INTRODUZIONE AL BUS IIC

È uno standard da tempo impiegato in molti apparati consumer e anche in alcuni progetti proposti nei mesi passati. Ci siamo però accorti che non tutti conoscono i fondamenti su cui si basa questo sistema di trasmissione dati

di Andrea Sbrana - 1ª parte

Il bus IIC è stato ideato dalla Philips con lo scopo di far dialogare più chip e periferiche con il minor numero di fili possibile, ma al tempo stesso con la massima affidabilità.

I primi esperimenti furono eseguiti con dialoghi tra prodotti per la cucina, come forno a microonde, frigorifero, lavastoviglie, poi, vista la piena funzionalità ed affidabilità, questo bus fu impiegato anche per far dialogare chip all'interno di una stessa scheda, come per esempio il telaio di un televisore.

Oggi, infatti, una delle più comuni apparecchiature consumer che fa uso del bus IIC è proprio il televisore: ci sono collegati i controller di gestione, il VFO, la memoria EEPROM, il ricevitore del telecomando, il decoder del televideo e quello del PAL, i sincronizzatori video, i PLL, i processori audio, ecc.

Ma l'IIC bus viene anche impiegato con successo nei telefoni cellulari, nelle centraline auto, nelle schede di comunicazione, nei controller industriali e, comunque, dove non sia necessaria un'elevata velocità, perché, come vedremo, la massima velocità standard raggiungibile è di 100 kHz, anche se poi ci sono periferiche dette FAST che possono anche raggiungere i 400 kHz ed alcune anche 1 MHz.

Lo standard IIC bus è nato infatti per rispondere a poche ma importanti regole: numero ridotto di componenti sulla scheda, linee di connessione (quindi costi) ridotte al minimo, velocità non eccessiva.

Per realizzare sistemi con queste caratteristiche, era necessario ideare una struttura a bus di tipo seriale. Per struttura a bus, si intende una architettura in cui tutti i chip del circuito sono collegati per mezzo di linee comuni, mentre per "seriale" si intende che l'informazione passa serialmente, ovvero su di una sola linea un dato dopo l'altro.

Per facilitare la comprensione, pensiamo allo schema di un computer attuale: il processore è connesso (vedi Figura 1) alle periferiche con un bus PARALLELO, ovvero sia i dati che gli indirizzi vengono passati uno per ogni filo, quindi il dato D0 passerà sempre per il filo D0, il dato D1 per il filo D1 e così via.

Chiaramente, in questo modo la velocità è la massima possibile, perché in un solo colpo di clock viene passato sia un indirizzo completo (esempio 32 bit), sia un dato (altri 32 bit) che altri segnali di controllo, ma ovviamente per ogni linea, sullo stampato dovrà corrispondere una pista fisica.

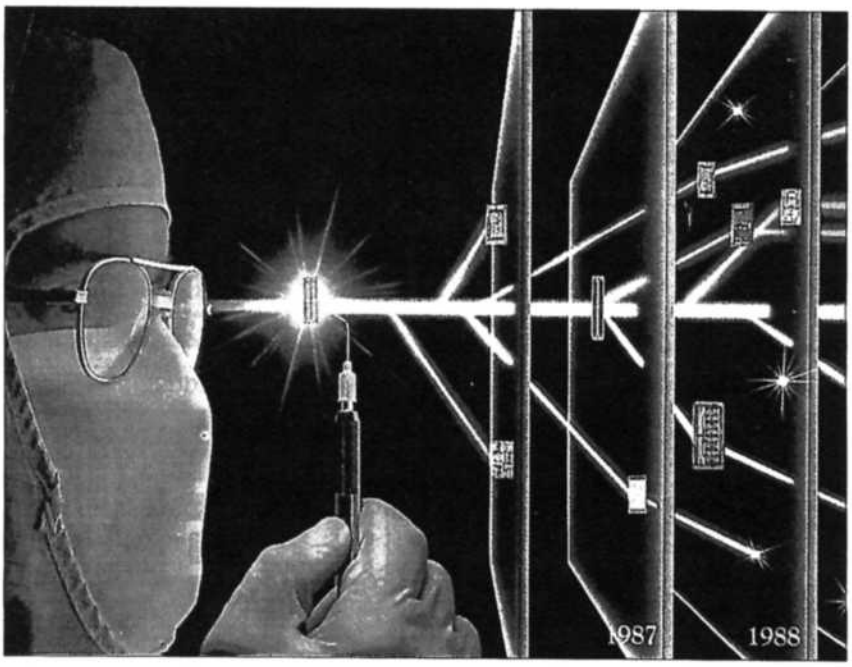
In Figura 2 possiamo vedere la struttura di un bus IIC, ovvero con due soli fili. Questa volta, per far passare un ipotetico indirizzo di 32 bit e un dato di altri 32 ci vorranno almeno 64 colpi di clock, uno per ogni bit, più quelli per i segnali di controllo. Capite quindi la differenza tra un collegamento di tipo PARALLELO ed un altro di tipo SERIALE. Poiché nella maggior parte delle applicazioni consumer non è rilevante la velocità di esecuzione, i costruttori preferiscono realizzare piste con pochi collegamenti a scapito della velocità.

Analisi dei collegamenti

Il bus IIC supporta sia processori che periferiche di tipo NMOS, CMOS, IIL. Sono necessari, ed al tempo stesso sufficienti, due soli fili di collegamento: SDA (Serial DAta) e SCL (Serial CLock) che si incaricano rispettivamente di portare i dati e di sincronizzare i vari dialoghi.

Ogni chip IIC viene univocamente determinato da un indirizzo e può sia ricevere che trasmettere informazioni con un protocollo che poi vedremo. Chiaramente ci saranno anche periferiche che saranno solo abilitate alla sola ricezione, come un controller per display LCD, altre abilitate alla sola trasmissione, come interfacce per tastiere, altre infine abilitate ad entrambe le funzioni, come per esempio le memorie.

Vedremo poi che ci sono anche periferiche che possono modificarsi da master in slave (dopo chiariremo anche questo concetto) e viceversa, come per esempio i microcontroller. Il bus IIC infatti è stato concepito come MULTI-MASTER-BUS,



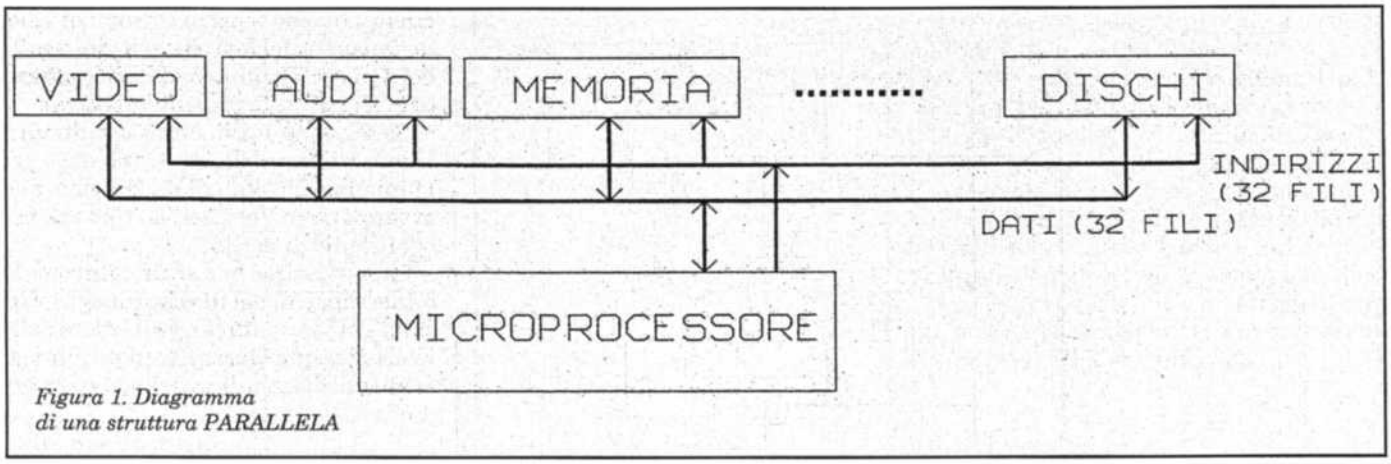


Figura 1. Diagramma di una struttura PARALLELA

ovvero che permette la coesistenza di più master, ovviamente in attività uno per volta.

