Realizzazione su FPGA di un dispositivo master per la comunicazione secondo protocollo I2C

di De Pin Alessandro

Indice

Introduzione

Lo scopo di questo lavoro è quello di implementare su scheda FPGA Spartan 2 un dispositivo master in grado di gestire la comunicazione tra dispositivi slave su bus secondo il protocollo I2C.
In particolare si vuole che legga da una memoria i dati che poi scriverà sul dispositivo corrispondente.
Come circuito slave di test si è fatto uso dell'integrato della Philips SAA1064, ovvero un driver per display a sette segmenti, funzionante con il sopraccitato protocollo.
Il master realizzato può comunque avere anche un utilizzo più generale; uno degli intenti infatti è quello di un eventuale impiego per il pilotaggio di convertitori analogico/digitale presenti su scheda Virtex XCV2000.

Breve cenno al protocollo I2C

Il protocollo I2C è stato pensato dalla Philips e prevede l'utilizzo di due fili bidirezionali seriali e connessi all'alimentazione tramite resistenze di pull-up:

- SDA, ovvero la linea per il transito dei dati;
- SCL, la linea per il clock

Ogni dispositivo collegato al bus I2C funge o da slave o da master ed è identificato da un indirizzo univoco a otto bit; il secondo, in particolare, ha la funzione di generare il segnale di clock sulla linea SCL e trasferire i dati.
Le velocità di trasferimento dati previste sono tre, ma per il proposito di questo lavoro si è impiegato lo Standard-Mode, la cui bitrate ha un massimo di 100Kbit/s.

Il protocollo prevede un segnale di inizio trasmissione ed uno di fine della stessa:

- segnale di start: con SCL alto, SDA passa dallo stato alto a quello basso (Fig.1);
- segnale di stop: con SCL alto, SDA passa dallo stato basso a quello alto (Fig.2).

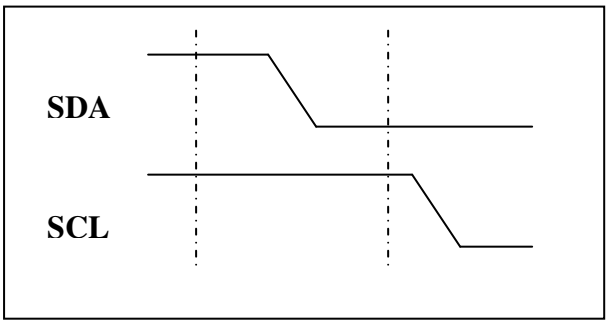


Fig.1

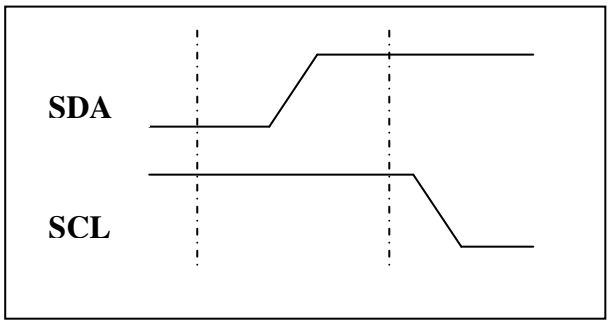


Fig.2

I dati su SDA vengono trasmessi un byte alla volta, a cui segue una conferma di avvenuta ricezione da parte del dispositivo slave interessato; ovvero vengono trasmessi otto bit di dati ed il nono ha la funzione di acknowledgement.

Un byte, per essere valido, dev'essere tale da mantenere la linea SDA stabile per l'intera durata del fronte positivo di SCL (Fig.3)

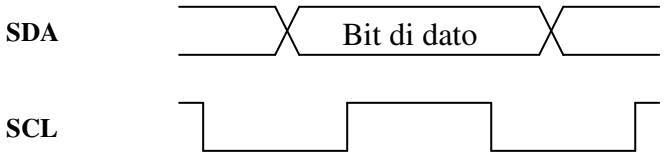


Fig.3

Durante il "bit di acknowledge", il trasmittente lascia SDA alta ed il ricevente la abbassa in caso di avvenuta ricezione o la lascia alta in caso contrario.
Un not-acknowledge può avvenire, quindi, nel caso in cui il byte immediatamente precedente non sia stato ricevuto correttamente: questo può essere un byte di dati effettivo, l'indirizzo della cella di memoria su cui andare a scrivere il dato oppure l'indirizzo stesso del dispositivo.
Nel caso di non ricevimento del dato, bisogna ricominciarne la trasmissione.

