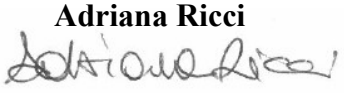
UNIVERSITÀ POLITECNICA DELLE MARCHE
SCUOLA DI DOTTORATO DI RICERCA IN SCIENZE DELL'INGEGNERIA
CURRICULUM IN INGEGNERIA BIOMEDICA, ELETTRONICA E
DELLE TELECOMUNICAZIONI

Gestione della complessità hardware nella metodologia di progettazione ad alto livello di sistemi multicore

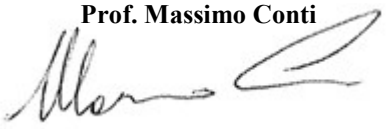
Tesi di Dottorato di:

Adriana Ricci



Tutor:

Prof. Massimo Conti



Coordinatore del Curriculum:

Prof. Francesco Piazza

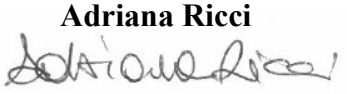
XXXI ciclo – A.A. 2017/2018
(XVII ciclo – nuova serie)

UNIVERSITÀ POLITECNICA DELLE MARCHE
SCUOLA DI DOTTORATO DI RICERCA IN SCIENZE DELL'INGEGNERIA
CURRICULUM IN INGEGNERIA BIOMEDICA, ELETTRONICA E
DELLE TELECOMUNICAZIONI

Hardware complexity management in high-level design methodology of multicore systems

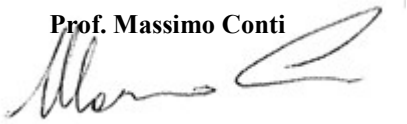
Ph.D. Dissertation of:

Adriana Ricci



Advisor:

Prof. Massimo Conti



Curriculum Supervisor:

Prof. Francesco Piazza

XXXI edition – A.A. 2017/2018
(XVII edition– new series)

Università Politecnica delle Marche
Scuola di Dottorato di Ricerca in Scienze dell'Ingegneria
Facoltà di Ingegneria
Via Brecce Bianche — 60131 - Ancona, Italy

Sintesi

Negli ultimi decenni, lo sviluppo della tecnologia di progettazione dei sistemi elettronici, affiancata dall'Electronic Design Automation (EDA), ha permesso la realizzazione di sistemi elettronici con complessità e performance crescenti.
Inoltre, in questi ultimi anni è possibile integrare un sistema elettronico complesso, equivalente a milioni di transistor, in un singolo chip di silicio: System-on-a-Chip (SoC). Il design di un SoC complesso non può essere gestito tramite i metodi tradizionali di progettazione ma necessita di metodologie di modellazione efficienti a livello di sistema e tool di sintesi ad alto livello (High Level Synthesis - HLS) affidabili, che generano un'architettura RTL ottimizzata a partire da specifiche funzionali ad alto livello.
Le ultime versioni di strumenti HLS hanno compiuto progressi significativi tali da poter pensare di inserire la metodologia HLS nel flusso di progettazione.
L'utilizzo di HLS permette di gestire sistemi complessi, pur garantendo alte performance, portando al complessivo aumento di produttività.

La ricerca proposta presenta progetti complessi come casi di studio sia per la modellazione sia per la sintesi ad alto livello, incluso il confronto delle soluzioni HLS rispetto al design sviluppato mediante metodologia di progettazione tradizionale. Sono state implementate diverse soluzioni con gradi di complessità crescenti, allo scopo di verificare i limiti della metodologia di progettazione a livello di sistema.
Infine, è stato implementato su FPGA un sistema funzionante che prevede l'integrazione di componenti progettati mediante metodologie diverse.
Parte della ricerca è stata effettuata in collaborazione con Korg Italy, con sede a Osimo (AN), che ha fornito know-how e consulenza per quanto riguarda il campo applicativo della sintesi digitale di segnali musicali.

Abstract

In recent decades, the development of electronic systems design technology, supported by Electronic Design Automation (EDA), has allowed the creation of electronic systems with increasing complexity and performance.
Moreover, in recent years it is possible to integrate a complex electronic system, equivalent to millions of transistors, into a single silicon chip: System-on-a-Chip (SoC). The design of a complex SoC can't be managed by traditional design methods but requires system-level efficient modeling methodologies and high-level synthesis tools (High Level Synthesis - HLS) that generate an optimized RTL architecture from high-level functional specifications. The latest versions of HLS tools have made significant advances that make it possible to consider including the HLS methodology in the design flow.
The use of HLS allows the management of complex systems, while ensuring high performance, leading to the increase in productivity.

The research presents complex projects as case studies for both modeling and high-level synthesis, including comparison of HLS solutions with respect to design using traditional methodology. Several solutions have implemented with increasing degrees of complexity, to verify the limits of the system-level design methodology.
Finally, a functioning system has been implemented on FPGA which provides for the integration of components designed using different methods.
Part of the research was carried out in collaboration with Korg Italy, based in Osimo (AN), which provided know-how and advice regarding the field of application of digital synthesis of musical signals.

Indice

Elenco delle Figure

Elenco delle Tabelle

Capitolo 1.

Introduzione

Negli ultimi trentacinque anni, la tecnologia di progettazione dei sistemi elettronici, e in particolare lo sviluppo dell'Electronic Design Automation (EDA), hanno avuto una crescita eccezionale, superata solo dai progressi nella fabbricazione di semiconduttori.
Una delle soluzioni comunemente accettate per ridurre il productivity gap, è quello di aumentare il livello di astrazione nel processo di progettazione. Al fine di aumentare la produttività e per colmare il divario semantico tra livelli di astrazione più alti e quelli più bassi d'implementazione, si tende ad automatizzare il più possibile il flusso di progettazione del sistema.
Nel momento in cui i problemi di progetto ai livelli più bassi di astrazione sono diventati umanamente ingestibili e dispendiosi in termini di tempo, sono stati sviluppati degli strumenti CAD (Computer-Aided Design) di supporto per la simulazione e la sintesi logica dei circuiti elettronici e introdotti nel processo di progettazione.
Tuttavia, i continui miglioramenti nella tecnologia del silicio e l'aumento della richiesta di applicazioni sempre più complesse ed eterogenee hanno costretto ricercatori e progettisti a concentrarsi su livelli di design più elevati rispetto alla logica.
Poiché i livelli più alti di astrazione riducono di un ordine di grandezza il numero di oggetti che il designer deve prendere in considerazione, è stato possibile progettare e realizzare circuiti integrati complessi per applicazioni specifiche in tempi più brevi.
Quindi, come è stato per la sintesi logica, la sintesi RTL (Register-Transfer Level) ha contribuito a innalzare i livelli di astrazione nella metodologia di progettazione.
Tuttavia, i linguaggi di descrizione dell'hardware (HDL – Hardware Description Language) vengono utilizzati per la progettazione di un sistema specifico (processore, interfaccia o componente di comunicazione) sviluppato ad hoc per un'applicazione. Questi sistemi, insieme ai processori standard e alle memorie, possono essere usati come componenti in soluzioni più complesse la cui metodologia di progettazione richiede livelli ancora più alti di astrazione: livello di sistema (Electronic System Level - ESL).
A livello di sistema, il design si concentra sulla specifica dei sistemi tramite la definizione di modelli comportamentali, utilizzando tipi di dato strutturati e canali di comunicazione; mentre i dettagli implementativi a livello di logica sono nascosti al designer e gestiti dai tool CAD che si occupano della sintesi.
Tuttavia, l'industria e il mondo accademico non si sono concentrati sufficientemente sullo sviluppo e la formalizzazione di una metodologia di progettazione a livello di sistema, anche se c'era una chiara necessità.
Inoltre, in questi ultimi anni, grazie agli sviluppi nel campo della tecnologia dei semiconduttori, è possibile integrare un sistema elettronico complesso, equivalente a milioni di transistor, in un singolo chip di silicio che viene definito System-on-a-Chip (SoC). Il design di un SoC complesso non può essere gestito tramite i metodi tradizionali di progettazione ma necessita di metodologie di modellazione efficienti a livello di sistema.

L'incremento continuo della complessità e le richieste sempre crescenti del mercato, che puntano a minimizzare il time-to-market e i costi, rendono inevitabile lo sviluppo di una metodologia di progettazione a livello di sistema ben chiara e definita e strumenti di sintesi ad alto livello (High Level Synthesis - HLS) affidabili che generano un'architettura RTL ottimizzata a partire da specifiche funzionali ad alto livello.
A causa degli scarsi risultati delle prime generazioni di sintesi commerciale ad alto livello (i primi sistemi HLS sono apparsi in letteratura negli '70), la metodologia ESL non è stata largamente usata fino ad oggi. Le ultime versioni di strumenti HLS hanno compiuto progressi significativi nel fornire un'ampia gamma di linguaggi accettati in ingresso e nella modellazione platform-based, specialmente per Field-Programmable Gate Array (FPGA). Di conseguenza, questo ha riaperto la sfida per lo sviluppo della metodologia ESL, che punta a essere inserita nel flusso di progettazione.
L'uso di HLS permette di creare diverse architetture RTL usando direttive al compilatore che vincolano la sintesi e l'implementazione, a partire dallo stesso codice sorgente in cui è definita la specifica funzionale del sistema ad alto livello. Questo consente una maggiore esplorazione del design space in modo più efficiente e rapido, anche per applicazioni che richiedono sistemi complessi, portando al complessivo aumento della produttività.

In questa ricerca sono stati utilizzati strumenti e piattaforme sviluppate da Xilinx come esempio per dimostrare l'efficacia della progettazione a livello di sistema per diversi domini applicativi. Vengono presentati progetti complessi come casi di studio sia per la modellazione sia per la sintesi C-to-FPGA, incluso il confronto delle soluzioni ESL rispetto al design mediante metodologia tradizionale HDL. Sono state implementate diverse soluzioni con gradi di complessità crescenti, allo scopo di verificare i limiti della metodologia di progettazione a livello di sistema.
Infine, è stato implementato su FPGA un sistema funzionante che prevede l'integrazione di componenti progettati mediante metodologie diverse.

1.1. Storia della sintesi ad alto livello: dagli anni 70 ad oggi

Questo paragrafo presenta la storia e l'evoluzione della sintesi ad alto livello, dalla ricerca accademica all'adozione in campo industriale. In particolare, vengono presentate le motivazioni per cui la metodologia ad alto livello, nonostante sia un argomento di cui si parla ormai da quasi quarant'anni, rappresenta ancora oggi una valida alternativa alla metodologia basata su sintesi RTL per colmare il productivity gap sempre in crescita.
Come mostrato in Fig. 1.1, nella storia delle metodologie di progettazione, l'aumento della produttività del design è sempre stato associato all'innalzamento del livello di astrazione nel flusso di progettazione.

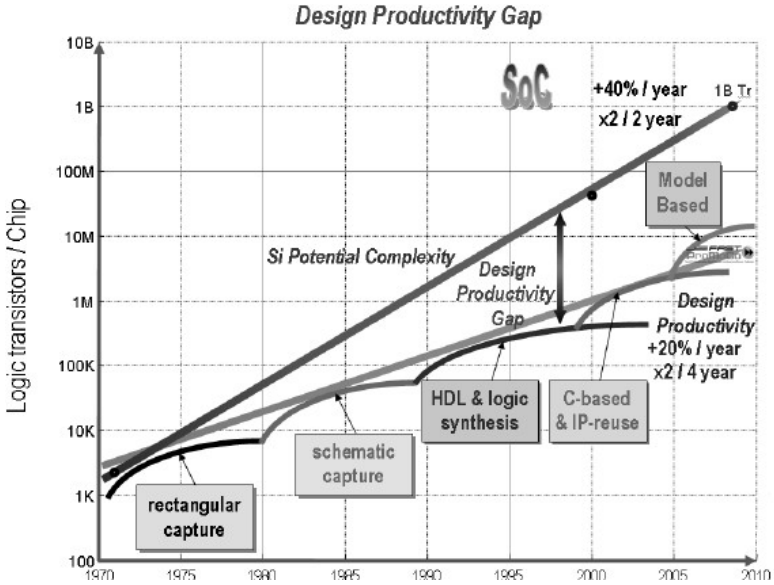


Fig. 1.1 - Design productivity gap

1.1.1. Preistoria dell'HLS (1970)

Questo periodo fu interessato da un precoce lavoro di ricerca nel campo della sintesi ad alto livello, in quanto i soli prodotti EDA in commercio ai tempi erano strumenti che eseguivano il layout fisico del circuito e la sintesi automatica era limitata al livello di transistor [1]. La ricerca è stata estremamente innovativa per il contesto tecnologico dell'epoca per cui ha avuto un piccolissimo impatto in campo industriale.
Nonostante la scarsa diffusione, questo periodo vede lo sviluppo di concetti fondamentali che saranno la base per l'evoluzione dell'HLS nelle future generazioni.
Alberto Sangiovanni-Vincentelli li classifica in cinque macro-aree [2]:

1. Simulazione circuitale: il grande successo della simulazione circuitale è stato la spinta principale per lo sviluppo successivo di strumenti EDA in campo industriale; nel 1975 all'interno dell'Università di Berkeley in California viene sviluppato Spice (Simulation Program for Integrated Circuits Emphasis), un programma di simulazione circuitale che oggi è divenuto lo standard per la simulazione dei circuiti elettronici analogici;
2. Simulazione logica e testing: molte compagnie si affidarono a questa tecnologia per il debug dei circuiti logici da loro progettati; gli innumerevoli contributi in questo campo hanno avuto un ruolo fondamentale nello sviluppo futuro di strumenti per la generazione automatica dei test pattern e la simulazione degli errori;
3. Simulazione timing: nel 1975, per la prima volta si introduce il concetto di approssimare il comportamento del circuito nel tempo a una soluzione di

equazioni differenziali ordinarie; inoltre, eseguire la simulazione nei periodi tempo in cui avvengono gli eventi di commutazione dei segnali permette di rendere la simulazione più veloce e attendibile; la corretta posizione nel tempo degli eventi è ancora oggi la base per eseguire velocemente la simulazione;
4. Wire routing: le principali tecniche usate oggi dai tool si basano sui risultati scientifici dell'epoca;
5. Regular arrays: viene sviluppato lo stile di progettazione tramite standard cells o gate array come alternativa al layout custom.

Di fatto il fallimento della preistorica generazione di EDA è dovuto principalmente al fatto che questi prodotti erano tarati esclusivamente su hardware customizzati e prevedevano l'uso di software complessi scritti in codice assembly. Quindi era molto difficile mantenere il passo con i progressi in campo tecnologico.
Questi fattori rendevano di fatto i tool obsoleti.

1.1.2. Generazione 1 (anni '80 - '90)

In questo periodo si ha una forte crescita nel campo EDA sotto vari aspetti.
È in questo particolare contesto tecnologico che, per rappresentare i circuiti digitali in modo più efficiente rispetto alle funzioni booleane, si sviluppano i linguaggi di descrizione dell'hardware (Hardware Description Languages – HDL), come ad esempio Verilog e VHDL (1987).
La mancata diffusione della generazione di strumenti di sintesi comportamentale dell'epoca è dovuta a vari fattori.
Innanzitutto, la necessità di colmare il productivity gap era già soddisfatta grazie all'adozione della sintesi RTL (Register Tranfer Level); di conseguenza, la sintesi comportamentale non era necessaria.
Inoltre, in quegli anni i designer stavano imparando a usare i linguaggi di descrizione dell'hardware, quindi l'idea di adottare altri nuovi linguaggi fu visto come un ostacolo alla progettazione.
Infine, anche la scarsa qualità dei risultati, di fatto, ha reso inaccettabile l'uso dei tool di sintesi comportamentale nel flusso di progettazione.

1.1.3. Generazione 2 (anni '90 - 2000)

Questo periodo coincide con l'esplosione del Web e delle sue applicazioni. Quindi, la ricerca si è indirizzata verso quel trend emergente con il conseguente calo di innovazione nel campo EDA.
I tool di sintesi usavano linguaggi di descrizione dell'hardware come linguaggi in ingresso; di conseguenza, il tempo di simulazione era circa lo stello della sintesi RTL.
Inoltre, spesso i risultati erano variabili e imprevedibili.
Quindi neanche questa generazione di tool di sintesi ad alto livello è riuscita ad espandersi nel mercato, non offrendo, di fatto, nessun vantaggio dalla sua adozione.

1.1.4. Generazione 3 (dal 2000 ad oggi)

Nel frattempo, i progressi nel settore tecnologico hanno continuato a seguire la legge di Moore, incrementando il grado di integrazione di componenti circuitali sullo stesso chip. Nascono così i cosiddetti System-on-Chip (SoC).
Date le alte potenzialità offerte dai SoC, le specifiche richiedono design sempre più complessi il cui progetto diventa ingestibile tramite metodi tradizionali, come la sintesi RTL.
Una strategia utilizzata è stata il design reuse cioè inglobare all'interno del sistema che si sta progettando dei componenti già precedentemente progettati e testati che prendono il nome di IP core (Intellectual Property core), e considerate come se fossero delle black box che svolgono una determinata funzionalità ma di cui non interessa come questa funzionalità è stata implementata internamente e va semplicemente collegata ai pin di ingresso e uscita. Le IP possono anche essere fornite da terze parti non necessariamente progettate dallo stesso team interno al progetto principale quindi questa tecnica riduce notevolmente il tempo di sviluppo complessivo. Tuttavia, questa strategia da sola non è sufficiente a colmare il productivity gap [3].
Dal 1998 la Semiconductor Industry Association (SIA) elabora l'International Technology Roadmap for Semiconductors (ITRS) [4], che predice i principali trend nell'industria dei semiconduttori quindici anni avanti nel futuro. L'ITRS è una valida guida per l'industria dei semiconduttori per seguire le tendenze tecnologiche e del mercato mondiale dei circuiti integrati (IC). Negli ultimi anni, i documenti ITRS sono diventati un riferimento comune per l'intera industria dei semiconduttori.
Come mostrato dalle Fig. 1.2 e 1.3 estratte dal ITRS executive summary del 2011 [5] e dalla Tabella 1.1 dell'ultimo ITRS executive report del 2015 [6], anche per i prossimi anni il trend tecnologico prevede lo scaling dimensionale per quanto riguarda la fabbricazione dei circuiti integrati. Mentre la tendenza per l'integrazione di sistemi eterogenei punta non più solo verso maggiori performance ma anche verso la riduzione del consumo di potenza.
Vista la tendenza, la ricerca si è focalizzata maggiormente sulla metodologia di progettazione ad alto livello. Sono stati prodotti strumenti di sintesi che accettano in ingresso linguaggi più famigliari ai progettisti (varianti del C e del MATLAB) che ottengono risultati di qualità superiore rispetto alle passate generazioni.
Le ultime versioni di strumenti HLS hanno compiuto progressi significativi tali da poter pensare di inserire la metodologia HLS nel flusso di progettazione.
Inoltre, questo periodo vede l'ascesa delle Field Programmable Gate Array (FPGA), circuiti integrati standard le cui funzionalità possono essere programmate direttamente dall'utente tramite HDL. Usare la sintesi ad alto livello basata su FPGA permette di mappare velocemente l'algoritmo su un hardware.

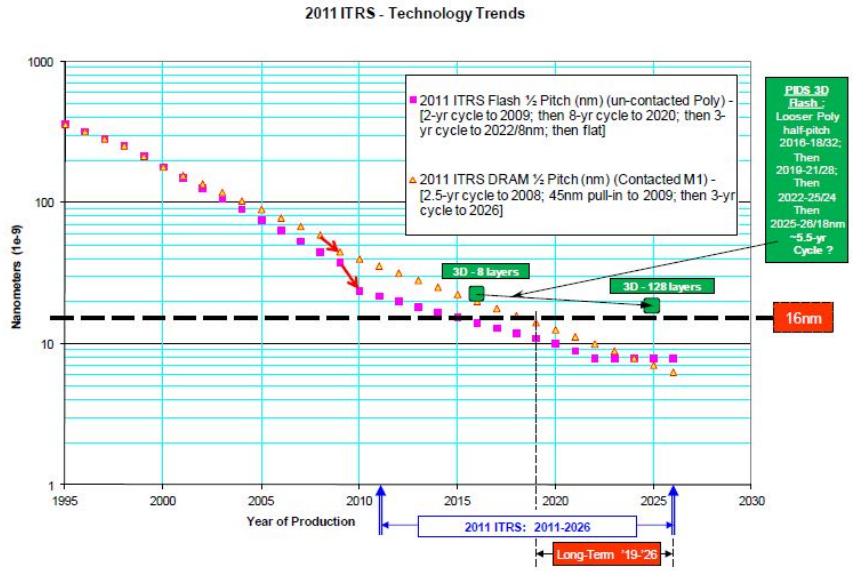


Fig. 1.2 – DRAM e Flash Memories Half Pitch Trend [5]

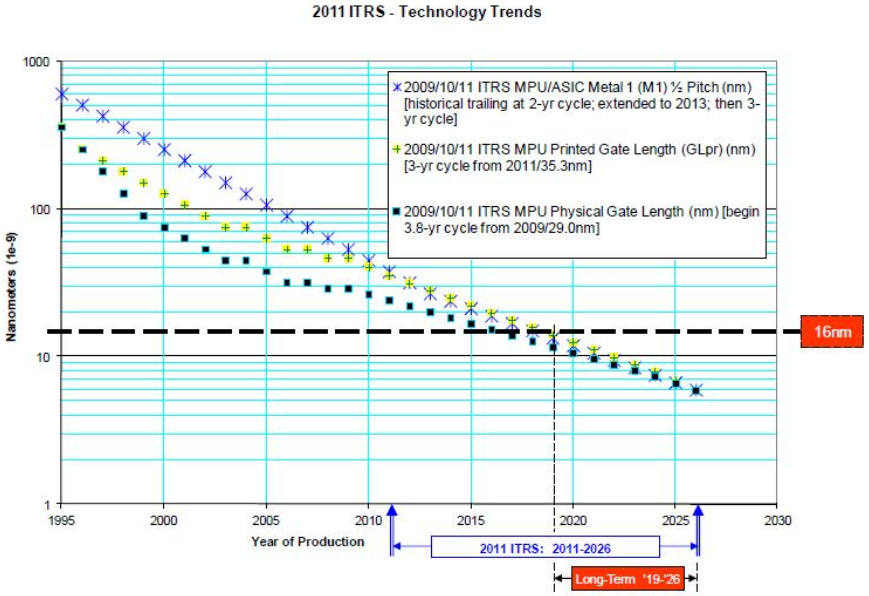


Fig. 1.3 – MPU/high performance ASIC Half Pitch and Gate Lenght Trend [5]

Tabella 1.1 – ITRS Technology Trend Targets [6]

Year of Production	2015	2017	2020	2021	2024	2027	2030
Logic ½ Pitch (nm)	32	25	20	16	13	10	7
DRAM ½ Pitch (nm)	24	20	17	14	11	8.4	7.7
DRAM cell size (µm2)	0.00346	0.0024	0.00116	0.00078	0.00048	0.00028	0.00024
2D Nand Flash ½ Pitch (nm)	15	13	12	12	12	12	12
NAND Flash Product highest density (Kgates/mm))	256G	384G	768G	1T	1.5T	3T	4T
3D NAND flash number of memory layers	32	32-48	64-96	96-128	128-192	256-384	384-512
Power Supply Voltage - Vdd (V)	0.80	0.75	0.70	0.65	0.55	0.45	0.40
1/(CV/I) (1/psec)	1.53	1.75	1.97	2.10	2.29	2.52	3.17
On-chip local clock MPU HP	5.95	6.44	6.96	7.53	8.14	8.8	9.9
Maximum number wiring levels	13	14	14	15	15	16	17
MPU High-Performance Physical Gate Length (GLph) (nm)	17	14	12	10	8	7	5

1.2. Caratteristiche della progettazione a livello di sistema

Electronic System Level (ESL) design è una metodologia di progettazione emergente che permette al progettista di lavorare a livelli di astrazione più alti rispetto la descrizione RTL.
La spinta è dovuta alla crescente complessità dei circuiti integrati che ha reso di fatto l'implementazione RTL meno efficiente [7]. L'enorme capacità dei semiconduttori oggi richiede l'astrazione del design a livelli più alti in modo da poter controllare la complessità, aumentando così la produttività del design.
La sintesi ad alto livello genera automaticamente un'architettura RTL a partire da specifiche funzionali fornite tramite linguaggi di programmazione ad alto livello come il C, C++, MATLAB o derivati. Tramite HLS è possibile creare diverse strutture RTL a partire dalla stessa specifica ad alto livello, usando direttive fornite al compilatore. Questo permette di esplorare più velocemente il design space, sperimentando diverse architetture che prevedono differenti trade-off tra performance, occupazione d'area e dissipazione di potenza.
Inoltre, si mantengono le caratteristiche tipiche dei linguaggi di programmazione ad alto livello come il riuso e la portabilità del software; di conseguenza, l'IP generato mediante HLS può essere facilmente trasportato su diverse piattaforme di implementazione (FPGA) e diversi domini applicativi [8].
Grazie alla sintesi automatica, l'adozione dell'ESL design riduce notevolmente il tempo di debug e verifica della struttura RTL [9]. I dettagli implementativi vengono gestiti dai tool di sintesi. Utilizzando linguaggi di alto livello non si tiene conto di costrutti specifici della progettazione hardware, come il timing, i bit di precisione, la concorrenzialità, i livelli di

gerarchia e la sincronizzazione dei segnali. Per non perdere consapevolezza e controllo sull'hardware sono state introdotte delle estensioni, librerie e direttive aggiuntive per adattare i linguaggi di programmazione ad alto livello alla progettazione hardware [10].

Capitolo 2.

System Level Design

Per poter sfruttare le potenzialità messe a disposizione dai progressi tecnologici nel campo dei semiconduttori, solitamente si fa ricorso a tre approcci di base [11]:

1. Alzare il livello di astrazione del design,
2. Introdurre la metodologia di design reuse mediante l'uso di componenti IP,
3. Automatizzazione tramite tool EDA di una parte del flusso di progetto.

In generale, l'astrazione mira a ridurre lo sforzo di specificare una funzionalità desiderata omettendo l'informazione non necessaria.
Componenti pre-progettati chiamati IP vengono assemblati in un nuovo design per migliorarne la produttività. Il numero di blocchi da definire da zero diventa più piccolo se il progettista può riutilizzare componenti IP esistenti. I blocchi IP sono presi da precedenti progetti di sviluppo o utilizzando provider IP esterni. L'IP deve soddisfare una serie di requisiti per poter essere considerato adatto al riutilizzo. In primo luogo, dovrebbero essere di alta qualità in termini di correttezza e completezza. In secondo luogo, le interfacce del componente IP devono essere chiaramente definite e documentate. In terzo luogo, i blocchi IP dovrebbe separare la comunicazione dalle parti funzionali per facilitare l'integrazione in un'architettura di design esistente.
Oggi, il design è supportato da diversi tool EDA che rendono automatica la sintesi del progetto dal livello alto di astrazione al livello più basso, riducendo così il tempo di progettazione.

È necessaria una metodologia di progettazione all'avanguardia che soddisfi certi requisiti [12]:

- alto grado di riutilizzo dei blocchi IP e in particolare dei sistemi IP (IP-reuse),
- efficiente modellazione degli IP a livello di sistema, per la creazione di prototipi virtuali completamente certificati (TLM-based),
- partizionamento hardware/software ottimale,
- hardware e software co-design basato su un livello elevato di astrazione del sistema (platform-based),
- co-simulazione e co-verifica efficienti.

Al fine di rendere gestibile la complessità di sistemi eterogenei e l'integrazione di IP differenti, serve una metodologia di progettazione ad alto livello (ESL design) che integra i nuovi concetti degli stili di modellizzazione ad alto livello con un corrispondente set di strumenti EDA system-oriented.