A grandi linee, il dialogo tra due chip A e B, potrebbe svolgersi nel seguente modo: se A vuole inviare un'informazione a B, allora:

- A inizia il dialogo (se possibile)
- A (in questo caso MASTER) invia l'indirizzo di B (SLAVE)
- A invia il dato (o i dati) a B
- A termina il dialogo

Se, invece, A vuole ricevere informazioni da B allora:

- A inizia il dialogo (se possibile)
- A invia l'indirizzo di B
- A attende il dato da B
- A termina il dialogo

Abbiamo specificato che A inizia il dialogo solo se è possibile, ovvero se il bus è ritenuto LIBERO, e non impegnato da altre comunicazioni.

La sincronizzazione del dialogo avviene, negli esempi precedenti, sempre da A. Per prevenire collisioni sul bus, è prevista una procedura di arbitrazione del bus.

Terminologia del bus IIC

TRANSMITTER
Il chip che invia dati sul bus
RECEIVER
Il chip (o i chip) che riceve dati dal bus
MASTER
Il chip che inizia un dialogo, genera il clock di riferimento e poi termina il dialogo
SLAVE
Il chip indirizzato dal master
MULTI-MASTER
Quando sono attivi contemporaneamente più chip alla volta
ARBITRATION
Meccanismo che garantisce la correttezza dei dialoghi
SYNCHRONIZATION
Procedura di sincronizzazione tra due o più chip

Caratteristiche generali

In Figura 3 possiamo vedere come ogni singola periferica venga connessa al bus IIC ed in particolare anche l'equivalente elettrico sui due segnali SDA e SCL.

Quest'ultimi sono entrambi bidirezionali e connessi al positivo per mezzo di una resistenza di pull-up.

Quando il bus è libero, entrambe le linee sono mantenute ad alto livello logico tramite queste resistenze di pull-up. Il numero di periferiche che è possibile connettere al bus è quasi illimitato, poiché vengono offerti "ripetitori" di segnali per aumentare le distanze tra i chip.

L'alimentazione dei chip sotto bus IIC è rigorosamente di 5 volt: lo standard impiegato da tutti i chip delle ultime generazioni, ma ci sono periferiche che riescono a funzionare anche sotto i 3 volt.

I valori delle due resistenze di pull-up, non devono essere casuali, ma vanno calcolati in base a specifiche scelte

