Modellazione di un protocollo di *bus arbitration* con SMCube

Federico D'Amato, Alessio Sarullo

1 Introduzione

L'obiettivo dell'elaborato è quello di modellare e simulare il problema di *bus arbitration*, utilizzando il protocollo Linear Two-Phase Commit. Lo scenario è quello in cui n processori fanno periodiche richieste di accesso ad un bus qualunque tra un insieme di m disponibili.

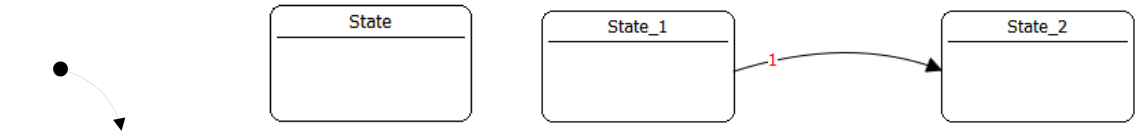
Ogni processore ed ogni bus viene modellato tramite una macchina a stati finiti; il tool utilizzato per la realizzazione di tali macchine è SMCube, della suite Scicos. L'intero sistema è rappresentato da un insieme di macchine a stati finiti realizzati con SMCube e da un certo numero di blocchi funzionali propri dell'ambiente Scicos, interconnessi fra loro.

Nella prima parte della relazione viene descritto il tool SMCube, sia dal punto di vista dell'interfaccia utente, sia da quello delle funzionalità offerte. Successivamente, il problema di *bus arbitration* viene descritto nel dettaglio, così come gli obiettivi dell'elaborato ed i vincoli realizzativi. Nella sezione successiva vengono mostrate le implementazioni delle macchine a stati (processori e bus) e del sistema di interconnessione risultante. Infine vengono illustrate le simulazioni effettuate, cercando di evidenziare eventuali punti di forza o debolezze dell'approccio realizzativo utilizzato.

2 SMCube

SMCube [3] è un tool per la modellazione, simulazione e generazione di codice per automi a stati finiti (*ASF*) a tempo discreto. È stato progettato per essere integrato all'interno del framework *Scicos* [2], permettendo la creazione di data-flow diagram che contengono ASF. In questo modello ibrido, i componenti del data-flow diagram vengono usati per gestire il flusso dei dati, mentre gli automi a stati finiti vengono usati per aggiungere comportamenti più complessi, non facilmente implementabili altrimenti.

Nelle sezioni seguenti verrà data un'idea generale dell'applicazione. Per maggiori dettagli si rimanda a [1].



(a) Stato iniziale. **(b)** Stato generico. **(c)** Transizione di priorità 1 dallo stato `State_1` allo stato `State_2`.

Figura 1: Componenti fondamentali di un ASF in SMCube.

2.1 Modellazione di un ASF tramite SMCube

In SMCube la modellazione di un ASF avviene per via grafica. Le figure 1a e 1b mostrano, rispettivamente, lo stato iniziale e uno stato generico dell'automa, mentre in figura 1c viene mostrata una transizione tra due stati.

Il tempo all'interno di SMCube è discreto ed è scandito dal verificarsi di *eventi*. L'integrazione con Scicos consente di definirne uno solo, che nel nostro caso è un clock comune a tutti i nodi. Al verificarsi dell'evento viene effettuata la transizione eseguibile con la priorità più alta, o si permane nello stato attuale se non ci sono transizioni eseguibili.

Nella modellazione di una macchina a stati con SMCube esiste la possibilità di definire delle *azioni*. L'esecuzione delle azioni può avvenire con diverse modalità:

- durante una transizione
- entrando in uno stato
- permanendo in uno stato
- uscendo dallo stato.

Le azioni consentono di inviare segnali in output o modificare i valori delle variabili locali; questi valori, insieme agli input ricevuti, possono poi essere utilizzati all'interno delle *guardie*, che permettono di controllare l'esecuzione delle transizioni. In figure 2 è mostrato un esempio di azioni e guardie che fanno utilizzo sia delle variabili locali che dei segnali inviati e ricevuti.

2.2 Integrazione con Scicos

Scicos è un modellatore grafico, simulatore e generatore di codice per sistemi dinamici. Un diagramma Scicos è costituito da una serie di blocchi comunicanti tra loro attraverso due tipi di porte: dati ed eventi. I dati costituiscono le informazioni di interesse che vengono manipolate dai vari blocchi, mentre gli eventi sono usati per comunicare ad un blocco quando eseguire determinate azioni. In figura 3 è mostrato un esempio di diagramma Scicos.

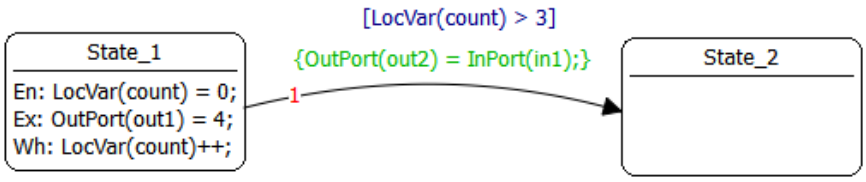


Figura 2: Esempio di azioni e guardie. Le label *en*, *ex* e *wh* indicano rispettivamente le azioni compiute entrando, uscendo o permanendo nello stato. Sulla transizione sono presenti una guardia (in blu, tra parentesi quadre) e un'azione (in verde, tra graffe). Le keyword `LocVar`, `OutPort` e `InPort` consentono di far riferimento rispettivamente a variabili locali, porte di output e porte di input.

