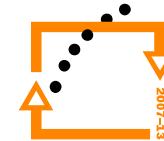
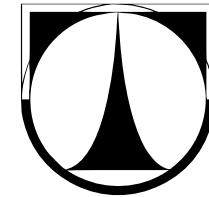




MINISTERSTVO ŠKOLSTVÍ,
MLÁDEŽE A TĚLOVÝCHOVY



OP Vzdělávání
pro konkurenční schopnost
2007-13



INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

ŘPS – úlohy MODBUS

Ing. Josef Grošman

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií

Tento materiál vznikl v rámci projektu ESF CZ.1.07/2.2.00/07.0247

Reflexe požadavků průmyslu na výuku v oblasti automatického řízení a měření,
který je spolufinancován Evropským sociálním fondem a státním rozpočtem ČR



Řídicí počítačové systémy

Úlohy pro samostatná cvičení

Implementace protokolu MODBUS ASCII na PC a mikropočítačích řady '51 pro uzly Master (Klient) i Slave (Server).

Rozhraní: RS232 resp. RS485, standardní rámec 7,N,2

Implementace protokolu MODBUS RTU na PC a mikropočítačích řady '51 pro uzly Master (Klient) i Slave (Server).

Rozhraní: RS232 resp. RS485, standardní rámec 8,N,2

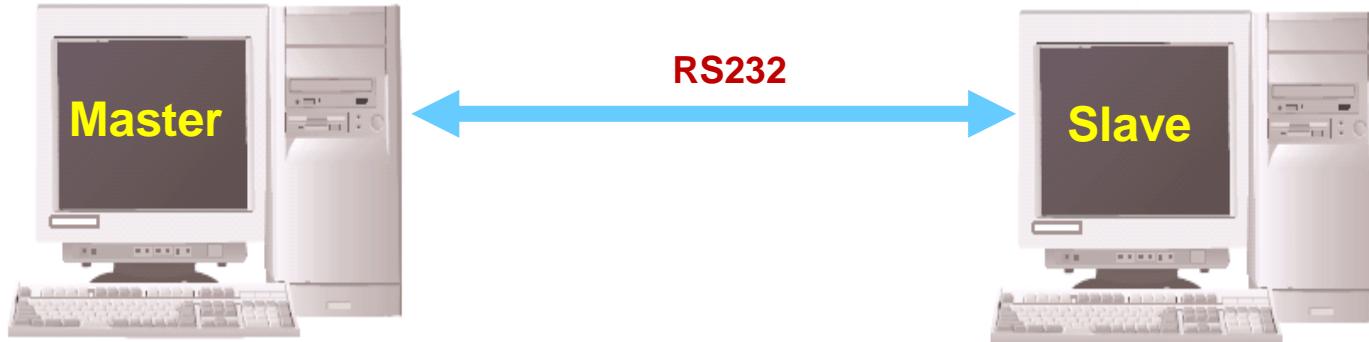
1. část: propojení PC – PC (C# MSVS nebo Pascal DELPHI) (RS232)
2. část: propojení PC - mikropočítač AT89C51CC03 (RS232-USB)
3. část: mikropočítač AT89C51CC03 - mikropočítač AT89C51CC03 (RS485)

Funkce pro podporu aplikace protokolu MODBUS:

v souboru Modbus.dll a Modbus.cs pro PC (MsVS-C#),
v souboru Modbus.H a Modbus.C pro mikropočítač AT89C51CC03



pro PC (1. resp. 2. část)



Podpora pro PC **Modbus.dll** (zdrojový kód **Modbus.cs**)

C:\PRS_podklady\modbus\sharp\
N:\RPS\cviceni_04_modbus\sharp\

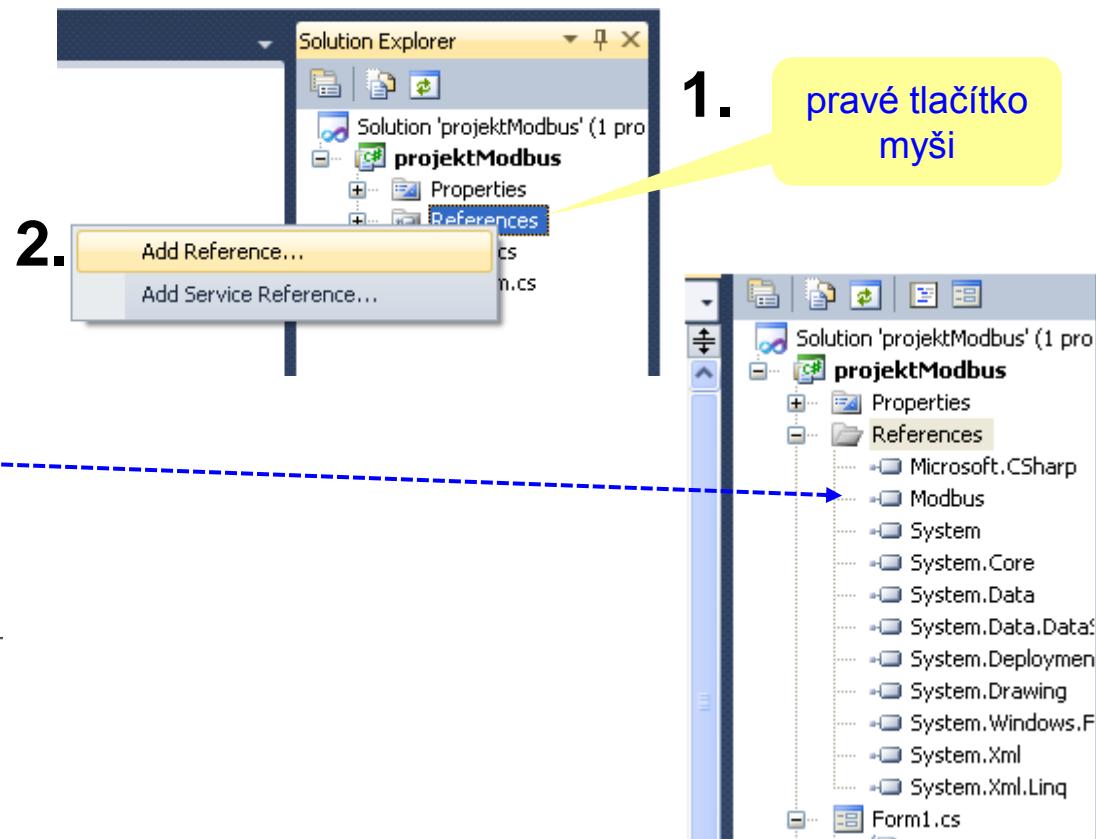
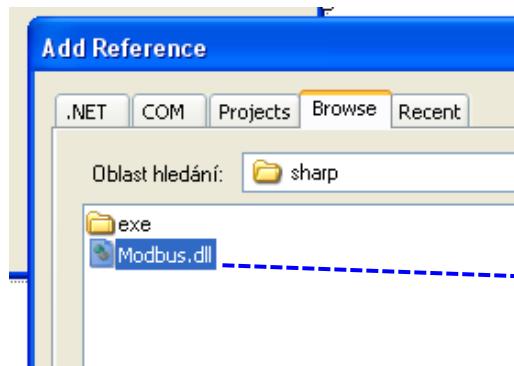
modbus.dll

Podpora pro testování ModbusMaster.exe a ModbusSlave.exe

C:\PRS_podklady\modbus\sharp\exe\
N:\RPS\cviceni_04_modbus\sharp\exe\

ModbusMaster.exe
ModbusSlave.exe

Zařazení Modbus.dll do aplikace



4.

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO.Ports;
using Modbus;

```

```

namespace projektModbus
{
}

```



pro mikropočítač (2. část)



Podpora pro mikropočítač

Modbus.c, Modbus.h

C:\RPS_podklady\modbus\C\
N:\RPS\cviceni_04_modbus\C\

MODBUS.C
MODBUS.H
MAIN.C
ADC.C
LCD.C
LEDBAR.C
TYPY.H



Pro 3.část : mikropočítač – mikropočítač

je nezbytné

- 1. správně nastavit propojky pro modul UART
buď přenos konektorem USB
nebo přenos konektory RS232/485**

- 2. správně přepínat budič RS485
pro příjem
nebo pro vysílání**

**Propojky volby
pro modul UART**

USB x RS

Pro nahrávání
programu : **USB**
Aplikace : **RS**



Propojky volby RS

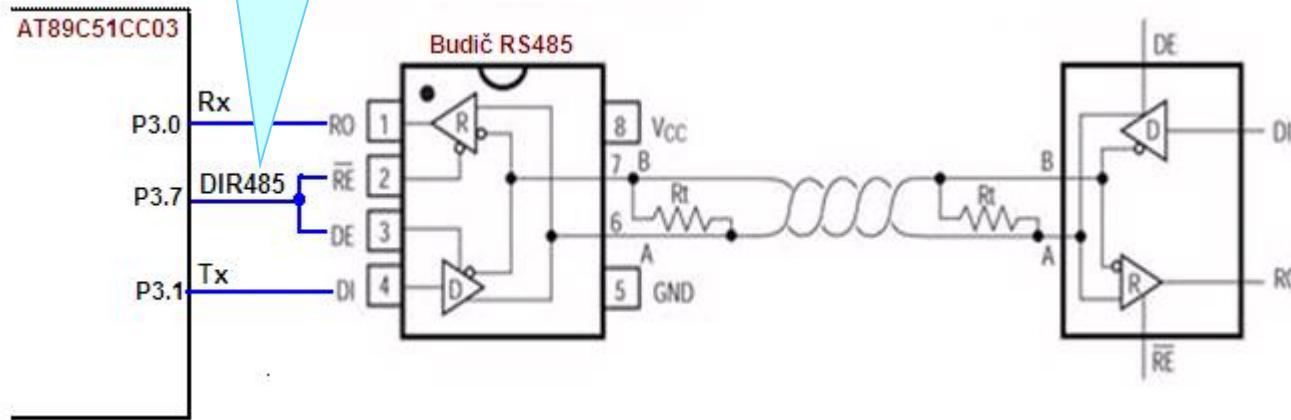
RS232 x RS45

Aplikace : **RS85**

RS45 konektory



#define DIR485 P3_7



DIR485	směr
0	Rx (příjem)
1	Tx (vysílání)

1. Nastavit na příjem (0)
2. Před vysláním zprávy nastavit na vysílání (1) a po vyslání zprávy zpět na příjem (0)



Řídicí počítačové systémy

Úlohy pro samostatná cvičení

úloha	MA1	MA2	MA3	MA4	MA5	MA6	MR1	MR2	MR3	MR4	MR5	MR6
ASCII str. 11	MA1M MA1S	MA2M MA2S	MA3M MA3S	MA4M MA4S	MA5M MA5S	MA6M MA6S						
RTU str. 139							MR1M MR1S	MR2M MR2S	MR3M MR3S	MR4M MR4S	MR5M MR5S	MR6M MR6S
Funkce	1 6	3 5	1 3	5 6	3 6	1 5	1 6	3 5	1 3	5 6	3 6	1 5
str.	36	53	70	88	105	122	169	183	203	221	238	255

1	Master vysílá požadavek na čtení 1bitové informace (Coil) ze Slavu
3	Master vysílá požadavek na čtení 16bitové informace (Registr) ze Slavu
5	Master zapisuje 1bitovou informaci (Coil) do Slavu
6	Master zapisuje 16bitovou informaci (Registr) do Slavu



Definované a doporučené hodnoty

význam	symbol	hodnota	úlohy
Adresa uzlu Slave	ADR_S	1	všechny
Funkce čtení bitu	FCE_RBIT	1	MA1, MA3, MA6,
Adresa čteného bitu	BIT_RD	0	MR1, MR3, MR6
Funkce zápis bitu	FCE_WBIT	5	MA2, MA4, MA6,
Adresa zapisovaného bitu	BIT_WR	0	MR2, MR4, MR6
Funkce čtení registru	FCE_RREG	3	MA2, MA3, MA5,
Adresa čteného registru	REG_RD	0	MR2, MR3, MR5
Funkce zápis registru	FCE_WREG	6	MA1, MA4, MA5,
Adresa zapisovaného registru	REG_WR	0	MR1, MR4, MR5



MINISTERSTVO ŠKOLSTVÍ,
MLÁDEŽE A TĚLOVÝCHOVY



ŘPS – úlohy MODBUS ASCII

Reflexe požadavků průmyslu na výuku v oblasti automatického řízení a měření

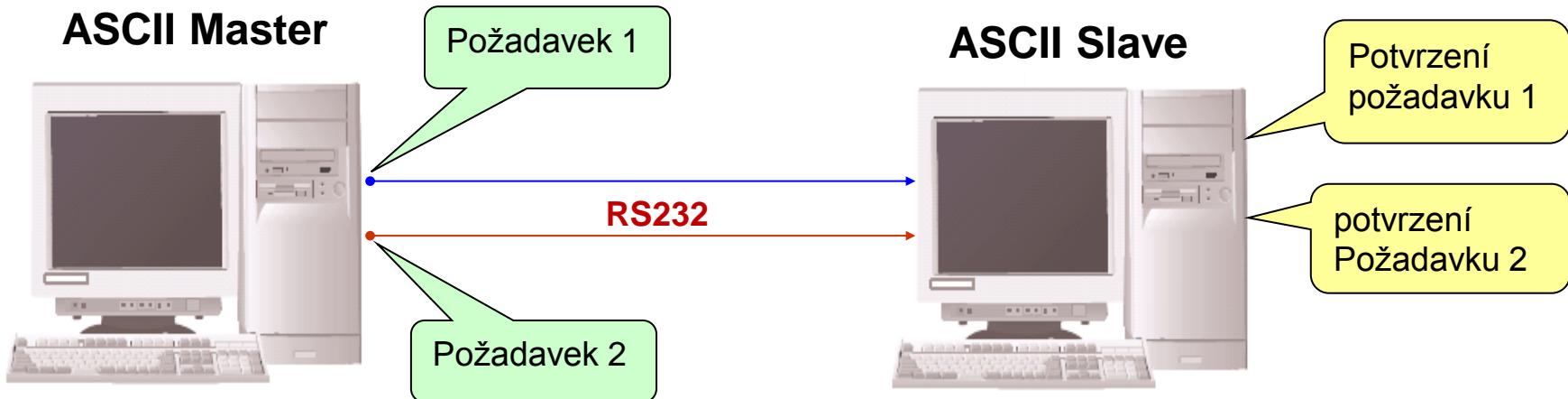
MODBUS ASCII

Úlohy MA1, MA2, MA3, MA4, MA5, MA6

MA1M, MA1S, MA2M, MA2S, MA3M, MA3S, MA4M, MA4S, MA5M, MA5S, MA6M, MA6S



PC-PC (varianta C#)





Podpora pro PC

Class lib **Modbus.dll** - zdrojový kód **Modbus.cs**

```
namespace Modbus;
```

```
class ModbusASCII
```

```
byte AHHex(byte b);  
byte HexAsc(byte b);
```

```
int wrByte(byte b, byte[] bf, int n);  
int wrWord(ushort w, byte[] bf, int n) ;  
int wrEoT(byte[] bf, int n)::
```

```
int wrOne(byte adr, byte fce, ushort reg, ushort val, byte[] bf);  
int Rd(byte adr, byte fce, ushort reg, ushort val, byte[] bf);
```

```
byte RdByte(byte[] bf, int n);  
ushort RdWord(byte[] bf, int n);
```

```
int AnsRd(byte adr, byte fce, byte bytes, byte[] vals, byte[] bf);  
int AnsWr(byte adr, byte fce, ushort reg, ushort val, byte[] bf);  
int AnsErr(byte adr, byte fce, byte er, byte[] bf);
```

```
byte Lrc(byte[] bf, int len):byte;
```



Užité metody třídy ModbusASCII v aplikaci z Modbus.dll

aplikační	pomocné
WrOne	RdByte
Rd	WrByte
AnsWr	Lrc
AnsRd	WrEoT
AnsErr	RdWord

Poznámka: v hlavním programu v sekci **using** přidat **Modbus**



:, adr. slavu, kód funkce, adr. registrů a bitů, hodnoty, LRC, CRLF



bfout

→ RS232



bfin

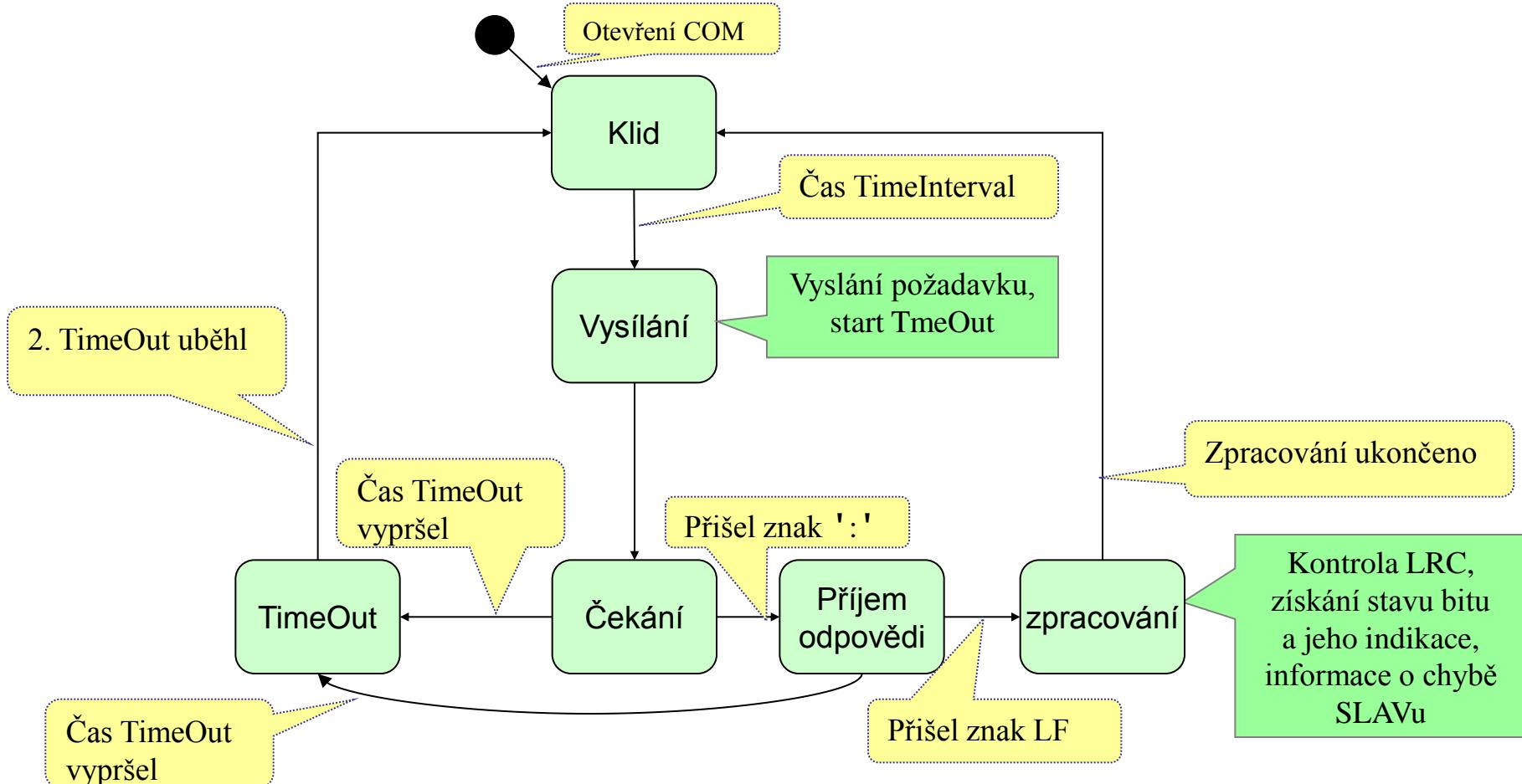
← RS232

```
byte []bfin = new byte[512];  
byte []bfout = new byte[512];
```

bfout[0]	:
bfout[1],bfout[2]	adresa slavu
bfout[3],bfout[4]	kód funkce



Master – zjednodušený stavový diagram



enum Tstav{stKlid,stVysilani,stCekani,stPrijem,stTimeOut};



Master – vyslání požadavku

Časovač Sample

střídavě každých cca 200 ms vysílá rámec s funcí **f1** a **f2**



ModbusASCII Ma;
bool prep;
Tstav stav;

Tick
Sample

```
if(comPort.isOpen && stav == Tstav.stKlid)
{
```

```
    stav=Tstav.stVysilani;
    prep= !prep;
    if(prep) n=Ma.Rd(ADR_S,...);
    else n=Ma.Rd(ADR_S,...);
```

n=Ma.Rd(ADR_S,...)
n=Ma.WrOne(ADR_S,...)

příprava

```
n=Ma.WrByte(Ma.Lrc(bfout,n-1),bfout,n);
n=Ma.WrEoT(bfout,n);
comPort.write(bfout,0,n);
TimerOut.Interval=500;
TimerOut.Enabled=true;
```

vyslání

aktivace TimeOut

Stav čekání

}

```
stav=Tstav.stCekani;
```

Master – příjímání odpovědi

DataReceived

začátek zprávy

další znaky

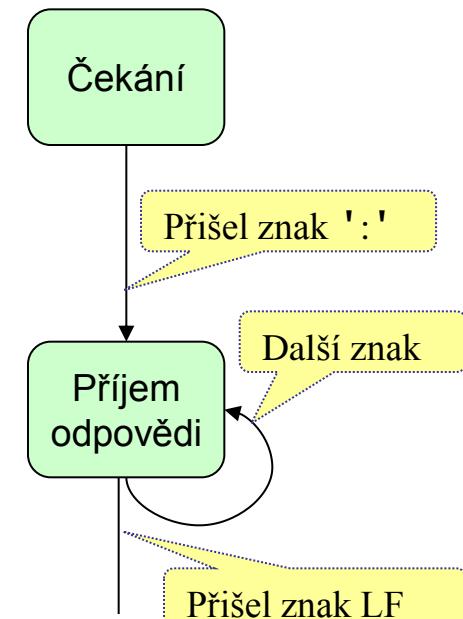
konec zprávy,
vypnutí TimeOut

```

while(comPort.BytesToRead > 0) {
    byte b = (byte)comPort.ReadByte();
    switch (stav) {
        case Tstav.stCekani:
            if(b==(byte)':')
            {
                stav=Ttav.stPrijem;
                bfin[ix=0]=b;
            } break;
        case Tstav.stPrijem:
            {
                if(b==(byte)':')ix=0 else ix++;
                bfin[ix]=b;
                if (b==(byte)'\n')
                {
                    TimerOut.Enabled=false;
                    .
                    .
                }
            }
    }
}

