vss_a)
101   begin
102     if enable_cond then
103       if (ss_keep_low = '1') then              -- fast discharge condition
104         vcap_temp <= vss_a;
105       else
106         if (ss_disch_cap = '1') then           -- slow discharge condition
107           vcap_temp <= ( v_cap - discharge_ampl_step ) after ss_time_step;
108         elsif (ss_enable = '1')    then        -- charge condition
109           vcap_temp <= ( v_cap + charge_ampl_step )    after ss_time_step;
110         end if;
111       end if; -- ss_keep_low
112       if (vcap_temp > vref) then
113         v_cap <= vref;
114       elsif (vcap_temp < vss_a) then
115         v_cap <= vss_a;
116       else
117         v_cap <= vcap_temp;
118       end if;
119     end if;  -- enable_cond
120   end process;  -- cap_reg
121 pmos_M_ESD:
122   process(pok, v_cap, vss_a)
123   begin
```

```
124       if (pok = '1') then
125         comp_reset_in_m <= v_cap;
126         comp_latch_in_p <= v_cap;
127       else
128         comp_reset_in_m <= vss_a;
129         comp_latch_in_p <= vss_a;
130       end if;
131     end process; -- pmos_M_ESD
132   -- Instances in the design ----
133   i_comp_reset:
134     comparator_pmos_medium
135     PORT MAP(
136               out_o   => ss_reset,
137               in_m    => comp_reset_in_m,
138               hyst_n  => hyst_n,
139               ibp_2u5 => net35,
140               en_n    => ss_enable_n,
141               ibp_5u  => net33,
142               in_p    => vref_0v2
143               );
144   i_comp_latch:
145     comparator_pmos_medium
146     PORT MAP(
147               out_o   => ss_latch,
148               in_m    => vref_1v75,
149               hyst_n  => hyst_n,
150               ibp_2u5 => net37,
151               en_n    => ss_enable_n,
152               ibp_5u  => net36,
153               in_p    => comp_latch_in_p
154               );
155   END beh;
```

6. Modello del convertitore BUCK-BOOST

6.1. Sistema di partenza

Dopo che si è completato il lavoro di creazione dei modelli dei blocchi analogici interni del dispositivo si tratta ora di verificarne il funzionamento.

Come primo approccio si sarebbe potuto andare a pilotare i PIN d'ingresso del sistema con valori opportuni e poi andare a controllare manualmente che i valori in risposta dai PIN di uscita fossero quelli che si aspettavano. Questa possibilità può funzionare per i primi test base di funzionamento, ma sul lungo periodo si dimostra molto onerosa e poco naturale nella valutazione dei risultati.

Dunque risulta conveniente sviluppare un modello a tempi discreti per la tipologia BUCK-BOOST non invertente presentata nella sezione 2.2 a pagina 32, e inserire come carico una serie di diodi per emulare una catena di LED, ovvero trasportare in linguaggio VHDL lo schema in figura 6.1 il quale è il circuito utilizzato nelle simulazioni SPECTRE di riferimento:

Nel seguito della trattazione ci si riferirà al circuito appena presentato con il termine inglese PLANT, preso in prestito dalla teoria del controllo dove indica il sistema di cui si vuole ottenere una regolazione tramite il loop di feedback.

Nel nostro caso il dispositivo analizzato – di cui si è già realizzato il modello – rappresenta il sistema di controllo e la grandezza che si vuole regolare è la corrente che attraversa i diodi di carico tramite il sensing della tensione della resistenza R_F.

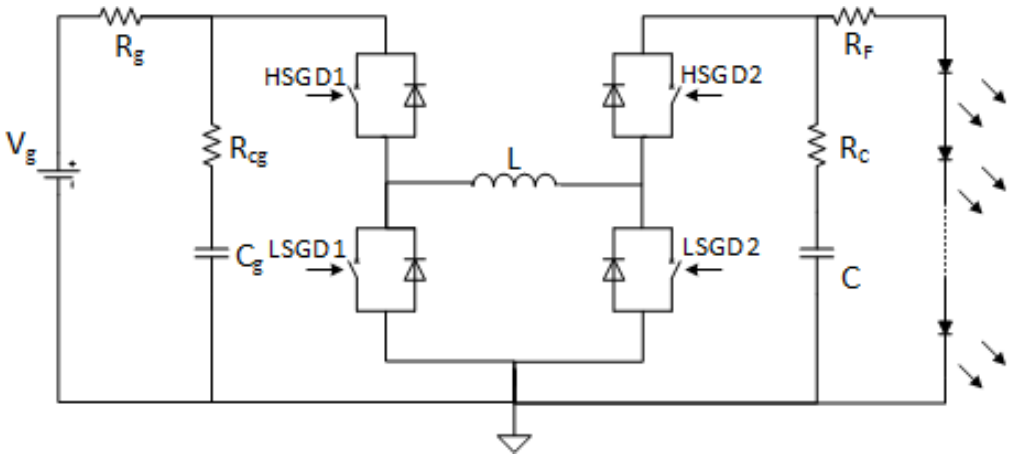

Figura 6.1.: Schema del PLANT

Per lo sviluppo del modello si sono utilizzati come base di partenza due concetti: il primo è di utilizzare come variabili di stato le tensioni dei capacitori e le correnti degli induttori, il secondo è di sostituire gli interruttori con dei resistori il cui valore di resistenza varia in funzione del gate driver.

Confrontando lo schema rappresentato in figura 6.1 con lo schema in figura 2.3 si può notare che si sono sostituiti i MOSFET con il parallelo d'interruttori ideali e diodi.

Questi diodi rappresentano i diodi parassiti di Body-Drain, essi sono essenziali alla simulazione in quanto la logica anticrossing interna al dispositivo prevede dei dead time in cui tutti gli interruttori dei ponti in fase di switching sono spenti.

Nei dead time i diodi permettono la formazione di un percorso per la corrente che attraversa l'induttore. La quale altrimenti subirebbe una brusca interruzione e questo nelle simulazioni analogiche di SPICE non permetterebbe alla simulazione di convergere, mentre nelle simulazioni discrete che si vogliono realizzare, si andrebbe ad annullare una variabile di stato del sistema.

Si è scelto di rappresentare anche il diodo come un resistore, il cui valore di resistenza viene aggiornato ad ogni passo temporale in funzione della corrente secondo le formule 6.1 e 6.2

$$V_D = \ln\left(\frac{I_D + I_S}{I_S}\right) \cdot \eta V_{th} \tag{6.1}$$

$$R_D = \frac{V_D}{I_D + 1 \cdot 10^{-12}} \cdot R_S \tag{6.2}$$

Il passo temporale a cui si acccenna non è il periodo di switching del dispositivo, ma un periodo di campionamento diverso e necessariamente più breve. Si è scelto di inserire nel modello un fattore di sovraccampionamento che l'utilizzatore può modificare.

Per semplificare il modello, il parallelo d'interruttori e diodi e la serie dei diodi di carico vengono trattati come un'unica impedenza il cui valore viene calcolato ad ogni iterazione. Quindi lo schema in figura 6.1 diventa:

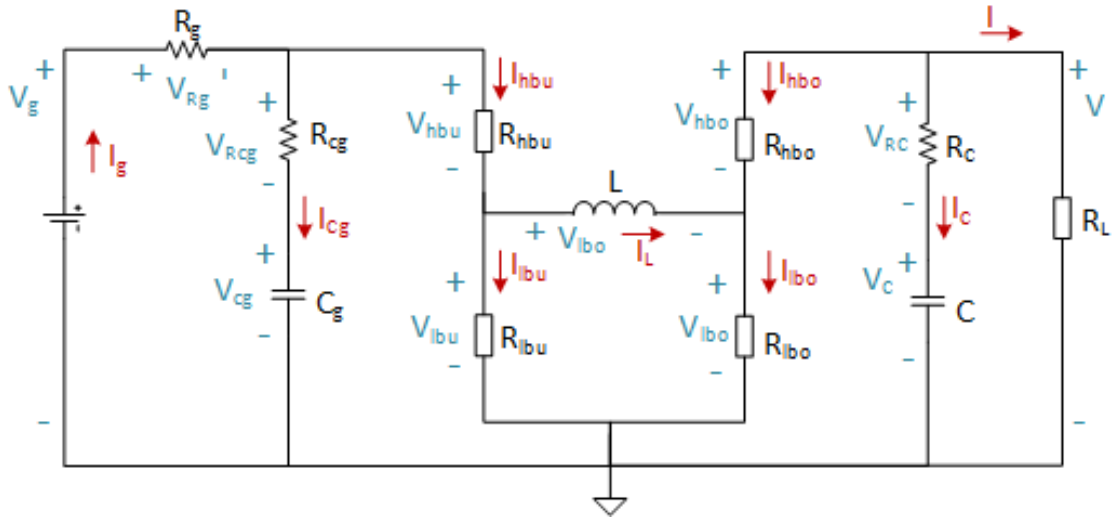

Figura 6.2.: Schema del modello del PLANT

Il quale si può rappresentare anche con il sistema:

$$\begin{cases} V_{rg} - I_g R_g = 0 \\ V_{Rcg} - I_{cg} R_{cg} = 0 \\ V_{cg} - I_{cg}/C_g = V_{cg0} \\ V_{hbu} - I_{hbu} R_{hbu} = 0 \\ V_{lbu} - I_{lbu} R_{lbu} = 0 \\ I_L - V_L/L = I_{L0} \\ V_{hbo} - I_{hbo} R_{hbo} = 0 \\ V_{lbo} - I_{lbo} R_{lbo} = 0 \\ V_C - I_C/C = V_{C0} \\ V_{RC} - I_C R_C = 0 \\ V - IR = 0 \\ -V_{Rg} - V_{cg} - V_{rcg} = -V_g \\ -V_{Rg} - V_{hbu} - V_{hbu} = -V_g \\ V_{lbu} - V_l - V_{lbo} = 0 \\ -V_{hbo} - V_{lbo} + V = 0 \\ -V_C - V_{RC} + V = 0 \\ I_g - I_{cg} - I_{hbu} = 0 \\ I_{hbu} - I_{lbu} - I_l = 0 \\ I_l - I_{hbo} - I_{lbo} = 0 \\ -I_{hbo} - I_c - I = 0 \end{cases} \tag{6.3}$$

6.2. MATLAB

Prima di creare il modello in VHDL, si è deciso di realizzare un riferimento in MATLAB.

MATLAB® (contrazione di MATrix LABoratory) è un linguaggio di alto livello e un ambiente interattivo per il calcolo numerico, l'analisi e la visualizzazione dei dati e la programmazione. MATLAB consente di analizzare dati, sviluppare algoritmi e creare modelli e applicazioni.

Il linguaggio, gli strumenti e le funzioni matematiche incorporate consentono di esplorare diversi approcci e di raggiungere una soluzione più velocemente rispetto all'uso di fogli di calcolo o di linguaggi di programmazione tradizionali, quali C/C++ o Java™.

Inoltre, i sistemi di equazioni a tempo discreto in linguaggio MATLAB sono – escludendo qualche differenza nella sintassi – completamente trasportabili in VHDL.

In MATLAB per rappresentare il sistema 6.3 nella pagina precedente si è utilizzato una rappresentazione matriciale. Ad ogni ciclo vengono aggiornati i parametri della matrice e si calcola il risultato.

```
1  function [V, iL, Vsw_buck, Vsw_boost, Iload, debug]=
       plant_model_buck_boost_led_v_0_0( Vg,  hs_drv_buck, ls_drv_buck,
       hs_drv_boost, ls_drv_boost,ShortTrigger, parameters);
2  %% 'parameters' è un argomento di tipo struct che contiene tutti valori
       numerici
3
4  %% DEFINIZIONE DELLE VARIABILI DI STATO
5   persistent Vcgo Vco Ilo
6   persistent Rh_on Rh_off Rl_on Rl_off Rg  Rcg Cg L C Rc R
7  %% INIZIALIZZAZIONE DELLE VARIABILI DI STATO
8     if isempty (Vco)
9       Vco  = 0;
10      Ilo  = 0;
11      Vcgo = 0;
12
13      Rh_on  = parameters.BB.Rp_on;
14      Rh_off = parameters.BB.Rp_off;
15      Rl_on  = parameters.BB.Rn_on;
16      Rl_off = parameters.BB.Rn_off;
17
18      Rg  = parameters.BB.Rg;
19      Rcg = parameters.BB.Rcg;
20      Cg  = parameters.BB.Cg*parameters.Fsw*parameters.analog_OS;
21
22      L   = parameters.BB.L*parameters.Fsw*parameters.analog_OS;
23      C   = parameters.BB.C*parameters.Fsw*parameters.analog_OS;
24      Rc  = parameters.BB.Rc;
25      R   = 1e6;
26    end
27 %% CODICE
28 % ASSEGNAZIONE DELLE RESISTENZE DEGLI INTERRUTTORI
29    Is = 1e-4;
30    n  = 1;
```

```
31    Vt = 25e-3;
32
33    if hs_drv_buck > 0.5
34      Rhbu= Rh_on;
35    else
36      Rhbu = Rh_off;
37    end
38    if ls_drv_buck > 0.5
39      Rlbu= Rl_on;
40    else
41      Rlbu = Rl_off;
42    end
43    if hs_drv_boost > 0.5
44      Rhbo= Rh_on;
45    else
46      Rhbo = Rh_off;
47    end
48    if ls_drv_boost > 0.5
49      Rlbo= Rl_on;
50    else
51      Rlbo = Rl_off;
52    end
53  %% RAPPRESENTAZIONE DEL SISTEMA A MATRICE
54  % X =[ Ig; Vrg; Vcg; Vrcg; Icg; Vhbu; Ihbu; Vlbu; Ilbu; Il;
55  %      Vl; Vhbo; Ihbo; Vlbo; Ilbo; Ic; Vc; Vrc; I; V]
56  BB =[
57       -Rg  1    0    0     0    0     0     0    0      0    0     0    0      0    0      0    0    0    0  0;
58         0  0    0    1  -Rcg    0     0     0    0      0    0     0    0      0    0      0    0    0    0  0;
59         0  0    1    0  -1/Cg   0     0     0    0      0    0     0    0      0    0      0    0    0    0  0;
60         0  0    0    0     0    1  -Rhbu    0    0      0    0     0    0      0    0      0    0    0    0  0;
61         0  0    0    0     0    0     0     1  -Rlbu    0    0     0    0      0    0      0    0    0    0  0;
62         0  0    0    0     0    0     0     0    0      1  -1/L    0    0      0    0      0    0    0    0  0;
63         0  0    0    0     0    0     0     0    0      0    0     1  -Rhbo    0    0      0    0    0    0  0;
64         0  0    0    0     0    0     0     0    0      0    0     0    0      1  -Rlbo    0    0    0    0  0;
65         0  0    0    0     0    0     0     0    0      0    0     0    0      0    0   -1/C    1    0    0  0;
66         0  0    0    0     0    0     0     0    0      0    0     0    0      0    0    -Rc    0    1    0  0;
67         0  0    0    0     0    0     0     0    0      0    0     0    0      0    0      0    0    0   -R  1;
68         0 -1   -1   -1     0    0     0     0    0      0    0     0    0      0    0      0    0    0    0  0;
69         0 -1    0    0     0   -1     0    -1    0      0    0     0    0      0    0      0    0    0    0  0;
70         0  0    0    0     0    0     0     1    0      0   -1     0    0     -1    0      0    0    0    0  0;
71         0  0    0    0     0    0     0     0    0      0    0    -1    0     -1    0      0    0    0    0  1;
72         0  0    0    0     0    0     0     0    0      0    0     0    0      0    0      0   -1   -1    0  1;
73         1  0    0    0    -1    0    -1     0    0      0    0     0    0      0    0      0    0    0    0  0;
74         0  0    0    0     0    0     1     0   -1     -1    0     0    0      0    0      0    0    0    0  0;
```