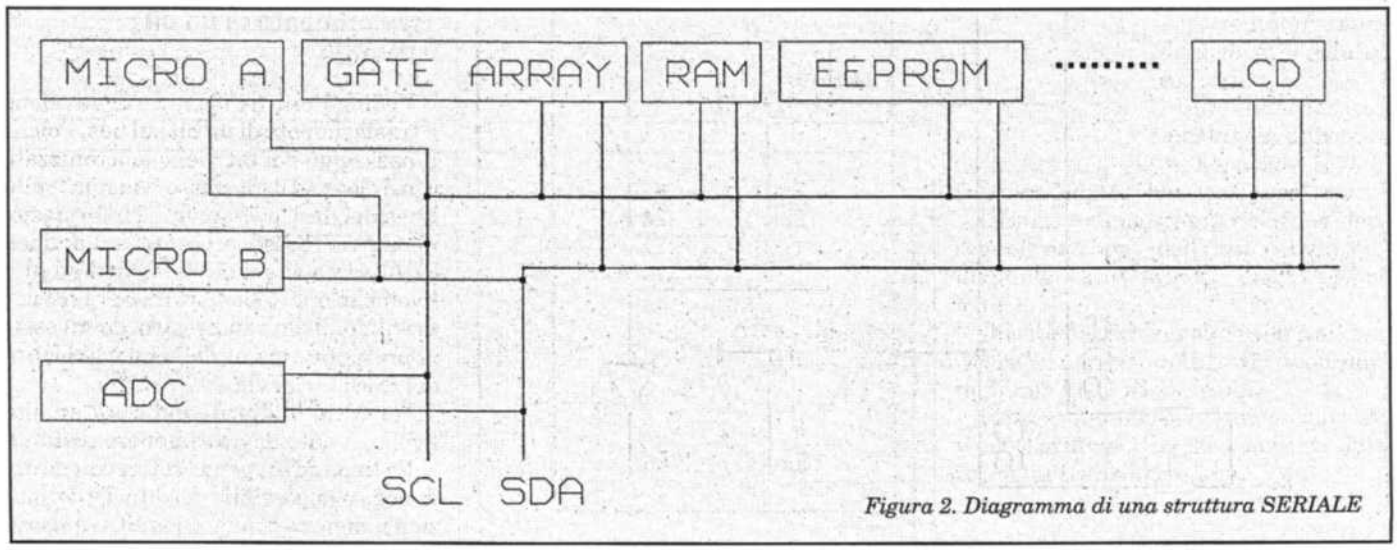


Figura 2. Diagramma di una struttura SERIALE

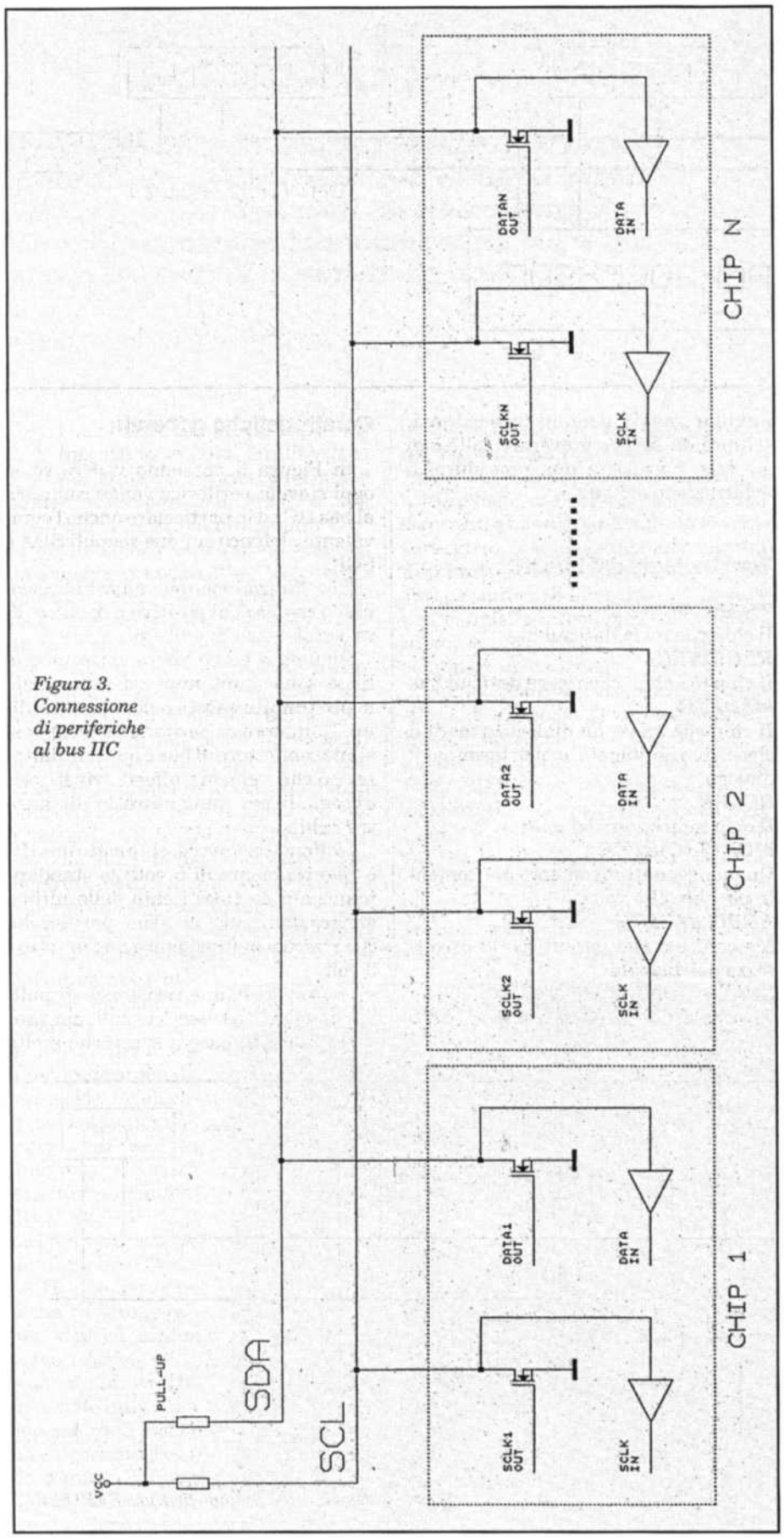


Figura 3. Connessione di periferiche al bus IIC

circuitali come tensione di alimentazione, capacità del bus (che non deve eccedere i 400 pF), numero di chip connessi (che identifica la corrente relativa).

La tensione di alimentazione limita il valore minimo delle due resistenze per la minima corrente di sink dichiarata, ovvero 3 mA a Vol-max, cioè 0,4 volt per ogni stadio di uscita.

La capacità del bus è calcolata come la totale capacità del filo che collega i vari chip. Tale capacità limita il valore delle resistenze per garantire dei fronti di salita e di discesa di massimo 1 microsecondo.

Condizione di Start e di Stop

Con condizione di start e stop viene inteso rispettivamente il protocollo di inizio di un dialogo e quello di terminazione. In Figura 4 possiamo vedere entrambe queste condizioni.

La condizione di START può ovviamente avvenire solo quando il bus è libero, ovvero solo quando sia la linea SDA che quella SCL si trovano ad alto livello.

Allora abbiamo una condizione di START se, mantenendo la linea SCL ad alto livello, si porta la linea SDA a basso livello. Successivamente sarà possibile il dialogo sincronizzandoci con il clock e con modalità che adesso vedremo.

La condizione di STOP, invece, si potrà verificare solo al termine di un dialogo, quindi viene identificata quando, con la linea SCL ad alto livello, la linea SDA viene portata da basso ad alto livello. Vedremo che non è possibile avere questa condizione durante il normale dialogo.

Trasferimento di un bit e dialogo

Vediamo ora in Figura 5 come avviene il trasferimento di un bit sul bus. Poiché il passaggio del bit viene sincronizzato con il clock ed il bit stesso "viaggia" sulla linea dei dati, ovviamente l'informazione dovrà essere presente sulla linea SDA prima che il clock si porti ad alto livello. In quel momento il dato è ritenuto valido. Il dato successivo, dovrà essere presentato prima della nuova risalita del clock e così via.

Per tutta la durata del clock ad alto livello, il dato dovrà rimanere costante sulla linea SDA, pena l'interruzione del dialogo o la possibile perdita d'informazione, mentre con la linea SCL a basso

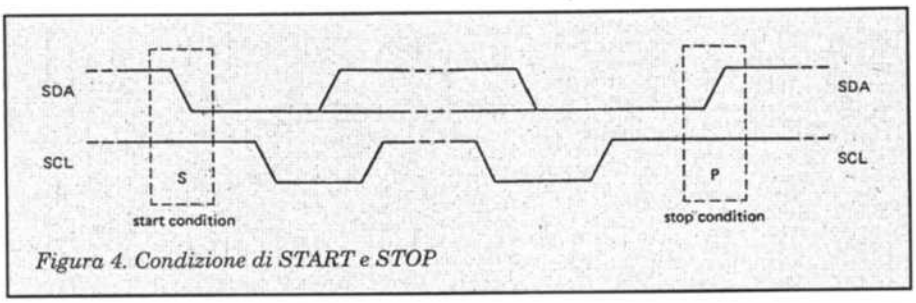


Figura 4. Condizione di START e STOP

livello è possibile anche mettere le periferiche in three-state.

In Figura 6 troviamo un esempio di dialogo che avviene sul bus.

Ogni byte inviato sulla linea SDA, sia esso di dato, di indirizzo o di controllo, deve necessariamente essere di 8 bit.

Al contrario, il numero di byte che possono transitare durante un dialogo può variare a discrezione del master (quindi in definitiva del programmatore del controller che svolge tale funzione).

Ognuno di questi byte però, deve essere seguito da un bit detto di acknowledge, ovvero di riconoscimento.

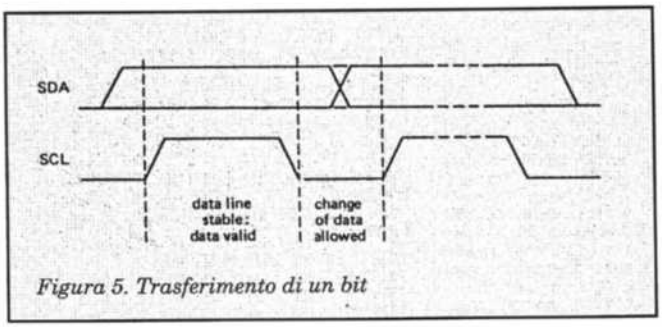


Figura 5. Trasferimento di un bit

Vedremo che alcune periferiche non rispettano però completamente questa direttiva. Il byte viene inviato con il MSB (Major Significant Bit = bit di peso maggiore) per primo.

Quindi, se per esempio volessimo inviare il numero 156 decimale, che in binario viene espresso come 10011100, dovremmo prima inviare "1", poi "0", poi "0", poi "1", poi "1", poi "1", poi "0" ed infine "0".

Ci sono casi in cui il ricevitore, dopo la ricezione di un byte, non possa ricevere altri byte per un certo tempo perché impegnato in attività proprie, come per esempio la memorizzazione di un dato in una memoria di tipo EEPROM che richiede circa 10 mS di tempo. In queste situazioni il ricevitore stesso tiene bassa la linea SCL fino a quando non sarà ancora disponibile a ricevere ancora dati (come il DTR nella RS-232).

Acknowledge

Il trasferimento dei dati con l'acknowledge è obbligatorio sul bus IIC. Il clock relativo al bit di acknowledge viene generato dal master che, al tempo stesso rilascia la linea SDA (livello alto) per verificare se l'acknowledge viene prodotto dallo slave indirizzato. Il chip slave infatti, in corrispondenza dell'impulso di clock relativo all'acknowledge, deve portare a basso livello la linea SDA e tenerla così per tutta la durata dell'impulso stesso (vedi Figura 7).

Generalmente, un ricevitore che è stato indirizzato dal master, è obbligato a generare un acknowledge dopo ogni byte ricevuto). Quando ciò non accade, ad esempio perché lo slave è impegnato, lo stesso slave deve tenere alta la linea SDA.

In questo modo, per recuperare del tempo prezioso, il master può decidere di inviare una condizione di STOP ed interrompere così il collegamento in modo da dedicarsi ad altri compiti.

Allo stesso modo, il master deve abortire il dialogo se lo slave, dopo aver inviato l'acknowledge, si blocca per vari motivi.

Un master, mentre è in ricezione, deve segnalare l'invio dell'ultimo byte non inviando l'acknowledge dopo l'ultimo byte.

Questa pratica è molto utile per "scaricare" in una sola volta tutto il contenuto di una memoria seriale.

Il prossimo mese vedremo come avviene l'arbitrazione ed i formati delle trame in transito sul bus.

