Protocollo Modbus su RS485

OVERDIGIT srl

innovative automation

Protocollo Modbus su RS485 - Introduzione

Il Modbus è ancora oggi uno dei protocolli di comunicazione più diffusi nel settore dell'automazione industriale nonostante le sue origini risalgano al 1979. La sua solida e duratura presenza nella vasta panoramica dei protocolli industriali è dovuta al fatto che il Modbus è un protocollo semplice e diretto. Inoltre non è soggetto royalty ed è flessibile per cui può essere liberamente implementato ed adattato alle più svariate applicazioni.

Le risorse hardware e software richieste dal protocollo Modbus sono piuttosto esigue e quindi esso è utilizzabile anche in sistemi basati su piccoli ed economici microprocessori. La semplice ed essenziale struttura dei dati trasmessi permette una comunicazione efficace con un buon rapporto tra la parte utile dell'informazione ed il numero totale di bytes trasmessi. Il protocollo si presta anche ad implementazioni parziali dei comandi, limitate alle specifiche esigenze, nonché all'espansione con l'aggiunta di comandi custom da parte del produttore del dispositivo.

Molto spesso i produttori di dispositivi per l'automazione "cadono" nella tentazione di sviluppare un protocollo di comunicazione completamente proprietario pensando di fare la cosa più semplice ed efficace. Tuttavia questi protocolli sono frequentemente delle "brutte copie" di un protocollo già semplice e diretto come il Modbus ma non sono però compatibili ne con questo ne con altri protocolli standard. Ciò può costituire un vantaggio se si intende vincolare i clienti al proprio protocollo ma esclude la interoperabilità con tanti altri dispositivi commerciali e la possibilità che il prodotto sia esso stesso più commerciabile.

Il protocollo Modbus è basato su bus di campo e questo significa che è adatto allo scambio di informazioni tra apparecchiature fisicamente distinte e posizionabili anche a notevoli distanze tra loro. La realizzazione di un macchina automatica o di un impianto mediante un insieme di dispositivi tra loro interconnessi ha molteplici vantaggi come:

- Possibilità di realizzare l'impianto di automazione utilizzando dispositivi commerciali, realizzati da diversi produttori e facilmente sostituibili tra loro.
- Distribuzione dei vari dispositivi in posizioni strategiche per l'impianto, anche remote, con notevole riduzione e risparmio nei cablaggi degli I/O.
- Frazionamento dell'impianto con conseguente semplificazione dello sviluppo progettuale e delle operazioni di manutenzione e riparazione.

I dispositivi sono connessi tra loro mediante cavi dotati di pochi poli conduttori per formare una rete distribuita anche su elevate distanze. Il protocollo Modbus prevede diverse tipologie di connessione fisica tra i dispositivi. In particolare i mezzi di connessione più diffusi sono:

- Rete RS485 - Protocollo Modbus seriale
- Rete Ethernet - Protocollo Modbus TCP/IP

Il tipo di connessione adottata definisce quello che normalmente si chiama "Physical layer" del protocollo ossia l'hardware sul quale avviene lo scambio delle informazioni. In questa serie di articoli verrà trattato solo il protocollo Modbus su rete seriale RS485. Una seriale di questo tipo necessita di cavi con solo due poli per far transitare dati da un qualsiasi dispositivo all'altro.

Il successivo livello del protocollo, il “Data Link layer”, è invece di tipo software e comprende tutte le specifiche relative allo scambio dei frames di dati (sequenze di bytes) tra un dispositivo e l’altro. Questa parte prescinde dal contenuto informativo dei bytes e si occupa esclusivamente del loro invio sul bus di campo, del controllo delle temporizzazioni e degli errori mediante checksum.

Il protocollo Modbus è di tipo Master/Slave e quindi nella rete è presente sempre e solo un dispositivo Master che gestisce la comunicazione nei confronti di uno o più dispositivi Slave. Ogni scambio di informazioni è originato dal Master il quale invia un frame di bytes sul bus di campo contente una particolare richiesta, normalmente un comando di lettura o di scrittura delle informazioni contenute in uno degli Slave. Tutti gli Slave sono normalmente in ricezione ed ascoltano le richieste del Master. Solo lo specifico Slave interrogato cattura le informazioni inviate dal Master, provvede all’esecuzione del comando e risponde al Master inviando a sua volta le proprie informazioni sulla rete.

Le modalità, secondo le quali le possibili richieste del Master e le relative risposte degli Slave sono codificate all’interno dei frames di comunicazione, sono definite dal successivo livello del protocollo, detto “Application protocol”, anch’esso di tipo software. E’ con questo livello finale del protocollo che interagisce l’applicazione specifica del dispositivo, nel caso del Master solitamente il programma di automazione del PLC, nel caso degli Slave il firmware di gestione degli specifici I/O del dispositivo. Questo livello costituisce l’interfaccia del protocollo con tutta la parte restante del software del dispositivo.

La descrizione del protocollo frazionata in successivi livelli di implementazione è definita dal modello ISO/OSI frequentemente utilizzato per rappresentare i protocolli di comunicazione:

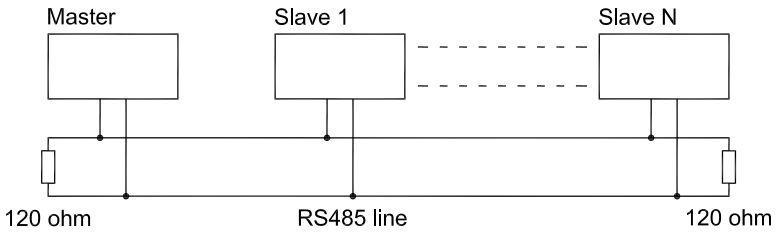
Layer	ISO/OSI Model	
7	Application	Modbus Application Protocol
6	Presentation	Not used
5	Session	Not used
4	Transport	Not used
3	Network	Not used
2	Data Link	Modbus Serial Line Protocol
1	Physical	EIA/TIA-485 standard

Si noti che, nel caso del protocollo Modbus seriale, i livelli da 3 a 6 non sono utilizzati mentre questi invece fanno parte di una comunicazione strutturata su rete Ethernet.

Negli articoli successivi della serie “Modbus su RS485” saranno analizzati con maggior dettaglio i tre livelli utilizzati dal protocollo seriale facendo riferimento alla documentazione ufficiale del Modbus. Inoltre, in un articolo finale saranno fornite informazioni relative alla specifica implementazione del protocollo nei moduli Slave Overdigit. Questo articolo costituisce un esempio concreto di implementazione della comunicazione Modbus su RS485 e di come il protocollo possa essere ampliato per incrementarne le caratteristiche di base.

