Introduzione a ScicosLab\Scicos

Lombardi Lorenzo

1 settembre 2011

Indice

Premessa

Il Sistema Operativo su cui sono stati svolti gli esempi presenti nel documento è Windows XP SP3 su cui sono state installate le versioni 4.4b8 di ScicosLab e la versione 1.5.1 di Erika Enterprise, nome in codice "Giacomo", con lo ScicosLab Pack 9.3. Sistemi Operativi diversi e/o versioni successive dei software potrebbero differenziarsi sensibilmente con quanto di seguito descritto, spero comunque che il tutorial vi sia ugualmente d'aiuto.

1.1 ScicosLab

ScicosLab è un pacchetto software open source il cui sviluppo è portato avanti dal gruppo METALAU[1] che racchiude alcuni ricercatori appartenenti ad INRIA[2] ed ENPC[3] basato sulla versione 4 di Scilab e contenente al suo interno un importante toolbox chiamato Scicos.

Scilab è un software di computazione numerica che include, all'interno di librerie, centinaia di funzioni utili ai più svariati utilizzi: dal calcolo matematico alla simulazione di sistemi di controllo. Offre inoltre la possibilità di progettare delle personali funzioni in linguaggio *C*, *C++* o *Scilab*.

Scicos è un toolbox con una interfaccia grafica che permette di modellare degli schemi a blocchi in grado di simulare sistemi dinamici di varia natura, anche per questo toolbox possono essere progettati dei blocchi ad hoc adatti ad esigenze specifiche in linguaggio *C* o *Scilab*.

1.2 Introduzione all'uso di ScicosLab/Scicos

Generalmente è buona norma creare una nuova cartella che racchiuda tutti i file interessanti il progetto in via di lavorazione: una volta creata questa verrà impostata come directory di lavoro di ScicosLab per la sessione corrente (è comunque possibile cambiare cartella di lavoro senza dover riavviare il programma)

Aprite ScicosLab. Per posizionare la directory di lavoro nella cartella desiderata, digitare il comando

```
cd("percorso cartella");
```

Gli apici che racchiudono il percorso della cartella servono per evitare problematiche legate ad eventuali presenze di spazi all'interno del percorso (es.: C:\Documents and Settings\cartella)

Per avviare Scicos, nella barra in alto, sotto la voce "Application", selezionare "scicos" (oppure lanciare il comando 'scicos();' -Senza gli apici!- in ScicosLab)
Si aprirà una finestra. In questa finestra, nel menù in alto, andare sulla voce "Palette" e selezionare "pal tree": comparirà una terza finestra con varie cartelle espandibili contenenti tutti i blocchi ("palette") utilizzabili in Scicos. Aprire il menù "sources" e posizionarsi sul blocco "GEN_SQR"

1 METhodes, Algorithmes et Logiciels pour l' Automatique, http://www-rocq.inria.fr/metalau/
2 Istitut National de Recherche en Informatique et en Automatique, http://www.inria.fr/
3 Ecole Nationale des Ponts es des Chaussèes, http://www.enpc.fr/english/int_index.htm

Fig. 1.1: Blocco GEN_SQR.

(scorrendo lentamente col mouse sopra i vari blocchi presenti nei menù viene visualizzato il nome degli stessi).

Trascinarlo all'interno della finestra di Scicos tramite un singolo click del tasto sinistro del mouse, senza rilasciarlo finchè non si è posizionato il blocco nel punto desiderato.

È comunque possibile spostare un blocco già posizionato nel grafico con lo stesso sistema, oppure cliccando col tasto destro del mouse sul blocco e selezionare la voce "Move".

Andare poi sul menù "Sinks" e fare lo stesso col blocco "CSCOPE".

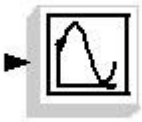

Fig. 1.2: Blocco CSCOPE.

Posizionare il mouse sulla freccia di uscita del blocco GEN_SQR, cliccare col tasto sinistro del mouse e, tenendolo premuto, tracciare una linea che va da questo blocco all'ingresso del blocco CSCOPE

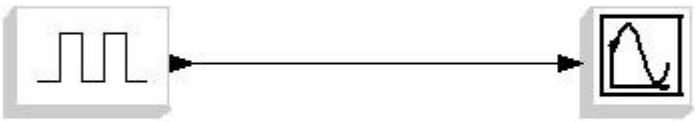

Fig. 1.3: Semplice esempio di collegamento tra blocchi.

Adesso, sulla barra in alto presente nella finestra di Scicos, andare alla voce "Simulate" e selezionare "Run". Si aprirà una quarta finestra dal nome "ScicosLab Graphic" e quello che mostrerà non sarà esattamente l'onda quadra che ci aspettavamo:

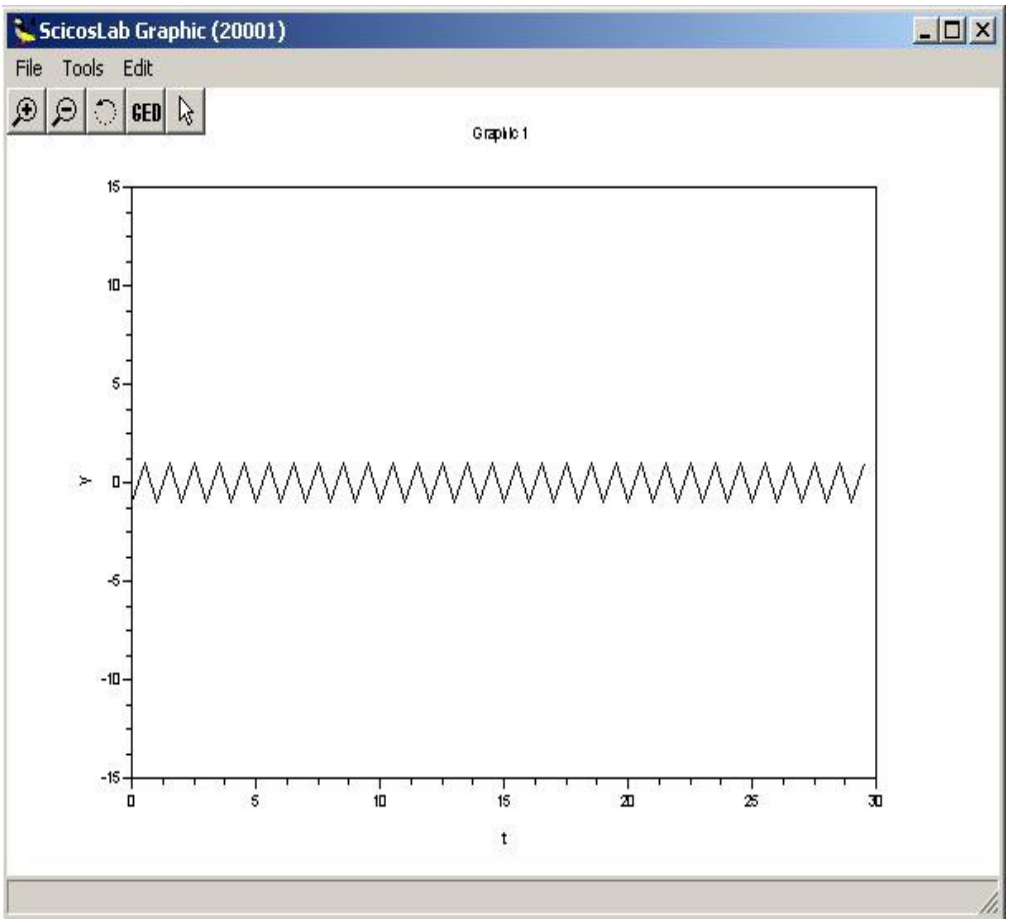

Fig. 1.4: Grafico del blocco CSCOPE risultante dallo schema di Fig. 3.

Il dente di sega visualizzato e' un "effetto di bordo" derivante dalle modalità con cui lo scope unisce due campioni per default. Il meccanismo è' un po' complicato da spiegare e non rientra negli obbiettivi di questa introduzione, quanto invece vedere come risolvere questo inconveniente.

Chiudere la finestra del grafico e, tornando su Scicos, trascinare nello schema il blocco "CLOCK_c" presente nel menù Sources:

Fig. 1.5: Blocco CLOCK_c.

Andare poi sul blocco CSCOPE e, cliccando due volte col tasto sinistro del mouse su di esso (oppure facendo un singolo click col tasto destro e selezionare l'opzione "Open/Set") cambiare nella finestra di impostazione del blocco che si aprirà la voce "*Accept herited events*" da uno a zero come mostrato in Fig. 1.6.

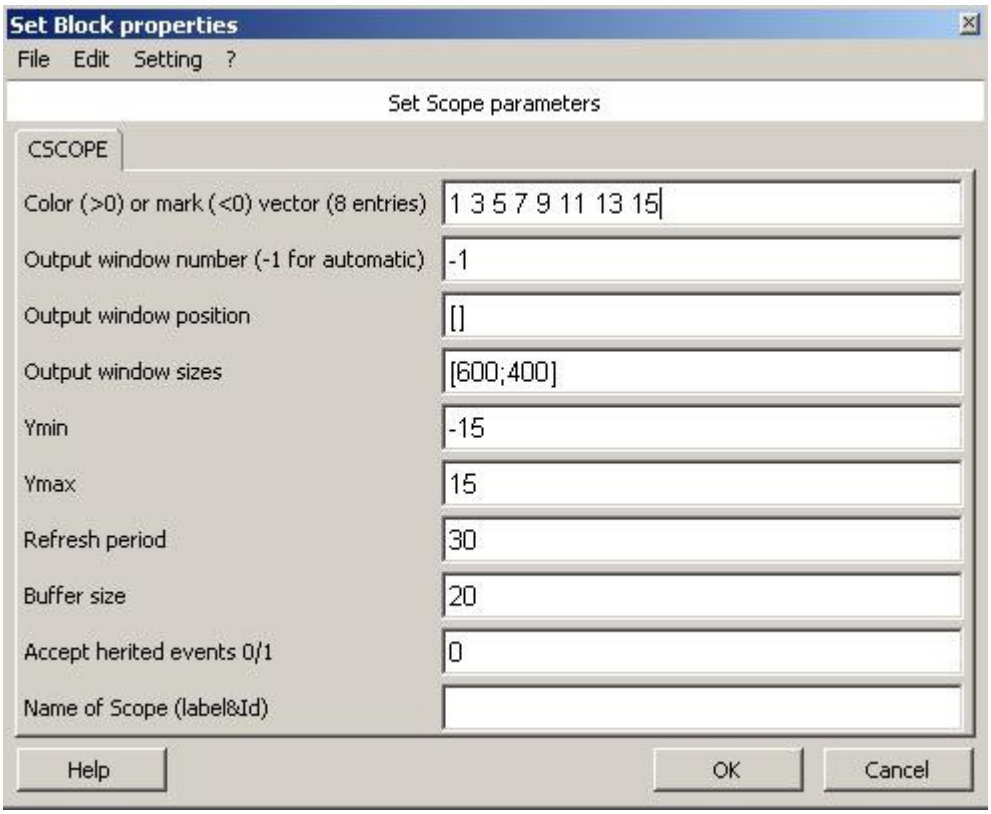

Fig. 1.6: Finestra di settaggio del blocco CSCOPE.

