POLITECNICO DI TORINO

III Facoltà di Ingegneria dell'Informazione
Corso di Laurea in Ingegneria Elettronica

Tesi di Laurea

Sviluppo di un ambiente di progettazione automatica di circuiti analogici

Relatori:
prof. Leonardo Reyneri
prof. Dante Del Corso

Candidato:
Musumeci Domenico Onorio - 126530

Novembre 2006

Sommario

Negli ultimi anni sistemi elettronici misti hardware/software si sono diffusi sia a livello industriale che nell'ambito dell'elettronica di consumo. Questa tendenza ha alimentato la ricerca e lo sviluppo di nuovi ambienti di progettazione che aiutassero il progettista nella realizzazione di sistemi ibridi, riducendo notevolmente i tempi che intercorrono fra l'ideazione di di un sistema hardware/software e la sua realizzazione fisica.

Il progetto del tool *CodeSimulink*, avviato dal Dipartimento di Elettronica del Politecnico di Torino, si inserisce in questo contesto: CodeSimulink è un ambiente di coprogettazione hardware/software che utilizza Simulink^{TM} come interfaccia utente e strumento di simulazione. Per ogni blocco si possono definire dei parametri che permettono di emulare, le caratteristiche del dispositivo fisico che esegue la stessa funzione del blocco. Inoltre, in fase di simulazione, è possibile fornire un'analisi funzionale o delle prestazioni del sistema, per ottenere stime sulla potenza dissipata, la velocità o il costo del sistema reale. Infine, il tool permette di estrarre, a partire da un modello Simulink^{TM} così definito, una descrizione hardware digitale in linguaggio VHDL o una descrizione software nel linguaggio C.

Il progetto sviluppato in questa tesi si propone di estendere le caratteristiche del tool CodeSimulink anche nell'ambito hardware analogico: la prima parte del lavoro consiste nel definire per un certo numero di blocchi CodeSimulink, il circuito elettronico reale che esegue la stessa funzione del blocco. A questo punto sarà possibile scegliere i componenti elettronici reali che costituiscono il circuito equivalente partire dai parametri introdotti per descrivere il comportamento "analogico" del blocco in simulazione. La seconda parte della tesi è incentrata sulla traduzione di un modello CodeSimulink descritto in hardware analogico in un file circuito (.cir) da utilizzare per la simulazione in Spice. Sia la prima parte del progetto che la seconda sono state realizzate con la programmazione in codice MATLAB^{TM}.

La tesi viene strutturata nel seguente modo:

Capitolo 1 : viene presentato in generale il problema della *coprogettazione* e viene descritto l'ambito in cui si inserisce il progetto svolto.

Capitolo 2 : viene illustrato il problema del compilatore hardware analogico e vengono analizzati i principali comandi necessari per implementare i sottocircuiti in questo linguaggio.

Capitolo 3 : viene descritta la realizzazione del database di componenti elettronici reali, necessario per descrivere i circuiti equivalenti di ogni blocco CodeSimulink.

Capitolo 4 : vengono presentati gli algoritmi per selezionare i componenti elettronici del database. Per ogni blocco CodeSimulink trattato vengono presentati i rispettivi algoritmi di ricerca, sia per i componenti passivi che per quelli attivi.

Capitolo 5 : vengono descritte le modifiche apportate al compilatore e le nuove funzioni introdotte per l'estensione hardware analogica del tool.

Capitolo 6 : in un modello di esempio, vengono illustrate le tecniche di scelta dei componenti e la conseguente compilazione in linguaggio Spice del diagramma.

Capitolo 7 : vengono proposte le possibili modifiche implementabili per estendere ulteriormente la descrizione hardware analogica di CodeSimulink.

Appendice A : tutti i circuiti equivalenti dei blocchi trattati vengono analizzati nello specifico e per ognuno di essi vengono ricavate le formule principali che hanno permesso di definire gli algoritmi di ricerca dei componenti

Appendice B : vengono riportati i listati per tutte le funzioni realizzate.

Indice

Elenco delle figure

Capitolo 1

Introduzione

1.1 La coprogettazione hardware/software

Negli ultimi anni nell'industria elettronica si è assistito ad una crescita dello sviluppo di sistemi elettronici realizzati in tecnologia mista hardware e software.

Questo sistemi offrono molteplici vantaggi, combinando la velocità e le prestazioni garantite dall'hardware con la flessibilità e la potenza descrittiva del software e si sono diffusi sia a livello dell'elettronica di consumo sia nell'ambito puramente industriale. Uno dei limiti introdotti da questi sistemi riguarda la progettazione vera e propria: mentre prima il progetto era suddiviso fra più team di lavoro che si occupavano individualmente della componente software e di quella hardware[1], ora ai progettisti di questi sistemi ibridi è sono richieste competenze specifiche in entrambi gli ambiti tecnologici.

Oltre a questi sistemi misti si sono conseguentemente diffusi degli ambienti di sviluppo concepiti per aiutare il progettista nel design di questi sistemi, riducendo così i tempi di progettazione e di implementazione, permettendo una vera e propria *coprogettazione hardware/software* (detta anche *hardware/software codesign*).

Presso il Politecnico di Torino è stato avviato il progetto di sviluppo di un ambiente di elaborazione di sistemi misti chiamato CodeSimulink. Esso si propone come uno strumento di *codesign hardware/software* che sfrutta le interfacce grafiche di MATLAB^{TM} e la possibilità di simulare dei sistemi fisici (modellizzati con Simulink^{TM}).

1.2 Il *tool* CodeSimulink

Simulink^{TM} è un software per la modellizzazione, la simulazione e l'analisi di sistemi dinamici incluso in MATLAB^{TM}. Supporta sistemi lineari e non lineari modellizzati in tempo continuo o campionati. Per la modellizzazione si utilizza l'interfaccia grafica (GUI[2]) di Simulink^{TM} che associa ad ogni modello un blocco: ognuno di essi "nasconde" le equazioni

[1] naturalmente era necessario aver prima stabilito delle specifiche comuni di progetto fra i vari gruppi di lavoro

[2] Graphic User Interface

che descrivono il funzionamento del blocco stesso. Con questa interfaccia è possibile combinare diversi blocchi/modelli in un unico file *model* (.mdl) e disegnare, come su carta, il sistema che si desidera progettare.

Il *tool* di codesign hardware/software CodeSimulink si inserisce in questo ambito e permette di rappresentare il sistema ibrido che si desidera realizzare attraverso un diagramma a blocchi. Quando il comportamento del sistema ha fornito dei corretti risultati in simulazione, il modello può essere compilato, ovvero trasformato in una descrizione software, hardware digitale, hardware analogica o mista.

Oltre alla possibilità di progettare rapidamente dei sistemi ibridi[3], CodeSimulink permette anche:

- l'analisi comparativa delle prestazioni dello stesso sistema descritto con differenti tecnologie.

- la simulazione rapida di modelli complessi, attraverso la compilazione e l'implementazione in tecnologia hardware.

Inoltre CodeSimulink prevede un paradigma d'interfacciamento fra le varie tecnologie supportate che permette al progettista di concetrarsi sulla soluzione del problema in generale senza dover occuparsi di interfacciare blocchi descritti in modo differente.

1.3 I modelli e la stima delle prestazioni

I blocchi CodeSimulink riprendono quelli di Simulink^{TM} e gli aggiungono i parametri necessari per caratterizzarne l'implementazione *estendendo* così le loro funzionalità iniziali. Vengono per questo chiamato *blocchi estesi*. I parametri di descrizione dell'implementazione vengono settati attraverso delle finestre di dialogo e, i valori introdotti, sono utilizzati sia in fase di simulazione del comportamento del sistema che per effettuare stime di costo e prestazioni. Nei blocchi hardware i modelli di stima sono implementati attraverso delle curve d'interpolazione sui dati reali che a loro volta sono ricavati da implementazioni di ogni blocco per valori significativi dei suoi parametri. Per i blocchi software invece si tiene conto di parametri come il tempo di esecuzione del codice e la sua dimensione: in questo caso ci si riferisce comunque al caso peggiore o *worst case*.

In tutti i casi, i modelli di costo e di prestazioni sono riconducibili a delle funzioni parametriche che utilizzano i valori specificati dall'utente per i parametri di progetto. La stima delle prestazioni globali del sistema è ottenuta sfruttando le capacità computazionali offerte da MATLAB^{TM}. I risultati forniti dale stime richiedono differenti parametri a seconda della tecnologia d'implementazione.

[3]detta anche *fast prototyping*

1.4 Le tecnologie, le architetture ed i protocolli di comunicazione

Come già accennato in precedenza, un blocco *esteso* di CodeSimulink può essere descritto in modo diverso a seconda della rappresentazione che gli si vuole attribuire nel modello del sistema. Sono disponibili tre differenti tecnologie:

Software : un blocco realizzato in questa tecnologia viene interpretato come una porzione di codice da eseguire su di un microprocessore. In questo caso le stime effettuate terranno conto delle dimensioni del codice e del suo tempo di esecuzione.

Hardware Digitale : in questo caso il blocco verrà interpretato come un circuito logico. I risultati delle stime permetteranno di valutare

- la *latenza* che fornisce informazioni sul ritardo introdotto dai blocchi fra l'ingresso e l'uscita del sistema in termini di numero di clock. Il parametro in questione è necessario nei sistemi sincroni.
- la massima frequenza di clock del sistema
- l'energia dissipata dal sistema in un periodo di clock
- l'area occupata dall'implementazione in termini di celle per le FPGA[4] ed in termini di superficie di silicio occupata per ASIC[5].

Hardware Analogica : il blocco così definito verrà interpretato come un circuito analogici costituito da componenti discreti sia attivi che passivi. In questo caso le stime effettuate permetteranno di determinare quali componenti elettronici utilizzano i circuiti equivalenti.

Per i blocchi realizzati in tecnologia hardware (sia digitale che analogica) è possibile anche specificare diversi tipi di architetture:

- per la tecnologia hardware digitale è possibile scegliere fra le seguenti architetture: parallela asincrona, seriale sincrona (*MSB first* o *MSB last*), seriale asincrona, adiabatica parallela, sistolica sincrona, e così via.
- per la tecnologia hardware analogica si possono individuare architetture *single-ended* o differenziali, in tensione o in corrente, a modulazione d'impulsi o di frequenza e così via.

La connessione fra blocchi appartenenti a diverse tecnologie può essere effettuata tramite dei blocchi d'interfaccia come ad esempio convertitori A/D e D/A, circuiti di codifica e decodifica, generatori di impulsi PWM. È definito inoltre un paradigma di comunicazione che tiene conto del tipo di dato scambiato fra i blocchi che può essere scalare, vettoriale o

[4] *Field Programmable Gate Array* è un dispositivo contenente componenti in logica programmabile e interconnessioni programmabili

[5] *Application Specific Integrated Circuit* è un circuito integrato realizzato per un'applicazione specifica

matriciale. In CodeSimulink non vengono trattati i dati complessi[6]. La rappresentazione dei dati digitali può avvenire solo in virgola fissa, nelle forme con segno o senza segno, in modulo e segno o in complemento a due, o con altre forme (p.es. sistemi ridondanti di numerazione per sistemi a bassa potenza). Un blocco in tecnologia hardware digitale si può considerare composto da più celle, ognuna con una specifica funzione. Una cella di protocollo è dedicata alla gestione di tutti e soli i segnali di protocollo, indipendentemente dalla funzione del blocco. Una cella di conversione dati è invece dedicata ad attuare conversioni tra larghezze dei segnali (in bit) e formato dei dati, ed a gestire il troncamento e la non rappresentabilità di essi. Infine vi deve sempre essere una cella funzionale, che descrive il comportamento del blocco, ovvero la relazione tra dati in ingresso e dati in uscita. Eventualmente quest'ultima cella può essere composta da più celle interagenti tra loro. La cella funzionale non si deve preoccupare della gestione dei segnali di protocollo e della conversione dei dati, dato che queste azioni vengono compiute dalle altre celle del blocco. Il progetto di un blocco si riduce quindi al progetto della sua cella funzionale.

1.5 La compilazione in CodeSimulink

Dopo aver progettato un modello ed averlo collaudato, è possibile compilarlo, ovvero tradurlo in un linguaggio di programmazione diverso a seconda della tecnologia dei blocchi presenti. Questo passo risulta necessario per realizzare il sistema in tecnologia mista hardware/software. Prima di compilare il sistema sono necessarie delle fasi intermedie: innanzitutto si ricava un diagramma equivalente a quello originale in cui tutti i blocchi estesi si trovano al livello di gerarchia più alto. In poche parole, ogni blocco viene sostituito con il suo diagramma interno (livello di gerarchia più bassa) mantenendo inalterate le connessioni del blocco iniziale con gli altri oggetti del sistema.

Dopo aver eliminato la gerarchia si può ancora semplificare la compilazione, suddividendo il diagramma in tre parti distinte a seconda delle tecnologie in cui sono rappresentati i blocchi. I diagrammi "specifici" così ricavati possono essere tradotti nei seguenti linguaggi:

- il modello puramente software viene tradotto in linguaggio di programmazione C grazie al *Real-Time Workshop* incluso in MATLAB^{TM}. Per completare questa fase è richiesta una fase di configurazione iniziale.

- il modello puramente hardware digitale viene tradotto nel linguaggio di descrizione analogica VHDL. La compilazione deve essere eseguita in forma strutturale, nella quale non è necessario descrivere il comportamento dei blocchi, ma è sufficiente rendere nota la loro presenza nel diagramma tramite una dichiarazione. Successivamente verranno specificate le corrispondenti interconnessioni. L'implementazione di un blocco viene detta *cella*. Un protocollo di comunicazione di tipo *dataflow* definito in CodeSimulink si occuperà di gestire le connessioni fra le varie celle e la presenza dei blocchi d'interfaccia.

[6] nel caso in cui si desideri rappresentarli si può sempre utilizzare in contemporanea due dati reali che rappresentano rispettivamente la componente reale e quella immaginaria del dato

- la traduzione del modello hardware analogico è uno degli argomenti trattati in questo progetto di tesi e verrà analizzato di seguito nel paragrafo 1.6.

Per implementare il sistema nel linguaggio codificato è necessaria un'ulteriore fase di compilazione in particolare per il codice in VHDL e quello in C.

1.6 Gli obiettivi del progetto di tesi

L'obiettivo principale del progetto descritto nella presente tesi è quello di ampliare le capacità di progettazione analogica di CodeSimulink. Allo stato attuale del lavoro, i blocchi presenti nel tool permettono emulare il comportamento dei circuiti elettronici reali, a patto che vengano esplicitati determinati *parametri analogici* per descrivere il segnale d'uscita del blocco CodeSimulink[7].

Nella prima parte dl progetto si cercherà di determinare quali siano i circuiti reali emulati dai blocchi CodeSimulink e, in particolare, i componenti elettronici presenti sul mercato che permettono di realizzare un circuito che abbia un comportamento equivalente a quello simulato con il blocco CodeSimulink di origine.

Per poter scegliere i componenti elettronici che meglio realizzano un determinato circuito, sarà necessario disporre di una vasta gamma di possibili componenti realmente esistenti e distinti fra di loro in base ad alcune caratteristiche elettriche (offset, alimentazioni, consumi ...) ed al loro prezzo. Questo problema verrà ampiamente discusso nel capitolo 3.

Oltre ad un elenco di componenti fra cui scegliere, sarà inoltre necessario definire dei criteri di ricerca che, considerando i *parametri analogici* dei blocchi CodeSimulink, permettano di risalire ai componenti elettronici che realizzano circuiti analogici appropriati. Questa parte del lavoro verrà trattata nel capitolo 4.

Una volta determinati i circuiti elettronici ed i relativi componenti, si può pensare alla traduzione di un diagramma CodeSimulink dal punto di vista hardware analogico. Come punto di partenza, si è scelto di utilizzare un linguaggio di simulazione supportato da Spice: a partire da un diagramma CodeSimulink opportunamente descritto dal punto di vista analogico, è possibile risalire ad un file circuito con estensione *.cir* facilmente simulabile con qualsiasi versione di Spice. In questo file ogni blocco CodeSimulink verrà rappresentato dal suo equivalente circuito analogico: i circuiti che utilizzano degli integrati come degli operazionali possono essere descritti correttamente grazie ai modelli Spice che vengono messi a disposizione dalle aziende di componenti elettronici.

Questa fase del lavoro verrà analizzata nel capitolo 2, in cui verranno forniti i primi rudimenti necessari per descrivere in questo linguaggio i circuiti analogici equivalenti ai blocchi CodeSimulink.

Tutte le modifiche del compilatore, implementate in linguaggio di programmazione MATLAB™, vengono trattate nel capitolo 5.

[7] I blocchi CodeSimulink vengono anche detti *estesi* in quanto estendono le proprietà previste in origine da Simulink™

Capitolo 2

La compilazione in linguaggio di simulazione Spice

Prima di approfondire il progetto descritto nel paragrafo 1.6, in particolare la definizione dei circuiti e la ricerca dei componenti, è conveniente approfondire la conoscenza del simulatore Spice. I circuiti che saranno assegnati ad ogni blocco CodeSimulink devono risultare facilmente descrivibili in questo linguaggio, senza per questo essere troppo approssimati.

Un aiuto per questo problema viene dato dalle aziende produttrici di componenti che mettono a disposizione dei modelli di circuiti integrati descritti in Spice che permettono di simulare adeguatamente il comportamento del dispositivo.

Di seguito vengono illustrate le nozioni base per riuscire poter descrivere agevolmente i circuiti in linguaggio Spice. Queste informazioni sono state tratte dalle fonti [2] 3 [5] riportate nella bibliografia.

2.1 La storia del simulatore Spice

Spice fu sviluppato in origine nei laboratori di ricerca elettronica nella University of California, Berkeley nel 1975 da Larry Nigel e Donald Pederson. Le prime due versioni vennero scritte in codice Fortran mentre le successive vennero codificate in C, utilizzando però una sintassi tipo quella di Fortran per la descrizione del circuito. Molte versioni commerciali di Spice hanno rimpiazzato quella originale, aggiungendo estensioni proprietarie che limitano la portabilità dei modelli di descrizione del circuito. Le versioni più recenti includono inoltre un'interfaccia grafica per la descrizione dei circuiti. La prima versione di Spice utilizzava il metodo dei nodi per la risoluzione dei circuiti, ma questo non permetteva di utilizzare generatori di tensione ideali e induttori. Le versioni successive utilizzano invece il metodo dei nodi modificato che risolve il difetto della prima versione. Spice utilizza diversi algoritmi di linearizzazione (come quello di Newton-Raphson) per convertire un problema complesso di analisi di un circuito in una serie di sottoproblemi lineari, semplici da risolvere nel loro punto di lavoro.

È utile introdurre qualche nozione di base della programmazione in Spice, per avvicinare

l'utente a un ambiente di sviluppo non molto comune, ma non per questo troppo complesso. Le regole base adottate sono le seguenti:

1. PSpice non è "*case sensitive*". Questo significa che nomi come *Vbus, VBUS, vbus* e anche *vBuS* sono equivalenti nel programma;
2. Tutti i nomi degli elementi devono essere unici, perciò non possono essere presenti due resistori entrambi chiamati "Rbias", per esempio;
3. La prima linea nel file è utilizzata come titolo. Spice ignora questa riga come dato del circuito;
4. Ci deve essere un nodo designato come "0". Questo è il nodo di riferimento rispetto a cui saranno calcolate tutte le tensioni;
5. Ogni nodo deve avere collegare almeno due elementi;
6. L'ultima linea del file deve essere sempre ".END";
7. Tutte le linee non bianche (eccetto il titolo) devono avere un carattere nella prima colonna di ogni linea
   - Si usa "*" per creare una linea di commento;
   - Si usa "+" per continuare la linea precedente (migliora la leggibilità);
   - Si usa "." per passare un'istruzione speciale al programma;
   - Si usa una lettera (seguita dal resto del nome) per identicare un elemento (resistenza, generatore di tensione, sottocircuito);
8. Si usano le spaziature per separare i campi dei dati all'interno della stessa linea;
9. Si usa ";" per terminare le informazioni di quella linea. È possibile aggiungere un commento per quella linea subito dopo il punto e virgola.

A queste regole base si aggiunge inoltre la seguente notazione numerica per le potenze:

2.2 I principali comandi Spice

Si analizzano ora i principali elementi e comandi utilizzabili:

Generatore ideale di tensione: per semplicità verranno trattati solo il caso in DC (Direct Current o corrente continua).e il caso in AC (Alternate Current) La prima lettera della linea deve essere "V", seguito poi dal resto del nome. Il nome completo è poi seguito dal nodo positivo, dal nodo negativo e (opzionalmente) dall'etichetta "DC".

Esempio:

Tabella 2.1. Notazione potenze in Spice.

Numero	Prefisso	Nome
10^{12}	"T" o "t"	*tera*
10^{9}	"G" o "g"	*giga*
10^{6}	"MEG" o "meg"	*mega*
10^{3}	"K" o "k"	*kilo*
10^{-3}	"M" o "m"	*milli*
10^{-6}	"U" o "u"	*micro*
10^{-9}	"N" o "n"	*nano*
10^{-12}	"P" o "p"	*pico*
10^{-15}	"F" o "f"	*femto*

name	*+node*	*-node*	*type*	*value*	*comment*
Va	4	2	DC	16V;	"V" dopo "16" è opzionale come anche "DC"

Esempio:

name	*+node*	*-node*	*type*	*value*	*phase*	*comment*
Vdep	15	27	AC	10;	-30;	la fase è espressa in gradi

Resistenza: in questo caso la prima lettera della linea deve essere "R". Il nome viene poi seguito dai nodi, prima il positivo e poi il negativo, e dal valore in ohm. Il valore nullo produce un errore. Poiché il resistore non è un elemento attivo, la polarità dei nodi non ha effetto sul valore della tensione e della corrente riportati nella soluzione. Il flusso della corrente è comunque riportato a partire dal nodo sinistro a quello destro.

Esempio:

name	*+node*	*-node*	*value*	*comment*
Rabc	31	0	14k;	la corrente riportata va dal nodo 31 al nodo 0

Capacità: per il capacitore la prima lettera deve essere la C. Come per il resistore i primi due numeri che seguono il nome del componente corrispondono rispettivamente al nodo positivo e al nodo negativo della capacità. Il valore del capacitore (in Farad) segue la lista dei nodi. Infine, poiché il componente può immagazzinare energia, Spice permette di specificarne il valore iniziale di tensione (IC) che si inserisce come ultimo dato nella stringa di codice e definisce la condizione iniziale da utilizzare per l'analisi del transitorio.

Esempio:

name	*+node*	*-node*	*value*	*initial_voltage*
Cfb	4	5	50uF;	IC=20V

Come per i casi precedenti, Spice ignora la lettera "F" per Farad e la lettera "V" per Volt che possono quindi essere omesse.

Analisi in DC: questo comando permette di variare un parametro (nel nostro caso la tensione d'ingresso, ovvero l'unico generatore indipendente) entro un range di valori, per poter così stampare in una volta sola l'andamento delle tensioni e delle correnti del circuito. Il comando utilizzato per avviare questa modalità è ".DC", seguito dal nome della sorgente indipendente di tensione (o corrente), il valore di partenza della variabile, quello finale e il valore del passo, ovvero dell'incremento con cui si passa dal valore iniziale a quello finale. Nel caso in cui il valore finale e quello iniziale coincidano, l'analisi verrà effettuata solo in quel determinato punto.

Esempio:

Command	*source_name*	*start_value*	*stop_value*	*increment_value*	
.DC	Vs	5	20.0	20.0	1.0

Nel caso in cui si effettui un'analisi in DC il valore di tensione precedentemente assegnato alla sorgente verrà ignorato.

Analisi in AC: analogamente al caso precedente, il comando ".AC" permette di effettuare un'analisi parametrica al variare della frequenza stampando la risposta in frequenza delle variabili del circuito per diversi valori. Subito dopo il comando è necessario aggiungere il tipo di variazione, descritto di seguito, il numero di punti su cui si vuole effettuare l'analisi in frequenza e infine il range di frequenza in cui analizzare la risposta. La variazione di frequenza può assumere tre valori diversi:

LIN Il range di frequenza verrà diviso equamente per il numero dei punti indicato dall'utente;

DEC Il numero dei punti corrisponde al numero di punti per decade;

OCT Il numero dei punti corrisponde al numero di punti per ottava;

Il discorso delle variabili da stampare va affrontato a parte poiché, diversamente dal caso in DC, una stessa variabile può essere stampata in diversi modi a seconda della lettera successiva a V o I. Di seguito è riportata una tabella che riassume i possibili tipi di nome per le variabili di uscita in AC.

Esempio:

Command	*type_variation*	*#points*	*start*	*stop*
.AC	LIN	11	100	200;

L'analisi del transitorio: utilizzando il comando ".TRAN" è possibile realizzare un'analisi sul comportamento del circuito nel tempo. Bisogna indicare in ordine l'incremento che deve essere utilizzato per il plot del segnale, e l'istante finale dell'analisi del transitorio. Si possono indicare anche dei termini opzionali dell'analisi, in ordine l'istante in cui inizare a

Tabella 2.2. Denominazione variabili di uscita in AC.

Variabile d'uscita	Significato
VR o IR	parte reale del valore complesso V o I
VI o II	parte immaginaria del valore complesso V o I
VM o IM	Modulo del numero complesso V o I
VDB o IDB	Valore in dB del modulo, ovvero $20\log_{10}\|V\|$ o $20\log_{10}\|I\|$

stampare i risultati, la dimensione massima dell'incremento utilizzato da Spice per effettuare i calcoli della simulazione (nel caso in cui ad esempio si desideri che l'incremento di computazione sia inferiore a quello per il plot) ed infine l'eventuale valore delle condizioni iniziali che avranno capacità e induttori.Di seguito è riportato un esempio tipico del suo utilizzo:

Esempio:

Command	*plotting_increment*	*final_time*
.TRAN	1e-4	1e-3

Sorgente di onde impulsive: questa sorgente di segnale è usata in combinazione con il comando ".TRAN". Le onde generate con l'istruzione "PULSE" saranno impulsive e i parametri inclusi indicano nell'ordine:

Start è il valore iniziale della pulsazione: per un onda quadra definita fra -5V e 5V il valore iniziale sarà il primo.

End è il valore finale della pulsazione: nel caso descritto sopra il valore finale sarà 5V.

Td è il ritardo (*delay*) dopo il quale inizia l'oscillazione. Il valore di default è 0.

Tr è il tempo di salita (*raise*) del segnale pulsante. Di default assume il valore dell'incremento di plottaggio (del comando ".PLOT") o di stampa (del comando ".PRINT").

Tf è il tempo di discesa del degnale pulsante. Anche in questo caso il valore di default è l'incremento dei comandi ".PRINT" o ".PLOT".

PulWidth indica l'ampiezza della pulsazione ossia il tempo che il segnale ha valore alto. In un'onda quadra con duty cycle del 50% l'ampiezza di pulsazione varrà metà del periodo dell'onda. Di default gli viene attribuito l'istante finale dell'analisi del transitorio (comando ".TRAN").

Period corrisponde al periodo dell'onda pulsante. Nel caso in cui non sia indicato, di default gli viene attribuito il valore ti tempo finale dell'analisi del transitorio.

il valore iniziale della pulsazione, il valore finale, il ritardo (di default è 0), il tempo di salita del segnale, il tempo di discesa, l'ampiezza della pulsazione (il valore di default è l'incremento di ".TRAN") ed infine il periodo (di default è il tempo finale di ".TRAN"). È

possibile trovarlo nella seguente forma:

Esempio:

Command	*Start*	*End*	*Td*	*Tr*	*Tf*	*PulWidth*	*Period*
PULSE(	-5	5	0	1e-5	1e-5	0.1e-3	1e-3)

Il comando .PRINT: questo comando è utilizzato per stampare i valori delle uscite in modo tabulare. I Il codice per richiamare questo comando è ".PRINT", che deve essere seguito dall'etichetta dell'analisi (ad esempio "DC") e dalle variabili da stampare separati da uno spazio. Le variabili di uscita possono essere sia tensioni ai nodi che correnti attraverso sorgenti di tensione.

Esempio:

Command	*type_ analysis*	*output_ variables*
.PRINT	DC	V(1) V(1,2)

In questo caso stampa le tensioni (nel primo caso la tensione è riferita al nodo 0)

Command	*type_ analysis*	*output_ variables*
.PRINT	DC	I(Ra)

In questo caso stampa la corrente dal nodo + al nodo - di Ra

Command	*type_ analysis*	*output_ variables*
.PRINT	AC	VM(1) VDB(1);

In questo caso stampa il modulo della tensione e il suo valore in dB tra i nodi 1 e 0

Il comando .PROBE: questo comando permette di utilizzare il processore grafico Probe messo a disposizione da Spice, che permette di plottare i risultati delle simulazioni. Con Probe è possibile accedere a qualunque punto del grafico ed è possibile ottenere i corrispondenti valori numerici. Infine Probe mette a disposizione molte built-in function che permettono all'utente di calcolare e visualizzare le espressioni matematiche che modellizzano il comportamento del circuito. Il codice Spice per richiamare il comando è ".PROBE", che deve essere seguito dalle variabili che verranno plottate durante la simulazione del codice generato. Nel caso in cui il comando ".PROBE" non sia seguito da variabili, memorizza comunque tutte quelle presenti nel circuito (ed eventualmente nei sottocircuiti) in un file .dat generato da Spice dopo la simulazione.

Esempio:

Command	*output_ variables*
.PRINT	V(1) V(1,2);

Il comando .LIB: questo comando permette di andare a recuperare in una delle librerie a disposizione di Spice, un determinato sottocircuito. Nel caso in esame, l'unica libreria da aggiungere a quelle esistenti è quella degli amplificatori operazionali, come si è visto nel paragrafo YYYYY. La sintassi da utilizzare per richiamare la libreria è la seguente:

Esempio:

Command	*library_name*
.LIB	OpAmp.lib;

Per richiamare il sottocircuito presente nella libreria all'interno del codice Spice, bisogna utilizzare "X" come prima lettera della riga di codice. La sintassi da adottare è descritta di seguito.

Sottocircuito: permette di raggruppare una parte di codice, chiamata sottocircuito, che può essere richiamata più volte all'interno del programma. Ogni sottocircuito deve avere un nome univoco, già come accadeva per gli altri elementi. Deve essere inoltre presente una lista di almeno due nodi che rappresentano le connessioni verso l'esterno del sottocircuito. La prima linea della dichiarazione deve essere ".SUBCKT", seguita dal nome e dalla lista dei nodi mentre l'ultima linea è indicata dal comando ".ENDS". Per richiamare nel programma un sottocircuito dichiarato, si utilizza il comando "X" come prima lettera della linea, seguita dal resto del nome e dalla lista dei nodi dichiarati nella definizione del sottocircuito. L'esempio riportato di seguito si riferisce al codice che implementa un amplificatore operazionale realizzato con un generatore di tensione dipendente.

```
.SUBCKT OpAmp p_in n_in com out
Ex   int   com   p_in   n_in   1e5
Ri   p_in  n_in  500k
Ro   int   out   50.0
.ENDS
```

Il circuito rappresentato è il seguente A questo punto è possibile richiamare il sottocircuito

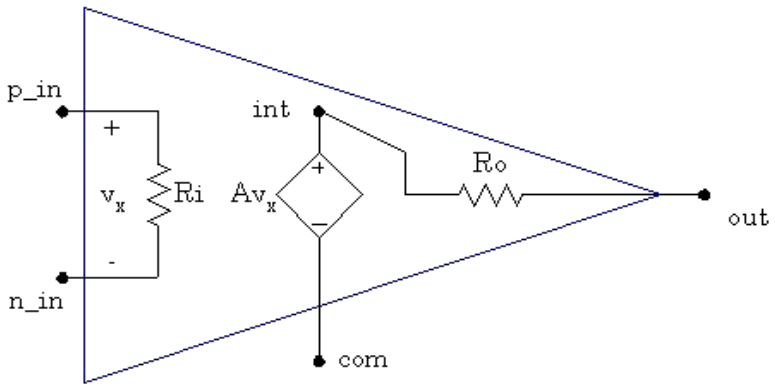

Figura 2.1. Modello amplificatore operazionale

in un programma più esteso come ad esempio quello per implementare un amplificatore invertente:

```
Vg   1     0     DC      50mV
```

```
Rg    1       2       5k
Rf    2       3       50k
RL    3       0       20k
X1    0       2       0       3     OpAmp
.END
```

Il circuito simulato è raffigurato di seguito.

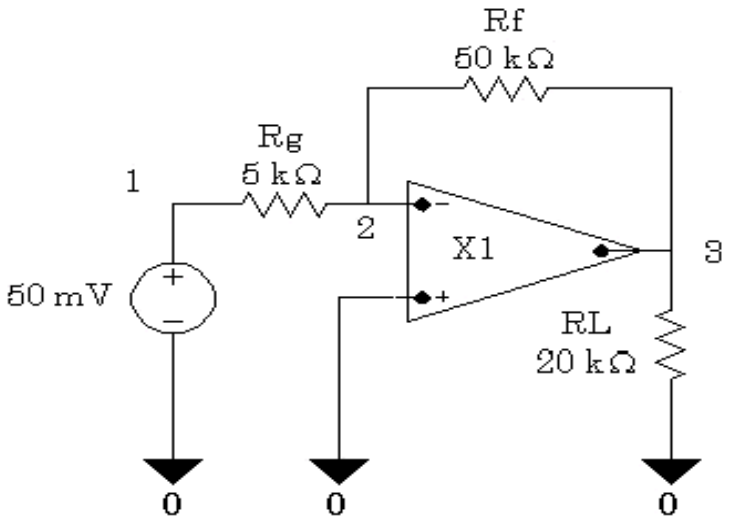

Figura 2.2. Amplificatore Invertente

2.3 Dal diagramma a blocchi alla *netlist*

Con gli algoritmi di ricerca descritti nel capitolo 4 sarà possibile risalire ai componenti elettronici che costituiscono i circuiti equivalenti dei blocchi CodeSimulink. Il passo successivo consisterà nel tradurre il sistema analogico descritto dal diagramma in un file di estensione *.cir*, utilizzato da Spice per simulare il comportamento del circuito analogico globale.

Ad ogni blocco CodeSimulink è associata una descrizione Spice del circuito che rappresenta. Per i blocchi *sim_ gain* e *sim_ sum2* che utilizzano degli amplificatori operazionali, la simulazione Spice risulterà molto accurata, in quanto i modelli dei dispositivi sono realizzati dalle corrispondenti aziende produttrici. Per il blocco *sim_ zoh* è stata realizzata una descrizione Spice adatta solo per scopi simulativi, che non rappresenta individualmente gli amplificatori di Sample & Hold inclusi nel database.

Le simulazioni che ne derivano permettono di avere una stima coerente delle grandezze elettriche presenti nel circuito simulato.

Per passare da un sistema a blocchi ad un file circuito (con estensione .cir), è necessario

effettuare una mappatura del diagramma: ogni blocco viene considerato come un sottosistema a se stante, chiamato *sottocircuito*[1] che è collegato agli altri in un certo modo. Identificando le connessioni fra tutti blocchi presenti nel diagramma è possibile risalire alla lista dei nodi o *netlist* del circuito analogico globale equivalente. In essa sono contenuti esclusivamente i nodi che rappresentano i fili di collegamento fra i vari blocchi/sottocircuiti, ma non sono compresi certi nodi specifici di alcuni circuiti, come le alimentazioni. Sarà sempre il compilatore hardware analogico a correggere, in una seconda fase, i nodi mancanti, andando ad analizzare caso per caso tutti i circuiti rappresentati dai blocchi CodeSimulink.

La descrizione del codice MATLAB^{TM} necessario ad implementare il compilatore hardware analogico sono descritte accuratamente nel paragrafo 5.4.

[1] il termine sottocircuito è utilizzato per analogia con il comando *.SUBCKT* descritto nel paragrafo 2.2 necessario per invocarlo

Capitolo 3

Il database dei componenti

In questo capitolo verranno trattati vari argomenti inerenti la creazione del database contenente i componenti elettronici utilizzati dai circuiti. Per prima cosa verranno spiegate le ragioni che hanno portato alla sua creazione e successivamente si analizzeranno aspetti più tecnici sulla sua realizzazione.

3.1 La necessità di realizzare un database

Si può facilmente stabilire quale circuito analogico esegue la stessa funzione di un blocco CodeSimulink: pensando al blocco di guadagno *sim_ gain* viene subito in mente che un circuito che amplifica un segnale di tensione è facilmente realizzabile con un amplificatore operazionale e due resistenze disposte in modo differente a seconda del segno del guadagno. Questa prima approssimazione risulta però essere insufficiente se vengono stabilite delle specifiche più restringenti sulle caratteristiche del segnale d'uscita, come ad esempio un offset massimo ammesso o un valore massimo del segnale d'uscita. In questo caso è necessario poter scegliere fra diversi componenti quelli che realizzano una controparte analogica del blocco CodeSimulink mantenendo le caratteristiche che aveva in simulazione il segnale d'uscita.

Per scegliere i componenti più adatti a realizzare un circuito sarebbe necessaria un'approfondita ricerca dei componenti disponibili sul mercato ed un'analisi dei loro data sheet anche nel caso in cui fosse necessario conoscere solo un numero limitato di parametri.

Molti siti web che vendono componenti online permettono di effettuare delle ricerche parametriche fra i componenti disponibili. Uno dei leader del settore è il sito *Digikey*[1] che oltre alla possibilità di effettuare ricerche parametriche, permette di scaricare gratuitamente una versione stampabile dei prodotti disponibili e dei relativi prezzi. Sfortunatamente non è possibile risalire ai database utilizzati per effettuare le ricerche e non è possibile estrarre tutte le informazioni necessarie su determinati componenti e scegliere i componenti migliori per un circuito con determinate caratteristiche.

A questo punto si sono cercate strade alternative, passando attraverso i siti web di alcune delle aziende leader nel settore tra cui la *Texas Instruments*, la *Analog Devices* e la

[1] la pagina web è raggiungibile all'indirizzo *http://www.digikey.com*

National Semiconductor. Ognuna di esse mette a disposizione dei database di componenti delle loro linee di produzione. Andando nello specifico:

- sui siti web della *Texas Instruments* e della *Analog Device* sono disponibili dei database che permettono di effettuare una ricerca parametrica dei componenti e di impostare i criteri di selezione desiderati. I database sono anche liberamente scaricabili in formato pdf;

- la *National Semiconductor* permette di scaricare nel suo sito web un software contenente un ampio database di componenti. Il programma permette anche di stabilire dei criteri di selezione dei componenti e di confrontarne fino a due contemporaneamente;

Questi database sono abbastanza completi (non includono tutti i parametri presenti in un data sheet, ma solo quelli principali) ma implicano comunque un lavoro di ricerca e di calcoli a posteriori per verificare che il circuito è effettivamente realizzabile all'interno delle specifiche.

A questo punto si comincia a comprendere la necessità di realizzare un database *ad hoc* che contenga i dispositivi provenienti da più aziende e permetta, fra l'altro, di automatizzare la ricerca dei componenti migliori, non con l'utilizzo delle *query*.[2] ma applicando degli algoritmi di ricerca che considerino il componente all'interno di un circuito elettronico. Come si vedrà in seguito i criteri di ricerca possono essere implementati come degli algoritmi di matlab ed essere facilmente modificati o aggiunti nel caso in cui si desiderino dei criteri di selezione più restrittivi.

Infine bisogna prendere in considerazione un aspetto finora trascurato: nel caso in cui sia possibile trovare più componenti che soddisfano i requisiti di scelta, a questo punto la scelta di uno di essi dipenderà il più delle volte dal prezzo del componente. Utilizzando un database che contenga anche il prezzo dei componenti fra i propri campi, sarà molto semplice implementare un algoritmo per trovare il quello più economico. Di seguito verranno illustrati le modalità di realizzazione del database con il programma Microsoft Access e successivamente l'interfacciamento di quest'ultimo con MATLAB^{TM}.

3.2 I componenti trattati

Come verrà spiegato nel paragrafo 4.1, i circuiti elettronici sono essenzialmente costituiti da due tipi ci componenti, quelli attivi e quelli passivi. Il database *Component_Database* contiene una tabella per ogni componente supportato. Di seguito vengono illustrati i componenti inclusi e le loro caratteristiche.

3.2.1 I componenti *attivi*

I componenti elettronici attivi (detti anche *dispositivi*) attualmente inclusi nel database sono gli amplificatori operazionali e gli amplificatori di Sample & Hold. Essi sono contenuti

[2] si definisce query una forma di interrogazione del database eseguita per estrarre delle informazioni in esso incluse

in tabelle distinte del database *Component_Database*. I nomi attribuiti alle tabelle sono fondamentali poiché l'estrazione dei dati in esse contenute avverà a partire da tali nomi. Questi nomi saranno fondamentali anche nella realizzazione del codice MATLAB^{TM} per-

Tabella 3.1. Nomi assegnati alle tabelle dei componenti attivi

Componente analogico	Nome tabella
Amplificatore operazionale	*OpAmp*
Amplificatore si Sample & Hold	*SHA*
Resistenza	*Res*

ché permetteranno un'identificazione univoca dei parametri appartenenti ad uno specifico dispositivo.

L'amplificatore operazionale é utilizzato in diversi circuiti che rappresentano la controparti analogiche dei blocchi CodeSimulink, come negli amplificatori invertente e non, l'amplificatore sommatore e quello differenziale. Molte aziende producono questo tipo di circuito integrato, il database risultante risulterà infatti abbastanza corposo. I campi inclusi per la tabella dell'operazionale rappresentano i principali parametri che lo descrivono e sono gli stessi comunemente indicati nei database trovati in rete. Di seguito vengono descritti i vari campi e, per ognuno di essi, viene indicata l'unità di misura del dato in esso contenuto. Fra parentesi sono riportati i nomi abbreviati dei campi: essi saranno utilizzati nelle funzioni per estrarre i dati dalla tabella.

Azienda produttrice : (*Producer*) indica il nome dell'azienda a cui appartiene quel determinato componente; sono inclusi i principali amplificatori operazionali provenienti dalle tre aziende indicate nel paragrafo 3.1. Questo campo ha solo valore informativo non è utilizzato in alcun modo negli algoritmi di ricerca.

Nome del componente : (*Name*) indica la sigla alfanumerica con cui è identificato il componente; questi nomi saranno anche utilizzati per definire la lista degli operazionali che rientrano nelle specifiche.

Descrizione del componente : (*Description*) contiene una breve lista delle caratteristiche dell'operazionale. Ha solo valore informativo ma viene utilizzato per informare l'utente sul dispositivo che intende utilizzare.

Alimentazione minima : (*MinSupply*) indica la minima alimentazione positiva che permette il corretto funzionamento del dispositivo. Verrà utilizzato all'interno degli algoritmi di ricerca essendo uno dei parametri principali considerati. Il valore riportato è inteso in V.

Alimentazione massima : (*MaxSupply*) indica la massima alimentazione che permette il corretto funzionamento del dispositivo. Analogamente al valore di alimentazione minimo è utilizzato negli algoritmi di ricerca in quanto parametro determinante per la scelta di un componente piuttosto che un altro. Il valore riportato è inteso in V.

Offset d'ingresso : (*VoltageOffset*) indica la tensione differenziale all'ingresso fra i terminali d'ingresso dell'amplificatore operazionale nel caso in cui non viene applicato alcun segnale all'ingresso del dispositivo[3]. Il suo valore è da intendersi come l'intervallo di tensione entro cui può variare l'offset d'ingresso: 5 mV di offset implicano un offset che può variare da $+5mV$ a -5mV. Viene utilizzato negli algoritmi di ricerca in quanto fattore limitante delle prestazioni dell'operazionale. I valori riportati nel database sono indicati in mV.

Corrente di Bias : (*BiasCurrent*) è definita come la media delle correnti (continue) di polarizzazione che fluiscono nei morsetti invertente e non invertente. Si manifestano quando i terminali sono posti a massa e quindi non viene applicata alcuna tensione d'ingresso. Generalmente il suo valore è piccolo (soprattutto per gli integrati che utilizzano tecnologia MOS) e viene indicato in modo diverso nei database trovati. I valori inclusi nel database sono comunque considerati in nA

Tensione di rumore : (*VoltageNoise*) è indicata come densità spettrale di rumore[4] (ovvero alla tensione di rumore normalizzata su una larghezza di banda di $1Hz$. Per conoscere il rumore in tutto lo spettro è necessario integrare il suo valore quadrato su tutta la banda di lavoro. Anche questo parametro indica un limite dell'amplificatore operazionale ed è necessario verificare che rientri nelle specifiche. I valori riportati nel database sono considerati in $\frac{nV}{\sqrt{Hz}}$.

Slew rate : (*SlewRate*) corrisponde alla velocità massima di variazione della tensione d'uscita è misurato in $\frac{V}{\mu s}$: uno slew rate di 1 $V/\mu s$ indica che l'uscita cresce o diminuisce ad una velocità non superiore di 1 V ogni microsecondo. Questo parametro verrà visualizzato insieme agli altri ma non costituirà una variabile degli algoritmi di ricerca.

Prodotto banda guadagno : (*GainBand*) indica il prodotto fra il guadagno ad anello aperto (senza controllo in reazione) dell'amplificatore operazionale e la sua frequenza in cui il guadagno scende di 3dB. Questo prodotto è costante per tutta la banda di funzionamnto del dispositivo: questo significa che all'aumentare della frequenza diminuirà il guadagno del dispositivo. Come si può vedere dalla figura alla massima frequenza corrisponde un guadagno unitario (0 dB in unità logaritmiche).

Viene utilizzato negli algoritmi di ricerca per ricavare la banda di lavoro dell'amplificatore retroazionato e risalire alla tensione di rumore.

Prezzo : (*Price*) indica il prezzo del componente; è utilizzato in una fase successiva alla ricerca dei componenti per trovare quelli di costo minimo fra quelli che rientrano di per se nelle specifiche. dati riportati nel database si riferiscono al prezzo in dollari.

La tabella degli amplificatori operazionali contiene altri due campi che non sono stati menzionati. Il primo è un campo obbligatorio, chiamato *ID_ OpAmp* (o *ID_ SHA* nel caso

[3] viene anche indicata come la tensione continua di correzione da fornire in ingresso per avere uscita nulla

[4] per maggiori informazioni sul rumore si rimanda al paragrafo A.4

del Sample & Hold), identifica univocamente i vari record inseriti nel database. L'altro campo è chiamato *SpiceDescription* e può contenere o 1 o 0. I valori non nulli specificano se per quel componente è disponibile un modello Spice fornito dall'azienda. Successivamente verrà ampiamente spiegato il ruolo di questo campo.

Per quanto riguarda gli amplificatori di Sample & Hold il discorso si complica un po'. Questo tipo di componente viene utilizzato soprattutto per stabilizzare un segnale che deve essere acquisito da un convertitore analogico/digitale: attualmente questo dispositivo si trova più facilmente integrato all'interno dei convertitori ADC che non come dispositivo a se stante. La tabella *SHA* del database contiene quindi un numero limitato di dispositivi. Gli amplificatori S/H inclusi all'interno del database hanno tutti una capacità di mantenimento interna (essendo presenti in numero maggiore rispetto a quelli con capacità esterna) ma possono avere diverse architetture interne[5]. A causa di queste limitazioni, è risultato difficile trovare degli algoritmi di ricerca che fossero comuni alle diverse architetture e, nonostante fossero disponibili molti parametri specifici per l'amplificatore di Sample & Hold, ne sono stati utilizzato un numero limitato. I campi del database (che rappresentano i suoi parametri) inclusi per questo componente sono i seguenti:

Azienda produttrice : (*Producer*) è il nome dell'azienda che ha realizzato il componente. Solo due aziende producono componenti che soddisfano i requisiti sopra illustrati e sono la *Analog Devices* e la *Intersil*.

Nome del componente : (*Nome*) è la sigla con cui è identificato il dispositivo. Analogamente al caso degli operazionali si utilizzano questi nomi per comporre la lista dei dispositivi utilizzabili.

Descrizione del componente : (*Description*) fornisce un'informazione generale sulle principali caratteristiche del componente. Si visualizzeranno solamente le descrizioni degli amplificatori S/H che rientrano nelle specifiche.

Alimentazione massima : (*MaxSupply*) indica la tensione massima a cui può essere alimentato il dispositivo; è uno dei parametri di confronto utilizzati negli algoritmi di ricerca. I valori riportati nel database sono intesi in V. L'alimentazione minima non è riportata in quanto non indicata in tutti i data sheet analizzati.

Range di uscita : (*FullScale*) detto anche *Full Scale*, indica il valore massimo di tensione che può raggiungere il segnale d'uscita. In tabella è indicato in V.

Tempo di acquisizione : (*AcquisitionTime*)è il massimo intervallo di tempo richiesto per acquisire una nuova tensione d'ingresso dopo che è stato dato il comando di Sample. Un segnale si può definire acquisito quando il suo valore rientra entro un certo errore (generalmente l'1%, lo 0.1% o lo 0.01% del *Full Scale*. È un parametro specifico della transizione dalla fase di Hold a quella di Sample e può essere ridotto diminuendo il valore della capacità di mantenimento (anche se questo causerebbe un incremento del *Droop rate*). Non è utilizzato negli algoritmi di ricerca in quanto

[5]per maggiori chiarimenti sulle architetture dei sample & hold si rimanda al paragrafo A.7

dipendente dal valore della capacità di mantenimento che talvolta non viene indicata nei data sheet. Nel database è indicato in μs.

Tempo di apertura : (*ApertureTime*) conosciuto anche come ritardo di apertura indica è generalmente definito come il tempo che intercorre fra l'istante in cui è stato dato il segnale di hold e l'istante in cui il segnale d'ingresso è effettivamente collegato alla capacità di Hold. Questo parametro è specifico della transizione dalla fase di Sample a quella di Hold. Non è utilizzato come variabile negli algoritmi di selezione ma viene mostrato comunque fra i parametri che identificano il componente. Nel database è indicato in ns.

Jitter d'apertura : (*ApertureJitter*) corrisponde all'incertezza a cui è soggetto il tempo di apertura (è anche conosciuto come *aperture uncertainty*). È dovuto al rumore che si sovrappone al comando di Hold. Viene spesso rappresentato in valore efficace e rappresenta la deviazione standard del tempo di apertura. Questo tipo di errore pone un limite superiore alla massima frequenza del segnale sinusoidale che può essere campionata in modo corretto dal Sample & Hold. Come il tempo di apertura viene indicato in ns all'interno del database.

Tempo di assestamento : (*SettlingTime*) è il tempo necessario affinché la tensione d'uscita si assesti entro un determinato errore di banda dopo che il segnale di Hold è stato dato. Gli errori di banda comunemente specificati sono l'1%, lo 0.1% o lo 0.01% del valore del *Full Scale*. È importante poiché pone un limite superiore alla massima frequenza di campionamento insieme al tempo di acquisizione del S/H e al tempo di conversione dell'ADC. Non è utilizzato negli algoritmi di ricerca non essendo possibile conoscere il tempo di conversione del (successivo) convertitore. Il valore riportato nel database è misurato in μs

Droop Rate : (*DroopRate*) indica di quanto varia la tensione d'uscita a causa delle perdite della capacità di mantenimento. Viene specificata nel data sheet se la capacità del componente è integrata, altrimenti è necessario calcolarla. Questo parametro è importante in applicazioni che richiedono il mantenimento della tensione campionata entro una certa banda di errore. Può essere ridotto incrementando il valore della capacità di mantenimento. Anche questo parametro non è stato utilizzato negli algoritmi di ricerca poiché dipende strettamente dall'architettura interna dell'amplificatore e dal valore della sua capacità di mantenimento. Nel database è indicato in $\frac{\mu V}{\mu s}$

Larghezza di banda (piccolo segnale) : (*FullPowerBandwidth*) è la frequenza alla quale il guadagno (di tensione) scende di 3dB nel caso in cui il segnale d'ingresso (in continua) sia inferiore di almeno 20dB al massimo segnale d'ingresso ammesso (il *Full Scale*). Questo parametro è importante per quelle applicazioni che richiedono la conversione di segnali non troppo ampi ma ad alte frequenze. Anche questo segnale non è attualmente considerato come variabile degli algoritmi di ricerca. Nel database è riportato in Mhz.

Hold step : (*HoldStepOffset*) conosciuto anche come *Pedestal error* o *sample-to-hold offset* è il gradino di tensione che appare in uscita in seguito ad una transizione dalla fase di Sample a quella di Hold. È causato da un trasferimento di carica verso la capacità di mantenimento dovuto all'apertura dello switch. Nel database è riportato in mV.

Risoluzione massima : (*ADCRes*) indica la massima risoluzione in bit che può raggiungere l'amplificatore di S/H. Questo parametro è importante se il dispositivo deve essere connesso ad un convertitore analogico/digitale: è necessario che la risoluzione del componente sia maggiore o uguale a quella del convertitore per evitare di perdere l'informazione. È utilizzato all'interno degli algoritmi di ricerca per verificare se l'amplificatore è in grado di garantire almeno una risoluzione minima. La sua unità di misura nel database è il *bit*.

Prezzo : (*Price*) indica semplicemente il prezzo dell'amplificatore di S/H. Come per il caso degli operazionali è indicato in dollari nel database.

Analogamente al caso dell'amplificatore operazionale è presente un ulteriore campo chiamato *ID_SHA* che permette di identificare univocamente i record inseriti nel database grazie ad una numerazione progressiva. Non è presente invece un campo *SpiceDescription*, non essendo disponibili dei modelli personalizzati per questo tipo di dispositivo. Si ricorda che i nomi dei campi riportati fra parentesi sono quelli utilizzati per estrarre le informazioni di quel determinato campo.

In conclusione, come si è visto molti di questi parametri sono stati utilizzati per verificare che il dispositivo rientrasse nelle specifiche richieste. Ciò non toglie che si possano aggiungere altri parametri per migliorare i criteri di selezione o, più semplicemente, utilizzare alcuni di quelli già presenti (come lo *Slew rate* nel caso dell'amplificatore operazionale) che non sono stati presi in considerazione.

3.2.2 I componenti *passivi*

Attualmente il database *Component_Database* contiene soltanto la tabella per le resistenze. Analogamente al caso dei componenti attivi, la tabella che contiene i vari record è stata nominata nel seguente modo: Nella fase iniziale del progetto non si era previsto di realizzare

Tabella 3.2. Nomi assegnati alle tabelle dei componenti passivi

Componente analogico	Nome tabella
Resistenza	*Res*

delle tabelle specifiche anche per i componenti passivi. Per questo motivo non è stato possibile realizzare un database completo che elencasse i resistori effettivamente presenti sul mercato. Al fine di testare il funzionamento del codice si sono comunque introdotti dei nomi fittizzi di resistori. I campi del database che individuano i parametri specifici delle

resistenze sono i seguenti. Come per i casi precedenti fra parentesi è riportato il nome del campo della tabella.

Azienda Produttrice : (*Producer*) è il nome dell'azienda che produce quella serie di resistenze.

Nome Componente : (*Name*) è il nome attribuito a quella serie.

Descrizione : (*Description*) contiene, in genere, la tecnologia in cui è stato realizzata quella serie di resistenze.

Tolleranza : (*Tolerance*) indica la massima differenza in valore assoluto che intercorre fra il valore nominale e quello reale della resistenza. I dati riportati nella tabella devono essere considerati in percentuale. Nella seguente tabella sono riportati i valori percentuali associati alle corrispondenti serie di resistenze.

Tabella 3.3. Tolleranze associate alle principali serie di resistori

Serie	Tolleranza (%)
E6	20
E12	10
E24	5
E48	2
E96	1
E192	0.5/0.25/0.1

Dissipazione : (*Wattage*) indica la massima potenza che può sopportare la resistenza. I dati introdotti in tabella sono intesi come Watt (W).

Prezzo : (*Price*) indica il costo del resistore espresso in dollari come nei casi precedenti.

3.3 La creazione del database

3.3.1 Impostazione del separatore decimale

Prima di creare un nuovo database in cui inserire i parametri stabiliti nel paragrafo 3.2 può essere necessario modificare delle impostazioni del sistema operativo. Nel caso in cui si utilizzi una versione italiana di Windows (XP o 2000), sarà necessario modificare il separatore decimale utilizzato di default, ovvero la virgola (","). Il programma di calcolo MATLAB™ accetta numeri decimali separati dal punto ("."), è dunque necessario andare nella seguente directory

Start/Impostazioni/Pannello di controllo/Opzioni internazionali

e modificare il campo "*Separatore decimale*" che si trova nella scheda "*Numeri*".

3.3.2 Introduzione a Microsoft Access

Access è un programma di casa Microsoft per la gestione di database, appartenente alla categoria dei DBMS (*Database Management System*).

Viene incluso nel pacchetto Ms Office, ragion per cui esistono diverse versioni del software legate alle edizioni rilasciate di Office; quella utilizzata per creare il database è la versione per Windows XP, sostanzialmente uguale a quella del 2000, su cui è stato comunque testato il corretto funzionamento. Problemi di compatibilità potrebbero verificarsi nel caso in cui si utilizzino versioni di Access precedenti a quella del 97. Le ultime versioni, infatti, non solo supportano più connessioni in contemporanea, ma permettono di creare file di database molto grandi (si passa da 1 Gb a 2Gb).

Per quanto il suo utilizzo sul Web sia diffuso ed ormai anche ben supportato, soprattutto nell'interazione con ASP ed ASP.NET, Access non è un DBMS pensato per il Web, bensì per applicazioni Desktop, implementabili con le maschere di Access stesso o con altri linguaggi di casa Microsoft, come il Visual Basic. Ad ogni modo è possibile interfacciarsi ad Access anche in altri ambiti, ad esempio col PHP e con Macromedia Cold Fusion per il Web, con Java, ed altri per la programmazione Desktop. Il caso esaminato all'interno di questo progetto di tesi prenderà in considerazione solo l'interfacciamento con il software di calcolo MATLAB^{TM}.

