Progetto Controllo Digitale

Prof. Aldo Balestrino

Studio e modifica di un sistema ad anello chiuso in ambiente RTAI-Lab con aggiunta di nuove caratteristiche.

Francesco Serafini e Stefano Zingarini

Indice

Appendice

Real-Time

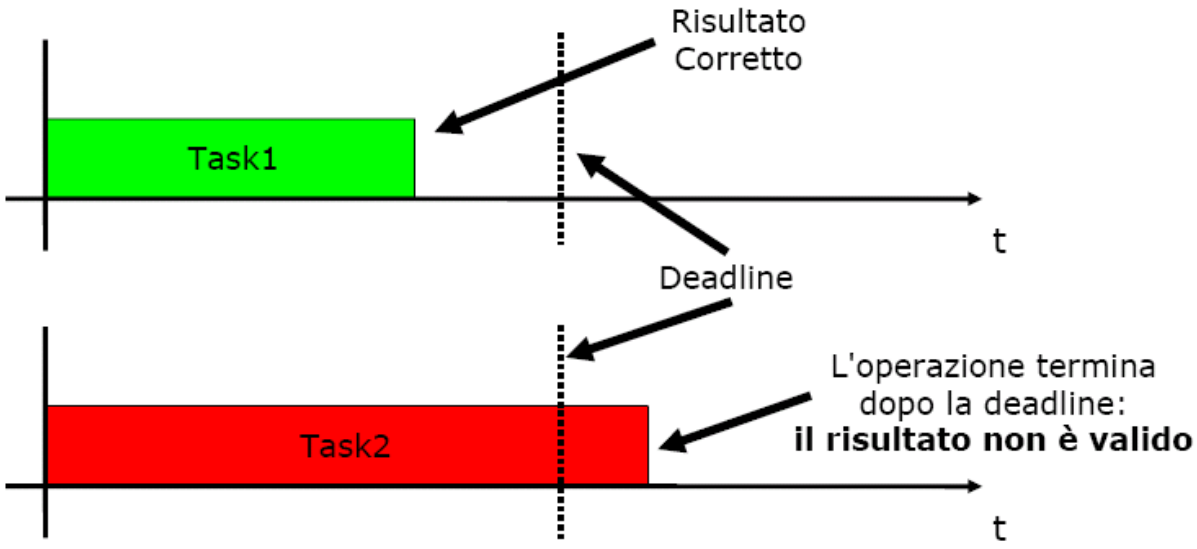
Real-time (*tempo reale*) è un termine utilizzato in ambito informatico per indicare quei programmi per i quali la correttezza del risultato dipende dal tempo di risposta. Ciò comporta che tali programmi devono rispondere ad eventi esterni entro tempi prestabiliti.
Il termine viene usato per quelle applicazioni dove le problematiche relative ai tempi di risposta sono preponderanti rispetto alla complessità dell'algoritmo da utilizzare.
All'interno delle problematiche da trattare in tempo reale si tende a distinguere le applicazioni in *tempo reale stretto* in cui i tempi di risposta si esprimono tipicamente in millisecondo o comunque in frazioni di secondo, da un tempo reale più lasco, in cui i tempi di risposta sono normalmente esprimibili in secondi.
Per la risoluzione di problematiche real-time si usano di solito architetture hardware dedicate, sistemi operativi appositamente concepiti, programmi applicativi pensati appositamente.
Le tre componenti (hardware, software di base, software applicativo) sono spesso strettamente legate, in modo da conseguire le necessarie ottimizzazioni sui tempi.

Perché real-time?

Rispondere ad un evento in tempo reale significa farlo ad una velocità che sia possibile predeterminare, quali che siano i contesti operativi dell'apparato. Questo richiede una architettura hardware in grado di gestire e registrare gli eventi esterni e di un software in grado di reagire su più processi per attuare una reazione adeguata.

Definizione di sistema operativo real-time

E' un sistema operativo in cui per valutare la correttezza delle operazioni si considera anche la variabile tempo:



Due tipologie di correttezza devono essere garantite:

- Correttezza logica: i risultati/risposte forniti devono essere quelli previsti (normalmente richiesta a qualunque sistema di elaborazione)

- Correttezza temporale: i risultati devono essere prodotti entro certi limiti temporali fissati (deadlines) (specifica dei sistemi real-time)

Caratteristiche dei S.O. real-time (RTOS)

Tipologie di Sistemi real-time

- Sistemi Hard real-time:
  Il non rispetto delle deadlines temporali NON e' ammesso, porterebbe al danneggiamento del sistema (applicazioni safety critical)

- Sistemi Soft real-time:
  Il non rispetto occasionale delle deadlines e' ammissibile, le specifiche temporali indicano solo in modo sommario i tempi da rispettare degrado delle performace accettabile.

In generale i sistemi real time complessi sono ibridi hard/soft

Vi saranno deadline inderogabili:

- gestione delle condizioni di allarme pericolose
- controllo digitale in retroazione
- reazione ad eventi interni

Mentre altre non saranno stringenti:

- effettivo avviamento di una macchina dopo il comando ricevuto da un sistema di supervisione
- salvataggio dei dati su dispositivi di archiviazione
- interazione con l'uomo

Spesso si confonde il concetto di Hard e Soft real-time con quello di real-time "Stretto" e "Largo"

- Real-time "Stretto": vincoli temporali (deadlines) stretti rispetto ai tempi di calcolo necessari per eseguire le operazioni richieste.
- Real-time "Largo": vincoli temporali (deadlines) larghi rispetto ai tempi di calcolo necessari per eseguire le operazioni richieste.

La "larghezza" o "strettezza" di un sistema di elaborazione dell'informazione real-time dipende dalla piattaforma hardware utilizzata.
I tempi di esecuzione dipendono dalla 'potenza' dell'unità di elaborazione (velocità + risorse). Purtroppo spesso i sistemi Hard Real-Time sono anche stretti.
Normalmente le deadlines vicine sono anche inderogabili e per motivi di costo si sceglie sempre una piattaforma di elaborazione non sovrabbondante rispetto alle esigenze ne consegue massimo sfruttamento delle risorse HW forte legame tra il codice e l'HW di elaborazione difficile abbinare astrazione e ottimizzazione.
Nel sistemi REAL-TIME STRETTI e' fondamentale la corretta organizzazione delle fasi di elaborazione per il rispetto delle deadlines.

- organizzazione delle attività di elaborazione al fine di ottimizzare i tempi
  - efficienza e flessibilità dello scheduler
  - gestione efficiente delle priorità
- necessità di ridurre al minimo l'indeterminazione nell'esecuzione dei processi
  - minimizzazione dei tempi di latenza
  - massimizzazione della preemption