Può avvenire anche che il dispositivo slave con cui si vuole comunicare non sia pronto a ricevere; in tal caso, la linea SCL viene tenuta bassa da quest'ultimo al momento di ricevere il dato.

Un tipico esempio di comunicazione è riportato in Fig.4:

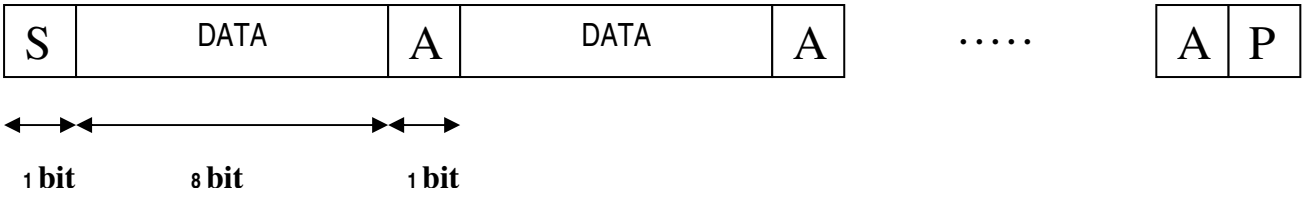


Fig.4

in cui il primo ottetto identifica sempre l'indirizzo dello slave su cui andare a scrivere i byte successivi nei corrispondenti registri interni; ovvero si procede come segue:

- si dà il segnale di start;
- si trasmette l'indirizzo dello slave su cui andare a scrivere;
- se lo slave risponde (ack), si trasmettono i bit successivi (Fig.4);
- in caso di not-ack o di fine trasmissione, si dà il segnale di stop.

In questo caso si è scelto di effettuare una trasmissione per ogni registro su cui si deve scrivere; ovvero si è scelto di effettuare una serie di trasmissioni caratterizzate da

- segnale di start;
- trasmissione indirizzo dello slave;
- trasmissione indirizzo del particolare registro interno su cui andare a scrivere;
- trasmissione del dato da scrivere sul particolare registro;
- segnale di stop.

Naturalmente ciascun ottetto è seguito dal bit di conferma/non conferma di avvenuta ricezione del byte.

La trasmissione si presenta quindi come descritta in Fig.5

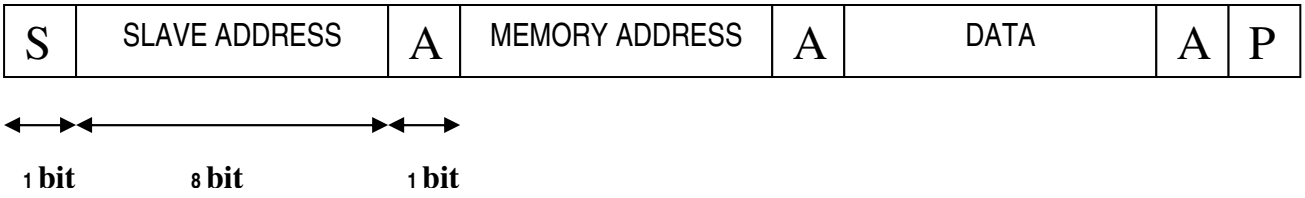


Fig.5

Realizzazione

In fase di progettazione si è utilizzato il software messo a disposizione dalla Xilinx: l'ISE 5.2. Per le varie simulazioni invece si è fatto uso del software Modelsim XE II 5.6.

