Protocollo MODBUS

PREMESSA

Un protocollo di comunicazione, non è altro che un insieme di regole che determinano come deve avvenire la comunicazione tra dispositivi.

Per poter comunicare tra di loro, due o più dispositivi, hanno bisogno essenzialmente di tre cose:

- un mezzo di trasporto del segnale che in genere può essere un cavo, una fibra ottica o le onde radio,
- dei dispositivi elettronici che consentono l'invio e la ricezione dei dati su questo canale trasmissivo (antenne, trasmettitori e ricevitori)
- delle regole che definiscono come devono essere i segnali elettrici, o come deve avvenire lo scambio di dati a livello software.

Ormai la trasmissione di dati, avviene sempre in formato digitale, inviando cioè l'informazione sottoforma di bit (0 ed 1) utilizzando svariate tecniche trasmissive.

Bisogna però differenziare le tecniche di trasmissione del segnale, dalle regole imposte da un protocollo che operano ad un livello più software e cioè al livello applicazione del modello ISO/OSI.

Ci sono tantissimi protocolli di comunicazione e molti di loro operano sulle reti LAN che tutti noi utilizziamo per collegare insieme vari computer, uno dei più storici ancora oggi utilizzato, è il protocollo Modbus, utilizzato in ambito industriale e non, ed utile anche ad un approccio didattico nel mondo delle telecomunicazioni.

MODBUS

Il protocollo MODBUS nasce nel 1979 dall'azienda Modicon (da qui il nome MODBUS) per la comunicazione con i propri PLC.

Nato per essere utilizzato in ambito industriale, con il tempo questo protocollo è stato utilizzato anche in altri settori diventando uno dei più diffusi, ed ancora oggi, nonostante abbia ormai più di 40 anni, lo troviamo disponibile in molti dispositivi, come pannelli operatore, PLC ecc.…

Il MODBUS è un protocollo libero da royalties ed ha una struttura molto semplice che lascia spazio anche a proprie personalizzazioni.

La comunicazione secondo questo protocollo, può avvenire su base hardware seriale RS485, RS232 o anche su rete Ethernet.

A seconda del formato con cui vengono trasmessi i dati, il protocollo si divide in MODBUS RTU, dove i dati vengono trasmessi in formato esadecimale, o MODBUS ASCII, dove i dati transitano invece in formato ASCII.

Anche il controllo errori si differenzia nei due casi, nel MODBUS RTU viene utilizzato un CRC (Cyclic Redundancy Check, controllo di ridondanza ciclico) inviato in coda ai comandi, mentre nel MODBUS ASCII un LRC (longitudinal Redundancy Check, controllo di ridondanza orizzontale) sempre inviato dopo i comandi.

Il MODBUS su base Ethernet, si chiama MODUBS TCP che utilizza come nel caso dell'RTU il formato esadecimale, ma i pacchetti vengono inglobati in pacchetti TCP/IP.

Altre differenze tra il MODBUS ASCII e RTU sono rappresentate nelle seguenti tabelle, ma saranno più chiare al termine del tutorial, dopo aver compreso la struttura dei messaggi del Modbus RTU.

Confronto tra Modbus/Ascii and Modbus/RTU

	Modbus/Ascii	Modbus/RTU
Caratteri usati	Simboli Ascii delle cifre esadecimali 0, 1, 2, ... E, F	Valori binari compresi tra 0 e 255
Inizio frame	Carattere ':' la durata di un byte	Silenzio di 3,5 volte
Fine frame	Sequenza CR/LF la durata di un byte	Silenzio di 3,5 volte
Bit iniziale	1	1
Bit dati	7	8
Pausa nel messaggio	1 sec	1,5 volte la durata di un byte
Controllo di ridondanza	LRC Longitudinal Redundancy Check	CRC Cyclic Redundancy Check

Frame Modbus/Ascii

Inizio	Indirizzo	Codice funzione	Campo dati	LRC check	Fine
1 carattere	2 caratteri	2 caratteri	N caratteri	2 caratteri	CR+LF

Frame Modbus/RTU

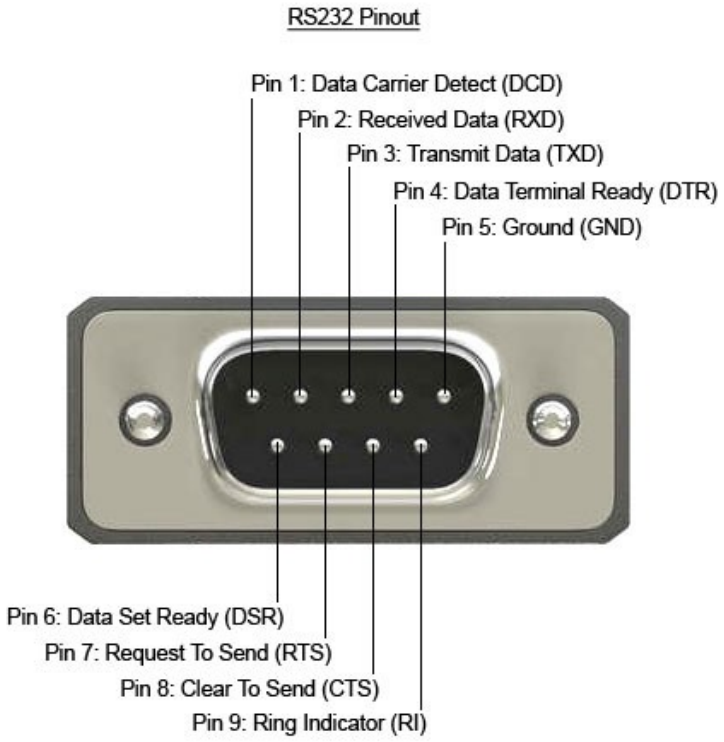
Inizio	Indirizzo	Codice funzione	Campo dati	LRC check	Fine
Silenzio di durata 3,5 byte	1 byte	1 byte	N byte	2 byte	Silenzio di durata 3,5 byte

Prima di proseguire nella descrizione del protocollo, analizziamo i due standard RS232 e RS485 utilizzati dal MODBUS.

Standard RS232

Lo standard prevede una comunicazione seriale asincrona tra due dispositivi, mediante due pin RXD e TXD.
Viene utilizzato un connettore a vaschetta a 9 poli, con la seguente piedinatura:

Num. pin	Descrizione
1	DCO – Data Carrier Detect
2	**RXD – Received Data**
3	**TXD – Transmit Data**
4	DTR – Data Terminal Ready
5	**GND - Ground**
6	DSR – Data Set Ready
7	RTS – Request To Send
8	CTS – Clear To Send
9	RI – Ring Indicator
shield	Schermo connettore



Il collegamento minimo per lo standard seriale, prevede l'utilizzo delle sole due linee RXD e TXD, in questo modo possiamo collegare due dispositivi utilizzando un cavo incrociato, dove il pin RX del primo dispositivo va a collegarsi con il pin TX del secondo e viceversa.

Lo standard RS232 nasce per far comunicare due dispositivi chiamati DTE (Data Terminal Equipment) e DCE (Data Comunication Equipment) come ad esempio il collegamento tra PC e Modem seriale.

In questo caso l'inversione dei pin RX e TX non è necessaria, in quanto nel DCE il funzionamento dei pin RX e TX viene invertito nel dispositivo.

Esiste inoltre la possibilità di utilizzare un connettore a 25 pin anziché 9 pin.

L'incrocio dei pin RX e TX dipende se il collegamento avviene tra un DTE e DCE o tra due dispositivi dello stesso tipo.

DTE Device (Computer) — DB9 — **DTE to DCE Connections** — DCE Device (Modem) — DB9

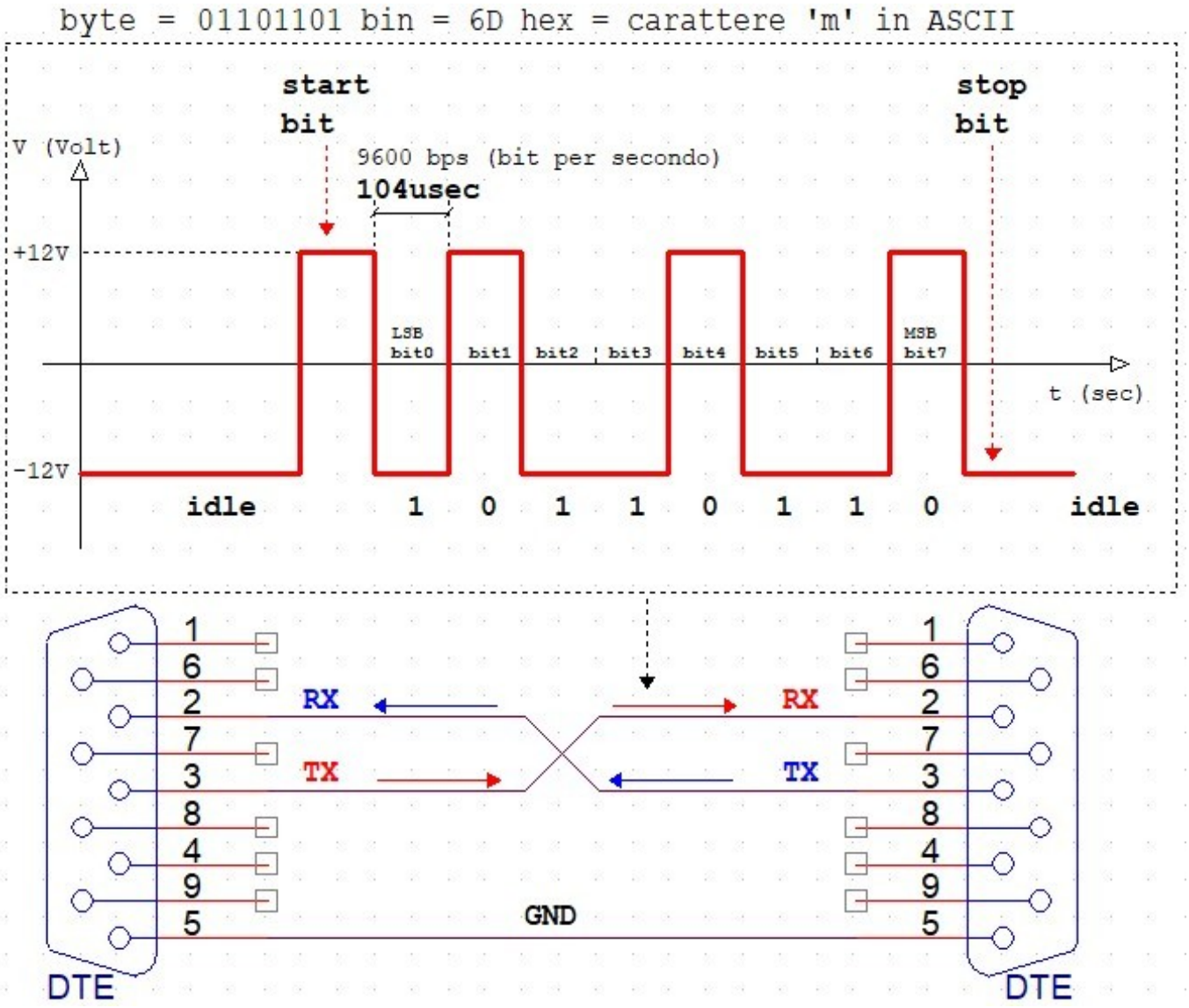
Pin# DB9	RS-232 Signal Names		Signal Direction	Pin# DB9	RS-232 Signal Names	
#1	Carrier Detector (DCD)	CD	←	#1	Carrier Detector (DCD)	CD
#2	Receive Data (Rx)	RD	←	#2	Receive Data (Rx)	RD
#3	Transmit Data (Tx)	TD	→	#3	Transmit Data (Tx)	TD
#4	Data Terminal Ready	DTR	→	#4	Data Terminal Ready	DTR
#5	Signal Ground/Common (SG)	GND	—	#5	Signal Ground/Common (SG)	GND
#6	Data Set Ready	DSR	←	#6	Data Set Ready	DSR
#7	Request to Send	RTS	→	#7	Request to Send	RTS
#8	Clear to Send	CTS	←	#8	Clear to Send	CTS
#9	Ring Indicator	RI	←	#9	Ring Indicator	RI
Soldered to DB9 Metal - Shield		FGND	—	Soldered to DB9 Metal - Shield		FGND