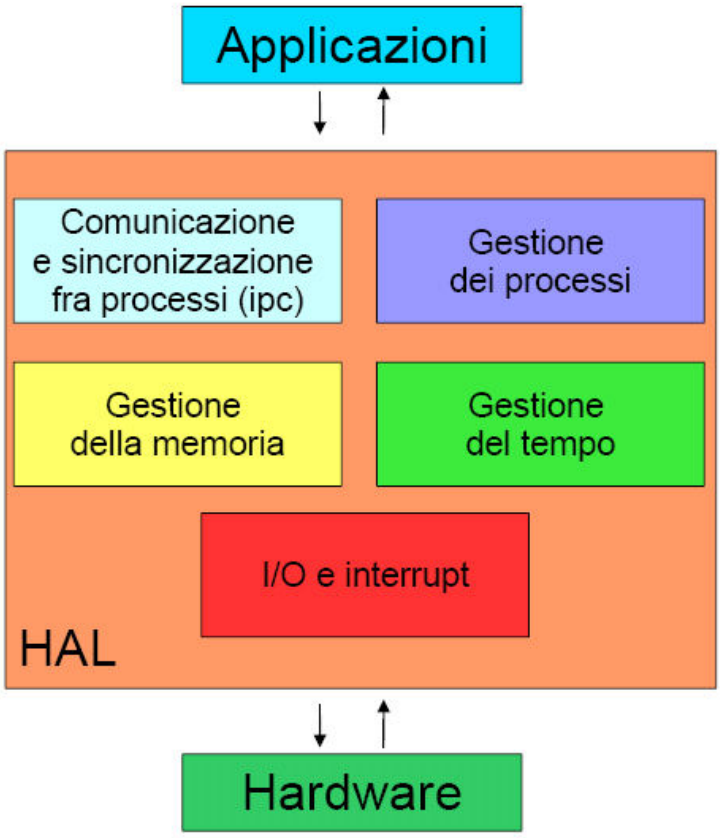
I sistemi real-time devono in genere svolgere più attività (task) che possono anche essere completamente indipendenti.
Le attività sono usualmente innescate a seguito del verificarsi di particolari condizioni (eventi) e devono terminare prima di certe deadlines.
L'intervallo di tempo tra evento e deadline viene detto "time scope" (finestra temporale) dell'attività. I time scope di diverse attività possono essere sovrapposti.
Ne consegue esecuzione parallela.
La realizzazione parallela delle attività dipende dalla architettura HW utilizzata.

- Sistemi multiprocessore (parallelismo reale)
  soluzione tipica nelle applicazioni real time
  - ravvicinati (collegamento molto veloce, es.: sulla stessa board)
  - distribuiti (collegamento più lento, es.: CAN bus, I2C, rete ethernet ecc...)

- Sistemi monoprocessore (parallelismo logico) condivisione della CPU
  problemi di gestione delle priorità
  algoritmi di scheduling dinamico

RTOS: Architettura

I sistemi operativi real-time fornisco alle applicazioni un'astrazione dell'hardware (Hardware Abstraction Layer, HAL). L'Hardware Abstraction Layer mette a disposizione delle applicazioni una serie di servizi (primitive real time).



Campi di utilizzo dei RTOS

Campi di applicazione dei sistemi operativi real-time:

- Macchine automatiche
- Automobili
- Computers
- Sistemi di telecomunicazione
- Centrali Elettriche
- Avionica

ecc...

I sistemi operativi real-time sono “nascosti”:

- Non sono studiati per interagire con l'utente (uomo);
- La programmazione di questi ambienti avviene solitamente con l'ausilio di altri sistemi operativi non real-time;
- La loro principale applicazione è in sistemi embedded.

La maggior parte dei sistemi operativi real-time consiste in soluzioni proprietarie studiate per particolari piattaforme hardware (progetto hardware/software).

RTOS: Soluzioni

Alcuni dei sistemi operativi real time “generici”:

Soluzioni proprietarie:
VxWorks
QNX
RTLinuxPro
...

Soluzioni opensource:
RTLinuxFree (solo per kernel 2.4)
RTAI
...

RTLinux e RTAI sono soluzioni basate sul kernel di Linux Possono sfruttare tutte le applicazioni e l'ambiente del sistema operativo di partenza.

RTAI Linux

Introduzione a RTAI Linux

L'ambiente real-time è realizzato tramite moduli kernel:
- ogni modulo gestisce un particolare servizio;
- disponibilità di diversi scheduler;
- possibilità di attivare e disattivare il supporto alle applicazioni real time;
- modifica trasparente alle applicazioni non real time.

Come Aggiunge al kernel di Linux le primitive real-time

Linux (kernel+applicazioni) diventa un processo di RTAI eseguito quando la CPU non è occupata nell'esecuzioni di processi real time.
Attenzione se i processi real time occupano la CPU al 100%, le applicazioni non real time risulteranno bloccate...

Caratteristiche di RTAI

Infrastruttura hard real-time sia in kernel space che in user space:

- programmazione in user space più semplice;
- implementazione in kernel space più efficiente ma potenzialmente più pericolosa per il sistema.

Funzionalità real-time avanzate:
- supporto al C++ (anche in kernel space per kernel 2.4);
- supporto al calcolo in virgola mobile in kernel space
- possibilità di backtrace delle applicazioni real time;
- disponibilità di una interfaccia grafica;
- ...

Installazione di RTAI

Si applica la patch di RTAI ai sorgenti del kernel di Linux (sono disponibili patch per i kernel più recenti) Si ricompila ed installa il nuovo kernel.
Si configura RTAI

rtaiuser@rtaibox:/usr/src/rtai$ make menuconfig
si seleziona quali servizi abilitare
rtaiuser@rtaibox:/usr/src/rtai$ make
rtaiuser@rtaibox:/usr/src/rtai$ make install

Sviluppo di applicazioni real-time con RTAI

Si possono sfruttare tutti gli editor e gli ambienti di sviluppo normalmente disponibili per Linux.
Come compilatore viene utilizzato solitamente il gcc (GNU Compiler Collection)
Disponibilità di ambienti di sviluppo dedicati per CACSD (Computer Aided Control Systems Design)

- piattaforma proprietaria: ambiente MatLab / Simulink / Real Time Workshop
- piattaforma opensource: ambiente Scilab/Scicos

RTAI-Knoppix

Si tratta di un LiveCD che mette a disposizione dell'utente un ambiente hard real time completo senza la necessità di alcuna installazione o configurazione.

- basata sulla distribuzione Knoppix-3.9
- Linux kernel versione 2.6.9
- RTAI versione 3.2
- Boot del sistema:

Guida RTAI-Knoppix

Introduzione

Questa guida mostra i passi necessari per l'installazione e l'utilizzo di applicazioni con sorgente pubblico CACSD (Computer Aided Control System Design) e di controllo in tempo reale chiamate RTAI-Lab.
Oltre ad applicazioni per lo sviluppo di sistemi di controllo tempo reale, RTAI-Lab integra simulatori e controllori real-time generati da altri programmi ad esempio Scilab & Scicos (sorgente pubblico).
Inoltre, RTAI-Lab:

• Sviluppa e esegue programmi real-time su strade locali, remote e distribuite,

• Monitora e controlla esecuzioni locali, remote e distribuite,

• Cambia parametri del controllore durante l'esecuzione.

Insieme di applicazioni RTAI-Lab

RTAI-Lab insieme di applicazioni basate su:

• **Scilab/Scicos**. Scilab è un sorgente pubblica CACSD per computazioni numeriche. Scilab include Scicos, un programma per realizzare un diagramma a blocchi che può essere usato per creare simulazioni e automaticamente generare e compilare codice.
Vedere www.scilab.org e www.scicos.org.

• **Comedi.** Comedi fornisce librerie per interagire con periferiche di acquisizione. Centinaia di periferiche sono supportate.
Vedere www.comedi.org

• **RTAI**. Interfaccia Applicazione in Tempo Reale è distribuita come un pacchetto di aggiornamento da applicare al sistema operativo Linux, aggiungendo un sotto nucleo dove i processi in tempo reale hanno la priorità. La tecnica FIFO e la menoria condivisa possono essere usate per trasferire dati tra processi utente e processi in tempo reale.
Vedere www.rtai.org.

• **RTAI-Lib**. RTAI-Lib è un insieme di componenti grafici di blocchi Scicos che permette l'utente di sviluppare diagrammi con sensori e attuatori. Esso fornisce un interfaccia per RTAI e per dispositivi che acquisiscono segnali. I diagrammi a blocchi che usano RTAI-Lib possono essere compilati in RTAI. Esso è incluso nel pacchetto RTAI.