2.1. Livelli di astrazione

I livelli di astrazione si estendono su più livelli di accuratezza, che vanno dal livello di sistema al livello fisico. Ogni livello introduce nuovi dettagli al modello.

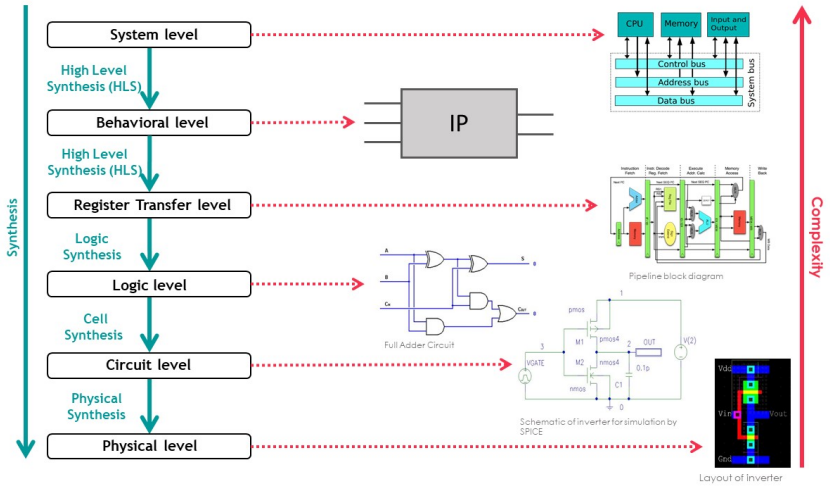


Fig. 2.1 – Livelli di astrazione

2.1.1. System level

A livello di sistema, non vengono descritte le strutture interne dei blocchi hardware.
Il sistema viene descritto attraverso la definizione separata di moduli, che contengono processi concorrenti che implementano il comportamento del sistema, e canali che modellano i meccanismi di comunicazione tra i moduli.
Questo livello viene chiamato anche transazionale (Transaction Level - TL) perché focalizza l'attenzione sulle transazioni ovvero sullo scambio di informazioni tra i canali di comunicazione.
La transazionale (Transaction Level Modeling - TLM) è una rappresentazione ad alto livello di un sistema particolarmente utile per la progettazione e l'analisi del protocollo di comunicazione.
I modelli transazionali mappano le transazioni su un ciclo di clock (clock-accuracy), a differenza dei modelli comportamentali che invece sono un-timed. Quindi, protocolli sincroni, ritardi delle connessioni e i tempi di accesso possono essere accuratamente modellati.

2.1.2. Behavioral level

A questo livello di astrazione, un sistema generico viene rappresentato in un Control and Data Flow Graph (CDFG), in cui vengono individuate le operazioni e le loro dipendenze.

Un CDFG ha una stretta correlazione con la rispettiva macchina a stati finiti FSM (Finite-State-Machine) che viene definito a livello strutturale, con la differenza che possono essere necessari più cicli di clock per eseguire uno stato del CDFG di un sistema.
Lo scopo principale di un modello comportamentale è sviluppare e testare il funzionamento di un componente.
Spesso viene chiamato anche livello algoritmico in quanto il comportamento del sistema viene descritto tramite linguaggi di programmazione.

2.1.3. Structural level

A livello strutturale o RTL (Register Tranfer Level), un sistema viene descritto in termini di datapath (registri, circuiti combinatori, bus di basso livello) e unità di controllo, generalmente implementati come macchine a stati finiti (FSM). L'accuratezza dei dettagli prevede la specifica del numero di bit per i tipi di dato, la definizione dei pin di ingresso e uscita per le interfacce I/O e il flusso dei segnali fra registri e le operazioni logiche o aritmetiche che vengono svolte su tali segnali.
Lo scopo principale dei modelli RTL è di sviluppare e testare l'architettura interna e la logica di controllo di un componente in modo che il progetto soddisfi le specifiche richieste e i vincoli temporali.

2.2. Metodologia di progettazione ad alto livello

Le metodologie di progettazione si sono evolute insieme con tecnologia di produzione, la complessità del progetto e l'automazione del design.
Di seguito sono spiegati i concetti chiave che identificano la progettazione ad alto livello.

2.2.1. Metodologia TLM-based

La progettazione di un sistema inizia dalla formulazione del modello ad alto livello.
La rappresentazione TLM è ad oggi considerata uno standard nella progettazione a livello di sistema [13].
Al fine di semplificare il processo di progettazione, i progettisti utilizzano in genere dei modelli intermedi a diversi livelli di timing che aggiungono gradualmente dei dettagli implementativi. L'intero design viene suddiviso in modelli più semplici, ognuno dei quali ha uno specifico obiettivo progettuale.
Fig. 2.2 [14] riporta la relazione tra i gradi di accuratezza del timing e i modelli del design. In ascissa sono indicati i gradi di accuratezza per quanto riguarda i blocchi computazionali, mentre in ordinata sono indicati i gradi di accuratezza dei canali di comunicazione.
Il livello un-timed rappresenta la funzionalità del design senza dettagli di implementazione.
Il livello approximate-timed contiene dettagli d'implementazione a livello di sistema, come l'architettura di sistema, le relazioni di mappatura tra processi delle specifiche e gli elementi di elaborazione dell'architettura. Consente una stima preliminare delle prestazioni e del consumo di potenza.

Il livello cycle-timed contiene i dettagli d'implementazione a livello dei cicli di clock (cycle-accurate) [15].
Vengono definiti sei modelli:

1) Modello di specifica: consiste nel modello comportamentale del sistema ed è privo di dettagli di implementazione
2) Modello component-assembly: il sistema viene descritto tramite processi eseguiti in modo concorrente e memorie globali che comunicano attraverso i canali. La parte di comunicazione del modello (canale) non è temporizzata, mentre la parte computazionale è temporizzata secondo la stima dell'esecuzione dei processi.
3) Modello bus-arbitration: rispetto al modello precedente viene definito il modello di bus astratto che implementa l'arbitraggio del bus tra i canali e i processi; non sono specificati né le temporizzazioni né i protocolli di comunicazione per i canali, mentre il modello di bus astratto è definito a livello approximate-timed.
4) Modello bus-functional: la parte di comunicazione viene descritta a livello cycle-timed che specifica il vincolo temporale della comunicazione secondo il protocollo, mentre la parte computazionale è descritta a livello approximate-timed.
5) Modello computazionale cycle-accurate: la parte di comunicazione viene descritta a livello approximate-timed, mentre la parte computazionale è descritta a livello cycle-timed, i componenti hardware sono modellati in RTL.
6) Modello d'implementazione: sia la parte di comunicazione sia la parte computazionale sono descritte a livello cycle-timed.

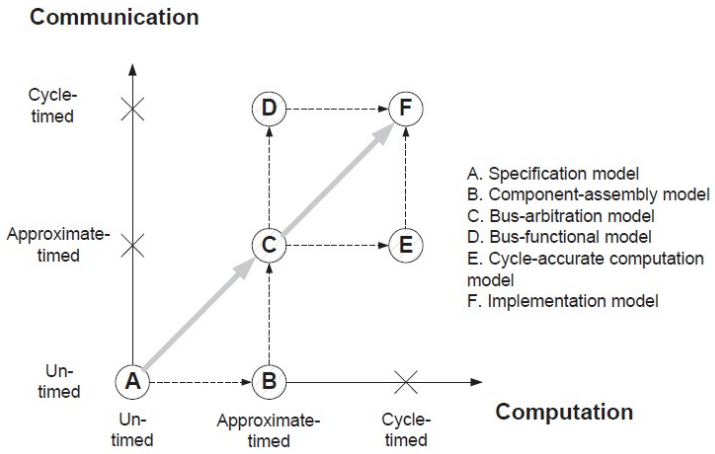


Fig. 2.2 - Grafico di modellazione TLM [14]

2.2.2. Metodologia Platform-based

La maggior parte degli approcci di progettazione ad alto livello utilizzati nell'industria hanno l'inconveniente di dedicarsi principalmente all'hardware o al software ma non entrambi. La progettazione software perde il concetto del tempo e concorrenza che rende praticamente impossibile descrivere, sintetizzare e verificare l'hardware.

La progettazione hardware ha una semantica troppo specifica per poter permettere ai progettisti software di lavorare bene.
Un approccio più potente sarebbe quello di utilizzare una metodologia mista supportata da strumenti EDA che includa sia hardware che software design, favorisca l'uso di alti livelli di astrazione per la descrizione iniziale del progetto, offra un'efficace esplorazione delle possibili strutture del design e permetta di raggiungere un'implementazione dettagliata attraverso la sintesi.
Alberto Sangiovanni-Vincentelli presenta l'approccio platform-based come un metodo di progettazione che soddisfa questi requisiti e lo identifica quindi come il più adatto per la progettazione ad alto livello [16, 17].
Ci sono molte definizioni di piattaforma che dipendono dal campo di applicazione. Nel dominio IC, una piattaforma si considera un circuito integrato flessibile la cui personalizzazione per una particolare applicazione si ottiene programmando uno o più componenti del chip. Rientrano tra questi le FPGA (Field-Prorammable-Gate-Array) o il software eseguito su un microprocessore o un DSP (Digital Signal Processor).

La metodologia platform-based è una metodologia intermedia che prevede una piattaforma esistente come substrato utilizzata come libreria di componenti.
Può essere vista come la combinazione di due approcci [18]:

- Top-down: si mappa un'istanza della funzionalità del progetto in un'istanza della piattaforma,
- Bottom-up: si personalizza la piattaforma scegliendo i componenti programmabili tra quelli disponibili in libreria.

2.2.3. Linguaggi di specifica ad alto livello

In seguito alla formulazione del modello, questo viene scritto in un linguaggio eseguibile.
I linguaggi solitamente usati sono linguaggi di programmazione ad alto livello come il C o C++ o derivati.

2.2.3.1. SystemC

SystemC è una piattaforma di progettazione composta da un set di librerie di classi C ++ e un kernel di simulazione che supporta i concetti di modellazione hardware TLM e RTL. La libreria di classi aggiuntiva serve per espandere il linguaggio C/C++ verso concetti di modellazione hardware come la concorrenza, processi event-driven, protocolli di comunicazione, timing e tipi di dato tipici dell'hardware (bit, tipi di dato fixed-point con un numero arbitrario di bit di precisione).
SystemC è utilizzato progettare, simulare ed eseguire il debug di un System-on-Chip descritto a livello di sistema [19]. SystemC è molto adatto alla progettazione dei sistemi HW / SW dal livello di sistema fino al livello RT. SystemC fornisce un kernel di simulazione basato su eventi che pianifica l'esecuzione dei processi che implementano le funzionalità di moduli e canali di comunicazione.

2.2.3.2. SystemC-WMS

SystemC-WMS è un'ulteriore libreria del C++, sviluppata per estendere il SystemC a blocchi analogici eterogenei. Si basa sull'uso di un'interfaccia analogica standard per la connessione dei moduli basati sulla propagazione delle onde [20, 21].
I moduli possono descrivere un sistema nei domini elettrico, termico, idraulico, rotazionale, e tri-fase, chiamati nature. Per ogni natura, l'ambiente fornisce una libreria base di moduli.
Ogni modulo analogico è descritto da un sistema di equazioni differenziali ordinarie non lineari come segue:

$$\begin{cases} \dot{x} = f(x;a) \\ b = g(x;a) \end{cases} \tag{3.1}$$

dove *f* e *g* sono funzioni vettoriali che descrivono la dinamica del sistema, *x*, *a* e *b* sono, rispettivamente, il vettore della variabile di stato, dell'ingresso e dell'uscita.
L'uso del modello dell'onda incidente/riflessa semplifica il problema di rappresentazione delle connessioni tra moduli eterogenei, assumendo che i moduli utilizzino onde incidenti come input e producano onde riflesse come output. Tipi di connessioni diverse (serie, parallelo, etc.) possono essere spiegate utilizzando un canale appropriato che invia le onde ai moduli collegati insieme e consente la formulazione di un'interfaccia analogica generica e standard utilizzabile per la connessione di moduli di nature differenti. SystemC-WMS è una libreria SystemC gratuita che può essere scaricata da [22].

2.3. Flusso di progettazione ad alto livello

In letteratura [23], il flusso di progettazione ESL è suddiviso in varie fasi, come mostrato in Fig. 2.3 [24]:

1. Specifica,
2. Modellazione,
3. Analisi pre-partizionamento,
4. Partizionamento HW/SW,
5. Analisi post partizionamento e debug,
6. Verifica post partizionamento,
7. Implementazione HW/SW.

A partire da documento di testo in cui sono indicati i requisiti e i vincoli di progetto, il progettista traduce la specifica iniziale in vari modelli eseguibili in linguaggi a livello di sistema. Il modello eseguibile risultante che descrive l'applicazione generale viene sottoposto a una fase di analisi preliminare da cui si può avere una stima iniziale delle prestazioni, occupazione d'area, risorse utilizzate e consumo di potenza. Questo serve per esplorare diverse alternative del design funzionale.
Sulla base di tale analisi preliminare, la successiva fase di progettazione definisce la struttura del sistema, ottenuta dal suo modello comportamentale.
Il processo di partizionamento hardware/software definisce quali parti della funzionalità sono implementati in software o in hardware. L'effetto del partizionamento hardware/software viene analizzato nella fase di analisi post partizionamento e debug. Se

necessario, si effettua un nuovo partizionamento per soddisfare meglio i requisiti di progetto.
Quindi, la fase di verifica post-partizionamento assicura che il comportamento originariamente previsto dei componenti software e hardware partizionati sia conservato. Infine, la fase di implementazione HW/SW crea i modelli RTL sintetizzabili dell'hardware utilizzando tecniche di sintesi ad alto livello e produce il software che deve essere eseguito dall'hardware di destinazione.

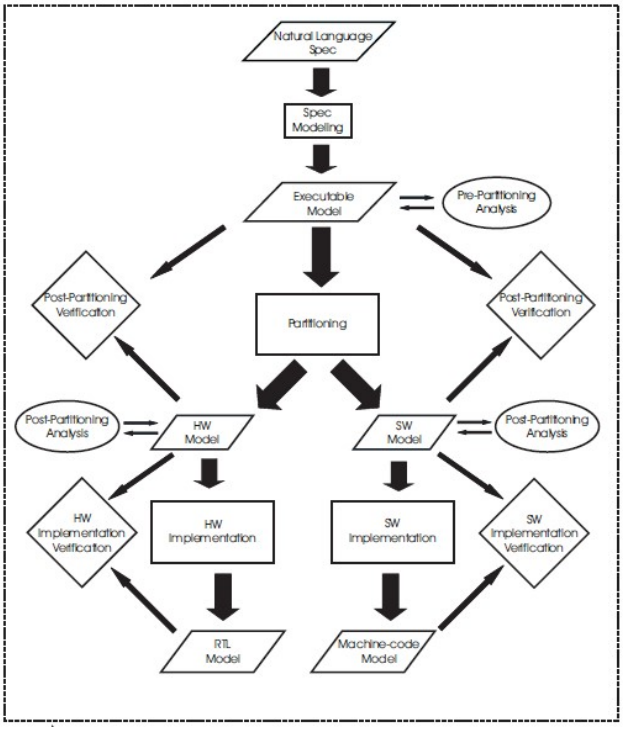


Fig. 2.3 – Flusso di progettazione ESL [24]

2.4. Sintesi ad alto livello

Il modello RTL del sistema è generato a partire dal suo modello comportamentale tramite un processo che viene chiamato sintesi ad alto livello (High-Level Synthesis - HLS) che ad oggi è reso automatico tramite tool EDA.
La procedura di implementazione di un sistema mediante HLS, mostrata in Fig. 2.4 [10], è costituita da vari step:

1) Descrizione iniziale eseguibile: il testbench e comportamento del sistema vengono scritti in linguaggi di programmazione ad alto livello, come il C, C++ e derivati (SystemC), a cui si aggiungono eventuali direttive e/o vincoli di progettazione forniti al compilatore,

2) Simulazione C: viene eseguita la simulazione funzionale del sistema sulla base del testbench scritto in precedenza,
3) Sintesi: viene eseguita la sintesi che crea una struttura RTL del sistema sulla base delle direttive o vincoli dati precedentemente,
4) Co-simulazione C-RTL: la struttura RTL viene simulata tramite il testbench scritto in C,
5) Esportazione in linguaggi HDL: la struttura RTL viene tradotta nel linguaggio HDL specificato in modo che possa essere utilizzato come IP in progetti a livelli di astrazione più bassi.

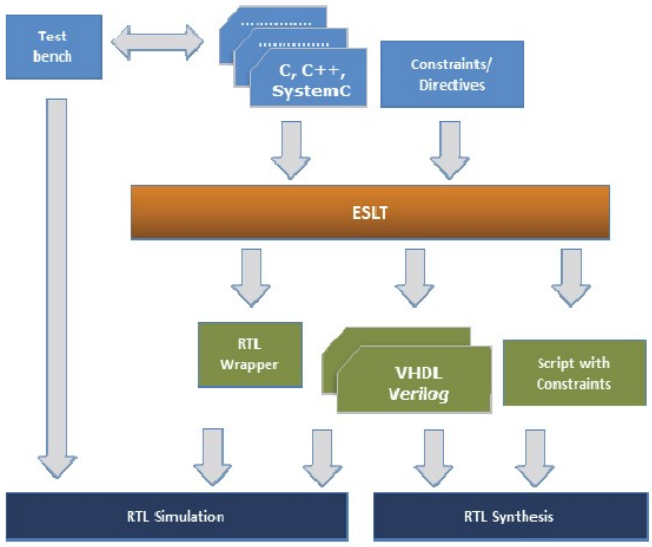


Fig. 2.4 – Procedura d'implementazione mediante tool di sintesi ad alto livello [10]

2.4.1. Algoritmo di sintesi

L'algoritmo di sintesi ad alto livello vede l'esecuzione di differenti task [25, 26, 27]:

1) Compilazione

La specifica eseguibile che definisce il comportamento funzionale del sistema viene codificata e analizzata. Il codice viene ottimizzato eliminando le righe inutili, le false dipendenze e la propagazione delle costanti. Da qui viene generato un Control and Data Flow Graph (CDFG) in modo che il datapath esegua gli assegnamenti delle variabili e l'unità di controllo implementi i costrutti di controllo. In questa fase vengono individuate le dipendenze tra le operazioni e il grado di parallelismo possibile.

2) Allocazione delle risorse:

L'attività di allocazione determina il tipo e la quantità di risorse da utilizzare sia per il funzionamento sia per le comunicazioni. L'obiettivo dell'allocazione è di fare il

compromesso più opportuno tra il costo del design e le prestazioni. Se la descrizione iniziale contiene vincoli inerenti al parallelismo delle operazioni, questo comporta l'allocazione di più risorse hardware aumentando così l'occupazione d'area e i costi ma permette anche l'esecuzione parallela delle operazioni e degli accessi in memoria, con conseguente miglioramento delle prestazioni. Invece, allocando meno risorse diminuiscono l'occupazione d'area e i costi ma si forza l'esecuzione sequenziale delle operazioni con conseguente peggioramento delle prestazioni.

3) Scheduling

L'attività di scheduling determina quale operazione viene eseguita in ogni control step. I control step hanno una stretta correlazione con i cicli di clock della FSM definita nel modello RTL finale.
L'obiettivo della fase di scheduling è ottimizzare il throughput quindi ridurre il più possibile il numero di cicli di clock. In altre parole, lo scheduling massimizza l'utilizzo delle risorse allocate.

4) Binding

L'attività di binding assegna le operazioni, gli accessi alla memoria e le interfacce di comunicazione in ogni ciclo di clock alle risorse hardware disponibili.
Il binding è composto da tre sottotask in base al tipo di unità:

- Storage binding assegna le variabili alle unità di memoria (due variabili che non sono attive contemporaneamente possono essere assegnate allo stesso registro, due variabili a cui non si accede contemporaneamente possono essere assegnate alla stessa porta);
- Functional-unit binding assegna ciascuna operazione in un control step a una unità funzionale. Un'unità funzionale o uno stadio della pipeline può eseguire solo un'operazione per ogni ciclo di clock;
- Interconnection binding assegna un'unità di interconnessione come un multiplexer o un bus ad ogni trasferimento di dato tra unità.

5) Estrazione della logica di controllo

Viene estratta la logica di controllo sulla base del CDFG che avrà una stretta correlazione con la macchina a stati definita nel modello RTL sintetizzato
Infine, viene generata l'architettura RTL.

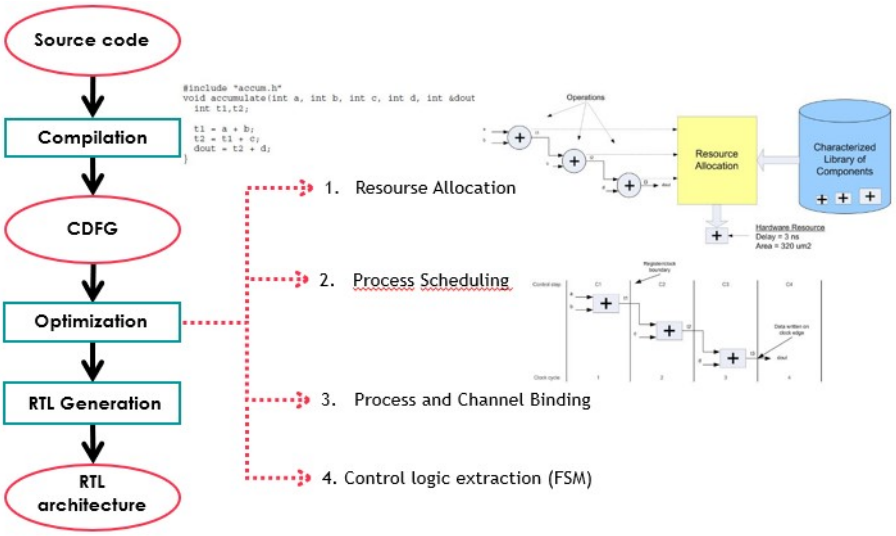


Fig. 2.5 – Step dell'algoritmo di sintesi ad alto livello

2.4.2. Ottimizzazione con direttive

La sintesi in modalità standard mappa le funzioni e sotto-funzioni della descrizione eseguibile rispettivamente in entity e componenti in RTL con una corrispondenza 1 a 1. Le risorse sono allocate in modo da ottimizzare sempre il throughput. I corpi delle funzioni vengono eseguiti in modo sequenziale.
I loop vengono mantenuti "rolled" (arrotolati) cioè le iterazioni vengono eseguite in modo sequenziale.
Il top-level del codice C viene mappato in semplici porte d'ingresso e di uscita secondo il protocollo Wire Handshake.

Se la mappatura standard non soddisfa i requisiti richiesti è possibile pilotare la sintesi tramite le direttive di ottimizzazione date al compilatore di sintesi.
Di seguito sono riportate le direttive più usate:

- LOOP UNROLL:

I loop delle funzioni in C/C ++ vengono mantenuti "rolled" di default, quindi la sintesi crea la logica per un'iterazione del loop e la progettazione RTL esegue questa logica per ogni iterazione in modo sequenziale.
La direttiva loop unroll "srotola" i loop creando copie multiple del corpo del loop nel progetto RTL, il che consente di eseguire alcune o tutte le iterazioni del loop in parallelo.
È possibile srotolare i loop per aumentare l'accesso ai dati e il throughput.
LOOP UNROLL consente al loop di essere completamente o parzialmente srotolato. Lo srotolamento completo crea una copia del corpo del loop nell'RTL per ogni iterazione, in modo che l'intero loop possa essere eseguito contemporaneamente. Lo srotolamento parziale consente di specificare un fattore N, per creare N copie del corpo del loop e ridurre

di conseguenza le iterazioni. Per srotolare completamente un loop, il numero di iterazioni deve essere noto al momento della compilazione. Questo non è richiesto per lo srotolamento parziale.

– RESOURCE:

Specifica quale una risorsa (core) deve essere utilizzata per implementare una variabile (array, operazione aritmetica o argomento di una funzione) nella struttura RTL. Il tipo di risorse disponibili dipende dalla piattaforma d'implementazione utilizzata.

– PIPELINE

Riduce l'intervallo di inizializzazione per una funzione o un loop consentendo l'esecuzione parallela delle operazioni.
Una funzione o loop in pipeline può elaborare nuovi input ogni N clock, dove N è l'intervallo di inizializzazione (II) dell'operazione. L'intervallo di inizializzazione predefinito per la pipeline è pari a 1, ciò significa che un nuovo input viene elaborato ad ogni ciclo di clock. La pipeline di un loop consente di implementare le operazioni del loop in modo concorrente, come mostrato in Fig. 2.6 [28].
È possibile implementare una struttura pipeline per massimizzare il throughput.

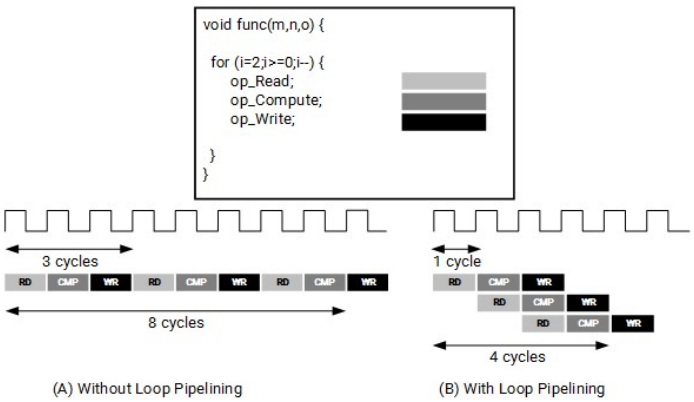


Fig. 2.6 – Loop pipeline [28]

– ALLOCATION

Specifica il numero massimo di istanze allocabili per una determinata risorsa per limitare l'allocazione delle risorse nella struttura RTL.

– INTERFACE

Nel design basato su C, tutte le operazioni di input e output sono eseguite a tempo zero attraverso gli argomenti delle funzioni. In una progettazione RTL, queste stesse operazioni di input e output devono essere eseguite attraverso le porte dell'interfaccia di comunicazione utilizzando un protocollo specifico di I/O.
La direttiva INTEFACE specifica come vengono create le porte RTL dalla definizione della funzione durante la sintesi dell'interfaccia.