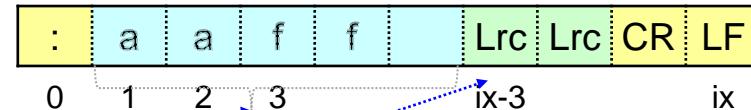
```

Poznámka:
Hlídat index!



Master – zpracování odpovědi

bfin



1. LRC

```
if(Ma.Lrc(bfin,ix-4)!=Ma.RdByte(bfin,ix-3)
{
    ... informace o chybné LRC
}
```

2. adresa a kód funkce

```
adr_r=Ma.RdByte(bfin,1);
kod_r=Ma.RdByte(bfin,3);
```

3. reakce na odpověď

```
switch(kod_r){
    case f1:
        .
        break;
    case f2:
        .
        break;
    default: if(kod_r>=0x80)
    {
        er= Ma.RdByte(bfin,5);
        switch(er){
            informace o chybě stavu
        }
    } break;
}
stav=Tstav.stklid;
```

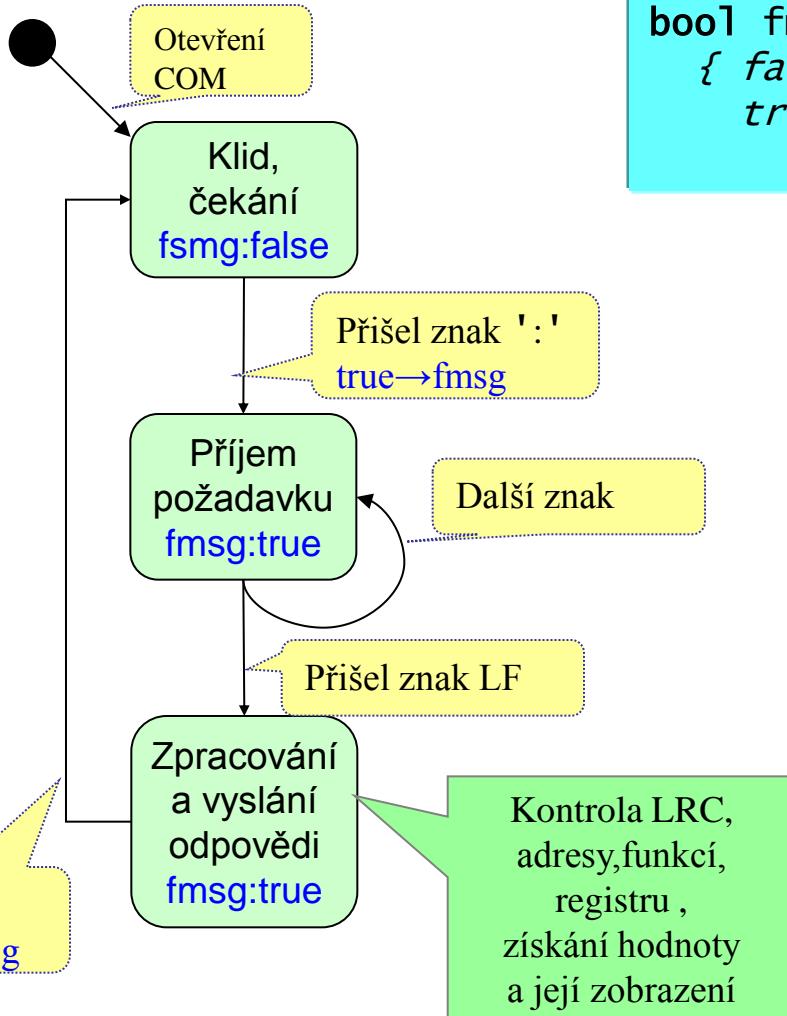
4. informace o chybě Slavu

5. klidový stav

úloha	str.
MA1	36
MA2	53
MA3	70
MA4	88
MA5	105
MA6	122



Slave – zjednodušený stavový diagram



```

bool fsmg;
{ false - čekání na požadavek
  true - příjem,zpracování požadavku
  a vyslaní odpovědi }

```



Slave – příjem požadavku

DataReceived

začátek zprávy

konec zprávy

```
while(comPort.BytesToRead > 0)
{
    byte b=(byte)comPort.ReadByte();
    if(b==(byte)':')
    {
        ix=0;
        fmsg=true;
    }
    else if(fmsg) ix++;
    bfin[ix]=b;
}
```

```
if(b==(byte)'\n' && fmsg)
{
    //zpracování požadavku
    //příprava a odeslání odpovědi
    .
    fmsg=false;
}
```

Klid,
čekání
fmsg:false

Přišel znak ':'
true→fmsg

Příjem
požadavku
fmsg:true

Další znak

Přišel znak LF



Slave – zpracování požadavku

1. LRC

```
if(Ma.Lrc(bfin,ix-4) != Ma.RdByte(bfin,ix-3)) {
    .. informace o chybné LRC
}
else {
```

2. adresa

```
adr_r = Ma.RdByte(bfin,1);
if(adr == ADR_S) {
```

3. kód funkce

```
kod_r = Ma.RdByte(bfin,3);
```

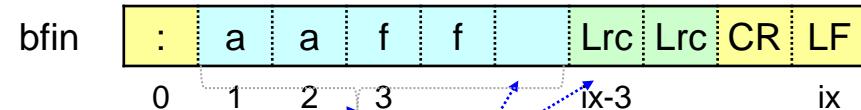
adresy objektů
a hodnoty

.. zpracování požadavku podle funkce a příprava odpovědi
`reg = Ma.RdWord(bfin,5);`
`val = Ma.RdWord(bfin,9);`

kontrola požadavku
- funkce
- adresa objektu
- hodnoty

er=0;

Požadovaná funkce není ve Slavu implementována: **er=1;**
Požadovaná adresa objektu ve Slavu mimo rozsah: **er=2;**
Zapisovaná data do objektu ve Slavu mimo rozsah: **er=3;**





Slave – vyslání odpovědi

if(er==0)

```
n = Ma.AnSRD(ADR_S,kod_r,pocet,vals,bfout);
```

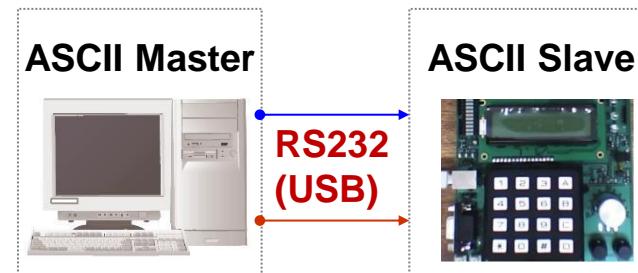
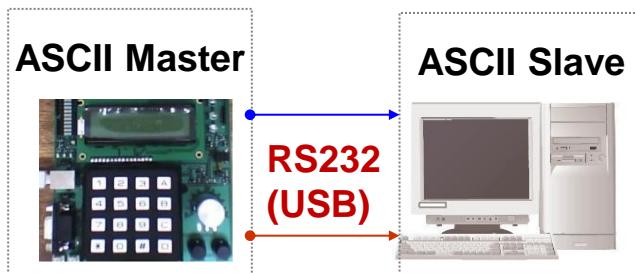
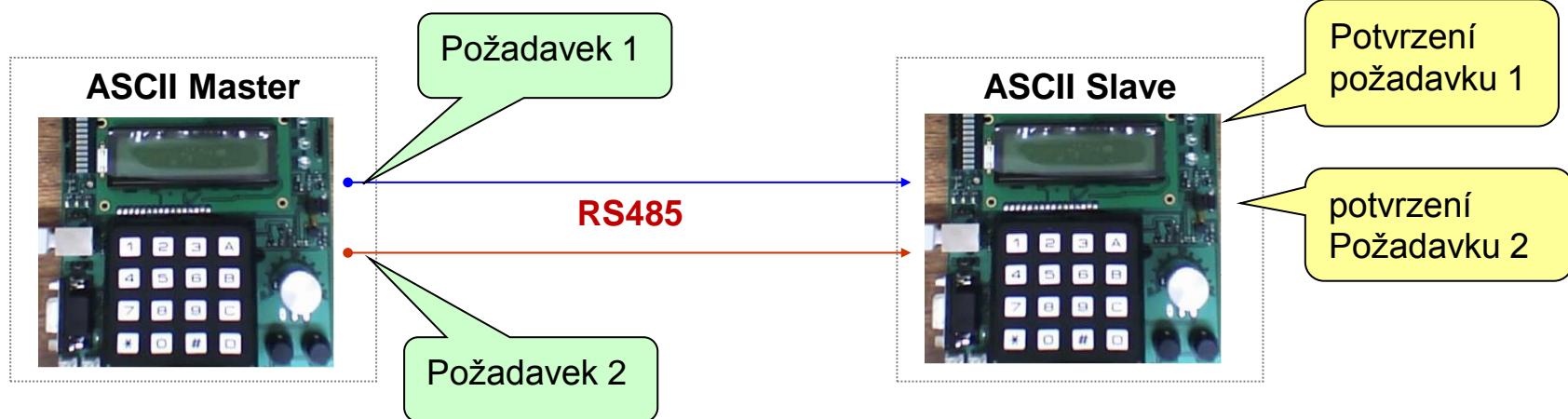
```
n = Ma.AnSwr(ADR_S,kod_r,reg,val,bfout);
```

```
if(er > 0) n=Ma.AnErr(adr_r,(byte)(kod_r|0x80),er,bfout);
```

```
n = Ma.WrByte(Mb.Lrc(bfout,n-1),bfout,n);
n = Ma.WrEoT(bfout[n]);
comPort.write(bfout,0,n);
```

úloha	str.
MA1	36
MA2	53
MA3	70
MA4	88
MA5	105
MA6	122

mikropočítač – mikropočítač (PC – mikropočítač)





Podpora pro mikropočítač prototypy fukcí **Modbus.H** – zdrojový kód **Modbus.C**

```
byte AHHex(byte c);
byte HexAsc(byte b);

byte WrWord(word val,byte *bf);
word RdWord(byte *bf);
byte MbRdByte(byte *bf);
word MbRdWord(byte *bf);
byte MbWrByte(byte b,byte *bf);
byte MbWrWord(word w,byte *bf);

byte MbRd(byte adr,byte fce,word reg,word val,byte *bf);
byte MbWrOne(byte adr,byte fce,word reg,word val,byte *bf);
byte MbWr(byte adr,byte fce,word reg,word nbr,byte *vals,byte *bf);

byte MbAnsWr(byte adr,byte fce,word reg,word val,byte *bf);
byte MbAnsRd(byte adr, byte fce, byte bytes, byte *vals,byte *bf);
byte MbAnsErr(byte adr,byte fce,byte er,byte *bf);

byte MbLrc(byte *bf,byte len);
byte MbWrEoT(byte *bf);
```

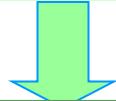


Užité funkce v aplikaci ze souboru Modbus.C

aplikáční	pomocné
MbWrOne	MbRdByte
MbRd	MbWrByte
MbAnsWr	MbLrc
MbAnsRd	MbWrEoT
MbAnsErr	MbRdWord
Poznámka: v hlavním programu	#include "Modbus.H"



: , adr. slavu, kód funkce, adr. registrů a bitů, hodnoty, LRC, CRLF



bfout

→ RS232

bfin

← RS232

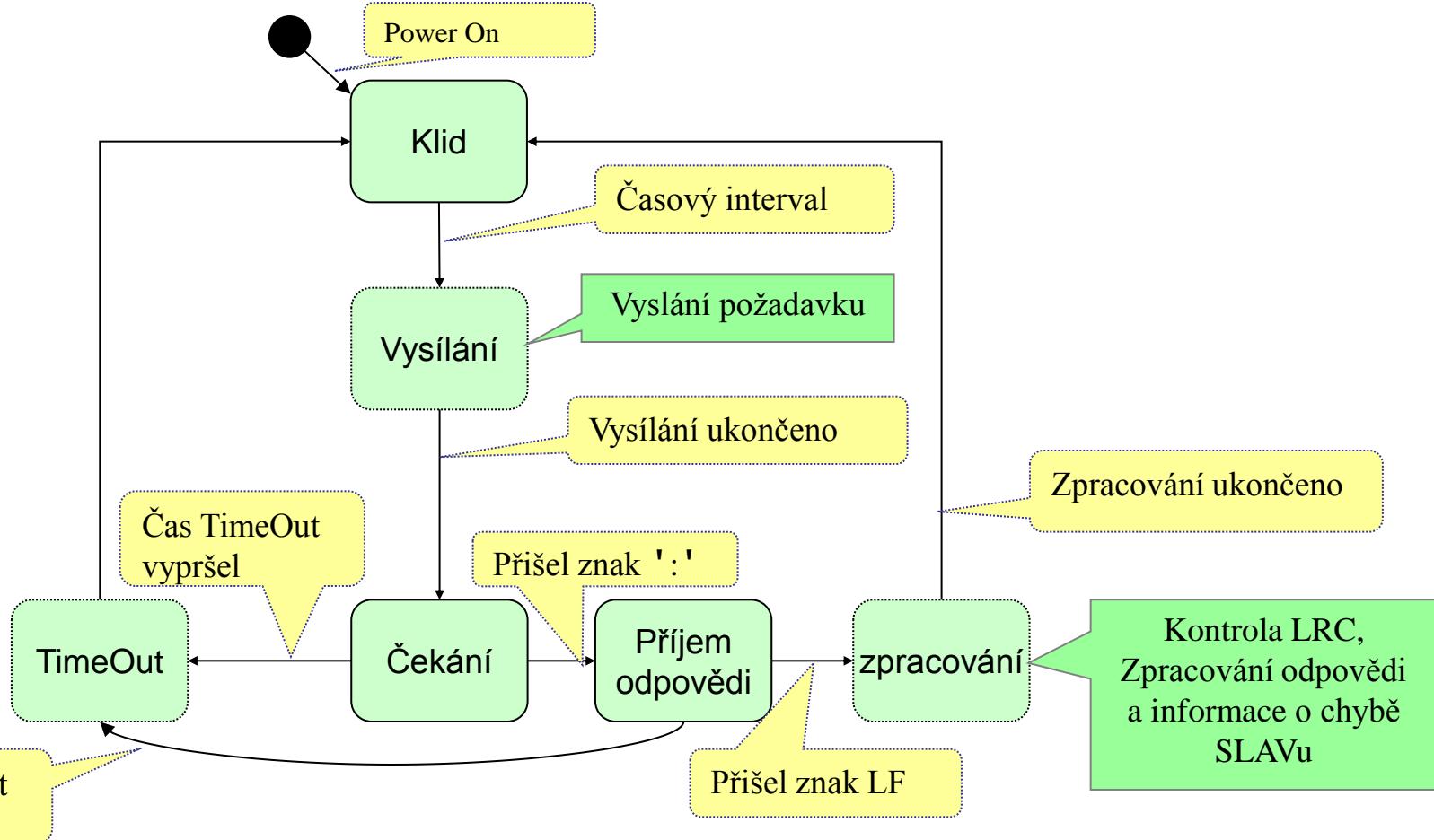
```
xbyte bfin[256],bfout[256];
```

bfout[0]	:
bfout[1],bfout[2]	adresa slavu
bfout[3],bfout[4]	kód funkce
.	
.	

funkce pro vyslání zprávy:
-bf: pointer na pole znaků
-len: počet bytů k vyslání

```
void SendBuf(byte *bf,byte len)
{
    while(len--)
    {
        SBUF=*bf++ | 0x80;
        while(!TI);
        TI=0;
    }
}
```

Master – zjednodušený stavový diagram



```
enum {stKlid, stCekani, stPrijem} stav;
```



Master – vyslání požadavku

střídavě každých cca 210 ms vysílá rámec s funcí **f1** a **f2**

časovač T0



```
if(++cnt_ticks>=N_TICKS && stav==stKlid)
{
    cnt_ticks=0;
    DIR485=1; /* na vysílání */
    prep=!prep;
    if(prep)itx=MbWrOne(ADR_S,...);
    else itx=MbRd(ADR_S,...);
    itx+=MbWrByte(MbLrc(bfout+1,itx-1),bfout+itx);
    itx+=MbWrEoT(bfout+itx);
    SendBuf(bfout,itx);
    stav=stCekani;
    DIR485=0; /* zpět na příjem */
}
```

```
#define N_TICKS 7
#define TIMEOUT 17
bit prep;
```

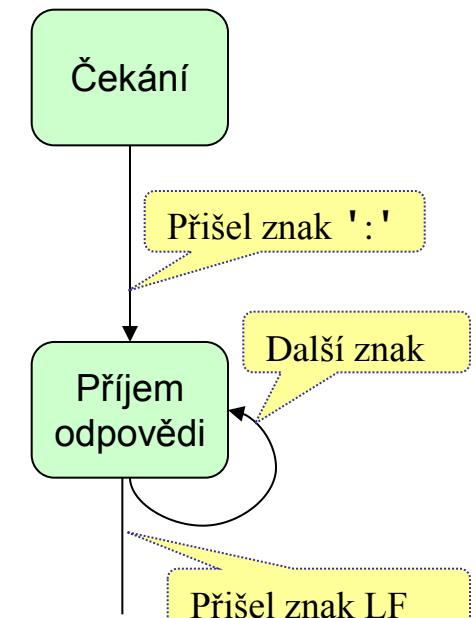
```
itx=MbRd(ADR_S,...)
itx=MbWrOne(ADR_S,...)
```

TimeOut

```
if(cnt_ticks>=TIMEOUT)
{
    cnt_ticks=0;
    LED_R=!LED_R; // signalizace TimeOutu
    stav=stKlid;
}
```

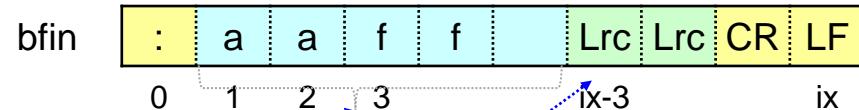
Master – příjem odpovědi

```
if (RI)
{
    byteIn=SBUF&0x7F;
    RI=0;
    if(stav==stCekani && byteIn==':')
    {
        ix=0;
        stav=stPrijem;
        bfin[0]=byteIn;
    }
    else if(stav==stPrijem)
    {
        if(byteIn==':')ix=0;
        else ix++;
        bfin[ix]=byteIn;
        if(byteIn=='\n')
        {
            . // zpracování odpovědi
            .
            stav=stKlid;
        }
    }
}
```





Master – zpracování odpovědi



1. LRC

```
if(MbLrc(bfin+1, ix-4)==(lrc=MbRdByte(bfin+ix-3)))
{
```

2. adresa
a kód funkce

```
adr_r=MbRdByte(bfin+1);
kod_r=MbRdByte(bfin+3);
```

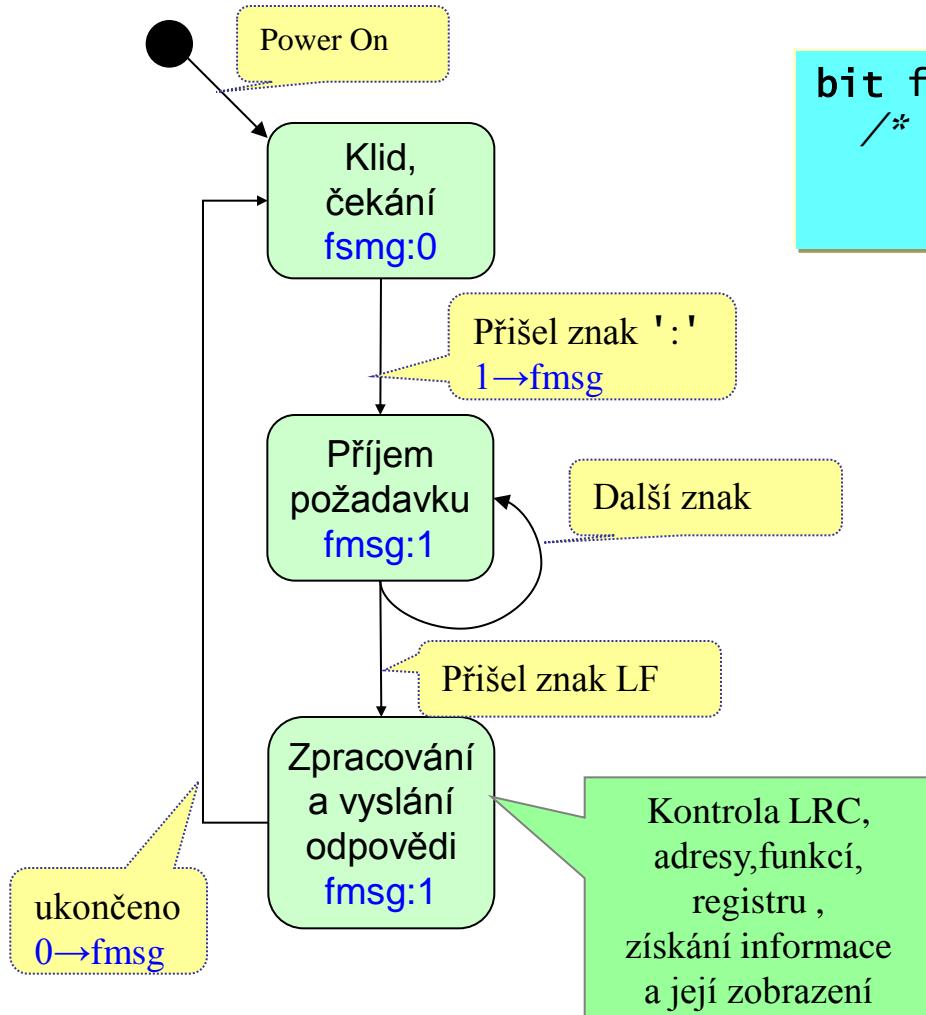
3. reakce
na odpověď

```
switch(kod_r){
    case f1:
        .
        break;
    case f2:
        .
        break;
    default: if(kod_r>=0x80)
    {
        informace o chybě stavu
    }
}
stav=stKlid;
```

4. informace
o chybě Slavu
(není nutné)

úloha	str.
MA1	36
MA2	53
MA3	70
MA4	88
MA5	105
MA6	122

Slave – zjednodušený stavový diagram



```

bit fmsg;
/* 0 - čekání na požadavek
   1 - příjem, zpracování požadavku
   a vyslaní odpovědi */

```

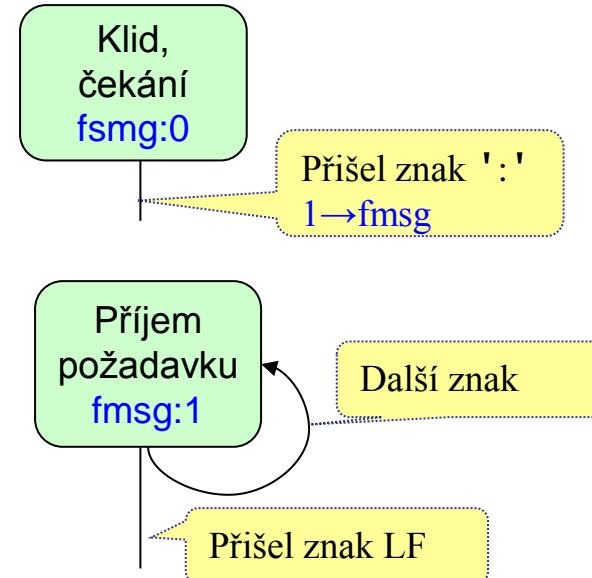


Slave – příjem požadavku

