UNIVERSITÀ DEGLI STUDI DI PADOVA

FACOLTÀ DI INGEGNERIA

CORSO DI LAUREA IN INGEGNERIA MECCATRONICA

TESI DI LAUREA TRIENNALE

SCHEDA DI INTERFACCIAMENTO TRA AZIONAMENTI ELETTRICI E UN SISTEMA DI SUPERVISIONE

Relatore: Ch.mo Prof. SIMONE BUSO

Laureando: GABRIELE VIERO

Matricola 563771-IMC

ANNO ACCADEMICO 2011-2012

Sommario

Con il crescere dello sviluppo e il bisogno di strutture più ampie e articolate, soprattutto nel settore industriale, si è arrivati alla necessità di possedere più di un ingresso e/o più di un' uscita, quindi un aumento del numero di cancelli automatici nella stessa struttura o proprietà.
Questo porta ad un cablaggio importante per poter comandare dalla stessa posizione ogni cancello presente, con conseguente aumento del costo dei lavori per far passare tutti i cavi necessari al collegamento.
Si è quindi pensato di creare un sistema che rendesse i cancelli automatici capaci di comunicare tra loro tramite protocollo di comunicazione Modbus, in serie, riducendo di fatto il numero di cavi necessari. Infatti con questa soluzione basta connettere i cancelli tra loro con un solo cavo, ed è sufficiente che un solo cancello sia collegato alla stazione di comando.
La scheda in progetto possiede delle porte di comunicazione seriale, in particolare RS232 con cui comunica internamente con altre schede vicine, e RS485 con cui può interfacciarsi con schede e altri sistemi di supervisione esterni, molto più lontani. Incorpora un microprocessore che supporta la gestione contemporanea di due porte di comunicazione.

Indice

1. INTRODUZIONE 4
1.1 Presentazione azienda....4
1.2 Presentazione del progetto e obbiettivi....6
1.21 Scelte progettuali....7
2. CORPO DELL'ELABORATO 8
2.1 Panoramica del prodotto....8
2.11 Caratteristiche Microchip....8
2.2 Generalità del progetto....11
2.21 Descrizione schema generale....11
2.3 Fase Iniziale: Apprendimento, esercitazioni....12
2.31 Studio dei Timer....12
2.32 Codice programma anti-rimbalzo....14
2.33 Descrizione Codice anti-rimbalzo....15
2.4 Fase di studio: IL PROTOCOLLO MODBUS....15
2.41 Formato dei messaggi....16
2.42 L'indirizzo....16
2.43 Il codice funzione....17
2.44 Il CRC16....17
2.45 Diagramma di flusso del CRC....18
2.46 Sincronizzazione dei messaggi....19
2.47 La gestione degli errori....19
2.48 Diagramma di flusso Modbus (slave)....20
2.5 Fase Software: Implementazione del codice....21
2.51 Codice shift register....21
2.52 Gestione del Buffer di memoria....23
2.53 Interfacce seriali: UART0 e UART2....23
2.54 Impostazione del bit rate....24
2.55 Codice implementato Buffer RS232 e RS485....25
2.56 Descrizione del codice buffer RS232 e RS485....27
2.57 Approfondimento: porta RS232 e RS485....28
2.6 Fase Hardware: Realizzazione scheda SIS....31
2.61 Collegamento scheda-pc. Test di trasmissione....32
2.62 Primo testo spedito con SIS....32
2.63 Collegamento scheda-scheda. Test di trasmissione....34
2.64 Spiegazione codice implementato....37
2.65 Caratteristiche del segnale trasmesso....39

3. CONCLUSIONI 40
3.1 In futuro... 40

Ringraziamenti 41
Bibliografia 41
Appendici 42

1. INTRODUZIONE

1.1 Presentazione azienda

Fig.1.1 Facciata dell'azienda Benincà.

Benincà Holding è un gruppo costituito da 5 aziende: Automatismi Benincà (fig. 1.1), Automatismi Cab, Byou, Hi Motions e Rise. Il gruppo porta con sé la storicità del marchio Benincà brand di riferimento nel panorama internazionale che da oltre 30 anni progetta e vende in tutto il mondo sistemi per l'automazione di porte e cancelli ad uso residenziale e industriale. Il 2009, anno del trentesimo anniversario del brand Benincà, è stato anche l'anno della scelta strategica: accanto alla nascita di 3 nuovi brand, Byou produttrice di sistemi di automazione in kit ad uso residenziale, Hi Motions produttrice di accessori meccanici per cancelli, Rise produttrice di dissuasori di sosta, le 5 realtà si concretizzano in una holding che fa della specializzazione e del know how i suoi punti di forza indiscussi in grado di offrire soluzioni tecnologiche per la messa in sicurezza di ambienti residenziali ed industriali.
Automatismi Benincà, presso la quale ho svolto il tirocinio, è situata a Sandrigo (VI).
Attraverso precise scelte commerciali, di marketing e di prodotto negli ultimi dieci anni ha saputo costruire su se stessa una storia nuova, un percorso che ha portato il brand ad essere uno dei cinque riferimenti nel mercato dell'automazione e a raggiungere un obiettivo di internazionalizzazione attraverso 108 paesi coperti e 11 filiali.

Questi i prodotti principali:

Fig.1.2 Azionamento per cancelli ad ante battenti.

BOB (fig.1.2), la gamma di automazioni caratterizzata da design e prestazioni uniche è in grado di soddisfare le diverse esigenze del mercato. L'applicazione di un'esclusiva coppia di ingranaggi consentono di azzerare completamente la rumorosità e le vibrazioni del motore facilitando al contempo la manutenzione da parte dell'installatore. I vantaggi di questa applicazione patent pending sono non solo l'azzeramento della rumorosità, ma anche l'aumento della vita stessa degli ingranaggi.

Fig. 1.3 Telecomando per cancelli automatici.

IO (fig.1.3), il nuovo trasmettitore rolling code dalle dimensioni ridotte disponibile a due canali.

Fig. 1.4 Centralina per il comando da cellulari.

CALL (fig.1.4), il nuovo accessorio GSM per il comando dell'automatismo attraverso l'utilizzo di un normale cellulare.

Fig. 1.5 Sistema di alimentazione da pannelli fotovoltaici.

SUN SYSTEM (fig.1.5), il nuovo dispositivo per il funzionamento dell'automatismo attraverso pannello fotovoltaico.

Fig. 1.6 Fotocellule alimentate da pannello fotovoltaico.

PUPILLA.B (fig.1.6), coppia di fotocellule con trasmettitore alimentato a batteria.
Il sistema di ricarica della batteria tramite pannello fotovoltaico consente di ottenere un funzionamento continuo senza necessità di manutenzione e senza rischi di blocco dell'impianto a seguito dell'esaurimento della batteria stessa. La facilità d'installazione è garantita dall'assenza di collegamenti elettrici e dalle dimensioni ridotte della fotocellula.

Fig. 1.7 LAMPI.LED, lampeggiante a led.

LAMPI.LED (fig. 1.7), lampeggiante a led disponibile a 230 Vac e a 24 Vdc. Caratterizzato da un basso assorbimento, consente una durata superiore, circa 80/100.000 ore contro le 7000 del lampeggiante a lampadina.

1.2 Presentazione del progetto e obbiettivi

Nella logica di risparmio sia di costi che di tempo di installazione dei cancelli automatici, l'azienda Benincà mi ha chiesto di progettare una scheda che avesse proprio tali fini.

Il mio lavoro è stato pertanto quello di progettare questa nuova scheda, sia dal punto di vista hardware che software.
Ho deciso di utilizzare una scheda a microprocessore per la comunicazione seriale, basata sulla porta RS485, utile al controllo remoto di motori in corrente continua e/o corrente alternata.
La scheda è in grado di interfacciarsi con scheda comando motori di un cancello elettrico, e comunicare con periferiche esterne utilizzando il protocollo modbus.
Il processore utilizzato fa parte della famiglia dei microchips Renesas.

1.21 Scelte progettuali

Si è scelto di utilizzare il microchips Renesas R8C/32 perché ha la possibilità di gestire due comunicazioni seriali contemporaneamente ad una buona velocità. In più offre il miglior rapporto prestazioni/prezzo.
La porta RS485 permette di raggiungere distanze elevate (1.2 km), poiché è un mezzo di trasmissione fisico che si basa su un segnale differenziale, quindi risulta maggiormente immune ai disturbi rispetto ad altri protocolli fisici di trasmissione.
Questa esperienza si suddivide principalmente in:

- fase iniziale, in cui prendo confidenza con il progetto con esercitazioni standard per conoscere ed apprendere il funzionamento del microchip della scheda;
- fase di studio, dove faccio conoscenza del protocollo di comunicazione Modbus e ne comprendo i meccanismi fondamentali per poterlo implementare via software;
- fase Software, nella quale sviluppo il codice necessario per il funzionamento della scheda;
- fase Hardware, dove realizzo la scheda, eseguo dei test di trasmissione, e arrivo alle conclusioni.

2. CORPO DELL'ELABORATO

2.1 Panoramica del prodotto

Il microchip R8C/32C è basato sull'R8C CPU Core. La massima frequenza operativa è di 20 MHz. All'interno è disponibile anche una Flash Memory. Questa memoria è programmabile su una sorgente singola di alimentazione.

2.11 Caratteristiche Microchip

Il gruppo R8C/32C formato da singoli chip MCU (MicroController Unit), incorpora le cpu R8C, in cui sono presenti istruzioni sofisticate per un alto livello di efficienza. Con un 1 Mbyte di spazio di allocazione è capace di eseguire istruzioni ad alta velocità. In più il cuore della CPU supporta un moltiplicatore per processare le operazioni ad alta velocità.
Il consumo di potenza è basso e le supportate modalità di operare consentono un maggiore controllo della potenza. Queste MCU sono progettate per massimizzare le performance di EMI/EMS (ElectroMagnetic Interferance/ElectroMagnetic Signal).
L'integrazione di molte funzioni periferiche, includendo timer multifunzionali e un'interfaccia seriale, riduce il numero di componenti di sistema.
Il gruppo R8C/32C ha una memoria dati flash (1 KB × 4 blocchi) con la funzione di svolgere operazioni in background.

Applicazioni

Elettrodomestici, attrezzature per ufficio, apparecchiature audio, apparecchiature di consumo, etc.

Caratteristiche Specifiche del microchip R8C/32C:

Tabella 2.1 Dati del Microchip (1), da Renesas R8C/32C Hardware Manual

Item	Function	Specification
CPU	Central processing unit	R8C CPU core • Number of fundamental instructions: 89 • Minimum instruction execution time: 50 ns (f(XIN) = 20 MHz, VCC = 2.7 to 5.5 V) 200 ns (f(XIN) = 5 MHz, VCC = 1.8 to 5.5 V) • Multiplier: 16 bits × 16 bits → 32 bits • Multiply-accumulate instruction: 16 bits × 16 bits + 32 bits → 32 bits • Operation mode: Single-chip mode (address space: 1 Mbyte)
Memory	ROM, RAM, Data flash	Refer to **Table 1.3 Product List for** R8C/32C Group.
Power Supply Voltage Detection	Voltage detection circuit	• Power-on reset • Voltage detection 3 (detection level of voltage detection 0 and voltage detection 1 selectable)
I/O Ports	Programmable I/O ports	• Input-only: 1 pin • CMOS I/O ports: 15, selectable pull-up resistor • High current drive ports: 15
Clock	Clock generation circuits	4 circuits: XIN clock oscillation circuit, XCIN clock oscillation circuit (32 kHz) High-speed on-chip oscillator (with frequency adjustment function), Low-speed on-chip oscillator, • Oscillation stop detection: XIN clock oscillation stop detection function • Frequency divider circuit: Dividing selectable 1, 2, 4, 8, and 16 • Low power consumption modes: Standard operating mode (high-speed clock, low-speed clock, high-speed on-chip oscillator, low-speed on-chip oscillator), wait mode, stop mode
		Real-time clock (timer RE)
Interrupts		• Number of interrupt vectors: 69 • External Interrupt: 7 ($\overline{\text{INT}}$ × 3, Key input × 4) • Priority levels: 7 levels
Watchdog Timer		• 14 bits × 1 (with prescaler) • Reset start selectable • Low-speed on-chip oscillator for watchdog timer selectable
DTC (Data Transfer Controller)		• 1 channel • Activation sources: 21 • Transfer modes: 2 (normal mode, repeat mode)

Tabella 2.2 Caratteristiche timer, da Renesas R8C/32 Hardware Manual

Timer	Timer RA	8 bits × 1 (with 8-bit prescaler) Timer mode (period timer), pulse output mode (output level inverted every period), event counter mode, pulse width measurement mode, pulse period measurement mode
	Timer RB	8 bits × 1 (with 8-bit prescaler) Timer mode (period timer), programmable waveform generation mode (PWM output), programmable one-shot generation mode, programmable wait one-shot generation mode
	Timer RC	16 bits × 1 (with 4 capture/compare registers) Timer mode (input capture function, output compare function), PWM mode (output 3 pins), PWM2 mode (PWM output pin)
	Timer RE	8 bits × 1 Real-time clock mode (count seconds, minutes, hours, days of week), output compare mode

- 8-bit Multifunction Timer with 8-bit prescaler (Timer RA and RB): 2
- 16-bit Input Capture/Output Compare Timer (Timer RC): 1
- 8-bit Real-Time Clock Timer with compare match function (Timer RE): 1
- UART/Clock Synchronous Serial Interface: 2 channels
- I^2C-bus Interface (IIC)/Synchronous Serial Communication Unit: 1 channel
- LIN Module: 1 channel (Timer RA, UART0)
- 10-bit A/D Converter: 4 channels
- Comparator: 2
- DTC (Data Transfer Controller): 1 channel