2.4.3. Tool commerciali e accademici

Dagli anni 2000, sono stati sviluppati nuovi tool HLS che, a differenza delle precedenti generazioni, accettano linguaggi di input basati su C/C ++ o derivati (SystemC). Questo rende i tool molto più accessibili anche ai progettisti software rispetto alle precedenti versioni che accettavano solo linguaggi di descrizione dell'hardware (HDL). Consente inoltre di progettare sia hardware che software utilizzando modelli e linguaggi comuni, facilitando il co-design e la co-verifica software/hardware.
L'uso di linguaggi basati su C rende facile sfruttare le più recenti tecnologie di compilatori software per la parallelizzazione e l'ottimizzazione nei tool di sintesi.
Nei moderni strumenti HLS basati su C si sono introdotte delle estensioni semantiche per adattare i linguaggi di input alla sintesi hardware in modo da poter specificare i bit di precisione, il clock, la concorrenza delle operazioni e altri costrutti fondamentali nell'hardware design e poter sintetizzare espressioni tipiche del software come i puntatori, le recursioni e il polimorfismo.
I tool vengono suddivisi in base anche al dominio di progetto, come mostrato in Fig. 2.7 [29].
Le due categorie sono distinte in:

1) Domain-Specific Language (DSL)
2) General-Purpose Programmable Language

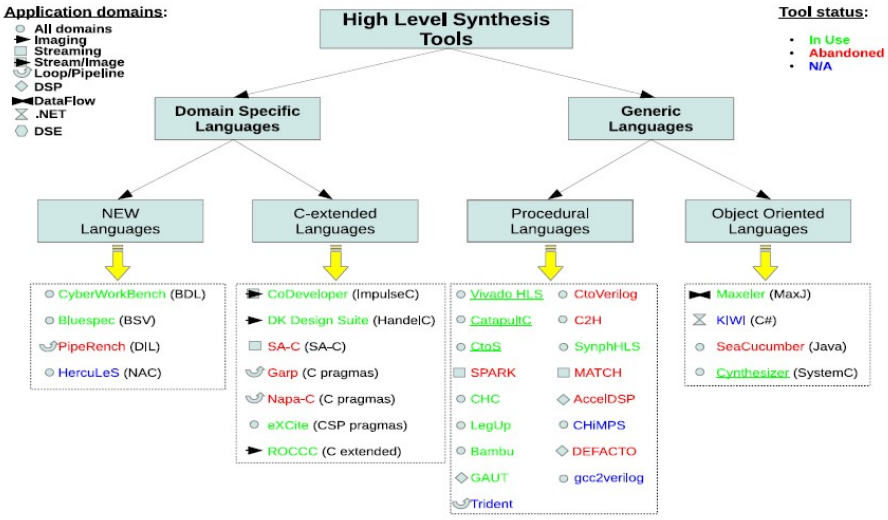


Fig. 2.7 – Classificazione dei tool HLS in base al linguaggio accettato in ingresso [29]

Si sono studiate le caratteristiche che contraddistinguono i più recenti tool HLS [29, 30].
La tabella 2.1 [29] riporta i tool HLS in uso e obsoleti, indicando anche il tipo di licenza, i linguaggi di input e output supportati, il dominio di progetto e altre caratteristiche.

Tabella 2.1 – Tool EDA per HLS [29]

Status	Compiler	Owner	License	Input	Output	Year	Domain	TestBench	FP	FixP
In Use	eXCite	Y Explorations	Commercial	C	VHDL/Verilog	2001	All	Yes	No	Yes
	CoDeve-loper	Impulse Accelerated	Commercial	Impulse-C	VHDL Verilog	2003	Image Streaming	Yes	Yes	No
	Catapult-C	Calypto Design Systems	Commercial	C/C++ SystemC	VHDL/Verilog SystemC	2004	All	Yes	No	Yes
	Cynthesizer	FORTE	Commercial	SystemC	Verilog	2004	All	Yes	Yes	Yes
	Bluespec	BlueSpec Inc.	Commercial	BSV	SystemVerilog	2007	All	No	No	No
	CHC	Altium	Commercial	C subset	VHDL/Verilog	2008	All	No	Yes	Yes
	CtoS	Cadence	Commercial	SystemC TLM/C++	Verilog SystemC	2008	All	Only cycle accurate	No	Yes
	DK Design Suite	Mentor Graphics	Commercial	Handel-C	VHDL Verilog	2009	Streaming	No	No	Yes
	GAUT	U. Bretagne	Academic	C/C++	VHDL	2010	DSP	Yes	No	Yes
	MaxCompiler	Maxeler	Commercial	MaxJ	RTL	2010	DataFlow	No	Yes	No
	ROCCC	Jacquard Comp.	Commercial	C subset	VHDL	2010	Streaming	No	Yes	No
	Synphony C	Synopsys	Commercial	C/C++	VHDL/Verilog SystemC	2010	All	Yes	No	Yes
	Cyber-WorkBench	NEC	Commercial	BDL	VHDL Verilog	2011	All	Cycle/ Formal	Yes	Yes
	LegUp	U. Toronto	Academic	C	Verilog	2011	All	Yes	Yes	No
	Bambu	PoliMi	Academic	C	Verilog	2012	All	Yes	Yes	No
	DWARV	TU. Delft	Academic	C subset	VHDL	2012	All	Yes	Yes	Yes
	VivadoHLS	Xilinx	Commercial	C/C++ SystemC	VHDL/Verilog SystemC	2013	All	Yes	Yes	Yes
N/A	Trident	Los Alamos NL	Academic	C subset	VHDL	2007	Scientific	No	Yes	No
	CHiMPS	U. Washington	Academic	C	VHDL	2008	All	No	No	No
	Kiwi	U. Cambridge	Academic	C#	Verilog	2008	.NET	No	No	No
	gcc2verilog [45]	U. Korea	Academic	C	Verilog	2011	All	No	No	No
	HercuLeS	Ajax Compiler	Commercial	C/NAC	VHDL	2012	All	Yes	Yes	Yes
Abandoned	Napa-C	Sarnoff Corp.	Academic	C subset	VHDL/Verilog	1998	Loop	No	No	No
	DEFACTO	U. South Calif.	Academic	C	RTL	1999	DSE	No	No	No
	Garp	U. Berkeley	Academic	C subset	bitstream	2000	Loop	No	No	No
	MATCH	U. Northwest	Academic	MATLAB	VHDL	2000	Image	No	No	No
	PipeRench	U.Carnegie M.	Academic	DIL	bitstream	2000	Stream	No	No	No
	SeaCucumber	U. Brigham Y.	Academic	Java	EDIF	2002	All	No	Yes	Yes
	SA-C	U. Colorado	Academic	SA-C	VHDL	2003	Image	No	No	No
	SPARK	U. Cal. Irvine	Academic	C	VHDL	2003	Control	No	No	No
	AccelDSP	Xilinx	Commercial	MATLAB	VHDL/Verilog	2006	DSP	Yes	Yes	Yes
	C2H	Altera	Commercial	C	VHDL/Verilog	2006	All	No	No	No
	CtoVerilog	U. Haifa	Academic	C	Verilog	2008	All	No	No	No

Nel 2017 Xilinx ha sviluppato Vivado HLx, una versione di Vivado HLS con licenza accademica e open source [31]. Ha alcune limitazione rispetto a Vivado HLS, come ad esempio che non è in grado di sintetizzare più progetti simultaneamente. Questo viene giustificato dal fatto che è stato sviluppato per essere utilizzato nel contesto accademico.
In ogni caso, mantiene molte delle caratteristiche del suo analogo commerciale, come il fatto di fornire una gamma completa di linguaggi supportati in ingresso e in uscita e la creazione automatica del testbench per la simulazione RTL.
Per questi motivi è stato usato come tool di sintesi nel progetto di ricerca.

2.4.3.1. Vivado HLS

Vivado HLS e Vivado HLx utilizzano AutoPilot come algoritmo di sintesi, sviluppato dall'azienda AutoESL che è stata poi acquisita da Xilinx.
Il flusso di sintesi è mostrato in Fig. 2.8 [32].
AutoPilot genera una struttura RTL in Verilog, VHDL e SystemC.
Per poter eseguire la co-simulazione, AutoPilot crea wrapper di test bench (TB) in Verilog/VHDL/SystemC modo che il progettista possano sfruttare il testbench originale scritto in C/C ++/ SystemC per verificare la correttezza della struttura RTL in uscita.

AutoPilot genera anche uno script di simulazione da utilizzare con simulatori RTL di terze parti. Così, i progettisti possono facilmente utilizzare il loro ambiente di simulazione esistente per verificare l'RTL generato.
Oltre a generare RTL, AutoPilot crea anche dei report di sintesi che stimano l'utilizzo delle risorse FPGA, come il timing, la latenza, il throughput e il ritardo massimo del design sintetizzato. I report includono una ripartizione delle prestazioni e delle risorse allocate per singoli moduli, funzioni e loop nel codice sorgente. Ciò consente al progettista di identificare rapidamente aree specifiche che impattano maggiormente sulla specifica richiesta e quindi regolare la sintesi tramite le direttive di ottimizzazione o modificare il codice sorgente in base alla mappatura RTL desiderata. Infine, i file HDL generati e i vincoli di progettazione sono importati in tool Xilinx per essere trasformati in forma di Bitstream per programmare l'FPGA.

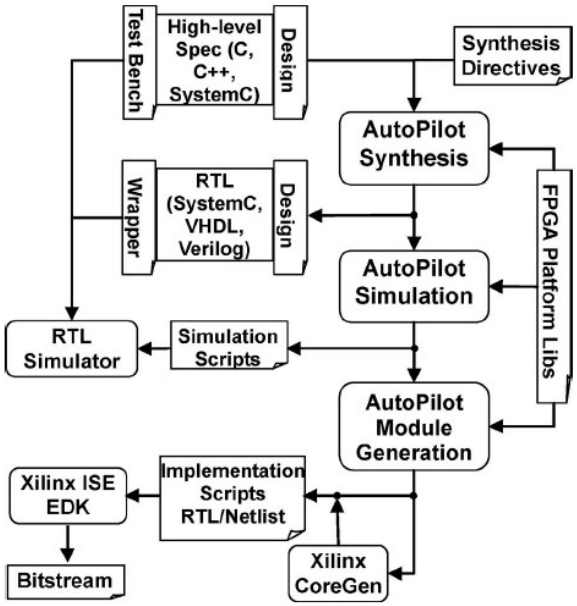


Fig. 2.8 – Flusso di progetto dell'algoritmo AutoPilot su FPGA Xilinx [32]

Capitolo 3.

Sintesi ad alto livello per il progetto di sintetizzatori audio digitali

Si è scelto lo studio dei sintetizzatori audio digitali come caso applicativo della metodologia di sintesi ad alto livello su FPGA perché per la loro intrinseca complessità progettuale sono molto adatti ad essere usati come casi di studio per sviscerare tutte le potenzialità della metodologia HLS. Infatti, se si considera che un suono deve essere riprodotto in 20ms in modo che il ritardo non venga percepito dall'orecchio umano e che gli strumenti oggi sul mercato prevedono almeno 128 voci di polifonia, il progetto di un sintetizzatore audio digitale deve soddisfare svariati vincoli progettuali per garantire le elevate performance richieste dai musicisti.
I sintetizzatori sono strumenti in grado di generare autonomamente segnali audio, generalmente comandati tramite tastiera.
Possono generare suoni fedeli quanto più possibile a quelli prodotti da strumenti musicali reali o creare suoni ed effetti non esistenti in natura. Questo dipende dal tipo di sintesi effettuata e dal numero di bit di quantizzazione.
In questo filone applicativo, la ricerca è stata svolta in collaborazione con il team di progettazione hardware della Korg Italy, azienda specializzata nel design di pianoforti e arranger digitali.

3.1. Sintesi musicale

Negli strumenti musicali tradizionali il suono è prodotto dalla vibrazione di parti meccaniche. Negli strumenti digitali, la vibrazione è descritta da segnali, che riproducono la variazione nel tempo della pressione acustica. Tradizionalmente, nella musica occidentale, il suono è caratterizzato da parametri musicali quali altezza, intensità, durata metrica e timbro [33].
Per sintesi musicale digitale si intende la generazione di un suono tramite l'elaborazione di segnali digitali eseguita su metriche che caratterizzano i segnali stessi, ovvero frequenza, ampiezza e forma d'onda, in base ai parametri musicali scelti in funzione del risultato sonoro che si vuole ottenere [34].
La tabella 3.1 riporta la corrispondenza tra le caratteristiche del segnale e i parametri musicali.

Tabella 3.1 - Corrispondenza tra le caratteristiche del segnale e i parametri musicali

Caratteristica del segnale	Parametro musicale	Sensazione sonora
Frequenza	Altezza	Acuto - Grave
Ampiezza	Intensità	Piano-Forte
Forma d'onda	Timbro	Chiaro-Scuro Armonico-Inarmonico

Si possono individuare le seguenti classi di algoritmi di sintesi:

- generazione diretta: campionamento, sintesi additiva e granulare;
- feed-forward: sintesi sottrattiva, modulazioni, distorsione non lineare;
- feed-back: sintesi per modelli fisici

Sintesi addittiva
Il suono è sintetizzato sommando le sinusoidi corrispondenti alle sue componenti armoniche. Nella sintesi additiva, suoni complessi sono prodotti mediante la sovrapposizione di suoni elementari, generalmente sinusoidali.
Il problema della sintesi additiva è che ha bisogno di un gran numero di parametri di controllo per ciascuna nota.

Sintesi granulare
La sintesi granulare condivide con la sintesi additiva l'idea di comporre suoni complessi a partire da suoni più semplici. Mentre la sintesi additiva si basa sulla sovrapposizione temporale di sinusoidi, la sintesi granulare invece si basa sulla successione di forme d'onda di breve durata (tipicamente da 1 a 100 msec) chiamate grani.

Sintesi sottrattiva
Mentre la sintesi additiva costruisce suoni complessi sommando insieme semplici suoni sinusoidali tempo varianti, la sintesi sottrattiva è basata sull'idea complementare di passare un segnale a larga banda (contenente molte armoniche) attraverso un filtro tempo-variante in modo da lasciare solo le armoniche volute per produrre la forma d'onda desiderata.

Sintesi per modulazione di frequenza (Frequency Modulation - FM) o ampiezza (Amplitude Modulation - AM)
Si ottiene il segnale voluto attraverso la modulazione in frequenza o in ampiezza di un segnale portante.

Sintesi per distorsione non lineare
La sintesi per distorsione non lineare, nota anche come waveshaping, avviene passando una sinusoide attraverso un blocco distorcente. Infatti, se una sinusoide viene distorta attraverso un filtro lineare modifica la sua ampiezza e fase, ma non la forma d'onda. Se invece l'amplificatore è non lineare la forma d'onda del segnale viene modificata e vengono create altre componenti spettrali.

Sintesi per modelli fisici
Gli algoritmi di sintesi visti si basano su modelli generativi del segnale musicale prodotto.
La sintesi per modelli fisici segue invece un approccio alternativo che si basa sulla rappresentazione della dinamica degli oggetti (reali o virtuali) responsabili della produzione del suono. La sintesi è quindi basata sull'uso di modelli formali di strumenti musicali tradizionali; il suono viene generato simulando numericamente la dinamica dello strumento che lo produce.

Sintesi per campionamento
Vengono campionate alcune forme d'onda scelte come riferimento e i campioni salvati in memoria.
La sintesi basata su campionamento consiste nel generare un segnale sulla base delle forme d'onda campionate modificandone la frequenza e l'ampiezza.
Esistono vari tipi di sintesi per campionamento, come la Direct Digital Synthesis (DDS) e la sintesi vettoriale.

3.1.1. Direct Digital Synthesis

La Direct Digital Synthesis (DDS) [35] è un metodo per produrre una forma d'onda analogica, solitamente un'onda sinusoidale, generando un segnale variabile nel tempo in forma digitale a partire da un segnale di riferimento precedentemente campionato e poi eseguendo una conversione da digitale ad analogico.
Un blocco DDS è capace di produrre e controllare in modo accurato forme d'onda a varie frequenze e profili. Quindi è possibile programmare e riprogrammare in modo digitale la forma d'onda di uscita.

Un dispositivo DDS è costituito principalmente da vari blocchi, come mostrato in Fig. 3.1:
- un accumulatore di frequenza che contiene l'indirizzo del campione in memoria da prelevare chiamato Current Frequency Address (CFA),
- un registro di frequenza in cui è memorizzato il valore digitale dell'incremento dell'accumulatore chiamato Frequency Control Word (FCW),
- una look-up table o una ROM che contiene i campioni del segnale sinusoidale di riferimento,
- un convertitore digitale/analogico.

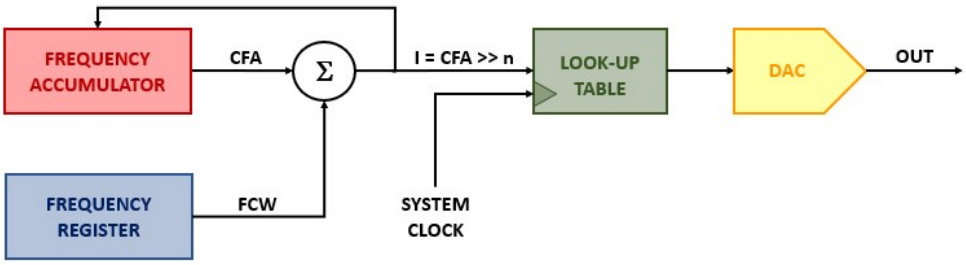


Fig. 3.1 – Schema a blocchi di un dispositivo DDS

Ad ogni ciclo di clock, il valore di FCW memorizzato nel frequency register viene sommato al valore dell'accumulatore di frequenza che è inizialmente inizializzato a zero.
FCW rappresenta il valore digitale del passo tra la frequenza del segnale voluto f_o e la frequenza del segnale di riferimento f_{rif} ed è pari a f_o/f_{rif}.
Immaginando l'oscillazione sinusoidale come un vettore che ruota intorno a un cerchio di fase, come mostrato graficamente in Fig. 3.2, ogni punto disegnato sul cerchio corrisponde a un valore equivalente della fase in periodo di un'onda sinusoidale. Un giro del vettore attorno al cerchio, a velocità costante, produce un periodo completo dell'onda sinusoidale di uscita. L'overflow del contatore che implementa l'accumulatore di frequenza riproduce l'andamento periodico della sinusoide d'uscita.
Di conseguenza, considerando che servono n bit per indirizzare una look-up table di dimensione della pari a 2n, CFA è una parola di n bit.

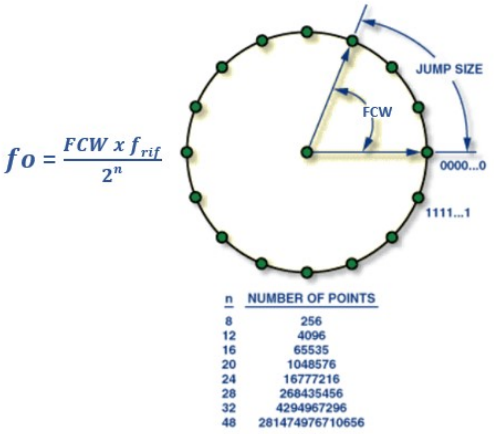


Fig. 3.2 – Rappresentazione grafica del funzionamento di un blocco DDS

L'uscita troncata dell'accumulatore di frequenza (CFA / 2^n) funge da indirizzo I per la look-up table (o ROM) in cui sono memorizzati i campioni della sinusoide di riferimento campionata a frequenza f_s. Ogni indirizzo nella look-up table corrisponde a un punto di fase sull'onda sinusoidale da 0 a 2π. La fase di un segnale sinusoidale φ è pari $2\pi f + \varphi_0$. La look-up table contiene le informazioni relative all'ampiezza digitale per un periodo

completo della sinusoide di riferimento. Vengono prelevati due campioni all'indirizzo I e I + 1. Il valore dell'interpolazione dei due campioni è il valore digitale dell'ampiezza della sinusoide di uscita alla frequenza $f_o = CFA * f_{rif} / 2^n$. Il valore di ampiezza digitale viene poi convertito in valore analogico dal DAC.

Sulla base di questo funzionamento è possibile eseguire il pitch shifting ovvero la variazione della frequenza della sinusoide di uscita modificando il valore di FCW. In questo modo si ottengono sinusoide a diverse frequenza partendo dagli stessi campioni della sinusoide di riferimento.

3.2. Sintetizzatore audio digitale

Un sintetizzatore digitale è costituito da vari blocchi funzionali:

- Oscillatore controllato che si occupa di generare la forma d'onda fondamentale in base alla tecnica di sintesi musicale scelta. È pilotato da un segnale di controllo detto pitch che indica la frequenza fondamentale del segnale voluto,
- Filtri controllati che modificano la risposta in frequenza del segnale fondamentale,
- Amplificatore controllato che determina l'ampiezza del segnale d'uscita,
- Generatore d'inviluppo che modifica i segnali di controllo generati dall'LFO in base ai comandi ricevuti esternamente,
- Low-Frequency Oscillator (LFO) che è un oscillatore a bassa frequenza che genera la forma d'onda dei segnali di controllo.

La struttura fondamentale di un sintetizzatore digitale è schematizzata in Fig. 3.3.

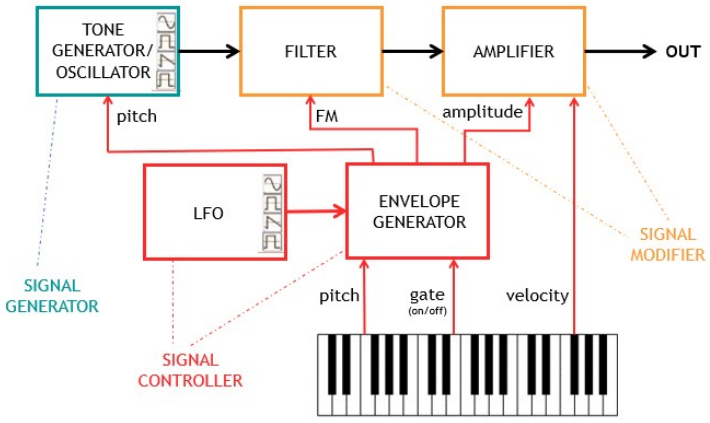


Fig. 3.3 – Schema a blocchi di un sintetizzatore audio digitale

Da notare che Fig. 3.3 schematizza la struttura di un sintetizzatore digitale per un'unica voce (tone) ovvero l'oscillatore controllato genera un singolo segnato che verrà poi elaborato dai blocchi a valle. Si consideri che oggi in commercio possiamo trovare anche sintetizzatori musicali da 128 voci di polifonia. Una simile struttura necessita di un numero

di blocchi dedicati alla generazione del suono pari al numero di voci che deve essere in grado di suonare contemporaneamente che siano in grado di sintetizzare il segnale in uscita in un tempo inferiore ai 20ms. Per limitare il numero di risorse utilizzate è necessario applicare tecniche di progettazione complesse come il pipeling.

3.2.1. Filtro di Chamberlin

I filtri di un sintetizzatore audio digitale possono essere di vari tipi.
Nel progetto di ricerca si è presa in esame la struttura del filtro di Chamberlin, che per le sue caratteristiche intrinseche, rappresenta un'alternativa interessante ai blocchi di filtraggio utilizzati più comunemente.

Il filtro di Chamberlin, introdotto da [36] è un tipo di filtro IIR (Infinite Impulse Response) del secondo ordine a variabili di stato.
La particolarità di questo filtro, oltre alla alta qualità di filtraggio che caratterizza i filtri IIR, è che permette di avere 4 tipi di filtraggio in uscita (low-pass, high-pass, band-pass, notch) con un'unica struttura.

Il comportamento del filtro è pilotato dai parametri di controllo F e D indipendenti tra loro, come mostrato in Fig. 3.4.
F controlla la frequenza naturale f_0; mentre D setta il fattore di smorzamento, che è inversamente proporzionale al fattore di qualità Q. La frequenza naturale f_0 e il fattore di qualità Q dipendono da entrambi i coefficienti di filtro F, D. Nel caso con smorzamento nullo (D = 0), la frequenza naturale f_0 coincide con la frequenza di taglio f_c.
F è legato alla frequenza di taglio f_c e alla frequenza di campionamento f_s secondo la seguente relazione:

$$F = 2\sin(\pi f_c / f_s) \tag{3.1}$$

Per applicazioni musicali, f_s è pari a 48kHz.
Di seguito sono riportate le equazioni delle funzioni di trasferimento delle 4 uscite del filtro [37]:

$$\begin{gathered} H_{LP}(z) = F^2 z / [z^2 + (F^2 + DF - 2)\, z + (1 - DF)] \\ H_{BP}(z) = Fz\,(z - 1) / [z^2 + (F^2 + DF - 2)\, z + (1 - DF)] \\ H_{HP}(z) = (z - 1)^2 / [z^2 + (F^2 + DF - 2)\, z + (1 - DF)] \\ H_{Notch}(z) = H_{LP}(z) + H_{HP}(z) \end{gathered} \tag{3.2}$$

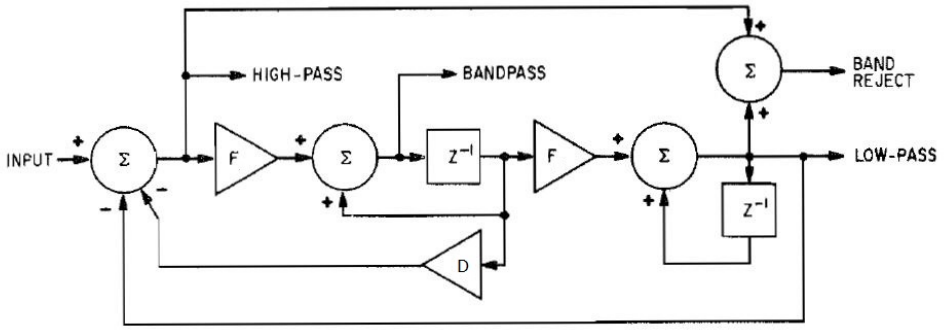


Fig. 3.4 – Struttura del filtro di Chamberlin

Un'analisi sulle funzioni di trasferimento delle uscite del filtro Chamberlin mostra il comportamento e i vincoli di stabilità per valori variabili dei parametri F e D.
La deviazione più importante dal comportamento ideale si ha per valori grandi di D, dove il filtro inizia con picco intorno a $f_s/2$ e alla fine diventa instabile con l'aumento F, come mostrato in Fig. 3.5 (b) e Fig. 3.5 (f). Di conseguenza, il limite di stabilità per il filtro Chamberlin è $0 < F < 1$ e $0 < D < 2$. Dall'eq (3.1) si ottiene che $f_c = f_s/6$; quindi per le applicazioni musicali la frequenza massima di taglio f_c è pari a 8kHz.
Alcuni dei risultati dell'analisi sulle funzioni di trasferimento del filtro sono riportati in Fig. 3.5 (a)-(f).

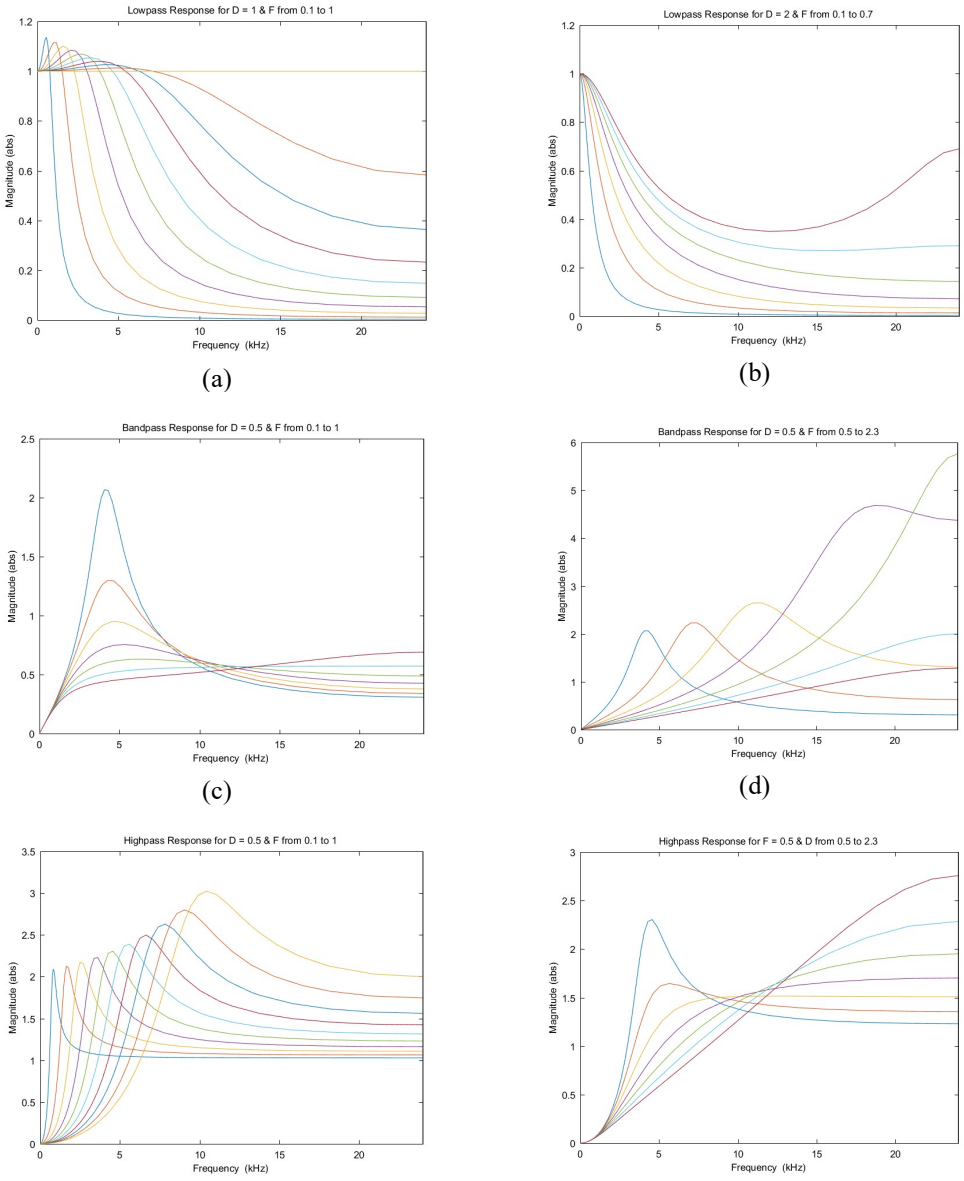


Fig. 3.5 – Risposta in frequenza delle uscite del filtro per diversi valori dei parametri F e D: (a), (b) caso stabile e instabile del filtro passa-basso; (c), (d) caso stabile e instabile del filtro passa-banda; (e), (f) caso stabile e instabile del filtro passa-alto;

3.3. Progetto del filtro di Chamberlin

La parte di ricerca mirata sulla metodologia di sintesi ad alto livello ha preso in esame la progettazione di un filtro di Chamberlin come caso applicativo e non come vero argomento di ricerca.
Di seguito sono riportate tutte le implementazioni del filtro sviluppate durante la ricerca.