A destra due esempi di collegamento tra DTE e DCE, o tra DCE e DCE, o anche tra due DTE, come ad esempio tra due PC.

DCE Device (Modem) — DB9 — **DCE to DCE Connections** — DCE Device (Modem) — DB9

Pin# DB9	RS-232 Signal Names		Signal Direction	Pin# DB9	RS-232 Signal Names	
#1	Carrier Detector (DCD)	CD		#1	Carrier Detector (DCD)	CD
#2	Receive Data (Rx)	RD	↔	#2	Receive Data (Rx)	RD
#3	Transmit Data (Tx)	TD	↔	#3	Transmit Data (Tx)	TD
#4	Data Terminal Ready	DTR	↔	#4	Data Terminal Ready	DTR
#5	Signal Ground/Common (SG)	GND	—	#5	Signal Ground/Common (SG)	GND
#6	Data Set Ready	DSR	↔	#6	Data Set Ready	DSR
#7	Request to Send	RTS	↔	#7	Request to Send	RTS
#8	Clear to Send	CTS	↔	#8	Clear to Send	CTS
#9	Ring Indicator	RI		#9	Ring Indicator	RI
Soldered to DB9 Metal - Shield		FGND	—	Soldered to DB9 Metal - Shield		FGND

Considerando il collegamento più semplificato tra due DTE, dovendo inviare un byte avremmo la seguente situazione:

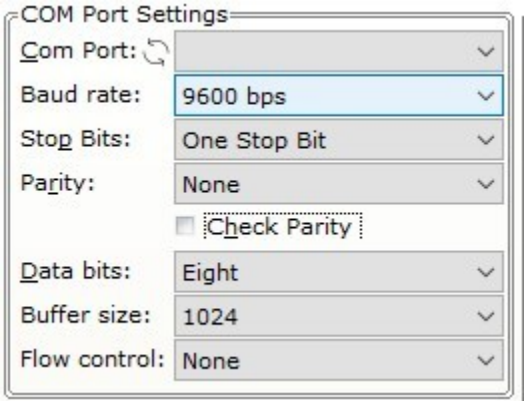


Il segnale sulla linea TX viene tenuto normalmente ad un livello di IDLE che corrisponde a -12Volt, quando invece la trasmissione inizia, viene inviato lo start bit che corrisponde ad un livello di +12Volt, e successivamente vengono inviati gli 8 bit considerando il livello logico 0 pari a +12Volt ed il livello logico 1 pari a -12Volt.
Al termine del byte viene inviato un bit di stop corrispondente a -12Volt e la linea rimane poi nello stato di IDLE.
Il livello logico **0** viene definito **space**, il livello logico **1** viene definito **mark**.
Lo standard prevede l'invio anche di un numero diverso di bit (5,6,7,8,9) o di un numero diverso di bit di stop (1, 2 o 1,5). Inoltre può essere prevista l'aggiunta un bit di parità per il controllo della correttezza del byte ricevuto, nello specifico ci sono le seguenti opzioni:

- nessuna parità — *non viene aggiunto alcun bit*
- pari (even) — *viene aggiunto un bit in modo che il numero di bit ad 1 (mark) sia pari*
- dispari (odd) — *viene aggiunto un bit in modo che il numero di bit ad 1 (mark) sia dispari*
- mark — *viene aggiunto sempre un bit di mark*
- space — *viene aggiunto sempre un bit di space*

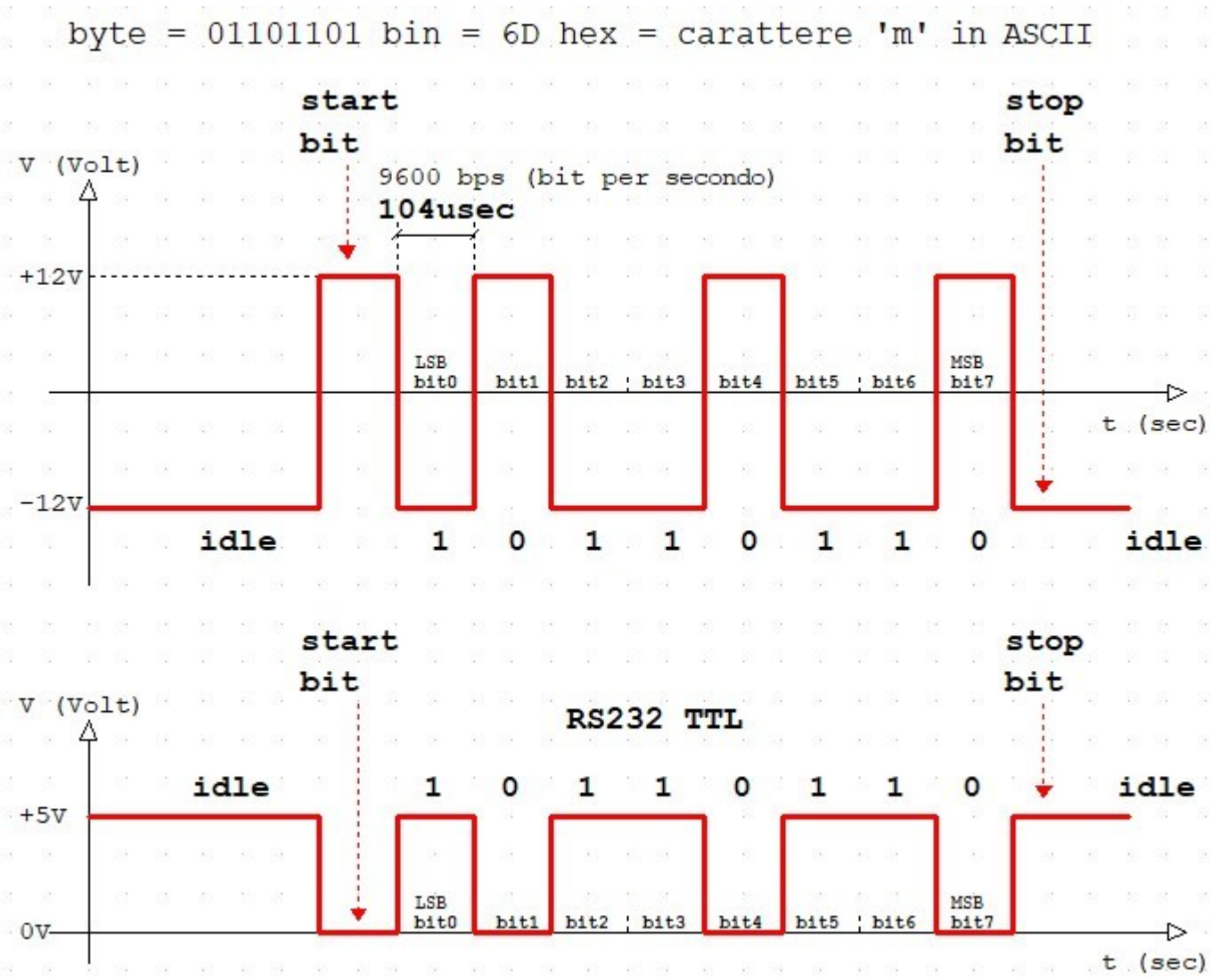
Essendo una comunicazione di tipo asincrono, occorre definire la velocità della comunicazione sui due dispositivi comunicanti. La velocità viene definita in bit per secondo (Baud rate) alcune velocità possibili sono: 1200, 2400, 4800, 9600, 14400,19200,38400, 57600, 115200.

Altro parametro che si può impostare in una comunicazione seriale, è il controllo di flusso (handshaking, stretta di mano) che può avvenire in maniera hardware utilizzando gli altri bit della seriale, in maniera software mediante l'invio di ulteriori bit, o può non essere utilizzato.