- Watchdog Timer
- Clock Generating Circuits: XIN Clock Generation Circuit, On-chip Oscillator (High/Low Speed), Sub Clock Generation Circuit
- Oscillation Stop Detection Function
- Voltage Detection Circuit
- Power-on Reset Circuit
- I/O Ports: 15 (incl. LED drive ports)
- External Interrupt Pins: 7
- Data Flash with Background Operation (BGO) function: 4 KB

Tabella 2.3 Dati del Microchip (2), da Renesas R8C/32 Hardware Manual

Item	Function	Specification
Serial Interface	UART0	Clock synchronous serial I/O/UART
	UART2	Clock synchronous serial I/O/UART, I^2C mode (I^2C-bus), multiprocessor communication function
Synchronous Serial Communication Unit (SSU)		1 (shared with I^2C-bus)
I^2C bus		1 (shared with SSU)
LIN Module		Hardware LIN: 1 (timer RA, UART0)
A/D Converter		10-bit resolution × 4 channels, includes sample and hold function, with sweep mode
Comparator B		2 circuits
Flash Memory		• Programming and erasure voltage: VCC = 2.7 to 5.5 V • Programming and erasure endurance: 10,000 times (data flash) 1,000 times (program ROM) • Program security: ROM code protect, ID code check • Debug functions: On-chip debug, on-board flash rewrite function • Background operation (BGO) function
Operating Frequency/Supply Voltage		f(XIN) = 20 MHz (VCC = 2.7 to 5.5 V) f(XIN) = 5 MHz (VCC = 1.8 to 5.5 V)
Current consumption		Typ. 6.5 mA (VCC = 5.0 V, f(XIN) = 20 MHz) Typ. 3.5 mA (VCC = 3.0 V, f(XIN) = 10 MHz) Typ. 3.5 μA (VCC = 3.0 V, wait mode (f(XCIN) = 32 kHz)) Typ. 2.0 μA (VCC = 3.0 V, stop mode)
Operating Ambient Temperature		-20 to 85°C (N version) -40 to 85°C (D version) (1)
Package		20-pin LSSOP Package code: PLSP0020JB-A (previous code: 20P2F-A)

2.2 Generalità del progetto

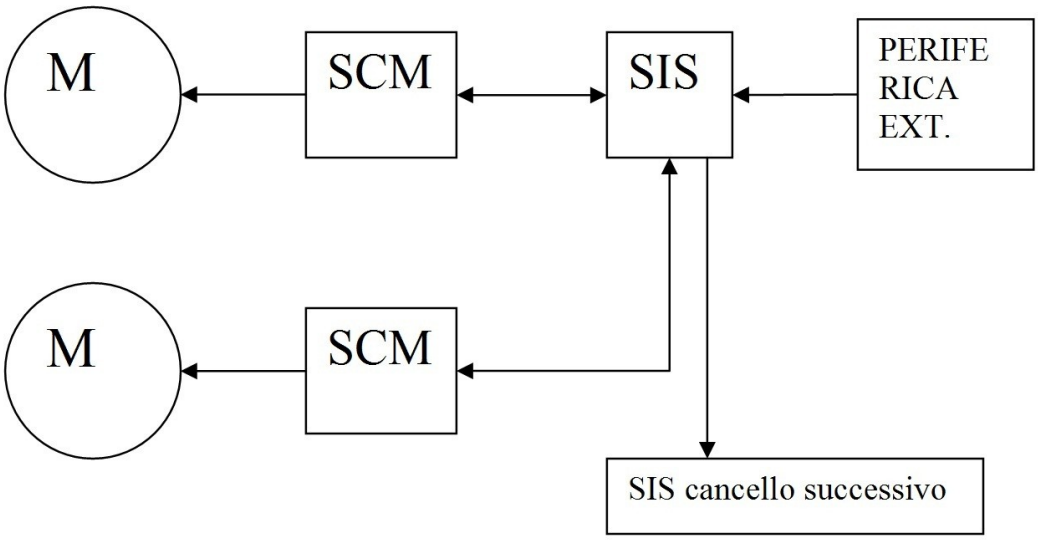

Fig. 2.1 Diagramma a blocchi generale del progetto in un singolo cancello.

Legenda: M = motore ac/dc;
SCM = scheda comando motore;
SIS = Sistema di Interfacciamento Seriale;
PERIFERICA EXT.=pc, sistema di supervisione ecc.

2.21 Descrizione schema generale

La figura 2.1 rappresenta un diagramma a blocchi del controllo di un cancello automatico ad ante battenti.
In questo esempio, si considera un cancello a due ante, fornito di un motore elettrico per ogni anta.
Ogni motore presenta una scheda opportuna per il suo comando, la quale si occupa di fornire anche la potenza necessaria ad azionare il motore.
La scheda SIS invece si occupa di inviare i comandi alle schede SCM.
Essa permette di essere comandata anche da periferiche esterne come pc, di monitorare i dati della posizione delle ante, di risolvere eventuali problemi di software e di effettuare aggiornamenti da remoto.

Schema con 2 o più cancelli.

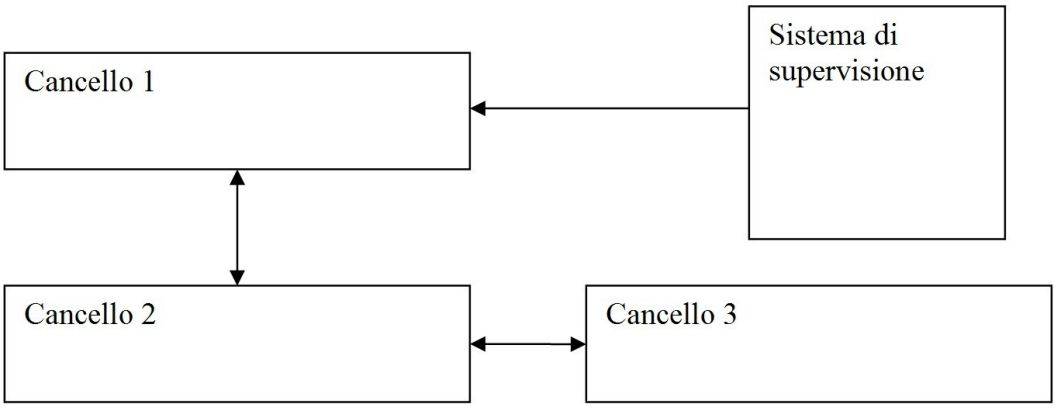

Fig. 2.2 Schema generale a più cancelli, concetto dello scopo del progetto.

Descrizione

Questo modello in fig. 2.2 rappresenta in maniera più generale quel che è lo scopo del progetto. Si vedono infatti i cancelli che sono in comunicazione seriale tra loro, comandati e controllati da un unico sistema esterno che può osservare lo stato di ciascun cancello e dare ordini singolarmente. Questo avviene grazie all'implementazione del protocollo seriale Modbus.
Le schede SIS comunicano tra loro attraverso la porta seriale RS485 e via RS232 con le SCM.

2.3 Fase Iniziale: Apprendimento, esercitazioni

2.31 Studio dei Timer

Nella fase iniziale ho studiato i Timer presenti nei microchips Renesas, ed eseguito delle prove con schedine già esistenti. Usando il linguaggio C, ho costruito un codice per poter impostare la frequenza dell'oscillatore e comandare dei led presenti nella scheda; riuscire a farli accendere e spegnere con una data frequenza, combinazione di colori, accensione e spegnimento led soffuso. Per fare questo nel programma del microchip si andava ad agire in alcuni registri specifici riguardo la frequenza, l'abilitazione di alcune modalità. In seguito riporto una parte di codice che mostra alcune configurazioni possibili.

Nel diagramma in fig. 2.3 sono riportati i blocchi che costituiscono il timer RA.
Ho messo in evidenza due parti: la prima riguarda la scelta della frequenza del clock principale su cui si basa il circuito del timer; la seconda, riguarda due registri a 8 bit ciascuno, con i quali poter selezionare l'effettiva frequenza con cui far contare il timer. Sono i registri TRAPRE e TRA che dividono la frequenza di partenza del circuito ottenendo quella desiderata.

La formula per ottenere la base dei tempi voluta è: F/((n+1)*(m+1)).

F=frequenza di partenza del circuito;
n=numero decimale, da inserire in forma binaria nel registro TRAPRE;
m=numero decimale, da inserire in forma binaria nel registro TRA.

Ad esempio, se vogliamo ottenere una F=1kHz, partendo da una frequenza iniziale di 2,5MHz dovrò dividere per il numero 2500, 2.5MHz/2500=1KHz.
Quindi (n+1)*(m+1)=2500. Scelgo n=99, m=24 e li traduco in due numeri binari:
n=TRAPRE=99 (decimale)=01100011 (binario)=0x63h (esadecimale)
m=TRA=24 (decimale)=00011000 (binario)=0x18h (esadecimale)

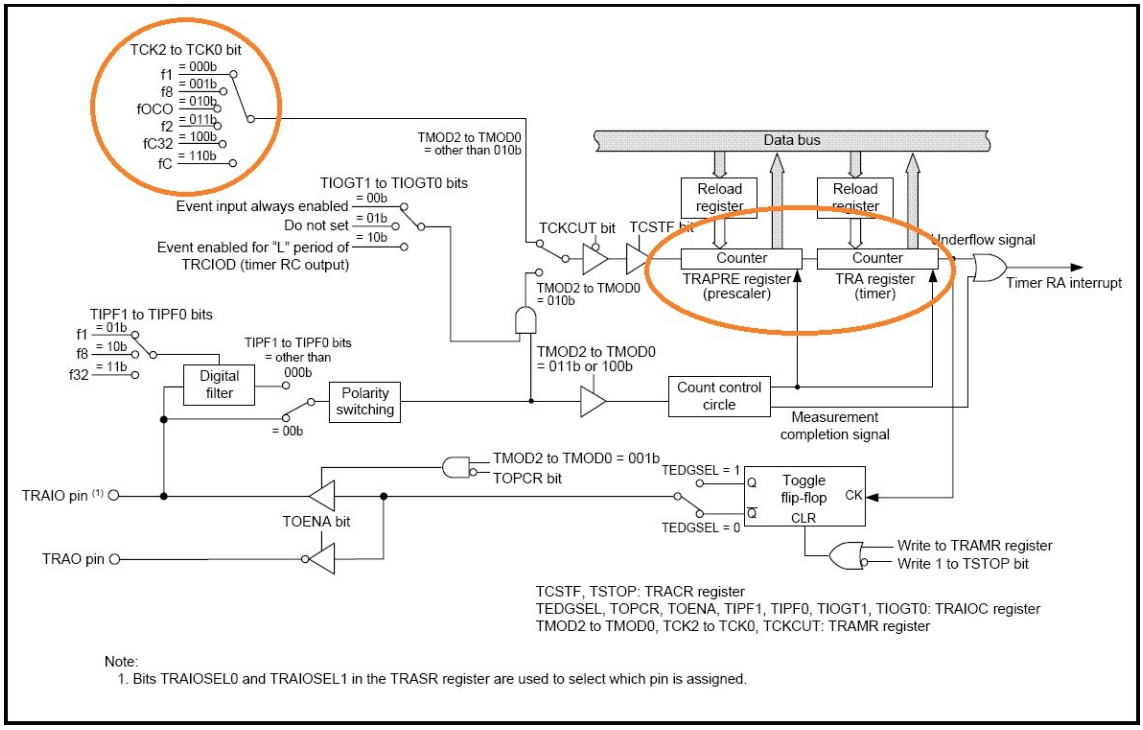

Fig. 2.3 Diagramma a blocchi del Timer RA, da Renesas R8C/32 Hardware Manual

esempio di impostazione del timer:
TIMER RA – 1ms
tramr=0x10; Fosc=F/8
trapre=0x18; Fosc/(25*100)=1Khz-->1ms
tra=0x63;

Nota: i valori preceduti da 0x...sono da intendersi in esadecimale.

Legenda:
Tramr = Timer RA Mode Register;
Trapre = Timer RA prescaler;
Tra = timer RA;
Fosc = frequenza di oscillazione del sistema.

Un esempio di esercitazione è il seguente:

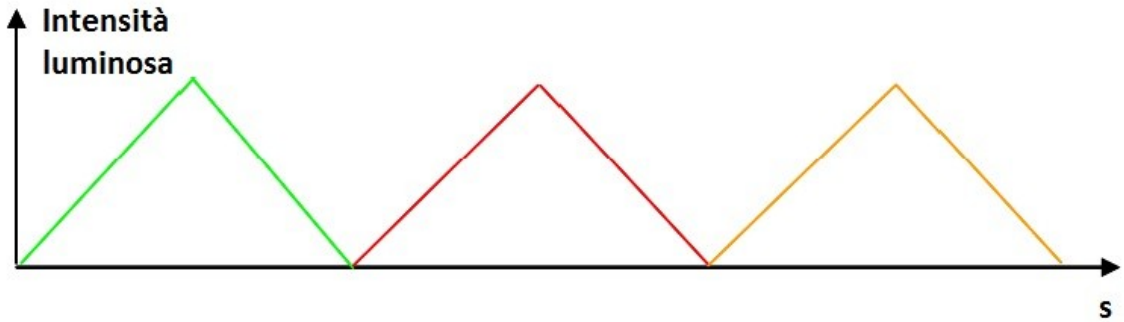

Fig. 2.4 Diagramma indicativo accensione/spegnimento led.

La figura 2.4 mostra accensione e spegnimento in modo graduale e consecutivo i led di luce verde, rosso, e arancione (dato dalla combinazione del verde e il rosso). Per fare questo bisogna innanzitutto impostare il funzionamento del microchip, attivandone i timer e le porte necessarie.