• **xrtailab**. xrtailab è un programma simile ad un oscilloscopio che può essere connesso all'eseguibile in tempo reale. Esso permette di visualizzare e monitorare segnali e eventi in tempo reale. Xrtailab permette anche di aggiustare parametri dell'eseguibile in tempo reale mentre esso è in esecuzione.

Sviluppare con RTAI-Lab

I programmi in tempo reale sono sviluppati con applicazioni RTAI-Lab realizzati con diagrammi a blocchi Scilab/Scicos.
La verifica dell'eseguibile in tepo reale può essere fatta con xrtailab. Nuovi blocchi Scicos possono essere programmati usando il linguaggio Scilab ed eseguono componenti programmati in C.

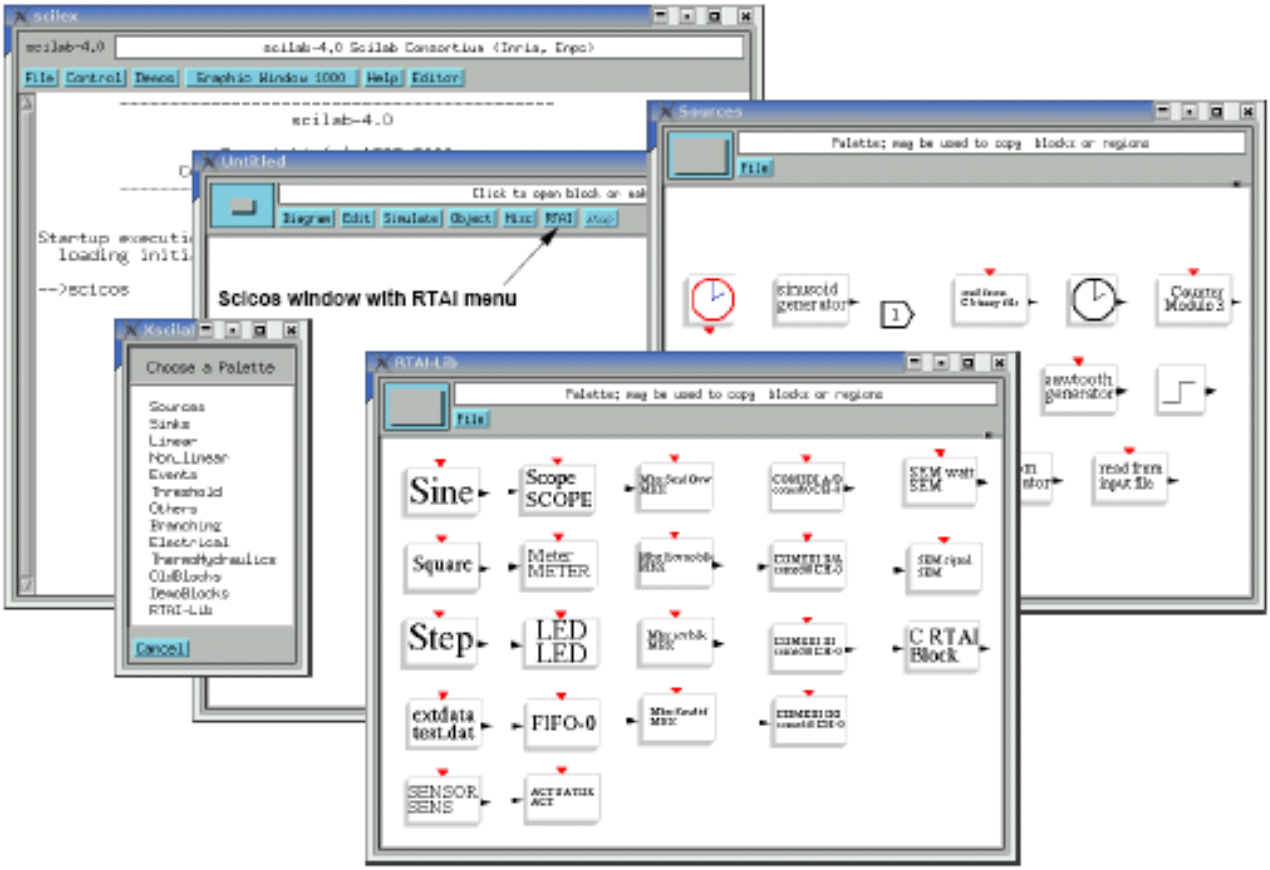


Figura 1: Scilab, Scicos

Avvio Linux-RTAI

All'avvio di Linux-RTAI sono caricati i moduli associati.

Start Scicos

Per lanciare Scilab scrivere sulla finestra dei comandi: scilab.
Nella finestra dello Scilab scrivere scicos.
Ciò apre la finestra di Scicos in cui traccerete gli schemi a blocchi. Per una lezione privata sul controllo http://www.scicos.org/TUTORIAL/tutorial.html

RTAI-Lab modifica 2 funzioni di Scicos

• C'è un menu RTAI sulla destra della finestra Scicos
• Ci sono delle gamme di blocchi RTAI

RTAI-Lib palette

Il RTAI-Lab fornisce la gamma di blocchi RTAI-Lib di Scicos. Per visualizzare la gamma di blocchi da Scicos cliccare sopra il menu Edit ->Pallettes, si apre una nuova finestra, poi cliccare sopra RTAI-Lib che dovrebbe essere elencato nella parte inferiore della finestra RTAI-Lab.