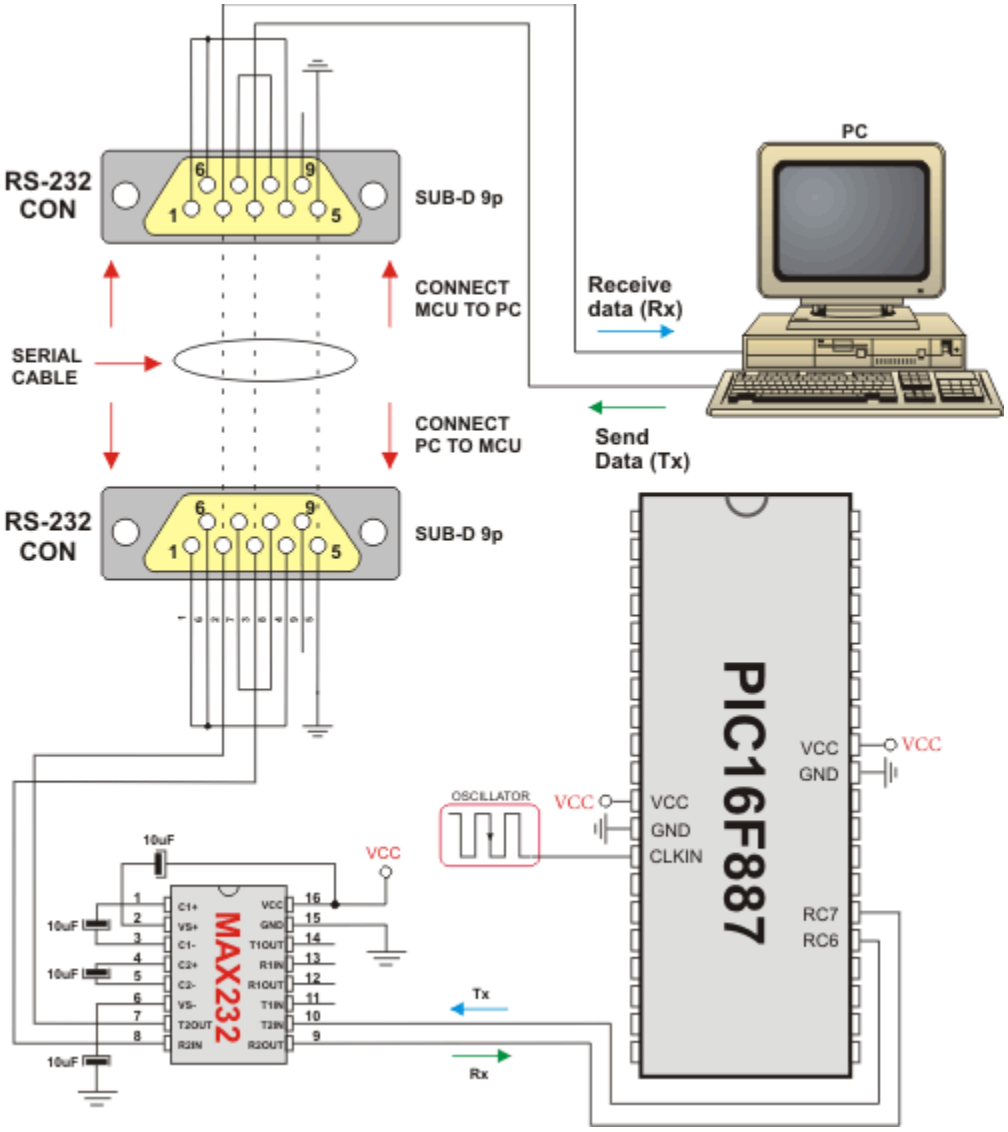


In un PC le porte seriali vengono definite con la sigla COM e numerate in base alla quantità di porte presenti (COM1, COM2 ecc…) la distanza massima consigliata tra due dispositivi è di 15metri.

Lo standard seriale può essere anche utilizzato con livelli di segnale TTL (ad esempio nei microcontrollori) in questo caso valgono tutte le regole precedentemente descritte, fatta eccezione per il livello di tensione che necessariamente deve essere 0Volt per lo zero logico e 5Volt per l'uno logico. Pertanto occorre solamente traslare il livello di tensione, ed il byte precedentemente visto si presenta su una linea seriale TTL nel seguente modo.



Esistono in commercio dei circuiti integrati come il MAX232, che consentono la trasposizione dei livelli di tensione, consentendo il collegamento tra un dispositivo che funziona con livelli di tensione 0-5V e lo standard ufficiale che prevede livelli diversi.
La seguente immagine è tratta dall'help del compilatore mikroC della ditta Mikroelektronika (www.mikroe.com) che fornisce tutte le librerie necessarie per instaurare una comunicazione seriale RS232, insieme a molte alte librerie per i diversi standard e protocolli di comunicazione.



Copyright (c) 2002-2019 mikroElektronika. All rights reserved.

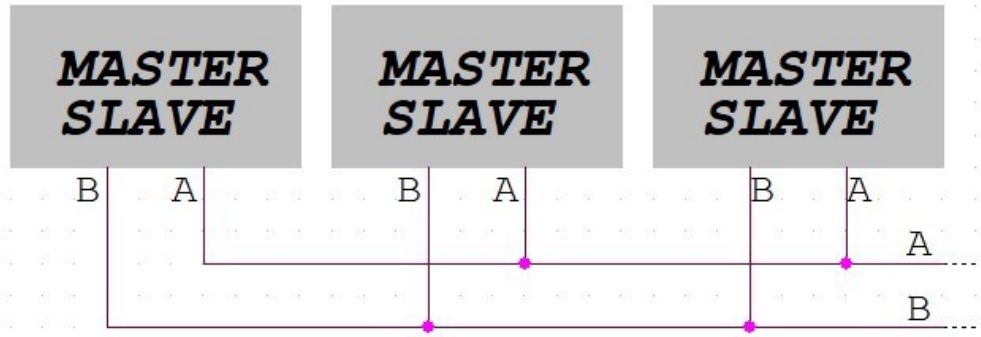
Questo standard di comunicazione non è però adatto ad una comunicazione tra più dispositivi, come ad esempio in un sistema Master-Slave, dove gli slave possibili sono in genere più di uno.
Il suo utilizzo è limitato alla comunicazione tra solo due dispositivi a meno che non vengano adottate soluzioni particolari normalmente non utilizzate.

Per collegare più dispositivi, occorre pertanto un sistema di comunicazione a bus, come ad esempio lo standard RS485.

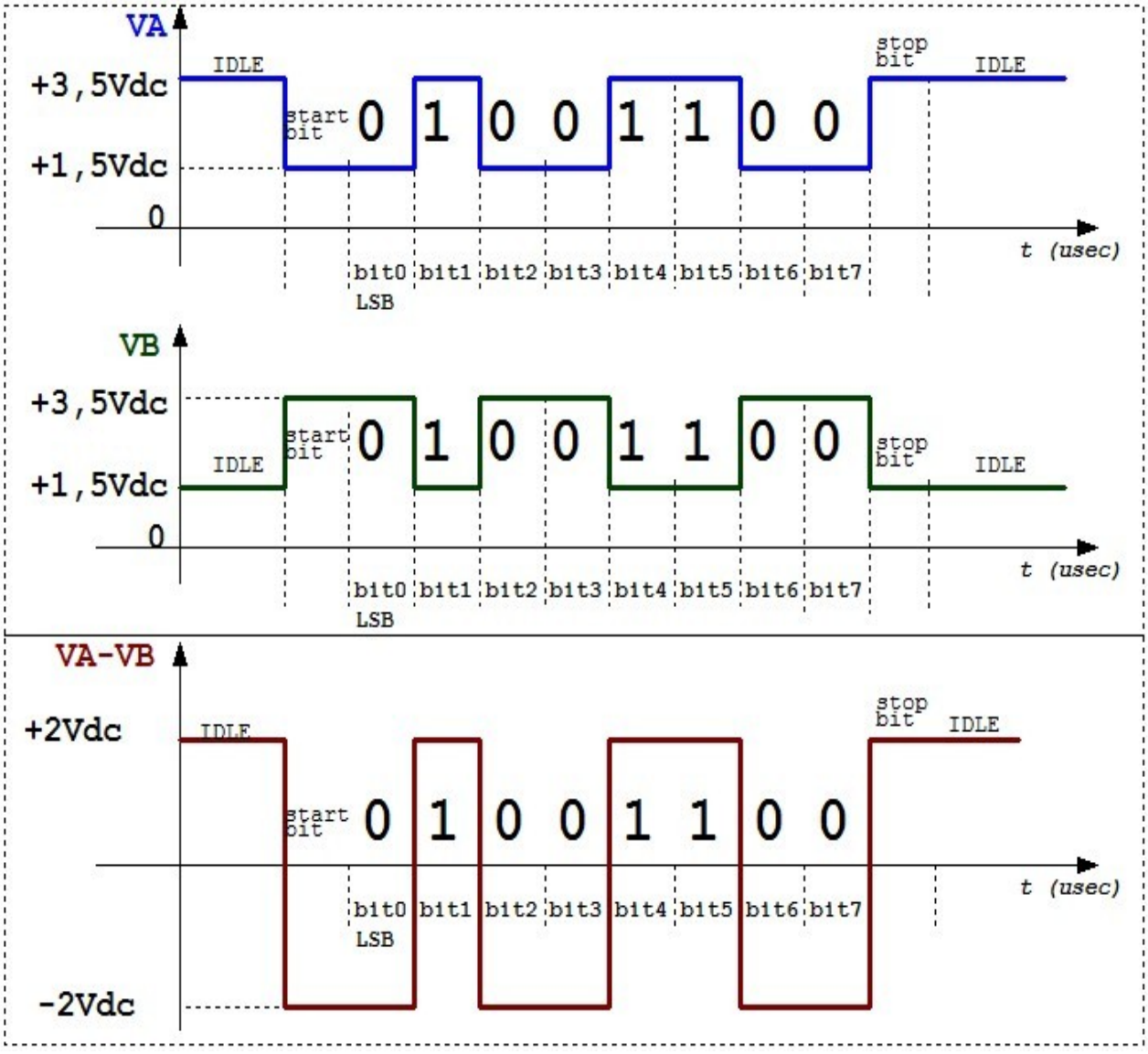
Standard RS485

A differenza della RS232 dove il dato transita su due canali differenti RX e TX, il cui valore di tensione è riferito allo stesso potenziale GND, nella comunicazione RS485, si prevede il transito dei dati su un bus composto da due collegamenti dove il valore di tensione è di tipo differenziale.

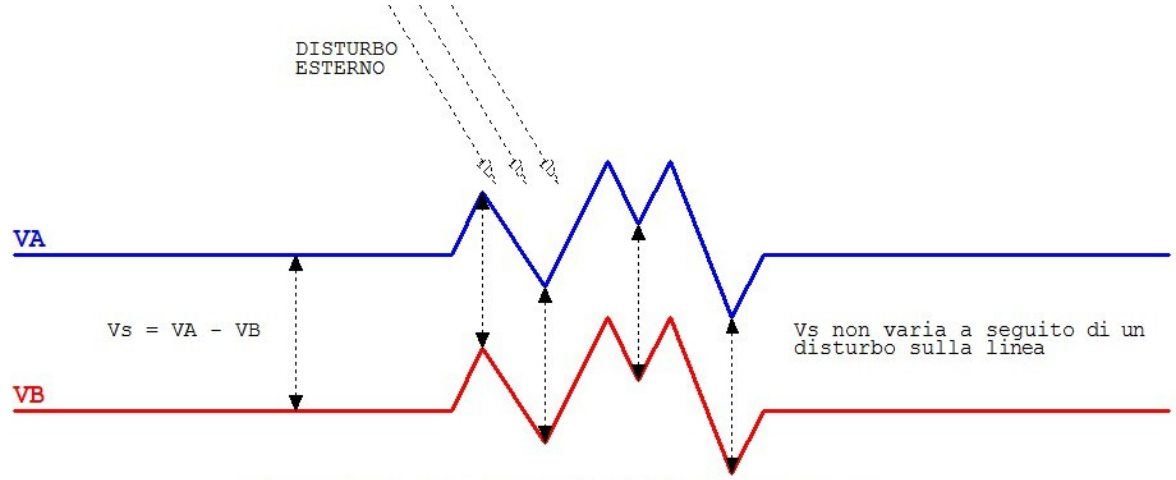
Questo tipo di comunicazione si presta per un collegamento tra più dispositivi, come nel caso di un collegamento Master-Slave.