Il capitolo proseguirà illustrando le nozioni utilizzate per costruire (e gestire) il database, utilizzabile sia attraverso un linguaggio di programmazione (SQL) che attraverso le funzionalità interne di Access (le maschere).

3.3.3 Le tabelle, le maschere e le query

Avviando il programma Microsoft Access come prima cosa ci si trova di fronte la finestra in figura 3.1 che permette di creare o modificare diversi tipi di oggetti: di seguito verranno trattati quelli principali che sono serviti per realizzare il database dei componenti. Selezionando come oggetto *Tabelle* (o *Table*) è si presentano diverse opzioni che permettono di creare una tabella. Nel caso in cui il file .db contenga già, i loro nomi saranno qui elencati. Le *tabelle* non sono altro che strutture bidimensionali identificate da campi (colonne) e record (righe) in cui vengono riorganizzati i dati che si vogliono inserire nel database. I *record* corrispondono ai dati che si desidera immagazzinare, nel caso trattato nella presente tesi ogni record della tabella *OpAmp* corrisponde ad un diverso amplificatore operazionale di cui si desidera memorizzare una serie di valori come ad esempio il nome o la tensione di alimentazione. Ogni valore distinto che si desidera memorizzare deve essere incluso in un *campo* specifico che riporti per l'appunto, il tipo di dato da salvare. A questo punto si può scegliere la modalità di creazione della tabella:

- si può creare una tabella *ad hoc* indicando il nome dei campi e il loro tipo selezionando l'opzione "*Design view*"
- si può creare una tabella scegliendo fra una lista di campi prefabbricati (sconsigliato).
- si può creare una tabella inserendo direttamente i dati che si intende includere in ogni campo.

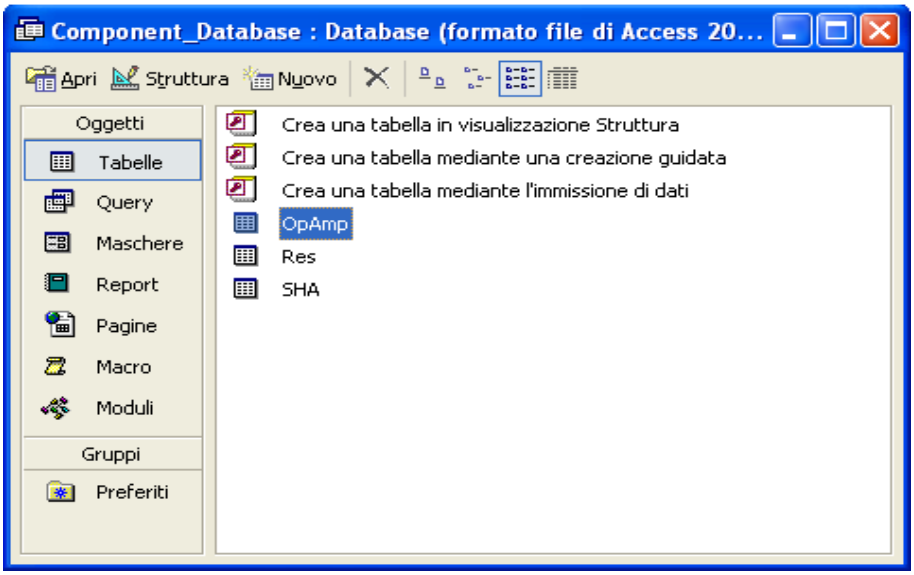

Figura 3.1. Finestra di selezione di tabelle, maschere e query

Si è scelto di utilizzare la prima opzione poiché permette di modificare la tabella in modo semplice, di aggiungere campi e di personalizzarli in maniera semplice.

Per aggiungere i campi alla struttura è necessario intervenire nella *Design view* che permette di introdurre oltre ai campi anche il tipo di dato (numerico, testuale ...) che esso conterrà. In tabella sono riportati i principali i tipi di dati disponibili e la relativa spiegazione di ognuno di essi. È possibile settare il numero di cifre decimali che possono accettare i campi di tipo *Numerico*.

Le *maschere* rappresentano l'interfaccia grafica mediante la quale l'utente "dialoga" con il database. esse permettono sia la visualizzazione dei dati esistenti che l'inserimento di nuovi record. Sono state utilizzate nella fase di aggiunta dei dati nel database ma non sono indispensabili per la sua realizzazione. Per poterle creare si seleziona l'oggetto *Maschere* (*Form* per la versione inglese) nella finestra iniziale e si sceglie come realizzare la maschera:

- utilizzando un wizard disponibile che mette a disposizioni delle maschere preconfezionate in cui è sufficiente indicare i campi che si desidera aggiungere;

- realizzando una maschera personalizzata disponendo gli oggetti grafici necessari (*Design view*).

Il termine *query* vuol dire letteralmente interrogazione, ma si utilizza per indicare in modo generale, le operazioni che è possibile effettuare su una o più tabelle di un database. Come per i casi precedenti è possibile realizzare una query con un approccio visuale oppure scrivere il codice SQL (*Structured Query Language*) necessario per realizzare una determinata interrogazione. Questo tipo di struttura non è stato utilizzata nel presente

Tabella 3.4. Tipi di dato ammessi in Access

Tipo di dato	Descrizione
Testo	Accetta caratteri alfanumerici e simboli fino a 255 caratteri spazi compresi
Memo	Raccoglie dati di tipo *Testo* per una lunghezza di gran lunga superiore ad un campo *Testo* reale
Numerico	Per default accetta numeri interi, ma può essere impostato anche su numeri con porzioni decimali
Data/ora	Accetta vari formati della data e dell'ora
Valuta	Accetta la valuta corrente in base alla lingua del programma.È possibile scegliere altre valute scaricando gli aggiornamenti dal sito della Microsoft
Contatore	Campo numerico ad incremento automatico di una unità per ogni nuovo record
Si/No	Campo booleano, accetta True e False e graficamente è rappresentato da un quadrato nero spuntabile, dove il segno di spunta rappresenta True

progetto. Le interrogazioni del database dovevano essere eseguite da MATLAB^{TM} per avere a disposizione i dati da inserire negli algoritmi di ricerca.

A questo punto si anno tutte le informazioni necessarie per realizzare il database di componenti, non è dunque più necessario soffermarsi sulla creazione del database stesso ma è possibile passare alla fase d'interfacciamento con il programma di calcolo scelto (MATLAB^{TM}).

3.3.4 Manutenzione del database

Un limite di Access è legato alla quantità di spazio occupato su disco che aumenta notevolmente non appena si effettuano poche e semplici operazioni. Un file di Access 2000 appena salvato occupa circa un centinaio di Kb ed è sufficiente creare una tabella ed inserire pochi dati per occupare circa 3 Mb su disco.

Il database dei componenti dovrà contenere diverse tabelle (una per ogni componente elettronico supportato) e alcune di esse possono arrivare a contenere oltre 1000 elementi inoltre ogni volta che l'utente effettua una ricerca, inserisce, modifica e cancella dati la dimensione del database aumenta ulteriormente.

Access mette a disposizione una funzione per compattare il database, in modo da ottimizzare lo spazio su disco e da migliorare le prestazioni. Per accedere a questa opzione bisogna seguire il percorso indicato:

Strumenti / Utilità database / Compatta e ripristina database

3.4 L'interfacciamento con MATLAB™

In questo paragrafo verrà presentato il codice realizzato per permettere la comunicazione fra il database appena realizzato con Access e il il software di calcolo MATLAB™. Si è scelto di includere la descrizione del codice in questo capitolo per fornire al lettore una visione più generale del lavoro eseguito sul database, dalla fase iniziale di definizione fino al suo utilizzo finale.

3.4.1 Il Database Toolbox

Il *Database Toolbox* è una collezione di costrutti da utilizzare in MATLAB™ per abilitare lo spostamento (sia importazione che esportazione) di dati fra MATLAB™ e i database realizzati con i più software più diffusi. Questo toolbox rende quindi possibile ricavare dei dati numerici da un database esterno, effettuare ogni sorta di calcolo ed analisi all'interno di MATLAB™ ed infine immagazzinare i dati calcolati nello stesso database o in uno diverso. In questa sezione sarà trattata solo la fase di *import* dei dati, non essendo richieste modifiche al database esistenti o salvataggio dei dati in altri file. Questo tool ha comunque le seguenti caratteristiche:

- permette di gestire più database in una singola sessione;
- permette di estrarre sia dati numerici che stringhe di dati e di memorizzarli in matrici o array di celle
- si possono estrarre solo determinati set di dati utilizzando determinate query modificabili di volta in volta all'interno della stessa connessione.
- è presente il *Visual Query Builder*[6] un'interfaccia grafica del toolbox che permette di estrarre i dati anche con una limitata conoscenza dell SQL.

I limiti di questo toolbox sono legati soprattutto alla velocità di estrazione dei dati che, nel caso di database grandi può richiedere anche diversi secondi.

3.4.2 La connessione del database a MATLAB™

Prima di proseguire con la stesura del codice necessario ad estrarre i dati, è necessaria far conoscere a MATLAB™ la fonte dei dati. Poiché si sta utilizzando una piattaforma Windows, i driver da utilizzare sono gli ODBC (*Open Database Connectivity*). Le operazioni da eseguire sono di seguito elencate:

- dal menu **Start** si segue il percorso

[6] è possibile accedere al VQL digitando *querybuilder* nella Command Window di MATLAB™

Impostazioni / Pannello di controllo / Strumenti di amministrazione

e si apre il programma **Origine dati (ODBC)** (**ODBC Data Sources** nella versione inglese);

- si seleziona il tab chiamato **DSN Utente** (**User DSN** nella versione inglese);
- cliccando su **Aggiungi** si sceglie il tipo di driver che utilizza il database (nel caso in esame è ovviamente il driver di Microsoft Access).
- scelto il driver dovrebbe apparire una finestra di configurazione che richiede di inserire un nome per il database connesso (non necessariamente quello del database sorgente) e (volendo) una breve descrizione.
- cliccando su **Seleziona** comparirà un ulteriore finestra in cui si potrà selezionare il database sorgente. Nel caso in analisi il database si trova nella seguente directory a partire dalla cartella in cui è stato salvato CodeSimulink:

CodeSimulink / Compiler / analog

- selezionando il database dei componenti *Component_ Database* si clicca su OK salvando le impostazioni scelte.

A questo punto il database è pronto per essere letto da MATLAB^{TM}, è necessario scrivere il codice per aprire la connessione, ed estrarre i dati desiderati. Si ricorda che questa connessione deve essere eseguita nei seguenti casi:

- se tale connessione non è mai stata eseguita prima su quel computer;
- se si è cambiato il nome del database sorgente.

Nel caso in cui l'utente dimentichi di indicare il database sorgente, il programma realizzato fornirà comunque tutte le informazioni qui elencate per effettuare con successo la connessione con MATLAB^{TM}. Nel caso in cui si verificassero altri problemi si rimanda al manuale del Database Toolbox presente all'indirizzo qui indicato.

3.4.3 La funzione *data_ extraction*

A questo punto è possibile illustrare i comandi MATLAB^{TM} utilizzati per poter estrarre i dati. Il programma è presente nell'appendice B. Il nome del database sorgente è memorizzato in una stringa di questa funzione, nel caso lo si volesse cambiare si ricorda che sarà necessario rieffettuare la connessione del database a MATLAB^{TM} illustrata nel paragrafo 3.4.2. La funzione riceve in ingresso un solo argomento, ovvero la stringa *Component* che identifica la tabella del database *Component_ Database* da cui saranno estratti i dati. In uscita restituirà due array di celle, la prima, chiamata *dati*, conterrà in una sorta di matrice tutti i record e i campi estratti dalla tabella, la seconda (chiamata *nomicolonne*) conterrà

invece tutti i nomi dei campi estratti dalla tabella. Tali nomi costituiranno il riferimento necessario per estrarre i dati specifici di ogni record. Le operazioni eseguite dalla funzione sono le seguenti:

- si stabilisce il tempo per tentare una connessione al database con il comando *logintimeout*.
- si apre la connessione con il comando *database* includendo fra gli argomenti d'ingresso la stringa contenente il nome del database, e il nome utente e la password. Nel presente caso si è preferito non utilizzare password sul database, quindi è sufficiente includere delle stringhe vuote. La funzione restituirà una struttura dati valida (chiamata *conn* nella funzione) contenente le informazioni della connessione.
- si utilizza il comando *ping(conn)* necessario per determinare lo stato della connessione. Nel caso in cui questa non sia avvenuta un messaggio di errore verrà visualizzato e si riporteranno le istruzioni precedenti necessarie a settare il database come sorgente dati.
- si utilizza il comando *exec* per estrarre un determinato numero di campi della tabella. Il primo argomento corrisponde allo stato della connessione *conn*, mentre il secondo è una stringa contenente una query scritta in SQL. Poiché si desidera estrarre tutti i campi presenti nella tabella, la query utilizzata sarà del tipo:

  SELECT * FROM *Nome_ Tabella*

  dove l'asterisco indica che saranno estratte tutti i campi presenti nella tabella *Nome_ Tabella*. La funzione *exec* restituirà un oggetto cursore chiamato *curs*.
- si definisce il formato in cui verranno estratti i dati con il comando *setdbprefs* che riceve in ingresso la stringa 'DataReturnFormat' e la stringa corrispondente al tipo di dato: poiché nella tabella sono presenti sia stringhe di caratteri che numeri il tipo di dato obbligatorio è l'array di celle (l'argomento da utilizzare sarà 'cellarray').
- si estraggono i dati con l'istruzione *fetch* che riceve in ingresso il cursore *curs* e restituisce una struttura dati contenente (nel campo *Data*) l'array di celle con i dati della tabella *Component*.
- in ultimo si estraggono i nomi dei campi della tabella con il comando *columnnames* che riceve in ingresso il cursore *curs*. La funzione restituisce una stringa con tutti i nomi separati da virgole: si utilizza la funzione *get_ string_ piece* descritta nel capitolo 5.5.1 per memorizzare il un array di celle le stringhe dei nomi.
- si può chiudere la connessione con le istruzioni *close(conn)* e *close(curr)*.

Nel caso in cui si fossero verificati degli errori la funzione *data_ extraction* restituirà sempre e comunque due celle vuote.

Capitolo 4

Gli algoritmi per la scelta dei componenti

In questo capitolo verranno analizzati i circuiti scelti per rappresentare le controparti analogiche dei blocchi CodeSimulink e verranno illustrati gli algoritmi di calcolo che permettono di ricavare i componenti reali per poter realizzare fisicamente i circuiti.

4.1 Introduzione: i componenti elettronici di un circuito

Per trovare una possibile rappresentazione analogica di un blocco CodeSimulink, è necessario identificare con quali componenti analogici è realizzato il suo circuito equivalente. Tipicamente, i componenti elettronici di cui è composto un circuito possono essere suddivisi nelle seguenti categorie:

Passivi : componenti elettronici come resistori, capacitori e induttori che non richiedono una sorgente di energia per eseguire una determinata funzione.

Attivi : componenti elettronici che forniscono un *guadagno*[1] in un circuito elettronico e dunque richiedono una sorgente di energia esterna per poter funzionare correttamente. Tra questi componenti si ricordano i *transistor*, i *diodi* e i *MOSFET*. I circuiti integrati sono a loro volta da considerarsi attivi, in quanto includono questo tipo di componenti.

Un circuito elettronico comune che realizza una determinata funzione, può essere costituito di componenti appartenenti ad una sola delle categorie elecate o ad entrambe. Per poter trovare un circuito analogico che modella un determinato blocco CodeSimulink sarà dunque necessario partire dalla sua funzione per poi risalire al circuito che si comporta in modo analogo a livello elettronico. Di seguito sono riportati i blocchi analizzati, la loro funzione e l'equivalente elettronico con cui sono stati rappresentati. Alcuni di questo circuiti

[1] In elettronica il guadagno è generalmente inteso come il rapporto fra un segnale di uscita ed un segnale d'ingresso: un guadagno maggiore di uno implica che ci sia stato un proporzionale aumento di tensione o potenza all'interno del circuito

Tabella 4.1. Blocchi CodeSimulink trattati.

Blocco Esteso	Funzione	Circuito/i Analogico
sim_gain	amplifica il segnale d'ingresso di un valore noto	*amplificatore non invertente e amplificatore invertente*
sim_sum2	somma due segnali d'ingresso	*amplificatore sommatore e amplificatore differenziale*
sim_zoh	campiona il segnale d'ingresso e lo mantiene costante per il periodo di campionamento	*amplificatore Sample & Hold*
sim_constant	genera un segnale costante in uscita	*partitore di tensione*

conterranno esclusivamente componenti passivi, come ad esempio il partitore di tensione realizzabile con due resistori in serie, oppure utilizzeranno solo componenti attivi, come ad esempio il Sample & Hold amplifier che è un circuito integrato a se stante oppure entrambi come ad esempio l'amplificatore invertente composto da due resistenze e un amplificatore operazionale (ossia un circuito integrato).

4.2 Le specifiche dei blocchi analogici

In ogni blocco esteso di CodeSimulink è possibile indicare le caratteristiche che dovrà avere l'equivalente segnale analogico in simulazione. Attraverso quei parametri che lo descrivono è possibile risalire ai componenti elettronici di un circuito reale che che permettono di ottenere un analoga uscita. Si ricorda che il segnale è inteso in tensione sia in ingresso che in uscita, di conseguenza nella determinazione degli algoritmi di ricerca bisognerà considerare che i parametri analogici, descritti di seguito, sono riferiti ad un segnale in tensione. Per ulteriori spiegazioni su come modificarne il valore si rimanda al paragrafo 5.2.

Max Gain Error(random) Specifica l'errore di guadagno casuale e simula l'effetto di discrepanze nei parametri dei circuiti analogici. L'effetto finale è una variazione casuale del segnale che potrebbe essere più grande o più piccolo di un valore casuale. Un errore di guadagno pari a 0.01, per esempio, specifica che l'uscita potrebbe essere fino all'1% più grande o all'1% più piccolo del valore atteso.

Max Offset(random) Specifica l'offset casuale che simula l'effetto di discrepanza nei parametri analogici già considerata per l'errore di guadagno. Poiché il segnale di uscita è considerato i tensione, l'effetto di offset considerato sul segnale sarà misurato in Volt. Un offset pari a 0.01 indicherà che il segnale di uscita sarà più alto o più basso, fino a 0.01V rispetto al valore atteso.

Non Linearity (Polinomial Coeffs) Specifica il vettore $[a_1 \quad a_2 \quad a_3 \quad \ldots]$ dei coefficienti polinomiali che permettono di rappresentare la non linearità del segnale d'uscita nel

seguente modo:

$$y = a_1 \cdot x + a_2 \cdot x^2 + a_3 \cdot x^3 + \dots$$

Dove ogni a_i è espresso in tensione come V^{-i}. Questo parametro non viene utilizzato da nessuno degli algoritmi di ricerca in quanto richiederebbe conoscenza approfondita della struttura interna dei dispositivi per poter definire un modello di calcolo sufficientemente accurato.

White Noise std.dev. Specifica la deviazione standard di un rumore a media nulla che affligge il segnale di uscita in tensione. Viene considerato in $Vrms$.

Lower Saturation Specifica il valore minimo al di sotto del quale non può andare il segnale d'uscita del blocco. È inteso in V.

Upper Saturation Specifica il valore massimo oltre il quale non può andare il segnale d'uscita del blocco. È inteso in V.

Questi parametri sono quelli generalmente utilizzati per descrivere le controparti analogiche dei blocchi estesi di CodeSimulink, compresi quelli elencati nella tabella 4.1. Nella maggior parte dei casi non saranno utilizzati tutti i parametri elencati per descrivere il modello analogico del blocco, poiché in alcuni dei circuiti equivalenti, ad esempio il partitore di tensione, non ha senso definire una tensione di alimentazione in quanto il circuito è formato esclusivamente da componenti passivi. In altri casi (come ad esempio si vedrà con il blocco *sim_ zoh*) alcuni parametri non sono specificati all'interno dei data sheet dei circuiti integrati e non è dunque possibile utilizzarli come termini di ricerca.

Di seguito verranno analizzati individualmente i circuiti elettronici scelti per rappresentare i blocchi CodeSimulink e per ognuno di essi, oltre a fornire una descrizione dei componenti da cui è costituito, verrà data una descrizione degli algoritmi necessari trovare i componenti più adatti a partire dai parametri analogici effettivamente utilizzati.

4.3 I circuiti equivalenti dei blocchi CodeSimulink

In questo paragrafo vengono illustrati i circuiti scelti per rappresentare le controparti analogiche dei blocchi CodeSimulink. Per non appesantire la descrizione si è scelto di non includere le equazioni matematiche che descrivono i vari circuiti ma di riportarle in appendice. Per ulteriori approfondimenti si rimanda al capitolo A.

4.3.1 Il blocco *sim_ gain*

Il blocco *sim_ gain* prende in ingresso un segnale e lo riporta in uscita moltiplicato per il valore costante memorizzato nel parametro "Gain" del blocco. Può essere agevolmente modificato nel box di dialogo principale che si apre cliccando sul blocco, modificando il valore di guadagno esistente e salvando (premendo i pulsanti "Apply" o "Ok"). I circuiti analogici che descrivono in modo completo il blocco sono legati al valore del guadagno che devono rappresentare:

- se *Gain >1* il blocco è assimilabile ad un amplificatore non invertente che amplifica semplicemente la tensione che riceve in ingresso. Il circuito è costituito da due resistenze e un amplificatore operazionale e viene descritto nel paragrafo 4.4.1.

- se *Gain <0* il blocco è assimilabile ad un amplificatore invertente che inverte una tensione e la amplifica. Anche in questo caso il circuito è costituito da due resistenze e un amplificatore operazionale ed è descritto nel paragrafo 4.4.2.

- se *Gain=1* il blocco è assimilabile ad amplificatore inseguitore, utilizzato in genere per disaccoppiare ingresso e uscita in applicazioni che richiedono di limitare gli effetti di carico o dove è necessaria un'interfaccia per connettere un dispositivo ad alta impedenza ad uno a bassa. Il circuito è realizzato con un'amplificatore operazionale in cui l'ingresso non invertente è collegato al segnale mentre quello non invertente è connesso all'uscita. Il presente circuito non verrà ulteriormente trattato

- se *0<Gain<1* il blocco è assimilabile ad un partitore di tensione che restituisce una frazione della tensione d'ingresso in uscita. Il circuito, costituito da due resistenze in serie, è descritto nel paragrafo 4.4.3.

4.3.2 Il blocco *sim_ sum2*

Il blocco *sim_ sum2* prende in ingresso due segnali distinti e ne restituisce una somma algebrica che dipende dalla polarità indicata per ciascuno dei due ingressi. L'informazione sul segno che deve essere assegnato ad ogni ingresso è contenuto nel parametro "Direction" interno al blocco. Si possono avere tre circuiti equivalenti a seconda delle polarità indicate per ogni ingresso:

- se Direction = "+ +" il blocco è rappresentabile come un sommatore non invertente che riceve in ingresso al morsetto non invertente dell'operazionale entrambi i segnali. Il circuito corrispondente è descritto nel paragrafo 4.4.5.

- se Direction = "+ -" o "- +" il blocco è rappresentabile come un amplificatore differenziale in cui il segnale con polarità positiva entra nel morsetto non invertente dell'operazionale. Questo circuito è spesso utilizzato come stadio d'ingresso per linee bilanciate o come blocco di confronto in sistemi retroazionati, cioè in quelle applicazioni che richiedono la rilevazione di una differenza di tensioni. Il circuito corrispondente è descritto nel paragrafo 4.4.4.

- se Direction = "- -" il blocco è rappresentabile come un sommatore invertente in cui entrabi i segnali sono entranti nel morsetto invertente dell'operazionale. Il circuito corrispondente è descritto nel paragrafo 4.4.6.

Come si può notare tutti i circuiti sono costituiti da un amplificatore operazionale più un certo numero (3 o 4) di resistenze che devono avere il medesimo valore sia che si trovino su un ramo d'ingresso che su quello di reazione. Solo in questo modo la somma algebrica dei due segnali non risulterà pesata.

4.3.3 Il blocco *sim_ zoh*

Il blocco *sim_ zoh* permette di campionare con un certo periodo, indicata dal parametro "*sim_ Sample_ Time*" e di mantenere costante il valore acquisito per tutto il periodo di campionamento. A livello analogico è assimilabile ad un amplificatore di Sample & Hold con guadagno unitario. Diversamente dai circuiti precedenti che erano composti sia da componenti attivi (gli amplificatori operazionali) che passivi (le resistenze), il S/H amplifier è presente come componente attivo integrato, che dovrà essere cercato all'interno del database di componenti. In realtà sul mercato sono presenti pochi dispositivi Sample & Hold poiché attualmente si preferisce integrali all'interno dei sistemi di conversione analogico/digitale in cui vengono generalmente utilizzati. Essi permettono infatti di mantenere costante un segnale per un determinato intervallo di tempo affinché venga correttamente campionato e convertito dal sistema di acquisizione. Inoltre, il circuito dell'amplificatore di Sample & Hold non è realizzato in maniera standard, ma può avere diverse architetture a seconda della specificità dell'applicazione in cui è utilizzato: nel caso sià richiesta un'alta velocità di conversione si preferiscono le architetture *Open-Loop* non limitate da reazioni interne, mentre se è richiesta una maggiore precisione nella conversione si utilizzano architetture *Closed-Loop*, meno veloci ma più precise. Il database per questi componenti è costituito da un numero limitato di amplificatori di Sample & Hold con capacità di mantenimento integrata.

4.3.4 Il blocco *sim_ constant*

Questo blocco può avere differenti rappresentazioni analogiche che dipendono dal significato che assume il blocco all'interno del circuito. Una tensione stabile in un circuito elettronico può avere una delle seguenti rappresentazioni:

- come partitore di tensione in cui la tensione sorgente proviene dall'esterno (ad esempio l'alimentazione).
- come componente attivo (regolatore di tensione) che genera una tensione fissa a partire da una determinata tensione di alimentazione;
- come componente attivo (diodo Zener) su cui cade una tensione fissa utilizzata come valore costante

Attualmente il blocco *sim_ constant* è stato rappresentato come un partitore di tensione, a cui è stato aggiunto un amplificatore inseguitore (buffer) per aumentarne la precisione e limitare gli assorbimenti di corrente nel caso in cui il riferimento di tensione sia utilizzato da altri dispositivi.

Il caso di tensioni fisse provenienti dall'esterno del circuito, non viene trattato dal blocco *sim_ constant*, ma dal blocco *sim_ analogIn* descritto nel paragrafo 4.3.5.

4.3.5 il blocco *sim_ analogIn*

Questo blocco permette di interfacciare i blocchi Simulink™ che rappresentano delle sorgenti di segnale analogico, con quelli CodeSimulink che rappresentano il circuito vero e

proprio. In realtà *sim_ analogIn* non ha un circuito analogico equivalente[2] ma nella fase di compilazione del codice Spice, assume il ruolo di un generatore di tensione d'ingresso. Utilizzandolo in combinazione con il blocco *To Workspace* presente fra i *sink* di Simulink™, permette di rappresentare il segnale proveniente da un qualsiasi blocco sorgente di Simulink™ o CodeSimulink come un'onda lineare definita a tratti: in questo modo è possibile rappresentare sia tensioni sinusoidali che tensioni costanti provenienti dall'esterno del circuito.

4.4 Gli algoritmi di ricerca dei componenti

Gli algoritmi di calcolo dei componenti possono essere suddivisi in tre tipologie distinte:

- gli algoritmi che permettono di determinare i *valori numerici* dei componenti passivi utilizzati. Vengono immagazzinati in un'unica stringa presente nell'opzione *PassCompAlgorithm* della "_ParamUI" del blocco CodeSimulink che si desidera descrivere analogicamente.

- gli algoritmi che permettono di scegliere fra i componenti passivi disponibili all'interno del database *Component_ Database* quelli che rientrano in determinate specifiche di precisione e di consumo. Vengono immagazzinati in un'unica stringa sotto l'opzione [NomeComponentePassivo _ *Algorithm*] della "_ParamUI" del blocco CodeSimulink che si desidera descrivere analogicamente.

- gli algoritmi che permettono di scegliere fra i componenti attivi (dispositivi) presenti nel database *Component_ Database* quelli che rientrano nelle specifiche imposte dai parametri analogici descritti nel paragrafo 4.2. Come nel caso precedente, questi algoritmi vengono immagazzinati in un'unica stringa sotto l'opzione [NomeComponenteAttivo _ *Algorithm*] della "_ParamUI" del blocco CodeSimulink che si desidera descrivere analogicamente.

Si ricorda, come già spiegato nel capitolo 3, che nel database non sono elencati tutti i parametri presenti nel data sheet del dispositivo, ma solo quelli principali comunemente utilizzati per scegliere un componente piuttosto che un altro.

Gli algoritmi di ricerca si trovano in realtà sotto forma di disequazioni che, contemporaneamente verificate, forniscono la lista dei componenti che soddisfano tutte le specifiche considerate. Di seguito è riportato un esempio di disequazione:

Formula	*Segno*	*Parametro Analogico*
$\frac{V_{Supply+}}{2}$	$>=$	$max(UpperSaturation, abs(LowerSaturation))$

I parametri presenti in *Formula* contengono i valori numerici estratti dal database per un determinato parametro oppure sono stati calcolati in precedenza dagli algoritmi dei componenti passivi. Con *Parametro Analogico* invece s'intende una combinazione di uno o più parametri utilizzati per descrivere la controparte analogica del blocco CodeSimulink

[2] in realtà il blocco *sim_ analogIn* può essere interpretato come un connettore d'ingresso

come spiegato nel paragrafo 4.2). Questa disequazione in MATLAB^{TM} restituirà un vettore di uni e zeri in cui ogni elemento rappresenta un componente presente nel database: se l'n-esimo elemento del vettore vale 1, allora l'n-esimo componente del database avrà soddisfatto la condizione imposta dalla disequazione, altrimenti quel componente verrà escluso dalla lista finale di quelli utilizzabili. Per un approfondimento sul codice MATLAB^{TM} per la ricerca dei componenti attivi e passivi si rimanda alla funzione descritta nel paragrafo 5.3.1.

4.4.1 L'amplificatore non invertente

In figura 4.1 è riportato il circuito equivalente dell'amplificatore non invertente. Per prima

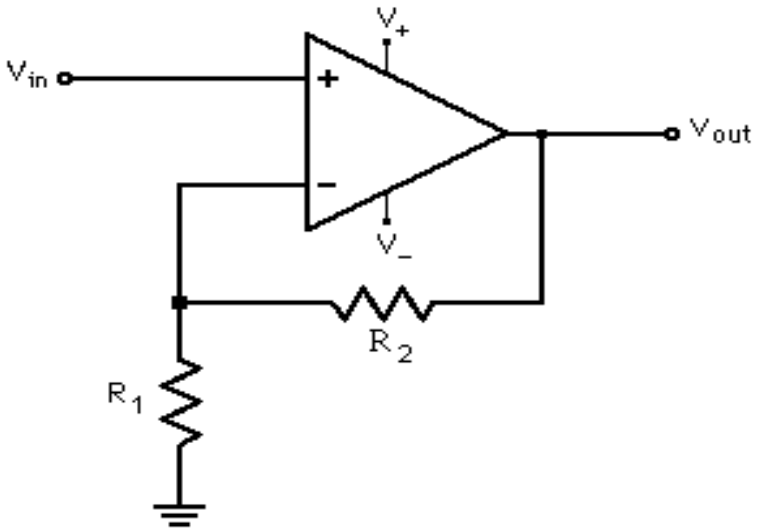

Figura 4.1. Amplificatore Non Invertente

cosa è necessario determinare il valore numerico delle resistenze $R1$ e $R2$ con l'algoritmo presente nell'opzione '*PassCompAlgoritm*' presente nella "_ParamUI" del blocco *sim_gain*. Inoltre verranno restituiti anche l'errore di guadagno reale e la tolleranza dei resistori, entrambi determinati a partire dai valori di resistenza trovati.

A questo punto si procede con la scelta dell'amplificatore operazionale che più si adatta alle specifiche introdotte. In particolar modo si è tenuto conto dei parametri riportati in tabella seguente.
Lower Saturation ($MinSat$) e *Upper Saturation* ($MaxSat$) pongono un limite all'alimentazione massima: sono ammessi quei dispositivi in grado di coprire un range di tensione che va dal valore minimo di saturazione a quello massimo. In termini di equazione si può dire

$$\frac{Dev\_MaxSupply}{2} >= max(MaxSat,abs(MinSat)) \tag{4.1}$$

Tabella 4.2. Parametri Analogici per l'amplificatore non invertente

Parametro Analogico	Nome Variabile Algoritmo
Lower Saturation	MinSat
Upper Saturation	MaxSat
Max Offset	MaxOffset
Max Noise	MaxNoise

Dove $Dev\_MaxSupply$ corrisponde all'alimentazione massima accettata dal dispositivo. Nel caso in cui $MinSat$ sia nulla, il range da coprire risulterà ridotto e l'equazione equivalente sarà data da

$$Dev\_MaxSupply >= MaxSat \tag{4.2}$$

Max Offset ($MaxOffset$) è da intendersi come il massimo offset del segnale in uscita quando in ingresso è presente una tensione nulla. Bisogna tener conto sia dell'offset d'ingresso implicito dell'amplificatore operazionale sia della corrente di Bias dell'amplificatore, la formula che ne risulta è dunque

$$Dev\_Vos \cdot A + Dev\_Ibias \cdot R2 <= MaxOffset \tag{4.3}$$

In questo caso Dev_Vos corrisponde all'offset d'ingresso dell'amplificatore, ovvero la tensione presente fra i due terminali dell'operazionale, mentre Dev_Ibias corrisponde alla corrente di Bias che entra in entrambi i terminali.

Queste formule sono state ricavate analizzando il circuito dell'amplificatore non invertente nel caso in cui la tensione d'ingresso sia nulla. Per ulteriori spiegazioni si rimanda al capitolo A.3

Max Noise ($MaxNoise$) è da intendersi come la massima tensione di rumore in uscita dall'amplificatore. Poichè viene dato come deviazione standard rispetto ad un rumore a media nulla, si considera espresso come valore efficace. Analogamente all'offset è dato dalla somma di più contributi di rumore scorrelati fra di loro. Per l'amplificatore invertente le principali componenti di rumore sono:

- il contributo di rumore dell'amplificatore operazionale che si manifesta all'ingresso del dispositivo stesso; è indicato come densità spettrale di rumore nei data sheet.
- il contributo di rumore dovuto alle resistenze del circuito, per non indicare troppe notazioni si è scelto di indicarlo anch'esso come densità spettrale di tensione (di rumore) quindi come

$$Nn = 4K \cdot R \cdot T$$

L'algoritmo che si ottiene è dunque il seguente:

$$CF \cdot \sqrt{A^2 \cdot Dev\_Vn^2 + \left(\frac{4K \cdot T}{R1} + \frac{4K \cdot T}{R2}\right) R2^2} <= MaxNoise \tag{4.4}$$

Dove

- CF è un fattore di correzione che permette di effettuare il confronto fra un rumore espresso in valore efficace (*MaxNoise*) e un rumore espesso come densità spettrale, misurate in $\frac{nV}{\sqrt{Hz}}$). Il fattore di correzione permette di integrare nella banda di lavoro le densità spettrali dei rumori.

  Per ottenere il valore numerico della banda di funzionamento dell'operazionale si deve partire dal suo prodotto banda-guadagno, incluso nel database. Poiché tutti i dispositivi presenti in *Component_Database* sono compensati internamente, il *GBP*[3] sarà costante in tutto il range di frequenze. Moltiplicando il guadagno del circuito di amplificazione A_f per questo prodotto è possibile risalire alla massima banda supportata. Si ha quindi:

$$Band = \frac{GainBand}{A_f}$$

  Poiché non ci sono altri termini in frequenza integrare la densità di rumore significa moltiplicare per la banda. A questo punto il fattore di correzione varrà:

$$CF = \sqrt{Band} = \sqrt{\frac{GainBand}{A_f}} \tag{4.5}$$

  La radice considera che i dati riportati nel database sono espressi in densità spettrale di potenza quindi in $\frac{nV}{\sqrt{(Hz)}}$.

- K è la costante di Boltzmann e vale circa $1{,}38^{-23}\,\frac{J}{K}$
- T è la temperatura di lavoro fissata a 300°K (27°C)

L'ultimo punto consiste nel risalire alla serie di resistori presenti nella tabella *Res* del database *Component_Database* a partire dalla precisione delle resistenze e dalla dissipazione di potenza su di esse. In particolare il seguente algoritmo

$$Res\_Tolerance <= ComputedTolerance \tag{4.6}$$

include nella lista delle serie di resistenze possibili quelle che hanno una tolleranza minore uguale rispetto a quella calcolata dalle resistenze $R1$ e $R2$ trovate.

Per quanto riguarda la potenza dissipata dai resistori ci si pone nel caso peggiore di funzionamento del circuito amplificatore non invertente: la tensione in uscita in questo caso vale $MaxSat$ e, con il guadagno del blocco è possibile ricavare la tensione d'ingresso e quindi le potenze dissipate sui resistori $R1$ e $R2$. I componenti nel database verranno selezionati in base al seguente algoritmo

$$Res\_Wattage > max(Pow\_R1, Pow\_R2) \tag{4.7}$$

dove Pow_R1 e Pow_R2 sono le potenze dissipate sulle due resistenze mentre *Res_Wattage* è il valore di potenza dissipata riportata nel campo *Wattage* della tabella *Res* nel database.

[3] Gain Band Product

4.4.2 L'amplificatore invertente

In figura 4.2 è riportato il circuito equivalente dell'amplificatore invertente. Analogamente

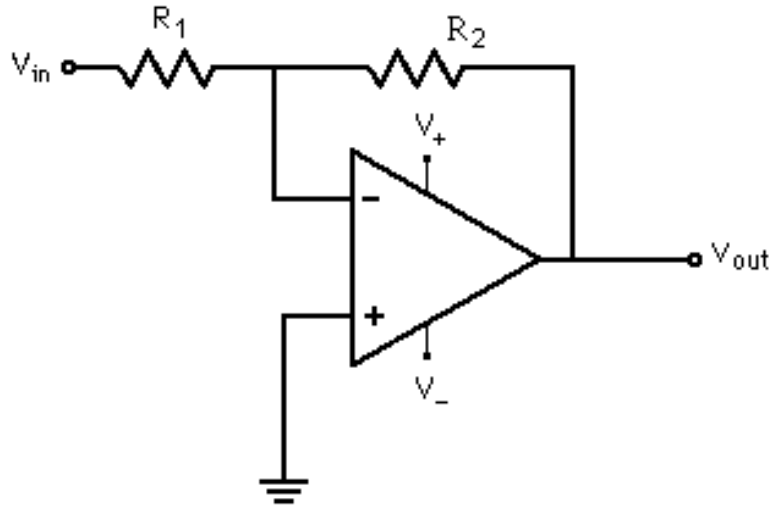

Figura 4.2. Amplificatore Invertente

al caso precedente, per prima cosa vengono determinati i valori delle resistenze $R1$ e $R2$, nonché l'errore di guadagno e la tolleranza calcolate con esse. L'unica differenza rispetto al caso precedente riguarda il valore del guadagno.

A questo punto si può trovare la lista di amplificatori operazionali che permettono di soddisfare le seguenti specifiche analogiche: *Lower Saturation* e *Upper Saturation* sono

Tabella 4.3. Parametri Analogici per l'amplificatore invertente

Parametro Analogico	Nome Variabile Algoritmo
Lower Saturation	MinSat
Upper Saturation	MaxSat
Max Offset	MaxOffset
Max Noise	MaxNoise

utilizzati nuovamente per determinare le alimentazioni dell'amplificatore operazionale, gli algoritmi sono gli stessi:

$$\frac{Dev\_MaxSupply}{2} >= max(MaxSat,abs(MinSat)) \tag{4.8}$$

nel caso in cui l'alimentazione negativa sia diversa da 0 e

$$Dev\_MaxSupply >= MaxSat \tag{4.9}$$

nel caso in cui $MinSat = 0$.

L'algoritmo relativo a *Max Offset* è lo stesso utilizzato per l'amplificatore non invertente, poiché il circuito su cui si effettua l'analisi (spegnendo i generatori di segnale esterno) è il medesimo. Si ha dunque

$$Dev\_Vos \cdot \left(1 + \frac{R2}{R1}\right) + Dev\_Ibias \cdot R2 <= MaxOffset \tag{4.10}$$

Anche l'algoritmo che considera *Max Noise* (la tensione di rumore in uscita) è identico a quello già utilizzato per l'amplificatore non invertente: come risulta dal paragrafo A.4 il modello di rumore è identico per i due amplificatori. La disequazione utilizzata sarà dunque:

$$CF \cdot \sqrt{\left(1 + \frac{R2}{R1}\right)^2 \cdot Dev\_Vn^2 + \left(\frac{4K \cdot T}{R1} + \frac{4K \cdot T}{R2}\right) R2^2} <= MaxNoise \tag{4.11}$$

Per il significato delle variabili si rimanda al paragrafo 4.4.1.

Per ultimo possono essere eseguiti gli algoritmi di selezione delle serie di resistori utilizzabili nell'amplificatore invertente. Come per il caso dell'amplificatore non invertente, si ricorre all'algoritmo di selezione per la precisione e a quello per la tolleranza. Per quanto riguarda il primo caso si ha

$$Res\_Tolerance <= ComputedTolerance \tag{4.12}$$

Dove $ComputedTolerance$ è la precisione delle due resistenze scelte in precedenza mentre $Res\_Tolerance$ è quella riportata nel database. Per quanto riguarda la dissipazione di potenza si ha nuovamente

$$Res\_Wattage > max(Pow\_R1, Pow\_R2) \tag{4.13}$$

Anche in questo caso è le potenze dissipate sulle resistenze $R1$ e $R2$ sono quelle calcolate ponendosi nel caso peggiore.

4.4.3 Il partitore di tensione

Questo circuito, diversamente dai casi precedenti, non utilizza componenti attivi, ma soltanto due resistori in serie, come si può vedere in figura 4.3. Si utilizza sia per modellizzare l'equivalente analogico del blocco *sim_ gain* con guadagno compreso fra 0 e 1, sia per l'equivalente analogico del blocco *sim_ constant*. L'algoritmo utilizzato è lo stesso, ma mentre nel primo caso la partizione viene effettuata sul segnale d'ingresso, per il blocco *sim_ constant* viene eseguita sull'alimentazione (positiva o negativa a seconda del segno del valore costante).

Per trovare le resistenze bisognerà conoscere solo il guadagno (o il valore costante per il blocco *sim_ constant*) e l'errore di guadagno. L'algoritmo corrispondente restituirà, come per i casi degli amplificatori trattati in precedenza, i valori di $R1$, $R2$ nonché la tolleranza e l'errore di guadagno calcolato con esse.

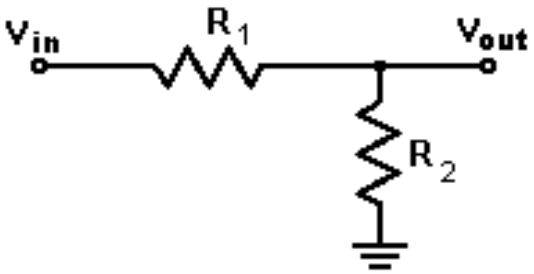

Figura 4.3. Partitore di Tensione

Non essendoci componenti attivi, si può passare immediatamente alla selezione delle serie di resistenze utilizzabili. Come per i casi precedenti si avranno i seguenti algoritmi

$$Res\_Tolerance <= ComputedTolerance \tag{4.14}$$

per quanto riguarda la tolleranza, e

$$Res\_Wattage > max(Pow\_R1,Pow\_R2) \tag{4.15}$$

per quanto riguarda la dissipazione di potenza. Per una descrizione accurata degli algoritmi si rimanda al paragrafo 4.4.1.

4.4.4 L'amplificatore differenziale

Il circuito equivalente dell'amplificatore differenziale è utilizzato per rappresentare l'equivalente analogico del blocco *sim_sum2* nel caso in cui le polarità dei due ingressi siano discordi (ovvero il parametro *Direction* vale "+ -" o "- +"). In figura 4.4 è riportato il circuito che rappresenta il caso "- +". Il segnale che in uscita avrà polarità negativa corrisponde a quello che entra nel morsetto invertente dell'amplificatore operazionale: infatti il circuito risulta essere la combinazione di un amplificatore invertente e di un amplificatore non invertente. Nonostante siano presenti dei componenti passivi (resistenze) non è necessario definire un algoritmo specifico per il calcolo delle loro grandezze; affinché in uscita sia restituita la differenza dei due segnali d'ingresso tutte le resistenze devono essere uguali. Per la determinazione dell'amplificatore operazionale più adatto si utilizzano i seguenti parametri analogici del blocco *sim_sum2* Gli algoritmi per il calcolo dell'alimentazione sono gli stessi già utilizzati per gli altri circuiti con amplificatori, ovvero

- $\frac{Dev\_MaxSupply}{2} >= max(MaxSat,abs(MinSat))$ se l'alimentazione negativa è diversa da 0
- $Dev\_MaxSupply >= MaxSat$ se l'alimentazione minima è nulla

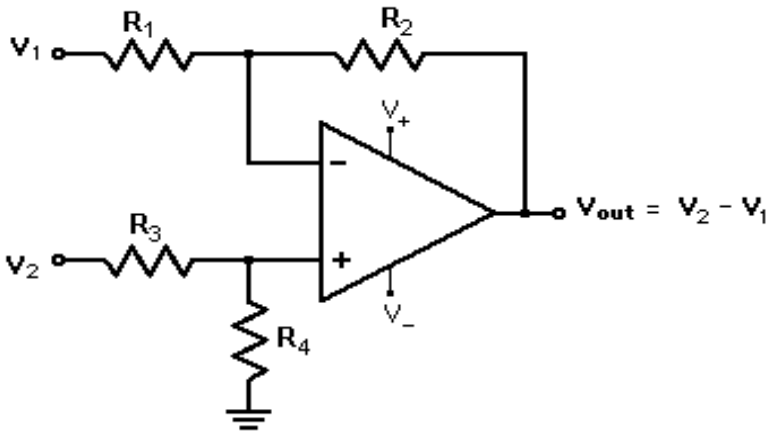

Figura 4.4. Amplificatore Differenziale

Tabella 4.4. Parametri Analogici per l'amplificatore differenziale

Parametro Analogico	Nome Variabile Algoritmo
Lower Saturation	MinSat
Upper Saturation	MaxSat
Max Offset	MaxOffset
Max Noise	MaxNoise

L'algoritmo di ricerca che include *MaxOffset* deve tener conto sia della tensione d'offset d'ingresso sia delle correnti di Bias che entrano in entrambi i morsetti. Poiché la resistenza sul morsetto positivo è uguale al parallelo delle resistenze entranti nel morsetto negativo, il contributo della corrente di Bias è automaticamente compensato. Di conseguenza l'algoritmo di ricerca è il seguente

$$2Dev\_Vos <= MaxOffset \tag{4.16}$$

L'algoritmo che include *MaxNoise* deve tener conto sia del rumore dell'amplificatore operazionale che del rumore dovuto a tutte le resistenze. Come per i casi dll'amplificatore invertente, è necessario un fattore di correzione (CF) per riportare a tensione di rumore le densità spettrali di rumore dell'operazionale e delle resistenze. L'algoritmo che si ottiene è il seguente

$$CF \cdot \sqrt{4(Dev\_Vn^2 + 8R \cdot K \cdot T) + 8R \cdot K \cdot T} <= MaxNoise \tag{4.17}$$

Per ulteriori spiegazioni sull'origine di questi algoritmi si rimanda al paragrafo A.6.5.

Come accadeva per i casi precedenti bisogna determinare quale fra le serie di resistori presenti nel database *Component_Database* è utilizzabile per realizzare il circuito fisico.

Poiché il valore della resistenza è stabilito a priori, così come la tolleranza (la massima possibile), l'unico algoritmo utilizzabile in questo caso è il seguente

$$Res\_Wattage > Pow\_R \tag{4.18}$$

La potenza massima Pow_R è calcolata considerando sempre il caso peggiore in cui la tensione di uscita dell'amplificatore differenziale vale $MaxSat$.

4.4.5 Il sommatore non invertente

Questo circuito è utilizzato per rappresentare l'equivalente analogico del blocco *sim_ sum2* nel caso in cui il parametro *Direction* valga "+ +". Come si può notare dalla figura 4.5,

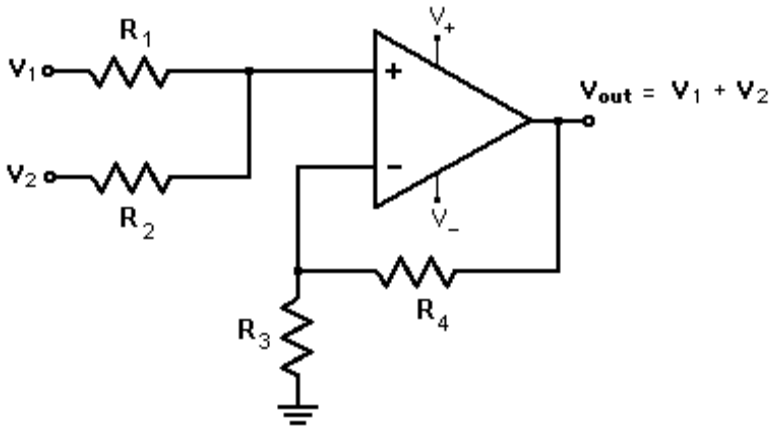

Figura 4.5. Sommatore Non Invertente

il circuito è equivalente ad un amplificatore non invertente con più tensioni entranti nel morsetto positivo dell'operazionale. Come per i circuiti precedenti che rappresentano il blocco *sim_ sum2*, è necessario che le resistenze siano uguali affinché la somma dei segnali d'ingresso non risulti pesata[4]. Di seguito sono riportati i parametri analogici necessari per la definizione degli algoritmi di ricerca. Come per i casi precedenti, gli algoritmi per il calcolo dell'alimentazione sono i seguenti

- $\frac{Dev\_MaxSupply}{2} >= max(MaxSat, abs(MinSat))$ se l'alimentazione negativa è diversa da 0
- $Dev\_MaxSupply >= MaxSat$ se l'alimentazione minima è nulla

Come già accadeva per il sommatore non invertente, il contributo di offset in tensione non considera la corrente di Bias che viene automaticamente compensata. L'algoritmo risultante sarà dunque

$$2Dev\_Vos <= MaxOffset \tag{4.19}$$

[4]cioè i segnali d'ingresso non siano moltiplicati per delle costanti che dipendano dai rapporti fra le resistenze

Tabella 4.5. Parametri Analogici per l'amplificatore differenziale

Parametro Analogico	Nome Variabile Algoritmo
Lower Saturation	MinSat
Upper Saturation	MaxSat
Max Offset	MaxOffset
Max Noise	MaxNoise

L'algoritmo che include *MaxNoise* è lo stesso già utilizzato per l'amplificatore differenziale descritto nel paragrafo 4.4.4. Il fattore di correzione CF riporta le densità spettrali di rumore a tensioni di rumore.

$$CF \cdot \sqrt{4(Dev\_Vn^2 + 8R \cdot K \cdot T) + 8R \cdot K \cdot T} <= MaxNoise \tag{4.20}$$

Come per il caso precedente si rimanda al paragrafo A.6.1 per una spiegazione approfondita dell'origine degli algoritmi.

A questo punto è possibile determinare le resistenze del database che possono essere utilizzate. Come per il caso descritto nel paragrafo 4.4.4 è sufficiente utilizzare l'algoritmo per la dissipazione, ovvero

$$Res\_Wattage > Pow\_R \tag{4.21}$$

4.4.6 Il sommatore invertente

Il circuito è utilizzato per rappresentare la controparte analogica del blocco *sim_ sum2* nel caso in cui il parametro *Direction* valga "- -". Come si può osservare dalla figura 4.6 il

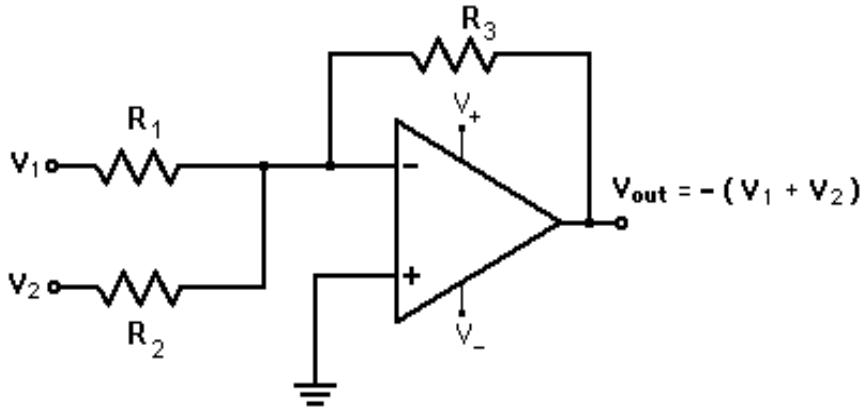

Figura 4.6. Sommatore Invertente

circuito è equivalente al sommatore non invertente ma in questo caso i segnali d'ingresso

entrano nel morsetto invertente. Come prima non è necessario definire un algoritmo per il calcolo delle resistenze che devono essere per forza uguali tra di loro. Analogamente ai casi precedenti, sono necessari i seguenti parametri per definire gli algoritmi di ricerca dell'amplificatore operazionale più adatto: Analogamente ai casi precedenti, gli algoritmi

Tabella 4.6. Parametri Analogici per l'amplificatore differenziale

Parametro Analogico	Nome Variabile Algoritmo
Lower Saturation	MinSat
Upper Saturation	MaxSat
Max Offset	MaxOffset
Max Noise	MaxNoise

per il calcolo dell'alimentazione sono:

- $\frac{Dev\_MaxSupply}{2} >= max(MaxSat,abs(MinSat))$ se l'alimentazione negativa è diversa da 0
- $Dev\_MaxSupply >= MaxSat$ se l'alimentazione minima è nulla

L'algoritmo di ricerca che tiene conto dell'offset è diverso rispetto a quelli utilizzati per l'amplificaore differenziale e sommatore non invertente. In questo caso è presente il contributo di offset della corrente di Bias non essendo presenti resistenze sul morsetto non invertente.

$$Dev\_Vos \cdot \left(1 + \frac{R}{R/2}\right) + Dev\_Ibias \cdot R2 <= MaxOffset \tag{4.22}$$

Anche l'algoritmo relativo al rumore è differente rispetto ai casi precedenti: il contributo di rumore fornito dalle resistenze è presente solo sul morsetto invertente, si ha quindi

$$CF \cdot \sqrt{9Dev\_Vn^2 + 12R \cdot K \cdot T} <= MaxNoise \tag{4.23}$$

Tutti gli algoritmi sono stati ricavati a partire dalle equazioni riportate nel paragrafo A.6.9.

Per determinare le resistenze del database utilizzabili si ricorre allo stesso algoritmo della dissipazione di potenza descritto nei paragrafi 4.4.4 e 4.4.5, ovvero

$$Res\_Wattage > Pow\_R \tag{4.24}$$

L'unica differenza significativa riguarda la formula utilizzata per ricavare Pow_R differente rispetto a quella del sommatore non invertente e dell'amplificatore differenziale.

4.4.7 L'amplificatore di Sample & Hold

Questo circuito deve essere trattato in modo differente rispetto ai precedenti per vari motivi. Innanzitutto l'amplificatore di S/H è considerato un circuito integrato (quindi un componente attivo) con la capacità di mantenimento integrata, non sono quindi utilizzati

algoritmi di ricerca dei componenti passivi. In secondo luogo, i dipositivi inclusi nel database non sono realizzati con la stessa architettura interna[5].Prima di proseguire conviene dunque fornire una breve descrizione delle rappresentazioni circuitali più diffuse per i S/H.

Open-Loop : è realizzata con due buffer che servono a disaccoppiare il segnale d'ingresso e il segnale d'uscita, uno switch che permette di campionare l'ingresso e un condensatore di mantenimento che serve a mantenera costante la tensione per il periodo di campionamento. Questa configurazione è molto rapida poiché non prevede l'utilizzo di un feedback ma risulta essere poco preciso per applicazioni che richiedono periodi di campionamento brevi.

Closed-Loop : è un'architettura che prevede l'utilizzo di un feedback in cui può essere anche posta la capacità di mantenimento (in questo caso la configurazione viene detta a integratore). Rispetto all'atchitettura *Open-Loop* la precisione è garantita dalla reazione ma la velocità è inferiore soprattutto nel caso in cui si utilizzi la capacità nella reazione.

Gli algoritmi di ricerca realizzati non tengono conto dell'architettura interna dell'amplificatore, ma piuttosto della sua precisione nel campionare il segnale correttamente: i parametri utilizzati negli algoritmi di ricerca sono riportati di seguito Rispetto a tutti i casi descrit-

Tabella 4.7. Parametri Analogici per l'amplificatore di Sample & Hold

Parametro Analogico	Nome Variabile Algoritmo
Upper Saturation	MaxSat
Max Offset	MaxOffset
Max Gain Error	MaxGainError

ti in precedenza il rumore non è stato utilizzato come termine di confronto: tutti i data sheet dei dispositivi non riportavano tale grandezza e, non essendo sempre possibile risalire all'architettura interna, si è preferito non effettuare stime potenzialmente scorrette.

MaxSat è utilizzato per risalire all'alimentazione del dispositivo. Come per i casi precedenti si ha

$$Dev\_MaxSupply >= MaxSat \tag{4.25}$$

Tutti i dispositivi trovati e riportati nel datasheet hanno alimentazione duale.