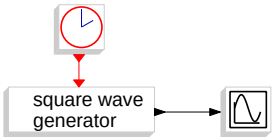


Figura 3: Un diagramma Scicos che stampa a video un'onda quadra. In rosso sono rappresentati i collegamenti e le porte relativi agli eventi, in nero quelli relativi ai dati. Ad ogni colpo di clock il generatore esegue un'azione, cambiando il valore in output. Il risultato è un'onda quadra con periodo pari a 2 colpi di clock.

L'integrazione di ASF progettati con SMCube all'interno di diagrammi Scicos è molto semplice: è sufficiente inserire un blocco SMCube all'interno del diagramma e specificare il file in cui è stato definito l'automa a cui il blocco si riferisce, come mostrato in figura 4. Una volta integrato il blocco, è possibile eseguire la simulazione in due modalità: batch e interattiva. La modalità batch è efficiente, ma non produce output ed è quindi più indicata per eseguire la simulazione dell'intero modello, assumendo che l'ASF rappresentato dal blocco sia corretto. La modalità interattiva, invece, produce un output grafico e consente anche un'esecuzione step-by-step, al costo ovviamente della velocità di esecuzione; è pertanto più indicata per eseguire il debug dell'ASF o anche dell'intero sistema, perché consente di controllare i dati che arrivano all'automa, il sui stato interno o gli output che esso produce.

3 Descrizione del problema

Il problema in esame è quello di *bus arbitration*, problema tipico in ambienti multi-processore. Il caso è quello in cui n processori $p_1, \ldots, p_n$ fanno periodiche richieste di accesso ad uno qualunque degli m bus fra $b_1, \ldots, b_m$. In un qualsiasi istante di tempo, un bus è assegnato al più ad un processore ed un processore sta utilizzando al più un unico bus.

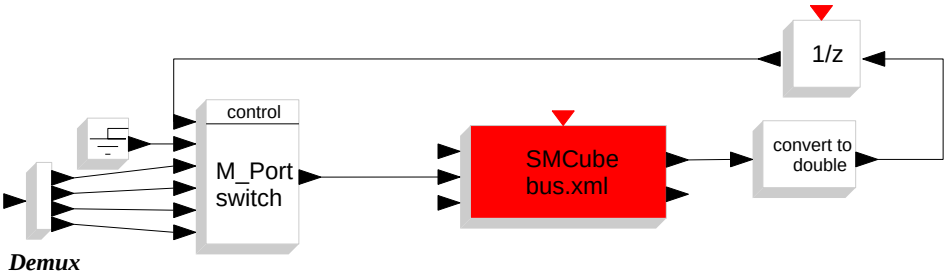


Figura 4: Blocco SMCube rappresentante un bus (in rosso) all'interno di un diagramma Scicos.

In Figura 5 sono mostrate delle macchine a stati finiti che modellano, in prima approssimazione, il comportamento di processori e bus. Un processore invia un segnale di `request` per richiedere l'accesso ad un bus. Se la richiesta ha buon fine, usa il bus fino a quando decide di rilasciarlo (segnale `release`). Il controllore del bus ha un comportamento speculare: a seguito di una richiesta manda un segnale di `grant` al processore, aspettando un segnale di avvenuta ricezione (`ack`), e poi attende che il processore lo rilasci.

Rispetto alle macchine a stati di Figura 5, si rende necessaria la realizzazione di un meccanismo che garantisca l'unicità della segnalazione di `grant` per una coppia processore-bus. Inoltre deve essere possibile, in un dato istante, riconoscere l'impossibilità temporanea di servire un processore e poter segnalare questa situazione al processore richiedente con un segnale di `nack`.

Nel caso di più richieste di utilizzo di uno stesso bus da parte di processori distinti, si pone anche il problema di decidere a quale processore assegnare il bus. Vengono prese in esame due diverse politiche di gestione:

- Equal priority: tutti i processori hanno la stessa priorità ed eventuali richieste in conflitto vengono servite in ordine FIFO di arrivo
- Fixed priority: i processori hanno una diversa priorità e in caso di richieste "simultanee" (nello stesso ciclo di clock) si serve quella del processore prioritario

Per la generazione dei segnali di `grant` e `nack`, i nodi bus eseguono il protocollo *Two-Phase Commit* (o *2PC*) in versione lineare. In questa versione del protocollo, i nodi bus sono collegati a catena, come in Figura 6. Ogni partecipante ha un numero progressivo e conosce il precedente e il successivo; l'ultimo nodo della catena sa di essere l'ultimo. Il protocollo procede attraverso due fasi successive:

- Fase 1: ogni nodo, partendo dal primo fino all'ultimo, comunica di essere arrivato al punto di *commit* (o *abort* nel caso qualcosa non abbia funzionato)

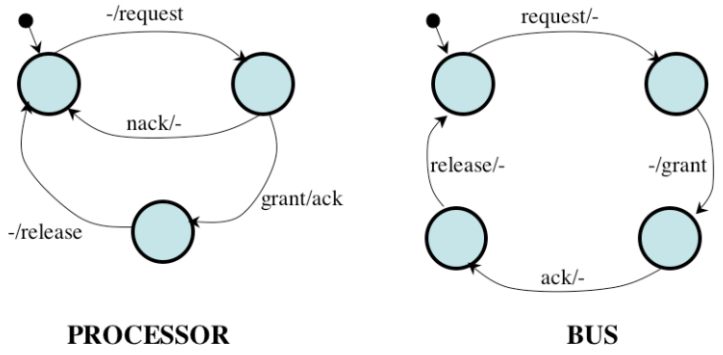


Figura 5: Schemi funzionali di processori e bus

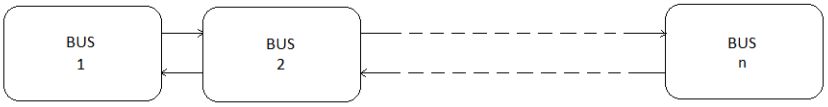


Figura 6: Collegamento a catena dei bus

- Fase 2: l'ultimo nodo risponde *OK* e il messaggio percorre in senso inverso la catena

Una realizzazione alternativa del protocollo prevede un collegamento aggiuntivo tra l'ultimo elemento della catena ed il primo; in questo caso la topologia diventa ad *anello* e il protocollo rimane sostanzialmente invariato.