2.32 Codice programma anti-rimbalzo

Quando si preme un pulsante, essendo esso costituito da un meccanismo a molla che mette in contatto due lamelle (che chiudono il contatto), le lamelle continueranno a vibrare (**rimbalzare**) per alcuni millisecondi andando in on/off centinaia, migliaia di volte nell'arco di un tempo brevissimo: in queste condizioni, essendo il nostro programma molto veloce, rileverebbe continuamente queste transizioni on/off, accendendo e spegnendo il led migliaia di volte al secondo, con il risultato che, alla fine di questo "trambusto" non si saprà se il led rimarrà acceso o spento (si verifica quindi una situazione random). Per tale motivo viene inserito un piccolo ritardo software il quale ci eviterà di ricontrollare lo stato del pulsante prima che i rimbalzi siano finiti. Spesso al posto del termine antirimbalzo si utilizza il termine anglosassone **antibounce**.
Generalmente un antirimbalzo di 100 millisecondi è *sufficiente* (dipende comunque molto dalla qualità, dal meccanismo e dallo stato di usura del pulsante), molto spesso conviene aumentare tale ritardo per lavorare in sicurezza, oppure ricorrere ad espedienti hardware per evitare i rimbalzi del pulsante.

Per evitare il cambiamento continuo e involontario dello stato logico, ad esempio di un ingresso a tasto, dovuto alle piccole vibrazioni fisiche (impercettibili dall'uomo, ma ben presenti per il microchips) dell'interruttore, si utilizza una soluzione software. In figura 2.5 è mostrato il problema del rimbalzo, senza e con antirimbalzo software:

Fig. 2.5 Diagramma che evidenzia il problema del rimbalzo della tensione.

Di seguito è riportato il codice risolutivo.

```
void gestione_ingressi(void)
{
 //antirimbalzo tasto S1
 if(!pin_tasto)
 {
  if(cont_tasto==ANTIRIMBALZO_INGRESSI)
  {
   tasto=1;
   cont_tasto=255;
  }
```

```
   else if(cont_tasto<ANTIRIMBALZO_INGRESSI) cont_tasto++;
  }
  else
  {
   cont_tasto=0;
  }
}
```

2.33 Descrizione Codice anti-rimbalzo

Il programma si attiva con la pressione del tasto che manda a livello logico basso il "pin_tasto"; si entra così nella condizione "if (!pin_tasto)". Nel passaggio successivo si esegue un confronto tra una variabile utilizzata come temporizzatore, "cont_tasto", e una costante "ANTIRIMBALZO_INGRESSI" fissata in una parte di codice a parte. Quando il cont_tasto raggiungerà lo stesso valore della costante di antirimbalzo si presume che l'interruttore fisico abbia smesso di vibrare permettendo al livello logico di rimanere stabile. Una volta che si verifica questo, il led viene acceso (tasto=1). Altrimenti il confronto continua tra la variabile cont_tasto che ad ogni ciclo viene incrementata (cont_tasto++), e la costante antirimbalzo fintanto che non si sarà verificata l'uguaglianza cont_tasto== ANTIRIMBALZO_INGRESSI. Nel caso in cui non si verifichi più la condizione di tasto premuto si procede con lo spegnimento del led e l'azzeramento del contatore.

2.4 Fase di studio: IL PROTOCOLLO MODBUS

II protocollo MODBUS definisce il formato e la modalità di comunicazione tra un "master" che gestisce il sistema e uno o più "slave" che rispondono alle interrogazioni del master.
II protocollo definisce come il master e gli slave stabiliscono ed interrompono la comunicazione, come trasmettitore e ricevitore devono essere identificati, come i messaggi devono venire scambiati e come gli errori rilevati.
Le principali ragioni della scelta di questo protocollo rispetto ad altri sono:

1. È un protocollo pubblicato apertamente e royalty-free.
2. Può essere implementato in pochi giorni, non in mesi.
3. Muove raw bits e words senza porre molte restrizioni ai produttori.

Si possono connettere un master e fino a 247 slave su una linea comune , occorre notare che questo è un limite logico del protocollo, l'interfaccia fisica può peraltro limitare ulteriormente il numero di dispositivi, per esempio l'interfaccia standard RS-485 prevede un massimo di 31 slave connessi alla linea.
Sostituendo l'ultimo elemento della linea con un apposito "bridge o ripetitore", si possono connettere altri 31 slave e cosi via sino al raggiungimento del numero massimo logico di dispositivi applicati.
Solo il master può iniziare una transazione.
Una transazione può avere il formato domanda/risposta diretta ad un singolo slave o broadcast in cui il messaggio viene inviato a tutti i dispositivi sulla linea che non danno risposta.
Una transazione è composta da una struttura singola domanda/singola risposta o una struttura singolo messaggio broadcast/nessuna risposta.
Alcune caratteristiche del protocollo sono definite e sono:

- standard di interfaccia
- parità

- numero di stop bit
- ed il formato RTU (binario).

Esiste anche il protocollo MODBUS di tipo ASCII ma normalmente viene implementato il modo RTU (unità terminale remota) in quanto più efficiente. Nel MODBUS RTU gli indirizzi partono da zero (0000 = 1° indirizzo).

2.41 Formato dei messaggi

Per poter comunicare tra due dispositivi, il messaggio deve essere contenuto in un "involucro" L'involucro lascia il trasmettitore attraverso una "porta" ed è "portato" lungo la linea fino ad una analoga "porta" sul ricevitore.
MODBUS stabilisce il formato di questo involucro (mostrato in fig. 2.6) , tanto per il master che per lo slave.

Indirizzo dello slave
Funzione da svolgere
Dati
CRC16

Fig. 2.6 Struttura di un messaggio.

Comprende:
L'indirizzo del dispositivo con cui il master ha stabilito la transazione (l'indirizzo O corrisponde ad un messaggio broadcast inviato a tutti i dispositivi slave).

- II codice della funzione che deve essere o è stata eseguita.
- I dati che devono essere scambiati.
- II controllo d'errore composto secondo l'algoritmo CRC16.

Se un dispositivo individua un errore nel messaggio ricevuto (di formato, di parità o nel CRC16) il messaggio viene considerato non valido e scartato, uno slave che rilevi un errore nel messaggio quindi non eseguirà l'azione e non risponderà alla domanda, cosi come se l'indirizzo non corrisponde ad un dispositivo in linea.

Formato dei caratteri
Normalmente i dispositivi che adottano il protocollo MODBUS utilizzano il formato 8, N, 1
Ovvero : 8 bit di dati, senza alcun controllo di parità e con 1 bit di stop.

2.42 L'indirizzo

Come sopra menzionato, le transazioni MODBUS coinvolgono sempre il master,che gestisce la linea, ed uno slave per volta (tranne nel caso di messaggi broadcast).
Per identificare il destinatario del messaggio viene trasmesso come primo carattere un byte che contiene l'indirizzo numerico del dispositivo slave selezionato.
Ciascuno degli slave quindi avrà assegnato un diverso numero di indirizzo che lo identifica univocamente.
Gli indirizzi ammissibili sono quelli da 1 a 247, mentre l'indirizzo 0, che non può essere assegnato ad uno slave, posto in testa al messaggio trasmesso dal master indica che questo è "broadcast", cioè diretto a tutti gli slave contemporaneamente.
Possono essere trasmessi come broadcast solo messaggi che non richiedano risposta per espletare la loro funzione, quindi solo le assegnazioni.

2.43 II codice funzione

II secondo carattere del messaggio identifica la funzione che deve essere eseguita nel messaggio trasmesso dal master, cui lo slave risponde a sua volta con lo stesso codice ad indicare che la funzione è stata eseguita.
Normalmente le MODBUS più utilizzate sono quelle riportate in tabella 2.1:

Tabella 2.1 Alcune funzioni Modbus.

Funzione	Descrizione
01	Read Coil Status
02	Read Input Status
03	Read Holding Registers
04	Read Input registers
05	Force Single Coil
06	Preset Single register
07	Read Status
15	Force multiple Coils
16	Preset Multiple Registers

2.44 Il CRC16

Gli ultimi due caratteri del messaggio contengono il codice di ridondanza ciclica (Cyclic Redundancy Check) calcolato secondo l'algoritmo CRC16.
Per il calcolo di questi due caratteri il messaggio (indirizzo, codice funzione e dati scartando i bit di start, stop e l'eventuale parità) viene considerato come un unico numero binario continuo di cui il bit più significativo (MSB) viene trasmesso prima.
Il messaggio viene innanzitutto moltiplicato per 2^16 (spostato a sinistra di 16 bit) e poi diviso per 2^16+2^15+2^2+1 espresso come numero binario (1100000000000101).
Il quoziente intero viene poi scartato e il resto a 16 bit (inizializzato a FFFFh all'inizio per evitare il caso di un messaggio di soli zeri) viene aggiunto di seguito al messaggio trasmesso.
Il messaggio risultante, quando viene diviso dal dispositivo ricevente per lo stesso polinomio (2^16+2^15+2^2+1) deve dare zero come resto, se non sono intervenuti errori (il dispositivo ricevente ricalcola il CRC).
Di fatto, dato che il dispositivo che serializza i dati da trasmettere (UART) trasmette prima il bit meno significativo (LSB) anziché il MSB come dovrebbe essere per il calcolo del CRC, questo viene effettuato invertendo il polinomio.
Inoltre, dato che il MSB del polinomio influenza solo il quoziente e non il resto, questo viene eliminato rendendolo quindi 1010000000000001.
La procedura passo-passo per il calcolo del CRC16 è la seguente:

2.45 Diagramma di flusso del CRC (Fig. 2.7)

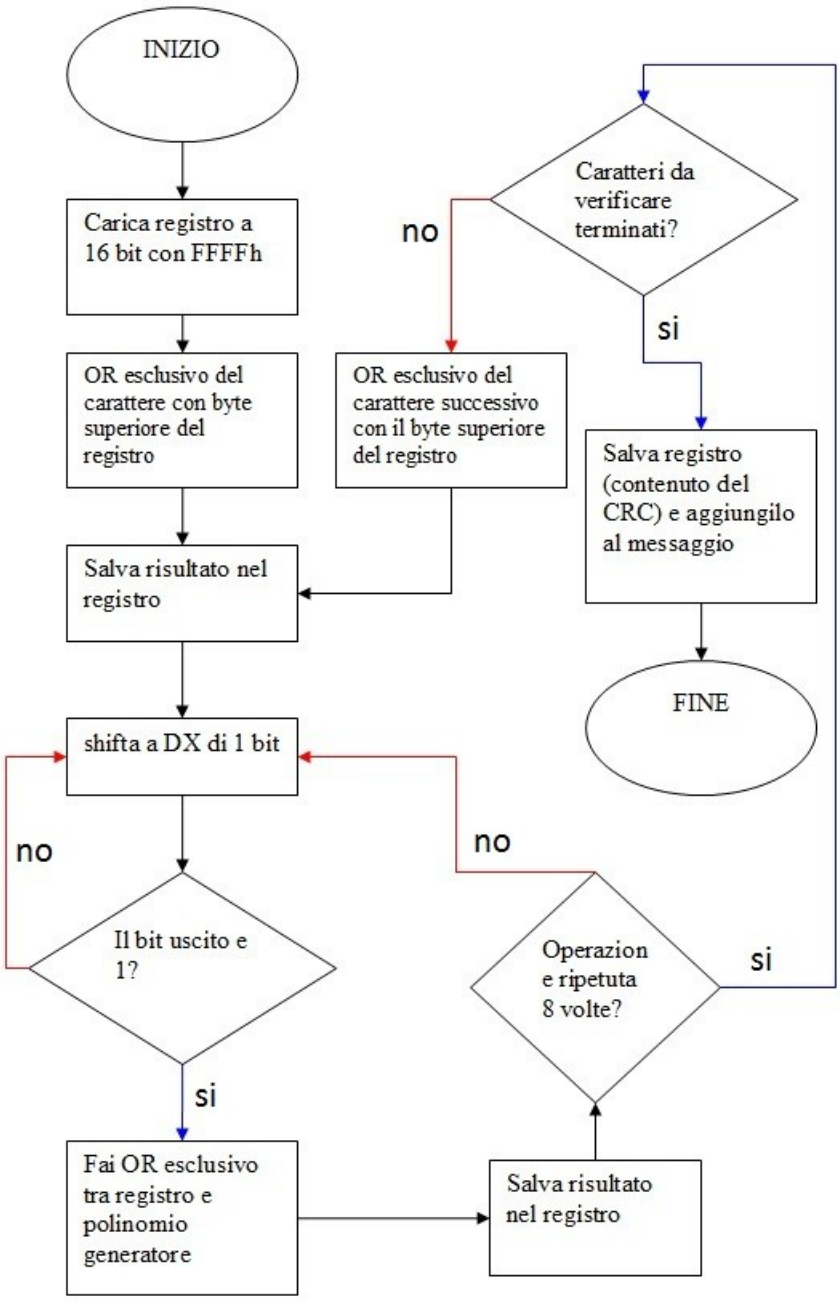

Fig. 2.7 Diagramma di flusso del CRC.

1. Caricare un registro a 16 bit con FFFFh (tutti i bit a 1).
2. Fare l'OR esclusivo del primo carattere con il byte superiore del registro , porre il risultato nel registro.
3. Spostare il registro a destra di un bit.
4. Se il bit uscito a destra dal registro (flag) è un 1, fare l'OR esclusivo del polinomio generatore 1010000000000001 con il registro.
5. Ripetere per 8 volte i passi 3 e 4.

6. Fare l'OR esclusivo del carattere successivo con il byte superiore del registro, porre il risultato nel registro.
7. Ripetere i passi da 3 a 6 per tutti i caratteri del messaggio.
8. II contenuto del registro a 16 bit è il codice di ridondanza CRC che deve essere aggiunto al messaggio.

2.46 Sincronizzazione dei messaggi

La sincronizzazione del messaggio tra trasmettitore e ricevitore viene ottenuta interponendo una pausa tra i messaggi pari ad almeno 3.5 volte il tempo di un carattere.
Se il dispositivo ricevente non riceve per un tempo di 3,5 caratteri, ritiene completato il messaggio precedente e considera che il successivo byte ricevuto sarà il primo di un nuovo messaggio e quindi un indirizzo.