3.3.1. Sviluppo mediante metodologia di progettazione tradizionale (HDL)

Come punto di partenza, si è sviluppata la struttura del filtro di Chamberlin a basso livello in ambiente HDL utilizzando il tool Vivado 2017.3 e testata per diversi tipi di segnali d'ingresso audio. Infine, questa struttura è stata implementata Xilinx FPGA Zinq7020 integrata sulla board Zedboard.
Per i test, D è stato fissato a 0.5. La tabella 3.2 riporta i valori di F e f_s per diversi segnali d'ingresso.

Tabella 3.2 – Test audio per diversi segnali di ingresso

Input	f_s [Hz]	F	Duration[s]
Dirac Impulse	48000	0.1308	0.02
Piano Audio	48000	0.1308	6
Guitar Audio	11025	0.5622	13
White noise	4410	0.1424	10
Pink noise	4410	0.1424	10

Nell'implementazione hardware del filtro, i numeri rappresentativi delle variabili e dei coefficienti devono essere memorizzati in registri di dimensione finiti. Nell'aritmetica a virgola fissa, i numeri utilizzati sono frazionari e definiti nell'intervallo [-1, 1] e possono essere rappresentati da un numero finito di bit. Questa rappresentazione comporta errori di quantizzazione. Varie notazioni sono usate per rappresentare la lunghezza della parola e la parte decimale in un numero binario a virgola fissa. Si è utilizzata la notazione Qm.n, dove m rappresenta il numero di bit della parte intera e n rappresenta il numero di bit frazionari.
Il segno è rappresentato dal bit più significativo (MSB) di m in quanto l'intera parola viene rappresentata in complemento a 2. Il numero totale di bit è b = m + n.
La struttura del filtro VHDL viene testata con diversi valori di n (Qm.4, Qm.7, Qm.9, Qm.11, Qm.13, Qm.15), fino al numero totale massimo di bit pari a 20.
Il segnale d'ingresso è normalizzato quindi il suo valore è compreso in [-1, 1]; di conseguenza, si è verificato che è sufficiente 1 bit per rappresentare la parte intera m per evitare l'overflow.
Questa analisi ha dimostrato che la deviazione dal caso in virgola mobile diventa rilevante per n <9, come mostrato in Fig. 3.6. Da un'analisi soggettiva dell'output dei suoni di chitarra e piano si può considerare che la differenza tra il valore floating point e fixed point è trascurabile per n > 12.

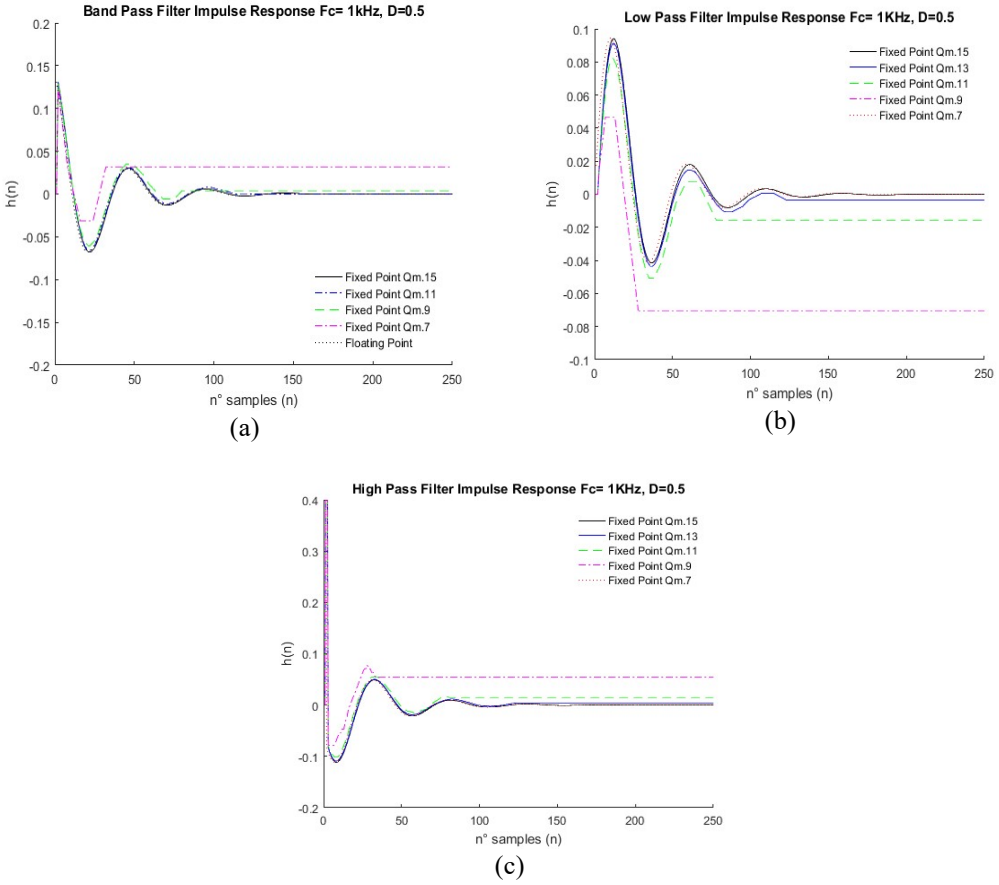


Fig. 3.6 – Risposta all'impulso delle uscite del filtro di Chamberlin per diversi valori di b: (a) Passa-banda; (b) Pssa-basso; (c) Passa-alto

Il tool Vivado fornisce i report delle risorse utilizzate in termini di numero di Look-Up Table (LUT), di Flip-Flop (FF), Digital Signal Processor (DSP).
Inoltre, fornisce anche i valori della dissipazione di potenza.
Fig. 3.7 mostra i valori dell'utilizzazione delle risorse e del consumo di potenza della struttura del filtro di Chamberlin testata con diversi valori del numero totale di bit b di rappresentazione fixed point.

Risorse utilizzare Zedboard (Zinq 7020)									
N.bit	LUT	FF	DSP	Potenza totale [mW]	Clocking [mW]	Logic [mW]	DSP [mW]	Potenza Statica [mW]	Potenza Dinamica [mW]
8	249	16	0	104,3640915	0,000380784	3,93E-04	0	104,363318	0,000773425
9	341	18	0	104,3641272	0,000388416	4,19E-04	0	104,3633202	0,000806977
10	394	20	0	1,04E+02	0,000396048	4,87E-04	0	104,3633291	8,83E-04
11	78	22	3	104,3639079	0,000435408	1,49E-04	2,29061E-05	104,3633008	0,000607012
12	86	24	3	104,3645763	0,00044304	2,11E-04	0,000608628	104,3633135	0,00126275
13	92	26	3	104,3646114	0,00044304	1,91E-04	0,000662632	104,3633142	0,00129716
14	99	28	3	104,3647005	0,000450672	2,20E-04	0,000708446	104,3633213	0,001379261
15	106	30	3	104,3649863	0,000497664	2,28E-04	0,000911919	104,3633485	1,64E-03
16	114	32	3	104,3663803	0,000497664	4,81E-04	0,001983545	104,3634176	0,002962689
17	120	34	3	104,3677639	0,000505296	6,18E-04	0,003162155	104,3634782	0,004285734
18	127	36	3	104,3692426	0,000505296	8,09E-04	0,004354585	104,3635742	0,005668427
19	134	38	3	104,3709908	0,000512928	8,39E-04	0,005951403	104,3636879	0,007302871
20	363	40	3	104,3721977	0,00052056	3,07E-03	0,004643799	104,3639674	8,23E-03

Fig. 3.7 - Utilizzazione delle risorse e consumo di potenza dell'implementazione HDL del filtro di Chamberlin per vari valori di b

Sulla base dell'analisi precedente si è deciso di sviluppare in ambiente HDL tramite tool Vivado la struttura del filtro con 3 uscite (passa-basso, passa-alto, passa-banda) nel formato di rappresentazione a virgola fissa Q5.15.
Per i test, D è stato fissato a 0.5; f_s è pari a 48kHz, f_c è pari a 1kHz; di conseguenza, F risulta essere pari a 0.1308. Fig. 3.8 riporta la struttura RTL di questa implementazione.
Si è notato che 2 delle 3 moltiplicazioni di cui fa uso la struttura del filtro vengono sintetizzate in RTL in DSP. L'uscita HP (passa-alto) del filtro non viene elaborata tramite DSP ma solo tramite LUT.

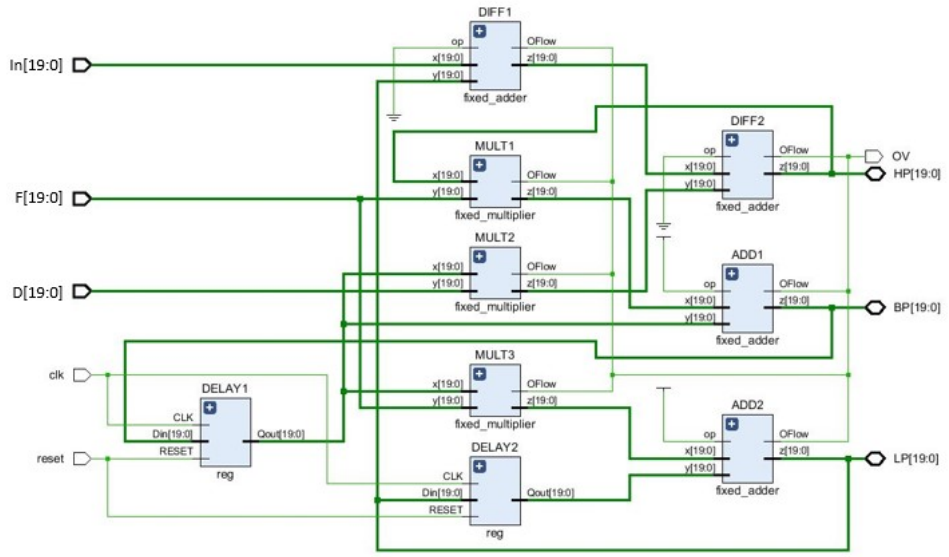


Fig. 3.8 - Struttura RTL del filtro di Chamberlin in ambiente HDL con 3 uscite e formato Q5.15 delle variabili del sistema

3.3.2. Sviluppo mediante metodologia di sintesi ad alto livello (HLS)

Si è sviluppata in ambiente HLS la stessa struttura del filtro (3 uscite, Q5.15) per confrontare le due implementazioni tramite il tool VIVADO HLx.

```
// CFQ5_15_3output.h
#include "ap_cint.h"

void CFQ5_15_3output(
    int20 Input,
    int20 D,      //  D = 1 / Q
    int20 F,     //  F = 2 * sin(M_PI * Fc / Fs)
    int20 *LowpassOutput,
    int20 *HighpassOutput,
    int20 *BandpassOutput
    );
```

La direttiva di inclusione #include “ap_cint.h” è stata inserita per poter utilizzare il tipo di dato int20 che definisce un tipo di dato intero che verrà sintetizzato in VHDL in un numero a 20 bit; altrimenti il tipo di dato int standard sarebbe stato sintetizzato in un numero a 16 bit.
I report post-sintesi riportano una stima dei valori delle risorse allocate in termini di numero di Look-Up Table (LUT), di Flip-Flop (FF), Digital Signal Processor (DSP) e banchi di memoria (BRAM), suddivisi per i componenti che le utilizzano, come riportato in Fig. 3.9.
Come si nota, a differenza dell’implementazione HDL, il tool HLS stima 3 DSP; di conseguenza, il numero di LUT stimate è diminuito per alcune operazioni che nell’implementazione HDL con 2 DPS venivano svolte dalle LUT, nel progetto HLS vengono eseguite dal DSP aggiuntivo.

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	3	0	130
FIFO	-	-	-	-
Instance	-	-	-	-
Memory	-	-	-	-
Multiplexer	-	-	-	-
Register	-	-	41	-
Total	0	3	41	130
Available	280	220	106400	53200
Utilization (%)	0	1	~0	~0

Fig. 3.9 – Report post-sintesi delle risorse allocate per la struttura del filtro di Chamberlin

3.3.3. Confronto HLS - HDL

A questo punto, per poter confrontare strutture simili si è deciso di utilizzare le direttive di ottimizzazione nel progetto HLS per limitare a 2 il numero di DSP allocati.
Si è notato che il tool VIVADO HLx non permette l'applicazione di alcune direttive a certi oggetti; questo si è attributo a una limitazione della licenza accademica voluta dalla casa produttrice Xilinx.
Quindi non è stato possibile imporre l'elaborazione dell'uscita HP con solo l'utilizzo di LUT tramite la direttiva RESOURCE (#pragma HLS RESOURCE variable=HighpassOutput core=Mul_LUT) per l'incompatibilità tra il set di operazioni necessarie per l'elaborazione e la direttiva stessa (WARNING: [SYN 201-303] Cannot apply functional unit assignment of 'Mul_LUT' on 'sub' operation due to incompatible operation sets.).
Quindi si è ricorso all'utilizzo della direttiva ALLOCATION (#pragma HLS ALLOCATION instances=mul limit=2 operation) per limitare a 2 il numero di operazioni di moltiplicazione per ciclo di clock. Così facendo però si aumenta la latenza e quindi il throughput di un ciclo di clock perché le prime due moltiplicazioni vengono eseguite in parallelo mentre la terza viene forzata a essere eseguita al ciclo di clock successivo. Però viene ridotto il ritardo sul critical path.
La tabella 3.3 riporta i dell'occupazione d'area, delle prestazioni e del consumo di potenza mettendo a confronto le due strutture del filtro sviluppate in HLS e le relative architetture RTL sintetizzate con l'implementazione HDL.

Tabella 3.3 – Confronto tra vari progetti del filtro di Chamberlin e relative strutture sintetizzate

	HLS	HLS Allocation	Post-HLS	Post-HLS Allocation	HDL
LUT	130	163	317	272	199
FF	41	97	43	85	40
DSP	3	2	3	2	2
Throughput	1	2	1	2	1
Critical path delay (ns)	20.13	12.08	19.195	12.307	20
Power consumption (mW)	-	-	104	104	104.37

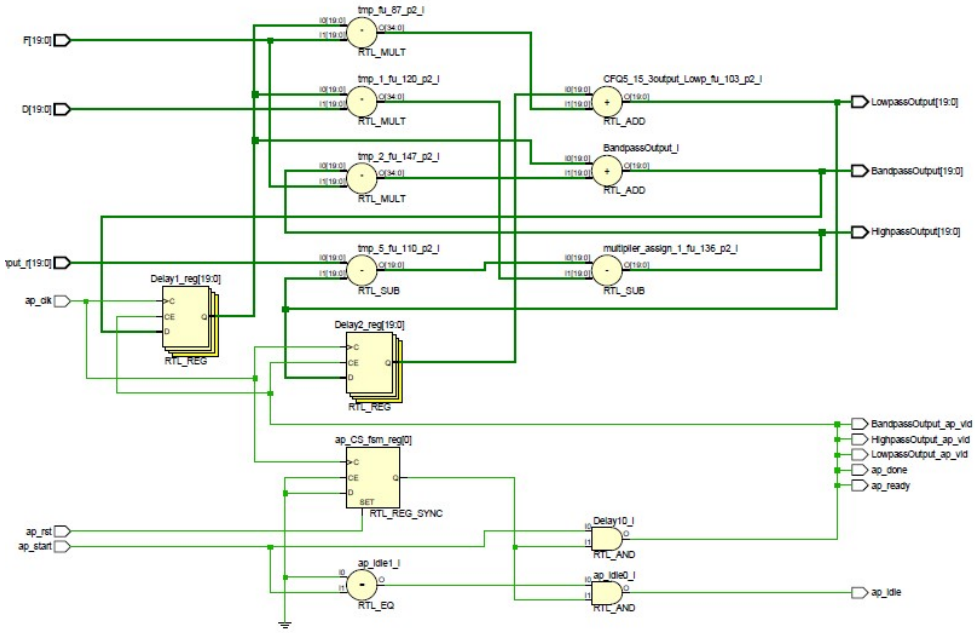


Fig. 3.10 – Struttura RTL sintetizzata dal progetto HLS senza utilizzo di direttive

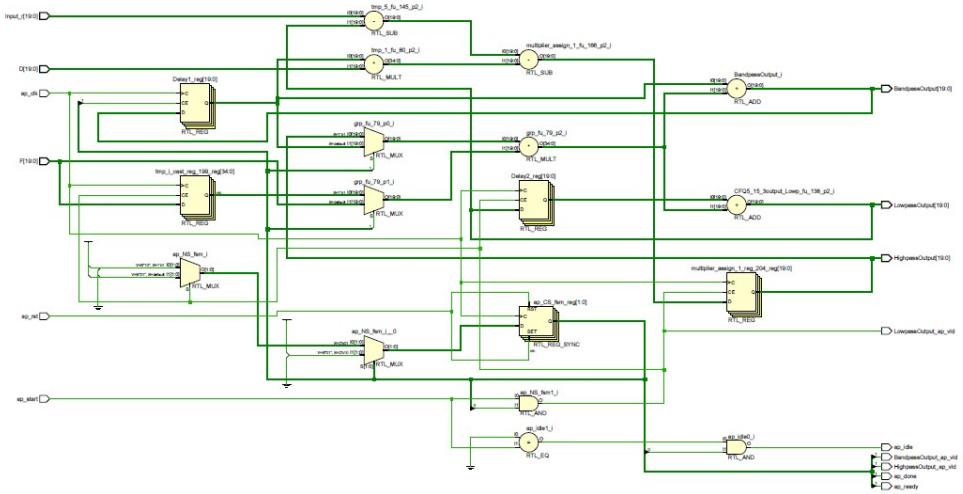


Fig. 3.11 – Struttura RTL sintetizzata dal progetto HLS utilizzando la direttiva ALLOCATION

3.3.4. Progetto HLS del filtro di Chamberlin a una sola voce

Sulla base dei risultati ottenuti confrontando le due metodologie di progettazione, si è deciso di sviluppare in HLS la struttura del filtro di Chamberlin che fornisce in uscita tutte 4 le uscite possibili nel formato Q8.16 e Q8.24.
Nel caso ad una sola voce di polifonia, il filtro riceve in ingresso una sola voce (un valore) e produce in uscita il valore corrispondente a quella voce.
Di seguito sono riportate le implementazioni sviluppate.

3.3.4.1. Progetto HLS del filtro nel formato Q8.16

Si è sviluppato in HLS il progetto del filtro di Chamberlin che fornisce tutte 4 le uscite possibili, con formato Q8.16 di bit di rappresentazione fixed point per vari valori del clock. Di seguito sono riportati i risultati ottenuti per valori di clock estremi considerati più significativi a 48 kHz e a 48 MHz.

```
// ChamberlinFilterQ8_16.h
#include "ap_cint.h"

void ChamberlinFilter(
    int24 Input,
    int24 D,     //  D = 1 / Q
    int24 F,    //  F = 2 * sin(M_PI * Fc / Fs)
    int24 *LowpassOutput,
    int24 *HighpassOutput,
    int24 *NotchOutput,
    int24 *BandpassOutput
    );
```

La direttiva di inclusione #include "ap_cint.h" è stata inserita per poter utilizzare il tipo di dato int24 che definisce un tipo di dato intero che verrà sintetizzato in VHDL in un numero a 24 bit, come desiderato.

Si è esplorato il design space attraverso l'applicazione di direttive di ottimizzazione:

- min latency: non è stata applicata nessuna direttiva, corrisponde al caso in cui viene massimizzato il throughput;
- max latency: si è limitata l'esecuzione di una moltiplicazione per ciclo di clock, corrisponde al caso di massima latenza.

Le tabelle 3.4 e 3.6 riportano i risultati delle stime HLS delle metriche di progetto rispettivamente con frequenza di clock pari a 48kHz e 48MHz per le diverse direttive applicate.
Le tabelle 3.5 e 3.7 invece riportano i corrispondenti valori per la struttura sintetizzata. Questi valori dimostrano che HLS stima le metriche nel caso peggiore quindi con massima allocazione delle risorse; l'implementazione RTL ottimizza le risorse allocate fornendo i valori effettivi delle prestazioni, dei consumi e dell'occupazione d'area.

Si nota che rispetto alla stima HLS il numero dei DSP utilizzati nel progetto sintetizzato è raddoppiato. Inoltre, i DSP sono dispositivi in grado di eseguire tante operazioni in breve tempo. Quindi la riduzione del numero di LUT allocate e del ritardo associato alla propagazione dei dati sono maggiori quanto è maggiore il numero di DSP allocati, perché la sintesi decide di attribuire ai DSP l'esecuzione di operazioni che altrimenti sarebbero state eseguite dalle LUT.

Si fa presente che è possibile in linea di principio limitare il numero di DSP totali ad una quantità specifica. Nel progetto in esame questo non è stato possibile; la causa è imputabile, come detto in precedenza, alla versione gratuita della licenza di utilizzo del software che ha funzionalità limitate rispetto alla versione commerciale.

Tabella 3.4 – Metriche del progetto HLS in formato Q8.16 e Clk = 48kHz

	Min latency	Max latency
LUT	261	266
FF	49	114
DSP	3	2
Throughput	1	2
Critical path delay (ns)	25.48	15.31

Tabella 3.5 – Metriche del progetto HLS sintetizzato in formato Q8.16 e Clk = 48kHz

	Min latency	Max latency
LUT	145	195
FF	48	122
DSP	6	4
Throughput	1	2
Critical path delay (ns)	20.375	13.529

Tabella 3.6 – Metriche del progetto HLS in formato Q8.16 e Clk = 48MHz

	Min latency	Max latency
LUT	276	266
FF	116	114
DSP	3	2
Throughput	1	2
Critical path delay (ns)	15.31	15.31

Tabella 3.7 – Metriche del progetto HLS sintetizzato in formato Q8.16 e Clk = 48MHz

	Min latency	Max latency
LUT	147	195
FF	74	122
DSP	6	4
Throughput	1	2
Critical path delay (ns)	11.859	13.539

Da notare che sia per entrambi i clock considerati i valori delle risorse occupate non cambiano in modo sostanziale; questo significa che i DSP hanno una potenza di calcolo tale da permettere di avere clock a frequenze più elevate. Si deve arrivare a un valore di clock pari o superiore a 100 MHz per vedere un significativo aumento delle risorse utilizzate e della latenza.
A dimostrazione di questo punto, si è calcolato che il critical path per entrambi i valori del clock, per entrambe le direttive utilizzate, è costituito dagli stessi blocchi, mostrato in Fig. 3.12 e Fig. 3.13.

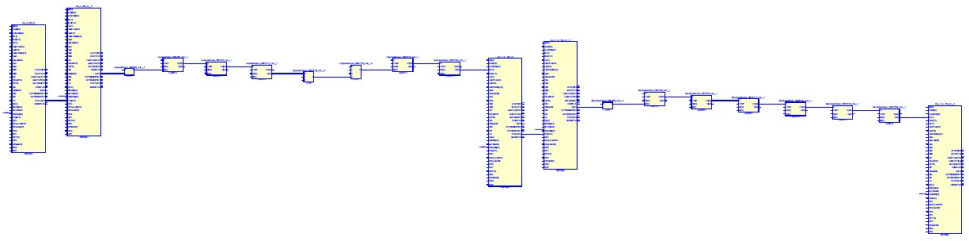


Fig. 3.12 – Critical path dell'implementazione con minima latenza con formato Q8.16 per clock a 48 kHz e 48 MHz

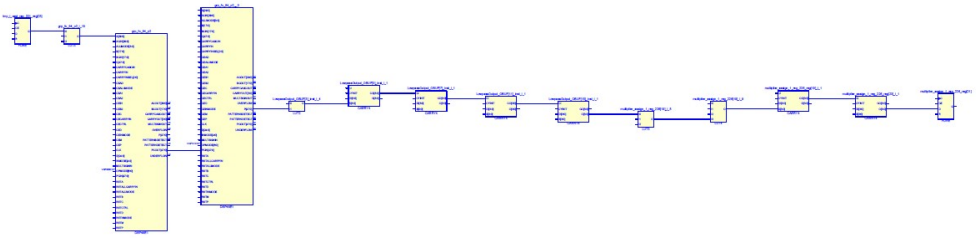


Fig. 3.13 – Critical path dell'implementazione con massima latenza con formato Q8.16 per clock a 48 kHz e 48 MHz

In seguito, si è deciso di limitare il numero di pin di uscita dell'FPGA; quindi si è sviluppato in HLS il progetto del filtro di Chamberlin che fornisce una delle 4 uscite possibili attraverso un selettore, con formato Q8.16 di bit di rappresentazione fixed point per vari valori del clock.
Il filtro così dimensionato riceve in ingresso una sola voce (un valore) e produce in uscita il valore corrispondente in base al settaggio sul selettore.

```
// ChamberlinFilterQ8_16_Selettore.h
#include "ap_cint.h"

void ChamberlinFilter (

    int24 Input,
    uint2 SEL,
    int24 D,      //  D = 1 / Q
    int24 F,     //  F = 2 * sin(M_PI * Fc / Fs)
    int24 *Output
    );
```

Si è esplorato il design space attraverso l'applicazione di direttive di ottimizzazione:

- min latency: non è stata applicata nessuna direttiva, corrisponde al caso in cui viene massimizzato il throughput;
- max latency: si è limitata l'esecuzione di una moltiplicazione per ciclo di clock, corrisponde al caso di massima latenza;
- pipeline: si introduce la struttura della pipeline per l'operazione di moltiplicazione.

Le tabelle 3.8 e 3.10 riportano i risultati delle stime HLS delle metriche di progetto rispettivamente con frequenza di clock pari a 48kHz e 48MHz per le diverse direttive applicate.
Le tabelle 3.9 e 3.11 invece riportano i corrispondenti valori per la struttura sintetizzata.