```

if(RI)
{
    if((byteIn=SBUF&0x7F)==' : ')
    {
        ix=0;
        fmsg=1;
    }
    else if(fmsg) ix++;
    RI=0;
    bfin[ix]=byteIn;
    if(fmsg && byteIn=='\n')
    {
        // zpracování požadavku
        // příprava a odeslání odpovědi
        .
        .
        .
        fmsg=0;
    }
}

```



Slave – zpracování požadavku

1. LRC
a adresa

```
if((MbLrc(bfin+1,ix-4)==(lrc=MbRdByte(bfin+ix-3))) &&
(MbRdByte(bfin+1)==ADR_S))
{
    .. LRC a adresa OK
}
```

2. kód funkce

```
kod_r=MbRdByte(bfin+3)
```

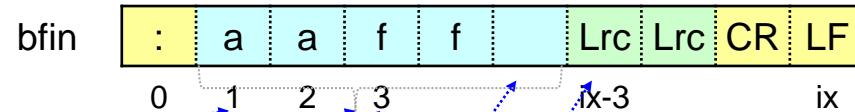
adresy objektů
a hodnoty

```
.. zpracování požadavku podle funkce a příprava odpovědi
reg = MbRdWord(bfin+5);
val = MbRdWord(bfin+9);
.
```

kontrola požadavku
- funkce
- adresa objektu
- hodnoty

er=0;

Požadovaná funkce není ve Slavu implementována: **er=1;**
Požadovaná adresa objektu ve Slavu mimo rozsah: **er=2;**
Zapisovaná data do objektu ve Slavu mimo rozsah: **er=3;**





Slave – vyslání odpovědi

if(er==0)

```
itx = MbAnsRD(ADR_S,kod_r,pocet,vals,bfout);  
itx = MbAnsWr(ADR_S,kod_r,reg,val,bfout);
```

```
if(er) itx = MbAnsErr(ADR_S,kod_r|0x80,er,bfout);
```

```
itx += MbWrByte(MbLrc(bfout+1,itx-1),bfout+itx);  
itx += MbWrEoT(bfout+itx);  
SendBuf(bfout,itx);
```

úloha	str.
MA1	36
MA2	53
MA3	70
MA4	88
MA5	105
MA6	122



Řídicí počítačové systémy

Úloha pro samostatná cvičení - MA1

Implementace protokolu MODBUS ASCII na PC a mikropočítacích řady '51 pro uzly Master (Klient) i Slave (Server).

Požadované implementované funkce:

- zápis jediného vnitřního registru (Holding) do uzlu Slave,
- čtení bitového stavu (Coil) z uzlu Slave.

Rozhraní: RS232, standardní rámec 7,N,2

1. část: propojení PC – PC (C# MSVS)

2. část: propojení PC – mikropočítač

Rozhraní: RS485, standardní rámec 7,N,2

3. část: propojení mikropočítač – mikropočítač

Funkce pro podporu aplikace protokolu MODBUS:

v souboru Modbus.dll a Modbus.cs pro PC (C#),

v souboru Modbus.H a Modbus.C pro mikropočítač



MASTER (klient)

V pravidelných časových intervalech generuje 16 bitovou hodnotu a předává požadavek na zápis do uzlu SLAVE

V pravidelných časových intervalech generuje požadavek na čtení bitové informace z uzlu SLAVE a zobrazuje ji

kód fukce: 06
+ data

potvrzení

kód fukce: 01

data

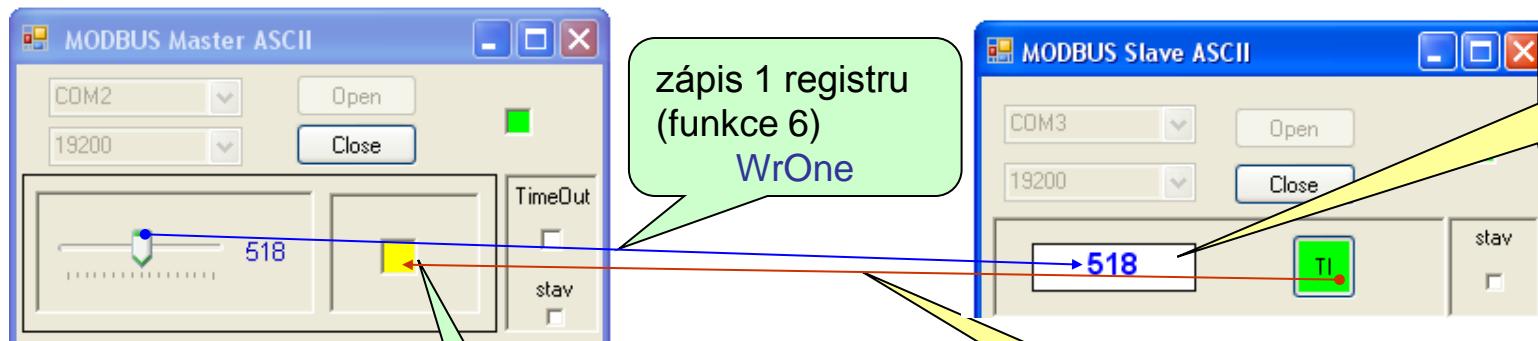
SLAVE (server)

Po příjmu hodnotu zobrazí a odešle potvrzovací odpověď

Po příjmu požadavku hodnotu bitu odešle



1.část : PC-PC (varianta C#)



čtení bitu
(funkce 1)
Rd

zápis 1 registru
(funkce 6)
WrOne

stav bitu
AnsRd

potvrzení
požadavku
AnsWr



Master – implementace na PC (klient)

Konfigurace:

Realizuje funkce (požadavky na server)

- požadavek na zápis jediného vnitřního registru (hodnota 0 až 1023) – funkční kód 6

metoda WrOne třídy Modbus ASCII s kódem funkce 6 (FCE_WREG)

- požadavek na čtení bitové hodnoty – funkční kód 1

metoda Rd třídy Modbus ASCII s kodem funkce 1 (FCE_RBIT)

Požadavky odesílat střídavě v pravidelných časových intervalech 200 ms,

jen když je sériový kanál otevřen a Master je ve stavu **klidu**

realizace časovačem intervalu

Implementovat generování čekacího TimeOut intervalu 500 ms na odpověď od Slave,

po vypršení TimeOutu vyčkat 500 ms a vátit se do stavu **klidu**

Zjednodušený příjem odpovědi

příchozí adresu Slave není nutno testovat, pouze správnost LRC

zpracovat jen odpověď na požadavek čtení bitu (FCE_RBIT)

informovat o chybové odpovědi od Slave

Master – vyslání požadavku

Časovač Sample

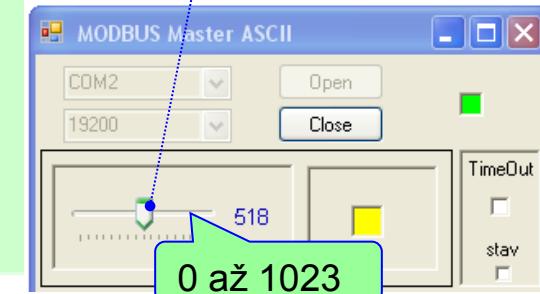
střídavě každých cca 200 ms vysílá rámec s funcí **1** (čtení bitu) a **6** (zápis registru)



ModbusASCII Ma;
bool prep;
Tstav stav;

Tick
Sample

```
if (comPort.IsOpen && stav == Tstav.stklid)
{
    stav=Tstav.stVysilani;
    prep = !prep;
    if (prep) n=Ma.wrOne(ADR_S,FCE_WREG,REG_WR,pot,bfout)
    else n=Ma.Rd(ADR_S,FCE_RBIT,BIT_RD,1,bfout);
    n=Ma.WrByte(Ma.Lrc(bfout,n-1),bfout,n);
    n=Ma.WrEoT(bfout,n);
    comPort.Write(bfout,0,n);
    TimerOut.Interval=500;
    TimerOut.Enabled=true;
    stav=Tstav.stCekani;
}
```





Master – zpracování odpovědi

1. LRC

```
if(Ma.Lrc(bfin,ix-4)!=Ma.RdByte(bfin,ix-3)
{
    .. informace o chybné LRC
}
```

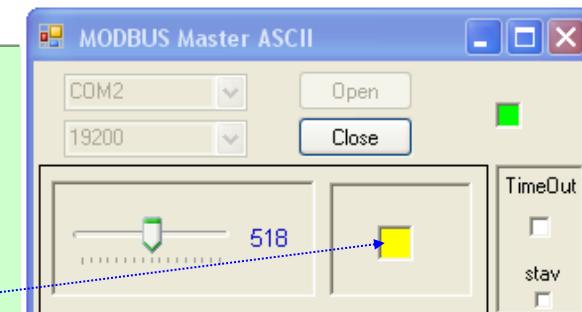
2. adresa
a kód funkce

```
adr_r=Ma.RdByte(bfin,1);
kod_r=Ma.RdByte(bfin,3);
```

3. reakce
na odpověď

```
if(kod_r== FCE_RBIT)
{
    pocet=Ma.RdByte(bfin,5);
    val=Ma.RdByte(bfin,7);
    if (adr_r==ADR_S && pocet==1)
        if (val & 1 ==1) { žlutá }
        else { bílá }
    }
else if (kod_r>=0x80)
{
    er= Ma.RdByte(bfin,5);
    switch(er) {
        informace o chybě slavu
    }
}
stav=Tstav.stklid;
```

4. informace
o chybě Slavu





Slave – implementace na PC (server)

Konfigurace:

Přijímá požadavky od klienta a vrací odpovědi

- požadavek na zápis jediného vnitřního registru (hodnota 0 až 1023) – funkční kód 6, hodnotu zobrazí a vrací potvrzení o přijetí požadavku

metoda `AnsWr` třídy `Modbus ASCII` s kódem přijaté funkce

- požadavek na čtení bitové hodnoty – funkční kód 1 a vrací stav požadovaného bitu

metoda `AnsRd` třídy `Modbus ASCII` s kódem přijaté funkce

Kontrolovat přijatý požadavek a vracet chybovou odpověď v případě neimplementované funkce, neexistující adresy registru nebo bitu a hodnoty mimo rozsah

metoda `AnsErr` třídy `Modbus ASCII` s upraveným kódem funkce a typem chyby

Skupinové vysílání ignorovat .



Slave – zpracování požadavku, kontrola položek, příprava odpovědi

1. LRC

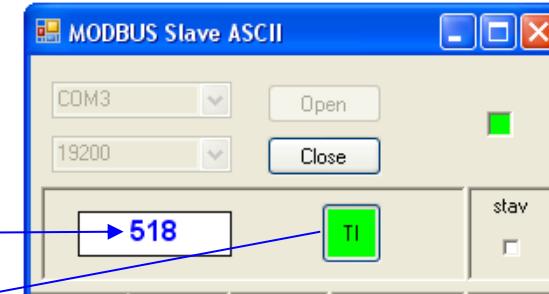
```
if(Ma.Lrc(bfin, ix-4) != Ma.RdByte(bfin, ix-3)
{
    .. možná informace o chybné LRC
}
else {
```

2. adresa

```
adr_r=Ma.RdByte(bfin,1);
if(adr_r == ADR_S)
{
```

3. kód funkce

```
kod_r=Ma.RdByte(bfin,3);
er=0;
switch(kod_r) {
    case FCE_WREG:
        .
        .
    case FCE_RBIT:
        .
        .
default er=1;
}
```





Slave – zpracování požadavku, kontrola položek, příprava odpovědi

FCE_WREG:

```
reg=Ma.Rdword(bfin,5);
val=Ma.Rdword(bfin,9);
if(reg!=REG_WR) er=2;
else if(val>1023) er=3;
else .. zobrazení hodnoty
if(er==0) n= Ma.Answr(ADR_S,kod_r,reg,val,bfout);
```



FCE_RBIT:

```
reg=Ma.Rdword(bfin,5);
pocet=Ma.RdWord(bfin,9);
if(reg!=BIT_RD || pocet!=1) er=2;
else {
    tlačítko -> vals
    n= Ma.AnswRD(ADR_S,kod_r,1,vals,bfout);
}
```

byte []vals = byte[1];

4. chyba

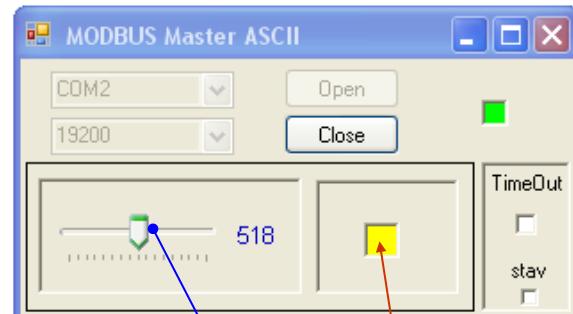
if(er>0) n=Ma.AnswErr(adr_r,(byte)(kod_r|0x80),er,bfout);

5. odeslání
odpovědi

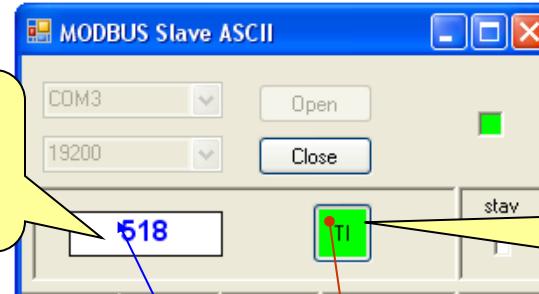
```
n=Ma.WrByte(Ma.Lrc(bfout,n-1),bfout,n);
n=Ma.WrEoT(bfout,n);
comPort.write(bfout, 0, n);
```



2.část : PC – mikropočítač



potvrzení
požadavku
AnsWr

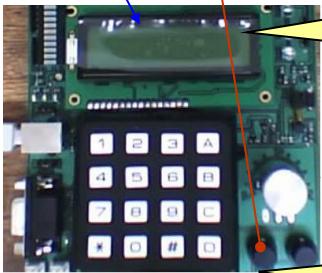


stav bitu
AnsRd

čtení bitu
(funkce 1)
Rd

zápis 1 registru
(funkce 6)
WrOne

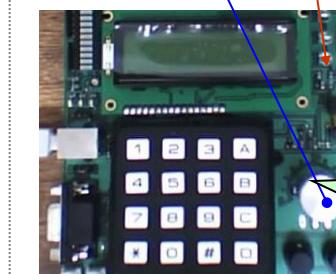
ASCII Slave



potvrzení
požadavku
MbAnsWr

stav tlačítka
MbAnsRd

ASCII Master



čtení bitu
(funkce 1)
MbRd

zápis hodnoty
potenciometru
(funkce 6)
MbWrOne



Master – implementace na mikropočítači (klient)

Konfigurace:

Realizuje funkce (požadavky na server)

- požadavek na zápis jediného vnitřního registru (hodnota 0 až 1023) – funkční kód 6
[aplikativní funkce MbWrOne s kódem funkce 6 \(FCE_WREG\)](#)
- požadavek na čtení bitové hodnoty – funkční kód 1
[aplikativní funkce MbRd s kodem funkce 1 \(FCE_RBIT\)](#)

Požadavky odesílat střídavě v pravidelných časových intervalech cca 200 ms,
jen když je sériový kanál otevřen a Master je ve stavu **klidu**

realizace časovačem T0 se základními tiky 30 ms ($30 * 7 = 210$)

Implementovat generování čekacího TimeOut intervalu 500 ms na odpověď od Slave
($30 * 17 = 510$)

Zjednodušený příjem odpovědi

příchozí adresu Slave není nutno testovat, pouze správnost LRC

zpracovat jen odpověď na požadavek čtení bitu (FCE_RBIT)

informace o chybě Slave: jen omezeně, nebo vůbec

Master – vyslání požadavku

střídavě každých cca 210 ms vysílá rámec s funcí **1** (čtení bitu) a **6** (zápis registru)



časovač T0

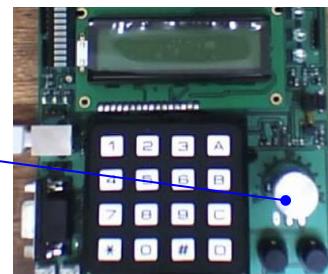
```
#define N_TICKS 7
#define TIMEOUT 17
bit prep;

if(++cnt_ticks>=N_TICKS && stav==stKlid){
    cnt_ticks=0;
    DIR485=1;      /* na vysílání - pro RS485*/
    prep=!prep;
    if(prep){ val= ... ;
        itx=MbWrOne(ADR_S,FCE_WREG,REG_WR,val,bfout);}
    else itx=MbRd(ADR_S,FCE_RBIT,BIT_RD,1,bfout);
    itx+=MbWrByte(MbLrc(bfout+1,itx-1),bfout+itx);
    itx+=MbWrEoT(bfout+itx);
    SendBuf(bfout,itx);
    stav=stCekani;
    DIR485=0;      /* zpět na příjem - pro RS485 */
}
```

```
#define N_TICKS 7
#define TIMEOUT 17
bit prep;
```

TimeOut

```
if(cnt_ticks>=TIMEOUT){
    cnt_ticks=0;
    LED_R=!LED_R; // signalizace TimeOutu
    stav=stKlid;
}
```





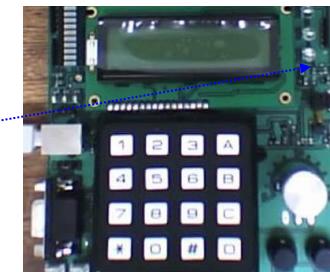
Master – zpracování odpovědi

1. LRC

```
if(MbLrc(bfin+1,ix-4)==(lrc=MbRdByte(bfin+ix-3)))
{
```

2. kód funkce
a reakce
na odpověď

```
if( (kod_r=MbRdByte(bfin+3))==FCE_RBIT)
{
    pocet=MbRdByte(bfin+5);
    val=MbRdByte(bfin+7);
    if(val & 1) .. ; // LED svítí
    else .. ; // LED nesvítí
}
```





Slave – implementace na mikropočítači (server)

Konfigurace:

Přijímá požadavky od klienta a vrací odpovědi

- požadavek na zápis jediného vnitřního registru (hodnota 0 až 1023) – funkční kód 6, hodnotu zobrazí na LCD a vrací potvrzení o přijetí požadavku

aplikáční funkce MbAnsWr s kódem přijaté funkce

- požadavek na čtení bitové hodnoty – funkční kód 1 a vrací stav tlačítka

aplikáční funkce MbAnsRd s kódem přijaté funkce

Kontrolovat přijatý požadavek a vracet chybovou odpověď v případě neimplementované funkce, neexistující adresy registru nebo bitu a hodnoty mimo rozsah

aplikáční funkce MbAnsErr s upraveným kódem funkce a typem chyby

Skupinové vysílání ignorovat .



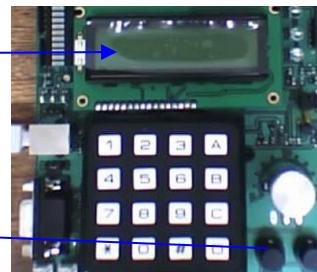
Slave – zpracování požadavku, kontrola položek, příprava odpovědi

1. LRC
a adresa

```
if((MbLrc(bfin+1,ix-4)==(lrc=MbRdByte(bfin+ix-3)))  
    && (MbRdByte(bfin+1)==ADR_S))  
{
```

2. kód
funkce

```
kod_r=MbRdByte(bfin+3);  
er=0;  
switch(kod_r)  
{  
    case FCE_WREG:  
        .  
        .  
    case FCE_RBIT:  
        .  
        .  
    default: er=1;  
}
```

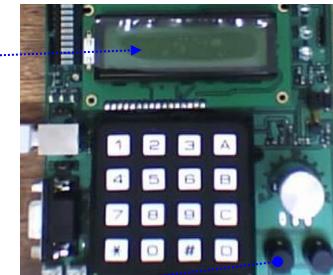




Slave – zpracování požadavku, kontrola položek, příprava odpovědi

FCE_WREG:

```
if((reg=MbRdWord(bfin+5))!=REG_WR) er=2;
else if((val=MbRdWord(bfin+9))>1023) er=3;
else printf(...);
if(er==0)itx=MbAnsWr(ADR_S,kod_r,reg,val,bfout);
break;
```



FCE_RBIT:

```
if((reg=MbRdWord(bfin+5))==BIT_RD&&(pocet=MbRdWord(bfin+9))==1)
{
    bity[0]= ...;
    itx=MbAnsRd(ADR_S,kod_r,1,bity,bfout);
}
else er=2;
break;
```

byte bity[1];

4. chyba

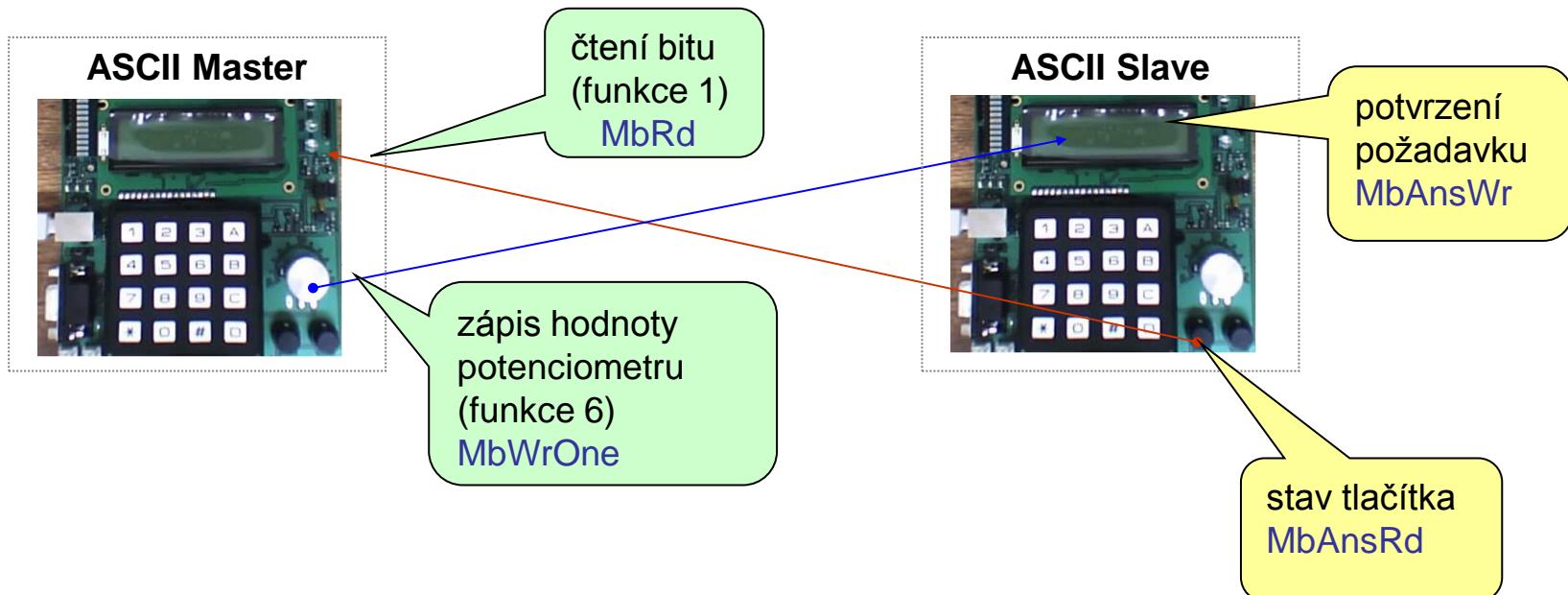
```
if(er) itx=MbAnsErr(adr_r,kod_r|0x80,er,bfout);
```

odeslání
odpovědi

