# Simulazione di una comunicazione fra dispositivi che utilizzano il protocollo i2c

Piccolo Fabio
Matricola: 83101395
E-mail: fbipic@yahoo.com

# Simulazione di una comunicazione fra dispositivi che utilizzano il protocollo i2c

## *Sommario*

La comunicazione fra due o più dispositivi elettronici si realizza attraverso un insieme di linee, chiamate bus. Affinché gli elementi interagiscano correttamente, è necessario stabilire delle specifiche di comunicazione. L'insieme delle regole di trasmissione prende il nome di protocollo.
La Philips ha brevettato un protocollo di comunicazione, chiamato i2c, il quale consente il colloquio fra più dispositivi collegati fra loro attraverso un bus composto da due linee.
In questa relazione si vuole simulare il comportamento di due elementi che colloquiano utilizzando le regole previste dal protocollo dalla Philips. In particolare è stato creato un unico elemento, descritto in linguaggio VHDL, il quale dovrà consentire sia la trasmissione che la ricezione di dati.
La simulazione dell'elemento è stata effettuata con il tool Sirocco della Synopsys. Si presentano inoltre i passi necessari ad utilizzare tale software per visualizzare la trasmissione fra due elementi collegati allo stesso bus.

## *Introduzione*

Il protocollo i2c specifica le regole di comunicazione fra più dispositivi collegati ad un bus formato da due linee. Esso è stato ideato dalla Philips per permettere a dispositivi differenti di colloquiare fra di essi in modo da consentire lo scambio di informazioni. Sul mercato esistono molte realizzazioni che adottano questo protocollo per comunicare fra loro.
In questa relazione, viene descritto tramite il linguaggio VHDL, il comportamento di un unico dispositivo che può funzionare sia da master che da slave; ossia può sia dare inizio alla comunicazione, che adempiere ad una richiesta ricevuta.
Dopo aver studiato il protocollo di comunicazione i2c sono stati individuati quali sono gli elementi necessari per simulare il comportamento di un dispositivo di comunicazione che sfrutti le specifiche previste.
Una volta definiti i registri necessari è stata eseguita la loro descrizione con il linguaggio VHDL. Per verificarne il funzionamento è stata fatta una simulazione mediante l'utilizzo di un banco di prova specifico per ogni componente.
Dopo averli testati separatamente sono stati interconnessi, e simulati "globalmente" in modo da coordinarne il funzionamento.
Nella presente tesina si presenta innanzitutto il protocollo i2c.

Si illustra, quindi, il funzionamento dei componenti che consentono di simulare il dispositivo di comunicazione.
La coordinazione di tutti gli elementi è effettuata da un unico componente che utilizza al suo interno una macchina a stati. Questo elemento, e il funzionamento della macchina a stati, sono descritti in maniera molto approfondita.
In seguito si presentano i risultati della simulazione e le conclusioni ottenute.
Successivamente sono descritti tutti i passaggi per effettuare ulteriori simulazioni.

## *Specifiche del protocollo i2c*

Il protocollo i2c prevede l'utilizzo di un bus formato da due linee bidirezionali. Le due linee, chiamate "scl" e "sda" rispettivamente, trasportano la tempistica di sincronizzazione (chiamata anche "clock") e i dati.
I segnali che transitano sulla linea hanno valore '1' o '0' e le tensioni che li rappresentano sono quelle d'alimentazione e di massa rispettivamente.
Le due linee non sono lasciate ad un valore indefinito, ma vengono collegate all'alimentazione attraverso una resistenza di pull-up. In questo modo le due linee permangono ad un valore "alto-debole", facilmente modificabile da un dispositivo.
Grazie all'utilizzo delle resistenze di pull-up, per ottenere un valore '1' sulla linea sarà sufficiente mettere il segnale d'uscita in alta impedenza. In questo modo se un altro dispositivo impone un valore '1' ed un altro un valore '0', quest'ultimo segnale sarà prevalente sul precedente. Questo consente di adoperare un meccanismo d'arbitraggio, spiegato in dettaglio di seguito.
Ogni dispositivo collegato a queste linee è dotato di un indirizzo univoco, di 7 o 10 bit e può agire sia da master che da slave, secondo le funzioni previste al suo interno. Il master si occupa di iniziare la trasmissione e di generare la tempistica del trasferimento, mentre lo slave è quello che riceve una richiesta. Entrambe le due categorie appena descritte possono assumere il ruolo di trasmittente o ricevente.
Le modalità di trasferimento dati possono essere riassunte come segue:

I) A trasmette dati a B
   1. A(master) spedisce l'indirizzo di B(slave) sul bus
   2. A (master-transmitter) trasmette i dati a B (slave-receiver)
   3. A termina il trasferimento

II) A vuole ricevere dati da B
   1. A (master) spedisce l'indirizzo di B(slave) sul bus
   2. B ( slave-transmitter) spedisce i dati ad A (master-receiver)
   3. A termina il trasferimento

Un esempio di trasmissione completa utilizzando il protocollo i2c è quello che compare in Fig. 1. In particolare si fa riferimento a dispositivi con indirizzo lungo sette bit.

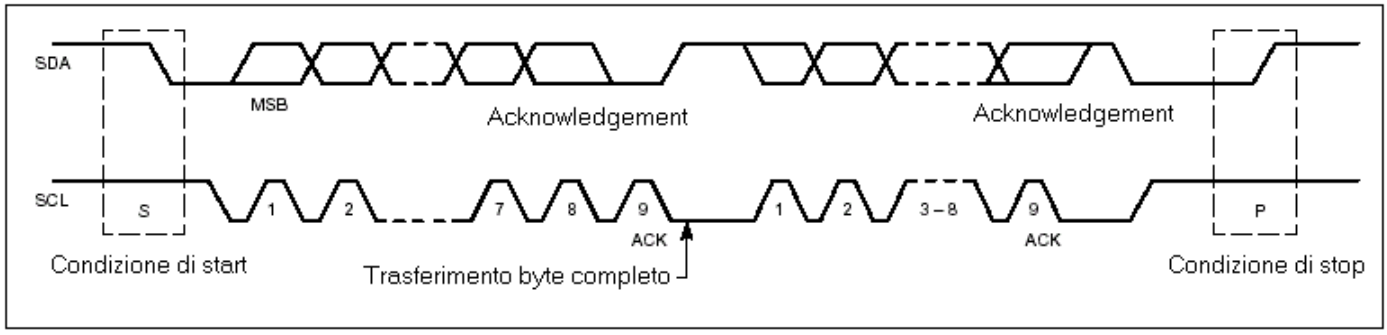



Fig.1

La trasmissione inizia con la generazione di un segnale di start, immesso sulla linea dal master, dopo un controllo sull'occupazione del bus.
La condizione di start consiste nel lasciare la linea "scl" allo stato "alto", mentre la linea "sda", subisce una transizione dallo stato '1' allo stato '0'. Nella Fig. 1 è rappresentata la condizione di start nella parte a sinistra.
Dopo la generazione del segnale di start inizia la trasmissione dei dati vera e propria.
Il primo byte trasmesso è quello composto dall'indirizzo dello slave con l'aggiunta di un ulteriore valore che indica al ricevente quale è l'operazione a lui richiesta. In base allo standard previsto dal protocollo questo bit può assumere due valori con i seguenti significati:

- '0' : ricezione dati;
- '1' : trasmissione dati;

L'ordine di trasmissione dei bit è quello dal più significativo al meno significativo.
Le successive comunicazioni seguono sempre quest'ultimo criterio.
Dopo la trasmissione d'ogni byte chi trasmette ha l'obbligo di lasciare la linea "sda" allo stato "alto", in modo da permettere a chi riceve dei dati, di darne conferma tramite il meccanismo dell'acknowledgement: esso consiste nell'abbassare la linea "sda" in corrispondenza del nono impulso presente sulla linea "scl".
Un esempio del processo appena esposto lo si può vedere nella Fig. 2 così come è presente nella Fig. 1.

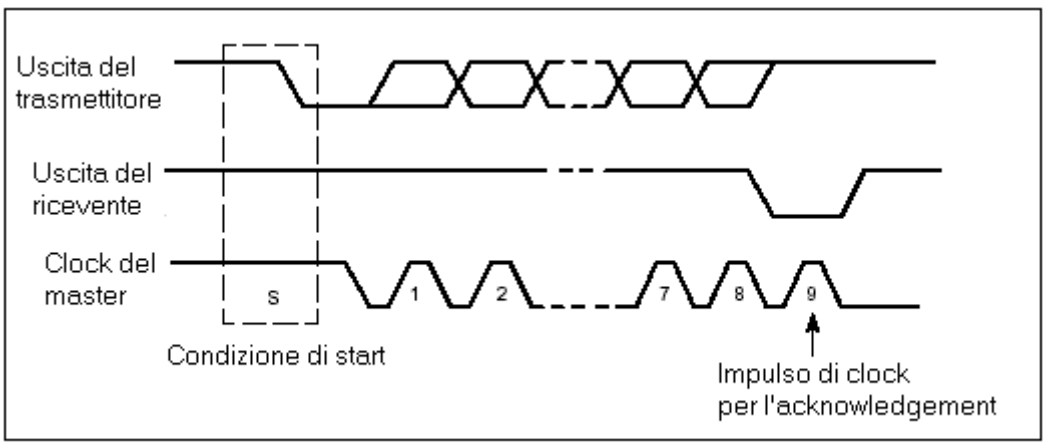



Fig. 2

Se l'ack non venisse generato, chi ha iniziato la trasmissione può interrompere la comunicazione, utilizzando la condizione di stop.
Un dispositivo può non generare il segnale di acknowledgement ad esempio, quando è inabilitato a ricevere, perché sta eseguendo delle funzioni in real-time, oppure quando non può più immagazzinare altri dati.
Il segnale di stop è sempre generato dal master, come pure quello dello start. Esso consiste nel far variare la linea sda dallo stato "basso" a quello "alto" in corrispondenza del periodo "alto" della linea "scl". Un esempio di questa condizione è visibile nella parte a destra della Fig. 1.

Durante la trasmissione avvengono contemporaneamente due processi:

1. la sincronizzazione dei clock
2. l'arbitraggio

Il primo processo permette a due dispositivi con velocità di funzionamento differenti di comunicare senza incorrere nella perdita di dati.
Il clock di un elemento è contraddistinto da un periodo "alto" o "positivo", in cui assume un valore '1', e da uno "basso" o "negativo", in cui assume il valore '0'. Entrambi i valori devono essere mantenuti per un certo intervallo di tempo, il quale può essere diverso nei due casi.
Un esempio di sincronizzazione del "clock" compare in Fig. 3.

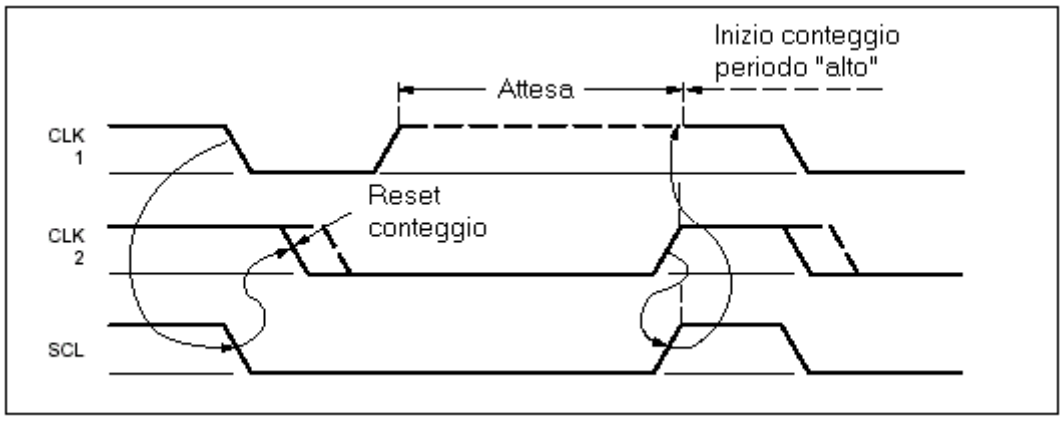


Fig. 3

Nell'immagine sono rappresentati i segnali dei clock interni di due dispositivi, chiamati rispettivamente "clk1" e "clk2", che hanno periodi, sia "alti" che "bassi", con durate differenti.
La sincronizzazione dei due segnali sfrutta il criterio di "prevalenza" fra i valori '1' e '0', esposto in precedenza.
Nella figura si vede che entrambi i dispositivi lasciano sulla linea "scl" un valore "alto" per la durata del periodo del fronte "positivo" del clock. Una volta esaurito questo intervallo di tempo, i due elementi mettono sulla linea "scl" un valore '0', mantenendolo per il tempo di durata di questo periodo. Il fronte negativo di "clk2"

risulta superiore a quello di “clk1”, quindi sulla linea “scl” avremo un valore negativo fino a che il secondo dispositivo non terminerà il periodo “basso”.
La regola di sincronizzazione prevede che il primo elemento, rilevando sulla linea “scl” un valore ‘0’, deve attendere fino a che la linea non ritorni al valore ‘1’. Una volta che questo accade, manterrà il valore ‘1’ per la durata del periodo “alto”, per poi passare al valore ‘0’ o “basso”. Il processo continuerà sempre nello stesso modo.
Usando questa tecnica il “clock” generato ha un periodo “basso” pari al più lungo periodo “basso” dei dispositivi, mentre il periodo alto è il più breve fra i periodi “alti” degli elementi collegati al bus. Dalla Fig. 3 si vede questa situazione ed inoltre si nota che il segnale sulla linea “scl” è differente da quello dei segnali “clk1” e “clk2”.

L’arbitraggio consente di utilizzare il bus come un multi-master, nel senso che è possibile collegare fra loro più elementi con la facoltà di iniziare trasferimenti di dati, senza che avvengano perdite d’informazione.
Il processo consiste nel paragonare ciò che si trasmette con quello che effettivamente si trova sulla linea “sda”. Quando due dispositivi trasmettono due valori differenti, quello prevalente risulta essere il valore “basso”.
Nel caso in cui un master trasmetta un livello ‘0’ e un altro trasmetta un livello ‘1’, quest’ultimo dovrà disabilitare il suo stato d’uscita, poiché sulla linea vedrà un valore diverso da quello che gli voleva trasmettere.
Nel caso in cui il dispositivo abbia capacità sia di master che di slave, è possibile che l’altro elemento lo stia contattando, quindi dovrà passare all’istante, dallo stato master allo stato slave.
La procedura d’arbitraggio permette a due elementi di iniziare entrambi la trasmissione e di continuarla fino a che il processo va a buon fine. In questo modo non si perdono dati.
La Fig.4 illustra un esempio d’arbitraggio fra due elementi.