In questo modo sopra il blocco CSCOPE apparirà una piccola freccia rossa: posizionare il mouse sulla freccia di uscita del blocco CLOCK_c e trascinare una linea (rossa) da qui alla nuova freccia rossa di ingresso apparsa sopra il blocco CSCOPE:

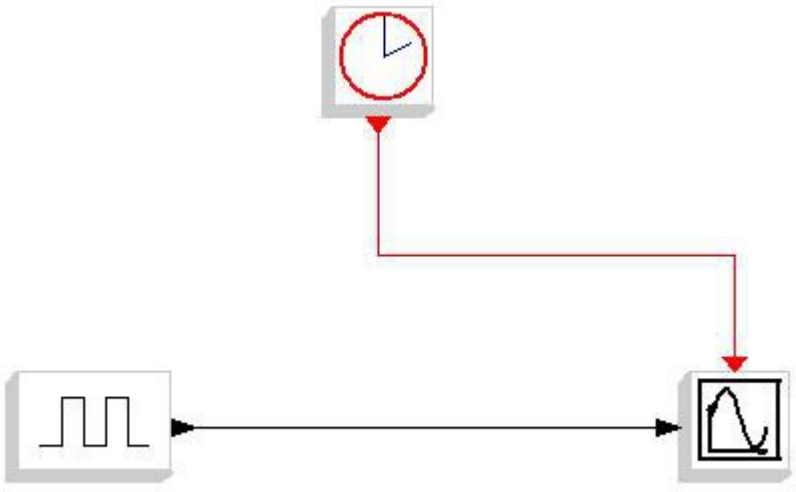

Fig. 1.7: Schema comprendente il blocco di sincronizzazione CLOCK_c.

Per creare dei segmenti tracciando una linea, mentre la si sta tracciando fare un click singolo col mouse nel punto dove si vuole cambiare direzione.

A questo punto, eseguendo di nuovo il comando "Simulate --> run" il grafico mostrerà l'onda quadra desiderata:

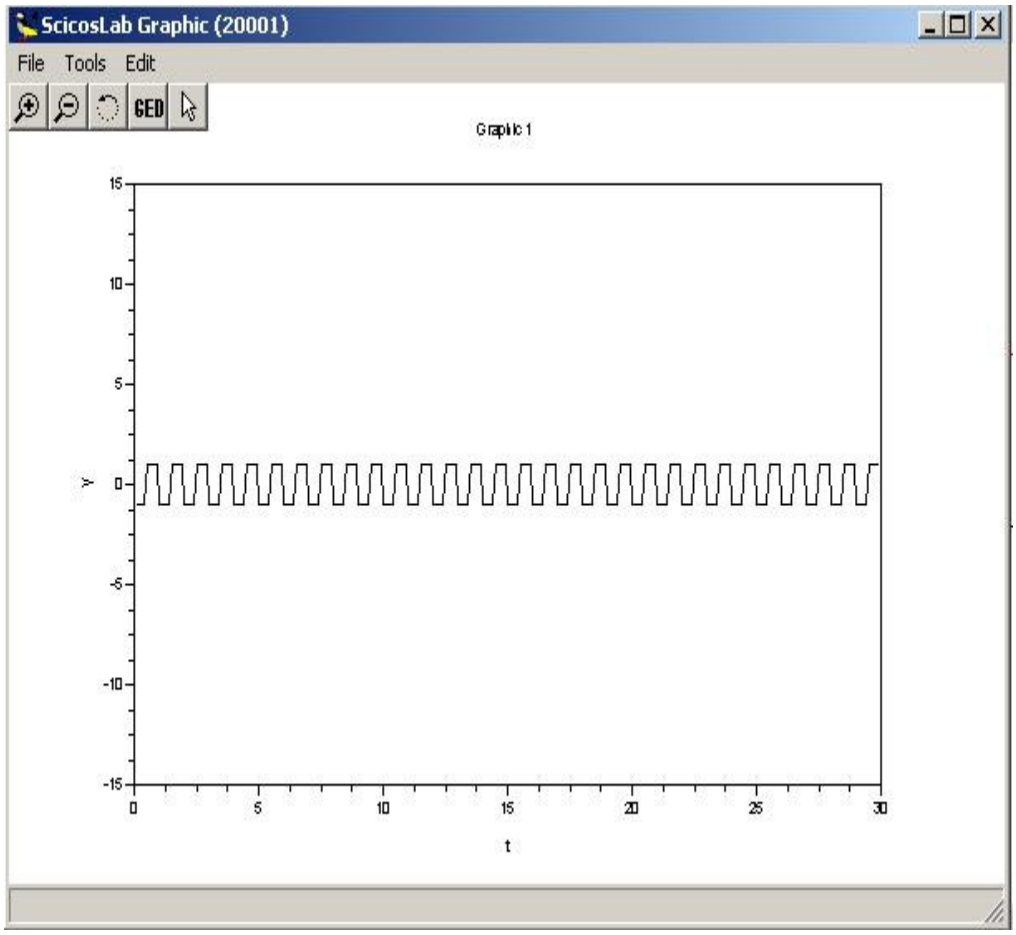

Fig. 1.8: Grafico del blocco CSCOPE risultante dallo schema di Fig. 1.7.

Il blocco CLOCK_c forza i blocchi ad esso collegati a lavorare alla frequenza da lui impostata: cliccando due volte col tasto sinistro del mouse su questo blocco si aprirà la finestra di impostazione delle opzioni del blocco CLOCK_c: la strada migliore per capirne il funzionamento è fare varie prove e vedere cosa succede.

Provate a cambiare la voce "Period" facendola variare tra 0.001 e 1, ricordando che una frequenza di calcolo maggiore *Non sempre* significa maggior accuratezza di simulazione, ma anzi se questa aumenta troppo si rischia di sovraccaricare il sistema con calcoli inutili che nel migliore dei casi non portano alcun giovamento al processo di calcolo e simulazione.

Provate anche a sostituire il blocco "GEN_SQR" con altri blocchi presenti nel menù palette "Soucres", ad esempio col blocco "GENSIN_f" (per cancellare il blocco "GEN_SQR" evidenziarlo cliccandoci una volta col tasto destro del mouse e selezionare l'opzione "Delete": cancellando un blocco scompariranno anche tutte le connessioni che questo ha con gli altri blocchi presenti nel grafico)

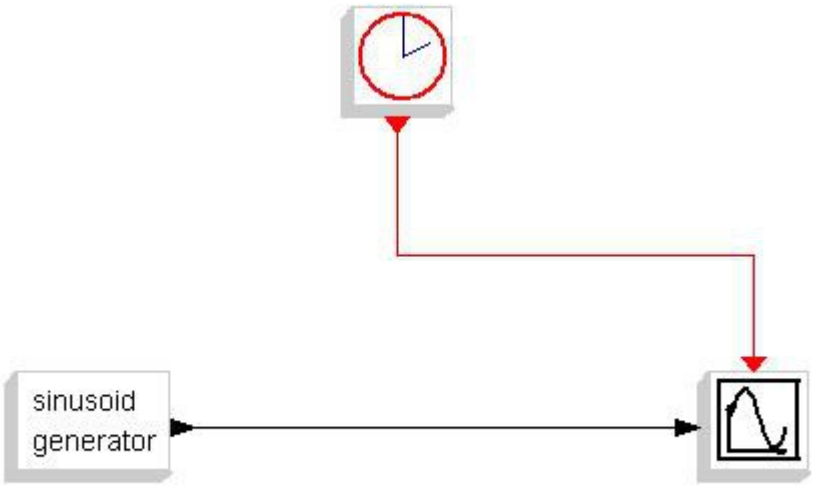

Fig. 1.9: Schema di Fig. 1.7 con generatore sinusoidale.

Al pari del blocco GEN_SQR (e di molti altri) il blocco GENSIN_f permette di impostare le proprietà desiderate (nel caso specifico ampiezza, frequenza e fase) aprendo la finestra di settaggio delle proprietà del blocco tramite il doppio click del tasto sinistro del mouse sul blocco in esame.

Per impostare la durata della simulazione, in Scicos, andare nel menù "Simulate --> setup" e modificare la voce "Final integration time".

Alla voce "Realtiming scaling" è possibile selezionare la velocità di avanzamento della simulazione: 0 = massima velocità possibile dal sistema in uso, 0.5 = il grafico evolve a velocità *Doppia* di come farebbe il sistema reale simulato, 1 = la simulazione evolve in tempo reale, 2 = rallentato di un fattore 0.5 e così via.

Mentre è in corso una simulazione, è possibile fermarla tramite l'opzione "Stop" presente nella barra dei menù in alto nella finestra di Scicos, è possibile poi riavviarla, ricominciarla o terminarla del tutto tramite la funzione "Simulate -> run"

Per modificare la grandezza degli assi del grafico visualizzato, aprire la finestra di settaggio del blocco CSCOPE e modificare i campi "Ymin", "Ymax" e "Refresh period" che indica la lunghezza dell'asse X del grafico: fare attenzione che questa non sia inferiore alla voce "Final integration time" impostata nei parametri di simulazione!

Per modificare le indicazioni presenti sugli assi invece, ad esempio per poter poi utilizzare il grafico creato in una presentazione o simile, fare click sul menù "Edit --> Figure properties" della finestra ScicosLab Graphic e, nella finestra di opzioni che vi apparirà, modificare il campo "Label" presente nel riquadro "Label Options" delle varie schede presenti a destra nel riquadro "Object Properties" che appare selezionando il menù "Axes" presente sulla sinistra. È così possibile modificare le indicazioni riportate sugli assi del grafico, più il titolo del grafico stesso contenuto nel campo "Name of Scope (label&Id)". (vedi Fig. 1.6) Questo sarà utile quando avremo schemi con molti grafici.

Per salvare un grafico occorre, dalla finestra ScicosLab Graphic, eseguire il comando "Save" presente sotto il menù "File" in alto a sinistra, in questo modo (ricordandosi di aggiungere l'estensione ".scg" alla fine del nome desiderato in quanto questo non viene fatto automaticamente da ScicosLab) è possibile salvare il grafico in formato ScicosLab Graphic in modo da poterlo riaprire successivamente per ulteriori analisi (ad esempio tramite il comando "Zoom", le due lenti di ingrandimento presenti sotto il menù file, che consentono di ingrandire una specifica sezione di grafico evidenziata all'interno di un rettangolo che si crea tenendo premuto il tasto sinistro del mouse e trascinandone i limiti all'interno del grafico).

Per caricare un grafico salvato, una volta aperta una finestra ScicosLab Graphic, eseguire il comando "Edit --> Erase figure" e successivamente "File --> Load" caricando il grafico desiderato. Tramite il comando "File --> export" è possibile salvare i grafici anche con altre estensioni (GIF, BMP ecc)

Le variabili in uso possono essere inserite esplicitamente tra i parametri dei blocchi componenti lo schema oppure essere indicizzate in maniera indiretta tramite il CONTEXT dello schema.

Per fare un esempio considerate lo schema seguente, composto dai già visti blocchi GENSIN_f, GEN_SQR, CLOCK_c e CSCOPE, più il blocco PRODUCT presente nel menù palette "Non_linear":

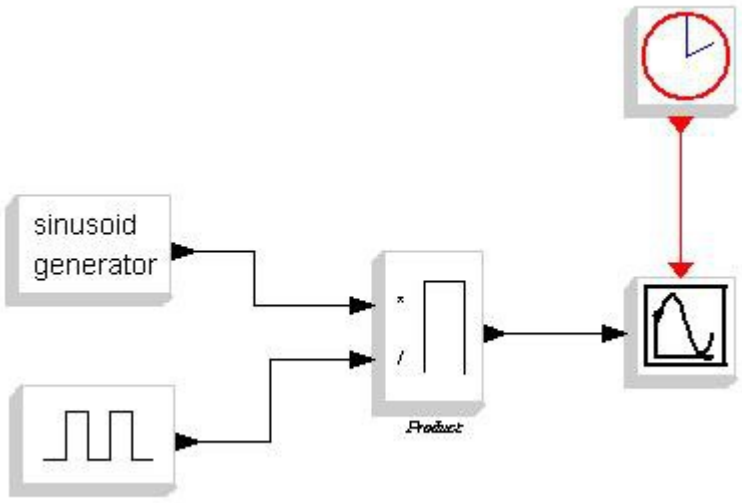

Fig. 1.10: Schema con grandezze indicizzate tramite context.