L'idea è quella di realizzare un controllore che funzioni sulla base del seguente algoritmo:

1. *Inizializzazione della memoria da cui prendere i dati da scrivere sui vari registri alla prima cella;*
2. *Segnale di inizio trasmissione;*
3. *Trasmissione 8 bit di indirizzo dispositivo;*
4. *Aspetta ack: se arriva prosegue, altrimenti fa stop con messaggio d'errore;*
5. *Trasmissione 8 bit di indirizzo di memoria su cui andare a scrivere il dato;*
6. *Aspetta ack: se arriva prosegue, altrimenti fa stop con messaggio d'errore;*
7. *Trasmissione 8 bit di dati;*
8. *Aspetta ack: se arriva prosegue, altrimenti fa stop con messaggio d'errore;*
9. *Attesa per qualche ciclo di clock;*
10. *Incremento dell'indirizzo di memoria;*
11. *Se era l'ultima cella esce con messaggio di corretta trasmissione, altrimenti ricomincia da 2.*

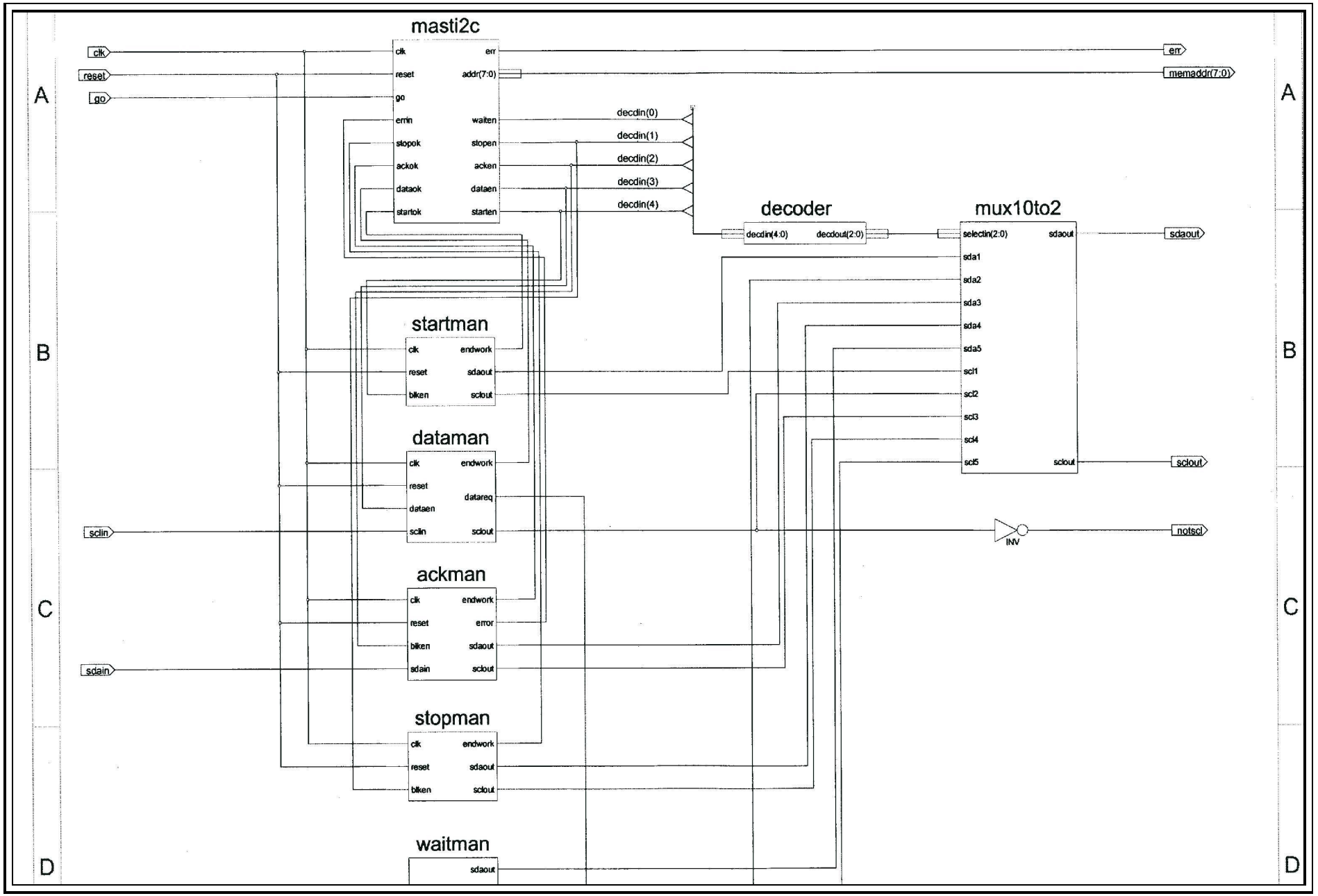
Per la realizzazione si è sviluppato un "core" in grado di gestire l'algoritmo precedente, collegato a dei dispositivi secondari che implementano loro stessi la generazione dei segnali sulle due linee SDA ed SCL; questi apparati sono (Fig.6)