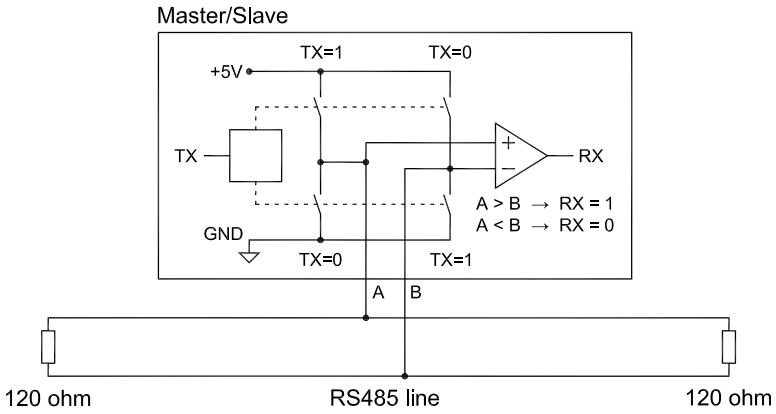
Protocollo Modbus su RS485 - Physical layer

Il protocollo Modbus su rete seriale RS485 è sicuramente una delle implementazioni più diffuse grazie alla sua semplicità, economia ma anche affidabilità in ambiente industriale.
Essa è realizzata mediante un cavo bifilare che unisce in parallelo tutti i dispositivi presenti sulla rete. Essendo un protocollo di tipo Master/Slave saranno presenti sempre un solo dispositivo Master ed uno o più dispositivi Slave:



La connessione bifilare è la più economica possibile ma al tempo stesso offre delle ottime prestazioni in termini di velocità di comunicazione ed immunità ad eventuali segnali di disturbo elettromagnetico. Questo grazie alla particolare tecnica di utilizzo dei segnali elettrici applicati alla coppia di fili che verrà descritta nel seguito.

La comunicazione dei bits 0/1 tra i dispositivi avviene applicando sui due fili della coppia una tensione continua di piccola entità la cui polarità cambia in funzione del livello logico 0/1 da trasmettere:



Normalmente, in assenza di trasmissione da parte del dispositivo, tutti quattro gli interruttori del blocco di trasmissione sono OFF e quindi entrambi i conduttori della linea RS485 non sono connessi ad alcun potenziale elettrico. Tuttavia è preferibile non lasciare la linea fluttuante e per questo si applica sempre un potenziale di default, positivo su A e negativo su B. Questa polarizzazione è ottenuta collegando due resistenze, una tra l'alimentazione +5V ed il segnale A ed un'altra tra il segnale B ed il riferimento GND.

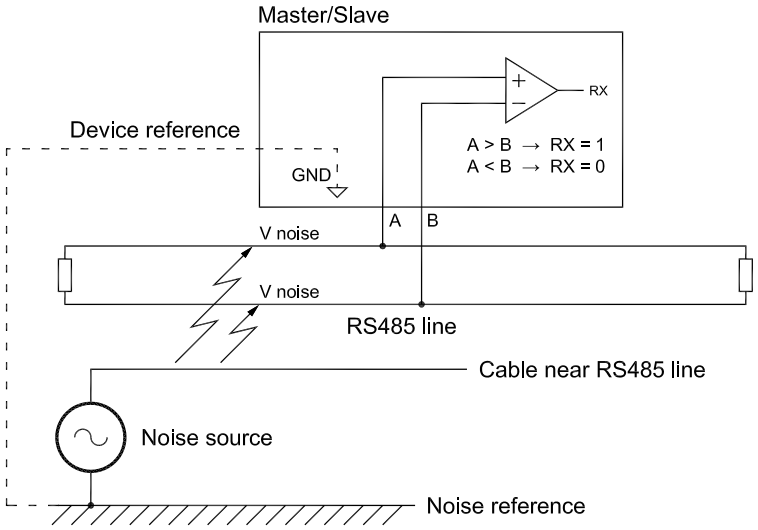
Quando un dispositivo deve trasmettere dei bits, esso prende momentaneamente il controllo della linea accendendo i propri interruttori a due a due in diagonale. Per trasmettere uno 0 logico la linea viene forzata con A=GND e con B=+5V, mentre per trasmettere un 1 logico la linea viene forzata con A=+5V e con B=GND. In questo modo la tensione misurata tra il conduttore A ed il conduttore B sarà +5V oppure -5V rispettivamente per trasmettere il bit 1 oppure il bit 0.

Ogni dispositivo comprende anche un blocco di ricezione che può determinare la polarità del segnale A relativamente a quella del segnale B. Per questo l'interfaccia di ricezione è in grado di misurare la differenza di tensione tra il polo A ed il polo B.
Se tale differenza è positiva (A > B) la ricezione corrisponde allo stato logico 1, se la differenza è negativa (A < B) lo stato logico che si sta ricevendo è 0. Il blocco di ricezione permette quindi di eseguire il processo inverso, decodificando l'informazione 0/1 precedentemente codificata con il blocco trasmettitore.

NOTA: il blocco trasmettitore è stato rappresentato idealmente con degli interruttori. In realtà questi interruttori sono realizzati con transistors in grado di commutare i segnali ad elevata velocità. Una vasta gamma di apposti circuiti integrati sono stati sviluppati come driver di interfaccia RS485 inserendo al loro interno sia la parte di trasmissione che di ricezione.

Questa tecnica di pilotaggio della linea seriale da origine al nome di "segnale differenziale" riferendosi a quello applicato ai fili A/B della linea. Si noti che in una connessione differenziale non è strettamente necessario collegare il riferimento comune GND di un dispositivo con quello degli altri come normalmente avviene quando si connettono dei segnali tra due apparecchiature distinte. Infatti nel caso del segnale differenziale quello che conta non è la tensione di A o B rispetto a GND ma la differenza relativa tra A e B. I segnali differenziali sono molto utilizzati, come ad esempio nella connessione USB, proprio per le loro elevate prestazioni abbinate alla massima semplicità del supporto fisico.

Un grande vantaggio derivante dai segnali differenziali lo si ha realizzando il cavo mediante una coppia di conduttori tra loro strettamente intrecciati, ottenendo un elevato grado di uguaglianza dei due fili in termini di posizione fisica nello spazio. Una eventuale interferenza, causata dalla vicinanza della linea ai cavi di altri sistemi, indurrà una tensione di disturbo su entrambi i fili della coppia, sovrapponendo ad essi delle tensioni indesiderate con la stessa polarità (positiva o negativa) rispetto al riferimento:



I ricevitori di ogni dispositivo, che effettuano una valutazione della differenza di tensione tra i fili A e B, saranno quindi in grado di eliminare questa componente di disturbo proprio perché di uguale segno ed entità su entrambi i fili. Per questo è importante che l'eventuale disturbo interferisca allo stesso modo con entrambi i conduttori della linea.