Per quanto riguarda il numero di DSP utilizzati, si possono osservare le stesse considerazioni fatte nel caso senza selettore: il numero di DSP nel progetto sintetizzato raddoppia.

Tabella 3.8 – Metriche del progetto HLS in formato Q8.16 e Clk = 48kHz con selettore

	Min latency	Max latency	Pipeline
LUT	337	320	337
FF	49	179	49
DSP	3	1	3
Throughput	1	3	1
Critical path delay (ns)	21,85	15,31	25,48

Tabella 3.9 – Metriche del progetto HLS sintetizzato in formato Q8.16 e Clk = 48kHz con selettore

	Min latency	Max latency	Pipeline
LUT	169	220	169
FF	48	147	48
DSP	6	2	6
Throughput	1	3	1
Critical path delay (ns)	21,529	14,305	20,305

Tabella 3.10 – Metriche del progetto HLS in formato Q8.16 e Clk = 48MHz con selettore

	Min latency	Max latency	Pipeline
LUT	352	320	327
FF	138	179	98
DSP	3	1	2
Throughput	2	3	2
Critical path delay (ns)	15,31	15,31	17,12

Tabella 3.11 – Metriche del progetto HLS sintetizzato in formato Q8.16 e Clk = 48MHz con selettore

	Min latency	Max latency	Pipeline
LUT	171	220	195
FF	52	147	75
DSP	6	2	4
Throughput	2	3	2
Critical path delay (ns)	15,159	14,308	15,995

3.3.4.2. Progetto HLS nel formato Q8.24

In seguito, si è eseguito lo stesso studio sulla struttura del filtro con formato Q8.24 di bit di rappresentazione fixed point.
Come nel caso precedente, si è studiata la struttura del filtro di Chamberlin che fornisce tutte 4 le uscite possibili, per vari valori del clock.

```
// ChamberlinFilterQ8_24.h

void ChamberlinFilterQ8_24(
    long int Input,
    long int D,      //  D = 1 / Q
    long int F,    //  F = 2 * sin(M_PI * Fc / Fs)
    long int *LowpassOutput,
    long int *HighpassOutput,
    long int *NotchOutput,
    long int *BandpassOutput
    );
```

In questo caso non è stato necessario inserire la direttiva di inclusione #include “ap_cint.h” dato che il tipo di dato long int è un tipo di dato standard che che verrà sintetizzato in VHDL in un numero a 32 bit.
Di seguito sono riportati i risultati ottenuti per valori di clock estremi considerati più significativi a 48 kHz e a 48 MHz.

Si è esplorato il design space attraverso l’applicazione di direttive di ottimizzazione:

- min latency: non è stata applicata nessuna direttiva, corrisponde al caso in cui viene massimizzato il throughput;
- max latency: si è limitata l’esecuzione di una moltiplicazione per ciclo di clock, corrisponde al caso di massima latenza.

Le tabelle 3.12 e 3.14 riportano i risultati delle stime HLS delle metriche di progetto rispettivamente con frequenza di clock pari a 48kHz e 48MHz per le diverse direttive applicate.
Le tabelle 3.13 e 3.15 invece riportano i corrispondenti valori per la struttura sintetizzata.
Questi valori dimostrano che HLS stima le metriche nel caso peggiore quindi con massima allocazione delle risorse; l'implementazione RTL ottimizza le risorse allocate fornendo i valori effettivi delle prestazioni, dei consumi e dell'occupazione d'area.

Tabella 3.12 – Metriche del progetto HLS in formato Q8.24 e Clk = 48kHz

	Min latency	Max latency
LUT	241	266
FF	65	154
DSP	9	6
Throughput	1	2
Critical path delay (ns)	26.49	17.98

Tabella 3.13 – Metriche del progetto HLS sintetizzato in formato Q8.24 e Clk = 48kHz

	Min latency	Max latency
LUT	310	337
FF	64	130
DSP	12	8
Throughput	1	2
Critical path delay (ns)	24.475	15.572

Tabella 3.14 – Metriche del progetto HLS in formato Q8.24 e Clk = 48MHz

	Min latency	Max latency
LUT	256	266
FF	154	154
DSP	9	6
Throughput	1	2
Critical path delay (ns)	17.98	17.98

Tabella 3.15 – Metriche del progetto HLS sintetizzato in formato Q8.24 e Clk = 48MHz

	Min latency	Max latency
LUT	312	337
FF	98	130
DSP	12	8
Throughput	1	2
Critical path delay (ns)	14.067	15.574

Da notare che sia per entrambi i clock considerati i valori delle risorse occupate non cambiano in modo sostanziale; questo significa che i DSP hanno una potenza di calcolo tale da permettere di avere clock a frequenze più elevate. Si deve arrivare a un valore di clock pari o superiore a 100 MHz per vedere un significativo aumento delle risorse utilizzate e della latenza.

Si nota che rispetto alla stima HLS il numero dei DSP utilizzati nel progetto sintetizzato è moltiplicato per un fattore 4/3.

In seguito, analogamente a quanto fatto nel progetto con formato Q8,16, anche in questo caso si è sviluppato in HLS il progetto del filtro di Chamberlin che fornisce una delle 4 uscite possibili attraverso un selettore, con formato Q8.24 di bit di rappresentazione fixed point per vari valori del clock.
Il filtro così dimensionato riceve in ingresso una sola voce (un valore) e produce in uscita il valore corrispondente in base al settaggio sul selettore.

```
// ChamberlinFilterQ8_24_Selettore.h
#include "ap_cint.h"

void ChamberlinFilter (

    long int Input,
    uint2 SEL,
    long int D,       //  D = 1 / Q
    long int F,     //  F = 2 * sin(M_PI * Fc / Fs)
    long int *Output
    );
```

Di seguito sono riportati i risultati ottenuti.
Per quanto riguarda il numero di DSP utilizzati, si possono osservare le stesse considerazioni fatte nel caso senza selettore: il numero di DSP nel progetto sintetizzato è moltiplicato per un fattore 4/3.

Tabella 3.16 – Metriche del progetto HLS in formato Q8. 24 e Clk = 48kHz con selettore

	Min latency	Max latency	Pipeline
LUT	317	340	317
FF	65	243	65
DSP	9	3	9
Throughput	1	3	1
Critical path delay (ns)	26,49	17,98	30,12

Tabella 3.17 – Metriche del progetto HLS sintetizzato in formato Q8. 24 e Clk = 48kHz con selettore

	Min latency	Max latency	Pipeline
LUT	342	331	342
FF	64	195	64
DSP	12	4	12
Throughput	1	3	1
Critical path delay (ns)	24,475	16,51	24,475

Tabella 3.18 – Metriche del progetto HLS in formato Q8. 24 e Clk = 48MHz con selettore

	Min latency	Max latency	Pipeline
LUT	332	340	327
FF	186	243	130
DSP	9	3	6
Throughput	2	3	2
Critical path delay (ns)	17,98	17,98	17,25

Tabella 3.19 – Metriche del progetto HLS sintetizzato in formato Q8.24 e Clk = 48MHz con selettore

	Min latency	Max latency	Pipeline
LUT	344	331	305
FF	130	195	98
DSP	12	4	8
Throughput	2	3	2
Critical path delay (ns)	17,257	16,408	15,815

3.3.4.3. Precisione del progetto HLS

Fig. 3.14 mostra l'errore sulla risposta all'impulso dell'uscita passa-basso tra il caso floating point e i casi fixed-point considerati.

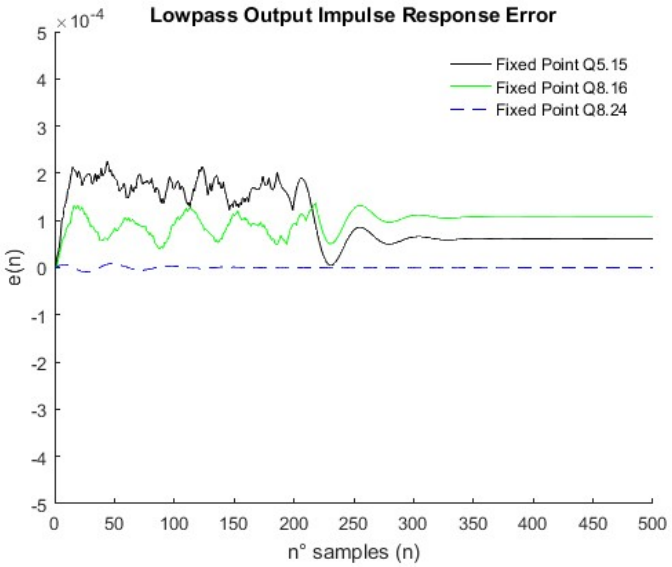


Fig. 3.14 - Errore sulla risposta all'impulso dell'uscita passa-basso tra floating point e fixed-point (Q5.15, Q8.16, Q8.24)

3.3.5. Progetto HLS del filtro di Chamberlin a più voci

Sulla base dell'analisi eseguita sul progetto del filtro di Chamberlin a una voce si è deciso di sviluppare la struttura del filtro con formato Q8.24 in grado di gestire più voci contemporaneamente, ovvero riceve in ingresso più di un valore e produce le uscite corrispondenti.

```
// ChamberlinFilter_multiVoiceQ8_24.h
#include "ap_cint.h"

void CFMVQ8_24(
    long int Input[],
    uint9    SEL[],
    long int D[],    //  D = 1 / Q
    long int F[],    //  F = 2 * sin(M_PI * Fc / Fs)
    long int Output[]
    );
```

Si è esplorato il design space attraverso l'applicazione di direttive di ottimizzazione per diversi valori della dimensione del vettore d'ingresso DIM:

- BUS: è stata applicata la direttiva che implementa l'interfaccia BUS sulle porte I/O;
- FIFO: è stata applicata la direttiva che implementa l'interfaccia FiFo sulle porte I/O;
- AXI: è stata applicata la direttiva che implementa l'interfaccia AXI verso microprocessori;
- LOOP UNROLL: è stata applicata la direttiva che "srotola" il loop, massimizzando il throughput

Di seguito sono riportati i risultati per DIM = 2, 128, 256.

Tabella 3.20 – Metriche del progetto HLS del filtro a più voci DIM = 2

	BUS	FIFO	LOOP UNROLL
LUT	573	590	983
FF	679	296	328
DSP	9	9	18
BRAM	0	0	0
Throughput	13	7	4
Critical path delay (ns)	21.289	30,13	33,76

Tabella 3.21 – Metriche del progetto HLS del filtro a più voci DIM = 128

	FIFO	LOOP UNROLL
LUT	493	23576
FF	33	2955
DSP	9	7
BRAM	0	9
Throughput	258	259
Critical path delay (ns)	33,76	42,02

3.4. Implementazione di FPGA

Per l'implementazione si è scelto di utilizzare ZedBoard Zynq Evaluation and Development Kit (Famiglia Zynq-7000) con chip FPGA xc7z020clg484-1 di Xilinx. Il dispositivo è mostrato in Fig. 3.15.
Si è implementato il progetto completo di un sintetizzatore audio funzionante che integra blocchi progettati a basso livello sia il filtro di Chamberlin Q8.16 realizzato tramite sintesi HLS, secondo lo schema riportato in Fig. 3.16.

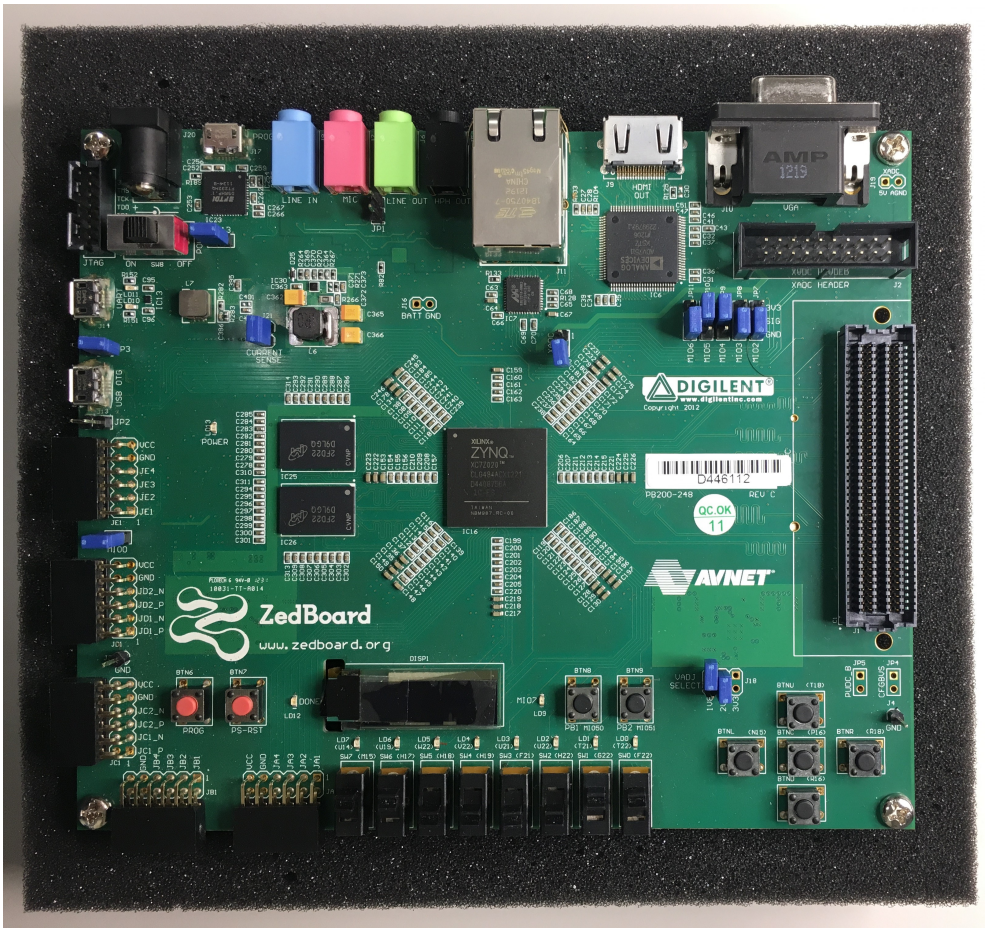


Fig. 3.15 – Zedboard Zynq Evaluation and Development Kit

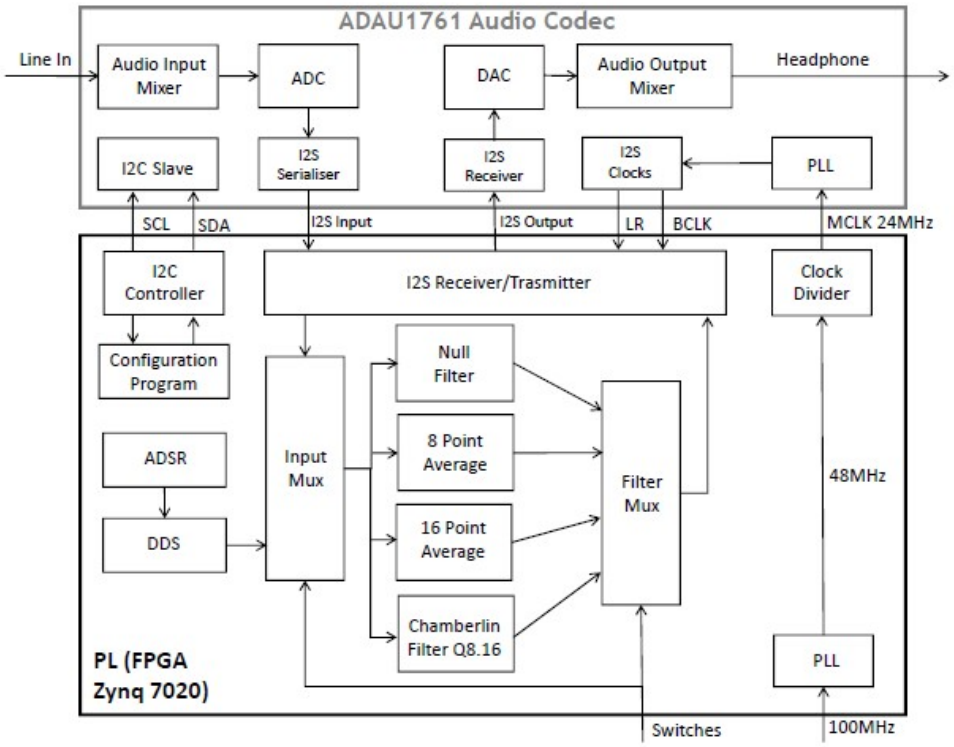


Fig. 3.16 – Schema a blocchi del sintetizzatore audio digitale implementato su FPGA

Di seguito la descrizione delle modalità implementative di ogni singolo blocco.

Il blocco ADSR è un generatore di inviluppo. La sigla ADSR è composta dalle iniziali dei parametri definibili su questo tipo di generatore di inviluppo (Attack-Decay-Sustain-Release). Genera in uscita un segnale come quello raffigurato in Fig. 3.17, che è utilizzato per modificare i segnali di controllo. È usato principalmente per agire sul settaggio l'amplificatore, modellando in questo modo l'inviluppo della nota suonata secondo i parametri desiderati.

Appena un tasto viene premuto, due segnali di controllo partono dalla tastiera:

1. Il segnale Pitch che rappresenta il valore della frequenza della nota suonata e quindi comunica al generatore di suono la frequenza del segnale fondamentale che deve produrre;
2. il segnale Gate che è un segnale ON-OFF che agisce sull'ADSR.

Avendo ricevuto il segnale Gate a ON, l'ADSR comincerà la fase di attacco emettendo un segnale crescente, fino ad arrivare al livello massimo in un tempo corrispondente al parametro Attack. A questo punto ha inizio la fase di decadimento: il segnale emesso passa,

in un tempo pari al parametro Decay, dal valore massimo al valore indicato dal parametro Sustain. A questo punto l'ADSR permane nella fase di Sustain fintanto che il segnale Gate è ancora attivo (ON). Alla disattivazione del segnale di Gate, ovvero quando il tasto sulla tastiera viene rilasciato (Gate OFF), inizia la fase di rilascio: il segnale emesso dall'ADSR passa dal valore di Sustain a zero in un tempo pari al parametro Release.
Si noti che la fase di rilascio ha sempre inizio quando il segnale di Gate passa dallo stato ON a OFF, anche nell'eventualità in cui le altre fasi non si siano ancora concluse o non siano ancora iniziate.

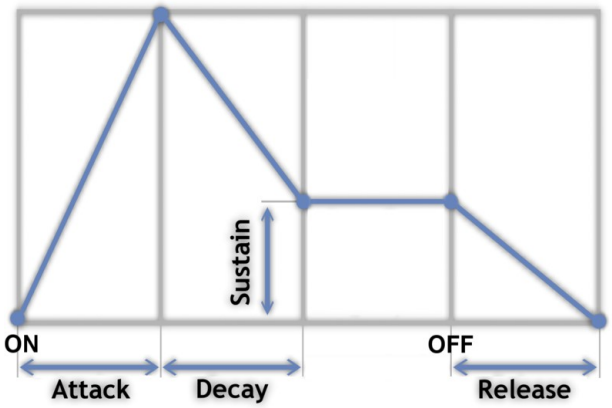


Fig. 3.17 – Segnale di controllo in uscita da un blocco ADSR

Fig.3.18 mostra l'Algorithmic State Machine Diagrams (ASMD) che schematizza il funzionamento del blocco ADSR.
Ogni stato del ASMD (Idle, Lunch, Attack, Decay, Sustain, Release), controllato dal segnale di Gate, agisce sul segnale z di uscita del blocco ADSR.
Quando il segnale di Gate assume valore logico basso in qualsiasi stato che non sia Idle o Launch il circuito passa nello stato Release.

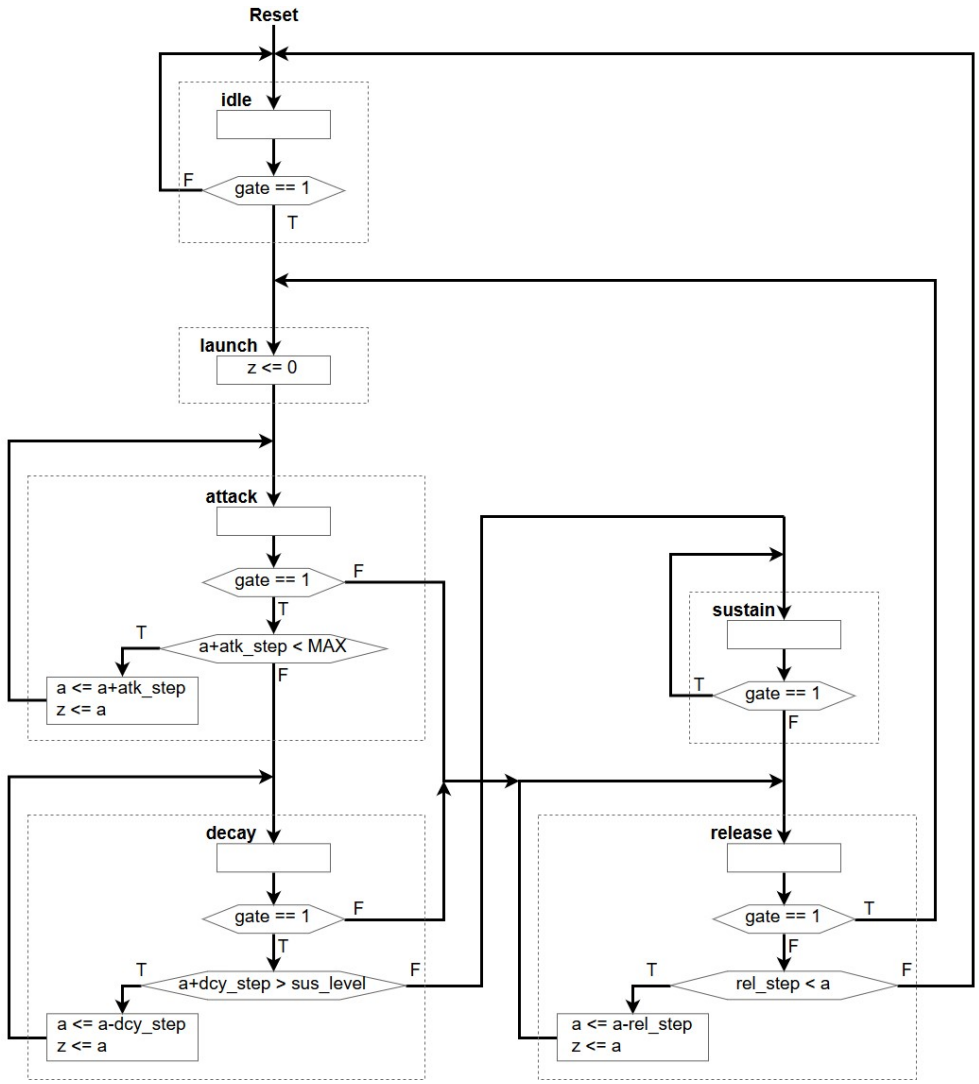


Fig. 3.18 – ASMD del blocco ADSR

Quando il segnale Pitch viene ricevuto dal blocco DDS, questo attiva i segnali di enable per la lettura dei campioni in memoria.
Nel blocco ROM sono stati memorizzati i campioni di un segnale sinusoidale con frequenza pari a 600 Hz nel formato Q8.16 campionato ad una frequenza f_s = 48kHZ. A partire da questa fondamentale il blocco DDS genera in uscita un segnale sinusoidale alla frequenza desiderata. Si è testato il sistema per valori del Pitch pari a 523 Hz (DO4) e 440 Hz (LA4).

Le uscite del blocco DDS e ADSR vengono moltiplicate tra loro tramite un moltiplicatore a 16 bit.
In seguito, il segnale risultante viene inviato in ingresso al set di filtri attivi secondo la configurazione indicata dagli switch che riproducono i comandi che dovrebbero arrivare dal microcontrollore.
Infine, il segnale di uscita del filtro viene inviato in uscita tramite il protocollo I2S il codec audio ADAU1761 di Analog Devices integrato sulla board Zedboard.

Capitolo 4.

Modellazione ad alto livello di sistemi eterogenei

Si è scelto lo studio delle batterie al litio e degli algoritmi di BMS (Battery Management System) come caso applicativo della metodologia di modellizzazione ad alto livello di sistemi eterogenei.
SystemC-WMS è stato usato per lo sviluppo di un ambiente di simulazione per emulare il funzionamento di un pacco batterie agli ioni di litio.
Le batterie agli ioni di litio sono state ampiamente utilizzate nei veicoli elettrici moderni, a causa del basso tasso di scarica e delle migliori prestazioni rispetto alle batterie tradizionali. Infatti, la tecnologia agli ioni di litio offre maggiore energia e densità di potenza.
L'affidabilità, l'efficienza e le condizioni di funzionamento sicure delle batterie agli ioni di litio richiedono monitoraggio, controllo e gestione costanti.
Il Battery Management System (BMS) protegge la batteria dai danni e la mantiene in condizioni operative sicure. Il BMS stima inoltre lo stato di carica (State-of-Charge - SoC), lo stato di salute (State-of-Health - SoH) e la vita utile residua della batteria. La durata e le prestazioni della batteria dipendono dal profilo di carica e scarica, dalle condizioni termiche e dalla frequenza di ricarica.
Inoltre, in molte applicazioni dove è richiesta alta tensione o corrente, le singole batterie sono collegate in combinazioni serie-parallelo, e vengono chiamate “celle” di un pacco batteria. A causa delle variazioni nel processo di produzione, nello stesso pacco ogni cella differisce dalle altre. Quindi è possibile che le celle dello stesso pacco batteria non raggiungano la piena carica nello stesso momento durante la fase di carica e abbiano diversi valori del SoC durante la fase di scarica. Il BMS si occupa anche del bilanciamento delle celle.
Le variazioni dei parametri interni della batteria (capacità nominale, SoC, resistenza interna, parametri termici) possono essere molto elevate e influire sulle prestazioni della batteria [38].
Quindi un modello elettro-termico accurato di una singola cella e un modello della variabilità statistica della singola cella in un pacco batteria sono fondamentali per la definizione di un efficiente algoritmo BMS.
L'ambiente SystemC-WMS consente, in modo estremamente semplice, la definizione e la simulazione di un pacco batteria con molte celle in serie e in parallelo considerando variazioni casuali della capacità nominale, del SoC e dei parametri termici ed elettrici delle singole celle. L'ambiente di simulazione può essere utilizzato per una simulazione statistica di un pacco batterie e per l'analisi dell'effetto di diverse strategie BMS di bilanciamento delle celle su un pacco batterie, considerando le variazioni statistiche delle celle.

4.1. Modello elettrotermico di una singola cella di una batteria agli ioni di litio

4.1.1. Modello Elettrico

La parte elettrica del modello proposto si basa sullo schematico in Fig. 4.1b.