MaxGainError è utilizzato per ricavare la risoluzione minima che deve garantire il Sample & Hold affinché il segnale sia campionato dall'ADC con la giusta precisione. Si ricorda che il guadagno A per tutti i S/H è considerato unitario. La formula di partenza è

$$\Delta A = \frac{V_{OUT} - V_{IN}}{V_{IN}} < \frac{1}{2^{n+1}}$$

[5] alcuni data sheet di amplificatori di Sample & Hold non riportano neanche l'architettura utilizzata per realizzare il componente

Dove n indica la risoluzione del convertitore ed è il valore su cui viene operato il confronto. L'algoritmo di ricerca risultante è il seguente:

$$Dev\_Resolution >= \left\lceil log_2 \left(\frac{1}{MaxGainError} \right) - 1 \right\rceil \tag{4.26}$$

MaxOffset è utilizzato per ricavare l'offset che affligge il S/H. In realtà gli offset in tensione si manifestano in fasi diverse del funzionamento del dispositivo e assumono di conseguenza nomi diversi:

- l'*Offset Voltage* indica la tensione che sia ha in uscita durante la fase di Sample, quando in ingresso viene dato segnale nullo. Si ricava in base alla seguente formula

  $$OffsetSample = \frac{Dev\_FullScale}{2^{Dev\_Resolution} + 1}$$

  Dove $Dev\_FullScale$ indica la massima tensione che può raggiungere il segnale di uscita e $Dev\_Resolution$ è la risoluzione massima che può raggiungere il S/H.

- l'*Hold Step* si verifica durante la transizione dalla fase di Sample alla fase di Hold ed è causato dal trasferimento di carica verso la capacità durante la chiusura dello switch. Nei data sheet viene già riportato il suo valore quindi

  $$OffsetSampleToHold = Dev\_HoldStep$$

L'algoritmo di ricerca risultante ipotizza che i due offset siano combinati essendo le fasi ravvicinate, di conseguenza l'algoritmo risultante sarà

$$OffsetSample + OffsetSampleToHold <= MaxOffset \tag{4.27}$$

L'ultimo algoritmo di ricerca utilizzato tiene conto della massima frequenza del segnale entrante nel blocco *sim_zoh* che dipende dalla risoluzione del componente e dal jitter di apertura[6] secondo la seguente formula:

$$SignalFrequency <= \frac{1}{2\pi \cdot 2^{Dev\_Resolution} \cdot Dev\_ApertureJitter}$$

[6]il jitter di apertura corrisponde all'incertezza che si ha sul tempo di apertura ovvero il tempo che intercorre fra quando è stato dato il segnale di Sample e quando effettivamente l'interruttore si apre

Capitolo 5

La descrizione del compilatore in codice MATLAB^{TM}

In questo capitolo verrà analizzato il codice in linguaggio MATLAB^{TM} implementato per permettere la ricerca dei componenti nel database e la traduzione del diagramma a blocchi in un file circuito (*.cir* da simulare in Spice. Le funzioni presenti possono essere divise nelle seguenti categorie:

- funzioni per la creazione delle finestre di dialogo necessarie per visualizzare i componenti trovati che traducono il blocco in un circuito analogico equivalente;
- funzioni per la ricerca dei componenti del database
- funzioni per la definizione della netlist e la compilazione di quest'ultima in un file circuito;
- funzioni utilizzate per scopi generici come la manipolazione di stringhe o l'estrazione di dati da un database;

5.1 Premessa: le notazioni considerate

Prima di descrivere il codice vero e proprio è conveniente fornire un'introduzione generale alla logica di funzionamento seguita nella stesura dei programmi e, soprattutto ad alcune notazioni che sono state adottate per semplificare il codice finale. L'idea di fondo è quella di realizzare dei programmi che risultino flessibili per tutte le descrizioni analogiche dei blocchi CodeSimulink.

La più importante notazione adottata riguarda le funzioni "_ParamUI" ovvero quelle associate alla callback *OpenFcn* di ogni blocco esteso. Queste funzioni sono generalmente costituite da un costrutto switch in cui ogni *case* contiene delle informazioni specifiche del blocco esteso o chiama l'esecuzione di determinate istruzioni. All'interno di alcuni *case* sono stati aggiunti gli elementi necessari a fornire la descrizione analogica del blocco, tra cui la lista dei componenti e dei parametri da inserire nelle interfacce e gli algoritmi di

calcolo. In questo modo risulta molto semplice accedere a tali informazioni: è sufficiente ricavare il tipo di blocco esteso *ExtendedBlockType* e utilizzare le seguenti istruzioni

```
eval([ExtendedBlockType '_ParamUI(''OPZIONE'', BlockHandle);']);
```

dove OPZIONE non è altro che la scelta del costrutto switch della "_ParamUI" che contiene i dati che si sta cercando di recuperare. Tra le varie opzioni aggiunte si trovano:

AnHwDescription : la presenza stessa di questa opzione fra quelle disponibili, implica che è disponibile una descrizione analogica per quel determinato blocco. Questa opzione è necessaria alla funzione di CodeSimulink *PutDefaultButtons* per aggiungere il pulsante "Component" nell'interfaccia dei parametri analogici relativa a quel blocco esteso.

AnalogSource : è necessaria per stabilire se un determinato blocco si comporta da sorgente di segnale analogico, ovvero se può essere considerato un generatore di tensione nell'analisi Spice. Mentre *sim_ constant* e *sim_ repeating_ sequence* sono considerate delle sorgenti di segnale analogico a priori, al blocco d'interfaccia *sim_ analogIn* può venire attribuito o meno il ruolo di sorgente, se è precedeuto o meno da un blocco *source* di Simulink^{TM} e da un *To Workspace*. Per ulteriori informazioni si rimanda ai paragrafi 5.4.6 e 5.4.10.

NeededVariables : restituisce una stringa contenente i nomi (separati da "|") dei parametri analogici di cui devono essere recuperati i valori numerici. Essi costituiscono le specifiche del segnale di uscita e saranno utilizzati dagli algoritmi di ricerca dei componenti. Per la lista completa dei parametri si rimanda al paragrafo 4.2 e per la lista di quelli specifici per ogni blocco si rimanda al paragrafo 4.4.

ActualCircuit : è inclusa solo nei blocchi estesi che possono avere una o più descrizioni analogiche (a seconda del valore di alcuni parametri)e restituisce una stringa con il nome attribuito a quella descrizione.

DeviceOfCircuit : restituisce una stringa contenente i nomi abbreviati (separati da "|") dei componenti attivi che sono utilizzati nella descrizione circuitale del blocco.

PassCompOfCircuit : analogamente al caso '*DeviceOfCircuit*' restituisce la stringa che contiene i nomi abbreviati dei componenti passivi utilizzati nella controparte analogica del blocco CodeSimulink.

PassCompParam : restituisce due stringhe:

- la prima contiene i nomi estesi dei componenti passivi separati dal simbolo "|"; essi saranno utilizzati nella finestra di dialogo per etichettare i text box contenenti i (corrispondenti) valori numerici calcolati.
- la seconda contiene i nomi abbreviati dei componenti passivi separati dal simbolo "|"; saranno utilizzati per definire i nomi dei campi in cui saranno memorizzati i valori numerici corrispondenti.

PassCompAlgorithm : contiene le stringhe che valutate con il comando *eval* permettono di ottenere i valori numerici dei componenti passivi utilizzando alcuni dei parametri analogici dei blocchi.

Res_Algorithm : contiene le stringhe che, passate al comando *eval*, permettono di risalire alle serie di resistenze nel database *Component_Database* utilizzabili nel circuito equivalente al blocco CodeSimulink. Nel caso in cui fossero presenti ulteriori componenti passivi sarà necessario aggiungere un'opzione con i corrispondenti algoritmi di ricerca implementati.

OpAmp_Algorithm/SHA_Algorithm : come per il caso di '*Res_Algorithm*' contiene le stringhe che valutate con il comando *eval* permettono di ottenere la lista dei componenti del database *Component_Database* che rientrano nelle specifiche e possono essere utilizzati per fornire una descrizione analogica del blocco CodeSimulink.

Si analizza, per esempio il caso del blocco *sim_gain*:

```
case 'AnHwDescription'
    varargout{1} = 1;
```

Questo caso è comune a tutti i blocchi estesi trattati e verrà sempre restituito il valore numerico 1.

```
case 'NeededVariables'
    varargout{1} = ['sim_GainError_Y1|sim_Offset_Y1|sim_NoiseFigure_Y1|' ...
        'sim_LowerSaturation_Y1|sim_UpperSaturation_Y1'];
    varargout{2} = ['MaxGainError|MaxOffset|MaxNoise|' ...
        'MinSat|MaxSat'];
```

Negli algoritmi di ricerca dei componenti analogici di *sim_gain* è necessario conoscere i seguenti parametri contenuti tutti nella maschera del blocco:

- *l'errore di guadagno* contenuto nel parametro *sim_GainError_Y1* della maschera e memorizzato nella variabile *MaxGainError*.
- *l'offset del segnale di uscita* contenuto nel parametro *sim_Offset_Y1* della maschera e memorizzato nella variabile *MaxOffset*.
- *il rumore che affligge il segnale di uscita* contenuto nel parametro *sim_NoiseFigure_Y1* della maschera e memorizzato nella variabile *MaxNoise*.
- *la dinamica minima del segnale* contenuto nel parametro *sim_LowerSaturation_Y1* della maschera e memorizzato nella variabile *MinSat*.
- *la dinamica massima del segnale* contenuto nel parametro *sim_UpperSaturation_Y1* della maschera e memorizzato nella variabile *MaxSat*.

```
case 'ActualCircuit'
    if gain > 1
        varargout{1} = 'sim_gain_P'; %Positive
    else
        if gain < 0
            varargout{1} = 'sim_gain_N'; %Negative
        else
            if gain >0 & gain <1
                varargout{1} = 'sim_gain_ZO'; %Zero to One
            else
                errordlg (['Unsupported gain 0 or 1 in block ' ...
                    getfullname(BlockID)],'Parameter Error');
            end
        end
    end
```

A seconda del valore attuale di *gain* si avranno differenti descrizioni circuitali che verranno identificate con nomi differenti. Nello specifico si avrà:

sim_ gain_ P quando il blocco *sim_ gain* è considerato un amplificatore non invertente.

sim_ gain_ N quando il blocco *sim_ gain* è considerato un amplificatore invertente.

sim_ gain_ ZO quando il blocco *sim_ gain* è considerato un partitore di tensione.

```
case 'DeviceOfCircuit'
    if gain >1 | gain<0
        varargout{1} = 'OpAmp';
    else
        varargout{1} ='';
    end
```

Anche in questo caso è necessario osservare il valore del guadagno poiché, fra i circuiti disponibili, soltanto l'amplificatore invertente e l'amplificatore non invertente utilizzano dei dispositivi attivi, ne caso specifico gli operazionali indicati con la stringa '*OpAmp*'. Grazie ad essa sarà possibile

- recuperare la tabella *OpAmp* del database *Component_ Database* contenente tutti gli amplificatori operazionali selezionabili.
- denominare i parametri della maschera di *sim_ gain* che conterranno i valori delle specifiche del circuito integrato come spiegato più avanti nel paragrafo.

```
case 'PassCompOfCircuit'
    varargout{1} = 'Res';
```

Tutti i circuiti equivalenti di *sim_ gain* utilizzano delle resistenze, sarà dunque necessario utilizzare la stringa '*Res*' come nel caso precedente per recuperare la corrispondente tabella del database e per denominare le relativi specifiche (tolleranza, dissipazione e prezzo) da memorizzare nella maschera.

```
case 'PassCompParam'
    varargout{1} = 'R1(Ohm):|R2(Ohm):|Computed Gain Error:';
    varargout{2} = 'R1|R2|ComputedGainError';
```

Tutti i circuiti equivalenti del blocco *sim_ gain* utilizzano le resistenze R_1, R_2 di cui dovranno essere calcolati i valori numerici e l'errore di guadagno *ComputedGainError* da esse derivato. In particolare:

- *R1* permetterà di definire il parametro *sim_ R1* della maschera contente il valore numerico della prima resistenza. Nella finestra di dialogo il suo valore sarà visualizzato nel text box etichettato come *R1(Ohm):*.
- *R2* permetterà di definire il parametro *sim_ R2* della maschera contente il valore numerico della seconda resistenza. Nella finestra di dialogo il suo valore sarà visualizzato nel text box etichettato come *R2(Ohm):*.
- *ComputedGainError* permetterà di definire il parametro *sim_ ComputedGainError* della maschera contente il valore numerico della prima resistenza. Nella finestra di dialogo il suo valore sarà visualizzato nel text box etichettato come *Computed Gain Error:*.

```
case 'PassCompAlgorithm'
    if gain > 1
        varargout{1} = [' ... '];
    else
        if gain <0
            varargout{1} = [' ... '];
        else
            if gain >0 & gain<1
                varargout{1} = [' ... '];
            else
                varargout{1} = '';
            end
        end
    end
```

Analogamente ad altri casi trattati in precedenza gli algoritmi per il calcolo dei componenti passivi (per il caso di *sim_ gain* si intendono le resistenze $R1$ e $R2$ ed l'errore di guadagno calcolato *ComputedGainError*) dipendono dal valore attuale del guadagno *gain*. Per non complicare la lettura del codice gli algoritmi, descritti nei paragrafi 4.4.1, 4.4.2 e 4.4.3, non sono stati riportati.

```
case 'Res_Algorithm'
    if gain > 1
        varargout{1} = [' ... '];
    else
        if gain < 0
            varargout{1} = [' ... '];
        else
            if gain >0 & gain<1
                varargout{1} = [' ... '];
            else
                varargout{1} = '';
            end
        end
    end
```

Anche in questo caso gli algoritmi per scegliere le serie di resistenza più adatte dipendono dal tipo di circuito analizzato e quindi dal valore del guadagno. Per semplificare la comprensione del codice non sono stati riportati gli algoritmi di scelta descritti comunque nel paragrafo 4.4.1 per le resistenze dell'amplificatore non invertente.

```
case 'OpAmp_Algorithm'
    if gain > 1
        varargout{1} = [' ... '];
    else
        if gain < 0
            varargout{1} = [' ... '];
        else
            if gain >0 & gain <1
                varargout{1} = '';
            else
                errordlg (['Unsupported gain 0 or 1 in block ' ...
                    getfullname(BlockID)], 'Parameter Error');
            end %if gain >0 & gain <1
        end %if gain < 0
    end %if gain > 1
```

Anche in questo caso, a seconda del valore di *gain*, saranno disponibile una serie di differenti algoritmi di selezione degli amplificatori operazionali del database. Analogamente agli altri casi trattati si è deciso di non riportare il codice degli algoritmi, trattati nei paragrafi 4.4.1, 4.4.2 e 4.4.3.

Un'altra notazione adottata riguarda la costruzione delle finestre di dialogo per la descrizione analogica: come spiegato nel paragrafo 4.1 i circuiti analizzati possono essere composti da combinazioni di componenti elettronici passivi e attivi: i rispettivi parametri

o valori devono essere rappresentati in *frame*[1] distinti per facilitare all'utente la lettura dell'interfaccia. Sia il numero dei frame che dei parametri in essi compresi sono variabili, in particolare l'elenco dei parametri relativi ai componenti passivi è presente nelle funzioni "_ParamUI" dei blocchi trattati come *case* dello statement switch sotto il nome di *PassCompParam*. Se un determinato circuito non prevede l'uso di tali componenti il relativo frame non deve essere aggiunto. Per quanto riguarda il frame dei componenti attivi il discorso è simile: se fra i *case* delle funzioni "_ParamUI" non è presente *DeviceOfCircuit*[2] allora il blocco non avrà componenti attivi (e non dovrà quindi includere tale frame). La lista dei parametri che descrivono i dispositivi sono riportati invece a parte nella funzione *AnHwDevParameters* descritta nel paragrafo 5.2.5. La lista dei parametri è memorizzata sotto forma di stringa in cui ogni parametro è separato dall'altro dal simbolo "|" (sarà necessario effettuare un parsing per separarli).

Un'altra notazione adottata riguarda il nome dei componenti attivi e passivi che, come affermato nel paragrafo 3.2 mantengono i nomi assegnati alle rispettive tabelle del database. Questo semplifica anche la memorizzazione dei valori dei parametri nel blocco, salvati con un nome assegnato secondo il seguente criterio: Il nome del parametro del blocco (aggiunto

Identificativo	Componente	Nome parametro
sim	*OpAmp*	*Price*
sim	*Res*	*Tolerance*

con la funzione *add_blockparam*) sarà dunque *sim_OpAmp_Price* (naturalmente l'esempio è riferito al prezzo del componente ma è applicabile per tutti gli altri parametri ad esso relativi).

Analogamente si utilizza il nome del componente (attivo o passivo) per identificare gli algoritmi necessari a ricavare i suoi parametri specifici: nelle varie "_ParamUI" è possibile trovare un *case* che è stato nominato secondo il seguente criterio: La scelta del comando

Componente	Identificativo
OpAmp	*Algorithm*
Res	*Algorithm*

switch sarà dunque *OpAmp_Algorithm* o *Res_Algorithm*.

5.2 La creazione delle interfacce utente

Le interfacce utente (UI) sono quelle finestre di dialogo che si aprono in seguito alla pressione di un pulsante o cliccando direttamente su di un blocco. È conveniente fornire una breve descrizione dei principali tipi di interfacce presenti:

Interfaccia Principale : è la finestra di dialogo che si genera cliccando due volte su un blocco esteso. Le funzioni Callback che vengono chiamate in questo caso sono le

[1] sono le cornici che raggruppano gli oggetti grafici nell'interfaccia che dovrebbero avere un significato comune

[2] contiene una stringa con la lista dei componenti attivi da includere nella descrizione

"_ParamUI". Tutti i blocchi CodeSimulink presentano *almeno* quest'interfaccia che è composta da:

- una serie di parametri specifici del blocco (come ad esempio il guadagno per *sim_ gain*.
- dei pulsanti che permettono di specificare le proprietà del segnale di uscita[3].
- dei pulsanti standard[4] di gestione della finestra;

In figura 5.1 è riportata l'interfaccia principale per il blocco *sim_ gain*. Questa finestra

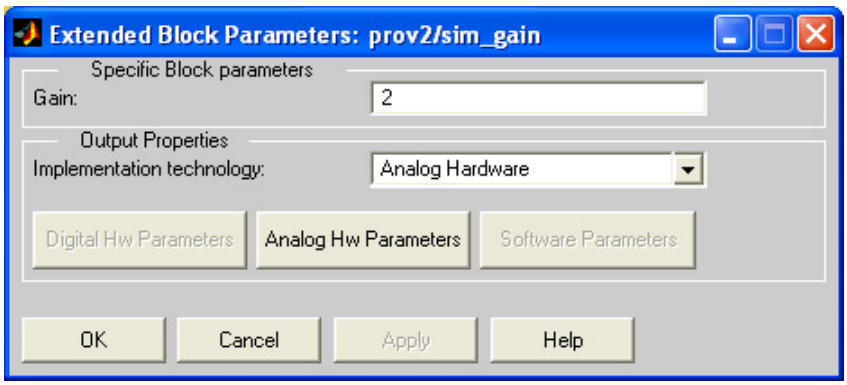

Figura 5.1. Interfaccia Principale per il blocco *sim_ gain*

non verrà trattata in quanto non è stato necessario apportare modifiche essendo la meno specifica di quelle presentate.

Interfaccia Parametri : questa finestra si apre cliccando su uno dei tre pulsanti che indicano le proprietà del segnale di uscita: a seconda del tipo di descrizione che si desidera fornire si avranno interfacce differenti. Nel caso in analisi ci si riferirà esclusivamente alla finestra per i parametri analogici, ottenuta cliccando sul pulsante *Analog Hw Parameters*. In questo caso sarà costituita da

- un menù a tendina in cui scegliere il tipo di segnale in uscita (verrà sempre considerato un segnale d'uscita in tensione *single-ended*);
- una serie di parametri analogici che servono a descrivere il segnale in uscita del blocco. Sono il punto di partenza su cui verranno applicati gli algoritmi di ricerca.
- i pulsanti standard di gestione dell'interfaccia già citati in precedenza.

L'unica modifica apportata all'interfaccia consiste nell'aggiunta del pulsante "Component" a quelli standard già presenti. In figura 5.2 è visualizzata l'interfaccia parametri standard, valida per la maggior parte dei blocchi estesi. Cliccando sul pulsante

[3]Digital Hw Parameters, Analog Hw Parameters e Software Parameters
[4]OK, Cancel, Help e Apply

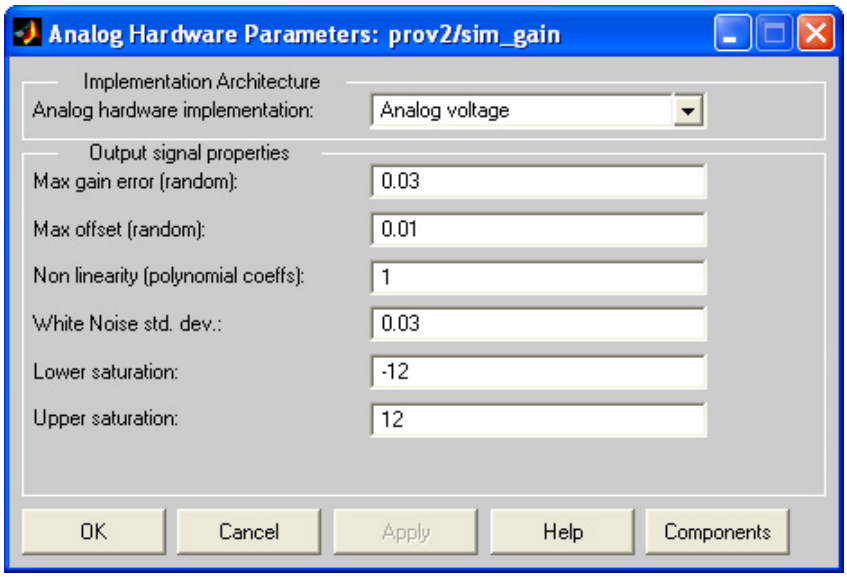

Figura 5.2. Interfaccia Parametri standard

"Component" verrà generata l'interfaccia descritta in seguito.

Interfaccia Componenti : è generata cliccando sul pulsante "Component" aggiunto all'interfaccia parametri (descritta sopra). È costituita da:

- un frame contenente i componenti passivi (in genere resistenze) che compongono il circuito analogico equivalente del blocco esteso. Per ogni tipologia di componente passivo verranno inoltre generati un popup menù che conterrà la lista di quelli utilizzabili[5] e un pulsante etichettato come "Cheap" che permette di includere nella lista solo quelli di costo minimo.
  Il frame dei componenti passivi non sarà presente se il circuito equivalente del blocco esteso è costituito solo da dispositivi integrati;
- un frame contenente i componenti attivi della rappresentazione circuitale associata a quel blocco. Analogamente al frame descritto in precedenza, per ogni tipo di componente attivo trovato verranno aggiunti un menù a tendina per includere i dispositivi scelti con i relativi algoritmi di scelta ed il pulsante "Cheap" per l'inclusione dei componenti più economici.
  L'assenza di componenti integrati corrisponderà all'assenza di questo frame
- una serie di pulsanti standard per la gestione dell'interfaccia[6].

Uno degli obiettivi principali di questa tesi riguarda la creazione di queste interfacce e la gestione delle funzioni di ricerca dei componenti ad esse associate. In figura 5.3

[5] la lista dei componenti passivi utilizzabili è ricavata grazie agli algoritmi di ricerca associati a quel determinato circuito

[6] OK, Cancel, Find Comp e Help

è riportata la finestra dei componenti analogici del blocco *sim_gain* quando il suo circuito equivalente è un amplificatore non invertente. Ogni blocco trattato avrà una

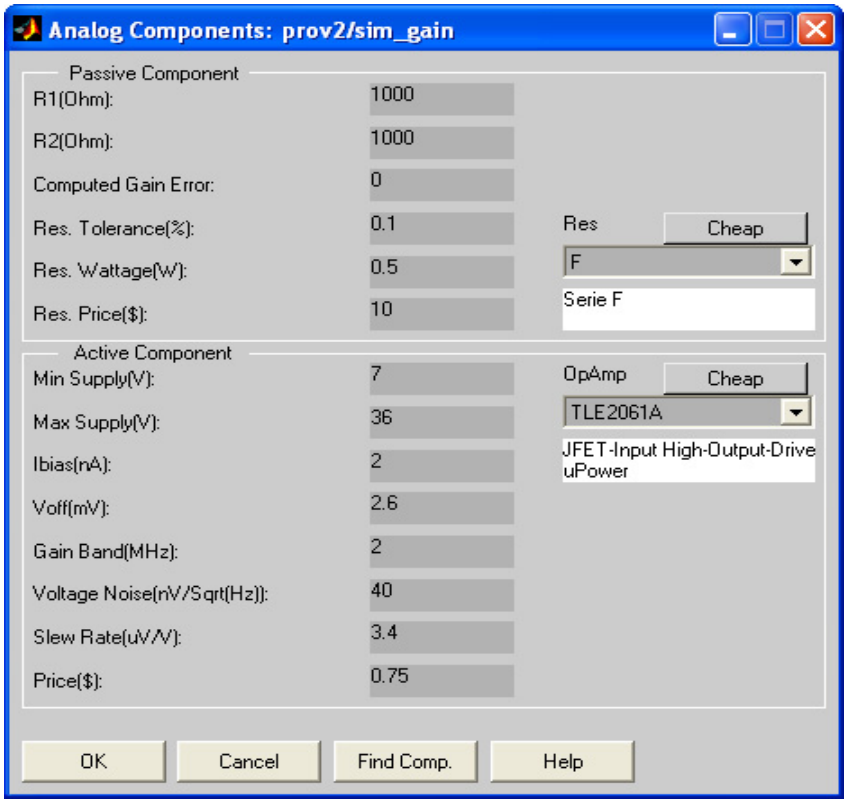

Figura 5.3. Interfaccia Componenti per il blocco *sim_gain*

o più interfacce componenti a seconda del numero di circuiti equivalenti che possono rappresentare

Di seguito viene riportato il codice necessario per generare le interfacce dei componenti. Esso è incluso nelle seguenti funzioni:

- *sim_standard2in_addHW_ParamUI* per il blocco *sim_gain*.
- *sim_standard2in_ParamUI* per i blocchi *sim_sum2* e *sim_zoh*.
- *sim_standard2in_varOut_ParamUI* per il blocco *sim_constant*.

Poiché esse gestiscono la creazione delle altre interfacce presentate si è ritenuto opportuno includere il seguente codice di generazione delle interfacce componenti al loro interno.

```
ControlHeight = 15;
ButtonHeight = 25;
GeneralControlsHeight = 40;
```

```
NumberOfFrames = 2;

FigHandle = gcbf;
BlockHandle = getfield(get(FigHandle,'UserData'), 'BlockHandle');
ExtendedBlockType = get_param(BlockHandle, ...
  'sim_ExtendedBlockType');
NewCircuit = eval([ExtendedBlockType
  '_ParamUI(''ActualCircuit'', BlockHandle);'],'ExtendedBlockType');

try
  ActualCircuit = get_param(BlockHandle, 'sim_ActualCircuit');
catch
  add_blockParam(BlockHandle, 'sim_ActualCircuit', '&', NewCircuit);
end

NControlsInsideFrame1 = AnalogCircuitDescription('ControlsInside',...
  FigHandle);

NControlsInsideFrame2 = AnalogDeviceDescription('ControlsInside',...
  FigHandle);

FigHeight = 5 + GeneralControlsHeight + 10*NumberOfFrames + ...
  (ControlHeight+5)*(NControlsInsideFrame1 + NControlsInsideFrame2);

[FigHandle AlreadyPresent] = PutDlg(BlockHandle, FigHeight, ...
  'AnHwCmp'); if AlreadyPresent; return; end

LastCtrlBottom = FigHeight -5;

FrameHeight1 = 5 + NControlsInsideFrame1*(ControlHeight + 5);
FrameHeight2 = 5 + NControlsInsideFrame2*(ControlHeight + 5);

PutAnCmpControls(FigHandle,LastCtrlBottom, FrameHeight1,...
FrameHeight2);

PutDefaultAnCmpButtons(FigHandle, [ExtendedBlockType ...
'_ParamUI(''AnCmpDlg_CallBack'');']);
```

Passando attraverso le "_ParamUI", viene chiamato in seguito alla pressione del pulsante "Component" precedentemente descritto. A questo punto è possibile descrivere le funzioni che permettono di generare l'interfaccia componenti. La prima parte del codice si occupa di definire la grandezza degli oggetti grafici da deporre e valuta il tipo di descrizione che sarà attuata in base al valore di *NewCircuit* (eventualmente è presente anche *NewDevice*): esso identifica la descrizione circuitale del blocco CodeSimulink in base ai parametri indicati

in precedenza (come il guadagno per *sim_ gain* e la polarità degli ingressi per *sim_ sum2*). Questa stringa verrà memorizzata come parametro del blocco in "*sim_ ActualCircuit*" e verrà utilizzata nel caso in cui la descrizione analogica dello stesso blocco cambi più volte. A questo punto vengono chiamate le funzioni *AnalogCircuitDescription* e *AnalogDeviceDescription* a cui viene passato la stringa '*ControlsInside*'. Entrambe le funzioni restituiranno il numero di componenti che dovranno essere inclusi rispettivamente nel frame dei componenti passivi e nel frame dei componenti attivi. Queste funzioni in realtà hanno diverse opzioni di funzionamento, si rimanda quindi ai paragrafi successivi per la descrizione completa del loro funzionamento.

In base al numero degli oggetti di controllo presenti sull'interfaccia è possibile ricavare le dimensioni della finestra di dialogo e aprirla con la funzione *PutDlg* (non descritta in quanto già presente nel tool di CodeSimulink). A questo punto verranno chiamate due funzioni realizzate ex novo ovvero la *PutAnCmpControls* per deporre tutti gli oggetti di controllo associati ai componenti attivi e passivi che descrivono il circuito, e la funzione *PutDefaultAnCmpButtons* che aggiungerà invece i pulsanti standard per l'interfaccia componenti.

5.2.1 AnalogCircuitDescription

```
[varargout] = AnalogCircuitDescription(Option, ...
    FigHandle, varargin)
```

Questa funzione è utilizzata per effettuare diverse operazioni sull'interfaccia componenti a seconda del valore che assume la stringa *Option*. Per prima cosa vengono inizializzate le variabili e si ricava la stringa $ExtendedBlockType$ che identifica il blocco esteso. Essa viene utilizzata per determinare la tipologia di componenti passivi del circuito equivalente[7]. Per ognuno di essi vengono estratte le stringhe dei parametri specifici (come la tolleranza e la dissipazione delle resistenze) e il loro numero.

Successivamente, sempre utilizzando $ExtendedBlockType$, vengono estratte le due stringhe *PassCompInfoStr* e *PassCompVarStr* che conterranno rispettivamente l'elenco dei nomi estesi dei componenti passivi e l'elenco dei nomi ridotti che verranno utilizzati per memorizzare i valori calcolati. Il numero dei componenti del circuito equivalente vengono sommato al numero dei parametri specifici calcolato in precedenza. A questo punto, a seconda dell'opzione si avrà:

- se *Option = 'ControlsInside'* restituisce il numero dei componenti passivi usati nel circuito equivalente sommato al numero dei parametri specifici di ogni componente passivo. Questa somma era stata ricavata in precedenza.

- se *Option = 'Generate'* viene deposto il frame dei componenti passivi e tutti gli oggetti di controllo in esso contenuti con la funzione *PutTextBox2*: per ogni componente verrà deposto un text box con il nome esteso del componente (ad esempio "*R2(Ohm):*") e un text box in cui verrà in seguito visualizzato il suo valore numerico.

[7] Tutti i circuiti analizzati utilizzano solo le resistenze come componente passivo

Questo valore deve essere ancora calcolato ma intanto al blocco viene aggiunto un nuovo parametro denominato nel seguente modo: A questo punto, per ogni tipo di

Identificativo	Componente
sim	*R2*

componente passivo presente (resistenza, capacità) verranno aggiunti degli ulteriori text box che descrivono i parametri specifici del componente passivo come le tolleranze o la dissipazione e un popup menu, contenente le possibili serie di componenti utilizzabili. Per maggiori informazioni si rimanda al paragrafo 5.2.3.

- se *Option = 'Update'* vorrà dire che l'interfaccia componenti è già presente ed è necessario aggiornarla essendo stati calcolati dei nuovi valori di componenti o parametri. Si utilizza sempre *PassCompVarStr* e si memorizzeranno i nuovi valori nel blocco e nella proprietà "Userdata" della finestra.

- se *Option = 'Delete'* la funzione rimuoverà tutti gli oggetti di controllo finora deposti ed il frame che li conteneva.

- se *Option = 'PassComp_Structure'* verrà creata una struttura dati chiamata *PassCompStruct* contenente un campo nullo per ogni componente (o parametro ad esso associato) presente in *PassCompVarStr*. Questa struttura verrà restituita alla funzione chiamante (*AllComponentSearch*) che si occuperà di memorizzare in essa i valori numerici calcolati.

5.2.2 AnalogDeviceDescription

```
[varargout] = AnalogDeviceDescription(Option, FigHandle, varargin)
```

Questa funzione è del tutto analoga a *AnalogCircuitDescription* (descritta in precedenza) ma si occupa di gestire il frame dei componenti attivi, che contiene i parametri del dispositivo integrato scelto per rappresentare il circuito equivalente del blocco. Esattamente come nel caso precedente per prima cosa vengono inizializzate le variabili in gioco e si ricava la stringa *ExtendedBlockType*. Quindi:

- se *Option = 'ControlsInside'* si ricava da *ExtendedBlockType* la lista dei componenti attivi e, per ognuno di essi si ricava a sua volta la lista dei parametri che lo descrivono utilizzando la funzione *AnHwDevParameters* che contiene le stringhe con i nomi estesi e quelli ridotti. Analogamente al caso di componenti passivi il numero è ricavato in base a quanti simboli "|" sono presenti nella stringa.

- se *Option = 'Generate'* la funzione si occupa di deporre il frame del componente attivo al cui interno vengono posti tutti i text box dei suoi parametri, un listbox che contiene l'elenco di quelli "utilizzabili" e un altro text box che contiene la descrizione dell'ultimo integrato selezionato. Come per il caso precedente vengono aggiunti

al blocco i parametri specifici del componente seguendo il seguente criterio di assegnazione dei nomi descritto all'inizio del paragrafo che considera anche il tipo di componente a cui è associato il parametro.

Identificativo	Componente	Nome parametro
sim	*OpAmp*	*Price*

- se *Option = 'Delete'* verranno cancellati tutti gli oggetti di controllo (text box, popup menù e pulsanti[8]) presenti nel frame oltre naturalmente al frame stesso.
- se *Option = 'Update'* come per il caso precedente verranno aggiornati i valori numerici dei parametri dell'interfaccia e i relativi parametri memorizzati nel blocco.

Attualmente nessuna descrizione analogica contiene più di un dispositivo attivo. La funzione è comunque progettata per gestire descrizioni circuitali con più dispositivi inserendo nello stesso frame tutti i parametri di ogni componente.

5.2.3 PutPopupPush

```
[PopupMenuHandle, PushButtonHandle] = PutPopupPush(FigHandle, ...
    Position, CompType, ...
    SelectedCompParamTag, SelectedCompParamName, ...
    CompDescriptionParamTag, CompDescriptionParamName, ...
    CompListParamTag, CompListParamName, ...
    CompIndicesParamTag, CompIndicesParamName)
```

Questa funzione è utilizzata sia da *AnalogDeviceDescription* che da *AnalogCircuitDescription* per deporre quattro oggetti di controllo:

- un text box che contiene il tipo di componente (attivo o passivo) considerato.
- un popup menù che contiene la lista dei componenti, estratti dalla relativa tabella del database *Component_Database*, selezionati dagli algoritmi in quanto rientrante nelle specifiche.
- un pulsante etichettato con "Cheap" che permette di includere nella lista dei componenti esclusivamente quelli di costo minimo.
- un text box che contiene la descrizione del componente selezionato nel menù a tendina.

Oltre all'handle *FigHandle* che indica in quale figura aggiungere gli oggetti di controllo, la funzione riceve in ingresso il tipo di componente *CompType* ed una serie di stringhe che permettono di memorizzare nella maschera del blocco esteso rispettivamente:

[8] si intende il pulsante "Cheap"

- l'ultimo componente selezionato (e quindi visualizzato) nel popup menù.È memorizzato nella maschera con il nome di *SelectedCompParamTag*.

- la descrizione del componente visualizzato nel popup menù memorizzato nella maschera con il nome di *CompDescriptionParamTag*.

- la lista dei componenti trovati con i relativi algoritmi di selezione. Essi sono separati dal simbolo "|" e costituiscono l'elenco completo dei componenti selezionabili nel popup menù. Il parametro della maschera che contiene questa lista è denominato *CompListParamTag*.

- la lista degli indici dei record del database in cui si trovano i componenti elencati nel popup menù. Anche in questo caso i vari elementi della lista sono separati dal carattere "|"[9]. Fra questa lista e quella dei nomi dei componenti sopra descritta esiste una associazione univoca: il quinto componente si troverà sicuramente nel database nella posizione indicata dal quinto indice della relativa lista. Il parametro della maschera che contiene la lista degli indici è denominato *CompIndicesParamTag*.

5.2.4 set_PopupMenu_value

```
set_PopupMenu_value(FigHandle, ...
    CompSelected, ParamTag1, ...
    CompDescription, ParamTag2, ...
    varargin)
```

Questa funzione è utilizzata per aggiornare sull'interfaccia componenti il popup menù e tutti gli oggetti di controllo ad esso associati descritti nel paragrafo 5.2.3. Possono verificarsi due casi distinti di aggiornamento:

- se è stato selezionato uno dei componenti elencati nel menù sarà necessario aggiornare solo la descrizione del componente e memorizzare nella proprietà '*Userdata*' della figura l'ultimo valore selezionato.

- quando l'interfaccia viene generata è necessario definire anche la lista dei componenti da aggiungere nel popup menù. In questo caso viene anche memorizzata la stringa contenente le posizioni dei componenti nel database elencati nel popup menù. In questa modalità di funzionamento alla funzione vengono passati i tag degli oggetti da aggiornare ed i valori che devono essere memorizzati.

5.2.5 AnHwDevParameters

```
[varargout] = AnHwDevParameters(AnalyzedDevice)
```

[9] anche se gli indici del database sono utilizzati come numeri vengono immagazzinati come un'unica stringa

Questa funzione contiene la lista dei parametri specifici dei componenti attivi che devono essere visualizzati nell'interfaccia componenti. Mentre le liste dei componenti passivi, specifici per ogni blocco, sono state incluse nelle "_ParamUI" corrispondenti, i dispositivi possono essere utilizzati da diversi circuiti non necessariamente appartenenti allo stesso blocco. La funzione riceve in ingresso la stringa *AnalyzedDevice* che identifica il tipo di componente attivo e, in base al suo valore, restituisce due stringhe:

- la prima stringa contiene i nomi dei parametri che verranno visualizzati nei text box dell'interfaccia componenti separati fra di loro dal carattere "|".
- la seconda stringa contiene invece i nomi ridotti (separati da "|") che verranno utilizzati per creare i nomi dei nuovi parametri da aggiungere alla maschera del blocco. Il criterio di denominazione è descritto nel paragrafo 5.1.

In entrambi i casi le sequenze di caratteri comprese fra due simboli "|" verranno ottenuti con un parsing effettuato nella funzione *AnalogDeviceDescription*, descritta nel paragrafo 5.2.2.

5.2.6 AnHwPassCompSpec

```
[varargout] = AnHwPassCompSpec(AnalyzedPassComp)
```

Come *AnHwDevParameters* descritta nel paragrafo 5.2.5, questa funzione restituisce due stringhe contenenti le informazioni relative al componente passivo *AnalyzedPassComp*, ossia

- la prima stringa contiene i nomi delle specifiche del componente passivo che verranno visualizzati nei text box dell'interfaccia componenti. Anche questo elenco è rappresentato da una stringa in cui i nomi delle specifiche sono separate dal carattere "|".
- la seconda contiene i nomi ridotti delle specifiche elencate in precedenza che serviranno per denominare i parametri della maschera contenti i corrispondenti valori numerici. Il criterio di denominazione è descritto nel paragrafo 5.1.

5.2.7 PutAnCmpControls

```
[] = PutAnCmpControls(FigHandle, LastCtrlBottom, ...
   FrameHeight1, FrameHeight2)
```

Questa funzione è utilizzata per deporre i frame dei componenti ed i relativi oggetti di controllo. Riceve in ingresso le dimensioni dei due frame e le passa a sua volta alle funzioni *AnalogCircuitDescription* e *AnalogDeviceDescription* descritte rispettivamente nel paragrafo 5.2.1 e 5.2.2. In entrambi casi l'opzione che deve essere eseguita è quella di generazione dell'interfaccia. Per ulteriori spiegazioni si rimanda ai rispettivi paragrafi.

5.2.8 PutDefaultAnCmpButtons

```
[] = PutDefaultAnCmpButtons(ParentFig, ButtonClickCallBack)
```

Questa funzione è stata realizzata sulla falsariga di *PutDefaultButtons*, già presente in CodeSimulink e utilizzata per aggiungere i pulsanti standard (OK, Cancel, Apply e Help) alle interfacce dei vari blocchi. In questo caso i pulsanti deposti sono:

- "*Find Comp*" permette di avviare la ricerca dei componenti da utilizzare (attivi e passivi) per descrivere l'equivalente analogico del blocco;
- "*Cancel*" chiude l'interfaccia componenti.
- "*Help*" apre l'help relativo alla descrizione analogica del blocco. Diversamente da quanto accadeva con *PutDefaultButtons* il brower dell'help è aperto utilizzando il comando *web* di MATLAB^{TM}.

A tutti i pulsanti è associata la medesima callback che porta all'esecuzione della funzione *DefaultAnCmpCallBack*.

5.2.9 DefaultAnCmpCallBack

```
[] = DefaultAnCmpCallBack
```

La funzione stabilisce quale pulsante sia stato premuto e, in base a quest'informazione chiama la funzione *AllComponentSearch* (trattata nel paragrafo successivo) passandogli una stringa diversa a seconda del pulsante selezionato.

5.2.10 AnDescriptionChange

```
[] = AnDescriptionChange
```

Questa funzione è in realtà una callback associata al pulsante "Find Comp" nell'interfaccia componenti ed al pulsante "Component" nell'interfaccia dei parametri analogici.

Permette di gestire il *refresh* dell'interfaccia componenti nel caso in cui si passi da una descrizione circuitale ad un'altra relativa al medesimo blocco esteso. In questo caso la funzione chiede all'utente se è sicuro di modificare la prima descrizione analogica e, in caso affermativo, modifica la finestra per visualizzare quella nuova. Nel caso in cui si sia stato premuto il pulsante "Find Comp" verranno calcolati anche i nuovi parametri relativi alla nuova descrizione, altrimenti verrà riproposta l'interfaccia nello stato iniziale. Si ricorda che:

- il nome della prima descrizione analogica è contenuto nel parametro del blocco *sim_ActualCircuit*
- il nome della seconda è ricavato chiamando la corrispondente "_ParamUI" e passandogli la stringa '*ActualCircuit*'.

Nel caso in cui sia necessario modificare l'interfaccia verranno prima cancellati gli oggetti precedenti, verrà ridimensionata la finestra e infine verranno deposti i nuovi oggetti di controllo. L'ultima operazione effettuata sarà l'aggiornamento del parametro *sim_ ActualCircuit* con nome corrispondente alla descrizione analogica appena effettuata.

5.2.11 PutTextBox2

```
ControlHandle = PutTextBox2(FigHandle, ...
  Position, Label,...
  CtrlTag, ParamName)
```

Questa funzione viene chiamata per depositare sull'interfaccia una coppia di text box affiancati orizzontalmente l'un l'altro: quello di sinistra conterrà la stringa *Label* che serve per identificare il valore numerico contenuto nel text box di destra, ricavato nel parametro *ParamName* nella maschera del blocco. Nel caso in cui tale parametro non sia ancora stato definito, nel text box verrà caricata la stringa '*-???-*'.

5.3 La ricerca dei componenti

Dopo aver definito l'interfaccia grafica è finalmente possibile stabilire i valori che assumono i componenti passivi (le resistenze) e l'elenco dei componenti attivi che soddisfano le specifiche. Per avviare la ricerca è necessario premere il pulsante "Find Component" nell'interfaccia dei componenti, gli algoritmi di ricerca verranno eseguiti e la finestra verrà aggiornata con i valori dei parametri appena trovati.

5.3.1 AllComponentSearch

```
[] = AllComponentSearch(Option, FigHandle)
```

La prima operazione che esegue questa funzione, indipendentemente dal valore della stringa *Option*, è ricavare le informazioni che provengono dal database *Component_ Database*. Questo può verificarsi in due modi:

1. nel caso in cui il database non sia mai stato letto si utilizzerà la funzione *data_ extraction* per memorizzare in un cell array le tabelle dei componenti che si desidera recuperare. Questi dati saranno memorizzati nella variabile *Database* che è una struttura dati contenente:

   - la lista delle tabelle estratte nel campo *DeviceList*;
   - il cell array contenente i dati estratti della tabella per quel determinato componente. Il campo della struttura sarà nominato secondo il seguente criterio:

   - il cell array che contiene i nomi dei campi (le colonne del database) per quel determinato componente. Anche in questo caso il campo della struttura *Database* seguirà un criterio di denominazione:

Componente	Identificativo
OpAmp	*Data*
Res	*Data*

Componente	Identificativo
OpAmp	*NamesOfColumns*
Res	*NamesOfColumns*

2. se invece la variabile globale *Database* contiene già le informazioni estratte in precedenza, non sarà necessario prenderle dal database e quindi il tempo impiegato a cercare i componenti sarà inferiore[10].

Naturalmente se una descrizione circuitale non utilizza un determinato tipo di componenti attivi o passivi, l'estrazione delle tabelle dal database non verrà effettuata e i dati saranno caricati, quando necessario, dalla variabile globale *Database*. A questo punto *AllComponentSearch* esegue diverse operazioni a seconda del valore che assume la stringa *Option* che verranno analizzate nei seguenti sottoparagrafi.

L'opzione *'Find'*

Questa opzione è il cuore del sistema di ricerca dei componenti poiché esegue fisicamente gli algoritmi dopo aver recuperato le specifiche di partenza e, in ultimo, visualizza i risultati trovati nell'interfaccia componenti. Il funzionamento del codice si basa sull'utilizzo del comando *eval* che permette di eseguire le istruzioni memorizzate sotto forma di stringa. Come si è potuto osservare nei paragrafi precedenti, gli algoritmi di ricerca, i nomi dei componenti e altri parametri sono tutti memorizzati sotto forma di stringhe nelle corrispondenti "_ParamUI" del blocco. Utilizzando il comando *eval* nel workspace (altrimenti detto *stack*) di questa funzione, è possibile effettuare tutti i confronti richiesti dagli algoritmi di scelta dei componenti e di memorizzare in determinate variabili tali valori.

In particolare, la funzione *AllComponentSearch* si occuperà di:

- recuperare dalla maschera del blocco CodeSimulink i parametri analogici (quelli inseriti nell'interfaccia parametri) che costituiranno le specifiche di partenza da cui derivare i componenti del circuito equivalente. Questi parametri sono differenti a seconda del blocco[11] e il loro elenco è memorizzato in una stringa presente nell'opzione '*NeededVariable*' della corrispondente "_ParamUI".

- recuperare gli algoritmi di ricerca dei componenti passivi contenuti, sotto forma di stringa, nell'opzione '*PassCompAlgorithm*' della "_ParamUI" del blocco.

- recuperare la lista dei componenti passivi e attivi

[10] si ricorda che uno dei limiti del Database Toolbox consiste nel tempo impiegato a estrarre i dati come si può leggere nel paragrafo 3.4.1

[11] si rimanda al paragrafo 4.4 per eventuali approfondimenti

- risalire, per ogni componente, ai corrispondenti algoritmi di ricerca inclusi nell'opzione [Nome_Componente '_*Algorithm*'] della "_ParamUI".
- trovare la lista dei componenti che rientrano nelle specifiche in base agli algoritmi di ricerca estratti;
- aggiornare la finestra di dialogo visualizzando tutti i valori trovati sia per i componenti attivi che per quelli passivi. I parametri riportati per il dispositivo si riferiscono al primo della lista di quelli potenzialmente utilizzabili.

Si ricorda che tutti i valori trovati vengono memorizzati sia nella maschera del blocco che nel campo della struttura *BlockData* memorizzata nella proprietà '*Userdata*' della finestra di dialogo. Per quanto riguarda i componenti attivi oltre alla lista con i nomi dei componenti adatti, è memorizzata anche una corrispondente lista che identifica la posizione di quel componente nel database. La visualizzazione dei parametri associati a quel determinato dispositivo si basa proprio sulla sua posizione all'interno del database.

L'opzione *'CompList'*

Quest'opzione corrisponde alla callback associata al popup menu dei componenti attivi o passivi. Selezionandone uno fra quelli elencati si risalirà alla sua posizione nel database e successivamente si visualizzeranno i parametri ad esso associati presenti nel corrispondente record.

L'opzione *'Cheap'*

L'opzione è richiamata dalla pressione dei pulsanti "Cheap" associati ad ogni componente passivo o attivo presente nella finestra di dialogo. L'azione eseguita si può definire complementare a quella di ricerca: nella lista di componenti vengono inclusi solo quelli di costo minimo (attivi o passivi a seconda del pulsante) potenzialmente utilizzabili. Il confronto viene eseguito sul campo *Price* dei componenti inclusi nella lista. Si ricorda che il pulsante "Cheap" viene abilitato solamente se è stato effettivamente ricavato un elenco di componenti.

5.3.2 find_res

```
[R1, R2, CompError, Tolerance] = find_res(Gain, GainError, ...
varargin)
```

Questa funzione è spesso richiamata nelle stringhe degli algoritmi passivi e permette di ricavare le resistenze che sono state utilizzate nei circuiti di amplificazione (invertente e non invertente) e nei partitori di tensione. In base al tipo di circuito è innanzitutto necessario ricondursi al rapporto delle resistenze: se si ha un amplificatore invertente sia sa che

$$Gain = -\frac{R2}{R1} \tag{5.1}$$

quindi il rapporto di resistenze sarà dato da

$$\frac{R2}{R1} = abs(Gain) \tag{5.2}$$

In modo analogo vengono trattati gli altri casi. A questo punto sarà necessario trovare le due resistenze reali il cui rapporto si avvicina di più a quello ottenuto in precedenza a partire da *Gain* (in parole povere si cerca il rapporto porti ad un errore di guadagno minore o uguale all'errore *GainError* che si aveva con il guadagno *Gain*). La strategia utilizzata per determinare il miglior rapporto possibile è la seguente:

- a partire dalla serie di resistenze E6 si determina la matrice contenente tutti i possibili rapporti fra le resistenze disponibili.
- si normalizza la matrice riportando tali rapporti allo stesso ordine di grandezza del rapporto di resistenze ottenuto da *Gain* iniziale.
- si sottrae ad ogni elemento della matrice normalizzata il valore del guadagno ottenendo gli errori.
- si risale alle resistenze che hanno portato ad un errore minimo inferiore uguale a *GainError*.
- nel caso non sia stata trovata nessuna coppia di resistenze si procede ad analizzare una serie di precisione maggiore[12].

Una volta trovata una coppia di resistenze adatta, viene calcolata la tolleranza in base alla serie utilizzata e si ricava l'effettivo errore di guadagno.

5.3.3 find_supply

```
[MaxVoltageReference, Vconst] = find_supply(BlockHandle)
```

Questa funzione è richiamata nelle stringhe degli algoritmi di ricerca dei componenti passivi. Permette di determinare il riferimento di tensione che si dovrà utilizzare per ottenere il circuito equivalente di un blocco esteso. Nel caso specifico, la variabile *BlockHandle* si riferisce al blocco esteso *sim_constant* che ha come equivalente analogico un partitore di tensione[13] che riceve in ingresso l'alimentazione del circuito e genera in uscita una tensione *Vconst*.

Per prima cosa la funzione *find_supply* estrae dal blocco *sim_DigHwInterface* i valori delle alimentazioni attribuite all'equivalente analogico del diagramma a blocchi. Nel caso in cui non sia presente il blocco d'interfaccia o le alimentazioni indicate siano scorrette, verranno segnalati degli errori e la ricerca dei componenti passivi sarà interrotta. Se i valori delle alimentazioni sono coerenti la funzione verificherà se è effettivamente possibile ottenere una tensione *Vconst* a partire da una delle due alimentazioni. In caso affermativo *find_supply* restituirà il riferimento adottato (o l'alimentazione positiva o quella negativa) e il valore costante *Vconst* d'uscita.

[12] nell'ordine vengono analizzate le serie di resistenze E6, E12, E24, E48, E96 ed E192

[13] per il circuito equivalente di *sim_constant* si veda il paragrafo 4.3.4

5.3.4 find_zoh_freq

```
BlockFrequency = find_zoh_frequency(BlockHandle, ...
  BlockFrequency)
```

Questa funzione ricorsiva permette di ricavare la massima frequenza del segnale in ingresso al blocco esteso *sim_zoh*. A partire dal suo handle *BlockHandle*, la funzione chiama se stessa finché non è stato trovato un blocco sorgente analogica[14]. Nel caso in cui i durante la fase di *backtracking* si trovino dei blocchi con più ingressi, ognuno di essi verrà percorso per trovare il blocco sorgente originale. la variabile *BlockFrequency* tiene memoria del valore di frequenza più alto trovato. Per ogni *Source* trovata, viene chiamata la funzione *RecoverBlockFreq*, descritta nel paragrafo 5.3.5 che si occuperà di calcolare la frequenza del segnale generato.

Nel caso in cui non siano state trovate delle sorgenti, la funzione restituirà il valore di frequenza passatole alla prima chiamata.

5.3.5 RecoverBlockFreq

```
Frequency = RecoverBlockFreq(SourceBlock, Frequency)
```

Questa funzione calcola materialmente la frequenza del segnale uscente dal blocco *SourceBlock*. I blocchi sorgenti devono avere nella loro maschera il parametro *sim_time_sequence* che contiene in un vettore la sequenza temporale di simulazione (l'ascissa di un diagramma cartesiano) ed il parametro *sim_value_sequence* che contiene l'evoluzione del segnale nel tempo (l'ordinata del diagramma cartesiano). A questo punto si calcola la trasformata di Fourier del segnale e si va a ricercare la componente dello spettro di potenza che si trova a frequenza più alta. La frequenza così trovata viene restituita alla funzione chiamante.

5.3.6 data_extraction

```
[dati, nomicolonne] = data_extraction(Component)
```

Questa funzione è utilizzata per estrarre i dati contenuti nelle tabelle del database *Component_Database* realizzato con Il software Microsoft Access. Riceve in ingresso la stringa *Component* che identifica il nome della tabelle di componenti da cui devono essere estratti i dati. I dati eventualmente recuperati verranno restituiti nel cell array *dati*. Invece il cell array *nomicolonne* conterrà i nomi dei campi estratti nella tabella *Component*. Nel caso si desiderino ulteriori dettagli si rimanda alla descrizione della medesima funzione nel paragrafo 3.4.3.

5.4 La traduzione in codice Spice

Finora ci si è occupati dei descrivere analogicamente un singolo blocco CodeSimulink cercando di determinare i componenti analogici che si utilizzerebbero per realizzare un suo

[14] nella maggior parte dei casi il ruolo di sorgente di segnale analogico è interpretato dal blocco *sim_analogIn*

equivalente circuito elettronico. Come si è notato in precedenza, ogni blocco è stato considerato come "a se stante", cioè è stato preso come un circuito isolato. Il passo successivo consiste nell'analisi circuitale di più blocchi analogici connessi insieme, passando attraverso la determinazione della *netlist* del diagramma CodeSimulink e attraverso la sua traduzione in un equivalente file d'ingresso di Spice, il *file circuito* (.cir). Il codice contenuto in questo file può essere letto da tutte le versioni (commerciali e non) del simulatore Spice e permette di poter simulare il comportamento del circuito analogico descritto nel codice. Per ulteriori spiegazioni sul linguaggio di descrizione dei circuiti in Spice si rimanda all'appendice 2. Come per il caso delle interfaccie componenti, anche in questo caso è stato necessario apportare delle modifiche al codice esistente descritte nel paragrafo successivo.

5.4.1 Le modifiche iniziali

Come già avvenuto nel caso delle interfaccie utente, descritto nel paragrafo 5.2, è stato necessario apportare delle modifiche al codice esistente, in particolar modo al compilatore già presente per la traduzione in linguaggio VHDL dei diagrammi a blocchi e alle interfaccie ad esso relative. In questo caso non verrà descritto il codice aggiunto (o modificato) in quanto i cambiamenti apportati non sono riconducibili ad un unico programma e si è preferito descrivere in linea generale le modifiche apportate ai programmi esistenti e illustrare nello specifico quelli realizzati *ex novo*. Di seguito sono comunque elencati i programmi già presenti che sono stati aggiornati per permettere la compilazione della netlist:

- *sim_DigHwInterface_ParamUI*: modificata la finestra di dialogo principale aggiungendo due edit box per l'inserimento delle alimentazioni adottate dal circuito analogico equivalente. Per la scelta *Custom target* della board è stato inoltre introdotto un popup menu che permettesse di selezionare una scheda analogica su cui sarà in seguito mappato il circuito[15]: in caso di selezione di una delle due comparirà un frame che permette di assegnare ai segnali principali del circuito equivalente (le alimentazioni e il riferimento) un piedino della scheda analogica. Queste modifiche sono state implementate in prospettiva delle successive versioni del tool CodeSimulink

- *Get_CompilerOptionDlg*: modificata la finestra di dialogo che permette di avviare la compilazione analogica digitale (VHDL) e software (in linguaggio C), aggiungendo la possibilità di compilare il diagramma in un file circuito. È stato inoltre aggiunto un edit box per la scelta del nome del file di uscita. Di default è assegnato il nome del file *.mdl*.