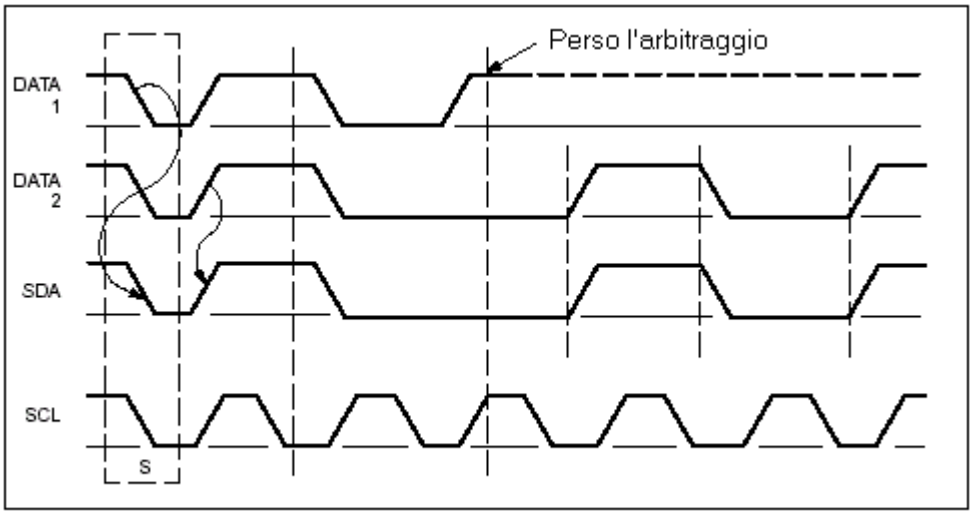


Fig. 4

Il formato delle trasmissioni potrà assumere quindi i raggruppamenti di bytes presentati nelle figure seguenti. In particolare esse rappresentano i tipi di comunicazione :

- Fig. 5 : Master-transmitter Slave-receiver
- Fig. 6 : Master-receiver Slave-transmitter
- Fig. 7 : modalità combinata

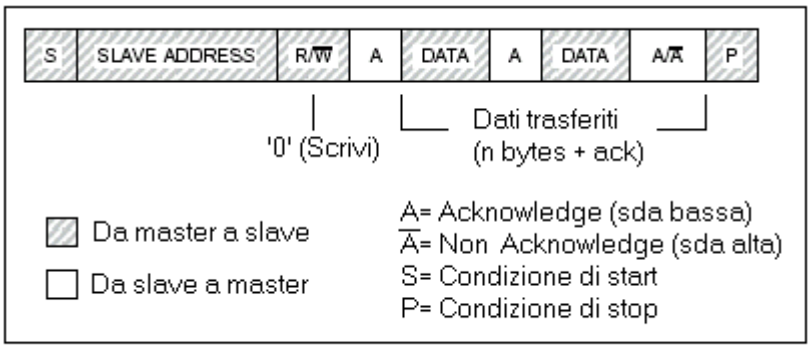



Fig. 5

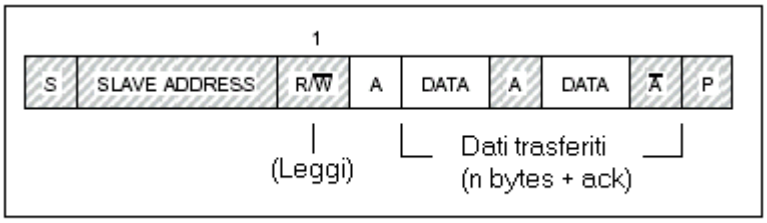



Fig. 6


Fig. 7

Si nota nella Fig. 7 che la direzione dei dati e dei bit dipendono dal bit che imposta la lettura e la scrittura.

### *Descrizione del dispositivo*

L'elemento descritto attraverso il linguaggio VHDL, con funzioni sia di master che di slave, è quello illustrato in Fig. 8.

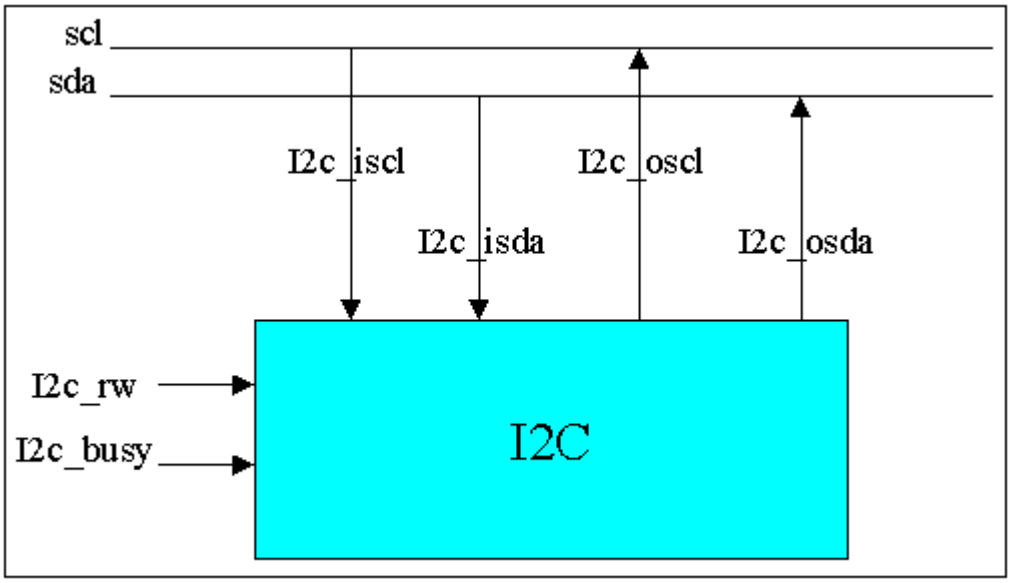



Fig. 8

I pin rappresentati nella parte alta della figura vengono collegati alle linee del bus. Una coppia di questi è utilizzata come ingresso, mentre l'altra è utilizzata come uscita.
Il collegamento chiamato I2c_rw serve per specificare la modalità di funzionamento del dispositivo. Questa è scelta in base alla seguente logica:

- '0' : master-transmitter
- '1': master-receiver

L'elemento si configura, invece, nello stato di slave quando rileva una condizione di start, indipendentemente dal valore del pin I2c_rw.
Specificare il valore di questo consente al dispositivo di iniziare la comunicazione, configurandosi nello stato master. La prima azione compiuta è quella di leggere un byte da un file. Quest'ultimo è costituito da una serie di righe di otto caratteri ciascuna, le quali vengono intrepretate come indirizzi o come dati a seconda dell'istante in cui vengono lette. Nel qual caso si tratti della prima comunicazione fra due elementi, esse rappresentano un indirizzo ( 7 bit più un ulterriore bit utilizzato dallo slave per capire quale operazione gli viene richiesta ). In caso contrario le righe vengono intrepretate come dati da trasmettere.
Nel caso ci siano due elementi settati in modo da svolgere entrambi la funzione di master, chi vince la procedura d'arbitraggio può richiedere all'altro delle informazioni, facendolo commutare nello stato di slave.
Il pin I2c_busy serve per indicare al dispositivo che non può trasmettere e/o ricevere più dati, quindi a porre fine al trasferimento generando il segnale di stop, nel caso

funzioni da master, oppure non creando il segnale d'acknowledgement se svolge il compito di slave.
La realizzazione del master/slave è stata portata a termine utilizzando i componenti visibili in Fig.9.

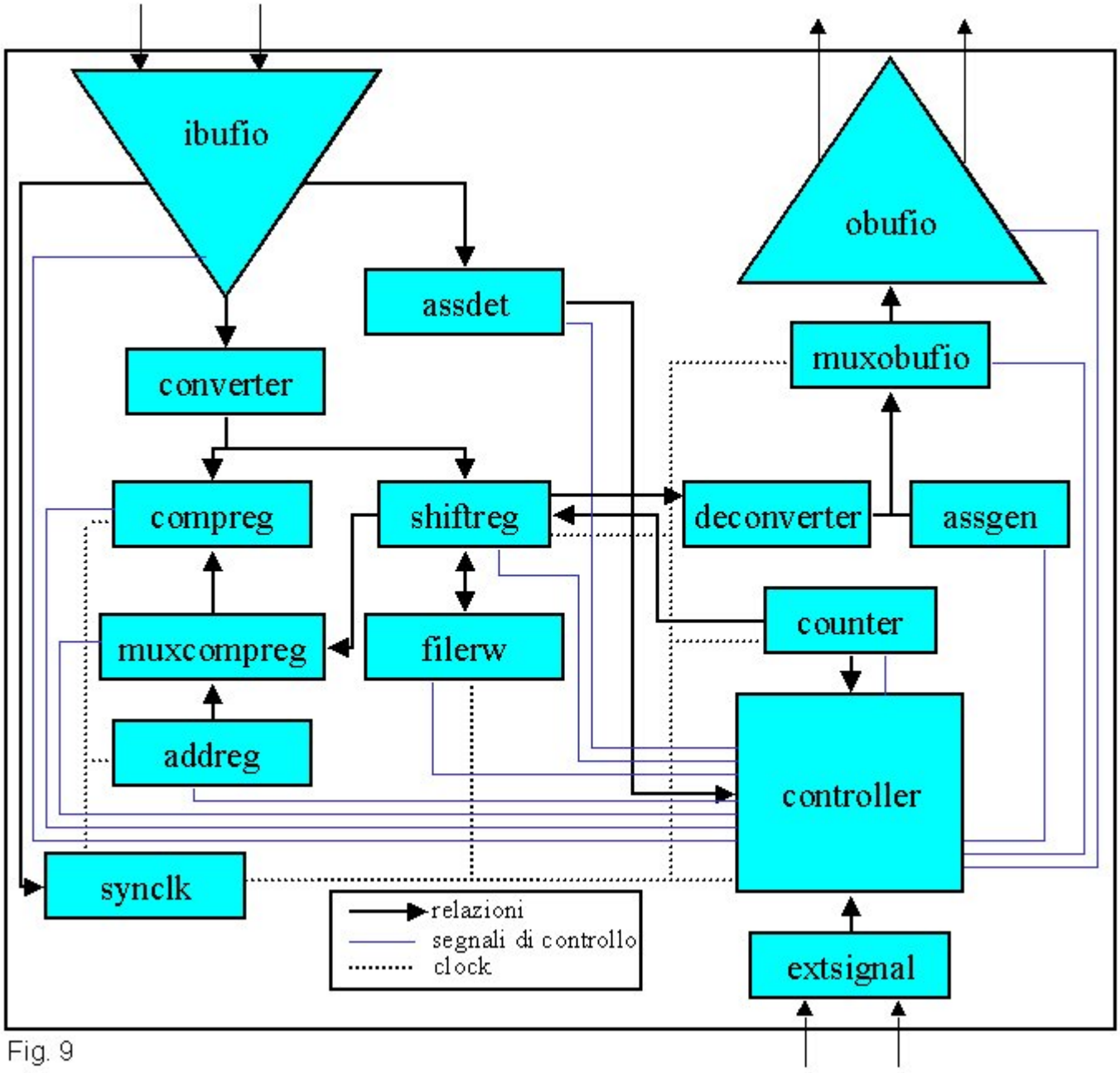



Fig. 9

In quest'immagine si rappresenta quali sono le relazioni funzionali fra i vari elementi, le linee del clock e quelle dei segnali di controllo che partono dal controller. Non sono rappresentati tutti i fili che collegano gli elementi per problemi d'elevata complessità e per difficoltà di rappresentazione.
Dalla figura si può notare l'elemento fondamentale nella realizzazione software di questo dispositivo è il registro <controller> al quale arrivano molti segnali e dal quale partono tutti i segnali di controllo. Con questa definizione s'intendono sia i segnali d'abilitazione sia quelli per la scelta della modalità di trasferimento.

## *Funzionamento dei singoli elementi*

### Addreg (Address Register)

Il registro <addreg> contiene al suo interno l'indirizzo identificativo del dispositivo e lo fornisce in maniera seriale ad ogni fronte positivo di clock, quando l'elemento viene abilitato da parte del controller. Per la scansione dell'indirizzo non è usato un indice esterno.

### Assdet (Acknowledgement Start Stop Detector)

Una caratteristica particolare dell'elemento <assdet> è quella di non avere un clock di funzionamento in quanto esso, una volta attivato, deve essere in grado di riconoscere, in ogni istante, un segnale "tipico" indipendentemente dal segnale che regola il funzionamento interno di un dispositivo.
Lo scopo di questo registro è quello di rilevare i vari segnali "particolari" che caratterizzano la trasmissione utilizzando il protocollo i2c. Essi sono:

- bus libero
- condizione di start
- acknowledgement
- condizione di stop

Il tipo di segnale da rilevarsi, è deciso dal <controller> attraverso un segnale che seleziona la modalità opportuna in ogni fase del trasferimento e/o ricezione di dati.
Il registro viene collegato direttamente al buffer di ingresso in quanto è stato progettato in modo da riconoscere i segnali direttamente.

### Assgen (Acknowledgement Start Stop Generator)

Anche per il registro <assgen> vale il discorso fatto in precedenza per il registro <assdet>. In questo caso però, esso non è collegato direttamente al buffer d'uscita <obufio>, ma connesso attraverso il multiplexer <muxobufio>.
Questo componente, collegato al buffer di uscita attraverso il multiplexer <muxobufio>, genera i segnali caratteristici del protocollo i2c. Essi sono:

- condizione di start
- acknowledgement
- condizione di stop

Anche in quest'evento esiste un bus che permette al <controller> di scegliere la modalità di funzionamento del dispositivo più opportuna in ogni fase della trasmissione.

### Compreg (Comparator Register)

Il registro <compreg> compie la comparazione fra due dati restituendo il valore '1': quando i due valori in ingresso sono uguali e '0' quando sono differenti. Esso è usato per eseguire l'arbitraggio durante la trasmissione da parte del master e dallo slave,

per controllare se l'indirizzo presente sulla linea è quello relativo al dispositivo di cui fa parte. Il componente dovrà quindi essere collegato in maniera permanente al buffer di ingresso <ibufio> e con uno dei registri <shiftreg> o <addreg>. Per questo motivo sarà necessario adottare un multiplexer che decide quali dati mandare in ingresso al registro. Questa è propriamente la funzione dell'elemento <muxcompreg>.

### Controller

L'elemento <controller> è il cuore di coordinazione di tutti i segnali che arrivano al dispositivo e ne regola in maniera precisa tutto il funzionamento. Il suo comportamento sarà discusso ampiamente in seguito.

### Converter

Il linguaggio VHDL consente al progettista, di assegnare ai segnali, diversi valori con significato analogo. Ad esempio per indicare uno stato logico "alto" si possono usare i valori 'H' oppure '1'.
Tutti i registri realizzati sono stati progettati utilizzando dei valori "standard", cioè '1' o '0'. L'elemento <converter> rappresenta una soluzione "software" per mantenere la compatibilità fra tutti gli altri componenti e i banchi di prova, creati precedentemente per effettuare i test di funzionamento.
Lo stesso tipo di problema si troverà nella riconversione dei dati da immettere sul buffer di uscita e verrà risolto con l'analogo elemento <deconverter>.
Il registro <converter>, collegato direttamente al buffer di ingresso <ibfuio>, si occupa quindi di trasformare i segnali provenienti dalle due linee scl e sda in segnali con un valore "standard", '0' od '1', in modo da permettere a tutti gli altri elementi che compongono il dispositivo di funzionare in maniera corretta.

### Counter

Il registro <counter> è un contatore ad otto bit ed è utilizzato per regolare la trasmissione e la ricezione di dati. Esso inoltre funge da indice per caricare o scaricare i bit nel registro <shiftreg>.

### Deconverter

Il registo <deconverter> riconverte i segnali del dispositivo con valori da trasmettere sulla linea. I dati da immettere in uscita provengono sempre dal registro <shiftreg>, quindi è collegato a quest'elemento da un lato, mentre dall'altro è collegato al multiplexer di uscita <muxobufio>.

### Extsignal (External Signal)

L'elemento <extsignal> si occupa di portare all'interno del dispositivo i segnali provenienti dall'esterno riguardanti la modalità di disposizione del dispositivo e la sua occupazione, quindi rispettivamente i piedini I2c_rw ed I2c_busy. Questi segnali sono utilizzati dal registro <controller>, e quindi andranno a collegarsi ad esso.

### Filerw (File Reader Writer)

Il registro <filerw> compie la lettura oppure la scrittura di dati, da o su file, secondo la modalità decisa sempre dal controller. Quando si esegue una trasmissione, è necessario leggere da file, mentre quando si riceve sarà necessario scrivere sul supporto fisico.
I byte sono immessi nel registro <shiftreg> o prelevati da quest'ultimo.
Ogni lettura o scrittura avviene sul fronte positivo del clock.
Inoltre il dispositivo utilizza delle funzioni interne per la conversione dei segnali in bit e viceversa.

### Ibufio(Input Buffer I/O)

Il buffer <ibufio> legge i valori presenti sulle linee del bus, fornendoli in ingresso al dispositivo. Questo componente e quello <obufio> sono entrambi realizzati, per via software, attraverso un unico elemento con due pin di ingresso, due pin di uscita e un bus per la scelta della modalità di funzionamento. Per mezzo di queste linee, si può scegliere quali pin di uscita mettere in alta impedenza.
Nel buffer <ibufio> i pin di ingresso sono collegati alle linee "scl" ed "sda", mentre quelli di uscita sono utilizzati all'interno del dispositivo. E' possibile scegliere quali uscite porre in alta impedenza attraverso il bus a due bit.