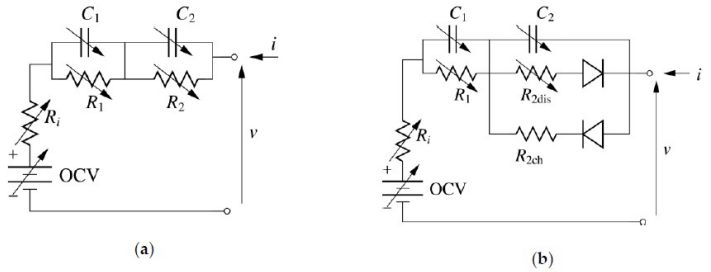


Fig. 4.1 – (a) Schematico classico del circuito che modella la parte elettrica di una cella al litio; (b) Schematico modificato in base ai riscontri sperimentali

Il modello elettrico della singola cella è descritto da un generatore ideale di tensione *OCV* (Open-Circuit Voltage), che rappresenta la tensione a circuito aperto della cella, connesso in serie alla resistenza interna R_i e in parallelo a due capacitori e due resistori (R_1, C_1, R_2, C_2).
A causa del comportamento non simmetrico della cella durante la carica e scarica, notato durante le misurazioni sperimentali, sono stati usati diversi valori di R_2 per la fase di carica e scarica, considerando due diversi valori di $R2$ durante le fasi di carica e scarica: R_{2ch}, R_{2dis} (Fig. 4.1b). R_{2ch} è un valore costante trascurabile rispetto a R_1, quindi il transitorio di $R_{2ch}C_2$ è molto più veloce rispetto a R_1C_1.
Il circuito è descritto dalle seguenti equazioni:

$$\begin{cases} \dot{v}_1 = \dfrac{i}{C_1} - \dfrac{v_1}{R_1 C_1} \\ \dot{v}_2 = \dfrac{i}{C_2} - \dfrac{v_2}{R_2 C_2} \end{cases} \quad . \qquad (4.2)$$
$$v = iR_i + v_1 + v_2 + \mathrm{OCV}$$

dove v_1 e v_2 sono delle variabili di stato delle celle.
Il comportamento della parte elettrica della cella, descritta dall'eq. (4.2), può essere riscritta in SystemC-WMS in termini dei parametri *a* e *b* come segue:

$$\begin{cases} \dot{v}_1 = \dfrac{1}{R_1 C_1}\left(2a\dfrac{R_1\sqrt{R_0}}{R_i+R_0} - v_1\dfrac{R_1+R_i+R_0}{R_i+R_0} - (v_2+\mathrm{OCV})\dfrac{R_1}{R_i+R_0}\right) \\ \dot{v}_2 = \dfrac{1}{R_2 C_2}\left(2a\dfrac{R_2\sqrt{R_0}}{R_i+R_0} - v_2\dfrac{R_2+R_i+R_0}{R_i+R_0} - (v_1+\mathrm{OCV})\dfrac{R_2}{R_i+R_0}\right) \end{cases} \qquad (4.3)$$
$$b = \frac{1}{R_i+R_0}\left(a(R_i - R_0) + (v_1 + v_2 + \mathrm{OCV})\sqrt{R_0}\right)$$

dove R_0 è la resistenza normalizzata utilizzata per la conversione tra tensione/corrente e onda incidente/riflessa.

4.1.2. Procedura di estrazione dei parametri elettrici della cella

A partire dalle misurazioni, si è caratterizzato il comportamento deterministico di una cella agli ioni di litio. I parametri elettrici della cella (OCV, R_i, R_1, C_1, R_2, C_2) hanno una dipendenza non lineare dalla temperatura T della cella e dallo State-of-Charge (SoC).
Esistono diverse metodologie di stima dei parametri di un modello elettrico a parametri concentrati. L'algoritmo utilizzato per l'estrazione dei parametri del modello elettrico in funzione di T e di SoC si basa sulla procedura Hybrid Pulse Power Characterization (HPPC), più usato in letteratura [39, 40, 41].
La batteria della cella è forzata da una sequenza di impulsi di carica e scarica. Partendo da una batteria completamente carica (SoC = 100%), viene prelevata una corrente costante dalla cella per il tempo necessario a ridurre del 10% il valore di SoC. Quindi la corrente viene azzerata fino a raggiungere uno stato stazionario. La procedura viene ripetuta fino a quando SoC raggiunge il 10% del valore nominale.
Il valore di SoC è ottenuto dalla seguente relazione:

$$\mathrm{SoC}_{\%} = \left(\mathrm{SoC}_{\mathrm{init}} - \frac{1}{\mathrm{Q}_{\mathrm{nom}}} \int_0^t \mathrm{i}(\tau)\, \mathrm{d}\tau\right) \tag{4.4}$$

dove Q_{nom} è il valore della carica nominale della cella e SoC_{init} è il valore iniziale di SoC. Quindi la batteria è completamente carica con una corrente costante.
In Fig. 4.2 sono riportati la forma della corrente di ingresso (in alto a sinistra), il comportamento tipico della tensione della cella (in basso a sinistra) e i dettagli delle fasi di carica e scarica (lato destro). V1, V3 e V6 sono i valori di tensione della cella negli stati stazionari; V2 e V4 sono i valori della tensione dopo le fasi di discesa e di salita della corrente di ingresso. V5 è il valore della tensione di uscita in un breve tempo (dt) dopo la fase di salita.

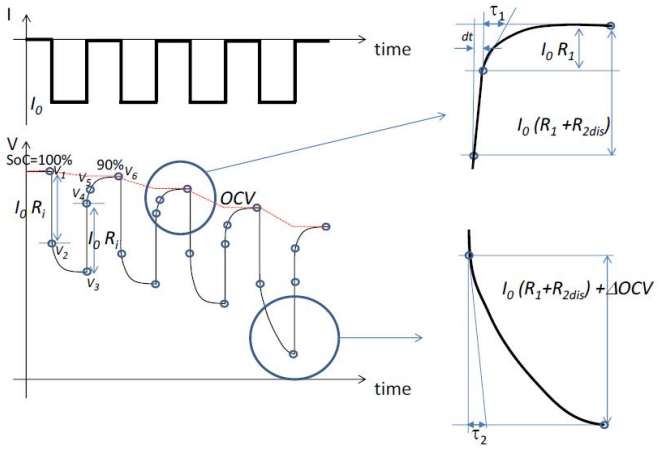


Fig. 4.2 - Procedura di estrazione dei parametri elettrici

Il valore di *OCV* è ottenuto dai valori della tensione negli stati stazionari per *SoC* = 100%, 90%, 80%, ..., 10%.
Un impulso della corrente d'ingresso causa una caduta di tensione istantanea su R_i. Da qui si ottengono i valori di R_i per *SoC* = 100%, 90%, 80%, ..., 10%:
R_i (*SoC* = 100%) = (V1 – V2) / I_0; R_i (*SoC* = 90%) = (V4 – V3) / I_0; ...
I valori di R_1, C_1, R_2, C_2 non possono essere ottenuti facilmente. $\tau_2 = R_2C_2$ è minore di $\tau_1 = R_1C_1$, quindi il primo transitorio più breve è dovuto alla scarica di C_2; mentre il rimanente è dovuto alla scarica di C_1. Pertanto, si possono stimare i valori di R_1 e C_1 e R_2 e C_2 come segue:

$$R_1(\mathrm{SoC}=90\%)=\frac{(V_6-V_5)}{I_0} \qquad R_{2\,\mathrm{dis}}(\mathrm{SoC}=90\%)=\frac{(V_2-V_3)-\Delta\mathrm{OCV}}{I_0}-R_1$$

$$\tau_1(\mathrm{SoC}=90\%)=\frac{(V_6-V_5)}{\left.\frac{dV}{dt}\right|_{V_5}} \qquad \tau_2(\mathrm{SoC}=90\%)=\frac{(V_3-V_2)}{\left.\frac{dV}{dt}\right|_{V_2}} \tag{4.5}$$

$$C_1=\frac{\tau_1}{R_1} \qquad C_2=\frac{\tau_2}{R_2}$$

dove ΔOCV (*SoC* = 90%) = *OCV* (*SoC* = 100%) - *OCV* (*SoC* = 90%).

La procedura di estrazione dei parametri appena descritta è stata applicata per il modello di una cella LFP (lithium ferrophosphate) con capacità nominale 50Ah, tensione nominale 3,2 V, dimensioni 37x101x192 mm, limite di tensione di scarica 2,5 V, limite di tensione di carica 3,8 V. Il profilo corrente, leggermente modificato rispetto al classico HPPC, è riportato in Fig. 4.3. Una corrente costante di 25A viene prelevata dalla batteria per 720 s, corrispondente a una carica di 25 * 720/3600 = 5 Ah che è 10% della capacità nominale della cella. Dopo un periodo di riposo di 300 s vengono applicati due impulsi di scarica e carica consecutivi per un valore di corrente pari a 25A. Quindi, dopo un altro periodo di riposo di 300 vengono applicati due impulsi successivi di scarica e carica per un valore di corrente pari a 75. Gli impulsi sono utilizzati per verificare l'indipendenza di R_i con il valore della corrente applicata. Non modificano il *SoC*. Il ciclo viene ripetuto per 9 volte, in modo che il *SoC* raggiunga il 10% della capacità nominale. Infine, una corrente costante di 50 A viene applicata fino alla ricarica completa della batteria.

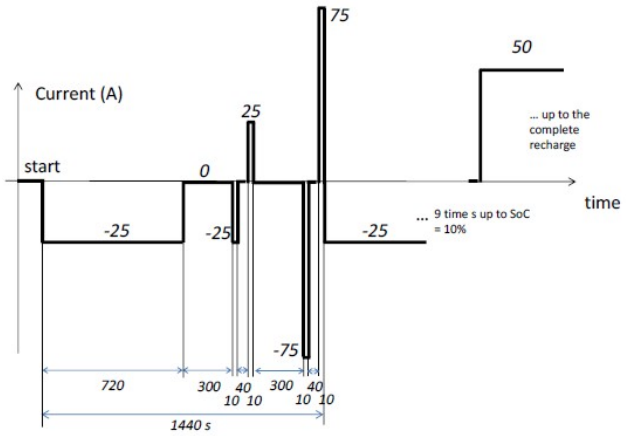


Fig. 4.3 – Profilo di corrente utilizzato per la caratterizzazione della cella

L'estrazione dei parametri è stata effettuata mantenendo costante la temperatura a 0°C, 25°C e 40°C. In questo modo, si sono ottenuti i valori dei parametri *OCV*, R_i, R_1, C_1, R_{2ch}, R_{2dis}, C_2 per diversi valori della temperatura *T* e di *SoC*. A partire da questi valori, viene ricavato un modello bidimensionale (in funzione di *SOC* e della temperatura) lineare a tratti, riducendo al minimo l'errore tra il modello e le misure sperimentali.
Fig. 4.4 (a) - (f) riportano in punti i valori dei parametri *OCV*, R_i, R_1, C_1, R_{2dis}, C_2 ottenuti dalla procedura di estrazione, e in linee continue il modello bidimensionale lineare a tratti dopo l'ottimizzazione. L'*OCV* (Fig. 4.4a) ha la tipica forma non lineare: aumenta rapidamente quando il *SOC* è vicino al 100% e diminuisce quando il *SOC* è basso. L'*OCV* aumenta leggermente con la temperatura. Le resistenze interne R_i, R_1, R_2 aumentano con la temperatura, compatibilmente con quanto mostrato in altri lavori [42, 43].
L'incremento delle resistenze in concomitanza con la riduzione dell'*OCV* provoca l'effetto ben noto della riduzione dell'efficienza della batteria a bassa temperatura. R_{2dis} è trascurabile a 25°C e 40°C, come mostrato in Fig. 4.4d. Al contrario, il suo valore aumenta durante la scarica a 0°C quando il *SOC* è basso.
Il valore di R_{2dis} è basso durante la fase di carica finale. Per questo motivo, si è utilizzato il modello in Fig. 4.1b con i diodi invece del modello classico riportato in Fig. 4.1a. Questo effetto è stato discusso anche in [44, 45, 46] con isteresi in *OCV*.

Fig. 4.4c mostra che la procedura di estrazione (punti) evidenzia un incremento di R_1 a 0°C quando il *SoC* è basso, in modo simile a R_2. Nel modello finale lineare a tratti ottimizzato, questo effetto è attribuito solo a R_2.
Fig. 4.4e-f mostrano che le costanti di tempo τ_1 e τ_2 hanno una grande variabilità irregolare in funzione della temperatura e di *SOC* nella stima della procedura di estrazione (punti). La procedura di estrazione non è in grado di discriminare facilmente tra i due contributi di R_1C_1 e $R_{2dis}C_2$. La successiva procedura di ottimizzazione garantisce un andamento più regolare in funzione della temperatura e di *SoC*.

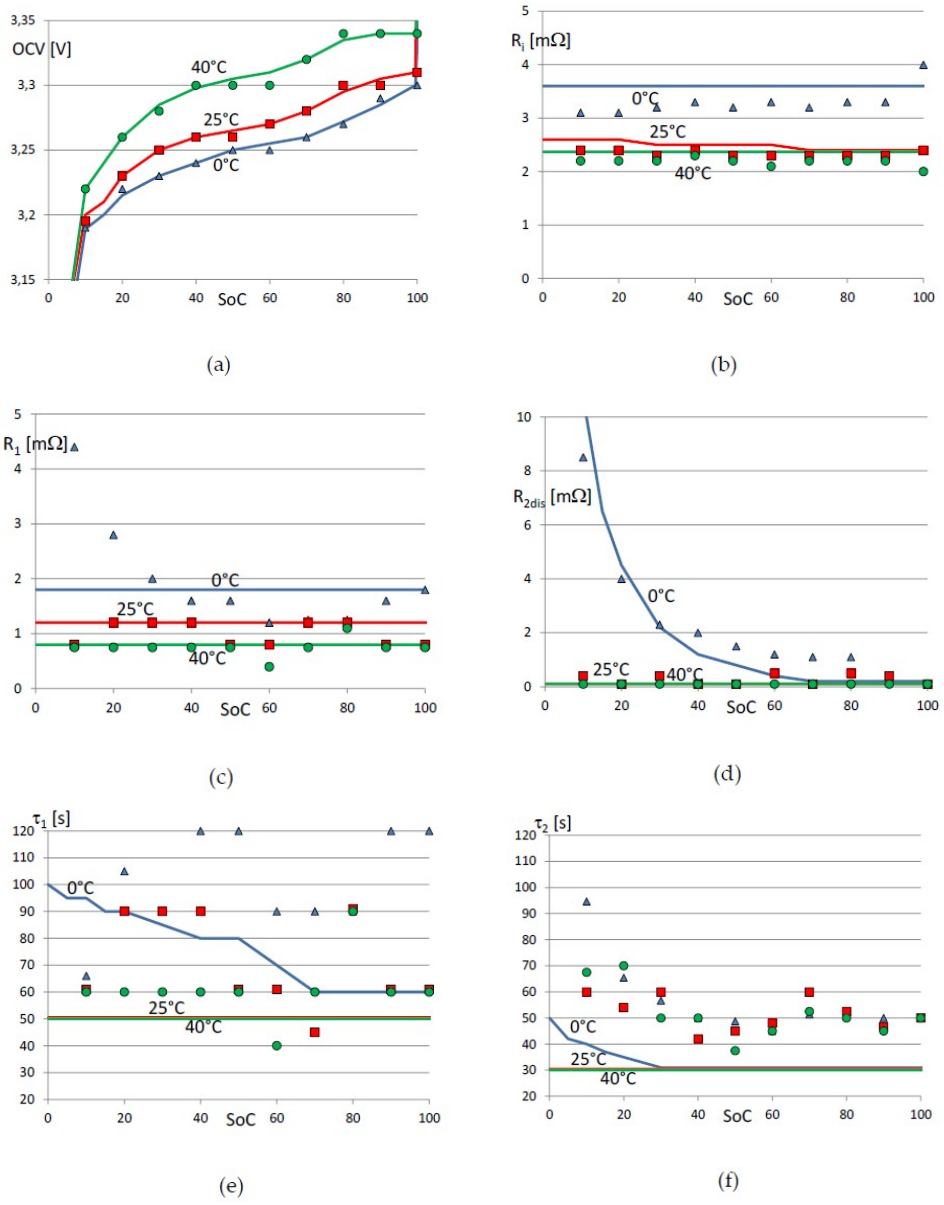


Fig. 4.4 – Valori dei parametri elettrici della cella: dopo la procedura di estrazione (punti) e dopo la successiva ottimizzazione: (a) *OCV*; (b) R_i; (c) R_1; (d) R_{2dis}; (e) τ_1; (f) τ_2. Tutti i parametri sono in funzione di *SoC*

4.1.3. Modello termico

Un modello termico a parametri concentrati è stato usato per descrivere il comportamento termico della singola cella. È ottenuto dall'equilibrio energetico tra la dissipazione del calore per effetto Joule e la conduzione termica, come indicato dalle seguenti equazioni:

$$G_{in}(T_{in} - T_{ext}) + C\ \frac{dT_{in}}{dt} = R_{th}i^2 \tag{4.6}$$

$$G_{in}(T_{in} - T_{ext}) = G_{conv}(T_{ext} - T_{amb}) \tag{4.7}$$

dove T_{in} è la temperatura interna della cella, T_{ext} è la temperatura sulla superficie della cella, T_{amb} è la temperatura ambiente, C è la capacità termica interna della cella, $G_{in} = kA/L$ è la conduttanza termica interna (k è la conduttività termica, L è la lunghezza di conduzione, $A = WH$ è l'area di convezione/conduzione), $G_{conv} = hA$ tiene conto del trasferimento di calore per convezione, (h è il coefficiente di trasferimento termico convettivo) e $R_{th}i^2$ è la portata di calore interna generata.

In questo modello si è assunto che la generazione di calore della batteria è dominata dal calore ohmico generato per effetto Joule dalle resistenze interne della batteria. Dato che questo termine è proporzionale al quadrato della corrente di carico, la generazione di calore della batteria è modellata come $R_{th}i^2$, dove R_{th} è la somma delle resistenze interne della cella (R_i, R_1, R_2).

Si considera anche che la temperatura interna della cella sia uniforme e che la generazione di calore sia uniformemente distribuita all'interno della cella. Si presume che la conduzione del calore e la convezione siano le uniche forme di trasferimento di calore tra la cella interna e l'ambiente. Si assume inoltre che la quantità di flusso di calore trasferita per effetto di conduzione è uguale a quella assorbita dalla convezione, come indicato dall'eq. (4.7). Nel modello della singola cella, la resistenza termica del case della batteria si considera trascurabile; mentre verrà introdotta nelle simulazioni del pacco batteria, per modellare lo scambio di calore tra celle vicine, quindi l'eq. (4.7) sarà modificata di conseguenza.

La Fig. 4.5 riporta la rappresentazione geometrica di una cella con un circuito elettro-termico equivalente che implementa l'eq. (4.6), (4.7); mentre la Fig. 4.6 confronta lo stesso circuito con una rappresentazione schematica del modello nell'ambiente di simulazione SystemC-WMS in termini dei parametri a e b.

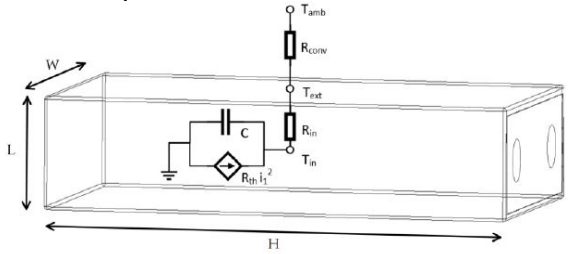


Fig. 4.5 – Rappresentazione termica e geometrica di una singola cella

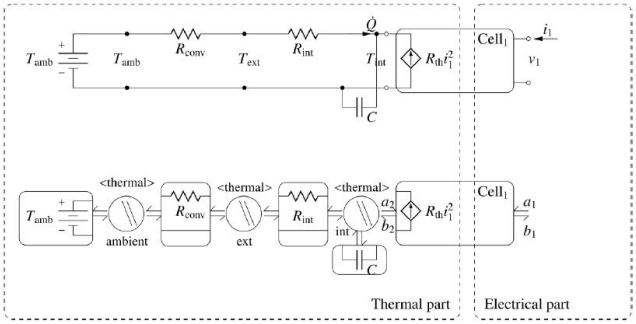


Fig. 4.6 – Modello termico di una singola cella (sopra) e implementazione in SystemC-WMS (sotto)

La tabella 4.1 riporta i valori dei coefficienti termici e parametri geometrici usati per la simulazione della singola cella.

Tabella 4.1 – Parametri fisici e termici della cella

Massa M	1.4 kg
Dimensioni L, W, H	(37, 101, 192) mm
Conduttività termica k	1.11 W/m K
Capacità termica specifica c_p	830 J/kg K
Resistenza termica interna R_{in}	1.735 K/W
Capacità termica interna C	1162 J/K
Coefficiente di trasferimento termico convettivo h	38.953 W/ m^2 K

4.1.4. Risultati

La precisione del modello è mostrata nelle Fig. 4.7-4.9. Riportano i valori di *SoC*, della tensione misurata e simulata in SystemC-WMS e Simulink in riferimento al modello della cella di Fig. 4.1b con i parametri ottenuti con la procedura di estrazione e successiva ottimizzazione. La temperatura della cella è stata forzata a una temperatura costante, rispettivamente a 0°C, 25°C e 40°C, in camera climatica.
Le simulazioni di SystemC-WMS e Simulink sono molto vicine tra loro; la differenza è dovuta principalmente al fatto che il modello Simulink non considera la parte termica ma, d'altra parte, il valore della resistenza convettiva è basso per simulare la camera climatica costante.
Inoltre, si evidenzia che la tensione di uscita della cella a basse temperature è inferiore rispetto a 25°C e 40°C, specialmente quando il *SoC* è basso.

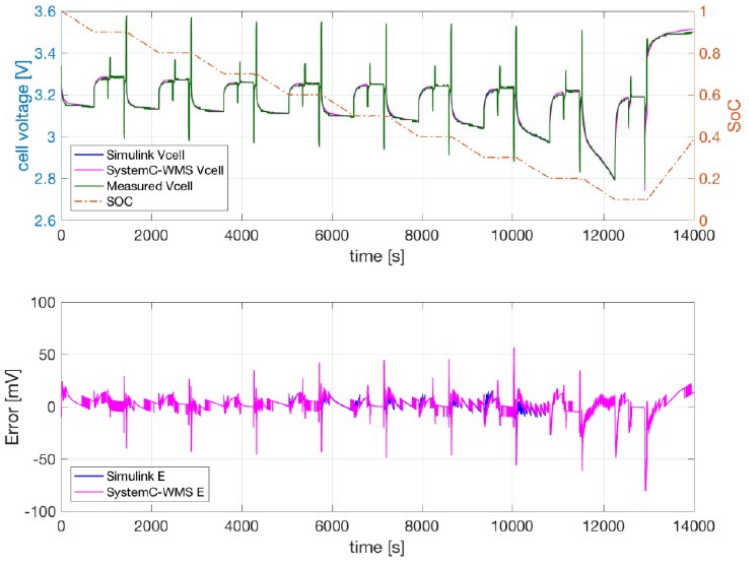


Fig. 4.7 – *SoC* e tensione della cella misurata e simulata tramite Simulink e SystemC-WMS con parametri ottenuti dalla procedura di estrazione e successiva ottimizzazione a $T = 0°C$ (sopra); errore tra i valori di tensione misurati e quelli simulati in Simulink e SystemC-WMS (sotto)

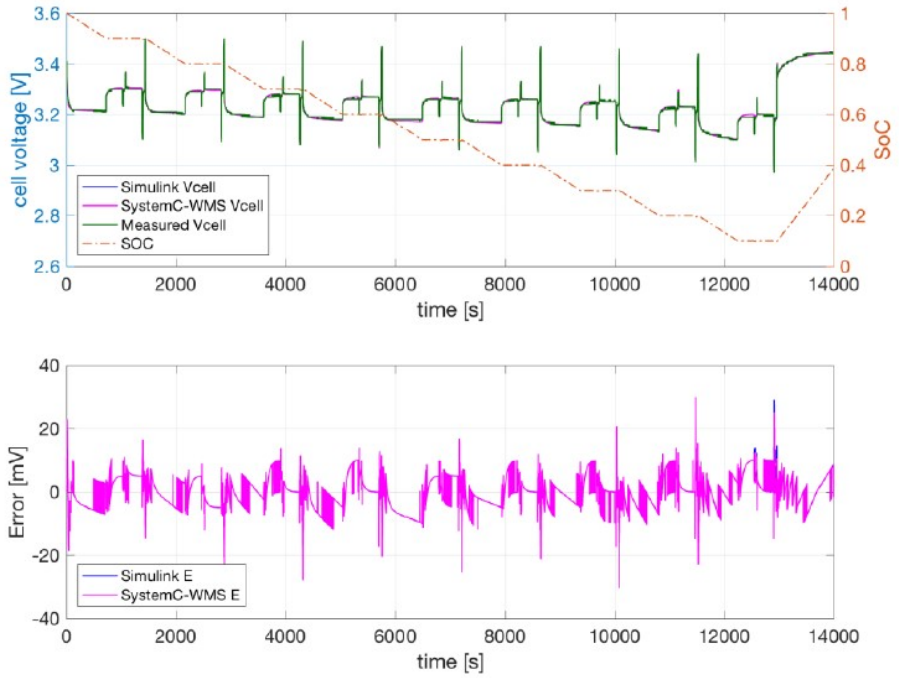


Fig. 4.8 – *SoC* e tensione della cella misurata e simulata tramite Simulink e SystemC-WMS con parametri ottenuti dalla procedura di estrazione e successiva ottimizzazione a $T = 25°C$ (sopra); errore tra i valori di tensione misurati e quelli simulati in Simulink e SystemC-WMS (sotto)

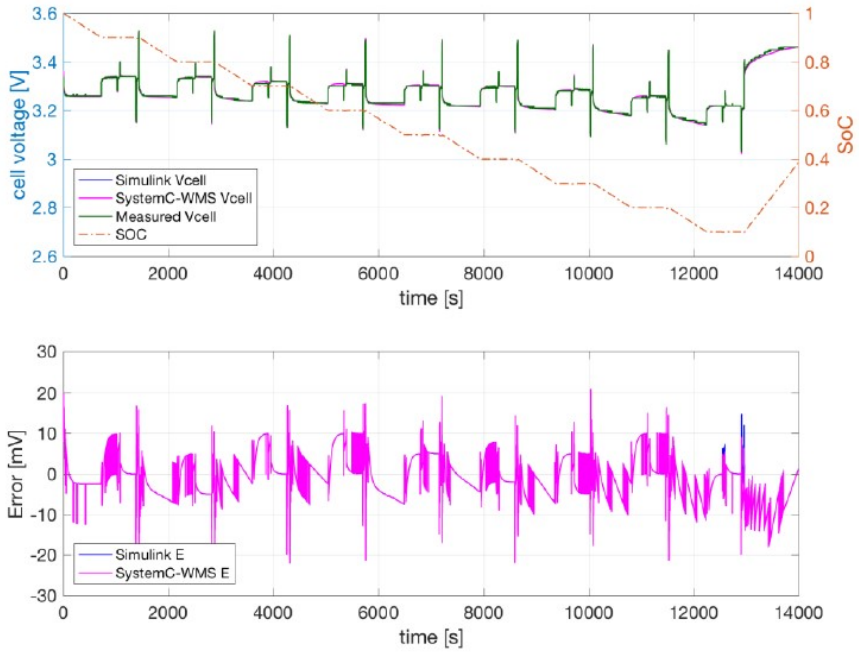


Fig. 4.9 – *SoC* e tensione della cella misurata e simulata tramite Simulink e SystemC-WMS con parametri ottenuti dalla procedura di estrazione e successiva ottimizzazione a $T = 40°C$ (sopra); errore tra i valori di tensione misurati e quelli simulati in Simulink e SystemC-WMS (sotto)

4.2. Modello elettrotermico di un pacco batteria agli ioni di litio

Un pacco batteria è in genere composto da un gruppo di celle dello stesso lotto di produzione, collegate tra loro in configurazioni serie o parallelo.
L'effetto dello sbilanciamento tra le celle causa il degrado delle prestazioni del pacco batteria. Durante la carica della batteria, OCV differenti in celle collegate in serie potrebbero portare al sovraccarico delle celle con OCV più alto. Analogamente, le celle con OCV più basso possono essere sovra-scaricate in fase di scarica.
Un problema duale è presente anche quando le celle sono collegate in parallelo: le celle sono forzate ad avere la stessa tensione ma il loro valore di SoC potrebbe essere diverso. Il risultato è che alcune celle raggiungono un SoC dello 0% mentre altre hanno ancora carica disponibile.