- *make_CS*: aggiunti alcuni parametri della maschera (tipici della compilazione analogica) che dovranno essere ereditati dalle versioni software, digitale e analogica del diagramma originale.

[15] le schede analogiche aggiunte fra le scelte sono fittizie, utilizzate solo per verificare il corretto funzionamento delle modifiche apportate

- *CS_ compile*: aggiunta una funzione di precompilazione del diagramma CodeSimulink originale e aggiunto il codice necessario per invocare il compilatore analogico nel caso in cui esso sia stato selezionato nella nella relativa interfaccia.

- *custom_ ParamUI*: aggiunto un popup menu di selezione dalla board analogica e il corrispondente frame per l'assegnazione dei piedini della scheda ai segnali analogici principali (alimentazioni e riferimento).

- *sim_ standardSink_ ParamUI*: aggiunto il codice per la creazione dell'interfaccia utente per il blocco *sim_ analogOut* (descritto nel paragrafo 5.4.7).

- *sim_ standardSource_ ParamUI*; aggiunto il codice per la creazione dell'interfaccia utente per il blocco *sim_ analogIn* (descritto nel paragrafo 5.4.6).

in particolare al blocco *sim_ DigHwInterface* che permette generare un file in linguaggio VHDL che descriva per descrivere un diagramma costituito da blocchi con descrizione hardware digitale.

5.4.2 AnHwCompile

```
[OutputFileName] = AnHwCompile(DiagramHandle, varargin)
```

Questa funzione è il cuore del compilatore analogico realizzato per ottenere i file circuito a partire da un diagramma a blocchi CodeSimulink. Viene richiamata dalla funzione *CS_ compile* subito dopo una fase di precompilazione del diagramma originale. L'handle $DiagraHandle$ si riferisce ad una versione flatenizzata del diagramma originale che comprende solo blocchi estesi in sui siano stati definiti i parametri analogici (i blocchi "gialli" e i blocchi d'interfaccia analogica *sim_ analogIn* e *sim_ analogOut*).

Dopo una fase di inizializzazione e (eventuale) assegnazione degli argomenti d'ingresso contenuti in *varargin*, *AnHwCompile* chiama la funzione *CircuitMapping* che attribuisce un ruolo preciso ad ogni blocco esteso mappando il circuito analogico equivalente al diagramma CodeSimulink. A questo punto è possibile determinare la netlist del circuito, chiamando la funzione *NodalAssignation*. In realtà vengono numerati in modo corretto solo gli ingressi e le uscite dei blocchi, ma non vengono considerati tutti i nodi Spice del circuito equivalente. Questa numerazione permette di analizzare il circuito globale come se fosse costituito da tanti sottosistemi "linkati" fra di loro: ognuno corrisponde ad un *sub circuit* di Spice che dovrà essere richiamato nel file circuito.

A questo punto si ricava la descrizione in Spice di ogni blocco richiamando la funzione *SpiceDescription* e per ognuna si assegnano:

- i valori numerici scelti per i componenti passivi (resistenze, condensatori ...).

- i nomi di eventuali modelli Spice presenti (per i circuiti che utilizzano gli amplificatori operazionali).

- i nodi Spice che non erano stati attribuiti in precedenza ai circuiti equivalenti

Come già detto ogni blocco è rappresentato come un circuito a se stante ed è dunque necessario assegnare i parametri in modo differente a seconda del tipo di circuito rappresentato.

In seguito a questa analisi si ottengono le stringhe di codice Spice finali che vengono memorizzate in due variabili distinte:

SubCircuitFunctions : contiene il codice dei sottocircuiti opportunamente corretto con i valori specifici dei componenti analogici. Questo codice verrà riportato immediatamente sotto l'header del file di uscita.

SubCircuitPrototypes : contiene le istruzioni Spice necessarie per richiamare i sottocircuiti definiti in precedenza. Ogni istruzione contiene i nodi ricavati in precedenza con la funzione *NodalAssignation*.

In ultimo viene aperto il file di uscita con estensione *.cir* e vengono stampate tutte le stringhe che compongono il codice Spice. La funzione *AnHwDescription* restituirà infine il nome del file generato che è stato memorizzato nella directory indicata in precedenza.

5.4.3 CircuitMapping

```
[BlockHandles,ListOfAdj, NumberOfSourceBlocks, ...
  NumberOfInterfaceBlocks] = CircuitMapping(DiagramHandle)
```

Questa funzione permette di effettuare una mappatura del circuito equivalente al diagramma: per prima cosa si determinano i blocchi presenti in *DiagramHandle* e le matrici di adiacenza di ogni blocco, utilizzando la funzione *adjlist* di CodeSimulink. A questo punto si prendono gli handle dei blocchi estesi e li si suddivide nelle seguenti categorie:

Blocchi Sorgente : in questa fascia rientrano il blocco *sim_ analogIn* e, in alcuni casi anche *sim_ constant*. Rappresentano delle sorgenti di segnale analogico, nello specifico, i blocchi d'interfaccia sono intese come delle sorgenti di segnale proveniente dall'esterno, mentre invece il blocco *sim_ constant* è considerato una sorgente di segnale interna al circuito, come descritto nel paragrafo 4.3.4.

Blocchi Sottocircuito : questa categoria comprende i blocchi *sim_ constant* e *sim_ repeating_ sequence* descritti come dei circuiti analogici a se stanti. Per ulteriori informazioni si rimanda al paragrafo 4.3.

Blocchi Interfaccia : solo i blocchi *sim_ analogOut* sono considerati d'interfaccia da questa funzione in quanto nel circuito equivalente del diagramma CodeSimulink svolgono esclusivamente il ruolo di connettori. Il blocco *sim_ analogIn*, nonostante sia anch'esso considerabile un connettore, in questa analisi interpreta il ruolo di sorgente poiché in esso viene memorizzato il segnale proveniente dall'esterno grazie all'utilizzo del blocco *To Workspace* di Simulink^{TM}. Per maggiori informazioni si rimanda alla funzione *AnaloSourceSignal* descritta nel paragrafo 5.4.10.

Sia gli handle che le matrici di adiacenza dei blocchi sopra descritti vengono ordinati con il seguente criterio: *Sorgenti Sottocircuiti Interfacce* In particolare vengono restituiti in uscita il vettore con gli handle dei blocchi $BlockHandles$ e l'array di celle $ListOfAdj$ con le rispettive matrici di adiacenza. Gli altri due argomenti di uscita sono invece il numero dei blocchi sorgente $NumberOfSourceBlocks$ e il numero dei blocchi d'interfaccia $NumberOfInterfaceBlocks$, necessari per rintracciare all'interno del vettore e del cell array i diversi tipi di blocchi.

5.4.4 NodalAssignation

```
[NodeList, NodeNumber] = NodalAssignation(...
AllBlockHandles, ListOfAdj, varargin)
```

Questa funzione è presente all'interno del compilatore *AnHwCompile*: permette di determinare la netlist del diagramma a blocchi in base al vettore $AllBlochHandles$ contenente gli handle dei blocchi presenti e all'array di celle $ListOfAdj$ contenente le corrispondenti matrici di adiacenza. Vengono inoltre passati altri tre argomenti che sono

- l'handle $BlockHandle$ del blocco attualmente "esplorato";
- l'array di celle $NodeList$ che contiene la lista dei nodi assegnati al blocco: ogni elemento contiene un vettore in cui sono riportati per primi i nodi d'ingresso e per ultimi quelli di uscita.
- il numero $NodeNumber$ che corrisponde al contatore dei nodi assegnati ed aumenta ogni volta che viene esplorato un blocco che non sia d'interfaccia.

La funzione lavora in modo ricorsivo e ogni volta che chiama se stessa numera nello stesso modo l'uscita del blocco $BlockHandle$ e l'ingresso del blocco successivo $NextBlockHandle$. Quest'ultimo handle viene poi passato alla successiva chiamata della funzione insieme ai valori aggiornati di $NodeList$ e $NodeNumber$ finché tutte le porte (sia ingressi che uscite) del blocco non siano state numerate o finché non sia possibile identificarne il successivo. L'esplorazione avviene in modo del tutto analogo a quella che si effettuerebbe per un albero binario. La stessa operazione viene ripetuta a partire da ogni blocco sorgente finché tutte le porte di ogni blocco non vengono numerate. I blocchi di interfaccia mantengono la stessa numerazione sia per l'ingresso che per l'uscita essendo una sorta di buffer non rappresentato in Spice.

5.4.5 SpiceDescription

```
[varargout] = SpiceDescription(BlockHandle, varargin)
```

Questa funzione può essere utilizzata in due modi a seconda del numero degli argomenti d'ingresso passati:

- nel caso in cui riceva in ingresso solo la variabile *BlockHandle*, la funzione controllerà se la descrizione Spice del blocco *BlockHandle* prevede l'utilizzo di modelli circuitali complessi per i componenti attivi. Attualmente solo i circuiti che utilizzano gli amplificatori operazionali possono richiamare muduli dalla libreria *OpAmp.lib*. Come già detto essa contiene i sottocircuiti, realizzati dalle aziende, che descrivono in modo accurato questo tipo di dispositivo. Per ulteriori spiegazioni sul funzionamento dei modelli Spice di libreria si rimanda al paragrafo 2.2 dell'appendice.

- nel caso in cui oltre all'handle del blocco venga passata anche la stringa che identifica il tipo di blocco esteso, allora la funzione restituirà in uscita due stringhe; la prima contiene la descrizione in linguaggio Spice del circuito equivalente del blocco CodeSimulink: essa non è altro che la codifica in *sub circuit* del circuito equivalente del blocco. Poiché alcuni blocchi posseggono più di una descrizione analogica, sarà necessario determinare quella attualmente utilizzata. Il codice in questione contiene dei simboli "@" in corrispondenza dei valori numerici che corrispondono ai componenti attivi e passivi del circuito. La seconda stringa restituita dalla *SpiceDescription* contiene i riferimenti necessari per estrarre della maschera valori numerici descritti prima. La funzione *AnHwCompile* utilizzando le suddette stringhe effettuerà un parsing con cui si otterrà il sottocircuito equivalente descritto nella sua interezza e pronto per essere incluso nel file circuito d'uscita.

5.4.6 sim_analogIn_ParamUI

```
[varargout] = sim_analogIn_ParamUI(Action, varargin)
```

Questa funzione è la "_ParamUI" (ovvero la funzione che genera l'interfaccia di dialogo principale) del blocco esteso *sim_analogIn*[16]. Questo blocco è stato realizzato durante il progetto ed è analogo al suo corrispondente digitale (*sim_digIn*). Il suo scopo principale è quello di mappare in una scheda analogica i segnali presenti nel diagramma che sono considerati ingressi esterni del circuito analogico equivalente: è l'utente che sceglie nella finestra di dialogo i piedini della scheda analogica da assegnare a quella linea nel diagramma CodeSimulink. Il numero di pin supportati varia da uno a due a seconda che il segnale (rappresentato nel diagramma a blocchi) sia inteso *single-ended* o differenziale nel circuito analogico equivalente. La scheda visualizzata corrisponde a quella selezionata nel blocco *sim_DigHwInterface*. Nel caso in cui il blocco non sia presente verrà segnalato l'errore. Come per tutti gli altri blocchi estesi è possibile selezionare i descrivere analogicamente il segnale di uscita e, in questo caso, anche quello di uscita, utilizzando i pulsanti "Analog Hw Parameters".

Il blocco *sim_analogIn* può venire interpretato dal compilatore *AnHwCompile* come una sorgente di segnale analogico nel caso in cui ad esso siano connessi un blocco *source* di Simulink™ ed un blocco *To Workspace* nel modo illustrato in figura 5.4. Eseguendo una simulazione del modello, il segnale in ingresso al blocco *sim_analogIn* viene immagazzinato

[16] in realtà, come accadeva per il blocco *sim_digIn*, le istruzioni per generare la finestra di dialogo sono contenute nella funzione *sim_standardSource_ParamUI*

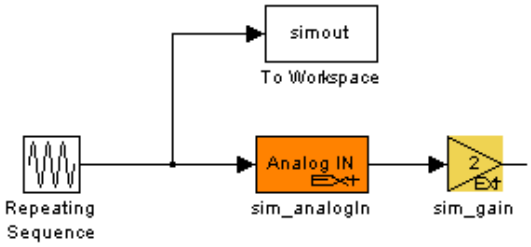

Figura 5.4. Collegamento del blocco *To Workspace* al blocco *sim_ analogIn*

come un vettore di valori nel workspace di base grazie al blocco Simulink^{TM} *To Workspace*. In fase di precompilazione la funzione *AnalogSourceSignal*, descritta nel paragrafo 5.4.10 memorizza il vettore di che rappresenta la sorgente e il suo andamento nel tempo di simulazione (entrambi definiti come dei vettori) fra i parametri del blocco *sim_ analogIn*. Questo blocco, a differenza del *To Workspace* e delle sorgenti di Simulink^{TM}, non verrà eliminato durante la divisione del diagramma nelle sue componenti software, analogica e digitale e potrà essere interpretato come una sorgente di segnale analogica[17].

5.4.7 sim_analogOut_ParamUI

```
[varargout] = sim_analogOut_ParamUI(Action, varargin)
```

Questa funzione è la "_ParamUI" del blocco esteso *sim_ analogOut*. Questo blocco d'interfaccia permette di mappare all'interno di un diagramma CodeSimulink quelle linee che che devono essere considerate dei segnali di uscita nel circuito analogico equivalente. Analogamente al caso precedente, la funzione crea una finestra di dialogo[18] che permette all'utente di selezionare i pin della scheda analogica da attribuire a quella linea. Anche in questo caso se il blocco *sim_ DigHwInterface* non è presente nel diagramma non sarà possibile indicare la scheda selezionata e di conseguenza verrà segnalato errore. Diversamente dal blocco *sim_ analogIn* non è prevista la possibilità di ereditare dei parametri dai blocchi precedenti ma è comunque possibile indicare le caratteristiche analogiche del segnale d'uscita cliccando sul pulsante "Analog Hw Parameters".

5.4.8 Analog_Board_PinMapDlg

```
[varargout] = Analog_Board_PinMapDlg(Action, varargin)
```

Questa funzione permette di generare e modificare le finestre di dialogo relative alla scheda analogica. Per questo motivo è legata alla generazione delle interfacce principali dei blocchi:

[17] in questo caso il blocco *sim_ analogIn* verrà considerato da Spice come una sorgente di onde lineari definite a tratti

[18] analogamente al blocco *sim_ digOut* per generare l'interfaccia principale verrà chiamata la funzione *sim_ standardSink_ ParamUI*

- *sim_ analogIn* ⇒ *sim_ analogIn_ ParamUI*
- *sim_ analogOut* ⇒ *sim_ analogOut_ ParamUI*
- *sim_ DigHwInterface* ⇒ *custom_ ParamUI*

Oltre ai blocchi sono indicate anche le funzioni "_ParamUI" da cui verranno ricavati i parametri da inserire nei list box e nei popup menu. In tutti i casi elencati la scheda visualizzata dipende comunque dalla selezione effettuata nel popup menu "Analog" presente nella finestra di dialogo principale del blocco *sim_ DigHwInterface*. Il menù a tendina è disponibile se e solo se è stata selezionato *Custom target* nel popup menu delle "Target Board". La funzione in questione non si occupa solo di generare gli oggetti di controllo, ma anche di gestire la memorizzazione delle scelte effettuate e di cancellare gli oggetti stessi. Le azioni eseguite dipendono strettamente dal valore assunto dalla stringa *Action*:

- se *Action = 'VERTDIM'* sarà restituita la dimensione del frame di assegnazione dei pin della scheda. Questa dimensione(standard) è necessaria per stabilire l'altezza finale della finestre di dialogo in cui saranno piazzati i vari oggetti di controllo.
- se *Action = 'CREATE'* verranno creati gli oggetti di controllo per la selezione dei piedini della scheda e del tipo di segnale analogico rappresentato. Tutte le informazioni da riportare dentro i list box o i popup menu sono contenute nella "_ParamUI" corrispondenti, come indicato nell'elenco iniziale. Il box di dialogo sarà composto sempre da un popup menu etichettato "Cell Port (AnHw)" che contiene i segnali analogici a cui dovranno essere assegnati i pin della scheda. Per il blocco *sim_ DigHwInterface* i segnali da assegnare sono le alimentazioni, il riferimento di tensione e (eventualmente) il clock del sistema ovvero i segnali sempre presenti all'interno di un circuito elettronico. Essendo tutti *single-ended* ad ognuno potrà essere assegnato al massimo un pin della scheda. Nel caso dei blocchi d'interfaccia i segnali non avranno un nome specifico[19] e sarà possibile scegliere se quel determinato segnale nel circuito analogico equivalente utilizza uno o due pin (caso differenziale). L'attribuzione dei piedini scelti per il segnale selezionato nel popup menù avviene attraverso l'utilizzo di due listbox: il primo, etichettato come "Port pin (AnHw)" contiene il numero di piedini specifici per quel segnale, mentre il secondo, etichettato come "Conn pin (AnHw)" contiene la lista con i nomi dei piedini della scheda analogica selezionata nel blocco *sim_ DigHwInterface*. L'assegnazione avviene cliccando in uno dei pin qui elencati, dopo aver selezionato il pin del segnale. Gli ultimi oggetti di controllo della finestra sono due pulsanti di reset che permettono di assegnare a tutti i piedini presenti il valore *Not used* o *Any pin*.
- se *Action = 'STORE'* tutte le assegnazioni dei piedini eseguite verranno memorizzate. Questa opzione si verifica nel caso in sui sia stato premuto il pulsante "Apply" dell'interfaccia e nella maschera del blocco verranno memorizzati i pin assegnati per ogni segnale presente nel popup menù.

[19] è l'utente che decide quali segnali del diagramma rappresentano nella controparte analogica i segnali di uscita del circuito

- se *Action = 'DELETE'* sarà necessario rimuovere dalla finestra tutti gli oggetti di controllo (popup menù, text box e list box) precedentemente aggiunti. Questa opzione può essere solo chiamata dalla *custom_ ParamUI* nel caso in cui sia stata selezionata un'altra *Target board* diversa dalla custom oppure nel popup menù "Analog" sia stata selezionata l'opzione *none*.

5.4.9 Analog_Board_Callback

```
[varargout] = Analog_Board_Callback(FigHandle, ...
CallbackType, varargin)
```

Questa funzione contiene tutte le callback associate agli oggetti di controllo utilizzati per l'assegnazione dei pin della scheda analogica. A seconda del valore della stringa *CallbackType* vengono eseguite diverse azioni:

- *CallbackType = 'MAIN'* è associata alla selezione delle opzioni del popup menù "Analog" presente soltanto nell'interfaccia di dialogo del blocco *sim_ DigHwInterface*. Se è stata selezionata l'opzione *none*, il frame di assegnazione dei pin della scheda analogica dovrà essere rimosso e con esso tutti gli oggetti di controllo presenti. In questo caso la dimensione finale della finestra risulterà ridotta. Nel caso in cui si selezioni una scheda analogica, la finestra dovrà essere allargata e gli oggetti di controllo saranno nuovamente depositati. In entrambi i casi la funzione prevede un riposizionamento nella finestra degli oggetti di controllo non eliminati e un aggiornamento dell'altezza della finestra.

- *CallbackType = 'PORT'* è associata alla selezione del segnale nel popup menù "Cell port (AnHw)": a seconda del numero di pin richiesti dal segnale verrà aggiornato il list box "Port pin (AnHw)". Naturalmente saranno riportate (se presenti) le assegnazioni effettuate in precedenza e memorizzate nella maschera del blocco.

- *CallbackType = 'PORTPIN'* in realtà non porta all'esecuzione di nessuna operazione. Serve esclusivamente per selezionare il pin del segnale (indicato nel list box "Port pin (AnHw)") a cui verrà in seguito assegnato un pin della scheda.

- *CallbackType = 'CONNPIN'* è associata alla selezione di uno dei pin elencati nel list box "Conn pin (AnHw)" e permette di attribuire il piedino della scheda analogica selezionato al pin evidenziato in quel momento nel list box "Port pin (AnHw)". In questo caso accanto a quel determinato pin verrà riportato fra parentesi quello della scheda che gli è stato assegnato.

- *CallbackType = 'ANY PIN'* o *CallbackType = 'NOT USED'* sono associate ai rispettivi pulsanti e permettono di resettare le assegnazioni effettuare fino ad allora riportandole rispettivamente a "Any pin" o a "Not used".

5.4.10 AnalogSourceSignal

```
[varargout] = AnalogSourceSignal(DiagramHandle, varargin)
```

Questa funzione è utilizzata in fase di precompilazione, cioè prima che avvenga la traduzione del diagramma nel suo equivalente file circuito. Come già spiegato nel paragrafo 5.4.6 il blocco *sim_ analogIn* viene considerato una sorgente di segnale analogico in fase di compilazione, nel caso in cui ad esso siano collegati un blocco *source* e il blocco *To Workspace* di Simulink^{TM}. In questo caso, il vettore che rappresenta l'evoluzione del segnale del *source* nel tempo, viene immagazzinato nel workspace di base. La funzione *AnalogSourceSignal* esegue le seguenti operazioni:

- recupera i parametri *StartTime* e *StopTime* memorizzati nella maschera del diagramma a blocchi;
- a partire da ognuno dei blocchi *sim_ analogIn* trova i corrispondenti blocchi Simulink^{TM} sorgenti connessi;
- verifica che ad ogni blocco sorgente trovato sia connesso (oltre al corrispondente *sim_ analogIn* un blocco *To Workspace*.

Se non si sono verificati degli errori durante questa analisi ed è stato trovato il blocco *To Workspace*, viene ricavato il vettore dei tempi di simulazione a partire dagli istanti *StartTime* e *StopTime*. La dimensione del vettore dei tempi viene determinata in base al numero di campioni del segnale sorgente, associato al relativo blocco *To WorkSpace*. In Spice questo segnale viene rappresentato come un'onda lineare a tratti e simulata quindi con il comando *PWL*.

5.5 Altre funzioni

In questa categoria rientrano alcune funzioni utilizzate da quelle descritte in precedenza. Si è preferito descriverle a parte poiché sono state pensate per funzionare correttamente in un contesto più generale rispetto a quello in cui sono state adoperate. In particolare *get_ string_ piece* può essere impiegata per effettuare parsing di qualunque tipo di stringa con qualsiasi carattere di delimitazione interno. Analogamente *IsNextPrevBlock* permette di determinare le connessioni di un blocco Simulink^{TM}, indipendentemente dal fatto che sia esteso o meno.

5.5.1 get_string_piece

```
c = get_string_piece(DataString, varargin)
```

Questa funzione permette di trovare all'interno di una stringa *DataString* le sequenze di caratteri che sono separate da dei caratteri delimitatori come ad esempio virgole, asterischi o underscore. Tali sequenze vengono riconosciute e memorizzate nel cell array *c*. Come ulteriore argomento d'ingresso è possibile passare una stringa contenente i caratteri

considerati delimitatori. Nel caso in cui questa non venga passata verranno considerati delimitatori i seguenti caratteri: virgola (,), et (@) e apostrofo ('). La funzione viene utilizzata per effettuare un parsing delle stringhe che descrivono il blocco CodeSimulink nell'equivalente sotto-circuito Spice. In questo caso il carattere di delimitazione "@" è sostituito con il nome del sotto-circuito (corrispondente al nome del blocco), i parametri che identificano i componenti passivi del circuito (resistenze) e con i nomi dei componenti attivi scelti (nel caso in cui questi ultimi siano a loro volta dei sotto-circuiti). In queste occasioni si è preferito utilizzare *get_string_piece* piuttosto che il comando *str_tock* congiunto ad un ciclo while per migliorare la leggibilità del codice di compilazione dei singoli blocchi.

5.5.2 IsNextPrevBlock

```
[SearchedBlock] = IsNextPrevBlock(BlockHandle, ...
Direction, varargin)
```

Questa funzione in base al valore *Direction* cerca i blocchi che precedono o succedono *BlockHandle*. A questo punto il vettore di uscita cambia a seconda del numero di argomenti d'ingresso:

- se non sono presenti ulteriori argomenti la funzione restituisce il vettore degli handle trovati.
- se è presente un terzo argomento (rinominato poi *BlockType*), la funzione cerca fra gli handle trovati quelli che corrispondono ai blocchi dello stesso tipo indicato da *BlockType*.

In realtà la variabile *SearchedBlock* non è sempre un vettore: nel caso in cui si cerchino i blocchi precedenti (l'analisi partirà quindi dagli ingressi di *BlockHandle*) potrà essere restituito uno e un solo handle poiché in Simulink^{TM} non sono ammessi più segnali entranti nella stessa porta d'ingresso. Se invece si stanno cercando i blocchi che seguono *BlockHandle* è possibile che sia restituito un vettore di handle poiché una stessa porta di uscita può essere connessa a più di un blocco.

Capitolo 6

Un esempio di simulazione

In questo capitolo verrà illustrato un esempio del funzionamento del codice descritto nel capitolo 5. Per prima cosa verranno scelti i componenti che definiscono il circuito analogico equivalente di ogni blocco CodeSimulink presente e, una volta fatto ciò, verrà generato il codice Spice per il medesimo circuito, utilizzando il blocco *sim_ DigHwInterface*.

6.1 Il diagramma di prova

Si ha il seguente modello CodeSimulink riportato in figura 6.1

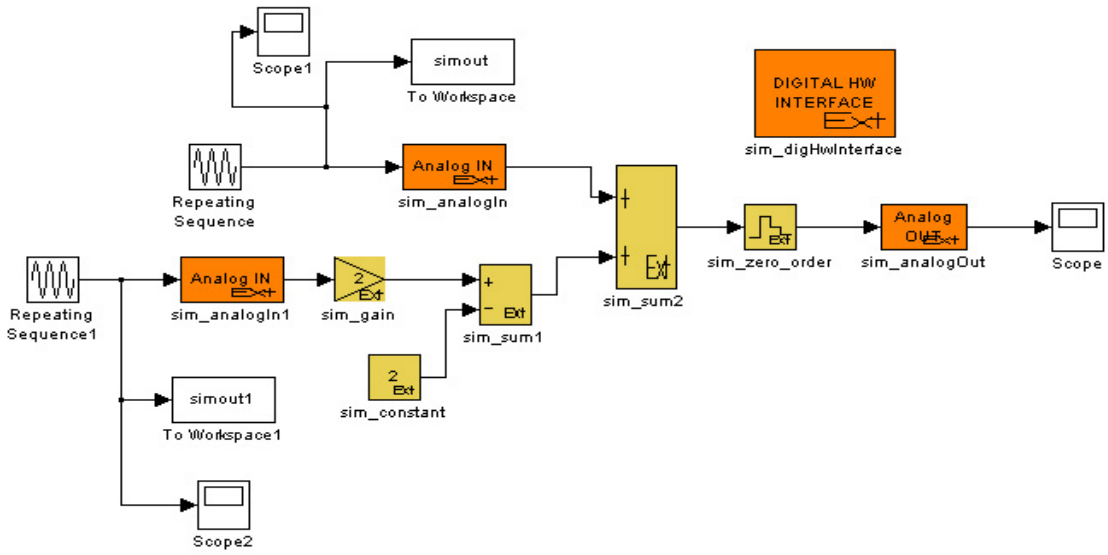

Figura 6.1. Diagramma a blocchi di esempio

costituito da due blocchi *sim_ sum2*, un blocco *sim_ gain* un blocco *sim_ constant* ed infine un blocco *sim_ zoh*. I segnali d'ingresso sono forniti da due blocchi di tipo *Repeating Sequence*. Essendo delle sorgenti di Simulink^{TM}, è necessario utilizzare i blocchi *sim_ analogIn* per interfacciarli con CodeSimulink. Il segnale di uscita del circuito è quello campionato dal *sim_ zoh* e si interfaccia verso l'esterno attraverso un blocco *sim_ analogOut*. Si può notare infine la presenza di tre *Scope* per visualizzare il segnale e di due *ToWorkSpace* per immagazzinare i segnali di uscita dalle rispettive sorgenti Simulink^{TM}.

La controparte analogica di questo modello si ottiene unendo fra di loro tutte le descrizioni circuitali dei blocchi presenti nel diagramma, descritti nel paragrafo 4.3. In figura 6.2 si può notare il circuito equivalente per il diagramma rappresentato in figura 6.1. Come si può notare i due blocchi *sim_ sum2* hanno due rappresentazioni analogiche distin-

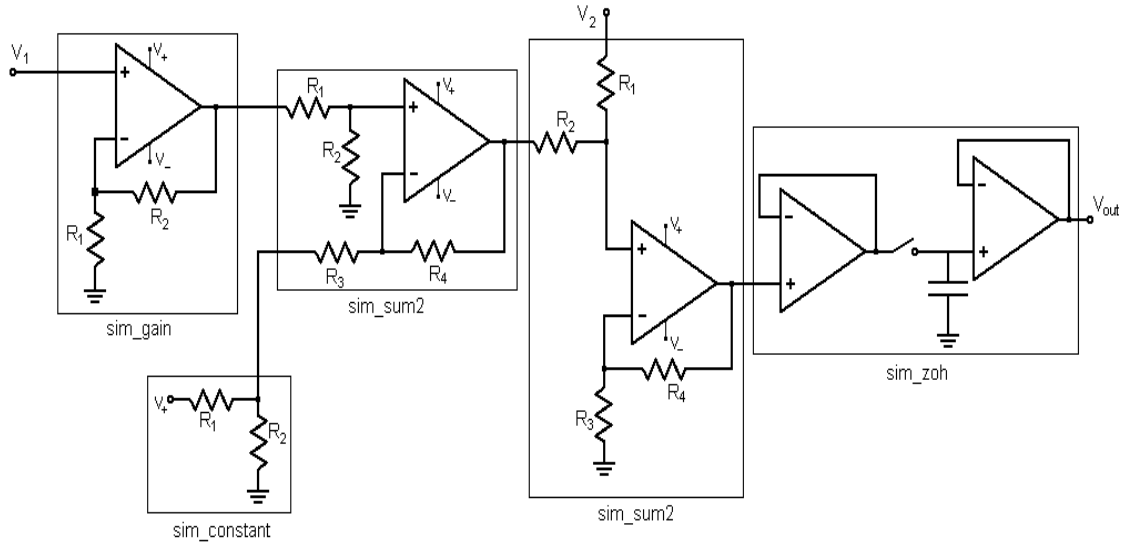

Figura 6.2. Circuito analogico equivalente del diagramma a blocchi

te uno come sommatore non invertente e l'altro come amplificatore differenziale. Il blocco guadagno è rappresentato come un amplificatore non invertente con guadagno pari a 2, mentre il blocco *sim_ constant* è un partitore sulla tensione di alimentazione con guadagno[1] unitario. Per quanto riguarda il blocco *sim_ zoh* si è adottata una rappresentazione standard riportata in figura A.19 e descritta nel paragrafo A.7. Gli altri blocchi fra cui il *sim_ DigHWInterface* ed *sim_ analogIn* non avendo una controparte analogica circuitale non sono stati rappresentati.

Si suppone che i blocchi siano già stati collegati e che siano già stati assegnati i parametri analogici standard (illustrati nel paragrafo 4.2) nelle interfacce parametri corrispondenti

[1]per il blocco *sim_ constant* il guadagno è inteso sempre come rapporto fra il segnale d'uscita e quello d'ingresso non come il rapporto di partizione

descritte nel paragrafo 5.2. Nei paragrafi successivi verranno presentati i passi necessari per la selezione dei componenti e la compilazione del codice Spice equivalente

6.2 La scelta dei componenti

Poiché le azioni da eseguire sono le medesime per tutti i blocchi CodeSimulink, le operazioni da effettuare verranno presentate soltanto per il blocco *sim_ gain* che utilizza sia componenti passivi (resistenze $R1$ e $R2$) che attivi (gli amplificatori operazionali).

Cliccando sul blocco *sim_ gain* si aprirà innanzitutto l'interfaccia principale, come si può vedere in figura 6.3.

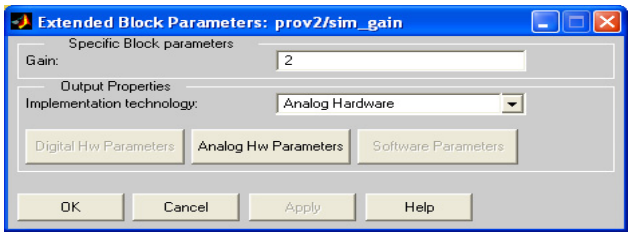

Figura 6.3. Interfaccia Principale del blocco *sim_ gain*

In essa è possibile settare il valore del guadagno e, cliccando sul pulsante "Analog Hw Parameters", aprire la finestra di dialogo in cui vengono inserite le specifiche del segnale *analogico* di uscita. L'interfaccia parametri è illustrata in figura 6.4. Come già detto nel

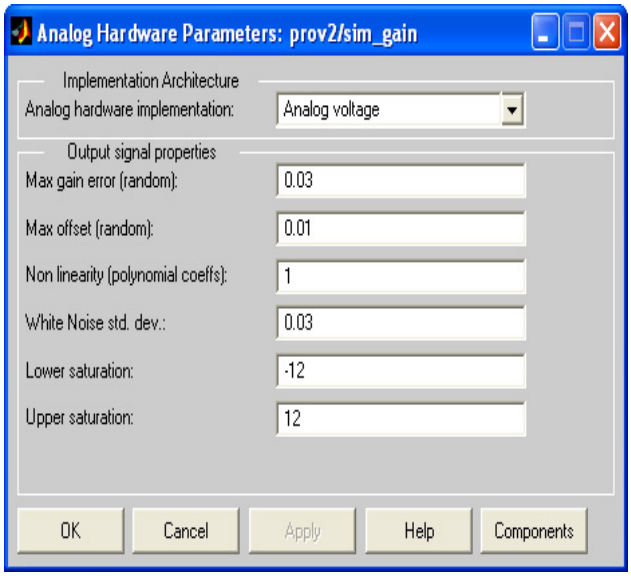

Figura 6.4. Interfaccia Parametri

paragrafo 6.1 si parte dall'ipotesi che le specifiche del segnale analogico siano già state assegnate. A questo punto cliccando sul pulsante "Components", si aprirà l'interfaccia componenti: nel caso in cui sia la prima volta che si descrive in modo analogico il blocco, l'interfaccia avrà l'aspetto riportato in figura 6.5. Nel caso in cui siano state effettuate delle

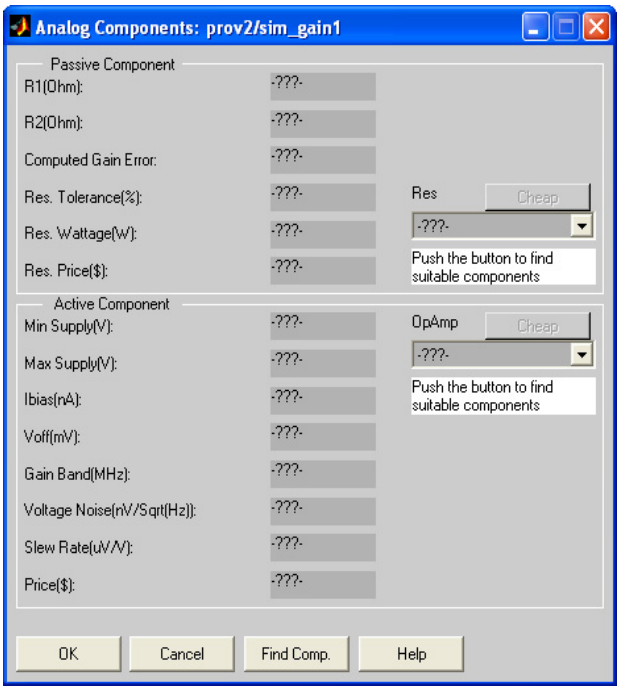

Figura 6.5. Interfaccia Componenti Inizializzata

precedenti descrizioni analogiche verranno visualizzati i dati sui circuiti precedenti. Come già affermato, alcuni blocchi prevedono più circuiti equivalenti a seconda della selezione dei parametri nell'interfaccia principale: se l'utente desidera effettuare una descrizione analogica differente rispetto a quella inizialmente specificata (ad esempio desidera che il blocco *sim_ gain* sia interpretato come un partitore di tensione[2] piuttosto che come un amplificatore non invertente) verrà visualizzato un messaggio di avvertimento che chiederà conferma all'utente del cambiamento di descrizione e, nel caso in cui sia premuto il pulsante "OK", darà avvio all'aggiornamento vero e proprio dell'interfaccia.

A questo punto è possibile procedere con la scelta dei componenti veri e propri che viene lanciata cliccando sul pulsante "Find Comp." Se tutti i parametri analogici sono stati inseriti correttamente l'interfaccia componenti visualizzerà i valori aggiornati per tutti gli oggetti presenti: nel caso del blocco *sim_ gain* tradotto in amplificatore non invertente, si avrà la finestra riportata in figura 6.6 con i valori calcolati. All'utente è lasciata la scelta del componente finale da utilizzare, attraverso la selezione nei relativi popup menu dell'operazionale e della serie di resistenze più adatte. Prima di selezionare una delle

[2] il valore del guadagno sarà compreso fra 0 e 1

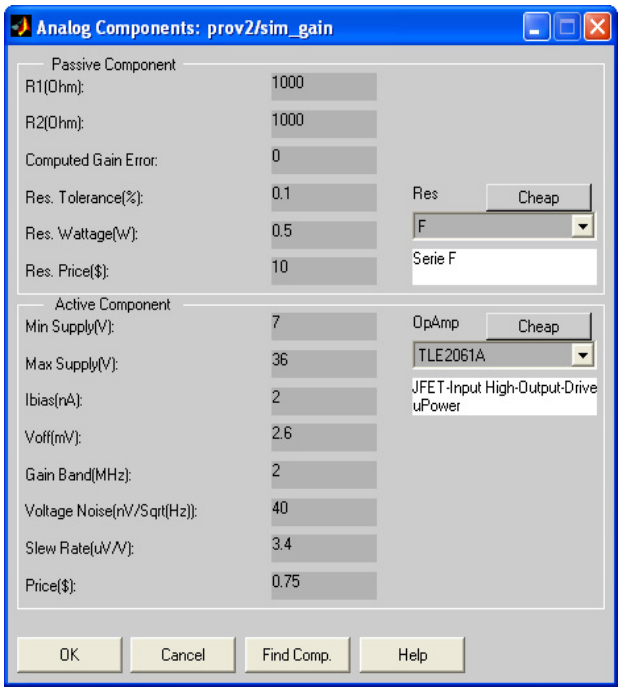

Figura 6.6. Interfaccia Componenti con i valori

opzioni nel menù a tendina l'utente può scartare a priori i componenti più costosi, andando a includere solo i più economici della lista: questa operazione è avviata con la pressione del corrispondente pulsante "Cheap".

La scelta dei componenti viene ripetuta in modo analogo per tutti gli altri blocchi estesi (esclusi i blocchi d'interfaccia *sim_analogIn*, *sim_analogOut* e *sim_DigHwInterface*). Si ricorda che la descrizione analogica di alcuni blocchi può essere effettuata solo se sono state verificate determinate condizioni, in particolare:

- per descrivere il circuito del blocco *sim_constant* è necessario che sul diagramma sia presente il blocco *sim_DigHwInterface* e in esso sia stato indicato il valore delle alimentazioni globali del circuito analogico: la controparte analogica del blocco costante non è altro che un partitore di tensione su una delle alimentazione (a seconda del segno del valore costante). In caso di errore un box di dialogo allerterà l'utente sulla natura del problema.

- per effettuare la ricerca dell'amplificatore di Sample & Hold più adatto è necessario conoscere la frequenza massima del segnale entrante nel blocco *sim_zoh*: nel caso in cui si effettui la ricerca dei componenti senza che il blocco sia connesso non sarà possibile risalire alla frequenza del segnale che verrà interpretata come nulla. L'utente sarà comunque allertato con un box di dialogo di questo problema.

Una volta che sono stati descritti analogicamente tutti i blocchi si può lanciare la simulazione del diagramma ottenuto e controllare con i blocchi *Scope* le forme d'onda ottenute.

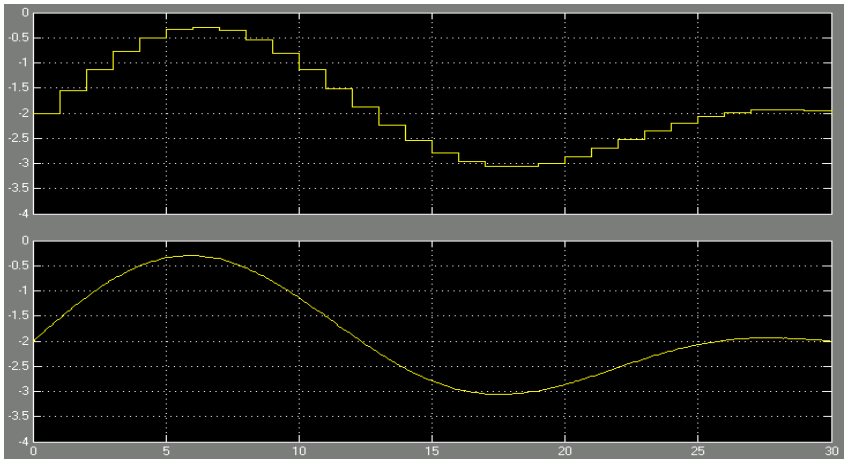

Figura 6.7. Segnali d'ingresso (in alto) e d'uscita (in basso) del blocco*sim_ zoh*

In figura 6.7 è riportato il risultato della simulazione, in particolare nel diagramma in alto è rappresentato il segnale d'ingresso del blocco *sim_ zoh* mentre nell'altro è rappresentato il suo segnale di uscita.

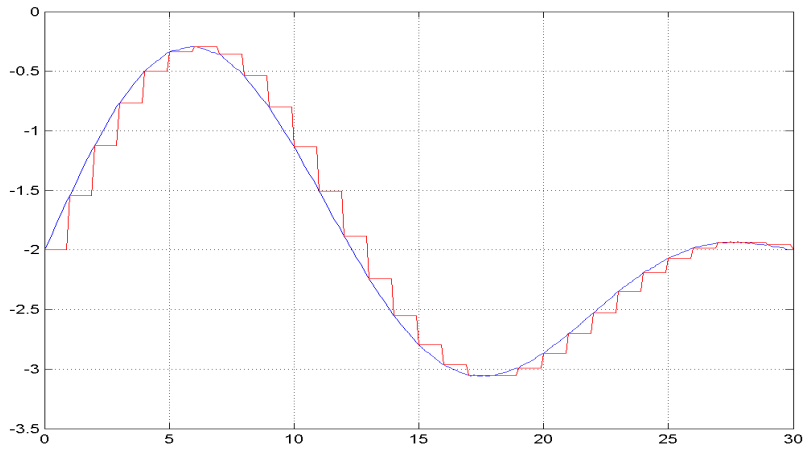

Figura 6.8. Plotting dei segnali d'ingresso e d'uscita dal blocco *sim_ zoh*

Per confrontare le due forma d'onda si può utilizzare MATLAB^{TM} per plottarle entrambe. Successivamente verrà effettuato un confronto fra il plot e la simulazione in Spice

del circuito analogico equivalente al diagramma CodeSimulink. In figura 6.8 è riportato il plotting dei segnali d'ingresso e d'uscita del blocco esteso *sim_ zoh*.

La compilazione per ottenere il file circuito da simulare è descritta di seguito.

6.3 La compilazione analogica del diagramma a blocchi

Per effettuare questa operazione è necessario che sia presente un blocco *sim_ DigHwInterface* nel diagramma. Cliccandoci sopra si aprirà la finestra di dialogo mostrata in figura 6.9. In questa fase di compilazione è necessario che siano indicate almeno le alimentazioni del

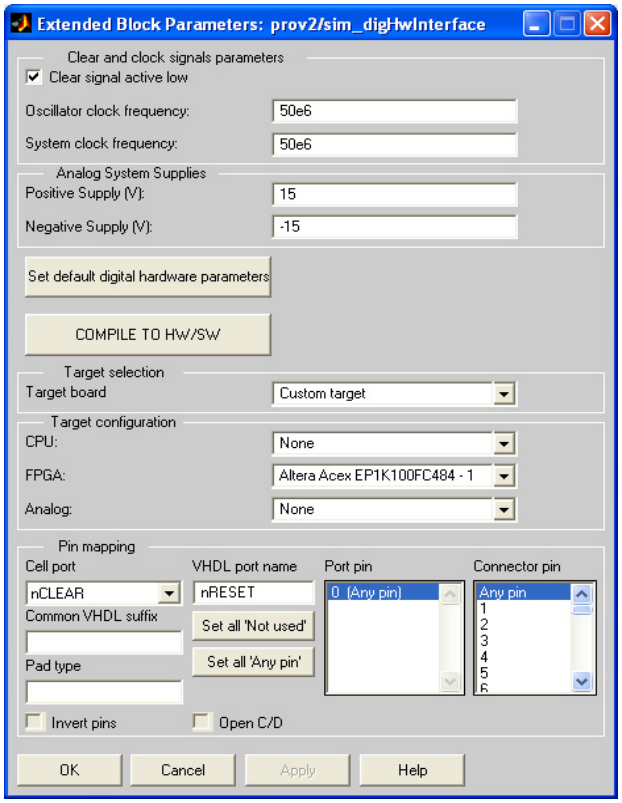

Figura 6.9. Interfaccia Principale di *sim_ DigHwInterface*

sistema. È possibile selezionare anche la scheda analogica, nel popup menù etichettato con "Analog:": questa opzione è presente solo se è stata selezionata la scheda custom nel menù "Target Board". I nomi delle schede presenti sono fittizzi, non essendo stata ancora implementata questa caratteristica. Selezionando uno di essi verrà aggiunto un frame per il *mapping* sulla scheda dei segnali analogici principali (le alimentazioni, il riferimento ed un eventuale clock).

Cliccando sul pulsante "COMPILE TO HW/SW" verrà aperta la finestra di dialogo in figura 6.10. Per generare il file circuito sarà necessario spuntare l'opzione nel check

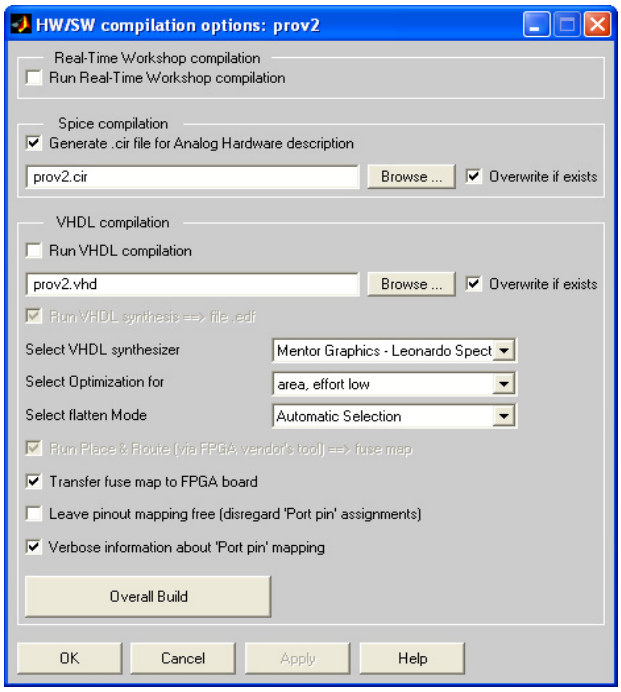

Figura 6.10. Finestra delle opzioni di compilazione di *sim_ DigHwInterface*

box relativo alla compilazione Spice[3]. È anche possibile indicare il nome del file di uscita, inizialmente nominato come il modello CodeSimulink. A questo punto è possibile avviare la compilazione vera e propria cliccando sul pulsante "Overall Build".

Nel caso in cui non si siano verificati errori, verrà generato un file di uscita con estensione *.cir* che descriverà il circuito analogico rappresentato dal modello CodeSimulink. In figura

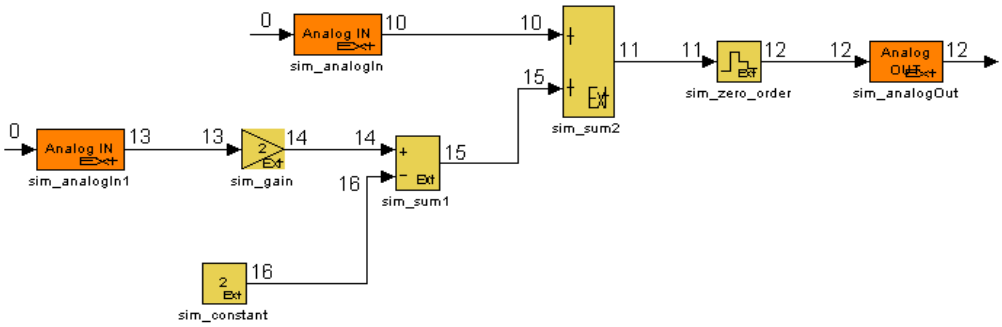

Figura 6.11. Nodi assegnati al diagramma a blocchi

[3]si consiglia di spuntare gli altri tipi di compilazione che potrebbero generare errori nel caso cui sia stata fornita esclusivamente una descrizione analogica del sistema

6.11 si può notare come sono stati assegnati i nodi Spice del circuito a partire dal blocco denominato *sim_ analogIn* che, essendo considerato come un blocco sorgente, ha l'ingresso scollegato e numerato con 0. Sarà il compilatore *AnHwCompile* descritto nel paragrafo 5.4.2 a considerare validi esclusivamente i nodi non nulli.

Si utilizza a questo punto un simulatore Spice che supporti un numero sufficiente di nodi (per questo esempio è stato utilizzato il simulatore *PSpice 9.1* della Orcad) e si va a simulare il file circuito ottenuto in uscita. Analizzando il segnale d'uscita dal Sample & Hold (ovvero il blocco CodeSimulink *sim_ zoh*) si ha la seguente forma d'onda Il segnale

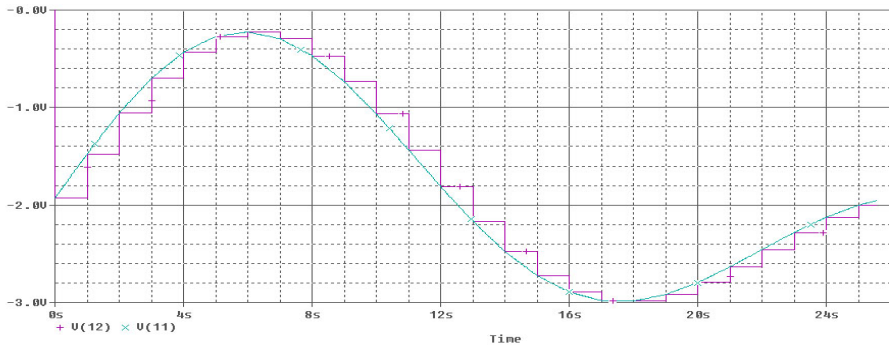

Figura 6.12. Simulazione Spice del segnale d'uscita

in verde rappresenta la tensione al nodo 11 ossia quella in ingresso al circuito di Sample & Hold. Il segnale rosso corrisponde invece alla tensione campionata con la frequenza indicata nel blocco *sim_ zoh*.

Confrontando la caratteristica trovata con la simulazione Simulink^{TM} del diagramma originale, proposta in figura 6.7 si può notare che l'andamento è uguale come anche il periodo di campionamento del circuito di Sample & Hold. Di contro si può notare che la tensione di partenza nella simulazione in Spice si assesta su valori inferiori rispetto a quella in MATLAB^{TM}. Questo effetto è sicuramente dovuto alla presenza del blocco *sim_ constant* che è descritto analogicamente come un partitore di tensione sull'alimentazione (in questo caso quella positiva) che assorbe una parte della corrente destinata all'alimentazione degli operazionali dei blocchi *sim_ gain* e *sim_ sum2*. Questo effetto può essere limitato in simulazione aggiungendo un buffer prima del partitore di tensione di *sim_ constant*, limitando in questo modo l'assorbimento di corrente di questo circuito.

6.4 Un caso di studio: il termostato

Il diagramma riportato in figura 6.13 si riferisce al modello di un termostato che serve a mantenere costante la temperatura dell'acqua al suo interno. Il modello è composto da blocchi descritti in hardware analogico (i circuiti di amplificazione del segnale) ed in software (il sistema di controllo e di confronto della temperatura di riferimento). Il blocco *heater* rappresenta il riscaldatore che riceve in ingresso la tensione di controllo necessaria

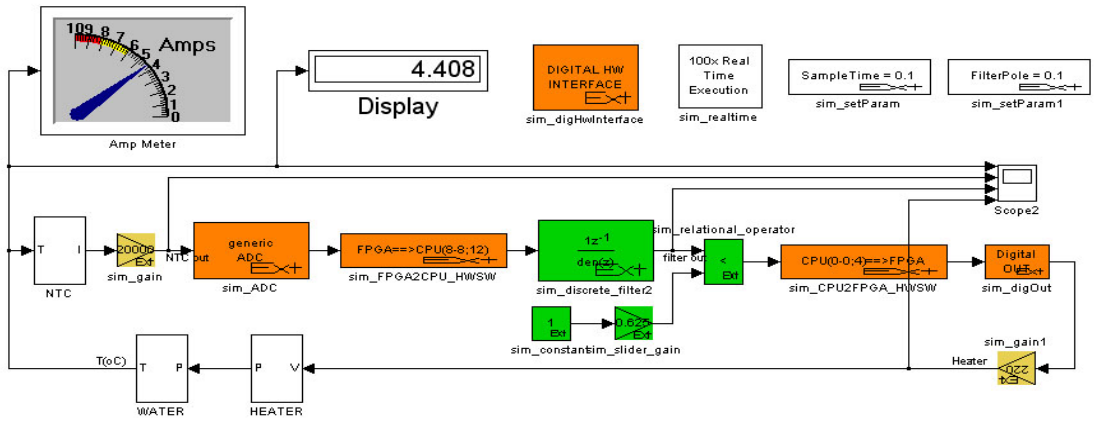

Figura 6.13. Modello CodeSimulink di un termostato

a riscaldare (quando il segnale è alto) o raffreddare (quando è a 0) l'acqua, rappresentata dal blocco *water*. Questo modello simula il comportamento reale del fluido al variare della temperatura, raffreddando l'acqua nel caso in cui il riscaldatore non sia acceso. Il software rappresentato dai blocchi "verdi" e dalle opportune interfacce, pilota l'alimentazione del riscaldatore per inseguire le variazioni di temperatura.

Il blocco *slider_ gain* rappresenta la temperatura a cui si vuole portare l'acqua, nel caso particolare:

- se la temperatura aumenta, al riscaldatore verrà dato un segnale alto di alimentazione per più tempo, finché non sarà stata raggiunta la temperatura desiderata. A quel punto il riscaldatore riceverà nuovamente il segnale periodico per inseguire soltanto il naturale raffreddamento dell'acqua.

- se la temperatura diminuisce, per raffreddare l'acqua sarà sufficiente non alimentare il riscaldatore finché il fluido non avrà raggiunto la temperatura desiderata. A quel punto l'alimentazione tornerà a impulsi periodici per mantenere mantenere la temperatura raggiunta.

In figura 6.14 si possono notare l'andamento dei seguenti segnali:

T(oC) : la temperatura dell'acqua;

NTC out : l'uscita dell'NTC;

filter out : l'uscita del filtro;

Heater : la tensione di alimentazione del riscaldatore

il blocco *NTC* permette di inseguire le variazioni di temperatura dell'acqua, generando un segnale (poi amplificato) che verrà elaborato dal software per pilotare il blocco *heater*.

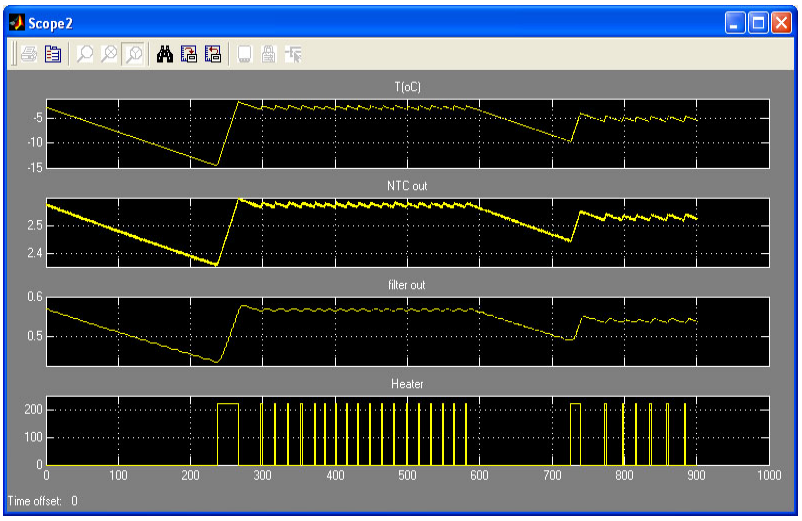

Figura 6.14. Segnali nel termostato

Se si effettua la scelta dei componenti per l'amplificatore da 20000, si noterà immediatamente che il numero di operazionali che rientrano nelle specifiche sono soltanto 3. Questo è dovuto all'alto valore di guadagno del blocco che rende accessibili soltanto gli operazionali a basso offset.