Negli schemi precedenti sono state evidenziate anche due resistenze da 120 ohm poste agli estremi della linea RS485. Lo scopo di queste resistenze è di definire l'impedenza di carico nelle parti terminali della linea per evitare che, nella propagazione dei segnali lungo i cavi, si generino, in corrispondenza alle estremità aperte, delle riflessioni indietro con conseguente alterazione della forma d'onda dei segnali stessi.
Questo fenomeno è tanto più evidente quanto più alte sono le frequenze dei segnali ma già alle più basse frequenze di utilizzo del bus di campo si possono notare questi effetti. Quindi l'inserimento delle resistenze di terminazione della linea è richiesto tassativamente in tutte le applicazioni.

Oltre alla corretta terminazione della linea differenziale è molto importante che la connessione di questa ai vari dispositivi avvenga seguendo un unico percorso lineare ossia senza diramazioni. Occorre prestare attenzione a questa regola anche nel punto di connessione della linea ai due appositi morsetti di ogni dispositivo. La linea di cavo va interrotta solo in corrispondenza della morsettiera di ogni dispositivo, serrando il cavo in arrivo con quello in partenza direttamente nei stessi morsetti. Solo il primo ed ultimo dispositivo, indipendentemente se Master o Slave, devono anche avere la connessione alla resistenza di terminazione. Questa resistenza è solitamente incorporata nei dispositivi ed abilitabile da un'apposito interruttore.

La rete seriale RS485 permette di realizzare impianti distribuiti posizionando i vari dispositivi anche a notevoli distanze tra loro. La lunghezza massima della connessione dipende da molti fattori come la velocità di comunicazione (generalmente da 9600 a 115200b/s per il protocollo Modbus), dalla qualità dei cavi e dalle caratteristiche del driver hardware dei dispositivi. Utilizzando cavi specifici per le linee differenziali, come quelli prodotti da Belden, la distanza massima raggiungibile è di 1200m con velocità di comunicazione nel range da 10Kb/s a 100Kb/s mentre con velocità crescenti fino a 1Mb/s la distanza scende proporzionalmente fino a 120m. Questi sono dati caratteristici specificati per i soli cavi ma esistono tanti altri fattori, dipendenti dalla particolare applicazione, come l'ambiente circostante e l'accuratezza dell'installazione, che possono portare ad una significativa riduzione della distanza massima raggiungibile per una determinata velocità di comunicazione.

Il numero massimo di dispositivi collegabili alla rete RS485 dipende principalmente dalle caratteristiche dei circuiti integrati driver adottati nelle interfacce dei singoli nodi.
La specifica EIA/TIA-485 richiede che il circuito trasmettitore sia capace di pilotare fino a 32 unità di carico dove una unità di carico (UL) equivale ad una impedenza di circa 12Kohm. Le tecnologie dei circuiti integrati driver, in continua evoluzione in termini di prestazioni, hanno permesso di introdurre nel mercato dispositivi capaci di corrispondere a frazioni dell'unità di carico per cui il numero massimo di dispositivi può essere maggiore di 32. Per esempio componenti con 1/2UL permettono di connettere tra loro fino a 64 dispositivi, mentre con 1/8UL si può arrivare a 256 nodi. Si consideri comunque che questi dati si basano esclusivamente sulla valutazione dell'unità di carico, senza considerare le resistenze di polarizzazione del bus e tanti altri elementi della reale applicazione, in particolare la qualità e lunghezza delle connessioni ed anche la velocità di comunicazione, che possono ridurre in modo sostanziale il numero massimo dei dispositivi.

Protocollo Modbus su RS485 - Data Link layer

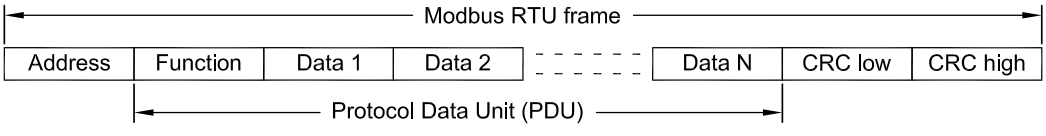
Il “Data Link layer” rappresenta il primo dei livelli di tipo software del protocollo Modbus e definisce come i bytes vengano scambiati tra un dispositivo e l’altro prescindendo tuttavia dal loro specifico significato. Questo livello prevede due diversi metodi di trasmissione delle informazioni.

La trasmissione di tipo RTU comporta uno scambio di frames binari dove i bytes possono assumere tutti i possibili valori 0÷255. Non potendo identificare l’inizio ed il termine di ogni frame di comunicazione in base a valori specifici dei bytes (caratteri di sincronizzazione) è necessario ricorrere alla gestione di timers che controllino l’esatta temporizzazione della trasmissione. In alcuni sistemi la gestione di queste temporizzazioni, i cui tempi possono essere anche molto brevi alle più elevate velocità di comunicazione, può essere onerosa e costituire un problema.

La trasmissione ASCII invece è ottenuta codificando il valore di ogni byte dell’informazione con la sequenza di due caratteri di testo esadecimali (cifre 0÷9, A÷F). Il protocollo ASCII è meno efficiente del protocollo RTU in quanto ogni byte di informazione è codificato da due caratteri. Tuttavia questa codifica consente di riservare dei caratteri speciali allo scopo di essere utilizzati come indicatori (caratteri di sincronizzazione) di inizio e fine frame. Questo semplifica notevolmente la gestione della trasmissione in particolare in sistemi più lenti o non dotati di particolari dispositivi hardware per implementare il controllo di temporizzazione necessario invece al protocollo RTU.
In questa serie di articoli sarà trattato solo il protocollo seriale Modbus RTU.

La comunicazione su rete seriale RS485 è di tipo half-duplex e ciò significa che solo un dispositivo alla volta può trasmettere informazioni sul bus. Questo richiede che la trasmissione effettuata dai vari nodi della rete sia coordinata in modo tale che non ci siano sovrapposizioni del pilotaggio della linea differenziale da parte di più di un dispositivo alla volta.
Il coordinamento della comunicazione è ottenuto assegnando ad uno dei nodi il ruolo di Master mentre a tutti gli altri quello di Slave. In genere la funzione di Master viene assunta dal dispositivo di controllo della logica dell’impianto, ad esempio il PLC, mentre tutti i dispositivi periferici e di espansione degli I/O operano come Slave.