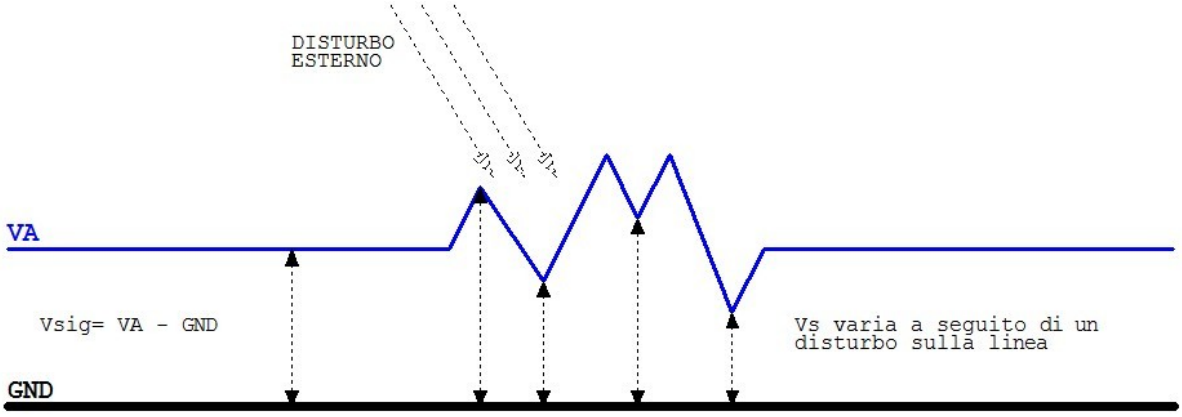
Come nella RS232 si prevedono due canali RX e TX oltre ad altri canali utilizzati per il controllo del flusso come RTS e CTS, tutti operanti con segnali differenziali, dove il valore di tensione tra i due terminali + e – corrisponde ad un valore positivo o negativo a seconda del livello logico 0 o 1. La tensione può variare da 1,5 a 3,5Volt.



Come detto in precedenta, in un segnale differenziale,il livello di tensione di un terminale è riferito all'altro terminale, ed i due terminali viaggiano in genere su collegamenti twistati (incrociati) in queso modo un eventuale disturbo di origine elettrica, andrebbe a modificare il valore di tensione di entrambi i terminali, non influendo sul valore della loro differenza. Una situazione completamente diversa da quella del segnale della RS232, un segnale sbilanciato, definito anche Single Ended, riferito alla GND. In questo caso un eventuale disturbo andrebbe a modificare il valore di tensione della linea causando possibili errori di trasmissione.



SEGNALE DIFFERENZIALE



SEGNALE SBILANCIATO
(Single Ended)

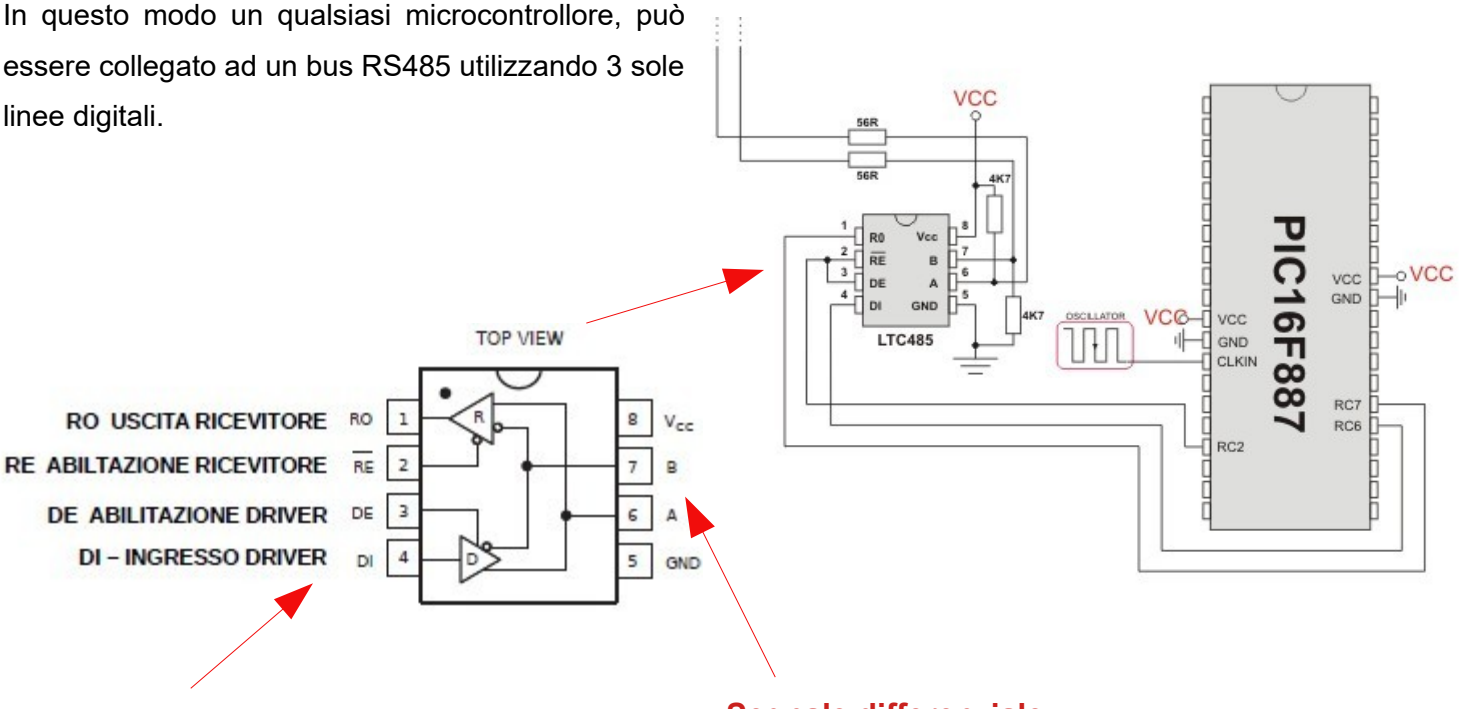
Anche in questo caso esiste uno standard nel pin-out di un connettore a vaschetta a 9 poli, anche se questo tipo di connettore viene utilizzato normalmente in una connessione punto-punto tra due dispositivi.

Num. pin	Segnale	Descrizione
1	TXD –	Transmit Data
2	TXD +	
3	RTS -	Request To Send
4	RTS +	
5	GND	Ground
6	RXD –	Received Data
7	RXD +	
8	CTS-	Clear To Send
9	CTS+	
shield		Schermo connettore

Come nella precedente comunicazione esistono in commercio dei dispositivi in grado di gestire il bus 485, come ad esempio l'integrato MAX485 o LTC485.

Il segnale digitale TX viene inviato sul pin 4 DI ed il segnale RX viene ricevuto dal pin 1 R0. I due pin RE e DE permettono di scegliere la direzione del dato, se in lettura o scrittura ed in genere possono essere collegati insieme come nell'esempio di seguito, tratto sempre dall'help del compilatore mikroC prodotto dalla mikroelektronika.

In questo modo un qualsiasi microcontrollore, può essere collegato ad un bus RS485 utilizzando 3 sole linee digitali.



Segnali riferiti a GND, lato microcontrollore

Segnale differenziale, BUS di comunicazione

Dal datasheet dello stesso integrato possiamo vedere un esempio di collegamento con più dispositivi.

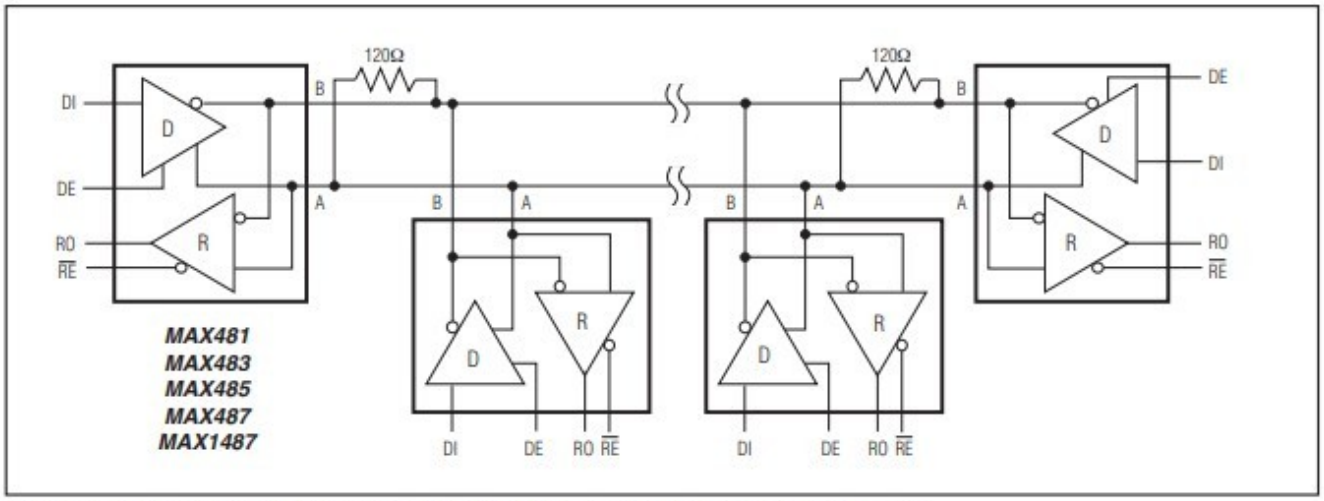
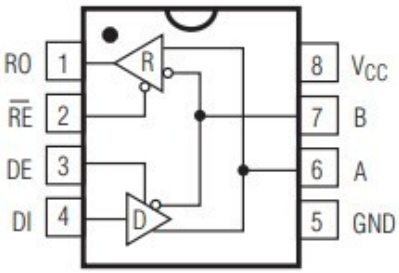


Figure 21. MAX481/MAX483/MAX485/MAX487/MAX1487 Typical Half-Duplex RS-485 Network

Come si può notare dall'immagine sopra, il bus è unico e comune a tutti i dispositivi, ed essendo costituito da un solo canale differenziale, la comunicazione può essere solo HALF-DUPLEX (il dato transita o in una direzione o nell'altra, mai contemporaneamente).



Il circuito integrato MAX488 invece ha al suo interno due canali differenziali indipendenti.

In questo modo si possono ottenere due canali indipendenti, uno per la tramissione ed uno per la ricezione, in modo da avere la possibilità di una comunicazione FULL-DUPLEX (il dato può transitare contemporaneamente in entrambe le direzioni).