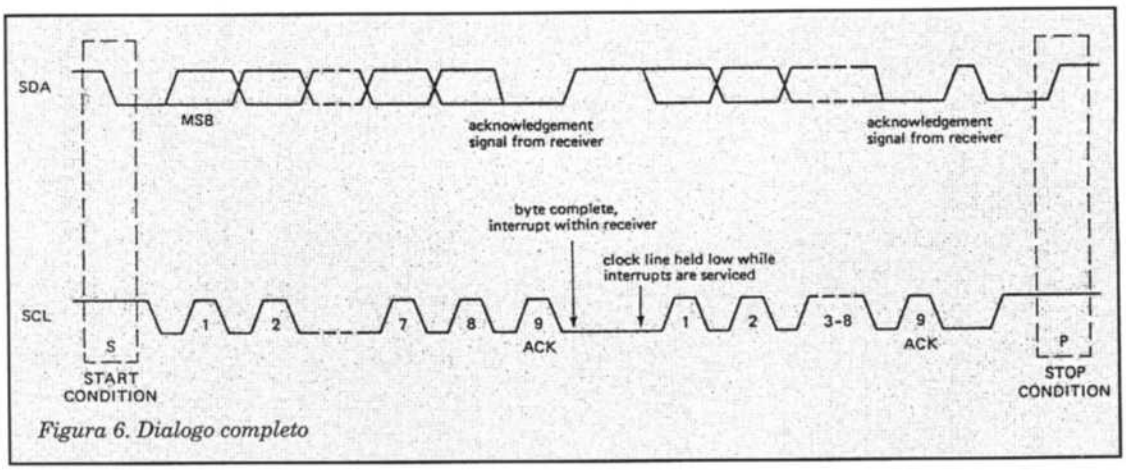


Figura 6. Dialogo completo

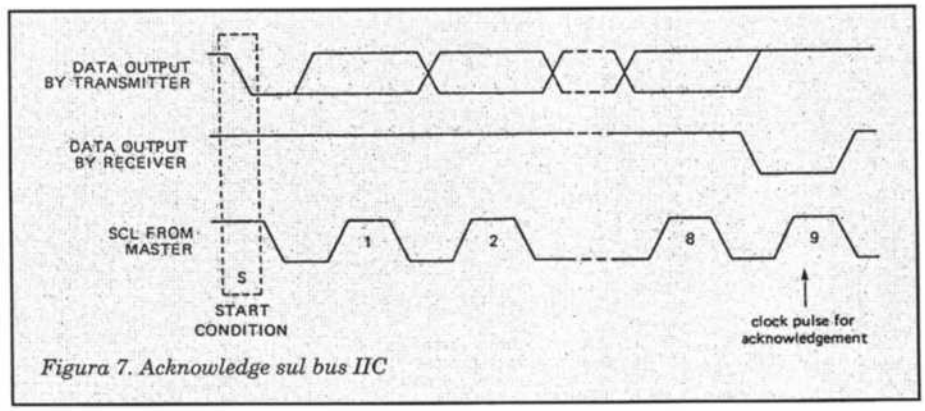


Figura 7. Acknowledge sul bus IIC

continua

INTRODUZIONE AL BUS IIC

Nella prima parte abbiamo imparato a conoscere i segnali di trasmissione del bus più diffuso in ambito elettronico. In questa seconda e conclusiva invece, ci dedichiamo agli aspetti più pratici della questione

Andrea Sbrana - 2ª parte

Riprendiamo la trattazione del bus IIC con le modalità di sincronizzazione che avevamo brevemente descritto il mese scorso.

Ogni master genera il proprio clock sulla linea SCL per trasferire dati sul bus IIC. Un dato è valido solamente durante il periodo in cui la linea SCL è ad alto livello, quindi si capisce come sia necessario un clock ben definito affinchè abbia luogo la procedura di arbitrazione.

La sincronizzazione con il clock viene realizzata sfruttando la connessione detta "wired-AND" sulla linea SCL del bus.

Questo significa che con una transizione della linea da alto a basso livello, tutte le periferiche dovranno attendere che la linea venga liberata, poiché per la particolare implementazione basta che un chip porti a livello basso una linea per avere uno zero logico anche se tutti gli altri chip sono a livello logico 1.

Un esempio è dato proprio dalla porta AND: se dei due ingressi almeno uno è a zero, sicuramente l'uscita rimane a zero.

Ogni chip ha un suo tempo di interdizione dopo che ha constatato che la linea SCL è a basso livello, quindi attende tale tempo e poi riprova a partire.

In questo modo viene garantita la sincronizzazione di più master agendo solo sulla linea SCL.

In Figura 8 è possibile vedere come ciò avviene.

Arbitrazione

In Figura 9, invece, è possibile vedere come avviene l'arbitrazione di due master per mezzo della linea SDA.

Come per la linea SCL, l'arbitrazione ha luogo con lo stesso meccanismo della logica wired-AND.

Quando un master trasmette un livello alto e, contemporaneamente, un altro master sta trasmettendo un basso livello, il primo metterà in off il suo buffer di uscita poiché il livello sulla linea SDA non corrisponde a quello che dovrebbe esserci.

L'arbitrazione può continuare per molti bit, basta pensare a due byte complementari da trasmettere con due master. Il primo passo consiste nella verifica dell'indirizzo: se i due master stanno entrambi tentando di connettere lo stesso slave, l'arbitrazione continua con la comparazione dei dati.

Da notare che in questo processo non viene persa alcuna informazione, poiché sia gli indirizzi che i dati sono impiegati per l'arbitrazione.

Un master che ha perso l'arbitrazione può generare degli impulsi di clock fino alla fine del byte con il quale ha perso l'arbitrazione.

Se un master perde l'arbitrazione durante la fase di indirizzamento, è possibile che il master vincitore stia tentando di indirizzare proprio il master perdente. Quest'ultimo deve allora porsi immediatamente in configurazione slave.

Inoltre, è possibile sfruttare il meccanismo di arbitrazione come handshake per abilitare i ricevitori a far fronte ad esempio a richieste di trasferimento veloce (FAST mode) dei dati.

Formati delle frame

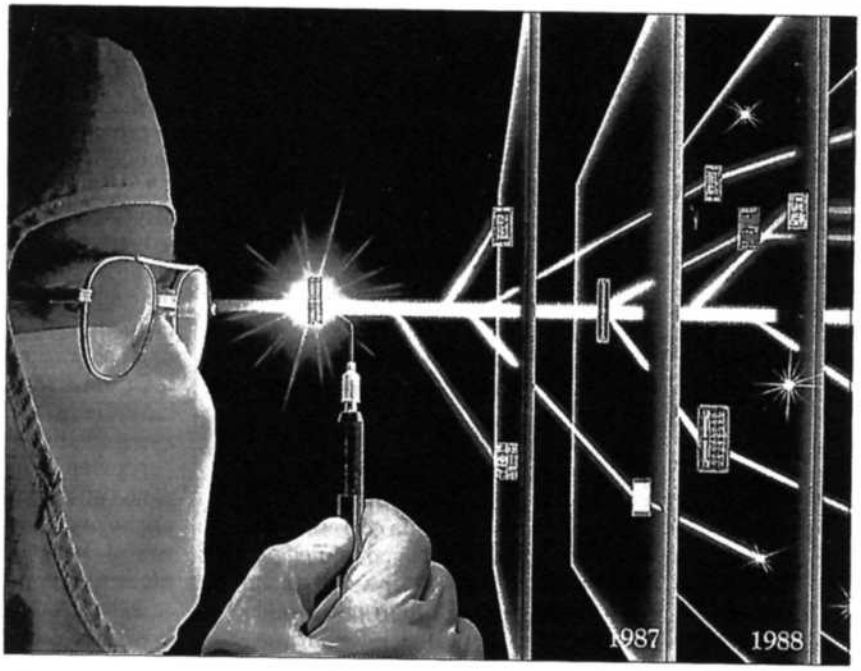
Il formato più comune è quello visibile in Figura 10: la prima azione è ovviamente una condizione di START, seguita da sette bit che definiscono lo SLAVE ADDRESS, poi viene inviato un bit per dire alla periferica se si vuole leggere o scrivere (bit R/W).