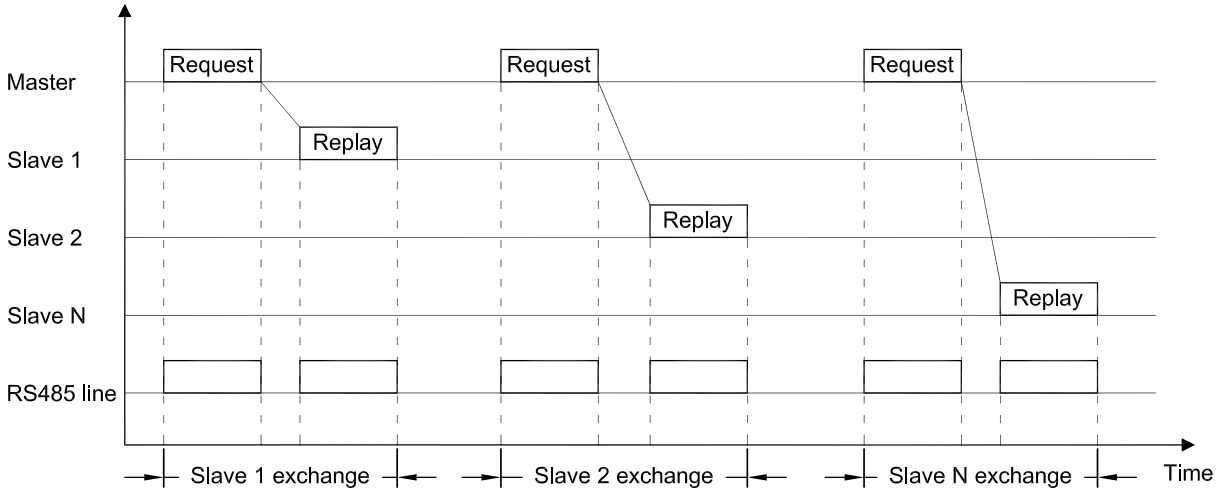
Il Master controlla la comunicazione nel senso che è l’unico autorizzato ad iniziare uno scambio di informazioni. Periodicamente o su necessità, il Master inizia la comunicazione con uno degli Slave, inviando un pacchetto di bytes sul bus contenente l’indirizzo dello Slave interessato, la funzione da eseguire, gli eventuali dati associati alla funzione ed il checksum di controllo del pacchetto:



Il campo Address è costituito da un solo byte contenente l’indirizzo dello Slave interessato. I valori ammessi dal protocollo sono nel range 0÷247.

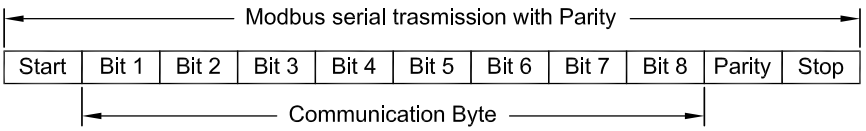
Anche per il campo Function è utilizzato un solo byte e questo contiene il codice comando della specifica richiesta allo Slave. Al codice funzione possono essere associati anche alcuni bytes di Dato specifici del comando fino ad un massimo di 252 bytes. I valori dei codici funzione ed il significato dei dati associati al comando saranno definiti nel livello successivo del protocollo (Application layer). L'insieme del campo funzione e del campo dati viene denominato Protocol Data Unit (PDU) e rappresenta la parte utile della comunicazione.
Al termine del frame sono aggiunti due bytes contenenti la parte bassa e la parte alta della word di checksum (CRC) calcolata su tutti i bytes precedenti. Questa word costituisce una firma relativa al contenuto del frame al fine di identificare eventuali errori nella trasmissione dei bytes.

Tutti gli Slave, normalmente in "ascolto" sul bus, riceveranno il frame trasmesso dal Master ma solo quello indirizzato continuerà la comunicazione mentre gli altri ignoreranno la richiesta.
Lo Slave interessato risponderà al Master inviando a sua volta un frame con la stessa struttura di quello appena descritto.
Il Master, una volta terminato lo scambio dei frames con uno Slave, potrà iniziare una nuova sequenza di comunicazione con lo stesso Slave oppure con uno degli altri procedendo allo stesso modo:



Lo Slave interrogato deve rispondere entro il più breve tempo possibile anche per non rallentare le operazioni di comunicazione con tutti gli altri Slave. Inoltre è importate gestire nel Master un tempo massimo prefissato (Timeout) entro il quale lo Slave deve rispondere per evitare di bloccare la comunicazione nel caso di Slave assente o guasto.
Il protocollo Modbus non specifica come eseguire l'aggiornamento delle informazioni nei confronti degli Slave per cui è possibile che il Master esegua le richieste in modo arbitrario, a seconda delle necessità, oppure in modo periodico (polling) per mantenere costantemente aggiornato la stato delle risorse degli Slave.

Ciascun byte del frame è trasmesso sul bus di campo RS485 mediante l'invio seriale di un totale di 11 bits, comprendendo il bit di start, il bit di parità ed il bit di stop:

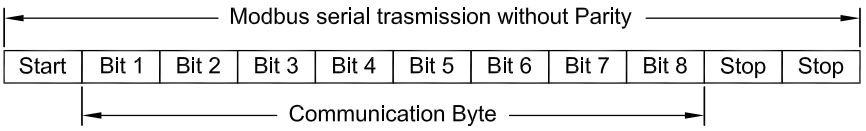


Il bit di start vale sempre 0 e costituisce un riferimento di sincronismo per inizializzare la ricezione di tutti i bits del carattere. Dopo il bit di start sono inviati gli 8 bits del byte da trasmettere iniziando dal bit meno significativo. Successivamente è trasmesso un bit di parità per il controllo degli errori su un bit del dato ed infine un bit di stop (valore fisso ad 1).
Il protocollo Modbus permette di utilizzare una delle seguenti modalità per il bit di parità:

Parity	Rule for define the parity bit
No	Fixed to 1
Even	Total of bits (data + parity) equal to 1 is even
Odd	Total of bits (data + parity) equal to 1 is odd

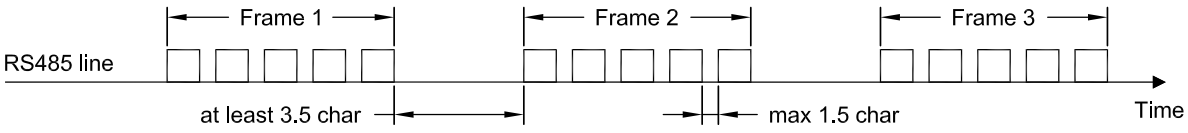
In realtà la specifica Modbus, anche se lascia piena libertà all'utilizzatore del dispositivo nella scelta della parità, obbliga al produttore di implementare nel dispositivo almeno la parità Even mentre raccomanda la disponibilità anche della configurazione "No parity".

Si noti che, nel caso di "No parity" il bit di parità è comunque presente in decima posizione e fisso al valore logico 1. Considerato anche il bit di stop in undicesima posizione, il carattere trasmesso ha quindi sempre un totale di 11 bits, anche in assenza di parità, come se la configurazione "No parity" corrispondesse alla presenza di due bits di stop:



E' piuttosto comune, in varie implementazioni del protocollo Modbus, trovare una configurazione di bits del carattere al di fuori della specifica ufficiale. Infatti, nel caso di "No parity", alcuni dispositivi considerano la totale assenza del bit di parità portando a 10 il numero di bits totali del carattere. Questa configurazione corrisponde, per esempio, alla classica "9600,N,8,1" (nel caso di velocità di comunicazione a 9600b/s) particolarmente utilizzata su sistemi che non consentono di gestire la parità oppure su sistemi PC.