3.1 Caso specifico

In questo elaborato viene realizzato un sistema con $n = 4$ processori ed $m = 3$ bus. Sul modello risultante, viene richiesto di verificare tramite simulazione le seguenti proprietà:

- Safety: mutua esclusione nell'accesso ai bus
- Liveness: assenza di *livelock*, cioè di computazioni nelle quali un processore ripete indefinitamente richieste di accesso che non vengono mai servite
- Liveness: assenza di *deadlock*, ovvero di computazioni in cui un processore non è in grado di accedere al bus perché il protocollo non garantisce di completare la comunicazione.

4 Implementazione

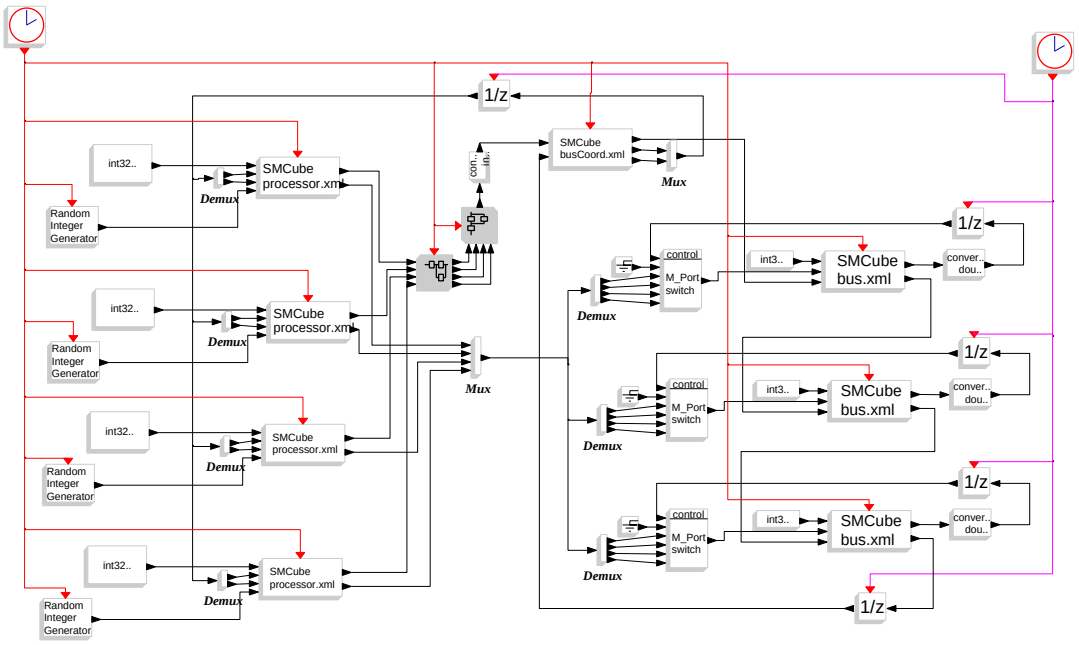
Il modello descritto nella sezione 3 è stato realizzato tramite un diagramma Scicos, utilizzando SMCube per modellare processori, bus e coordinatore dei bus. In figura 7a è mostrato il diagramma relativo al caso Equal Priority, mentre la figura 7b rappresenta il modello Fixed Priority. Ovviamente, i due modelli sono del tutto identici a parte per come viene gestita la coda delle richieste che arrivano al coordinatore.

È importante evidenziare due decisioni di progetto che sono state prese nella rappresentazione del modello:

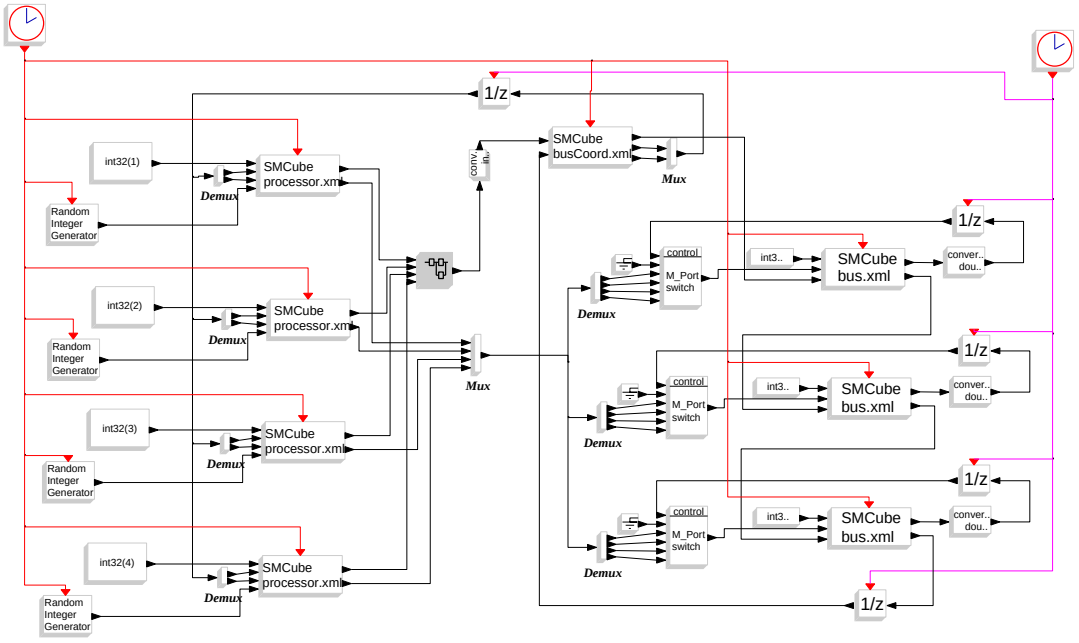
1. **Disporre i bus secondo una topologia ad anello orientato**. Il motivo di questa scelta deriva da un controllo che Scicos effettua in fase di compilazione del modello per rilevare *loop algebrici*, ossia cicli di input e output potenzialmente infiniti che portano al fallimento della simulazione (figura 8a). È quindi necessario aggiungere un elemento di ritardo qualora due blocchi debbano comunicare reciprocamente (o un blocco utilizzi una sua risposta come input, come accade nell'esempio di figura 8b). Questo significa che, nel caso di una struttura lineare come quella mostrata in figura 6, ogni messaggio della seconda fase del protocollo subirebbe un ritardo di un colpo di clock, rendendo la complessità temporale del protocollo lineare rispetto al numero di nodi. Con una struttura ad anello orientato, invece, occorre inserire un blocco ritardo soltanto tra due bus. La configurazione ottimale prevede il ritardo tra l'ultimo bus dell'anello (ossia quello che precede il coordinatore del 2PC) ed il coordinatore: così facendo, in un colpo di clock viene deciso quale bus soddisferà la richiesta e nel secondo colpo di clock viene raggiunta conoscenza comune.

2. **Separare la logica del coordinatore del 2PC da quella dei bus**. In questo modo tutti i bus hanno lo stesso comportamento, evitando di dover differenziare la logica del bus coordinatore da quella degli altri. Inoltre, l'automa a stati finiti di un coordinatore "semplice" è ovviamente meno complesso di quello di un coordinatore che è anche un bus, diminuendo quindi la possibilità di errori in fase di implementazione. Il costo da pagare è soltanto l'aggiunta del nodo coordinatore all'interno dell'anello dei bus.

Il modello è costruito per via grafica ed è abbastanza autoesplicativo: è facile vedere come i processori, il coordinatore ed i bus sono collegati tra di loro. Vale la pena però porre l'attenzione sugli switch antecedenti ai bus che filtrano gli output dei processori. Mentre per effettuare la richiesta i processori comunicano di fatto solo con il coordinatore, una volta garantito l'accesso la comunicazione con i bus avviene per via diretta, per evitare di aggiungere ulteriore carico di lavoro al coordinatore e rallentare le comunicazioni. Questo vuol dire che tutti i processori sono collegati a tutti i bus, ma *non* è vero che tutti i segnali inviati dai processori arrivano a tutti i bus: ogni bus, infatti, controlla il proprio switch in modo che solo i segnali inviati dal processore che ha l'accesso possano arrivare.

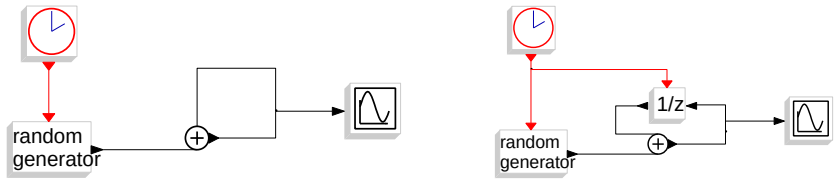


(a) Diagramma del modello Equal Priority.



(b) Diagramma del modello Fixed Priority.

Figura 7: I due modelli implementati: Equal Priority (a) e Fixed Priority (b). In grigio sono evidenziati i blocchi che costituiscono la coda di priorità.



(a) **(b)**

Figura 8: La figura (a) mostra un diagramma Scicos che contiene un loop algebrico, mentre in figura (b) è raffigurato un diagramma equivalente che risolve il problema inserendo un blocco ritardo.

Nel seguito descriveremo gli automi che modellano i processori, i bus e il coordinatore dei bus.

4.1 Processore

L'automa del processore è riprodotto in figura 9. È funzionalmente equivalente a quello riportato nelle specifiche e mostrato in figura 5. Da notare che il tempo di soggiorno negli stati `Idle` e `Using` è random e viene campionato al momento dell'entrata nello stato.