In ambiente SystemC-WMS è stato sviluppato il modello di una batteria complessa con collegamento in serie M moduli costituiti da N celle connesse in parallelo.
Il numero di celle in serie e in parallelo può essere modificato semplicemente cambiando i parametri M e N. Fig. 4.10 mostra un modulo costituito da celle collegate in parallelo.

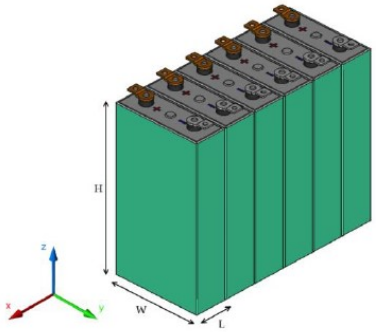


Fig. 4.10 – Modulo del pacco batteria costituito da 6 celle connesse in parallelo

In questo caso, il calore generato da ciascuna cella viene propagato nelle celle adiacenti e viene trasferito verso la superficie per conduzione; quindi le resistenze termiche R_{cnt} tra le celle non possono essere trascurate.
Pertanto, l'eq. (4.6), (4.7) sono sostituite dall'eq. (4.8):

$$G_{\mathrm{conv}i}(T_{\mathrm{ext}i} - T_{\mathrm{amb}}) + \sum_{j} G_{\mathrm{cnt}j}(T_{\mathrm{ext}i} - T_{\mathrm{ext}j}) = \\ = G_{\mathrm{int}i}(T_{\mathrm{int}i} - T_{\mathrm{ext}i}) + C\frac{\mathrm{d}T_{\mathrm{int}i}}{\mathrm{d}t} = R_{\mathrm{th}}i^2 \tag{4.8}$$

dove il pedice i indica il parametro relativo alla i-esima cella del pacco batteria; mentre G_{cntj} è la conduttanza di contatto con la j-esima cella adiacente.
Il valore di della G_{conv} cambia a seconda che la cella considerata sia interna o esterna al modulo, come mostrato in Fig. 4.11.
Per le celle esterne del pacco batteria, $G_{conv} = G_{conv} = hA_2$ dove A_2 è l'area del radiatore data dall'area periferica più l'area della sezione trasversale della cella. Per le celle interne invece batteria $G_{conv} = G_{conv1} = hA_1$ è l'area periferica della cella.
La Tabella 4.2 riporta i valori dei coefficienti termici e dei parametri geometrici, utilizzati per la simulazione del pacco batteria, oltre ai parametri della singola cella, riportati nella Tabella 4.1.

Tabella 4.2 – Parametri fisici e termici del pacco batteria

Area della sezione trasversale della cella $A = WH$	0.019 m^2
Area periferica $A_1 = 2(W + H)L$	0.022 m^2
Area periferica $A_2 = A_1 + A$	0.041 m^2
Conduttività termica del case k_s [39]	58 W/m K
Resistenza termica di conduzione $R_{cnt} = d/k_s A$	0.003 K/W
Resistenza termica di convezione $R_{conv1} = 1/ hA_1$	1.184 K/W
Resistenza termica di convezione $R_{conv2} = 1/ hA_2$	0.625 K/W

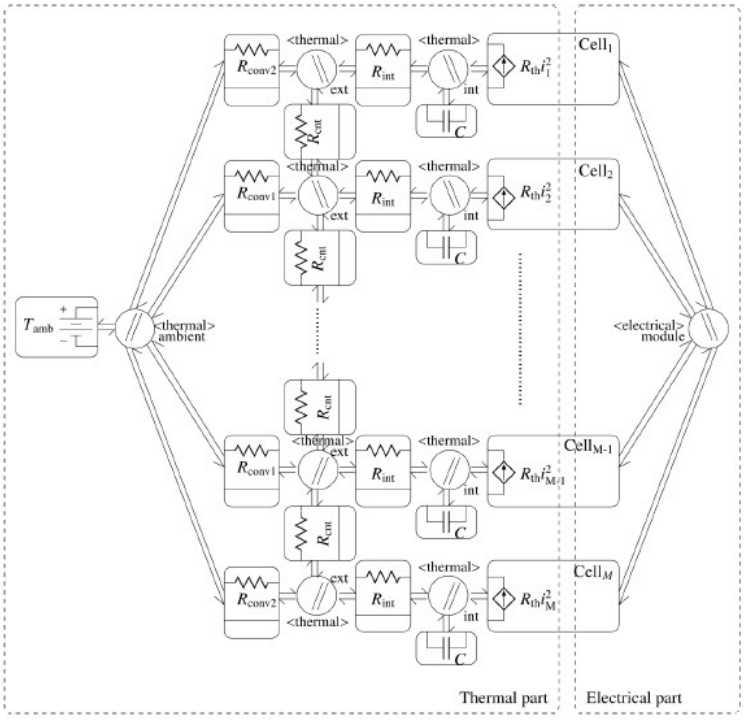


Fig. 4.11 – Schematico SystemC-WMS di un modulo del pacco batteria

4.2.1. Simulazione e analisi statistica della variazione dei parametri interni della cella

Uno degli aspetti critici dell'uso e della gestione di pacchi batteria agli ioni di litio sono le variazioni statistiche delle caratteristiche elettrotermiche e chimiche delle singole celle.
Nonostante la rilevanza del problema, non esiste in letteratura un ambiente di simulazione statistica in grado di verificare l'efficienza di strategie di BMS che affrontano il problema del mismatch della tensione delle singole celle.

Nell'ambiente di simulazione sviluppato, l'effetto di sbilanciamento delle celle viene considerato aggiungendo un valore casuale costante, con distribuzione gaussiana a valore medio nullo e valori diversi della deviazione standard definiti dall'utente, agli 8 parametri della cella (*OCV, R_i, R_1, C_1, R_2, C_2, Q_{nom}, SoC_{init}*).
Le variazioni statistiche dei parametri della cella possono andare dall'1% al 10% [47, 48, 49, 50], ma il designer può misurare i parametri delle singole celle prima di connetterle nel pacco batteria in serie o parallelo. Pertanto, può selezionare le celle per ridurre le variazioni.
L'analisi dell'effetto dello sbilanciamento delle celle sulle prestazioni della batteria è utile per definire le specifiche sul massimo mismatch consentito.

Il modello elettrotermico in SystemC-WMS del pacco batteria che include variazioni statistiche è stato utilizzato per effettuare una simulazione di un modulo con 6 celle collegate in parallelo. La capacità media di ciascuna cella è di 50 Ah; quindi, la capacità nominale del modulo è di 300 Ah con una tensione nominale di 3,2 V. La temperatura ambiente e iniziale delle celle è stata fissata a 25 ° C, mentre il SoC_{init} iniziale medio delle celle al 95%. La durata delle simulazioni è di 3600 secondi, in cui la corrente viene mantenuta costante a 200 A. Essendo costante, la corrente media su ogni cella è pari al 1/6 della corrente totale (200/6 = 33.33A) e il *SoC* medio delle 6 celle diminuisce linearmente.
Inoltre, le simulazioni sono state eseguite considerando le variazioni statistiche di un singolo parametro con diversi valori della deviazione standard. Ciò consente di analizzare l'effetto delle variazioni del singolo parametro sulle prestazioni della batteria.
Per ogni parametro e per ogni valore della deviazione standard sono state eseguite 50 simulazioni.

Fig. 4.12 riporta il valore medio della tensione di uscita del modulo nel tempo considerando le variazioni di tutti i 48 parametri (8 parametri *OCV,* R_i, R_1, C_1, R_2, C_2, Q_{nom}, SoC_{init} per ciascuna delle 6 celle nel pacco batteria), con una deviazione standard dell'1%, 1,5%, 2% e 3%. Il valore della tensione del modulo ideale con sbilanciamento nullo è riportato come riferimento. La differenza tra il valore medio e il caso ideale è dovuta alla tipica forma non lineare dell'*OCV* in funzione del *SoC*. Nei primi secondi la tensione di uscita è superiore a quella del caso ideale. Ciò è dovuto al fatto che alcune celle hanno un SoC_{init} superiore al valore nominale. Alla fine della simulazione, alcune celle hanno un *SoC* molto più basso delle altre e quindi la tensione di uscita diminuisce. Pertanto, la variazione statistica tra le celle causa un decremento della vita utile del pacco batteria. Considerando che un pacco batterie ideale raggiunge la tensione di uscita minima consentita dal tipo di chimica della cella in 3600 s, in media lo stesso limite viene raggiunto dopo 2000 s se la deviazione standard dei parametri delle celle è del 3%. La durata del pacco batteria diminuisce quindi del 55% a causa del mismatch tra le celle.

Fig. 4.13a riporta la deviazione standard media sulle 50 simulazioni della tensione di uscita del pacco batteria nel tempo considerando le variazioni di tutti i parametri con una deviazione standard dell'1%, 1,5%, 2% e 3%; mentre Fig. 4.13 (b)-(e) riportano gli stessi risultati considerando le variazioni dei singoli parametri (R_i, R_1, R_2, *OCV,* $SoC_{init,}$ Q_{nom}) per diversi valori di deviazione standard. Si noti che Fig. 4.13 (a)-(e) hanno scala diversa sull'asse y e diversi valori delle deviazioni standard dei parametri al fine di evidenziare l'effetto dei vari parametri sulla tensione di uscita.
Il comportamento della deviazione standard della tensione di uscita dipende dalla relazione non lineare delle 48 variabili casuali (8 parametri *OCV,* R_i, R_1, C_1, R_2, C_2, Q_{nom}, SoC_{init} per ciascuna delle 6 celle nel pacco batteria) con il *SoC* e la temperatura e dalle equazioni differenziali non lineari che descrivono il funzionamento elettrico della singola cella (4.2), (4.4). Le simulazioni nel tempo permettono di effettuare analisi più approfondite del comportamento nel tempo.
Nel transitorio iniziale, le variazioni della tensione di uscita sono rilevanti a causa delle variazioni del SoC_{init}. In media il SoC_{init} delle 6 celle è del 95%, ma alcune celle raggiungono il 100%.

La relazione non lineare tra tensione di uscita e *SoC* quando il *SoC* è vicino al 100% causa il forte incremento nella deviazione standard della tensione di uscita.
Dopo 100 s tutte le celle hanno un *SoC* compreso tra il 90% e il 20% e in questa regione la tensione di uscita è indipendente dal *SoC*. Quindi la deviazione standard della tensione di uscita non è influenzata dalle variazioni di SoC_{init}, come mostrato in Fig. 4.13a. In questa zona le variazioni dell'*OCV* influenzano fortemente la tensione di uscita; mentre tutti gli altri parametri hanno un effetto trascurabile.
Alla fine della simulazione, dopo circa 2500s, alcune delle celle hanno una forte scarica (valore di *SoC* inferiore al 20%); di conseguenza la deviazione standard della tensione di uscita aumenta. Questo è fondamentale per le prestazioni del pacco batteria: in molti casi la tensione di uscita dei pacchi batteria è inferiore alla tensione nominale (3.2 V in questo caso) e il carico non è correttamente alimentato. La ragione principale di questo malfunzionamento è il mismatch tra l'*OCV* delle celle, come si può vedere confrontando le Fig. 4.13a e Fig. 4.13d. Il mismatch su SoC_{init} e le resistenze interne non è rilevante, come si può vedere dalle Fig. 4.13b e Fig. 4.13c. Le variazioni delle capacità nominali delle celle diventano rilevanti alla fine della simulazione quando le celle con capacità nominale ridotta possono essere sovra-scaricate rispetto alle altre, come si può vedere in Fig. 4.13e.
In sintesi, l'effetto delle variazioni dei diversi parametri sulla deviazione standard della tensione di uscita è diverso nelle diverse fasi della scarica: le variazioni del SoC_{init} tra le celle sono rilevanti nella fase iniziale, le variazioni di Q_{nom} sono rilevanti solo nella fase finale. Le variazioni dell'*OCV* sono sempre rilevanti sulla variazione della tensione di uscita, mentre le variazioni delle resistenze interne sono trascurabili.

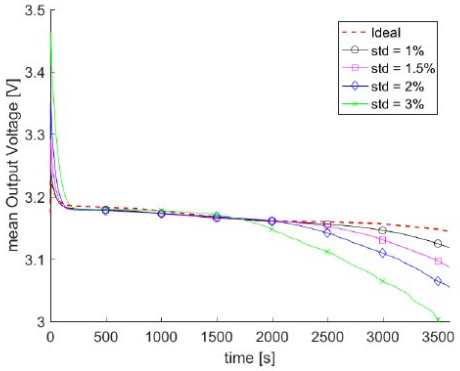


Fig. 4.12 - Valore medio sulle 50 simulazioni della tensione di uscita del modulo nel tempo

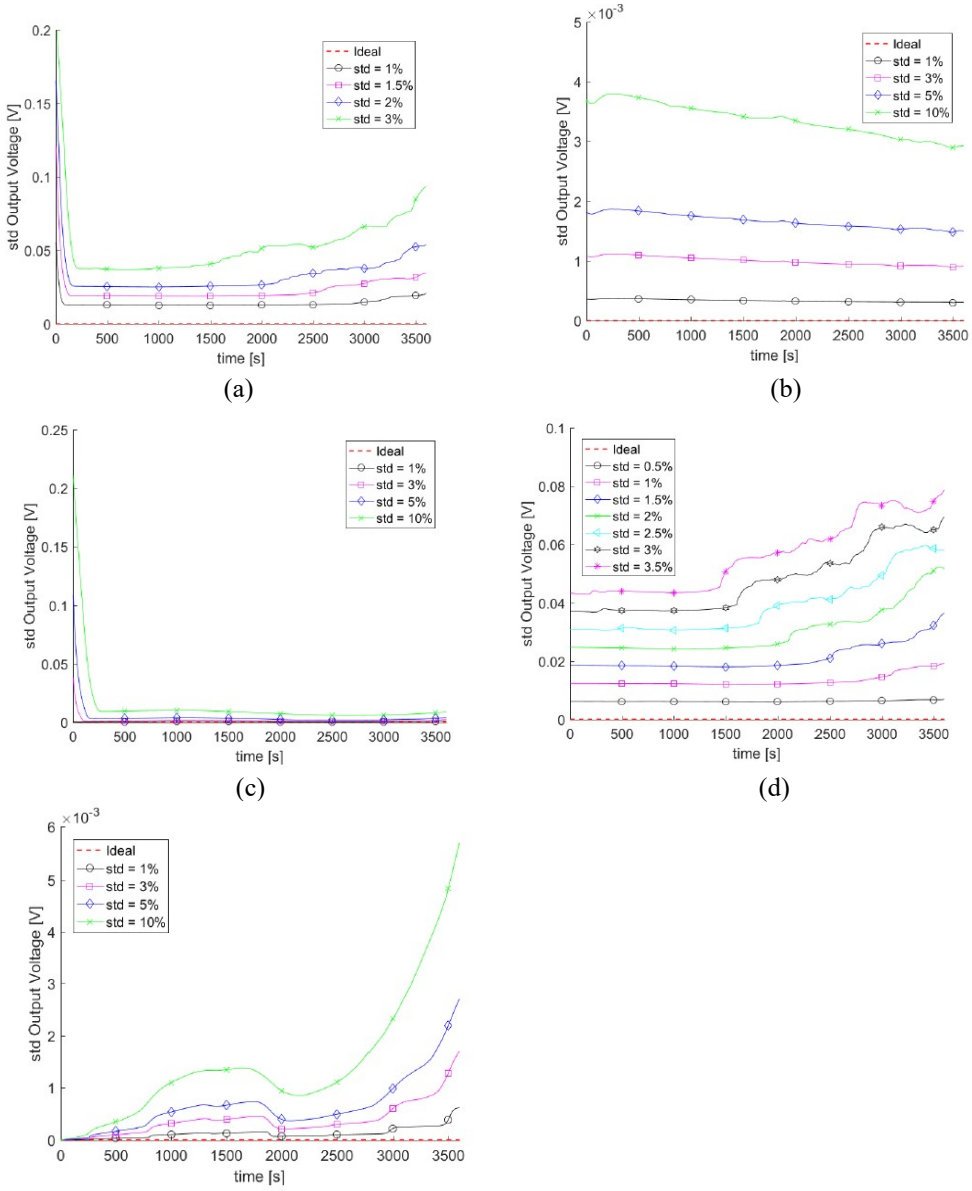


Fig. 4.13 - Deviazione standard della tensione di uscita del modulo nel tempo considerando la variazione: (a) di tutti i parametri delle 6 celle; (b) delle resistenze interne delle celle, R_i, R_1, R_2; (c) di SoC_{init}; (d) di OCV; (e) di Q_{nom}

Analogamente, Fig. 4.14a riporta il valore medio delle 50 simulazioni della deviazione standard del SoC delle 6 celle del modulo nel tempo considerando le variazioni di tutti i 48 parametri (8 parametri OCV, R_i, R_1, C_1, R_2, C_2, Q_{nom}, SoC_{init} per ciascuna delle 6 celle nel pacco batteria), con una deviazione standard dell'1%, 1.5%, 2% e 3%; mentre Fig. 4.14 (b)-(e) riportano gli stessi risultati considerando le variazioni dei singoli parametri (R_i, R_1, R_2, OCV, SoC_{init}, Q_{nom}) per diversi valori di deviazione standard.
In Fig. 4.14c si nota che la variazione iniziale del 10% del *SoC* diminuisce rapidamente fino a circa il 5%. La corrente delle celle con *SoC* più alto è maggiore nelle prime fasi, questo causa una riduzione più rapida del loro *SoC*.
L'effetto delle variazioni delle resistenze interne e di Q_{nom}, mostrato in Fig. 4.14b e Fig. 4.14e, è inizialmente trascurabile e aumenta durante la fase di scarica ma è meno rilevante rispetto alle variazioni dell'*OCV*, come mostrato in Fig. 4.14d.
Valori alti delle variazioni standard del *SoC* implicano che alcune celle sono sovra-scaricate, mentre altre sono ancora in buone condizioni. Ciò causa un rapido degrado delle celle già "stressate" e una conseguente riduzione rapida del loro stato di salute e, di conseguenza, del pacco batteria. Pertanto, la deviazione standard del *SoC* delle celle del pacco batteria deve essere ridotta e questo può essere ottenuto riducendo la deviazione standard dell'*OCV* tra le celle. L'effetto degli altri parametri è meno rilevante.

Fig. 4.15a riporta il valore medio sulle 50 simulazioni della deviazione standard della corrente di uscita delle 6 celle del modulo nel tempo considerando le variazioni di tutti i parametri con una deviazione standard dell'1%, 1,5%, 2% e 3%; mentre Fig. 4.15 (b)-(e) riportano gli stessi risultati considerando le variazioni dei singoli parametri (R_i, R_1, R_2, OCV, SoC_{init}, Q_{nom}) per diversi valori di deviazione standard.
Il valore medio della corrente di uscita delle 6 celle è di 33 A ed è costante nel tempo.
In Fig. 4.15c si nota che la variazione iniziale del *SoC* causa una forte differenza tra le correnti delle diverse celle. La corrente delle celle con *SoC* più alto è maggiore nelle prime fasi. L'effetto delle variazioni di SoC_{init} diminuisce rapidamente ed è trascurabile rispetto all'effetto delle variazioni dell'*OCV*, riportato in Fig. 4.15d.
L'effetto delle variazioni delle resistenze interne e di Q_{nom}, mostrato in Fig. 4.15b e Fig. 4.15e, è sempre trascurabile rispetto all'effetto delle variazioni dell'*OCV*, riportato in Fig. 4.15d.
La deviazione standard della corrente di uscita dipende principalmente dalle variazioni dell'*OCV* e questa dipendenza è quasi costante durante tutte le fasi della scarica.
Un alto valore di deviazione standard della corrente di uscita di ciascuna cella corrisponde al fatto che alcune celle possono fornire una corrente più elevata delle altre. Questo causa una rapida diminuzione del *SoC* e un maggiore riscaldamento per effetto Joule di quelle celle. Entrambi gli effetti possono essere pericolosi per lo stato di salute e per la sicurezza del pacco batteria.

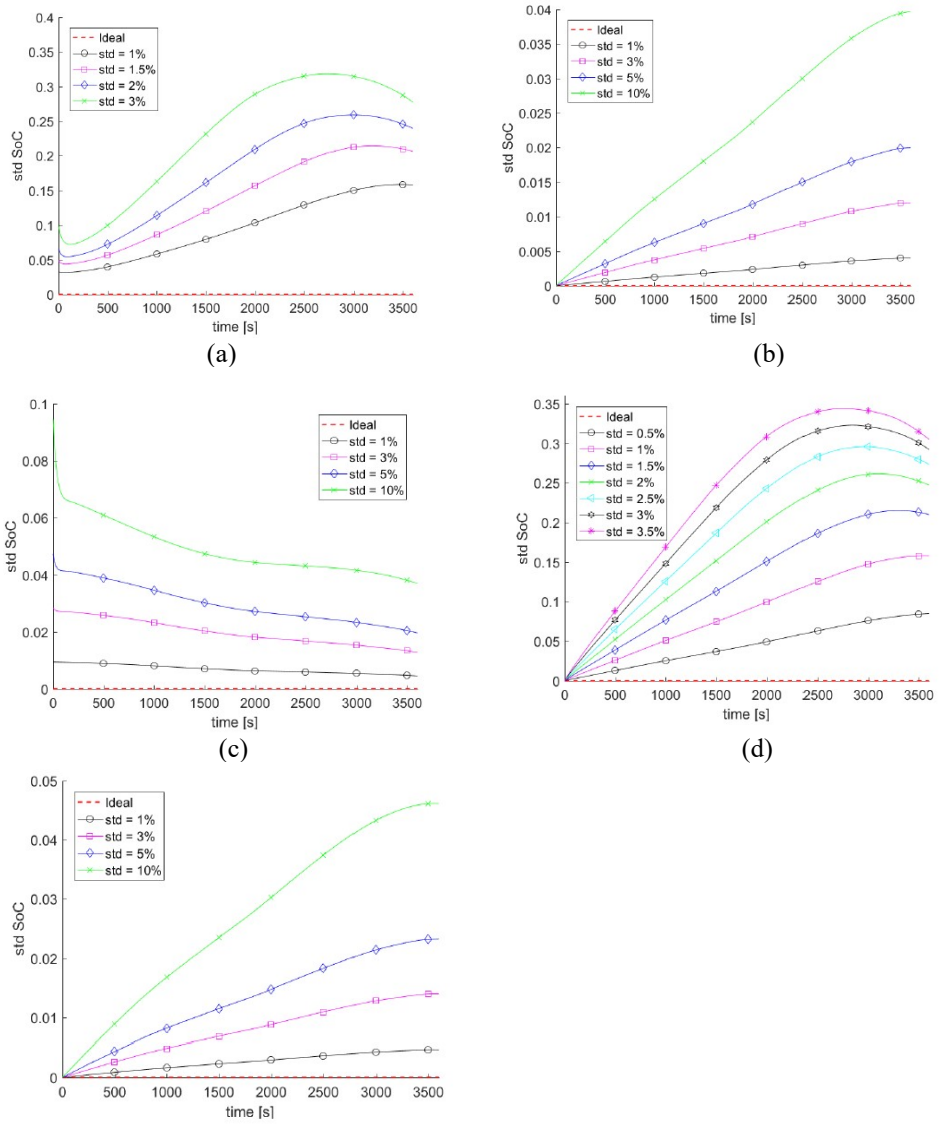


Fig. 4.14 – Valore medio sulle 50 simulazioni della deviazione standard di *SoC* nel tempo considerando la variazione: (a) di tutti i parametri delle 6 celle; (b) delle resistenze interne delle celle, R_i, R_1, R_2; (c) di SoC_{init}; (d) di *OCV*; (e) di Q_{nom}

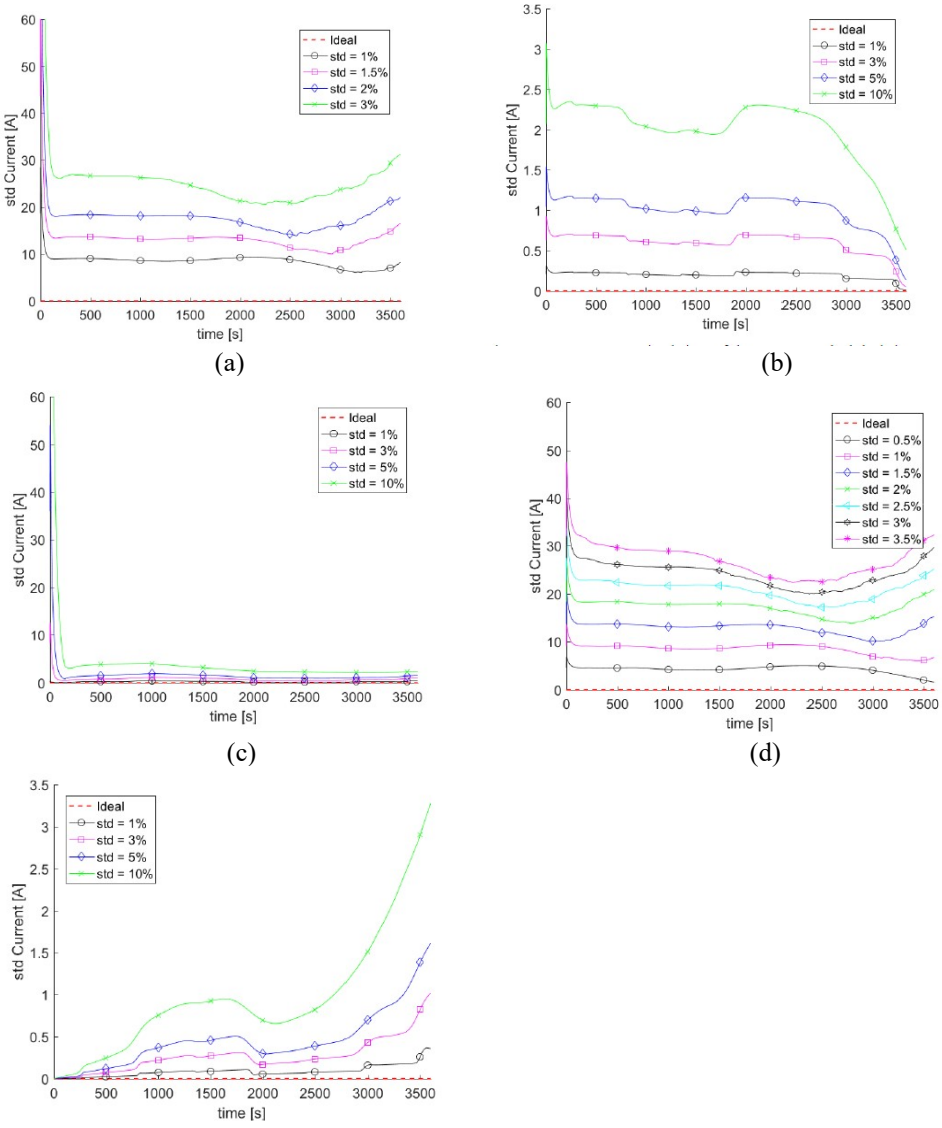


Fig. 4.15 – Valore medio sulle 50 simulazioni della deviazione standard della corrente nel tempo considerando la variazione: (a) di tutti i parametri delle 6 celle; (b) delle resistenze interne delle celle, R_i, R_1, R_2; (c) di SoC_{init}; (d) di OCV; (e) di Q_{nom}

I diversi parametri hanno effetti differenti sulle prestazioni e il loro effetto cambia nel tempo nelle diverse fasi della scarica. Le variazioni dell'OCV hanno una rilevanza maggiore rispetto agli altri parametri.