```
DIR485=1; /* na vysílání */
itx+=MbWrByte(MbLrc(bfout+1,itx-1),bfout+itx);
itx+=MbWrEoT(bfout+itx);
SendBuf(bfout,itx);
DIR485=0; /* zpět na příjem */
```



3.část : mikropočítač – mikropočítač



Řídicí počítačové systémy

Úloha pro samostatná cvičení - MA2

Implementace protokolu MODBUS ASCII na PC a mikropočítačích řady '51 pro uzly Master (Klient) i Slave (Server).

Požadované implementované funkce:

- zápis jediného bitového stavu (Coil) do uzlu Slave,
- čtení 16 bitového vnitřního registru (Holding) z uzlu Slave.

Rozhraní: RS232, standardní rámec 7,N,2

1. část: propojení PC – PC (C# MSVS)

2. část: propojení PC – mikropočítač

Rozhraní: RS485, standardní rámec 7,N,2

3. část: propojení mikropočítač – mikropočítač

Funkce pro podporu aplikace protokolu MODBUS:

v souboru Modbus.dll a Modbus.cs pro PC (C#),

v souboru Modbus.H a Modbus.C pro mikropočítač



MASTER (klient)

V pravidelných časových intervalech generuje 1bitovou informaci a předává požadavek na zápis do uzlu SLAVE

V pravidelných časových intervalech generuje požadavek na čtení 16bitové hodnoty z uzlu SLAVE a zobrazuje ji

SLAVE (server)

Po příjmu informaci zobrazí a odešle potvrzovací odpověď

Po příjmu požadavku hodnotu odešle

kód fukce: 05

+ data

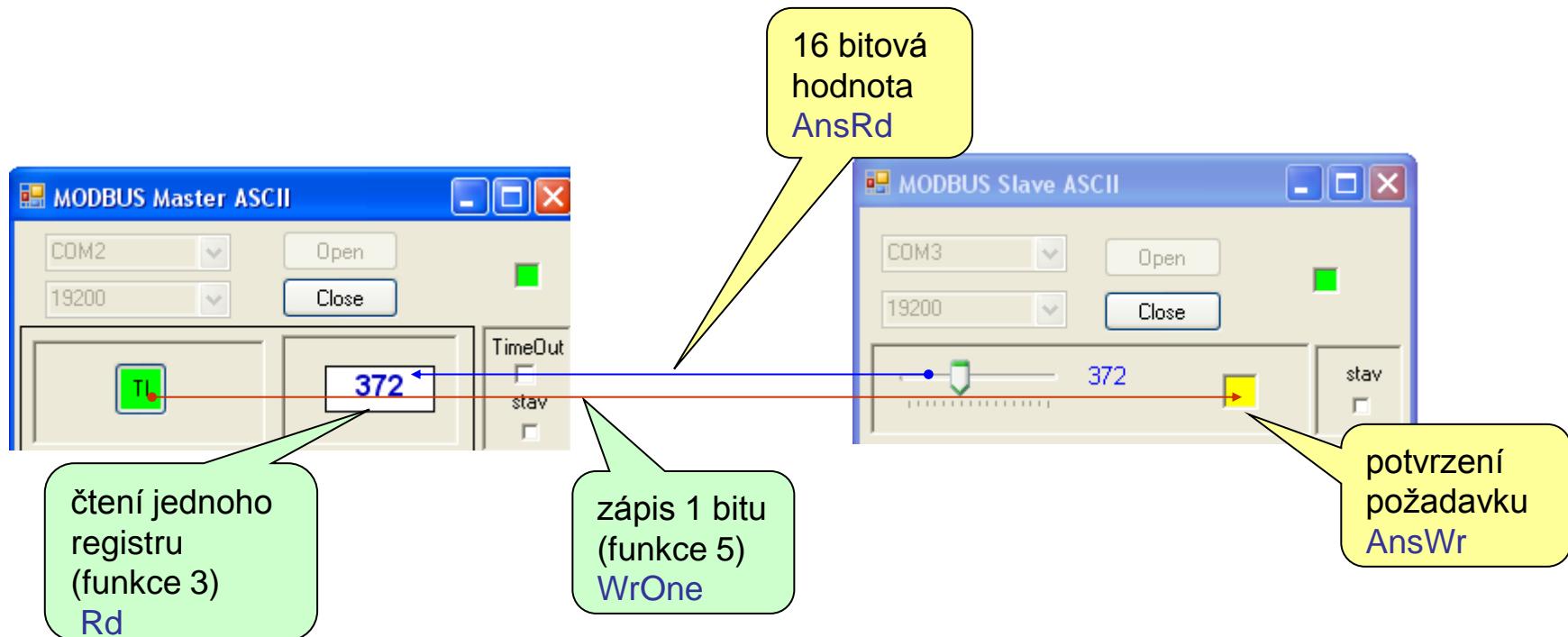
potvrzení

kód fukce: 03

data



1.část : PC-PC (varianta C#)





Master – implementace na PC (klient)

Konfigurace:

Realizuje funkce (požadavky na server)

- požadavek na zápis jediného bitového stavu – funkční kód 5

metoda WrOne třídy Modbus ASCII s kódem funkce 5 (FCE_WBIT)

- požadavek na čtení 16 bitové hodnoty vnitřního registru – funkční kód 3

metoda Rd třídy Modbus ASCII s kodem funkce 3 (FCE_RREG)

Požadavky odesílat střídavě v pravidelných časových intervalech 200 ms,

jen když je sériový kanál otevřen a Master je ve stavu **klidu**

realizace časovačem intervalu

Implementovat generování čekacího TimeOut intervalu 500 ms na odpověď od Slave,

po vypršení TimeOutu vyčkat 500 ms a vátit se do stavu **klidu**

Zjednodušený příjem odpovědi

příchozí adresu Slave není nutno testovat, pouze správnost LRC

zpracovat jen odpověď na požadavek čtení registru (FCE_RREG)

informovat o chybové odpovědi od Slave



Master – vyslání požadavku

Časovač Sample

střídavě každých cca 200 ms vysílá rámec s funcí 3 (čtení registru) a 5 (zápis bitu)

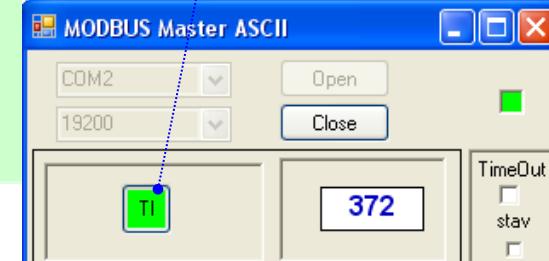


ModbusASCII Ma;
bool prep;
Tstav stav;

Tick
Sample

```
if (comPort.IsOpen && stav == Tstav.stklid)
{
    stav=Tstav.stVysilani;
    prep = !prep;
    if (prep) n=Ma.wrOne(ADR_S,FCE_WBIT,BIT_WR,val,bfout)
    else n=Ma.Rd(ADR_S,FCE_RREG,REG_RD,1,bfout);
    n=Ma.WrByte(Ma.Lrc(bfout,n-1),bfout,n);
    n=Ma.WrEoT(bfout,n);
    comPort.Write(bfout,0,n);
    TimerOut.Interval=500;
    TimerOut.Enabled=true;
    stav=Tstav.stCekani;
}
```

stisk: 0xFF00
jinak: 0



Master – zpracování odpovědi

1. LRC

```
if(Ma.Lrc(bfin, ix-4) != Ma.RdByte(bfin, ix-3)
{
    .. informace o chybné LRC
}
```

2. adresa a kód funkce

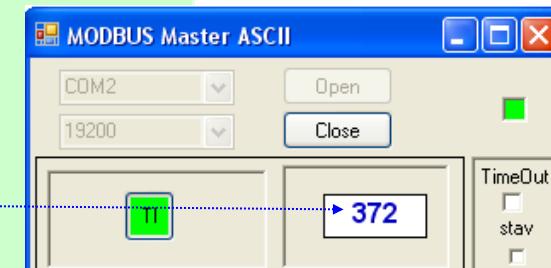
```
adr_r=Ma.RdByte(bfin,1);
kod_r=Ma.RdByte(bfin,3);
```

3. reakce na odpověď

```
if(kod_r== FCE_RREG)
{
    pocet=Ma.RdByte(bfin,5);
    if (adr_r==ADR_S && pocet==1)
    {
        val=Ma.Rdword(bfin,7);
        .
        .
    }
}
```

4. informace o chybě Slavu

```
else if (kod_r>=0x80)
{
    er= Ma.RdByte(bfin,5);
    switch(er) {
        .
        .
    }
    stav=Tstav.stklid;
```





Slave – implementace na PC (server)

Konfigurace:

Přijímá požadavky od klienta a vrací odpovědi

- požadavek na zápis jediného bitového stavu – funkční kód 5, stav indikuje a vrací potvrzení o přijetí požadavku

metoda `AnsWr` třídy `Modbus ASCII` s kódem přijaté funkce

- požadavek na čtení 16 bitové hodnoty – funkční kód 3 a vrací požadovanou hodnotu

metoda `AnsRd` třídy `Modbus ASCII` s kódem přijaté funkce

Kontrolovat přijatý požadavek a vracet chybovou odpověď v případě neimplementované funkce, neexistující adresy registru nebo bitu a hodnoty mimo rozsah

metoda `AnsErr` třídy `Modbus ASCII` s upraveným kódem funkce a typem chyby

Skupinové vysílání ignorovat .



Slave – zpracování požadavku, kontrola položek, příprava odpovědi

1. LRC

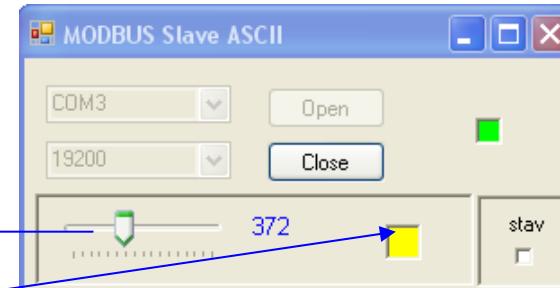
```
if(Ma.Lrc(bfin, ix-4) != Ma.RdByte(bfin, ix-3)
{
    .. možná informace o chybné LRC
}
else {
```

2. adresa

```
adr_r=Ma.RdByte(bfin,1);
if(adr_r == ADR_S)
{
```

3. kód funkce

```
kod_r=Ma.RdByte(bfin,3);
er=0;
switch(kod_r) {
    case FCE_RREG:
        .
        .
    case FCE_WBIT:
        .
        .
    default: er=1;
}
```



Slave – zpracování požadavku, kontrola položek, příprava odpovědi

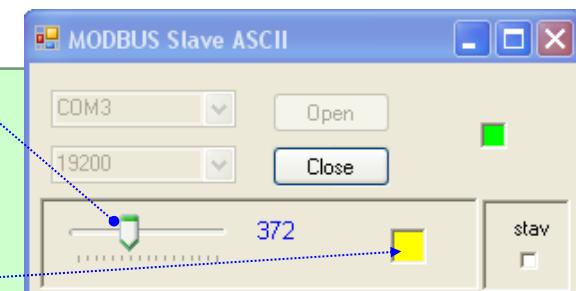
FCE_RREG:

```
reg=Ma.RdWord(bfin,5);
pocet=Ma.RdWord(bfin,9);
if(reg!=REG_RD || pocet!=1) er=2;
else .. MSB -> vals[0] a LSB -> vals[1]
if(er==0) n= Ma.AnswRD(ADR_S,kod_r,2,vals,bfout);
```

byte []vals = byte[2];

FCE_WBIT:

```
reg=Ma.RdWord(bfin,5);
val=Ma.RdWord(bfin,9);
if(reg!=BIT_WR) er=2;
else switch(val){
    case 0xFF00: .. žlutá ; break;
    case 0x0000: .. bílá ; break;
    default: err=3;
}
if(err==0) n= Ma.AnswWR(ADR_S,kod_r,reg,val,bfout);
```



4. chyba

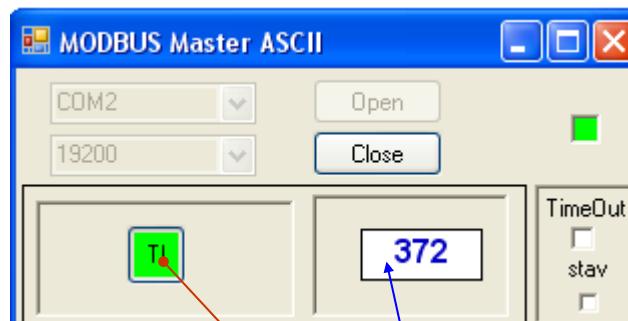
if(er>0) n=Ma.AnswErr(adr_r,(byte)(kod_r|0x80),er,bfout);

5. odeslání
odpovědi

```
n=Ma.WrByte(Ma.Lrc(bfout,n-1),bfout,n);
n=Ma.WrEoT(bfout,n);
comPort.Write(bfout, 0, n);
```



2.část : PC – mikropočítač



zápis 1 bitu
(funkce 5)
WrOne

16 bitová
hodnota
AnsRd



potvrzení
požadavku
AnsWr



čtení jednoho
registru
(funkce 3)
Rd

potvrzení
požadavku
MbAnsWr

hodnota
potenciometru
MbAnsRd



čtení jednoho
registru
(funkce 3)
MbRd

ASCII Master

zápis stavu
tlačítka
(funkce 5)
MbWrOne



Master – implementace na mikropočítači (klient)

Konfigurace:

Realizuje funkce (požadavky na server)

- požadavek na zápis jediného bitového stavu – funkční kód 5

[aplikativní funkce MbWrOne s kódem funkce 5 \(FCE_WBIT\)](#)

- požadavek čtení 16 bitové hodnoty vnitřního registru – funkční kód 3

[aplikativní funkce MbRd s kodem funkce 3 \(FCE_RREG\)](#)

Požadavky odesílat střídavě v pravidelných časových intervalech cca 200 ms,

jen když je sériový kanál otevřen a Master je ve stavu **klidu**

realizace časovačem T0 se základními tiky 30 ms ($30 * 7 = 210$)

Implementovat generování čekacího TimeOut intervalu 500 ms na odpověď od Slave
($30 * 17 = 510$)

Zjednodušený příjem odpovědi

příchozí adresu Slave není nutno testovat, pouze správnost LRC

zpracovat jen odpověď na požadavek čtení registru (FCE_RREG)

informace o chybě Slave: jen omezeně, nebo vůbec



Master – vyslání požadavku

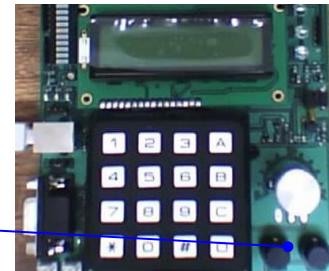
střídavě každých cca 210 ms vysílá rámec s funcí **3** (čtení bitu) a **5** (zápis registru)

3 5 3 5 3 5

časovač T0

```
if(++cnt_ticks>=N_TICKS && stav==stKlid)
{
    cnt_ticks=0;
    DIR485=1;      /* na vysílání - pro RS485*/
    prep=!prep;
    if(prep) {val= ... ;
        itx=MbWrOne(ADR_S,FCE_WBIT,BIT_WR,val,bfout);}
    else itx=MbRd(ADR_S,FCE_RREG,REG_RD,1,bfout);
    itx+=MbWrByte(MbLrc(bfout+1,itx-1),bfout+itx);
    itx+=MbWrEoT(bfout+itx);
    SendBuf(bfout,itx);
    stav=stCekani;
    DIR485=0;      /* zpět na příjem - pro RS485 */
}
```

```
#define N_TICKS 7
#define TIMEOUT 17
bit prep;
```



TimeOut

```
if(cnt_ticks>=TIMEOUT)
{
    cnt_ticks=0;
    LED_R=!LED_R; // signalizace TimeOutu
    stav=stKlid;
}
```

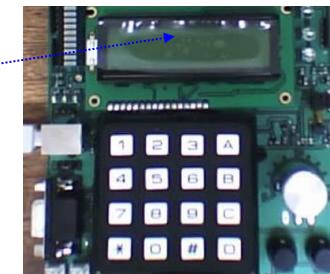
Master – zpracování odpovědi

1. LRC

```
if(MbLrc(bfin+1,ix-4)==(lrc=MbRdByte(bfin+ix-3)))
{
```

2. kód funkce
a reakce
na odpověď

```
if( (kod_r=MbRdByte(bfin+3))==FCE_RREG)
{
    pocet=MbRdByte(bfin+5);
    val=MbRdWord(bfin+7);
    printf(...);
}
```





Slave – implementace na mikropočítači (server)

Konfigurace:

Přijímá požadavky od klienta a vrací odpovědi

- požadavek na zápis jediného bitového stavu – funkční kód 5,
stav indikuje a vrací potvrzení o přijetí požadavku

aplikáční funkce MbAnsWr s kódem přijaté funkce

- požadavek na čtení 16 bitové hodnoty – funkční kód 3 a vrací požadovanou hodnotu
aplikáční funkce MbAnsRd s kódem přijaté funkce

Kontrolovat přijatý požadavek a vracet chybovou odpověď v případě neimplementované funkce, neexistující adresy registru nebo bitu a hodnoty mimo rozsah

aplikáční funkce MbAnsErr s upraveným kódem funkce a typem chyby

Skupinové vysílání ignorovat .



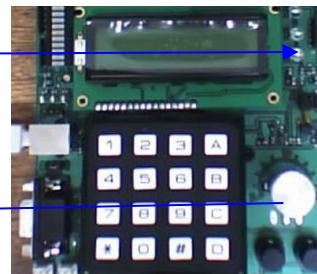
Slave – zpracování požadavku, kontrola položek, příprava odpovědi

1. LRC
a adresa

```
if((MbLrc(bfin+1,ix-4)==(lrc=MbRdByte(bfin+ix-3)))  
    && (MbRdByte(bfin+1)==ADR_S))  
{
```

2. kód
funkce

```
kod_r=MbRdByte(bfin+3);  
er=0;  
switch(kod_r)  
{  
    case FCE_WBIT:  
        .  
        .  
    case FCE_RREG:  
        .  
        .  
    default: er=1;  
}
```

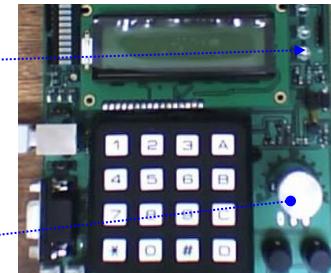




Slave – zpracování požadavku, kontrola položek, příprava odpovědi

FCE_WBIT:

```
if((reg=MbRdword(bfin+5))!=BIT_WR) er=2;
else if((val=MbRdword(bfin+9))!=0&&val!=0xFF00) er=3;
else LED_G ...;
if(er==0)itx=MbAnsWr(ADR_S,kod_r,reg,val,bfout);
break;
```



FCE_RREG:

```
if((reg=MbRdword(bfin+5))==REG_RD&&(pocet=MbRdword(bfin+9))==1){
    val = ... ;
    vals[0]= val>>8;
    vals[1]=val;
    itx=MbAnsRd(ADR_S,kod_r,1,vals,bfout);
}
else er=2;
break;
```

byte vals[2];

4. chyba

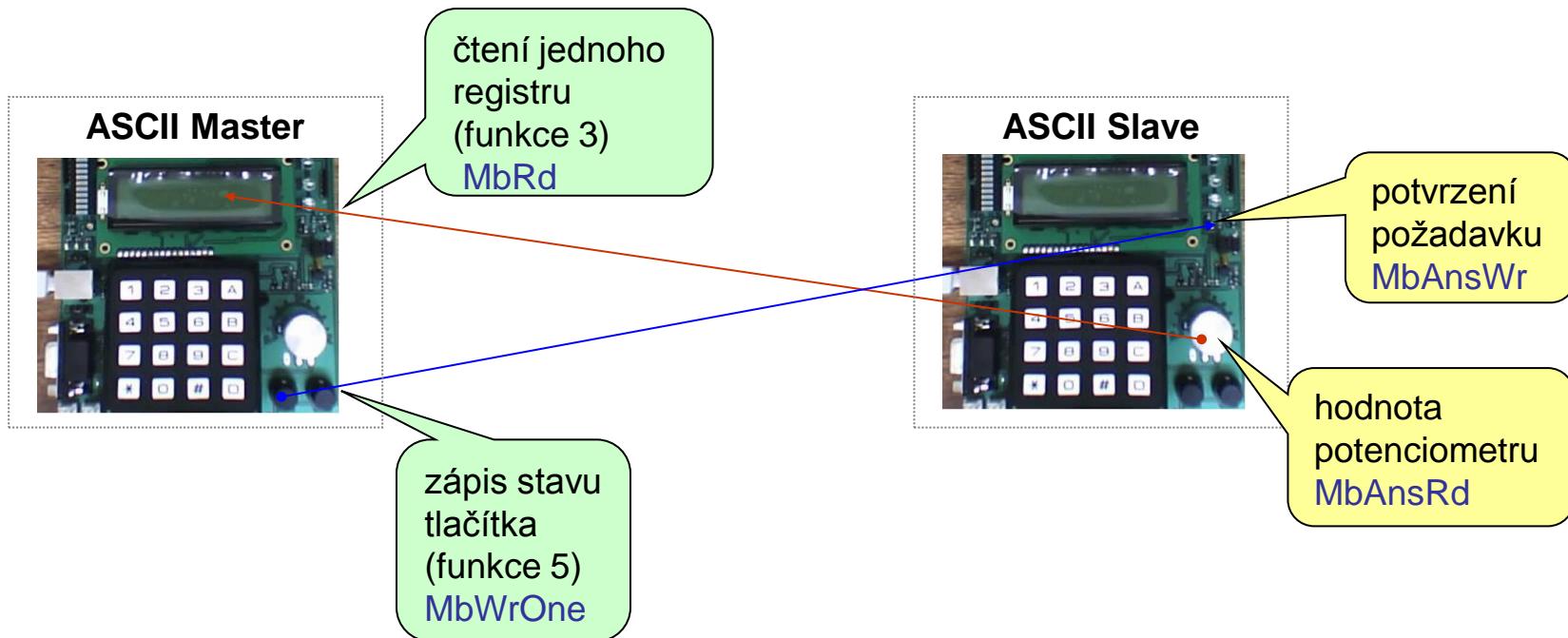
```
if(er) itx=MbAnsErr(adr_r,kod_r|0x80,er,bfout);
```

odeslání
odpovědi