- un dispositivo per la generazione del segnale di start (startman, acronimo di start manager);
- un dispositivo per la generazione del segnale di stop (stopman);
- un dispositivo per la gestione dei dati contenuti in memoria, nonchè per la generazione del segnale di temporizzazione opportuno sulla linea SCL (dataman);
- un dispositivo per la gestione dell'ack (ackman);
- un dispositivo per gestire le attese (waitman).

Ciascuno di essi viene abilitato dal core con un segnale di Chip Enable e, una volta terminata la loro funzione, avvisano il controllore tramite apposito filo (startok, stopok, dataok, ackok rispettivamente).
Da evidenziare inoltre che il dispositivo per la gestione dei dati abilita la memoria tramite il piedino d'uscita denominato con "datareq" e gli fornisce la temporizzazione tramite la linea SCL in uscita. Il dato dalla memoria invece arriva direttamente su SDA dalla pad d'ingresso denominata come "data".
Le uscite di questi sono inviate sui due fili tramite un opportuno multiplexer, che fornisce in uscita le linee SDA ed SCL ("sdaout" e "sclout").

Sono previsti due ulteriori ingressi, "sda in" ed "scl in", in ack-manager, per permettere il riconoscimento di ack/not ack, ed in data-manager, per fornire indicazione eventuale al master di dispositivo slave occupato, rispettivamente.

Il controller fornisce in aggiunta un filo in uscita, che poi andrà su un decoder per display a sette segmenti, in cui transita il segnale di stato per indicare un eventuale errore di trasmissione ed una linea per comunicare alla memoria sulla board l'indirizzo della cella da cui fornire il dato.


masti2c
clk
reset
go
err
addr(7:0)
ermin
stopok
ackok
dataok
startok
waiten
stopen
acken
dataen
starten
err
memaddr(7:0)
decdin(0)
decdin(1)
decdin(2)
decdin(3)
decdin(4)
decoder
decdin(4:0)
decdout(2:0)
mux10to2
selectin(2:0)
sdaout
sda1
sda2
sda3
sda4
sda5
scl1
scl2
scl3
scl4
scl5
sclout
sdaout
sclout
startman
clk
reset
blken
endwork
sdaout
sclout
dataman
clk
reset
dataen
sclin
endwork
datareq
sclout
sclin
notscl
INV
ackman
clk
reset
blken
sdain
endwork
error
sdaout
sclout
sdain
stopman
clk
reset
blken
endwork
sdaout
sclout
waitman
sdaout
A
B
C
D

Il controllore inizia la trasmissione non appena si preme il tasto designato per tale funzione (“go”), di default è stato impostato nel file di vincoli il pushbutton presente sulla board; questo pulsante, inoltre, è utilizzato come restart nel caso si termini la trasmissione o si arrivi ad una situazione d’errore (ad esempio, in 4.,6. e 8.).

Questo dispositivo è riassunto nella seguente macchina a stati finiti (Fig.7):