### Muxcompreg (MultipleXer Comparator Register)

Il multiplexer <muxcompreg> ha lo scopo di scegliere quali dati presentare in ingresso al registro <compreg>. E' possibile sceglierne la modalità di funzionamento e far transitare i segnali provenienti dal registro <addreg> o dallo <shiftreg>.

### Muxobufio (MultipleXer Output Buffer I/O)

Il multiplexer <muxobufio> è utilizzato per collegare al buffer di uscita i vari segnali.
In questo caso, il componente utilizzato anche per realizzare <ibufio>, viene collegato in maniera opposta alla situazione precedente, nel senso che i suoi ingressi vengono presi dall'interno del dispositivo, mentre le uscite vengono collegate al bus.
I segnali di uscita possono provenire dai registri <assgen>, <shiftreg> o <synclock>.
La scelta di quali segnali far transitare viene effettuata dal <controller>.

### Obufio (Output Buffer I/O)

Il componente, creato anche per realizzare <ibufio>, viene utilizzato in modo opposto, ossia gli ingressi sono i segnali provenienti dall'interno del dispositivo mentre le uscite vengono collegate al bus.
Il buffer <obufio> emette in uscita dal dispositivo i segnali provenienti dal multiplexer <muxobufio>. Anche per questo buffer è possibile scegliere quali linee porre in alta impedenza e quali copiare dall'ingresso.

### Shiftreg (Shift Register)

Il registro <shiftreg> ha lo scopo di memorizzare i dati che sono trasmessi o ricevuti dal dispositivo, prima di venire memorizzati o trasmessi.

Esso ha vari modi di funzionamento, i quali sono rispettivamente:

- carico di un byte
- scarico di un byte
- carico di un bit
- scarico di un bit

Il carico e lo scarico dei byte avviene attraverso il registro <filerw>, mentre il carico e lo scarico di bit in modo seriale sono effettuati attraverso il <converter> e il <deconverter> rispettivamente.
La modalità di funzionamento viene decisa dal controller in ogni fase della trasmissione o ricezione di dati.

### Synclock (Syncronizator Clock)

L'elemento <synclock> si occupa di sincronizzare i clock del trasmittente e del ricevente, secondo i modi previsti dal protocollo i2c. Esso riceve in ingresso il valore della linea "scl" e secondo questa comanda gli elementi interni del dispositivo dettandone la tempistica.

## *Funzionamento del <controller>*

Il registro <controller> svolge la fondamentale funzione di coordinare tutti i segnali all'interno del dispositivo. Esso deve essere capace di passare dallo stato di master a quello di slave o viceversa secondo i segnali esterni che capta sui piedini a sua disposizione. Inoltre entrambe le configurazioni possono trovarsi a trasmettere o a ricevere. Analizziamo quindi brevemente quali sono le funzioni di chi invia dati e di chi li memorizza. Per il primo esse saranno le seguenti:

1. Generazione del segnale di start
2. Trasmissione di un byte
3. Attesa dell'acknowledgement
4. Eventuale generazione della condizione di stop

Si nota che le operazioni 1. e 4. sono effettuate solo da un master e mai dallo slave.
Il ricevente invece eseguirà i seguenti comportamenti:

1. Rilevazione del segnale di start
2. Ricezione di un byte
3. Generazione dell'acknowledgement
4. Rilevazione della condizione di stop

## Configurazioni del dispositivo

Passiamo ora ad analizzare le varie configurazioni in cui si può disporre il dispositivo. Secondo lo stato nel quale si trova, le azioni compiute cambiano nettamente.

### Master

Il dispositivo passa allo stato master quando non rileva sulla linea un segnale di start ed inoltre il valore del pin I2c_rw non è in alta impedenza. Il tipo è scelto con il seguente criterio:

- '0': master-transmitter
- '1': master-receiver

Il master quando inizia la trasmissione dovrà, dopo essersi accertato che il bus è libero, generare innanzitutto il segnale di start. Una volta eseguite queste due operazioni dovrà caricare i dati, ed iniziare la trasmissione effettiva.
Il primo byte letto da file, corrisponde all'indirizzo di un altro dispositivo collegato al bus, con l'ultimo bit che indica allo slave in che stato disporsi.
La spedizione di bit sul bus procederà fino ad arrivare ad un byte. Durante questa prima trasmissione avviene anche la fase di arbitraggio. Nel caso sulla linea il valore sia differente da quello che si vuole imporre, il master dovrà passare allo stato di slave, eventualmente procedendo con la comparazione di quanto si trova sulla linea con il proprio indirizzo. Sul nono impulso di clock lascerà libera la linea in modo da permettere allo slave di generare il segnale di acknowledgement. Se questo segnale non fosse ricevuto, il master generà il segnale di stop e si disporrà nello stato di attesa.
Una volta ottenuta conferma della ricezione, le operazioni successive dipenderanno dallo stato in cui si trova il dispositivo.

### Master-transmitter

Nel caso il segnale proveniente dal morsetto I2c_rw sia allo stato "alto", la trasmissione proce in modo analogo a quanto visto nel primo passaggio. Sono quindi spediti otto bit e successivamente atteso il segnale di acknoweldgement. La procedura d'arbitraggio, se necessario, continuerà in questa fase, ma senza ricorrere all'indirizzo del dispositivo per confronto. Nel caso l'elemento perda questa procedura, si pone nello stato d'attesa.
La trasmissione continuerà finché il master avrà dati da spedire o chi riceve deciderà di non spedire il segnale di conferma. In questo caso verrà quindi generato il segnale di stop da parte del master.

### Master-receiver

La comunicazione fra i due dispositivi procede, anche in questo caso, con la trasmissione di dati da parte dello slave.

Il segnale “scl” continua ad essere generato da parte d’entrambi i dispositivi.
Il master si preparerà ad ottenere dei dati dal bus, memorizzandoli a gruppi di otto bit su file. In questo tipo di modalità vista l’impossibilità da parte del master di conoscere quanti dati debba trasmettere lo slave si suppone che quest’ultimo risponda ad una richiesta di questo tipo con un unico byte. Una volta ricevuto il byte il master genererà il segnale di acknowledgement e successivamente anche quello di stop. Dopo, esso si collocherà nello stato d’attesa.

Slave

Il dispositivo si dispone nello stato di slave nel qual caso veda sulla linea un segnale di start. Una volta predisposto in questa configurazione andrà a testare se i primi sette bit trasmessi corrispondono con il suo indirizzo. Se questo non accade ritorna nello stato di testing della linea. In caso d’esito positivo l’ottavo bit indica la modalità nella quale disporsi, con il seguente criterio:

- ‘0’: slave-receiver
- ‘1’: slave-transmitter

In entrambe le configurazioni egli genererà il segnale d’acknowledgement, a meno che non pervenga indicazione diversa da parte del segnale sul piedino I2c_busy. Nel qual caso quest’ultimo venga settato al valore “alto”, lo slave ritornerà nello stato di attesa. Se questo non accade la continuazione delle operazioni dipenderà dall’ultimo bit visto sulla linea sda.

Slave-transmitter

In questa configurazione è richiesto allo slave di trasmettere un dato, prelevato da un file. I bit sono spediti nello stesso modo in cui venivano effettuate le precedenti comuncazioni. Una volta trasmesso il byte egli ritornerà nello stato di attesa.

Slave-receiver

Quando il dispositivo si trova in questa configurazione riceve i bit dalla linea. Una volta arrivati all’ottavo bit dovrà generare il segnale di acknowledgement, sempre nel caso in cui non ci sia indicazione contraria proveniente dall’esterno attraverso il pin I2c_busy. Durante la ricezione, l’elemento deve anche porsi il problema di riconoscere un eventuale segnale di stop, testando in maniera continuata le due linee. In entrambi gli ultimi due casi citati, il dispositivo ritornerà nello stato d’attesa.

## Macchina a stati

La macchina a stati che descrive tutte le configurazioni si trova in Fig. 10.

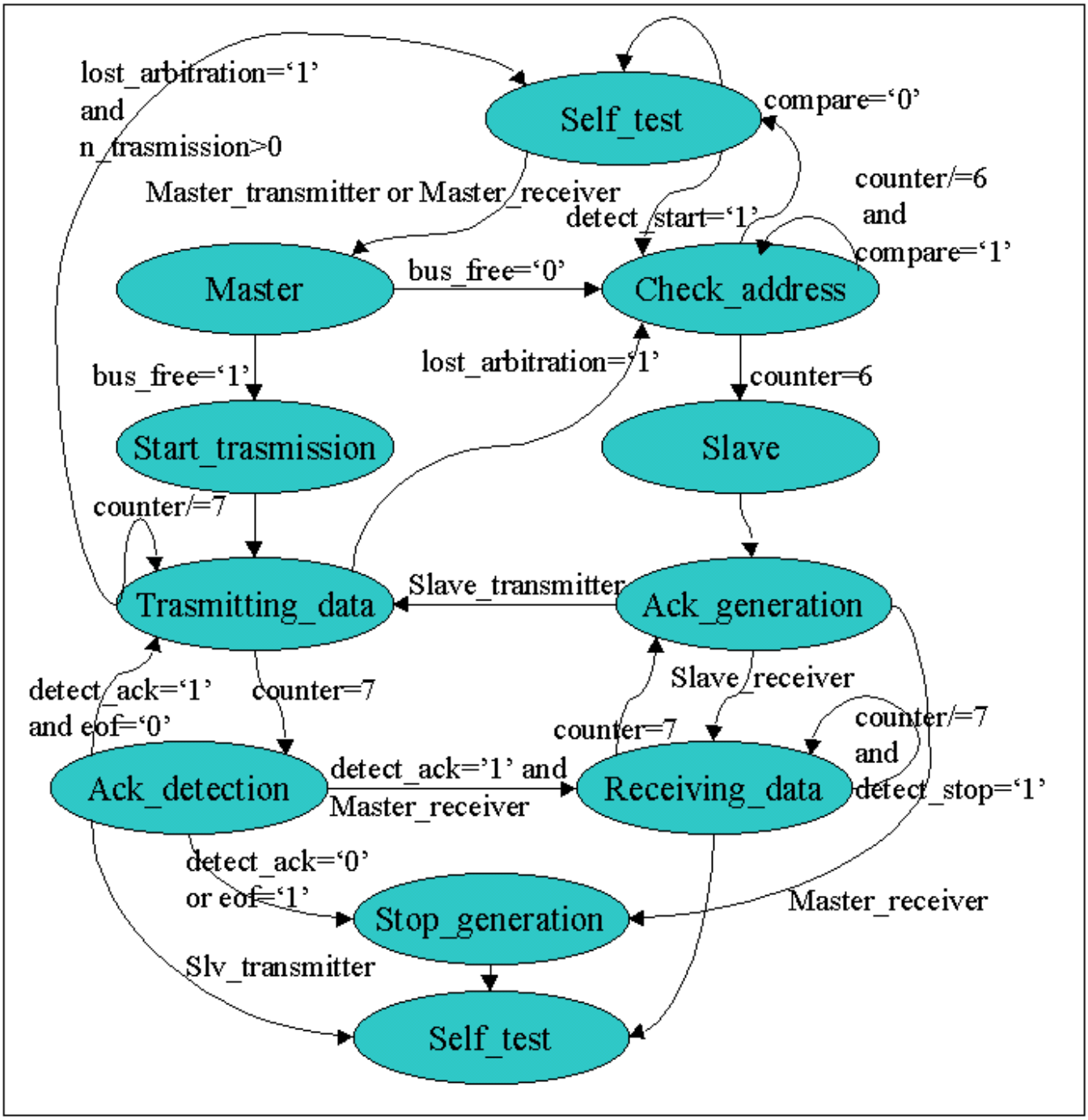



Fig. 10

Nella Fig. 10 lo stato Self_test è ripetuto due volte solo per comodità di rappresentazione. Si nota innanzitutto che la sua descrizione appare molto complessa. Questo è dovuto soprattutto al fatto che ci sono molti collegamenti fra gli stati.
Da notare inoltre, le condizioni di passaggio, riportate sui rami del diagramma, non indicano effettivamente i segnali attraverso i quali decidere il passaggio da uno stato logico ad un altro, ma consentono una descrizione più intuitiva del criterio di scelta.

La macchina realizzata è descritta utilizzando due stati: uno presente ed uno attuale, chiamati rispettivamente present_state e next_state.
Le azioni compiute nei due casi sono differenti. In particolare si nota che sono effettuate nello stato presente, tutte le operazioni che riguardano la trasmissione di segnali sulla linea, come ad esempio la generazione del segnale di start. Nello stato next_state sono invece effettuate tutte le commutazioni dei segnali in modo da predisporre il dispositivo per la prossima configurazione.
Il passaggio da uno stato presente ad un altro avviene durante la soglia positiva del clock del dispositivo.
La scelta del prossimo stato viene invece effettuata durante la soglia negativa.
Passiamo ora a descrivere i vari stati in cui si può predisporre il dispositivo e le azioni che dovranno effettuarsi. Prima di passare all'analisi dettagliata delle varie fasi di funzionamento bisogna precisare che in questa descrizione i registri non citati vengono ritenuti in alta impedenza e quindi non influenti.

*Ack_detection*
Nello stato ack_detection, il dispositivo si pone in fase d'attesa del segnale d'acknoweldgement. Bisognerà quindi predisporre il buffer d'uscita <obufio> in modo da lasciare la linea sda libera. Verrà inoltre attivato il rilevatore di segnali "particolari": <assdet> ed abilitato il buffer di ingresso <ibufio>.
Le altre azioni da compiere dipenderanno sostanzialmente dalla modalità in cui si è disposto il dispositivo.
Nel caso l'elemento si trovi nello stato di master-transmitter, si predisporrà a caricare nuovi dati nello <shiftreg> leggendoli tramite il registro <filerw>. Una volta caricati il nuovo byte da trasmettere verrà cambiato il segnale shift_state, il quale individua lo stato del registro shift.
Nel caso il dispositivo si trovi nello stato di master-receiver o di slave-transmitter verrà disabilitato il registro <shiftreg>.
Il prossimo stato da eseguire sarà quello di trasmissione dati se viene rilevato il segnale di acknoweldgement e durante la fase di lettura non ci sono state segnalazioni da parte del registro <filerw> che indichino la fine del file.
Nel caso una delle due condizioni sopra esposte non si verifichi il dispositivo, se è configurato come master, passerà allo stato di stop_generation. Se invece il dispositivo si trova nello stato di slave-transmitter si porrà nello stato self-test.
Se l'elemento si trova nello stato di master_receiver, ed abbia ricevuto l'acknowledgement, si porterà nello stato receiving_data.

*Ack_generation*
Questo stato rappresenta il simmetrico del precedente. Quando due elementi comunicano uno con l'altro devono trovarsi, sul nono impulso di clock, uno nello stato ack_generation e l'altro nello stato ack_detection.

In questo caso si provvede a generare il segnale di acknowledgement, abilitando opportunamente il generatore <assgen> e collegandolo al buffer di uscita <obufio> attraverso la commutazione del multiplexer <muxobufio>.
Nel caso il dispositivo si trovi nella configurazione di master-receiver o di slave-receiver, i dati contenuti nello <shiftreg> verranno memorizzati su file attraverso l'utilizzo di <filerw>, mentre lo stato del registro shift viene impostato a vuoto.
Se il dispositivo si trova invece nella configurazione di slave-transmitter viene invece caricato un byte da file seguendo il percorso inverso a quello appena enunciato.
In questo caso il prossimo stato sarà stop_generation se il dispositivo si trova nella configurazione di master-receiver. Nel caso invece funzioni da slave_transmitter il prossimo stato sarà trasmitting_data. Se si trova nella configurazione slave_receiver il prossimo stato sarà receiving_data.

*Check_address*
In questo stato il dispositivo confronta il proprio indirizzo con quello che si presenta sulla linea. Verrà quindi attivato il registro <addreg> e confrontati i bit in uscita da questo con quelli presenti sulla linea attraverso il comparatore <compreg>. Perché il secondo bit sia quello proveniente dalla linea scl, sarà necessario configurare correttamente il multiplexer <muxcompreg>.
Per permettere un corretto controllo verrà inoltre abilitato il contatore <counter>.
Nel caso si arrivi al settimo bit, senza incorrere in esito negativo da parte del <compreg>, il prossimo stato sarà quello di slave. In caso contrario si ritorna nello stato di self_test.