```
DIR485=1; /* na vysílání */
itx+=MbWrByte(MbLrc(bfout+1,itx-1),bfout+itx);
itx+=MbWrEoT(bfout+itx);
SendBuf(bfout,itx);
DIR485=0; /* zpět na příjem */
```



3.část : mikropočítač – mikropočítač



Řídicí počítačové systémy

Úloha pro samostatná cvičení - MA3

Implementace protokolu MODBUS ASCII na PC a mikropočítačích řady '51 pro uzly Master (Klient) i Slave (Server).

Požadované implementované funkce:

- čtení 16bitového vnitřního registru (Holding) z uzlu Slave,
- čtení bitového stavu (Coil) z uzlu Slave.

Rozhraní: RS232, standardní rámec 7,N,2

1. část: propojení PC – PC (C# MSVS)

2. část: propojení PC – mikropočítač

Rozhraní: RS485, standardní rámec 7,N,2

3. část: propojení mikropočítač – mikropočítač

Funkce pro podporu aplikace protokolu MODBUS:

v souboru Modbus.dll a Modbus.cs pro PC (C#),

v souboru Modbus.H a Modbus.C pro mikropočítač



MASTER (klient)

V pravidelných časových intervalech generuje požadavek na čtení 16bitové hodnoty z uzlu SLAVE a zobrazuje ji

V pravidelných časových intervalech generuje požadavek na čtení bitové informace z uzlu SLAVE a zobrazuje ji

SLAVE (server)

Po příjmu požadavku hodnotu odešle

Po příjmu požadavku stavu bitu odešle

kód funkce: 03

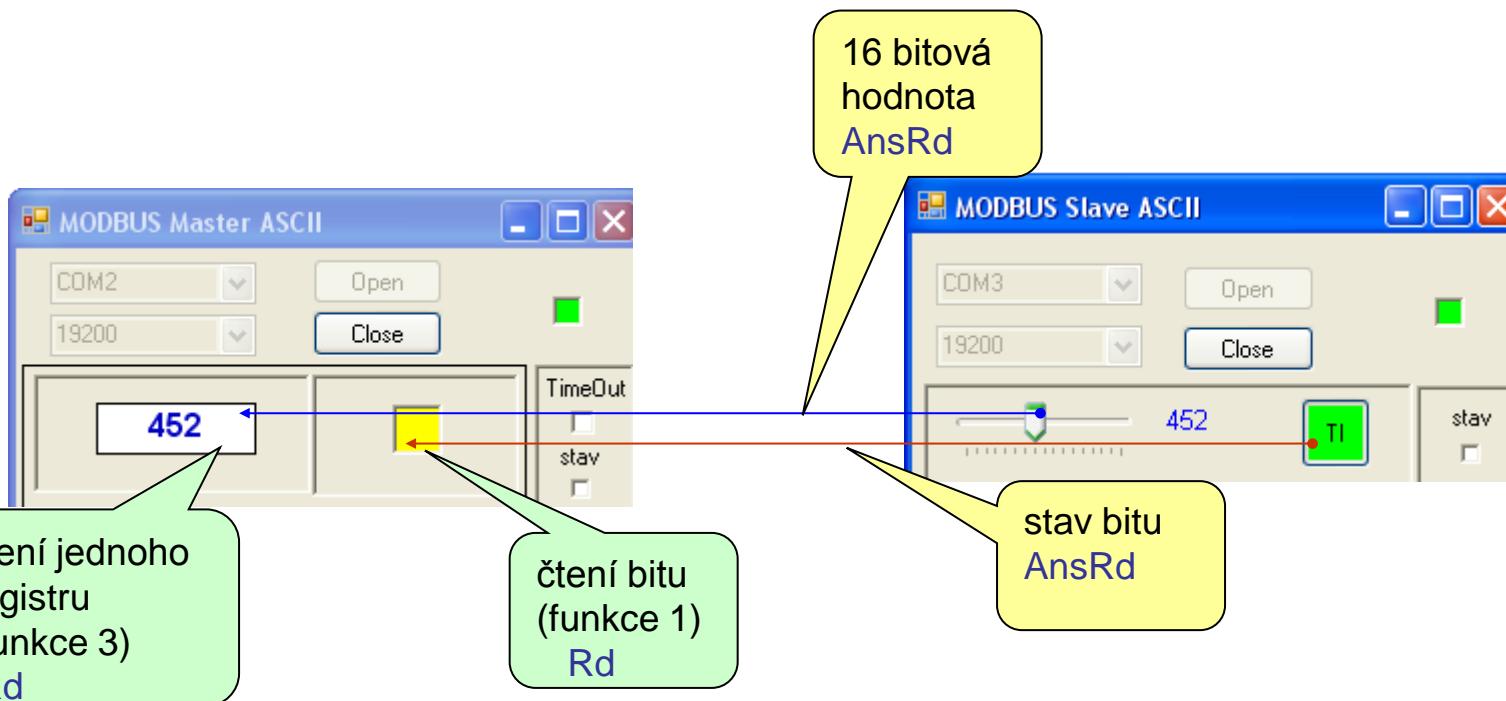
data

kód funkce: 01

data



1.část : PC-PC (varianta C#)





Master – implementace na PC (klient)

Konfigurace:

Realizuje funkce (požadavky na server)

- požadavek na čtení 16bitové hodnoty vnitřního registru – funkční kód 3

metoda Rd třídy Modbus ASCII s kódem funkce 3 (FCE_RREG)

- požadavek na čtení bitové hodnoty – funkční kód 1

metoda Rd třídy Modbus ASCII s kodem funkce 1 (FCE_RBIT)

Požadavky odesílat střídavě v pravidelných časových intervalech 200 ms,

jen když je sériový kanál otevřen a Master je ve stavu **klidu**

realizace časovačem intervalu

Implementovat generování čekacího TimeOut intervalu 500 ms na odpověď od Slave,

po vypršení TimeOutu vyčkat 500 ms a vátit se do stavu **klidu**

Zjednodušený příjem odpovědi:

příchozí adresu Slave není nutno testovat, pouze správnost LRC

zpracovat odpovědi na požadavky čtení registru (FCE_RREG)

a čtení bitu (FCE_RBIT)

informovat o chybové odpovědi od Slave



Master – vyslání požadavku

Časovač Sample

střídavě každých cca 200 ms vysílá rámec s funcí **1** (čtení bitu) a **3** (čtení registru)



ModbusASCII Ma;
bool prep;
Tstav stav;

Tick
Sample

```
if (comPort.IsOpen && stav == Tstav.stklid)
{
    stav=Tstav.stVysilani;
    prep = !prep;
    if (prep) n=Ma.Rd(ADR_S,FCE_RREG,REG_RD,1,bfout)
    else n=Ma.Rd(ADR_S,FCE_RBIT,BIT_RD,1,bfout);
    n=Ma.WrByte(Ma.Lrc(bfout,n-1),bfout,n);
    n=Ma.WrEoT(bfout,n);
    comPort.Write(bfout,0,n);
    TimerOut.Interval=500;
    TimerOut.Enabled=true;
    stav=Tstav.stCekani;
}
```



Master – zpracování odpovědi

1. LRC

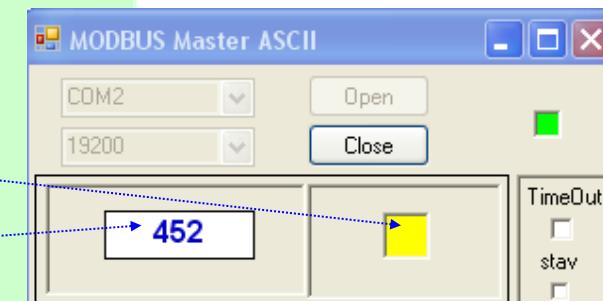
```
if(Ma.Lrc(bfin,ix-4)!=Ma.RdByte(bfin,ix-3)
{
    .. informace o chybné LRC
}
```

2. adresa a kód funkce

```
adr_r=Ma.RdByte(bfin,1);
kod_r=Ma.RdByte(bfin,3);
```

3. reakce na odpověď

```
switch (kod_r) {
    case FCE_RBIT:
        .
        .
    case FCE_RREG:
        .
        .
default: if (kod_r>=0x80){
    er= Ma.RdByte(bfin,5);
    switch (er) {
        informace o chybě stavu
    }
}
stav=Tstav.stklid;
```



4. informace o chybě Slavu



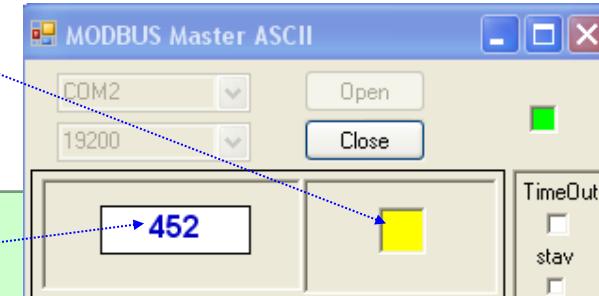
Master – zpracování odpovědi

FCE_RBIT:

```
pocet=Ma.RdByte(bfin,5);
val=Ma.RdByte(bfin,7);
if (val&1 ==1) žlutá ;
else bílá ;
break;
```

FCE_RREG:

```
pocet=Ma.RdByte(bfin,5);
val=Ma.RdWord(bfin,7);
.
.
.
break;
```





Slave – implementace na PC (server)

Konfigurace:

Přijímá požadavky od klienta a vrací odpovědi

- požadavek na čtení 16bitové hodnoty – funkční kód 3 a vrací požadovanou hodnotu,
[metoda AnsRd třídy Modbus ASCII s kódem přijaté funkce](#)
- požadavek na čtení bitové hodnoty – funkční kód 1 a vrací stav požadovaného bitu
[metoda AnsRd třídy Modbus ASCII s kódem přijaté funkce](#)

Kontrolovat přijatý požadavek a vracet chybovou odpověď v případě neimplementované funkce, neexistující adresy registru nebo bitu a hodnoty mimo rozsah

[metoda AnsErr třídy Modbus ASCII s upraveným kódem funkce a typem chyby](#)

Skupinové vysílání ignorovat .



Slave – zpracování požadavku, kontrola položek, příprava odpovědi

1. LRC

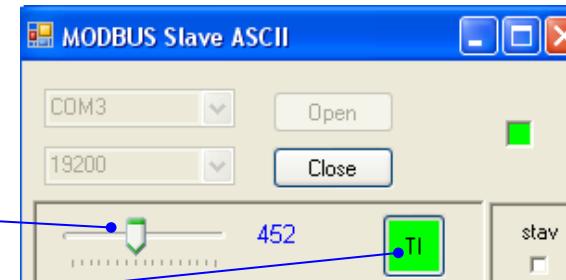
```
if(Ma.Lrc(bfin, ix-4) != Ma.RdByte(bfin, ix-3)
{
    .. možná informace o chybné LRC
}
else {
```

2. adresa

```
adr_r=Ma.RdByte(bfin,1);
if(adr_r == ADR_S)
{
```

3. kód funkce

```
kod_r=Ma.RdByte(bfin,3);
er=0;
switch(kod_r) {
    case FCE_RREG:
        .
        .
    case FCE_RBIT:
        .
        .
default: er=1;
}
```





Slave – zpracování požadavku, kontrola položek, příprava odpovědi

FCE_RREG:

```
reg=Ma.Rdword(bfin,5);
pocet=Ma.Rdword(bfin,9);
if(reg!=REG_RD || pocet!=1) er=2;
else .. MSB -> vals[0] a LSB -> vals[1]
if(er==0) n= Ma.AnSRD(ADR_S,kod_r,2,vals,bfout);
```

byte []vals = byte[2];

FCE_RBIT:

```
reg=Ma.Rdword(bfin,5);
pocet=Ma.Rdword(bfin,9);
if(reg!=BIT_RD || pocet!=1) er=2;
else {
    tlačítka -> vals[0]
    n= Ma.AnSRD(ADR_S,kod_r,1,vals,bfout);
}
```



4. chyba

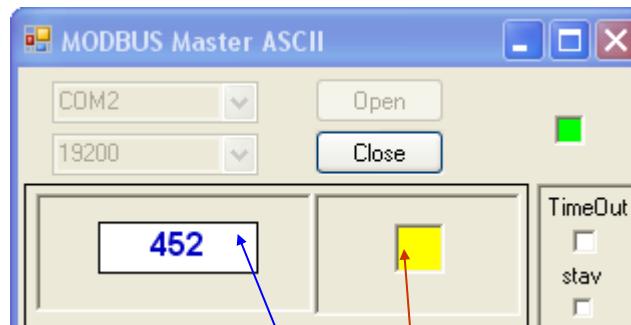
if(er>0) n=Ma.AnSerr(adr_r,(byte)(kod_r|0x80),er,bfout);

5. odeslání
odpovědi

```
n=Ma.WrByte(Ma.Lrc(bfout,n-1),bfout,n);
n=Ma.WrEoT(bfout,n);
comPort.write(bfout, 0, n);
```

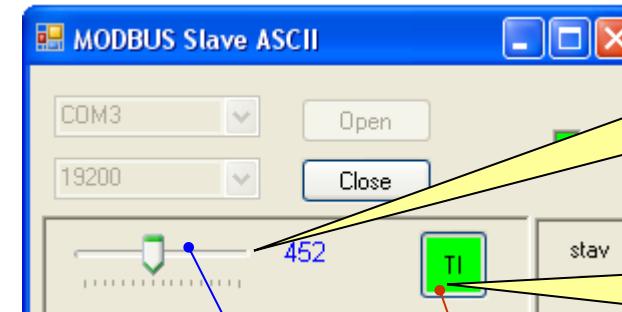


2.část : PC – mikropočítač



čtení jednoho registru
(funkce 3)
Rd

čtení bitu
(funkce 1)
Rd



16 bitová hodnota
AnsRd

stav bitu
AnsRd



hodnota potenciometru
MbAnsRd

stav tlačítka
MbAnsRd



čtení jednoho registru
(funkce 3)
MbRd

čtení bitu
(funkce 1)
MbRd



Master – implementace na mikropočítači (klient)

Konfigurace:

Realizuje funkce (požadavky na server)

- požadavek čtení 16 bitové hodnoty vnitřního registru – funkční kód 3

aplikativní funkce MbRd s kódem funkce 3 (FCE_RREG)

- požadavek na čtení bitové hodnoty – funkční kód 1

aplikativní funkce MbRd s kodem funkce 1 (FCE_RBIT)

Požadavky odesílat střídavě v pravidelných časových intervalech cca 200 ms,
jen když je Master ve stavu **klidu**

realizace časovačem T0 se základními tiky 30 ms ($30 * 7 = 210$)

Implementovat generování čekacího TimeOut intervalu 500 ms na odpověď od Slave
($30 * 17 = 510$)

Zjednodušený příjem odpovědi

příchozí adresu Slave není nutno testovat, pouze správnost LRC

zpracovat jen odpověď na požadavek čtení registru (FCE_RREG)

a čtení bitu (FCE_RBIT)

informace o chybě Slave: jen omezeně, nebo vůbec



Master – vyslání požadavku

střídavě každých cca 210 ms vysílá rámec s funcí **1** (čtení bitu) a **3** (čtení registru)



časovač T0

```
#define N_TICKS 7
#define TIMEOUT 17
bit prep;

if(++cnt_ticks>=N_TICKS && stav==stK1id)
{
    cnt_ticks=0;
    DIR485=1;      /* na vysílání - pro RS485*/
    prep=!prep;
    if(prep) itx=MbRd(ADR_S,FCE_RREG,REG_RD,1,bfout);
    else itx=MbRd(ADR_S,FCE_RBIT,BIT_RD,1,bfout);
    itx+=MbWrByte(MbLrc(bfout+1,itx-1),bfout+itx);
    itx+=MbWrEoT(bfout+itx);
    SendBuf(bfout,itx);
    stav=stCekani;
    DIR485=0;      /* zpět na příjem - pro RS485 */
}
```

TimeOut

```
if(cnt_ticks>=TIMEOUT){
    cnt_ticks=0;
    LED_R=!LED_R; // signalizace TimeOutu
    stav=stK1id;
}
```

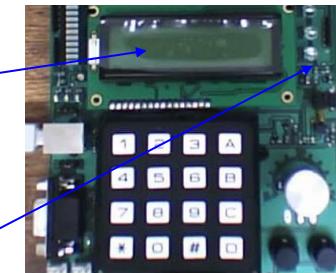
Master – zpracování odpovědi

1. LRC

```
if(MbLrc(bfin+1,ix-4)==(lrc=MbRdByte(bfin+ix-3)))  
{
```

2. kód funkce a reakce na odpověď

```
if((kod_r=MbRdByte(bfin+3))==FCE_RREG)  
{  
    pocet=MbRdByte(bfin+5);  
    val=MbRdWord(bfin+7);  
    printf(...);  
}  
  
else if(kod_r==FCE_RBIT)  
{  
    pocet=MbRdByte(bfin+5);  
    val=MbRdByte(bfin+7);  
    if(val & 1) .. ; // LED svítí  
    else .. ; // LED nesvítí  
}
```





Slave – implementace na mikropočítači (server)

Konfigurace:

Přijímá požadavky od klienta a vrací odpovědi

- požadavek na čtení 16bitové hodnoty – funkční kód 3 a vrací požadovanou hodnotu,
[aplikativní funkce MbAnsRd s kódem přijaté funkce](#)
- požadavek na čtení bitové hodnoty – funkční kód 1 a vrací stav tlačítka
[aplikativní funkce MbAnsRd s kódem přijaté funkce](#)

Kontrolovat přijatý požadavek a vracet chybovou odpověď v případě neimplementované funkce, neexistující adresy registru nebo bitu a hodnoty mimo rozsah

[aplikativní funkce MbAnsErr s upraveným kódem funkce a typem chyby](#)

Skupinové vysílání ignorovat .



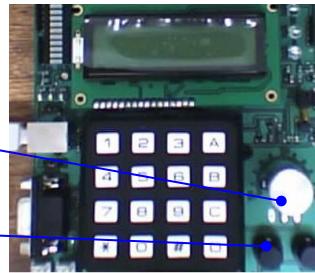
Slave – zpracování požadavku, kontrola položek, příprava odpovědi

1. LRC
a adresa

```
if((MbLrc(bfin+1,ix-4)==(lrc=MbRdByte(bfin+ix-3)))  
    && (MbRdByte(bfin+1)==ADR_S))  
{
```

2. kód
funkce

```
kod_r=MbRdByte(bfin+3);  
er=0;  
switch(kod_r)  
{  
    case FCE_RREG:  
        .  
    case FCE_RBIT:  
        .  
    default: er=1;  
}
```



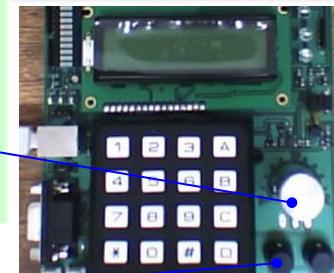


Slave – zpracování požadavku, kontrola položek, příprava odpovědi

FCE_RREG:

```
if((reg=MbRdword(bfin+5))==REG_RD&&(pocet=MbRdword(bfin+9))==1){
    val=... ;
    vals[0]=val>>8; vals[1]=val;
    itx=MbAnsRd(ADR_S,kod_r,2,vals,bfout);
}
else er=2;
break;
```

byte vals[2];



FCE_RBIT:

```
if((reg=MbRdword(bfin+5))==BIT_RD&&(pocet=MbRdword(bfin+9))==1){
    vals[0]= ... ;
    itx=MbAnsRd(ADR_S,kod_r,1,vals,bfout);
}
else er=2;
break;
```

4. chyba

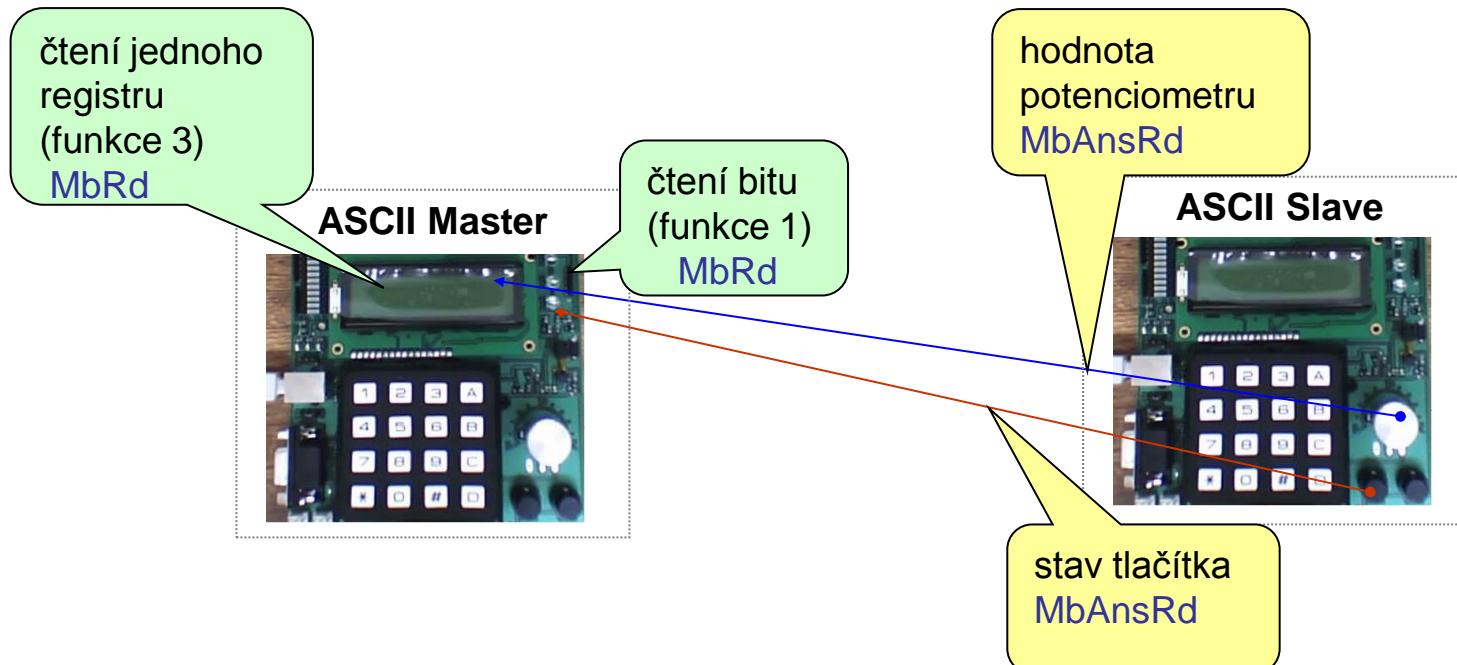
if(er) itx=MbAnsErr(adr_r,kod_r|0x80,er,bfout);

odeslání
odpovědi