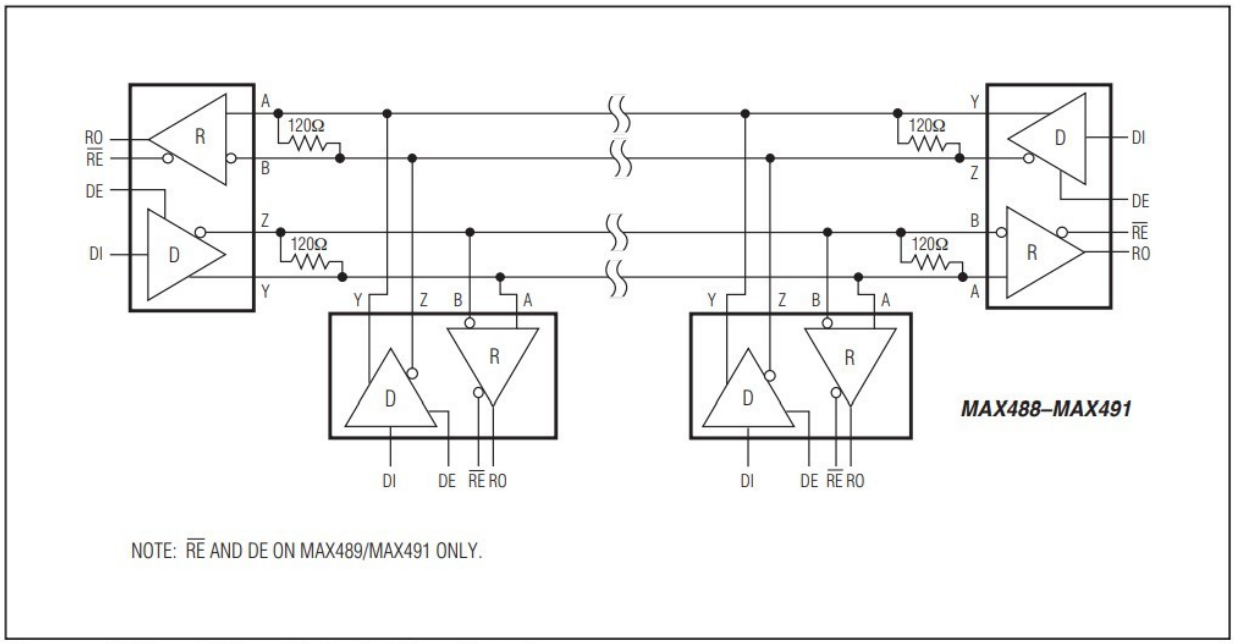


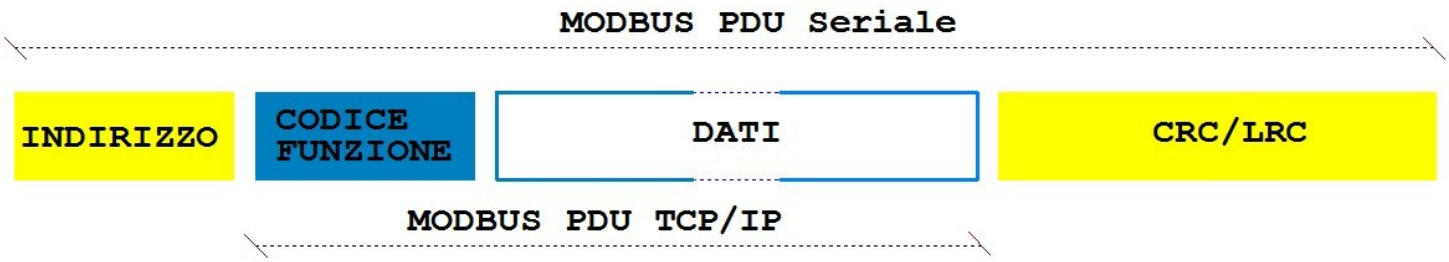
Figure 22. MAX488–MAX491 Full-Duplex RS-485 Network

Il vantaggio di una comunicazione su bus RS485, è essenzialmente una maggiore immunità ai disturbi, in quanto vengono utilizzati dei canali trasmissivi differenziali. Questo consente anche il raggiungimento di distanze molto maggiori rispetto alla RS232, distanze dell'ordine dei 1200metri con velocità elevate fino a 20Mbps.
Inoltre la comunicazione RS485, consente un collegamento tra più dispositivi e non solo tra un DTE-DCE o DTE-DTE, come visto nella comunicazione RS232.
Nel caso del MODBUS, lo standard RS485 offre pertanto una valida struttura hardware su cui appoggiarsi, dal punto di vista logico inoltre, la comunicazione RS485 e la RS232 presentano molte analogie, che consentono un loro facile utilizzo tramite le normali periferiche UART presenti nei microcontrollori.
Dopo questa analisi degli standard hardware utilizzati, andiamo ad approfondire la struttura del protocollo MODBUS.

Struttura del Protocollo MODBUS

Questo protocollo, definisce le modalità di comunicazione tra un master ed uno o più slave. Si possono connettere fino a 247 slave su un bus comune, limitazione questa di tipo logico, ma fisicamente il numero massimo di slave dipende dallo standard hardware utilizzato. Ad esempio nella RS485 si possono collegare al massimo 31 dispositivi, ma inserendo nel bus uno o più ripetitori nel bus, si può espandere il numero di dispositivi fino al massimo logico di 247 dispositivi.
La trasmissione viene avviata dal MASTER, e può essere indirizzata ad un singolo SLAVE o in **broadcast** a tutti gli SLAVE presenti nel bus.
Come detto inizialmente, a seconda del tipo di MODBUS e cioè ASCII o RTU, il controllo degli errori avviene con LRC o CRC, in caso di MODBUS TCP invece il controllo errori è implementato nello stesso protocollo TCP.
Il formato del messaggio del MODBUS (PDU Protocol Data Unit) è il seguente:

- **Indirizzo dello slave** (nel MODBUS TCP non è presente, in quanto l'indirizzo IP viene gestito dallo stesso protocollo TCP/IP)
- **Codice funzione da eseguire**
- **Dati** (indirizzo e quantità)
- **CRC o LRC** (nel MODBUS TCP non è presente)



Indirizzo dello slave

Gli indirizzi vanno da 1 a 247, il valore 0 su questo campo, indica che il messaggio è indirizzato a tutti gli SLAVE (indirizzo di broadcast). Si ha una comunicazione **UNICAST**, quando il Master invia una richiesta ad un singolo Slave, pertanto nel pacchetto dati inviato troveremo l'indirizzo univoco dello Slave selezionato.
Si ha una comunicazione **BROADCAST**, quando il Master invia una richiesta a tutti gli Slave, in questo caso nel campo indirizzo ci sarà il valore 0.

Codice funzione

Nella seguente tabella sono riassunte sinteticamente le funzioni previste dal protocollo MODBUS.

Codice	Descrizione	Accesso	Descrizione
01	Read Coil Status	bit	Legge lo stato (ON/OFF) di una o più variabili binarie
02	Read Input Status	bit	Identica alla precedente, utilizzata per leggere ingressi fisici
03	Read Holding Registers	16 bit	Legge il valore di un registro a 16 bit
04	Read Input registers	16 bit	Identica alla precedente, utilizzata per leggere ingressi fisici
05	Force Single Coil	bit	Scrivi lo stato (ON/OFF) di una variabile binaria
06	Preset Single register	16 bit	Scrivi lo stato di un registro a 16 bit.
07	Read Status	diagnostics	Legge lo stato di 8 bit, in maniera compatta con l'invio di un byte. Gli 8 bit possono rappresentare lo stato del dispositivo remoto, il significato dipende comunque dal dispositivo.
08	Diagnostic	diagnostics	Usata per testare la comunicazione
11	Get Com event counter	diagnostics	Usata per conteggio eventi, o parola di stato del dispositivo remoto
12	Get Com Event Log	diagnostics	Simile alla precedente
15	Force multiple Coils	bit	Scrivi lo stato (ON/OFF) di più variabili binarie
16	Preset Multiple Registers	16 bit	Scrivi lo stato di più registri a 16 bit.
17	Report Server ID	diagnostics	Usata per ricevere informazioni dal dispositivo remoto
20	Read File record	File record	Lettura di record di file
21	Write File record	File record	Scrittura di record di file
22	Mask Write Register	16 bit	Modifica lo stato di un registro con un'operazione che utilizza una maschera AND o OR
23	Read/Write Multiple Registers	16 bit	Scrittura e lettura su più registri
24	Read FIFO queue	16 bit	Lettura di un registro FIFO del dispositivo remoto
43	Read device Identification	diagnostics	Legge le informazioni di identificazione del dispositivo remoto

I dati inviati dopo il codice funzione, dipendono ovviamente dalla funzione, come anche la risposta del dispositivo. Analizziamo di seguito in maniera più approfondita le principali funzioni, che nella maggior parte dei casi sono le seguenti: 01 – 02 – 03 – 04 – 05 – 06 – 07 – 08 – 11 -12 - 15 – 16 - 23.

Per ogni informazione, e per il funzionamento delle altre funzioni, occorre leggere attentamente le specifiche riportate nei documenti ufficiali disponibili sul sito https://modbus.org/ nella sezione **Technical Resources**.

FUNZIONE 01 – READ COIL STATUS

Con questa funzione vengono letti gli stati di singoli bit (coil) massimo 2000 bit per ogni lettura su 65535 possibili.

RICHIESTA ALLO SLAVE - 8 Bytes							
INDIRIZZO SLAVE da 1 a 247	***FUNZIONE***	***INDIRIZZO DEL DATO da 0000 a FFFF da 0000 a 65535***		***NUMERO DI BIT da 0001 a 07D0 da 0001 a 2000***		***CRC***	
		HIGH	*LOW*	*HIGH*	*LOW*	*HIGH*	*LOW*
0F	**01**	**00**	**03**	**00**	**14**	**CD**	**2B**

Il valore di ogni Byte è in esadecimale, pertanto tradotto in decimale abbiamo che:

Indirizzo dispositivo=15 (0Fhex) - Indirizzo del dato=3 *(il primo registro ha indirizzo 0)* - Numero di bit= 20 (14hex)

Perciò in questo caso viene chiesto al dispositivo con indirizzo 15 (0F) il valore di 20 bit partendo dal quarto bit (indirizzo 3). Il metodo per il calcolo del CRC verrà approfondito successivamente.