*Master*
Nello stato di slave il master si configura in una delle due modalità consentite, ossia o quella di trasmissione o quella di ricezione a seconda del valore percepito sul pin I2c_rw. Inoltre in questa fase il dispositivo testa la linea per controllare che sia libera. Se questo non accade va allo stato check_address. In caso invece non ci siano altri elementi che stanno iniziando una comunicazione si porterà nello stato start_trasmission.

*Receiving_data*
In questo stato il dispositivo si dispone per ricevere i dati presenti sulla linea. Verrà quindi abilitato il registro <shifreg> in modalità di carico dei bit dal buffer di ingresso <ibufio>. La corretta scrittura dei bit nel registro a scorrimento è effettuata tramite il contatore <counter>, il quale svolge la funzione d'indice.
Nel caso l'elemento sia configurato in modalità di master-receiver verrà anche disabilitato il registro <assdet> per la rilevazione dello stop, il quale viene attivato nel caso si trovi in configurazione slave-receiver.
La macchina a stati rimarrà in questa condizione fino a che non sono stati trasmessi tutti i bit, quindi fino a quando il contatore <counter>, debitamente abilitato, non arriverà ad otto. L'altra condizione richiesta per la permanenza in questo stato è che non venga rilevato il segnale di stop. Se questo accade la macchina si sposta nello

stato di self_test. Contrariamente se il contatore indica il valore massimo, si passerà allo stato di ack_generation.

*Self_test*
In questo stato il dispositivo non ha ancora assunto una configurazione specifica. Verrà abilitato il buffer di ingresso <ibufio> e il rilevatore del segnale di start, il registro <assdet>. Nel qual caso si rilevi questa condizione, il prossimo stato sarà quello di check_address. Contrariamente se non il pin I2c_rw rimane ad un valore non specifico, l'elemento rimarrà in questo stato. Se al morsetto in ingresso viene applicato un valore '1' o '0', il prossimo stato sarà quello master.

*Slave*
Questo stato serve solo per configurare in modo corretto il dispositivo in base all'ultimo bit della prima trasmissione. Al fine di decidere se lo slave è di tipo trasmitter o receiver, verrà usato il registro <compreg> al quale si pone in ingresso il valore proveniente dalla linea sda e il segnale '1' proveniente dal multiplexer <muxcompreg> opportunamente abilitato. Il prossimo stato della macchina sarà quello ack_generation.

*Start_trasmission*
In questa fase il dispositivo genera la condizione di start attraverso il registro <assgen> collegato al buffer d'uscita <obufio> attraverso il multiplexer <muxobufio> opportunamente settato. Soltanto se non ci sono informazioni non trasmesse, avviene il caricamento dei dati da file attraverso il registro <filerw>.
Il byte è memorizzato nello <shiftreg>. Il prossimo stato sarà trasmitting_data.

*Stop_generation*
Nello stato stop_generation, il master genera la condizione di start che pone fine alle comunicazioni con l'altro dispositivo. Essa è sempre un segnale creato dal master. In questo caso è abilitato il buffer d'uscita <obufio> e collegato attraverso il multiplexer <muxobufio> al registro <assgen> per la generazione del segnale di stop.

*Trasmitting_data*
In questa fase, vengono inviati sulla linea i bit contenuti nello <shiftreg>. Quest'ultimo è collegato al buffer d'uscita <obufio> attraverso il multiplexer <muxobufio> opportunamente settato. Il dispositivo rimane in questo stato finché il contatore <counter>, debitamente abilitato, non arriva al valore otto. Questo registro viene utilizzato come indice per lo <shiftreg>.
Nel caso in cui il dispositivo si trovi nella configurazione di master-transmitter avviene anche l'arbitraggio attraverso il registro <compreg> il quale riceve come segnali da paragonare quelli provenienti dalla linea sda e dal registro <addreg>. Per disporre quest'ultimo segnale in ingresso al comparatore, viene abilitato il multiplexer <muxcompreg> in modo opportuno. Nel qual caso il master perderà l'arbitraggio, il prossimo stato sarà quello di check_address.

L’arbitraggio è svolto in questo modo soltanto per il primo byte trasmesso, ma non per i successivi, per i quali si esegue il confronto fra quanto presente nel registro a scorrimento e quanto è visto sulla linea. Se il master perde l’arbitraggio, in questo caso si riporta il dispositivo nello stato self_test.
Nel caso in cui il dispositivo sia configurato come slave-transmitter, l’arbitraggio ha luogo in modo analogo a quanto avviene per il master dopo la prima trasmissione.

## *Risultati della Simulazione*

Sono ora presentati i risultati ottenuti con la simulazione.
I segnali considerati per questa simulazione sono quelli ritenuti più indicativi. Questi sono:

- le linee scl e sda
- i segnali di configurazione (rw e busy)
- la configurazione del dispositivo (config_device)
- lo stato presente (present_state)
- il prossimo stato (next_state)
- il valore del contatore (counter_value)
- il contenuto del registro shift (shiftreg_data)
- lo stato del registro a scorrimento (shift_state)
- il byte letto e quello scritto dal registro <filerw> (filerw_rbyte e filerw_wbyte)

Nelle Fig. 11-12-13 è presentata una comunicazione svolta fra un master-transmitter e uno slave-receiver. Per comodità lo scambio di dati è stato suddiviso in tre parti ognuna della durata di 500 nano secondi.
Nelle Fig. 14-15-16 è illustrata la comunicazione fra due dispositivi, uno in configurazione master-receiver, l’altro in slave-transmitter. Anche in queste figure viene suddivisa la comunicazione in tre parti ognuna della durata di 480 nano secondi.
La Fig. 17 illustra il riconoscimento di un indirizzo diverso da quello proprio del dispositivo. Siccome non corrisponde con il suo, torna nello stato di self_test.

Fig. 11

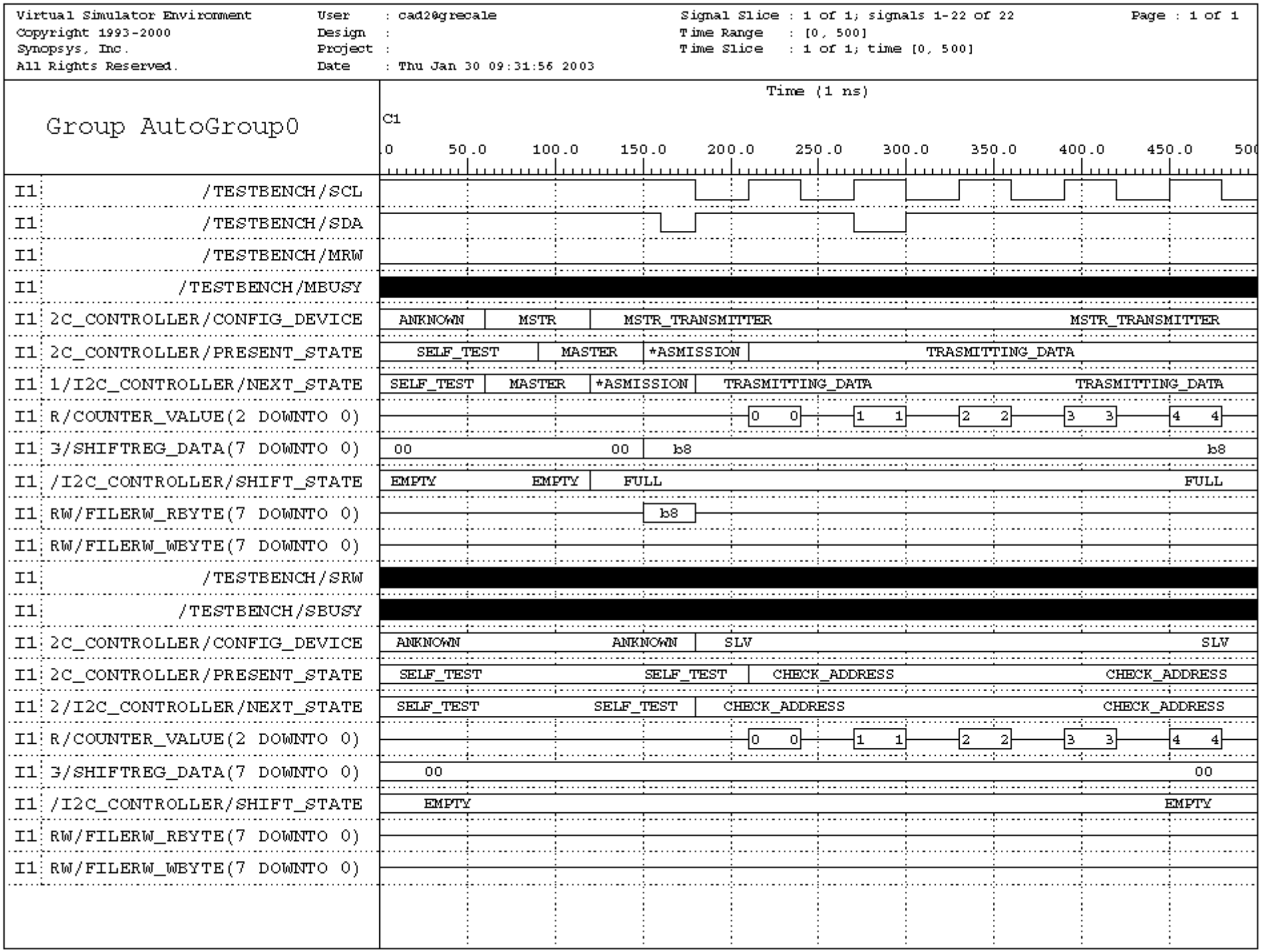

Virtual Simulator Environment
Copyright 1993-2000
Synopsys, Inc.
All Rights Reserved.
User : cad20grecale
Design :
Project :
Date : Thu Jan 30 09:31:56 2003
Signal Slice : 1 of 1; signals 1-22 of 22
Time Range : [0, 500]
Time Slice : 1 of 1; time [0, 500]
Page : 1 of 1
Group AutoGroup0
Time (1 ns)
C1
0
50.0
100.0
150.0
200.0
250.0
300.0
350.0
400.0
450.0
500
I1 /TESTBENCH/SCL
I1 /TESTBENCH/SDA
I1 /TESTBENCH/MRW
I1 /TESTBENCH/MBUSY
I1 2C_CONTROLLER/CONFIG_DEVICE
ANKNOWN
MSTR
MSTR_TRANSMITTER
MSTR_TRANSMITTER
I1 2C_CONTROLLER/PRESENT_STATE
SELF_TEST
MASTER
*ASMISSION
TRASMITTING_DATA
I1 1/I2C_CONTROLLER/NEXT_STATE
SELF_TEST
MASTER
*ASMISSION
TRASMITTING_DATA
TRASMITTING_DATA
I1 R/COUNTER_VALUE(2 DOWNTO 0)
0 0
1 1
2 2
3 3
4 4
I1 3/SHIFTREG_DATA(7 DOWNTO 0)
00
00
b8
b8
I1 /I2C_CONTROLLER/SHIFT_STATE
EMPTY
EMPTY
FULL
FULL
I1 RW/FILERW_RBYTE(7 DOWNTO 0)
b8
I1 RW/FILERW_WBYTE(7 DOWNTO 0)
I1 /TESTBENCH/SRW
I1 /TESTBENCH/SBUSY
I1 2C_CONTROLLER/CONFIG_DEVICE
ANKNOWN
ANKNOWN
SLV
SLV
I1 2C_CONTROLLER/PRESENT_STATE
SELF_TEST
SELF_TEST
CHECK_ADDRESS
CHECK_ADDRESS
I1 2/I2C_CONTROLLER/NEXT_STATE
SELF_TEST
SELF_TEST
CHECK_ADDRESS
CHECK_ADDRESS
I1 R/COUNTER_VALUE(2 DOWNTO 0)
0 0
1 1
2 2
3 3
4 4
I1 3/SHIFTREG_DATA(7 DOWNTO 0)
00
00
I1 /I2C_CONTROLLER/SHIFT_STATE
EMPTY
EMPTY
I1 RW/FILERW_RBYTE(7 DOWNTO 0)
I1 RW/FILERW_WBYTE(7 DOWNTO 0)