```
DIR485=1; /* na vysílání */
itx+=MbWrByte(MbLrc(bfout+1,itx-1),bfout+itx);
itx+=MbWrEoT(bfout+itx);
SendBuf(bfout,itx);
DIR485=0; /* zpět na příjem */
```



3.část : mikropočítač – mikropočítač





Řídicí počítačové systémy

Úloha pro samostatná cvičení - MA4

Implementace protokolu MODBUS ASCII na PC a mikropočítacích řady '51 pro uzly Master (Klient) i Slave (Server).

Požadované implementované funkce:

- zápis jediného vnitřního registru (Holding) do uzlu Slave,
- zápis jediného bitového stavu (Coil) do uzlu Slave,

Rozhraní: RS232, standardní rámec 7,N,2

1. část: propojení PC – PC (C# MSVS)

2. část: propojení PC – mikropočítač

Rozhraní: RS485, standardní rámec 7,N,2

3. část: propojení mikropočítač – mikropočítač

Funkce pro podporu aplikace protokolu MODBUS:

v souboru Modbus.dll a Modbus.cs pro PC (C#),

v souboru Modbus.H a Modbus.C pro mikropočítač



MASTER (klient)

V pravidelných časových intervalech generuje 16 bitovou hodnotu a předává požadavek na zápis do uzlu SLAVE

V pravidelných časových intervalech generuje 1bitovou informaci a předává požadavek na zápis do uzlu SLAVE

kód fukce: 06
+ data

potvrzení

kód fukce: 05
+ data

potvrzení

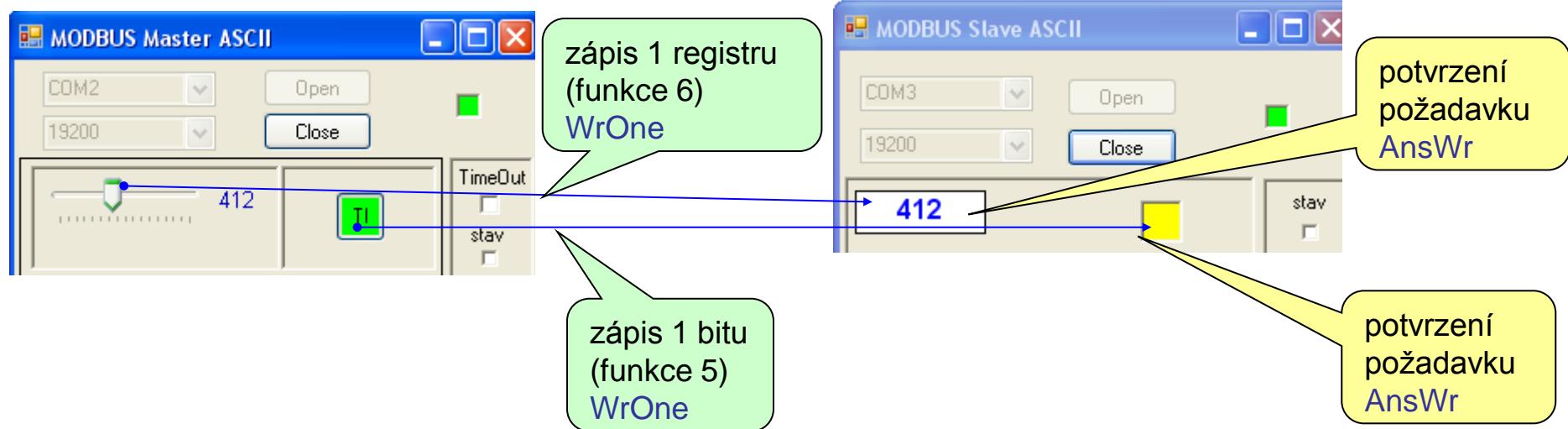
SLAVE (server)

Po příjmu hodnotu zobrazí a odešle potvrzovací odpověď

Po příjmu informaci zobrazí a odešle potvrzovací odpověď



1.část : PC-PC (varianta C#)





Master – implementace na PC (klient)

Konfigurace:

Realizuje funkce (požadavky na server)

- požadavek na zápis jediného vnitřního registru (hodnota 0 až 1023) – funkční kód 6

metoda WrOne třídy Modbus ASCII s kódem funkce 6 (FCE_WREG)

- požadavek na zápis jediného bitového stavu – funkční kód 5

metoda WrOne třídy Modbus ASCII s kódem funkce 5 (FCE_WBIT)

Požadavky odesílat střídavě v pravidelných časových intervalech 200 ms,

jen když je sériový kanál otevřen a Master je ve stavu **klidu**

realizace časovačem intervalu

Implementovat generování čekacího TimeOut intervalu 500 ms na odpověď od Slave,

po vypršení TimeOutu vyčkat 500 ms a vátit se do stavu **klidu**

Zjednodušený příjem odpovědi

příchozí adresu Slave není nutno testovat, pouze správnost LRC

zpracovat jen chybové odpovědi od Slave a informovat o nich



Master – vyslání požadavku

Časovač Sample

střídavě každých cca 200 ms vysílá rámec s funcí 5 (zápis bitu) a 6 (zápis registru)

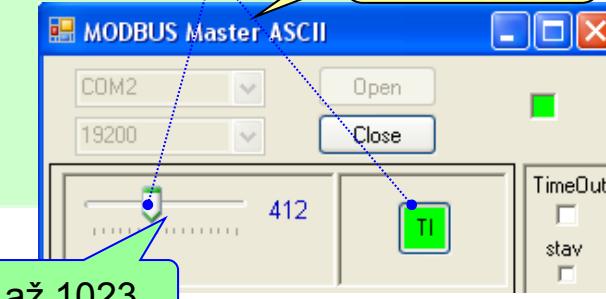


ModbusASCII Ma;
bool prep;
Tstav stav;

Tick
Sample

```
if (comPort.IsOpen && stav == Tstav.stklid)
{
    stav=Tstav.stVysilani;
    prep = !prep;
    if (prep) n=Ma.wrOne(ADR_S,FCE_WREG,REG_WR,pot,bfout)
    else n=Ma.wrOne(ADR_S,FCE_WBIT,BIT_WR,val,bfout);
    n=Ma.wrByte(Ma.Lrc(bfout,n-1),bfout,n);
    n=Ma.wrEoT(bfout,n);
    comPort.Write(bfout,0,n);
    TimerOut.Interval=500;
    TimerOut.Enabled=true;
    stav=Tstav.stCekani;
}
```

stisk: 0xFF00
jinak: 0





Master – zpracování odpovědi

1. LRC

```
if(Ma.Lrc(bfin,ix-4)!=Ma.RdByte(bfin,ix-3)
{
    .. informace o chybné LRC
}
```

2. adresa a kód funkce

```
adr_r=Ma.RdByte(bfin,1);
kod_r=Ma.RdByte(bfin,3);
```

3. informace o chybě Slavu

```
if (kod_r>=0x80)
{
    er= Ma.RdByte(bfin,5);
    switch(er) {
        informace o chybě slavu
    }
}
stav=Tstav.stKlid;
```



Slave – implementace na PC (server)

Konfigurace:

Přijímá požadavky od klienta a vrací odpovědi

- požadavek na zápis jediného vnitřního registru (hodnota 0 až 1023) – funkční kód 6, hodnotu zobrazí a vrací potvrzení o přijetí požadavku

metoda `AnsWr` třídy `Modbus ASCII` s kódem přijaté funkce

- požadavek na zápis jediného bitového stavu – funkční kód 5, stav indikuje a vrací potvrzení o přijetí požadavku

metoda `AnsWr` třídy `Modbus ASCII` s kódem přijaté funkce

Kontrolovat přijatý požadavek a vracet chybovou odpověď v případě neimplementované funkce, neexistující adresy registru nebo bitu a hodnoty mimo rozsah

metoda `AnsErr` třídy `Modbus ASCII` s upraveným kódem funkce a typem chyby

Skupinové vysílání ignorovat .



Slave – zpracování požadavku, kontrola položek, příprava odpovědi

1. LRC

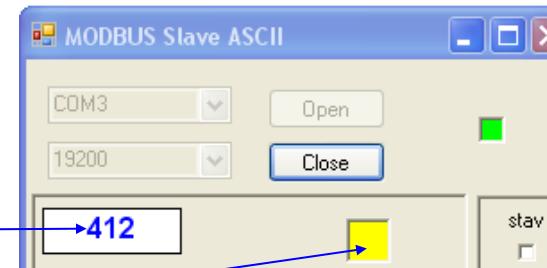
```
if(Ma.Lrc(bfin, ix-4) != Ma.RdByte(bfin, ix-3)
{
    .. možná informace o chybné LRC
}
else {
```

2. adresa

```
adr_r=Ma.RdByte(bfin,1);
if(adr_r == ADR_S)
{
```

3. kód funkce

```
kod_r=Ma.RdByte(bfin,3);
er=0;
switch(kod_r) {
    case FCE_WREG:
        .
        .
    case FCE_WBIT:
        .
        .
default: er=1;
}
```

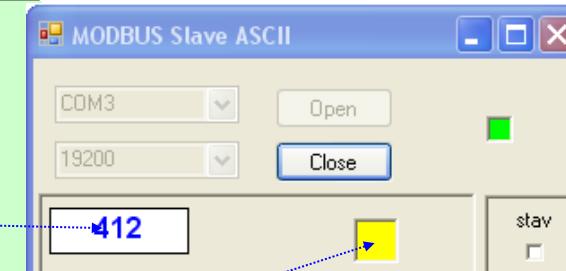




Slave – zpracování požadavku, kontrola položek, příprava odpovědi

FCE_WREG:

```
reg=Ma.Rdword(bfin,5);
val=Ma.Rdword(bfin,9);
if(reg!=REG_WR) er=2;
else if(val>1023) er=3;
else .. zobrazení hodnoty
if(er==0) n= Ma.Answr(ADR_S,kod_r,reg,val,bfout);
```



FCE_WBIT:

```
reg=Ma.Rdword(bfin,5);
val=Ma.Rdword(bfin,9);
if(reg!=BIT_WR) er=2;
else switch(val){
    case 0xFF00: .. žlutá ; break;
    case 0x0000: .. bílá ; break;
    default: err=3;
}
if(err==0) n= Ma.Answr(ADR_S,kod_r,reg,val,bfout);
```

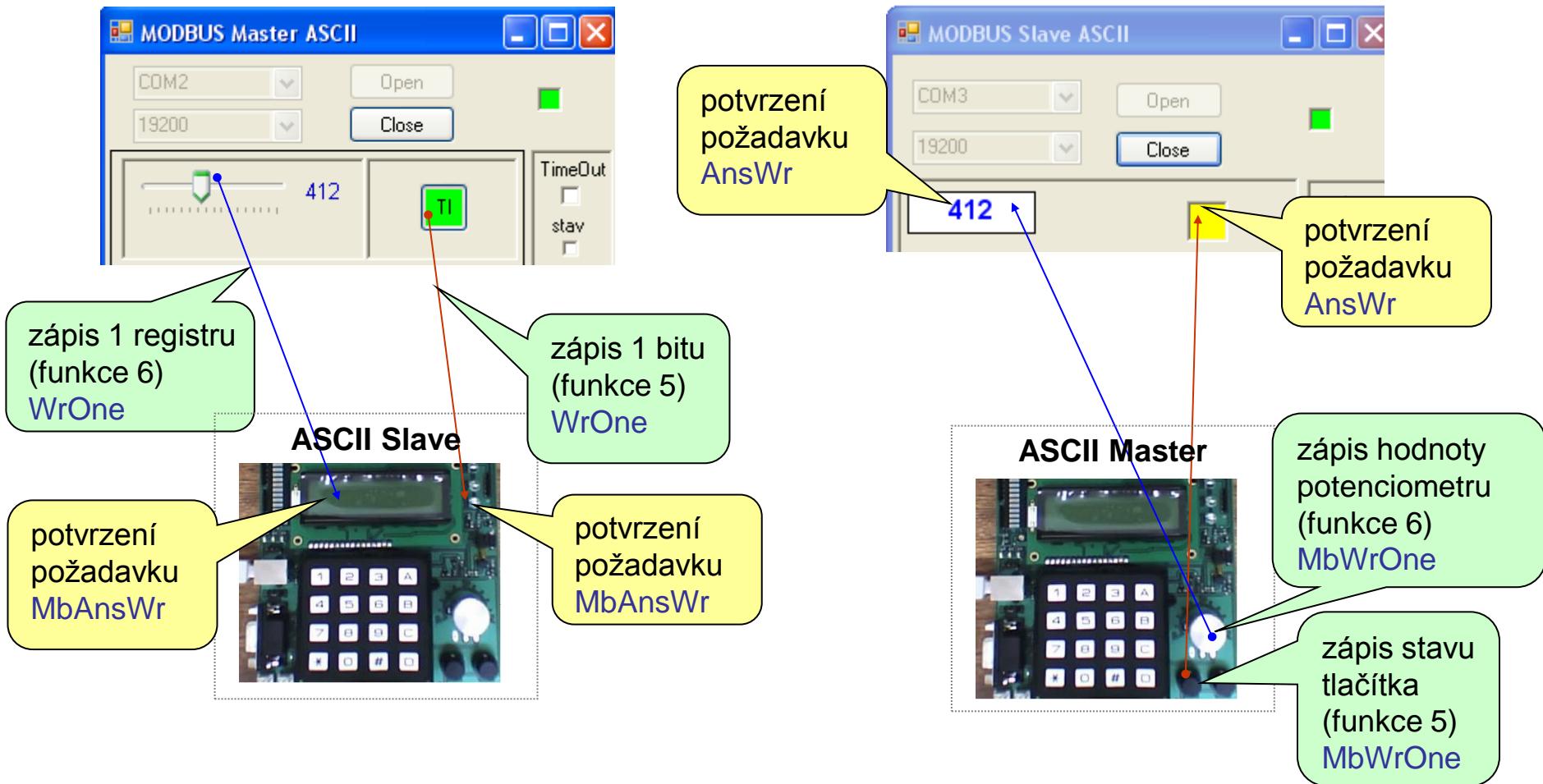
4. chyba

```
if(er>0) n=Ma.Answr(adr_r,(byte)(kod_r|0x80),er,bfout);
```

5. odeslání
odpovědi

```
n=Ma.WrByte(Ma.Lrc(bfout,n-1),bfout,n);
n=Ma.WrEoT(bfout,n);
comPort.Write(bfout, 0, n);
```

2.část : PC – mikropočítač





Master – implementace na mikropočítači (klient)

Konfigurace:

Realizuje funkce (požadavky na server)

- požadavek na zápis jediného vnitřního registru (hodnota 0 až 1023) – funkční kód 6
[aplikativní funkce MbWrOne s kódem funkce 6 \(FCE_WREG\)](#)
- požadavek na zápis jediného bitového stavu – funkční kód 5
[aplikativní funkce MbWrOne s kódem funkce 5 \(FCE_WBIT\)](#)

Požadavky odesílat střídavě v pravidelných časových intervalech cca 200 ms,
jen když je sériový kanál otevřen a Master je ve stavu **klidu**

realizace časovačem T0 se základními tiky 30 ms ($30 * 7 = 210$)

Implementovat generování čekacího TimeOut intervalu 500 ms na odpověď od Slave
($30 * 17 = 510$)

Zjednodušený příjem odpovědi

příchozí adresu Slave není nutno testovat, pouze správnost LRC

zpracovat informaci o chybě Slave: informovat jen omezeně (např žlutá LED,
nebo vůbec)



Master – vyslání požadavku

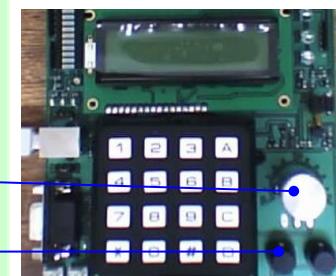
střídavě každých cca 210 ms vysílá rámec s funcí **5** (zápis bitu) a **6** (zápis registru)



časovač T0

```
if(++cnt_ticks>=N_TICKS && stav==stK1id){
    cnt_ticks=0;
    DIR485=1;      /* na vysílání - pro RS485*/
    prep=!prep;
    if(prep) { val = ...;
                itx=MbWrOne(ADR_S, FCE_WREG, REG_WR, val, bfout); }
    else { val = ... ;
                itx=MbWrOne(ADR_S, FCE_WBIT, BIT_WR, val, bfout); }
    itx+=MbWrByte(MbLrc(bfout+1, itx-1), bfout+itx);
    itx+=MbWrEoT(bfout+itx);
    SendBuf(bfout, itx);
    stav=stCekani;
    DIR485=0;      /* zpět na příjem - pro RS485 */
}
```

```
#define N_TICKS 7
#define TIMEOUT 17
bit prep;
```



TimeOut

```
if(cnt_ticks>=TIMEOUT)
{
    cnt_ticks=0;
    LED_R=!LED_R; // signalizace TimeOutu
    stav=stK1id;
}
```



Master – zpracování odpovědi

1. LRC

```
if(MbLrc(bfin+1,ix-4)==(lrc=MbRdByte(bfin+ix-3)))  
{
```

2. kód funkce
a reakce
na chybu
SLAVE

```
if( (kod_r=MbRdByte(bfin+3))>=0x80)  
    LED_Y=0;  
else  LED_Y=1;
```





Slave – implementace na mikropočítači (server)

Konfigurace:

Přijímá požadavky od klienta a vrací odpovědi

- požadavek na zápis jediného vnitřního registru (hodnota 0 až 1023) – funkční kód 6, hodnotu zobrazí na LCD a vrací potvrzení o přijetí požadavku

aplikáční funkce MbAnsWr s kódem přijaté funkce

- požadavek na zápis jediného bitového stavu – funkční kód 5, stav indikuje LED (zelené) a vrací potvrzení o přijetí požadavku

aplikáční funkce MbAnsWr s kódem přijaté funkce

Kontrolovat přijatý požadavek a vracet chybovou odpověď v případě neimplementované funkce, neexistující adresy registru nebo bitu a hodnoty mimo rozsah

aplikáční funkce MbAnsErr s upraveným kódem funkce a typem chyby

Skupinové vysílání ignorovat .



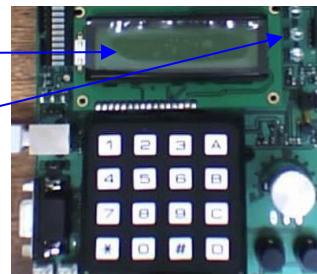
Slave – zpracování požadavku, kontrola položek, příprava odpovědi

1. LRC
a adresa

```
if((MbLrc(bfin+1,ix-4)==(lrc=MbRdByte(bfin+ix-3)))  
    && (MbRdByte(bfin+1)==ADR_S))  
{
```

2. kód
funkce

```
kod_r=MbRdByte(bfin+3);  
er=0;  
switch(kod_r)  
{  
    case FCE_WREG:  
        .  
        .  
    case FCE_WBIT:  
        .  
        .  
    default: er=1;  
}
```

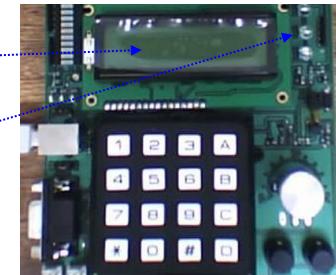




Slave – zpracování požadavku, kontrola položek, příprava odpovědi

FCE_WREG:

```
if((reg=MbRdWord(bfin+5))!=REG_WR) er=2;
else if((val=MbRdWord(bfin+9))>1023) er=3;
else printf(...);
if(er==0) itx=MbAnsWr(ADR_S,kod_r,reg,val,bfout);
break;
```



FCE_WBIT:

```
if((reg=MbRdWord(bfin+5))!=BIT_WR) er=2;
else if((val=MbRdWord(bfin+9))!=0&&val!=0xFF00) er=3;
else LED_G ...;
if(er==0) itx=MbAnsWr(ADR_S,kod_r,reg,val,bfout);
break;
```

4. chyba

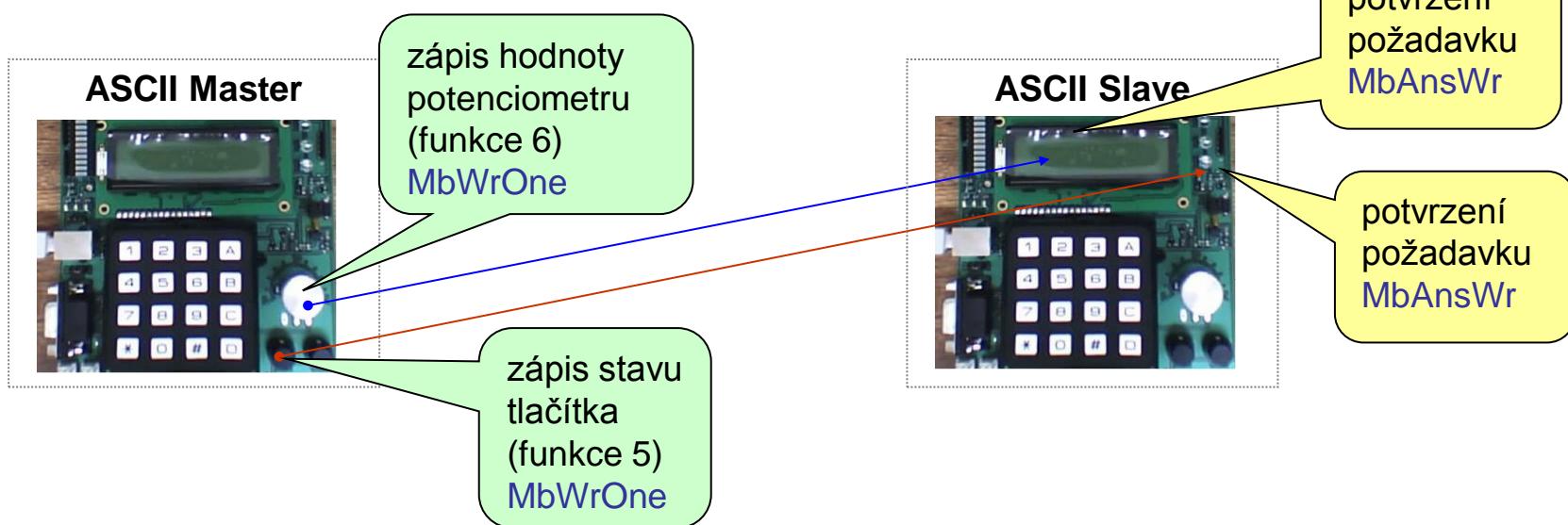
```
if(er) itx=MbAnsErr(adr_r,kod_r|0x80,er,bfout);
```

odeslání
odpovědi

