Översikt

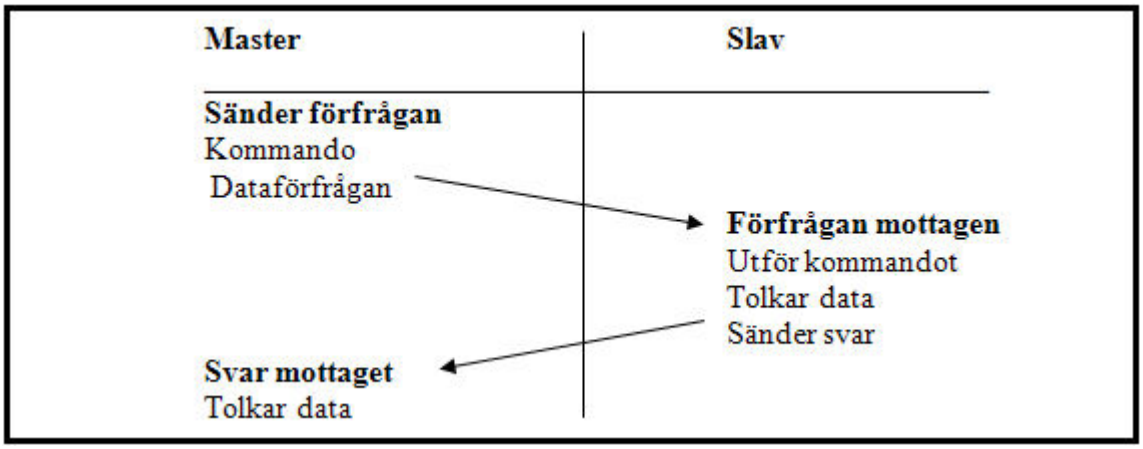
Modbus protokollet är en allmänt använd och väldokumenterad kommunikationsmetod.

Det är ett enkelt och effektivt sätt att programmera våra olika produkter

A typiskt Modbus-paket ser ut så här:

Byte1	Enhet ID, måladress för ett bestämt meddleande
Byte2	Funktion
Byte3	Startadress för det bestämda lagringregistret(n) att läsas eller skrivas, hi byte,
Byte4	Startadress low byte
Byte5	Antal register att läsa/skriva (hi byte)
Byte6	Antal register att läsa/skriva (low byte)
Byte7	CRC hi byte
Byte8	CRC low byte

Under den seriella kommunikationen kommer slaven genast att skicka ett svar på masterns förfrågan.



[Observera]: Ett fel i en meddelandeöverföring ses normalt som ett timeout-fel.

Om man sänder en felaktig förfrågan beroende på att bytes blivit förvanskade eller saknas, kommer detta inte att ge något svar som resulterar i ett time-out fel.

Mjukvaruverktygen kan hittas på: http://www.modbustools.com/modbus_poll.asp

Om din applikation kan läsa och skriva till en separat PC som använder applikationen ¨Master-Slave¨kan Du läsa och skriva till våra regulatorer MPC..

Observera: När man använder mjukvara Modbus Poll ska adresseringen anges som¨Protocol Adresses (base 0)¨under display-menyn.

Exempel på Modbus

Kommandot READ(0x03):

Denna funktion används för att läsa innehållet i multipla minnesregister.

Modbus Master överordnade enhet måste ange enhetens ID, dess startregister och hur många register som önskas.

Om ett datapaket innehåller 2 byte så sänds först Hi byte och sedan Lo byte.