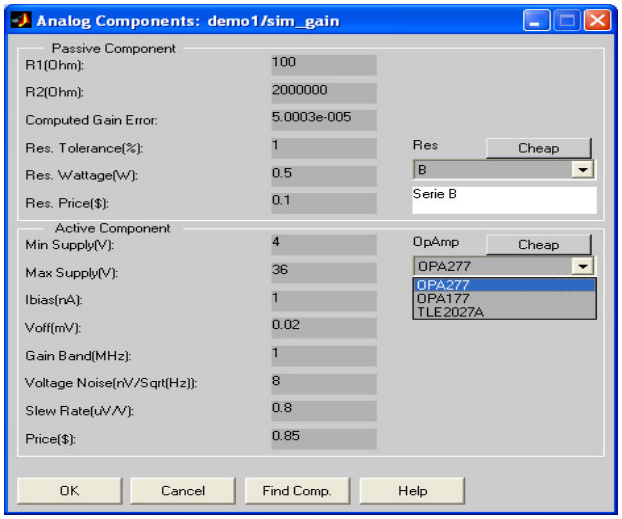

Figura 6.15.

Capitolo 7

Conclusioni

Con lo svolgimento del presente progetto di tesi sono state poste le basi per l'estensione del tool di CodeSimulink alla progettazione hardware analogica definendo, fra le altre cose:

- delle strategie nella scelta dei componenti analogici da assegnare ai blocchi estesi.
- le tecniche di estrazione dei record del database a seconda del tipo di componente cercato.
- una tecnica di realizzazione delle maschere che può essere facilmente estesa a tutti i circuiti implementabili.
- un compilatore in linguaggio di simulazione Spice.

Tutti questi elementi costituiscono un punto di partenza nella progettazione di un ambiente completo di coprogettazione. Tra i possibili sviluppi del progetto nell'ambito hardware analogico, sono inclusi:

- l'estensione analogica per i blocchi di CodeSimulink non considerati nella presente trattazione.
- l'aggiunta di ulteriori circuiti equivalenti ai blocchi già trattati che considerino tutti i possibili comportamenti dei blocchi estesi di CodeSimulink.
- la definizione di un compilatore hardware analogico che permetta, oltre alla simulazione in Spice, di sintetizzare il diagramma CodeSimulink in un circuito integrato (PCB).

Bibliografia

[1] Analog Devices, Inc. *Operational Amplifiers (Op Amps)* (consultato il 07/08/2006) Disponibile all'indirizzo: *http://www.analog.com*

[2] Attia, John O. (2002) *Pspice and Matlab for Electronics: An Integrated Approach*, Boca Raton, CRC Press (VLSI Circuits Series).

[3] Attia, John O. (1999) *Electronics and circuit analysis using Matlab*, Boca Raton, CRC Press.

[4] Belin, Howard M. (1988) *Il manuale degli amplificatori operazionali*, Milano, Jackson.

[5] Dillon, William E. *Pspice Tutorials*, (Modificato il 08/07/2006; consultato il 08/08/2006). Disponibile all'indirizzo *http:77dave.uta.edu/dillon/pspice/index.html*

[6] Franco, Sergio (1992), *Amplificatori operazionali e circuiti integrati analogici : tecniche di progetto, applicazioni*, Milano, Hoepli.

[7] Google, *Google Groups: comp.soft-sys.matlab* (consultato il 05/08/2006). Disponibile all'indirizzo: *http://groups.google.it/group/comp.soft-sys.matlab?lnk=lr*

[8] Horrocks, David H. (1996), *Circuiti retroazionati e amplificatori operazionali*, Bresso, Jackson Libri.

[9] Intersil, *Operational Amplifier Noise Prediction (All OpAmps)* Application Note AN519.1-November 1996

[10] Marchand, P. & Holland, O.T. (2003) *Graphics and GUI with Matlab*, 3rd edition, Boca Raton, Chapman & Hall/CRC.

[11] National Semiconductors, Inc. *Selguide: Amplifier Product Selector Guide Software for Windows* (consultato il 07/08/2006). Disponibile all'indirizzo: *http://www.national.com/appinfo/amps/ amplifiers_ selection_ guide.html*

[12] National Semiconductors, Inc. *Specification and Architecture of Sample-And-Hold Amplifiers*, Application Note AN-775-Luglio 1992

[13] Ramakant, Gayakwad A (1990), *Amplificatori operazionali e circuiti integrati lineari*, Milano, Jackson

[14] Ruggiero, Luca, *Guida a Microsoft Access* (consultato il 07/08/2006) Disponibile all'indirizzo: *http://www.mrwebmaster.it/database/msaccess/guida_ msaccess/*

[15] Ruggiero, Luca, *Guida a SQL* (consultato il 07/08/2006) Disponibile all'indirizzo: *http://www.mrwebmaster.it/database/sql/guida_ sql/*

[16] Schwarz, Urs (us), *str2cell: a pedestrian cell creator* (consultato il 07/08/2006) Disponibile all'indirizzo: *http://www.mathworks.com/matlabcentral/*

[17] Texas Instruments, Inc. *Amplifiers and Linear: Operational Amplifiers* (consultato il 07/08/2006) Disponibile all'indirizzo *http://www.ti.com/*

[18] Texas Instruments, Inc. *Handbook of Operational Amplifier Applications*, Application Report SBOA092A-Ottobre 2002

[19] The MathWorks, *Inc. Documentation for MathWorks products, R2006a* , (consultato il 07/08/2006). Disponibile all'indirizzo: *http://www.mathworks.com/access/helpdesk /help/helpdesk.html*

[20] Torzo Giacomo (1991), *Capire e sperimentare gli amplificatori operazionali: un approccio sperimentale all' elettronica analogica con una introduzione ai circuiti integrati digitali e ai trasduttori di grandezze fisiche*, Bologna, Zanichelli.

[21] Wikipedia, *The Free Encyclopedia*, (consultato il 07/10/2006). Disponibile all'indirizzo: *http://www.wikipedia.org/*

Appendice A

I circuiti trattati

In questo capitolo sono riportate tutte le nozioni di base sui circuiti elettronici che utilizzano amplificatori operazionali. In particolar modo sono riportate le formule principali che hanno portato alla definizione degli algoritmi di ricerca dei componenti ottimi utilizzati nel paragrafo 4. La maggior parte delle nozioni riguardanti i gli amplificatori invertenti e non invertenti sono state ricavate da [4] e da [18] e sono state applicate agli altri circuiti che utilizzano operazionali. Per quanto riguarda l'amplificatore di Sample & Hold le informazioni sono state ricavate da [6] e [12].

A.1 Il modello dell'amplificatore operazionale

Tutti i circuiti che utilizzano gli amplificatori operazionali si riferiranno a questo modello del dispositivo che è sicuramente il più diffuso in letteratura proposto in figura A.1. La

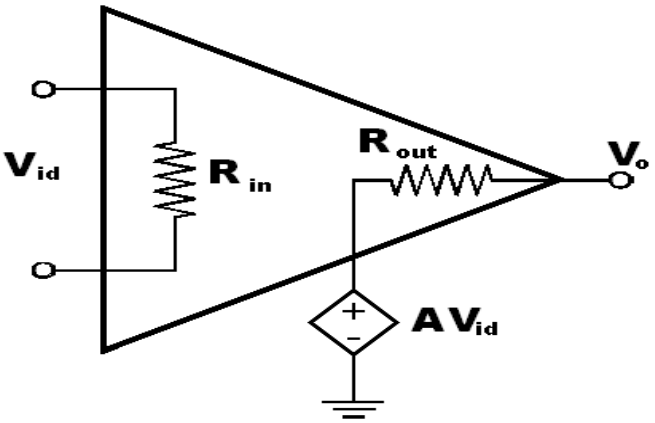

Figura A.1. Modello di amplificatore operazionale

tensione di uscita in questo amplificatore *non* reazionato è data dalla seguente formula

$$V_{out} = A \cdot V_{id} \tag{A.1}$$

dove A corrisponde al guadagno in tensione ad *anello aperto* mentre V_{id} è la differenza di tensione che si ha fra i morsetti invertente e non invertente. Nel modello A corrisponde al guadagno del generatore di tensione pilotato che riporta in uscita la tensione differenziale V_{id}. In un buon amplificatore il guadagno deve essere molto grande, tendente all'infinito (caso ideale) mentre la differenza di tensione fra i morsetti deve tendere a 0 (caso ideale). Le resistenze presenti nel modello sono la resistenza d'ingresso R_{in} e quella d'uscita R_{out}: la prima deve essere molto grande ed in genere il suo valore è considerato infinito, mentre la seconda è spesso trascurata in quanto molto piccola.

Prima di passare all'analisi dei circuiti con reazione che utilizzano il modello appena descritto è opportuno distinguere i termini di guadagno che verranno utilizzati nella descrizione:

$A = \frac{V_o}{V_{id}}$ è il guadagno ad anello aperto descritto sopra. Come già detto il suo valore tende all'infinito tanto più la tensione V_{id} fra i due morsetti tende a 0.

$A_f = \frac{V_o}{V_{in}}$ è il guadagno ad anello chiuso e verrà utilizzato in tutti i circuiti presentati di seguito. Rappresenta il guadagno vero e proprio del circuito di amplificazione che utilizza l'amplificatore operazionale.

$\beta = \frac{V_f}{V_o}$ è il guadagno del circuito di reazione ed è dato dal rapporto fra la tensione di reazione V_f e la tensione di uscita.

A.2 Le formule dei principali circuiti di amplificazione

Nei successivi paragrafi saranno riportate le formule di base dell'amplificatore invertente e di quello non invertente. Tutti gli altri circuiti che utilizzano amplificatori saranno descritti in maniera meno approfondita poiché le corrispondenti formule derivano da quelle illustrate nel paragrafo seguente.

A.2.1 L'amplificatore non invertente

Il circuito ha reazione negativa ed è costituito da un operazionale e due resistenze come si può vedere in figura A.2. Le equazioni di partenza sono le seguenti

$$V_o = A \cdot (V_1 - V_2) \tag{A.2}$$

$$A_f = \frac{V_o}{V_{in}} \tag{A.3}$$

Dalla figura si possono ricavare i valori delle tensioni ai morsetti: V_1 corrisponde al segnale in ingresso V_{in} mentre V_2 è la tensione sulla resistenza R_1 che si calcola come la partizione

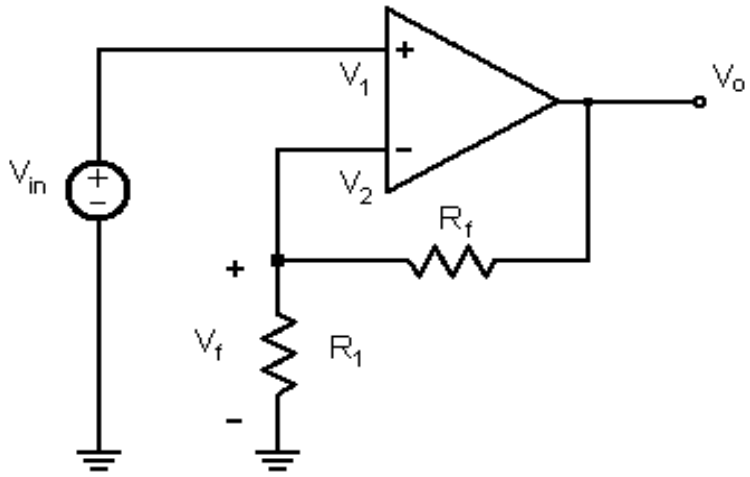

Figura A.2. Amplificatore Non Invertente

della tensione d'uscita sulla resistenza a patto che la resistenza d'ingresso R_i sia molto maggiore di R_1. Si ha dunque

$$V_1 = V_1 \tag{A.4}$$

$$V_2 = V_f = \frac{R_1}{R_1 + R_f} V_o \tag{A.5}$$

Sostituendo le tensioni trovate nell'equazione nella A.2 si ottiene

$$V_o = \frac{A \cdot (R_1 + R_f)}{R_1 + R_f + A \cdot R_1} V_{in} \tag{A.6}$$

A questo punto si ottiene il guadagno ad anello chiuso A_f

$$A_f = \frac{V_o}{V_{in}} = \frac{A \cdot (R_1 + R_f)}{R_1 + R_f + A \cdot R_1} \tag{A.7}$$

Se il guadagno ad anello aperto A è molto grande si può ipotizzare che

$$A \cdot R_1 \gg (R_1 + R_f) \Rightarrow (A \cdot R_1 + R_1 + R_f) \simeq A \cdot R_1$$

Si può quindi dire che

$$A_f = 1 + \frac{R_f}{R_1} \tag{A.8}$$

Questa formula è la più importante in assoluto dell'amplificatore non invertente poiché corrisponde al guadagno del circuito di amplificazione ed è necessario per determinare i componenti passivi che compongono il circuito a partire da un guadagno dato.

Infine si può esprimere il guadagno ad anello chiuso A_f della formula A.7 in termini del guadagno ad anello aperto raccogliendo $R_1 + R_f$. In questo modo si ottiene

$$A_f = \frac{A\left(\frac{R_1+R_f}{R_1+R_f}\right)}{\frac{R_1+R_f}{R_1+R_f} + \frac{A \cdot R_1}{R_1+R_f}} \Rightarrow A_f = \frac{A}{1 + A \cdot \beta} \tag{A.9}$$

Sapendo che il guadagno di reazione β si ricava dalla formula A.5 come

$$\beta = \frac{V_f}{V_o} = \frac{R_1}{R_1 + R_f} \tag{A.10}$$

A.2.2 L'amplificatore invertente

Anche questo circuito ha reazione negativa ed è costituito da un operazionale e due resistenze come si può osservare in figura A.3. Diversamente dal caso dell'amplificatore non

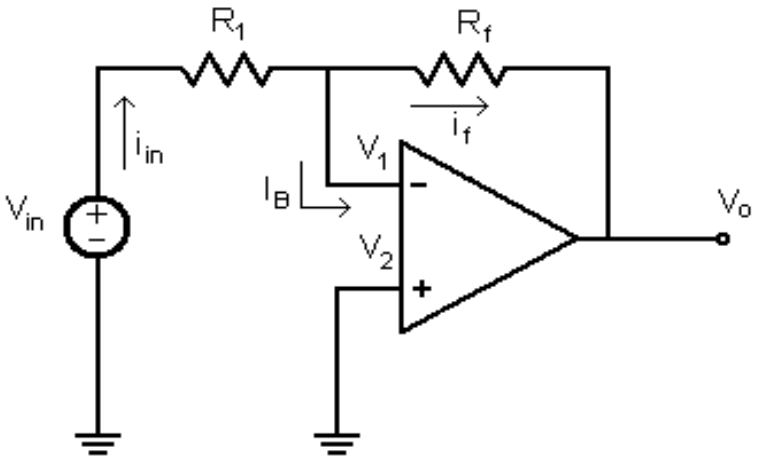

Figura A.3. Amplificatore Invertente

invertente, la reazione è costituita dalla sola resistenza R_f. L'equazione di partenza è, come in precedenza

$$V_o = A \cdot (V_1 - V_2) \tag{A.11}$$

$$A_f = \frac{V_o}{V_{in}} \tag{A.12}$$

In questo caso il guadagno ad anello chiuso A_f non si può calcolare immediatamente come il rapporto fra le tensioni d'ingresso e d'uscita, è necessario passare attraverso le equazioni di Kirchoff in corrente

$$i_{in} = i_f + I_B \Rightarrow i_{in} \simeq i_f \tag{A.13}$$

dove I_B è la corrente di polarizzazione dell'operazionale ed è considerata trascurabile in entrambi i morsetti. Sapendo che la tensione al nodo invertente si chiama V_2 si ottiene

$$\frac{V_{in} - V_2}{R_1} = \frac{V_2 - V_o}{R_f} \tag{A.14}$$

A partire dalla A.11, sapendo che $V_1 = 0$ si ottiene

$$V_2 = -\frac{V_0}{A}$$

che sostituito nella A.14 porta a

$$\frac{V_{in} + V_o/A}{R_1} = \frac{-V_o/A - V_o}{R_f} \tag{A.15}$$

Si ha quindi che

$$V_{in} = -\frac{R_1}{R_f}\left(1 + \frac{1}{A}\right)V_o - \frac{V_o}{A} \tag{A.16}$$

Da cui si può ricavare il guadagno ad anello chiuso A_f

$$A_f = \frac{V_o}{V_{in}} = -\frac{A \cdot R_f}{A \cdot R_{R1} + R_1 + R_f} \tag{A.17}$$

come per il caso precedente supponendo che $A \cdot R_1 \gg R_1 + R_f$ si ottiene quindi

$$A_f = \frac{V_o}{V_{in}} = -\frac{R_f}{R_1} \tag{A.18}$$

Analogamente al caso dell'amplificatore non invertente, questa formula è quella che meglio rappresenta il circuito di amplificazione. Naturalmente la tensione di uscita avrà segno opposto rispetto a quella d'ingresso. Infine,si esprime il guadagno ad anello chiuso in funzione di quello ad anello aperto a partire da A.17 raccogliendo $R_1 + R_f$ a numeratore e a denominatore. Si ottiene la seguente equazione:

$$A_f = -\frac{\frac{A \cdot R_f}{R_1 + R_f}}{\frac{A \cdot R_1 + R_1 + R_f}{R_1 + R_f}} = -\frac{\frac{A \cdot R_f}{R_1 + R_f}}{\frac{A \cdot R_1}{R_1 + R_f} + 1} \tag{A.19}$$

Raccolgo i termini

$$K = \frac{R_f}{R_1 + R_f}$$

e

$$\beta = \frac{R_1}{R_1 + R_f}$$

e analogamente al caso precedente ottengo una formula che esprime A_f in funzione di A

$$A_f = -\frac{K \cdot A}{1 + \beta \cdot A} \tag{A.20}$$

A.3 L'offset dell'amplificatore invertente e non invertente

L'offset è quella tensione di errore che si manifesta in uscita quando la tensione d'ingresso è nulla. Negli operazionali reali è possibile identificare tre grandezze che danno origine all'offset di uscita in tensione nei circuiti:

Tensione d'offset d'ingresso : V_{ios} è la tensione differenziale che è presente fra i due terminali d'ingresso dell'operazionale senza che venga applicato nessun segnale d'ingresso esterno.

Corrente di polarizzazione d'ingresso (Bias) : I_B è definita come la media delle due correnti di polarizzazione d'ingresso I_{B1} e I_{B2} entranti rispettivamente nel morsetto non invertente e non invertente dell'operazionale. Si manifestano quando i terminali d'ingresso sono posti a massa e quindi non viene applicata alcuna tensione d'ingresso all'operazionale. Durante la trattazione si ipotizza che le due correnti siano uguali per motivi spiegati di seguito. Si ha quindi

$$I_B = \frac{I_{B1} + I_{B2}}{2} = I_{B1} = I_{B2}$$

È una corrente continua specificata nei data sheet dei dispositivi.

Correnti di offset d'ingresso : I_{io} indica il grado si disadattamento fra le correnti di polarizzazione ed è generalmente calcolata come la differenza fra le due correnti. Generalmente il suo valore è molto inferiore alle correnti di polarizzazione e, anche se il suo valore viene riportato fra i data sheet dell'operazionale si è dunque preferito non trattarla di seguito applicando l'ipotesi che le correnti di polarizzazione siano uguali, si ha quindi

$$I_{io} = |I_{B1} - I_{B2}| \simeq 0$$

L'offset di tensione globale che si manifesta in uscita è calcolabile utilizzando sovrapposizione degli effetti. A questo punto vengono trattati singolarmente i contributi di offset in uscita causati dalla tensione di offset d'ingresso e dalla corrente di polarizzazione. In entrambi i casi, per calcolarli, e necessario cortocircuitare la tensione d'ingresso V_{in} valutare singolarmente la propagazione in uscita dei due effetti di offset. Il circuito risultante è equivalente sia per l'amplificatore invertente che per quello non invertente.

A.3.1 Il contributo della tensione d'offset d'ingresso

Si parte dal circuito in figura A.4. Come già detto, si suppone $V_{in} = 0$ e, per sovrapposizione degli effetti, che le correnti di polarizzazione siano nulle, si ha quindi che la tensione V_2 al terminale invertente vale

$$V_2 = \frac{R_1}{R_1 + R_f} V_{oos} \tag{A.21}$$

dove V_{oos} corrisponde proprio al contributo in uscita che si sta cercando di trovare. Esplicitando si ha dunque

$$V_{oos} = \frac{R_1 + R_f}{R_1} V_2 \tag{A.22}$$

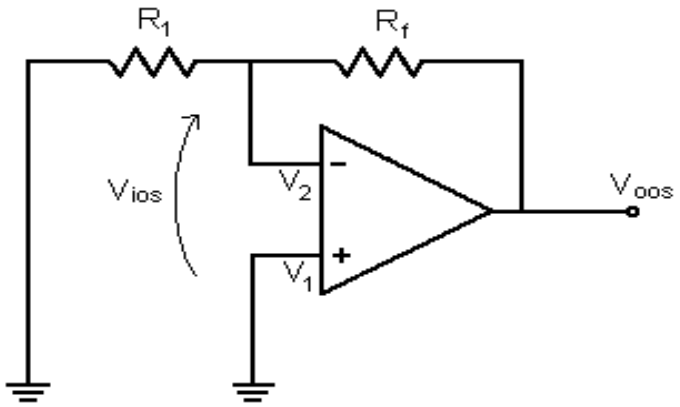

Figura A.4. Contributo di offset della tensione di offset d'ingresso

Per definizione V_{ios} è la tensione fra i morsetti invertente e non invertente, si ha dunque

$$V_{ios} = |V_2 - V_1| = V_2 \tag{A.23}$$

poiché il morsetto non invertente è collegato al riferimento. Il valore assoluto si prende poiché non si conosce il verso della tensione di offset d'ingresso. A questo punto sostituendo V_2 nella A.22 si ottiene

$$V_{oos} = \frac{R_1 + R_f}{R_1} V_{ios} = A_f \cdot V_{ios} \tag{A.24}$$

A.3.2 Il contributo della corrente di polarizzazione

Si riporta in figura A.5 il circuito analizzato con le tensioni d'ingresso e di offset d'ingresso nulle. Diversamente dai casi precedenti è necessario considerare il modello dell'operazionale esposto nel paragrafo A.1 e utilizzare le equazioni di Kirchoff in corrente. Avendo imposto $V_{ios} = 0$ e non essendoci idealmente tensione fra i due terminali la tensione di pilotaggio del generatore vale

$$A \cdot V_{id} = A \cdot V_{ios} = 0 \tag{A.25}$$

quindi le resistenze R_f e R_1 sono in parallelo e la tensione V_2 si può calcolare come

$$V_2 = (R_1 \parallel R_f) I_{B2} = \frac{R_1 \cdot R_f}{R_1 + R_f} I_{B2} \tag{A.26}$$

Scrivendo l'equazione di Kirchoff al nodo invertente si ha

$$I_{B2} = I_1 + I_2 \Rightarrow \frac{V_2}{R_i} = -\frac{V_2}{R_1} + \frac{V_{oIb} - V_2}{R_f} \tag{A.27}$$

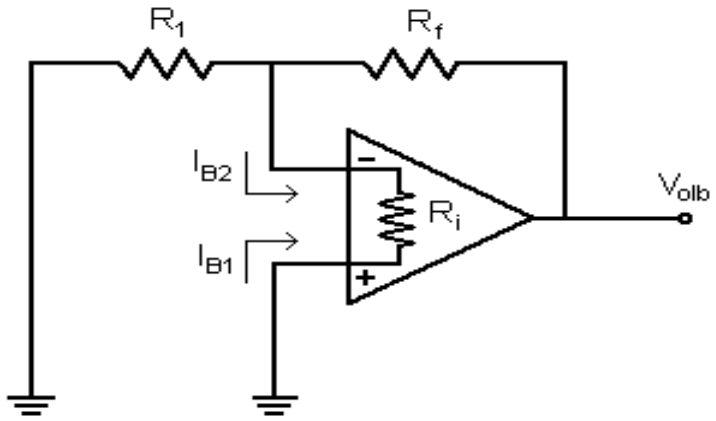

Figura A.5. Contributo di offset della corrente di polarizzazione

Dove V_{oIb} è il contributo in uscita della corrente di polarizzazione d'ingresso che si sta cercando. Esplicitando rispetto ad esso si ottiene

$$V_{oIb} = R_f \cdot V_2 \left(\frac{1}{R_1} + \frac{1}{R_f} + \frac{1}{R_i} \right) \tag{A.28}$$

Essendo R_i molto grande

$$\frac{1}{R_i} \simeq 0$$

quindi si ha

$$V_{oIb} = R_f \cdot V_2 \left(\frac{R_1 + R_f}{R_1 \cdot R_f} \right) = V_2 \left(\frac{R_1 + R_f}{R_1} \right) \tag{A.29}$$

Sostituendo la A.26 nell'ultima equazione si ottiene

$$V_{oIb} = \frac{R_1 \cdot R_f}{R_1 + R_f} \left(\frac{R_1 + R_f}{R_1} \right) I_{B2} = R_f \cdot I_{B2} \tag{A.30}$$

Diversamente dal contributo di offset descritto nel paragrafo A.3.1, è possibile compensare l'effetto delle correnti di polarizzazioni aggiungendo fra il morsetto non invertente e il riferimento una resistenza uguale al parallelo di tutte le resistenze entranti nel morsetto invertente.

A.4 Il rumore dell'amplificatore invertente e non invertente

Il rumore corrisponde al disturbo casuale che si manifesta sul segnale di uscita. È dovuto a varie cause ed è spesso difficile da prevedere e analizzare. Prima di procedere con l'identificazione delle fonti di rumore nei circuiti di amplificazioni è conveniente fornire una breve descrizione sulle diverse notazioni adottate per descrivere il rumore.

A.4.1 Le definizioni di rumore

Il rumore elettronico può essere indicato in molti modi diversi che cambiano per significato e unità di misura. Di seguito è proposto una breve descrizione delle principali definizioni di rumore utilizzabili per descrivere un resistore.

La tensione di rumore (*noise voltage*)

Il rumore di una resistenza viene rappresentato come una **tensione di rumore** seguita da una resistenza. Il suo valore è predicibile a patto di conoscere:

- il valore della resistenza
- la banda del circuito
- la temperatura di lavoro

Tutti i termini sono conosciuti nei circuiti trattati, infatti

- la resistenza è calcolata tramite un determinato algoritmo;
- la banda del circuito è calcolata a partire dalla banda di lavoro del dispositivo;
- la temperatura è fissata a 300°K (27°C) che corrisponde alla tipica condizione di lavoro del dispositivo;

Di conseguenza il valore del rumore espresso in valore efficace (rms[1]) è dato da

$$En = \sqrt{4K \cdot T \cdot R \cdot \Delta f} \quad (V_{rms}) \tag{A.31}$$

Dove K è la costante di Boltzmann e vale circa $1{,}38^{-23}\frac{J}{K}$. Questo rumore è detto *termico* perché descrive il movimento casuale degli elettroni eccitati termicamente.

La potenza di rumore(*noise power*)

La potenza di rumore corrisponde semplicemente al quadrato della tensione di rumore, quindi

$$Pn = En^2 = 4K \cdot R \cdot T \cdot \Delta f \quad (V^2) \tag{A.32}$$

Questo tipo di definizione del rumore è utilizzata perché rende possibile sommare linearmente più contributi di rumore espressi appunto come potenze di rumore, cosa impossibile da fare con i valori efficaci.

[1]Root Mean Square ovvero la radice quadrata della somma dei quadrati dei campioni acquisiti

La densità spettrale di potenza (*power spectral density*)

Per osservare il comportamento del rumore nel dominio delle frequenze, si calcola la potenza di rumore in una banda di 1 Hz e si ottiene

$$Sn = \frac{Pn}{\Delta f} = \frac{En^2}{\Delta f} = 4K \cdot T \cdot R \quad (V^2/Hz) \tag{A.33}$$

Quello che si ottiene viene chiamata *densità spettrale di potenza* ed è costante a tutte le frequenze per il caso analizzato.

La densità spettrale di tensione (*voltage spectral density*)

Come si può intuire, la *densità spettrale di tensione* è la radice quadrata della densità spettrale di potenza

$$Nn = \sqrt{Sn} = \frac{En}{\sqrt{\Delta f}} = \sqrt{4K \cdot T \cdot R} \quad (V/\sqrt{Hz}) \tag{A.34}$$

In molti dispositivi elettronici, tra cui gli amplificatori operazionali, il rumore è espresso in questa forma.

A.4.2 I contributi di rumore nei circuiti di amplificazione

Per poter calcolare il rumore che affligge il segnale di uscita bisogna conoscere:

- il modello di rumore del circuito;
- la notazione che si adotta per il rumore;

I *modelli di rumore* permettono di rappresentare il rumore con dei generatori simili a quelli già visti nel calcolo degli offset nel paragrafo A.3. Il contributo totale sarà ottenuto con la sovrapposizione degli effetti di tutti i generatori presi singolarmente. Diversamente dal caso dell'offset però i calcoli dovranno essere eseguiti sulle densità spettrali di potenza e eventuali moltiplicazioni per resistenze o guadagni si trasformeranno in prodotti per i loro moduli al quadrato; ad esempi se si ha una corrente I_n^2 che attraversa una resistenza R, la densità spettrale di tensione sarà data da

$$V_n^2 = I_n^2 \cdot |R|^2 = I_n^2 \cdot R^2$$

Eventuali correzioni sulla notazione del rumore (come il passaggio da densità spettrale di potenza a tensione di rumore) verranno eseguite sul contributo globale in uscita. Di seguito sono riportati i principali modelli di rumore dei componenti elettronici che compongono i circuiti di amplificazione.

Rumore dell'operazionale: per modellare il rumore si utilizzano tre generatori considerati scorrelati fra di loro sebbene derivino da sorgenti interne e talvolta comuni dell'operazionale. Il rumore è espresso come densità spettrale e i valori dipendono dall'amplificatore e dalla tecnologia utilizzata per realizzarlo. Il rumore di corrente

riportato nei data sheet degli operazionali è inferiore al suo corrispondente in tensione, per questo motivo si è preferito non includere nel modello i generatori di corrente, ma solo quello di tensione. L'unità di misura di questo rumore nel data sheet è $\frac{nV}{\sqrt{Hz}}$.

Rumore delle resistenze : la maggior sorgente di rumore nei resistori è il rumore termico, che dipende dall'agitazione termica degli elettroni. Esso non dipende dalla corrente che passa nella resistenza, ma solo dalla temperatura del resistore. È un tipico rumore bianco. Si possono avere due modelli di rumore per la resistenza:

- il modello di Thevenin include la resistenza in serie al generatore V_n avente densità spettrale (di potenza)

$$V_n^2 = 4K \cdot T \cdot R \tag{A.35}$$

  Dove K è la costante di Boltzmann e T è la temperatura di lavoro.

- il modello di Norton è costituito da un generatore di corrente di rumore I_n in parallelo alla conduttanza $G = 1/R$ con densità spettrale (di potenza) generatore V_n avente densità spettrale (di potenza)

$$I_n^2 = \frac{V_n^2}{R^2} = \frac{4K \cdot T}{R} \tag{A.36}$$

In figura A.6 sono riportati i due modelli di rumore della resistenza: a sinistra quello Thevenin e a desctra quello Norton.

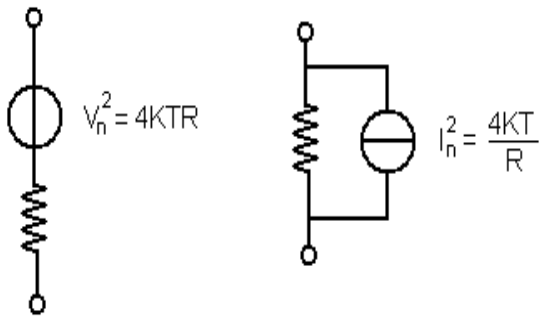

Figura A.6. Modelli di rumore del resistore

A questo punto è possibile definire il modello di rumore per i circuiti dell'amplificatore invertente e non invertente. Si ricorda che, come accadeva per l'offset, è necessario cortocircuitare al riferimento il generatore del segnale d'ingresso: anche in questo caso i modelli di rumore dei due circuiti sono identici e si possono rappresentare nel seguente modo Essendo il rumore casuale i generatori di corrente e di tensione non è possibile stabilire a priori il verso delle tensioni delle correnti. Mettendosi nell'ipotesi di determinare il rumore nel caso peggiore i segni saranno attribuiti in modo tale che i vari contributi si sommino. Di seguito sono analizzati gli effetti dei singoli generatori.

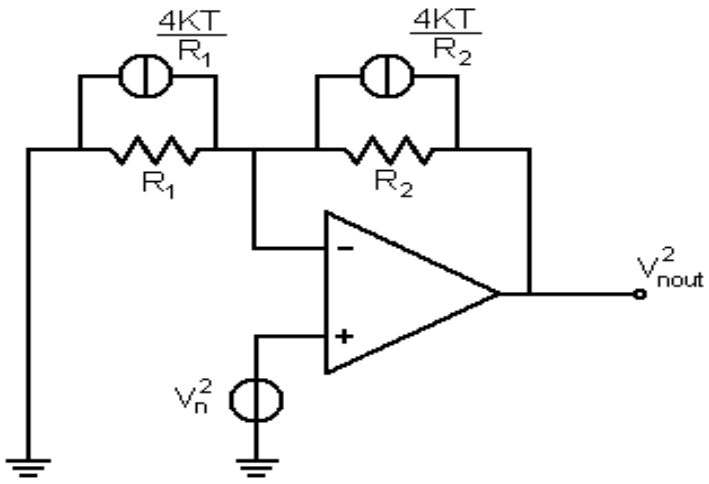

Figura A.7. Modelli degli amplificatori invertente e non invertente

A.4.3 Il contributo di rumore dell'operazionale

Non considerando i generatori dovuti alle resistenze R_1 e R_2 si analizza il la propagazione del segnale V_n^2 in uscita. Analizzando il circuito si nota che il circuito corrisponde ad un amplificatore non invertente, di conseguenza la tensione di rumore in uscita sarà pari a

$$V_{nout}^2 = V_n^2 \cdot \left|1 + \frac{R_2}{R_1}\right|^2 = V_n^2 \cdot \left(1 + \frac{R_2}{R_1}\right)^2 \tag{A.37}$$

Ricordandosi che i generatori sono densità spettrali di potenza.

A.4.4 Il contributo di rumore delle resistenze

In modo analogo al caso precedente si trascura il rumore dell'amplificatore e si considera solo l'effetto dei generatori di corrente associati alle resistenze. La somma delle correnti attraversa la resistenza di reazione R_2 in questo modo si ha

$$V_{nout}^2 = \left(\frac{4K \cdot T}{R_1} + \frac{4K \cdot T}{R_2}\right) \cdot |R_2|^2 = 4K \cdot T \cdot \left(\frac{1}{R_1} + \frac{1}{R_2}\right) \cdot R_2^2 \tag{A.38}$$

La densità di rumore globale sarà data dalla somma di questo contributo più il precedente indicato nella A.37, si avrà quindi

$$V_{nout}^2 = V_n^2 \cdot \left(1 + \frac{R_2}{R_1}\right)^2 + 4K \cdot T \cdot \left(\frac{1}{R_1} + \frac{1}{R_2}\right) \cdot R_2^2 \tag{A.39}$$

A.5 La dissipazione di potenza negli amplificatori invertente e non invertente

La stima sulla potenza dissipata su questi circuiti di amplificazione viene effettuata solo sulle due resistenze presenti. È necessario mettersi nelle condizioni peggiori di funzionamento del circuito, quando la tensione di uscita raggiunge la sua massima dinamica indicata dal parametro analogico *Upper Saturation*. Conoscendo la tensione massima di uscita ed il guadagno in tensione dei circuiti di amplificazione è possibile determinare la potenza assorbita da R_1 ed R_2 nei due circuiti. In una resistenza la potenza dissipata si calcola con la seguente formula

$$P = V \cdot I = I^2 \cdot R = \frac{V^2}{R} \tag{A.40}$$

A questo punto, a seconda del circuito, è possibile determinare la potenza dissipata sulle due resistenze.

A.5.1 La potenza assorbita nell'amplificatore non invertente

Si considera il circuito in figura A.8. Sapendo che la tensione massima in uscita vale V_{out}

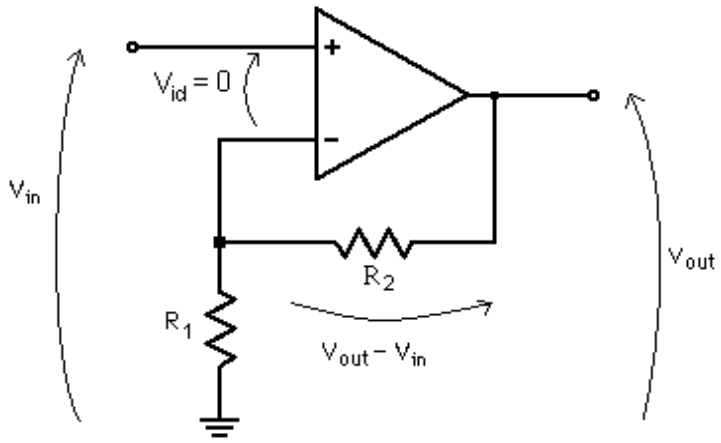

Figura A.8. Dissipazione di potenza nell'Amplificatore Non Invertente

e conoscendo il guadagno ad anello chiuso A_f si ha che

$$V_{in} = \frac{V_{out}}{A_f} \tag{A.41}$$

A questo punto si può facilmente calcolare la potenza dissipata su R_1 e R_f, infatti

- sulla resistenza R_1 cade la tensione V_{in} poiché la tensione fra i morsetti invertente e non invertente è idealmente nulla, si ha quindi che

$$P_{R1} = \frac{V_{in}^2}{R_1} \tag{A.42}$$

- la tensione sulla resistenza R_2 è data dalla differenza fra la tensione d'uscita e la tensione d'ingresso, si ha quindi

$$P_{R2} = \frac{(V_{out} - V_{in})^2}{R_2} \tag{A.43}$$

A.5.2 La potenza assorbita nell'amplificatore invertente

Analogamente al caso dell'amplificatore non invertente, è possibile calcolare la tensione d'ingresso a partire da quella di uscita considerata massima. Il circuito dell'amplificatore invertente è riportato in figura A.9. Una volta conosciute V_{in} e V_{out} si può risalire alle

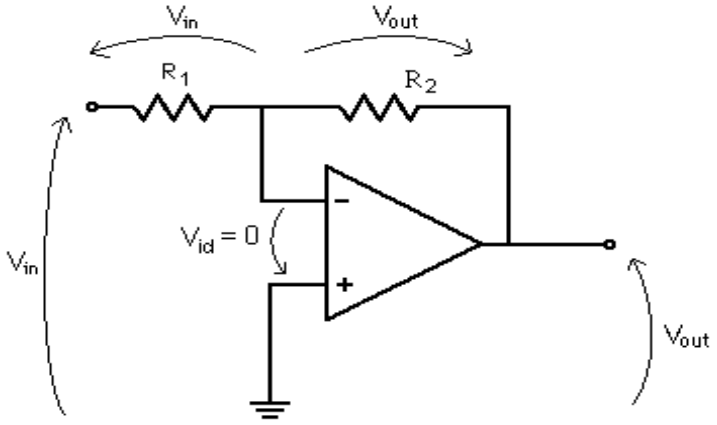

Figura A.9. Dissipazione di potenza nell'Amplificatore Invertente

potenze dissipate sulle due resistenze. Si ha che

- sulla resistenza R_1 cade la tensione V_{in} poiché, come in precedenza, la tensione fra i morsetti invertente e non invertente è idealmente nulla, si ha quindi che

$$P_{R1} = \frac{V_{in}^2}{R_1} \tag{A.44}$$

- sulla resistenza R_2 cade solo la tensione d'uscita in quanto quella fra i morsetti invertente e non invertente è idealmente nulla, si ha quindi

$$P_{R2} = \frac{V_{out}^2}{R_2} \tag{A.45}$$

A.6 L'amplificatore sommatore e l'amplificatore differenziale

In questa sezione verranno trattati i circuiti che forniscono la descrizione analogica del blocco *sim_sum2* e verranno ricavate le formule principali con gli stessi criteri utilizzati per gli amplificatori invertente e non invertente.

A.6.1 Il sommatore non invertente

Il circuito, proposto in figura A.10, permette di sommare le tensioni di due segnali entranti nel morsetto non invertente. Come si può osservare in figura, il circuito è simile a quello

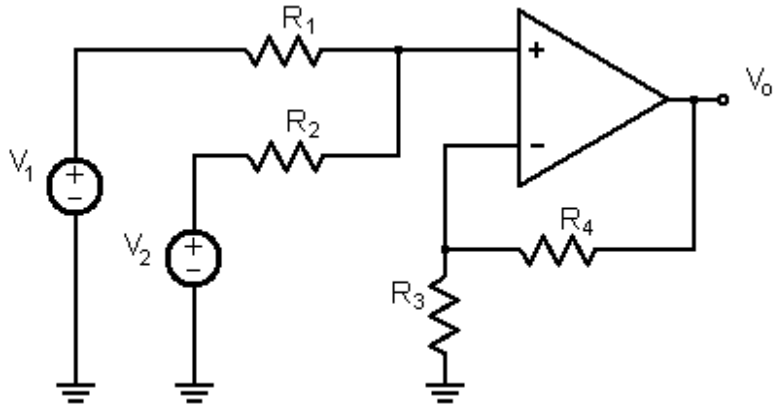

Figura A.10. Sommatore Non Invertente

dell'amplificatore non invertente. Con sovrapposizione dei circuiti si ottengono in uscita i contributi dei due generatori d'ingresso.

- se $V_2 = 0$ la V_1 viene ripartita tra le resistenze R_2 e R_4. All'ingresso del terminale non invertente si ha
$$V_+ = \frac{R_2}{R_1 + R_2} V_1 \tag{A.46}$$
Questa tensione viene riportata in uscita moltiplicata per il guadagno ad anello chiuso che vale
$$A_f = 1 + \frac{R_4}{R_3}$$
Di conseguenza il primo contributo di tensione in uscita varrà
$$V_{out1} = \left(1 + \frac{R_4}{R_3}\right) \cdot \frac{R_2}{R_1 + R_2} V_1 \tag{A.47}$$

- se $V_1 = 0$ la V_2 viene nuovamente ripartita fra le resistenze R_2 e R_4, ma questa volta si avrà
$$V_+ = \frac{R_1}{R_1 + R_2} V_2 \tag{A.48}$$
Poiché il guadagno ad anello chiuso è lo stesso si avrà in uscita
$$V_{out2} = \left(1 + \frac{R_4}{R_3}\right) \cdot \frac{R_1}{R_1 + R_2} V_2 \tag{A.49}$$

La formula finale che tiene conto dei due effetti sarà dunque

$$V_{out} = V_{out1} + V_{out2} = \left(1 + \frac{R_4}{R_3}\right) \cdot \frac{R_2}{R_1 + R_2} V_1 + \left(1 + \frac{R_4}{R_3}\right) \cdot \frac{R_1}{R_1 + R_2} V_2 \tag{A.50}$$

Se $R_1 = R_2 = R_3 = R_4$ l'equazione diventa

$$V_{out} = (1 + 1) \cdot \frac{1}{2} V_1 + (1 + 1) \cdot \frac{1}{2} V_2 = V_1 + V_2 \tag{A.51}$$

A questo punto è possibile trovare l'offset ed il rumore per questo circuito.

A.6.2 L'offset del sommatore non invertente

Come per l'amplificatore non invertente per determinare la tensione di offset globale in uscita si utilizza sovrapposizione degli effetti sui due generatori di offset. Il circuito in

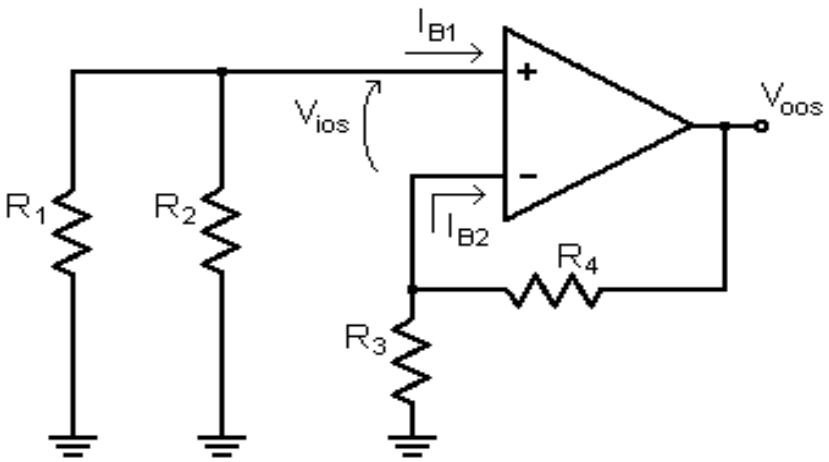

Figura A.11. Contributi di offset nel Sommatore Non Invertente

figura A.11 verrà anche utilizzato per determinare l'offset dell'amplificatore differenziale, come è possibile osservare nel paragrafo A.6.6: i generatori delle tensioni V_1 e V_2 sono spenti e le uniche sorgenti di segnale sono:

- la tensione d'offset d'ingresso V_{ios1} tra i due terminali dell'operazionale si propaga in uscita moltiplicata per il guadagno ad anello, si ha quindi

  $$V_{oos1} = \left(1 + \frac{R_4}{R_3}\right) V_{ios1} \tag{A.52}$$

  Nel morsetto non invertente scorre una corrente nulla quindi anche la tensione che cade sul parallelo di R_1 e R_2 sarà 0 e in uscita sarà riportata solo V_{ios}.

- la corrente di Bias in ingresso non fornirà in uscita alcun contributo: il circuito è automaticamente compensato grazie alla presenza del parallelo di R_1 e R_2 sul terminale non invertente che è esattamente uguale al parallelo fra R_3 e R_4, infatti

$$V_{oos2} = R_2 \cdot I_{B1} - R_2 \cdot I_{B2} = 0 \tag{A.53}$$

La tensione di offset globale varrà quindi

$$V_{oos} = V_{oos1} + V_{oos2} = \left(1 + \frac{R_4}{R_3}\right) V_{ios1} \tag{A.54}$$

A.6.3 Il rumore del sommatore non invertente

In figura A.12 è riportato il modello di rumore di questo circuito Il circuito è lo stesso

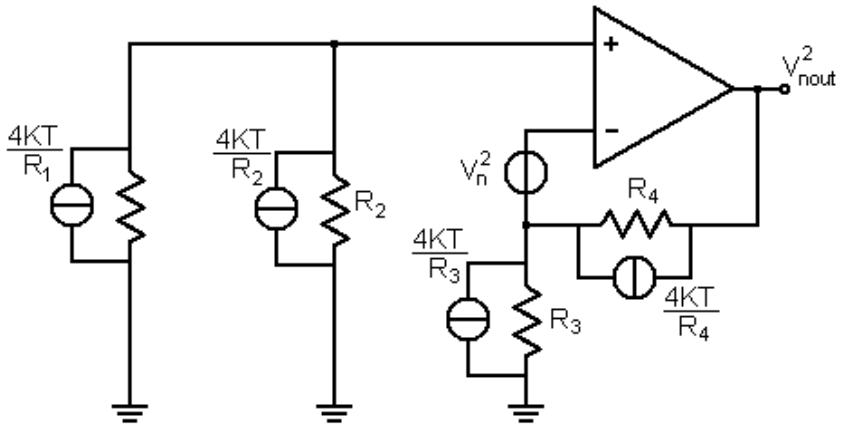

Figura A.12. Modello di rumore del Sommatore Non Invertente

utilizzato anche nel caso dell'amplificatore differenziale spiegato nel paragrafo A.6.7. Il rumore globale in uscita si ottiene con la sovrapposizione degli effetti sui generatori presenti nel modello. Invece di prendere singolarmente gli effetti di ogni generatore del modello si considerano la combinazione dei generatori ai terminali invertente e non invertente:

- nel terminale non invertente sono presenti i generatori di tensione dovuti alle resistenze R_1 e R_2 più il generatore di tensione di rumore dell'amplificatore operazionale; combinandoli e riportando in uscita il segnale globale si ha:

$$V^2_{nout1} = \left|1 + \frac{R_4}{R_3}\right|^2 \cdot \left(4K \cdot T \cdot R_1 + 4K \cdot T \cdot R_2 + V^2_{nin}\right) \tag{A.55}$$

- nel terminale invertente entrano i contributi di rumore delle resistenze R_3 e R_4. Per semplicità conviene considerare il modello a corrente di rumore dei resistori. In questo caso si avrà

$$V^2_{nout2} = |R_4|^2 \cdot \left(\frac{4K \cdot T}{R_3} + \frac{4K \cdot T}{R_4}\right) \tag{A.56}$$

Tenendo conto che le resistenze sono identiche si sommano i due termini di rumore e si ottiene

$$V_{nout}^2 = V_{nout1}^2 + V_{nout2}^2 = 4 \cdot (8K \cdot T \cdot R + V_{nin}^2) + 8K \cdot T \cdot R \tag{A.57}$$

A.6.4 La dissipazione nel sommatore non invertente

Poiché in questo circuito tutte le resistenze hanno il medesimo valore R per sapere su quale resistenza viene dissipata più potenza è sufficiente trovare la resistenza su cui cade la tensione maggiore, si sa infatti che

$$P = \frac{V^2}{R}$$

Diversamente dal caso degli amplificatori invertente e non invertente, trattati nel paragrafo A.5, non è possibile risalire alle tensioni d'ingresso V_1 e V_2 a partire dalla tensione d'uscita V_{out}. Ci si mette allora nella condizione in cui uno degli ingressi sia nullo e l'altro abbia invece dinamica massima, si avrà dunque

$$V_{out} = V_1 + V_2 = 0 + V_2 = V_2 \tag{A.58}$$

Anche invertendo le tensioni d'ingresso si otterrà il circuito equivalente in figura A.13. in

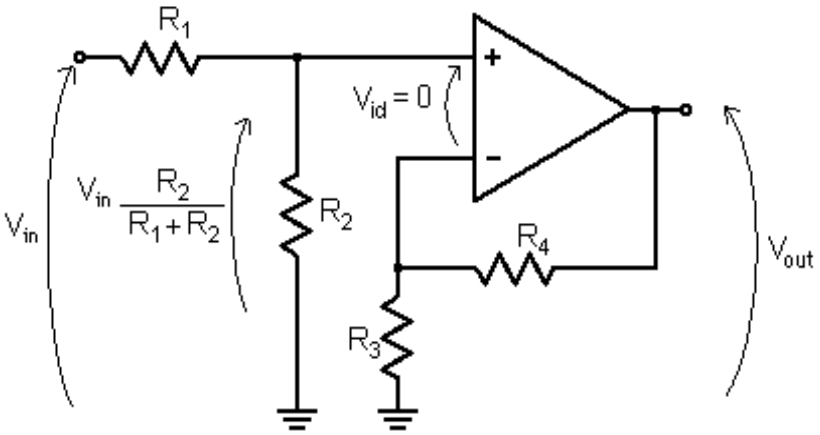

Figura A.13. Dissipazione di potenza nel Sommatore Non Invertente

cui la tensione che cade su ognuna delle resistenze presenti vale

$$V_R = \frac{R}{R+R} V_2 = \frac{V_2}{2} \tag{A.59}$$

Di conseguenza in questo caso che è sicuramente il peggiore, la potenza dissipata su ogni resistenza vale

$$P_R = \frac{V_R^2}{R} = \frac{1}{R}\left(\frac{V_2}{2}\right)^2 \tag{A.60}$$

A.6.5 L'amplificatore differenziale

Questo circuito, riportato in figura A.14 permette di sottrarre due segnali in tensione. Il

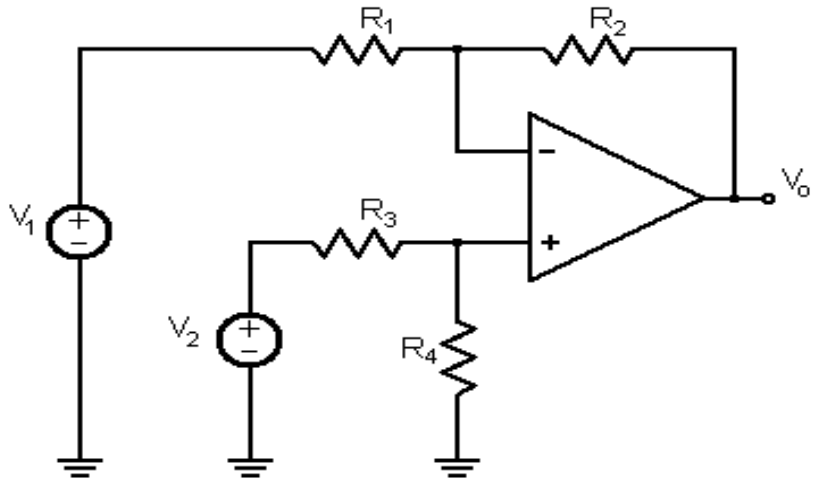

Figura A.14. Amplificatore Differenziale

circuito è una combinazione dell'amplificatore invertente e di quello non invertente. Per analizzarlo si utilizza nuovamente sovrapposizione degli effetti per i generatori di tensione V_1 e V_2:

- se $V_2 = 0$ il circuito è riconducibile ad un amplificatore invertente, la tensione in uscita sarà pari a
$$V_{out1} = -\frac{R_2}{R_1} V_1 \tag{A.61}$$
- se $V_1 = 0$ il circuito è assimilabile ad un amplificatore non invertente con un partitore di tensione sul morsetto positivo, si avrà dunque
$$V_{out2} = \left(1 + \frac{R_2}{R_1}\right) \frac{R_4}{R_3 + R_4} V_2 \tag{A.62}$$

Sommando le due tensioni parziali e sapendo che le resistenze sono uguali si ottiene

$$V_{out} = V_{out1} + V_{out2} = V_2 - V_1 \tag{A.63}$$

A questo punto si può passare all'analisi degli offset e del rumore.

A.6.6 L'offset dell'amplificatore differenziale

Se si azzerano le sorgenti di segnale V_1 e V_2 si ottiene il circuito in figura A.10, di conseguenza le equazioni dell'offset saranno le medesime. Si rimanda al paragrafo A.6.2 per ulteriori spiegazioni.

A.6.7 Il rumore dell'amplificatore differenziale

Il modello di rumore dell'amplificatore differenziale è identico a quello già utilizzato per il sommatore non invertente, come si può notare dalla figura A.11. Le equazioni da trovare sono dunque identiche a quelle esposte nel paragrafo A.6.3.

A.6.8 La dissipazione nell'amplificatore differenziale

Questo caso è del tutto analogo a quello trattato nel paragrafo A.6.4 per il sommatore non invertente. Per potersi mettere nel caso peggiore è necessario imporre che la tensione in uscita sia massima e che una delle due tensioni d'ingresso sia nulla, ipotizzando V_2 nulla si avrà quindi

$$V_{out} = V_1 - V_2 = V_1 - 0 = V_1 \tag{A.64}$$

A questo punto, analizzando il circuito equivalente si ricade nello stesso caso trattato in precedenza per il sommatore non invertente: ipotizzando che tutte le resistenze siano uguali su ognuna di esse cadrà una tensione pari a

$$V_R = \frac{R}{R+R} V_1 = \frac{V_1}{R} \tag{A.65}$$

Come nel caso precedente, la massima potenza dissipata varrà quindi

$$P_R = \frac{V_R^2}{R} = \frac{1}{R}\left(\frac{V_1}{2}\right)^2 \tag{A.66}$$

A.6.9 Il sommatore invertente

Il circuito riportato in figura A.15 permette di invertire la somma di due segnali d'ingresso. Il circuito è amplificatore invertente con più di un segnale entrante nel morsetto negativo. utilizzando sovrapposizione degli effetti sui due generatori V_1 e V_2:

- se $V_2 = 0$ il circuito è esattamente identico ad un amplificatore invertente, la resistenza R_2 si trova infatti fra due tensioni nulle (il riferimento e la massa virtuale del morsetto negativo) e quindi non deve essere considerata. in uscita si avrà

$$V_{out1} = -\frac{R_3}{R_1} V_1 \tag{A.67}$$

- anche se $V_1 = 0$ il circuito diventa un amplificatore invertente, in questo caso sarà la resistenza R_1 a trovarsi fra due tensioni nulle. La tensione in uscita varrà

$$V_{out2} = -\frac{R_3}{R_2} V_2 \tag{A.68}$$

Sommando i due contributi e sapendo che $R_1 = R_2 = R_3$ si ottiene che

$$V_o = V_{out1} + V_{out2} = -(V_1 + V_2) \tag{A.69}$$

A questo punto si può procedere con l'analisi dell'offset e del rumore.

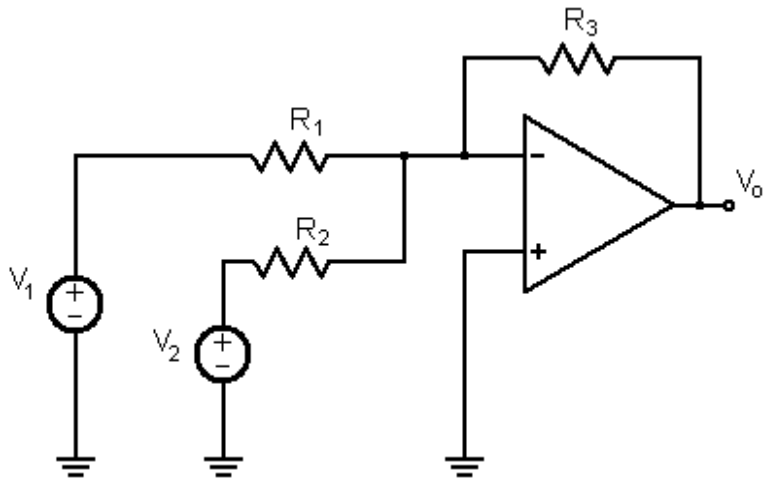

Figura A.15. Sommatore Invertente

A.6.10 L'offset del sommatore invertente

Il circuito che si ottiene cortocircuitando i generatori V_1 e V_2 è riportato in figura A.16. Si

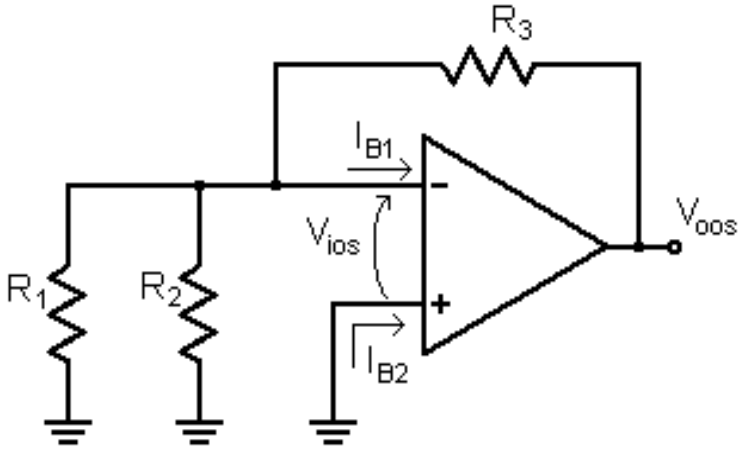

Figura A.16. Contributi d'offset nel Sommatore Invertente

procede con sovrapposizione degli effetti ad analizzare i contributi di offset presenti.