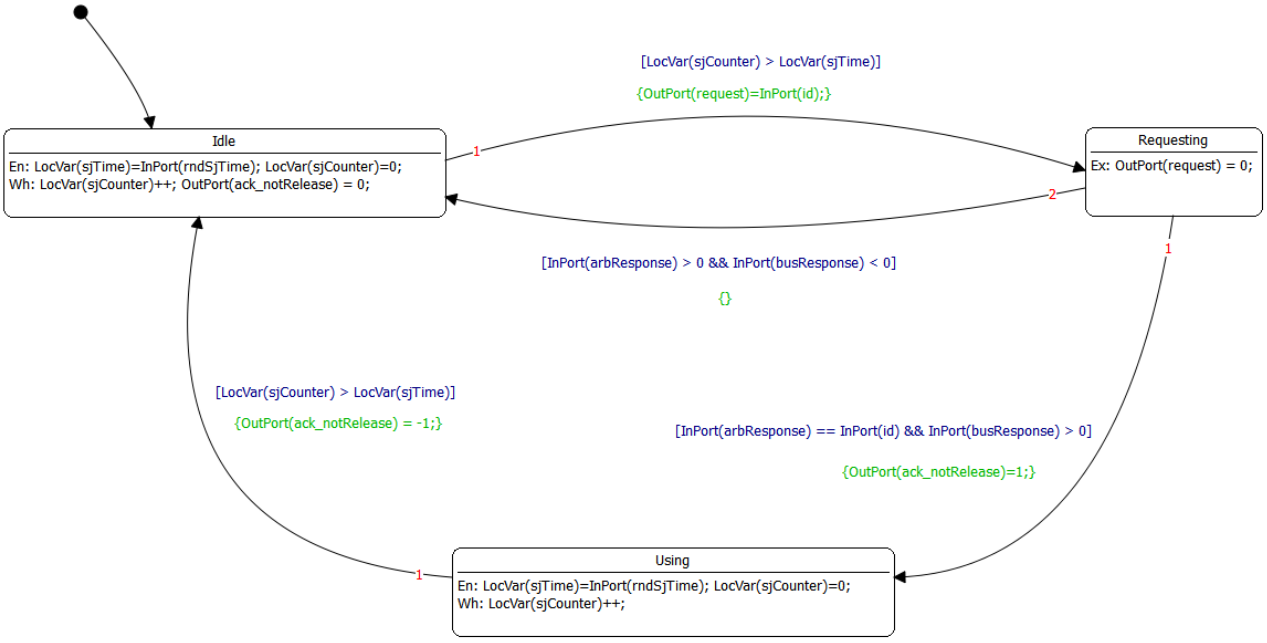


Figura 9: ASF di un processore.

4.2 Coordinatore dei bus

L'automa del coordinatore viene mostrato in figura 10. È composto da tre stati:

- `Idle`: il coordinatore è in attesa di richieste da parte dei processori.
- `NotifyingBuses`: è arrivata una richiesta ed è stata inoltrata ai bus, dando via alla prima fase del 2PC. Il coordinatore è in attesa di sapere quale bus ha accolto la richiesta.
- `Committing`: la richiesta è passata per tutti i bus dell'anello, tornando al coordinatore. Questo significa che a tutti i bus è arrivata la notifica della richiesta e sono tutti in attesa del commit. Il messaggio che arriva al coordinatore contiene l'id del bus che soddisferà la richiesta (o -1 se non ci sono bus disponibili) e il coordinatore non deve fare altro che inoltrarlo sia ai bus che ai processori. Quando il messaggio percorre tutto l'anello e ritorna al coordinatore, tutti i bus e i processori sono stati notificati del risultato della richiesta e il coordinatore può tornare nello stato di `Idle`.

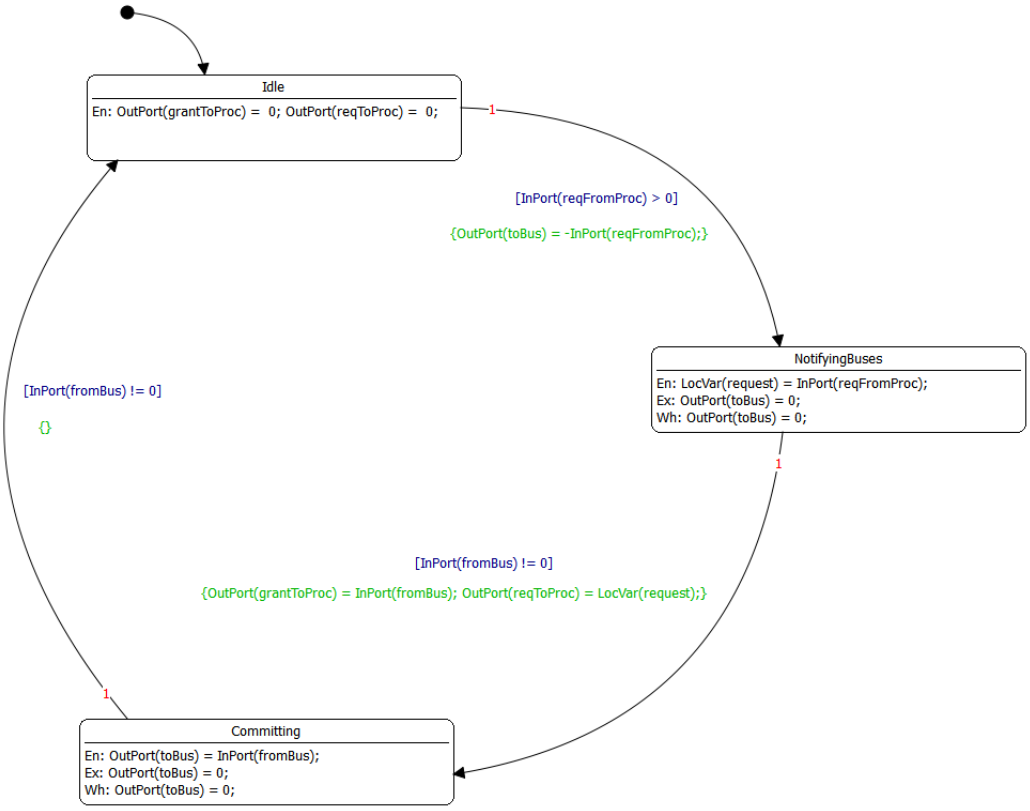


Figura 10: ASF del coordinatore dei bus.