Nel protocollo Modbus RTU, tutti i bytes appartenenti allo stesso frame devono sottostare a delle rigide specifiche temporali, proprio perché, non disponendo di caratteri speciali da utilizzare come sincronismo, i pacchetti vengono riconosciuti ed isolati gli uni dagli altri in base esclusivamente alle loro temporizzazioni:



Ogni frame deve necessariamente avere tutti i propri bytes vicini tra loro con un tempo massimo, tra la fine di trasmissione di un byte e l'inizio di trasmissione del successivo, pari a 1.5

volte la durata del carattere stesso.
Inoltre per distinguere un frame da quello adiacente occorre che, dopo la trasmissione di un frame, ci sia una pausa pari a 3.5 volte la durata di un carattere. Questa pausa a fine frame, abbinata alla “compattezza” dei bytes dello stesso frame, garantisce la possibilità di distinguere e separare tutti i frames tra loro.

Un dispositivo in trasmissione dovrà quindi assicurare l’invio contiguo dei bytes del frame, per esempio preparando tutti i bytes fino al checksum in un buffer di trasmissione ed attivando successivamente l’invio dell’intero buffer sulla linea seriale.
In ricezione i dispositivi dovranno “catturare” immediatamente tutti i bytes di un frame riattivando ad ogni byte ricevuto un timer (a 1.5 caratteri) per controllare la contiguità dei bytes ed un timer (a 3.5 caratteri) per verificare l’effettiva fine di un frame. Dopo la ricezione di tutto il frame verrà controllata la correttezza del checksum e la corrispondenza dell’indirizzo di Slave.
Solo il livello successivo del protocollo (Application layer) analizzerà la correttezza della PDU ricevuta e in caso positivo provvederà a processarla.

Riguardo al campo indirizzo del frame di comunicazione, il protocollo Modbus utilizza i valori nel range 1÷247 per comunicare con un solo specifico Slave alla volta (modalità Unicast) mentre l’indirizzo speciale 0 è utilizzato per la modalità Broadcast. Una richiesta del Master con indirizzo 0 viene processata da tutti gli Slave ma nessuno di questi provvederà a rispondere al Master proprio per evitare conflitti di trasmissione sul bus. Questa modalità è utilizzata per inviare lo stesso comando contemporaneamente a tutti gli Slave senza però ricevere risposta e quindi senza alcuna conferma dell’avvenuta esecuzione del comando da parte degli Slave. Infine gli indirizzi da 248 a 255 sono “riservati” e consentono al produttore di implementare delle specifiche funzionalità.

Il campo terminale del frame è costituito da due bytes corrispondenti alla parte bassa e alta della word di checksum (CRC). Questo valore permette di verificare l’integrità dei bytes del pacchetto in quanto costituisce una firma relativa ai valori dei bytes che precedono il campo CRC. In trasmissione il checksum è calcolato su tutti i bytes dal primo (indirizzo) all’ultimo del campo Dati ed aggiunto al termine del frame. In ricezione viene calcolato il checksum su tutti i bytes del frame ricevuto (campo CRC compreso). In tal caso, solo un risultato pari a zero indica la correttezza del frame ricevuto.
Per il calcolo del CRC si possono adottare due differenti approcci. Il primo è quello di un algoritmo completo che elabora i valori di tutti i bytes considerati e determina il valore a 16 bit del CRC. Il secondo è più semplice e veloce dal punto di vista del calcolo ma richiede l’utilizzo di una tabella di 256 bytes costanti precalcolate e questo su microprocessori con poca memoria programma può risultare più oneroso. Per maggiori dettagli su questi algoritmi fare riferimento al documento ufficiale del protocollo Modbus seriale:

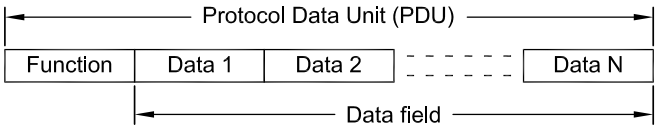
http://www.modbus.org/docs/Modbus_over_serial_line_V1_02.pdf

Durante la comunicazione dei frames possono verificarsi situazioni anomale come la ricezione di un frame incompleto oppure con checksum errato. In tal caso i dispositivi Slave devono scartare questi frames e non rispondere mentre il Master, se verifica degli errori in una risposta degli Slave, potrebbe decidere di ritentare lo scambio dei frames.

Protocollo Modbus su RS485 - Application layer

Il secondo ed ultimo livello software del protocollo Modbus, detto “Application layer“, si occupa della Protocol Data Unit (PDU) ossia della parte utile di informazione contenuta nei frames scambiati tra il Master e gli Slave. Infatti nel precedente livello (Data Link layer) sono definite solo le parti del protocollo relative alla trasmissione delle PDU senza considerarne il loro significato.

La PDU contiene tutte le informazioni necessarie affinché il Master possa effettuare delle operazioni su uno Slave, per esempio la scrittura o la lettura di un parametro.
Per questo i bytes di una PDU sono suddivisi in primo campo (di lunghezza 1 byte) che definisce la particolare operazione da svolgere (codice funzione) ed un eventuale secondo campo Dati specifico del comando da eseguire:



Il campo funzione contiene un valore predefinito per ogni comando implementato. Il protocollo Modbus prevede che solo i valori da 1 a 127 siano utilizzati per codificare il comando, lasciando i valori da 128 a 255 per la codifica dello stesso comando (ma con il bit più alto ad 1) allo scopo di segnalare, nella risposta dello Slave, un eventuale “eccezione” ossia un errore rilevato durante la sua esecuzione.
Il protocollo Modbus suddivide l’insieme dei codici numerici in due gruppi. I codici compresi nei campi 1÷64, 73÷99 e 111÷127 sono utilizzati per le funzioni definite “pubbliche” ossia documentate ufficialmente e quindi univoche nell’ambito del protocollo. I rimanenti codici nei campi 65÷72 e 100÷110 sono liberi per l’implementazione di funzioni custom del protocollo.

Il significato e la lunghezza del campo Dati dipendono dallo specifico comando e questa parte può anche non essere presente. La lunghezza massima del campo Dati è di 252 bytes in quanto la somma complessiva dei bytes di un frame, secondo la specifica, è di 256 bytes.
Il protocollo gestisce valori di tipo 16bit-word utilizzando la notazione “big-Endian” la quale richiede che il byte più significativo della word sia trasmesso per primo. Si noti che ciò non vale per l’invio del CRC che va trasmesso con il byte meno significativo per primo.