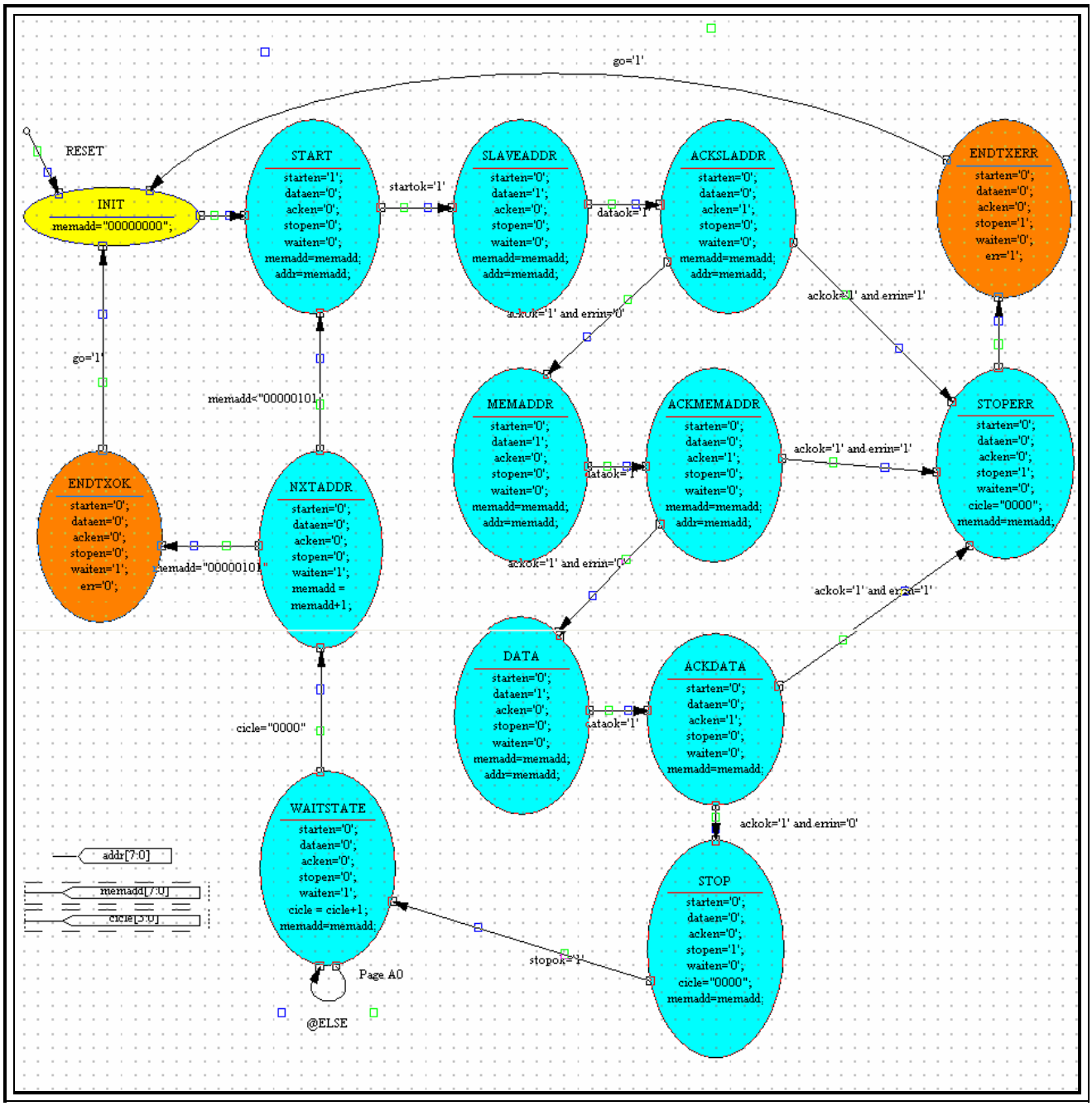


Fig.7 - Schematico di masti2c

Si può notare che ad ogni stato viene attivato il dispositivo atto a svolgere la funzione designata da quello stato (pilotare adeguatamente le due linee bidirezionali), tramite il rispettivo Chip Enable; a loro volta, ciascuno di questi “manager”, quando terminano, restituiscono un segnale di ok al controllore centrale, che passa allo stato successivo.

Si osservi che il segnale cicle è interno ed è stato fissato arbitrariamente a 16 ed è utilizzato come contatore di cicli di clock in cui il dispositivo resta nello stato di attesa (con SDA ed SCL nello stato alto) a fine trasmissione, prima di iniziarne una nuova: questo ha lo scopo di prevenire un'eventuale congestione sullo slave.