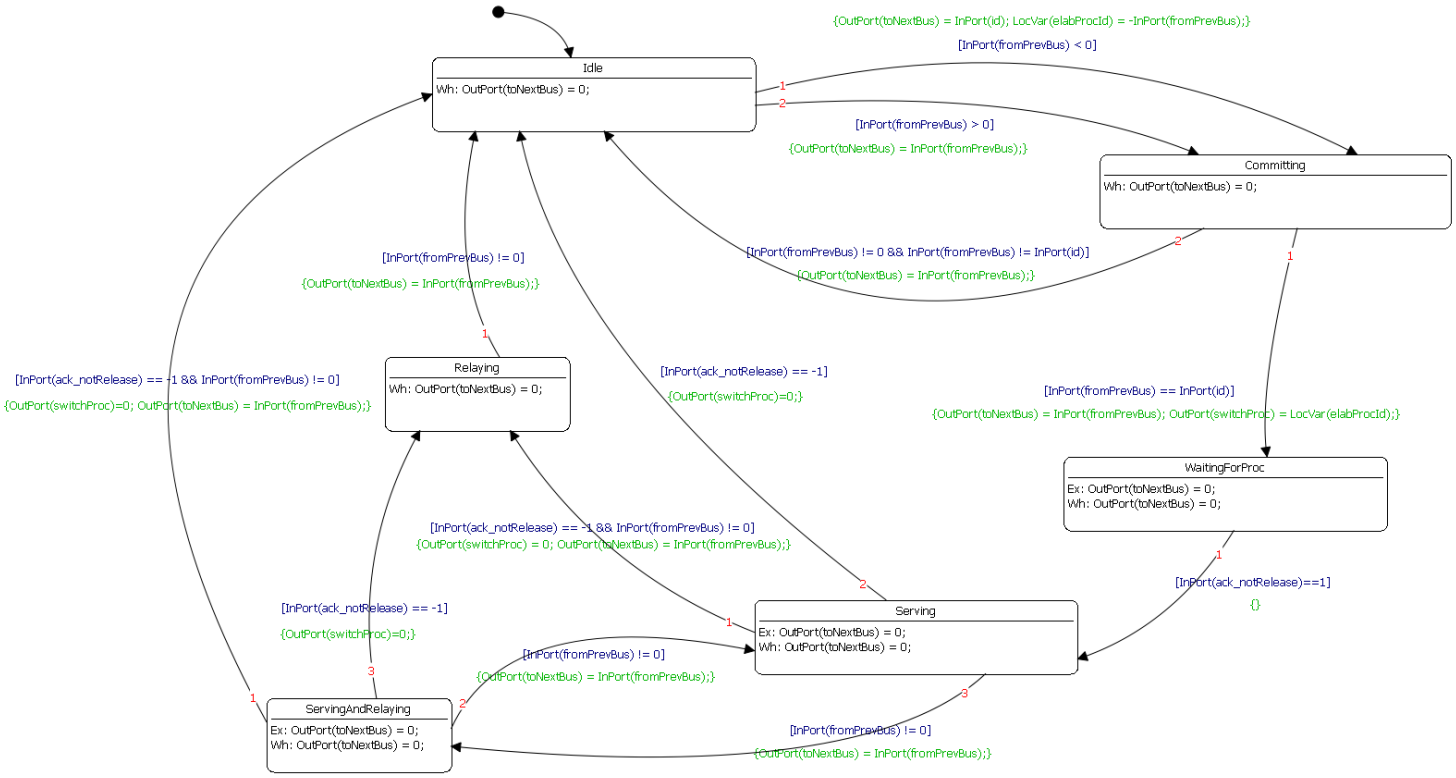


Figura 11: ASF dei bus.

4.3 Bus

I bus sono caratterizzati dall'ASF riportato in figura 11. Rispetto alla macchina di figura 5 sono presenti stati aggiuntivi, derivanti dalla necessità di poter eseguire il 2PC e di doverlo fare sia mentre il bus è `Idle` che mentre serve un processore (stato `Serving`).

Gli stati `Idle` e `Committing` sono i due stati necessari per il funzionamento del 2PC.

Inizialmente, il bus è `Idle`. Quando arriva una richiesta il bus entra nello stato `Committing`. Questo passaggio di stato può essere effettuato in due modi possibili: se in esso viene comunicato che la richiesta è già stata accolta, inoltrando semplicemente il messaggio; altrimenti, comunicando di incaricarsi della richiesta. Quando il messaggio compie il giro dell'anello e ritorna al bus, il 2PC è (localmente) completato e il bus può uscire dallo stato di `Committing`, andando nello stato `WaitingForProc` se deve servire il processore o tornando nello stato `Idle` se è stato invece un altro bus ad accogliere la richiesta.

Nello stato `WaitingForProc` il bus è in attesa del segnale `ack` da parte del processore che deve servire. Da notare che in questo stato il bus *non* può prendere parte al 2PC. Questo però non è necessario, in quanto il sistema modellato

è sincrono ed esente dai guasti: il segnale `ack` viene inviato dal processore e ricevuto dal bus prima che un ulteriore richiesta possa essere processata dal coordinatore. Una volta ricevuto il segnale `ack` il bus può servire il processore, passando allo stato `Serving`.

In linea di principio, un bus rimane `Serving` fino a quando non riceve un segnale `release` dal processore, ritornando conseguentemente nello stato `Idle`. Questo meccanismo è reso più complicato dalla necessità del bus di dover eseguire il 2PC anche mentre serve un processore, per evitare il blocco delle comunicazioni.

Se un 2PC viene iniziato mentre il bus sta servendo un processore, la richiesta non può essere localmente accolta: l'unica azione che il bus può eseguire è quella di ritrasmettere il messaggio nell'anello (*relaying*), passando quindi nello stato `ServingAndRelaying`. Una volta che la prima fase del 2PC viene portata a termine e arriva il messaggio della seconda fase, il processore può tornare nello stato `Serving`.