Il risultato della simulazione, dopo aver opportunamente settato il tempo di calcolo del blocco CLOCK_c, è riportata in Fig. 1.11.

Occorre precisare che per far partire una simulazione è necessario che tutti gli ingressi dei blocchi presenti nello schema siano collegati per evitare l'indeterminazione del loro valore, mentre lo stesso non è obbligatorio per quanto riguarda le uscite.

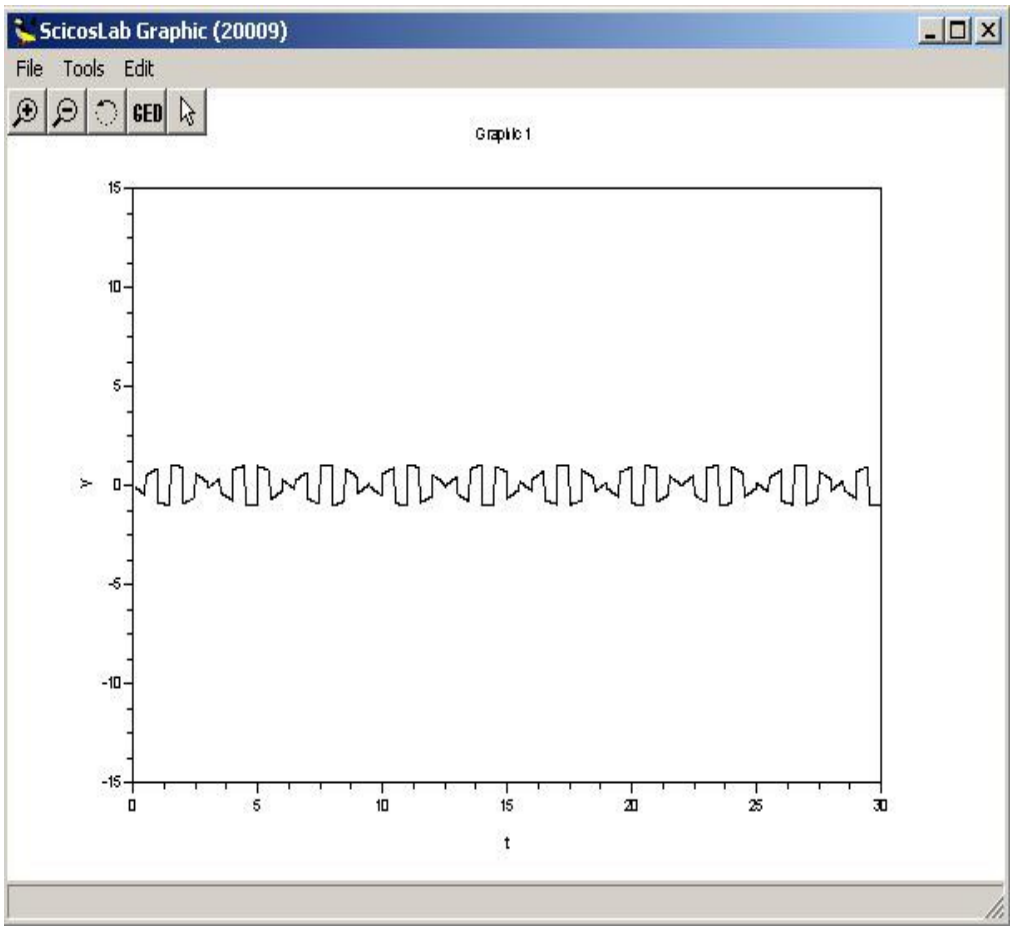

Fig. 1.11: Grafico del blocco CSCOPE risultante dallo schema di Fig. 1.10.

Torniamo adesso sul grafico Scicos e andiamo nel menù "Diagram --> Context" raggiungibile dalla barra dei menù in alto alla finestra di Scicos: si aprirà una finestra di testo in cui inserire i parametri di interesse.

Nella fattispecie inseriamo quanto segue:

```
a  = 2
f1 = 2
b  = 3
f2 = 4
```

Tab. 1.1: Context dello schema di Fig. 1.10.

Clicchiamo "Ok" in basso e torniamo sul grafico Scicos: aprendo con un doppio click del tasto sinistro del mouse la finestra di impostazione del blocco GENSIN_f inseriamo nel campo "Magnitude" il valore "a" definito nel context e nel campo "Frequency (Rad/s)" il valore "f1" come visibile in Fig. 1.12.

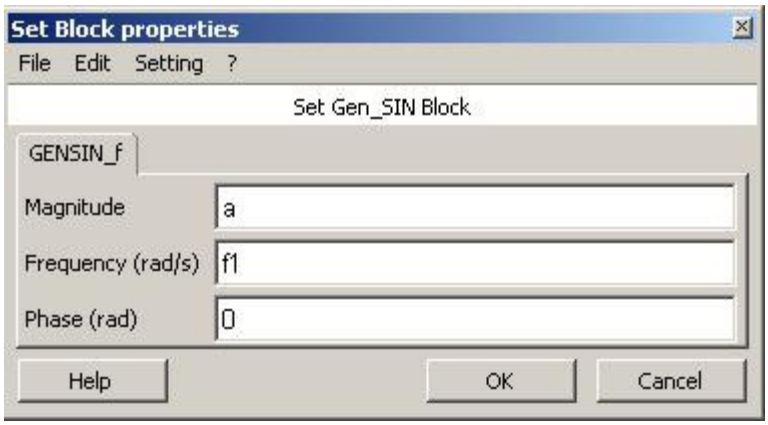

Fig. 1.12: Finestra di settaggio del blocco GENSIN_f.

In questo modo l'ampiezza della sinusoide viene settata al valore relativo alla variabile "a" impostata nel context e la frequenza della stessa al valore f1. Aprite poi la finestra di settaggio delle impostazioni del blocco GEN_SQR e modificate i valori "Minimum value" e "Maximum value" rispettivamente a "-b" e "+b", dopodichè impostare il valore "Period (sec)" al valore "1/f2".

Effettuare nuovamente la simulazione per vedere gli effetti delle modifiche apportate al grafico.

L'utilizzo del context in questo modo può apparire superfluo, ma quando si hanno schemi complessi, composti da molti blocchi annidati uno dentro l'altro in cui la stessa variabile può ripetersi all'interno di svariati blocchi, o in cui la stessa espressione matematica viene ripetuta in più parametri, il context consente di modificare questa espressione o questa variabile una sola volta senza dover andare a ricercare ogni blocco dove questa è utilizzata col rischio di perderne qualcuna e consentendo così di modificare più velocemente gli schemi e di velocizzare il processo di messa a punto della simulazione stessa.

Ma cosa significa avere "Più blocchi annidati uno dentro l'altro"? Quando si hanno schemi Molto grandi, spesso è possibile suddividerli in sezioni funzionali che eseguono scopi precisi, in tal caso per semplificare la comprensione d'insieme dello schema è utile racchiudere queste sezioni in appositi blocchi, detti "Superblocchi", in modo che lo schema generale sia composto da superblocchi funzionali connessi l'uno all'altro.

Per creare un superblocco, cliccare sullo schema Scicos col tasto sinistro del mouse in uno spazio bianco e, senza rilasciarlo, tracciare un rettangolo che includa tutti i blocchi che si vuole siano compresi nel superblocco. Per esempio, tornando allo schema di prima, evidenziare Tutti i blocchi escluso il blocco CLOCK_c (è possibile inserire anche questo nel superblocco, ma a noi conviene non inserirlo) in modo da avere la situazione mostrata in Fig. 1.13.

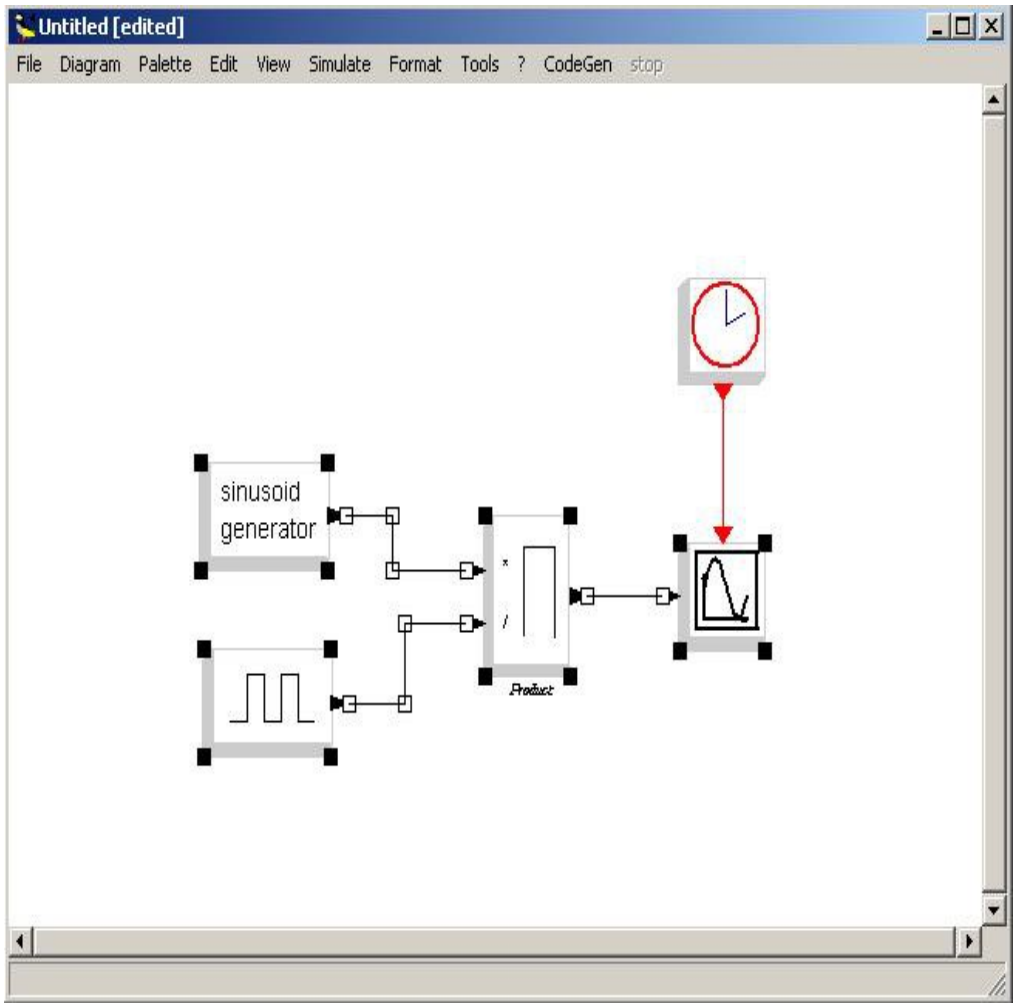

Fig. 1.13: Evidenziamento dei blocchi dello schema di Fig. 1.10 da inserire nel superblocco.

Dopodichè cliccare col tasto destro del mouse e selezionare l'opzione "Region to superblock". Ciò che otterremo sarà uno schema simile a quello riportato in Fig. 1.14.

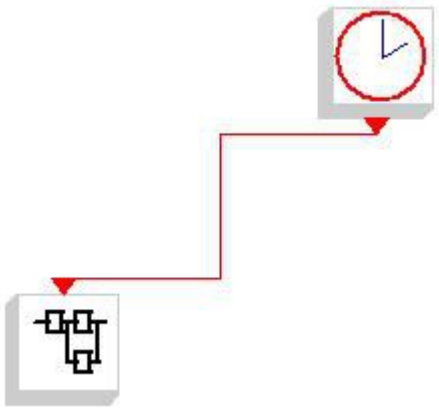

Fig. 1.14: Schema di Fig. 1.10 racchiuso in un superblocco.