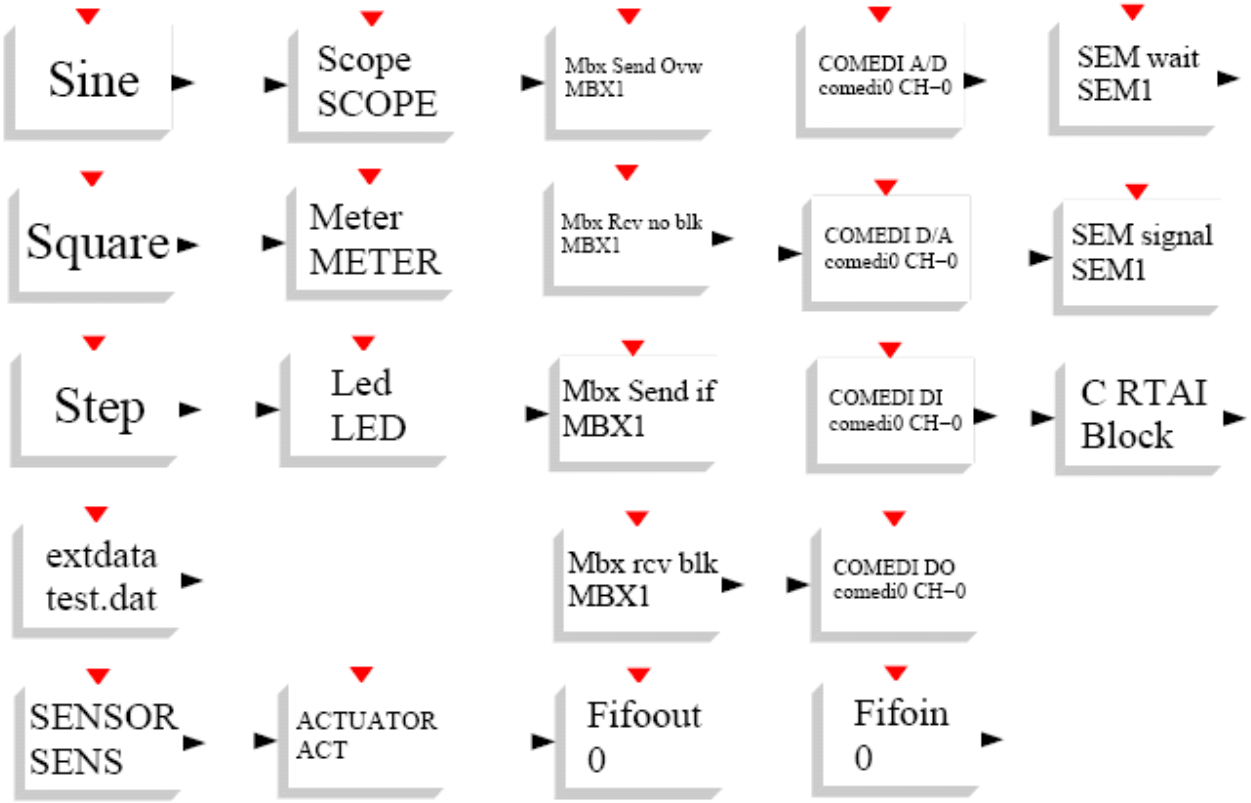


Figura 2: La gamma di blocchi RTAI-Lib che caratterizza (da sinistra a destra) gli ingressi, le uscite, le mailboxes, le interfacce dei dispositivi ed i semafori.

I blocchi (più o meno) sono organizzati in colonne tematiche:

1. ***Ingressi:*** i generatori di funzioni (seno, quadrato, gradino), legge file di dati, sensore generico programmabile.
2. **Uscite:** Visualizza (oscilloscopio, LED), attuatori generici programmabili
3. **Mailboxes:** trasmettere il messaggio e lo sovrascrive, ricevere senza ostruire l'operazione, ricevere senza condizioni, trasmettere il messaggio se l'operazione di sincronizzazione lo consente.
4. **Comedi:** procedure per dispositivi: A/D, D/A, ingressi, uscite digitali.
5. **Semafori:** attesa, segnale, blocchetto generico programmabile di C Se avete installato correttamente il RTAI-Laboratorio potete trovare il codice sorgente per i blocchi dentro /usr/local/scilab-4.x.x/macros/RTAI. Comunque i blocchi originali forniti della distribuzione di RTAI sono in /usr/src/rtai/rtai-lab/scilab/macros/RTAI. La maggior parte di questi blocchi sono chiamati dalle funzioni del RTAI-Lab scritte in C e compilate nella libreria /usr/realtime/lib/libsciblk.a.
Il codice sorgente delle funzioni scritte in C è in /usr/src/rtai/rtai-lab/scilab/devices. Se si vuol guardare il codice sorgente delle funzioni RTAI-Lab, si deve vedere come le funzioni RTAI API sono chiamate. Queste funzioni RTAI API sono documentate in HTML e altri formati in /usr/src/rtai/doc/generated se si specifica che la documentazione deve essere generata durante l'installazione di RTAI.
Segue una spiegazione dettagliata dei blocchi RTAI.
I valori di default sono indicati tra parentesi quadre.

Ingressi:

• **Seno.** Generatore della forma d'onda seno in tempo reale. Il blocco seno permette di aggiustare ampiezza [1], frequenza [1] (Hz), fase [0], bias [0] e ritardo [0]. Inoltre si possono aggiustare questi parametri da xrtailab o su programma esternomentre l'eseguibile in tempo reale sta girando.

• **Onda Quadra.** Forma d'onda quadra in tempo reale. Il blocco permette di aggiustare ampiezza [1], periodo [4] (s), impulso con [2] (s), bias [0]e ritardo [0] (s). Inoltre si possono aggiustare questi parametri da xrtailab o su programma esterno mentre l'eseguibile in tempo reale sta girando.

• **Gradino .** Funzione gradino. Il blocco gradino permette di aggiustare ampiezza [1] e ritardo [0] (s).

• **Dati esterni.** Carica dati da un file. Il file [test.dat] deve contenere una singola colonna con valori ASCII.

• **SENSORI.** Generico ingresso sensore. Si può aggiustare il numero di uscite [1] e un identificatore [SENS].

Uscite:
Scope, Meter e LED sono blocchi di uscita che inviano dati a xrtailab.

• **Scope.** Oscilloscopio xrtailab multicanale. Si può modificare il numero di ingressi [1] e il nome [SCOPE].

• **Meter.** Tester xrtailab singolo canale. Si può modificare il nome [METER].

• **LED.** Blocco Multi-LED xrtailab multicanale. Si può modificare il numero di ingressi/LEDs [1] e il nome blocco LED [LED].

• **ATTUATORE.** Generico attuatore uscita. Simile al blocco sensore. Si può modificare il numero di ingressi [1] e l'identificatore [ACT].

Mailboxes:

Mailboxes permette ai processi in tempo reale di scambiare messaggi di diverse dimensioni arbitrariamente. Il messaggio che passa può essere interrotto dai vincoli in tempo reale. Vedere /usr/src/rtai/doc/generated/html/api/group mbx.html e [Sarolahti, 2001].
Il blocco del processo locale può accadere quando scambia un pacchetto fra un elaboratore locale e una mailbox su un elaboratore remoto. Inoltre, la trasmissione del pacchetto ad una mailbox su un elaboratore remoto attualmente è limitata dal formato del pacchetto del UDP (1500 byte).

• **Mbx Send Ovw.** Trasmette un messaggio ad una mailbox, possibilmente sovrascrive ciò che è già nella mailbox. Si può registrare il numero di porte del messaggio di ingresso[1], il nome della mailbox [MBX] e l'indirizzo IP della mailbox [127.0.0.1] (localhost). Vedere inoltre la documentazione di RTAI api per la funzione rt mbx ovrwr di trasmissione.

• **Mbx Rcv no blk.**
Riceve un messaggio soltanto se il messaggio intero può essere passato senza ostruire l'operazione di chiamata. Tuttavia, se la mailbox è su un elaboratore remoto, l'operazione di chiamata può essere bloccata. Si può modificare il nmero di porte dei messaggi in uscita [1], nome mailbox [MBX] e l'indirizzo IP della mailbox [127.0.0.1] (localhost). Vedere inoltre la documentazione di RTAI api per la funzione rt mbx di ricezione.

• **Mbx Rcv blk.** Riceve un messaggio. L'operazione di chiamata sarà bloccata fino a che tutti i byte del messaggio non arrivino o un errore accada. Si può cambiare il numero di porte di uscita del messaggio [1], il nome della mailbox [MBX] e l'indirizzo IP della mailbox [127.0.0.1] (localhost). Vedere inoltre la documentazione di RTAI api per la funzione rt mbx per la ricezione.

• **Mbx Send if.** Trasmette un messaggio soltanto se il messaggio intero può essere passato senza bloccare la chiamata dell'operazione. Tuttavia, se la mailbox è su un elaboratore remoto, l'operazione di chiamata può essere bloccata. Si può modificare il numero di porte in ingresso del messaggio [1], il nome della mailbox [MBX] e l'indirizzo IP della mailbox [127.0.0.1] (localhost). Vedere inoltre la documentazione di RTAI api per la funzione rt mbx per la trasmissione.

• **FIFOout e FIFOin.** Multicanale FIFO. Si puó cambiare il numero di ingressi [1], il numero di FIFO's [0], e la dimensione FIFO's [50000] (in bytes).

Comedi procedure per dispositvi:

• **Comedi A/D.** Comedi supporta l'ingresso analogico. Si può selezionare il canale d'ingresso della scheda di acquisizione [0], il dispositivo se si hanno parecchie schede di acquisizione [comedi0], l'insieme dei valori come specificato nel manuale delle schede di acquisizione [0] (avvertendo: il valore da entrare non è espresso nei volt) e nel tipo di riferimento di tensione [0] Si veda inoltre i dati letti di comedi di Comedi api.