Lo stato `Relaying` e le altre transizioni uscenti dagli stati `Serving` e `ServingAndRelaying` sono necessarie per gestire i casi in cui il processore rilascia il bus prima che il 2PC venga portato a termine.

5 Simulazione e verifica delle proprietà

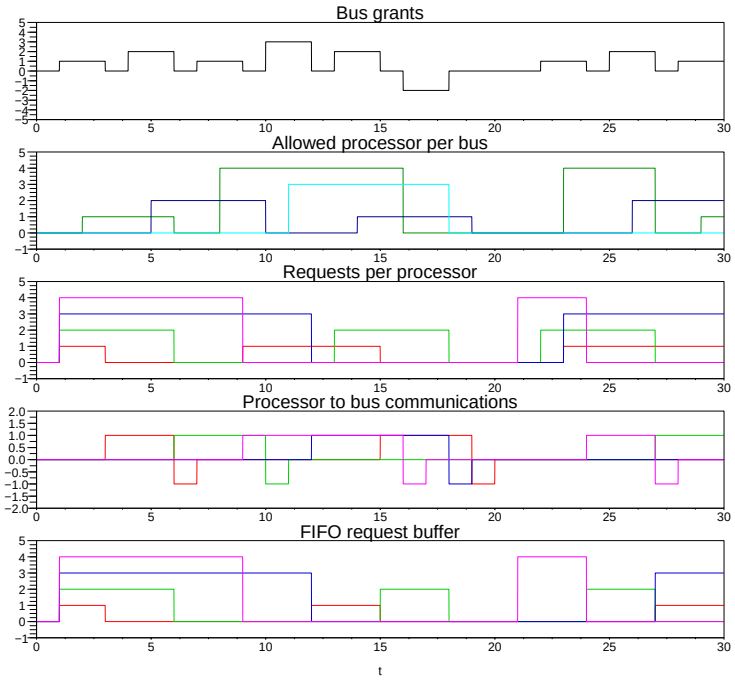
L'ultima fase dell'elaborato è verificare tramite simulazione che il modello funzioni e soddisfi le proprietà di safety e liveness richieste (si veda §3.1). Nell'ultimo paragrafo si discuterà anche della scalabilità del modello proposto.

In figura 12 sono mostrate le simulazioni effettuate osservando i segnali scambiati per 30 colpi di clock. Esamineremo prima il funzionamento delle code in tutti e due gli scenari e, poiché questa è l'unica differenza tra i due modelli, successivamente non tratteremo necessariamente entrambi i casi.

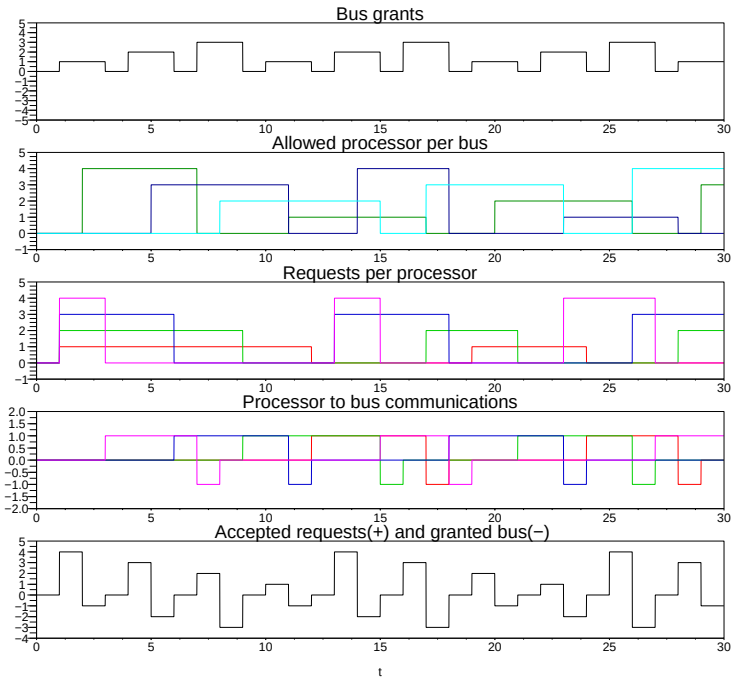
5.1 Verifica della correttezza delle code di priorità

Consideriamo prima il modello Equal Priority. In figura 12a vengono mostrati sia le richieste dei processori che l'output del buffer FIFO. Il comportamento atteso è che, all'interno del buffer, una richiesta acquista tanta più priorità quanto più tempo è trascorso da quando è stata fatta. Questo è esattamente quello che succede: richieste che arrivano quando il buffer non è vuoto non vengono inoltrate al coordinatore fino a quando tutte le precedenti non sono state accolte, come è possibile notare dalla richiesta che il processore 1 effettua al tempo $t = 9$. Se più richieste hanno priorità massima, ne viene scelta una in modo casuale.

Consideriamo ora il caso Fixed Priority, mostrato in figura 12b. In questo scenario, un processore ha priorità tanto più alta quanto maggiore è il suo ID. Questo significa che, se ci sono richieste contemporanee da più processori, viene



(a) Equal Priority



(b) Fixed Priority.

Figura 12: Simulazioni di durata 30 colpi di clock eseguite con entrambi i modelli.