Fig. 12

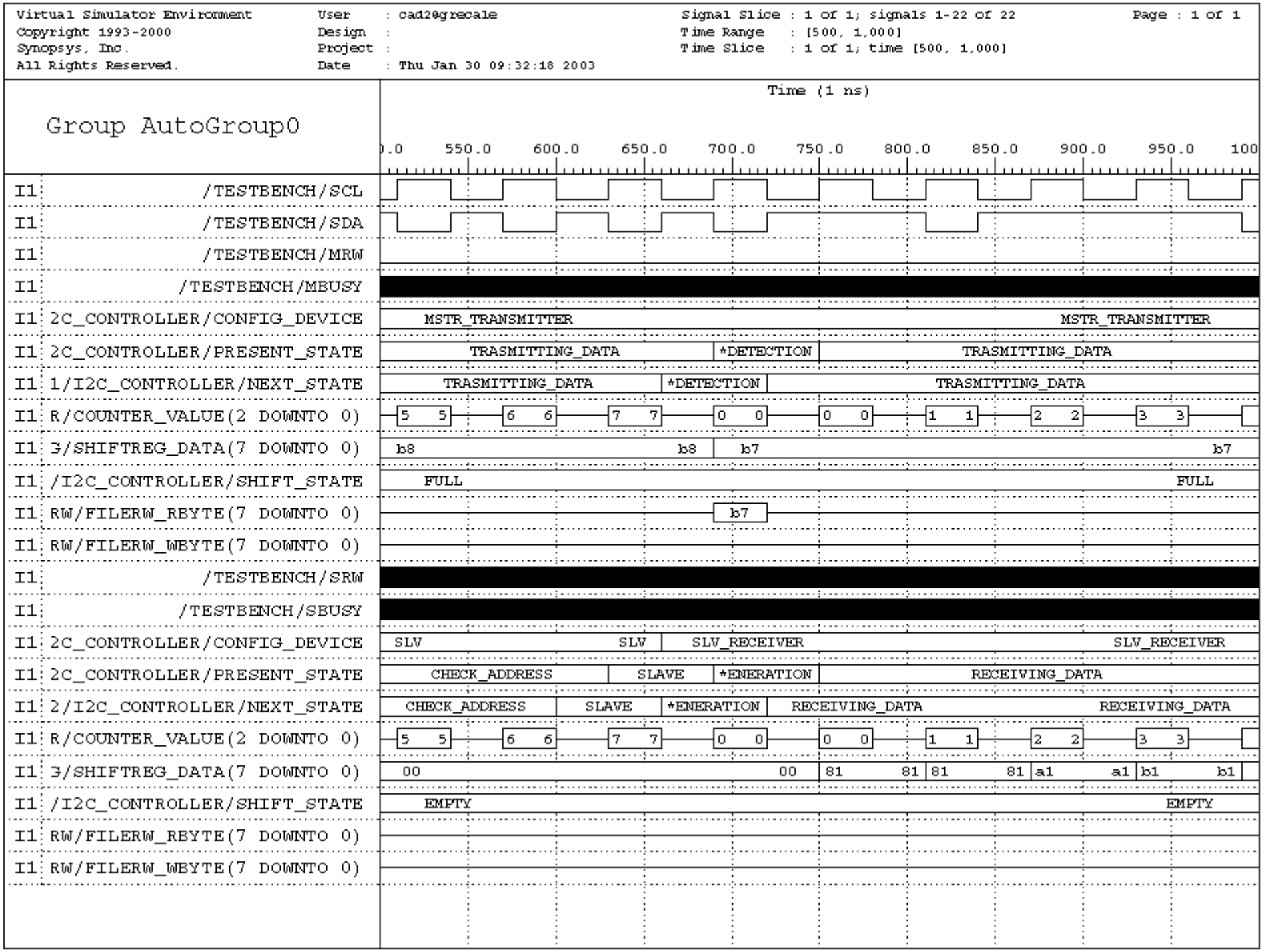

Virtual Simulator Environment
Copyright 1993-2000
Synopsys, Inc.
All Rights Reserved.
User : cad20grecale
Design :
Project :
Date : Thu Jan 30 09:32:18 2003
Signal Slice : 1 of 1; signals 1-22 of 22
Time Range : [500, 1,000]
Time Slice : 1 of 1; time [500, 1,000]
Page : 1 of 1
Group AutoGroup0
Time (1 ns)
550.0
600.0
650.0
700.0
750.0
800.0
850.0
900.0
950.0
I1 /TESTBENCH/SCL
I1 /TESTBENCH/SDA
I1 /TESTBENCH/MRW
I1 /TESTBENCH/MBUSY
I1 2C_CONTROLLER/CONFIG_DEVICE
MSTR_TRANSMITTER
MSTR_TRANSMITTER
I1 2C_CONTROLLER/PRESENT_STATE
TRASMITTING_DATA
*DETECTION
TRASMITTING_DATA
I1 1/I2C_CONTROLLER/NEXT_STATE
TRASMITTING_DATA
*DETECTION
TRASMITTING_DATA
I1 R/COUNTER_VALUE(2 DOWNTO 0)
5 5
6 6
7 7
0 0
0 0
1 1
2 2
3 3
I1 3/SHIFTREG_DATA(7 DOWNTO 0)
b8
b8
b7
b7
I1 /I2C_CONTROLLER/SHIFT_STATE
FULL
FULL
I1 RW/FILERW_RBYTE(7 DOWNTO 0)
b7
I1 RW/FILERW_WBYTE(7 DOWNTO 0)
I1 /TESTBENCH/SRW
I1 /TESTBENCH/SBUSY
I1 2C_CONTROLLER/CONFIG_DEVICE
SLV
SLV
SLV_RECEIVER
SLV_RECEIVER
I1 2C_CONTROLLER/PRESENT_STATE
CHECK_ADDRESS
SLAVE
*ENERATION
RECEIVING_DATA
I1 2/I2C_CONTROLLER/NEXT_STATE
CHECK_ADDRESS
SLAVE
*ENERATION
RECEIVING_DATA
RECEIVING_DATA
I1 R/COUNTER_VALUE(2 DOWNTO 0)
5 5
6 6
7 7
0 0
0 0
1 1
2 2
3 3
I1 3/SHIFTREG_DATA(7 DOWNTO 0)
00
00
81
81
81
81
a1
a1
b1
b1
I1 /I2C_CONTROLLER/SHIFT_STATE
EMPTY
EMPTY
I1 RW/FILERW_RBYTE(7 DOWNTO 0)
I1 RW/FILERW_WBYTE(7 DOWNTO 0)

Fig 13

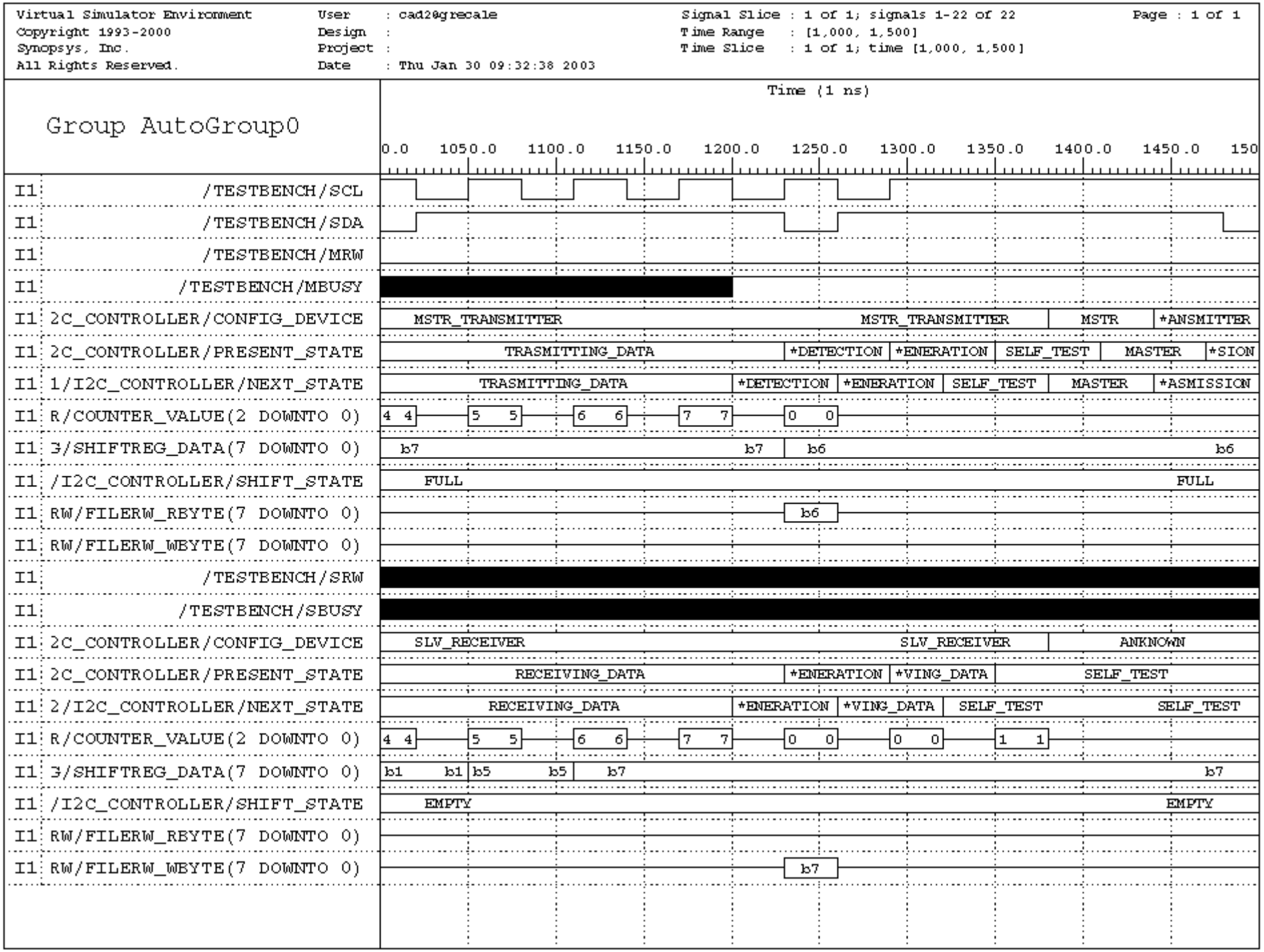

Virtual Simulator Environment
Copyright 1993-2000
Synopsys, Inc.
All Rights Reserved.
User : cad20grecale
Design :
Project :
Date : Thu Jan 30 09:32:38 2003
Signal Slice : 1 of 1; signals 1-22 of 22
Time Range : [1,000, 1,500]
Time Slice : 1 of 1; time [1,000, 1,500]
Page : 1 of 1
Group AutoGroup0
Time (1 ns)
0.0 1050.0 1100.0 1150.0 1200.0 1250.0 1300.0 1350.0 1400.0 1450.0 150
I1 /TESTBENCH/SCL
I1 /TESTBENCH/SDA
I1 /TESTBENCH/MRW
I1 /TESTBENCH/MBUSY
I1 2C_CONTROLLER/CONFIG_DEVICE
MSTR_TRANSMITTER
MSTR_TRANSMITTER
MSTR
*ANSMITTER
I1 2C_CONTROLLER/PRESENT_STATE
TRASMITTING_DATA
*DETECTION
*ENERATION
SELF_TEST
MASTER
*SION
I1 1/I2C_CONTROLLER/NEXT_STATE
TRASMITTING_DATA
*DETECTION
*ENERATION
SELF_TEST
MASTER
*ASMISSION
I1 R/COUNTER_VALUE(2 DOWNTO 0)
4 4
5 5
6 6
7 7
0 0
I1 3/SHIFTREG_DATA(7 DOWNTO 0)
b7
b7
b6
b6
I1 /I2C_CONTROLLER/SHIFT_STATE
FULL
FULL
I1 RW/FILERW_RBYTE(7 DOWNTO 0)
b6
I1 RW/FILERW_WBYTE(7 DOWNTO 0)
I1 /TESTBENCH/SRW
I1 /TESTBENCH/SBUSY
I1 2C_CONTROLLER/CONFIG_DEVICE
SLV_RECEIVER
SLV_RECEIVER
ANKNOWN
I1 2C_CONTROLLER/PRESENT_STATE
RECEIVING_DATA
*ENERATION
*VING_DATA
SELF_TEST
I1 2/I2C_CONTROLLER/NEXT_STATE
RECEIVING_DATA
*ENERATION
*VING_DATA
SELF_TEST
SELF_TEST
I1 R/COUNTER_VALUE(2 DOWNTO 0)
4 4
5 5
6 6
7 7
0 0
0 0
1 1
I1 3/SHIFTREG_DATA(7 DOWNTO 0)
b1
b1
b5
b5
b7
b7
I1 /I2C_CONTROLLER/SHIFT_STATE
EMPTY
EMPTY
I1 RW/FILERW_RBYTE(7 DOWNTO 0)
I1 RW/FILERW_WBYTE(7 DOWNTO 0)
b7

Fig. 14

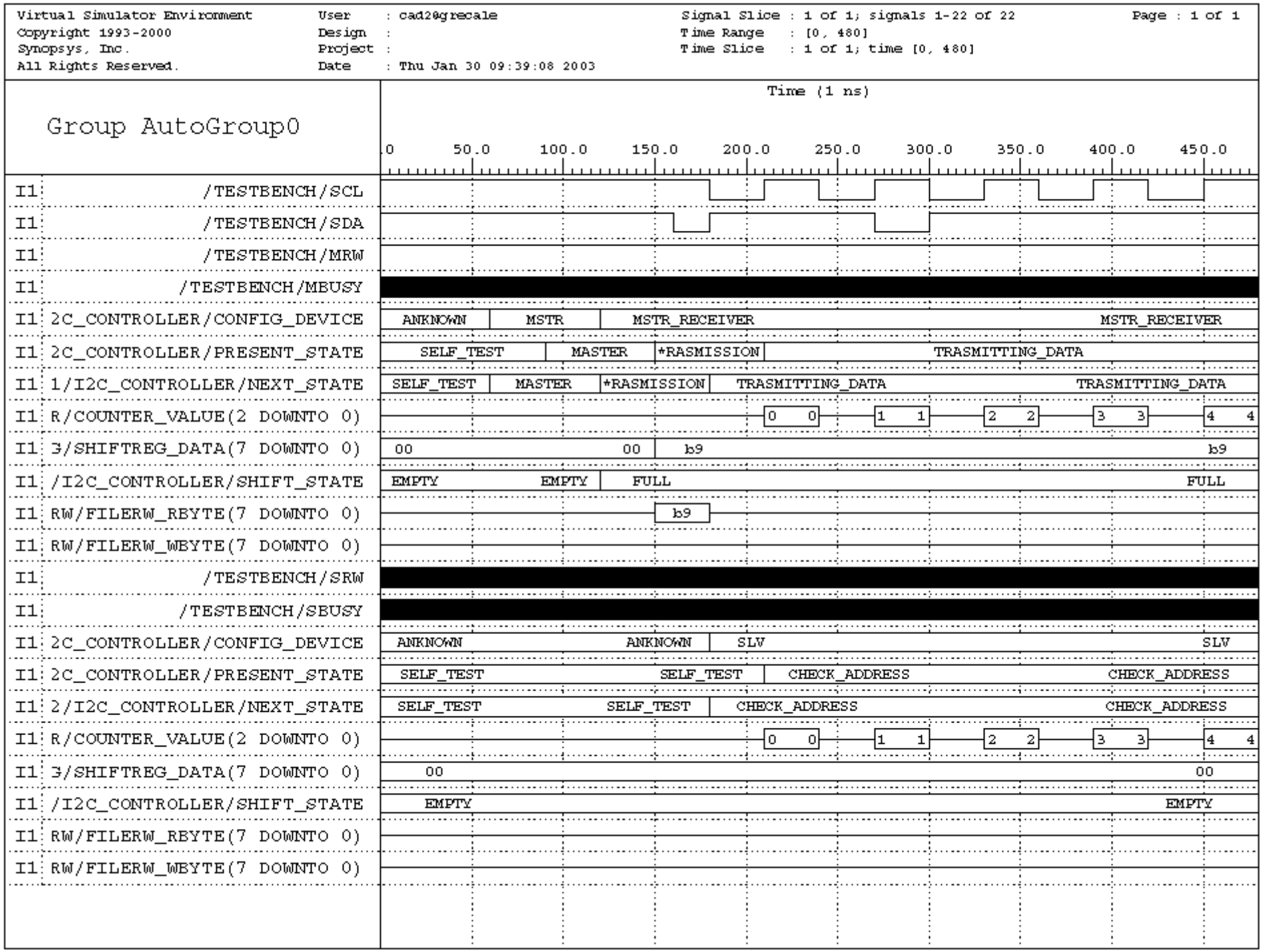

Virtual Simulator Environment
Copyright 1993-2000
Synopsys, Inc.
All Rights Reserved.
User : cad20grecale
Design :
Project :
Date : Thu Jan 30 09:39:08 2003
Signal Slice : 1 of 1; signals 1-22 of 22
Time Range : [0, 480]
Time Slice : 1 of 1; time [0, 480]
Page : 1 of 1
Group AutoGroup0
Time (1 ns)
0
50.0
100.0
150.0
200.0
250.0
300.0
350.0
400.0
450.0
I1 /TESTBENCH/SCL
I1 /TESTBENCH/SDA
I1 /TESTBENCH/MRW
I1 /TESTBENCH/MBUSY
I1 2C_CONTROLLER/CONFIG_DEVICE
ANKNOWN
MSTR
MSTR_RECEIVER
MSTR_RECEIVER
I1 2C_CONTROLLER/PRESENT_STATE
SELF_TEST
MASTER
*RASMISSION
TRASMITTING_DATA
I1 1/I2C_CONTROLLER/NEXT_STATE
SELF_TEST
MASTER
*RASMISSION
TRASMITTING_DATA
TRASMITTING_DATA
I1 R/COUNTER_VALUE(2 DOWNTO 0)
0 0
1 1
2 2
3 3
4 4
I1 3/SHIFTREG_DATA(7 DOWNTO 0)
00
00
b9
b9
I1 /I2C_CONTROLLER/SHIFT_STATE
EMPTY
EMPTY
FULL
FULL
I1 RW/FILERW_RBYTE(7 DOWNTO 0)
b9
I1 RW/FILERW_WBYTE(7 DOWNTO 0)
I1 /TESTBENCH/SRW
I1 /TESTBENCH/SBUSY
I1 2C_CONTROLLER/CONFIG_DEVICE
ANKNOWN
ANKNOWN
SLV
SLV
I1 2C_CONTROLLER/PRESENT_STATE
SELF_TEST
SELF_TEST
CHECK_ADDRESS
CHECK_ADDRESS
I1 2/I2C_CONTROLLER/NEXT_STATE
SELF_TEST
SELF_TEST
CHECK_ADDRESS
CHECK_ADDRESS
I1 R/COUNTER_VALUE(2 DOWNTO 0)
0 0
1 1
2 2
3 3
4 4
I1 3/SHIFTREG_DATA(7 DOWNTO 0)
00
00
I1 /I2C_CONTROLLER/SHIFT_STATE
EMPTY
EMPTY
I1 RW/FILERW_RBYTE(7 DOWNTO 0)
I1 RW/FILERW_WBYTE(7 DOWNTO 0)