Per ottenere uno schema più ordinato è possibile cancellare il collegamento rosso che collega il blocco CLOCK_c al superblocco creato

(posizionandovisi sopra col mouse e cliccare col tasto destro per poi selezionare l'opzione "Delete") spostare detto blocco più vicino e ricollegare i due in maniera meno segmentata, senza perdere alcuna funzionalità. Difatti, eseguendo nuovamente il comando "Simulate --> run" il grafico ottenuto sarà identico a quello ottenuto in precedenza.

Cliccando due volte col tasto sinistro del mouse sul superblocco, o cliccandovi sopra col tasto destro e selezionando l'opzione "Open/set", è possibile aprire un secondo schema Scicos comprendente i blocchi annidati nel superblocco.

Il pentagono irregolare rosso con dentro il numero uno rappresenta il punto in cui, all'interno del superblocco, entra il segnale esterno di controllo proveniente dal blocco CLOCK_c.

Se il superblocco avesse avuto più ingressi, anche non di attivazione, avremmo avuto altri pentagoni (Neri, per i segnali) mentre se il superblocco avesse avuto delle uscite avremmo avuto degli analoghi blocchi pentagonali indicanti però le uscite dal superblocco.

È comunque possibile inserire nuovi ingressi ed uscite anche quando il superblocco è già stato creato. Per aggiungere un ingresso al superblocco, nella schermata che lo rappresenta aggiungere il blocco "IN_f" presente nel menù Sources delle palette

Fig. 1.15: Blocco IN_f.

Per non modificare troppo il nostro esempio, aprire la schermata di impostazione delle proprietà del blocco PRODUCT e modificare il campo "Number of imput or sign vector" da "[1;-1]" a "[1;-1; 1]": in questo modo si aggiunge un ingresso al blocco moltiplicatore.

L'operazione che viene eseguita sugli ingressi è una moltiplicazione se l'ingresso è segnato con un "1" e una divisione se "-1". L'ordine con cui gli ingressi appaiono all'esterno del blocco sono, dall'alto in basso, nello stesso ordine con cui vengono elencati nel vettore di ingresso da sinistra a destra.

In questo modo i collegamenti ai due ingressi precedentemente collegati rimarranno inalterati, ma a causa della diversa disposizione degli ingressi sul lato del blocco PRODUCT le linee di collegamento potrebbero non essere più dritte.

Per modificare il percorso di un collegamento posizionarvisi sopra col mouse ed eseguire un singolo click col tasto sinistro: nei punti dove la linea cambia direzione appariranno dei rettangoli bianchi; andandovi sopra col mouse e premendo il tasto sinistro, senza rilasciarlo, è possibile spostare l'angolo selezionato nella posizione desiderata.

Posizionare il mouse sulla punta del blocco IN_f e tracciare una linea che va da questo al nuovo ingresso al blocco PRODUCT. Aggiungere allo schema anche il blocco "OUT_f" presente nel menù palette "Sinks"

Fig. 1.16: Blocco OUT_f.

Posizionare il mouse a metà della linea che congiunge il blocco PRODUCT al blocco CSCOPE e, tramite un doppio click col tasto sinistro del mouse (o, in alternativa, premendo il tasto "l" ("Link") della tastiera) collegare questa linea al blocco OUT_f. Il risultato finale sarà simile a quello mostrato in Fig. 1.17.

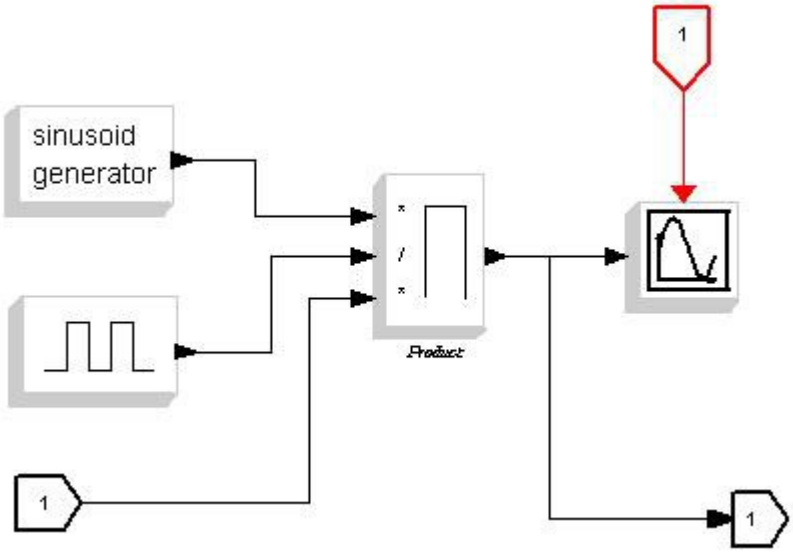

Fig. 1.17: Interno del sulerblocco di Fig. 1.14 con aggiunta di ingressi e uscite.

Adesso, chiudendo la schermata Scicos che racchiude i blocchi all'interno del superblocco e tornando allo schema di origine, il superblocco mostrerà, oltre all'ingresso di attivazione, anche un ingresso di segnale e una uscita, per adesso inutilizzati, come in Fig. 1.18.

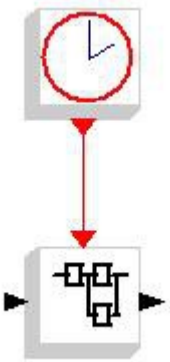

Fig. 1.18: Schema di Fig. 1.14 con ingresso ed uscita al superblocco.

Aggiungere in questo schema un blocco "CONST_m" disponibile nel menù palette "Sources" ed un ulteriore PRODUCT, impostare poi il valore Constant del blocco CONST_m a due e collegarlo all'ingresso creato nel superblocco, e in derivazione da questo collegamento all'ingresso moltiplicatore del blocco PRODUCT, mentre all'altro ingresso porterete il

segnale uscente dal superblocco. Aggiungere poi al grafico anche il blocco "CMSCOPE" presente nel menu palette "Sinks", che consente di visualizzare più di un grafico nella stessa schermata, e attivare l'ingresso degli eventi impostando a zero il valore del parametro "Accept herited events 0/1" come fatto in precedenza per il blocco CSCOPE.

Di default il blocco CMSCOPE accetta due ingressi e visualizza quindi due grafici: all'interno della sua finestra delle impostazioni è comunque possibile aggiungere nuovi ingressi e visualizzare quindi più grafici in una stessa schermata settando opportunamente detti parametri.

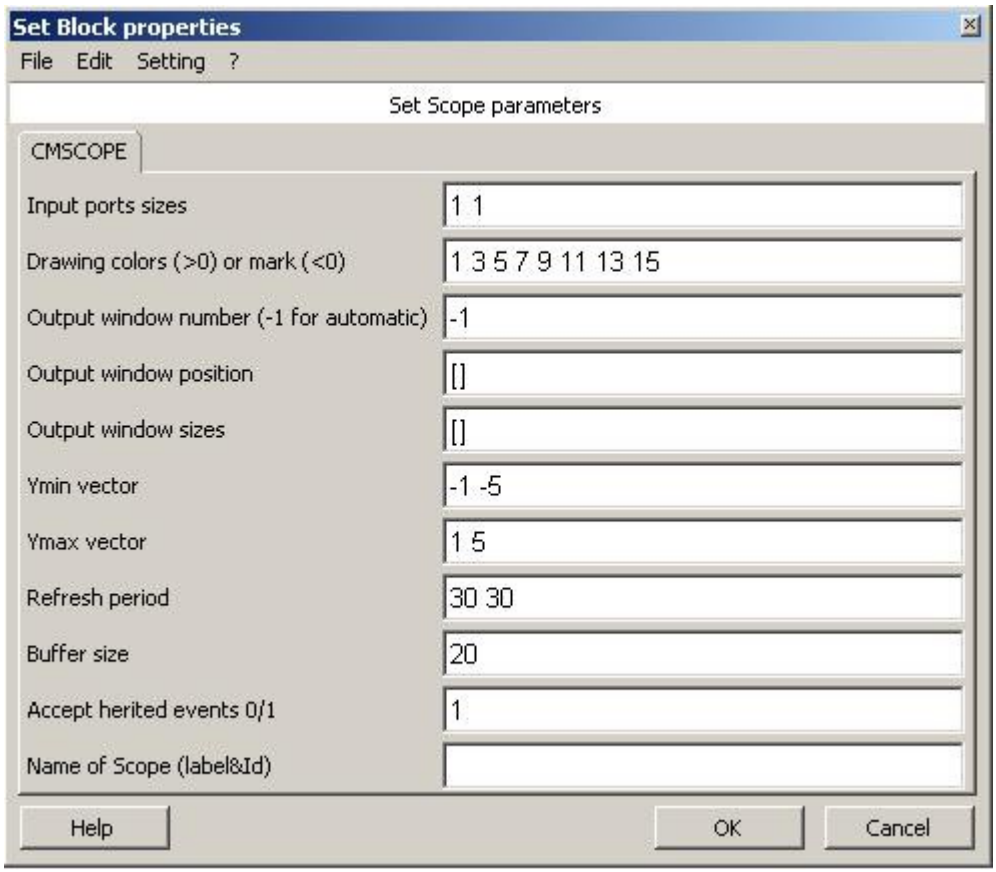

Fig. 1.19: Finestra di settaggio dei parametri del blocco CMSCOPE (Impostazioni di default).

Per avere un numero maggiore di ingressi occorre specificare, per ciascuno di essi, il tipo nel parametro "input port sizes". Un "Uno", come vedremo dopo, corrisponde ad un ingresso scalare reale. Per ciascuno degli ingressi quì dichiarato deve poi essere specificata la grandezza degli assi del grafico alle voci "Ymin vector", "Ymax vector" e "Refresh period", che assumono gli stessi significati del blocco CSCOPE.

Tornando al nostro esempio, collegare agli ingressi del blocco CMSCOPE il segnale di uscita dal superblocco e quello proveniente dal blocco PRODUCT aggiunto, come mostrato in Fig. 1.20.

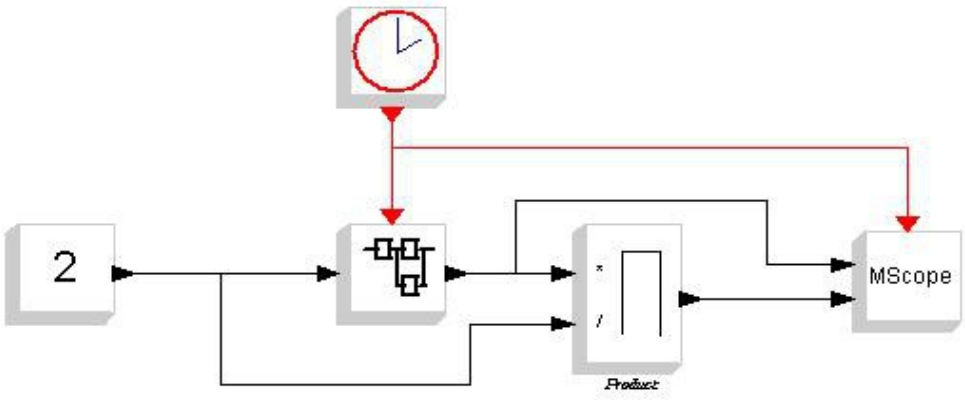

Fig. 1.20: Schema comprendente il superblocco di Fig. 1.14.

Adesso, eseguendo il comando "Simulate --> Run", appariranno due schermate ScicosLab Graphic: una con un solo grafico relativa al blocco CSCOPE interno al superblocco e una con due grafici relativi al blocco CMSCOPE appena inserito. Come per il blocco CSCOPE, anche per il blocco CMSCOPE è possibile modificare le etichette visualizzate sugli assi dei grafici tramite le impostazioni presenti nel menù "Edit --> Figure properties", con la differenza che adesso i menù "Axes" presenti nel riguadro "Objects browser" presente a sinistra sono due.