L'analisi effettuata consente di costruire una relazione tra le prestazioni del pacco batteria (valore medio e deviazione standard della tensione di uscita, corrente di uscita, *SoC*) e le variazioni dei parametri delle singole celle collegate in parallelo nel modulo. Questa relazione consente di definire delle specifiche richieste sul mismatch dei parametri delle celle collegate in parallelo.
La stessa analisi può essere utilizzata per dimostrare come l'incertezza sulla stima dei parametri delle singole celle possa causare un errore nella stima del *SoC* e di conseguenza sull'effetto dell'algoritmo di bilanciamento del BMS.

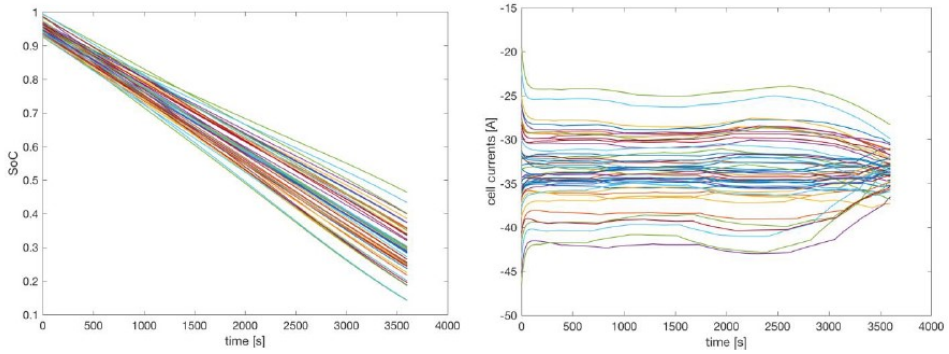
Si può concludere che la deviazione standard sull'*OCV* dovrebbe essere inferiore all'1%, mentre le variazioni sulla capacità nominale dovrebbero essere inferiori al 10%.
Inoltre, il BMS dovrebbe mantenere la deviazione standard dello stato iniziale di carica tra le celle inferiore al 10%.

4.2.2. Risultati

Sulla base dell'analisi statistica delle variazioni dei parametri interni della cella, si è simulato in SystemC-WMS un pacco batteria da 8 moduli in serie costituiti da 6 celle connesse in parallelo (8x6 celle). La capacità media delle celle è di 50 Ah, quindi la capacità nominale del pacco batteria è di 300 Ah con una tensione nominale di 3,2 * 8 = 25,6 V. La temperatura iniziale delle celle e la temperatura ambiente sono di 25°C, il valore medio iniziale di *SoC* delle celle è del 95%. La durata delle simulazioni è di 3600 secondi, in cui la corrente viene mantenuta costante a 200 A, con una corrente media di 33,33 A da ciascuna cella.
Le simulazioni sono state effettuate considerando le variazioni statistiche di un singolo parametro con valori diversi della deviazione standard da 0 a 10%. Per ogni parametro e per ogni valore della deviazione standard si sono eseguite 50 simulazioni.
Fig. 4.16, Fig. 4.17 e Fig. 4.18 mostrano i risultati delle simulazioni eseguite con:
$\sigma_{OCV} = 1\%$;
$\sigma_{Qnom} = 3.33\%$;
$\sigma_{SoCinit} = 1.66\%$;
$\sigma_{Ri} = \sigma_{R1} = \sigma_{R2dis} = \sigma_{C1} = \sigma_{C2} = 3.33\%$;

Fig. 4.16b evidenzia le variazioni correnti. A causa dei diversi valori dei parametri interni, le celle collegate in parallelo forniscono correnti diverse: quella con *OCV* più alto fornisce corrente più elevata rispetto a quella con *OCV* più basso.
Le celle che forniscono una corrente più elevata si scaricano più rapidamente rispetto alle altre: questo può essere visto dalla pendenza del *SoC* in Fig. 4.16a. Pertanto, alla fine della simulazione alcune celle hanno un valore di *SoC* vicino al 10%, mentre altre hanno un *SoC* vicino al 50%. Quando la cella è vicina all scarica completa, l'*OCV* diminuisce rapidamente, come mostrato in Fig. 4.4a. Le celle che nella prima parte della scarica hanno fornito una corrente più elevata, hanno un valore più basso di *SoC* alla fine della scarica e quindi riducono la corrente fornita. Questo fatto viene evidenziato dalla riduzione delle variazioni tra le correnti alla fine delle simulazioni, come mostrato in Fig. 4.16b.

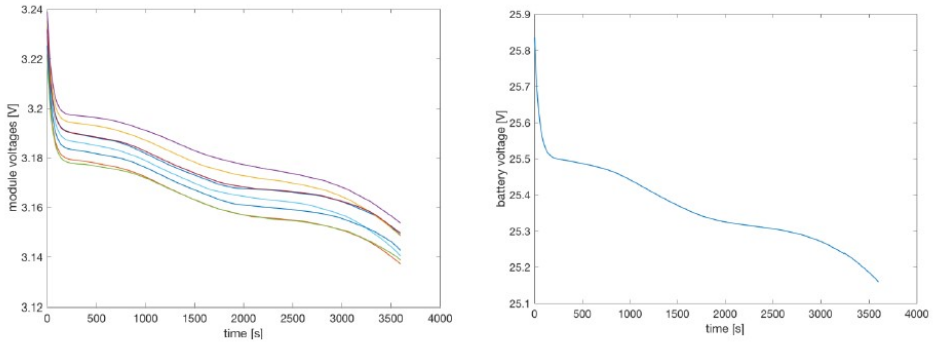
La carica totale fornita dalla batteria durante la simulazione è 200 Ah che corrisponde a 2/3 della capacità nominale. Dopo 1 ora, il *SoC* del pacco batteria ideale con celle identiche dovrebbe essere (0,95-200 / 300 * 1h) = 28,3%. Il pacco batteria nominale dovrebbe raggiungere il 10% di *SoC* (il valore minimo da raggiungere per mantenere la batteria in condizioni di sicurezza) dopo (0,95-0,1) * 300/200 = 1h 16 '. Al contrario, a causa della variabilità delle celle, Fig. 4.16a mostra che una delle celle raggiunge circa il 10% di *SoC* dopo un'ora, con una riduzione del 78% della durata reale della batteria, rispetto al pacco batteria ideale senza sbilanciamento delle celle.



(a) (b)

Fig. 4.16 – (a) *SoC* delle 48 celle; (b) corrente fornita da ogni cella

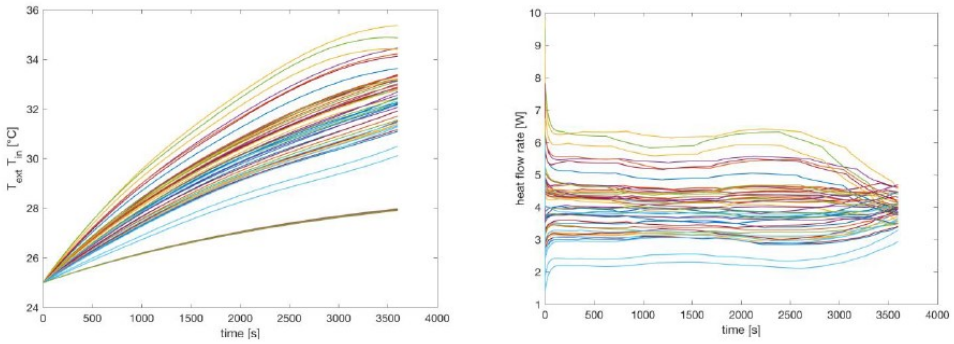
Fig. 4.17a mostra la tensione di uscita nel tempo degli 8 moduli collegati in serie; Fig. 4.17b mostra la tensione della batteria. La caduta di tensione iniziale nei primi 200s è dovuta alle resistenze e ai condensatori interni delle celle, quindi le tensioni sono ridotte a causa della riduzione del *SoC*. Le variazioni tra le tensioni di uscita degli 8 moduli sono evidenti, nonostante la connessione parallela delle 6 celle in ciascun pacchetto faccia una media della variazione delle celle.



(a) (b)

Fig. 4.17 – (a) Tensione di uscita degli 8 moduli connessi in serie; (b) Tensione di uscita del pacco batteria

Fig. 4.18a mostra le 48 temperature interne e le 48 temperature esterne delle celle. Le variazioni delle temperature esterne sono molto basse, in modo che sembrino un'unica linea spessa nella parte inferiore di Fig. 4.18a. Le linee superiori rappresentano le temperature interne. Le temperature aumentano nel tempo a causa della generazione di calore all'interno di ciascuna cella e la temperatura a regime costante viene raggiunta al termine della simulazione. Le variazioni tra le temperature interne sono dovute alle variazioni dei coefficienti di temperatura e, principalmente, alla diversa corrente che scorre in ciascuna cella. Fig. 4.18b mostra il flusso di calore di ciascuna cella. Le variazioni sono principalmente dovute alla diversa corrente che scorre in ogni cella.



(a) (b)

Fig. 4.18 – (a) Tensione di uscita degli 8 moduli connessi in serie; (b) Tensione di uscita del pacco batteria

L'ambiente di simulazione sviluppato consente un'analisi dettagliata del comportamento di un pacco batteria con variazioni statistiche tra le celle, e di conseguenza la definizione di un algoritmo di BMS mirato a ridurre l'effetto di queste variazioni sulle prestazioni della batteria.
È necessario eseguire una caratterizzazione accurata delle singole celle per selezionare le celle che devono essere connesse in parallelo.

Capitolo 5.

Conclusioni

La ricerca proposta ha presentato il progetto di un pacco batteria agli ioni di litio in ambiente SsytemC-WMS come casi di studio per la modellazione ad alto livello di sistemi eterogenei complessi.
Inoltre, ha presentato anche il flusso di gestione della complessità tramite uso di direttive di ottimizzazione nella metodologia di sintesi ad alto livello. Come campo applicativo in questo caso si è scelta la sintesi audio digitale, data l'elevata complessità del sistema richiesto.
In particolare, si è studiato il flusso di sintesi e ottimizzazione del progetto del filtro di Chamberlin. Si sono implementate diverse strutture RTL per valutare la gestione della complessità e implicazioni sulle prestazioni tramite l'applicazione di varie direttive di ottimizzazione.
Infine, si è implementato su FPGA un sintetizzatore audio funzionante che prevede l'integrazione di componenti progettati mediante metodologie a basso e alto livello.
Si è arrivati a implementare la struttura di un filtro di Chamberlin a 128 voci di polifonia.
Come sviluppo futuro, si può pensare allo studio dell'interfacciamento del sistema del sintetizzatore con il microprecessore tramite interfaccia AXI e all'estensione del progetto del filtro di Chamberlin a 256 voci di polifonia.

Bibliografia

1. G. Martin, G. Smith, “High-Level Synthesis: Past, Present, and Future”, IEEE Design & Test of Computers 2009
2. A. Sangiovanni-Vincentelli, “The Tides of EDA”, IEEE Design & Test of Computers 2003
3. H. D. Foster, “Why the design productivity gap never happened”, IEEE/ACM International Conference on Computer-Aided Design (ICCAD) 2013
4. International Technology Roadmap for Semiconductors (ITRS) http://www.itrs2.net/itrs-reports.html
5. ITRS executive summary 2011
6. ITRS executive report 2015
7. D. Chen, K. Choi, P. Coussy, Y. Xie, Z. Zhang, “ESL Design Methodology”, Hindawi Publishing Corporation Journal of Electrical and Computer Engineering Volume 2012, Article ID 358281, 2 pages doi:10.1155/2012/358281
8. J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, Z. Zhang, “High-Level Synghesis for FPGAs: from Prototyping to Deployment”, IEEE Transactions in Computer-Aided Design of Integrated Circuits and Systems, Vol.30, No. 4, April 2011
9. B. Bayley, F. Balarin, M. McNamara, G. Mosenson, M. Stellfox, Y. Watanabe, “TLM-Driven Design and Verification Methodology”, 2010, ISBN: 978-0-557-53906-2
10. D. Navarro, Ó. Lucía, L. A. Barragán, I. Urriza, J. I. Artigas - “Teaching Digital Electronics Courses Using High-Level Synthesis Tools”, 7th International Conference on e-Learning in Industrial Electronics (ICELIE) 2013, DOI: 10.1109/ICELIE.2013.6701269
11. L. Santos, S. Rigo, R. Azevedo, G. Araujo, “Electronic System Level Design: An Open-Source Approach”, Springer 2011
12. S. K. Shukla, C. Pixley, “Guest Editors’ Introduction: The True State of the Art of ESL Design”, IEEE Design & Test of Computers 2006
13. J. Sanguinetti, “Abstraction and Standardization in Hardware Design”, IEEE Design & Test of Computers 2012
14. L. Cai, D. Gajski, “Transaction Level Modeling: An Overview”, CODES+ISSS 2003
15. F. Rogin, R. Drechsler, “Debugging at the Electronic System Level”, Springer Science+Business Media B.V. 2010, DOI 10.1007/978-90-481-9255-7_2
16. A. Sangiovanni-Vincentelli, “Quo Vadis, SLD? Reasoning About the Trends and Challenges of System Level Design, IEEE Vol. 95, No. 3, 2007
17. R. Bergamaschi, L. Benini, K. Flautner, W. Kruijtzer, A. Sangiovanni-Vincentelli, K.Wakabayashi, “Roundtable: The State of ESL Design”, IEEE Design & Test of Computers 2008
18. D. D. Gajski, S.Abdi, A. Gerstlauer, G. Schirner, “Embedded System Design: Modeling, Synthesis and Verification”, Springer 2009

19. P. Coussy, A. Takach, M. McNamara - “An Introduction to the SystemC Synthesis Subset Standard”, CODESI/SSS'IO, October 24-29,2010, ACM 978-1-60558-905-3/10/10
20. G. Biagetti, M. Giammarini, M. Ballicchia, M. Conti, and S. Orcioni, “SystemC-WMS: wave mixed signal simulator for non-linear heterogeneous systems,” International Journal of Embedded Systems, vol. 6, no. 4, pp. 277–288, 2014
21. S. Orcioni, G. Biagetti, and M. Conti, “SystemC-WMS: Mixed signal simulation based on wave exchanges,” in Applications of Specification and Design Languages for SoCs, ser. ChDL, A. Vachoux, Ed. Springer, 2006, ch. 10, pp. 171–185, selected papers from FDL 2005]
22. https://github.com/orcioni/systemc-wms, “SystemC-WMS (Wave Mixed Signal simulator)”
23. G. Martin, B. Bailey, A. Piziali, “ESL Design and Verification: A Prescription for Electronic System Level Methodology, Elsevier 2007 ISBN: 9780080488837
24. L. Santos, S. Rigo, R. Azevedo, G. Araujo, “Electronic System Level Design: An Open-Source Approach”, Springer 2011
25. D. D. Gajski, L. Ramachandr, “Introduction to High-Level Synthesis”, IEEE Design & Test of Computers 1994
26. P. Coussy, D. D. Gajski, M. Meredith, A. Takach, “An introduction to High-Level Synthesis”, IEEE Design & Test of Computers 2009
27. H. Ren, “A brief introduction on contemporary High-Level Synthesis”, IEEE 2014
28. Xilinx High-Level synthesis user guide
29. R. Nane, Vlad-Mihai Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. Brown, F. Ferrandi, J. Anderson, K. Bertels, “A Survey and Evaluation of FPGA High-Level Synthesis Tools”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 35, No. 10, 2016
30. A. Madariaga, J. Jiménez, J. L. Martín, U. Bidarte, A. Zuloaga, “Review of Electronic Design Automation Tools for High-Level Synthesis”, IEEE International Conference on Applied Electronics 2010 ISBN: 978-80-7043-865-7
31. https://www.xilinx.com/products/design-tools/hardware-zone.html
32. J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, Z. Zhang, “High-Level Synthesis for FPGAs: From Prototyping to Deployment”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 30, No. 4, April 2011
33. G. De Poli, C. Drioli, F. Avanzini, “Sintesi dei segnali audio”, 1999
34. A. Cipriani, M. Giri, “Musica Elettronica e Sound Design”, ConTempoNet 2009
35. E. Murphy, C. Slattery, “All About Direct Digital Synthesis”, Analog Dialogue 2004
36. H. Chamberlin, “Musical Applications of Microprocessors”, ISBN 978-0810457683
37. Beat Frei, “Digital Sound Generation”, Institute for Computer Music and Sound Technology (ICST) Zurich University of the Arts
38. L. Lu, X. Han, J. Li, J. Hua and Minggao Ouyang, “A review on the key issues for lithium-ion battery management in electric vehicles,” Journal of Power Sources 2013, vol. 226, pp. 272 – 288

39. K. Makinejad, R. Arunachala, S. Arnold, H. Ennifar, H. Zhou, A. Jossen, W. Changyun, “Lumped Electro-Thermal Model for Li-Ion Cells in Electric Vehicle Application”, EVS28 KINTEX, Korea, May 3-6, 2015
40. R. Ahmed et al., “Model-Based Parameter Identification of Healthy and Aged Li-ion Batteries for Electric Vehicle Applications,” SAE Int. J. Altern. Powertrains, vol. 4, no. 2, pp. 2015-01–0252, 2015
41. M. Cacciato, G. Nobile, G. Scarcella, and G. Scelba, “Real-Time Model-Based Estimation of SOC and SOH for Energy Storage Systems,” IEEE Trans. Power Electron., vol. 32, no. 1, pp. 794–803, 2017
42. T. Huria, M. Ceraolo, J. Gazzarri, R. Jackey, “High Fidelity Electrical Model with Thermal Dependence for Characterization and Simula- tion of High Power Lithium Battery Cells,” Syst. Eng., pp. 1–8, 2012
43. F. Altaf, B. Egardt, L. Johannesson Mårdh, “Load Management of Modular Battery Using Model Predictive Control: Thermal and State-of-Charge Balancing”, IEEE Trans. on Control Systems technology, Vol.25, n.1, January 2017, pp. 47-62
44. K. Cheng, B. H. W. Divakar, K. Ding and F. Ho, "Battery management system (BMS) and SOC development for electrical vehicles," IEEE Trans. on Vehicular Technology, vol. 60, no. 1, pp. 76-88, January 2011
45. L. Liu, L. Yi Wang, Z. Chen, C. Wang, F. Lin, H. Wang, “Integrated System Identification and State-of-Charge Estimation of Battery Systems”, IEEE Transactions on Energy Conversion, Vol:28, Issue: 1, March 2013, pp. 13-23
46. F. Baronti, R. Saletti, W. Zamboni, “Open Circuit Voltage of Lithium-ion Batteries for Energy Storage in DC Microgrids”, proc. of IEEE International Conference on DC Microgrids (ICDCM), 7-10 June 2015
47. L. Lu, X. Han, J. Li, J. Hua, and M. Ouyang, “A review on the key issues for lithium-ion battery management in electric vehicles,” Journal of Power Sources, vol. 226, pp. 272 – 288, 2013
48. B. Pattipati, K. Pattipati, J. P. Christopherson, S. M. Namburu, D. V. Prokhorov, L. Qiao, “Automotive battery management systems,” in 2008 IEEE AUTOTESTCON, Sept 2008, pp. 581–586
49. S. Narayanaswamy, M. Kauer, S. Steinhorst, M. Lukasiewycz, S. Chakraborty, “Modular active charge balancing for scalable battery packs,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 25, no. 3, pp. 974–987, March 2017
50. L. McCurlie, M. Preindl, A. Emadi, “Fast model predictive control for redistributive lithium-ion battery balancing,” IEEE Transactions on Industrial Electronics, vol. 64, no. 2, pp. 1350–1357, Feb 2017

Pubblicazioni.

Capitolo su libro internazionale

1. “Chamberlin state-variable filter structure in FPGA for musical applications” - Adriana Ricci, Matteo Silvestrini, Massimo Conti, Marco Caldari, Franco Ripa, APPLEPIES 2018 6th Int. Conf. Applications in Electronics Pervading Industry Environment Society, Pisa, 26 Sept. 2018, in publication on the book Springer LNEE series

Rivista internazionale

1. “SystemC/TLM Controller for Efficient NAND Flash Management in Electronic Musical Instruments” - Massimo Conti, Marco Caldari, Matteo Gianfelici, Adriana Ricci, Franco Ripa – MDPI journal: Electronics, Special Issue "All-Digital Time-Mode Approaches for Mixed Analog-Digital Signal Processing", 2018, Volume 7, Issue 5, n.75 (18 May 2018)

2. “Lithium-ion Battery Electro-Thermal Model, Parameter Estimation and Simulation Environment” - Simone Orcioni, Luca Buccolini, Adriana Ricci, Massimo Conti – MDPI journal: Energies, Volume 10, Issue 3 (March 2017), 375, pp. 1-20; ISSN:1996-1073, doi: 10.3390/en10030375

Congresso internazionale

1. “Electric vehicles charging reservation based on OCPP” - Simone Orcioni, Luca Buccolini, Adriana Ricci, Massimo Conti – Proc. of IEEE 18th Int. Conference on Environmental and Electrical Engineering, EEEIC 2018 Conference: 12-15 June 2018, Palermo

2. “Effects of variability of the characteristics of single cell on the performance of a lithium-ion battery pack” - Simone Orcioni, Adriana Ricci, Luca Buccolini, Cristiano Scavongelli and Massimo Conti –2017 13th Workshop on Intelligent Solutions in Embedded Systems (WISES), 12-13 June 2017, Hamburg, Germany, Pages: 15 - 21; ISBN: ISBN:978-1-5386-1157-9, doi: 10.1109/WISES.2017.7986926

3. “Statistical Characterization of Lithium-ion Batteries using SystemC-WMS” - Simone Orcioni, Adriana Ricci, Luca Buccolini, Massimo Conti – Proc. of IEEE

17th Int. Conference on Environmental and Electrical Engineering, EEEIC 2017 Conference: 6-9 June 2017, Milano, doi: 10.1109/EEEIC.2017.7977744

4. “Battery Management System (BMS) Simulation Environment for Electric Vehicles” - Luca Buccolini, Adrianna Ricci, Cristiano Scavongelli, Giuseppe DeMaso-Gentile, Simone Orcioni, Massimo Conti – 2016 IEEE 16th International Conference on Environment and Electrical Engineering (EEEIC) Pubblication IEEE Xplore: 01/09/2016 Conference: 7-10 June 2016, Firenze

5. “Energy harvesting and wearable technologies for sleep monitoring", Massimo Conti, Roberto Lombardi, Adriana Ricci, Luca Buccolini, Simone Orcioni, Proc. of 2016 International Workshop on Analysis of biometric parameters to detect relationships between stress and sleep quality (AnBiPa 2016), Ancona, Italy, November 4, 2016, pp. 67-74, ISBN: 978-88-87548-09-9

Appendice A.

FPGA

I dispositivi FPGA (Field Programmable Gate Array) sono circuiti integrati digitali costituiti da blocchi logici configurabili collegati tra loro da interconnessioni programmabili. Il termine Field Programmable sta ad indicare che la programmazione avviene sul campo, ovvero da parte dell'utente finale, contrariamente a quei dispositivi le cui funzionalità interne sono prefissate dal produttore, come ad esempio i processori. Questo consente al progettista di riconfigurare la risorse logiche disponibili per implementare funzioni o sistemi adatti a specifiche applicazioni. Le prime board FPGA furono introdotte da Xilinx nel 1985, con lo sviluppo dell'XC2064.
Questielementi presentano caratteristiche intermedie tra i dispositivi ASIC (Application Specific Integrated Circuit) e i precedenti PLD (Programmable Logic Devices) come i PAL (Programmable Array Logic), GAL (Generic PAL) e i CPLD (Complex Programmable Logic Devices). Questo perchè la loro funzionalità può essere configurata sul campo, come i PLD, e possono contenere milioni di porte logiche ed essere usati per implementare funzioni estremamente grandi e complesse che, precedentemente, potevano essere realizzate solo usando dispositivi ASIC.
I vantaggi principali sono la velocità di implementazione e il fatto che permettono di apportare modifiche al sistema semplicemente riprogrammando il dispositivo in qualsiasi momento e in pochi secondi. Questo garantisce una notevole flessibilità e una riduzione di costi e tempi di progettazione, che per un ASIC sono molto dispendiosi, oltre al fatto che su questi il sistema è inciso sul silicio e quindi non modificabile. Tuttavia, gli FPGA presentano costi non competitivi per applicazioni con elevati volumi di produzione (milioni di pezzi) e la loro generalità si paga con prestazioni inferiori e una minore ottimizzazione delle risorse, contrariamente agli ASIC che permettono di ammortizzare gli elevati costi di progettazione e forniscono prestazioni e livelli di integrazione elevati.
La programmazione degli FPGA e ASIC si appoggia a strumenti di sviluppo software EDA in grado di eseguire la sintesi e la simulazione. Le specifiche possono essere rappresentate attraverso il disegno dello schema progettuale (schematic capture), oppure in forma testuale utilizzando un linguaggio HDL.
Sono spesso utilizzati per la prototipazione di dispositivi ASIC o come piattaforme per verificare l'implementazione fisica di nuove funzioni o algoritmi.

Struttura interna e programmazione

Un dispositivo FPGA è costituito da un elevato numero di blocchi logici (Slice) programmabili, circondati da una rete di interconnessioni programmabili, come mostrato in Fig. A.1. I singoli blocchi logici possono essere configurati per realizzare semplici funzioni e collegati opportunamente tra loro programmando le interconnessioni, realizzando così funzioni più complesse.

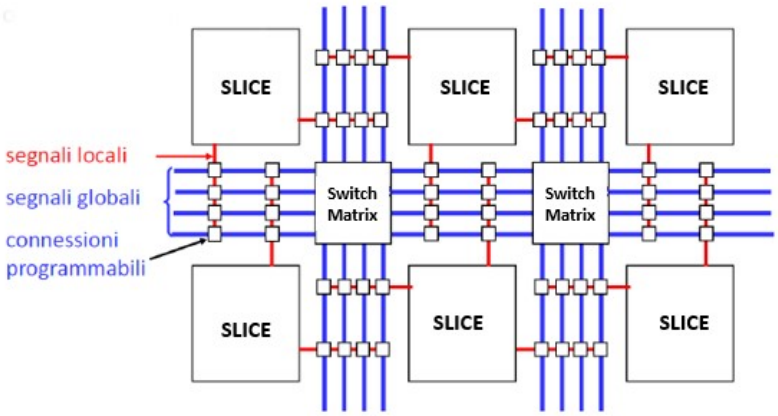


Fig A.1 – Struttura interna di una FPGA

La struttura interna di una slice (blocco logico) è mostrata in Fig. A.2.
La Look-Up Table (LUT) è composta da celle SRAM, che sono configurate per implementare una generica funzione combinatoria. L'uscita della LUT può essere utilizzata direttamente oppure può essere memorizzata in un flip-flop di tipo D, implementando così un circuito sequenziale. Inoltre, è presente un ingresso ausiliario per combinare più funzioni logiche attraverso un multiplexer (MUX).
La funzione logica svolta da ciascuna slice e i collegamenti tra queste sono determinati commutando degli interruttori programmabili costituiti da un MOSFET comandato da una memoria SRAM. Dato che tale meccanismo è volatile e le memorie perdono il loro contenuto quando il dispositivo viene sconnesso dall'alimentazione, è necessario ricaricare la configurazione (bistream) da una memoria esterna non volatile ogni volta
che si accende il componente.

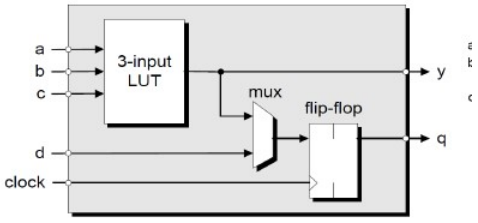


Fig. A.2 – Struttura interna di una slice