Fig.15

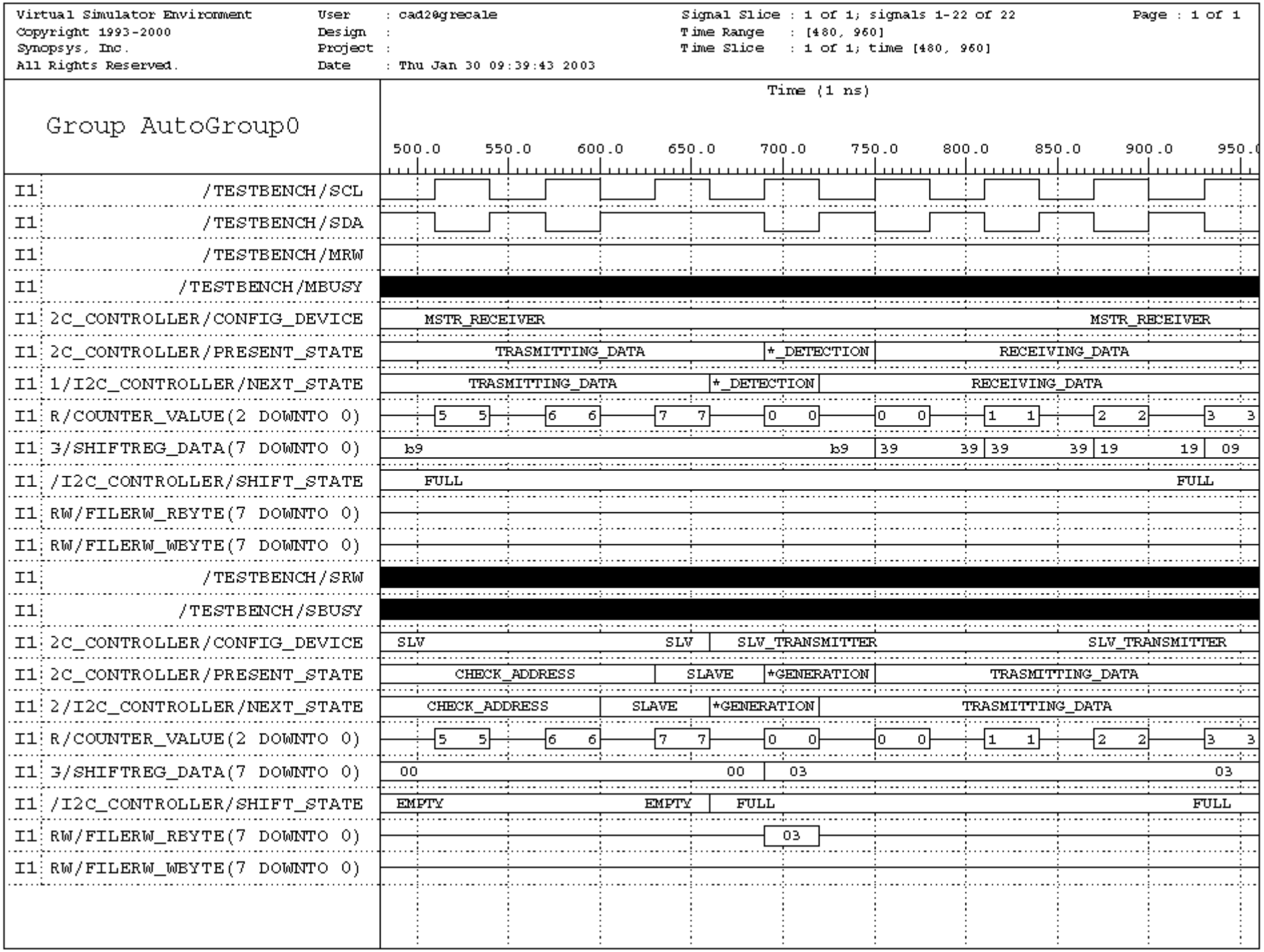

Virtual Simulator Environment
Copyright 1993-2000
Synopsys, Inc.
All Rights Reserved.
User : cad20grecale
Design :
Project :
Date : Thu Jan 30 09:39:43 2003
Signal Slice : 1 of 1; signals 1-22 of 22
Time Range : [480, 960]
Time Slice : 1 of 1; time [480, 960]
Page : 1 of 1
Group AutoGroup0
Time (1 ns)
500.0
550.0
600.0
650.0
700.0
750.0
800.0
850.0
900.0
950.0
I1 /TESTBENCH/SCL
I1 /TESTBENCH/SDA
I1 /TESTBENCH/MRW
I1 /TESTBENCH/MBUSY
I1 2C_CONTROLLER/CONFIG_DEVICE
MSTR_RECEIVER
MSTR_RECEIVER
I1 2C_CONTROLLER/PRESENT_STATE
TRASMITTING_DATA
*_DETECTION
RECEIVING_DATA
I1 1/I2C_CONTROLLER/NEXT_STATE
TRASMITTING_DATA
*_DETECTION
RECEIVING_DATA
I1 R/COUNTER_VALUE(2 DOWNTO 0)
5 5
6 6
7 7
0 0
0 0
1 1
2 2
3 3
I1 3/SHIFTREG_DATA(7 DOWNTO 0)
b9
b9
39
39
39
39
19
19
09
I1 /I2C_CONTROLLER/SHIFT_STATE
FULL
FULL
I1 RW/FILERW_RBYTE(7 DOWNTO 0)
I1 RW/FILERW_WBYTE(7 DOWNTO 0)
I1 /TESTBENCH/SRW
I1 /TESTBENCH/SBUSY
I1 2C_CONTROLLER/CONFIG_DEVICE
SLV
SLV
SLV_TRANSMITTER
SLV_TRANSMITTER
I1 2C_CONTROLLER/PRESENT_STATE
CHECK_ADDRESS
SLAVE
*GENERATION
TRASMITTING_DATA
I1 2/I2C_CONTROLLER/NEXT_STATE
CHECK_ADDRESS
SLAVE
*GENERATION
TRASMITTING_DATA
I1 R/COUNTER_VALUE(2 DOWNTO 0)
5 5
6 6
7 7
0 0
0 0
1 1
2 2
3 3
I1 3/SHIFTREG_DATA(7 DOWNTO 0)
00
00
03
03
I1 /I2C_CONTROLLER/SHIFT_STATE
EMPTY
EMPTY
FULL
FULL
I1 RW/FILERW_RBYTE(7 DOWNTO 0)
03
I1 RW/FILERW_WBYTE(7 DOWNTO 0)

Fig. 16

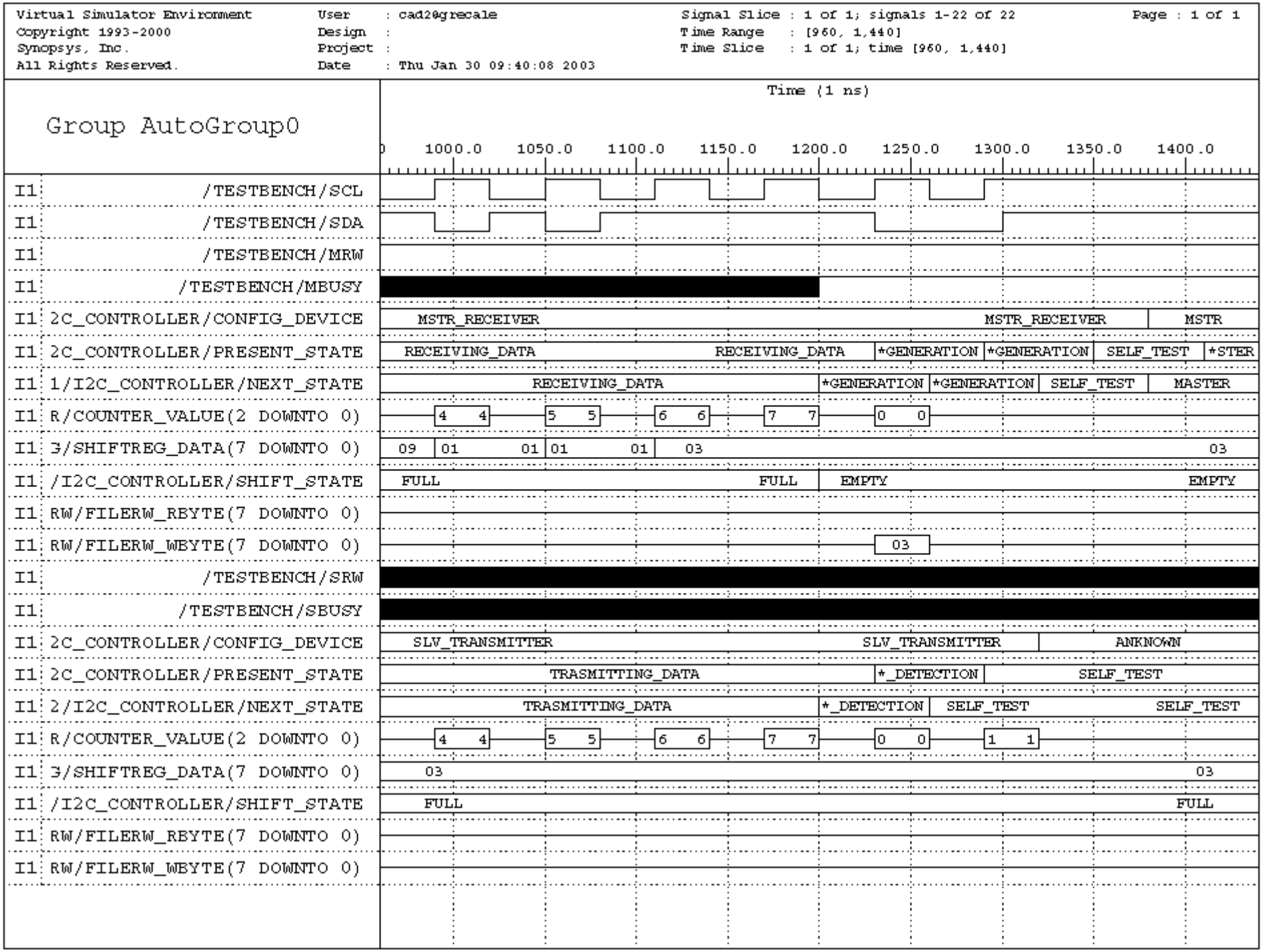
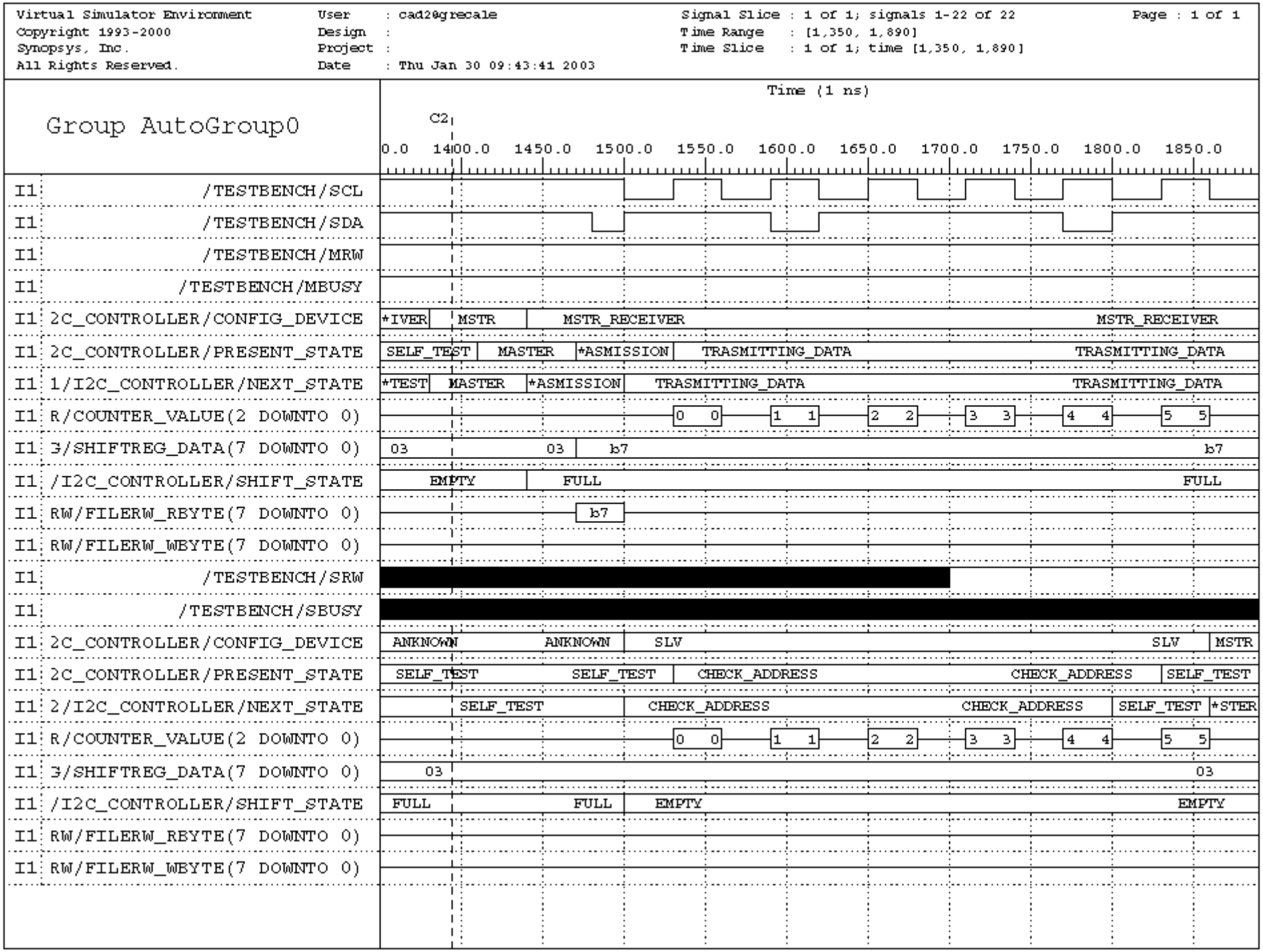

Virtual Simulator Environment
Copyright 1993-2000
Synopsys, Inc.
All Rights Reserved.
User : cad20grecale
Design :
Project :
Date : Thu Jan 30 09:40:08 2003
Signal Slice : 1 of 1; signals 1-22 of 22
Time Range : [960, 1,440]
Time Slice : 1 of 1; time [960, 1,440]
Page : 1 of 1
Group AutoGroup0
Time (1 ns)
1000.0
1050.0
1100.0
1150.0
1200.0
1250.0
1300.0
1350.0
1400.0
I1 /TESTBENCH/SCL
I1 /TESTBENCH/SDA
I1 /TESTBENCH/MRW
I1 /TESTBENCH/MBUSY
I1 2C_CONTROLLER/CONFIG_DEVICE
MSTR_RECEIVER
MSTR_RECEIVER
MSTR
I1 2C_CONTROLLER/PRESENT_STATE
RECEIVING_DATA
RECEIVING_DATA
*GENERATION
*GENERATION
SELF_TEST
*STER
I1 1/I2C_CONTROLLER/NEXT_STATE
RECEIVING_DATA
*GENERATION
*GENERATION
SELF_TEST
MASTER
I1 R/COUNTER_VALUE(2 DOWNTO 0)
4 4
5 5
6 6
7 7
0 0
I1 3/SHIFTREG_DATA(7 DOWNTO 0)
09
01
01
01
01
03
03
I1 /I2C_CONTROLLER/SHIFT_STATE
FULL
FULL
EMPTY
EMPTY
I1 RW/FILERW_RBYTE(7 DOWNTO 0)
I1 RW/FILERW_WBYTE(7 DOWNTO 0)
03
I1 /TESTBENCH/SRW
I1 /TESTBENCH/SBUSY
I1 2C_CONTROLLER/CONFIG_DEVICE
SLV_TRANSMITTER
SLV_TRANSMITTER
ANKNOWN
I1 2C_CONTROLLER/PRESENT_STATE
TRASMITTING_DATA
*_DETECTION
SELF_TEST
I1 2/I2C_CONTROLLER/NEXT_STATE
TRASMITTING_DATA
*_DETECTION
SELF_TEST
SELF_TEST
I1 R/COUNTER_VALUE(2 DOWNTO 0)
4 4
5 5
6 6
7 7
0 0
1 1
I1 3/SHIFTREG_DATA(7 DOWNTO 0)
03
03
I1 /I2C_CONTROLLER/SHIFT_STATE
FULL
FULL
I1 RW/FILERW_RBYTE(7 DOWNTO 0)
I1 RW/FILERW_WBYTE(7 DOWNTO 0)