I comandi standard del protocollo Modbus relativi alla lettura e scrittura dei valori delle risorse degli Slave fanno riferimento al concetto di “Oggetto” anziché all’indirizzo reale di memoria del dispositivo. Questo permette di svincolare l’indirizzamento delle risorse da parte del protocollo dalle effettive posizioni fisiche delle stesse nell’ambito della memoria interna dello Slave.
Per questo ogni dispositivo dovrà definire quelli che sono gli oggetti accessibili al protocollo assegnandogli un indirizzo Modbus e dovrà di conseguenza occuparsi della loro associazione con le variabili effettive interne nella propria memoria.

Gli oggetti sono stati raggruppati in 4 tipologie differenti con evidente riferimento al tipo di risorse che offrivano i dispositivi del periodo nel quale era stato definito il protocollo:

Primary tables	Object type	Access type	Comments
Discrete Inputs	Single bit	Read-Only	Provided by an I/O slave
Coils	Single bit	Read-Write	Can be alterable by Master
Input Registers	16-bit word	Read-Only	Provided by an I/O slave
Holding Registers	16-bit word	Read-Write	Can be alterable by Master

Le risorse di tipo “Discrete Inputs” fanno riferimento a valori di ingresso digitale (ON/OFF) mentre le risorse “Coils” a valori di uscita digitale quindi forzabili in scrittura ma al tempo stesso leggibili. Le risorse di tipo “Input Registers” e “Holding Registers” sono delle versioni analogiche (valori continui nel range 0-65535) delle precedenti ed utilizzate per ingressi/uscite analogiche o qualsiasi altro valore numerico del dispositivo come ad esempio un parametro di lavoro.

Ai suddetti oggetti sono associati sia dei numeri identificativi che degli specifici indirizzi al fine di poter accedere a queste risorse tramite le PDU del protocollo. Si ricordi che l’indirizzo della risorsa non coincide necessariamente con l’indirizzo di memoria fisica all’interno del dispositivo. E’ compito di quest’ultimo associare, al volo, l’indirizzo Modbus con la variabile della memoria interna, utilizzando per esempio delle tabelle di corrispondenza.
Queste tabelle definiscono quello che si chiama “Modello dati” dello specifico dispositivo e costituiscono una sorta di “manuale di riferimento del programmatore” in quanto elencano tutte le possibili risorse disponibili nello Slave e quelli che sono gli indirizzi ai fini del protocollo.

Gli indirizzi Modbus degli oggetti, essendo definiti mediante una word, possono estendersi nell’intero range da 0 a 65535 mentre per il numero identificativo degli oggetti si utilizza la numerazione da 1 a 65536. Tuttavia la numerazione degli oggetti è solo formale mentre ciò che conta per l’accesso alla risorsa è esclusivamente l’indirizzo inserito nella PDU.
Per ognuna delle 4 aree di suddivisione degli oggetti è possibile utilizzare l’intero range 0÷65535 di indirizzamento in quanto sarà lo stesso codice funzione a definire a quale delle quattro aree si riferisce l’indirizzo.
A questo modo esteso, che permette di indirizzare 65536 oggetti per ognuna delle quattro aree, si affianca un’altra convenzione definita originariamente dal protocollo e che è tuttora utilizzata. In questa convenzione si numerano le risorse in un range ristretto di valori da 0001 a 9999 aggiungendo un offset, multiplo di 10000, a seconda dell’area interessata:

Primary tables	Type prefix	Object Number	PDU Address
Coils	0x	00001÷09999	0000÷9998
Discrete Inputs	1x	10001÷19999	0000÷9998
Input Registers	3x	30001÷39999	0000÷9998
Holding Registers	4x	40001÷49999	0000÷9998

In questo modo è il numero di oggetto a definire a quale delle quattro aree si riferisce e conseguentemente con quale codice funzione eseguire le relative operazioni.

I codici funzione standard predefiniti dalla specifica Modbus sono pochi e buona parte di questi sono usati per leggere o scrivere le variabili di tipo bit e le variabili di tipo word:

Function code	Related area	Operation
1	Coils	Read up to 2000 contiguous memory bits
2	Discrete Inputs	Read up to 2000 contiguous input bits
3	Holding Registers	Read up to 125 contiguous memory words
4	Input Registers	Read up to 125 contiguous input words
5	Single Coil	Write one memory bit
6	Single Register	Write one memory word
15	Coils	Write up to 1968 contiguous memory bits
16	Holding Registers	Write up to 123 contiguous memory words
23	Holding Registers	Read up 125 and write up 121 memory words

Function code 1 - Read Coils			
Request	Function code	1 byte	0x01
	Starting address	2 bytes	0x0000 to 0xFFFF
	Quantity of Coils	2 bytes	1 to 2000 (0x001 to 0x7D0)
Replay	Function code	1 byte	0x01
	Bytes count	1 byte	N/8 or N/8+1 (N is the Quantity of Coils)
	Coils status	n bytes	n=N/8 or n=N/8+1 if (remainder of N/8) ≠ 0
Notes	The first bit addressed is in the bit 0 position of the first byte of replay. If N is not a multiple of eight the remaining bits in the last byte are 0 padded.		

Function code 2 - Read Discrete Inputs			
Request	Function code	1 byte	0x02
	Starting address	2 bytes	0x0000 to 0xFFFF
	Quantity of Inputs	2 bytes	1 to 2000 (0x001 to 0x7D0)
Replay	Function code	1 byte	0x02
	Bytes count	1 byte	N/8 or N/8+1 (N is the Quantity of Inputs)
	Inputs status	n bytes	n=N/8 or n=N/8+1 if (remainder of N/8) ≠ 0
Notes	The first bit addressed is in the bit 0 position of the first byte of replay. If N is not a multiple of eight the remaining bits in the last byte are 0 padded.		

Function code 3 - Read Holding Registers			
Request	Function code	1 byte	0x03
	Starting address	2 bytes	0x0000 to 0xFFFF
	Quantity of Registers	2 bytes	1 to 125 (0x01 to 0x7D)
Replay	Function code	1 byte	0x03
	Bytes count	1 byte	2 x N
	Registers value	2N bytes	N is the Quantity of Holding Registers

Function code 4 - Read Input Registers			
Request	Function code	1 byte	0x04
	Starting address	2 bytes	0x0000 to 0xFFFF
	Quantity of Registers	2 bytes	1 to 125 (0x01 to 0x7D)
Replay	Function code	1 byte	0x04
	Bytes count	1 byte	2 x N
	Registers value	2N bytes	N is the Quantity of Input Registers

Function code 5 - Write Single Coil			
Request	Function code	1 byte	0x05
	Coil address	2 bytes	0x0000 to 0xFFFF
	Coil value	2 bytes	0x0000 for OFF or 0xFF00 for ON
Replay	Function code	1 byte	0x05
	Coil address	2 byte	0x0000 to 0xFFFF
	Coil value	2 bytes	0x0000 for OFF or 0xFF00 for ON