```
75      0  0  0  0  0  0  0  0  0  1  0  0  1  0 -1  0  0  0  0 0;
76      0  0  0  0  0  0  0  0  0  0  0  0 -1  0  0 -1  0  0 -1 0];
77
78  E=[ 0; 0; -Vcgo; 0; 0; -Ilo; 0; 0; -Vco;
79      0; 0; Vg; Vg; 0; 0; 0; 0; 0; 0; 0];
80
81 X=BB\(-E);
82
83 %% USCITE
84   iL         = X(10);
85   Vsw_buck   = X(8);
86   Vsw_boost  = X(14);
87   V          = X(20);
88   Iload      = X(19);
89  %debug =  [Iin; -Irn; Ip];
90   debug =  [X(1); X(3)+X(4)];
91
92 %% CALCOLO DELLE RESISTENZE DEI LED
93   Is       = parameters.LED_Is;
94   N        = parameters.LED_N;
95   Vth      = parameters.LED_Vth;
96   Rs       = parameters.LED_Rs;
97   Rschain  = parameters.LED_Rschain;
98
99   if ShortTrigger < 0.5
100    LED_Nled = parameters.LED_Nled;
101  else
102    LED_Nled = parameters.LED_Nled - parameters.LED_Nshorted;
103  end
104  Iled=X(19);
105  if Iled > 0
106    Vd = log( (Iled + Is)/Is ) * N * Vth ;
107    Rled = Vd / (Iled+1e-12) + Rs;
108  else
109    Rled = 1e6;
110  end
111  R = LED_Nled * Rled +  Rschain;
112 %% AGGIORNAMENTO DELLE VARIABILI DI STATO
113  Vc   = X(17);
114  Il   = X(10);
115  Vcg  = X(3);
116  Vco  = Vc;
117  Ilo  = Il;
118  Vcgo = Vcg;
```

La funzione presentata è stata simulata all'interno in un modello Simulink, ottenendo una buona approssimazione dei risultati attesi dal modello ideale del PLANT.

In seguito si è convertito il modello MATLAB in forma matriciale in linguaggio VHDL e si è inserito in una simulazione di prova, grazie anche ad un package esterno per i calcoli sulle matrici. Quest'ultimo però contiene solo algoritmi base in alcun modo paragonabili agli algoritmi ottimizzati di MATLAB. In particolare il calcolo d'inversione della matrice – necessario ad ogni ciclo – viene realizzato ricorsivamente e nel caso presentato (matrice 20x20) il simulatore non stato in grado di raggiungere una soluzione.

Anche nell'ipotesi di riuscire a migliorare l'algoritmo, è evidente quanto la tecnica adottata non sia conveniente poiché il numero dei calcoli richiesti ad ogni passo temporale andrebbero ad aumentare il tempo di simulazione, facendo perdere il vantaggio guadagnato col modello.

6.3. MuPAD

Dopo queste considerazioni si è deciso di cambiare strategia. Il nuovo approccio è quello di scrivere all'interno del modello VHDL la soluzione esplicita del sistema di equazioni 6.3 a pagina 64. A questo scopo si è utilizzato MuPAD.

MuPAD comprende un linguaggio ottimizzato per operare su espressioni matematiche in forma simbolica ed un efficiente risolutore. Questo strumento è disponibile all'interno del Symbolyc Math Toolbox di MATLAB.

Listing 6.1: File MuPAD

```
1  n01 := V_R_gen - I_gen * R_gen = 0
2  n02 := V_R_cap_g - I_C_cap_g * R_cap_g = 0
3  n03 := V_C_cap_g - V_C_cap_g0 - I_C_cap_g / C_cap_g = 0
4  n04 := V_R_mos_hbu - I_R_mos_hbu * R_mos_hbu = 0
5  n05 := V_R_mos_lbu - I_R_mos_lbu * R_mos_lbu = 0
6  n06 := I_L_ind - I_L_ind0 - V_L_ind / L_ind = 0
7  n07 := V_R_mos_hbo - I_R_mos_hbo * R_mos_hbo = 0
8  n08 := V_R_mos_lbo - I_R_mos_lbo * R_mos_lbo = 0
9  n09 := V_C_cap_c - V_C_cap_c0 - I_C_cap_c / C_cap_c = 0
10 n10 := V_R_cap_c - I_C_cap_c * R_cap_c = 0
```

```
n11 := V_R_load - I_R_load * R_load = 0
n12 := - V_R_gen - V_C_cap_g - V_R_cap_g + V_gen = 0
n13 := - V_R_gen - V_R_mos_hbu - V_R_mos_lbu + V_gen = 0
n14 := V_R_mos_lbu - V_L_ind - V_R_mos_lbo = 0
n15 := - V_R_mos_hbo - V_R_mos_lbo + V_R_load = 0
n16 := - V_C_cap_c - V_R_cap_c + V_R_load = 0
n17 := I_gen - I_C_cap_g - I_R_mos_hbu = 0
n18 := I_R_mos_hbu - I_R_mos_lbu - I_L_ind = 0
n19 := I_L_ind + I_R_mos_hbo - I_R_mos_lbo = 0
n20 := - I_R_mos_hbo - I_C_cap_c - I_R_load = 0

S := linsolve({n01, n02, n03, n04, n05, n06, n07, n08, n09, n10,
    n11, n12, n13, n14, n15, n16, n17, n18, n19, n20}, {V_R_gen,
    I_gen, V_R_cap_g, V_C_cap_g, I_C_cap_g, V_R_mos_hbu,
   I_R_mos_hbu, V_R_mos_lbu, I_R_mos_lbu, I_L_ind, V_L_ind,
   V_R_mos_hbo, I_R_mos_hbo, V_R_mos_lbo, I_R_mos_lbo, V_C_cap_c
    , I_C_cap_c, V_R_cap_c, V_R_load, I_R_load})

fprint(Unquoted, Text, "solve_5.txt", S)
fprint(Unquoted, Text, "solve_BUCK-BOOST_uPAD_5.txt" , generate
   ::MATLAB(S))
```

Listing 6.2: Frammento del codice VHDL generato dal file MuPAD

```
            -- [...]
I_C_cap_c <= - ( C_cap_c*L_ind*R_gen*R_load*V_C_cap_c0
                 + C_cap_c*L_ind*R_gen*R_mos_hbo*V_C_cap_c0
                 + C_cap_c*L_ind*R_gen*R_mos_lbo*V_C_cap_c0
                 + C_cap_c*L_ind*R_load*R_mos_hbu*V_C_cap_c0
                 + C_cap_c*L_ind*R_load*R_mos_lbu*V_C_cap_c0
                 + C_cap_c*L_ind*R_mos_hbo*R_mos_hbu*V_C_cap_c0
                 + C_cap_c*L_ind*R_mos_hbo*R_mos_lbu*V_C_cap_c0
                 + C_cap_c*L_ind*R_mos_lbo*R_mos_hbu*V_C_cap_c0
                 + C_cap_c*L_ind*R_mos_lbo*R_mos_lbu*V_C_cap_c0
                 + C_cap_c*R_gen*R_load*R_mos_lbo*V_C_cap_c0
                 + C_cap_c*R_gen*R_load*R_mos_lbu*V_C_cap_c0
                 + C_cap_c*R_gen*R_mos_hbo*R_mos_lbo*V_C_cap_c0
                 + C_cap_c*R_gen*R_mos_hbo*R_mos_lbu*V_C_cap_c0
```