2.47 La gestione degli errori

In MODBUS esistono due tipi di errori, gestiti in modo diverso: errori di trasmissione ed errori operativi. Gli errori di trasmissione sono errori che alterano il messaggio, nel suo formato, nella parità (se è usata), o nel CRC16.
Il dispositivo che rilevi errori di questo tipo nel messaggio lo considera non valido e non da risposta. Qualora invece il messaggio sia corretto nella sua forma ma la funzione richiesta, per qualsiasi motivo, non sia eseguibile, si ha un errore operativo. A questo errore il dispositivo slave risponde con un messaggio di eccezione.
Questo messaggio è composto dall'indirizzo, dal codice delta funzione richiesta, da un codice d'errore e dal CRC. Per indicare che la risposta è la notifica di un errore il codice funzione viene ritornato con il bit più significativo a"1".

Esempio di trasmissione

Se il messaggio non contiene errori il bit più significativo è 0, altrimenti è 1.
Esempio:
Messaggio ok:

0000 0011 [03h];

Errore messaggio:

1000 0011 [83h];

L'errore viene segnalato ponendo un "1" sul MSB (Most Significative Bit).
Se c'è un errore nel CRC lo slave risponde con un codice funzione che servirà al master per fargli conoscere l'azione successiva da intraprendere.
Nel campo dati risposta dello slave, in caso di errore, non compare niente. Se invece non c'è errore, nel campo dati, ci saranno i dati richiesti dal master.

Se non c'è nessuna risposta da parte dello slave entro un determinato TIMEOUT, il master agisce di conseguenza annullando la chiamata; anche chiamando un indirizzo di slave inesistente avviene un TIMEOUT.

2.48 Diagramma di flusso Modbus (slave)

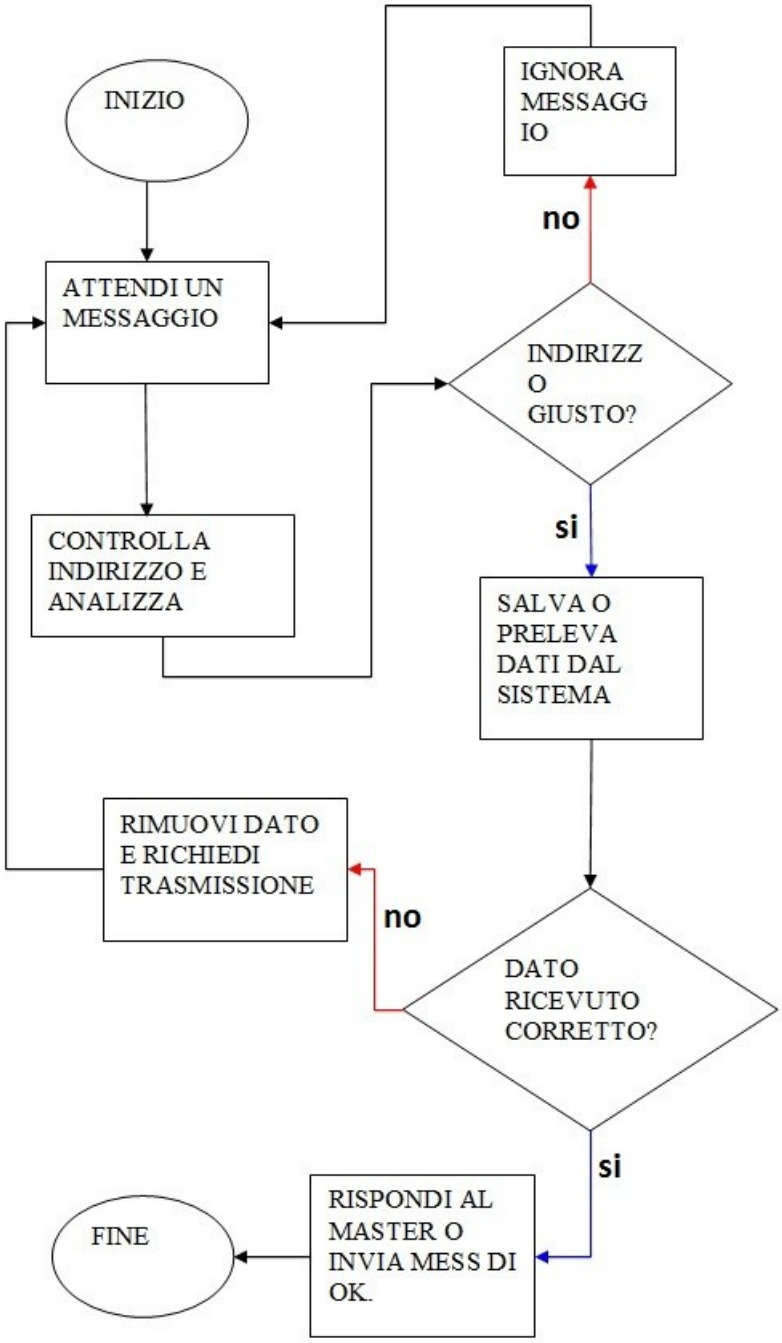

Fig. 2.8 Diagramma di flusso del Modbus Slave.

Legenda: freccia blu=affermazione;
freccia rossa=negazione;
forma rettangolo=azione;
forma rombo=confronto.
Ovale=inizio/fine

Descrizione

Questo modello in fig. 2.8 fornisce una descrizione molto generale del processo di comunicazione che avviene per un pacchetto, scambiato tra master e slave (il master può essere un pc, lo slave la nostra schedina).
Principalmente il master può inviare un comando, aprire chiudere un cancello, oppure richiedere lo stato della posizione del cancello.

2.5 Fase Software: Implementazione del codice

Con il programma C, ho iniziato a comprendere il funzionamento del protocollo modbus, con un codice pre-esitente che però andava opportunamente modificato, semplificato, ottimizzato alle esigenze della nostra scheda in progetto. Si sono ridotte il numero di istruzioni e comandi del modbus.

2.51 Codice shift register

```
void gestione_shift_register(void)
{
 static unsigned char fase_shift_register=0;

 switch(fase_shift_register)
 {
  case 0: //INIZIO ACQUISIZIONE INGRESSI
   pin_strobe_165=0;
   fase_shift_register++;
   break;

  case 1: //FINE ACQUISIZIONE INGRESSI
   pin_strobe_165=1;
   fase_shift_register++;
   break;

  case 2: //INIZIO SCARICAMENTO DATI IN SERIALE
  case 4:
  case 6:
  case 8:
  case 10:
  case 12:
  case 14:
  case 16:
   pin_clk_165=1;
   fase_shift_register++;
   break;

  case 3:
   pin_dip7=pin_data_165;
   pin_clk_165=0;
   fase_shift_register++;
   break;

  case 5:
```

```
    pin_dip6=pin_data_165;
    pin_clk_165=0;
    fase_shift_register++;
    break;

   case 7:
    pin_dip5=pin_data_165;
    pin_clk_165=0;
    fase_shift_register++;
    break;

   case 9:
    pin_dip4=pin_data_165;
    pin_clk_165=0;
    fase_shift_register++;
    break;

   case 11:
    pin_dip3=pin_data_165;
    pin_clk_165=0;
    fase_shift_register++;
    break;

   case 13:
    pin_dip2=pin_data_165;
    pin_clk_165=0;
    fase_shift_register++;
    break;

   case 15:
    pin_dip1=pin_data_165;
    pin_clk_165=0;
    fase_shift_register++;
    break;

   case 17: //FINE SCARICAMENTO DATI SERIALE
    pin_dip0=pin_data_165;
    pin_clk_165=0;
    fase_shift_register++;
    break;

   default: fase_shift_register=0;
  }
```

Spiegazione codice

La funzione principale di questo codice è quella di salvare i singoli bit ricevuti in serie raggruppandoli per formare un dato, senza perdere nessuna informazione. Il dato viene raccolto in un registro formato dal valore assunto dai dip0…dip7, e salvato in un buffer di memoria.
Il pin che si occupa di trasmettere/ricevere dati in serie è pin_data_165; il pin che serve ad abilitarlo è pin_clk_165.

Schema shift register in fig. 2.9:

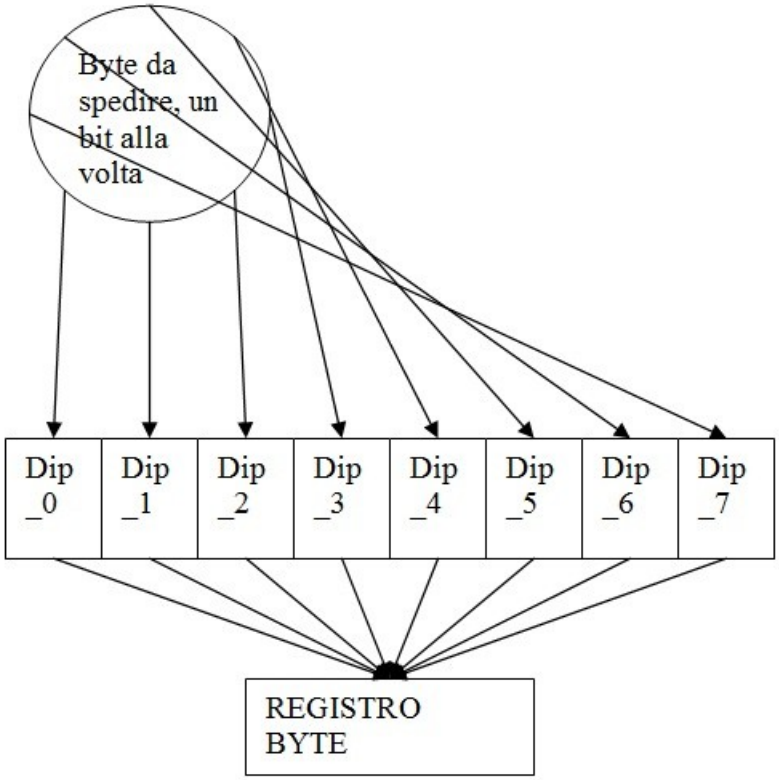

Fig. 2.9 Funzionamento indicativo dello Shift register.

2.52 Gestione del Buffer di memoria

Ogni byte da trasmettere viene dapprima salvato in un buffer, e messo in attesa per il suo "turno". I registri del micro che si occupano di raccogliere un byte e trasmetterlo sono u0tb e u2tb, dedicati rispettivamente alla porta RS232 e RS485. Quando, ad esempio, il registro u2tb è libero il buffer gli pone un byte, e dopo averlo trasmesso si ripete l'operazione per tutti i byte da spedire.
Note: u0tb (uart0 transmit buffer) e u2tb (uart2 transmit buffer) sono dei registri a 8 bit dedicati alla trasmissione seriale effettiva di un byte.

2.53 Interfacce seriali: UART0 e UART2

L'interfaccia seriale consiste in due canali: UART0 e UART2.

UART0

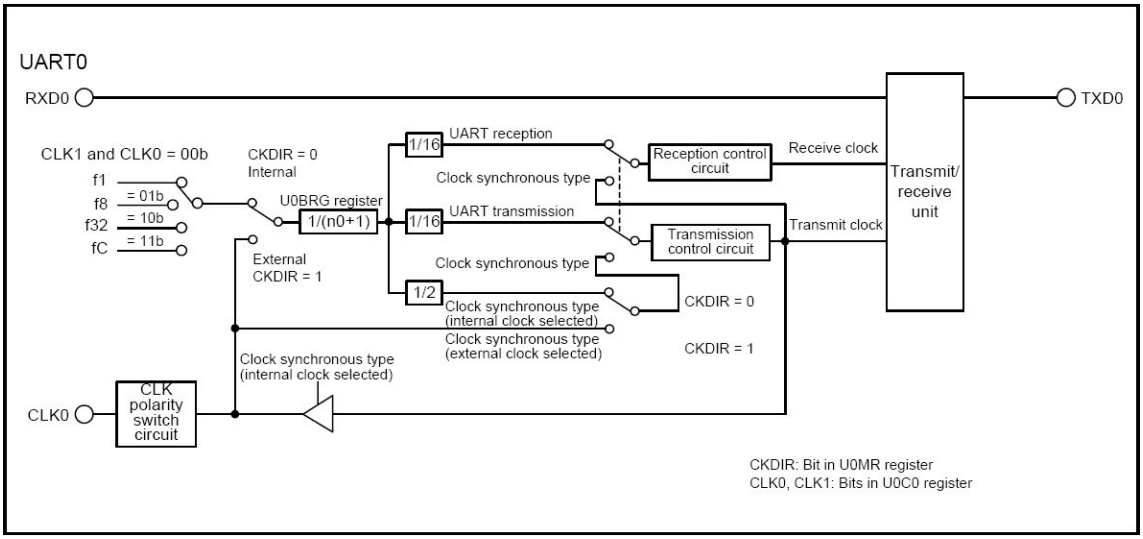

Fig. 2.10 Diadramma a blocchi dell'interfaccia Uart0, fonte Renesas R8C/32 Hardware Manual

UART0 (fig.2.10) ha un timer dedicato per generare un clock di trasferimento dati e opera in modo indipendente. UART0 supporta le modalità di trasmissione seriale I/O a clock sincrono e clock asincrono (UART mode). UART mode permette la trasmissione e la ricezione dati dopo aver impostato il bit rate e il formato del trasferimento dati desiderato.
Utile al progetto è stato L'UART mode. Il clock di trasferimento dati che ho scelto di utilizzare nei test è 19200bps, mentre nel progetto finale sarà impostato per motivi di sicurezza a 9600bps.

Bit Rate

In UART mode, il bit rate è dato dalla frequenza del sistema divisa dal registro U0BRG e divisa per 16.