Function code 6 - Write Single Register			
Request	Function code	1 byte	0x06
	Register address	2 bytes	0x0000 to 0xFFFF
	Register value	2 bytes	0x0000 to 0xFFFF
Replay	Function code	1 byte	0x06
	Register address	2 byte	0x0000 to 0xFFFF
	Register value	2 bytes	0x0000 to 0xFFFF

Function code 15 - Write Coils			
Request	Function code	1 byte	0x0F
	Starting address	2 bytes	0x0000 to 0xFFFF
	Quantity of Coils	2 bytes	1 to 1968 (0x001 to 0x7B0)
	Bytes count	1 byte	N/8 or N/8+1 (N is the Quantity of Coils)
	Coils status	n bytes	n=N/8 or n=N/8+1 if (remainder of N/8) ≠ 0
Replay	Function code	1 byte	0x0F
	Starting address	2 byte	0x0000 to 0xFFFF
	Quantity of Coils	2 bytes	1 to 1968 (0x001 to 0x7B0)
Notes	The first bit addressed is in the bit 0 position of the first byte of request. If N is not a multiple of eight the remaining bits in the last byte are 0 padded.		

Function code 16 - Write Holding Registers			
Request	Function code	1 byte	0x10
	Starting address	2 bytes	0x0000 to 0xFFFF
	Quantity of Registers	2 bytes	1 to 123 (0x01 to 0x7B)
	Bytes count	1 byte	2 x N
	Registers value	2N bytes	N is the Quantity of Holding Registers
Replay	Function code	1 byte	0x10
	Starting address	1 byte	0x0000 to 0xFFFF
	Quantity of Registers	2 bytes	1 to 123 (0x01 to 0x7B)

Function code 23 - Read/Write Holding Registers			
Request	Function code	1 byte	0x17
	Read start address	2 bytes	0x0000 to 0xFFFF
	Quantity to read	2 bytes	1 to 125 (0x01 to 0x7D)
	Write start address	2 bytes	0x0000 to 0xFFFF
	Quantity to write	2 bytes	1 to 121 (0x01 to 0x79)
	Write bytes count	1 byte	2 x N
	Write Registers value	2N bytes	N is the Quantity of writing Registers
Replay	Function code	1 byte	0x17
	Read bytes count	1 byte	2 x n
	Read Registers value	2n bytes	n is the Quantity of reading Registers

Il protocollo Modbus prevede una gestione degli errori anche a livello del “Application layer” controllando che le richieste del Master siano lecite. Se uno Slave riceve una PDU con valori non compatibili con le proprie risorse, esso non eseguirà il comando richiesto e risponderà al Master con una PDU particolare (Exception) contenente nel campo Function lo stesso comando richiesto ma con il bit più significativo ad 1. Successivamente al campo Function, trasmetterà un campo dati, con lunghezza 1 byte, contenente il codice del particolare errore verificato:

Exception			
Replay	Function code	1 byte	The function code of request + 128
	Exception code	1 byte	0x00 to 0xFF

Exception code	Name of error	Comments
1	ILLEGAL FUNCTION	Function code is not valid or implemented.
2	ILLEGAL DATA ADDRESS	Object address is not valid for the Slave.
3	ILLEGAL DATA VALUE	Writing value is not valid for the addressed object.
4	SLAVE DEVICE FAILURE	Fatal error occurred during the requested operation.

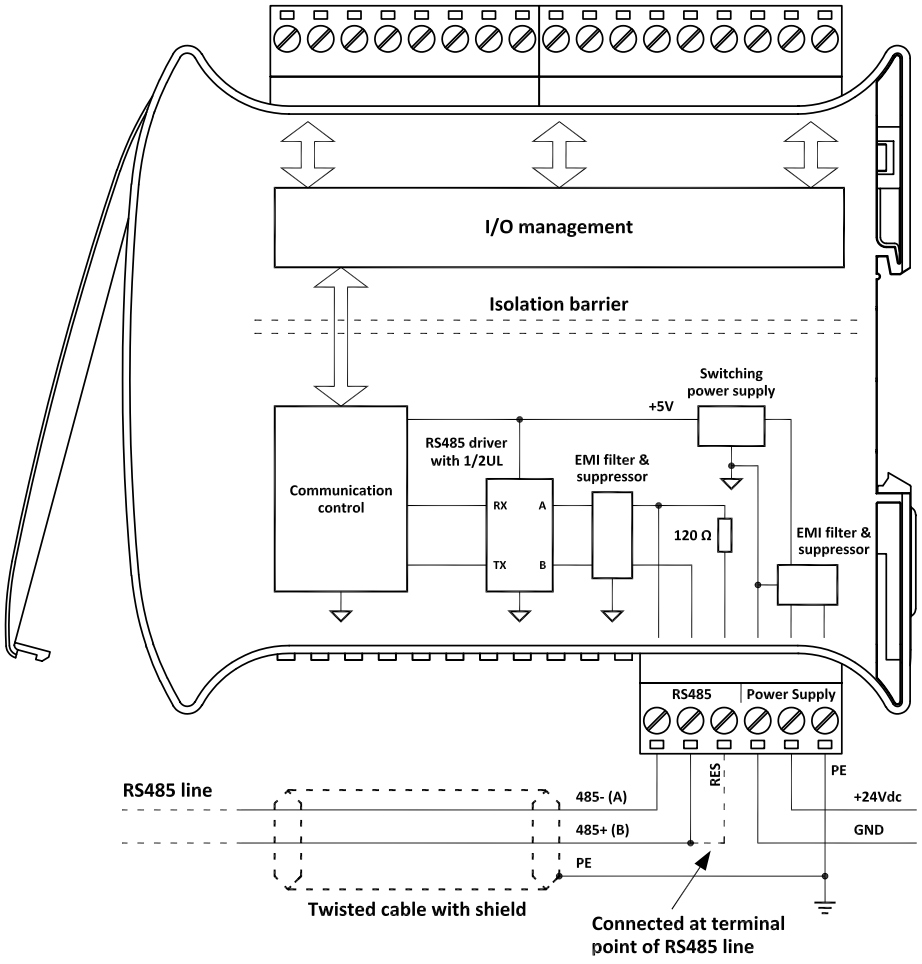
Il protocollo Modbus definisce anche altri codici funzione utilizzati per funzioni di servizio, lettura dello stato, diagnosi ed identificazione del dispositivo ed anche ulteriori altri codici di errore. Per maggiori dettagli a riguardo fare riferimento al documento ufficiale del protocollo Modbus:

http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf

Protocollo Modbus su RS485 - Moduli Slave Overdigit

I moduli Slave Overdigit sono conformi alla specifica ufficiale Modbus ma al tempo stesso sfruttano le caratteristiche di flessibilità del protocollo per ottenere delle prestazioni notevolmente superiori rispetto a qualsiasi altro dispositivo Modbus presente sul mercato. L'incremento di prestazioni può essere completamente sfruttato solo nel caso di utilizzo dei Web-PLCs Overdigit in quanto è richiesta anche l'implementazione di codici funzione custom. Tuttavia ciò dimostra la possibilità di utilizzare il protocollo Modbus anche per soddisfare le esigenze più specifiche, in apparenza realizzabili solo con lo sviluppo di protocolli totalmente proprietari. In questo modo è quindi possibile mantenere la compatibilità, sullo stesso bus di campo, tra dispositivi Modbus standard e sistemi che possono ottenere prestazioni superiori ma che comunque utilizzano anch'essi il protocollo Modbus.