Il passo successivo è la generazione, da parte dello slave, dell'acknowledge.

A questo punto è possibile lo scambio di 8 bit di dato seguiti dal solito acknowledge, ripetuti anche N volte, fino all'invio, da parte del master, della condizione di STOP.

Le condizioni di START e di STOP le abbiamo analizzate nella scorsa puntata, quindi adesso analizziamo come avviene l'indirizzamento.

Questa procedura è tale che il primo byte inviato dopo una condizione di START indichi sempre l'indirizzo della periferica che il master vuole selezionare. Un'eccezione è data dalla cosiddetta "chiamata generale", ovvero una



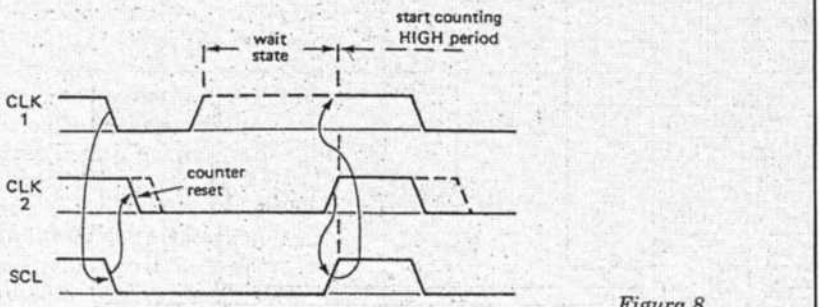


Figura 8. Sincronizzazione con il clock

chiamata che coinvolge tutti i chip.

Il secondo byte dopo la chiamata generale indica alle periferiche l'azione da compiere.

I primi sette bit del primo byte, indicano quindi lo SLAVE ADDRESS.

L'ottavo bit, indicato come R/W, in realtà non indica che la direzione del messaggio.

Uno zero dice che il master vuole scrivere il byte successivo nella periferica selezionata, mentre un uno indica che il master si aspetta di ricevere dalla periferica dei dati.

Ad ogni condizione di start, tutti i chip attendono lo slave address e lo confrontano con il loro posseduto internamente: se questo coincide, il chip interessato si pone nella condizione dettata dal bit R/W e cioè o in ricezione, o in trasmissione.

Lo slave address è generalmente composto da una parte fissa e da una parte programmabile. Generalmente la parte fissa è di 3 o 4 bit, in funzione dei pin disponibili per la formazione della parte programmabile.

Tanto per fare un esempio, è possibile collegare ad un bus IIC fino ad otto memorie EEPROM di tipo 2401, poiché queste hanno uno slave address composto da quattro bit fissi (1010) e da tre (2 elevato alla terza = 8 combinazioni) programmabili con tre pin esterni al chip.

La combinazione 1111xxx è riservata per future espansioni. L'indirizzo 1111111 è invece riservato per estensioni di indirizzo. Questo implica che il completamento dell'indirizzo potrà avvenire soltanto con l'invio di un ulteriore byte. Per esempio, le memorie di tipo 2432 e 2465 fanno uso di doppio indirizzamento, dato che con soli 7 bit non sarebbe stato possibile indirizzarne tutte le locazioni.

Chiamata generale

La chiamata generale è stata concepita per selezionare tutti i chip presenti sul bus, tuttavia se uno di questi sta eseguendo una propria funzione, oppure semplicemente ignora la chiamata, non ci sarà acknowledge di ritorno e quel chip rimarrà in slave mode.

Il compito che viene dato ai chip che rispondono, viene determinato dal secondo byte, e qui dobbiamo distinguere due casi: quando il bit meno significativo è zero, oppure uno.

Quando il bit meno significativo è zero, il secondo byte ha le seguenti definizioni:

00000110- Reset e scrittura della parte programmabile dello slave address per mezzo del software e dell'hardware. Questo può essere utile per un reset del sistema dopo una perdita di alimentazione.

00000010 - Scrittura dello slave address solo con metodo software. In questo modo non viene effettuato il reset del chip.

00000100 - Scrittura dello slave address solo mediante programmazione hardware. Anche qui non viene effettuato il reset.

00000000 - Codice non permesso.

Quando invece il bit meno significativo è 1, il significato è che la chiamata è stata effettuata da un componente hardware come ad esempio una tastiera, che non può essere programmata per trasmettere un ben preciso indirizzo.

Questa esigenza nasce dal fatto che esistono periferiche hardware che non possono conoscere a quali chip interessino i loro dati, quindi sono costretti ad avvisare tutti che sono pronti ad inviare alcuni dati.

I rimanenti sette bit del secondo byte contengono l'indirizzo del master hardware, per far capire ad un eventuale controller intelligente, chi vuole inviare dati.

In questi casi è obbligatorio l'uso di un microcontrollore sul bus.