```
DIR485=1; /* na vysílání - pro RS485*/
itx+=MbWrByte(MbLrc(bfout+1,itx-1),bfout+itx);
itx+=MbWrEoT(bfout+itx);
SendBuf(bfout,itx);
DIR485=0; /* zpět na příjem - pro RS485 */
```



3.část : mikropočítač – mikropočítač



Řídicí počítačové systémy

Úloha pro samostatná cvičení - MA5

Implementace protokolu MODBUS ASCII na PC a mikropočítačích řady '51 pro uzly Master (Klient) i Slave (Server).

Požadované implementované funkce:

- čtení 16bitového vnitřního registru (Holding) z uzlu Slave,
- zápis jediného vnitřního registru (Holding) do uzlu Slave,

Rozhraní: RS232, standardní rámec 7,N,2

1. část: propojení PC – PC (C# MSVS)

2. část: propojení PC – mikropočítač

Rozhraní: RS485, standardní rámec 7,N,2

3. část: propojení mikropočítač – mikropočítač

Funkce pro podporu aplikace protokolu MODBUS:

v souboru Modbus.dll a Modbus.cs pro PC (C#),

v souboru Modbus.H a Modbus.C pro mikropočítač



MASTER (klient)

V pravidelných časových intervalech generuje požadavek na čtení 16bitové hodnoty z uzlu SLAVE a zobrazuje ji

V pravidelných časových intervalech generuje 16 bitovou hodnotu a předává požadavek na zápis do uzlu SLAVE

SLAVE (server)

Po příjmu požadavku hodnotu odešle

Po příjmu hodnotu zobrazí a odešle potvrzovací odpověď

kód fukce: 03

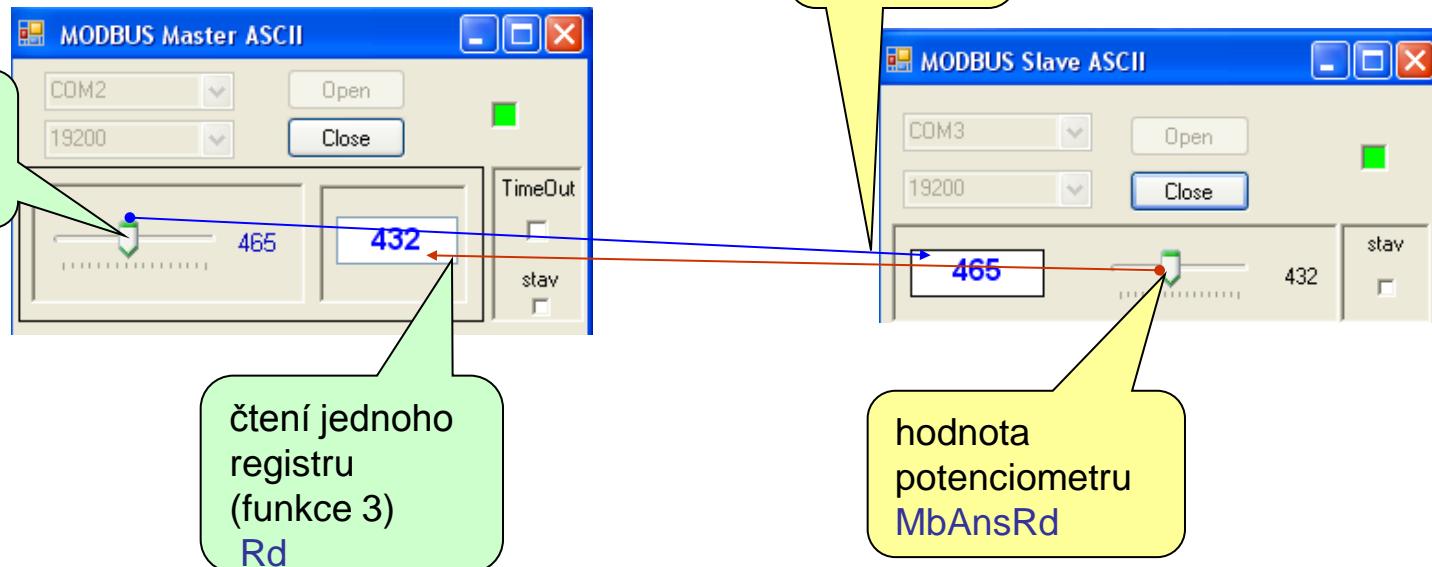
data

kód fukce: 06
+ data

potvrzení



1.část : PC-PC (varianta C#)





Master – implementace na PC (klient)

Konfigurace:

Realizuje funkce (požadavky na server)

- požadavek na čtení 16bitové hodnoty vnitřního registru – funkční kód 3
[metoda Rd třídy Modbus ASCII s kódem funkce 3 \(FCE_RREG\)](#)
- požadavek na zápis jediného vnitřního registru (hodnota 0 až 1023) – funkční kód 6
[metoda WrOne třídy Modbus ASCII s kódem funkce 6 \(FCE_WREG\)](#)

Požadavky odesílat střídavě v pravidelných časových intervalech 200 ms,

jen když je sériový kanál otevřen a Master je ve stavu **klidu**

realizace časovačem intervalu

Implementovat generování čekacího TimeOut intervalu 500 ms na odpověď od Slave,
po vypršení TimeOutu vyčkat 500 ms a vátit se do stavu **klidu**

Zjednodušený příjem odpovědi:

příchozí adresu Slave není nutno testovat, pouze správnost LRC
zpracovat jen odpověď na požadavek čtení registru (FCE_RREG)
informovat o chybové odpovědi od Slave



Master – vyslání požadavku

Časovač Sample

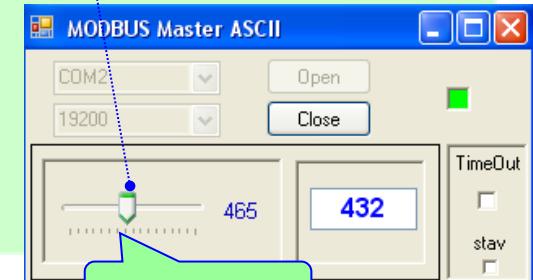
střídavě každých cca 200 ms vysílá rámec s funcí 3 (čtení registru) a 6 (zápis registru)



ModbusASCII Ma;
bool prep;
Tstav stav;

Tick
Sample

```
if (comPort.IsOpen && stav == Tstav.stklid)
{
    stav=Tstav.stVysilani;
    prep = !prep;
    if (prep) n=Ma.Rd(ADR_S,FCE_RREG,REG_RD,1,bfout)
    else n=Ma.WrOne(ADR_S,FCE_WREG,REG_WR,pot,bfout)
    n=Ma.WrByte(Ma.Lrc(bfout,n-1),bfout,n);
    n=Ma.WrEoT(bfout,n);
    comPort.Write(bfout,0,n);
    TimerOut.Interval=500;
    TimerOut.Enabled=true;
    stav=Tstav.stCekani;
}
```



0 až 1023



Master – zpracování odpovědi

1. LRC

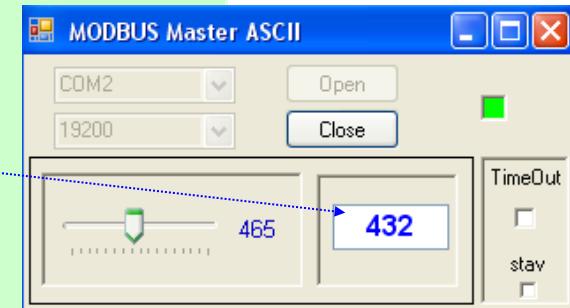
```
if(Ma.Lrc(bfin, ix-4) != Ma.RdByte(bfin, ix-3)
{
    .. informace o chybné LRC
}
```

2. adresa a kód funkce

```
adr_r=Ma.RdByte(bfin,1);
kod_r=Ma.RdByte(bfin,3);
```

3. reakce na odpověď

```
if(kod_r== FCE_RREG)
{
    pocet=Ma.RdByte(bfin,5);
    if (adr_r==ADR_S && pocet==1)
    {
        val=Ma.Rdword(bfin,7);
        .
    }
}
else if (kod_r>=0x80)
{
    er= Ma.RdByte(bfin,5);
    switch(er) {
        informace o chybě slavu
    }
}
stav=Tstav.stklid;
```





Slave – implementace na PC (server)

Konfigurace:

Přijímá požadavky od klienta a vrací odpovědi

- požadavek na zápis jediného vnitřního registru (hodnota 0 až 1023) – funkční kód 6, hodnotu zobrazí a vrací potvrzení o přijetí požadavku

metoda `AnsWr` třídy `Modbus ASCII` s kódem přijaté funkce

- požadavek na čtení 16 bitové hodnoty – funkční kód 3 a vrací požadovanou hodnotu

metoda `AnsRd` třídy `Modbus ASCII` s kódem přijaté funkce

Kontrolovat přijatý požadavek a vracet chybovou odpověď v případě neimplementované funkce, neexistující adresy registru nebo bitu a hodnoty mimo rozsah

metoda `AnsErr` třídy `Modbus ASCII` s upraveným kódem funkce a typem chyby

Skupinové vysílání ignorovat .



Slave – zpracování požadavku, kontrola položek, příprava odpovědi

1. LRC

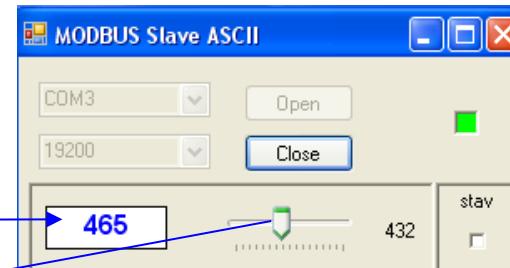
```
if(Ma.Lrc(bfin, ix-4) != Ma.RdByte(bfin, ix-3)
{
    .. možná informace o chybné LRC
}
else {
```

2. adresa

```
adr_r=Ma.RdByte(bfin,1);
if(adr_r == ADR_S)
{
```

3. kód funkce

```
kod_r=Ma.RdByte(bfin,3);
er=0;
switch(kod_r) {
    case FCE_WREG:
        .
        .
    case FCE_RREG:
        .
        .
default: er=1;
}
```





Slave – zpracování požadavku, kontrola položek, příprava odpovědi

FCE_WREG:

```
reg=Ma.Rdword(bfin,5);
val=Ma.Rdword(bfin,9);
if(reg!=REG_WR) er=2;
else if(val>1023) er=3;
else .. zobrazení hodnoty
if(er==0) n= Ma.Answr(ADR_S,kod_r,reg,val,bfout);
```



FCE_RREG:

```
reg=Ma.Rdword(bfin,5);
pocet=Ma.RdWord(bfin,9);
if(reg!=REG_RD || pocet!=1) er=2;
else .. MSB -> vals[0] a LSB -> vals[1]
if(er==0) n= Ma.Answr(ADR_S,kod_r,2,vals,bfout);
```

byte []vals = byte[2];

4. chyba

if(er>0) n=Ma.Answr(adr_r,(byte)(kod_r|0x80),er,bfout);

5. odeslání
odpovědi

```
n=Ma.WrByte(Ma.Lrc(bfout,n-1),bfout,n);
n=Ma.WrEoT(bfout,n);
comPort.write(bfout, 0, n);
```

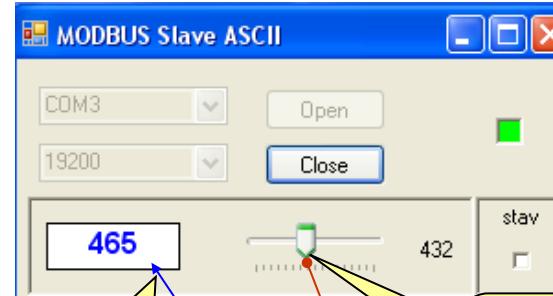


2.část : PC – mikropočítač

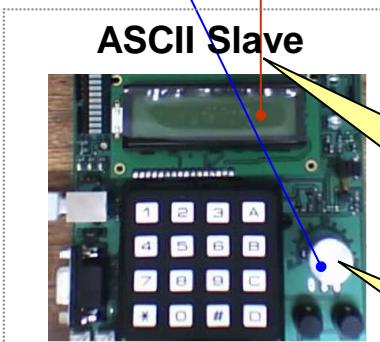


zápis 1 registru
(funkce 6)
WrOne

čtení jednoho
registru
(funkce 3)
Rd

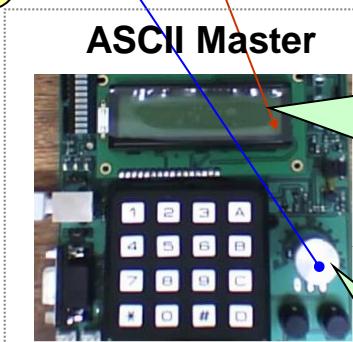


16 bitová
hodnota
AnsRd



potvrzení
požadavku
MbAnsWr

hodnota
potenciometru
MbAnsRd



čtení jednoho
registru
(funkce 3)
MbRd

zápis hodnoty
potenciometru
(funkce 6)
MbWrOne



Master – implementace na mikropočítači (klient)

Konfigurace:

Realizuje funkce (požadavky na server)

- požadavek na zápis jediného vnitřního registru (hodnota 0 až 1023) – funkční kód 6
[aplikativní funkce MbWrOne s kódem funkce 6 \(FCE_WREG\)](#)
- požadavek čtení 16 bitové hodnoty vnitřního registru – funkční kód 3
[aplikativní funkce MbRd s kodem funkce 3 \(FCE_RREG\)](#)

Požadavky odesílat střídavě v pravidelných časových intervalech cca 200 ms,
jen když je sériový kanál otevřen a Master je ve stavu **klidu**
realizace časovačem T0 se základními tiky 30 ms ($30 * 7 = 210$)

Implementovat generování čekacího TimeOut intervalu 500 ms na odpověď od Slave
($30 * 17 = 510$)

Zjednodušený příjem odpovědi

příchozí adresu Slave není nutno testovat, pouze správnost LRC
zpracovat jen odpověď na požadavek čtení bitu (FCE_RBIT)
informace o chybě Slave: jen omezeně, nebo vůbec



Master – vyslání požadavku

střídavě každých cca 210 ms vysílá rámec s funcí **3** (čtení registru) a **6** (zápis registru)



časovač T0

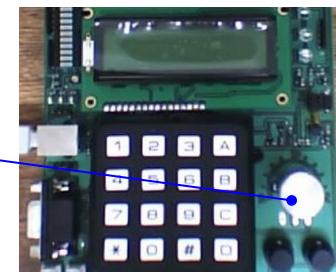
```
#define N_TICKS 7
#define TIMEOUT 17
bit prep;

if(++cnt_ticks>=N_TICKS && stav==stKlid){
    cnt_ticks=0;
    DIR485=1;      /* na vysílání - pro RS485*/
    prep=!prep;
    if(prep){ val= ... ;
        itx=MbWrOne(ADR_S,FCE_WREG,REG_WR,val,bfout);}
    else itx=MbRd(ADR_S,FCE_RREG,REG_RD,1,bfout);
    itx+=MbWrByte(MbLrc(bfout+1,itx-1),bfout+itx);
    itx+=MbWrEoT(bfout+itx);
    SendBuf(bfout,itx);
    stav=stCekani;
    DIR485=0;      /* zpět na příjem - pro RS485 */
}
```

```
#define N_TICKS 7
#define TIMEOUT 17
bit prep;
```

TimeOut

```
if(cnt_ticks>=TIMEOUT){
    cnt_ticks=0;
    LED_R=!LED_R; // signalizace TimeOutu
    stav=stKlid;
}
```





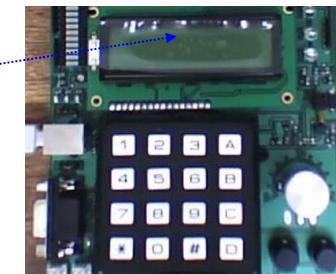
Master – zpracování odpovědi

1. LRC

```
if(MbLrc(bfin+1,ix-4)==(lrc=MbRdByte(bfin+ix-3)))
{
```

2. kód funkce
a reakce
na odpověď

```
if( (kod_r=MbRdByte(bfin+3))==FCE_RREG)
{
    pocet=MbRdByte(bfin+5);
    val=MbRdWord(bfin+7);
    printf(...);
}
```





Slave – implementace na mikropočítači (server)

Konfigurace:

Přijímá požadavky od klienta a vrací odpovědi

- požadavek na zápis jediného vnitřního registru (hodnota 0 až 1023) – funkční kód 6, hodnotu zobrazí na LCD a vrací potvrzení o přijetí požadavku
[aplikáční funkce MbAnsWr s kódem přijaté funkce](#)

- požadavek na čtení 16 bitové hodnoty – funkční kód 3 a vrací požadovanou hodnotu
[aplikáční funkce MbAnsRd s kódem přijaté funkce](#)

Kontrolovat přijatý požadavek a vracet chybovou odpověď v případě neimplementované funkce, neexistující adresy registru nebo bitu a hodnoty mimo rozsah

[aplikáční funkce MbAnsErr s upraveným kódem funkce a typem chyby](#)

Skupinové vysílání ignorovat .



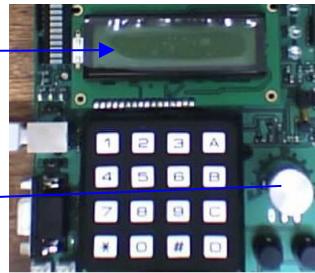
Slave – zpracování požadavku, kontrola položek, příprava odpovědi

1. LRC
a adresa

```
if((MbLrc(bfin+1,ix-4)==(lrc=MbRdByte(bfin+ix-3)))  
    && (MbRdByte(bfin+1)==ADR_S))  
{
```

2. kód
funkce

```
kod_r=MbRdByte(bfin+3);  
er=0;  
switch(kod_r)  
{  
    case FCE_WREG:  
        .  
        .  
    case FCE_RREG:  
        .  
        .  
    default: er=1;  
}
```

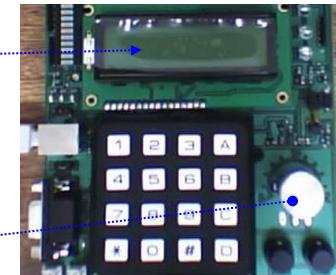




Slave – zpracování požadavku, kontrola položek, příprava odpovědi

FCE_WREG:

```
if((reg=MbRdword(bfin+5))!=REG_WR) er=2;
else if((val=MbRdword(bfin+9))>1023) er=3;
else printf(...);
if(er==0)itx=MbAnsWr(ADR_S,kod_r,reg,val,bfout);
break;
```



FCE_RREG:

```
if((reg=MbRdword(bfin+5))==REG_RD&&(pocet=MbRdword(bfin+9))==1){
    val = ... ;
    vals[0]= val>>8;
    vals[1]=val;
    itx=MbAnsRd(ADR_S,kod_r,1,vals,bfout);
}
else er=2;
break;
```

byte vals[2];

4. chyba

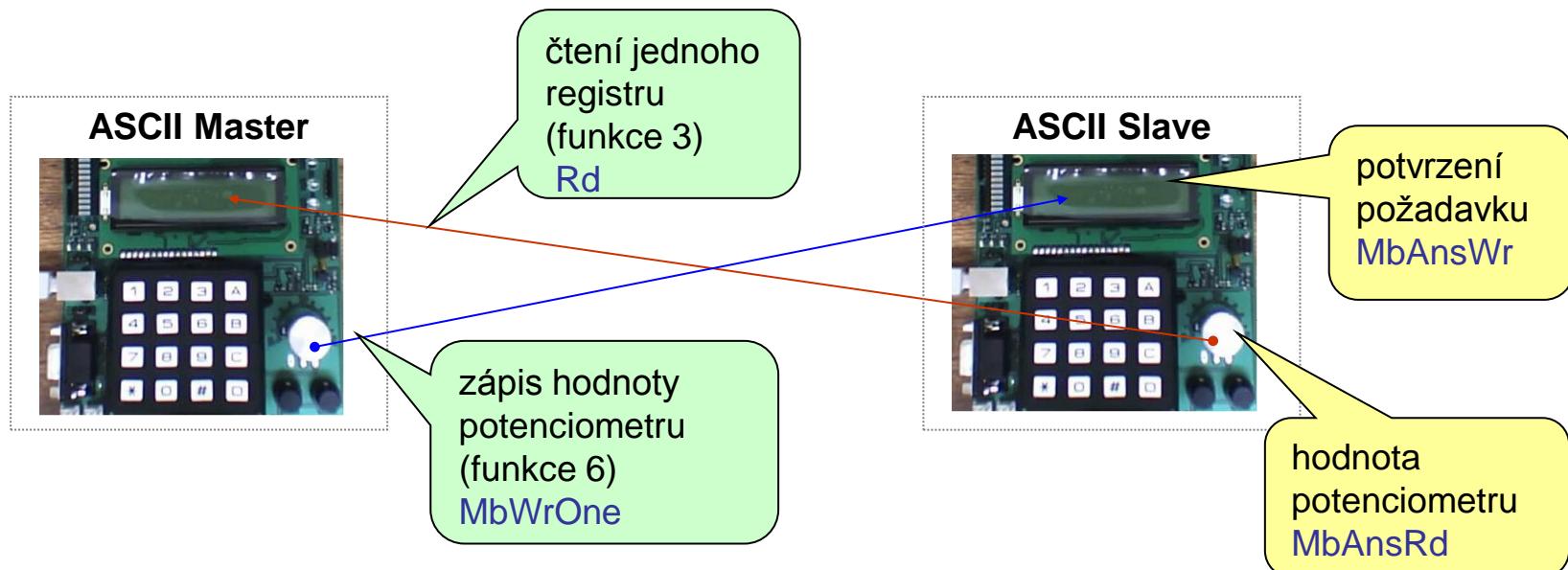
```
if(er) itx=MbAnsErr(adr_r,kod_r|0x80,er,bfout);
```

odeslání
odpovědi