Per quanto riguarda il "Physical layer" i moduli Overdigit sono dotati di un driver RS485 con 1/2 unità di carico. Ciò significa che è possibile collegare fino ad un massimo di 64 dispositivi sul bus senza utilizzare ripetitori di linea. Il connettore a 6 poli, standardizzato per tutti i moduli, consente di alimentare i dispositivi e di connettere l'interfaccia seriale al bus di campo. All'interno dei moduli è prevista la resistenza di terminazione il cui inserimento è ottenuto mediante un ponticello esterno nella morsettiera tra il polo del segnale B ed il polo RES:



I segnali A e B sono anche indicati rispettivamente come 485- e 485+ evidenziando la polarità della linea nello stato attivo di pilotaggio del bit=0, in contrapposizione allo stato non attivo della trasmissione. Si noti infatti che lo stato non attivo, per la presenza delle resistenze di polarizzazione di default, assume la stessa polarità associata alla trasmissione del bit=1.

Il driver della linea RS485 utilizzato nei moduli Overdigit permette comunicazioni seriali fino alla massima velocità di 1Mb/s anche se normalmente i sistemi Modbus presenti sul mercato non superano mai la velocità di 115200b/s.

Il “Data Link layer” implementato nei moduli Overdigit prevede quattro modalità del bit di parità:

Parity	Total bits	Comments
No	11	2 stop bits, compliant to specification
Even	11	Even parity + 1 stop bit, compliant to specification
Odd	11	Odd parity + 1 stop bit, compliant to specification
No-1stop	10	Parity bit not present (1 stop bit), out of specification

La modalità “No-1stop” non è prevista nella specifica del protocollo Modbus ma è stata comunque implementata per ottenere la compatibilità con i numerosi sistemi commerciali che hanno “deviato” dalla specifica e che hanno di fatto apportato una variante, abbastanza diffusa ma non ufficiale, al protocollo Modbus seriale.

Riguardo l’implementazione delle Protocol Data Unit (PDU), definite nel ”Application layer”, i moduli Overdigit prevedono tutti i codici comando standard necessari alle gestione delle risorse dello specifico modulo. Per l’indirizzamento degli oggetti è stato adottato l’indirizzamento esteso per cui il campo Address della PDU si può estendere su tutto il range 0÷65535 per ciascuna delle quattro aree dati definite dal protocollo. In pratica comunque, per il funzionamento dei vari dispositivi, sono normalmente necessari solo pochi indirizzi a partire dal valore 0.

Inoltre, nei moduli Overdigit, sono stati aggiunti alcuni codici funzione custom, secondo quanto consentito dal protocollo Modbus, che permettono di velocizzare notevolmente le operazioni di aggiornamento delle risorse del modulo.
In particolare sono stati aggiunti i codici 100, 101, 102 rispettivamente per aggiornare tutti i bytes di ingresso, di uscita e di ingresso/uscita del modulo. In tal caso tutte le risorse di I/O del modulo sono compattate in un unico stream di bytes nell’area Data della PDU.
Il codice comando che maggiormente permette di rendere efficace l’aggiornamento del modulo è quello per la lettura degli ingressi e la scrittura delle uscite:

Function code 102 - Read and Write I/O Area			
Request	Function code	1 byte	0x64
	Write Out bytes count	1 byte	N
	Write Out bytes value	N bytes	N is the Quantity of writing Output bytes
Replay	Function code	1 byte	0x64
	Read In bytes count	1 byte	n
	Read In bytes value	n bytes	n is the Quantity of reading Input bytes

L’utilizzo di questo comando permette di aggiornare tutta l’area immagine degli ingressi e delle uscite del modulo in un solo scambio di frames. Per ottenere ciò nei convenzionali moduli Slave con protocollo Modbus è necessario eseguire più interrogazioni successive ed utilizzare variabili di tipo word anche se l’informazione da scambiare può essere contenuta in un solo byte. Nel caso dei moduli Overdigit le risorse di I/O sono invece inserite all’interno di un array di bytes lungo quanto necessario per contenere lo stato di tutte le variabili associate alle risorse.

Le quantità dei bytes di I/O possono essere note a priori oppure determinate in modo dinamico, per esempio al power-on, mediante il codice comando standard 17 (Report Slave ID):

Function code 17 - Report Slave ID			
Request	Function code	1 byte	0x11
Replay	Function code	1 byte	0x11
	Bytes count	1 byte	N
	ID bytes value	N bytes	N is the Quantity of Identification bytes

Il campo dati nella PDU della risposta è costituito da un unico array di 20 bytes che contiene, in posizioni predefinite, le informazioni relative al modulo e alle dimensioni delle aree di I/O:

ID byte offset	Data type	Comments
0÷7	Fixed 8 bytes string	Module name, for example “EX1608DD“
8÷15	Fixed 8 bytes string	Firmware release, for example “r.01.008”
16-17	Word	Number of bytes for Input Area
18-19	Word	Number of bytes for Output Area

L’utilizzo del codice comando 102, in abbinamento con la massima velocità di comunicazione impostabile (1Mb/s), permette di aggiornare tutta l’area di I/O dei moduli in tempi notevolmente ridotti rispetto a quanto normalmente si ottiene su altri sistemi commerciali.

Per esempio, nel caso del modulo Overdigit EX1608DD con 16 ingressi + 8 uscite digitali, il tempo complessivo di comunicazione è di soli 250µs. La maggior parte dei moduli analoghi presenti sul mercato spesso limitano la velocità massima a 38400b/s e non prevedono comandi addizionali. Per questo, per effettuare l’aggiornamento degli I/O, è richiesta l’esecuzione di una lettura di un “Input Register” e di una scrittura di un “Holding Register”. Considerando anche un tempo nullo di elaborazione dei frames e quindi di una risposta immediata da parte dello slave, si ottiene un tempo complessivo di comunicazione di circa 13ms ossia ben 52 volte maggiore.

Ovviamente prestazioni così spinte richiedono l’implementazione anche di funzioni custom, come avviene nella libreria Modbus per CoDeSys dei Web PLCs Overdigit, ma il vantaggio principale, rispetto allo sviluppo di un protocollo completamente proprietario, è che comunque la struttura principale del protocollo rimane conforme allo standard Modbus permettendo la compatibilità di questi moduli con tutti i più tradizionali dispositivi.