Den överordnade eneheten Maste modbus i modbusnätverket kommer att sända ett READ-kommando:

- Enhet ID=11
- Läs 6 bytes med data
- Starta vid registernummer 107 (6Bh)

Byte #	Namn (Hex)	Data	Beskrivning
Byte1	Slaveadress	11	MPC med ID11 kommer att läsas
Byte2	Funktion	03	Läsfunktion
Byte3	Startadress Hög	00	
Byte4	Startadress Låg	6B	Läsning startar från register #6B
Byte5	antal register att läsa Hi	00	
Byte6	Antal register att läsa Lo	03	Läs totalt 3 rgister
Byte7	Felkontroll (CRC) HI byte	XX	CRC beräknas av en CRC rutin CRC
Byte8	Felkontroll (CRC) LO byte	XX	routin beskrivs nedan

Slavenheten med ID=11 kommer att svara den överordnade enheten Master inom några millisekunder med följande svar

Byte #	Field Name (Hex)	Data	Description
Byte1	Slavadress	11	Slaven med ID11 svarar
Byte2	Funktion	03	Vi besvara ett läskommando
Byte3	Räkna byte	06	6 bytes kommer
Byte4	Data1 Hög	02	byte1 av data
Byte5	Data1 Låg	2B	byte2 av data
Byte6	Data2 Hög	00	byte3 av data
Byte7	Data2 Låg	00	byte4 av data
Byte8	Data3 Hög	00	byte5 av data
Byte9	Data3 Låg	64	byte6 avdata
Byte10	Felkontroll (CRC) HI byte	XX	CRC beräknas av en CRC rutin
Byte11	Felkontroll (CRC) LO byte	XX	routin beskrivs nedan

Exempel på Read-kommando

The Master sends the Read querie:

Slave Address	Function	Starting Address Hi	Starting Address Lo	No. of Regs Hi	No. of Regs Lo	CRC Hi Byte	CRC Lo Byte
11	3	0	6Bh 107	0	3	xx	xx

The device node sends back the following response:

Slave Address	Function	Byte Count	Data1 Hi	Data1 Lo	Data2 Hi	Data2 Lo
11	3	6	(02h) 2	(2Bh) 43	(00h) 0	(00h) 0

Data3 Hi	Data3 Lo	CRC Hi Byte	CRC Lo Byte			
(00h) 0	(64h) 100	xx	xx			

Kommandot WRITE (0x06):

Denna funktion används för att skriva till ett enda minnesregister.

Modbus Master överordnade enhet måste ange enhetens ID, vilken registeradress som skall skrivas och önskat data.

Den överordnade enheten Master i modbusnätverket kommer att sända ett WRITE-kommando:

- Enhet ID=11
- Skriv till adress 11
- Ange data 3 (03h)

Byte #	Namn (Hex)	Data	beskrivning
Byte1	Slavadress	11	måladress
Byte2	Funktion	06	det här är ett skrivkommando
Byte3	Registeradress Hög	00	den adress man skriver till, hi byte
Byte4	Registeradress låg	01	den man adress man skriver till, low byte
Byte5	Data Hög	00	data som vi skriver, hi byte
Byte6	Data Låg	03	data som vi skriver, low byte
Byte7	Felkontroll (CRC) HI byte	XX	CRC beräknas av en CRC rutin
Byte8	Felkontroll (CRC) LO byte	XX	rutin beskrivs nedan

Slavenheten med ID=11 kommer att svara den överordnade enheten Master inom några millisekunder med följande svar:

Byte #	Namn (Hex)	Data	Description
Byte1	Slavadress	11	måladress
Byte2	Funktion	06	det här är ett skrivkommando
Byte3	Registeradress Hög	00	den adress man skriver till, hi byte
Byte4	Registeradress Låg	01	den adress man skriver till, low byte
Byte5	Data Hög	00	data som vi skriver, hi byte
Byte6	Data Låg	03	data som vi skriver, low byte
Byte7	Felkontroll (CRC) HI byte	XX	CRC beräknas av en CRC rutin
Byte8	Felkontroll (CRC) LO byte	XX	rutin beskrivs nedan

[Observera]: I detta fall sänder slavenheten bara tillbaka meddelandet för att informera den överordnade enheten Master om att frågan har blivit korrekt mottagen.