UART mode

• Internal clock selected

$$\text{Setting value in U0BRG register} = \frac{fj}{\text{Bit Rate} \times 16} - 1$$

fj: Count source frequency of U0BRG register (f1, f8, f32, or fC)

• External clock selected

$$\text{Setting value in U0BRG register} = \frac{fEXT}{\text{Bit Rate} \times 16} - 1$$

fEXT: Count source frequency of U0BRG register (external clock)

2.54 Impostazione del bit rate

Per impostare la velocità di trasmissione agisco nel registro U0BRG (Uart0 Bit Rate Register). Questo registro serve a dividere la frequenza di partenza scelta f1=20MHz; la frequenza iniziale viene divisa per la costante 16 dai circuiti interni. Per ottenere, nel mio caso, un bit rate di 19200bps devo dividere ulteriormente per 64; quindi imposto nel registro il valore in binario 0100000, equivalente a 0x40h in esadecimale.

Tabella 2.2 Esempi di impostazione bit rate, da Renesas R8C 32 Hardware Manual

Bit Rate (bps)	U0BRG Count Source	System Clock = 20 MHz			System Clock = 18.432 MHz (1)			System Clock = 8 MHz		
		U0BRG Setting Value	Actual Time (bps)	Setting Error (%)	U0BRG Setting Value	Actual Time (bps)	Setting Error (%)	U0BRG Setting Value	Actual Time (bps)	Setting Error (%)
1200	f8	129 (81h)	1201.92	0.16	119 (77h)	1200.00	0.00	51 (33h)	1201.92	0.16
2400	f8	64 (40h)	2403.85	0.16	59 (3Bh)	2400.00	0.00	25 (19h)	2403.85	0.16
4800	f8	32 (20h)	4734.85	-1.36	29 (1Dh)	4800.00	0.00	12 (0Ch)	4807.69	0.16
9600	f1	129 (81h)	9615.38	0.16	119 (77h)	9600.00	0.00	51 (33h)	9615.38	0.16
14400	f1	86 (56h)	14367.82	-0.22	79 (4Fh)	14400.00	0.00	34 (22h)	14285.71	-0.79
19200	f1	64 (40h)	19230.77	0.16	59 (3Bh)	19200.00	0.00	25 (19h)	19230.77	0.16
28800	f1	42 (2Ah)	29069.77	0.94	39 (27h)	28800.00	0.00	16 (10h)	29411.76	2.12
38400	f1	32 (20h)	37878.79	-1.36	29 (1Dh)	38400.00	0.00	12 (0Ch)	38461.54	0.16
57600	f1	21 (15h)	56818.18	-1.36	19 (13h)	57600.00	0.00	8 (08h)	55555.56	-3.55
115200	f1	10 (0Ah)	113636.36	-1.36	9 (09h)	115200.00	0.00	–	–	–

UART2

Il canale di comunicazione Uart2 è molto simile ad Uart0. Cambiano i nomi dei registri. Le modalità di trasmissione e l'assegnazione del clock di trasmissione avviene in modo uguale ad Uart0.

2.55 Codice implementato Buffer RS232 e RS485

```
/************ Buffer RS232 TO RS485 ************/
void gestione_buffer232(void)
{
 static unsigned char indice_rx_232=0;
 static unsigned char indice_tx_485=0;
 static unsigned char buffer_232[16];
 static unsigned char buffer_rs232_completo=0;


 if(ri_u0c1==1)
 {
  if(indice_rx_232<16)
  {
   buffer_232[indice_rx_232]=buffer_ricevuto;
   indice_rx_232++;

   if(indice_rx_232!=indice_tx_485)
   {
    if(ti_u2c1)
    {
     if(indice_tx_485<16)
     {
      u2tb=buffer_232[indice_tx_485];
      indice_tx_485++;
     }
     else indice_tx_485=0;
    }
   }
   else
   {
    indice_rx_232=0;
```

```
      buffer_rs232_completo=1;
     }
    }
   }
   else if(ri_u0c1==0 && buffer_rs232_completo==1)
   {
    if(indice_rx_232!=indice_tx_485 && ti_u2c1)
    {
     if(indice_tx_485<16)
     {
      u2tb=buffer_232[indice_tx_485];
      indice_tx_485++;
     }
     else indice_tx_485=0;
    }
   }
}

//---------------- Buffer RS485 TO RS232 --------------------
void gestione_buffer485(void)
{
 static unsigned char indice_rx_485=0;
 static unsigned char indice_tx_232=0;
 static unsigned char buffer_485[16];
 static unsigned char buffer_rs485_completo=0;

 if(ri_u2c1==1)
 {
  if(indice_rx_485<16)
  {
   buffer_485[indice_rx_485]=buffer_ricevuto;
   indice_rx_485++;
   if(indice_rx_485!=indice_tx_232)
   {
    if(ti_u0c1==1)
    {
     if(indice_tx_232<16)
     {
      te_u0c1=1;
      u0tb=buffer_485[indice_tx_232];
      indice_tx_232++;
     }
     else indice_tx_232=0;
    }
   }
  }
  else
  {
   indice_rx_485=0;
   buffer_rs485_completo=1;
  }
 }
 else if(ri_u2c1==0 && buffer_rs485_completo==1)
```

```
  {
   if(indice_rx_485!=indice_tx_232 && ti_u0c1==1)
   {
    if(indice_tx_232<16)
    {
     u0tb=buffer_485[indice_tx_232];
     indice_tx_232++;
    }
    else indice_tx_232=0;
   }
  }
}
```

Fine.

2.56 Descrizione del codice buffer RS232 e RS485

Questo codice si occupa di gestire l'invio e la ricezione di dati delle porte RS232 e RS485, e non solo. Deve anche mettere a disposizione i dati ricevuti da una porta per poterli spedire con l'altra, e viceversa.
Quando sono in fase di ricezione con RS232, il buffer di ricezione RS232 si riempie (1byte) e quando è completo si salva nel buffer_232[indice_rx232]. Appena il dato è disponibile e la porta RS485 non si trova in fase di ricezione, lo posso inviare seguendo l'indice di trasmissione indice_tx485 fino a che il buffer_232 non torna vuoto (riferimento a figura 2.11).
Se sono in fase di ricezione con RS485, appena il buffer di ricezione RS485 è completo (1byte) il dato si salva nel buffer_485[indice_rx485], fino al suo riempimento. E' poi possibile spedire i dati con RS232 seguendo l'indice di trasmissione indice_tx232.

Nota: nella scheda SIS con la porta RS485 non è possibile eseguire contemporaneamente sia trasmissione che ricezione; infatti lo standard RS485 è uno standard half-duplex, cioè la trasmissione dei dati è bidirezionale ma non contemporanea: occorre quindi che si indichi in modo esplicito se si vuole realizzare una ricezione o una trasmissione.

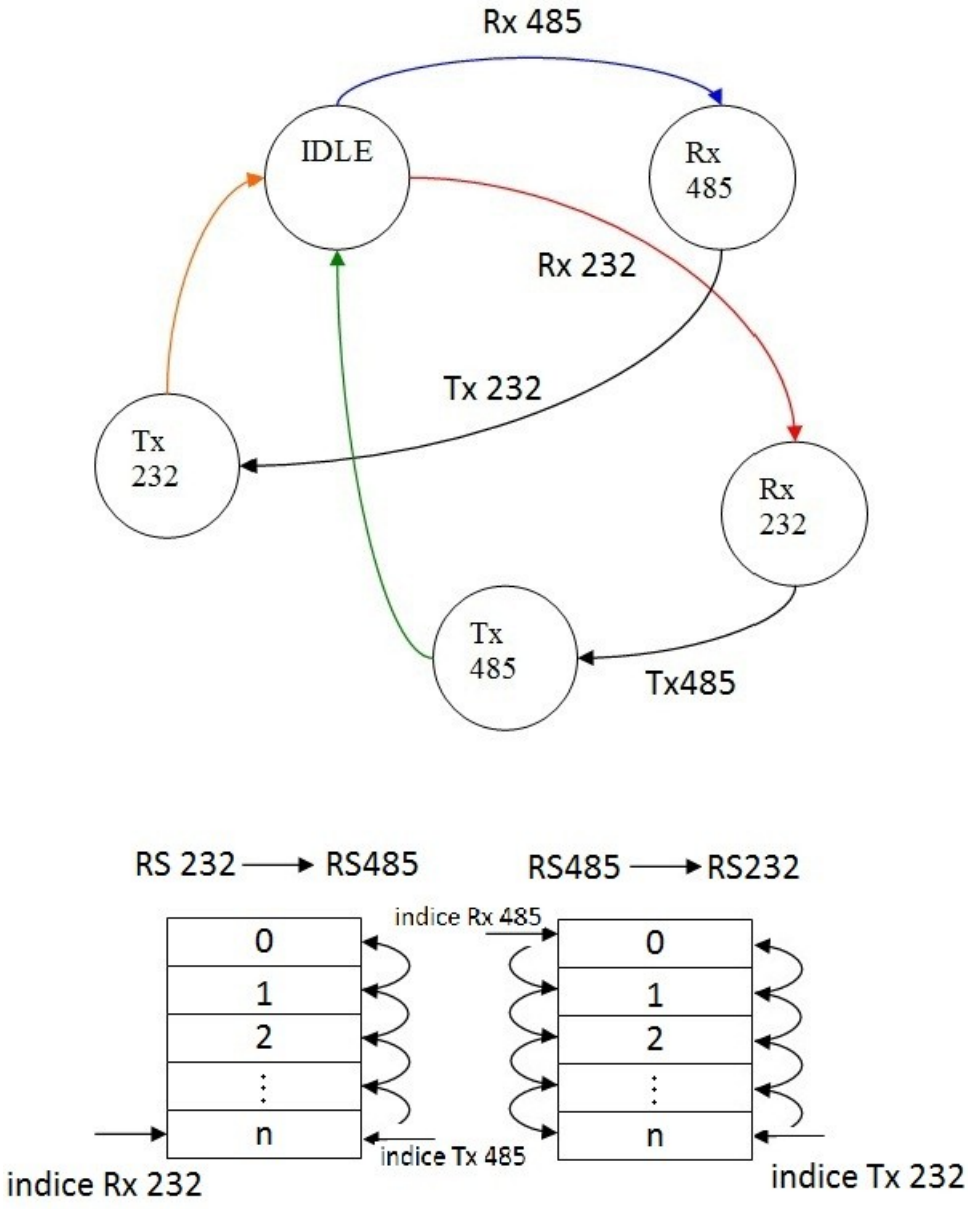

Figura 2.11 Diagramma a stati dei Buffer RS232 e RS485.

2.57 Approfondimento: porta RS232 e RS485

I segnali con cui si ha a che fare nei sistemi digitali sono spesso di tipo single ended o sbilanciati: la tensione è misurata rispetto ad un riferimento comune indicato come massa. Un esempio di tali segnali, per rimanere nel campo dei sistemi di comunicazione, è la porta seriale RS232; l'uno logico è identificato da una tensione negativa compresa, in modulo, tra 3 e 15 V e lo zero da una tensione positiva maggiore di 3 V.

Per trasmettere un singolo bit usando un segnale sbilanciato è evidentemente necessario usare un solo filo oltre alla massa che è unica e comune a tutti i segnali.

I limiti nell'uso di segnali non bilanciati nella trasmissione di informazioni derivano da due considerazioni:

- Se è vero, almeno in prima approssimazione, che il potenziale di massa è univocamente definito in un circuito di piccole dimensioni, tale concetto diventa una illusione quando le distanze sono elevate; infatti la corrente, scorrendo nel filo di riferimento, produce differenze di potenziali a causa dell'impedenza elettrica del collegamento. Questo fatto è già abbastanza rilevante in corrente continua (quando occorre tener conto della sola resistenza del cavo) ma è fondamentale quando, usando segnali ad alta velocità, l'induttanza del cavo diviene l'elemento predominante dell'impedenza.
- Lungo un filo di lunghezza non trascurabile la tensione subisce l'influsso casuale e continuamente variabile dei disturbi esterni: il ricevitore quindi osserva un segnale digitale sovrapposto a "rumore" che, se elevato, potrebbe portare ad interpretazioni errate del valore logico. Tale effetto si verifica in modo casuale sia nei confronti del segnale sia nei confronti del riferimento, per di più con diversa intensità considerando che l'impedenza equivalente verso massa è diversa.

Una soluzione ad entrambi i problemi è quella di adottare elevate escursioni del segnale al fine di aumentare il rapporto tra segnale e disturbo (p.e. la RS232 prevede escursioni tipiche di 24 V) oppure mantenere corti i collegamenti (soluzione ovviamente improponibile se i due oggetti da collegare sono fisicamente distanti) oppure ancora usare, almeno per il segnale di riferimento, cavi di elevata sezione (opzione con evidenti impatti negativi e comunque che porta a benefici solo marginali).

Il secondo dei due problemi accennati è inoltre parzialmente risolvibile utilizzando cavi schermati.
Nei sistemi bilanciati o differenziali la tensione associata alla trasmissione di un singolo bit è misurata come differenza di potenziale tra due fili, tra loro identici e pilotati da trasmettitori con la stessa impedenza di uscita: se la tensione è maggiore su un filo rispetto all'altro il valore logico è associato ad uno zero, se è minore ad un uno. Non ha invece nessuna importanza la tensione dei due fili rispetto a massa.

Questo metodo permette di superare i due problemi appena descritti:

- Il valore logico è associato alla differenza di potenziale tra due fili: il potenziale assoluto della massa è quindi teoricamente ininfluente. In questo modo, anche se sul filo di massa scorrono correnti e quindi si creano differenze di potenziale, non si generano effetti sui valori logici.
  Si usa dire che un sistema di trasmissione differenziale non è sensibile alla tensione di modo comune (Vcm), definita come la media della tensione dei due fili che trasportano il segnale misurata rispetto alla massa locale.
- Visto che la coppia di fili su cui il segnale viaggia è costituita da un "doppino" pilotato da trasmettitori con la stessa impedenza di uscita, i disturbi sono fortemente attenuati.

  Esistono ovviamente anche degli svantaggi:
- È necessario prevedere un numero doppio di conduttori: per ogni singolo segnale servono due fili, oltre alla massa comune a tutti i segnali ed in genere necessaria.
- Sono richiesti driver e ricevitori più complessi. Utilizzando circuiti integrati appositamente studiati e largamente diffusi, questo problema è però facilmente superabile.