Viene impiegato anche un contatore (memaddr) per riferire alla rispettiva cella di memoria, atta a contenere i dati: esso è inizializzato a 0 (prima cella, da 24bit) ad inizio trasmissione; quando si arriva al segnale di stop e dopo aver effettuato i cicli di attesa, viene valutato se si è raggiunta la dimensione massima della memoria (in questo caso si passa ad uno stato finale che mostra l'esito della trasmissione) oppure no (in questo caso il contatore viene incrementato di una unità in modo da 'puntare' alla cella successiva).

Data la tempistica ed i sincronismi alquanto stringenti richiesti dal protocollo I2C, si è pensato di scegliere una frequenza di lavoro in modo da avere un fitto campionamento e quindi un maggior controllo (Fig. 8)

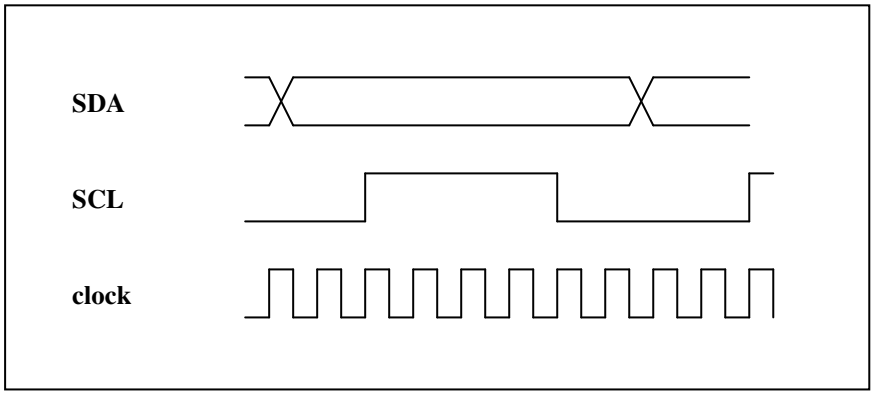


Fig.8

In particolare ogni periodo di SCL è stato campionato in otto sottoperiodi: questo ha permesso di gestire con un buon margine non solo i ritardi dovuti alla propagazione del segnale tra componenti diversi, ma anche all'interno di un componente stesso.

La memoria, invece, è stata realizzata direttamente in VHDL ed è stata concepita per avere una lunghezza variabile (ma predefinita in fase di avvio del dispositivo) di parole da 24bit, in cui i primi 8 indicano l'indirizzo dello slave, il secondo ottetto in dica l'indirizzo della cella di memoria di questo su cui andare a scrivere il dato rappresentato dall'ultimo byte.
Essa è quindi customisable; ovvero, è possibile inserirne i dati, espressi in binario incominciando dal bit più significativo per ogni ottetto, agendo terzina di otto bit per terzina direttamente sul rispettivo file vhdl.
Uno stralcio del listato della memoria viene presentato di seguito:

```
case datain is
        when "00000000" => data<= "011100000000000001001000";   --E00048
        when "00000001" => data<= "011100000000000100000001";   --E00101
        when "00000010" => data<= "011100000000001000000010";   --E00202
        when "00000011" => data<= "011100000000001100000011";   --E00303
        when "00000100" => data<= "011100000000010000000100";   --E00404
when others => data<= "000000000000000000000000";
```

end case;

in cui “datain” è la posizione della cella (incrementale) e fissa la dimensione della memoria (in questo caso datain è di 8bit, quindi la memoria ha una capacità massima di 256 celle); mentre “data” è il contenuto della singola parola (a 24 bit) in cui i primi 8 sono l’indirizzo dello slave (in questo caso in esadecimale E0, che è quello corrispondente all’SAA1064 di test), i secondi l’indirizzo di un suo registro interno e l’ultimo ottetto è il vero e proprio dato da scrivere.
Quindi, se si volesse aumentare la capacità della memoria è sufficiente ridefinire il vettore “datain” con il numero di bit desiderato; per immettere i dati nelle celle invece basta aggiungere righe del tipo

*when "*posizione cella *" => data<= "*parola di 24 bit*";*

all’interno del case.
Attenzione, però, visto che il core funziona fino a quando si raggiunge l’ultima cella di memoria piena (in questo caso la quinta), nel caso in cui il numero di parole da 24bit venga a variare rispetto a quello impostato di default (5 appunto), bisogna andare a modificare il segnale interno “memaddr” nel file masti2c.vhd o in masti2c.dia inserendo il valore e la dimensione opportuna.