Notare che, a parte un fattore di scala relativo alle diverse impostazioni dei valori massimi visualizzati sugli assi, il grafico realizzato dal blocco CSCOPE e il primo del blocco CMSCOPE sono identici, in quanto relativi allo stesso segnale.

Quando si hanno schemi complessi è consigliabile associare ai punti di uscita e di ingresso ai superblocchi delle etichette identificative dei segnali che vi devono essere collegati, per fare ciò aprire il superblocco e posizionarsi sui blocchi IN_f e OUT_f di ingresso ed uscita, e tramite la modifica del parametro "Block properties --> Identification" associare una etichetta al blocco di ingresso (o uscita) al superblocco. Nel nostro esempio possiamo associare l'etichetta "in" al blocco di ingresso e "out" al blocco di uscita, e i due schemi così modificati appariranno infine come in Fig. 1.21 e Fig. 1. 22.

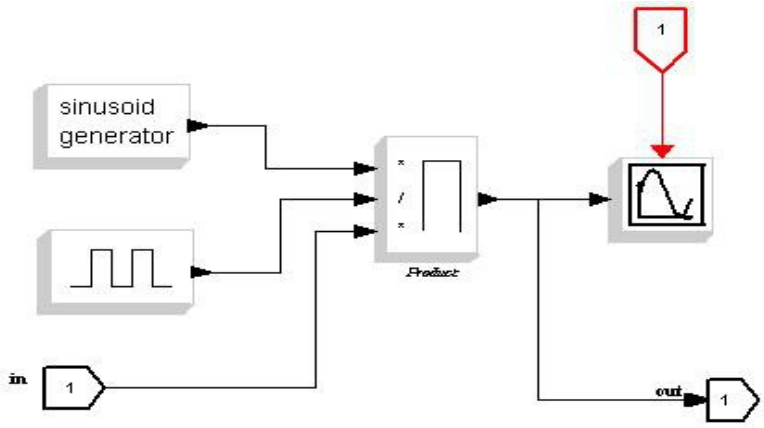

Fig. 1.21: Interno del superblocco di Fig. 1.20 con aggiunta di etichette ai blocchi di ingresso e uscita.

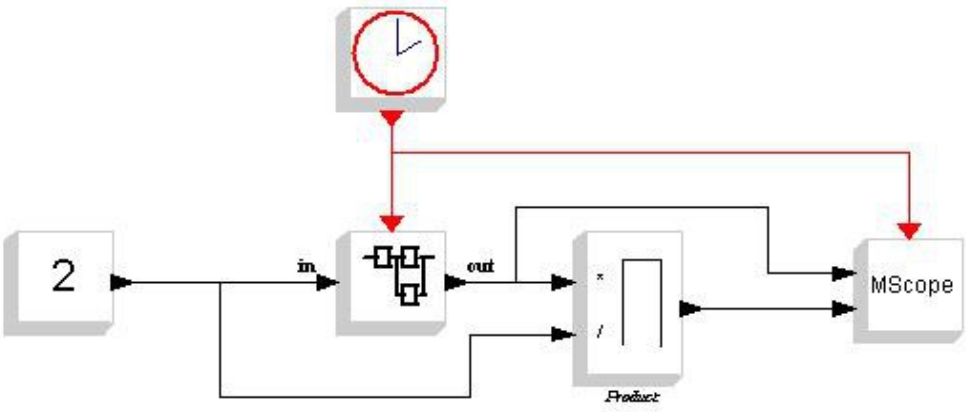

Fig. 1.22: Schema di Fig. 1.20 dopo l'aggiunta delle etichette di ingresso e uscita dal superblocco.

Volendo, è possibile inserire tutto il nuovo grafico in un ulteriore superblocco contenente anche il superblocco precedentemente creato: le variabili definite nel context dello schema generale rimarranno valide a qualunque livello di annidamento dei superblocchi.

Quando si sta visualizzando lo schema relativo al contenuto di un superblocco è possibile fare in modo di associare a questo un nome per non confondersi con altre schemrate relative ad altri superblocchi aperti: per farlo andare nel menù "Diagram --> Rename" ed inserire il nome desiderato per il superblocco.

Al pari dei grafici è possibile salvare anche immagini relative agli schemi Scicos presenti nelle finestre di Scicos: per farlo adoperare il comando "File --> export" tramite il quale è possibile salvare l'immagine come postscript file o tramite una "Graphic windows" che apre una finestra grafica che mostra il contenuto dello schema in esame adattato alle dimensioni dello schermo: una volta fatto questo tramite i comandi "File --> save" o "File --> Export" è possibile salvare l'immagine dello schema nei vari formati .scg, GIF, BMP...

In caso volessimo aggiungere ulteriori ingressi o uscite al superblocco, dovremmo aggiungere come visto ulteriori blocchi "IN_f" e "OUT_f": questi però vengono posizionati sempre col numero "1" segnato al suo interno, e questo causerebbe un errore al momento della chiusura del superblocco modificato. Il numero racchiuso nel blocco infatti rappresenta la posizione che l'ingresso (o l'uscita) relativo all'etichetta hanno all'esterno del blocco, e come è ovvio non è possibile avere due ingressi relativi alla stessa posizione. Per ovviare a questo inconveniente occorre ordinare in maniera crescente le etichette dei blocchi di ingresso e di uscita tramite al modifica del parametro "Port number" presente nella finestra di settaggio delle impostazioni del blocco IN_f o OUT_f, in modo da non creare conflitti.

1.2.1 Aggiungere nuovi blocchi a Scicos

Vediamo adesso come aggiungere un nuovo blocco a Scicos. Occorre fare una piccola premessa: i blocchi di Scicos possono essere costruiti a partire da due soli file di codice: uno che descrive l'azione che il blocco deve compiere, chiamato "Computational function", scritto generalmente in linguaggio C (ma è possibile adoperare anche altri linguaggi di programmazione) e uno che descrive l'interfaccia grafica del blocco in Scicos, definito "Interfacing function", scritto in linguaggio Scilab.

Il consiglio è di testare prima la computational function usando un blocco messo a disposizione da Scicos che consente di scrivere codice personalizzato dentro ad un blocco con interfaccia standard, questo blocco lo si trova all'interno del menù palette "Others" col nome "CBLOCK4"

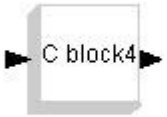

Fig. 1.23: Blocco CBLOCK4.

Il blocco che andremo a realizzare calcolerà la radice quadrata dei suoi due ingressi. Questa stessa funzione potrebbe essere svolta anche dal blocco "POWBLK_f del menù "Non_linear" o dal blocco EXPRESSION presente nello stesso menù, ma a Noi interessa un esempio semplice per descrivere la creazione di un nuovo blocco, non qualcosa di inedito.

Prendete, dal menù palette Sources due blocchi CONST_m e il blocco CLOCK_c, poi, dal menù Sink, per cambiare, il blocco AFFICH_m, poi dal menù Linear, il blocco SUMMATION e dal menù Others il citato CBLOCK4.

Fate doppio click col tasto sinistro del mouse sui blocchi CONST_m e nella finestra delle impostazioni che vi si aprirà impostate il valore del campo "Constant" rispettivamente a dieci e sei.

Aprite la finestra di settaggio delle impostazioni anche del blocco SUMMATION e modificate il valore del campo "Number of imput of sign vector (of +1, -1)" da "[1;-1]" a "[1; 1]": questo campo indica l'operazione che deve essere eseguita all'ingresso corrispondente: "+1" indica che l'ingresso dovrà essere sommato agli altri e "-1" che dovrà essere sottratto. Aggiungendo altri "1" o "-1" è possibile aggiungere altri ingressi al blocco. L'ordine con cui appariranno nell'interfaccia grafica del blocco è lo stesso, dall'alto in basso, di come vengono quì definiti da sinistra a destra.

Se il numero di ingressi dovesse essere considerevole e le freccie di ingresso dovessero risultare troppo vicine, è possibile **modificare le dimensioni del blocco** (come di qualsiasi altro blocco) facendo click col tasto destro del mouse sullo stesso e selezionando la voce "Block Properties --> Resize".

Tralasciando momentaneamente il blocco CBLOCK4, concludiamo la descrizione dello schema dicendo che il blocco AFFICH_m serve semplicemente a visualizzare il valore numerico dell'ingresso che riceve: per la prova che stiamo per fare le impostazioni vanno bene così, il consiglio comunque è sempre quello di fare diversi tentativi con le varie opzioni per vederne gli effetti e capirne il funzionamento in maniera autonoma: la pratica

insegna sempre meglio di mille parole. Al posto di questo blocco avremmo potuto usare il già visto blocco CSCOPE, e trattandosi di un sistema statico avremmo anche potuto omettere il blocco CLOCK_c, comunque andiamo avanti con quanto posizionato e realizziamo lo schema di Fig. 1.24.

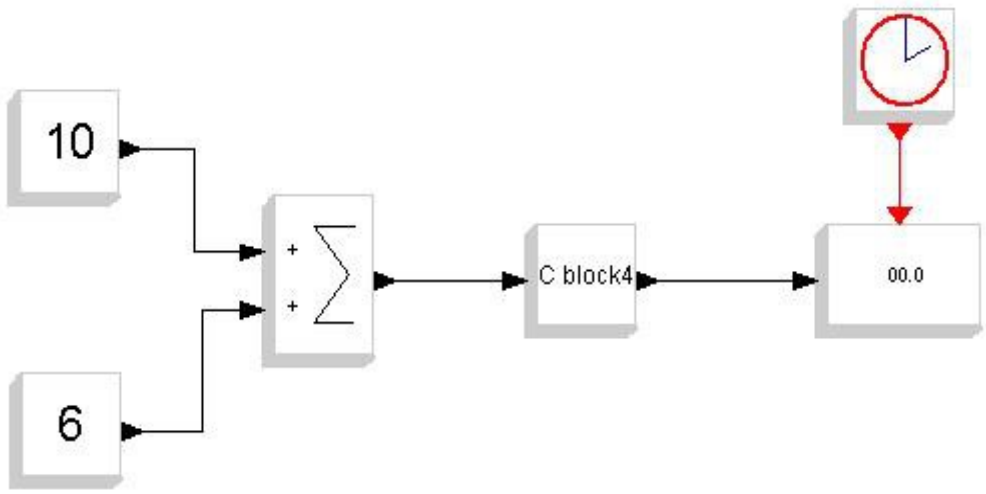

Fig. 1.24: Schema di prova del blocco CBLOCK4.

Adesso dobbiamo inserire il codice nel blocco CBLOCK4: per farlo apriamo la sua finestra di settaggio, che apparirà come in Fig. 1.25.