Lo standard RS485

Lo standard RS485 prevede il supporto delle linee multi-drop, cioè linee in cui coesistono più ricevitori e trasmettitori sulla stessa coppia di fili.

Analizziamo per semplicità un collegamento tra un solo trasmettitore e un ricevitore (fig.2.12):

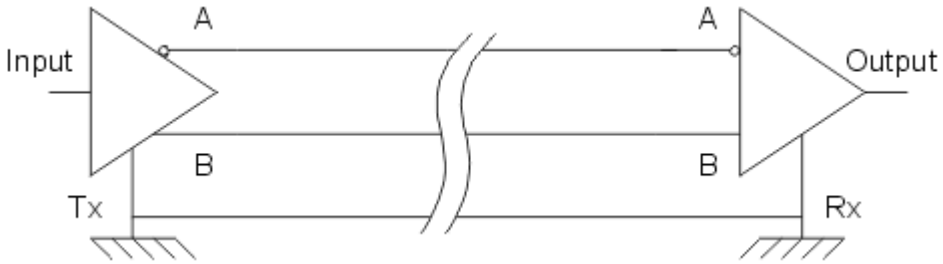

Fig. 2.12 Collegamento singolo tra trasmettitore e ricevitore, fonte http://www.vincenzov.net, ottobre 2011.

I due stati di ciascuna linea sono definiti nel seguente modo:

- Quando il terminale A è negativo rispetto a B, la linea rappresenta un uno binario. Tale stato rappresenta anche l'assenza di segnale (idle state).
- Quando il terminale A è positivo rispetto a B, la linea rappresenta uno zero binario.

Nella figura 2.13 viene mostrato l'andamento idealizzato dei segnali sui due fili A (in rosso) e B (in blu): come si vede si tratta di due segnali tra loro in opposizione di fase. Nell'immagine sono mostrati come variabili tra zero ed una tensione positiva (come del resto avviene il più delle volte anche nei sistemi reali) anche se questo non è richiesto dallo standard. La tensione differenziale è quella che effettivamente trasmette l'informazione ed è positiva o negativa in funzione del livello logico trasmesso.

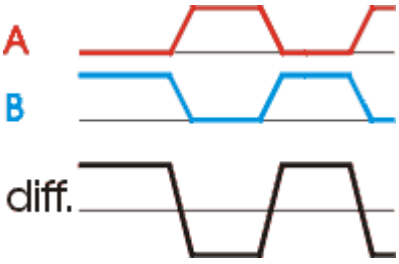

Fig. 2.13 Segnali a tensione differenziale

Da notare che in molte realizzazioni il terminale A è identificato come - ed il terminale B come + oppure con altri nomi in cui è evidenziato, anche graficamente, che hanno sempre valori logici opposti.
All'uscita del trasmettitore la differenza di potenziale tra le linee A e B deve essere di almeno 4 V e la tensione di modo comune deve essere minore di 7 V (normalmente una linea vale circa 0 V e l'altra circa 5 V). Il ricevitore deve essere in grado di interpretare correttamente lo stato della linea quando la differenza di potenziale è superiore in modulo a 200 mV.
Al fine di evitare conflitti è ovviamente necessario che un solo trasmettitore alla volta sia attivo. Questo implica l'uso di trasmettitori che, oltre alle uscite corrispondenti allo zero e all'uno, possano gestire anche un "terzo stato" in cui l'elettronica appare come fisicamente non collegata alla linea (stato detto ad alta impedenza, three-state o Hi-Z).
I ricevitori possono invece essere tutti attivi contemporaneamente ed in genere lo sono effettivamente.
La topologia più usata con questo protocollo è quella a due fili (oltre alla massa) rappresentata nello schema seguente. Questa connessione permette la trasmissione bidirezionale(ma ovviamente non contemporanea) tra due o più nodi che, dal punto di vista elettrico, sono tra loro equivalenti.

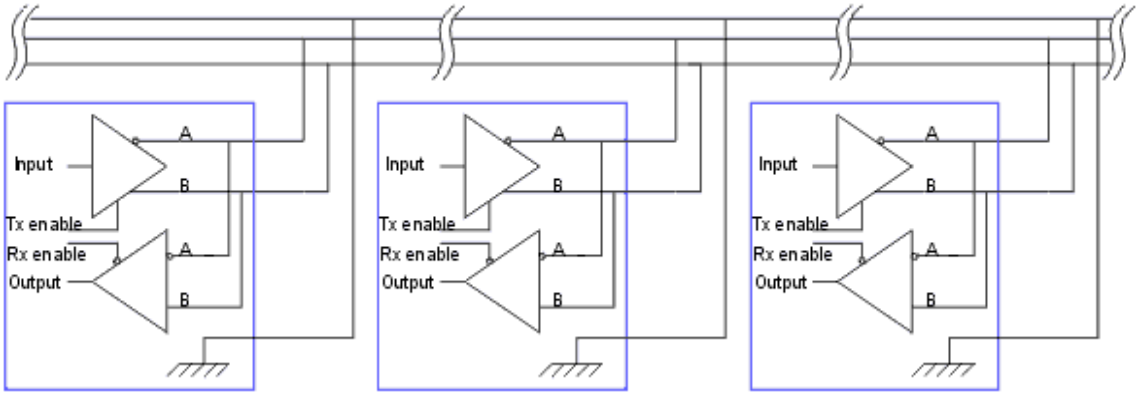

Fig. 2.14 Connessione tra più nodi, fonte http://www.vincenzov.net, ottobre 2011

La sezione rice-trasmittente di ciascuno dei nodi è evidenziato nella figura 2.14 da un rettangolo blu.
Le connessioni verso la linea di trasmissione sono costituite semplicemente dai due terminali A e B comuni sia alla sezione di ricezione che a quella di trasmissione e dalla massa. Possono essere ovviamente presenti dei moduli dotati del solo ricevitore o, caso meno probabile, del solo trasmettitore. Il collegamento di terra non ha funzioni nella trasmissione ma è richiesto.
Ciascun modulo trasmettitore deve possedere verso il dispositivo digitale di un ingresso dati e di un ingresso di abilitazione alla trasmissione, pilotato localmente, che permette di disabilitare il trasmettitore quando non serve: al fine di evitare conflitti è necessario prevedere un qualche meccanismo che impedisca l'attivazione contemporanea di più trasmettitori oppure sia in grado rilevare tali conflitti ed intervenire opportunamente. I driver RS485 sono comunque progettati per non riportare danni anche in caso di corto circuito permanente, limitando la corrente massima a 250 mA.
Nello schema disegnato è previsto anche un segnale di abilitazione del ricevitore, sebbene spesso non necessario: è infatti possibile lasciare tutti i ricevitori sempre attivi oppure collegare insieme i due ingressi di abilitazione essendo normalmente attivi su livelli logici opposti.
Lo standard originario permette la connessione di massimo 32 ricevitori ma utilizzando integrati a basso assorbimento tale limite può essere abbondantemente superato.

2.6 Fase Hardware: Realizzazione scheda SIS

Dopo aver preparato il codice, mi sono occupato di costruire la schedina seguendo uno schema fornito dal mio tutore.
Eseguite tutte le opportune saldature a stagno, ecco realizzata la schedina.

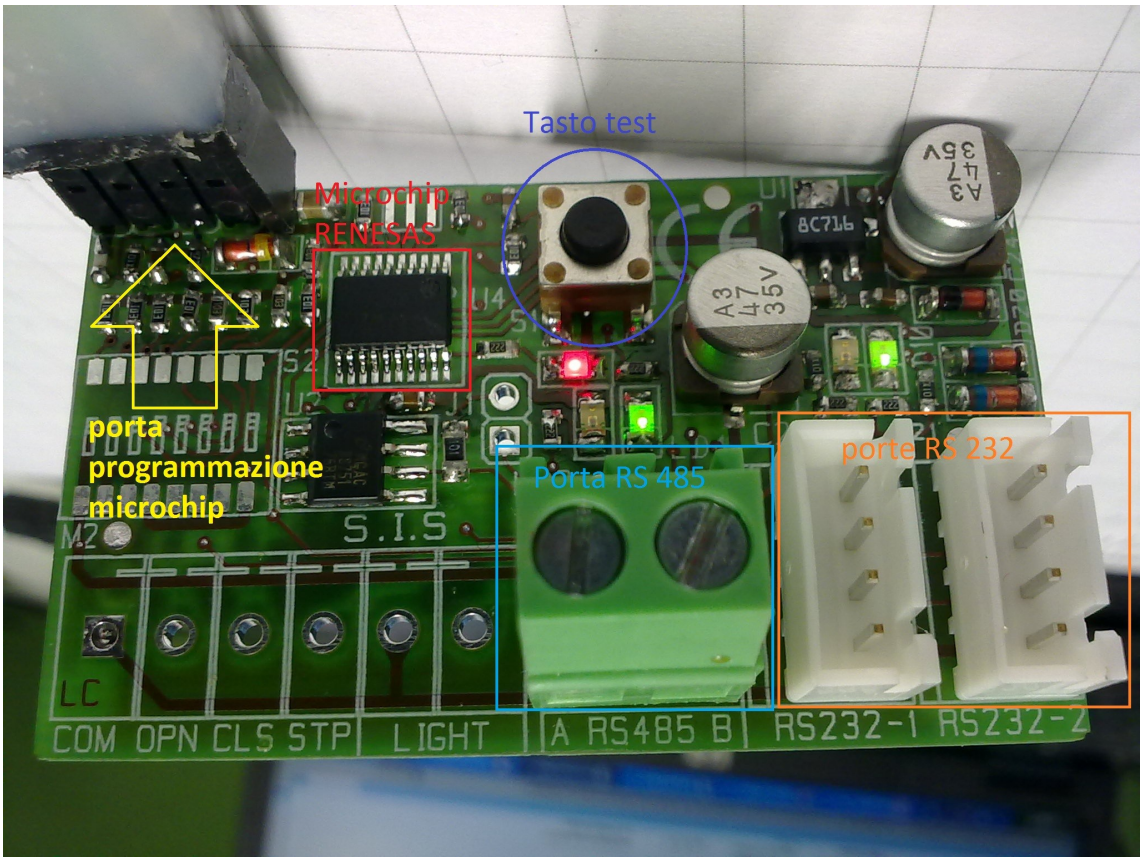

Figura 2.15, Scheda SIS con componenti d'interesse evidenziati.

Nella figura 2.15 sono evidenziati i componenti che riassumono le funzioni della scheda SIS.
In alto a sinistra c'è il connettore tramite il quale è possibile programmare il microchip;
subito a fianco si trova il Microchip Renesas, cuore della scheda. In alto al centro c'è il tasto con cui effettuare test e prove di funzionamento, infine, in basso, ci sono le porte di comunicazione RS485 e RS232, una per trasmissioni a lunga distanza (RS485, fino a un km) e l'altra per comunicare con schede vicine (RS232, fino a qualche metro). Le due porte RS232 servono a mandare comandi e ricevere informazioni dalle SCM (scheda comando motore) che gestiscono i due motori elettrici presenti; mentre l'unica porta RS485 è dedicata a raccogliere ordini dall'esterno, inviare informazioni e comandi ad altri cancelli.

2.61 Collegamento scheda-pc. Test di trasmissione

Collego la scheda al pc tramite un decoder fornito dalla ditta, e costruisco un adattatore con un chip che converte il segnale della scheda da TTL a CMOS in quanto la porta RS-232 del pc necessita di tale segnale.
Una volta compilato il programma nella scheda, procedo col primo tentativo di comunicazione.
Imposto nel programma inserito, un comando che verrà spedito tramite la pressione di un tasto nella scheda, verso il pc. Nel pc uso un programma per ricevere il segnale. Imposto la velocità di trasmissione a 19200bps.
Dopo alcuni tentativi e correzioni del programma, riesco finalmente a vedere il codice trasmesso dalla schedina.
Successivamente ho provato a far spedire intere frasi usando il codice ASCII.

2.62 Primo testo spedito con SIS

ASCII

ASCII è l'acronimo di **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange (ovvero *Codice Standard Americano per lo Scambio di Informazioni*)

Introduzione

È un sistema di codifica dei caratteri a 7 bit (vedi nota) comunemente utilizzato nei calcolatori, proposto dall'ingegnere dell'IBM Bob Bemer nel 1961, e successivamente accettato come standard dall'ISO (ISO 646). Per non confonderlo con le estensioni a 8 bit proposte successivamente, questo codice viene talvolta riferito come **US-ASCII**.
Alla specifica iniziale basata su codici di 7 bit fecero seguito negli anni molte proposte di estensione ad 8 bit, con lo scopo di raddoppiare il numero di caratteri rappresentabili. Nei PC IBM si fa per l'appunto uso di una di queste estensioni, ormai standard di fatto, chiamata *extended ASCII* o *high ASCII*. In questo ASCII esteso, i caratteri aggiunti sono vocali accentate, simboli semigrafici e altri simboli di uso meno comune. I caratteri *extended ASCII* sono codificati nei cosiddetti codepage.
L'asteroide 3568 ASCII prende il nome da questa codifica dei caratteri.
Nota: In realtà, di solito si aggiunge un 8º bit, usato come bit di parità, per rilevare eventuali errori.

Tabella 2.3 Caratteri ASCII (vedi appendice).

Ecco un esempio di come compariva lo schermo dopo l'utilizzo del codice ASCII.