```
15                + C_cap_c*R_gen*R_mos_lbo*R_mos_lbu*V_C_cap_c0
16                + C_cap_c*R_load*R_mos_lbo*R_mos_hbu*V_C_cap_c0
17                + C_cap_c*R_load*R_mos_lbo*R_mos_lbu*V_C_cap_c0
18                + C_cap_c*R_load*R_mos_hbu*R_mos_lbu*V_C_cap_c0
19                + C_cap_c*R_mos_hbo*R_mos_lbo*R_mos_hbu*V_C_cap_c0
20                + C_cap_c*R_mos_hbo*R_mos_lbo*R_mos_lbu*V_C_cap_c0
21                + C_cap_c*R_mos_hbo*R_mos_hbu*R_mos_lbu*V_C_cap_c0
22            -- [...]
```

Questo espediente presenta alcuni svantaggi rispetto alla versione matriciale. Il più evidente guardando il frammento 6.2 è la quasi totale perdita della leggibilità del codice e conseguenza diretta è che la ricerca di eventuali errori si può attuare solo sui risultati delle simulazioni. A titolo di esempio si pensi a cosa potrebbe comportare riportare nel sistema un segno invertito.

Inoltre si perde la riutilizzabilità del codice, ovvero in seguito ad una semplice modifica del sistema le equazioni devono essere riscritte in MuPad e riportate in VHDL, mentre in forma matriciale si sarebbe potuti andare a modificare direttamente il codice VHDL del blocco.

6.4. VHDL

Nella sua versione finale si è deciso di dividere il codice del modello del PLANT in due blocchi VHDL, il primo contiene le equazioni ricavate tramite MuPAD, il secondo istanzia il primo blocco e contiene quanto necessario per il suo utilizzo. Viene riportato nell'appendice a pagina 79, il codice del secondo blocco, in cui si può notare l'inizializzazione delle variabili che gestisce anche il reset delle stesse, il clock di aggiornamento del sistema, i registri per gestire le variabili di stato ed alcune funzioni di conversione per i segnali da passare al modello del dispositivo.

7. Risultati e conclusioni

Gli obiettivi che in fase di progetto tendevano ad una buona approssimazione dei segnali analogici in simulazioni con velocità molto superiori al simulatore SPICE, sono stati pienamente raggiunti ed approvati.

Con i modelli realizzati si riescono ad ottenere simulazioni top-level rappresentanti decine di millisecondi in appena qualche minuto, anche salvando tutti i segnali di ogni livello gerarchico.

Dalla figura 7.1 a pagina 73 alla figura 7.4 a pagina 76 vengono riportate diverse schermate del simulatore con i segnali tratti dal modello RN del convertitore DC/DC in diverse configurazioni del dispositivo.

In seguito si è sperimentata una simulazione equivalente con i blocchi analogici interamente a schematico SPICE per confrontare i tempi di elaborazione. Quello che ne è conseguito è stato l'impiego di alcuni giorni di tempo per infine concludere le simulazioni prima che riuscissero a completare la fase di Soft Start.

Infine, per avere comunque un termine di paragone, si è ricorso all'utilizzo del tool CustomSim-XA di Synopsys – il quale rientra nella categoria dei simulatori FastSPICE – con cui si è lanciata una simulazione dell'intero dispositivo a livello di transistor. In questa prova si è configurato il dispositivo affinché rimanesse nella modalità di regolazione BUCK, essendo essa la più rapida da raggiungere, inoltre si sono salvati solo i segnali dei livelli gerarchici superiori e si sono fissati livelli di risoluzione sufficientemente bassi per aumentare la velocità. Anche in queste condizioni la simulazione ha richiesto poco più di 20 ore per ottenere circa due millisecondi.

Nelle figure 7.5 e 7.6 a pagina 77 e 78 vengono proposti alcuni confronti tra le forma d'onde ricavate dal simulatore CustomSim-XA e il modello realizzato con la tecnica RN.

In generale gli esiti ottenuti con il modello RN si sono rivelati particolarmente validi da essere considerati per il loro utilizzo nel processo di verifica per le modifiche future del circuito – in particolare del blocco della logica –, con uno speciale interesse alla possibilità

di svolgere simulazioni top level e contemporaneamente salvare ogni segnale interno al dispositivo.

Nello specifico il modello è stato utilizzato per sviluppare una sequenza di test di alcuni casi particolari la cui durata complessiva di simulazione è 250 $[ms]$ ed il tempo di esecuzione poco superiore ai trenta minuti.

Con questo lavoro si è dimostrato che il Real Number Modeling dei circuiti a segnale analogici e MS, risulta essere alquanto accessibile ed intuitivo, nonostante sussistano numerose precauzioni e controlli da assumere al fine di garantire l'esecuzione col minimo rischio in un flusso di verifica.

Inoltre, questi modelli consentono di rimanere completamente all'interno dell'ambiente di simulazione digitale, permettendo di sfruttare in maniera più efficace le caratteristiche MDV quali la generazione randomica di test, le misure di coverage e le asserzioni.

Alcuni esempi sono stati presentati per illustrare i concetti di base della modellizzazione a numeri reali in simulatori digitali. Nonostante questi esempi risultino essere notevolmente semplici, si è brevemente illustrato come gli stessi concetti possano essere facilmente estesi a modelli più complessi.

Lo svolgimento di questo caso studio ha richiesto una particolare attenzione ed impegno nel momento in cui si è dovuta attuare la conversione di ogni schematico analogico in un insieme di soluzioni esplicite – soprattutto in questo caso in cui la stesura dei modelli non è stata effettuata dalla medesima persona che si è occupata degli schematici – e sempre mantenendo la stessa interfaccia dei blocchi analogici.

In aggiunta a quanto illustrato nel paragrafo precedente, la tecnica adottata per la generazione delle soluzioni delle equazioni del modello del convertitore BUCK-BOOST presenta una buona approssimazione dei segnali e può essere vantaggiosa per strutture facenti parte del testbench, come in questo caso analizzato. Ma è sconsigliabile da utilizzare per i modelli interni del sistema, poiché devono risultare facilmente modificabili per essere adeguati a possibili variazioni future del progetto.

Inoltre, si ribadisce che lo scopo principe rimane la velocità di simulazione e per questo dove possibile – senza compromettere la funzionalità globale del dispositivo – si è cercato di realizzare modelli semplici come quello visto per il regolatore ideale di tensione. Tuttavia, ogni semplificazione richiede comunque un esame che verifichi che quanto viene trascurato non sia una fonte di errore in simulazione.

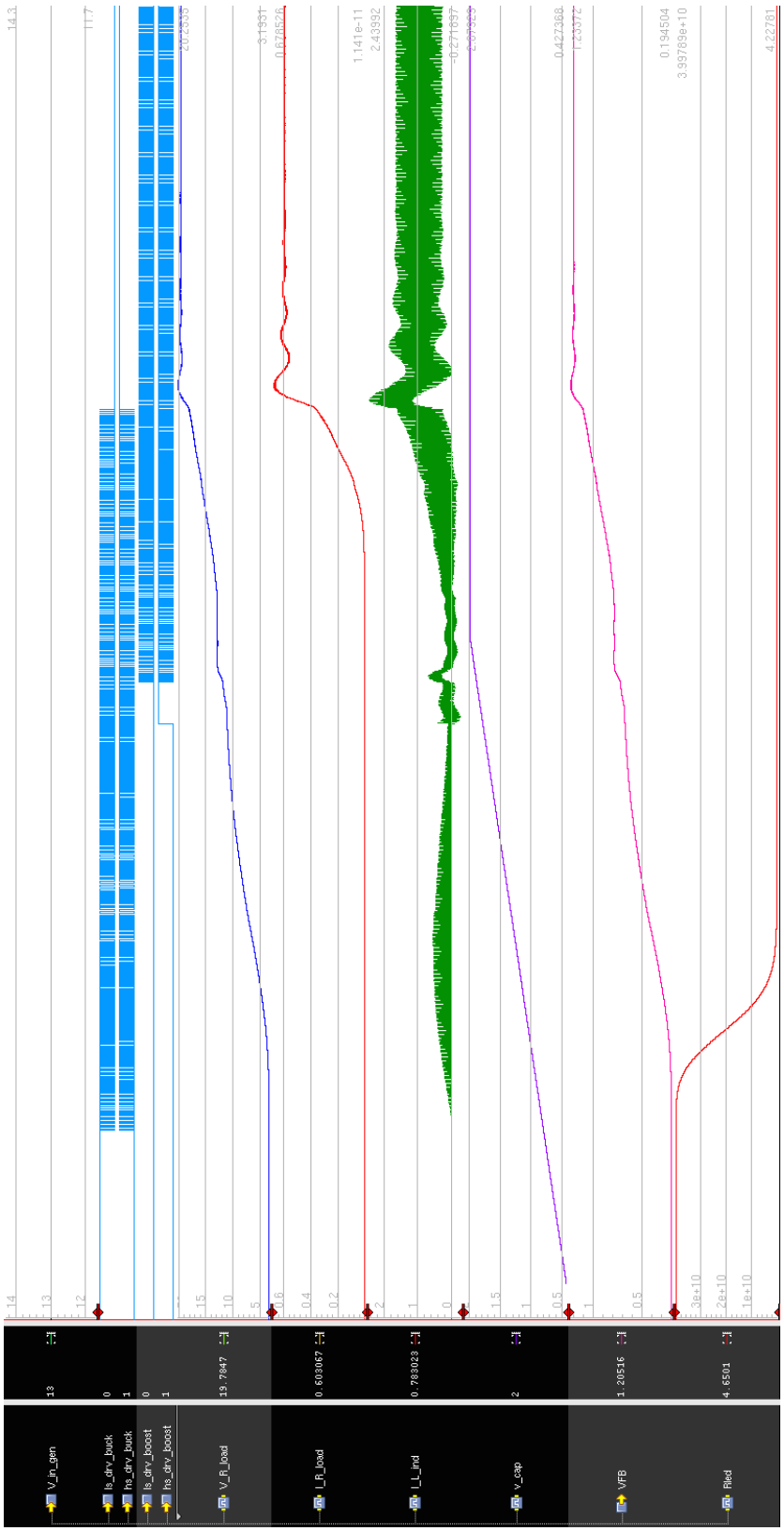

Figura 7.1.: Avvio e regolazione del dispositivo – si noti, guardando i segnali di controllo degli interruttori, come il dispositivo passi dalle modalità BUCK, BUCK-BOOST a BOOST. Nell'ultima riga si può osservare l'andamento della resistenza del Diodo

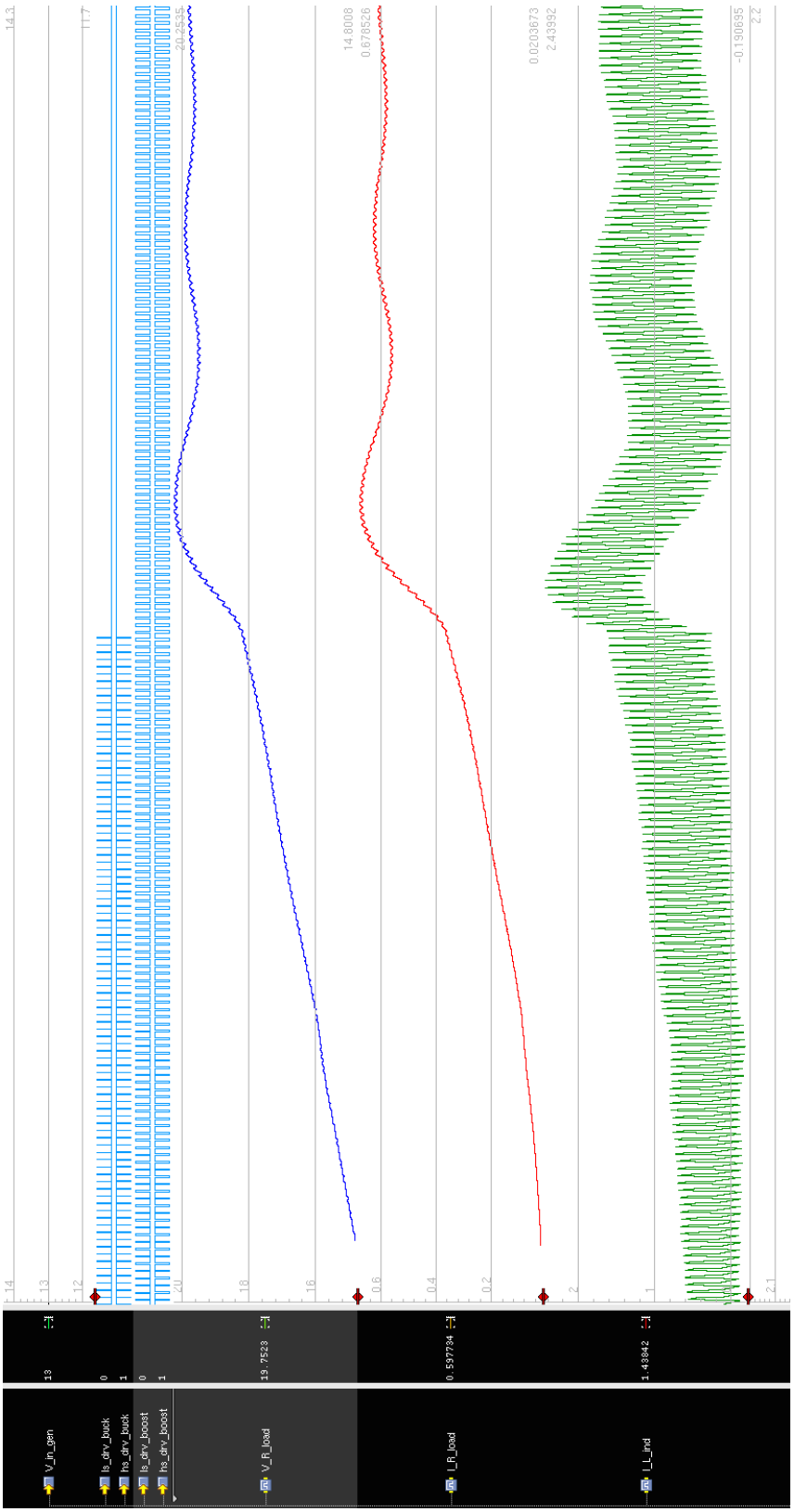

Figura 7.2.: Dettaglio del passaggio dalle modalità BUCK-BOOST alla modalità BOOST

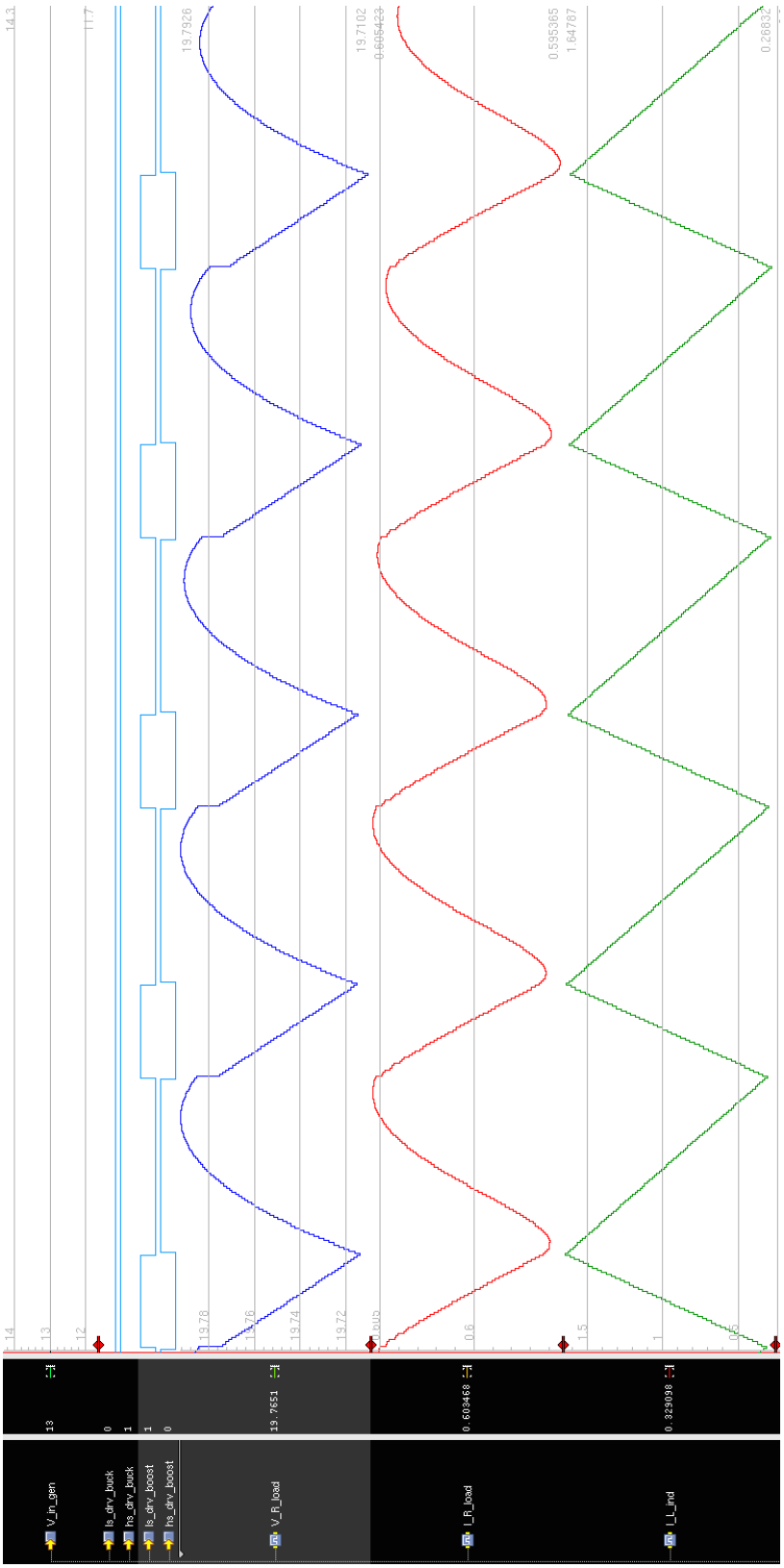

Figura 7.3.: Dettaglio di alcuni cicli dei segnali nella modalità di regolazione BOOST

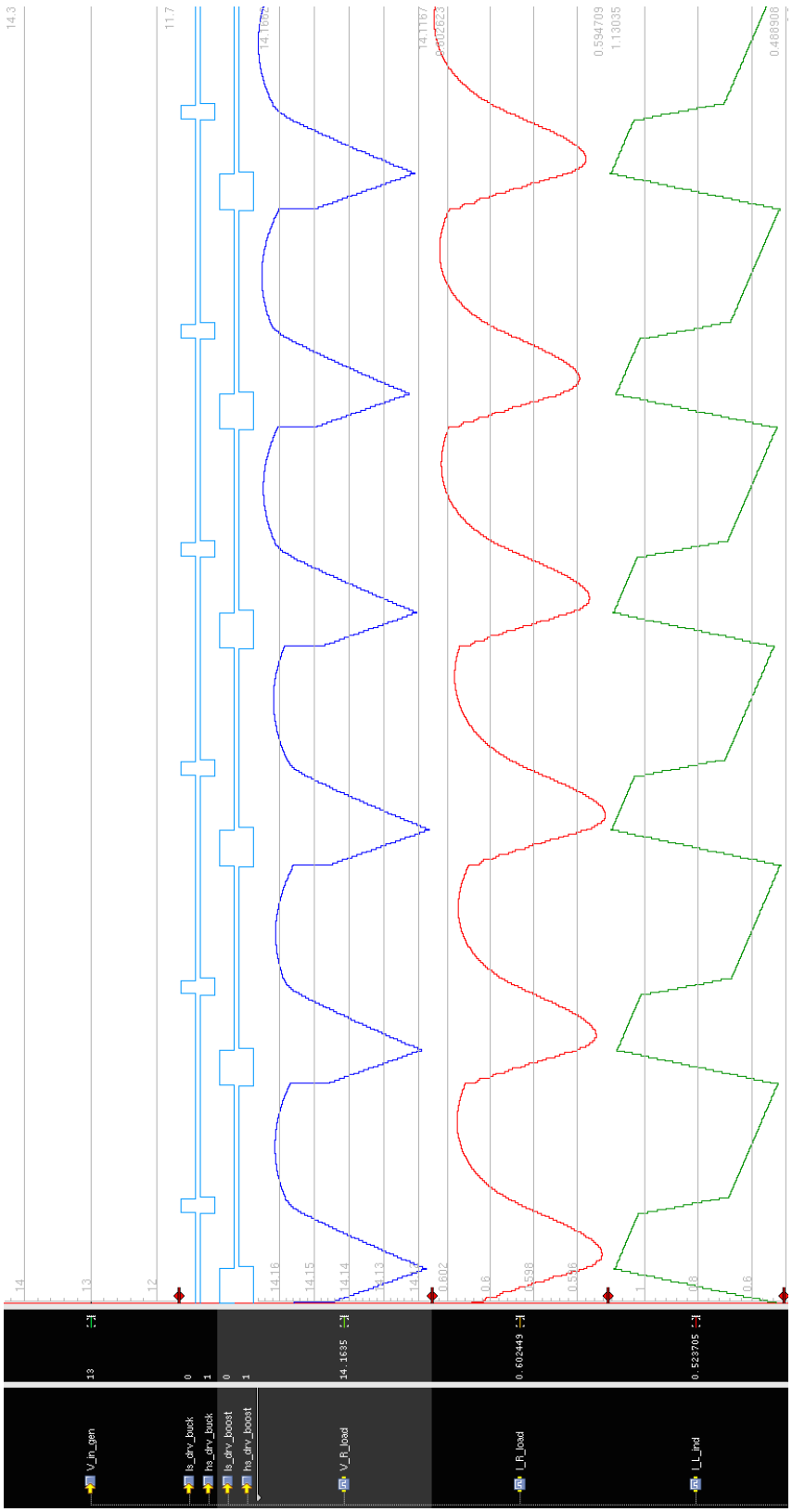

Figura 7.4.: Dettaglio di alcuni cicli dei segnali nella modalità di regolazioneBUCK-BOOST

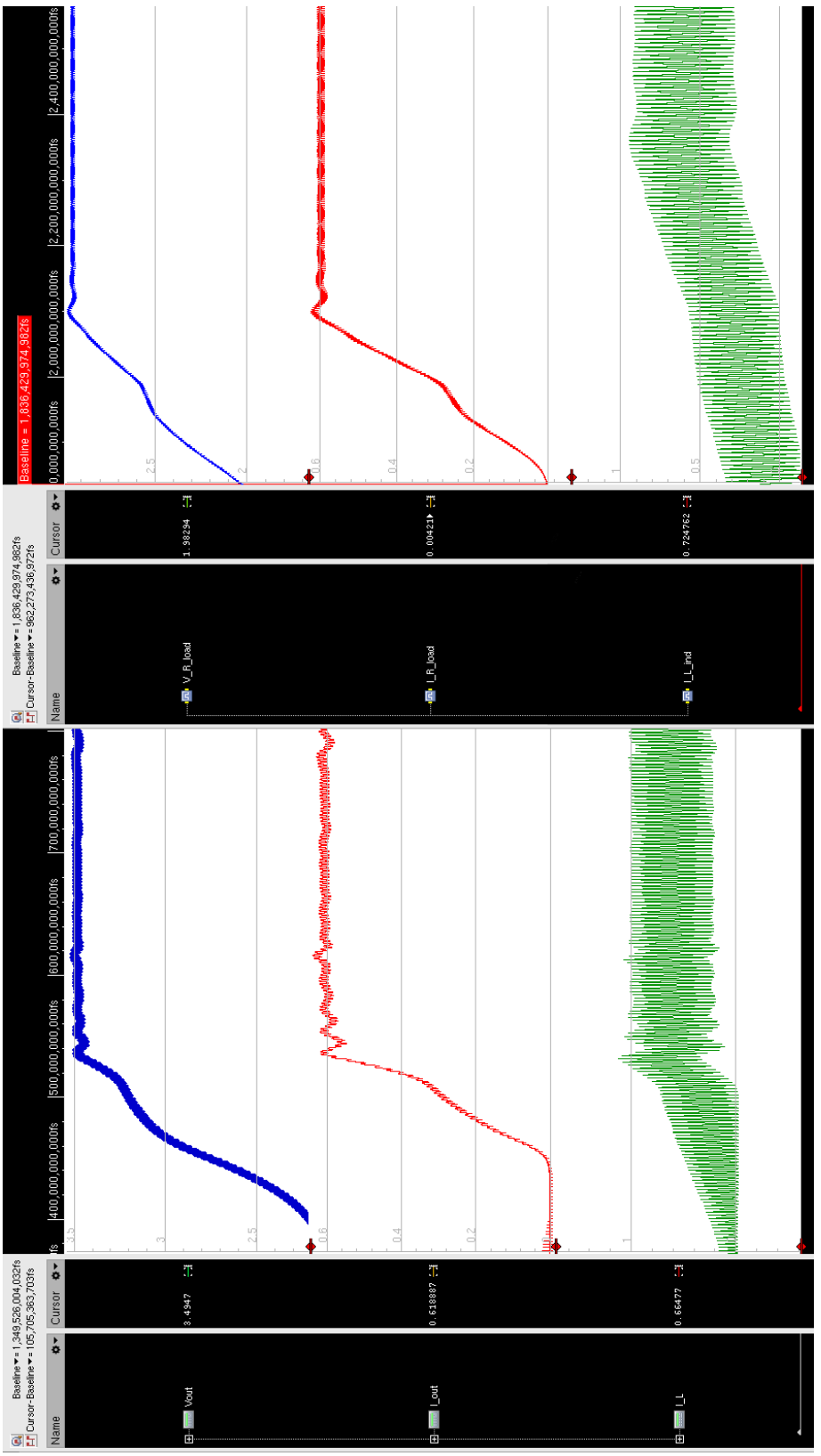

Figura 7.5.: Confronto della fase di partenza tra i segnali dela tensione sul carico, della corrente di uscita da regolare e della corrente che attraversa l'induttore. A sinistra dell'immagine sono presenti i segnali risultanti dalla simulazione FastSPICE , a destra invece i segnali del modello RNM

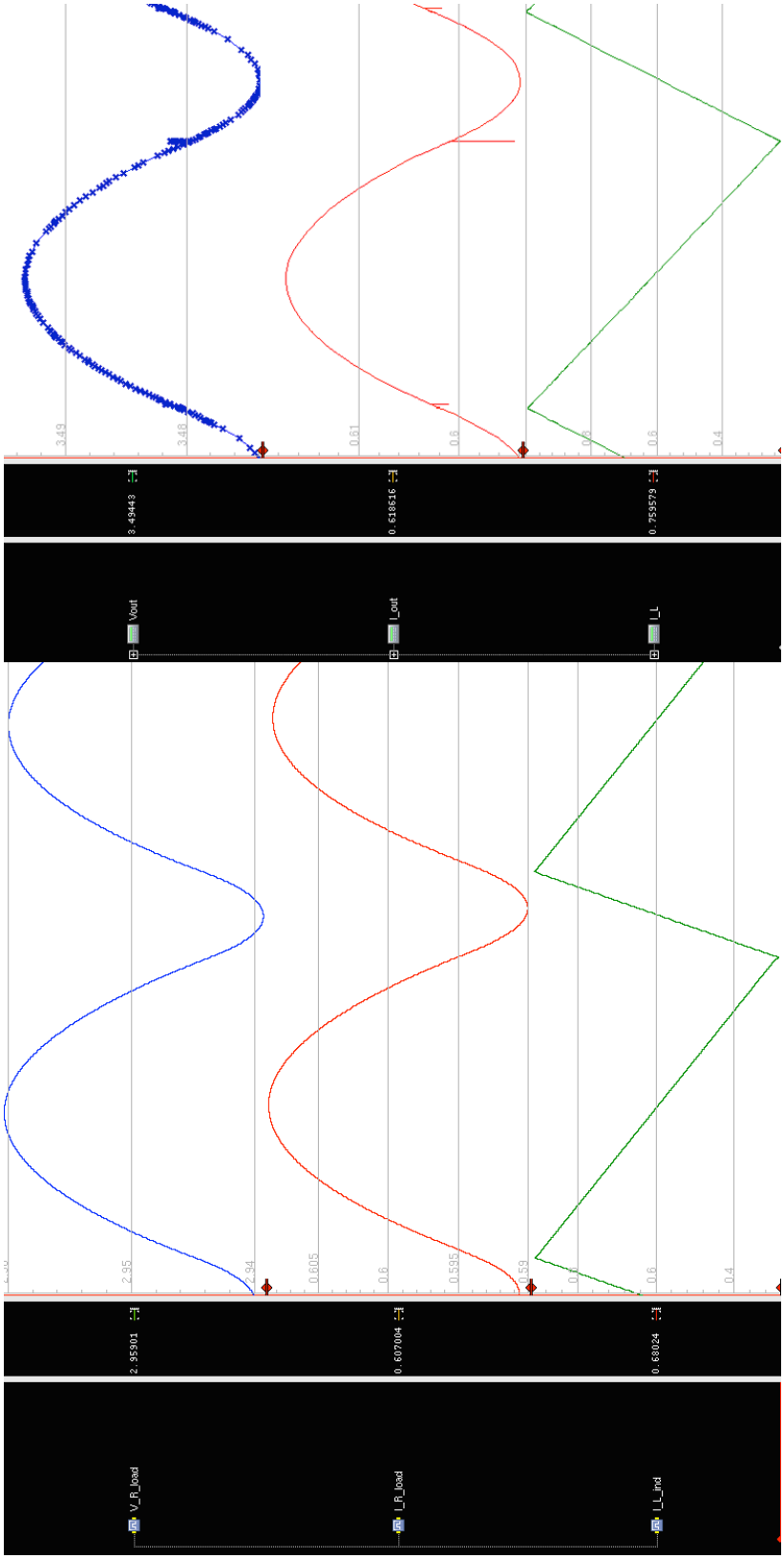

Figura 7.6.: Confronto più dettagliato tra i segnali dela tensione sul carico, della corrente di uscita da regolare e della corrente che attraversa l'induttore. A destra dell'immagine sono presenti i segnali risultanti dalla simulazione FastSPICE , a sinistra invece i segnali del modello RNM

8. Appendice

8.1. Codice VHDL