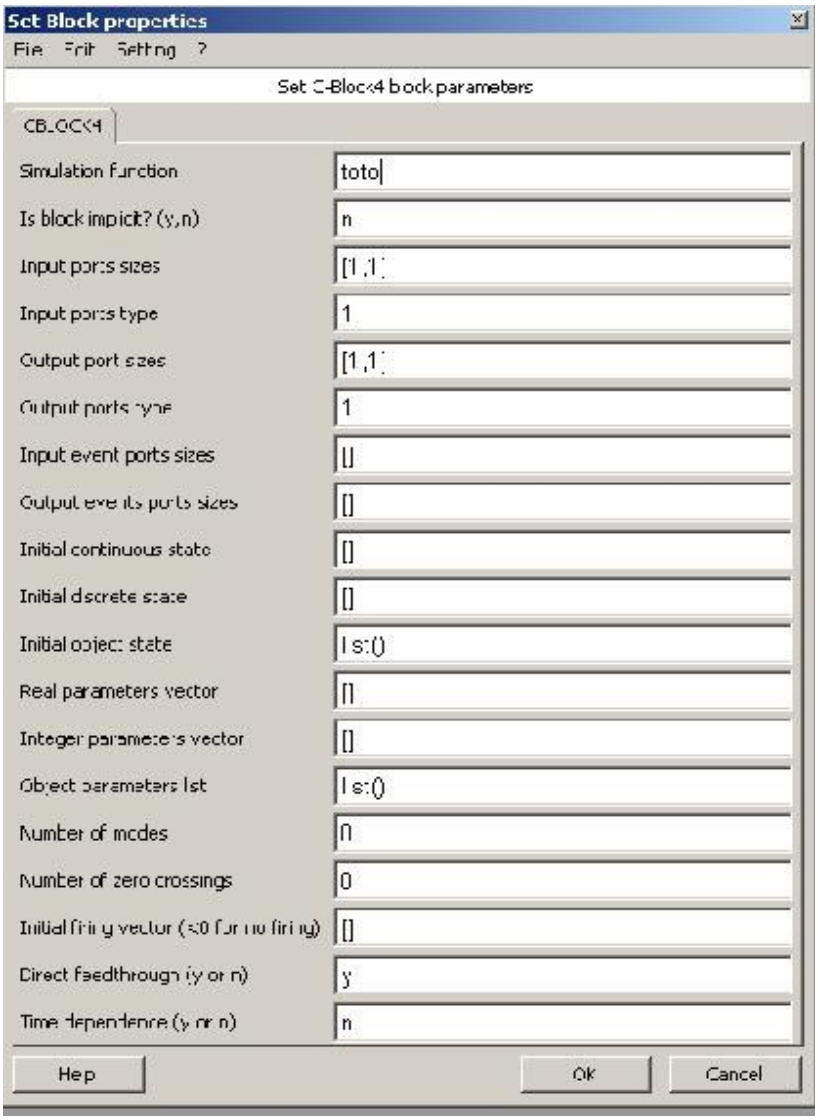

Fig. 1.25: Finestra di settaggio del blocco CBLOCK4.

Il campo "Simulation function" indica il nome della funzione che eseguirà il blocco: il consiglio è di impostare lo stesso nome della computational function scelta, in questo caso ad esempio "Square".

Il campo "Imput port sizes" indica il numero e la dimensione di ingressi del blocco: la scritta "1, 1" indica che abbiamo un ingresso di tipo scalare (1 * 1). Se per esempio volessimo due ingressi dello stesso tipo avremmo dovuto modificare questa voce scrivendo "[1, 1;1, 1]"

La voce "Imput port type" indica il tipo di ingresso al blocco: il numero 1 indica, come già visto, un ingresso reale. Vedremo meglio successivamente come sono associati i tipi di dato a questi numeri in Scicos, per adesso lasciatelo così.

Le voci "Output port size" e "Output port type" indicano le stesse cose rispettivamente delle voci "imput port type" e "imput port size" riferite però alle uscite.

Le voci "Input event port sizes" e "output event port sizes" indicano il numero e il tipo di ingressi e di uscite di eventi riferiti al blocco (le "freccie rosse"). Nel nostro caso non servono quindi possiamo lasciare queste voci

vuote.

La voce "Initial continuous state " indica, per blocchi contenenti uno stato interno, il valore che questo stato deve avere all'inizio della simulazione. La voce è riferita al caso specifico di una funzione relativa ad un ambito tempocontinuo, per blocchi contenenti funzioni tempodiscrete dovremo usare la successiva voce "Initial discrete state ".

Il significato del resto delle voci contenute nel menù, insieme a tutte le altre, è consultabile cliccando sul tasto "Help" in basso a sinistra nella finestra di settaggio delle impostazioni. Il menù "Help" è consultabile per quasi tutti i blocchi nella stessa maniera: dalla finestra di settaggio delle impostazioni tramite la selezione del tasto help.

Proseguendo col nostro esempio quindi, l'unica modifica che dovremo fare alle impostazioni di default è impostare il titolo della funzione in "Square". (sostituire, nel campo "Simulation function" di Fig. 24, il nome "toto" con "Square"). Successivamente si aprirà una finestra intitolata "C function" contenente lo scheletro di base delle computational function di Scicos, che appare come riportato in Tab. 1.2.

```
#include <scicos/scicos_block4.h>

void Square(scicos_block *block,int flag)
{
  /* init */
  if (flag == 4) {

  /* output computation */
  } else if(flag == 1) {

  /* ending */
  } else  if (flag == 5) {

  }
}
```

Tab. 1.2: Struttura base della computational function in C di un blocco Scicos.

I blocchi Scicos definiti con una computational function in C si basano sulla stessa struttura di base tramite la quale è possibile accedere a tutte le informazioni del blocco in esame: ingressi, uscite, parametri e stati interni.

Questa struttura è definita nel file "scicos_block4.h" ed è per questo che occorre includere questo header in tutte le computational function scritte in C.

Durante la simulazione, la computational function viene richiamata con un flag che corrisponde al task che deve essere eseguito. Alla prima esecuzione la funzione viene chiamata col flag = 4 che esegue le procedure di inzializzazione delle variabili (di in ingresso, stato e uscita) del blocco, per poi procedere con il flag = 1 che corrisponde alla computazione delle uscite in base ai valori degli ingressi e dello stato.

In caso siano definite anche delle variabili di stato, successivamente al flag 1 la computational function attiverà il task corrispondente al flag = 2 nel quale vengono aggiornate le variabili di stato definite all'interno del blocco.

Quando la simulazione termina, o in caso di un errore tale da richiedere

l'aborto della simulazione, l'ultimo flag ad essere chiamato è il 5 che richiama le procedure di rilascio di memoria per eventuali variabili interne dichiarate in fase di inizializzazione.

Sono attivabili anche altri flag relativi ad altri task per compiti specifici, ma nell'esempio in esame non ci interessano e trattandosi di funzionalità avanzate rimandiamo la loro spiegazione ad altri tutorial più specifici.

Per realizzare la funzione di calcolo della radice quadrata dell'ingresso la computational function dovrà essere modificata come mostrato in Tab. 1.3.

```
#include <scicos/scicos_block4.h>
#include <math.h>

void Square(scicos_block *block, int flag)
{
  double *in = GetRealInPortPtrs(block, 1);
  double *out = GetRealOutPortPtrs(block, 1);

  /* init */
  if (flag == 4) {
     out[0] = 0;
     }

  /* output computation */
  else if(flag == 1) {
     out[0] = sqrt(in[0]);
     }

  /* ending */
  else  if (flag == 5) {
   }
}
```

Tab. 1.3: Computational function dell' esempio di Fig. 1.23.

Una volta modificato il codice come sopra premere "ok" ed ignorare la successiva scheda in cui ci viene chiesto se linkare librerie esterne, in quanto non ne abbiamo bisogno. Premere "Ok" anche in questa finestra per tornare allo schema Scicos. A questo punto, avviando la simulazione, nel blocco AFFICH_m verrà visualizzato il valore 4.0

Analizziamo riga per riga il codice sopra: la direttiva "`#include <scicos/scicos_block4.h>`" abbiamo già detto a cosa serve, la successiva "`#include <math.h>`" serve ad includere la libreria matematica del C necessaria a poter usare la funzione "sqrt()" che calcola la radice quadrata del numero inserito come parametro.

La riga "`void Square(scicos_block *block, int flag)`" dichiara la funzione "Square" come una funzione che accetta due parametri, un puntatore ad una struttura di tipo "scicos_block" che rappresentante il file stesso e un intero che servirà per selezionare il task da eseguire ogni volta che la funzione verrà richiamata, come descritto in precedenza.

Le successive dichiarazioni

*double *in = GetRealInPortPtrs(block, 1);*
*double *out = GetRealOutPortPtrs(block, 1);*

inizializzano due puntatori a variabili double che rappresentano l'ingresso e l'uscita del nostro blocco, a cui sono associati tramite le macro "GetRealInPortPtrs(block, 1)" e "GetRealOutPortPtrs(block, 1)".

In Tab. 1.4 vengono riportate alcune delle macro più utilizzate per definire gli ingressi e le uscite più comuni per casi semplici:

Macro	ritorno	Descrizione
GetInPortPtrs(block, x)	void*	Restituisce un puntatore alla porta di ingresso "x" specificata.
GetRealInPortPtrs(block, x)	double*	Restituisce un puntatore alla parte reale della porta di ingresso "x" specificata
GetImagInPortPtrs(block, x)	double*	Restituisce un puntatore alla parte immaginaria della porta di ingresso "x" specificata
Getint8InPortPtrs(block, x)	char*	Restituisce un puntatore al carattere digitato nella porta di ingresso "x" specificata
Getint16InPortPtrs(block, x)	short*	Restituisce un puntatore al tipo di dato short presente sulla porta di ingresso "x" specificata
Getint32InPortPtrs(block, x)	long*	Restituisce un puntatore al tipo di dato long presente sulla porta di ingresso "x" specificata
Getuint8InPortPtrs(block, x)	unsigned char*	Restituise un puntatore al tipo di dato unsigned char presente alla porta di ingresso "x" specificata
Getuint16InPortPtrs(block, x)	unsigned short*	Restituisce un puntatore al tipo di dato unsigned short presente alla porta di ingresso "x" specificata
Getuint32InPortPtrs(block, x)	unsigned long*	Restituisce un puntatore al tipo di dato unsigned long presente alla porta di ingresso "x" specificata
GetOutPortPtrs(block, x)	void*	Restituisce un puntatore alla porta di uscita "x" specificata.

GetRealOutPortPtrs(block, x)	double*	Restituisce un puntatore alla parte reale della porta di uscita "x" specificata
GetImagOutPortPtrs(block, x)	double*	Restituisce un puntatore alla parte immaginaria della porta di uscita "x" specificata
Getint8OutPortPtrs(block, x)	char*	Restituisce un puntatore al carattere digitato nella porta di uscita "x" specificata
Getint16OutPortPtrs(block, x)	short*	Restituisce un puntatore al tipo di dato short presente sulla porta di uscita "x" specificata
Getint32OutPortPtrs(block, x)	long*	Restituisce un puntatore al tipo di dato long presente sulla porta di uscita "x" specificata
Getuint8OutPortPtrs(block, x)	unsigned char*	Restituisce un puntatore al tipo di dato unsigned char presente alla porta di uscita "x" specificata
Getuint16OutPortPtrs (block, x)	unsigned short*	Restituisce un puntatore al tipo di dato unsigned short presente alla porta di uscita "x" specificata
Getuint32OutPortPtrs (block, x)	unsigned long*	Restituisce un puntatore al tipo di dato unsigned long presente alla porta di uscita "x" specificata

Tab. 1.4: Elenco delle macro più utilizzate per definire ingressi ed uscite a computational function scritte in C.

Per chiarire meglio la relazione tra il tipo di dato restituito e il nome delle macro, nonchè l'interpretazione dei tipi di dato usata in ScicosLab, possiamo far riferimento a Tab. 1.5 che associa a ciascun tipo di dato espresso in linguaggio Scilab il corrispondente dato in C nonchè il numero con cui questo viene rappresentato nella dichiarazione degli ingressi e delle uscite nei blocchi di Scicos.

Scilab	C	Scicos block
real	double	1
complex	double	2
int32	long	3
int16	short	4
int8	char	5

uint32	unsigned long	6
uint16	unsigned short	7
uint8	unsigned char	8

Tab. 1.5: Corrispondenze tra variabili in linguaggio Scilab e C.