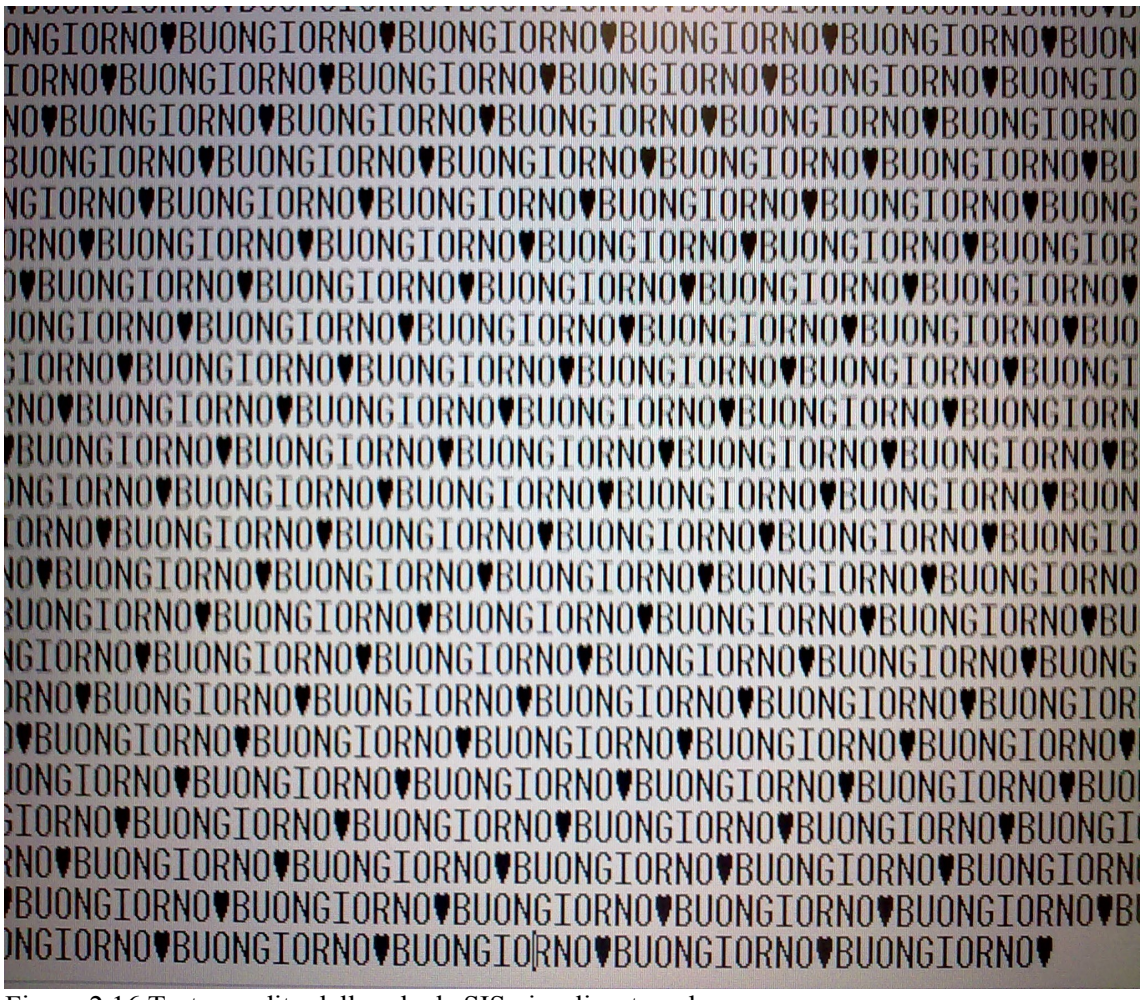

Figura 2.16 Testo spedito dalla scheda SIS visualizzato nel pc.

Il messaggio composto dai byte trasmessi è questo:

```
lettera[0]=0x42; //B
lettera[1]=0x55; //U
lettera[2]=0x4F; //O
lettera[3]=0x4E; //N
lettera[4]=0x47; //G
lettera[5]=0x49; //I
lettera[6]=0x4F; //O
lettera[7]=0x52; //R
lettera[8]=0x4E; //N
lettera[9]=0x4F; //O
lettera[10]=0x20;//spazio
```

In figura 2.16 viene mostrato il messaggio spedito dalla scheda con un bit rate fissato a 19200bps.
La scheda era impostata in modo tale che dopo una sola pressione del tasto spedisse in modo continuo e ripetitivo il messaggio.

2.63 Collegamento scheda-scheda. Test di trasmissione

Una volta accertato il corretto funzionamento di trasmissione tramite porta RS-232, verifichiamo la trasmissione tra due schede identiche, stavolta utilizzando le porte RS-485, ottime per trasmissioni a media distanza (10:1000 m).
Creo un codice che permetta alla scheda master di inviare un comando all'altra scheda (slave) in modo che cambi lo stato dei led presenti, e successivamente riceva un comando di ritorno dallo slave che modifichi lo stato dei led del master.

Ho utilizzato il codice seguente:

```
1.  void gestione_uart2(void)
2.  {
3.   #define START_TX         0xAA
4.   #define FINE_TX          0x55
5.   #define INDIRIZZO_UART    0x76
6.   #define COMANDO_LED_ON    0x10
7.   #define COMANDO_LED_OFF   0x01
8.   #define COMANDO_READ_BYTE 0x88
9.   #define COMANDO_W         0x05
10.
11.  static unsigned char fase_uart2=0;
12.  static unsigned char indice_ricezione;
13.  static unsigned char buffer_ricezione_uart2[16];
14.  static unsigned char indice_trasmissione;
15.  static unsigned char buffer_trasmissione_uart2[16];
16.
17.  if(tasto)
18.  {
19.    tasto=0;
20.    if(fase_uart2==0 || fase_uart2==1)
21.    {
22.      fase_uart2=200;
```

```
23.          }
24.        }
25.
26.
27.        switch(fase_uart2)
28.        {
29.          case 0: //inizializzazione uart ricezione
30.            pin_enable_RS485=0;
31.            re_u2c1=1;
32.            fase_uart2=1;
33.            break;
34.
35.          case 1: //attesa carattere di inizio trasmissione
36.            if(flag_ricevuto_uart2)
37.            {
38.              if(buffer_ricevuto==START_TX)
39.              {
40.                fase_uart2++;
41.                indice_ricezione=0;
42.              }
43.              flag_ricevuto_uart2=0;
44.            }
45.            break;
46.
47.          case 2: //inserimento dati nel vettore
48.            if(flag_ricevuto_uart2)
49.            {
50.              flag_ricevuto_uart2=0;
51.              if(buffer_ricevuto!=FINE_TX)
52.              {
53.                if(indice_ricezione<sizeof(buffer_ricezione_uart2)-1)
54.                {
55.                  if((indice_ricezione & 0x01) == 0x01)
56.                  {
57.                    buffer_ricezione_uart2[indice_ricezione/2]|=(unsigned char)...
                       ...(buffer_ricevuto&0x0F);
58.                  }
59.                  else
60.                  {
61.                    buffer_ricezione_uart2[indice_ricezione/2]=(unsigned char)...
                       ...(buffer_ricevuto<<4);
62.                  }
63.                  indice_ricezione++;
64.                }
65.                else
66.                  fase_uart2=0; //errore overflow buffer
67.              }
68.              else
69.                fase_uart2=100;
70.            }
71.            break;
72.
73.          case 100: //ricezione completata, analizzo il pacchetto
```

```
74.           if(buffer_ricezione_uart2[0]==INDIRIZZO_UART)
75.           {
76.            if(buffer_ricezione_uart2[1]==COMANDO_LED_ON)
77.            {
78.             pin_led=!pin_led;
79.            }
80.            else if(buffer_ricezione_uart2[1]==COMANDO_LED_OFF) pin_led=!pin_led;
81.            else if(buffer_ricezione_uart2[1]==COMANDO_READ_BYTE)
82.            {
83.             fase_uart2=30;
84.            }
85.           }
86.           fase_uart2=255;
87.           break;
88.
89.          case 200: //creazione pacchetto
90.           buffer_trasmissione_uart2[0]=START_TX;
91.           buffer_trasmissione_uart2[1]=INDIRIZZO_UART>>4;
92.           buffer_trasmissione_uart2[2]=INDIRIZZO_UART&0x0F;
93.           //effettuo il toggle del comando
94.           if(buffer_trasmissione_uart2[3]==(COMANDO_LED_ON>>4) &&
95.              buffer_trasmissione_uart2[4]==(COMANDO_LED_ON&0x0F))
96.           {
97.            buffer_trasmissione_uart2[3]=(COMANDO_READ_BYTE>>4);
98.            buffer_trasmissione_uart2[4]=(COMANDO_READ_BYTE&0x0F);
99.           }
100.          else
101.          {
102.           buffer_trasmissione_uart2[3]=(COMANDO_LED_ON>>4);
103.           buffer_trasmissione_uart2[4]=(COMANDO_LED_ON&0x0F);
104.          }
105.          buffer_trasmissione_uart2[5]=FINE_TX;
106.          indice_trasmissione=0;
107.          fase_uart2++;
108.          break;
109.
110.         case 201: //calcolo del checksum
111.
112.
113.          fase_uart2++;
114.          break;
115.
116.         case 202: //trasmissione pacchetto
117.          pin_enable_RS485=1;
118.          re_u2c1=0;
119.
120.          if(trasmissione_uart2(buffer_trasmissione_uart2[indice_trasmissione],0)==0)
121.          { //nessuna trasmissione in corso, posso iniziare a trasmettere
122.           if(buffer_trasmissione_uart2[indice_trasmissione]==FINE_TX)
123.           {
124.            trasmissione_uart2(buffer_trasmissione_uart2[indice_trasmissione],1);
125.            fase_uart2=255;
126.           }
```

```
127.            else if(trasmissione_uart2(buffer_trasmissione_uart2[indice_trasmissione],1)==1)
128.            { //trasmissione iniziata, incremento indice buffer
129.              indice_trasmissione++;
130.            }
131.          }
132.          break;
133.
134.        case 30: //creazione pacchetto BYTE
135.          buffer_trasmissione_uart2[0]=START_TX;
136.          buffer_trasmissione_uart2[1]=INDIRIZZO_UART>>4;
137.          buffer_trasmissione_uart2[2]=INDIRIZZO_UART&0x0F;
138.          //effettuo il toggle del comando
139.          if(buffer_trasmissione_uart2[3]==(COMANDO_READ_BYTE>>4) &&
140.             buffer_trasmissione_uart2[4]==(COMANDO_READ_BYTE&0x0F))
141.          { //inserisco COMANDO diverso
142.            buffer_trasmissione_uart2[3]=(COMANDO_LED_ON>>4);
143.            buffer_trasmissione_uart2[4]=(COMANDO_LED_ON&0x0F);
144.          }
145.        /*else
146.          {
147.            buffer_trasmissione_uart2[3]=(COMANDO_LED_ON>>4);
148.            buffer_trasmissione_uart2[4]=(COMANDO_LED_ON&0x0F);
149.          }*/
150.          buffer_trasmissione_uart2[5]=FINE_TX;
151.          indice_trasmissione=0;
152.          fase_uart2++;
153.          break;
154.
155.          case 31: //trasmissione pacchetto BYTE
156.          pin_enable_RS485=1;
157.          re_u2c1=0;
158.
159.          if(trasmissione_uart2(buffer_trasmissione_uart2[indice_trasmissione],0)==0)
160.          { //nessuna trasmissione in corso, posso iniziare a trasmettere
161.            if(buffer_trasmissione_uart2[indice_trasmissione]==FINE_TX)
162.            {
163.              trasmissione_uart2(buffer_trasmissione_uart2[indice_trasmissione],1);
164.              fase_uart2=255;
165.            }
166.            else if(trasmissione_uart2(buffer_trasmissione_uart2[indice_trasmissione],1)==1)
167.            { //trasmissione iniziata, incremento indice buffer
168.              indice_trasmissione++;
169.            }
170.          }
171.          break;
172.
173.        default: //fine trasmissione
174.          pin_enable_RS485=0;
175.          fase_uart2=0;
176.          break;
177.      }
178.    }
```
Fine.

2.64 Spiegazione codice implementato

Questo codice è stato creato appositamente per simulare uno scambio dati che contenessero dei comandi, come l'apertura o chiusura di un cancello. Nella mia simulazione avevo a disposizione i led della schedina a cui potevo semplicemente far variare lo stato da acceso a spento e viceversa.
La parte iniziale del codice definisce il messaggio del master, composto da: byte di START, INDIRIZZO, COMANDI e FINE (riga 3...9).
Alla riga 17, con la pressione del tasto del master, si ottiene l'invio del messaggio.
Dalla riga 27 inizia la fase di invio del messaggio byte per byte e termina con la fine della trasmissione alla riga 71.
Dalla riga 73, lo slave che ha appena ricevuto il messaggio, analizza il pacchetto ricevuto, dapprima controlla che il messaggio sia rivolto proprio a lui, verifica se l'indirizzo corrisponde con quello a cui è stato associato e legge i comandi da eseguire.
A questo punto avviene l'accessione o spegnimento del led dello slave, come descritto dalla riga 78. Il passo successivo è dello slave che rispedisce al master il messaggio ricevuto, con l'ordine di cambiare lo stato del suo led, ma questo avviene solo dopo la pressione del tasto dello slave.
Tutto si ripete con la pressione di uno o l'altro tasto delle schede.

In figura 2.17 è riportato il segnale di un messaggio visualizzato con oscilloscopio digitale.