- la tensione d'offset d'ingresso V_{ios1} tra i due terminali dell'operazionale si propaga in

uscita moltiplicata per il guadagno ad anello chiuso del circuito, si ha quindi

$$V_{oos1} = \left(1 + \frac{R_3}{R_1 \parallel R_2}\right) V_{ios} \tag{A.70}$$

- la corrente di I_B nel morsetto invertente si propaga in uscita secondo la seguente equazione già incontrata per l'amplificatore invertente nel paragrafo A.3.2:

$$V_{oos2} = R_3 I_B \tag{A.71}$$

Sommando i due contributi si ha infine

$$V_{oos} = V_{oos1} + V_{oos2} = \left(1 + \frac{R_3}{R_1 \parallel R_2}\right) V_{ios} + R_3 I_B \tag{A.72}$$

Poiché $R_1 = R_2 = R_3$ la tensione di offset globale sarà pari a

$$V_{oos} = 3V_{ios} + R_3 I_B \tag{A.73}$$

A.6.11 Il rumore del sommatore invertente

Come per i casi precedenti è necessario azzerare i generatori V_1 e V_2 ed estrarre il modello di rumore del circuito. Si considerano separatamente i contributi di rumori che si propagano

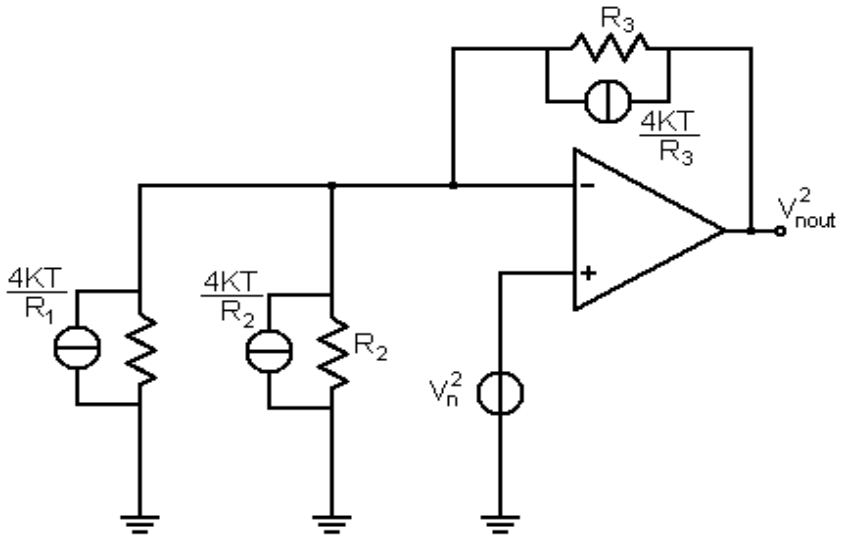

Figura A.17. Modello di rumore del Sommatore Invertente

a partire dai due terminali d'ingresso dell'operazionale:

- nel morsetto non invertente è presente solo il contributo di rumore dell'operazionale, si ha quindi

$$V_{nout1}^2 = \left|1 + \frac{R_3}{R_1 \parallel R_2}\right|^2 V_{nin}^2 \tag{A.74}$$

- nel morsetto invertente convergono i contributi di rumore dei tre resistori, utilizzando il modello in corrente si ottiene:

$$V_{nout2}^2 = |R_3|^2 \left(\frac{4K \cdot T}{R_1} + \frac{4K \cdot T}{R_2} + \frac{4K \cdot T}{R_3}\right) \tag{A.75}$$

Sommando i due contributi e tenendo conto che le resistenze devono essere identiche si ottiene infine

$$V_{nout}^2 = V_{nout1}^2 + V_{nout2}^2 = 9V_{nin}^2 + 12K \cdot T \cdot R \tag{A.76}$$

A.6.12 La dissipazione nel sommatore invertente

Come per i circuiti dell'amplificatore differenziale e del sommatore non invertente è necessario mettersi nel caso peggiore, in cui la tensione di uscita è massima ed una di quelle d'ingresso è nulla, si avrà quindi che

$$V_{out} = -(V_1 + V_2) = -(V_1 + 0) = -V_1 \tag{A.77}$$

In questo caso il circuito equivalente è il quello riportato i figura A.18 in cui la resistenza

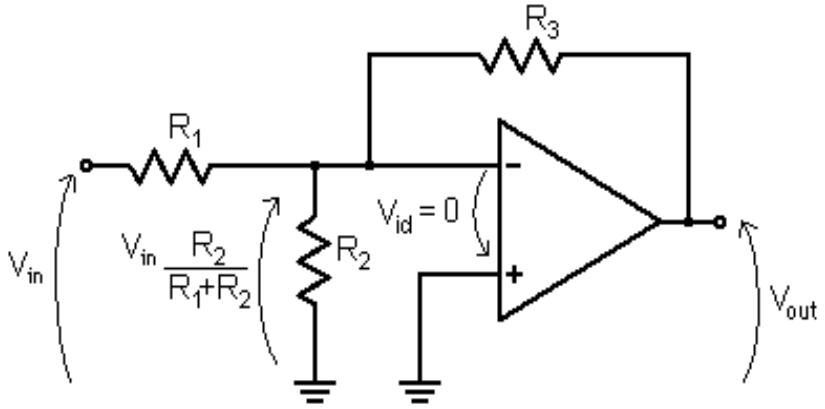

Figura A.18. Dissipazione di potenza nel Sommatore Invertente

collegata all'ingresso nullo non verrà considerata in quanto si trova fra il riferimento ed la massa virtuale del morsetto invertente e quindi non è percorsa da corrente. Questo caso è equivalente a quello visto nel paragrafo A.5.2 in cui la tensione che cade sulle due resistenze è la stessa in valore assoluto

$$V_R = |V_1| \tag{A.78}$$

Si ha dunque che la potenza dissipata vale

$$P_R = \frac{V_R^2}{R} = \frac{|V_1|^2}{R} \tag{A.79}$$

A.7 L'amplificatore di Sample & Hold

Come più volte ricordato all'interno di questa trattazione, non è possibile fornire una descrizione dettagliata e univoca del circuito di Sample & Hold, a causa della diverse architetture con cui sono stati realizzati i dispositivi presenti nel database. Sono esclusi dalla trattazione i modelli di rumore e di offset essendo impossibile risalire al circuito. Di seguito verranno descritte le fasi di funzionamento del S/H e per ognuna di esse verrà fornita una descrizione dei parametri in gioco che sono stati usati per selezionare un dispositivo piuttosto che un altro fra quelli disponibili. Questi parametri sono comuni a diversi tipi di architettura, ma per comodità ci si riferirà sempre a quella più semplice possibile, rappresentata da due buffer (uno in ingresso ed uno in uscita) per limitare l'assorbimento di corrente, un interruttore per campionare il segnale e una capacità di mantenimento per mantenerlo a quel valore per un determinato intervallo di tempo (generalmente quasi tutto il periodo di campionamento). Il circuito descritto è riportato in figura A.19.

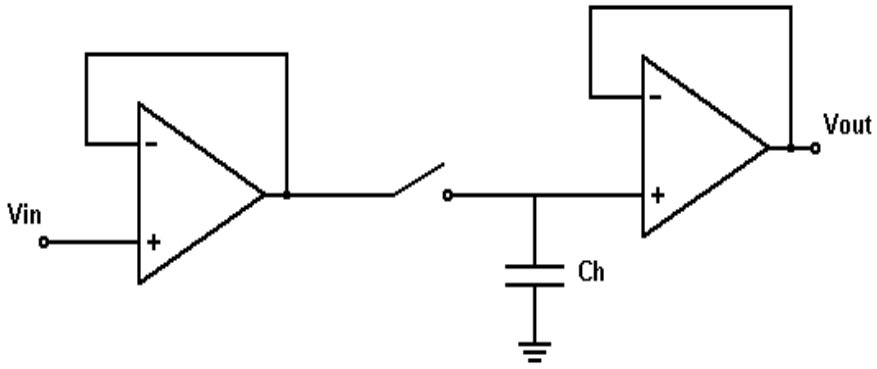

Figura A.19. Circuito d'esempio dell'amplificatore di Sample & Hold

A.7.1 La fase di Sample

Quando l'interruttore è chiuso il segnale viene campionato e la sua tensione viene "memorizzata" nella capacità di mantenimento. In questa fase è importante che il segnale venga mantenuto con una buona precisione per il tempo che serve al convertitore analogico/digitale per campionarlo e convertirlo. Tra le specifiche più importanti considerate in questa fase si ricordano:

Voltage Offset

Corrisponde alla variazione di tensione dallo 0 quando il dispositivo si trova nella fase di Sample e in ingresso viene dato un valore nullo. Per mantenere una precisione corretta nelle applicazioni di conversione analogico/digitale, l'offset deve essere inferiore a 1/2 LSB[2] oppure

$$V_{os} < \frac{FS}{2^{n+1}} \tag{A.80}$$

dove FS (*Full Scale*) è il range di tensioni acquisibili dal S/H mentre n è la risoluzione dell'ADC. in alcuni Sample & Hold è prevista la possibilità di annullare l'offset.

Gain Error

È definito come il rapporto fra la differenza di tensione fra ingresso e uscita fratto la tensione d'ingresso quando il dispositivo è nella fase di Sample.

$$\Delta A = \frac{V_{out} - V_{in}}{V_{in}} < \frac{1}{2^{n+1}} \tag{A.81}$$

Anche in questo caso se il S/H è utilizzato per applicazione di conversione A/D, è necessario che l'errore sia inferiore a 1/2 LSB. n indica nuovamente la risoluzione del convertitore.

Full-Power Bandwidth

Viene comunemente definita in due modi. Alcune aziende la considerano la frequenza alla quale il guadagno in tensione del S/H scende di 3 dB rispetto al guadagno in continua per un ingresso *full scale*. Altri derivano il valore della banda dalla misura dello slew rate. Per definizione la larghezza di banda è uguale alla frequenza dell'onda sinusoidale che ha la massima variazione di carica pari allo slew rate. Questo è dato da

$$BW_{fp} = \frac{SR}{2\pi V_p} \tag{A.82}$$

Dove SR è lo slew rate mentre $2V_p$ corrisponde al *full scale*

Small Signal Bandwidth

È intesa come la frequenza alla quale il guadagno in tensione scende di 3 dB rispetto allo stesso guadagno in continua, quando in ingresso è dato un segnale inferiore rispetto al *full scale* di almeno 20-40 dB. Questa specifica è importante per quelle applicazioni che richiedono le conversioni di piccoli segnali ad alta frequenza.

Slew rate

Corrisponde alla massima variazione della tensione di uscita che il dispositivo è in grado di seguire quando si trova nella fase di Sample. I)l suo valore dipende dalla capacità di mantenimento ed è condiziona la banda *Full-Power* ed il tempo di acquisizione.

[2]Less Significant Bit ovvero il valore di tensione che convertito in digitale è rappresentato solo dal bit meno significativo

A.7.2 La transizione da Sample a Hold

In figura A.20 sono riportate alcune delle sorgenti di errore che si verificano durante la transizione da Sample a Hold e durante la fase di Hold. Nella fase di passaggio si possono notare l'*hold step* descritto nel paragrafo A.7.2 il tempo di assestamento descritto nel paragrafo A.7.2.

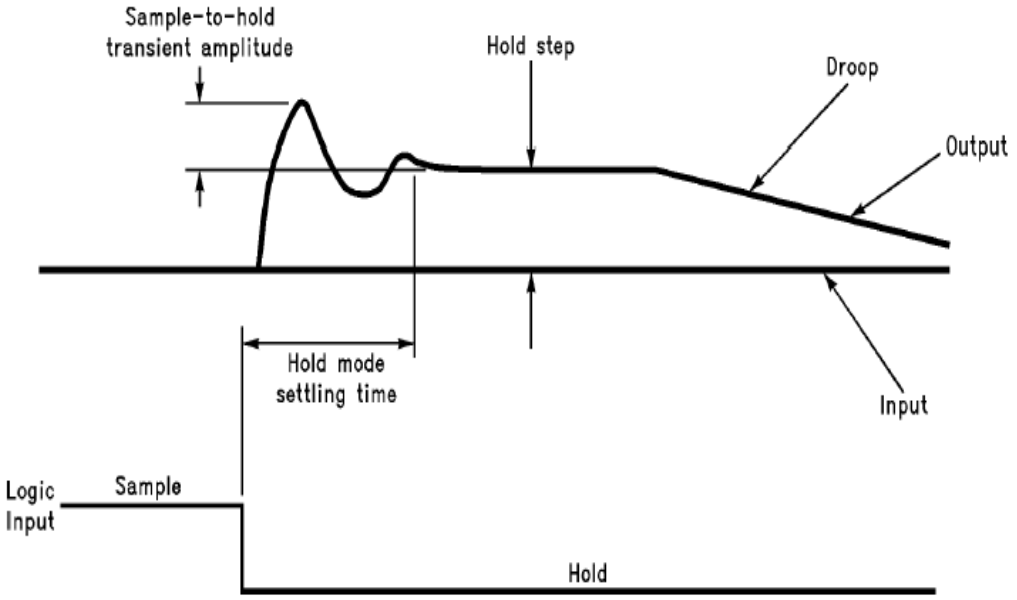

Figura A.20. Sorgenti di errore nelle fasi Sample-to-Hold e Hold

Aperture Time

Conosciuto anche come ritardo di apertura, può essere definito in più modi. La definizione più rigida afferma che è l'intervallo di tempo durante il quale il segnale è scollegato dalla capacità di mantenimento dopo che è stato dato un segnale di Hold. Secondo la definizione meno rigida, è il tempo che intercorre fra l'applicazione del comando di Hold e l'istante in cui il segnale d'ingresso è completamente scollegato dalla capacità. Quest'ultima definizione considera anche il ritardo digitale che si verifica fra quando viene applicato l'Hold e quando l'interruttore che connette l'ingresso alla capacità *inizia* ad aprirsi. Il tempo di apertura non pone un limite alla massima frequenza del segnale d'ingresso poiché per un segnale sinusoidale, l'errore di tensione causato dal tempo di apertura, si manifesta come uno sfasamento e non come un'alterazione del'ampiezza del segnale d'ingresso o della sua frequenza. In figura A.21 è riportato il tempo di apertura.

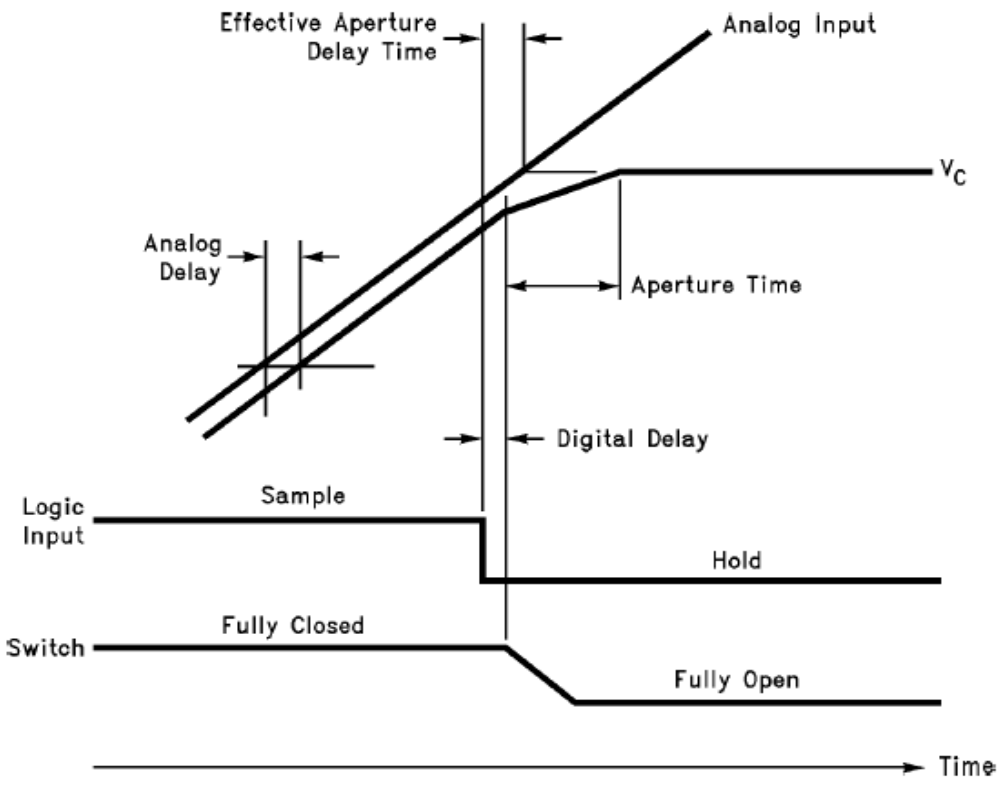

Figura A.21. Tempo d'Apertura

Aperture Jitter

Viene anche conosciuto come incertezza di apertura (*aperture uncertainty*). È dovuta al rumore che presente sul comando di Hold e si manifesta come un errore nella temporizzazione. Viene spesso indicato in rms come deviazione standard del tempo di apertura. Il jitter pone un limite superiore alla massima frequenza del segnale d'ingresso che può essere campionato dal S/H: affinché non si perda precisione è necessario che il segnale non cambi di più di $\pm 1/2$ LSB durante il tempo del jitter. Utilizzando un'onda sinusoidale del tipo $V = A \sin(2\pi \cdot f \cdot t)$ si ha

$$\frac{dV}{dt} = 2\pi \cdot f \cdot A \cos(2\pi \cdot f \cdot t) < \frac{\pm 1/2 LSB}{t_{aj}} \tag{A.83}$$

Dove A indica il range della tensione d'ingresso e t_{aj} è il jitter di apertura. Poiché $1/2LSB = A/2^n$ dove n è la risoluzione del convertitore, si ha

$$f < \frac{1}{2\pi \cdot 2^n \cdot t_{aj}} \tag{A.84}$$

Per esempio utilizzando un convertitore a 12 bit, il cui s/H ha un jitter d'apertura di $100ps$, il segnale campionato può avere una frequenza massima di $388kHz$. Tutto ciò è

naturalmente possibile se la frequenza di campionamento del Sample & Hold è almeno doppia rispetto alla frequenza del segnale.

Hold Step

Conosciuto anche come *pedestal error* o *sample-to-hold offset* è il gradino di tensione che appare in uscita in seguito alla transizione dalla fase di Sample a quella di Hold. È causato da un trasferimento di carica verso la capacità di mantenimento, dovuto all'apertura dell'interruttore. L'*Hold step* può essere determinato a partire dal trasferimento di carica con la seguente formula

$$V_{HS} = \frac{Q}{C_H} \tag{A.85}$$

Dove Q è la carica trasferita alla capacità di mantenimento C_H. Si deduce che questo offset può essere ridotto aumentando il valore della capacità: questo ovviamente aumenterà il tempo di acquisizione del segnale d'ingresso.

Settling Time

È il tempo richiesto affinché l'uscita si assesti all'interno di una determinata banda di errore dopo che il comando d Hold è stato dato. L'errore è generalmente specificato come lo 0.1% o lo 0.01% del massimo segnale d'ingresso. Per le applicazioni di conversione è necessario che l'uscita si assesti entro $\pm 1/2$ LSB prima che la conversione abbia inizio. Il tempo di assestamento è anche importante perché insieme al tempo di acquisizione del S/H e al tempo di conversione dall'ADC determina la massima frequenza di campionamento del sistema S/H-A/D. Si ha che

$$(f_s)_{MAX} = \frac{1}{t_{aq} + t_c + t_{HS}} \tag{A.86}$$

A.7.3 La fase di Hold

Droop Rate

È la variazione della tensione d'uscita nel tempo causata dalle correnti di perdita della capacità. Se il S/H ha la capacità integrata, il *droop rate* è indicato nel data sheet, altrimenti può essere calcolato con la seguente formula

$$\frac{dV_{CH}}{dt} = \frac{I_L}{C_H} \tag{A.87}$$

Dove I_L rappresenta la corrente di perdita (*leakage*) della capacità C_H.

Il droop rate è importante in quelle applicazioni in cui la tensione campionata deve essere mantenuta costante entro una determinata banda di errore per un certo intervallo di tempo. In applicazioni di conversione analogico/digitale la variazione di tensione deve essere inferiore a 1/2 LSB durante il tempo di conversione t_c, si ha dunque

$$\frac{dV_{CH}}{dt} = \frac{1/2LSB}{t_c} = \frac{FS}{(2^{n+1})t_c} \tag{A.88}$$

Dove FS è la tensione *full scale* del convertitore, n è la sua risoluzione e t_c è il suo tempo di conversione. Il *droop rate* può essere ridotto aumentando il valore della capacità di mantenimento. Si ricorda comunque che in questo caso verrà aumentato il tempo di acquisizione del Sample & Hold.

A.7.4 La transizione da Hold a Sample

Acquisition Time

È il massimo tempo richiesto per acquisire un nuovo ingresso in tensione quando da quando è stato dato il segnale di Sample. Un segnale si può dire acquisito quando si assesta all'interno di una determinata banda di errore rispetto al valore di uscita desiderato (generalmente intorno allo 0.1% o allo 0.01%). Il tempo di acquisizione massimo si ha se la capacità deve essere caricata con la tensione *full scale*. Il tempo di acquisizione può essere ridotto scegliendo una capacità di mantenimento piccola; questo andrà a inficiare sul *droop rate* e sull'*hold step*.

Appendice B

Listati

B.1 *AllComponentSearch.m*

```
function [varargout] = AllComponentSearch(Option, ...
  FigHandle, varargin)

% la funzioneè  chiamata in seguito alla pressione
% dei pulsanti find component o cheap component
% presenti sull'interfaccia o alla selezione di
% uno dei componenti presenti nel PopupMenu

% si definisce una variabile globale Database che
%è  vuota nel caso in cui non sia ancora stata
% inizializzata nel workspace di base, altrimenti
% àconterr una struttura descritta di seguito
global Database;

BlockData = get(FigHandle, 'UserData');
BlockHandle = getfield(BlockData, 'BlockHandle');
ExtendedBlockType = get_param(BlockHandle, ...
  'sim_ExtendedBlockType');
EmptyStr = '';
DatabaseCompList = '';

try DatabaseCompList = getfield(Database, 'CompList'); end

% se il circuito che rappresenta il blocco ha una
% lista di dispositivi che permettono di descriverlo,
%è  necessario recuperare tali informazioni
CircuitDeviceListStr = eval([ExtendedBlockType ...
    '_ParamUI(''DeviceOfCircuit'', BlockHandle);'], ...
  'EmptyStr');
CircuitPassCompListStr = eval([ExtendedBlockType ...
    '_ParamUI(''PassCompOfCircuit'', BlockHandle);'], ...
  'EmptyStr');
```

```
if isempty(CircuitPassCompListStr) | ...
    isempty(CircuitDeviceListStr)
  CompListStr = [CircuitDeviceListStr ...
      CircuitPassCompListStr];
else
  CompListStr = [CircuitDeviceListStr '|' ...
      CircuitPassCompListStr];                                    40
end

OrigCompListStr = CompListStr;

while(~isempty(CompListStr))
  [AnalyzedComp, CompListStr] = strtok(CompListStr, '|');
  % i dati del database vengono caricati solo se
  % non sono ancora stati caricati  (la variabile
  % che li contieneè  vuota oppure non
  % li contiene ancora)                                          50
  if isempty(Database) | ...
      isempty(findstr(DatabaseCompList, AnalyzedComp))
    % estraggo i dati dal database: se Database
    %è  vuoto oppure non contiene le informazioni
    % relative ad un determinato dispositivo, allora
    % devo aggiornarlo estraendo i dati relativi al
    % nuovo dispositivo e includerli nella
    % struttura Database
    [Data, NamesOfColumns] = data_extraction(AnalyzedComp);
    if isempty(Data) | isempty(NamesOfColumns)                    60
      continue;
    end

    % DeviceList: -->
    %           contiene la lista dei dispositivi
    %           di cui si sono àgi estratte
    %           informazioni dal database
    % [AnalyzedComp '_Data']: -->
    %           contiene un array di celle con
    %           tutti i dati estratti dal database                70
    %           per il dispositivo AnalyzedDevice
    % [AnalyzedComp '_NamesOfColumns']: -->
    %           contiene un array di celle con
    %           i nomi di tutte le colonne
    %           relative al database del
    %           dispositivo AnalyzedDevice
    DatabaseCompList = [DatabaseCompList AnalyzedComp '|'];

    Database = setfield(Database, ...
      'CompList', ...                                             80
      DatabaseCompList);
```

```
    Database = setfield(Database, ...
      [AnalyzedComp '_Data'], ...
      Data);
    Database = setfield(Database, ...
      [AnalyzedComp '_NamesOfColumns'], ...
      NamesOfColumns);
  end %isempty(Database) | ...
end % while

switch upper(Option)
  case 'FIND',

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%% ESTRAZIONE PARAMETRI NECESSARI PER GLI
    %%%% ALGORITMI DI SCELTA DEI COMPONENTI
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    ErrorFound = 0;
    [MaskParamTagStr, ParamNameStr] = eval([ExtendedBlockType ...
      '_ParamUI(''NeededVariables'', BlockHandle);']);
    NumberOfVariables = length(strfind(ParamNameStr,'|'))+1;

    % estrazione dei parametri analogici
    % del blocco che sono necessari per
    % effettuare le scelte dei componenti
    % ùpi adatti: i parametri principali sono
    % --> Max Gain Error
    % --> Max Offset
    % --> Non Linearity Polinomial Coefficent
    % --> White Noise (std dev)
    % --> Lower Saturation
    % --> Upper Saturation
    for k = 1:NumberOfVariables
      [MaskParamTag, MaskParamTagStr] = strtok(MaskParamTagStr,'|');
      [ParamName, ParamNameStr] = strtok(ParamNameStr,'|');
      try
        eval(...
          [ParamName '= eval_param(BlockHandle,' ...
            ' ''' MaskParamTag ''' );']);
      catch
        errordlg(...
          ['No admitted value for ' MaskParamTag '''s ' ...
            'tag parameter.'], ...
          'Value Parameter error');
        ErrorFound = 1;
      end
    end

    if ErrorFound return; end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% ANALISI DEI COMPONENTI PASSIVI E ATTIVI
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

PassCompAlgorithm = eval([ExtendedBlockType ...
    '_ParamUI(''PassCompAlgorithm'', BlockHandle);'],'0');
if PassCompAlgorithm
  % nel caso in cui siano presenti degli
  % algoritmi per la definiziome dei componenti
  % passivi (resistenze, àcapacit) del circuito                    140
  % corrispondente al blocco. La struttura (dinamica)
  % contiene nei vari campi tutte le variabili
  % relative ai corrispondenti componenti passivi
  PassCompAlgResults = AnalogCircuitDescription(...
    'PassComp_Structure', ...
    FigHandle);

  % calcolo i valori dei componenti passivi
  % con il comando eval, questi valori saranno
  % memorizzati nello stack della presente funzione                150
  % e saranno opportunamente salvati di seguito
  eval(PassCompAlgorithm);
end %PassCompAlgorithm


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% se nonè  presente una lista di dispositivi
% tra cui scegliere o siè  esaurita la lista di
% quelli analizzati, non si effettuano                             160
% ulteriori operazioni
while(~isempty(OrigCompListStr))
  [AnalyzedComp, ...
      OrigCompListStr] = strtok(OrigCompListStr, '|');
  ErrorFound = 0; %ri-inizializzo

  % estraggo l'algoritmo di ricerca dei
  % componenti attivi, ovvero i dispositivi
  % con cuiè  realizzato quel determinato blocco
  CompAlgorithm = eval([ExtendedBlockType                          170
    '_ParamUI(''' [AnalyzedComp '_Algorithm'] ''', ' ...
      'BlockHandle);'], 'EmptyStr');

  % estraggo i dati numerici (e i nomi delle
  % relative colonne) del database che erano
  % stati in precedenza memorizzati nella
  % struttura (variabile globale) chiamata Database
  Data = getfield(Database, [AnalyzedComp '_Data']);
  NamesOfColumns = getfield(Database, [AnalyzedComp ...
```

```
    '_NamesOfColumns']);                                                    180

% valuto l'algoritmo di ricerca di quel determinato
% dispositivo. époich potrebbero esserci ùpi istruzioni
% "find" all'interno dell'algoritmo di scelta del
% componente, si àdovr prendere in considerazione solo
% l'ultimo presente nella stringa
InitOfFindCommand = strfind(CompAlgorithm,'find');
eval(CompAlgorithm(1:InitOfFindCommand(end)-1));
ListOfCompIndex = eval(...
  CompAlgorithm(InitOfFindCommand(end):end)...                              190
  );

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Status = 'on'; % àabilit il pulsante cheap
if isempty(ListOfCompIndex)
  % se non sono stati trovati dispositivi adatti viene
  % aggiornata l'interfaccia e si disabilita il pulsante
  % di cheap component
  Status = 'off';
  ErrorFound = 1;                                                           200
  UnknownData = '-???-';
  errordlg(['No suitable ' AnalyzedComp ...
      ' found in database.'], ...
    'Component error');
  set_PopupMenu_value(FigHandle, ...
    UnknownData, ['sim_' AnalyzedComp '_Selected'], ...
    UnknownData, ['sim_' AnalyzedComp '_Description'], ...
    UnknownData, ['sim_' AnalyzedComp '_List'], ...
    UnknownData, ['sim_' AnalyzedComp '_Indices']);
  if ~isempty(strfind(CircuitDeviceListStr, AnalyzedComp))                  210
    AnalogDeviceDescription('Update', ...
      FigHandle, AnalyzedComp, ...
      UnknownData, UnknownData);
  else
    AnalogCircuitDescription('Update', ...
      FigHandle, AnalyzedComp, ...
      UnknownData, UnknownData);
  end
end
set(findobj(FigHandle, ...                                                  220
  'Tag', ['Cheap_' AnalyzedComp]), ...
  'Enable', Status);
%%%%%%%%%%%%%%%%%%%%%%%%%

% se si sono verificati degli errori si
% analizzano comunque gli algoritmi degli
% altri dispositivi eventualmente presenti
if ErrorFound continue; end
```

```
  ColumnOfNameOfComp = strmatch('Name', ...
    NamesOfColumns);
  ColumnOfDescriptionOfComp = strmatch('Description', ...
    NamesOfColumns);

  FirstCompName = Data{ListOfCompIndex(1), ...
      ColumnOfNameOfComp};
  FirstCompDescription = Data{ListOfCompIndex(1), ...
      ColumnOfDescriptionOfComp};
  ListOfCompName = FirstCompName;

  % creazione della stringa contenente l'elenco
  % dei nomi dei dispositivi adatti. I vari
  % componenti sono separati dal carattere pipeline
  % "|" per essere inseriti nel PopupMenu
  for j = 2 : length(ListOfCompIndex)
    ListOfCompName = [ListOfCompName '|' ...
        Data{ListOfCompIndex(j), ColumnOfNameOfComp}];
  end %for j = ...

  FirstNeededRow = {Data{ListOfCompIndex(1), :}};

  % aggiornamento dei componenti (sia
  % dispositivi che resistenze...)
  if strfind(CircuitDeviceListStr, AnalyzedComp)
    AnalogDeviceDescription('Update', FigHandle, ...
      AnalyzedComp, FirstNeededRow, NamesOfColumns);
  else
    AnalogCircuitDescription('Update', FigHandle, ...
      AnalyzedComp, FirstNeededRow, NamesOfColumns);
  end

  % aggiornamento del PopupMenu di
  % selezione del dispositivo
  set_PopupMenu_value(FigHandle, ...
    FirstCompName, ...
    ['sim_' AnalyzedComp '_Selected'], ...
    FirstCompDescription, ...
    ['sim_' AnalyzedComp '_Description'], ...
    ListOfCompName, ...
    ['sim_' AnalyzedComp '_List'], ...
    num2str(ListOfCompIndex), ...
    ['sim_' AnalyzedComp '_Indices']);
end %while(~...

% nel caso in cui siano presenti componenti
% passivi (precedentemente calcolati) si deve
% aggiornare il relativo frame. I valori
```

```
  % calcolati per tali componenti sono stati
  % memorizzati nella struttura (dinamica)
  % PassCompAlgResults                                          280
  if PassCompAlgorithm
    PassCompParam = fieldnames(PassCompAlgResults);

    for k = 1: length(PassCompParam)
      % a partire dalla lista delle variabili
      % che si ottengono come risultato dell'analisi
      % degli algoritmi per i componenti passivi del
      % circuito (resistenze, àcapacit,...) definisco
      % una struttura che contenga tali risultati
      % sotto forma di stringa                                  290
      ActualFieldName = PassCompParam{k};
      ActualFieldValue = num2str(eval(ActualFieldName));
      if isempty(ActualFieldValue)
        errordlg(['No data found for passive component ' ...
            ActualFieldName], ...
          'Passive Component error');
        ActualFieldValue = '-???-';
        ErrorFound = 1;
      end
      PassCompAlgResults = setfield(PassCompAlgResults, ...    300
        ActualFieldName, ActualFieldValue);
    end
    AnalogCircuitDescription('Update', ...
      FigHandle, PassCompAlgResults);
  end %if PassCompAlgorithm

  % if ErrorFound return; end

case 'COMPLIST'
                                                               310
  PopupMenuHandle = gcbo;
  % identifico il tipo di dispositivo a cui
  % si riferisce l'oggetto selezionato del PopupMenu
  PopupMenuTag = get(PopupMenuHandle, 'Tag');
  [Intro, Last] = strtok(PopupMenuTag, '_');
  AnalyzedComp = strtok(Last, '_');

  % estraggo i dati numerici (e i nomi delle
  % relative colonne) del database
  Data = getfield(Database, ...                                320
    [AnalyzedComp '_Data']);
  NamesOfColumns = getfield(Database, ...
    [AnalyzedComp '_NamesOfColumns']);

  % estrazione della stringa delle opzioni
  % memorizzate nel PopupMenu
```

```
  ListOfCompFromDB = getfield(BlockData, ...
    ['sim_' AnalyzedComp '_List']);

  % identificazione dell'indice corrispondente                    330
  % alla posizione di quel determinato
  % oggetto all'interno del PopupMenu
  ValueOfPopupMenuComp = get(PopupMenuHandle, 'Value');
  IndicesOfCompFromDB = str2num(getfield(BlockData, ...
    ['sim_' AnalyzedComp '_Indices']));

  if isempty(IndicesOfCompFromDB)
    return;
  end
  % estrazione della riga del database                             340
  % che contiene le informazioni
  % relative al nuovo dispositivo
  % selezionato
  CompRow = {...
      Data{IndicesOfCompFromDB(ValueOfPopupMenuComp),:}...
    };
  SelectedCompFromDB = PopupConversion(...
    ValueOfPopupMenuComp, ListOfCompFromDB);
  DescriptionOfCompFromDB = CompRow{...
      strmatch('Description', NamesOfColumns)};                  350

  % aggiornamento della finestra dei componenti
  % attivi/passivi con le grandezze relative
  % a quel determinato componente selezionato
  if strfind(CircuitDeviceListStr, AnalyzedComp)
    AnalogDeviceDescription('Update', FigHandle, ...
      AnalyzedComp, CompRow, NamesOfColumns);
  else
    % if strfind(CircuitPassCompListStr, AnalyzedComp)
    AnalogCircuitDescription('Update', FigHandle, ...            360
      AnalyzedComp, CompRow, NamesOfColumns);
    % end
  end

  set_PopupMenu_value(FigHandle, ...
    SelectedCompFromDB, ...
    ['sim_' AnalyzedComp '_Selected'], ...
    DescriptionOfCompFromDB, ...
    ['sim_' AnalyzedComp '_Description']);
                                                                 370
case 'CHEAPCOMP'
  % scelta del componente di costo minimo
  % all'interno di quelli inclusi nella lista
  % precedentemente ottenuta in seguito alla
  % pressione del pulsante cheap
```

```
PushButtonHandle = gcbo;

PushButtonTag = get(PushButtonHandle, 'Tag');
[Intro, Last] = strtok(PushButtonTag, '_');
AnalyzedComp = strtok(Last, '_');

Data = getfield(Database, [AnalyzedComp '_Data']);
NamesOfColumns = getfield(Database, ...
  [AnalyzedComp '_NamesOfColumns']);
IndicesOfCompFromDB = str2num(...
  getfield(BlockData, ...
  ['sim_' AnalyzedComp '_Indices'])...
  );

if isempty(IndicesOfCompFromDB)
  return;
end

ColumnOfNameOfComp = strmatch('Name', ...
  NamesOfColumns);
ColumnOfDescriptionOfComp = strmatch('Description', ...
  NamesOfColumns);
ColumnOfPriceOfComp = strmatch('Price', ...
  NamesOfColumns);
IndicesOfCheapCompFromDB = [];

% tutti i prezzi dei componenti utilizzabili
% sono messi in un vettore, si determina
% l'elemento di valore minimo ed il relativo
% indice nel vettore iniziale
CompPriceFromDB = [...
    Data{IndicesOfCompFromDB, ColumnOfPriceOfComp}...
  ];
IndexOfIndices = find(...
  CompPriceFromDB == min(CompPriceFromDB)...
  );
IndicesOfCheapCompFromDB = ...
                    IndicesOfCompFromDB(IndexOfIndices);

% estrazione dei dati relativi al componente di prezzo
% inferiore
NameOfFirstCheapComp = Data{IndicesOfCheapCompFromDB(1), ...
    ColumnOfNameOfComp};
DescriptionOfFirstCheapComp = Data{...
    IndicesOfCheapCompFromDB(1), ...
    ColumnOfDescriptionOfComp ...
};
ListOfCheapCompFromDB = NameOfFirstCheapComp;
```

```
  for k = 2 : length(IndicesOfCheapCompFromDB)
    ListOfCheapCompFromDB = [ListOfCheapCompFromDB '|' ...
        Data{IndicesOfCheapCompFromDB(k), ColumnOfNameOfComp}];
  end
                                                                        430
  set_PopupMenu_value(FigHandle, ...
    NameOfFirstCheapComp, ...
    ['sim_' AnalyzedComp '_Selected'], ...
    DescriptionOfFirstCheapComp, ...
    ['sim_' AnalyzedComp '_Description'], ...
    ListOfCheapCompFromDB, ...
    ['sim_' AnalyzedComp '_List'], ...
    num2str(IndicesOfCheapCompFromDB), ...
    ['sim_' AnalyzedComp '_Indices']);
                                                                        440
  FirstNeededRow = {Data{IndicesOfCheapCompFromDB(1), :}};

  if strfind(CircuitDeviceListStr, AnalyzedComp)
    AnalogDeviceDescription('Update', FigHandle, ...
      AnalyzedComp, FirstNeededRow, NamesOfColumns);
  else
    % se nonè  un componente attivoè
    % sicuramete un componente passivo
    AnalogCircuitDescription('Update', FigHandle, ...
      AnalyzedComp, FirstNeededRow, NamesOfColumns);                    450
  end

otherwise
  return;
end
```

B.2 *Analog_Board_Callback.m*

```
function [varargout] = Analog_Board_Callback(FigHandle, ...
  CallbackType, varargin)


% viene chiamata in seguito alla modifica
% dello stato della finestra relativa alla
% scelta della scheda analogica e dei suoi parametri .

switch upper(CallbackType)
  case 'MAIN'                                                    10
    %è  la callback relativa alla scelta
    % nel popup menu di una determinata
    % scheda analogica. per prima cosa
    % effettua la cancellazione degli
    % oggetti relativi alla precedente
    % scheda e successivamente alla
    % deposizione di quelli relativi
    % alla nuova scheda scelta.
    UserData = get(FigHandle, 'UserData');
                                                                 20
    [szBoard, Board] = custom_ParamUI('sim_Analog_Type');

    CurrentBoard = getfield(UserData, ...
      'Dinamic_Analog_Board');

    % altezza del frame attuale relativo
    % alla precedente scheda
    CurrentFrameHeight = eval(...
      [CurrentBoard '_ParamUI(''VertDim'');'] ...
      );                                                         30

    % cancellazione oggetti della
    % precedente scheda
    eval(...
      [CurrentBoard '_ParamUI(''Delete'', gcbf);']...
      );
    % trovo la nuova scheda scelta,
    % la sua dimensione del frame
    % e memorizzo le nuove informazioni
    % acquisite                                                  40

    TBHandle = findobj(FigHandle, ...
      'Tag', 'sim_Analog_Type');
    EditContent = get(TBHandle, 'String');
    NewBoard = PopupConversion(...
      PopupConversion(EditContent, szBoard), ...
      Board...
```

```
  );
UserData.Dinamic_Analog_Board = NewBoard;
NewFrameHeigth  = eval(...
  [NewBoard '_ParamUI(''VertDim''); ']...
  );

% aggiorno la dimensione della figura
% in base alla nuova dimensione
% del frame (tolgo la dimensione del
% frame vecchio e aggiungo la dimensione
% del frame nuovo)
FigurePosition = get(FigHandle, 'Position');
FigurePosition(4) = FigurePosition(4) - ...
  CurrentFrameHeight + NewFrameHeigth;
set(FigHandle, 'Position', FigurePosition);

NewFrameBottom = UserData.AnStaticCtrlBottom - ...
  CurrentFrameHeight + NewFrameHeigth;

UserData.StaticCtrlBottom = ...
  UserData.StaticCtrlBottom - ...
  CurrentFrameHeight + NewFrameHeigth;

% nel caso in cui la dimensione della
% nuova scheda sia uguale a quella vecchia
% nonè  necessario spostare gli oggetti
% presenti, époich la dimensione
% globale della figura nonè  cambiata
Objs = findobj(FigHandle);
for k = 1:length(Objs)
  ObjPosition = get(Objs(k), 'Position');
  if ObjPosition(2) >= ...
      NewFrameBottom - NewFrameHeigth
    set(Objs(k), ...
      'Position', ...
      ObjPosition - ...
    [0 (CurrentFrameHeight - NewFrameHeigth) 0 0]);
  end
end
% memorizzazione dell'altezza
% della nuova figura
UserData.AnStaticCtrlBottom = NewFrameBottom;
set(FigHandle, 'UserData', UserData);

% creazione del frame relativo
% alla nuova scelta effettuata
eval(...
  [NewBoard ...
    '_ParamUI(''Create'', gcbf, ' ...
```

```
      num2str(UserData.AnStaticCtrlBottom) ');'...
    ]);

case 'PORT'                                                          100
  %è  la callback relativa al popup di
  % scelta del segnale a cui si desidera
  % assegnare il pin della scheda
  BlockData = get(FigHandle, 'UserData');
  PortPinObjHandle = findobj(FigHandle, ...
    'Tag', 'AnPortPin');
  ConnPinObjHandle = findobj(FigHandle, ...
    'Tag', 'AnConnPin');

  % si ricava la posizione del segnale                               110
  % nella lista del popupmenu in cuiè  presente
  PortNumber = get(gcbo, 'Value');
  PortPins ='';

  % in base al numero di pin relativi
  % a quel segnale si
  for k=0:BlockData.Analog_Board_Data.PortWidth(PortNumber)-1
    PCBPinIdx = PopupConversion(...
      strtok(...
      BlockData.Analog_Board_Data.PCBPin{PortNumber}{k+1}, ...       120
      '"'), ...
      BlockData.sim_Conn2PCBPinMapping ...
      );

    if isempty(PCBPinIdx)
      PCBPinIdx = 1;
    end % if ~isempty(PCBPin)

    % si ricava la lista dei pin relativi
    % a quel segnale (dipendente da PortWidth)                       130
    PortPins = [PortPins num2str(k) '  (' ...
        PopupConversion(...
        PCBPinIdx, BlockData.szConn2PCBPinMapping...
        ) ')|'];
  end
  PortPins(end) = ''; % eliminazione ultimo carattere "|"
  set(PortPinObjHandle, ...
    'String', PortPins, ...
    'Value', 1, ...
    'ListBoxTop', 1);                                                140

  % in base al segnale selezionato,
  % si posiziona il selettore del listbox
  % relativo alla lista dei pin
  % della scheda analogica sull'ultimo
```

```
  % pin della scheda assegnato
  % al primo pin disponibile a quel
  % segnale. Nel caso in cui non sia
  % stato assegnato ancora alcun pin
  % si posiziona il selettore sulla                    150
  % prima scelta della lista dei pin
  % della scheda (not used)
  PinPosition = PopupConversion(...
    BlockData.Analog_Board_Data.PCBPin{PortNumber}{1}, ...
    BlockData.sim_Conn2PCBPinMapping ...
    );
  if isempty(PinPosition)
    PinPosition = 1;
  end % if isempty(ListBoxValue)
  set(ConnPinObjHandle, ...                            160
    'Value', PinPosition, ...
    'ListBoxTop', PinPosition);

case 'PORTPIN'

  % vuota ...

  % serve solo per evidenziare nel listbox
  % dei pin di una determinata porta
  % uno dei pin (ùpi pin solo per il                     170
  % caso differenziale) a cui àdovr
  % essere assegnato un pin della scheda
  % specificata in uno dei combobox precedenti.
  % Per tali pin àsar infatti necessario
  % conoscerne la posizione (ricavabile
  % dalla àpropriet value) all'interno del listbox

case 'CONNPIN'
  % permette di assegnare ai pin di una
  % determinata porta (1 solo in generale                180
  % 2 se il segnale di quella portaè
  % differenziale) il pin che à avr in
  % quella determinata scheda analogica
  BlockData = get(FigHandle, 'UserData');

  PortObjHandle = findobj(FigHandle, ...
    'Tag', 'AnPort');
  PortPinObjHandle = findobj(FigHandle, ...
    'Tag', 'AnPortPin');
                                                       190
  PortNumber = get(PortObjHandle, 'Value');
  PinNumber = get(PortPinObjHandle, 'Value');

  % Immagazzinamento del nuovo pin della
```

```
  % scheda assegnato a quel
  % determinato pin di quel
  % determinato segnale
  BlockData.Analog_Board_Data.PCBPin{PortNumber}{PinNumber}=...
    PopupConversion(...
    get(gcbo, 'Value'), BlockData.sim_Conn2PCBPinMapping...
    );

  % aggiornamento del contenuto del listbox
  % contenente l'elenco dei pin per quel
  % determinato segnale: si riporta la
  % lista dei pin del segnale e tra parentesi
  % si indica il pin della scheda analogica
  % cheè  stato assegnato
  PortPins = '';
  for k=0:BlockData.Analog_Board_Data.PortWidth(PortNumber)-1
    PCBPinIdx = PopupConversion( ...
      strtok(...
      BlockData.Analog_Board_Data.PCBPin{PortNumber}{k+1},'"'...
      ), BlockData.sim_Conn2PCBPinMapping ...
      );
    if isempty(PCBPinIdx)
      PCBPinIdx = 1;
    end % if ~isempty(PCBPin)

    PortPins = [PortPins num2str(k) '  (' ...
        PopupConversion(...
        PCBPinIdx, BlockData.szConn2PCBPinMapping ...
        ) ')|'];
  end % for k ...
  PortPins(end) = ''; % eliminazione ultimo carattere "|"
  set(PortPinObjHandle, 'String', PortPins);
  set(FigHandle, 'UserData', BlockData);

case {'ANY PIN','NOT USED'}
  % permette di assegnare a tutti i pin
  % di un determinato segnale
  % il valore "any pin" o "not used"
  BlockData = get(FigHandle, 'UserData');

  PortObjHandle = findobj(FigHandle, ...
    'Tag', 'AnPort');
  PortPinObjHandle = findobj(FigHandle, ...
    'Tag', 'AnPortPin');

  % si converte il numero del pin selezionato
  % nel numero di pin della scheda
  PortNumber = get(PortObjHandle, 'Value');
  PortWidth = ...
```

```
      BlockData.Analog_Board_Data.PortWidth(PortNumber);

    % Store data
    pinId = PopupConversion(...
      PopupConversion(...
      CallbackType, BlockData.szConn2PCBPinMapping ...
      ), ...
      BlockData.sim_Conn2PCBPinMapping ...
      );

    % creazione di un cell array di stringhe contenente
    % in ogni elemento il numero (convertito in stringa)
    % corrispondente ad una delle due opzioni scelte:
    % 0 --> not used
    % -1 --> any pin
    BlockData.Analog_Board_Data.PCBPin{PortNumber} = ...
      cellstr(repmat(num2str(pinId),PortWidth,1))';
    set(FigHandle, 'UserData', BlockData);
    set(gcbo, ...
      'Value', ...
      PopupConversion(...
      CallbackType, ...
      BlockData.szConn2PCBPinMapping...
      ));

    % riaggiornamento del listbox secondo
    % i nuovi valori prescelti
    Analog_Board_Callback(FigHandle, 'ConnPin');

  otherwise
end
```