```
1  LIBRARY IEEE,STD;
2  USE IEEE.std_logic_1164.all;
3  USE IEEE.math_real.all ;
4  LIBRARY analog_hw;
5  USE analog_hw.analog_pack.all;
6
7  ENTITY plant_model_buck_boost_led IS
8    PORT(
9          -- Tensione del generatore di ingresso
10         V_in_gen       : IN  analog_ut;
11         -- Numero di LED da pilotare
12         LED_Nled       : IN  integer;
13         -- Numero di LED che il comando short trigger cortocircuita
14         LED_Nshorted   : IN  integer;
15         -- gate driver del MOS M1 (= HSGD1)
16         hs_drv_buck    : IN  std_ulogic;
17         -- gate driver del MOS M2 (= LSGD1)
18         ls_drv_buck    : IN  std_ulogic;
19         -- gate driver del MOS M4 (= HSGD2)
20         hs_drv_boost   : IN  std_ulogic;
21         -- gate driver del MOS M3 (= LSGD2)
22         ls_drv_boost   : IN  std_ulogic;
23         reset_n        : IN  std_ulogic;
24         -- se '1' cortocircuita (LED_Nshorted) LED
25         short_trigger  : IN  std_ulogic;
26         -- Sensing di tensione del nodo intermedio del ponte 1
27         SWN_buck       : OUT analog_ut;
28         -- Sensing di tensione del nodo intermedio del ponte 2
29         SWN_boost      : OUT analog_ut;
30         -- Sensing di tensione per la misura della corrente sull'induttore
```

```
31          SWCS            : OUT analog_ut;
32          -- Feedback di tensione del carico
33          VFB             : OUT analog_ut;
34          -- Feedback di corrente positivo del carico
35          FBH             : OUT analog_ut;
36          -- Feedback di corrente negativo del carico
37          FBL             : OUT analog_ut
38          );
39  END plant_model_buck_boost_led;
40
41  ARCHITECTURE beh of plant_model_buck_boost_led IS
42    COMPONENT BB_plant_solved_system
43      PORT(
44           V_R_gen      : OUT real;
45           I_gen        : OUT real;
46           R_gen        : IN  real;
47           V_R_cap_g    : OUT real;
48           I_C_cap_g    : OUT real;
49           R_cap_g      : IN  real;
50           V_C_cap_g    : OUT real;
51           V_C_cap_g0   : IN  real;
52           C_cap_g      : IN  real;
53           V_R_mos_hbu  : OUT real;
54           I_R_mos_hbu  : OUT real;
55           R_mos_hbu    : IN  real;
56           V_R_mos_lbu  : OUT real;
57           I_R_mos_lbu  : OUT real;
58           R_mos_lbu    : IN  real;
59           I_L_ind      : OUT real;
60           I_L_ind0     : IN  real;
61           V_L_ind      : OUT real;
62           L_ind        : IN  real;
63           V_R_mos_hbo  : OUT real;
64           I_R_mos_hbo  : OUT real;
65           R_mos_hbo    : IN  real;
66           V_R_mos_lbo  : OUT real;
67           I_R_mos_lbo  : OUT real;
68           R_mos_lbo    : IN  real;
69           V_C_cap_c    : OUT real;
70           V_C_cap_c0   : IN  real;
71           I_C_cap_c    : OUT real;
```

```
72          C_cap_c      : IN  real;
73          V_R_cap_c    : OUT real;
74          R_cap_c      : IN  real;
75          V_R_load     : OUT real;
76          I_R_load     : OUT real;
77          R_load       : IN  real;
78          V_gen        : IN  real
79          );
80    END COMPONENT;
81  -- Local SIGNALS and CONSTANTS ---
82    -- PARAMETRI DEL CLOCK INTERNO
83    signal    semiperiod    : time;
84    signal    int_clk       : std_ulogic := '0';
85    signal    freq_sw       : real;                    -- Frequenza di Switching
86    -- this is 1 ns because '/' operation for type time is an Euclidean division
87    constant one_nsec       : time := 1 ns;
88    -- LED
89    signal    LED_Num_led   : real;
90    signal    I_led         : real;
91    constant LED_Is         : real := 15.538e-15;
92    constant LED_N          : real := 2.8911;
93    constant LED_Vth        : real := 0.026;
94    constant LED_Rs         : real := 0.7484;
95    constant LED_Rschain    : real := 0.25;
96    signal    Rled          : real := 1.0e6;
97    -- PARAMETRI DEL PLANT
98    constant Rh_on          : real := 1.5e-2;
99    constant Rh_off         : real := 10.0e6;
100   constant Rl_on          : real := 1.5e-2;
101   constant Rl_off         : real := 10.0e6;
102   constant MOS_diode_Is   : real := 1.0e-14;
103   constant MOS_diode_N    : real := 2.0;
104   signal    Cg_value      : real := 1.0e-6;
105   signal    L_value       : real := 10.0e-6;
106   signal    C_value       : real := 10.0e-6;
107   signal    Rg            : real := 0.0;
108   signal    Rcg           : real := 0.0;
109   signal    Cg            : real;
110   signal    Rhbu          : real;
111   signal    Rlbu          : real;
112   signal    L             : real;
```

```
113   signal    Rhbo          : real;
114   signal    Rlbo          : real;
115   signal    C             : real;
116   signal    Rc            : real :=  0.0;
117   signal    R             : real;
118   signal    Vcgo          : real;
119   signal    Ilo           : real;
120   signal    Vco           : real;
121   signal    Vg            : real;
122   -- SEGNALI PER il SISTEMA DEL PLANT
123   signal    V_R_gen       : real;
124   signal    I_gen         : real;
125   signal    V_R_cap_g     : real;
126   signal    I_C_cap_g     : real;
127   signal    V_C_cap_g     : real;
128   signal    V_R_mos_hbu   : real;
129   signal    I_R_mos_hbu   : real;
130   signal    V_R_mos_lbu   : real;
131   signal    I_R_mos_lbu   : real;
132   signal    I_L_ind       : real;
133   signal    V_L_ind       : real;
134   signal    V_R_mos_hbo   : real;
135   signal    I_R_mos_hbo   : real;
136   signal    V_R_mos_lbo   : real;
137   signal    I_R_mos_lbo   : real;
138   signal    V_C_cap_c     : real;
139   signal    I_C_cap_c     : real;
140   signal    V_R_cap_c     : real;
141   signal    V_R_load      : real;
142   signal    I_R_load      : real;
143   -- PARAMETRI PER LE USCITE
144   signal    R_SWCS        : analog_ut := 4.0e-3;
145   signal    R_VFBH        : analog_ut := 1.2e4;
146   signal    R_VFBL        : analog_ut := 1.58e5;
147 BEGIN
148 -- Concurrent Signal Assignments ----
149   -- Generatore del clock interno
150   semiperiod <= osc_period / (2.0*OS_factor);
151   int_clk    <= not(int_clk) after semiperiod when reset_n = '1' else '0';
152   freq_sw    <= 1.0 / (real(osc_period / one_nsec)/1.0e+9);
153   Vg         <= real(V_in_gen);
```

```
154   -- Normalizzazione al passo temporale delle componenti L e C
155   Cg          <= Cg_value * freq_sw * OS_factor;
156   L           <= L_value  * freq_sw * OS_factor;
157   C           <= C_value  * freq_sw * OS_factor;
158   -- Calcolo o connessioni alle uscite
159   SWN_buck    <= analog_ut(V_R_mos_lbu);
160   SWN_boost   <= analog_ut(V_R_mos_lbo);
161   SWCS        <= analog_ut( I_R_mos_lbo + I_R_mos_lbu ) * R_SWCS
162                        when     analog_known(analog_ut(I_R_mos_lbo))
163                                 or analog_known(analog_ut(I_R_mos_lbu))
164                        else analog_ut(0.0);
165   VFB         <= analog_ut(V_R_load) * ( R_VFBL / ( R_VFBH + R_VFBL ));
166   FBH         <= analog_ut(V_R_load);
167   FBL         <= analog_ut(V_R_load) - analog_ut(I_R_load * LED_Rschain);
168 -- processes ----
169   process(int_clk, reset_n)
170     variable Vd     : real;
171     variable Vd_hbu : real;
172     variable Vd_lbu : real;
173     variable Vd_hbo : real;
174     variable Vd_lbo : real;
175   begin
176     if not(reset_n = '1') then
177       LED_Num_led <=  real(LED_Nled);
178       I_led       <=  0.0;
179       Rled        <=  1.0e6;
180       Rhbu        <=  Rh_off;
181       Rlbu        <=  Rh_off;
182       Rhbo        <=  Rh_off;
183       Rlbo        <=  Rh_off;
184       R           <=  1.0e6;
185       Vcgo        <=  0.0;
186       Ilo         <=  0.0;
187       Vco         <=  0.0;
188     elsif ( int_clk'event and int_clk = '1' ) then
189       -- Aggiornamento del valore assunti dagli interruttori nel prossimo ciclo
190       if ( hs_drv_buck  = '1' ) then
191       -- interruttore chiuso
192         Rhbu <= Rh_on;
193       elsif  I_R_mos_hbu < 0.0 then
194       -- conduzione del diodo
```

```
195           Vd_hbu := log(( (-I_R_mos_hbu) + MOS_diode_Is)/ MOS_diode_Is
196                         * MOS_diode_N * LED_Vth ;
197           Rhbu   <= Vd_hbu / (-I_R_mos_hbu) ;
198         elsif  (I_L_ind < 0.0) and (ls_drv_buck  = '0') then
199         -- conduzione forzata del diodo
200           Vd_hbu := log(( (-I_L_ind) + MOS_diode_Is)/ MOS_diode_Is )
201                         * MOS_diode_N * LED_Vth ;
202           Rhbu   <= Vd_hbu / (-I_L_ind) ;
203         else
204         -- interruttore aperto
205           Rhbu <= Rh_off;
206         end if;
207         if ( ls_drv_buck  = '1' ) then
208           Rlbu <= Rl_on;
209         elsif  I_R_mos_lbu < 0.0 then
210           Vd_lbu := log(( (-I_R_mos_lbu) + MOS_diode_Is)/ MOS_diode_Is )
211                         * MOS_diode_N * LED_Vth ;
212           Rlbu   <= Vd_lbu / (-I_R_mos_lbu) ;
213         elsif  (I_L_ind > 0.0) and (hs_drv_buck  = '0') then
214           Vd_lbu := log(( (I_L_ind) + MOS_diode_Is)/ MOS_diode_Is )
215                         * MOS_diode_N * LED_Vth ;
216           Rlbu   <= Vd_lbu / (I_L_ind) ;
217         else
218           Rlbu <= Rl_off;
219         end if;
220         if ( hs_drv_boost = '1' ) then
221           Rhbo <= Rh_on;
222         elsif  I_R_mos_hbo < 0.0 then
223           Vd_hbo := log(( (-I_R_mos_hbo) + MOS_diode_Is)/ MOS_diode_Is )
224                         * MOS_diode_N * LED_Vth ;
225           Rhbo   <= Vd_hbo / (-I_R_mos_hbo) ;
226         elsif  (I_L_ind > 0.0) and (ls_drv_boost  = '0') then
227           Vd_hbo := log(( (I_L_ind) + MOS_diode_Is)/ MOS_diode_Is )
228                         * MOS_diode_N * LED_Vth ;
229           Rhbo   <= Vd_hbo / (I_L_ind) ;
230         else
231           Rhbo <= Rh_off;
232         end if;
233         if ( ls_drv_boost = '1' ) then
234           Rlbo <= Rl_on;
235         elsif  I_R_mos_lbo < 0.0 then
```

```
236           Vd_lbo := log(( (-I_R_mos_lbo) + MOS_diode_Is)/ MOS_diode_Is )
237                       * MOS_diode_N * LED_Vth ;
238           Rlbo   <= Vd_lbo / (-I_R_mos_lbo) ;
239         elsif  (I_L_ind < 0.0) and (hs_drv_boost  = '0') then
240           Vd_lbo := log(( (-I_L_ind) + MOS_diode_Is)/ MOS_diode_Is )
241                       * MOS_diode_N * LED_Vth ;
242           Rlbo   <= Vd_lbo / (-I_L_ind) ;
243         else
244           Rlbo <= Rl_off;
245         end if;
246         -- Aggiornamento delle variabili di stato per il ciclo successivo
247         Vco  <= V_C_cap_c;
248         Ilo  <= I_L_ind;
249         Vcgo <= V_C_cap_g;
250         -- Aggiornamento del valore di resistenza del carico per il ciclo successivo
251         if short_trigger = '1' then
252           LED_Num_led <= real(LED_Nled - LED_Nshorted);
253         else
254           LED_Num_led <= real(LED_Nled);
255         end if;
256         I_led <= I_R_load;
257         if I_led > 0.0 then
258           Vd   := log(( I_led + LED_Is)/ LED_Is ) * LED_N * LED_Vth ;
259           Rled <= Vd / (I_led + 1.0e-12) + LED_Rs;
260         else
261           Rled <= 1.0e6;
262         end if;
263           R <= LED_Num_led * Rled + LED_Rschain;
264       end if; -- int_clk
265     end process;
266
267 -- Instances in the design ---
268 plant_system:
269     BB_plant_solved_system
270     PORT MAP(
271               V_R_gen      =>  V_R_gen,
272               I_gen        =>  I_gen,
273               R_gen        => Rg,
274               V_R_cap_g    =>  V_R_cap_g,
275               I_C_cap_g    =>  I_C_cap_g,
276               R_cap_g      => Rcg,
```