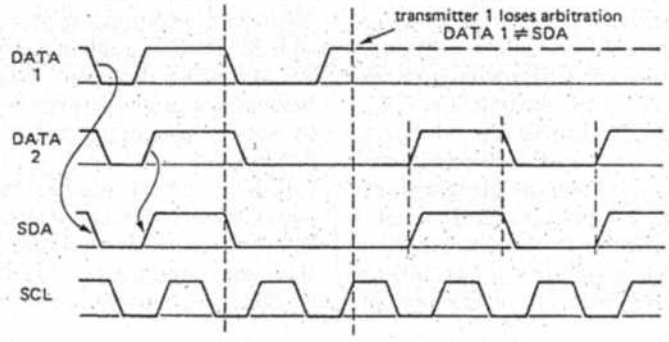


Figura 9. Arbitrazione di due master

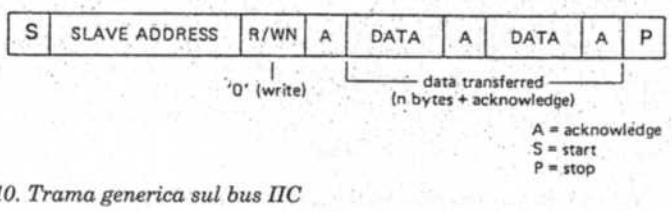


Figura 10. Trama generica sul bus IIC

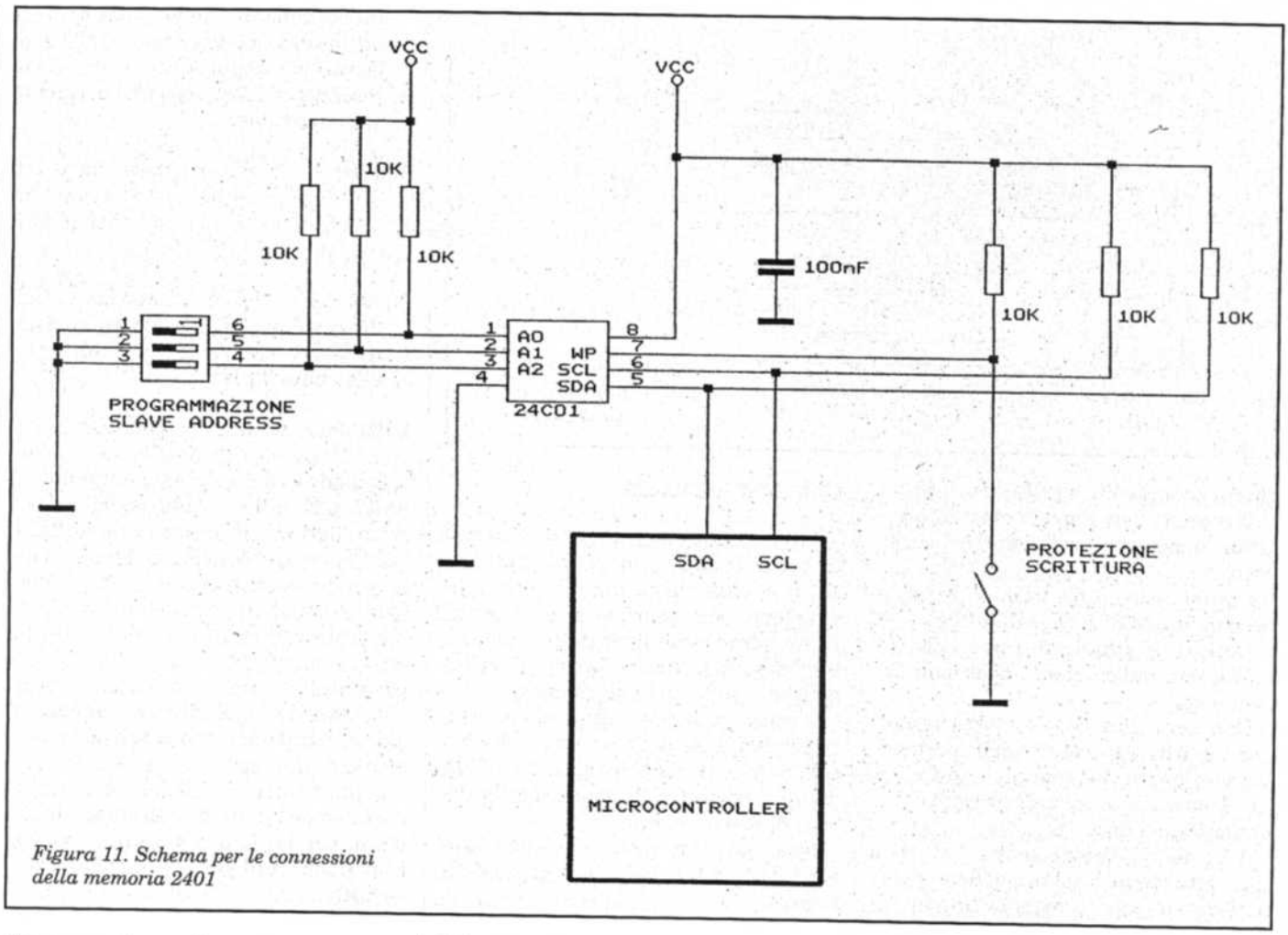


Figura 11. Schema per le connessioni della memoria 2401

Un esempio pratico

Poiché molti lettori avranno sicuramente voglia di sperimentare personalmente ciò che abbiamo fino ad ora esposto, proponiamo di sviluppare un semplice dialogo con memorie di tipo 2401, le più semplici da impiegare.

Proveremo a scriverle ed a rileggerle, ma chiaramente abbiamo bisogno di un controller intelligente, quindi adesso spiegheremo come realizzare il dialogo con queste memorie a livello di protocollo, poi ogni hobbista potrà "trasportarlo" nel linguaggio adatto per il suo controllore preferito.

In Figura 11 troviamo lo schema applicativo per connettere una memoria di tipo 2401 ad un qualsiasi microcontroller. I tre dip-switch posti sui pin 1, 2 e 3, servono per provare a vedere cosa succede variando lo slave address in modo hardware.

Il deviatore sul pin WP invece serve per verificare che con questo pin a livello 0 non è possibile scrivere nella memoria (attenzione che la serie delle memorie 85xx, sebbene simile, ha il significato di questo bit esattamente al contrario della serie 24xx). Le due resistenze di pull-up sono state calcolate per la specifica esigenza.

L'alimentazione della memoria dovrà essere di 5 volt, come quella del microcontroller.

Vediamo allora in Figura 12, come fare per scrivere un dato in una cella di memoria: come partenza si invia una condizione di START, sempre che il bus sia libero (ma è sicuramente libero perché l'unica periferica è la memoria!).

Poi si invia lo slave address, che nel

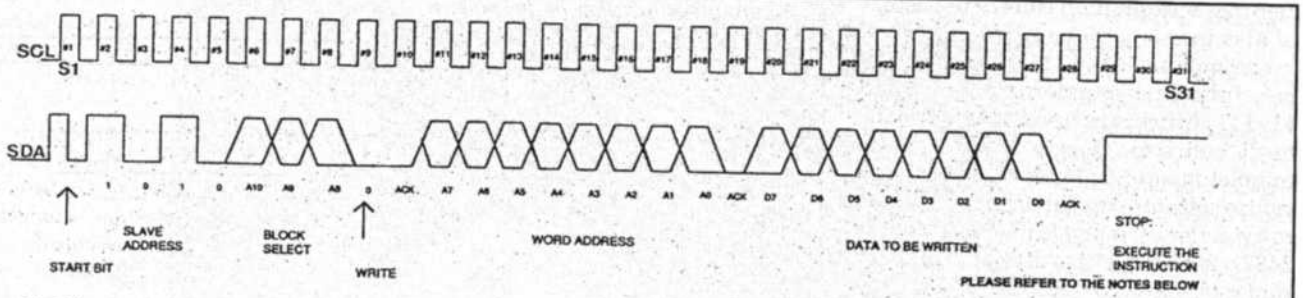


Figura 12. Diagramma per la scrittura di una memoria IIC bus

caso di questa memoria vale 1010ABC, dove al posto di ABC va scritto il livello sul pin corrispondente rispettivamente a A2, A1, A0. Questi ultimi tre bit vengono definiti BLOCK SELECT.

Successivamente si dovrà inviare uno "0" per dire alla memoria che vogliamo scrivere ed infine si aspetterà al ciclo di clock successivo un acknowledge. Terminato quest'ultimo, si invierà alla memoria prima il WORD ADDRESS, ovvero l'indirizzo della cella di memoria da scrivere, poi, dopo il solito acknowledge, si invierà il dato da scrivere ed infine si invierà, dopo un altro acknowledge, la condizione di STOP.

Tanto per dare qualche cifra, mantenendo i tre pin della selezione del blocco a zero, si potrebbe inviare come primo byte un 10100000, come indirizzo della cella di memoria un 00000010 (cioè la numero 2) e come dato un 10100110.

Resta ora da verificare se tale dato è stato effettivamente scritto nella memoria al voluto indirizzo.

Per far ciò potrete sfruttare il diagramma di Figura 13 che indica come leggere una determinata locazione: questa volta il protocollo è, paradossalmente, più lungo del precedente, perché prima bisogna indicare alla memoria che vogliamo leggere, poi va inviato l'indirizzo della cella ed infine ci sarà il passaggio dei dati.

Ma vediamo le varie fasi dettagliatamente: si parte come prima con uno START, poi si invia come precedentemente lo slave address ed il block select con l'istruzione di scrittura, poi si attende l'acknowledge e si invia il word address.

A questo punto però dobbiamo, dopo l'acknowledge, generare una nuova condizione di START, poi inviare il medesimo slave address, il medesimo block select, ma il bit di R/W dovrà essere posto a 1 perchè dobbiamo leggere dalla periferica, ed infatti, dopo un ciclo di clock, la memoria ci invierà in seriale il dato richiesto.

Tabella 1. Gli integrati attualmente più conosciuti che dialogano con questo bus

PCF8200	Sintetizzatore vocale
SAA1300	Tuner switching
SAA3028	Decodifica per ricevitore infrarossi
SAA4700	Processore per VPS (Stati Uniti)
SAA5243	ECCT decoder per televideo
SAA5245	USECCT decoder per televideo
SAA9041	Processore televideo DVTB
SAA9050	Decoder video PAL/NTSC
SAA9055	Decoder video SECAM
SAA9068	Picture in picture controller
SAB3035	Digital tuning per TV
SAF1135	Decoder 16 linee per VCR
TDA8370	Sincronizzatore per TV
TDA8405	Stereo/dual sound
TDA8420	Audio processor
TDA8440	Video/audio switch
TDA8442	Interfaccia decoder colore
TDA8443A	YUV/RGB switch a matrice
TDA8461	Decoder PAL/NTSC e processore RGB
TEA6000	FM/IF e tuning digitale per radio
TEA6300T	Sound fader per autoradio
TSA5510	PLL per TV e VCR
PCD3311	DTMF generator
PCD3315	Controller telefonico
PCD3341P	Gestione TONE/PULSE 100 memorie uso telefonico
PCF8566	LCD driver
PCF8570	RAM statica
PCF8572	EEPROM
PCF8573	Orologio/calendario
PCF8574	I/O expander
PCF8591	8 bit ADC/DAC
SAA1064	4 digit led driver
UMA1000	Data processor per cellulari

L'ultima fase sarà la chiusura del collegamento con una condizione di STOP.