• **Comedi D/A.** Comedi supporta l'uscita analogica. I parametri simile al Comedi A/D. Vedre anche i dati scritti Comedi API.

• **Comedi DI.** Comedi supporta ingressi digitali. Si può selezionare il canale [0] e il supporto [comedi0].
Veder anche Comedi API comedi dio lettura.

• **Comedi DO.** Comedi supporta uscite digitali. Si può selezionare il canale [0], il supporto [comedi0] e la soglia [1]. Se l'immissione dei dati al blocco è più grande del valore di soglia, allora un singolo bit sarà prodotta dal dispositivo. Vedere inoltre il dio write di comedi Comedi api.

Semafori e altro:

I Semafori possono essere utilizzati per sincronizzare i processi in tempo reale. Similmente alle mailbox, l'attenzione è necessaria riguardo al blocco che puó avvenire quando si scambia un semaforo fra un elaboratore locale e remoto.

• **SEM wait.** Decrementa il valore del semaforo e aspetto un evento del segnale. La chiamata del processo è bloccata e messa in coda finché il valore del semaforo è negativo. Si può modificare il nome del semaforo [SEM] ed l'indirizzo IP [127.0.0.1] (localhost). Vedere inoltre la documentazione di RTAI api per la funzione rt sem wait.

• **SEM signal.** Incrementa il valore del semaforo. Se il valore risultante non è positivo allora il primo processo nella coda di attesa del semaforo può andare in esecuzione. Si può registrare il nome del semaforo [SEM] ed indirizzo IP [127.0.0.1] (localhost). Vedere inoltre la documentazione di RTAI api per la funzione del segnale sem rt.

• **C RTAI.** É un blocco generico che lascia inserire il codice C. È basata su C-Block2 dello Scilab. Quando si clicca sopra il blocchetto di C RTAI una finestra di dialogo si apre dove si può editare i parametri del blocco quale il nome della funzione principale che conterrà, il numero di ingressi e di uscite per i dati, gli eventi e vari parametri. Dopo aver ciccato su OK una seconda finestra si apre con uno scheletro di codice per editare. Copiare il codice in questa finestra. Quando si clicca OK il codice è compilato e collegato. Se il codice è relativamente grande e meglio programmare il proprio blocco invece di usare il blocchetto di C RTAI.

Sinusoide in tempo reale: passo per passo
In questo esempio si usa Scilab/Scicos per tracciare uno schema a blocchi che genera un sinusoide. Automaticamente si genera e compila un eseguibile in tempo reale. Si testa il programma e si visualizza la sinusoide con xrtailab.

Creazione diagramma a blocchi
Creazione diagramma della sinusoide (vedere figura).

Seleziona blocchi. Avvia Scicos. Aprire sorgente palette: ciccare su menu Edit -> Palettes e seleziona sorgente. Aprire RTAI-Lib palette nello stesso modo.

Scorciatoie Tastiera. Si noti che per default, Scicos ha alcune scorciatoie,sono elencate con il menu Misc -> Shortcuts.

Seleziona Blocchi. Dai blocchi sogente, clicca col tasto sinistro sull'orologio rosso. Spostare il mouse verso la finestra principale Scicos. Si Dovrebbe vedere il profilo del blocco che è trascinato mentre il vostro cursore del mouse entra nella finestra principale di Scicos. Ricclicare col tasto sinistro per disporre l'orologio nella finestra principale di Scicos, in qualche luogo verso il centro superiore della finestra. Se si clicca col tasto destro si annulla la disposizione del blocco. Dai blocchi sorgente prendere il blocco del seno e disporlo verso sinistra nella finestra principale di Scicos. Porre anche un blocchetto Scope verso la destra più bassa.

Allineamento blocchi . In Scicos I blocchi non sono perfettamente allineati.Due metodi possono aiutare nell'allineamento dei blocchi.

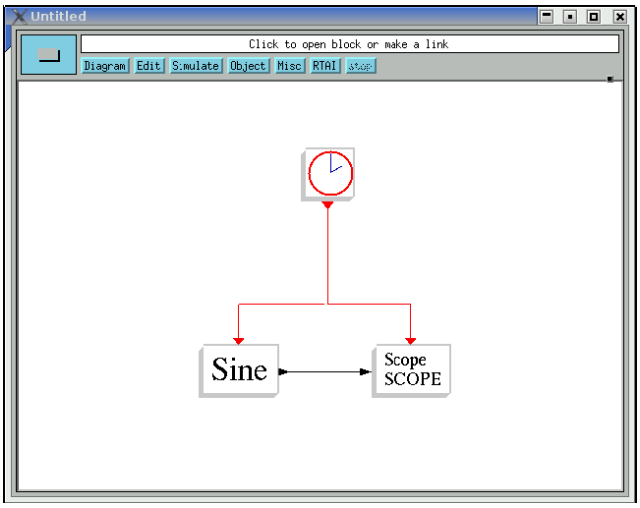


Figura: Diagramma a blocchi Di una sinusoide in tempo reale.

• Usare la funzione align di Scicos: menu Edit -> Align. Cliccare sulla porta di uscita del blocco Sine e poi sull'ingresso del blocco Scope.
Tramite la tastiera basta posizionare il mouse sull'ingresso o l'uscita del blocco, premere il tasto "a" e cliccare sulla porta del blocco da allineare.

• Modificare l'accelerazione del mouse sotto windows.

Collegamento Blocchi. Menu Edit -> Link. Cliccare sulla porta di uscita del blocco Sine.Cliccare sulla porta di ingresso del blocco Scope ed un link è disegnato tra il blocco Sine e il blocco Scope. Cliccare sulla porta di uscita dell'evento orologio (il triangolo rosso sotto l'orologio). Poi disegnare il collegamento al blocchetto Sine. Un collegamento rosso è disegnato dall'orologio al seno. Disegnare un altro collegamento di evento cliccando sulla linea rossa, quindi cliccare sopra la porta di ingresso del blocco Scope.

Settare parametri blocco. Cliccare sopra l'orologio. Si possono aggiustare i tempi di inizio e il periodo (valori sono in secondi). Cliccare sopra il seno, modificare i parametri di conseguenza, infine ciccare su OK. Si può anche editare il nome Scope e dire che può realmente avere più di un'immissione dei dati. Comunque, è necessario che Scicos valuti tutti i blocchi. Ciò è fatto quando cliccate sopra un blocco, modifica i relativi parametri ed infine si clicca su OK. Una via generale per valutare tutti i blocchi è selezionare il menu Simulate -> Eval. Si deve fare questa operazione Eval ogni volta che si aggiunge un blocco ad uno schema e si omettete il controllo dei relativi parametri interni.

Creazione Super Blocco. Menu -> Diagram -> Region to Super Block. Si deve disegnare una struttura elastica intorno ai blocchetti Scope e Sino, escludendo l'orologio. Il primo click sopra a sinistra del blocchetto Sine, poi un click a destra del blocchetto Scope. La regione allora si trasforma in un Super Blocco.

Compilazione
Settaggio target. Si può facoltativamente cliccare sopra il menu RTAI -> Set Target ed infine cliccare sopra il Super Blocco per compilare. Si apre una finestra di dialogo dove si può modificare:
• **Target.** Questo é il nome del Makefile. Vedere la descrizione riportata di seguito.
• **Funzioni Ode.** É una delle funzioni ordinarie di equazione differenziale disponibili in Scilab. Il valore di default è ode4. I valori possibili sono:
- ode1. Usa il metodo di Eulero (RK1)
- ode2. Usa il metodo di Heun (RK2), anche conosciuto come metodo di eulero migliorato.