Fig. 17


Virtual Simulator Environment
Copyright 1993-2000
Synopsys, Inc.
All Rights Reserved.
User : cad20grecale
Design :
Project :
Date : Thu Jan 30 09:43:41 2003
Signal Slice : 1 of 1; signals 1-22 of 22
Time Range : [1,350, 1,890]
Time Slice : 1 of 1; time [1,350, 1,890]
Page : 1 of 1
Group AutoGroup0
Time (1 ns)
C2
0.0
1400.0
1450.0
1500.0
1550.0
1600.0
1650.0
1700.0
1750.0
1800.0
1850.0
I1 /TESTBENCH/SCL
I1 /TESTBENCH/SDA
I1 /TESTBENCH/MRW
I1 /TESTBENCH/MBUSY
I1 2C_CONTROLLER/CONFIG_DEVICE
*IVER
MSTR
MSTR_RECEIVER
MSTR_RECEIVER
I1 2C_CONTROLLER/PRESENT_STATE
SELF_TEST
MASTER
*ASMISSION
TRASMITTING_DATA
TRASMITTING_DATA
I1 1/I2C_CONTROLLER/NEXT_STATE
*TEST
MASTER
*ASMISSION
TRASMITTING_DATA
TRASMITTING_DATA
I1 R/COUNTER_VALUE(2 DOWNTO 0)
0 0
1 1
2 2
3 3
4 4
5 5
I1 3/SHIFTREG_DATA(7 DOWNTO 0)
03
03
b7
b7
I1 /I2C_CONTROLLER/SHIFT_STATE
EMPTY
FULL
FULL
I1 RW/FILERW_RBYTE(7 DOWNTO 0)
b7
I1 RW/FILERW_WBYTE(7 DOWNTO 0)
I1 /TESTBENCH/SRW
I1 /TESTBENCH/SBUSY
I1 2C_CONTROLLER/CONFIG_DEVICE
ANKNOWN
ANKNOWN
SLV
SLV
MSTR
I1 2C_CONTROLLER/PRESENT_STATE
SELF_TEST
SELF_TEST
CHECK_ADDRESS
CHECK_ADDRESS
SELF_TEST
I1 2/I2C_CONTROLLER/NEXT_STATE
SELF_TEST
CHECK_ADDRESS
CHECK_ADDRESS
SELF_TEST
*STER
I1 R/COUNTER_VALUE(2 DOWNTO 0)
0 0
1 1
2 2
3 3
4 4
5 5
I1 3/SHIFTREG_DATA(7 DOWNTO 0)
03
03
I1 /I2C_CONTROLLER/SHIFT_STATE
FULL
FULL
EMPTY
EMPTY
I1 RW/FILERW_RBYTE(7 DOWNTO 0)
I1 RW/FILERW_WBYTE(7 DOWNTO 0)

## *Conclusioni*

Analizzando i grafici ottenuti dopo la simulazione si vede che il dispositivo descritto funziona correttamente e rispetta le specifiche del protocollo i2c. Inoltre è possibile farlo funzionare sia da master che da slave. Esso riconosce inoltre quello che è l'indirizzo che lo contraddistingue e lo differenzia dagli altri elementi presenti sulla linea.

## *Esecuzione di una simulazione*

Per eseguire una simulazione del funzionamento del protocollo i2c e' necessario innanzitutto dotarsi dei file sorgenti. Conviene quindi creare una cartella "i2c" nella quale inserire i seguenti file:

addreg.vhd
assdet.vhd
assgen.vhd
bufio.vhd
compreg.vhd
controller.vhd
converter.vhd
counter.vhd
deconverte.vhd
extsignal.vhd
filerw.vhd
i2c.vhd
muxcompreg.vhd
muxobufio.vhd
pullup.vhd
shiftreg.vhd
synclk.vhd
testbench.vhd

Una volta eseguito questo bisogna creare una sottocartella "dati" nella quale si inseriscono i file sui quali il dispositivo legge e scrive. Questi saranno:

mifile.txt
mofile.txt
sifile.txt
sofile.txt

La sotto directory che si consiglia di creare non e' necessaria, ma serve semplicemente per tenere separati i dati da elaborare dai file sorgenti. Bisognerà modificare le impostazioni presenti n sia corretto.

Bisogna poi lanciare dal prompt il comando “scirocco”. Non occorrono altre specifiche particolari. All'apertura del programma comparira' una videata chiamata "VirSim" in cui si trovano vari pulsanti i quali attivano i sottoprogrammi a disposizione del tool come si vede in Fig. 18.

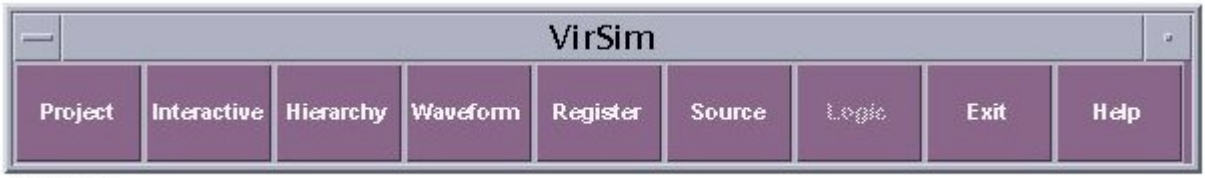



Fig. 18

Si preme quindi con il tasto destro del mouse sul pulsante "Project". Una volta eseguita quest’operazione compare la finestra grafica visualizzata in Fig. 19.

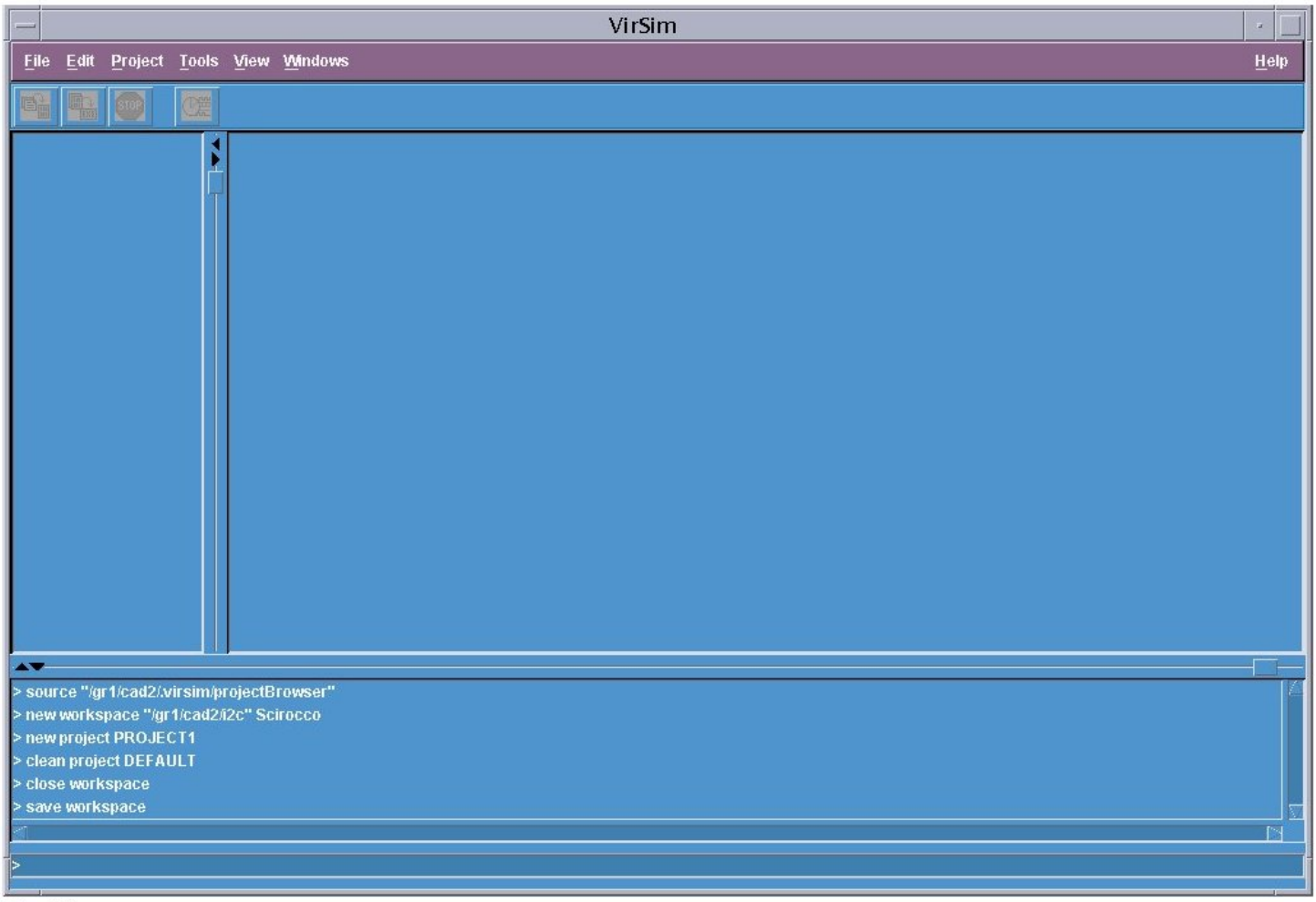



Fig. 19

Dalla figura si vede che è presente un menù a tendina e una serie di pulsanti i quali consentono di semplificare il richiamo di comandi particolari. Descrivendo i pulsanti da sinistra a destra, sono questi:

- Analyze active project
- Elaborate active project
- Stop
- Simulate design top

Oltre a questi è presente una zona che indica il “progresso delle operazioni”, la quale viene attivata durante le fasi di analisi e elaborazione.
Dal menu "File" selezioniamo la voce "New" e poi "Workspace". Comparira', quindi, una finestra di dialogo attraverso la quale si creerà un nuovo spazio di lavoro con il nome scelto. In questo caso si consiglia di chiamarlo "i2c" in modo da non creare ulteriori sottodirectory. La situazione è rappresentata nella Fig. 20.

Fig. 20

Dopo aver creato uno spazio di lavoro e' necessario aggiungere a questo i file sorgenti necessari per l'esecuzione della simulazione. Per fare questo sono possibili due vie:

1. Selezionare dal menu a tendina la voce "Project" e poi "Add file ...".
2. Spostarsi sulla parte a sinistra della finestra grafica e selezionare la voce "Source File" con il tasto sinistro. Comparira' quindi la voce "Add File" che dovra' venire selezionata.

In entrambi i casi sarà creata una finestra di dialogo che consente di inserire i file necessari per la simulazione, rappresentata in Fig. 21.

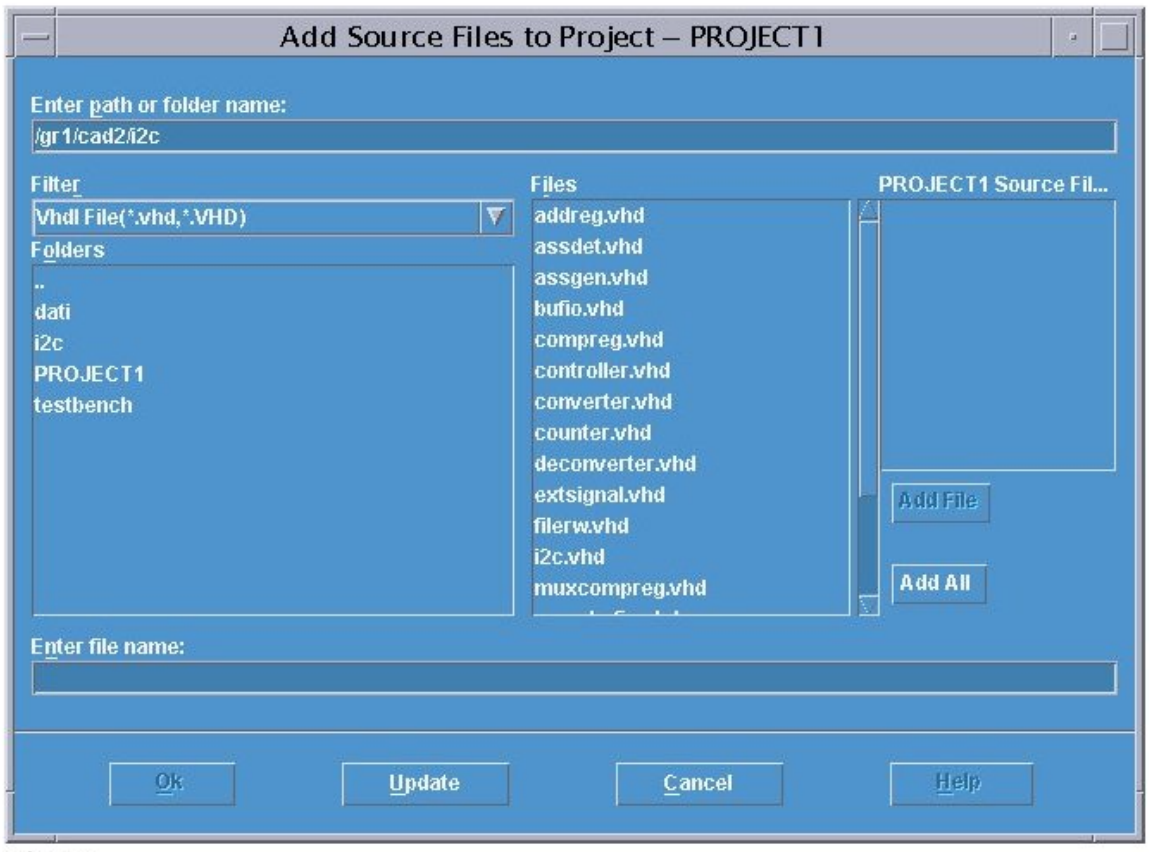


Fig. 21

I file dovranno essere selezionati da parte dell'utente e poi aggiunti. Si puo' premere il tasto "Add all". La situazione a cui si arriva è quella in Fig. 22.

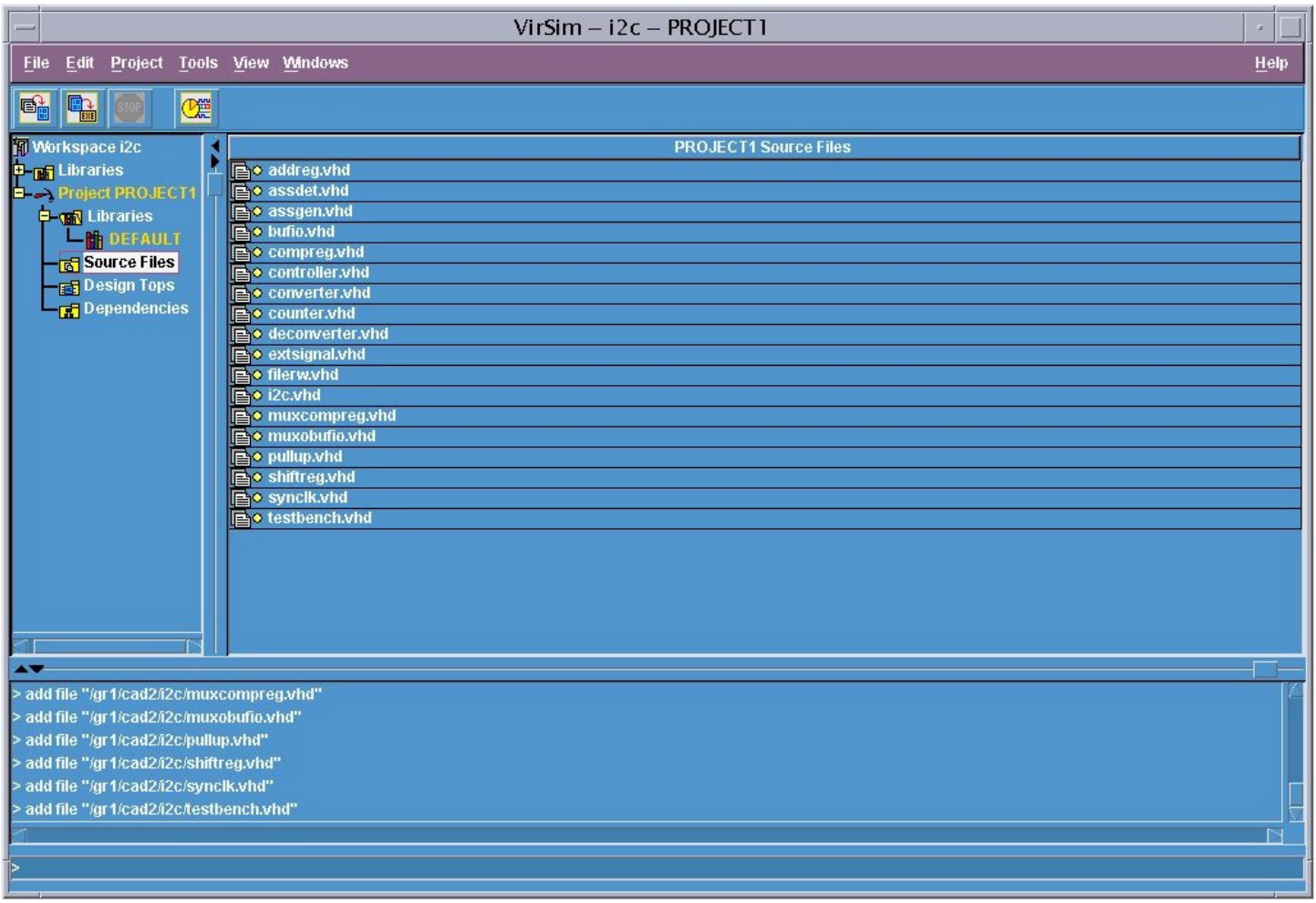


Fig. 22

Accanto ad ogni archivio è presente un indicatore del suo stato. Il suo significato dipende dal colore:

- bianco : file non analizzato
- verde : file analizzato, senza errori riscontrati
- rosso : file analizzato con errori riscontrati

Dopo aver inserito gli archivi sorgenti sarà necessario analizzarli.
Per fare questo le scelte disponibili sono:

1. Scegliere dal menù a tendina il sottomenù "Tool" e selezionare la voce "Analyze All".
2. Premere sull'icona che ha come commento la scritta "Analyze active project".