Vediamo allora anche questa volta i byte da trattare: il primo byte da inviare sarà 10100000, il secondo sarà sempre 00000010 (vogliamo leggere la stessa locazione precedentemente scritta), il terzo sarà invece 10100001 perchè vogliamo leggere. Se tutto è a posto, il byte che la memoria ci tornerà dovrà essere proprio 10100110, ciè ciò che prima avevamo scritto.

Per coloro che vogliono prendere dimestichezza con queste memorie, consigliamo di acquistare un lettore/program-

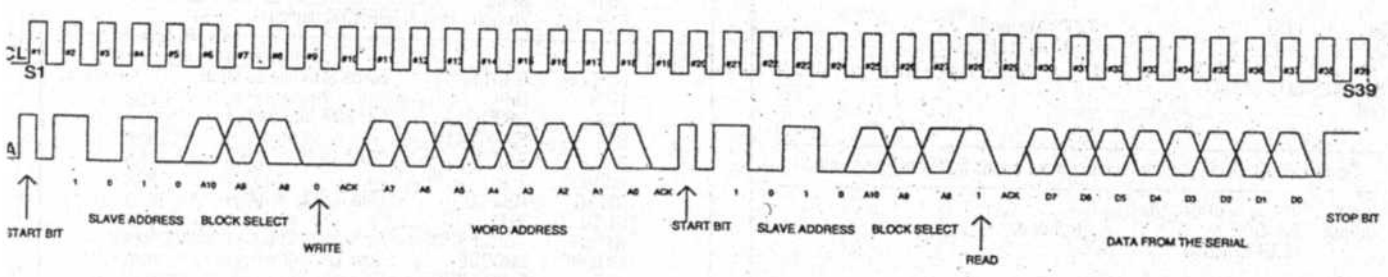


Figura 13. Diagramma per la lettura di una memoria IIC bus

matore dedicato, come per esempio quello pubblicato in aprile '95 oppure in febbraio '94 telefonando allo 0337/259730.

Per i nostri lettori che hanno invece seguito il corso sui microcontrollori PIC, proponiamo, in Tabella 2, le routine rilasciate direttamente dalla Microchip per una perfetta gestione. ■