Adesso dovrebbe apparire più chiaro perchè alle voci "Imput port type" e "Output port type" presenti nella finestra di settaggio delle impostazioni del blocco CBLOCK4 abbiamo inserito il numero 1: (Fig.1.24) gli ingressi al nostro blocco saranno definiti come "Real", ovvero come double in linguaggio C.

Proseguendo con l'analisi del codice, nel corpo della funzione "Square" si incontra un controllo di flusso basato sul costrutto switch: abbiamo già descritto a cosa servono i flag di attivazione in una computational function in Scicos, in questi campi andremo a scrivere il codice che nei vari casi dovranno essere via via eseguiti dalla funzione.

Nel nostro caso l'inizializzazione consiste semplicemente nell'azzeramento dell'uscita (flag = 4) mentre per la computazione delle uscite (flag = 1) si pone in uscita il valore della radice quadrata degli ingressi calcolato con la funzione sqrt().

L'utilizzo di una nozione vettoriale per accedere alla porta di uscita (e di ingresso) rappresenta in questo caso una raffinatezza più concettuale che pratica: avendo una porta di uscita scalare avremmo anche potuto assegnare l'uscita tramite il puntatore al nome della stessa:

**out = sqrt(*in);*

Nel caso generico in cui avessimo una porta di uscita vettoriale però, sapendo che il nome di un vettore rappresenta un puntatore al primo elemento dello stesso, per accedere ai vari elementi della porta occorrerebbe indirizzarli in maniera vettoriale, come fatto sopra.

Il flag = 5 risulta vuoto in quanto per la funzione Square non occorre eseguire alcuna operazione specifica al termine della simulazione.

Una volta verificata la validità della *computational function* tramite il blocco CBLOCK4, per aggiungere un nuovo blocco a Scicos, occorre scrivere la *interfacing function* del blocco stesso.

La interfacing function, come detto, è scritta in linguaggio Scilab e purtroppo non si trovano molti tutorial su come procedere: non nascondo che molte delle indicazioni di seguito sono state individuate "Per tentativi" e che quindi Potrebbero essere soggette ad errori, nel qual caso chiedo anticipatamente venia, sperando comunque che risultino ugualmente utili.

Ad ogni modo, come per la *computational function*, anche la *interfacing function* si basa su uno "Scheletro" di struttura base da modificare di volta in volta a seconda del blocco in questione: per la mia esperienza, lo scheletro base di una interfacing function è riportato in Tab. 1.6 (già adattato alla nostra funzione Square)

```
function [x,y,typ]=Square(job,arg1,arg2)
x=[];y=[];typ=[]

select job
case 'plot' then
  standard_draw(arg1)
case 'getinputs' then
  [x,y,typ]=standard_inputs(arg1)
case 'getoutputs' then
  [x,y,typ]=standard_outputs(arg1)
case 'getorigin' then
  [x,y]=standard_origin(arg1)

case 'set' then
  x=arg1;

case 'define' then
  model = scicos_model()
  model.sim = list("Square",4)
  model.in  = [1]
  model.out = [1]
  model.blocktype='c'
  model.dep_ut = [%t %f]
  gr_i =
['txt=[''Square''];';'xstringb(orig(1),orig(2),txt,sz(1),
sz(2),''fill'');']
  exprs=list()
  x = standard_define([2 2],model,exprs,gr_i)
  end
endfunction
```

Tab. 1.6: Interfacing funcion della funzione Square.

Analizziamo nuovamente riga per riga il codice proposto:
La prima riga,

function [x,y,typ]=Square(job,arg1,arg2)

serve all'editor di Scicos per avere un riferimento al blocco ogni volta che deve accedere alle sue funzioni. Per semplicità ho usato lo stesso nome della *computational function* ma questa non è una cosa strettamente necessaria.

Le righe successive in cui viene selezionato un "job" rappresentano l'*interfacing function* vera e propria: le prime 9 righe (righe 4-12) utilizzano le funzioni standard di Scilab per disegnare e posizionare il blocco nello schema ed assegnarne gli ingressi e le uscite (L'*interfacing function* non è del tutto scollegata dalla *computational function*: oltre all'ovvietà per la quale il numero di ingressi ed uscite definite nella *computational function* deve essere lo stesso dichiarato nella *interfacing function*, anche per aspetti in apparenza irrilevanti dal punto di vista grafico -Come la presenza di uno stato interno alla *computational function*- occorrerà prevedere precise definizioni nella *interfacing function*).

Il caso 'set' è adibito alla descrizione della finestra di settaggio dei parametri funzionali del blocco in esame (quando impostiamo il valore della costante interna ad un blocco CONST_m, ad esempio, la finestra che appare tramite il doppio click del mouse sul blocco è definita in questo "Case", e anche quì ovviamente il numero di parametri da impostare deve essere coerente con quello definito nella *computational function*)

Nel nostro caso non abbiamo alcun parametro da impostare quindi ci limitiamo alla riga

```
x=arg1;
```

Che inizializza gli ingressi.

Il case 'define' racchiude la parte più critica di collegamento tra *computational function* ed *interfacing function*: la riga

```
model = scicos_model()
```

indica che la *interfacing function* creata si riferisce ad un blocco Scicos il cui modello è definito dalla funzione "scicos_model()" (così come occorreva inserire la direttiva "`#include <scicos/scicos_block4.h>`" all'inizio della *computational function*)

La riga

```
model.sim = list("Square",4)
```

indica che la *computational function* richiamata dal blocco (SIMulation function) si chiama "Square" ed è scritta in C, quindi il blocco dovrà essere legato ad un file "Square.c".

Come detto, le *computational function* possono essere scritte anche in diversi linguaggi: se ad esempio avessimo scritto una computational function in linguaggio Scilab avremmo dovuto scrivere questa riga come "model.sim = list("Square",5)"

Le righe

```
model.in  = [1]
model.out = [1]
model.blocktype='c'
```

indicano che il nostro blocco avrà un solo ingresso (Se ne avessimo avuti due avremmo dovuto scrivere "model.in = [1; 1]") e una sola uscita (idem come per gli ingressi) e che, nuovamente, la *computational function* è scritta in C.

La riga

model.dep_ut = [%t %f]

serve ad indicare alla *interfacing function* il comportamento della *computational function* tramite due variabili booleane: dept_u (il primo parametro) e dep_t.

Se la variabile dep_u viene impostata a true (%t) il blocco è sempre attivo, ovvero le uscite dipendono costantemente *Anche* dal tempo (Il che non significa che debbano per forza variare!) mentre la variabile dep_t, se vera, indica che il blocco ha almeno una uscita la cui computazione dipende unicamente dai valori in ingresso (senza interessare quindi stati interni o parametri vari).

La riga

gr_i = ['txt=["Square"];';'xstringb(orig(1),orig(2),txt,sz(1),sz(2),"fill");']

rappresenta l'aspetto grafico del blocco: le scritte e la posizione delle stesse, la grandezza del font ecc...

La riga

exprs=list()

racchiude un vettore di stringhe che rappresenta i valori di default visualizzati nei campi di settaggio dei parametri del blocco: poichè il nostro blocco non ha nessun parametro da inizializzare e conseguentemente nessuna schermata di impostazione la lista è vuota.

La riga

x = standard_define([2 2],model,exprs,gr_i)

crea tramite una funzione standard di Scilab il blocco vero e proprio definendone la grandezza ed unendovi le indicazioni date nei vari casi settati precedentemente (i parametri dei campi "model." definiti nel caso 'define', la visualizzazione della finestra di impostazione definita nel case 'set', l'interfaccia grafica del campo "gr_i" ecc) affinchè il blocco sia pienamente funzionale.

Una volta testata la nostra *computational function* e creata la relativa *interfacing function*, occorre salvarle per poterle compilare e linkare in Scilab.

Per farlo aprire il blocco note e copiarci il codice della *computational function*, salvandola in una cartella predefinita come file c (oppure creare appositamente una nuova cartella) e fare lo stesso per la *interfacing function* salvandola come file sci ("Square.sci"). Dopodichè aprire Scilab ed impostare la directory di lavoro sulla cartella in cui sono stati salvati i file di cui sopra (utilizzando la funzione "Change directory" tramite il tasto raffigurante la cartella gialla nella barra delle icone sotto il menù di Scilab, tra le scritte

"Preferences" e "Control", oppure tramite il comando "cd('percorso')")

Una volta posizionata la directory di lavoro nella giusta cartella occorre compilare e linkare la computational function digitando il comando seguente:

libn=ilib_for_link("Square", "Square.o", [], "c", "Makelib");

Questo comando riceve cinque parametri: il primo, "Square", indica il nome della funzione che deve essere linkata in Scilab, il secondo, "Square.o", il nome dell'oggetto che verrà creato in seguito alla compilazione del nostro codice sorgente, il terzo è una lista di librerie esterne necessarie alla creazione dell'oggetto (nel nostro caso nessuna, quindi la lista è vuota) il quarto indica che il codice da linkare è scritto in linguaggio c e il quinto il nome del makefile di default.

In seguito a questo comando, nella cartella predefinita verranno creati sette nuovi file, tra cui un "loader.sce": questo dovrà essere eseguito in Scilab digitando il comando

exec loader.sce;

e, di seguito, provvedere a compilare la interfacing function digitando

exec Square.sci;

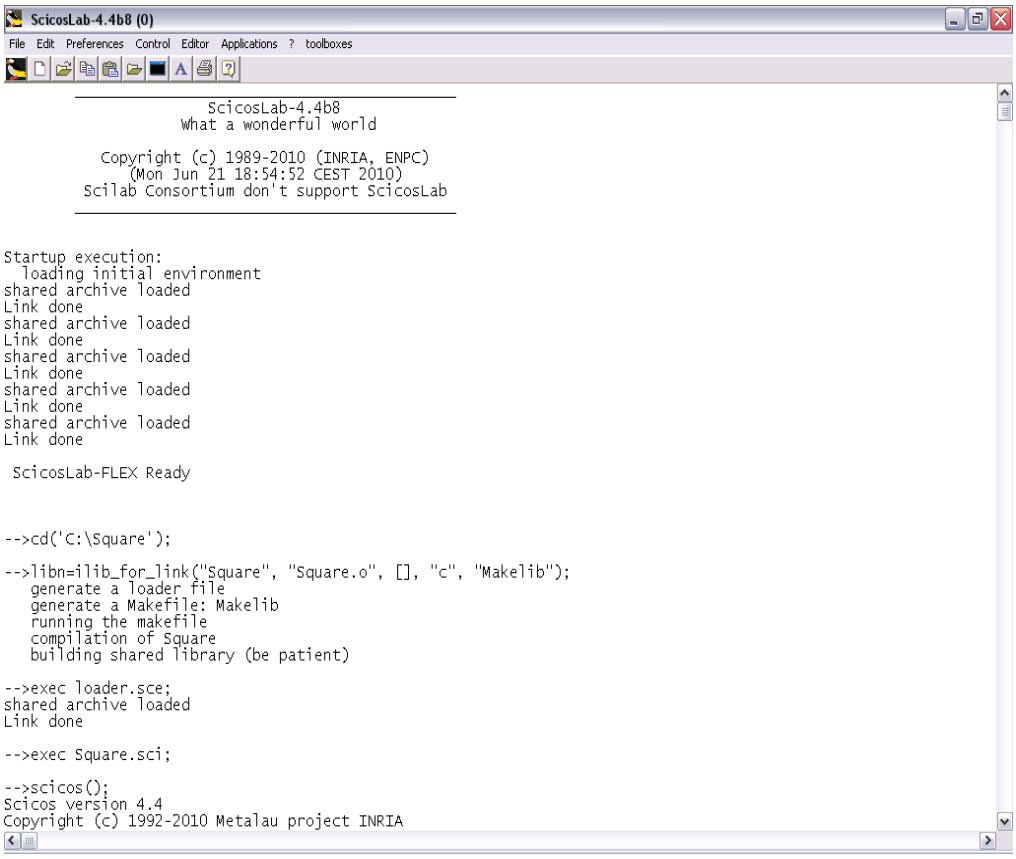

Fig. 1.26: Compilazione e linkaggio delle computational ed interfacing function in ScicosLab/Scicos.