B.3 *Analog_Board_PinMapDlg.m*

```
function [varargout] = Analog_Board_PinMapDlg(...
  Action, varargin)

% permette di generare e modificare
% la finestra di dialogo della
% scheda analogica scelta. Tra
% l'altro contiene anche informazioni
% sulla dimensione del frame
% della finestra di dialogo.
% A meno di eccezioni tutte le schede                    10
% analogiche selezionabili avranno il
% medesimo box di dialogo, àcambier
% solamente la lista dei pin disponibili
% per quella scheda

global sim___developper

switch upper(Action)
  case 'VERTDIM'
    % se serve solo la dimensione del                    20
    % frame nonè  necessario
    % proseguire con il resto
    % della funzione
    varargout{1} = 100; %115
    return;
end % switch upper(Action)

% dalla figura si ricava l'handle
% del blocco, necessario per ricavare le
% informazioni memorizzate nella maschera                30
FigHandle = varargin{1};
BlockHandle = getfield(...
  get(FigHandle, 'UserData'), ...
  'BlockHandle' ...
  );

% tipo di blocco esteso
ExtendedBlockType = get_param(BlockHandle, ...
  'sim_ExtendedBlockType');
                                                         40
switch upper(Action)
  case 'CREATE'
    Position = varargin{2};

    DiagramHandle = bdroot(BlockHandle);

    % si cerca all'interno del diagramma
```

```
% il blocco dighwinterface, in
% caso non sia presente viene
% restituito un vettore vuoto.
DigHwInterface = get_digHwInterface(BlockHandle);

if ~isempty(DigHwInterface)
  if nargin == 4
    Analog_Board = varargin{3};
  else
    try
      Analog_Board = eval_param(...
        DigHwInterface, 'sim_Analog_Type' ...
        );
    catch
      errordlg(['Cannot extract information on ' ...
          'Analog board from digHwInterface ' ...
          'block; target assumed as ''none'''], ...
        'Missing digHwInterface block', 'non-modal');
      Analog_Board = 'none';
    end % try
  end % if nargin == 4
else
  errordlg(['Cannot extract information on ' ...
      'target board because digHwInterface ' ...
      'block is not present into the block diagram; ' ...
      'target assumed as ''none'''], ...
    'Missing digHwInterface block', 'non-modal');
  Analog_Board = 'none';
end % if ~isempty(DigHwInterface)

% si ricavano le liste delle schede
% disponibili: in àrealt sono la
% medesima lista ma unaè  quella
% visualizzata nel popup menu,
% l'altraè  quella che contiene i
% nomi di riferimento per la ricerca
% all'interno di tale lista
szConn2PCBPinMapping = []; sim_Conn2PCBPinMapping = [];
eval(...
  ['[szConn2PCBPinMapping sim_Conn2PCBPinMapping] = ' ...
    '' Analog_Board '_Pin_Mapping;'] ...
  );

% inizializzazione della struttura
% relativa alla scheda
Analog_Board_Data = '';

% si ricavano informazioni sui segnali
% principali e sul numero di
```

```
% pin che servono per descriverli
Ports = []; PortsWidth = [];
eval( ...
  ['[Ports, PortsWidth] = ' ...                                    100
    ExtendedBlockType ...
  '_ParamUI(''AnalogPorts'',BlockHandle);']...
  );

ValidPins = sim_Conn2PCBPinMapping;
NPorts = length(find(Ports == '|')) + 1;

% Initialize the string to be put into
% the Port popup menu
szPorts = '';                                                      110

for k = 1:NPorts
  % per ogni segnale si ricava
  % il nome e il numero di pin
  [Port, Ports] = strtok(Ports, '|');
  [PortWidth, PortsWidth] = strtok(PortsWidth, '|');

  % costruzione dell'elenco dei segnali
  % da inserire nel popup menu
  szPorts = [szPorts Port '|'];                                    120

  Analog_Board_Data.Port{k} = Port;
  Analog_Board_Data.PortWidth(k) = str2num(PortWidth);

  try
    get_param(BlockHandle, Port);
  catch
    % nel caso in cui la maschera
    % non sia àgi stata aggiornata,
    % si aggiungono i parametri contenenti                         130
    % il nome del segnale e il riferimento ai
    % pin che lo descrivono, inizializzati
    % come "not used" --> "0"
    add_blockParam(BlockHandle, Port, '&', Port);
    InitPin = '"0"';
    FinalPin = InitPin;
    for m = 1:Analog_Board_Data.PortWidth(k)-1
      FinalPin = [FinalPin ',' InitPin];
    end
    add_blockParam(BlockHandle, ...                                140
      ['pinMap_', Port], '&', FinalPin);
  end

  PinMapParam = get_param(...
    BlockHandle, ['pinMap_' Port] ...
```

```
    );

  % nel caso tipico al segnaleè
  % associato un solo pinè,
  % sufficiente recuperare                                          150
  % l'informazione sul riferimento a quel
  % determinato pin
  if Analog_Board_Data.PortWidth(k) == 1
    Tok= strtok(PinMapParam, '"');
    Analog_Board_Data.PCBPin{k}{1} = Tok;
  else
    % nel caso in cui siano presenti
    % il segnale sia associato a ùpi pin,
    % si ricavano i riferimenti nell'elenco
    % a tutti i pin presenti e li si memorizza                      160
    % nella cella PCBPin della struttura
    % Analog_Board_Data
    if strfind(PinMapParam, ')')
      PinMapParam(1) = ''; PinMapParam(end) = '';
    end %strfind(PinMapParam...

    for m = 1:Analog_Board_Data.PortWidth(k)
      [Tok, PinMapParam] = strtok(PinMapParam, ' ');
      Analog_Board_Data.PCBPin{k}{m} = strtok(Tok,'"');
    end % for m                                                     170
  end
end % for k = 1:NPorts

% cancellazione ultimo carattere "|"
szPorts(end) = '';

% si ricava la dimensione del
% frame dipendente dalla scheda
% selezionata
FrameHeight = Analog_Board_PinMapDlg(...                          180
  'VERTDIM', FigHandle ...
);

Position = Position - 5;

PutFrame(FigHandle, ...
  Position - FrameHeight, ...
  FrameHeight, ...
  'Pin mapping (AnHw)', 90);
                                                                  190
LastCtrlBottom = Position - 20;

% inserimento textbox che si riferiscono
% agli oggetti di controllo da deporre
```

```
InsertTextBox(FigHandle, ...
  [10 LastCtrlBottom 90 13], 'Cell port (AnHw)');
InsertTextBox(FigHandle, ...
  [180 LastCtrlBottom 80 13], 'Port pin (AnHw)');
InsertTextBox(FigHandle, ...
  [265 LastCtrlBottom 70 13], 'Conn pin (AnHw)');

LastCtrlBottom = LastCtrlBottom - 15;

BlockData = get(FigHandle, 'UserData');

% inserimento popup menu con la
% lista dei segnali e assegnazione
% della relativa callback
CellPortPopupHandle = InsertPopupMenu(FigHandle, ...
  [10 LastCtrlBottom 90 15], szPorts, 'AnPort');
set(CellPortPopupHandle, 'Callback', ...
  [get(CellPortPopupHandle, 'Callback') ...
    'Analog_Board_Callback(gcbf, ''Port''); %']);

PortPins = '';
for k = 0:Analog_Board_Data.PortWidth(1)-1
  PCBPinIdx = PopupConversion(...
    strtok(...
    Analog_Board_Data.PCBPin{1}{k+1}, '"' ...
    ), sim_Conn2PCBPinMapping);
  if isempty(PCBPinIdx)
    PCBPinIdx = 1;
  end
  PortPins = [PortPins num2str(k) '  (' ...
      PopupConversion(PCBPinIdx, szConn2PCBPinMapping) ...
      ')|'];
end %k = 0:Analog_Board_Data.PortWidth(1)-1

% si elimina l'ultimo carettere "|"
PortPins(end) = '';

% inserimnto listbox con la lista
% dei pin relativi ai segnali
% inseriti nel popup menu: essendo
% la maggior parte dei segnali
% single ended, il numero dei pin
% àsar quasi sempre 1
PortPinListHandle = InsertListBox(FigHandle, ...
  [180 LastCtrlBottom-55 80 70], ...
  PortPins, 1, 'AnPortPin');
set(PortPinListHandle, 'Callback', ...
  [get(PortPinListHandle, 'Callback') ...
    'Analog_Board_Callback(gcbf,''PortPin''); %']);
```

```
ListBoxValue = PopupConversion(...
  Analog_Board_Data.PCBPin{1}{1}, ...
  sim_Conn2PCBPinMapping...
  );
if isempty(ListBoxValue)
  ListBoxValue = 1;
end % if isempty(ListBoxValue)

% inserimento listbox contenente la
% lista dei pin disponibili per
% una determinata scheda e assegnazione
% della relativa callback
ConnectorPinListHandle = InsertListBox(...
  FigHandle, [265 LastCtrlBottom-55 70 70], ...
  szConn2PCBPinMapping, ListBoxValue, 'AnConnPin');
set(ConnectorPinListHandle, 'Callback', ...
  [get(ConnectorPinListHandle, 'Callback') ...
    'Analog_Board_Callback(gcbf, ''ConnPin''); %']);

ApplyButtonCallback = ...
  ['ApplyButtonHandle = findobj(' ...
    'gcf, ''Tag'', ''Apply_PB''); ' ...
    'set(ApplyButtonHandle, ''Enable'', ''on''); ' ...
    'clear ApplyButtonHandle;'];

% aggiunta dei pulsanti di assegnazione
% di "any pin" e "not used"
uicontrol('Parent', FigHandle, ...
  'Units', 'points', ...
  'Callback', [ApplyButtonCallback ...
    ' Analog_Board_Callback(gcbf, ''Not used''); '],...
  'Position', [105 LastCtrlBottom-22 70 18], ...
  'Style', 'pushbutton', ...
  'String', 'Set all ''Not used''', ...
  'Tag', 'ALL_PBN_AN');

uicontrol('Parent', FigHandle, ...
  'Units', 'points', ...
  'Callback', [ApplyButtonCallback  ...
    ' Analog_Board_Callback(gcbf,''Any pin''); '],...
  'Position', [105 LastCtrlBottom-44 70 18], ...
  'Style', 'pushbutton', ...
  'String', 'Set all ''Any pin''', ...
  'Tag', 'ALL_PBT_AN');

LastCtrlBottom = LastCtrlBottom - 5;
```

```
  % memorizzazione della struttura dati
  % con le informazioni sulla
  % nuova scheda appena ricavate
  BlockData = get(FigHandle, 'UserData');
  BlockData.Analog_Board_Data = Analog_Board_Data;
  BlockData.szConn2PCBPinMapping = ...
    szConn2PCBPinMapping;
  BlockData.sim_Conn2PCBPinMapping = ...                          300
    sim_Conn2PCBPinMapping;
  set(FigHandle, 'UserData', BlockData);

case 'STORE'

  % immagazzinamento dei dati selezionati
  % nella finestra di dialogo.
  % viene chiamata in seguito alla
  % pressione del pulsante Apply
  BlockHandle = getfield(...                                      310
    get(FigHandle, 'UserData'), 'BlockHandle'...
    );
  ExtendedBlockType = get_param(...
    BlockHandle, 'sim_ExtendedBlockType' ...
    );
  Analog_Board_Data = getfield(...
    get(FigHandle, 'UserData'), 'Analog_Board_Data' ...
    );
  for k = 1:length(Analog_Board_Data.Port)
    if Analog_Board_Data.PortWidth(k) == 1                        320
      % caso tipico, segnali ad un solo pin
      % si verifica
      Ports = []; PortsWidth = [];
      eval( ...
        ['[Ports, PortsWidth] = ' ...
          ExtendedBlockType  ...
        '_ParamUI(''AnalogPorts'');'] ...
        );
      Port = '';
      while (~strcmp(Port, Analog_Board_Data.Port{k}))            330
        [Port, Ports] = strtok(Ports, '|');
        [PortWidth, PortsWidth] = strtok(PortsWidth, '|');

        % se la portaè  una stringa vuota
        % significa che il parametro non
        % era incluso nella maschera
        if isempty(Port)
          error(...
            ['Port ' Analog_Board_Data.Port{k} ...
              ' is not specified into ' ...                       340
              get_param(BlockHandle, 'Name') ...
```

```
            ' block''s mask function. ' ...
            'Compilation can''t continue.']);
        end % if isempty(Port)
      end % while (~strcmp(Port, ...
      PinArray = ['"' Analog_Board_Data.PCBPin{k}{1} '"'];
    else
      % caso particolare segnali differenziali
      % a 2 pin, la lista dei pinè  inclusa
      % fra parentesi tonde                                          350
      PinArray = '(';
      for m = 1:Analog_Board_Data.PortWidth(k)
        PinArray = [PinArray ...
            '"' Analog_Board_Data.PCBPin{k}{m} ...
            '", '];
      end % for m = 1:Target_Board_Data.PortWidth(k)

      % l'ultimo carattereè  la chiusura
      % della parentesi tonda
      PinArray(end) = ''; PinArray(end) = ')';                     360
    end % if Analog_Board_Data.PortWidth(k) == 1

    set_param(BlockHandle, ...
      ['pinMap_' Analog_Board_Data.Port{k}], PinArray);

  end % for k = 1:length(Analog_Board_Data.Port)

case 'DELETE'

  % si ricava la struttura dati                                    370
  % in figura e si rimuovono i campi
  % relativi alla precedente scheda analogica
  BlockData = get(FigHandle, 'UserData');
  BlockData = rmfield(BlockData, 'Analog_Board_Data');
  BlockData = rmfield(BlockData, 'szConn2PCBPinMapping');
  BlockData = rmfield(BlockData, 'sim_Conn2PCBPinMapping');
  set(FigHandle, 'UserData', BlockData);

  % si candella il frame chiamato Pin mapping
  % (AnHw) della precedente scheda analogica                       380
  DeleteFrame(FigHandle, 'Pin mapping (AnHw)');

  % cancellazione dei textbox
  delete(...
    findobj(FigHandle, ...
    'Style', 'text', 'String', 'Cell port (AnHw)'));
  delete(...
    findobj(FigHandle, ...
    'Style', 'text', 'String', 'Port pin (AnHw)'));
  delete(...                                                       390
```

```
      findobj(FigHandle, ...
      'Style', 'text', 'String', 'Conn pin (AnHw)'));

    % cancellazione dei controlli
    % (listbox, popupmenu e pulsanti)
    delete(...
      findobj(FigHandle, ...
      'Style', 'popup', 'Tag', 'AnPort'));
    delete(...
      findobj(FigHandle, ...                                          400
      'Style', 'listbox', 'Tag', 'AnPortPin'));
    delete(...
      findobj(FigHandle, ...
      'Style', 'listbox', 'Tag', 'AnConnPin'));
    delete(...
      findobj(FigHandle, ...
      'Style', 'pushbutton', 'Tag', 'ALL_PBN_AN'));
    delete(...
      findobj(FigHandle, ...
      'Style', 'pushbutton', 'Tag', 'ALL_PBT_AN'));                   410

  otherwise
    error(...
      ['Action ' Action ...
        ' not allowed for function Target_Board_PinMapDlg()']);
end % switch upper(Action)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                                                      420
% LOCAL FUNCTIONS

function InsertTextBox(FigHandle, Position, Label)
% inserisce un textbox
uicontrol('Parent', FigHandle, ...
  'Units', 'points', ...
  'BackgroundColor', [.8 .8 .8], ...
  'HorizontalAlignment', 'left', ...
  'Position', Position, ...
  'String', Label, ...                                                430
  'Style', 'text');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function ControlHandle = InsertPopupMenu(FigHandle, ...
  Position, MenuItemString, Tag)
% inserisce un popup menu
ControlHandle = uicontrol('Parent', FigHandle, ...
  'Units', 'points', ...
```

```
  'BackgroundColor', [1 1 1], ...
  'HorizontalAlignment', 'left', ...
  'Position', Position, ...
  'String', MenuItemString, ...
  'Style', 'popupmenu', ...
  'Tag', Tag, ...
  'Value', 1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function ControlHandle = InsertEditBox(FigHandle, Position, Tag)
% inserisce un edit box
ControlHandle = uicontrol('Parent', FigHandle, ...
  'Units', 'points', ...
  'BackgroundColor', [1 1 1], ...
  'HorizontalAlignment', 'left', ...
  'Position', Position, ...
  'Style', 'edit', ...
  'Tag', Tag);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function ControlHandle = InsertListBox(FigHandle, ...
  Position, MenuItemString, Value, Tag)
% inserisce un listbox
ControlHandle = uicontrol('Parent', FigHandle, ...
  'Units', 'points', ...
  'BackgroundColor', [1 1 1], ...
  'HorizontalAlignment', 'left', ...
  'Position', Position, ...
  'String', MenuItemString, ...
  'Style', 'listbox', ...
  'Tag', Tag, ...
  'Value', Value, ...
  'ListBoxTop', Value);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

B.4 *AnalogCircuitDescription.m*

```
function [varargout] = AnalogCircuitDescription(Option, ...
  FigHandle, varargin)

% permette di descrivere la finestra di dialogo
% relativa  alla descrizione analogica
% del circuito.
varargout = cell(1, nargout);
EmptyString = '';

TotalControls = 0;
NumOfPassCompParam = 0;
NumOfPassCompType = 0;

PassCompInfoStr = '';
PassCompVarStr = '';

BlockData = get(FigHandle,'Userdata');
BlockHandle = getfield( ...
  BlockData, 'BlockHandle');
ExtendedBlockType = get_param(...
  BlockHandle, 'sim_ExtendedBlockType');

try
  % si prova ad estrarre le stringhe
  % contenenti le informazioni sui
  % controlli da inserire nel frame
  % per descrivere i componenti passivi
  % del circuito. Nel caso in cui non
  % sia presente l'opzione
  % PassCompParam nella relativa
  % funzione _paramUI del blocco, il numero
  % dei controlliè  da considerarsi nullo
  [PassCompInfoStr, PassCompVarStr] = eval(...
    [ExtendedBlockType ...
      '_ParamUI(''PassCompParam'', BlockHandle);'] ...
    );
  NumOfPassCompParam = length( ...
    strfind(PassCompVarStr, '|') ...
    )+1;

  TotalControls = NumOfPassCompParam;

  OrigPassCompStr = eval(...
    [ExtendedBlockType ...
      '_ParamUI(''PassCompOfCircuit'', BlockHandle);'], ...
    'EmptyString' ...
    );
```

```
  PassCompStr = OrigPassCompStr;
  NumOfPassCompType = length(strfind(PassCompStr, '|'))+1;
  for i = 1:NumOfPassCompType
    [AnalyzedPassComp, PassCompStr] = strtok(PassCompStr, '|');
    [TextObj, InfoObj] = AnHwPassCompSpec(AnalyzedPassComp);
    TotalControls = TotalControls + ...
      length(strfind(TextObj,'|'))+1;
  end
end


switch upper(Option)
  case 'CONTROLSINSIDE',
    % nel caso in cui sia solo necessario
    % conoscere il numero degli oggetti di
    % controllo interni al frame dei
    % componenti passivi
    varargout{1} = TotalControls;
    return;

  case 'GENERATE',
    % generazione del frame della finestra
    % di dialogo relativo ai componenti
    % passivi (resistenze, àcapacit ...)
    % associate alla descrizione analogica
    % di quel blocco. Nel caso in cui non
    % siano presenti tali parametri, il
    % frame non viene generato

    LastCtrlBottom = varargin{1};
    FrameHeight = varargin{2};
    ControlHeight = varargin{3};

    % se non ci sono parametri dei
    % componenti passivi da inserire
    % nella finestra, il frame
    % non àverr generato
    if TotalControls
      PutFrame(...
        FigHandle, LastCtrlBottom-FrameHeight-5, ...
        FrameHeight, 'Passive Component', 80);
      LastCtrlBottom1 = LastCtrlBottom - 25;
    else
      varargout{1} = LastCtrlBottom-5;
      return;
    end

    % aggiunta degli eventuali parametri
    % dei componenti passivi che
```

```
% descrivono il circuito
for k = 1:NumOfPassCompParam
  [TextInfoBox, PassCompInfoStr]= ...
    strtok(PassCompInfoStr, '|');                                    100
  [ObjectTag, PassCompVarStr]= ...
    strtok(PassCompVarStr, '|');
  CtrlHandle = PutTextBox2(FigHandle, ...
    LastCtrlBottom1, TextInfoBox, ...
    ['sim_' ObjectTag], ['sim_' ObjectTag]);
  LastCtrlBottom1 = LastCtrlBottom1 - ...
    ControlHeight - 5;
end

for i = 1:NumOfPassCompType                                          110
  [AnalyzedPassComp, OrigPassCompStr] = ...
    strtok(OrigPassCompStr, '|');
  [PopupMenuHandle, PushButtonHandle] = PutPopupPush(...
    FigHandle, ...
    LastCtrlBottom1, AnalyzedPassComp, ...
    ['sim_' AnalyzedPassComp '_Selected'], ...
    ['sim_' AnalyzedPassComp '_Selected'], ...
    ...
    ['sim_' AnalyzedPassComp '_Description'], ...
    ['sim_' AnalyzedPassComp '_Description'], ...                    120
    ...
    ['sim_' AnalyzedPassComp '_List'], ...
    ['sim_' AnalyzedPassComp '_List'], ...
    ...
    ['sim_' AnalyzedPassComp '_Indices'], ...
    ['sim_' AnalyzedPassComp '_Indices'] ...
    );

  set(PopupMenuHandle, ...
    'Callback', ['AllComponentSearch(''CompList'', gcbf);']);        130
  set(PushButtonHandle, ...
    'Callback', ['AnDescriptionChange; AddStoreParameter; '...
      'AllComponentSearch(''CheapComp'', gcbf);']);

  [TextSpecStr, InfoSpecStr] = ...
    AnHwPassCompSpec(AnalyzedPassComp);
  while(~isempty(TextSpecStr))
    [TextSpec, TextSpecStr]= strtok(TextSpecStr, '|');
    [ObjectTag, InfoSpecStr]= strtok(InfoSpecStr, '|');
    CtrlHandle = PutTextBox2(FigHandle, ...                          140
      LastCtrlBottom1, TextSpec, ...
      ['sim_' AnalyzedPassComp '_' ObjectTag], ...
      ['sim_' AnalyzedPassComp '_' ObjectTag]);
    LastCtrlBottom1 = LastCtrlBottom1 - ControlHeight - 5;
  end
```

```
  end

  varargout{1} = LastCtrlBottom - FrameHeight - 5;

case 'UPDATE',

  switch nargin
    case 2
      %%%%%%%%%% TUTTI a -???-%%%%%%%%
      UnknownData  = '-???-';
      PassCompStruct = AnalogCircuitDescription(...
        'PassComp_Structure',FigHandle ...
      );
      UpdatePassCompStruct(FigHandle, PassCompStruct);

      NumOfPassCompType = length(strfind(OrigPassCompStr, '|'))+1;
      for i = 1:NumOfPassCompType
        [AnalyzedPassComp, OrigPassCompStr] = ...
          strtok(OrigPassCompStr, '|');
        try
          set_PopupMenu_value(FigHandle, ...
            UnknownData, ...
            ['sim_' AnalyzedPassComp '_Selected'], ...
            ...
            UnknownData, ...
            ['sim_' AnalyzedPassComp '_Description'], ...
            ...
            UnknownData, ...
            ['sim_' AnalyzedPassComp '_List'], ...
            ...
            UnknownData,...
            ['sim_' AnalyzedPassComp '_Indices']);

          UpdatePassCompSpec(FigHandle, ...
            AnalyzedPassComp, UnknownData, UnknownData);
        end
      end
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    case 3
      %%%%%%%%%%%%%% SOLO L'ELENCO DEI COMPONENTI
      PassCompAlgResults = varargin{1};
      UpdatePassCompStruct(FigHandle, PassCompAlgResults);
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    case 5
      %%%%% SOLO LE SPECIFICHE DEI COMPONENTI PASSIVI
      AnalyzedPassComp = varargin{1};
      PassCompRow = varargin{2};
      NamesOfColumns = varargin{3};
```

```
        UpdatePassCompSpec(FigHandle, ...
          AnalyzedPassComp, PassCompRow, NamesOfColumns);
    end

  case 'DELETE',                                                    200
    % elimina tutti gli oggetti inclusi nel
    % frame che descrive i componenti passivi
    % (resistenze, àcapacit ...) utilizzati
    % nella descrizione del circuito analogico
    % del blocco simulink

    if ~isempty(OrigPassCompStr)
      DeleteFrame(FigHandle, 'Passive Component');
      delete(...
        findobj(FigHandle, 'Style', 'text'));                       210
      delete(...
        findobj(FigHandle, 'Style', 'popupmenu'));
      delete(...
        findobj(FigHandle, ...
        'Style', 'pushbutton', ...
        'String', 'Cheap'));
    end

  case 'PASSCOMP_STRUCTURE'
    % creazione di una struttura di dimensione                    220
    % variabile a seconda del tipo di blocco
    % esteso, contenente tutti i parametri dei
    % componenti passivi che verranno aggiunti
    % nell'interfaccia per quel determinato blocco
    PassCompStruct = [];
    for i = 1:NumOfPassCompParam
      [PassCompParam, PassCompVarStr] = ...
        strtok(PassCompVarStr,'|');
      eval(...
        ['PassCompStruct.',PassCompParam,'=''-???-'';']...          230
        );
    end
    % restituisce una struttura inizializzata
    % che contiene tutte le variabili relative
    % ai componenti passivi
    varargout{1} = PassCompStruct;

  otherwise

end                                                                 240

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function [varargout] = UpdatePassCompStruct(FigHandle, ...
  PassCompStruct, varargin)

BlockData = get(FigHandle,'Userdata');
BlockHandle = getfield(BlockData, 'BlockHandle');

Fields = fieldnames(PassCompStruct);
% NumOfPassCompParam = length(Fields);
for k = 1:length(Fields)
  ObjectTag = Fields{k};
  Param = getfield(PassCompStruct, ObjectTag);
  ParamTag = ['sim_' ObjectTag];
  set_param(BlockHandle, ParamTag, Param);
  BlockData = setfield(BlockData, ParamTag, Param);
  set(...
    findobj(FigHandle, 'Tag', ParamTag), 'String', Param);
end %for k = 1: ...

set(FigHandle, 'Userdata', BlockData);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [varargout] = UpdatePassCompSpec(FigHandle, ...
  AnalyzedPassComp, PassCompRow, NamesOfColumns, varargin)

BlockData = get(FigHandle,'Userdata');
BlockHandle = getfield(BlockData, 'BlockHandle');

[PassCompSpecInfoStr, PassCompSpecVarStr] = ...
  AnHwPassCompSpec(AnalyzedPassComp);
NumOfPassCompSpec = length(...
  strfind(PassCompSpecVarStr,'|')...
  )+1;
for i = 1 : NumOfPassCompSpec
  [ObjectTag, PassCompSpecVarStr] = ...
    strtok(PassCompSpecVarStr, '|');
  if iscell(PassCompRow)
    Param = num2str(...
      PassCompRow{strmatch(ObjectTag, NamesOfColumns)}...
      );
  else
    Param = PassCompRow; % vale '-???-'
  end
  ParamTag = ['sim_' AnalyzedPassComp '_' ObjectTag];
  set_param(BlockHandle, ParamTag, Param);
  BlockData = setfield(BlockData,ParamTag, Param);
  set(findobj(FigHandle, ...
    'Tag', ParamTag), ...
    'String', Param);
end % for i
```

```
set(FigHandle, 'Userdata', BlockData);
```

B.5 *AnalogDeviceDescription.m*

```
function [varargout] = AnalogDeviceDescription(Option, ...
  FigHandle,varargin)

% questa funzione permette di descrivere
% ll frame della finestra di dialogo
% che contiene le informazioni sui
% dispositivi che descrivono il circuito
% analogico
BlockData = get(FigHandle, 'UserData');
BlockHandle = getfield(BlockData, 'BlockHandle');                    10
ExtendedBlockType = get_param(BlockHandle, ...
  'sim_ExtendedBlockType');
EmptyStr = '';
TotalControls = 0;

OrigDeviceListStr = eval( ...
  [ExtendedBlockType ...
    '_ParamUI(''DeviceOfCircuit'', BlockHandle);'], ...
  'EmptyStr' ...
  );                                                                 20
DeviceListStr = OrigDeviceListStr;

while(~isempty(DeviceListStr))
  [AnalyzedDevice, DeviceListStr] = ...
    strtok(DeviceListStr, '|');
  [DevParamInfoStr,DevParamVarStr] =  ...
    AnHwDevParameters(AnalyzedDevice);
  if DevParamVarStr
    DevParamNumber = length(strfind(DevParamVarStr, '|'))+1;
  else                                                               30
    DevParamNumber = 0;
  end
  TotalControls = TotalControls + DevParamNumber;
end


switch upper(Option)
  case 'CONTROLSINSIDE'
    % permette di calcolare il numero di
    % oggetti di controllo (textbox)                                 40
    % da inserire nel frame (o nei frame,
    % a seconda del numero) del
    % dispositivo (dei dispositivi) che
    % descrivono il circuito
    % analogico di un determinato blocco
    % esteso. nel caso in cui non
    % siano presenti dispositivi,
```

```
  % viene restituito 0.

  varargout{1} = TotalControls;

case 'GENERATE',
  % permette di creare un frame nella
  % finestra di dialogo che
  % contenga le informazioni relative
  % al dispoitivo, tra cui l'elenco
  % dei dispositivi che soddisfano le
  % specifiche immesse dall'utente
  % e i parametri principali (
  % contenuti nel database) che descrivono
  % quel dispositivo
  LastCtrlBottom = varargin{1};
  FrameHeight = varargin{2};
  ControlHeight = varargin{3};

  if TotalControls
    PutFrame(FigHandle, ...
      LastCtrlBottom-FrameHeight-5, ...
      FrameHeight, 'Active Component', 80);
    LastCtrlBottom = LastCtrlBottom - 25;
  end

  while ~isempty(OrigDeviceListStr)
    [AnalyzedDevice, OrigDeviceListStr] = ...
      strtok(OrigDeviceListStr,'|');

    % per un determinato dispositivo pongo
    % il listbox contenente
    % l'elenco dei dispositivi del
    % database che soddisfano determinate
    % specifiche inserite dall'utente
    [PopupMenuHandle, PushButtonHandle] = ...
      PutPopupPush(FigHandle, ...
      LastCtrlBottom, AnalyzedDevice, ...
      ['sim_' AnalyzedDevice '_Selected'], ...
      ['sim_' AnalyzedDevice '_Selected'], ...
      ...
      ['sim_' AnalyzedDevice '_Description'], ...
      ['sim_' AnalyzedDevice '_Description'], ...
      ...
      ['sim_' AnalyzedDevice '_List'], ...
      ['sim_' AnalyzedDevice '_List'], ...
      ...
      ['sim_' AnalyzedDevice '_Indices'], ...
      ['sim_' AnalyzedDevice '_Indices']);
```

```
    % si aggiungono le callback per la
    % selezione degli oggetti elencati
    % nel listbox e per la pressione
    % del pulsante cheap                                          100
    set(PopupMenuHandle, ...
      'Callback', ...
      ['AllComponentSearch(''CompList'', gcbf);']);
    set(PushButtonHandle, ...
      'Callback', ...
      ['AnDescriptionChange; AddStoreParameter; '...
        'AllComponentSearch(''CheapComp'', gcbf);']);

    [DevParamInfoStr, DevParamVarStr] = ...
      AnHwDevParameters(AnalyzedDevice);                          110
    NumOfDevParam = length(strfind(DevParamVarStr, '|'))+1;
    for i = 1 : NumOfDevParam
      [TextInBox, DevParamInfoStr]= ...
        strtok(DevParamInfoStr, '|');
      [ObjectTag, DevParamVarStr]= ...
        strtok(DevParamVarStr, '|');
      CtrlHandle = PutTextBox2(FigHandle, ...
        LastCtrlBottom, TextInBox, ...
        ['sim_' AnalyzedDevice '_' ObjectTag], ...
        ['sim_' AnalyzedDevice '_' ObjectTag]);                   120
      LastCtrlBottom = LastCtrlBottom - ControlHeight - 5;
    end %for i = ...
  end %while ~isempty(


  varargout{1} = LastCtrlBottom;

case 'DELETE'
  % elimina tutti gli oggetti inclusi
  % nei frame che descrivono i                                    130
  % componenti attivi (i dispositivi)
  % utilizzati nella descrizione
  % del circuito analogico
  % del blocco simulink

  if ~isempty(OrigDeviceListStr)
    DeleteFrame(FigHandle, 'Active Component');
    delete(...
      findobj(FigHandle, 'Style', 'text'));
    delete(...                                                    140
      findobj(FigHandle, ...
      'Style', 'pushbutton', ...
      'String', 'Cheap')...
      );
  end
```

```
  case 'UPDATE',
    % aggiornamento del frame precedentemente
    % generato per quello specifico dispositivo.
    % Si ricorda che, come nel caso dei                                    150
    % parametri relativi ai componenti passivi
    %è  sufficente aggiornare il textbox contenente
    % il valore del parametro senza modificare il
    % textbox che lo etichetta.

    switch nargin
      case 2
        UnknownData = '-???-';
        for j = 1:length(strfind(OrigDeviceListStr, '|'))+1
          [AnalyzedDevice, OrigDeviceListStr] = ...                      160
            strtok(OrigDeviceListStr, '|');
          try
            set_PopupMenu_value(FigHandle, ...
              UnknownData, ...
              ['sim_' AnalyzedDevice '_Selected'], ...
              ...
              UnknownData, ...
              ['sim_' AnalyzedDevice '_Description'], ...
              ...
              UnknownData, ...                                           170
              ['sim_' AnalyzedDevice '_List'], ...
              ...
              UnknownData, ...
              ['sim_' AnalyzedDevice '_Indices']);

            UpdateDevParam(FigHandle, ...
              AnalyzedDevice, UnknownData, UnknownData);
          end
        end
      case 5                                                             180
        AnalyzedDevice = varargin{1};
        DeviceRow = varargin{2};
        NamesOfColumns = varargin{3};

        UpdateDevParam(FigHandle, ...
          AnalyzedDevice, DeviceRow, NamesOfColumns);
    end

  otherwise
end                                                                      190

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% LOCAL FUNCTION
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function [varargout] = UpdateDevParam(FigHandle, ...
  AnalyzedDevice, DeviceRow, NamesOfColumns, varargin)

BlockData = get(FigHandle,'Userdata');
BlockHandle = getfield(BlockData, 'BlockHandle');                    200

[DevParamInfoStr,DevParamVarStr] = ...
  AnHwDevParameters(AnalyzedDevice);
NumOfDevParam = length(strfind(DevParamVarStr,'|'))+1;
for k = 1:NumOfDevParam
  [ObjectTag,DevParamVarStr] = ...
    strtok(DevParamVarStr, '|');
  if iscell(DeviceRow)
    Param = num2str(...
      DeviceRow{strmatch(ObjectTag, NamesOfColumns)}...              210
      );
  else
    Param = DeviceRow; % vale '-???-'
  end
  ParamTag = ['sim_' AnalyzedDevice '_' ObjectTag];
  set_param(BlockHandle, ParamTag, Param);
  BlockData = setfield(BlockData,ParamTag, Param);
  set(...
    findobj(FigHandle, 'Tag', ParamTag), 'String', Param ...
  );                                                                 220
end

set(FigHandle, 'Userdata', BlockData);
```

B.6 *AnalogSourceSignal.m*

```
function [varargout] = AnalogSourceSignal(DiagramHandle, ...
  varargin)

% questa funzione in fase di precompilazione
% assegna il segnale memorizzato nei
% blocchi To Workspace ai blocchi
% sim_analogIn a cuiè  collegato

AnalogInHandle = find_system(DiagramHandle, ...
  'sim_ExtendeBlockType', 'sim_analogIn');

StartTime = str2num(get_param(DiagramHandle, 'StartTime'));
StopTime = str2num(get_param(DiagramHandle, 'StopTime'));

for i = 1:length(AnalogInHandle)
  ActualAnalogIn = AnalogInHandle(i);
  PortConn = get_param(ActualAnalogIn, ...
    'PortConnectivity');
  for j = 1: length(PortConn)
    SourceBlock = PortConn(j).SrcBlock;
    SourceConn = get_param(SourceBlock, ...
      'PortConnectivity');

    if isempty(SourceConn)
      continue;
    end;

    DstOfSource = SourceConn.DstBlock;
    TypesOfBlocks = get_param(DstOfSource, 'BlockType');
    Index = strmatch('ToWorkspace',TypesOfBlocks);
    if Index & length(Index)==1
      try
        VarName =  get_param(DstOfSource(Index), ...
          'VariableName');
        y = evalin('base', [VarName '.signals.values']);
        dim_y = length(y);
        y_sequence = reshape(y, [1 dim_y]);
        t_sequence = linspace(StartTime, StopTime, dim_y);

        set_param(ActualAnalogIn, ...
          'sim_time_sequence', num2str(t_sequence));
        set_param(ActualAnalogIn, ...
          'sim_value_sequence', num2str(y_sequence));
      catch
        disp(sprintf(...
          '\n### No data extracted from ''%s'' Block...\n', ...
          get_param(DstOfSource(Index), 'Name')));
```

```
        set_param(ActualAnalogIn, 'sim_time_sequence', '');
        set_param(ActualAnalogIn, 'sim_value_sequence', '');
      end                                                          50
    else
      disp(sprintf(...
        ['\n### No ToWorkspace Block found' ...
          ' in diagram for ''%s'' block...\n'], ...
        get_param(ActualAnalogIn, 'Name')));
      set_param(ActualAnalogIn, 'sim_time_sequence', '');
      set_param(ActualAnalogIn, 'sim_value_sequence', '');
    end
  end % for j
end % for i                                                        60
```

B.7 *AnDescriptionChange.m*

```
function [varargout] = AnDescriptionChange(varargin)

% nel caso in cui un blocco abbia ùpi
% descrizioni circuitali analogiche
% a seconda del valore di alcuni suoi
% parametri (ad esempio sim gain
% òpu essere visto come un amplificatore
% operazionale invertente se Gain<0
% o un operazionale non invertente
% se Gain >1), nonè  detto che la finestra
% di dialogo di selezione dei componenti
% analogici sia la stessa (anche se
% si riferisce allo stesso blocco).
% In questo caso àsar necessario
% effettuare un "refresh" della finestra
% di dialogo adattandola alla nuova
% descrizione del circuito analogico
% del blocco questa funzione rappresenta
% una callback specifica dei pulsanti find
% component echeap component presenti nella
% finestra di dialogo per la
% scelta dei componenti analogici
% ùpi adatti a rappresentare il circuito

FigHandle = gcbf;
UnknownData = '-???-';
BlockData = get(FigHandle, 'UserData');
BlockHandle = getfield(BlockData, 'BlockHandle');
ExtendedBlockType = get_param(BlockHandle, ...
  'sim_ExtendedBlockType');

SFigHandle = findobj('Type', 'Figure', ...
  'Name', ['Analog Components: ' getfullnamer(BlockHandle)]);
if FigHandle ~= SFigHandle
  FigHandle = SFigHandle;
end

ActualCircuit = '';
LastCircuit = '';

try
  ActualCircuit = eval(...
    [ExtendedBlockType ...
      '_ParamUI(''ActualCircuit'', BlockHandle);']...
    );
  LastCircuit = get_param(BlockHandle, 'sim_ActualCircuit');
end
```

```
if ~strcmp(ActualCircuit, LastCircuit)
  % se la descrizione circuitaleè                                    50
  % stata modificataè  necessario
  % aggiornare la finestra di dialogo
  Choise = questdlg(...
    ['The analog circuit corrispondent to ' ...
      get_param(BlockHandle,'Name') ...
      ' is changed: refresh the' ...
      ' dialog for the new analog circuit?'], ...
    'Attention!');
  if isempty(strmatch(Choise, {'No' 'Cancel' ''}))
    ControlHeight = 15;                                              60
    ButtonHeight = 25;
    GeneralControlsHeight = 40;
    NumberOfFrames = 2;

    % per prima cosa si cancellano gli
    % oggetti di controllo presenti
    % che si riferiscono alla precedente
    % descrizione analogica
    AnalogDeviceDescription('Delete', FigHandle);
    AnalogCircuitDescription('Delete', FigHandle);                   70

    % si calcolano le dimensioni della nuova
    %   finestra e sia aggiorna la
    % dimensione della figure che deve
    % contenere i nuovi oggetti (allo
    % stato attuale delle cose non essendoci
    % variazioi significative fra
    % le finestre di dialogo di diverse
    % descrizioni circuitali questa
    % operazione nonè  necessaria)                                   80
    NControlsInsideFrame1 = ...
      AnalogCircuitDescription('ControlsInside', FigHandle);
    NControlsInsideFrame2 =...
      AnalogDeviceDescription('ControlsInside', FigHandle);
    FigHeight = ...
      5 + GeneralControlsHeight + 10*NumberOfFrames + ...
      (ControlHeight+5)*(NControlsInsideFrame1 + ...
      NControlsInsideFrame2);

    FigurePosition = get(FigHandle, 'Position');                     90
    FigurePosition(4) = FigHeight;
    set(FigHandle, 'Position', FigurePosition);

    LastCtrlBottom = FigHeight -5;
    FrameHeight1 = ...
      5 + NControlsInsideFrame1*(ControlHeight + 5);
```

```
    FrameHeight2 = ...
      5 + NControlsInsideFrame2*(ControlHeight + 5);
    PutAnCmpControls(FigHandle, ...
      LastCtrlBottom, FrameHeight1, FrameHeight2);                    100

    AnalogCircuitDescription('Update', FigHandle);
    AnalogDeviceDescription('Update', FigHandle);

    set(...
      findobj(FigHandle, ...
      'Style', 'pushbutton', ...
      'String', 'Cheap'), ...
      'Enable', 'off');
                                                                      110
    % memorizzazione della nuova descrizione
    % circuitale adottata per quel blocco
    set_param(BlockHandle, 'sim_ActualCircuit', ActualCircuit);
    ActualDevice = eval(...
      [ExtendedBlockType  ...
        '_ParamUI(''DeviceOfCircuit'', BlockHandle);'] ...
      );
    set_param(BlockHandle, ...
      'sim_ActualDevice', ActualDevice);
  else                                                                120
    return;
  end
end % if sign(Gain)...
```

B.8 *AnHwCompile.m*

```
function [OutputFileName, varargout] = ...
  AnHwCompile(DiagramHandle,varargin)

% in ingresso ho il diagramma analogico flattenizzato,
% a questo punto devo fare le seguenti operazioni:
% recuperare la lista di tutti i blocchi presenti nel
% diagramma e identificare i blocchi sorgenti; a partire
% dai blocchi sorgente, vado ad analizzare tutti i
% blocchi a cuiè  collegato e, ricorsivamente,
% tutti i blocchi ad essi collegati (scansione a
% mo' di albero binario). Di volta i volta vado
% ad assegnare i nodi spice corrispondenti:
% in genere i nodi corrispondono agli ingressi
% e alle uscite del blocco, ma alcuni blocchi
% contengono implicitamente ulteriori nodi
% (come ad esembio il blocco sim_zoh che
% rappresenta un S/H e deve avere anche il nodo
% corrispondente all'ingresso del segnale di
% campionamento). I blocchi d'interfaccia
% (sim_analogIn e sim_analogOut) sono considerati
% una sorta di buffer e quindi i nodi assegnati
% sono uguali. In fase di compilazione vera e
% propria vengono ignorati. Il compilatore funziona
% sia nel caso in cui sono effettivamente presenti
% questi blocchi sia nel caso in cui non
% siano stati inclusi

EntityName = get_param(DiagramHandle, 'Name');
OutputFileName = [get_param(DiagramHandle, 'Name') ...
    '.cir'];

%è  prevista una descrizione all'interno
% del file .cir generato nel caso in cui
% si siano verificati eventuali errori
% (sono stati considerati i ùpi comuni)
ErrorDescription = '';
SubCircuitFunctions = '';
SubCircuitPrototypes = '';
LibraryLines = '';
NegSupplyNode = '0';

% per poter simulare il comportamento
% del circuito in codice spiceè  necessario
% introdurre il comando .TRAN che osserva il
% comportamento dewl circuito in un
% determinato intevallo di tempo. Nel caso
% in cui non siano presenti blocchi che
```

```
% hanno segnali di frequenza diversa da 0
% àverr attribuito un periodo di 10 secondi di
% default, in caso contrario la frequenza
% corrispondente àverr calcolata.

PeriodOfSystem = 10;
MaxSignalFrequency = 0;

if nargin > 1

  for k = 1:2:(nargin-1)
    switch upper(varargin{k})
      case 'ENTITY'
        EntityName = varargin{k + 1};
      case 'COMPILESPICE'
        CompileSPICE = varargin{k + 1};
      case 'OUTPUTFILENAME'
        OutputFileName = varargin{k + 1};
      otherwise
        bdclose(DiagramHandle);
        error(...
          ['Option "' varargin{k} ...
            '" not available for ' ...
            'AnHwCompile() function']);
    end % switch(uvarargin{k})
  end % for k = 1:(nargin-1)/2
end %if nargin > 1

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%% ORDINAMENTO DEI BLOCCHI %%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% si effettua una sorta di mappatura del circuito
% in cui identifico i blocchi presenti nel diagramma
% (sia i corrispondenti handle dei blocchi che la
% matrice di adiacenza di ogni blocco ottenuta con
% la funzione adjlist). Il vettore BlockHandles e
% l'array di celle ListOfAdj sono ordinati secondo il
% seguente principio: i primi NumberOfSourceBlocks
% elementi si riferiscono ai blocchi che rappresentano
% una sorgente analogica (sim_constant e
% sim_repeating_sequence sono i casi trattati)
% I successivi blocchi sono i cosiddetti "sottocircuiti",
% époich in Spice verranno rappresentati come tali
% ed infine (nel caso siano presenti) gli ultimi
% NumberOfInterfaceBlocks blocchi sono quelli d'interfaccia
% che servono a mappare gli ingressi e
% le uscite del circuito in una scheda analogica (PCB)
% indicata nel blocco sim_digHwInterface
[BlockHandles, ListOfAdj, ...
```

```
    NumberOfSourceBlocks, NumberOfInterfaceBlocks] = ...
  CircuitMapping(DiagramHandle);

if isempty(BlockHandles)                                                    100
  % se siè  verificato un errore viene restituito
  % un vettore nullo e la compilazione si arresta
  return;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%% ASSEGNAZIONE NODI DEI SOTTOCIRCUITI %%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% informazioni sulle alimentazioni del sistema
try                                                                         110
  PositiveSupply = get_param(...
    DiagramHandle, 'sim_PositiveSupply');
  NegativeSupply = get_param(...
    DiagramHandle, 'sim_NegativeSupply');
  SubCircuitPrototypes = ...
    ['VCC   1   0   DC   ' PositiveSupply '\n'];
  if str2num(NegativeSupply) % se non nulla
    SubCircuitPrototypes = [SubCircuitPrototypes ...
        'VEE   2   0   DC   ' NegativeSupply '\n'];
    NegSupplyNode = '2';                                                    120
  end
catch
  errordlg(...
    ['No Supply Value found, ' ...
      'check the sim_DigHwInterface block.'], ...
    'No Supply Found'...
    );
  return;
end
                                                                            130
% la cella contiene nelle corrispondenti
% posizioni i vettori con i nodi assegnati
% per i vari blocchi secondo  la descrizione
% Spice. In ogni vettore contenuto nella cella,
% i primi elementi riportati corrispondono
% alle porte d'ingresso del blocco simulink,
% mentre i successivi corrispondono alle porte d'uscita.
NodeList = cell(1,length(BlockHandles));

                                                                            140
NodeNumber = 10;
% il numero da cui parte il conteggio dei nodiè  10
% époich tra i primi numeri sono assegnati di default:
% 0 --> riferimento
% 1 --> alimentazione positiva
```

```
% 2 --> alimentazione negativa
% 3 --> eventuale clock (non trattato)
% da 4 a 10 --> riservati x applicazioni future
% da 10 in poi --> nodi del circuito

% a partire da ogni sorgente analogica presente
% (NON alimentazione) si assegnano i nodi in
% modo crescente: si parte dall'uscita del blocco
% sorgente e si assegna a tale blocco il numero del
% nodo e, questo stesso numero viene anche assegnato
% ai blocchi ad esso connessi (tenendo in considerazione
% le porte del blocco. Nel caso in cui il blocco
% collegato a quello iniziale sia a sua volta
% connesso ad altri blocchi, viene la medesima funzione
% si richiama ricorsivamente èfinch nonè  stato
% analizzata ogni foglia dell'albero binario
% delle connessioni. Tutta la procedura viene
% effettuata per ogni sorgente analogica éfinch
% non sono asseganti tutti i nodi a tutte le porte
% di ogni blocco.
for k = 1:NumberOfSourceBlocks
  [NodeList, NodeNumber] = NodalAssignation(...
    BlockHandles, ListOfAdj, ...
    BlockHandles(k), NodeList, NodeNumber);
end %for k = ...

% i vettori di nodi trovati vengono semplicemente
% convertiti in stringhe per essere poi nella
% descrizione del file spice
NodeList = NodalCorrection(NodeList);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% ACQUISIZIONE VALORI NUMERICI DEL CODICE SPICE %%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% si effettua una descrizione spice dei
% blocchi sorgente trovati (esclusi quelli d'alimentazione),
% vengono òperci acquisiti anche i corrispondenti valori
% numerici (e non) da includere nella descrizione del
% corrispondente circuito

for k = 1:NumberOfSourceBlocks
  ActualBlock = BlockHandles(k);
  BlockName = get_param(BlockHandles(k), 'Name');
  BlockNumber = num2str(k);
  BlockNode = NodeList{k};
  ExtendedBlockType = get_param(...
    BlockHandles(k), 'sim_ExtendedBlockType');
```

```
disp(['--- Describing analog source version of block ' ...
    BlockName '...']);
  [SourceSpiceDescription, SourceSpiceVariableStr] = ...
    SpiceDescription(BlockHandles(k), ExtendedBlockType);
  if isempty(SourceSpiceDescription)                                  200
    errordlg(...
      ['No Spice description found for block ' ...
        '' BlockName ':' ...
        ' Spice Description aborted.'], ...
      'Missing Analog Description' ...
      );
    SubCircuitPrototypes = ...
      ['*No description for source block ' ...
        BlockName '\n'];
      SubCircuitFunctions = '*\n';                                    210
      ErrorDescription = [ErrorDescription ...
          ' missing spice description for block ' ...
          BlockName];
      disp(...
        ['>>>>>No Spice description for block ' ...
          BlockName ...
          '...']);
      break;
    end % if isempty(...
                                                                      220
    % a seconda del tipo di blocco vengono effettuate
    % delle descrizioni specifiche che sostituiscono
    % i valori contenuti nei parametri
    % SourceSpiceVariableStr dei corrispondenti
    % blocchi all'interno della stringa
    % SourceSpiceDescription estratta
    switch upper(ExtendedBlockType)
      case 'SIM_CONSTANT'
        ActualSupplyNode = '1';
        CircuitSpiceDescription = SourceSpiceDescription;             230
        [Token, LastPartOfString] = ...
          strtok(CircuitSpiceDescription, '@');
        CircuitSpiceDescription = ...
          [Token BlockName LastPartOfString(2:end)];

        CircuitSpiceVariable = ...
          get_string_piece(SourceSpiceVariableStr, '|');
        n =1;
        while ~isempty(strfind(CircuitSpiceDescription,'@'))
          [Token, LastPartOfString] = ...                             240
            strtok(CircuitSpiceDescription, '@');
          [StrValue, ErrorDescription] = ...
            RecoverParamValue(...
```

```
        BlockHandles(k), CircuitSpiceVariable{n}, ...
        ErrorDescription, BlockName);
      CircuitSpiceDescription = ...
        [Token StrValue LastPartOfString(2:end)];
      n = n+1;
    end % while
    SubCircuitFunctions = [SubCircuitFunctions ...
        CircuitSpiceDescription];

    if strfind(get_param(BlockHandles(k), ...
        'Const_Data_RealValue'),'-')
      ActualSupplyNode = '2';
    end
    %è  un caso particolare époich nonostante
    % sia una sorgente analogicaè  comunque
    % descritta da un sottocircuito cheè
    % necessario includere in SubCircuitFunctions.
    % Il relativo prototipo del sottocircuito
    % viene invece memorizzato nella stringa
    % SubCircuitPrototypes
    LatestCircuitPrototype = ['X' BlockNumber ...
        '   ' ActualSupplyNode ...
        '   ' BlockNode '   ' BlockName ';\n'];
    SubCircuitPrototypes = ...
      [SubCircuitPrototypes LatestCircuitPrototype];

  case {'SIM_REPEATING_SEQUENCE', 'SIM_ANALOGIN'}
    [Token, LastPartOfString] = ...
      strtok(SourceSpiceDescription, '@');
    SourceSpiceDescription = ...
      [Token BlockNumber LastPartOfString(2:end)];

    [Token, LastPartOfString] = ...
      strtok(SourceSpiceDescription, '@');
    SourceSpiceDescription = ...
      [Token BlockNode LastPartOfString(2:end)];

    [Token, LastPartOfString] = ...
      strtok(SourceSpiceDescription, '@');
    value_seq = get_param(...
      BlockHandles(k), 'sim_value_sequence');
    time_seq = get_param(...
      BlockHandles(k), 'sim_time_sequence');

    if isempty(time_seq) | isempty(value_seq)
      errordlg(...
        ['Unable to find signal from ' BlockName ...
          ' block: if a ''To Workspace'' block' ...
          ' is connected run the simulation' ...
```

```
          ' otherwise add a ''To Workspace''' ...
          ' block to catch ' BlockName ' signal'], ...
        'No Signal Found Error');
      ErrorDescription = ...
        [ErrorDescription '\n*        ' ...
          'no signal found in block ' BlockName];
    end

    PWL_string = '\n +';

    % se le sequenze di valori contengono le
    % parentesi quadre devo toglierle nella
    % descrizione Spice
    if strfind(value_seq, '[')
      value_seq = value_seq(2:end-1);
    end

    if strfind(time_seq, '[')
      time_seq = time_seq(2:end-1);
    end

    while(~isempty(strfind(value_seq, ' ')) & ...
        ~isempty(strfind(time_seq, ' ')))
      % époich esiste un limite di caratteri
      % per linea conviene riportare in ogni
      % linea solo due elementi (x, y) alla
      % volta e utilizzare il comando "+" per
      % gli altri elementi
      [t, time_seq] = strtok(time_seq,' ');
      [y, value_seq] = strtok(value_seq,' ');
      PWL_string = [PWL_string t ' ' y '\n +'];
    end

    % tolgo l'ultimo '\n' e '+'
    PWL_string = PWL_string(1:end-4);

    %non essendo rappresentato da un
    % sottocircuito nonè  necessario aggiungere
    % la "funzione" ma solo la descrizione
    % del prototipo
    SourceSpiceDescription = ...
      [Token PWL_string LastPartOfString(2:end)];
    SubCircuitPrototypes = ...
      [SubCircuitPrototypes SourceSpiceDescription];

  otherwise
    return;
 end %switch ExtendedBlockType
end %for k
```

```
% analogamente al caso precedente si
% recupera la descrizione spice del
% blocco che questa voltaè  effettivamente
% rappresentato da un
% sottocircuito
for z=NumberOfSourceBlocks+1:...
    length(BlockHandles)-NumberOfInterfaceBlocks
  BlockName = get_param(BlockHandles(z), 'Name');
  BlockNumber = num2str(z);
  BlockNode = NodeList{z};
  ExtendedBlockType = get_param(...
    BlockHandles(z), 'sim_ExtendedBlockType');
  disp(...
    ['--- Describing analog sub circuit'...
      ' version of block ' ...
      '' BlockName '...']);
  [CircuitSpiceDescription, CircuitSpiceVariableStr] = ...
    SpiceDescription(BlockHandles(z),ExtendedBlockType);
  if isempty(CircuitSpiceDescription)
    errordlg(...
      ['No Spice description found for block ' ...
        BlockName], ...
      'Missing Analog Description');
    SubCircuitFunctions = ...
      ['*No Sub Circuit description for ' ...
        BlockName '\n'];
    SubCircuitPrototypes = ...
      ['*No Circuit Prototypes for ' ...
        BlockName '\n'];
    ErrorDescription = ...
      [ErrorDescription '\n*        ' ...
        'missing spice description for block ' ...
        BlockName];
    disp(...
      ['>>>>>No Spice description for block ' ...
        '' BlockName '.']);
    break;
  end %if isempty(...

  CircuitSpiceVariable = get_string_piece(...
    CircuitSpiceVariableStr, '|');

  % per prima cosa si assegna il nome del blocco
  [Token, LastPartOfString] = ...
    strtok(CircuitSpiceDescription, '@');
  CircuitSpiceDescription = ...
    [Token BlockName LastPartOfString(2:end)];
```

```
switch upper(ExtendedBlockType)
  case 'SIM_ZOH'
    Tsam = eval_param(...
      BlockHandles(z), 'sim_Sample_Time');

    % selezione àcapacit da attribuire
    % al modello del sample and
    % hold. Il modelloè  definito a scopo
    % puramente simulativo, non rappresenta                    400
    % in maniera univoca il dispostivo scelto,
    % èpoich non esistono modelli spice
    % specifici per questo tipo di componente
    HoldCap = num2str(Tsam/1000); %approssimato
    [Token, LastPartOfString] = ...
      strtok(CircuitSpiceDescription, '@');
    CircuitSpiceDescription = ...
      [Token HoldCap LastPartOfString(2:end)];
    SubCircuitFunctions = ...
      [SubCircuitFunctions CircuitSpiceDescription];          410

    % come per la àcapacit anche i valori
    % che identificano il segnale di campionamento
    % sono stati ricavati in maniera
    % empirica al solo scopo simulativo
    PulseTime = num2str(Tsam);
    RaiseTime = num2str(Tsam/10000);
    FallTime = RaiseTime;
    PulseWidth = num2str(Tsam/1000);
    PulseSourceSpiceDescription = [...                        420
        'VNpulse' BlockNumber  ...
        '  PulseNode' BlockNumber '  0  PULSE  ' ...
        '0  ' PositiveSupply '   0' ...
        '  ' RaiseTime  '  '  FallTime ...
        '  ' PulseWidth '  '  PulseTime '\n'];
    SubCircuitPrototypes = ...
      [SubCircuitPrototypes PulseSourceSpiceDescription];

    LatestCircuitPrototype = ['X' BlockNumber ...
        '   ' BlockNode '   PulseNode' BlockNumber ...        430
        '   ' BlockName ';\n'];
    SubCircuitPrototypes = ...
      [SubCircuitPrototypes LatestCircuitPrototype];

  case {'SIM_SUM2'}
    n =1;
    while ~isempty(strfind(CircuitSpiceDescription,'@'))
      [Token, LastPartOfString] = ...
        strtok(CircuitSpiceDescription, '@');
```

```
    [StrValue, ErrorDescription] = ...
      RecoverParamValue(...
      BlockHandles(z), CircuitSpiceVariable{n}, ...
      ErrorDescription, BlockName);
    CircuitSpiceDescription = ...
      [Token StrValue LastPartOfString(2:end)];
    n = n+1;
  end % while
  SubCircuitFunctions = ...
    [SubCircuitFunctions CircuitSpiceDescription];

  % NegSupplyNodeè  stato deciso in precedenza
  LatestCircuitPrototype = ['X' BlockNumber ...
      '   ' BlockNode '   1   ' NegSupplyNode ...
      '   ' BlockName ';\n'];
  SubCircuitPrototypes = ...
    [SubCircuitPrototypes LatestCircuitPrototype];

case{'SIM_GAIN'}
  Gain = eval_param(BlockHandles(z), 'Gain');
  n =1;
  while ~isempty(strfind(CircuitSpiceDescription,'@'))
    [Token, LastPartOfString] = ...
      strtok(CircuitSpiceDescription, '@');
    [StrValue, ErrorDescription] = RecoverParamValue(...
      BlockHandles(z), CircuitSpiceVariable{n}, ...
      ErrorDescription, BlockName);
    CircuitSpiceDescription = ...
      [Token StrValue LastPartOfString(2:end)];
    n = n+1;
  end % while
  SubCircuitFunctions = ...
    [SubCircuitFunctions CircuitSpiceDescription];

  if Gain >1 | Gain<0
    % NegSupplyNodeè  stato deciso in precedenza
    LatestCircuitPrototype = ['X' BlockNumber ...
        '   ' BlockNode '   1   ' NegSupplyNode ...
        '   ' BlockName ';\n'];
  else
    %caso del partitore di tensione
    LatestCircuitPrototype = ['X' BlockNumber ...
        '   ' BlockNode '   ' BlockName ';\n'];
  end % if Gain >1 | Gain<0
  SubCircuitPrototypes = ...
    [SubCircuitPrototypes LatestCircuitPrototype];

otherwise % sim_analogIn non deve essere processato
```

```
  end %switch
                                                                    490
  ListOfDevicesStr = eval(...
    [ExtendedBlockType ...
      '_ParamUI(''DeviceOfCircuit'', '...
      'BlockHandles(z));'],'0');
  SpiceDeviceDescription = ...
    SpiceDescription(BlockHandles(z));
  % seè  presente una descrizione spice
  % dei dispositivi del del circuito
  % che rappresenta quel bloccoè,  necessario
  % aggiungere la chiamata a quella determinata                     500
  % libreria effettuata in spice con il comando .LIB
  if SpiceDeviceDescription
    DevicesOfCircuit = ...
      get_string_piece(ListOfDevicesStr, '|');
    for k = 1: length(DevicesOfCircuit)
      ActualDevice = DevicesOfCircuit{k};
      if isempty(strfind(LibraryLines, ActualDevice)) & ...
          ~isempty(ActualDevice)
        LibraryLines = ...
          [LibraryLines '.LIB ' ActualDevice '.lib;\n'];          510
      end %isempty(strfind(...
    end %for k = ...
  end %if SpiceDeviceDescription
end %for z

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% GENERAZIONE DEL FILE .CIR IN CODICE SPICE %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% si apre in scrittura il file .cir                                 520
% che intendo generare
ScriptFID = fopen(OutputFileName, 'wt');

for i=1:length(BlockHandles(1:NumberOfSourceBlocks))
  MaxSignalFrequency = RecoverBlockFreq(...
    BlockHandles(i), MaxSignalFrequency);
end

if MaxSignalFrequency
  PeriodOfSystem = 1/max(MaxSignalFrequency);                       530
end

% in caso di assenza di errori si include
% la seguente riga
if isempty(ErrorDescription)
  ErrorDescription = 'NO ERROR FOUND';
end
```

```
% la prima riga del file spiceè  sempre il titolo
TitleOfCirFile = ['Spice description for ' ...
    get_param(DiagramHandle, 'Name') ...
    '''s CodeSimulink Diagram.\n'];

% intestazione del file
HeaderOfComment = [...
    '*\n*\n' ...
    '* File generated on date ' datestr(now) '\n'...
    '* MatLab Version ' version '\n'...
    '* Analog Hardware description of ' ...
    get_param(DiagramHandle, 'Name') '\n'...
    '* Number of source element(s): ' ...
    num2str(NumberOfSourceBlocks) '\n'...
    '* Number of subcircuit element(s): ' ...
    num2str(length(BlockHandles)-NumberOfSourceBlocks-...
    NumberOfInterfaceBlocks) '\n'...
    '* Number of interface element(s): ' ...
    num2str(NumberOfInterfaceBlocks) '\n' ...
    '*\n' ...
    '* Error: ' ErrorDescription '.\n'...
    '*\n'];

% linee conclusive che permettono tra le
% altre cose di effettuare una
% simulazione in spice del
% circuito equivalente
EndingLines= [LibraryLines ...
    '.TRAN  1e-9  ' num2str(PeriodOfSystem) '\n' ...
    '.PROBE\n' ...
    '.END\n' ...
  ];

% ordinamento delle stringhe da memorizzare
% nel file secondo l'ordine stabilito
FinalFileScript = [ ...
    TitleOfCirFile ...
  HeaderOfComment ...
  '*\n' ...
    '*\n' ...
    SubCircuitFunctions ...
  '*\n' ...
    '*\n' ...
    SubCircuitPrototypes ...
  EndingLines ...
];

% scrittura su file di uscita
```

```
fprintf(ScriptFID, FinalFileScript);
fclose(ScriptFID);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%% FUNZIONI LOCALI %%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [NodeList, NodeNumber, varargout] = ...
  NodalAssignation(AllBlockHandles,ListOfAdj, varargin)

% oltre a verificare che il blocco successivo
% esista (il suo handle non sia nullo,
% si deve controllare anche il numero attuale
% dei nodi memorizzati per il blocco successivo
% èpoich, nel caso in cui si siano raggiunto +
% il numero massimo di nodi a disposizione
% per quel blocco, non àbisogner analizzarlo
% ulteriormente ma proseguire con quelli successivi.
% il primo caso si verifica quando ho dei circuiti
% aperti (la funzione adjlist àassegner
% al blocco successivo un handle nullo), mentre
% il secondo si verifica se ho ùpi sorgenti
% e blocchi a ùpi ingressi: come siè  visto
% l'algoritmo scandisce il circuito a mo' di albero
% binario a partire dalle sorgenti e assegna a
% tutti i blocchi lungo il suo cammino UN nodo
% d'ingresso e TUTTI i nodi d'uscita

BlockHandle= varargin{1};
NodeList = varargin{2};
NodeNumber = varargin{3};

PortHandles = get_param(...
  BlockHandle, 'PortHandles');
InPorts = PortHandles.Inport;
NumInPorts = length(InPorts);
OutPorts = PortHandles.Outport;
NumOutPorts = length(OutPorts);

BlockPosition = ...
  find(ismember(AllBlockHandles,BlockHandle));
BlockPortInfo = ListOfAdj{BlockPosition};


for j = 1:NumOutPorts
  % ricavo lo stato delle connessioni per
  % quella porta d'uscita
```

```
% --> Aè  l'handle della porta
%      d'uscita che sto analizzando
% --> X1 e X2 (...) sono gli handle dei
%      blocchi connessi alla porta A
%      dell'uscita analizzata                                    640
% --> Y1 e Y2 (...) sono gli handle delle
%      porte dei blocchi connessi alla porta A
%      dell'uscita analizzata
%[A   X1  X2 ...  Y1  Y2 ...]
OutPortStatusConnection = ...
  BlockPortInfo(NumInPorts+j,:);
% le porte delle uscite si
% trovano dopo quelle d'ingresso

ActualPortNumber= ...                                            650
  get_param(OutPortStatusConnection(1), 'PortNumber');
NodeList{BlockPosition}(ActualPortNumber+NumInPorts)= ...
  NodeNumber;
% cerco a quanti blocchiè  connessa quell'uscita
NumberOfConnectedBlock = ...
  (length(OutPortStatusConnection)-1)/2;

for i =1:NumberOfConnectedBlock
  % tutti i nodi che costituiscono un nodo
  % d'ingresso per il blocco successivo                            660
  % devono avere lo stesso valore del nodo
  % di uscita del blocco precedente
  NextBlockHandle = OutPortStatusConnection(1+i);
  if NextBlockHandle ~= 0
    ExtBlockTypeOfNextBlock = ...
      get_param(NextBlockHandle, 'sim_ExtendedBlockType');
    NextBlockPortHandle = ...
      OutPortStatusConnection(1+NumberOfConnectedBlock+i);
    NextPortNumber = ...
      get_param(NextBlockPortHandle, 'PortNumber');              670
    NextBlockPosition = ...
      find(ismember(AllBlockHandles,NextBlockHandle));
    NodeList{NextBlockPosition}(NextPortNumber) = ...
      NodeNumber;
    % ai blocchi d'interfaccia assegno
    % lo stesso valore del nodo
    % di ingresso e uscita per non alterare
    % l'assegnazione dei nodi
    % agli altri blocchi significativi.
    % Questi blocchi saranno                                     680
    % comunque esclusi dalla descrizione
    % in codice spice
    if ( strcmp(upper(ExtBlockTypeOfNextBlock), ...
        'SIM_ANALOGIN') & ...
```

```
          ~eval([ExtBlockTypeOfNextBlock ...
            '_ParamUI(''AnalogSource'', ' ...
            'NextBlockHandle)'],'1') ) | ...
          strcmp(upper(ExtBlockTypeOfNextBlock), ...
          'SIM_ANALOGOUT')
        NodeNumber = NodeNumber-1;                                    690
      end
    end
  end
  NodeNumber = NodeNumber +1;

  for i = 1:NumberOfConnectedBlock
    NextBlockHandle = OutPortStatusConnection(1+i);
    NextBlockPosition = find(...
      ismember(AllBlockHandles,NextBlockHandle));
    if NextBlockHandle ~= 0 % se esiste il blocco successivo          700
      PortsOfNextBlock = ...
        get_param(NextBlockHandle, 'Ports');
      MaxNumberOfNode = ...
        PortsOfNextBlock(1)+PortsOfNextBlock(2);
      NumberOfNodeOfNextBlock = ...
        length(NodeList{NextBlockPosition});
      if NumberOfNodeOfNextBlock < MaxNumberOfNode
        % se il blocco successivo non ha esaurito
        % tutti i nodi assegnabili
        [NodeList, NodeNumber] = ...                                  710
          NodalAssignation(AllBlockHandles,ListOfAdj, ...
          NextBlockHandle, NodeList, NodeNumber);
      end
    end
  end

end % for j =...