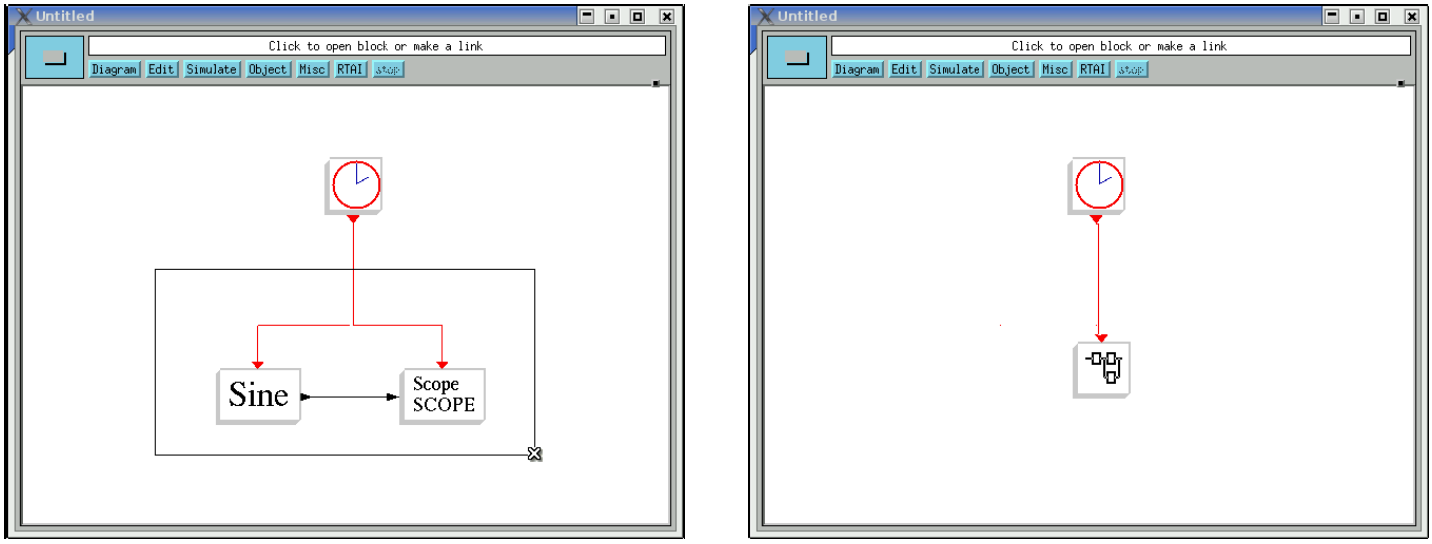


Figure: Sinistra: schema a blocchi preliminare di RTAI-Lib durante la creazione del Super Blocco. Tutti i blocchi tranne l'orologio dovrebbero essere contenuti nella regione. Destra: Super Blocco risultante.

– **ode4.** Usa la formula di Runge-Kutta del quarto ordine (RK4).

Il codice sorgente per queste funzioni è disponibile in $SCILAB/macros/RTAI/RTAICodeGen.sci. Le funzioni ODE dello Scilab sono dettagliate nella sezione 3.2 di [Campbell ed altri., 2006] ed anche in [Sallet, 2004].

• **Il passo tra due campioni.** Indica il numero di punti dei sottocampioni usati per varie funzioni come le funzioni ODE. Il valore di default é 10.

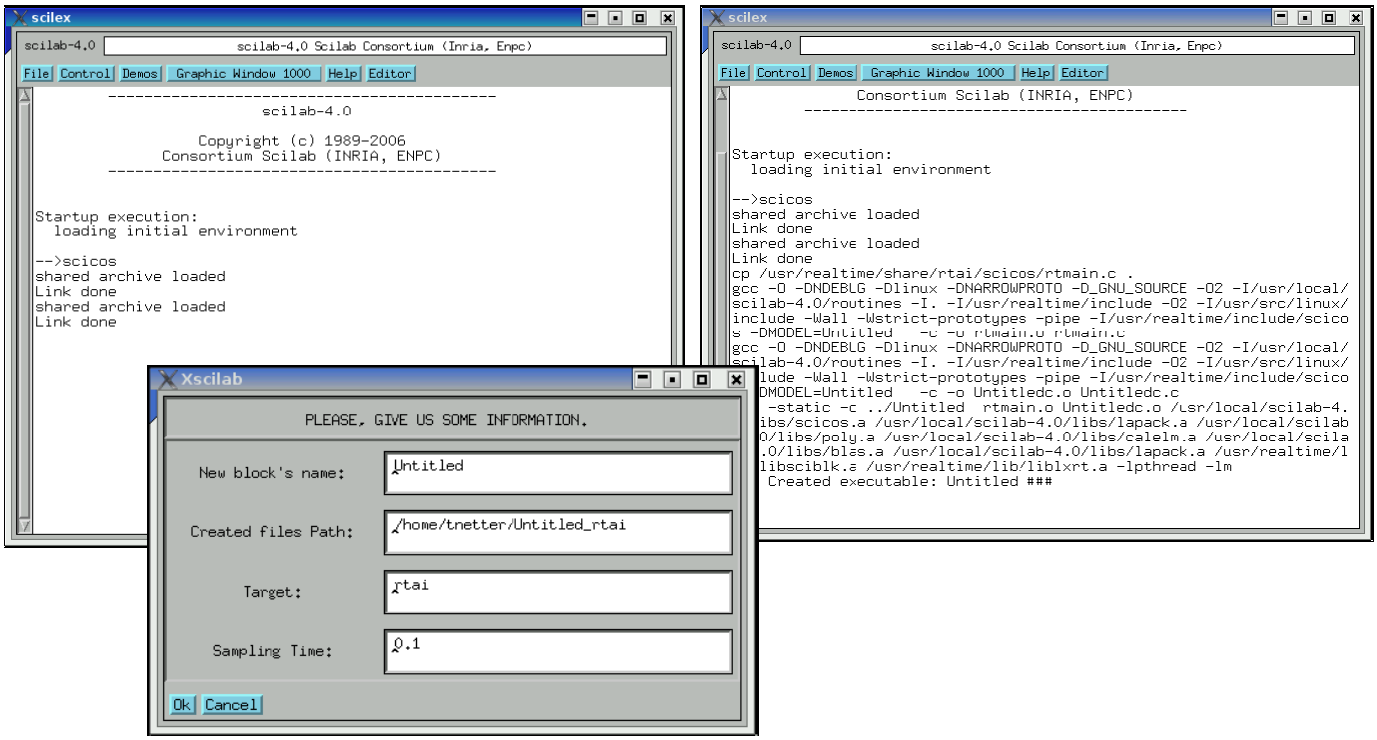


Figura: Schema a blocchi: generazione codice e compilazione. (a): Finestra di Scicos dopo la generazione del codice. (b): La finestra di dialogo per modificare il nome del file, il makefile ed il periodo dell'orologio. (c): Il testo prodotto dopo la compilazione.

Compilazione schema a blocchi. Menu RTAI -> RTAI CodeGen.
Poi ciccare sul Super Blocco. A questo punto Scilab converte il vostro schema a blocchi in codice C. Per ogni blocco all'interno del Super Blocco sono prodotte due le linee nella finestra di Scicos (5a): shared archive loaded Link done
Una finestra di dialogo si apre (5b) (mostra il dialogo con i valori di default) dove potete modificare:
• **Nome nuovo blocco.** Questo sarà il nome dell'eseguibile finale. Esso verrà salvato nella directory corrente dello Scilab. Per default é Untitled.
• **Creazione Cammino file.** Questa é la directory dove i file C generate sono salvati insieme al Makefile.
• **Target.** Questo é il nome del Makefile. Esso sarà copiato nella directory dei file generate e usato per la compilazione. Il valore di default è rtai e corrisponde al file $SCILAB/macros/RTAI/RT templates/rtai.mak
• **Tempo di campionamento.** Corrisponde all'insieme di valori del periodo nei parametri del blocchetto dell'orologio, cioè si può modificare l'orologio qui.