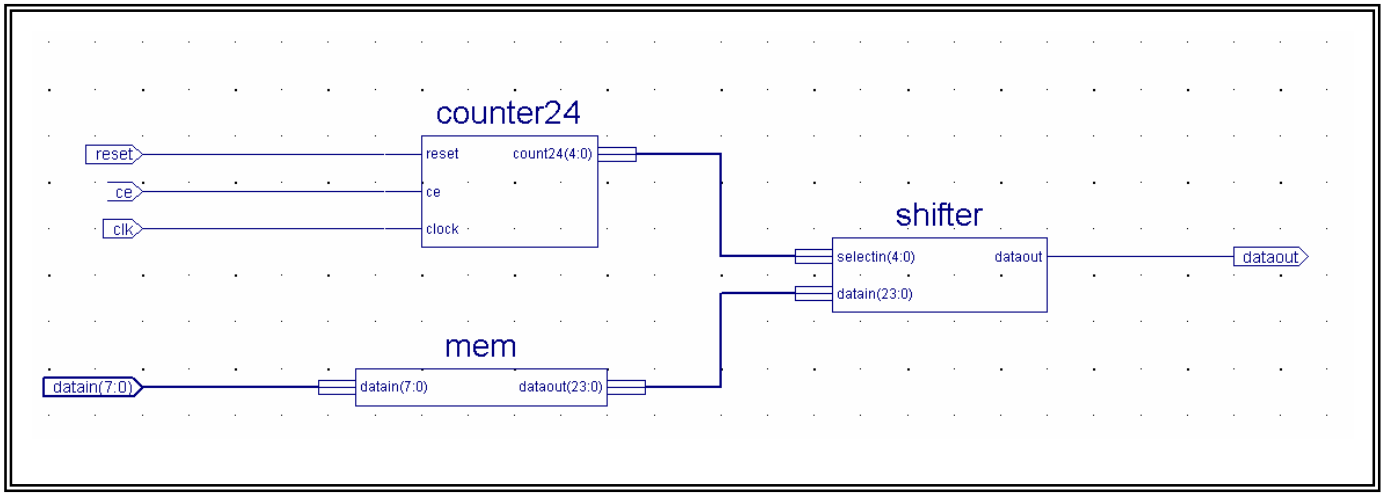


Fig.9 – Schematico di memory

Da essa viene estratta una parola alla volta e, tramite un contatore ciclico appropriato, i bit vengono serializzati uno ad uno con una cadenza appropriata, gestita dal controllore (Fig.9).
Per mantenere un perfetto sincronismo e permettere quindi che il dato sia stabile al fronte alto di SCL, viene utilizzata direttamente questa linea per pilotare l’estrazione della parola dalla memoria; in particolare si fa in modo che il dato da inviare su SDA venga preparato subito dopo il fronte di discesa di SCL relativo al dato precedente; in tal modo sono contemplati abbondantemente anche eventuali ritardi dovuti alla propagazione del segnale.

Lo schematico globale è raffigurato qui sotto (Fig.10):