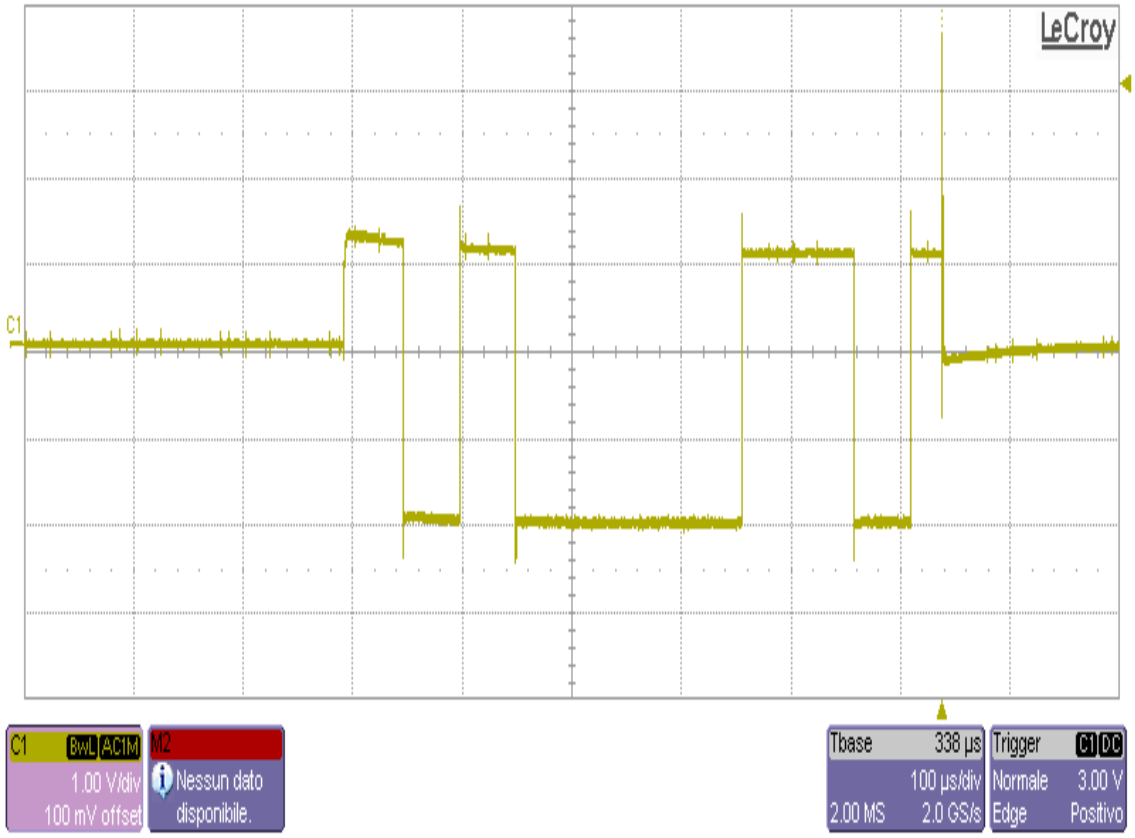

Figura 2.17 Segnale visualizzato all'oscilloscopio, messaggio del master.

Si tratta del segnale della scheda master, ovvero la prima che invia il comando alla scheda slave. Sotto, invece, riporto in figura 3.4 il segnale corrispondente alla risposta dello slave al master.

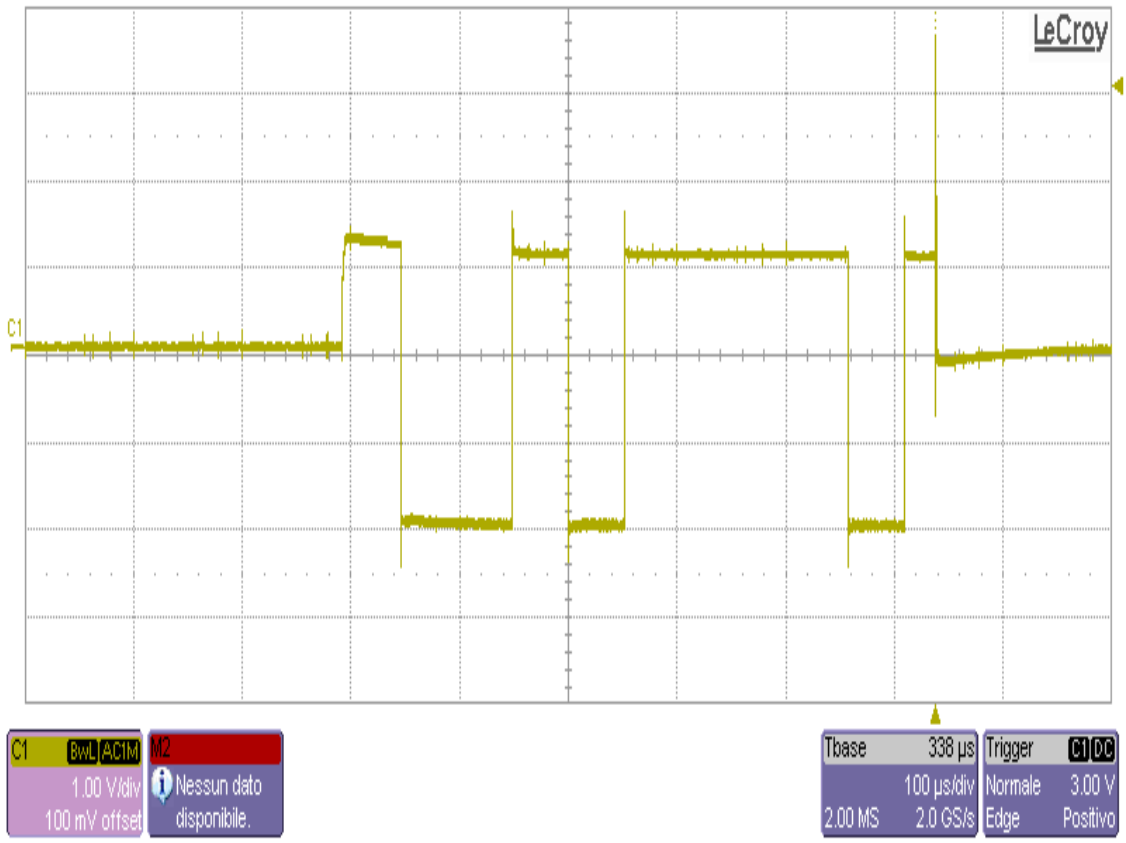

Figura 2.18 Segnale dello Slave.

2.65 Caratteristiche del segnale trasmesso

Il segnale mostrato in figura 2.18 (come anche nella figura precedente) presenta una successione di 11 bit. L'impulso generato dal microprocessore dura quanto un pacchetto di bit che forma un byte trasmesso, meno metà della durata di un bit, con una precisione di +/- ½ bit; l'abilitazione alla trasmissione cioè termina a metà dell'ultimo bit di stop.
Considerando che la trasmissione ha un bit rate impostato a 19200bps, 8 bit, con parità e un bit di stop, per ogni byte trasmesso sono necessari 11 bit:
1 bit di start + 8 + 1 bit di parità + 1 bit di stop, ciascuno della durata di 1/19200 = 52 uS. In tutto sono quindi necessari 0,546 mS (+/- 26 uS).

3. Conclusioni

Il progetto si è svolto senza problemi importanti; ho imparato a gestire i timer del microchip, approfondito lo studio del protocollo di comunicazione modbus, e delle porte fisiche di cui dovevo disporre; creato il codice necessario all'apprendimento, il codice necessario al funzionamento della scheda; e infine la realizzazione della stessa con test annessi.
Sono stati omessi alcuni step della mia esperienza come vari esercizi con il software, utili solo all'apprendimento del programma di linguaggio C di cui disponevo; ho omesso la costruzione a parte di un convertitore di segnali da TTL a CMOS, utilizzando un chip apposito, utile all'adattamento dei segnali della scheda con la porta RS232 del pc con cui lavoravo.
Dopo svariati test, siamo certi che la schedina è in grado di inviare e rispondere correttamente ai comandi inviati dall'esterno (schede periferiche e pc) senza problemi o errori di comunicazione. Rimane da implementare il codice opportuno per ulteriori sviluppi del progetto.

3.1 In futuro...

Come si è già intuito, il progetto non è ancora giunto al suo stato finale.
La fase successiva sarà quella di implementare il codice del protocollo di comunicazione modbus in modo completo; si lavorerà sul software. Si potrà implementare protocolli di comunicazione TCP/IP, Ethernet, e protocolli di accesso a terminali remoti per comunicare attraverso la rete internet.
L'idea ancora in fase embrionale è quella di arrivare ad un sistema che permetta all'azienda di effettuare manutenzione, aggiornare il software, o analizzare un problema già da “casa”. Si avrà il notevole vantaggio di migliorare l'assistenza tecnica, velocizzare l'intervento di riparazione che avverrà successivamente sul posto, vantaggi per il tecnico di essere già informato nei dettagli sul problema e poter preparare soluzioni possibili per risolverlo.

Ringraziamenti

Desidero innanzitutto ringraziare il mio relatore Prof. Simone Buso per il supporto che mi ha dato accompagnandomi passo-passo e in modo preciso alla prova finale;
il mio tutore aziendale Giuliano e il suo collaboratore Federico per i suoi consigli utili al progetto e per avermi fornito il materiale necessario alla realizzazione della tesi.

Desidero ringraziare tutta la mia famiglia che mi ha sostenuto economicamente e moralmente.
Infine ringrazio tutti i miei amici che mi sono stati vicini e i miei compagni di corso;
per ultima, ma non meno importante, la mia ragazza Pamela.

Bibliografia

Siti di riferimento: www.wikipedia.org, http://VincenzoV.net.

Appendici

La tabella 2.3 riporta alcuni dei caratteri principali.
E' relativa al codice US ASCII, ANSI X3.4-1986 (*ISO 646 International Reference Version*). I codici decimali da 0 a 31 e il 127 sono caratteri non stampabili (*codici di controllo*). Il 32 corrisponde al carattere di "spazio". I codici dal 32 al 126 sono caratteri stampabili.

Legenda:

• PR - La rappresentazione stampabile del carattere, se presente
• Dec - Il codice decimale del carattere
• Oct - Il codice ottale del carattere
• Hex - Il codice esadecimale del carattere

Non stampabili

Binario	Oct	Dec	Hex	Abbr	PR	CS	CEC	Descrizione
000 0000	000	0	00	NUL	□	^@	\0	Null character
000 0001	001	1	01	SOH	□	^A		Start of Header
000 0010	002	2	02	STX	□	^B		Start of Text
000 0011	003	3	03	ETX	□	^C		End of Text
000 0100	004	4	04	EOT	□	^D		End of Transmission
000 0101	005	5	05	ENQ	□	^E		Enquiry
000 0110	006	6	06	ACK	□	^F		Acknowledgment
000 0111	007	7	07	BEL	□	^G	\a	Bell
000 1000	010	8	08	BS	□	^H	\b	Backspace
000 1001	011	9	09	HT	□	^I	\t	Horizontal Tab
000 1010	012	10	0A	LF	□	^J	\n	Line feed
000 1011	013	11	0B	VT	□	^K	\v	Vertical Tab

000 1100	014	12	0C	FF	□	^L	\f	Form feed
000 1101	015	13	0D	CR	□	^M	\r	Carriage return
000 1110	016	14	0E	SO	□	^N		Shift Out
000 1111	017	15	0F	SI	□	^O		Shift In
001 0000	020	16	10	DLE	□	^P		Data Link Escape
001 0001	021	17	11	DC1	□	^Q		Device Control 1 (oft. XON)
001 0010	022	18	12	DC2	□	^R		Device Control 2
001 0011	023	19	13	DC3	□	^S		Device Control 3 (oft. XOFF)
001 0100	024	20	14	DC4	□	^T		Device Control 4
001 0101	025	21	15	NAK	□	^U		Negative Acknowledgement
001 0110	026	22	16	SYN	□	^V		Synchronous Idle
001 0111	027	23	17	ETB	□	^W		End of Trans. Block
001 1000	030	24	18	CAN	□	^X		Cancel
001 1001	031	25	19	EM	□	^Y		End of Medium
001 1010	032	26	1A	SUB	□	[^Z		Substitute
001 1011	033	27	1B	ESC	□	^[	\e	Escape
001 1100	034	28	1C	FS	□	^\		File Separator
001 1101	035	29	1D	GS	□	^]		Group separator
001 1110	036	30	1E	RS	□	^^		Record Separator
001 1111	037	31	1F	US	□	^_		Unit Separator

Stampabili

Binario	Oct	Dec	Hex	Glifo	
010 0000	040	32	20	Space	
010 0001	041	33	21	[[Punto esclamativo	]]
010 0010	042	34	22	"	
010 0011	043	35	23	#	
010 0100	044	36	24	$	
010 0101	045	37	25	%	
010 0110	046	38	26	&	
010 0111	047	39	27	'	
010 1000	050	40	28	(	
010 1001	051	41	29	)	
010 1010	052	42	2A	*	
010 1011	053	43	2B	+	
010 1100	054	44	2C	,	
010 1101	055	45	2D	-	

010 1110	056	46	2E	.	
010 1111	057	47	2F	/	
011 0000	060	48	30	0	
011 0001	061	49	31	1	
011 0010	062	50	32	2	
011 0011	063	51	33	3	
011 0100	064	52	34	4	
011 0101	065	53	35	5	
011 0110	066	54	36	6	
011 0111	067	55	37	7	
011 1000	070	56	38	8	
011 1001	071	57	39	9	
011 1010	072	58	3A	:	
011 1011	073	59	3B	;	
011 1100	074	60	3C	<	
011 1101	075	61	3D	=	
011 1110	076	62	3E	>	
011 1111	077	63	3F	?	

Binario	Oct	Dec	Hex	Glifo
100 0000	100	64	40	@
100 0001	101	65	41	A
100 0010	102	66	42	B
100 0011	103	67	43	C
100 0100	104	68	44	D
100 0101	105	69	45	E
100 0110	106	70	46	F
100 0111	107	71	47	G
100 1000	110	72	48	H
100 1001	111	73	49	I
100 1010	112	74	4A	J
100 1011	113	75	4B	K
100 1100	114	76	4C	L
100 1101	115	77	4D	M
100 1110	116	78	4E	N
100 1111	117	79	4F	O
101 0000	120	80	50	P
101 0001	121	81	51	Q
101 0010	122	82	52	R
101 0011	123	83	53	S

101 0100	124	84	54	T
101 0101	125	85	55	U
101 0110	126	86	56	V
101 0111	127	87	57	W
101 1000	130	88	58	X
101 1001	131	89	59	Y
101 1010	132	90	5A	Z