Ciccare su OK e la compilazione inizia. I passi nella compilazione possono essere controllati nella finestra di Scilab (5c). Nel caso la compilazione fallisce, si puó far partire manualmente la compilazione scrivendo nella directory in cui i file C sono stati generati. Ciò fornisce più uscite che poi vengono stampate nella finestra di Scilab. Se si mantengono i valori di default si dovrebbe trovare unafile eseguibile denominato Untitled nel directory corrente.

Esecuzione
Aprire due terminali.

Esecuzione in tempo reale. Nel primo terminale scrivere: Untitled –u Questo fornisce istruzioni su come usare le varie opzioni. Per far partire l'esecuzione in tempo reale in verbose mode: Untitled –v.

Xrtailab. xrtailab é eseguito solo quando RTAI Linux kernel é in esecuzione. xrtailab genererà un segmentation fault se viene avviato su un kernel Linux. Nel secondo terminale scrivere:
xrtailab -h
xrtailab
Select menu File -> Connect. . . (oppure da tastiera: alt-c) poi cliccare su OK(Fig. 6).

L'indirizzo IP è quello dell'host in cui l'operazione in tempo reale è in esecuzione. Gli identificatori IFTASK, RTS, RTL, RTE, RTM e RTY sono già definiti dentro rtmain.c, nella directory che contiene il codice sorgente generato. Se parecchi processi in tempo reale devono funzionare in parallelo, gli identificatori del processo in tempo reale possono essere regolati via comandi in linea (eseguire l'operazione con opzione -u per elencare la lista delle opzioni del comando) e modificando i valori "Connect to Target „. Si può persino editare rtmain.c per cambiare gli identificatori predefiniti del processo in tempo reale.

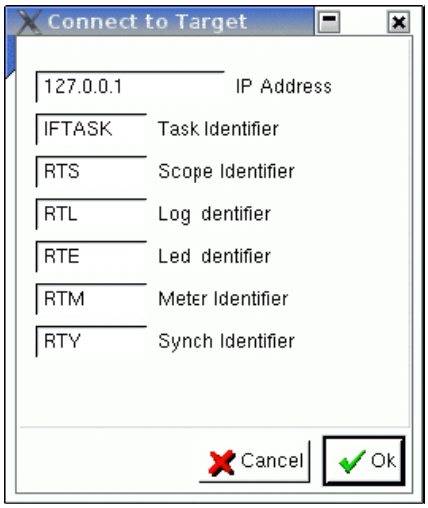


Figura6: finestra di dialogo per la connessione xrtailab.

I tasti di xrtailab (Figura 7) permettono di far partire ed arrestare l'esecuzione in tempo reale ed aprire in tempo reale vari display come Scope, tester e LED. Per aprire una finestra di gestione o ciccare sul tasto Scope o selezionarlo nel Menu (scorciatoia: alt-s). Poi cliccare sopra il checkbox per visualizzare lo Scope. Dovreste vedere una sinusoide da 1 hertz procedere serpeggiando uniformemente nella finestra

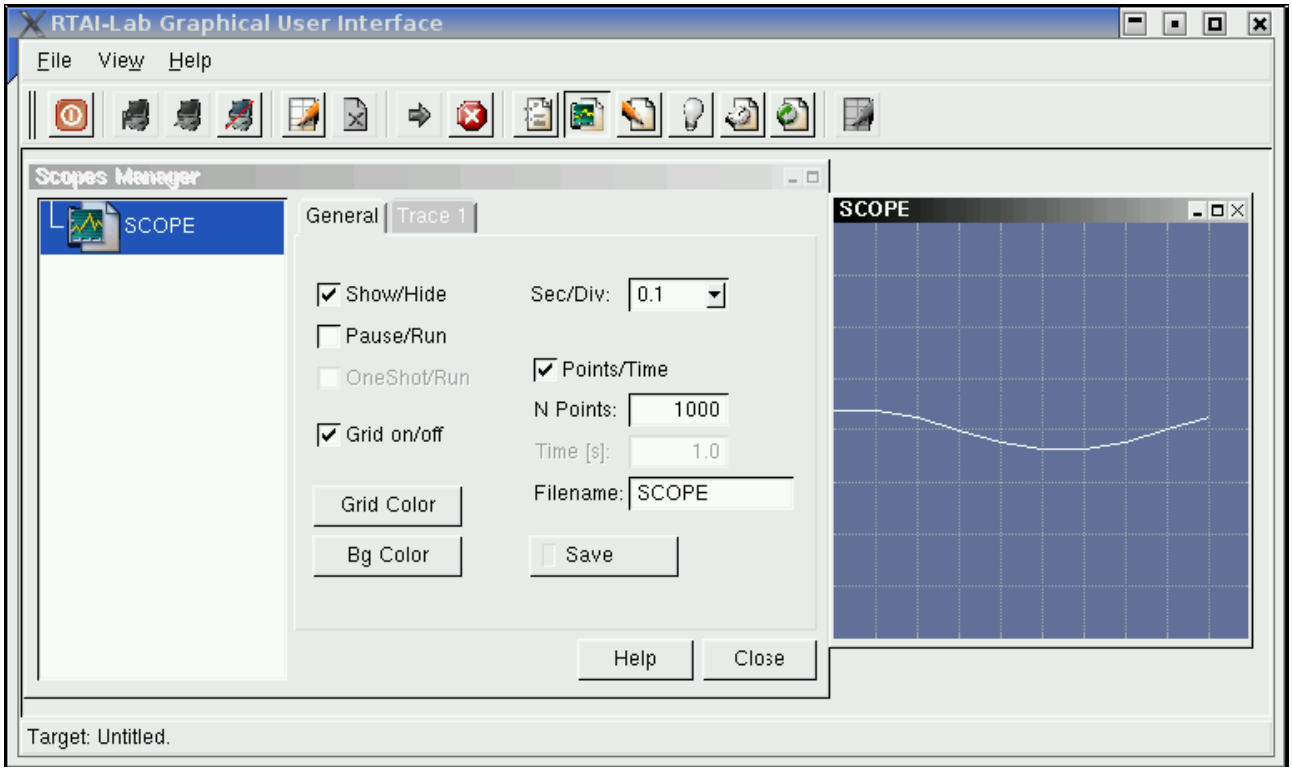


Figure 7: xrtailab con sinusoide a 1 HZ.

Modifica parametri

Xrtailab permette di cambiare "al volo" i parametri che sono stati definiti nello schema a blocchi di Scicos. Per esempio, il blocchetto Sine ha 5 parametri: l'ampiezza (valore [0]), la frequenza (valore [1]), la fase (valore [2]), bias (valore [3]) e il ritardo (valore [4]). Cliccare sopra il tasto di gestione dei parametri. La finestra del gestore dei parametri si apre. Si può ridimensionare la finestra di xrtailab e riposizionare la finestra del gestore dei parametri per ottenere una disposizione conveniente simile a quella come appare figura 8. Cambiare il valore [0] a 3 e premere INVIO. L'ampiezza della sinusoide aumenterà di conseguenza. Cambiare il valore [1] a 2. Ciò raddoppierà la frequenza della sinusoide a 2 hertz e noterete che l'onda ha perso la relativa scorrevolezza. Ciò è un effetto sottocampionamento indotto dal fatto che il periodo dell'orologio, come definito nello schema a blocchi di Scicos, è soltanto di 0.1 s o di 10 hertz. Si noti che il gestore dei parametri inoltre lascia trasferire i parametri dai processi tempo reale in esecuzione su elaboratori remoti ed riprenderli dopo la modifica.