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [varargout] = ...                                            720
  IsSourceBlock(BlockHandle, varargin)

% la funzione ritorna uno se il bloccoè  una
% sorgente analogica, altrimenti
% viene restituito il valore 0
ExtendedBlockType = ...
  get_param(BlockHandle, 'sim_ExtendedBlockType');
varargout{1} = eval(...
  [ExtendedBlockType ...
    '_ParamUI(''AnalogSource'', BlockHandle);'],'0'...                730
  );

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function [NodeList, varargout] = ...
  NodalCorrection(NodeList,varargin)

% converte i vettori contenuti
% ella cella NodeList in stringhe di
% caratteri da inserire poi
% nella descrizione spice                                          740
for i = 1: length(NodeList)
  % i nodi nulli corrrispondono a
  % dei blocchi d'interfaccia che si
  % comportano come dei blocchi sorgente
  NodeStr = num2str(setdiff(NodeList{i}(1),0));
  for j = 2:length(NodeList{i})
    NodeStr = [NodeStr '  ' num2str(NodeList{i}(j))];
  end % for j = ...
  NodeList{i} = NodeStr;
end % for i = ...                                                  750

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [StringOfValue, ErrorDescription, varargout] = ...
  RecoverParamValue(BlockHandle,NameOfVariable, ...
  ErrorDescription, BlockName);

% recupera i valori numerici (memorizzati
% comunque sotto forma di stringa)
% e verifica effettivamente siano presenti
% nella maschera del blocco e che                                  760
% non siano ancora sconosciuti
StringOfValue ='';
NameOfVariableParam = ['sim_' NameOfVariable];

try
  StringOfValue = ...
    get_param(BlockHandle, NameOfVariableParam);
  if strcmp(StringOfValue, '-???-')
    %è  presente nella maschera ma
    % non ha un valore corretto                                    770
    StringOfValue = '<<UNKNOWN>>';
    errordlg(...
      ['Unknown value for parameter ' ...
        NameOfVariableParam ...
        ' in ' BlockName '''s Block:'], ...
      'Unknown Value Of Parameter');
    ErrorDescription = ...
      [ErrorDescription '\n*        ' ...
        'missing value for ' NameOfVariableParam ...
        ' of ' BlockName];                                         780
    disp(['>>>>Unknown value found for ' ...
        NameOfVariableParam ...
```

```
        ' of ' BlockName '.']);
  end
catch
  % nonè  stato trovato quel parametro
  % nella maschera
  StringOfValue = '<<MISSING>>';
  errordlg(['Unable to find parameter ' ...
      NameOfVariableParam ...                                    790
      ' in ' BlockName ''''s Block:'],  ...
    'Missing Value Of Parameter');
  ErrorDescription = ...
    [ErrorDescription '\n*        ' ...
      'missing value for ' NameOfVariableParam ...
      ' of ' BlockName];
  disp(['>>>>Missed value ' NameOfVariableParam ...
      ' of ' BlockName '.']);
end
```

B.9 *AnHwDevParameters.m*

```
function [varargout] = ...
  AnHwDevParameters(AnalyzedDevice, varargin)

% questa funzione contiene la lista dei
% parametri ed dell'etichetta che li
% identifica

switch AnalyzedDevice
  case 'OpAmp'
    varargout{1} = ...                                          10
      ['Min Supply(V):|Max Supply(V):|'...
        'Ibias(nA):|Voff(mV):|'...
        'Gain Band(MHz):|'...
        'Voltage Noise(nV/Sqrt(Hz)):|' ...
        'Slew Rate(uV/V):|Price($):'];
    varargout{2} = ...
      ['MinSupply|MaxSupply|'...
        'BiasCurrent|VoltageOffset|'...
        'GainBand|VoltageNoise|' ...
        'SlewRate|Price'];                                      20
  case 'SHA'
    varargout{1} = ...
      ['Max Supply(V):|'...
        'Full Scale Range (V)|'...
        'Acquisition Time(us) H->S:|' ...
        'Aperture Time(ns) S->H:|'...
        'Aperture Jitter(ns) S->H:|'...
        'Settling Time(us) S->H:|' ...
        'Droop Rate (uV/us)|Hold Step Offset (mV):|'...
        'Small Signal Bandwidth (MHz)|' ...                     30
        'Max ADC Resolution (bit)|Price($):'];
    varargout{2} = ...
      ['MaxSupply|FullScale|'...
        'AcquisitionTime|ApertureTime|'...
        'ApertureJitter|SettlingTime|' ...
        'DroopRate|HoldStepOffset|'...
        'SmallSignalBandwidth|ADCRes|Price'];
  otherwise
    varargout{1} = '';
    varargout{2} = '';                                          40
end %end case
```

B.10 *AnHwPassCompSpec.m*

```
function [varargout] = ...
  AnHwPassCompSpec(AnalyzedPassComp, varargin)

% questa funzione contiene la lista dei
% parametri ed dell'etichetta che li
% identifica

switch upper(AnalyzedPassComp)
    case 'RES'
        varargout{1} = ...                                          10
          ['Res. Tolerance(%):|'...
            'Res. Wattage(W):|Res. Price($):'];
        varargout{2} = ['Tolerance|'...
            'Wattage|Price'];
    otherwise
        varargout{1} = '';
        varargout{2} = '';
end %end case
```

B.11 *CircuitMapping.m*

```
function [BlockHandles,ListOfAdj, ...
    NumberOfSourceBlocks, NumberOfInterfaceBlocks] = ...
  CircuitMapping(DiagramHandle, varargin)

% effettua una mappatura del diagramma a blocchi
% identificando quelli che rappresentano l'alimentazione
% del circuito, i sottocircuiti del sistema e i blocchi
% cosiddetti "d'interfaccia" che permenttono di
% assegnare ai segnali principali del circuito
% un connettore (un pin) di una eventuale
% circuito integrato

[BlockHandles, ListOfAdj] = adjlist(DiagramHandle);
NumberOfBlocks = length(BlockHandles);

OriginalBlockHandles = BlockHandles(:,1);
OriginalListOfAdj = ListOfAdj;

InterfaceBlockHandles = [];
InterfaceBlockListOfAdj = [];

SourceBlockHandles = [];
SourceBlockListOfAdj = [];

SubCircuitBlockHandles = [];
SubCircuitBlockListOfAdj = [];

for k = 1:NumberOfBlocks
    %matrice di adiacenza per quel blocco
    PortStatus = ListOfAdj{k};

    ActualBlock = BlockHandles(k);

    % indice del blocco analizzato all'interno
    % del vettore originale di
    % handles (passato come argomento d'ingresso
    IndexInOrderedVector = find(...
      ismember(OriginalBlockHandles, ActualBlock)...
      );

    ExtendedBlockType = ...
      get_param(ActualBlock, 'sim_ExtendedBlockType');

    switch upper(ExtendedBlockType)
        case {'SIM_CONSTANT' 'SIM_REPEATING_SEQUENCE'}
            if isempty(IsNextPrevBlock(...
                ActualBlock, 'next', 'sim_analogIn'))
```

```
                SourceBlockHandles = ...
                  [SourceBlockHandles ...
                    OriginalBlockHandles(IndexInOrderedVector)];
                SourceBlockListOfAdj = ...
                  [SourceBlockListOfAdj ...
                    OriginalListOfAdj(IndexInOrderedVector)];
            end
        case 'SIM_ANALOGIN'
                SourceBlockHandles = ...
                  [SourceBlockHandles ...
                    OriginalBlockHandles(IndexInOrderedVector)];
                SourceBlockListOfAdj = ...
                  [SourceBlockListOfAdj ...
                    riginalListOfAdj(IndexInOrderedVector)];
        case 'SIM_ANALOGOUT'
            InterfaceBlockHandles = ...
              [InterfaceBlockHandles ...
                OriginalBlockHandles(IndexInOrderedVector)];
            InterfaceBlockListOfAdj = ...
              [InterfaceBlockListOfAdj ...
                OriginalListOfAdj(IndexInOrderedVector)];
        case {'SIM_ZOH', 'SIM_SUM2', 'SIM_GAIN'}
            SubCircuitBlockHandles = ...
              [SubCircuitBlockHandles ...
                OriginalBlockHandles(IndexInOrderedVector)];
            SubCircuitBlockListOfAdj = ...
              [SubCircuitBlockListOfAdj ...
                OriginalListOfAdj(IndexInOrderedVector)];
        otherwise

    end %switch
end % for k = ...

if isempty(SourceBlockHandles)
    % non ci sono blocchi sorgenti
    errordlg(['No source block found in diagram ' ...
        get_param(DiagramHandle, 'Name') ...
            ': probably unsupported '...
            'Simulink source block.' ],...
          'Missing block');
    BlockHandles = [];
    ListOfAdj = [];
    NumberOfSourceBlocks = [];
    NumberOfInterfaceBlocks = [];
    return;
end

% a questo punto si possono ordinare i blocchi
% identificati e ìcos facendo
```

```
% mappare il circuito
BlockHandles = ...
  [SourceBlockHandles ...
    SubCircuitBlockHandles ...                                    100
  InterfaceBlockHandles];
ListOfAdj = ...
  [SourceBlockListOfAdj ...
    SubCircuitBlockListOfAdj ...
  InterfaceBlockListOfAdj];
NumberOfSourceBlocks = length(SourceBlockHandles);
NumberOfInterfaceBlocks = length(InterfaceBlockHandles);
```

B.12 *data_extraction.m*

```
function [dati, nomicolonne] = data_extraction(Component)

% funzione di estrazione degli elementi del
% database riferiti ad un determinato tipo
% di componente elettronico. Il databaseè  sempre
% il medesimo, cambiano le tabelle dei componenti,
% cheè  necessario chiamare con il nome di riferimento
% attribuito a quel
% componente: nel caso specifico:
% amplificatori operazionali --> OpAmp                          10
% sample and hold amplifier --> SHA
% resistori --> Res
% Per impostare il database sorgenteè  necessario
% andare in
% Start => Impostazioni => Pannello di controllo =>
% Strumenti di amministrazione => Origine dati (ODBC)
% e aggiungere il percorso in cui si trova il database
% (di dafaultè  nella cartella analog).
% Nel caso si verifichino degli
% errori, all'utente àsar comunque illustrata                   20
% la serie di operazioni da
% svolgere per linkare il database in modo
% che possa essere letto dal
% presente programma

logintimeout(30);

name_db = 'Component_Database';
conn = database(name_db, '', '');
                                                               30
try
  % tentativo di connessione al database
  ping(conn);
catch
  errordlg(['No connection to ' name_db ':' ...
      'follow the instructions in Matlab Command Window ' ...
      'to set the Data Source.'], 'Database error');

  w = what;
  database_path = w.path;                                      40
  fprintf(['First of all, if you are using a ' ...
      'non-english Windows OS version\n' ...
      'you change the default decimal separator:\n' ...
      'From the Window Start menu select:\n' ...
      'Setting => Control Panel => ' ...
      'International Options\n' ...
      'Select the ''Number'' tab.\n' ...
```

```
      'Change the ''decimal separator'' field in '...
      'this window.\n\n' ...
      'For setting the source database follow these '...
      'instructions:\n' ...
      'From the Window Start menu select:\n' ...
      'Setting => Control Panel => ' ...
      'Administrative Tools => Data Source (ODBC).\n' ...
      'Select the ''User DSN'' tab:\n' ...
      'Click Add in the ODBC Data Source Administrator '...
      'dialog box.\n' ...
      'Select the Microsoft Access Driver (*.mdb) that '...
      'the data source database' ...
      ' will use and click Finish.\n\n' ...
      'Provide the Data Source Name => \n     %s   \n' ...
      'in the dialog box.\n'...
      'Click Select to define the database that '...
      'the data source will use.\n'...
      'Find the database of component is the '...
      'following director: \n\n' ...
      '%s\n\n' ...
      'and select the database.\n' ...
      'Click OK to close the Select Database dialog box.\n' ...
      'In the ODBC Setup dialog box, click OK.\n\n' ...
      'For more information check the '...
      'Database Toolbox User Guide\n' ...
      'available in the MathWorks website.\n'], ...
    name_db, database_path);

  dati = {};
  nomicolonne = {};
  return;
end
% CURSOR = EXEC(CONNECT,SQLQUERY) esegue un codice in SQL.
% Bisogna riportare il codice in SQL
% della query realizzata nel database di Access per
% estrarre i valori che servono all'analisi.
% Il comando * permette di estrarre tutte le colonne
% di dati presenti nella tabella "Component"
SQL_query = ['SELECT * FROM ' Component];
curs = exec(conn, SQL_query);

% i dati vengono ritornati in un formato
% che supporta le stringhe in caso di soli
% risultati numerici si òpu utilizzare
% 'numeric' al posto di 'cellarray'
DataType = 'cellarray';
setdbprefs('DataReturnFormat', DataType);

% CURSOR = FETCH(INITIALCURSOR,ROWLIMIT) permette
```

```
% di richiamare fisicamente i dati del database
% in matlab. Si òpu opzionalmente segnalare
% il numero massimo di righe da importare
curs = fetch(curs);                                                    100
dati = curs.Data;

% righe = rows(curs);
% colonne = cols(curs);

try
  % nel caso in cui l'estrazione di dati
  % sia andata a buon fine si àavr un array
  % di celle non vuoto
  nomi_colonne_str = columnnames(curs);                                110
  nomicolonne = ...
    get_string_piece(nomi_colonne_str, ',''');
catch
  errordlg(...
    ['No ' Component '''s database in ' ...
      '' name_db '.'], ...
    'Database error');
  nomicolonne = {};
end
                                                                       120
% chiusura della connessione al database
close(curs);
close(conn);
```

B.13 *DefaultAnCmpCallBack.m*

```
function  [varargout] = DefaultAnCmpCallBack(varargin)

% questa funzione contiene le
% callback dei pulsanti nella finestra
% di dialogo che descrive i componenti
% analogici del circuito
% (escluso il pulsante cancel).

FigHandle = gcf;
PBHandle = gco;

PushButtonTag = get(PBHandle, 'Tag');

Option = strtok(PushButtonTag, '_');
% per risalire all'informazione del
% pulsante premuto si sfrutta la stringa
% contenuta nella àpropriet Tag dell'oggetto
switch upper(Option)
    case 'FIND',
    otherwise
        return;
end
```

B.14 *find_res.m*

```
function [R1, R2, CompError, Tolerance, varargout] = ...
  find_res(Gain, GainError, varargin)

% questa funzione calcola le resistenze
% a partire da un valore di guadagno per un
% amplificatore operazionale sia esso invertente
% che non invertente. La funzione òpu
% essere utilizzata indistintamente
% per calcolare le resistenze ùpi adatte
% per ottenere un determinato
% rapporto di partizione sia esso maggiore
% di 1 o compreso fra 0 e 1
% (il caso negativo viene ricondotto
% a quello positivo)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% INIZIALIZZAZIONI
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% riga e colonna che permettono di risalire
% ad un rapporto di resistenzecorretto
r = '';
c = '';
Series = [6 12 24 48 96 192];
count = 1;

R1 = '';
R2 = '';
CompError = '';
Tolerance = '';

exponent = ...
  [0.0001 0.001 0.01 0.1 1 ...
    10 100 1e3 1e4 1e5 1e6 1e7];
off = 6;

% serie di resistenze E6
AllResSeries.E6 = [1 1.5 2.2 3.3 4.7 6.8];

% serie di resistenze E12
AllResSeries.E12 = [...
    1.0 1.2 1.5 1.8 2.2 2.7 ...
    3.3 3.9 4.7 5.6 6.8 8.2];

% serie di resistenze E24
AllResSeries.E24 = [...
    1.0 1.1 1.2 1.3 1.5 1.6 1.8 2 ...
    2.2 2.4 2.7 3 3.3 3.6 3.9 4.3 ...
```

```
    4.7 5.1 5.6 6.2 6.8 7.5 8.2 9.1];

% serie di resistenze E48
AllResSeries.E48 = [...
    1.00 1.05 1.10 1.15 1.21 1.27 1.33 1.40 ...
    1.47 1.54 1.62 1.69 1.78 1.87 1.96 2.05 ...
    2.15 2.26 2.37 2.49 2.61 2.74 2.87 3.01 ...
    3.16 3.32 3.48 3.65 3.83 4.02 4.22 4.42 ...
    4.64 4.87 5.11 5.36 5.62 5.90 6.19 6.49 ...
    6.81 7.15 7.50 7.87 8.25 8.66 9.09 9.53];


% serie di resistenze E96
AllResSeries.E96 = [...
    1.00 1.02 1.05 1.07 1.10 1.13 1.15 1.18 ...
    1.21 1.24 1.27 1.30 1.33 1.37 1.40 1.43 ...
    1.47 1.50 1.54 1.58 1.62 1.65 1.69 1.74 ...
    1.78 1.82 1.87 1.91 1.96 2.00 2.05 2.10 ...
    2.15 2.21 2.26 2.32 2.37 2.43 2.49 2.55 ...
    2.61 2.67 2.74 2.80 2.87 2.94 3.01 3.09 ...
    3.16 3.24 3.32 3.40 3.48 3.57 3.65 3.74 ...
    3.83 3.92 4.02 4.12 4.22 4.32 4.42 4.53 ...
    4.64 4.75 4.87 4.99 5.11 5.23 5.36 5.49 ...
    5.62 5.76 5.90 6.04 6.19 6.34 6.49 6.65 ...
    6.81 6.98 7.15 7.32 7.50 7.68 7.87 8.06 ...
    8.25 8.45 8.66 8.87 9.09 9.31 9.53 9.76];

% serie di resistenze E192
AllResSeries.E192 =[...
    1.00 1.01 1.02 1.04 1.05 1.06 1.07 1.09 ...
    1.10 1.11 1.13 1.14 1.15 1.17 1.18 1.20 ...
    1.21 1.23 1.24 1.26 1.27 1.29 1.30 1.32 ...
    1.33 1.35 1.37 1.38 1.40 1.42 1.43 1.45 ...
    1.47 1.49 1.50 1.52 1.54 1.56 1.58 1.60 ...
    1.62 1.64 1.65 1.67 1.69 1.72 1.74 1.76 ...
    1.78 1.80 1.82 1.84 1.87 1.89 1.91 1.93 ...
    1.96 1.98 2.00 2.03 2.05 2.08 2.10 2.13 ...
    2.15 2.18 2.21 2.23 2.26 2.29 2.32 2.34 ...
    2.37 2.40 2.43 2.46 2.49 2.52 2.55 2.58 ...
    2.61 2.64 2.67 2.71 2.74 2.77 2.80 2.84 ...
    2.87 2.91 2.94 2.98 3.01 3.05 3.09 3.12 ...
    3.16 3.20 3.24 3.28 3.32 3.36 3.40 3.44 ...
    3.48 3.52 3.57 3.61 3.65 3.70 3.74 3.79 ...
    3.83 3.88 3.92 3.97 4.02 4.07 4.12 4.17 ...
    4.22 4.27 4.32 4.37 4.42 4.48 4.53 4.59 ...
    4.64 4.70 4.75 4.81 4.87 4.93 4.99 5.05 ...
    5.11 5.17 5.23 5.30 5.36 5.42 5.49 5.56 ...
    5.62 5.69 5.76 5.83 5.90 5.97 6.04 6.12 ...
    6.19 6.26 6.34 6.42 6.49 6.57 6.65 6.73 ...
```

```
    6.81 6.90 6.98 7.06 7.15 7.23 7.32 7.41 ...
    7.50 7.59 7.68 7.77 7.87 7.96 8.06 8.16 ...
    8.25 8.35 8.45 8.56 8.66 8.76 8.87 8.98 ...
    9.09 9.19 9.31 9.42 9.53 9.65 9.76 9.88];                          100
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

BlockType = '';
if nargin == 3
  BlockType = varargin{1};
  if ~ischar(BlockType)
    BlockHandle = BlockType;
    BlockType = ...
      get_param(BlockHandle,'sim_ExtendedBlockType');
  end                                                                  110
end

switch BlockType
  case 'sim_gain'
    if Gain > 1
      % amplificatore non invertente
      R2_div_R1 = Gain-1;
    else
      if Gain <0
        R2_div_R1 = abs(Gain);                                         120
      else
        if Gain >0 & Gain <1
          R2_div_R1 = 1/Gain -1;
        else
          errordlg (...
            ['Unsupported gain 0 or 1 in block '...
              getfullname(BlockHandle)], ...
            'Parameter Error', 'non-modal');
          return;
        end                                                            130
      end
    end
  otherwise
    % rapporto di partizione maggiore di 0
    % (non guadagno di amplificatore)
    % oppure guadagno dell'amplificatore invertente
    R2_div_R1 = abs(Gain);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%                       140
% si cerca la prima serie che garantisca
% al massimo l'errore di guadagno
% indicato da GainError. Nel caso non
% si riesca a trovare vengono
% restituiti delle variabili vuote
```

```
while isempty(r) & count < length(Series)+1
  Res = getfield(AllResSeries, ...
    ['E' num2str(Series(count))]);
  dim = length(Res); % dimensione del vettore
                                                                    150
  % matrice che conntiene tutti i possibili
  % rapporti fra le resistenze
  % disponibili: alcuni di questi rapporti
  % avranno valore >=1, àsar
  % necessario dividerli per 10 in modo da
  % ottenere i valori effettivi
  % dei rapporti tutti con valore compreso fra 0 e 1
  LUT = Res' * (1./Res);

  cond = (LUT>=1); % divido per 10 solo gli                         160
  % elementi maggiori uguali ad 1
  CLUT = cond.*LUT/10 + ~cond.*LUT;

  ODG = floor(log10(R2_div_R1+eps));
  NEWLUT = exponent(ODG+off)*CLUT;

  MU = abs(NEWLUT-R2_div_R1);

  % cerco la posizione (riga e colonna) nella
  % matrice dell'elemento valore minimo che deve                    170
  % avere un valore inferiore all'errore di guadagno
  [r,c]=find(MU==min(MU(:)) & ...
    GainError*R2_div_R1>=MU);

  count = count +1;
end %end while isempty(r)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% se ho trovato una coppia di resistenze
% si trova il valore fisico                                         180
% adatto per queste ultime, altrimenti
% viene restituito un valore vuoto
if ~isempty(r)
  R2 = Res(r(1))*...
    exponent(ODG+off-(cond(r(1),c(1))));
  R1 = Res(c(1));
  CompError = MU(r(1),c(1))/R2_div_R1;

  % correzione del valore
  % finale della resistenza                                         190
  switch ODG
    case -4,
      Pot = 100000;
    case -3,
```

```
      Pot = 100000;
    case -2,
      Pot = 10000;
    case -1,
      Pot = 10000;
    case 0,
      Pot = 1000;
    case 1,
      Pot = 1000;
    case 2,
      Pot = 1000;
    case 3,
      Pot = 100;
    case 4,
      Pot = 100;
    case 5,
      Pot = 10;
    case 6,
      Pot = 1;
    otherwise
      Pot = 1;
  end

  R2 = R2*Pot;
  R1 = R1*Pot;
  % se l'erroreè  troppo piccolo si riporta
  % semplicemente 0
  CompError = ...
    ~(floor(log10(CompError+eps))<-5)*CompError;
  ChosenSeries = ['E' num2str(Series(count-1))];
  switch ChosenSeries
    case 'E6'
      Tolerance = 20;
    case 'E12'
      Tolerance = 10;
    case 'E24'
      Tolerance = 5;
    case 'E48'
      Tolerance = 2;
    case 'E96'
      Tolerance = 1;
    case 'E192'
      Tolerance = 0.5;  %0.25, 0.1
  end % switch ChosenSeries
end % if ~isempty(r)
```

B.15 *find_supply.m*

```
function [MaxVoltageReference, Vconst] = ...
  find_supply(BlockHandle)

Vconst = '';
MaxVoltageReference = '';
DiagramHandle = bdroot(BlockHandle);

% si cerca all'interno del diagramma il
% blocco dighwinterface, in caso non sia
% presente viene restituito un vettore vuoto.
DigHwInterface = get_digHwInterface(BlockHandle);

if ~isempty(DigHwInterface)
  try
    PosSupply = ...
      eval_param(DigHwInterface, 'sim_PositiveSupply');
    NegSupply = ...
      eval_param(DigHwInterface, 'sim_NegativeSupply');
  catch
    errordlg(['Cannot extract information on '...
        'analogical supply from digHwInterface ' ...
        'block: probably non-numerical '...
        'value inserted.'], ...
      'Missing digHwInterface block', 'non-modal');
    return;
  end % try
else
  errordlg(['Cannot extract information on '...
      'analogical supplies because digHwInterface ' ...
      'block is not present into the block diagram.'], ...
    'Missing digHwInterface block', 'non-modal');
  return;
end % if ~isempty(DigHwInterface)

Vconst = eval_param(BlockHandle, ...
  'Const_Data_RealValue');
pol = sign(Vconst);

switch pol
  case 1
    if Vconst < PosSupply
      MaxVoltageReference = PosSupply;
    else
      errordlg(['Unable to obtain ' num2str(Vconst) ...
          ' from ' num2str(PosSupply) ...
          ' with a voltage divider:' ...
          ' positive supply is too low.'], ...
```

```
        'Analog Supply Error');
      return;
    end
  case -1
    if Vconst > NegSupply
      MaxVoltageReference = NegSupply;
    else
      errordlg(['Unable to obtain ' num2str(Vconst) ...
          ' from ' num2str(NegSupply) ...
          ' with a voltage divider:' ...
          ' negative supply is too high.'], ...
        'Analog Supply Error');
      return;
    end
  otherwise
    errordlg(['Constant value of block' ...
        get_param(BlockHandle, 'Name') ...
        ' cannot be zero: impossible finding' ...
        ' voltage divider components.'], ...
      'Analog Supply Error');
    return;
end
```

B.16 *find_zoh_frequency.m*

```
function [BlockFrequency,varargout] = ...
  find_zoh_frequency(BlockHandle, ...
  BlockFrequency, varargin)

% a partire dal blocco di S/H trova la
% frequenza massima del segnale in
% ingresso. Questo valore àsar necessario
% per stabilire quale sia il
% componente analogico (S/H amplifier)
% ùpi adatto per rappresentare
% l'equivalente circuitale del diagramma
% simulink e anche per calcolare il
% periodo da impostare in simulazione spice.
% si cerca la frequenza massima del
% sistema a partire dai blocchi che
% precedono sim_zoh. Vengono dunque
% analizzati in modo ricorsivo tutti
% i blocchi che precedono sim_zoh èfinch
% non si trova un blocco sorgente che
% genera un segnale ad una determinata
% frequenza. Nel caso siano presenti ùpi
% blocchi viene considerata la

% funzione ricorsiva che cerca i blocchi
% sorgenti connessi al blocco
% inizialmente passato come BlockHandle.
% Di volta in volta BlockHandle
% àcorrisponder all'handle del blocco
% che precede quello attuale, èfinch
% non àverr trovato un blocco sorgente

ExtendedBlockType = ...
  get_param(BlockHandle, 'sim_ExtendedBlockType');

if strcmp(upper(ExtendedBlockType), 'SIM_ANALOGIN');
  BlockFrequency = ...
    RecoverBlockFreq(BlockHandle, BlockFrequency);
  return;
end

PortHandles = ...
  get_param(BlockHandle, 'PortHandles');
InPorts = PortHandles.Inport;
for k = 1:length(InPorts)
  LineHandle = get_param(InPorts(k), 'Line');
  if LineHandle < 0
    warndlg(...
```

```
      ['Unable to find frequency in block''s path: block ' ...
        get_param(BlockHandle, 'Name') ...
        ' is not connected to a source block.'],  ...
      'Unconnected Block Error');
    continue;
  end

  SourceBlock = ...
    get_param(LineHandle, 'SrcBlockHandle');
  SourceBlockPortHandles = ...
    get_param(SourceBlock, 'PortHandles');
  InPortOfSourceBlock = SourceBlockPortHandles.Inport;

  if eval([ExtendedBlockType ...
        '_ParamUI(''AnalogSource'', BlockHandle)'], '0')
    BlockFrequency = ...
      RecoverBlockFreq(SourceBlock, BlockFrequency);
  else
    BlockFrequency = ...
      find_zoh_frequency(SourceBlock, BlockFrequency);
  end
end
```

B.17 *get_string_piece.m*

```
function c = get_string_piece(DataString, varargin)

% in base alla stringa che viene passata
% come argomento e agli eventuali
% delimitatori che vengono indicati in una
% stringa a parte, la funzione
% individua le sequenze di caratteri
% separate dai simboli inclusi nella
% stringa dei delimitatori e memorizza
% in un array di celle tali sequenzeì.
% di caratteri separate.

% varargout = cell(1, nargout);
if nargin < 2
   delim = '@''|';
else
   delim = varargin{1};
end

% si identificano i caratteri della
% stringa diversi da quelli
% elencati nella nella lista dei
% delimitatori e successivamente si
% identificano gli indici della
% stringa che identificano il primo e
% l'ultimo carattere di una sequenza
% valida di caratteri compresa fra due
% delimitatori
ix =~ismember(DataString,delim);
dx = diff(ix);
k = sort([findstr(dx,-1) findstr(dx,1)+1]);

if      ~isempty(k)
  if ix(1) == 1 %
         k=[1 k];
  end

  if ix(end) == 1
         k=[k length(ix)];
  end

else
    % nel caso non sia stato trovato
    % alcun delimitatore la stringa
    %è  restituita identica
    k = [1 length(DataString)];
end
```

```
nk = size(k,2)/2;
c=cell(1,nk); % una riga e nk colonne                                    50
for      i = 1 : nk
    c{i}=DataString(k(1+2*(i-1)):k(2+2*(i-1)));
end
```

B.18 *IsNextPrevBlock.m*

```
function [SearchedBlock, varargout] = ...
  IsNextPrevBlock(BlockHandle, Direction, varargin)

% si utilizza per determinare i blocchi che
% precedono o seguono un altro blocco. Nel caso
% in cui sia passato un terzo argomento d'ingresso si
% restituiscono in uscita gli handle dei
% blocchi estesi del tipo indicato dal terzo argomento,
% altrimenti viene solo restituito il vettore degli
% handle trovati (non si controlla il tipo di blocco)

SearchedBlock = [];
SearchedBlockHandle = [];
PortHandles = get_param(BlockHandle, 'PortHandles');

switch upper(Direction)
    case 'NEXT'
        OutHandle = PortHandles.Outport;
        LineHandle = get_param(OutHandle, 'Line');
        if LineHandle >0
            SearchedBlockHandle = ...
              get_param(LineHandle, 'DstBlockHandle');
        end
    case {'PREVIOUS' 'PREV'}
        InHandle = PortHandles.Inport;
        LineHandle = get_param(InHandle, 'Line');
        if LineHandle >0
            SearchedBlockHandle = ...
              get_param(LineHandle, 'SrcBlockHandle');
        end
    otherwise
        return;
end

if nargin >2
    SearchedExtBlockType = upper(varargin{1});
    for i = 1:length(SearchedBlockHandle)
        try
            ExtdendedBlockType = ...
              upper(get_param(SearchedBlockHandle(i), ...
              'sim_ExtendedBlockType'));
        catch
            ExtdendedBlockType = '';
        end

        if strfind(SearchedExtBlockType, ...
            ExtdendedBlockType)
```

```
            SearchedBlock = ...
              [SearchedBlock SearchedBlockHandle(i)];
        end %if strcmp(                                                    50
    end %for i =
else
   SearchedBlock = SearchedBlockHandle;
end
```

B.19 *PutAnCmpControls.m*

```
function PutAnCmpControls(FigHandle, ...
  LastCtrlBottom, FrameHeight1, FrameHeight2)

% questa funzione permette di aggiungere
% gli oggetti di controllo che descrivono il
% circuito analogico equivalente del blocco
% simulink. A seconda del tipo di blocco
% possono essere presenti al ùpi due frame:
% uno contenente i dispositivi integrati
% che realizzano il circuito e uno
% che contiene i componenti passivi
% (resistenze, condensatori) ed eventuali
% valori con essi calcolati. Nel caso
% in cui uno dei due frame non sia
% incluso non àverr stampato

ControlHeight = 15;
ButtonHeight = 25;

BlockHandle = getfield(...
  get(FigHandle, 'Userdata'), 'BlockHandle');
ExtendedBlockType = ...
  get_param(BlockHandle, 'sim_ExtendedBlockType');
EmptyStr = '';

[LastCtrlBottom] = ...
  AnalogCircuitDescription('Generate', FigHandle, ...
  LastCtrlBottom, FrameHeight1, ControlHeight);

[LastCtrlBottom] = ...
  AnalogDeviceDescription('Generate', FigHandle, ...
  LastCtrlBottom, FrameHeight2, ControlHeight);
```

B.20 *PutDefaultAnCmpButtons.m*

```
function [varargout] = ...
  PutDefaultAnCmpButtons(ParentFig, ButtonClickCallBack)

% aggiunge i pulsanti principali nella
% finestra di dialogo che descrive il
% circuito analogico equivalente di un
% blocco simulink

EmptyStr = '';
HelpClickCallBack = EmptyStr;                                    10
GraphUnits = 'points';
ButtonBottom = 5; ButtonSize = [60 20];
ButtonPrmVect = [ButtonBottom ButtonSize];

BlockHandle = getfield(...
  get(ParentFig,'Userdata'), 'BlockHandle');
ExtendedBlockType = ...
  ButtonClickCallBack(1:strfind(...
  ButtonClickCallBack,'_ParamUI')-1);
                                                                 20
if any(get_param(BlockHandle,'maskhelp'))
  helpFileName = [sim_getenv('CodeSimulinkHelp') ...
      filesep get_param(BlockHandle, 'MaskHelp')];
    if exist (helpFileName) == 2
      HelpClickCallBack = ['system(''' ...
          sim_getenv('CodeSimulinkBrowser') ...
          ' ' helpFileName ''');'];
    else
      HelpClickCallBack = ...
        get_param (BlockHandle, 'MaskHelp');                     30
    end
  else
    helpFileName = ['file:///' ...
        sim_getenv('CodeSimulinkHelp') filesep ...
        'analog_circuit.html#' ExtendedBlockType];
    try
      HelpClickCallBack = ...
        ['web ' helpFileName ' -browser'];
    catch
      HelpClickCallBack = '';                                    40
    end
  end

  % pulsante per ricavare i componenti
  % utilizzabili

  Left = 5;
```

```
uicontrol('Parent', ParentFig, ...
  'Units', GraphUnits, ...
  'Callback', ButtonClickCallBack,...                                    50
  'Position', [Left ButtonPrmVect], ...
  'Style', 'pushbutton', ...
  'String', 'OK', ...
  'Tag', 'OK_PB');

Left = 70; % 80
uicontrol('Parent', ParentFig, ...
  'Units', GraphUnits, ...
  'Callback', ButtonClickCallBack, ...
  'Position', [Left ButtonPrmVect], ...                                  60
  'Style', 'pushbutton', ...
  'String', 'Cancel', ...
  'Tag', 'Cancel_PB');

Left = 135; %155
uicontrol('Parent', ParentFig, ...
  'Unit', GraphUnits, ...
  'CallBack',['AnDescriptionChange; '...
    'AddStoreParameter;' ...
    'AllComponentSearch(''Find'', gcbf);'],...                           70
  'Position', [Left ButtonPrmVect], ...
  'Style', 'pushbutton', ...
  'String', 'Find Comp.', ...
  'Tag', 'Find_PB');

% Help button
Left = 200; %
uicontrol('Parent', ParentFig, ...
  'Units', GraphUnits, ...
  'Callback', HelpClickCallBack, ...                                     80
  'Position', [Left ButtonPrmVect], ...
  'Style', 'pushbutton', ...
  'String', 'Help', ...
  'Tag', 'Help_PB');
```

B.21 *PutPopupPush.m*

```
function [PopupMenuHandle, PushButtonHandle] = ...
  PutPopupPush(FigHandle, Position, CompType, ...
  SelectedCompParamTag, SelectedCompParamName, ...
  CompDescriptionParamTag, CompDescriptionParamName, ...
  CompListParamTag, CompListParamName, ...
  CompIndicesParamTag, CompIndicesParamName)

% questa funzione deposita sull'interfaccia
% quattro oggetti di controllo: tre text box e un
% PopupMenu, in particolare due text box etichettano
% l'altro text box e il PopupMenu. Il PopupMenu
% contiene la lista dei dispositivi che si adattano alle
% specifiche immesse dall'utente mentre il textbox
% contiene la descrizione del dispositivo selezionato

en_button = 'off';
CompDescription = '';
BlockData = get(FigHandle, 'UserData');
BlockHandle = getfield(BlockData, 'BlockHandle');

try
  % se i parametri sono àgi presenti
  % nella maschera vengono recuperati
  SelComp = get_param(BlockHandle, ...
    SelectedCompParamName); %--> nome dispositivo selezionato
  Description = get_param(BlockHandle, ...
    CompDescriptionParamName); % --> descrizione dispositivo s
                               %      elezionato
  CompList = get_param(BlockHandle, ...
    CompListParamName); % --> lista nomi dispositivi adatti
  IndicesComp = get_param(BlockHandle, ...
    CompIndicesParamName); % --> indice dei componenti
                           %     adatti nel database

  %se cè' indico l'ultimo valore selezionato
  Value = PopupConversion(SelComp, CompList);

catch
  % se i parametri non sono presenti nella
  % maschera vengono aggiunti e i
  % valori vengono inizializzati
  SelComp = '-???-';
  add_blockParam(BlockHandle, ...
    SelectedCompParamName, '&', SelComp);

  Description = ['Push the button to find ' ...
      'suitable components according to ' ...
```

```
      'Codesimulink block''s description.'];
  add_blockParam(BlockHandle, ...
    CompDescriptionParamName, '&', Description);

  CompList = '-???-';
  add_blockParam(BlockHandle, ...
    CompListParamName, '&', CompList);

  IndicesComp = '';
  add_blockParam(BlockHandle, ...
    CompIndicesParamName, '&', IndicesComp);

  Value = 1;
end


uicontrol('Parent', FigHandle, ...
  'Units', 'points', ...
  'BackgroundColor', [.8 .8 .8], ...
  'HorizontalAlignment', 'left', ...
  'Position', [230 Position 60 15], ...
  'String', CompType, ...
  'Style', 'text', ...
  'Tag', [CompListParamTag '_text']);



if ~isempty(get_param(BlockHandle, ...
    ['sim_' CompType '_Indices']))
  en_button = 'on';
end


PushButtonHandle = uicontrol(...
  'Parent', FigHandle, ...
  'Units', 'points', ...
  'BackgroundColor', [.8 .8 .8], ...
  'HorizontalAlignment', 'left', ...
  'Position', [272 Position 60 15], ...
  'String', 'Cheap', ...
  'Style', 'pushbutton', ...
  'Enable', en_button, ...
  'Tag', ['Cheap_' CompType]);

Position = Position-20;
PopupMenuHandle = uicontrol(...
  'Parent', FigHandle, ...
  'Units', 'points', ...
  'BackgroundColor', [.7 .7 .7], ...
```

```
  'HorizontalAlignment', 'left', ...
  'Position', [230 Position 105 20], ...
  'String', CompList, ...
  'Style', 'popupmenu', ...                                          100
  'Tag', CompListParamTag, ...
  'Value', Value ... %<-- dipende da SelComp
);

BlockData = setfield(BlockData, ...
  CompListParamTag, CompList);
BlockData = setfield(BlockData, ...
  SelectedCompParamTag, SelComp);

Position = Position - 20;                                            110
uicontrol('Parent', FigHandle, ...
  'Units', 'points', ...
  'BackgroundColor', [1 1 1], ...
  'HorizontalAlignment', 'left', ...
  'Position', [230 Position 105 20], ...
  'String', Description, ...
  'Tag', CompDescriptionParamTag, ...
  'Style', 'text');

BlockData = setfield(BlockData, ...                                  120
  CompDescriptionParamTag, Description);
BlockData = setfield(BlockData, ...
  CompIndicesParamTag, IndicesComp);

set(FigHandle, 'UserData', BlockData);
```

B.22 *PutTextBox2.m*

```
function ControlHandle = PutTextBox2(FigHandle, ...
  Position, Label, ...
  CtrlTag, ParamName)

% depone sulla finestra due textbox
% affiancati orizzontalmente: quello
% a sinistra etichetta quello a destra
% che contiene un valore numerico

BlockData = get(FigHandle, 'UserData');
BlockHandle = getfield(BlockData, 'BlockHandle');

try
  String = get_param(BlockHandle, ParamName);
catch
  String = '-???-';
  add_blockParam(BlockHandle, ...
    ParamName, '&', String);
end

uicontrol(...
  'Parent', FigHandle, ...
  'Units', 'points', ...
  'BackgroundColor', [.8 .8 .8], ...
  'HorizontalAlignment', 'left', ...
  'Position', [10 Position 140 13], ...
  'String', Label, ...
  'Style', 'text', ...
  'Tag', [CtrlTag '_txt']);
ControlHandle = uicontrol(...
  'Parent', FigHandle, ...
  'Units', 'points', ...
  'BackgroundColor', [.7 .7 .7], ...
  'HorizontalAlignment', 'left', ...
  'Position', [150 Position 60 15], ...
  'String', String, ...
  'Style', 'text', ...
  'Tag', CtrlTag);

BlockData = setfield(BlockData, CtrlTag, String);
set(FigHandle, 'UserData', BlockData);
```

B.23 *RecoverBlockFreq.m*

```
function [Frequency, varargout] = ...
  RecoverBlockFreq(SourceBlock, Frequency, varargin)

ExtendedBlockType = ...
  get_param(SourceBlock, 'sim_ExtendedBlockType');

switch upper(ExtendedBlockType)
  case {'SIM_REPEATING_SEQUENCE' 'SIM_ANALOGIN'}
    % si valuta la sequenza temporale su cui
    %è  inizialmente generato il segnale                              10
    try
      t = str2num(...
        get_param(SourceBlock,'sim_time_sequence')...
        );
      %si calcola lo step fra due campioni
      Tsamp = t(2)-t(1);

      % si calcola la sequenza dei valori
      % campionati del segnale
      y = str2num(...                                                 20
        get_param(SourceBlock,'sim_value_sequence')...
        );
      PointNumber = 512;

      % si calcola la trasformata di fourier
      % del segnale per determinare le componenti
      % dello spettro di potenza a frequenza ùpi alta
      Y = fft(y,PointNumber);
      Pyy = Y.*conj(Y)/PointNumber;
      f = 1/Tsamp*(0:PointNumber/2)/PointNumber;                      30

      % si cerca la componente di frequenza con massimo
      % valore dello spettro
      max_pw = max(Pyy(1:PointNumber/2+1));

      % % i seguenti comandi cercano fra ùpi
      % componenti dello spettro per determinare
      % quella massima fra le presenti, la frequenza ìcos
      % ottenuta risulta essere meno precisa
      % main_pw_comp = Pyy(1:PointNumber/2+1) >= 0.7*max_pw;           40
      % MaxFreqIndex = find(main_pw_comp);

      % si determina la componente di spettro massimo
      % che si trova alla frequenza ùpi alta
      MaxFreqIndex = find(...
        ismember(Pyy(1:PointNumber/2+1),max_pw)...
        );
```

```
      MaxFreq = f(MaxFreqIndex(end));
      Frequency = max(MaxFreq, Frequency);
    catch                                                    50
      %errore àgi identificato i precedenza:
      % la frequenza finaleè  l'ultima che
      %è  stata memorizzata
      return;
    end
  case 'SIM_CONSTANT'
    MaxFreq = 0;
    Frequency = max(MaxFreq, Frequency);
  otherwise
end %end switch                                              60
```

B.24 *set_PopupMenu_value.m*

```
function set_PopupMenu_value(FigHandle, ...
  CompSelected, ParamTag1, ...
  CompDescription, ParamTag2, ...
  varargin)
% CompList, ParamTag3, ...
% CompIndex, ParamTag4, ...

% funzione per l'aggiornamento del popup menu
% e degli oggetti di controllo ad esso associati

BlockData = get(FigHandle,'Userdata');
BlockHandle = getfield(BlockData, 'BlockHandle');

% CompList <-- lista nomi dispositivi
%              che soddisfano le specifiche
% IndexItem <-- indici dispositivi
%               nel database
% CompSelected <-- nome primo dispositivi
%                  della lista
% CompDescription <-- descrizione primo
%                     dispositivo

set_param(BlockHandle, ParamTag1, CompSelected);
BlockData = setfield(BlockData, ...
  ParamTag1, CompSelected);
% non lo salvo in figura èxchè  quello cheè  selezionato
% nel PopupMenu

set_param(BlockHandle, ...
  ParamTag2, CompDescription);
BlockData = setfield(BlockData, ...
  ParamTag2, CompDescription);
set(...
  findobj(FigHandle, 'Tag', ParamTag2), ...
  'String', CompDescription);

if nargin > 5
  CompList = varargin{1};
  ParamTag3 = varargin{2};
  CompIndex = varargin{3};
  ParamTag4 = varargin{4};
  set_param(BlockHandle, ParamTag3, CompList);
  BlockData = setfield(BlockData,ParamTag3, CompList);
  set(findobj(FigHandle, 'Tag', ParamTag3), 'Value', 1, ...
    'String', CompList);

  set_param(BlockHandle, ParamTag4, CompIndex);
```

```
  BlockData = setfield(BlockData, ParamTag4, CompIndex);
end

% salvataggio in figura della struttura aggiornata
set(FigHandle, 'Userdata', BlockData);
```

B.25 *sim_analogIn_ParamUI.m*

```
function [varargout] = ...
  sim_analogIn_ParamUI(Action, varargin)

% callback di apertura del blocco sim_analogIn
% permette di creare l'interfaccia di dialogo e
% contiene alcune informazioni importanti come le
% porte analogiche da assegnare.
% In fase di precompilazione gli vengono aggiunti
% dei parametri della maschera.

if nargin > 1
    BlockID = varargin{1};
else
    BlockID = gcbh;
end % if nargin > 1

switch Action
    case 'IsHierarchical'
        varargout{1} = 0;
        return;
    case 'AnalogSource'
        if isempty(IsNextPrevBlock(BlockID, 'previous'))
            varargout{1} = 1;
        else
            varargout{1} = 0;
        end
        return;
    case 'AnalogPorts'
        varargout{1} = 'SINGLE|DIFF';
        varargout{2} = '1|2';
        return;
    case 'NonStandardInputPorts'
        varargout{1} = 1;
        return;
    case 'NonStandardOutputPorts'
        varargout{1} = 0;
        return;
    case 'PreComp'
        try
            get_param(BlockID, 'sim_value_sequence');
        catch
            add_blockparam(BlockID, ...
              'sim_time_sequence', '&', '');
            add_blockparam(BlockID, ...
              'sim_value_sequence', '&', '');
        end
```

```
    otherwise
        [varargout{1}, varargout{2}] = ...
          sim_standardSource_ParamUI(Action, BlockID);
end
```

B.26 *sim_analogOut_ParamUI.m*

```
function [varargout] = ...
  sim_analogOut_ParamUI(Action, varargin)

% callback di apertura del blocco sim_analogOut
% permette di creare l'interfaccia di dialogo e
% contiene alcune informazioni importanti come le
% porte analogiche da assegnare

if nargin > 1
    BlockID = varargin{1};                                          10
else
    BlockID = gcbh;
end % if nargin > 1

switch Action
    case 'IsHierarchical'
        varargout{1} = 0;
        return;
    case 'AnalogPorts'
        varargout{1} = 'SINGLE|DIFF';                               20
        varargout{2} = '1|2';
        return;
    case 'NonStandardInputPorts'
        varargout{1} = 0;
        return;
    case 'NonStandardOutputPorts'
        varargout{1} = 1;
        return;
    case 'PreComp'
        return;                                                     30
    otherwise
        [varargout{1},varargout{2}] = ...
          sim_standardSink_ParamUI(Action, BlockID, 1);
end
```

B.27 *SpiceDescription.m*

```
function [varargout] = ...
  SpiceDescription(BlockHandle, varargin)

% funzione contenente le descrizioni
% in codice spice di determinati blocchi
% simulink. Tali descrizioni non sono altro
% che stringhe contenenti al loro interno
% dei caratteri "@" che indicano in quale
% posizione àdovr essere sostituito un
% determinato valore. Il secondo argomento
% d'uscita, contiene il riferimento ai
% parametri contenuti nella maschera
% di quel blocco, che dovranno essere
% recuperati e inclusi nella descrizione spice.
% Si tenga conto che il primo valore da inserire
% in ogni bloccoè  il nome del blocco stesso
% all'interno del diagramma e àverr incluso
% in sede di compilazione. Inoltre alcuni blocchi
% possono avere dei parametri non inclusi
% all'interno delle maschere, che verranno
% comunque aggiunti e/o calcolati in sede di
% compilazione. In ultimo, si ricorda che alcuni
% blocchi possono avere diverse descrizioni spice
% a seconda del valore di un determinato parametro
% incluso nella maschera, in questo caso la àscelt
% àdipender dal valore attuale di quel detrminato
% parametro

if nargin >1
  ExtendedBlockType = varargin{1};
else
  % si occupa di verificare se sono presenti
  % o meno modelli spice realizzati dalle aziende
  % per i dispositivi utilizzati da quei
  % circuiti
  ActualCircuit = ...
    get_param(BlockHandle,'sim_ActualCircuit');
  switch upper(ActualCircuit)
    case {'SIM_SUM2_++', 'SIM_SUM2_+-', 'SIM_SUM2_-+', ...
          'SIM_SUM2_--', 'SIM_GAIN_P', 'SIM_GAIN_N'}
      varargout{1} = 1;
    otherwise
      varargout{1} = 0;
  end
  return;
end
```

```
switch upper(ExtendedBlockType)
  case 'SIM_CONSTANT'
    %NO alimentazioni!!!
    varargout{1} = ...
      ['.SUBCKT  @  1  2\n'...
        'R1   1   2   @\n' ...
        'R2   2   0   @\n' ...
        '.ENDS\n'
    ];
    varargout{2} = 'R1|R2';

  case {'SIM_REPEATING_SEQUENCE', 'SIM_ANALOGIN'}
    varargout{1} = 'V@   @   0   PWL(@)\n';
    varargout{2} = '';

  case 'SIM_ZOH'
    varargout{1} = ...
      ['.SUBCKT  @  Vin+ Vout+  PulseNode\n' ...
        'E1   2  0  Vin+  0  1\n' ...
        'G1   2  3  VALUE  {V(PulseNode)*V(2,3)}\n' ...
        'R1   3  0  1G\n' ...
        'E2   Vout+  0  3  0  1\n' ...
        'C1   3  0  @\n' ...
        '.ENDS\n' ];
    varargout{2} = '';

  case 'SIM_GAIN'
    gain = eval_param(BlockHandle, 'Gain');

    if gain >1
      % ultimi nodi le alimentazioni
      % + (VCC--> 4) e - (VEE--> 5)
      varargout{1} = ...
        ['.SUBCKT  @  1  3  4  5\n' ...
          'R1   2   0   @\n' ...
          'R2   2   3   @\n' ...
          'X1   1   2   4   5   3   @;\n' ...
          '.ENDS\n' ...
        ];
      varargout{2} = 'R1|R2|OpAmp_Selected';
    else
      if gain < 0
        % ultimi nodi le alimentazioni
        % +(VCC--> 5) e - (VEE--> 6)
        varargout{1} = ...
          ['.SUBCKT @  1  3  5  6\n' ...
            'R1   1   2   @\n' ...
            'R2   2   3   @\n' ...
            'X1   0   2   5   6   3   @;\n' ...
```

```
          '.ENDS\n' ...
        ];
      varargout{2} ='R1|R2|OpAmp_Selected';
    else
      if gain >0 & gain <1
        %NO alimentazioni!!!
        varargout{1} = ['.SUBCKT  @  1  2\n'...
            'R1   1   2   @\n' ...
            'R2   2   0    @\n' ...
            '.ENDS\n'
        ];
        varargout{2} = 'R1|R2';
      else
        errordlg(...
          ['Unsupported gain 0 and 1 in block ' ...
            getfullname(BlockHandle)], ...
          'Parameter Error', 'non-modal');
      end
    end
  end

case 'SIM_SUM2'
  SumType = get_param(BlockHandle, 'Direction');
  switch SumType
    case '++'
      % gli ultimi nodi sono le alimentazioni,
      % i primi due gli ingressi il terzo l'uscita
      varargout{1} = ...
        ['.SUBCKT  @  1  2  5  6  7\n' ...
          'R1   1   3   @\n' ...
          'R2   2   3   @\n' ...
          'R3   4   0   @\n' ...
          'R4   4   5   @\n' ...
          'X1   3   4   6   7   5   @;\n' ...
          '.ENDS\n' ...
        ];
      varargout{2} = 'R|R|R|R|OpAmp_Selected';
    case '+-'
      % gli ultimi nodi sono le alimentazioni,
      % i primi due gli ingressi il terzo l'uscita
      varargout{1} = ...
        ['.SUBCKT  @  1  2  5  6  7\n' ...
          'R1   1   3   @\n' ...
          'R2   3   0   @\n' ...
          'R3   2   4   @\n' ...
          'R4   4   5   @\n' ...
          'X1   3   4   6   7   5   @;\n' ...
          '.ENDS\n' ...
        ];
```

```
        varargout{2} = 'R|R|R|R|OpAmp_Selected';
      case '-+'
        % gli ultimi nodi sono le alimentazioni,
        % i primi due gli ingressi il terzo l'uscita
        varargout{1} = ...
          ['.SUBCKT  @  1  2  5  6  7\n' ...
            'R1   1   3   @\n' ...
            'R2   3   5   @\n' ...
            'R3   2   4   @\n' ...
            'R4   4   0   @\n' ...
            'X1   4   3   6   7   5   @;\n' ...
            '.ENDS\n' ...
          ];
        varargout{2} = 'R|R|R|R|OpAmp_Selected';
      case '--'
        % gli ultimi nodi sono le alimentazioni,
        % i primi due gli ingressi il terzo l'uscita
        varargout{1} = ...
          ['.SUBCKT  @  1  2  4  5  6\n' ...
            'R1   1   3   @\n' ...
            'R2   2   3   @\n' ...
            'R3   3   4   @\n' ...
            'X1   0   3   5   6   4   @;\n' ...
            '.ENDS\n' ...
          ];
        varargout{2} = 'R|R|R|OpAmp_Selected';
    end
  otherwise
    varargout{1} = '';
    varargout{2} = '';
end
```