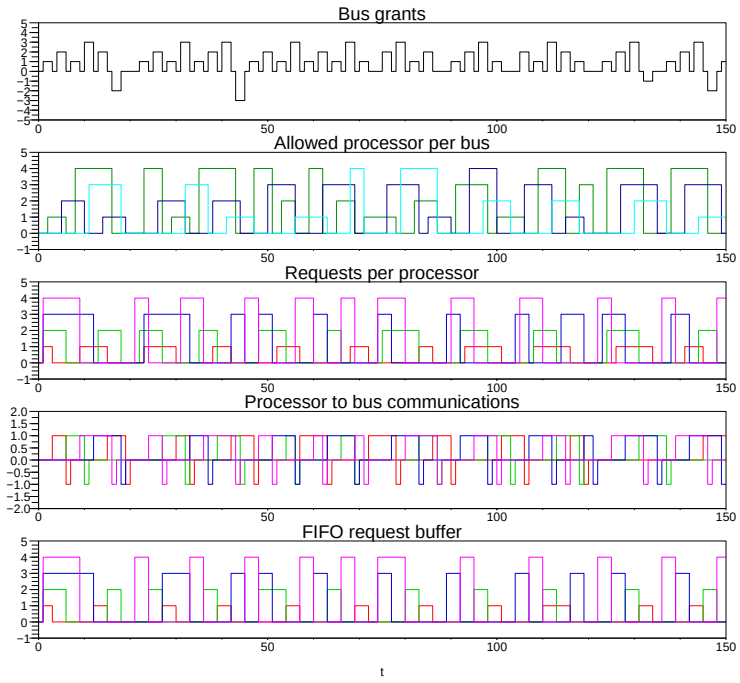


Figura 13: Simulazioni di durata 150 colpi di clock eseguite con il modello Equal Priority.

considerata solo quella del processore a priorità maggiore. Questo comportamento si può notare già al tempo $t = 1$: tutti i processori hanno fatto richiesta, ma, come si evince dall'ultimo grafico, è la richiesta del processore 4 ad essere accolta (ed è soddisfatta dal bus 1).

5.2 Mutua esclusione nell'accesso ad un bus

La proprietà di mutua esclusione può essere verificata già esaminando la macchina a stati del bus (figura 11). Violare la mutua esclusione significa che, mentre il bus sta servendo un processore, accetta una richiesta proveniente da un processore diverso. Vediamo perché questo non è possibile.

Il segnale inviato allo switch dei processori viene impostato soltanto entrando nello stato `WaitingForProc` e viene resettato quando si smette di servire il processore (i.e., all'uscita dagli stati `Serving` o `ServingAndRelaying`). Questo significa che, mentre sta servendo, un bus può comunicare solo con il processore che ha fatto richiesta. D'altro canto, l'unico modo che il bus ha per uscire dagli stati Serving è ricevere un segnale di rilascio da parte del processore. Poiché il processore invia il segnale `release` soltanto quando smette di usare il bus (figura 9), l'unico modo che il bus ha per iniziare la comunicazione con un altro processore è che il precedente effettui un rilascio. Questo garantisce la mutua esclusione nell'accesso ai bus.

La "dimostrazione" teorica appena fornita è confermata dall'esito della simulazione: un bus garantisce l'accesso ad un processore soltanto se nessun altro processore possiede i diritti di accesso al bus, come è possibile notare dal secondo diagramma delle figure 12.

5.3 Assenza di livelock

In linea di principio, con il modello adottato l'assenza di livelock non può essere garantita: è sempre possibile costruire un'esecuzione in cui più processori fanno richiesta e uno di essi non viene mai servito perché possiede priorità inferiore rispetto agli altri o perché tutti i bus sono già occupati. Tuttavia, come mostrano entrambe le figure 12 e 13, nella pratica questa eventualità è rara: se i processori fanno richieste e rilasciano i bus ad intervalli di tempo casuali (e dello stesso ordine di grandezza), la probabilità che un processore debba attendere un intervallo di tempo infinito per accedere ad un bus è nulla.

5.4 Assenza di deadlock

Teoricamente, l'assenza di deadlock può essere verificata già dal diagramma del modello e dalle macchine a stati del coordinatore e dei bus, esaminando quali sono le condizioni per cui un automa non può più uscire da uno stato e controllando che esse non possano essere soddisfatte. Questo tipo di considerazioni sono state quelle che hanno guidato la modellazione degli automi, per cui il sistema è stato progettato prestando particolare attenzione a soddisfare questa proprietà. L'assenza di deadlock è stata poi verificata per via sperimentale: come si può vedere in figura 13, in intervalli di tempo prossimi alla fine tutti i componenti funzionano ancora correttamente.

5.5 Analisi di scalabilità

Come si può vedere dalle figure 9, 10 e 11, nella definizione degli automi non vi è alcun parametro che faccia riferimento a quanti processori o bus siano presenti nel sistema. Inoltre, poiché all'interno di Scicos è possibile l'invio di messaggi broadcast e poiché un messaggio può essere ricevuto da un blocco e inoltrato nello stesso colpo di clock, l'incremento del numero di processori o di bus non comporta l'aumento del tempo necessario all'esecuzione dell'algoritmo di bus arbitration. Da questi punti di vista, quindi, il modello risulta facilmente scalabile.

La costruzione del modello, tuttavia, è stata effettuata con Scicos per via grafica. Questo significa che bisogna aggiungere manualmente ulteriori nodi processore o bus ed effettuare manualmente tutti i collegamenti necessari. Questa restrizione è piuttosto significativa ed è il motivo per cui, in ultima istanza, il modello risulta difficilmente scalabile.

Riferimenti bibliografici

[1] Manuale SMCube. `http://www.evidence.eu.com/download/smcube/smcube_manual_1.0.pdf`.

[2] Scicos. `http://www.scicos.org/`.

[3] SMCube. `http://erika.tuxfamily.org/drupal/sm-cube.html`.