Stop esecuzione

Cliccare sopra l'icona esagonale di arresto per disconnettere xrtailab e per terminarlo. Si noti che è possibile disconnettere soltanto xrtailab da un target (alt-d) e lasciare che esso continui a funzionare. Si può ricollegare più successivamente ad esso. Per concludere, si può anche interrompere l'eseguibile in tempo reale direttamente dal terminale in cui si é lanciata la relativa esecuzione: premere ctrl-c.

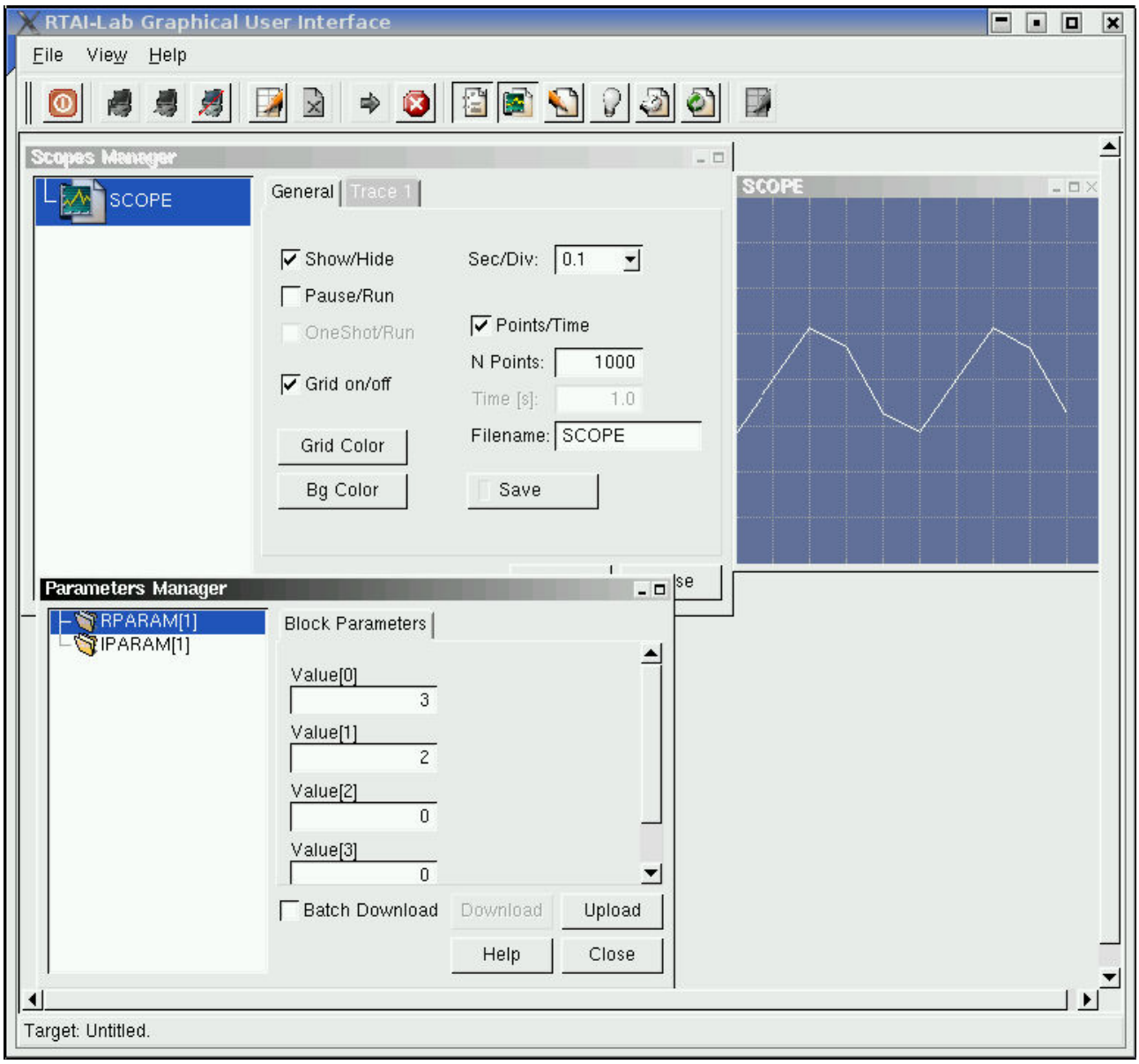


Figura 8: xrtailab con la finestra del gestore dei parametri. I parametri che sono stati cambiati sono ampiezza della sinusoide (valore [0]) da 1 a 3 e frequenza (valore [1]) da 1 a 2 hertz. La finestra rivela un sottocampionamento poiché l'orologio eseguibile in tempo reale è regolato a soltanto 10 hertz.

Simulazione e modellizzazione in tempo reale

Modificare ancora lo schema a blocchi usando i blocchi di RTAI-Lib:
• sostituire l'orologio ed il generatore di dell'onda quadra con un blocchetto onda quadra di RTAI-Lib. Parametri del blocchetto onda quadra: L'ampiezza = 1, periodo = 6, larghezza di impulso = 3, bias = 0, ritardo = 0
• sostituire il multiplexer e lo scope con un blocchetto Scope di RTAI-Lib. Regolare le porte dell'input = 3
• Menu Diagram!Region to Super Block: Selezionare tutti i blocchi tranne l'orologio. Aprire Super Blocco del • 14b. Menu Diagram -> Rename Cambiare il nome dello schema a rtex3. Chiudere la finestra del super blocco.
• Menu RTAI -> RTAI CodeGen e compilare
• In un terminale scrivere: ./rtex3 - v
• In un altro terminale scrivere: xrtailab
• Nel xrtailab: collegarsi al target (alt-c), aprire il gestore per visualizzare la forma d'onda. Nel gestore di Scope si può cliccare sopra le linguette per modificare il colore di ogni traccia (Figura 15).

Programmare funzioni C esterne

Notare in rtai proc.sci (pagina 37) che lo scheletro di codice in C nella funzione del getCode è suddiviso tramite le selezioni case. Nella nostra funzione di esempio, ogni case chiama userfunc init funzioni C esterne precompilate, l'uscita di userfunc e l'aggiornamento di userfunc. Vedere inoltre DirOutBit.c e DirInpBit.c alla pagina 34 per programmare i vari case in file .c separati piuttosto che nella funzione del getCode (ora obsoleta). Queste funzioni di C possono essere codificate normalmente. Devono tuttavia essere compilate ed archiviate in una libreria. Questa libreria sarà denominata da Scilab/Scicos quando converte il vostro schema a blocchi in codice C e compila e collega le funzioni risultanti.

Compilare funzioni C e generare una libreria

Il Makefile seguente è un esempio per compilare le funzioni C contenute in userfunc.c (per esempio userfunc init, userfunc output ed userfunc update) nella libreria libuser.a.

```
SRC = userfunc.c
LIB = libuser.a
OBJ = $(SRC:.c=.o)
RTAIDIR = $(shell rtai-config --prefix)
all: $(LIB)
CC_FLAGS = -c $(DBG) -I. -O2
%.o: %.c
cc $(CC_FLAGS) $<
$(LIB): $(OBJ)
ar -r $(LIB) $(OBJ)
cp $(LIB) ../lib
clean:
rm -f $(LIB) $(OBJ)
```