RISPOSTA DELLO SLAVE - lunghezza variabile							
INDIRIZZO SLAVE da 1 a 247	***FUNZIONE***	***NUMERO DI BYTE DEL DATO max 250 (FA in hex)***	***VALORE DEI BIT – DATO RICHIESTO***			***CRC***	
			1*byte bit da 4 a 11	***2*byte bit da 12 a 19***	***3*byte bit da 20 a 23***	*HIGH*	*LOW*
0F	**01**	**03**	**04**	**01**	**00**	**7D**	**31**

La situazione dei bit nella memoria dello SLAVE secondo il precedente esempio è la seguente:

BYTE	INDIRIZZO BIT	VALORE
	0	
	1	
	2	
1°BYTE 0000 0100 0x04	3	0
	4	0
	5	1
	6	0
	7	0
	8	0
	9	0
	10	0
2°BYTE 0000 0001 0x01	11	1
	12	0
	13	0
	14	0
	15	0
	16	0
	17	0
	18	0

BYTE	INDIRIZZO BIT	VALORE
3°BYTE 0000 0000 0x00	19	0
	20	0
	21	0
	22	0
	23	0
	24	0
	25	0
	26	0
	27	
	28	
	29	
	30	
	31	
	32	
	33	
	34	
	35	
	36	
	37	

Come si può notare sopra, dovendo leggere 20 bit vengono inviati 3 byte che contengono 24 bit, i 4 bit più significativi dell'ultimo byte vengono messi a zero in quanto non richiesti.

Un altro esempio potrebbe essere il seguente, leggere 32 bit (00 20 hex) partendo dall'indirizzo 13 (0C hex)

RICHIESTA ALLO SLAVE - 8 Bytes							
INDIRIZZO SLAVE	***FUNZIONE***	***INDIRIZZO DEL DATO***		***NUMERO DI BIT***		***CRC***	
		HIGH	*LOW*	*HIGH*	*LOW*	*HIGH*	*LOW*
0F	**01**	**00**	**0C**	**00**	**20**	**FC**	**FF**

RISPOSTA DELLO SLAVE								
INDIRIZZO SLAVE	***FUNZIONE***	***NUMERO DI BYTE DEL DATO***	***VALORE DEI BIT – DATO RICHIESTO***				***CRC***	
			1*byte bit da 12 a 19	***2*byte bit da 20 a 27***	***3*byte bit da 28 a 35***	***4*byte bit da 36 a 43***	*HIGH*	*LOW*
0F	**01**	**04**	**35**	**64**	**0D**	**18**	**5E**	**98**

BYTE	INDIRIZZO BIT	VALORE
	0	
	1	
	2	
	3	
	4	
	5	
	6	
	7	
	8	
	9	
	10	
	11	
1°BYTE 0011 0101 0x35	12	1
	13	0
	14	1
	15	0
	16	1
	17	1
	18	0
	19	0
2°BYTE 0110 0100 0x64	20	0
	21	0
	22	1
	23	0
	24	0
	25	1
	26	1
	27	0

BYTE	INDIRIZZO BIT	VALORE
3°BYTE 0000 1101 0x5D	28	1
	29	0
	30	1
	31	1
	32	0
	33	0
	34	0
	35	0
4°BYTE 0001 1000 0x18	36	0
	37	0
	38	0
	39	1
	40	1
	41	0
	42	0
	43	0
	44	
	45	
	46	
	47	
	48	
	49	
	50	
	51	
	52	
	53	
	54	
	55	

FUNZIONE 02 – READ INPUT STATUS

La funzione è praticamente identica alla precedente, ma in questo caso i bit letti rappresentano gli ingressi fisici del dispositivo e non una zona di memoria.

FUNZIONE 03 – READ HOLDING REGISTER

Con questa funzione vengono letti dei blocchi contigui di registri interni a 16 bit dal dispositivo Slave.

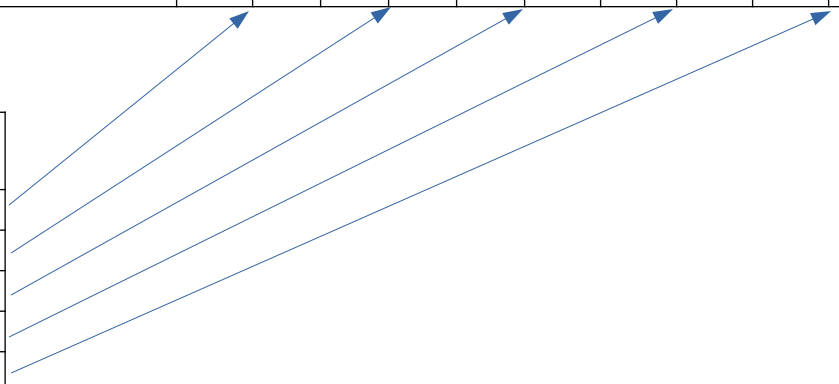
RICHIESTA ALLO SLAVE - 8 Bytes							
INDIRIZZO SLAVE	***FUNZIONE***	***INDIRIZZO INIZIALE***		***NUMERO DI REGISTRI***		***CRC***	
		HIGH	*LOW*	*HIGH*	*LOW*	*HIGH*	*LOW*
0F	**03**	**00**	**00**	**00**	**05**	**84**	**E7**

In questo caso viene chiesto allo slave con indirizzo 15 (0F hex) di fornire il valore di 5 registri partendo dal registro con indirizzo 0.
Al massimo possono essere letti 125 registri da 16 bit.

RISPOSTA DELLO SLAVE													
INDIRIZZO SLAVE	***FUNZIONE***	***NUMERO DI BYTE DEL DATO***	***VALORE DEI REGISTRI – DATO RICHIESTO***										***CRC***	
			REGISTRO 0		***REGISTRO 1***		***REGISTRO 2***		***REGISTRO 3***		***REGISTRO 4***		*HIGH*	*LOW*
0F	**03**	**0A**	**00**	**00**	**00**	**F0**	**00**	**00**	**7D**	**00**	**00**	**00**	**DA**	**5B**

Registri nello Slave (Server Modbus)

INDIRIZZO REGISTRO	VALORE
0	0
1	240
2	0
3	32000
4	0
5	
6	
7	



FUNZIONE 04 – READ INPUT REGISTER

La funzione è praticamente identica alla precedente, ma in questo caso i registri letti rappresentano gli ingressi fisici del dispositivo e non una zona di memoria.

FUNZIONE 05 – FORCE SINGLE COILS

Con questa funzione è possibile modificare lo stato di un bit che rappresenta una parte di memoria, o un'uscita fisica del dispositivo.

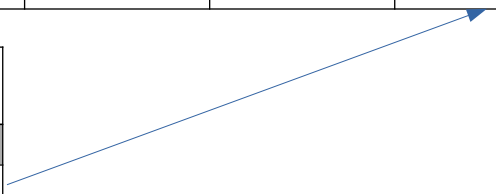
RICHIESTA ALLO SLAVE - 8 Bytes							
INDIRIZZO SLAVE	***FUNZIONE***	***INDIRIZZO INIZIALE DEL BIT***		***STATO DEL BIT 00=FALSE FF=TRUE***		***CRC***	
		HIGH	*LOW*	*STATO*	*SEMPRE A ZERO*	*HIGH*	*LOW*
0F	**05**	**00**	**01**	**FF**	**00**	**DC**	**D4**

In questo caso viene chiesto di forzare al valore TRUE (1) il bit con indirizzo 01 (il secondo) dello slave con indirizzo 15 (0F hex)..

La risposta sarà l'eco di quanto ricevuto e verrà inviata dopo aver modificato lo stato del bit.

RISPOSTA DELLO SLAVE - 8 Bytes							
INDIRIZZO SLAVE	***FUNZIONE***	***INDIRIZZO DEL BIT***		***STATO DEL BIT 00=FALSE FF=TRUE***		***CRC***	
		HIGH	*LOW*	*STATO*	*SEMPRE A ZERO*	*HIGH*	*LOW*
0F	**05**	**00**	**01**	**FF**	**00**	**DC**	**D4**

INDIRIZZO BIT	VALORE
0	
1	1 (TRUE)
2	
3	



FUNZIONE 06 – PRESET SINGLE REGISTER

Con questa funzione, si può impostare il valore di un registro a 16 bit. Come nella precedente funzione la risposta sarà l'eco dei caratteri inviati.

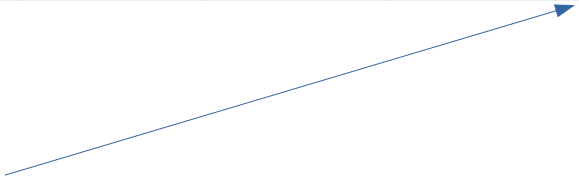
La funzione lavora sugli Holding register, cioè i registri della memoria interna dello Slave.

RICHIESTA ALLO SLAVE - 8 Bytes							
INDIRIZZO SLAVE	***FUNZIONE***	***INDIRIZZO DEL REGISTRO***		***VALORE DA SCRIVERE NEL REGISTRO***		***CRC***	
		HIGH	*LOW*	*HIGH*	*LOW*	*HIGH*	*LOW*
0F	**06**	**00**	**01**	**00**	**32**	**58**	**F1**

In questo caso viene chiesto di scrivere il valore 50 (32 hex) nel registro 1 dello Slave.

RISPOSTA DALLO SLAVE - 8 Bytes							
INDIRIZZO SLAVE	***FUNZIONE***	***INDIRIZZO DEL REGISTRO***		***VALORE DA SCRIVERE NEL REGISTRO***		***CRC***	
		HIGH	*LOW*	*HIGH*	*LOW*	*HIGH*	*LOW*
0F	**06**	**00**	**01**	**00**	**32**	**58**	**F1**

INDIRIZZO REGISTRO	VALORE
0	
1	50
2	
3	



FUNZIONE 07– READ STATUS

La funzione serve per leggere gli 8 bit di stato del dispositivo remoto.

RICHIESTA ALLO SLAVE - 4 Bytes			
INDIRIZZO SLAVE	***FUNZIONE***	***CRC - LOW***	***CRC - HIGH***
0F	**07**		

In questo caso la richiesta contiene solo l'indirizzo, il codice funzione ed il CRC.

RISPOSTA DALLO SLAVE - 5 Bytes				
INDIRIZZO SLAVE	***FUNZIONE***	***VALORE DEGLI 8 BIT DI STATUS***	***CRC - LOW***	***CRC - HIGH***
0F	**07**			

Le seguenti 3 funzioni, non sono esattamente di uso comune, in quanto si utilizzano per la diagnostica nel caso di comunicazione seriale.

FUNZIONE 08 – DIAGNOSTICS

Con questa funzione si esegue la diagnostica della linea seriale, pertanto non è utilizzabile nel MODBUS TCP.
Il formato prevede l'invio anche di un codice per una sottofunzione che definisce il tipo di test da effettuare.

RICHIESTA DALLO SLAVE - 6 Bytes					
INDIRIZZO SLAVE	***FUNZIONE***	***CODICE SOTTOFUNZIONE***		***DATI***	
0F	*08*	*HIGH*	*LOW*	*HIGH*	*LOW*

Sub-function code		Name
Hex	Dec	
00	00	Return Query Data
01	01	Restart Communications Option
02	02	Return Diagnostic Register
03	03	Change ASCII Input Delimiter
04	04	Force Listen Only Mode
	05.. 09	RESERVED
0A	10	Clear Counters and Diagnostic Register
0B	11	Return Bus Message Count
0C	12	Return Bus Communication Error Count
0D	13	Return Bus Exception Error Count
0E	14	Return Derver Message Count
0F	15	Return Derver No Response Count
10	16	Return Server NAK Count
11	17	Return Server Busy Count
12	18	Return Bus Character Overrun Count
13	19	RESERVED
14	20	Clear Overrun Counter and Flag
N.A.	21 ... 65535	RESERVED

Il contenuto del campo dati nel messagio di risposta dipende dal tipo di test che si vuole effettuare.
Nel caso il test non prevede la restituzione di un valore, la risposta sarà l'eco del messaggio inviato.
Di seguito i codici delle sottofunzioni, estratti dalle specifiche tecniche presenti sul sito modbus.org.