Exempel på Write-kommando

The Master sends the Write querie:

Slave Address	Function	Register Address Hi	Register Address Lo	Data Hi	Data Lo	CRC Hi Byte	CRC Lo Byte
11	6	0	(01h) 1	0	3	xx	xx

The device node sends back the following response:

Slave Address	Function	Register Address Hi	Register Address Lo	Data Hi	Data Lo	CRC Hi Byte	CRC Lo Byte
11	6	0	(01h) 1	0	3	xx	xx

Kommandot Multiple-Write (0x10):

Denna funktion används för att skriva till ett minnesregister med ett flertal adresser.
Den överordnade enheten Master måste ange enhetens ID, dess startadressregister, hur många register som önskas och datat.
OBS: Detta används endast för uppdatering av inbyggda program. Det används inte för att skriva enhetsregister

Den överordnade enheten Master på Modbusnätverket kommer att sända ett Multipelt write-kommande network will issue a multiple-write command:

- Enhet ID=11
- Skriv till adress 291 (123h)
- Antal register 3
- Data1 = 10 (000Ah)
- Data2 = 11 (000Bh)
- Data3 = 12 (000Ch)

Byte #	Field Name (Hex)	Data	Berskrivning
Byte1	Slavadress	11	Måladress ID 11
Byte2	Funktion	10	Detta är ett multipelt write-kommando
Byte3	Registrets startadress Hi	01	Detta är adressen som vi för närvarande skriver till i kodutrymmet på enheten
Byte4	Registrets startadress Lo	23	I det här fallet vill vi skriva till registeradress 0x0123
Byte5	Antal register att skriva till HI	00	Vi kommer att skriva ett växlande antal bytes åt gången
Byte6	Antal register att skriva LO	10	I det här fallet vill skriva till 10H eller 16 register
Byte7	Beräkna byte	20	Om beräkning av byte är samma som antalet register, behandla 8 bits Om beräkning av byte är dubbla antalet register, behandla 16 bits

Byte #	8 bits	Byte #	16 bits
Byte8	Data 1	Byte8	Data1 Hi
Byte9	Data 2	Byte9	Data1 Lo
Byte10	Data 3	Byte10	Data2 Hi
Byte11	Data 4	Byte11	Data2 Lo
[…]		[…]	
Byte22	Data 15	Byte38	Data16 Hi
Byte23	Data 16	Byte39	Data16 Lo
Byte 24	Felkontroll HI	Byte40	Felkontroll HI
Byte 25	Felkontroll LO	Byte41	Felkontroll LO

[Observera]: Byte 7 används som en byte-räknare.
Om byte-räkningen är samma som antal register att skriva, vet vi att vi behandlar ett 1 byte-register.
Likaledes vet vi att om byte-räkningen är dubbelt antal register behandlar vi inte ett 2 byte-register.

Slavenheten med ID=11 kommer att svara den överordnade enheten Master inom fem millisekunder med följande svar.

Byte #	Namn (Hex)	Data	Beskrivning
Byte1	Slavadress	11	Måladress nod ID
Byte2	Funktion	10	Detta är ett multipelt write-kommando
Byte3	Registrets startadress Hi	00	Startadress som vi just skriver till hi byte
Byte4	Registrets startadress Lo	01	Startadress low byte
Byte5	Antal register Hi	00	Antal register som skall skrivas till, hi byte
Byte6	Antal register Lo	0A	Antal register, low byte
Byte7	Felkontroll (CRC) HI byte	XX	CRC beräknas av en CRC rutin CRC
Byte8	Felkontroll (CRC) LO byte	XX	Rutin är tidigare beskriven

Exampel på kommandot Multiple-Write

The Master sends the Multiple-Write querie:

Slave Address	Function	Starting Address Hi	Starting Address Lo	Quantity. of Regs Hi	Quantity. of Regs Lo	Byte Count
11	(10h) 16	(01h) 1	(23h) 35	0	3	6

Data1 Hi	Data1 Lo	Data2 Hi	Data2 Lo	Data3 Hi	Data3 Lo	CRC Hi Byte	CRC Lo Byte
(00h) 00	(2Ah) 10	(00h) 00	(0Bh) 12	(00h) 00	(0Ch) 13	xx	xx