Quando viene eseguita questa operazione sulla finestra del progetto scorre una specie di cursore fino a quando l'analisi non finisce. Ad analisi terminata gli indicatori dello stato dei file diventano verdi.
Una volta analizzati i file bisogna elaborarli.
Prima di tutto pero' bisogna indicare al simulatore quale e l'entita "dominante". Nel nostro caso si tratta di quella chiamata come "TESTBENCH". Per selezionarla come top bisogna andare nella sottodirectory del "project PROJECT1" chiamata "Libraries" e ancora nel sotto ramo "DEFAULT". Cliccando con il tasto destro del mouse

comparira' una serie di entita. Fra queste bisogna selezionare quella chiamata "TESTBENCH" con il pulsante destro del mouse. Comparira' un sottomenù nel quale si andra' a selezionare la sottovoce "Add Top". Si giungerà quindi alla situazione illustrata nella Fig. 23.

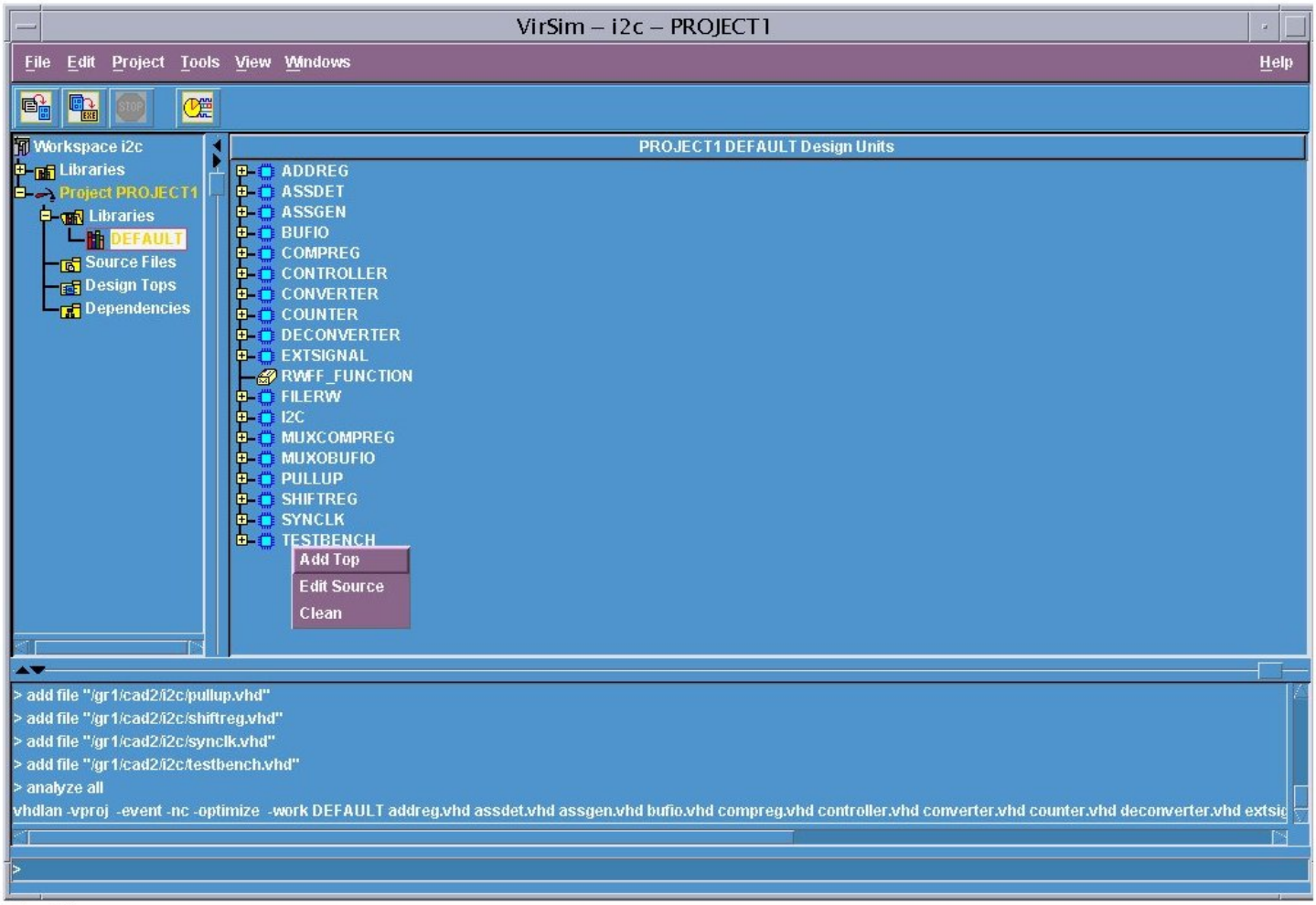


Fig. 23

Il passo successivo è quello della elaborazione.
Per eseguire questa fase le opzioni sono:

1. Scegliere dal menù a tendina la voce "Tool" e la sottovoce "Elaborate".
2. Premere con il tasto sinistro del mouse sull'icona indicante il commento "Elaborate active project".

Finita l'elaborazione si può procedere con la fase di simulazione. Per lanciare il simulatore sono disponibili due vie:

1. Selezionare dal menù a tendina la voce "Tool" e la sottovoce "Simulate".
2. Premere l'icona con indicazione "Simulate design top".

Comparirà quindi una finestra con il titolo "VirSim-Interactive-SIM-I1-Scirocco" come quella visibile in Fig. 24.

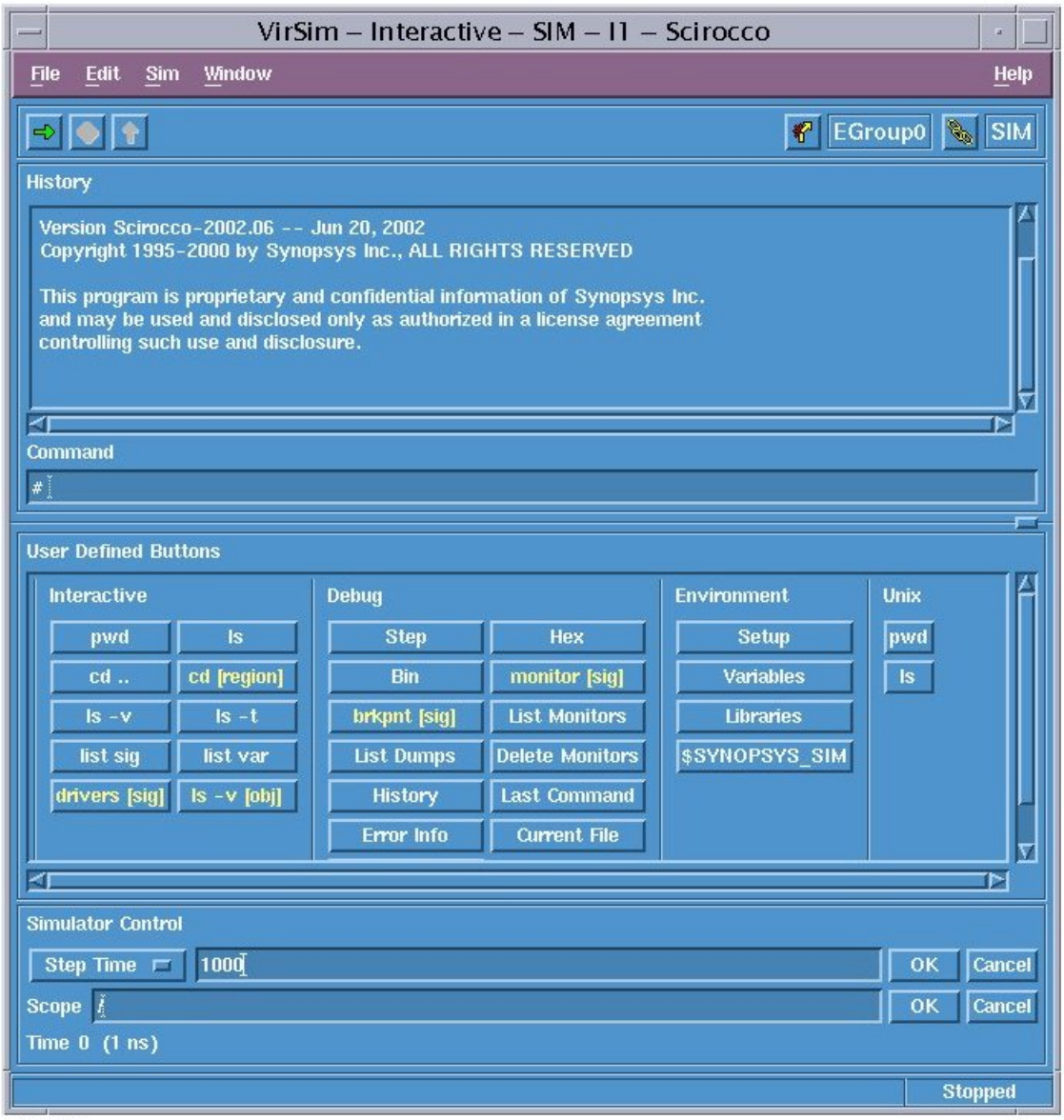


Fig. 24

Tramite la finestra illustrata qui sopra, si può scegliere il tempo da simulare. Nel nostro caso conviene procedere per passi di 1000 nano secondi.
Arrivati a questo punto bisogna decidere quali segnali visualizzare.
Dalla finestra "VirSim" selezioniamo premiamo ora il pulsante "Hierarchy" il quale consente di visualizzare la gerarchia del progetto illustrata in Fig. 25.
Il progetto nel suo complesso è visto come una sorta di sotto-directory collegate fra di loro in una struttura gerarchica. Selezionando una voce con il simbolo "freccia in basso" e possibile andare a vedere i segnali che contraddistinguono ogni sottoelemento.
Tramite questa finestra, chiamata "VirSim-Hierarchy" è possibile selezionare i segnali da visualizzare nell'elaborazione.

Fig. 25

Sempre dalla finestra "VirSim" bisogna premere il pulsante "Waveform" che permette di lanciare una finestra chiamata "VirSim-Waveform" in Fig.26. Essa consente di monitorare le forme d'onda durante la simulazione.

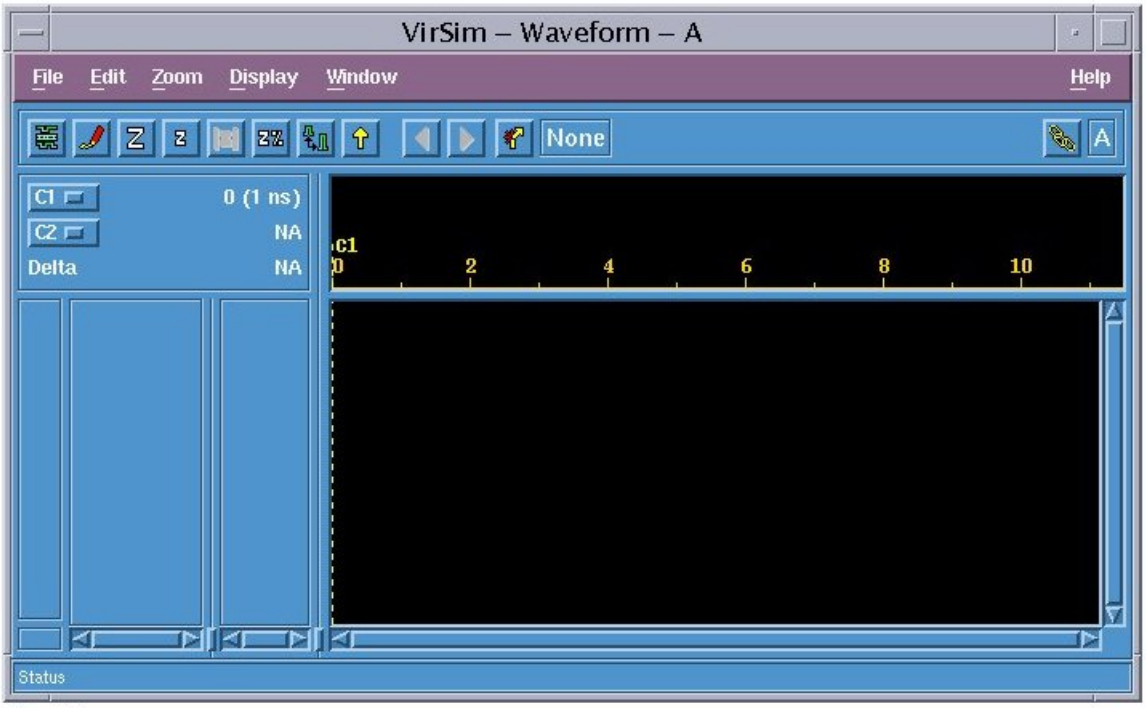


Fig. 26

Una volta attivate queste due finestre si ritorna a quella "VirSim-Hierarchy" per andare a selezionare i segnali da visualizzare durante la fase di simulazione. Per fare questo sono possibili due strade:

1. Utilizzare il pulsante "Add" dopo avere evidenziato con un click il segnale che ci interessa osservare.
2. Trascinare i segnali con il pulsante centrale del mouse.

Dopo aver compiuto queste operazioni nella finestra "VirSim-Waveform" avremo dei segnali con un valore indefinito.

Per simulare il comportamento del dispositivo e' sufficiente ritornare nella finestra "VirSim-Interactive", selezionare un valore di tempo da simulare e premere il tasto "Ok" a lato.

Nella finestra "VirSim-Waveform" compariranno i valori dei segnali per i primi istanti di tempo.

## *Bibliografia*


### Protocollo I2C

I2C Bus Specification, Philips 1995 ( http://www.philips.com )

### VHDL

1. “Introduzione al VHDL”
   ( http ://ticino.com/usr/pagna/Pagine/Documentazioni/Introduzione%20al%20VHDL.pdf )
2. “VHDL Reference Manual”, Synario
   (http://clsab.snu.ac.kr/course/cad99/vhdl_ref.pdf)
3. “VHDL”, Douglas L. Perry, McGraw-Hill 1998
   Collocazione biblioteca tecnico-scientifica 21a/206

### VirSim

1. Vhdl Simualtion Installation Guide – Synopsys
2. Vhdl Simulation Quick Reference – Synopsys
3. Vhdl Simulation Releas Note – Synopsys
4. Vhdl Simulation User Guide – Synopsys
5. VirSim Context Sensitive Help Menu – Synopsys
6. VirSim User Guide – Synopsys