Tabella 2. Software di gestione rilasciato dalla Microchip

▼

```
        TITLE " TWO WIRE/I2C BUS INTERFACE WITH PIC16C5x "
;
        LIST P=16C54
;*****************************************************************
;** Two wire/I2C Bus READ/WRITE Sample Routines of Microchip's
;** 24Cxx / 85Cxx serial CMOS EEPROM interfacing to a
;** PIC16C54 8-bit CMOS single chip microcomputer
;**         Part use = PIC16C54-XT/JW
;** Note: 1) All timings are based on a reference crystal frequency of 2MHz
;**          which is equivalent to an instruction cycle time of 2 usec.
;**       2) Address and literal values are read in octal unless otherwise
;**          specified.
;*****************************************************************
;
;-----------------------------------------------------------------
;       Files Assignment
;-----------------------------------------------------------------
;
PC      EQU  2          ; Program counter
STAT    EQU  3          ; PIC status byte
FSR     EQU  4          ; File Select Register
RA      EQU  5          ; Port A use to select device address
RB      EQU  6          ; RB7 = SDA, RB6 = SCL
;
STATUS  EQU  08         ; Status register
FLAG    EQU  09         ; Common flag bits register
EEPROM  EQU  0A         ; Bit buffer
ERCODE  EQU  0B         ; Error code (to indicate bus status)
ADDR    EQU  10         ; Address register
DATAI   EQU  11         ; Stored data input register
DATAO   EQU  12         ; Stored data output register
SLAVE   EQU  13         ; Device address (1010xxx0)
TXBUF   EQU  14         ; TX buffer
RXBUF   EQU  15         ; RX buffer
COUNT   EQU  16         ; Bit counter
TIMER0  EQU  18         ; Delay timer0
TIMER1  EQU  19         ; Delay timer1
;-----------------------------------------------------------------
;       Bit Assignments
;-----------------------------------------------------------------
;
; Status bits
;
Z       EQU  2
C       EQU  0
;
; FLAG Bits
;
ERROR   EQU  0          ; Error flag
;
; EEPROM Bits
;
DI      EQU  7          ; EEPROM input
DO      EQU  6          ; EEPROM output
;
; I2C Device Bits
;
SDA     EQU  7          ; RB7, data in/out
SCL     EQU  6          ; RB6, serial clock
;
;END FILES/BITS EQUATE
 PAGE
;
;-----------------------------------------------------------------
;   Two wire/I2C - CPU communication error status table and subroutine
;-----------------------------------------------------------------
;  input  : W-reg  = error code
;  output : ERCODE          = error code
;           FLAG(ERROR)     = 1
;
;           code   error status mode
;          ------  ------------------------------------------------
;           1   :  SCL locked low by device (bus is still busy)
;           2   :  SDA locked low by device (bus is still busy)
;           3   :  No acknowledge from device (no handshake)
;           4   :  SDA bus not released for master to generate STOP bit
;-----------------------------------------------------------------
;
;Subroutine to identify the status of the serial clock (SCL) and serial data
;(SDA) condition according to the error status table. Codes generated are
;useful for bus/device diagnosis.
;
ERR
        BTFSS   FLAG,ERROR  ; Remain as first error encountered
        MOVWF   ERCODE      ; Save error code
        BSF     FLAG,ERROR  ; Set error flag
        RETLW   0
;
;-----------------------------------------------------------------
;       START bus communication routine
;-----------------------------------------------------------------
;       input       : none
;       output  : initialize bus communication
;-----------------------------------------------------------------
;
;Generate START bit (SCL is high while SDA goes from high to low transition)
;and check status of the serial clock.
BSTART
        MOVLW   B'00111111' ; Put SCL, SDA line in output state
        TRIS    RB
;*****************************************************************
        bsf     RB,SDA      ;make sure sda is high
;*****************************************************************
        BSF     RB,SCL      ; Set clock high
        MOVLW   1           ; Ready error status code 1
        BTFSS   RB,SCL      ; Locked?
        CALL    ERR         ; SCL locked low by device
        BCF     RB,SDA      ; SDA goes low during SCL high
        NOP                 ; Timing adjustment
        NOP
        NOP
        BCF     RB,SCL      ; Start clock train
        RETLW   0
;-----------------------------------------------------------------
;       STOP bus communication routine
;-----------------------------------------------------------------
;       Input   :   None
;       Output  :   Bus communication, STOP condition
;-----------------------------------------------------------------
;
;Generate STOP bit (SDA goes from low to high during SCL high state)
;and check bus conditions.
;
BSTOP
;*****************************************************************
        MOVLW   B'00111111' ; Put SCL, SDA line in output state
        TRIS    RB
;*****************************************************************
        BCF     RB,SDA      ; Return SDA to low
        BSF     RB,SCL      ; Set SCL high
        nop
        nop
        nop
        MOVLW   1           ; Ready error code 1
        BTFSS   RB,SCL      ; High?
        CALL    ERR         ; No, SCL locked low by device
        BSF     RB,SDA      ; SDA goes from low to high during SCL high
        MOVLW   4           ; Ready error code 4
        BTFSS   RB,SDA      ; High?
        CALL    ERR         ; No, SDA bus not release for STOP
        RETLW   0
;-----------------------------------------------------------------
;       Serial data send from PIC to serial EEPROM, bit-by-bit subroutine
;-----------------------------------------------------------------
;       Input   :   None
;       Output  :   To (DI) of serial EEPROM device
;-----------------------------------------------------------------
BITIN
        MOVLW   B'10111111' ; Force SDA line as input
        TRIS    RB
        BSF     RB,SDA      ; Set SDA for input
        BCF     EEPROM,DI
        BSF     RB,SCL      ; Clock high
        MOVLW   1
        BTFSC   RB,SCL      ; Skip if SCL  is high
        GOTO    BIT1
        BTFSS   FLAG,ERROR  ; Remain as first error encountered
        MOVWF   ERCODE      ; Save error code
        BSF     FLAG,ERROR  ; Set error flag
BIT1
        BTFSC   RB,SDA      ; Read SDA pin
```

```
        BSF     EEPROM,DI       ; DI = 1
        NOP                     ; Delay
        BCF     RB,SCL          ; Return SCL to low
        RETLW   0
;-------------------------------------------------------------------------
;       Serial data receive from serial EEPROM to PIC, bit-by-bit subroutine
;-------------------------------------------------------------------------
;       Input   :   EEPROM file
;       Output  :   From (DO) of serial EEPROM device to PIC
;-------------------------------------------------------------------------
BITOUT
        MOVLW   B'00111111'     ; Set SDA, SCL as outputs
        TRIS    RB
        BTFSS   EEPROM,DO
        GOTO    BIT0
        BSF     RB,SDA          ; Output bit 0
        MOVLW   2
        BTFSC   RB,SDA          ; Check for error code 2
        GOTO    CLK1
        BTFSS   FLAG,ERROR      ; Remain as first error encountered
        MOVWF   ERCODE          ; Save error code
        BSF     FLAG,ERROR      ; Set error flag
        GOTO    CLK1            ; SDA locked low by device
BIT0
        BCF     RB,SDA          ; Output bit 0
        NOP                     ; Delay
        NOP
        NOP
CLK1
        BSF     RB,SCL
        MOVLW   1               ; Error code 1
        BTFSC   RB,SCL          ; SCL locked low?
        GOTO    BIT2            ; No.
        BTFSS   FLAG,ERROR      ; Yes.
        MOVWF   ERCODE          ; Save error code
        BSF     FLAG,ERROR      ; Set error flag
BIT2
        NOP
        NOP
        BCF     RB,SCL          ; Return SCL to low
        RETLW   0
;-------------------------------------------------------------------------
;       RECEIVE   DATA          subroutine
;-------------------------------------------------------------------------
;       Input   :   None
;       Output  :   RXBUF = Receive 8-bit data
;-------------------------------------------------------------------------
RX
        MOVLW   .8              ; 8 bits of data
        MOVWF   COUNT
        CLRF    RXBUF
;
RXLP
        RLF     RXBUF           ; Shift data to buffer
        SKPC
        BCF     RXBUF,0         ; carry ---> f(0)
        SKPNC
        BSF     RXBUF,0
        CALL    BITIN
        BTFSC   EEPROM,DI
        BSF     RXBUF,0         ; Input bit =1
        DECFSZ  COUNT           ; 8 bits?
        GOTO    RXLP
        BSF     EEPROM,DO       ; Set acknowledge bit = 1
        CALL    BITOUT          ; to STOP further input
        RETLW   0
;-------------------------------------------------------------------------
;       TRANSMIT  DATA          subroutine
;-------------------------------------------------------------------------
;       Input   :   TXBUF
;       Output  :   Data X'mitted to EEPROM device
;-------------------------------------------------------------------------
TX
        MOVLW   .8
        MOVWF   COUNT
TXLP
        BCF     EEPROM,DO       ; Shift data bit out.
        BTFSC   TXBUF,7         ; If shifted bit = 0, data bit = 0
        BSF     EEPROM,DO       ; Otherwise data bit = 1
        CALL    BITOUT          ; Serial data out
        RLF     TXBUF           ; Rotate TXBUF left
        SKPC    ; f(6) ---> f(7)
        BCF     TXBUF,0         ; f(7) ---> carry
        SKPNC   ; carry ---> f(0)
        BSF     TXBUF,0
        DECFSZ  COUNT           ; 8 bits done?
        GOTO    TXLP            ; No.
        CALL    BITIN           ; Read acknowledge bit
        MOVLW   3
        BTFSC   EEPROM,DI       ; Check for acknowledgement
        CALL    ERR             ; No acknowledge from device
        RETLW   0
;-------------------------------------------------------------------------
;       BYTE-WRITE, write one byte to EEPROM device
;-------------------------------------------------------------------------
;       Input   :   DATAO= data to be written
;                   ADDR   = destination address
;                   SLAVE  = device address (1010xxx0)
;       Output  :   Data written to EEPROM device
;-------------------------------------------------------------------------
        ORG     080             ; The location for BYTE-WRITE routine can be
                                ;assigned anywhere between (377-777) octal.
WRBYTE
        MOVF    SLAVE,W         ; Get SLAVE address
        MOVWF   TXBUF           ; to TX buffer
        CALL    BSTART          ; Generate START bit
        CALL    TX              ; Output SLAVE address
        MOVF    ADDR,W          ; Get WORD address
        MOVWF   TXBUF           ; into buffer
        CALL    TX              ; Output WORD address
        MOVF    DATAO,W         ; Move DATA
        MOVWF   TXBUF           ; into buffer
        CALL    TX              ; Output DATA and detect acknowledgement
        CALL    BSTOP           ; Generate STOP bit
        goto    wrt_end
;-------------------------------------------------------------------------
;       BYTE-READ, read one byte from serial EEPROM device
;-------------------------------------------------------------------------
;       Input   :   ADDR        = source address
;       SLAVE                   = device address (1010xxx0)
;       Output  :   DATAI       = data read from serial EEPROM
;-------------------------------------------------------------------------
        ORG     0C0             ; The location for BYTE-READ routine can be
                                ;assigned anywhere between (377-777) octal.
RDBYTE
        MOVF    SLAVE,W         ; Move SLAVE address
        MOVWF   TXBUF           ; into buffer (R/W = 0)
        CALL    BSTART          ; Generate START bit
        CALL    TX              ; Output SLAVE address. Check ACK.
        MOVF    ADDR,W          ; Get WORD address
        MOVWF   TXBUF
        CALL    TX              ; Output WORD address. Check ACK.
        CALL    BSTART          ; START READ (if only one device is
        MOVF    SLAVE,W         ; connected to the I2C bus)
        MOVWF   TXBUF
        BSF     TXBUF,0         ; Specify READ mode (R/W = 1)
        CALL    TX              ; Output SLAVE address
        CALL    RX              ; READ in data and acknowledge
        CALL    BSTOP           ; Generate STOP bit
        MOVF    RXBUF,W         ; Save data from buffer
        MOVWF   DATAI           ; to DATAI file.
        goto    rd_end
;Test program to read and write ramdom start
start   movlw   0AE             ;set A2=A1=A0=1
        movwf   SLAVE           ; /
        movlw   2               ;set r/w loc. = 2
        movwf   ADDR            ; /
        movlw   55              ;write 55 to SEEPROM
        movwf   DATAO           ; /
        goto    WRBYTE          ;write a byte
wrt_end
        call    delay           ;wait for write
                                ;operation (internal)
        goto    RDBYTE          ;read back data
rd_end
        movlw   55              ;test if read
        xorwf   DATAI,W         ;correct?
        btfss   STAT,Z          ;yes then skip
wrong
        goto    wrong
correct
        goto    correct
;At 2.0Mhz, delay = approx. 3mS.
delay
        clrf    1F              ;clear last location
dly1
        nop
        nop
        nop
        decfsz  1F              ;reduce count
        goto    dly1            ;loop
        retlw   0

        org     0x1FF
        goto    start
        END
```