Slave Address	Function	Starting Address Hi	Starting Address Lo	Quantity. of Regs Hi	Quantity. of Regs Lo	CRC Hi Byte	CRC Lo Byte
11	10	(01h) 1	(23h) 35	0	3	xx	xx

CRC FELKORRIGERINGAR

Följande är en samling koder somhjälper Dig att komma igång med applikationerna.

static unsigned char auchCRCHi[] = {

0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01,
0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81,
0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01,
0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01,
0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01,
0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40
} ;

/* Tabell med CRC värden för low–order byte */
static unsigned char auchCRCLo[] = {

0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06, 0x07, 0xC7, 0x05, 0xC5, 0xC4,
0x04, 0xCC, 0x0C, 0x0D, 0xCD, 0x0F, 0xCF, 0xCE, 0x0E, 0x0A, 0xCA, 0xCB, 0x0B, 0xC9, 0x09,
0x08, 0xC8, 0xD8, 0x18, 0x19, 0xD9, 0x1B, 0xDB, 0xDA, 0x1A, 0x1E, 0xDE, 0xDF, 0x1F, 0xDD,
0x1D, 0x1C, 0xDC, 0x14, 0xD4, 0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6, 0xD2, 0x12, 0x13, 0xD3,
0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3, 0xF2, 0x32, 0x36, 0xF6, 0xF7,
0x37, 0xF5, 0x35, 0x34, 0xF4, 0x3C, 0xFC, 0xFD, 0x3D, 0xFF, 0x3F, 0x3E, 0xFE, 0xFA, 0x3A,
0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38, 0x28, 0xE8, 0xE9, 0x29, 0xEB, 0x2B, 0x2A, 0xEA, 0xEE,
0x2E, 0x2F, 0xEF, 0x2D, 0xED, 0xEC, 0x2C, 0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7, 0xE6, 0x26,
0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0, 0xA0, 0x60, 0x61, 0xA1, 0x63, 0xA3, 0xA2,
0x62, 0x66, 0xA6, 0xA7, 0x67, 0xA5, 0x65, 0x64, 0xA4, 0x6C, 0xAC, 0xAD, 0x6D, 0xAF, 0x6F,
0x6E, 0xAE, 0xAA, 0x6A, 0x6B, 0xAB, 0x69, 0xA9, 0xA8, 0x68, 0x78, 0xB8, 0xB9, 0x79, 0xBB,
0x7B, 0x7A, 0xBA, 0xBE, 0x7E, 0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C, 0xB4, 0x74, 0x75, 0xB5,
0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71, 0x70, 0xB0, 0x50, 0x90, 0x91,
0x51, 0x93, 0x53, 0x52, 0x92, 0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94, 0x54, 0x9C, 0x5C,
0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E, 0x5A, 0x9A, 0x9B, 0x5B, 0x99, 0x59, 0x58, 0x98, 0x88,
0x48, 0x49, 0x89, 0x4B, 0x8B, 0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D, 0x4C, 0x8C,
0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42, 0x43, 0x83, 0x41, 0x81, 0x80,
0x40
} ;

Exempel för att beräkna CRC i meddelandedata lagrat i minnesplats *puchMsg

```
unsigned short CRC16(unsigned char *puchMsg, unsigned char usDataLen)
{
    unsigned char uchCRCHi = 0xFF ; /* high byte of CRC initialized */
    unsigned char uchCRCLo = 0xFF ; /* low byte of CRC initialized */
    unsigned uIndex ; /* will index into CRC lookup table */
     while (usDataLen—) /* pass through message buffer */
     {
          uIndex = uchCRCHi ^ *puchMsg++ ; /* calculate the CRC */
         uchCRCHi = uchCRCLo ^ auchCRCHi[uIndex] ;
          uchCRCLo = auchCRCLo[uIndex] ;
     }
    return (uchCRCHi << 8 | uchCRCLo) ;
}
```