```
277             V_C_cap_g    =>  V_C_cap_g,
278             V_C_cap_g0   => Vcgo,
279             C_cap_g      => Cg,
280             V_R_mos_hbu  =>  V_R_mos_hbu,
281             I_R_mos_hbu  =>  I_R_mos_hbu,
282             R_mos_hbu    => Rhbu,
283             V_R_mos_lbu  =>  V_R_mos_lbu,
284             I_R_mos_lbu  =>  I_R_mos_lbu,
285             R_mos_lbu    => Rlbu,
286             I_L_ind      =>  I_L_ind,
287             I_L_ind0     => Ilo,
288             V_L_ind      =>  V_L_ind,
289             L_ind        => L,
290             V_R_mos_hbo  =>  V_R_mos_hbo,
291             I_R_mos_hbo  =>  I_R_mos_hbo,
292             R_mos_hbo    => Rhbo,
293             V_R_mos_lbo  =>  V_R_mos_lbo,
294             I_R_mos_lbo  =>  I_R_mos_lbo,
295             R_mos_lbo    => Rlbo,
296             V_C_cap_c    =>  V_C_cap_c,
297             V_C_cap_c0   => Vco,
298             I_C_cap_c    =>  I_C_cap_c,
299             C_cap_c      => C,
300             V_R_cap_c    =>  V_R_cap_c,
301             R_cap_c      => Rc,
302             V_R_load     =>  V_R_load,
303             I_R_load     =>  I_R_load,
304             R_load       => R,
305             V_gen        => Vg
306             );
307 END beh;
```

Parte II.

Assertion Based Verification per la simulazione di circuiti Mixed Signal

9. Elementi di teoria

9.1. Mixed-Signal Verification

Il percorso di sviluppo di un circuito integrato parte dalla definizione delle specifiche del progetto, le quali possono derivare dall'applicazione finale, dal settore commerciale e dalle normative.

Parallelamente all'ideazione ed alla successiva implementazione del circuito, una parte del team di progettazione viene impiegata nello sviluppo della fase di verifica del prodotto. In questo ramo si pianificano e realizzano una serie di test al fine di garantire che le specifiche di progetto vengano rispettate nel prodotto finale.

La fase di verifica inizia con la scrittura di un programma o piano di verifica – maggiormente noto con i termini inglesi verification plan –, esso contiene gli obiettivi di progetto, espressi in termini misurabili, e definisce la strategia da adottare nella realizzazione delle prove che andranno a verificare il dispositivo, prima ancora di scendere nel dettaglio dell'implementazione.

Inoltre nel verification plan si delineano i termini per la valutazione di quanto il dispositivo venga controllato nella fase di verifica. Si identifica questo aspetto con l'espressione coverage metric – metrica della copertura –, esso viene utilizzato per determinare il completamento della verifica e per calcolarne il progresso durante la progettazione.

Il processo di verifica differisce significativamente tra dominio analogico e quello digitale, presentando diversi aspetti e considerazioni. In aggiunta osservando l'attuale tendenza ad integrare i due domini, e quindi parlando di ambito Mixed-Signal (MS), la complessità aumenta notevolmente.

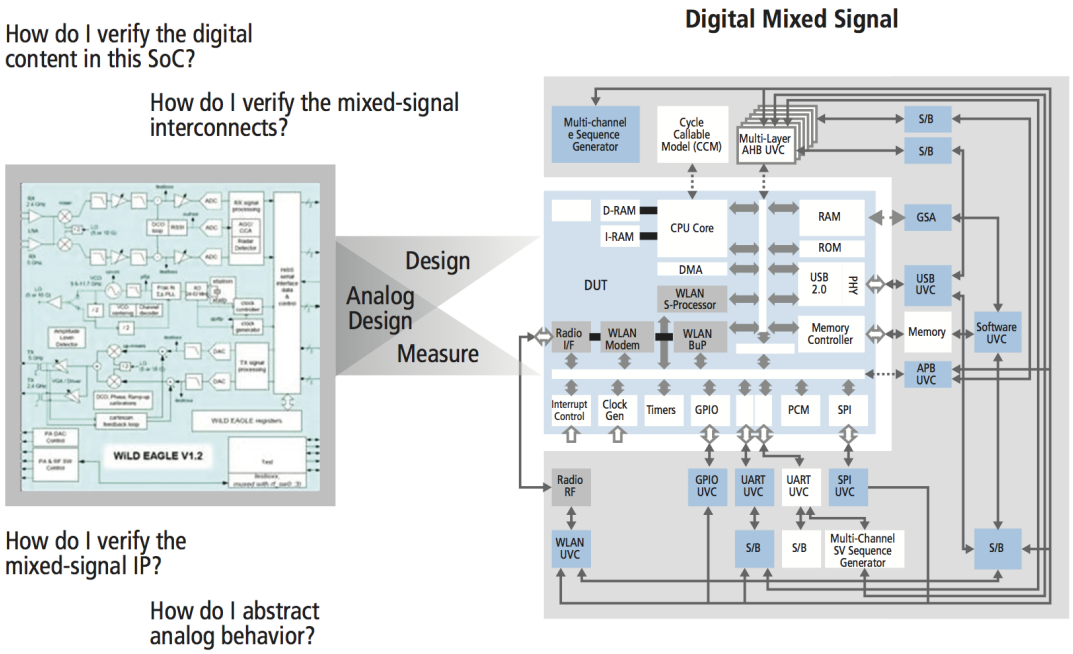

Figura 9.1.: Verifica in ambito Mixed-Signal [2]

Attualmente nel campo digitale sono presenti delle metodologie standardizzate per la pianificazione ed esecuzione della verifica. Tra le varie funzioni messe a disposizione sono comprese la generazione pseudo-casuale degli stimoli, l'automazione dei testbench, le asserzioni e gli strumenti per la valutazione della coverage.

Queste tecniche di verifica, come ad esempio l'Universal Verification Methodology (UVM), permettono agli ingegneri di determinare abbastanza accuratamente il livello di verifica e di potenziarlo fino a raggiungere l'obiettivo di coverage richiesto.

Per quanto riguarda, invece, l'ambito analogico, il processo di verifica è tradizionalmente costruito ad-hoc e viene realizzato tramite test diretti eseguiti su analisi del punto di lavoro, dei transitori, della risposta in frequenza, sweep, corner e Monte Carlo. Diversi simulatori analogici forniscono i dispositivi di verifica di basso livello, ma i supporti esistenti scarseggiano per un'adeguata pianificazione della verifica o una valida stima della coverage.

Durante il Design Automation Conference (DAC) tenutosi nel giugno 2011, si è discusso riguardo la necessità che la progettazione analogica e la sua verifica debbano diventare più affini alla progettazione digitale, più strutturate e più top-down – la concezione top-down prevede la descrizione iniziale della finalità principale senza scendere nel dettaglio delle

sue parti, le quali si andranno a rifinire in seguito; essa è una metodologia consolidata in ambito digitale.

Da quanto appena affermato si può concludere che l'ambiente della progettazione analogica abbia bisogno di strumenti di pianificazione della verifica come la Metric-Driven Verification (MDV) o la Universal Verification Methodology (UVM) e di metodologie di debug simili all'Assertion-Based Verification (ABV). In definitiva, si riscontra l'esigenza di introdurre testbench con controlli automatici anche per l'ambito analogico e Mixed-Signal.

Inoltre, una buona pianificazione della verifica del dispositivo permette di stabilire quali aspetti delle specifiche del progetto MS considerato, possano essere verificati nei livelli di astrazione più elevati e quali invece debbano necessariamente essere controllati a livello di transistor.

In questo modo gli ingegneri possono gestire i problemi derivanti dalla vasta quantità di dati richiesti dalle simulazioni per la verifica, e conseguentemente dall'elevata quantità di tempo necessario per generarli, rivolgendosi a tecniche di modellizzazione per aumentare la velocità di simulazione delle componenti analogiche. Queste includono l'utilizzo di Verilog-A, Verilog-AMS, e/o sono basate su tecniche di simulazione ad eventi come il Real Number Modeling (RNM).

Quindi, il partizionamento del progetto e la programmazione delle simulazioni risultano essere stadi fondamentali nella pianificazione della verifica. Oltre a ciò, nella stesura del piano di simulazione occorre prestare particolare attenzione ai test di regressione, poiché i progetti raramente presentano delle specifiche di sistema stabili nella parte iniziale del progetto.

9.2. Universal Verification Methodology

L'Universl Verification Methodology (UVM) rappresenta un procedimento che raggruppa le migliori applicazioni per una verifica efficiente ed esaustiva dei sistemi digitali.

Sotto lo stesso nome si presenta un libreria di classi appositamente realizzate per l'automazione ed il riutilizzo del lavoro nel processo di verifica.

Uno dei principi alla base dell'UVM è l'applicazione del flusso Metric-Driven Verification (MDV) che a sua volta si fonda sulla valutazione della coverage.

A questo proposito si vuole accennare che nei sistemi digitali si possono analizzare due tipologie di coverage.

La prima conosciuta col nome di 'code coverage' considera se nei test si è utilizzato ogni elemento del codice (assegnazioni, espressioni, stati, blocchi procedurali, etc) che descrive il circuito; il principale svantaggio di questo sistema è il fatto che non tiene in considerazione le intenzioni del progetto.

La seconda tipologia invece, chiamata 'functional coverage', prende come riferimento l'insieme delle specifiche del progetto e misura quanto il circuito rispecchia l'intento definito dal progettista.

9.2.1. Flusso MDV

Come già accennato il primo passaggio prevede la pianificazione delle attività di verifica e la redazione di un verification plan.

Dopodiché si passa alla creazione dei testbench, i quali devono presentare le seguenti caratteristiche comuni: generazione pseudo-casuale degli stimoli e disposizione di elementi per il monitoraggio dei segnali. Questi ultimi devono sia misurare la coverage ma anche controllare ed identificare comportamenti indesiderati.

In funzione del verification plan si procede alla disposizione dei riferimenti per la coverage, ed infine si eseguono le simulazioni.

Ciclicamente si analizza la coverage risultante dalle simulazioni e nell'eventualità si regolano i vincoli del verification plan e/o si aggiungono nuovi elementi di controllo nei testbench per sopperire le mancanze fino a raggiungere il grado di copertura auspicato.

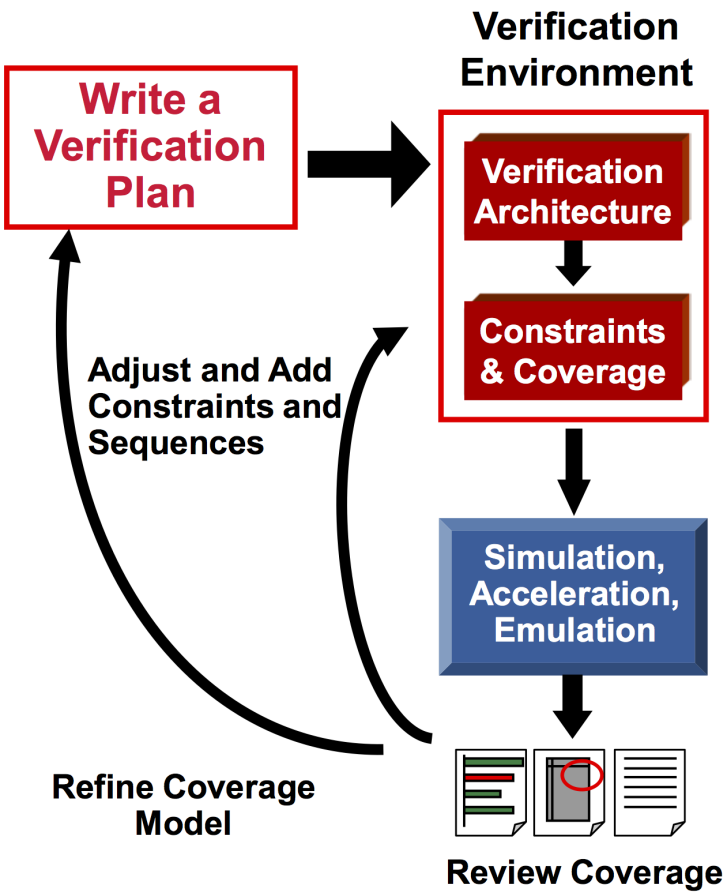

Figura 9.2.: Rappresentazione schematica del flusso MDV [14]

In questo flusso di verifica i test si devono prestare ad un'ampia possibilità di riutilizzo, e per conseguire ciò vengono scritti in modo indipendente dalla gerarchia – ovvero, si rendono i test riutilizzabili dal singolo blocco al sistema completo – e quindi, si risparmia la necessità di ridefinire la topologia per ogni nuova verifica.

Infine, si cerca di eseguire il maggior numero di simulazioni di test compilando una sola volta, ed evitando in questo modo la rielaborazione dell'ambiente di simulazione.

Uno degli obiettivi dell'UVM è quello di fornire dei test realizzati in maniera semplice per velocizzarne la lettura ed agevolarne la comprensione da parte degli utenti. Inoltre, in questo modo si riduce il background di conoscenze necessarie a creare i testbench basati su tali prove.

Così facendo si procura ai team di verifica un'interfaccia di base, facile e intuitiva per la scrittura dei test, offrendo comunque delle indicazioni orientative per cogliere eventuali casi limite e/o particolari.

Queste metodologie dunque, consentono di accellerare il processo di verifica, andando a ridurre la necessità di ricreare ogniqualvolta centinaia di test. La libreria di sviluppo UVM fornisce le classi di base e i meccanismi automatici per generare velocemente i testbench.

Inoltre, una delle qualità maggiormente apprezzata della metodologia UVM è la sua capacità di riutilizzo di queste componenti: su più livelli di astrazione dello stesso progetto – dal blocco funzionale al RTL – (riutilizzo verticale), o da un progetto al seguente (riutilizzo orizzontale).

9.3. Mixed Signal - MDV

Come accennato precedentemente, per via dell'andamento incrementale della complessità nella verifica dei circuiti MS, si ricorre ad un numero crescente di cicli di simulazione.

Per questo motivo si sta studiando come realizzare l'estensione del MDV nell'ambito Mixed-Signal (MS-MDV), e quindi di permettere al progettista di automatizzare il debug delle simulazioni e la verifica del successo o fallimento su tutte le prove applicate. La figura 9.3 mostra una rappresentazione schematica di alcuni esempi di controlli automatici che si vogliono implementare.

In aggiunta questo approccio presenta una maggiore capacità nel condurre adeguatamente il flusso di verifica nella catalogazione dei fallimenti e nella compilazione dei report.

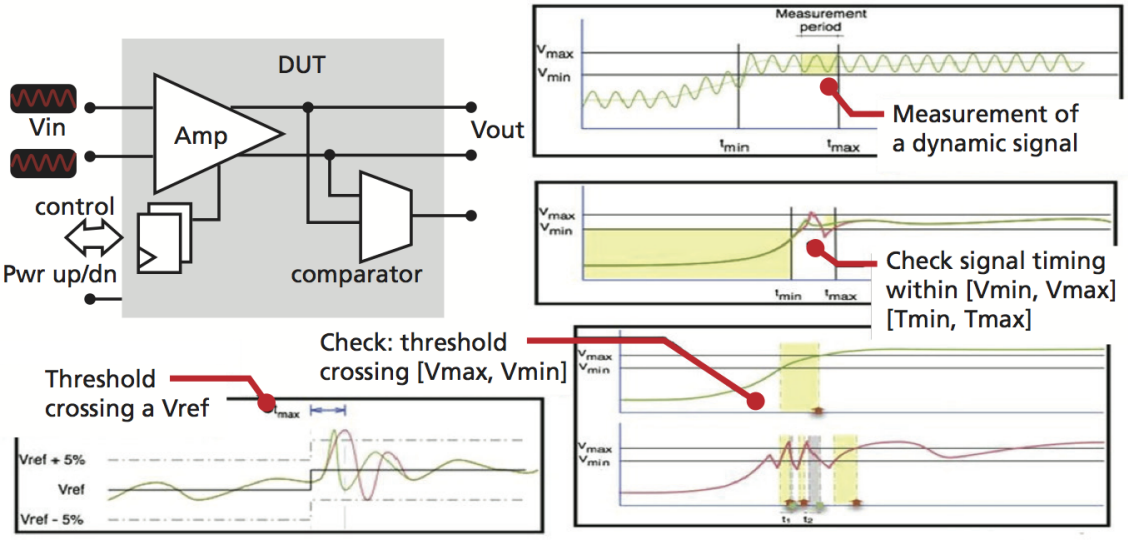

Figura 9.3.: Controlli automatici sui segnali analogici [2]

In definitiva l'obiettivo del metodo MS-MDV è quello di aumentare la capacità di verifica funzionale del SoC Mixed-Signal introducendo sistemi di ispezione nell'ambito analogico per misurare la copertura della verifica. Per ottenere un tale esito la prima operazione da svolgere considera di misurare nel tempo i valori della tensione, della corrente, della frequenza e del guadagno. Ed il secondo passo riguarda la scrittura di asserzioni che includono electrical e real number.

Inoltre ci si può eventualmente servire del RNM per i blocchi analogici, ed usufruire della verifica basata sull'UVM nei progetti MS attuali.

Un ambiente che supporta MS-MDV (Figura 9.4) permette di eseguire i modelli RNM in un simulatore esclusivamente digitale. Dunque gli utenti possono effettuare la verifica funzionale dell'intero circuito e delle sue interconnessioni con risolutori digitali per la simulazione. Quando si rende necessario un maggior grado di precisione, gli utenti possono comunque eseguire la simulazione a livello di transistor, o i modelli comportamentali analogici nello stesso ambiente.

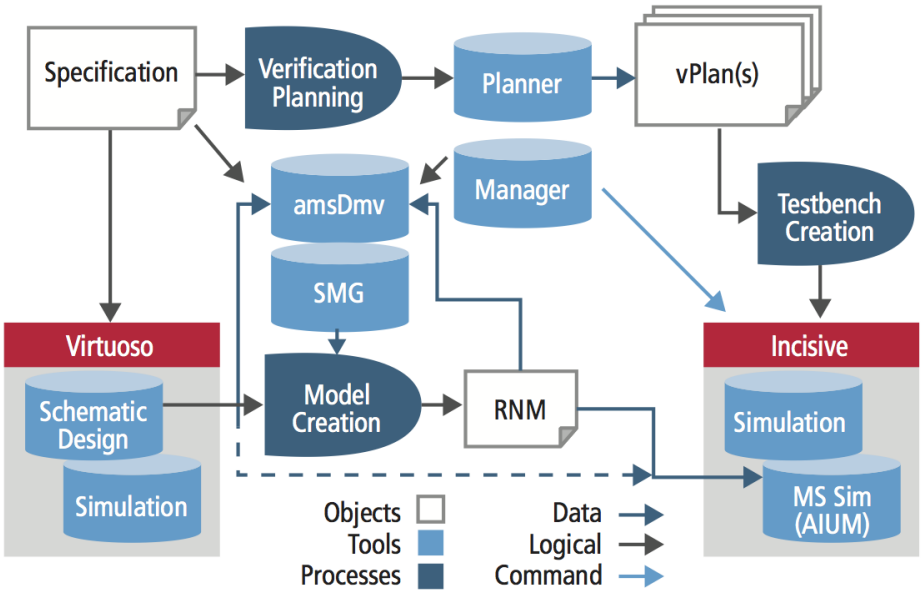

Figura 9.4.: Ambiente di simulazione MS-MDV [2]

9.4. Assertion Based Verification

Si passa ora ad introdurre uno dei possibili procedimenti atti ad applicare le teorie presentate nei paragrafi precedenti, l'Assertion Based Verification (ABV).

L'Assertion Based Verification è una metodologia che indipendentemente dagli strumenti di supporto e dai linguaggi utilizzati, mira a rendere le asserzioni parte integrante del progetto e del processo di verifica.

Le tecniche dell'ABV sono state sviluppate prevalentemente per l'ambito digitale, dove sono state impiegate con successo, e solo recentemente le relative organizzazioni responsabili degli standard hanno cominciato a sviluppare le estensioni per il Property Specification Language (PSL) e il SystemVerilog Assertion (SVA) – che attualmente risultano essere i principali linguaggi di sviluppo dell'ABV – da utilizzare nell'ambito MS.

All'interno di questo metodo si fa largo uso di alcuni termini tecnici di cui riportiamo le definizioni tradotte da il Language Reference Manual (IEEE 1850-2005) del PSL:

Proprietà È una raccolta di relazioni logiche e temporali tra due o più espressioni booleane concorrenti, espressioni sequenziali, o altre proprietà che, nel loro insieme, rappresentano un insieme di comportamenti.
Una proprietà viene valutata attraverso un booleano TRUE o FALSE, ma di per sé è completamente passiva.

Asserzione È la dichiarazione che una data proprietà debba necessariamente essere mantenuta vera ed una direttiva rivolta agli strumenti di test affinché controllino che essa venga verificata.

Coverage È la misura dell'occorrenza di determinate proprietà durante la valutazione della funzionalità (tipicamente dinamica) e quindi un criterio della completezza del processo di verifica della funzionalità.

L'adozione della tecnica ABV presenta alcuni svantaggi, a partire dal fatto che aggiungono codice non sintetizzabile ai file del progetto, e le asserzioni rendono il codice RTL più difficile da leggere ai progettisti. In generale rendono più gravosa la manutenzione del codice, e perdere leggibilità aumenta la probabilità di commettere errori.

Si deve anche tenere conto che le valutazioni delle asserzioni rappresentano per l'ambiente di sviluppo dei calcoli aggiuntivi che rallentano le simulazioni.

Inoltre, le asserzioni potrebbero accidentalmente interagire con i segnali sotto osservazione e modificare la funzionalità del progetto. Quest'aspetto rappresenta un'eventualità sufficientemente remota, ma è da tenere in considerazione e comporta necessariamente che anche il codice per la verifica venga controllato.

Considerando quanto detto finora risulta evidente che lo scrivere e debuggare le asserzioni comporta l'impiego di ulteriore tempo rispetto a quanto strettamente necessario per lo sviluppo del progetto, ed eventualmente questo può causare ritardi nei tapeout.

Tuttavia nel corso del tempo si è più volte dimostrato che un'applicazione sistematica di queste tecniche permette di ridurre il tempo totale speso per ricerca degli errori funzionali poichè le asserzioni consentono di rilevarli vicino alla loro causa di origine. La figura 9.5 nella pagina seguente rende visivamente ed in maniera molto semplice il concetto appena espresso.

Le asserzioni consentono anche di aumentare l'osservabilità durante le simulazioni di un progetto, andando ad esaminare condizioni ed elementi, causa di errori, che in situazioni normali di funzionamento non si propagano alle uscite.
Per giunta esse forniscono delle valutazioni continue durante tutte le simulazioni, portando a scoprire eventuali difetti anche in modo casuale.

Grazie alle asserzioni è possibile formalizzare in un linguaggio di programmazione le specifiche del blocco realizzato, precedentemente espresse solo in un linguaggio naturale.
Questo porta al miglioramento dell'integrazione e del riutilizzo degli IP realizzando un protocollo di conformità che controlla il corretto comportamento, e previene che il modulo venga utilizzato in maniera impropria.

Per rendere chiaro quanto appena affermato si pensi, come esempio, di scrivere asserzioni riguardanti l'intervallo di frequenza ammissibile, o nel caso in cui il modulo presenti dei parametri quali valori non possono essere accettati.

In definitiva l'approccio di verifica ABV ha dimostrato nello sviluppo dei circuiti integrati digitali di aiutare gli ingegneri a migliorare la qualità della progettazione e di ridurre il time to market.

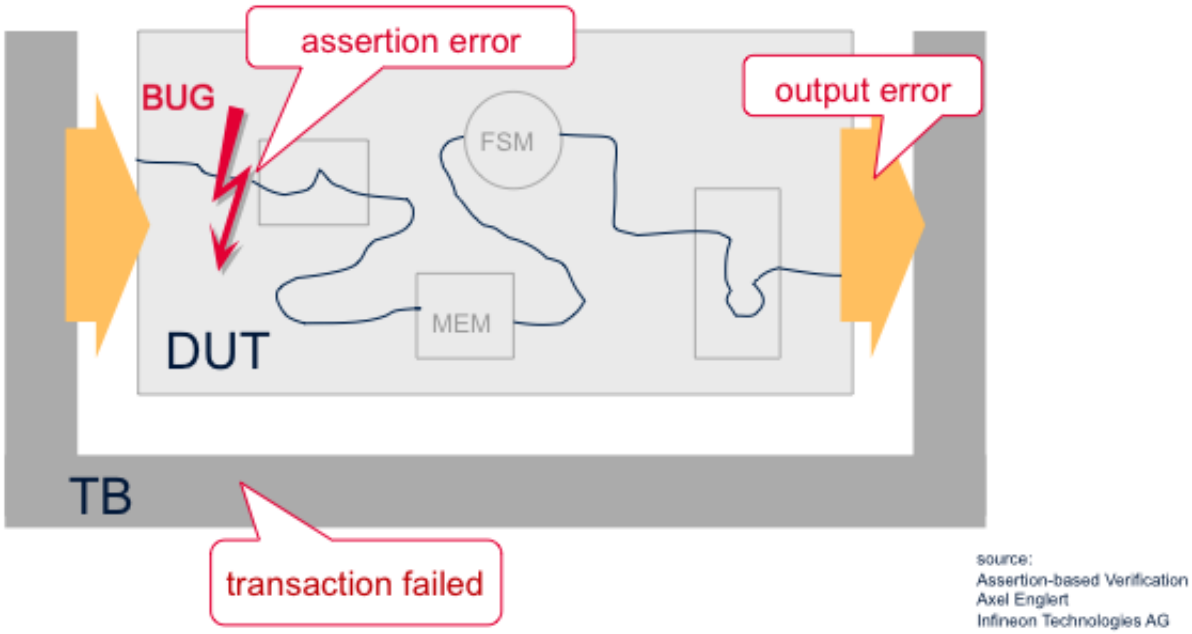

Figura 9.5.: L'asserzione rileva l'errore nella prossimità della causa di origine [15]

Le asserzioni vengono scritte sia durante la fase di progettazione che in fase di verifica del sistema, perciò sia progettisti che coloro che si occupano della verifica possono essere coinvolti nell'individuazione dei requisiti e nella traduzione in forma di asserzioni.

A questo proposito il progettista di un determinato blocco è particolarmente avvantaggiato:

- nell'individuare e scrivere le proprietà riguardanti le interfacce del blocco ed il resto del progetto (ad esempio prima, terza e quinta asserzioni in ordine da sinistra a destra in figura 9.6 nella pagina successiva)
- nel documentare sotto forma di asserzioni ulteriori ipotesi formulate sull'utilizzo del blocco
- nello scrivere asserzioni che vadano a verificare le previsioni sulle principali interazioni tra le gli elementi instanziati all'interno del blocco
- nello scrivere asserzioni che evitino l'incorrere in errori prevedibili derivanti dalla funzionalità nominale, dalle condizioni al contorno, dai comportamenti di avvio, o da altri elementi

- nella creazione di elementi per la coverage col fine di garantire che i casi limite e le aree più complesse del progetto vadano verificate adeguatamente.

L'ingegnere che si occupa della verifica è invece più portato a definire le asserzioni e i punti di coverage che derivano dalle specifiche globali sulla funzionalità del dispositivo.

Ad esempio, può andare a controllare se:

- Il dispositivo è sempre in una configurazione valida
- Il dispositivo e l'ambiente comunicano correttamente
- Il dispositivo risponde regolarmente agli ingressi (seconda e quarta asserzione in ordine da sinistra a destra in figura 9.6)

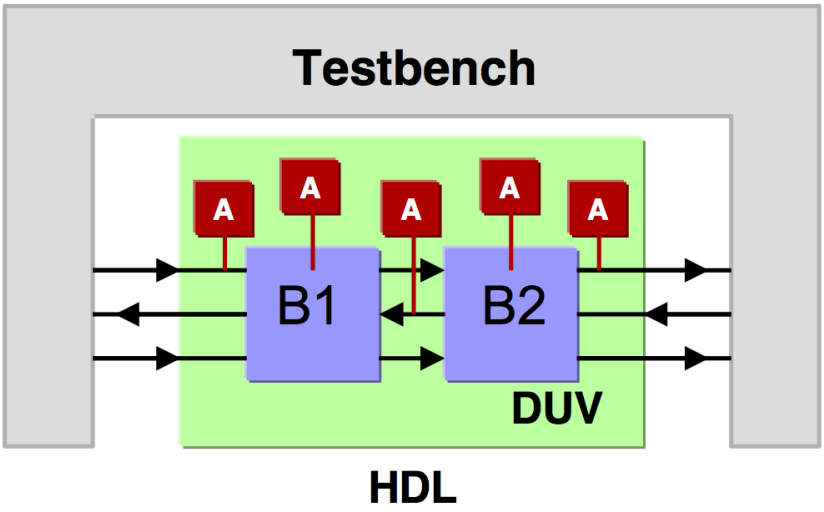

Figura 9.6.: Dispositivo in fase di verifica con asserzioni integrate [1]

10. SystemVerilog Assertions

10.1. Introduzione

Il SystemVerilog Assertion (SVA) è un sottoinsieme delle componenti per la verifica del linguaggio SystemVerilog, presentato brevemente a pagina 20.

Il SVA è, insieme al PSL, uno dei principali linguaggi per la scrittura delle asserzioni; le quali si possono semplicemente pensare come un controllo delle specifiche all'interno del progetto.

Questo concetto, alla base delle asserzioni, è possibile implementarlo con qualsiasi linguaggio di programmazione, ricorrendo ad esempio a strutture if...else... come nel seguente codice Verilog:

```
‘ifdef only_sim
  if (read & write)
      $display("Error: Mutually exclusive signals read
                        and write are both high\n");
‘endif
```

l'istruzione è stata inserita tra ‘ifdef e ‘endif per eventualmente escluderla durante la compilazione.

Perché allora passare al SVA?

Mentre nel Verilog alcuni tipi di controlli risultano facili esso presenta degli svantaggi tra cui il fatto che non esegue una buona valutazione delle tempistiche della dinamica, e rende difficile testare eventi paralleli nello stesso periodo.

Inoltre il Verilog è un linguaggio prolisso ed all'aumentare del numero delle asserzioni si complica la manutenzione del codice, e non ha al suo interno meccanismi per fornire la coverage funzionale.

Il SVA è un linguaggio dichiarativo che viene usato per specificare e verificare il comportamento del progetto. La natura dichiarativa lo rende particolarmente adatto a descrivere e controllare le relazioni temporali che intercorrono tra i segnali.

Il linguaggio è molto conciso e di più facile manutenzione, inoltre viene tenuto in considerazione solo dai simulatori ed ignorato automaticamente dagli strumenti di sintesi. Il SVA può essere utilizzato per fornire coverage funzionale ed eventualmente come input di strumenti per la Formal Verification.

10.2. Linguaggio SVA

10.2.1. Asserzioni

I principali elementi costitutivi di un'istruzione del SVA si possono riassumere in:

verification-directive (property) actions

La direttiva stabilisce l'azione da intraprendere per la proprietà che istanzia. La proprietà di per sé è completamente passiva e non viene mai considerata autonomamente.

Nel SVA sono presenti cinque differenti direttive:

assert Definisce la proprietà come un controllore, ovvero stabilisce che la proprietà debba risultare sempre verificata.

cover Monitora l'occorrenza della proprietà per la coverage.

assume Specifica la proprietà come un'assunzione per l'ambiente, ovvero assume che la proprietà venga verificata (in simulazione: assume = assert).

restrict Specifica la proprietà come un vincolo per la Formal Verification (non considerato in simulazione).

expect blocca l'esecuzione fino al completamento della proprietà.

Nel SystemVerilog sono presenti due tipologie di asserzioni: immediate (**assert**) e concorrenti (**assert property**).

10.2.1.1. Immediate Assertions

Le immediate assertions sono istruzioni procedurali e possono essere comparate all'istruzione di **if**.

La differenza si ha nel fatto che l'istruzione **if** non dichiara che un espressione deve essere vera, ma fa solo un controllo se è vera, ad esempio:

```
if (A == B) ...   // Controlla semplicemente se A è uguale a B
assert (A == B); // Asserisce che A è uguale a B, altrimenti
                 // viene generato un errore
```

Se la condizione espressa nell'asserzione risulta essere X, Z, o 0, l'asserzione fallisce ed il simulatore, di default, segnala un messaggio di errore.

Nell'esempio precedente non presenta altre istruzioni, ma anche le immediate assertions possono includere delle azioni da eseguire sia nel caso la valutazione abbia successo (pass) o che fallisca (fail). Ad esempio:

```
assert (A == B) $display ("OK. A è uguale B");

assert (A == B) $display ("OK. A è uguale B");
      else $error("L'asserzione è fallita");

assert (A == B) else $error("L'asserzione è fallita");
```

Se è presente l'istruzione di pass, essa viene eseguita immediatamente dopo la valutazione dell'espressione. L'istruzione di fail è associata ad un **else**, e si può utilizzare anche omettendo l'istruzione precedente.

Inoltre è possibile associare alle azioni un livello di gravità grazie ai system task:

$fatal segnala l'errore e termina la simulazione

$error segnala l'errore ma non interrompe la simulazione (opzione di default)

$warning segnala un warning

$info segnala un messaggio senza una gravità associata

```
ReadCheck: assert (data === correct_data)
                  else $error("Errore nella lettura dalla memoria");
Input10: assert (Input > 10)
                else $warning("Input è minore o uguale a 10");
```

Qualsiasi istruzione del SystemVerilog può essere inserita nella sezione actions delle asserzioni. Ad esempio si può stampare un messaggio di errore a console, pilotare una flag di errore, incrementare un contatore degli errori o segnalare un fallimento ad un'altra parte del testbench

```
A_B: assert (a == b)
          else begin
                 error_count++;
                 $error("A dovrebbe essere uguale a B");
               end
```

10.2.1.2. Concurrent Assertions

Da un punto di vista semantico ciò che distingue le concurrent assertions è la presenza della parola chiave **property** property che segue immediatamente l'**assert**.

Ad esempio

```
assert property ( !(read_en && write_en) );
```

asserisce che l'espressione read_en && write_en non deve mai risultare uguale a '1' durante la simulazione. Oppure:

```
assert property (@(posedge clock) Req |-> ##[1:2] Ack);
```

dove Req è una sequenza semplice (è giusto un espressione booleana) e ##[1:2] Ack è leggermente più complessa il cui significato è che il segnale Ack deve essere alto nel prossimo o successivo fronte di clock (o entrambi). E |-> è l'operatore d'implicazione.

Nel primo esempio, l'asserzione viene valutata ad ogni punto della simulazione, mentre nella seconda viene controllata solo sui fronti positivi del clock e tiene in considerazione più cicli.

Dunque rispetto alla immediate assertions le cui proprietà possono solo essere delle semplici espressioni booleane, nelle concurrent assertions si possono utilizzare sequenze che descrivono le relazioni temporali dei segnali.

Inoltre si può lavorare su segnali campionati sulla base degli eventi di un segnale di riferimento (tipicamente un clock) ed eventualmente condizionarne l'abilitazione.

In caso di asserzioni che considerano più cicli di campionamento, le valutazioni sono fatte in maniera sequenziale e quindi si possono presentare diverse code di valutazione

in parallelo. La figura 10.1 è presenta un semplice esempio per aiutare a capire questo punto.

A questo concetto bisogna prestare attenzione in quanto ritardi molto lunghi, lunghe ripetizioni, in definitiva lunghe sequenze possono incidere sulle performance di simulazione.

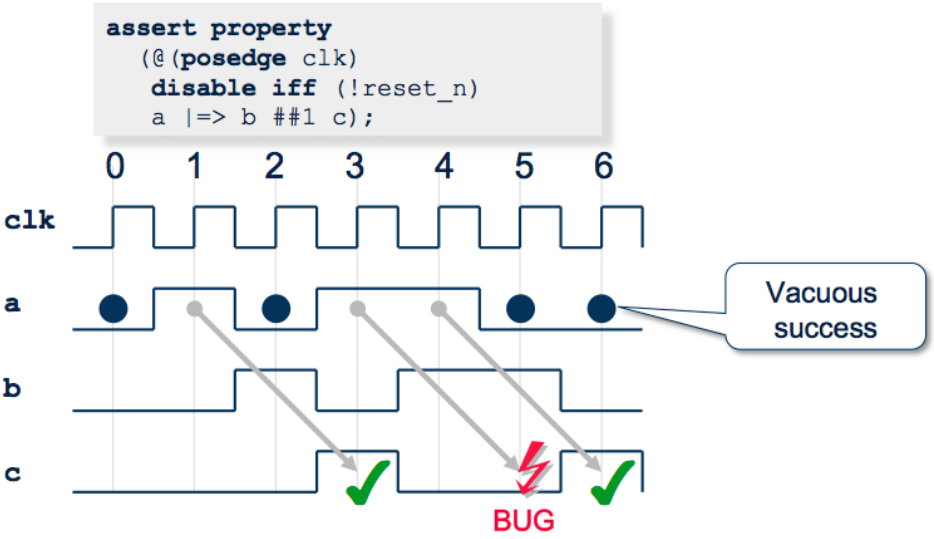

Figura 10.1.: Esempio di valutazione in maniera sequenziale [15]

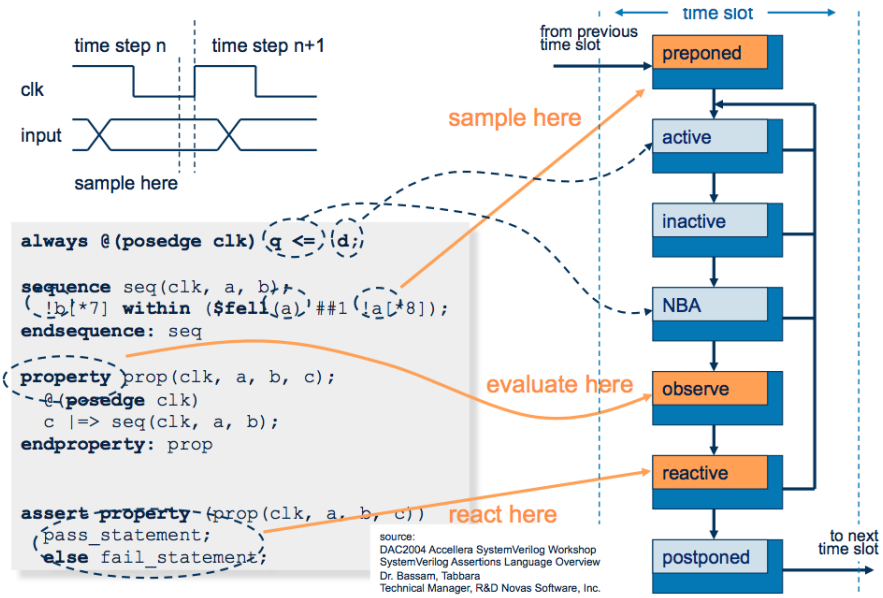

Figura 10.2.: Ciclo di campionamento e valutazione delle asserzioni nel SVA [15]

10.2.1.3. Operatore implicazione

Un'implicazione può essere utilizzata per creare una precondizione per la valutazione di una certa proprietà.

Un'implicazione consiste di una sequenza, un operatore di implicazione e un'ulteriore sequenza. La sequenza a sinistra dell'operatore è chiamata antecedente o condizione di abilitazione, mentre la sequenza a destra è chiamata conseguente o condizione di adempimento.

Se nel momento in cui viene valutata la proprietà che comprende l'implicazione c'è una corrispondenza dell'antecedente allora comincia la valutazione del conseguente. In questo caso l'implicazione ha successo, se la valutazione del conseguente restituisce un TRUE.
Se invece non c'è la corrispondenza dell'antecedente la valutazione ritornerà TRUE senza controllare il conseguente.

Ci sono due tipi di implicazione: l'implicazione sovrapposta il cui operatore è |−> , e quella non sovrapposta con l'operatore |=> .

```
s1 |-> s2;
s1 |=> s2;
```

Nel primo caso il conseguente incomincia ad essere valutato immediatamente conclusa la sequenza dell'antecedente, mentre nel secondo caso la valutazione del conseguente comincia sul fronte di clock successivo.
L'implicazione non sovrapposta è equivalente a:

```
s1 |-> ##1 s2;
```

10.2.1.4. Properties and Sequencies

Negli esempi visti finora, le proprietà asserite sono dichiarate e specificate nell'istruzione di asserzione. Ma le proprietà e le sequenze che le compongono possono essere dichiarate separatamente, per esempio:

```
sequence request
  req;
endsequence
```

```
sequence acknowledge
  ##[1:2] ack;
endsequence

property property_1;
  @(posedge clock) request |-> acknowledge;
endproperty

assert property (property_1);
```

Una sequenza è una lista di espressione booleane in un ordine crescente di tempo. Essa corrisponde se le espressioni booleane coincidono nell'ordine temporale. Si noti che una sequenza non fallisce o ha successo ma solo corrisponde o meno.

L'operatore per eccellenza delle sequenze è il ritardo `##N` , dove N rappresenta il numero di eventi di campionamento che intercorrono tra la sequenza che sta a sinistra dell'operatore con quella che sta a destra.

N può essere zero (caso particolare usato per unire le sequenze) o un numero naturale superiore o anche un intervallo chiuso o aperto; esso deve essere definito al momento della compilazione perciò può essere una costante o un parametro, e non può essere un intervallo di tempo – per intendere non può essere `5ns` .

Alcuni esempi, in cui se consideriamo a e b dei segnali logici affinché la sequenza totale corrisponda:

```
a ##1 b        // b deve essere '1' un ciclo di clock dopo che a
               // è passato a '1'

a ##[1:4] b    // b può essere a '1' nell'intervallo compreso da
               // uno a quattro cicli di clock dopo che a è
               // passato a '1'

a ##[1:$] b    // intervallo aperto
```

L'operatore $*$ è usato per specificare una ripetizione consecutiva dell'elemento a sinistra dell'operatore.

```
a ##1 b [*3] ##1 c      // Equivalente: a ##1 b ##1 b ##1 b ##1 c

(a ##2 b) [*2]          // Equivalente: (a ##2 b ##1 a ##2 b)

(a ##2 b)[*1:3]         // Equivalente: (a ##2 b)
                        // o (a ##2 b ##1 a ##2 b)
                        // o (a ##2 b ##1 a ##2 b ##1 a ##2 b)
```

Ci sono molti altri operatori per le sequenze tra cui quelli per specificare una sequenza o una ripetizione non consecutiva ([−> , [= , ...) e gli operatori per combinare le sequenze (**and**, **or**, intersect , first_match, ...) ma non è pretesa di questa breve introduzione al linguaggio di essere esaustiva e si rimanda a corsi specifici.

10.2.1.5. Clock nelle asserzioni

Le asserzioni concorrenti utilizzano gli eventi di un segnale per valutare le proprietà, esso può essere un singolo segnale clk , un gated clock ((clk && GatingSig)) o altre espressioni più complesse.

Tipicamente si sceglie un fronte di un clock per avere un passo temporale fisso per la valutazione dei segnali. Il clock per un proprietà può essere specificato in differenti modi:

- specificandolo esplicitamente in una sequenza:

```
sequence s;
  @(posedge clk) a ##1 b;
endsequence
property p;
  a |-> s;
endproperty
assert property (p);
```

- Specificandolo esplicitamente nella proprietà:

```
property p;
  @(posedge clk) a ##1 b;
endproperty
assert property (p);
```

- Specificandolo direttamente nell'asserzione:

```
assert property (@(posedge clk) a ##1 b);
```

- Inserendo l'asserzione in un blocco procedurale:

```
property p;
  a ##1 b;
endproperty
always @(posedge clk) assert property (p);
```

10.2.1.6. Reset asincrono

L'istruzione **disable iff** permette di specificare un reset asincrono per le asserzioni, come in esempio:

```
property p1;
  @(posedge clk)
              disable iff (Reset) a ##1 b;
endproperty
assert property (p1);
```

10.2.1.7. Assertion System Functions

Il SystemVerilog fornisce delle funzioni di sistema da utilizzare all'interno delle asserzioni. **$rose**, **$fell** ,**$changed** e **$stable** indicano se il valore dell'espressione ha cambiato o meno valore tra due eventi del clock adiacenti. Per esempio

```
assert property
  (@(posedge clk) $rose(in) |=> detect);
```

si asserisce che se in cambia da 0 a 1 tra un fronte di clock positivo ed il successivo, detect deve essere 1 sul fronte di clock successivo. In questa asserzione,

```
assert property
  (@(posedge clk) enable == 0 |=> $stable(data));
```

lo stato del segnale data non deve cambiare mentre enable è uguale a 0.

La funzione **$past** ritorna il valore di un'espressione nel ciclo di clock precedente. Per esempio l'asserzione:

```
assert property
  (@(posedge clk) disable iff (reset)
                  enable |=> q == $past(q+1));
```

controlla che q incrementi, mentre reset reset è basso ed enable enable alto.

Si noti che l'argomento di **$past** può essere un espressione, come mostrato sopra.

10.2.2. Istruzione di Bind

Come si è detto il codice del SVA si può includere nel codice sorgente dei moduli in cui si applicano. Alternativamente il SVA fornisce l'istruzione **bind**, la quale consente di scrivere le asserzioni in un modulo separato.

```
// modulo contenente il progetto da testare
module designModule (da, db, dclk);
  input  logic da, dclk;
  output logic db;
  // segnali interni
  logic  rda, rdb;
//  [...]
    always @(posedge dclk) db  <= da;
    always @(posedge dclk) rdb <= rda;
//  [...]
endmodule

// modulo contenente le asserzioni
module propertyModule (pa, pb, pclk);
 input pa, pb, pclk;
  property property_example1;
    pa |-> pb;
  endproperty
```

```
assert_example:
  assert property (
                   @(posedge pclk)
                     (property_example1)
                   )
          else $error("L'asserzione è fallita ");
endmodule

// modulo che lega il propertyModule con il designModule
module test_bindProperty;

  // l'istruzione di bind necessita del nome del modulo o
  // dell'istanza contenente il codice del progetto, il nome
  // del modulo delle asserzioni e del nome dell'istanza di bind
  // Negli esempi dpM_n è il nome dell'istanza di bind.
  // Per la compilazione è inoltre necessario che ogni
  // porta dell'interfaccia del modulo delle asserzioni
  // siano connesse ad un segnale del modulo del progetto
  //
  // L'istruzione sotto lega le asserzioni ad ogni istanza
  // del designModule all'interno del sistema
  bind designModule propertyModule dpM_1 (.pa(da), .pb(db), .pclk(dclk) );
  // mentre questa contiene il nome di una specifica istanza (dM)
  // ed il modulo delle asserzioni sarà legato solo ad essa
  bind dM propertyModule dpM_2 (.pa(da), .pb(db), .pclk(dclk) );
  // Nelle istruzioni precedenti si sono connesse le relative
  // interfacce dei moduli. Però l'istruzione di bind ha
  // piena visibilità all'interno del modulo connesso;
  // a questo proposito l'istruzione seguente mostra come
  // si possano anche collegare i segnali interni (rda ed rdb)
  bind dM propertyModule dpM_3 (.pa(rda), .pb(rdb), .pclk(dclk) );

endmodule
```

L'operazione di bind presenta alcuni vantaggi tra cui permettere:

- di mantenere il codice del progetto privo di costrutti non sintetizzabili
- il riutilizzo del codice delle asserzioni su più livelli di astrazione ed anche su altri progetti
- di rendere il codice delle asserzioni indipendente dal nome e dal percorso gerarchico dei segnali del progetto, facilitando la manutenzione
- di utilizzare l'SVA anche con altri linguaggi che non siano SystemVerilog come ad esempio il VHDL
- al team di verifica di lavorare contemporaneamente al progettista potendo scrivere su file diversi senza le restrizioni del sistema di gestione dei file
- di verificare le asserzioni separatamente

11. SVA asserzioni analogiche

Quando si parla di asserzioni analogiche, si sta parlando di essere in grado di monitorare i valori continui di segnali a tempo continuo.

I recenti miglioramenti del SystemVerilog consentono di utilizzare nelle asserzioni segnali a valori reali, ma ancora nel dominio del tempo discreto digitale.

11.1. analog_package

Nel momento in cui si è cercato di legare le asserzioni scritte in SVA con i modelli VHDL realizzati si è riscontrato che l'ambiente di simulazione Incisive di Cadence non consente la comunicazione diretta tra i segnali dichiarati col tipo di dati real tra moduli VHDL ed i moduli SystemVerilog.

Tuttavia al suo interno presenta dei sistemi di conversione per il tipo di dato wreal del VerilogAMS sia per il real del VHDL che per il real del SystemVerilog.

Come prima soluzione allora si è pensato di realizzare degli involucri (wrapper) in VerilogAMS per i moduli VHDL in modo da forzare le conversioni in wreal – procedimento comunque necessario nel caso si voglia utilizzare un modello VHDL all'interno di una simulazione mista con schematici SPICE – con questa soluzione però si perde la visibilità dei segnali interni in quanto, nel wrapper si considerano solo i segnali delle interfacce dei blocchi.

Nel cercare un soluzione alternativa si è venuti a conoscenza che Cadence mette a disposizione un package di nome cds_rnm_pkg che al suo interno definisce alcuni tipi di dato che emulano il tipo di dato wreal per il SystemVerilog.

Alla fine si è deciso di creare un package che riporta il codice fornito da Cadence e di modificarlo minimamente per uniformare i nomi con l'analog package presentato nel capitolo 3 a pagina 39. Per utilizzarlo all'interno dei moduli in cui si dichiarano le asserzioni sarà sufficiente aggiungere ad inizio del file l'istruzione `import analog_package::*;`

Si riporta come ultima nota che l'attuale versione del SVA non consente l'utilizzo di UDT all'interno delle asserzioni, e per questo come si potrà vedere negli esempi nel paragrafo 11.3 a pagina 116 si rende necessario l'assegnazione a segnali temporanei dichiarati col tipo di dato real prima del loro utilizzo all'interno delle asserzioni.