- **Return Query Data.** I dati inviati e quelli ricevuti coincidono, si crea un loop tra master e slave.
- **Restart Communication Options.** Lo slave viene riavviato, con o senza reset del log di eventi. Il campo dati trasmessi conterrà 00 00 se si vuole solo riavviare, FF 00 se si vuole resettare il log di eventi.
  La risposta sarà l'eco del messaggio.
- **Return Diagnostic Register.** Viene richiesto Il contenuto del registro a 16 bit di diagnostica dello slave. In questo caso nel campo dati della richiesta conterrà 00 00 ed in quello della risposta verrà inserito il valore di questo registro.
- **Change ASCII Input Delimiter.** Cambia il limitatore LF (Line Feed) del protocollo ASCII con un altro carattere. In questo caso nel campo dati, del messaggio inviato, ci satà il codice ASCII del nuovo delimitatore. La risposta sarà l'eco del messaggio inviato.

- **Force Listen Only Mode.** Mette lo slave in stato di ascolto, nel campo dati della richiesta viene messo il valore 00 00. Lo slave dopo aver ricevuto questo comando, non riceve più alcun comando, escluso il comando 08 – 01 (cioè nuovamente diagnostica seriale con sottofunzione 01). Non è prevista alcuna risposta da parte dello Slave.
- **Clear Counters and Diagnostics Register.** Azzera i contatori ed il registro di diagnostica a 16 bit, nel campo dati della richiesta viene messo 00 00. La risposta conterrà l'eco della richiesta.
- **Return Bus Message Count.** Restituisce i messaggi che lo slave ha ricevuto dall'ultima ripartenza o reset, dall'ultimo reset dei contatori o dall'ultimo avviamento. Il campo dati della richiesta contiene il valore 00 00, nel campo dati della risposta ci sarà il numero totale dei messaggi ricevuti.
- **Return Bus Communications Error Count.** Restituisce i numero di errori CRC rilevati dall'ultima ripartenza o reset, dall'ultimo reset dei contatori o dall'ultimo avviamento. Il campo dati della richiesta contiene il valore 00 00, nel campo dati della risposta ci sarà il numero totale degl errori CRC.

  Con le stesse modalità dell'ultima sottofunzione, ma con il conteggio di messaggi/errori o situazioni differenti, vengono gestite le sottofunzioni 13-14-15-16-17-18.

- **Clear Overrun Counter and Flag.** Azzera il contatore di sovraccarico caratteri ed i flag di errore.
  Il campo dati della richiesta contiene il valore 00 00, la risposta sarà l'eco della richiesta.

FUNZIONE 11 (0B hex) – GET COM EVENT COUNTER

Con questa funzione è possibile leggere lo stato ed il contatore degli eventi andati a buon fine durante la comunicazione seriale. Il contatore degli eventi si incrementa ogni messaggio ricevuto correttamente, non viene incrementato se ci sono risposte contenenti delle eccezioni. Il contatore viene resettato con la funzione 08-00-01 o 08-00-0A (come visto precedentemtente).

FUNZIONE 12 (0C hex) – GET COM EVENT LOG

Questa funzione legge le seguenti informazioni:

- stato
- contatore eventi
- contatore di messaggi.

In pratica restituisce la situazione complessiva dello slave. Sono previsti 2 bytes per lo stato, 2 per il contatore eventi e 2 per il contatore messaggi e fino a 64 per gl eventi.

Andando sempre in ordine crescente del codice numerico della funzione, torniamo ora quelle maggiormente utilizzate

FUNZIONE 15 (0F hex) – FORCE MULTIPLE COILS

La funzione consente la scrittura contigua, di più bit.

RICHIESTA ALLO SLAVE - il numero dei Bytes dipende da quanti bit si devono scrivere										
INDIRIZZO SLAVE	***FUNZIONE***	***INDIRIZZO DEL BIT***		***QUANTITA' DI BIT DA SCRIVERE***		***NUMERO DI BYTES CONTENENTI IL VALORE DEI BIT***	***VALORE DEI BIT***		***CRC***	
		HIGH	*LOW*	*HIGH*	*LOW*		*HIGH*	*LOW*	*HIGH*	*LOW*
0F	**0F**	**00**	**02**	**00**	**10**	**02**	**F6**	**30**	**E8**	**16**

In questo caso viene chiesto allo slave con indirizzo 15 (0F hex) di scrivere 10 bit partendo dal terzo bit (indirizzo 00 02). Il valore dei dieci bit è dato dai due bytes che valgono F6 e 30, pertanto i bit dovranno assumere sequenzialmente partendo dal terzo il seguente valore: 1111 0110 0011 0000.

La risposta conterrà quanto segue.

RISPOSTA DALLO SLAVE - 8 Bytes							
INDIRIZZO SLAVE	***FUNZIONE***	***INDIRIZZO DEL BIT***		***QUANTITA' DI BIT SCRITTI***		***CRC***	
		HIGH	*LOW*	*HIGH*	*LOW*	*HIGH*	*LOW*
0F	**0F**	**00**	**02**	**00**	**10**	**F4**	**E9**

Lo stato dei bit nella memoria dello slave sarà il seguente:

BYTE	INDIRIZZO BIT	VALORE		
	0			
	1			
1° byte F6 hex	2	0	LSB	
	3	1		
	4	1		
	5	0		**1111 0110**
	6	1		**F 6**
	7	1		
	8	1		
	9	1	MSB	
2° byte 30 hex	10	0	LSB	
	11	0		
	12	0		
	13	0		**0011 0000**
	14	1		**3 0**
	15	1		
	16	0		
	17	0	MSB	
	18			

FUNZIONE 16 (10 hex) – PRESET MULTIPLE REGISTER

La funzione consente la scrittura di più registri a 16 bit (holding registers) contigui tra di loro.

RICHIESTA ALLO SLAVE - il numero dei Bytes dipende da quanti registri si devono scrivere																
INDIRIZZO SLAVE	*FUNZIONE*	*INDIRIZZO DEL REGISTRO*		*NUMERO DEI REGISTRI DA SCRIVERE*		*NUMERO DI BYTES CONTENENTI IL VALORE DEI REGISTRI*	*1° BYTE*		*2° BYTE*		*3° BYTE*		*4° BYTE*		*CRC*	
		HIGH	*LOW*	*HIGH*	*LOW*		*HIGH*	*LOW*	*HIGH*	*LOW*	*HIGH*	*LOW*	*HIGH*	*LOW*	*HIGH*	*LOW*
0F	**10**	**00**	**01**	**00**	**04**	**08**	**00**	**0C**	**00**	**96**	**00**	**02**	**79**	**18**	**C3**	**FA**

In questo caso, viene chiesto di scrivere 4 registri a 16 bit (8 bytes) partendo dall'indirizzo 1 (secondo registro), sullo slave con indirizzo 15 (0F hex). I valori scritti partendo dal primo byte sono rispettivamente:
12 (00 0C) – **150** (00 96) – **2** (00 02) – **31000** (79 18)

La risposta conterrà oltre all'indirizzo dello slave, alla funzione ed all'indirizzo del primo registro, la quantità di registri scritti ed il CRC.

RISPOSTA DALLO SLAVE - 8 Bytes							
INDIRIZZO SLAVE	*FUNZIONE*	*INDIRIZZO DEL PRIMO REGISTRO*		*QUANTITA' DI REGISTRI SCRITTI*		*CRC*	
		HIGH	*LOW*	*HIGH*	*LOW*	*HIGH*	*LOW*
0F	**10**	**00**	**01**	**00**	**04**	**91**	**24**

Il contenuto dei registri interni dello slave (holding register) sarà il seguente:

INDIRIZZO REGISTRO	VALORE	
0		
1	12	1°byte
2	150	2°byte
3	2	3°byte
4	31000	4°byte

FUNZIONE 23 (17 hex) – READ/WRITE MULTIPLE REGISTER

La funzione consente la scrittura e la lettura di più registri a 16 bit contigui, con un unico ciclo modbus.

Viene prima effettuata la scrittura e successivamente la lettura.

Al massimo possono essere letti 125 registri e scritti 150 registri.

La struttura della richiesta è la seguente, anche se la dimensione dipenderà dal numero di registri che si vogliono leggere e scrivere.

INDIRIZZO DELLO SLAVE	***1 Byte***	
CODICE FUNZIONE	***1 Byte***	***17***
INDIRIZZO PRIMO REGISTRO DA LEGGERE	***2 Bytes***	***da 0000 a FFFF***
NUMERO DI REGISTRI DA LEGGERE	***2 Bytes***	***da 0001 a 007D***
INDIRIZZO PRIMO REGISTRO DA SCRIVERE	***2 Bytes***	***da 0000 a FFFF***
NUMERO DI REGISTRI DA SCRIVERE	***2 Bytes***	***da 0001 a 0079***
NUMERO DI BYTE CHE CONTERRANNO I VALORI DEI REGISTRI	***1 Byte***	
VALORE DEI REGISTRI	***N Bytes***	
CRC	***2 Bytes***	

La struttura della risposta invece sarà la seguente:

INDIRIZZO DELLO SLAVE	***1 Byte***	
CODICE FUNZIONE	***1 Byte***	***17***
INDIRIZZO PRIMO REGISTRO DA LEGGERE	***2 Bytes***	
VALORE DEI REGISTRI	***N Bytes***	
CRC	***2 Bytes***	

ERRORI

Durante la trasmissione si possono verificare due tipi di errori, l'errore di trasmissione e l'errore operativo.
Il primo si verifica qualora il messaggio inviato venga compromesso durante l'invio e venga percò ricevuto male.
In questo caso l'errore viene rilevato da un eventuale controllo di parità dei bit, se attivo nella trasmissione seriale, o dl CRC. Lo slave che rileva un errore di trasmissione, scarta il messaggio non considerandolo.

Se invece il messaggio viene inviato correttamente senza errori di trasmissione, si potrebbe verificare un errore nel contenuto del messaggio stesso, come ad esempio una funzione non eseguibile, o un indirizzo sbagliato. In questo caso il dispositivo slave risponde con un messaggio definito "messaggio d'eccezione".
Il messaggio contiene l'indirizzo dello slave, il codice della funzione richiesta con il bit più significativo messo ad 1, un codice d'errore ed il CRC.