A questo punto aprire Scicos e selezionare l'opzione "Add new block" disponibile sotto il menù "Edit" come in Fig. 1.27.

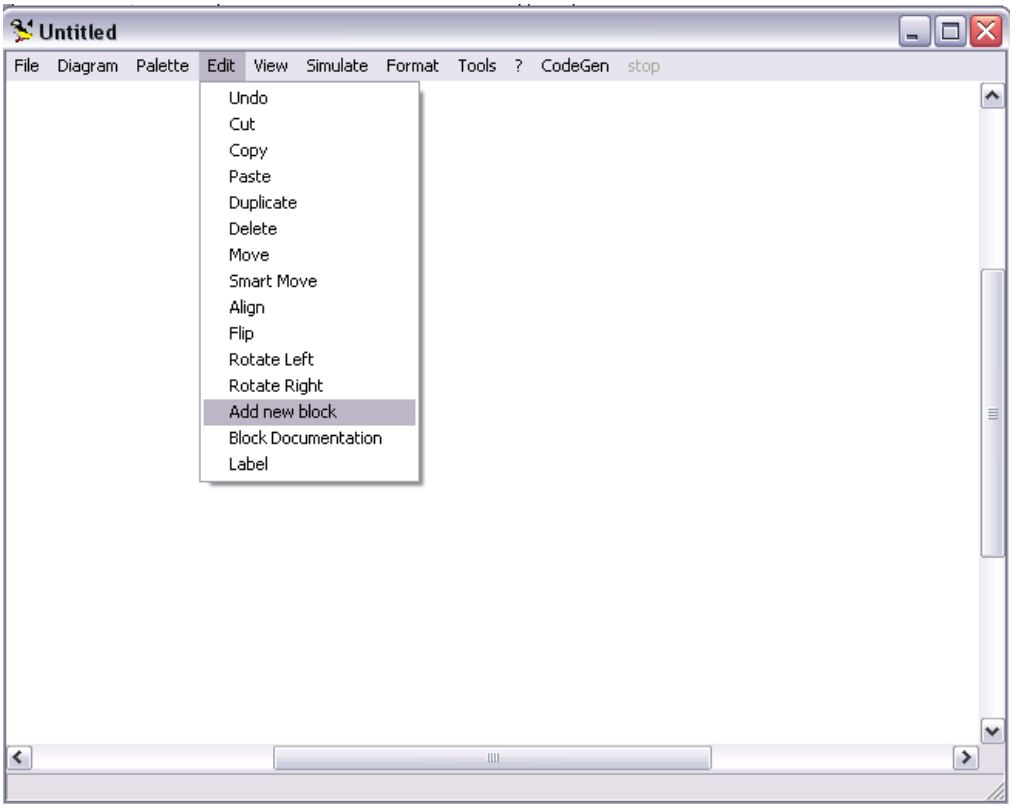

Fig. 1.27: Percorso per aggiungere un nuovo blocco a Scicos.

Nella finestra che apparirà (vedi Fig. 1.28) inserire "Square" e digitare "Ok"

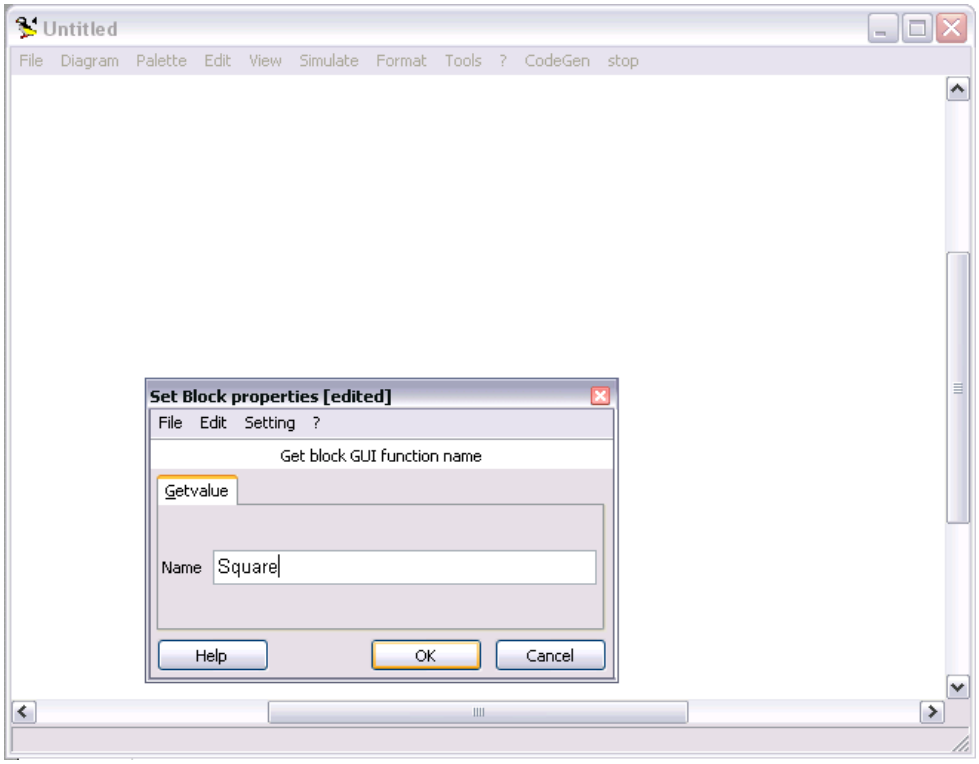

Fig. 1.28: Selezione del nome del nuovo blocco da aggiungere allo schema.

Se tutto è stato fatto correttamente vedremo comparire nello schema il nostro nuovo blocco, che apparirà come in Fig. 1.29.

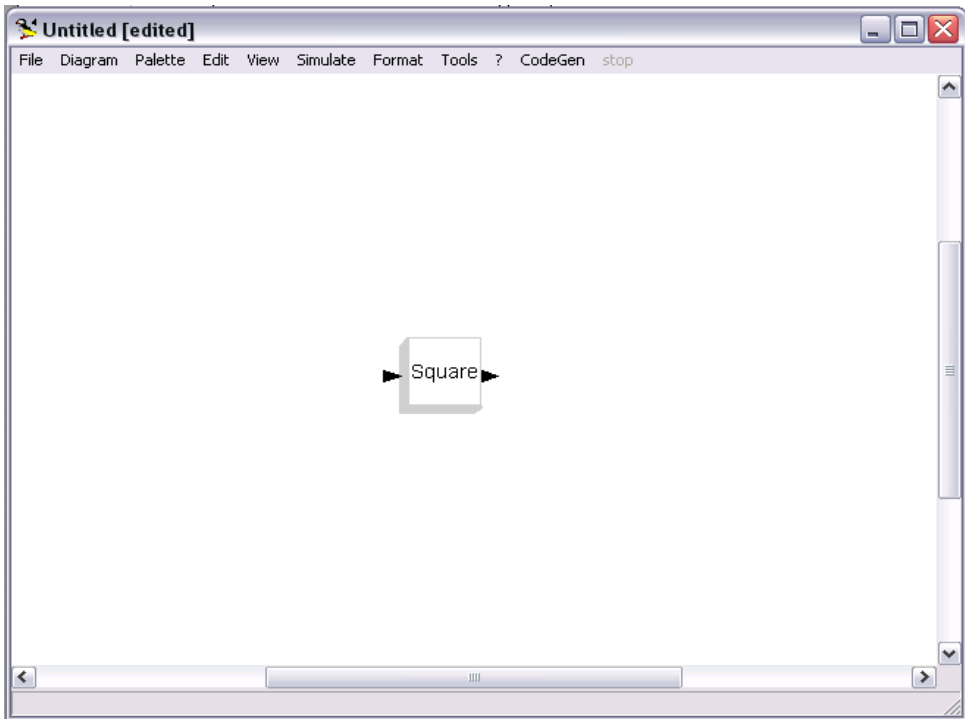

Fig. 1.29: Il nuovo blocco creato posizionato in Scicos.

Per provare il nuovo blocco creato insieme alla interfacing function non rimane che creare uno schema di prova: il risultato dovrebbe essere identico a quello ottenuto con il CBLOCK4 contenente la *computational function* precedentemente testata.

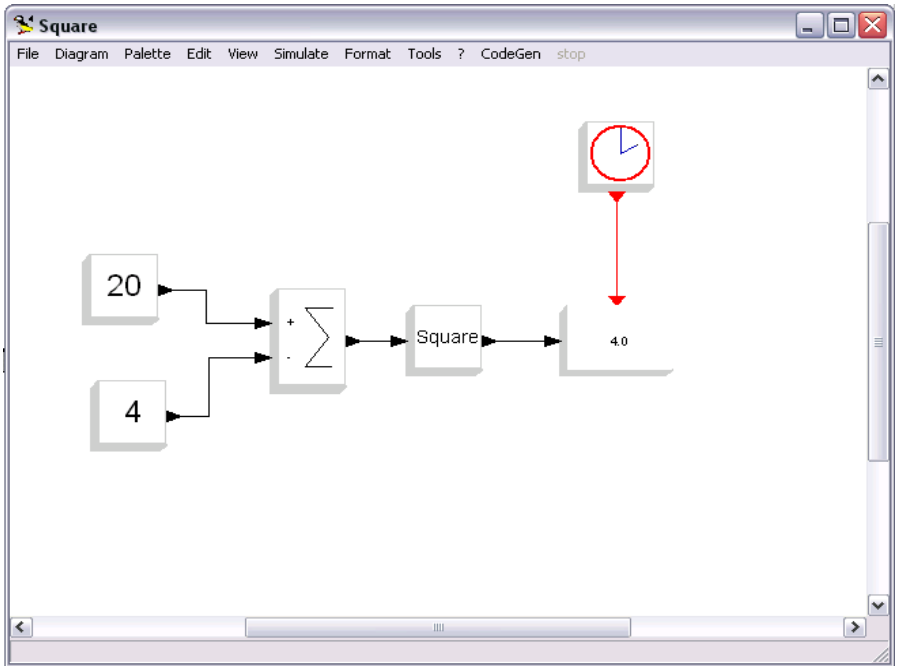

Fig. 1.30: Schema Scicos contenente il nuovo blocco creato.

Se, lavorando su Scicos, avessimo bisogno di tornare in ScicosLab per cambiare directory di lavoro o eseguire calcoli, è possibile farlo senza chiudere la finestra di Scicos tramite l'opzione "Activate ScicosLab Window" presente sotto il menù "Tools"; una volta finito il lavoro su ScicosLab sarà possibile tornare su Scicos digitando "scicos();" oppure tramite il menù "Applications" di ScicosLab (oppure, più semplicemente, cliccando nuovamente sulla finestra di Scicos)

1.3 Bibliografia

[1] TESI, Ing. Lorenzo Lombardi, *"Implementazione di algoritmi di controllo della traiettoria e di comunicazione per un robot a controllo remoto e relative verifiche sperimentali*", Laboratorio di ultrasuoni e controlli non distruttivi, università degli studi di Firenze, Italia, 2011.

[2] Stephen L. Campbell, Jean-Philippe Chancelier, and Ramine Nikoukhah *"Modeling and Simulation in Scilab/Scicos with ScicosLab 4.4"*, second edition, Springer.

[3] Phil Schmidt, *"Creating a C Function Block in Scicos"*, tutorial online, http://www.scicos.org/ScicosCBlockTutorial.pdf

[4] Scicos Team, *"Constructing new blocks in Scicos"*, tutorial online, http://www.scicos.org/Formation_scicos_mars_2008.pdf