```
/* cds_rnm_pkg - wreal equivalent SystemVerilog nettypes
 *
 * REVISION HISTORY:
 * Created: 9/4/12 aspratt
 *
 */

// changed name from cds_rnm_pkg
package analog_package;

   // Equivalent to VAMS wreal1drv
   nettype real wreal1driver with CDS_res_wreal1driver;
   // Equivalent to VAMS wreal4state
   nettype real wreal4state with CDS_res_wreal4state;
   // Equivalent to VAMS wrealmin
   nettype real wrealmin with CDS_res_wrealmin;
   // Equivalent to VAMS wrealmax
   nettype real wrealmax with CDS_res_wrealmax;
   // Equivalent to VAMS wrealsum
   nettype real wrealsum with CDS_res_wrealsum;
   // Equivalent to VAMS wrealavg
   nettype real wrealavg with CDS_res_wrealavg;

                    nettype wreal1driver analog_ut;
                    nettype wrealsum     current_ut;

endpackage : analog_package
```

11.2. Eventi di attivazione

Come accennato precedentemente le asserzioni analogiche hanno ancora bisogno di un segnale digitale per essere attivate. A questo proposito si sono individuate due soluzioni.

11.2.1. Utilizzo di un segnale di clock

Poichè si stanno considerando dispositivi MS, non è raro che i blocchi analogici che si vogliono verificare posseggano una logica di controllo digitale, e spesso in questa situazione si utilizza il segnale di clock di quest'ultima per campionare i segnali analogici sotto osservazione.

È comunque possibile utilizzare un segnale di clock anche in caso di segnali continui che non dipendono da un controllore digitale, e qui è a discrezione di colui che scrive le asserzioni scegliere una frequenza appropriata. Solitamente, si applica il criterio di Nyquist: la frequenza di campionamento è scelta per essere il doppio della frequenza maggiore che possono assumere i segnali del componente.

Chiaramente in questo modo si trascura qualsiasi instabilità inattesa tra i due fronti del clock e sfortunatamente, quando un segnale non si comporta come previsto (individuare queste eventualità è l'obiettivo della verifica), la frequenza massima teorica potrebbe non essere nota in anticipo.

Inoltre bisogna considerare che l'aumentare la frequenza di clock delle asserzioni può influire pesantemente sul tempo di simulazione.

11.2.2. Comparazione dei segnali con livelli di riferimento

Con questa soluzione si intende muovere al di fuori delle asserzioni tutte le valutazioni dei segnali analogici esprimendo i risultati come valori logici, in modo da poter valutare facilmente questi ultimi con le asserzioni.

Per farlo si realizzano delle soglie da poter confrontare con il segnale analogico da verificare, ed i segnali digitali rappresentano l'attraversamento o meno di questi livelli di riferimento.

Questo approccio risulta particolarmente adatto per costruire delle soglie di allerta, o per valutare proprietà statiche o quasi statiche (come ad esempio l'uscita di un regolatore di tensione).

Ma lo stesso principio può essere utilizzato per generare soglie dinamiche, che rappresentano i limiti del comportamento atteso del segnale considerato.

Se nella soluzione precedente si perde visibilità temporale, in questo caso si tralasciano tutti quei valori di ampiezza del segnale intermedi ai vari livelli scelti.

11.3. Esempi

L'asserzione nell'esempio seguente è stata pensata per essere inserita tra i controlli di un amplificatore.

Operativamente essa controlla che al variare della differenza delle tensione agli ingressi, l'uscita risponda rispettando le tolleranza di ampiezza e tempo previste.

Per questo ultimo aspetto la proprietà è legata ad un segnale di campionamento dei segnali (clk) e la sequenza viene valutata in un intervallo definibile dal parametro N.

Non viene presentata la dichiarazione dell'interfaccia del modulo che contiene l'asserzione, ma i segnali dell'amplificatore sono stati dichiarati col tipo di dati analog_ut e per questo si devono necessariamente assegnare ad un segnale di tipo real – il cast è implicito – , poiché all'interno della property non sarebbero considerati validi

```
real out_temp;
real v_in_diff;
real v_out_ideal;
real tol;

assign out_temp    = out;
assign v_in_diff   = (in_p - in_n);
assign v_out_ideal = ((in_p - in_n) * G_ideal);
assign tol         = 0.2;

// property
property prop_out_value;
  @(posedge clk)
  disable iff (reset)
  $changed(v_in_diff) |->
           ##[0:N]
           ((out_temp >= ((1-tol)*v_out_ideal)) &&
```

```
            (out_temp <= ((1+tol)*v_out_ideal)));
endproperty: prop_out_value
// assertion
assert property (prop_out_value);
```

Nell'esempio successivo viene considerato un segnale di ingresso del modulo la cui funzione è di fornire un riferimento fisso di tensione.

L'obiettivo dell'asserzione è controllare che durante il funzionamento del blocco – ovvero mentre il segnale di reset non è uguale a 1 – il riferimento rimanga all'interno della massima tolleranza consentita.

Concretamente si è comparato il segnale di ingresso con le soglie di riferimento e tradotto l'informazione in un segnale logico il quale si è poi utilizzato all'interno dell'asserzione.

```
      logic  vref_2v_value_flag;
      real   vref_2v_value_tol;
const real   vref_2v_value_ref = 2.0;       // 2 [V]

assign vref_2v_value_tol = 0.2;
// comparison
always_comb
begin
  if ((vref_2v < (1-vref_2v_value_tol)*vref_2v_value_ref) ||
      (vref_2v > (1+vref_2v_value_tol)*vref_2v_value_ref))
              vref_2v_value_flag = 1;
    else      vref_2v_value_flag = 0;
end
// property
property prop_vref_2v_value;
  disable iff (reset)
    vref_2v_value_flag === 0;
endproperty: prop_vref_2v_value
// assertion
assert property ( prop_vref_2v_value );
```

Di seguito si riporta un asserzione pensata per la valutazione dell'uscita di un Convertitore Digitale-Analogico (Digital-to-Analog Converter – DAC).

Prima della property il segnale d'ingresso digitale adim_range_i viene convertito nel corrispettivo reale e moltiplicato per il LSB al fine di ottenere il valore ideale della tensione in uscita dal DAC.

Nella property si esamina che l'uscita effettiva del DAC set_dac_adim sia all'interno della tolleranza rappresentata in questo caso dall'intervallo dell'LSB.

La valutazione è anche in questo caso in un intervallo di eventi di clock definibile tramite il parametro DAC_vout_value_N, poichè il segnale clk di riferimento è appositamente a frequenza maggiore rispetto a quella di adim_range_i perchè il tempo di risposta che si vuole ottenere dal blocco è strettamente minore il periodo del segnale digitale.

```
        real set_dac_adim_temp;
        real DAC_vout_ideal;
const real DAC_LSB = 0.008;                 // 8 [mV]

assign set_dac_adim_temp = set_dac_adim;
assign DAC_vout_ideal    = ( real'(adim_range_i) * DAC_LSB);

// property
property prop_DAC_vout_value;
  @(posedge clk)
  disable iff (!pok)
    $changed(set_dac_adim_temp) |->
      ##[0:DAC_vout_value_N]
      ((set_dac_adim_temp >= (DAC_vout_ideal - (DAC_LSB/2))) &&
       (set_dac_adim_temp <= (DAC_vout_ideal + (DAC_LSB/2))));
endproperty: prop_DAC_vout_value

// assertion
assert property ( prop_DAC_vout_value );
```

12. Conclusioni

Come supportato da alcuni rapporti di introduzione dell'Assertion Based Verification in azienda, si è potuto constatare che l'adozione di questa tecnica richiede un elevato costo iniziale per via del tempo necessario allo studio del linguaggio e degli strumenti a disposizione per applicarla, soprattutto nel caso in cui non si presenti già una famigliarità con il SystemVerilog.

E per questo, con il tempo a disposizione si è riusciti solo ad impostare il lavoro di applicazione dell'ABV al dispositivo presentato nel capitolo 6 a pagina 61 .

Solitamente nei progetti l'investimento porta dei benefici aumentando le capacità della fase di verifica. Le quali possono essere il fornire dei controlli automatici utili per continuare a monitorare tutti gli aspetti del progetto durante le simulazioni di test o anche mettere a disposizione una formalizzazione delle specifiche del dispositivo utili nelle prove di regressione in caso di modifiche.

In particolare nel lavoro si è analizzato come applicare lo strumento tipicamente digitale delle asserzioni anche in ambito Mixed-Signal, e riuscire a monitorare i segnali analogici delle simulazioni dei dispositivi.

Si sono riscontrate ancora delle pesanti carenze nell'applicazione delle asserzioni analogiche, rispetto all'utilizzo in ambito digitale che presenta già degli standard e librerie di componenti riutilizzabili.

Ma considerando che è ancora campo di ricerca per le commissioni di sviluppo dei linguaggi – come il SystemVerilog o il VerilogAMS –, è quindi molto probabile che con i prossimi aggiornamenti dei manuali di riferimento si presentino delle funzioni di verifica analogiche che consentano di aumentare efficacia ed efficienza di questi controlli.

Durante l'applicazione si è potuto constatare che l'aspetto delle asserzioni che necessita la maggiore attenzione, è la difficoltà di definire proprietà che devono risultare essere in totale corrispondenza delle specifiche di progetto.

A. Acronimi

ABV	Assertion Based Verification
AMS	Analog Mixed-Signal
ASIC	Application Specific Integrated Circuit
CM E2R	Connect module Electrical to Real
CM R2E	Connect module Real to Electrical
DAC	Digital to Analog Converter
DAE	Differential Algebraic Equation
	Equazione differenziale Algebrica
DC	Direct Current
EDA	Eletronic Design Automation
ESD	Electrostatic discharge
HDL	Hardware Description Language
IEEE	Institute of Electrical and Electronics Engineers
LED	Light Emitting Diode
LRM	Language Reference Manual
MDV	Metric-Driven Verification
MS	Mixed-Signal
PMU	Power Management Unit
PSL	Property Specification Language
PWC	Piece-Wise Constant
PWL	Piece-Wise Linear
PWM	Pulse-Width Modulation
RNM	Real Number Modeling
RTL	Register Transfer Level
SoC	System on Chip
SPI	Serial Peripheral Interface
SPICE	Simulation Program with Integrated Circuit Emphasis
SV	SystemVerilog
SVA	SystemVerilog Assertions

UDR	User-Defined Resolution
UDT	User-Defined Types
UVM	Universal Verification Methodology
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

Bibliografia

[1] Chen J., et al., 2012. *Mixed-Signal Methodology Guide*. 1° ed. USA: Cadence Design Systems, Inc. (Lulu Enterprises, Inc.)

[2] Balasubramanian S., Hardee P.; Cadence Design Systems, 2013. *Solutions for Mixed-Signal SoC Verification Using Real Number Models*. Disponibile su
`http://www.cadence.com/rl/Resources/white_papers/Mixed_Signal_Verification_wp.pdf`
[Data di accesso 04/12/2014]

[3] Serge Garcia Sabiro; Mentor Graphics. 2013. *Event-Driven (RN) Modeling for AMS Circuits.* IEEE. Published in *Specification & Design Languages (FDL), 2013 Forum on.* ISSN:1636-9874.

[4] Datasheet del dispositivo: *4-Switch Synchronous DC/DC Controller with SPI Interface for High Power LED Lighting*
Documentazione aziendale interna: Infineon Technologies, AG.

[5] Bizjak L., et al., 2013. *VHDL Modeling and Coding guidelines for AMS systems*
Documentazione aziendale interna: Infineon Technologies, AG.

[6] Bizjak L., et al., 2013. *VHDL Modeling and Coding guidelines for AMS systems*
Documentazione aziendale interna: Infineon Technologies, AG.

[7] Schaffer B., 2008. *Modeling analog and mixed signal blocks with VHDL*
Documentazione aziendale interna: Infineon Technologies, AG.

[8] Cadence Design Systems, Inc., 2013. *Virtuoso VHDL Toolbox User Guide, Product Version 6.1.5*

[9] Marsili S., 2010. *Building time discrete System Models with Matlab/Simulink*
Documentazione aziendale interna: Infineon Technologies, AG.

[10] Capodivacca G., 2014. *Discrete time simulation for DC/DC*
Documentazione aziendale interna: Infineon Technologies, AG.

[11] Ashok B., 2014. *SystemVerilog Assertions and Functional Coverage – Guide to Language, Methodology and Applications*. 1° ed. New York: Springer

[12] Neviani A., 2012. *Materiale del Corso di Progettazione e Sintesi di Circuiti Digitali*
Dipartimento di Ingegneria dell'Informazione - Università di Padova.

[13] Srikanth Vijayaraghavan, Meyyappan Ramanathan, 2005. *A Practical Guide for SystemVerilog Assertions*. 1° ed. USA: Springer

[14] Cadence Design Systems, Inc., 2013. *UVM-SV Introduction RAK - UVM Overview*

[15] Zinn A., 2013. *Assertion-based Verification*
Documentazione aziendale interna: Infineon Technologies, AG.

[16] Axel Englert, 2008. *Assertion-based Verification (ABV Workshop SMARTi LU)*
Documentazione aziendale interna: Infineon Technologies, AG.

[17] Doulos: SystemVerilog Assertions Tutorial. Disponibile su
`https://www.doulos.com/knowhow/sysverilog/tutorial/assertions/`
[Data di accesso 04/12/2014]

[18] Cadence Design Systems, Inc., 2013. *SystemVerilog Real Number Modeling (SV-RNM) RAK*

[19] Regis Santonja; Freescale Semiconductor. 2012. *Re-usable continuous-time analog SVA assertions.*

Ringraziamenti

Desidero ringraziare tutti coloro che mi hanno aiutato nella stesura della presente tesi con suggerimenti, critiche ed osservazioni, anche se solo a me spetta la responsabilità per ogni errore in essa contenuto.

Vorrei esprimere la mia gratitudine al mio relatore professor Andrea Neviani per il supporto e la pazienza dimostratami nel lavoro di tesi portato avanti in questi mesi.

Stesso riconoscimento vorrei rivolgerlo a Infineon Tecnhologies, presso la quale ho svolto il progetto di cui tratta questa tesi: ringrazio sinceramente tutti i colleghi che mi hanno accompagnato in questa avventura.

In particolar modo vorrei esprimere la mia riconoscenza verso l'Ing. Andrea Scenini, mio tutor all'interno dell'azienda e correlatore per la tesi, per avermi supportato e sopportato durante lo svolgimento del tirocinio.

Un sentito grazie ai miei genitori, per il sostegno fornitomi; senza il loro incoraggiamento e la loro perseveranza, probabilmente, oggi non sarei qui.

Ringrazio mio fratello Lorenzo e mia sorella Maria per il loro costante aiuto e per tutti i preziosi consigli.

Vorrei inoltre ringraziare Marco Serpelloni e tutti i compagni di studio che mi hanno aiutato durante questo percorso accademico.

Infine, a tutti gli amici semplicemente grazie.