La struttura della risposta è la seguente:

INDIRIZZO SLAVE	***FUNZIONE***	***CODICE D'ERRORE***	***CRC HIGH***	***CRC LOW***

CODICE D'ERRORE	***TIPO ERRORE***	***SIGNIFICATO***
01	ILLEGAL FUNCTION	Il codice della funzione non corrisponde ad una funzione dello slave
02	ILLEGAL DATA ADDRESS	l'indirizzo non è consentito
03	ILLEGAL DATA VALUE	Il valore non può essere assegnato nell'indirizzo scelto
04	SLAVE ERROR	Errore irreversibile, lo slave non può portare a termine la richiesta.
05	ACKNOWLDGE	Lo slave sta elabrando la richiesta, ma la stessa richiede del tempo per essere eseguita.
06	DEVICE BUSY	Lo slave è occupato.
07	NAK	La funzione non può essere eseguita o si voleva scrivere su un indirizzo a sola lettura
08	MEMORY ERROR	Un blocco dati (record) non ha superato il controllo di integrità dei dati.

CRC (Cyclic Redundancy Check)

Esaminiamo ora il funzionamento del CRC, cioè i due byte che vengono inseriti in coda al messaggio per verificare l'integrità dello stesso.

Il CRC non è presente nella comunicazione MODBUS TCP, in quanto in quel caso il controllo d'integrità viene eseguito all'interno del protocollo TCP/IP.

Come abbiamo visto nei precedenti esempi, in coda ad ogni messaggio, ci sono due byte che rappresentano il CRC16 (16 rappresenta il numero di bit utilizzati). Il calcolo di questi due byte, avviene considerando il resto del messaggio come un unico numero binario continuo di cui il bit più significativo (MSB) viene trasmesso per primo.

Consideriamo l'esempio rappresentato nella funzione 01 – Read Coils Status visto sopra.

RICHIESTA ALLO SLAVE - 8 Bytes							
INDIRIZZO SLAVE da 1 a 247	***FUNZIONE***	***INDIRIZZO DEL DATO da 0000 a FFFF da 0000 a 65535***		***NUMERO DI BIT da 0001 a 07D0 da 0001 a 2000***		***CRC***	
		HIGH	*LOW*	*HIGH*	*LOW*	*HIGH*	*LOW*
0F	**01**	**00**	**03**	**00**	**14**	**CD**	**2B**

Il messaggio su cui calcolare il CRC è composto dai numeri esadecimali evidenziati in giallo e cioè:

che in binario rappresentano il numero :

0F	**01**	**00**	**03**	**00**	**14**
0000 1111	**0000 0001**	**0000 0000**	**0000 0011**	**0000 0000**	**0001 0100**

Possiamo trovare dei siti online per il calcolo del CRC, come ad esempio https://crccalc.com/

Immettendo nel sito i valori in esadecimale dei campi evidenziati in giallo, avremo i due byte del CRC che saranno **CD** e **2B**.

Vediamo ora quali sono le fasei per effettuare il calcolo del CRC.

Su questo numero viene applicato un algoritmo che effettua le seguenti operazioni:

1. caricare un registro a 16 bit con il valore 0xFFFF (tutti 1)
2. eseguire l'OR esclusivo del registro con il primo byte, il risultato va nel registro stesso
3. spostare a destra di un bit il registro per 8 volte
4. se dopo ogni spostamento a destra, il bit fuoriuscito è 1 fare l'OR esclusivo del registro, con il byte 0xA001
5. eseguire l'OR esclusivo del byte successivo con il registro
6. ripetere i punti da 3 a 5 per tutti i byte del messaggio.

Un esempio di una funzione per il calcolo del CRC realizzata in linguaggio C con l'ambiente di sviluppo mikroC della mikroelektronika, è la seguente.

```
//CALCOLA IL CRC 16bit
//il vettore messaggio[] contiene i bytes su cui calcolare i CRC
void calcola_CRC(char numero_byte)
{
  crc=0xFFFF;   //inizializzo un registro a 16 bit con tutti 1
  cont=8;       //inizializzo i contatori
  cont_byte=0;

  while(cont_byte<numero_byte)      //eseguo il ciclo per ogni byte partendo dal primo
   {
    crc=crc^messaggio[cont_byte];  //OR esclusivo tra regitro crc e byte del messaggio
    while(cont)
     {
      crc>>=1;                     //sposto a destra di un bit
      if(status.c) crc=crc^0xA001; //se il bit uscito a destra è 1, OR esclusivo con 0xA001
      cont--;                      //decremento il contatore per fare
     }                             //8 volte la stessa operazione
    cont_byte++;                   //incremento il contatore dei byte per passare al byte successivo
    cont=8;
   }//while(cont_byte)
  messaggio[numero_byte+1]=Hi(crc); //salvo in coda al messaggio i due byte del CRC calcolati
  messaggio[numero_byte]=Lo(crc);
}//calcola_CRC
```

CAMPI DI APPLICAZIONE

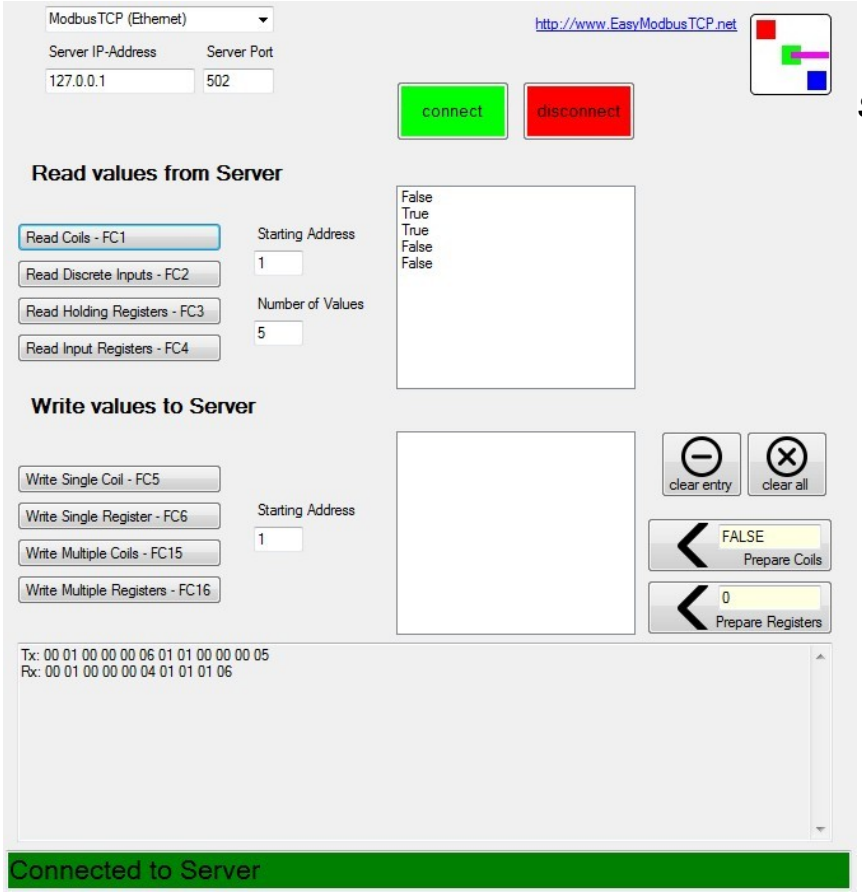
Come detto in premessa questo protocollo anche se obsoleto, è ancora molto utilizzato, soprattutto nel settore industriale. Possiamo trovarlo nei PLC, per realizzare una comunicazione con un pannello operatore, o in vari dispositivi nel settore dell'impiantistica o dell'automazione industriale, come ad esempio moduli di ingresso/uscita analogici e digitali.

In ogni caso essendo un protocollo molto flessibile, occorre studiare i datasheet dei vari dispositivi, o le funzioni software che vengono messe a disposizione nei vari PLC.

CONCLUSIONI E SUGGERIMENTI

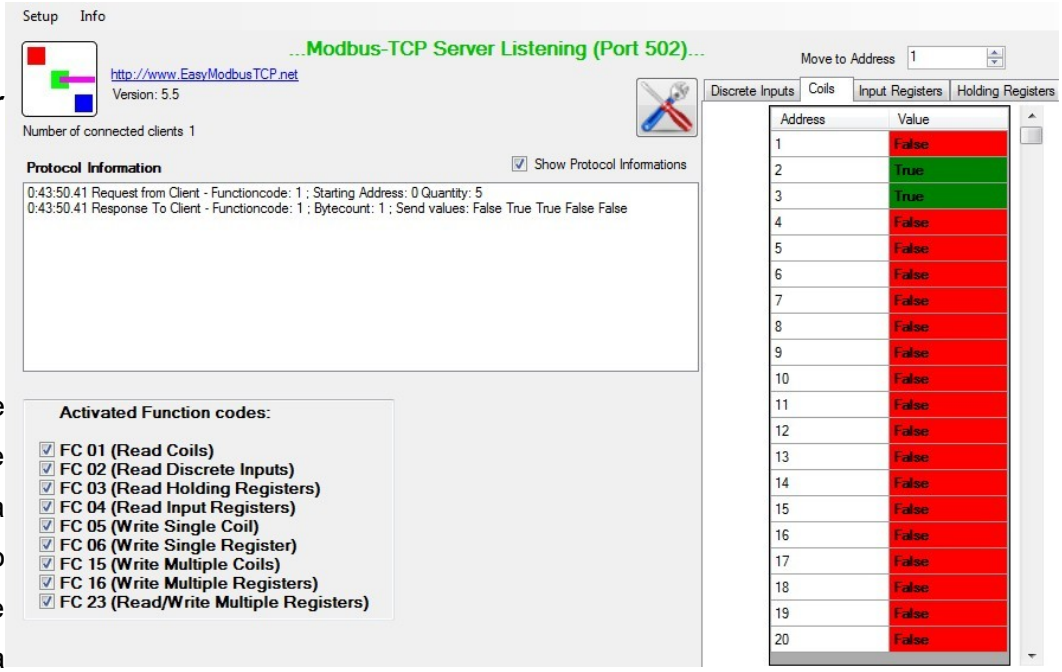
Volendo effettuare delle prove esistono in rete dei simulatori del protocollo modbus, quello che ho utilizzato per creare gli esempi precedenti, si può scaricare liberamente al seguente link.: http://easymodbustcp.net/en/

Il mio consiglio pertanto è quello di scaricare il software per il client ed il server e cominciare a provare ad utilizzare le funzioni precedentemente descritte.



Simulatore modbus Client (Master)

Simulatore modbus Server (Slave)



Il passo successivo potrebbe essere quello di creare uno Slave modbus, utiizzando una scheda a microcontrollore. In questo caso esistono delle librerie già pronte e funzionanti, anche se vista la semplicità della struttura del messaggio, ognuno potrebbe creare la propria funzione per inviare e ricevere messaggi utilizzando questo protocollo.