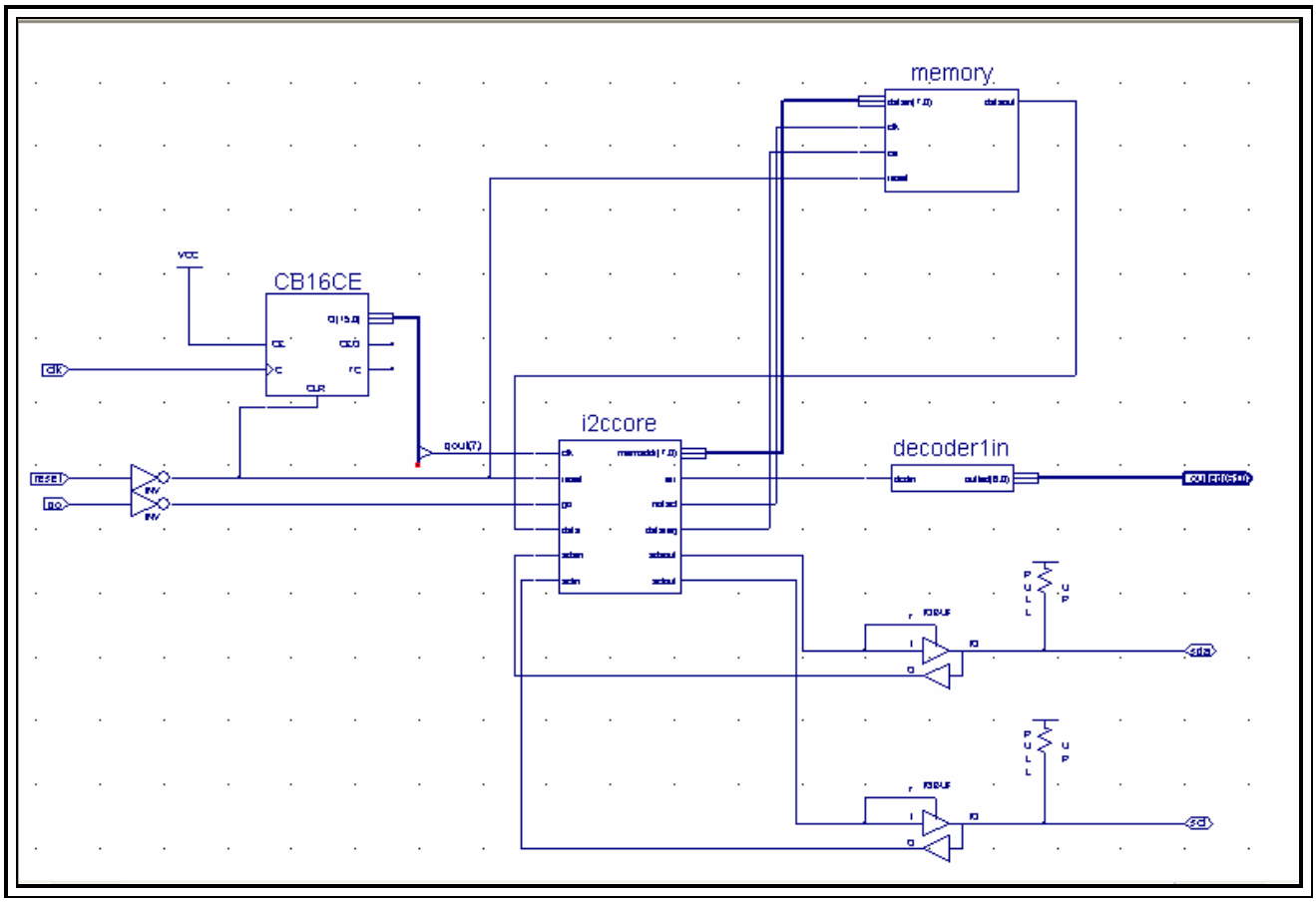


Fig.10

Per la sua esecuzione con il chip di test precedentemente menzionato è sufficiente effettuare il download su scheda del file .bit. In alternativa per comunicare con un qualsiasi altro dispositivo (slave) avente interfaccia I2C si deve riconfigurare la memoria con i dati voluti ed appropriati per questo come sopra descritto e ricompilare il tutto con il software della Xilinx per rigenerare un nuovo file con estensione .bit da caricare sulla board.

L'idea di base, come già detto, può essere implementata anche su board diverse con software differente, dato che tutto il codice è nato o convertito di seguito in linguaggio vhdl, a patto di un preventivo ed opportuno adattamento dei listati.

Elenco dei files utilizzati

Core I2C:	*masti2c.vhd*
Dispositivi di gestione SDA,SCL:	*startman.vhd* *dataman.vhd* *ackman.vhd* *stopman.vhd* *waitman.vhd*
Multiplexer per la gestione delle sorgenti di segnale:	*mux1.vhd* *mux2.vhd*
Decoder per il controllo del multiplexer:	*decoder.vhd*
Memoria:	*mem.vhd* *counter24.vhd* *shifter.vhd*
Decoder per display a sette segmenti:	*decoder1in.vhd*
File di vincoli:	*vincoli.ucf*