```
DIR485=1; /* na vysílání */
itx+=MbWrByte(MbLrc(bfout+1,itx-1),bfout+itx);
itx+=MbWrEoT(bfout+itx);
SendBuf(bfout,itx);
DIR485=0; /* zpět na příjem */
```



3.část : mikropočítač – mikropočítač





Řídicí počítačové systémy

Úloha pro samostatná cvičení - MA6

Implementace protokolu MODBUS ASCII na PC a mikropočítacích řady '51 pro uzly Master (Klient) i Slave (Server).

Požadované implementované funkce:

- zápis jediného bitového stavu (Coil) do uzlu Slave,
- čtení bitového stavu (Coil) z uzlu Slave.

Rozhraní: RS232, standardní rámec 7,N,2

1. část: propojení PC – PC (C# MSVS)

2. část: propojení PC – mikropočítač

Rozhraní: RS485, standardní rámec 7,N,2

3. část: propojení mikropočítač – mikropočítač

Funkce pro podporu aplikace protokolu MODBUS:

v souboru Modbus.dll a Modbus.cs pro PC (C#),

v souboru Modbus.H a Modbus.C pro mikropočítač



MASTER (klient)

V pravidelných časových intervalech generuje 1bitovou informaci a předává požadavek na zápis do uzlu SLAVE

V pravidelných časových intervalech generuje požadavek na čtení bitové informace z uzlu SLAVE a zobrazuje ji

kód fukce: 05
+ data

potvrzení

kód fukce: 01

data

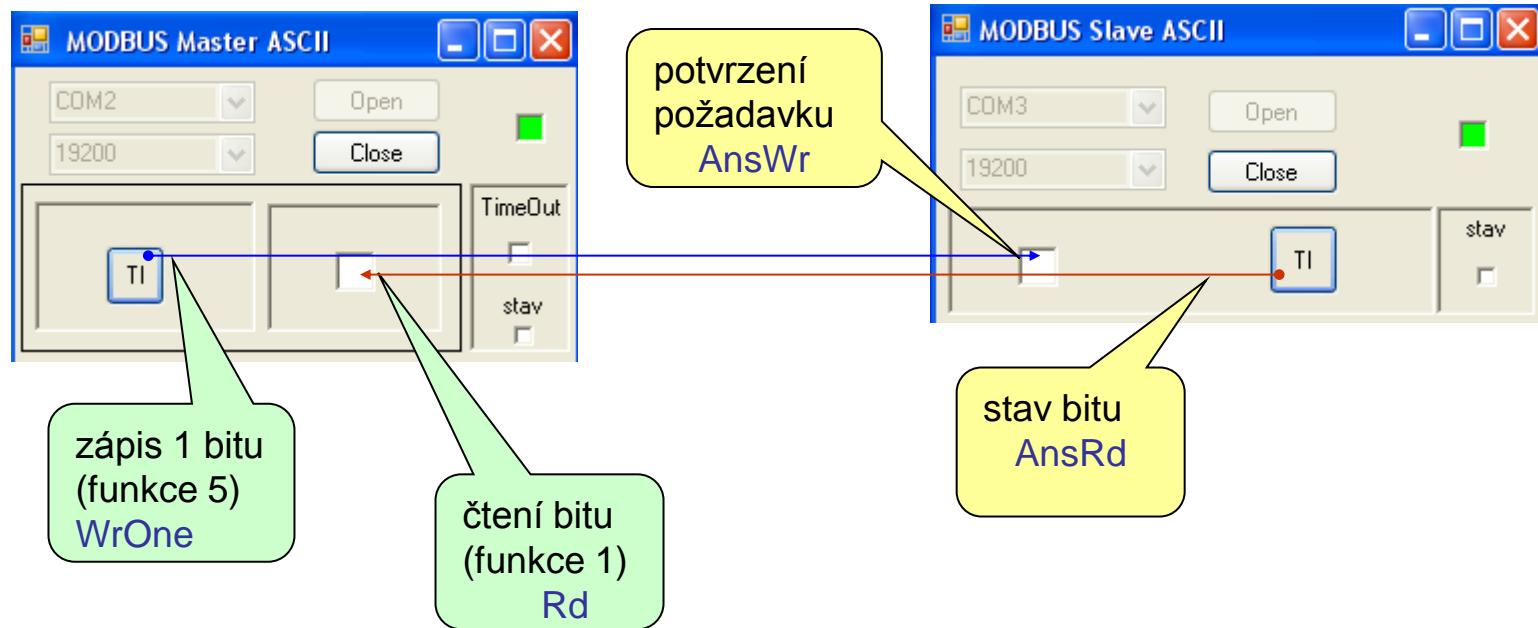
SLAVE (server)

Po příjmu informaci zobrazí a odešle potvrzovací odpověď

Po příjmu požadavku hodnotu bitu odešle



1.část : PC-PC (varianta C#)





Master – implementace na PC (klient)

Konfigurace:

Realizuje funkce (požadavky na server)

- požadavek na zápis jediného bitového stavu – funkční kód 5

metoda WrOne třídy Modbus ASCII s kódem funkce 5 (FCE_WBIT)

- požadavek na čtení bitové hodnoty – funkční kód 1

metoda Rd třídy Modbus ASCII s kodem funkce 1 (FCE_RBIT)

Požadavky odesílat střídavě v pravidelných časových intervalech 200 ms,

jen když je sériový kanál otevřen a Master je ve stavu **klidu**

realizace časovačem intervalu

Implementovat generování čekacího TimeOut intervalu 500 ms na odpověď od Slave,

po vypršení TimeOutu vyčkat 500 ms a vátit se do stavu **klidu**

Zjednodušený příjem odpovědi

příchozí adresu Slave není nutno testovat, pouze správnost LRC

zpracovat jen odpověď na požadavek čtení bitu (FCE_RBIT)

informovat o chybové odpovědi od Slave



Master – vyslání požadavku

Časovač Sample

střídavě každých cca 200 ms vysílá rámec s funcí **1** (čtení bitu) a **5** (zápis bitu)

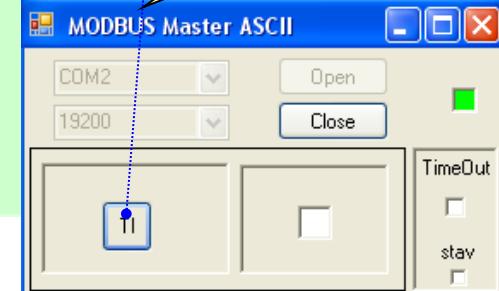


ModbusASCII Ma;
bool prep;
Tstav stav;

Tick
Sample

```
if (comPort.IsOpen && stav == Tstav.stklid)
{
    stav=Tstav.stVysilani;
    prep = !prep;
    if (prep) n=Ma.wrOne(ADR_S,FCE_WBIT,BIT_WR,va1,bfout)
    else n=Ma.Rd(ADR_S,FCE_RBIT,BIT_RD,1,bfout);
    n=Ma.WrByte(Ma.Lrc(bfout,n-1),bfout,n);
    n=Ma.WrEoT(bfout,n);
    comPort.Write(bfout,0,n);
    TimerOut.Interval=500;
    TimerOut.Enabled=true;
    stav=Tstav.stCekani;
}
```

stisk: 0xFF00
jinak: 0





Master – zpracování odpovědi

1. LRC

```
if(Ma.Lrc(bfin,ix-4)!=Ma.RdByte(bfin,ix-3)
{
    .. informace o chybné LRC
}
```

2. adresa
a kód funkce

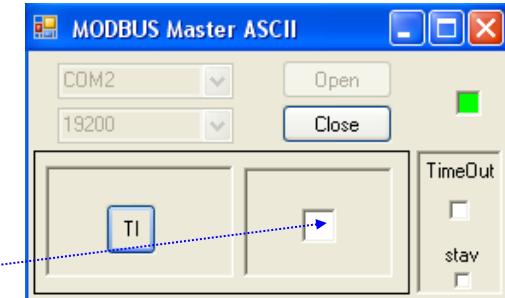
```
adr_r=Ma.RdByte(bfin,1);
kod_r=Ma.RdByte(bfin,3);
```

3. reakce
na odpověď

```
if(kod_r== FCE_RBIT)
{
    pocet=Ma.RdByte(bfin,5);
    val=Ma.RdByte(bfin,7);
    if (adr_r==ADR_S && pocet==1)
        if (val & 1 ==1) { žlutá }
        else { bílá }
}
```

4. informace
o chybě Slavu

```
else if (kod_r>=0x80)
{
    er= Ma.RdByte(bfin,5);
    switch(er) {
        informace o chybě slavu
    }
}
stav=Tstav.stklid;
```





Slave – implementace na PC (server)

Konfigurace:

Přijímá požadavky od klienta a vrací odpovědi

- požadavek na zápis jediného bitového stavu – funkční kód 5, stav indikuje a vrací potvrzení o přijetí požadavku

metoda `AnsWr` třídy `Modbus ASCII` s kódem přijaté funkce

- požadavek na čtení 1 bitové hodnoty – funkční kód 1 a vrací požadovanou hodnotu

metoda `AnsRd` třídy `Modbus ASCII` s kódem přijaté funkce

Kontrolovat přijatý požadavek a vracet chybovou odpověď v případě neimplementované funkce, neexistující adresy registru nebo bitu a hodnoty mimo rozsah

metoda `AnsErr` třídy `Modbus ASCII` s upraveným kódem funkce a typem chyby

Skupinové vysílání ignorovat .



Slave – zpracování požadavku, kontrola položek, příprava odpovědi

1. LRC

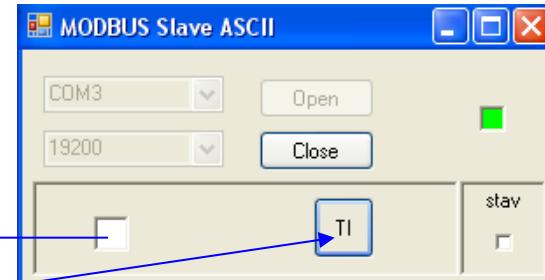
```
if(Ma.Lrc(bfin, ix-4) != Ma.RdByte(bfin, ix-3)
{
    .. možná informace o chybné LRC
}
else {
```

2. adresa

```
adr_r=Ma.RdByte(bfin,1);
if(adr_r == ADR_S)
{
```

3. kód funkce

```
kod_r=Ma.RdByte(bfin,3);
er=0;
switch(kod_r) {
    case FCE_RBIT:
        .
        .
    case FCE_WBIT:
        .
        .
    default: er=1;
}
```



Slave – zpracování požadavku, kontrola položek, příprava odpovědi

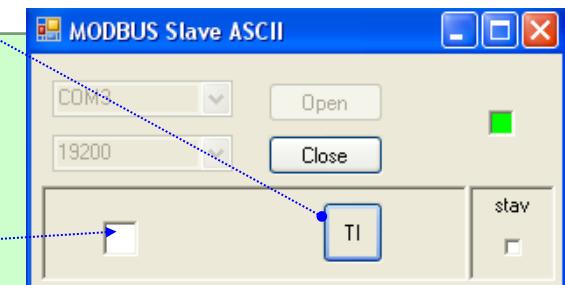
FCE_RBIT:

```
reg=Ma.RdWord(bfin,5);
pocet=Ma.RdWord(bfin,9);
if(reg!=REG_RD || pocet!=1) er=2;
else .. tlačítko -> vals
if(er==0) n= Ma.AnswRD(ADR_S,kod_r,1,vals,bfout);
```

byte []vals = byte[1];

FCE_WBIT:

```
reg=Ma.RdWord(bfin,5);
val=Ma.RdWord(bfin,9);
if(reg!=BIT_WR) er=2;
else switch(val){
    case 0xFF00: .. žlutá ; break;
    case 0x0000: .. bílá ; break;
    default: err=3;
}
if(err==0) n= Ma.AnswWR(ADR_S,kod_r,reg,val,bfout);
```



4. chyba

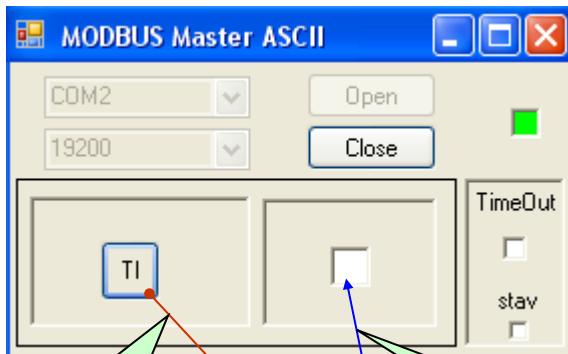
if(er>0) n=Ma.AnswErr(adr_r,(byte)(kod_r|0x80),er,bfout);

5. odeslání
odpovědi

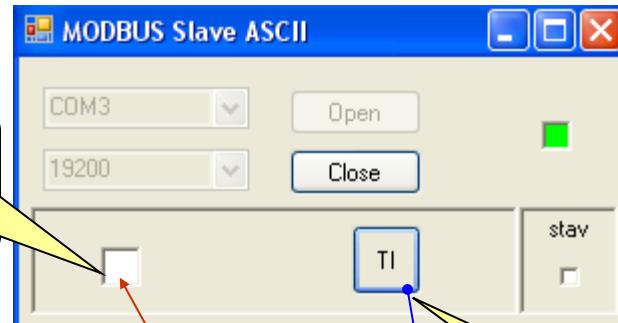
```
n=Ma.WrByte(Ma.Lrc(bfout,n-1),bfout,n);
n=Ma.WrEoT(bfout,n);
comPort.Write(bfout, 0, n);
```



2.část : PC – mikropočítač



potvrzení
požadavku
AnsWr



čtení bitu
(funkce 1)
Rd



potvrzení
požadavku
MbAnsWr

stav tlačítka
MbAnsRd



čtení bitu
(funkce 1)
MbRd

zápis stavu
tlačítka
(funkce 5)
MbWrOne



Master – implementace na mikropočítači (klient)

Konfigurace:

Realizuje funkce (požadavky na server)

- požadavek na zápis jediného bitového stavu – funkční kód 5

[aplikativní funkce MbWrOne s kódem funkce 5 \(FCE_WBIT\)](#)

- požadavek na čtení bitové hodnoty – funkční kód 1

[aplikativní funkce MbRd s kodem funkce 1 \(FCE_RBIT\)](#)

Požadavky odesílat střídavě v pravidelných časových intervalech cca 200 ms,

jen když je sériový kanál otevřen a Master je ve stavu **klidu**

realizace časovačem T0 se základními tiky 30 ms ($30 * 7 = 210$)

Implementovat generování čekacího TimeOut intervalu 500 ms na odpověď od Slave
($30 * 17 = 510$)

Zjednodušený příjem odpovědi

příchozí adresu Slave není nutno testovat, pouze správnost LRC

zpracovat jen odpověď na požadavek čtení registru (FCE_RREG)

informace o chybě Slave: jen omezeně, nebo vůbec



Master – vyslání požadavku

střídavě každých cca 210 ms vysílá rámec s funcí **1** (čtení bitu) a **5** (zápis bitu)

1 5 1 5 1 5

časovač T0

```
if(++cnt_ticks>=N_TICKS && stav==stKlid)
{
    cnt_ticks=0;
    DIR485=1;      /* na vysílání - pro RS485*/
    prep=!prep;
    if(prep) {val= ... ;
        itx=MbWrOne(ADR_S,FCE_WBIT,BIT_WR,val,bfout);}
    else itx=MbRd(ADR_S,FCE_RREG,REG_RD,1,bfout);
    itx+=MbWrByte(MbLrc(bfout+1,itx-1),bfout+itx);
    itx+=MbWrEoT(bfout+itx);
    SendBuf(bfout,itx);
    stav=stCekani;
    DIR485=0;      /* zpět na příjem - pro RS485 */
}
```

```
#define N_TICKS 7
#define TIMEOUT 17
bit prep;
```



TimeOut

```
if(cnt_ticks>=TIMEOUT)
{
    cnt_ticks=0;
    LED_R=!LED_R; // signalizace TimeOutu
    stav=stKlid;
}
```



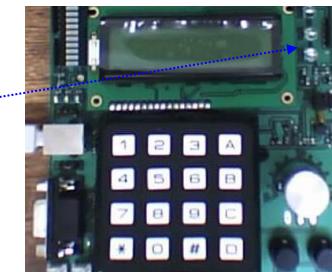
Master – zpracování odpovědi

1. LRC

```
if(MbLrc(bfin+1,ix-4)==(lrc=MbRdByte(bfin+ix-3)))
{
```

2. kód funkce a reakce na odpověď

```
if( (kod_r=MbRdByte(bfin+3))==FCE_RBIT)
{
    pocet=MbRdByte(bfin+5);
    val=MbRdByte(bfin+7);
    if(val & 1) .. ; // LED svítí
    else .. ; // LED nesvítí
}
```





Slave – implementace na mikropočítači (server)

Konfigurace:

Přijímá požadavky od klienta a vrací odpovědi

- požadavek na zápis jediného bitového stavu – funkční kód 5, stav indikuje a vrací potvrzení o přijetí požadavku

aplikáční funkce MbAnsWr s kódem přijaté funkce

- požadavek na čtení bitové hodnoty – funkční kód 1 a vrací stav tlačítka

aplikáční funkce MbAnsRd s kódem přijaté funkce

Kontrolovat přijatý požadavek a vracet chybovou odpověď v případě neimplementované funkce, neexistující adresy registru nebo bitu a hodnoty mimo rozsah

aplikáční funkce MbAnsErr s upraveným kódem funkce a typem chyby

Skupinové vysílání ignorovat .



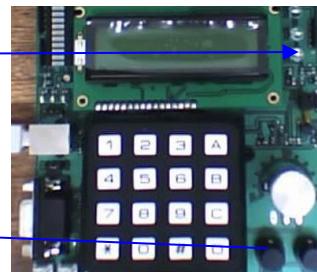
Slave – zpracování požadavku, kontrola položek, příprava odpovědi

1. LRC
a adresa

```
if((MbLrc(bfin+1,ix-4)==(lrc=MbRdByte(bfin+ix-3)))  
    && (MbRdByte(bfin+1)==ADR_S))  
{
```

2. kód
funkce

```
kod_r=MbRdByte(bfin+3);  
er=0;  
switch(kod_r)  
{  
    case FCE_WBIT:  
        .  
        .  
    case FCE_RBIT:  
        .  
        .  
    default: er=1;  
}
```

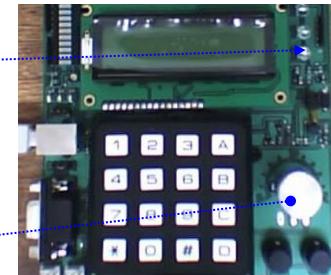




Slave – zpracování požadavku, kontrola položek, příprava odpovědi

FCE_WBIT:

```
if((reg=MbRdword(bfin+5))!=BIT_WR) er=2;
else if((val=MbRdword(bfin+9))!=0&&val!=0xFF00) er=3;
else LED_G . .
if(er==0)itx=MbAnsWr(ADR_S,kod_r,reg,val,bfout);
break;
```



FCE_RBIT:

```
if((reg=MbRdword(bfin+5))==BIT_RD&&(pocet=MbRdword(bfin+9))==1)
{
    bity[0]= . . ;
    itx=MbAnsRd(ADR_S,kod_r,1,bity,bfout);
}
else er=2;
break;
```

byte bity[1];

4. chyba

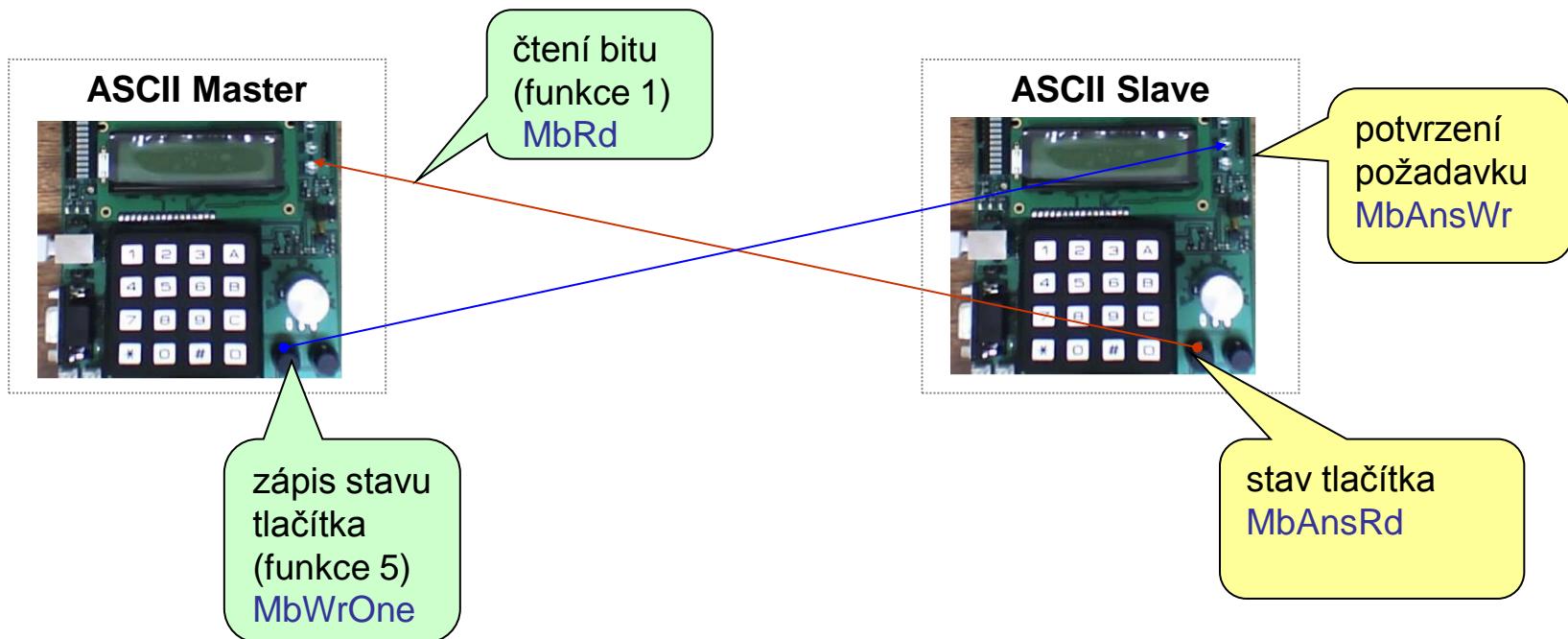
```
if(er) itx=MbAnsErr(adr_r,kod_r|0x80,er,bfout);
```

odeslání
odpovědi

```
DIR485=1; /* na vysílání */
itx+=MbWrByte(MbLrc(bfout+1,itx-1),bfout+itx);
itx+=MbWrEoT(bfout+itx);
SendBuf(bfout,itx);
DIR485=0; /* zpět na příjem */
```



3.část : mikropočítač – mikropočítač





MINISTERSTVO ŠKOLSTVÍ,
MLÁDEŽE A TĚLOVÝCHOVY



ŘPS – úlohy MODBUS RTU

Reflexe požadavků průmyslu na výuku v oblasti automatického řízení a měření

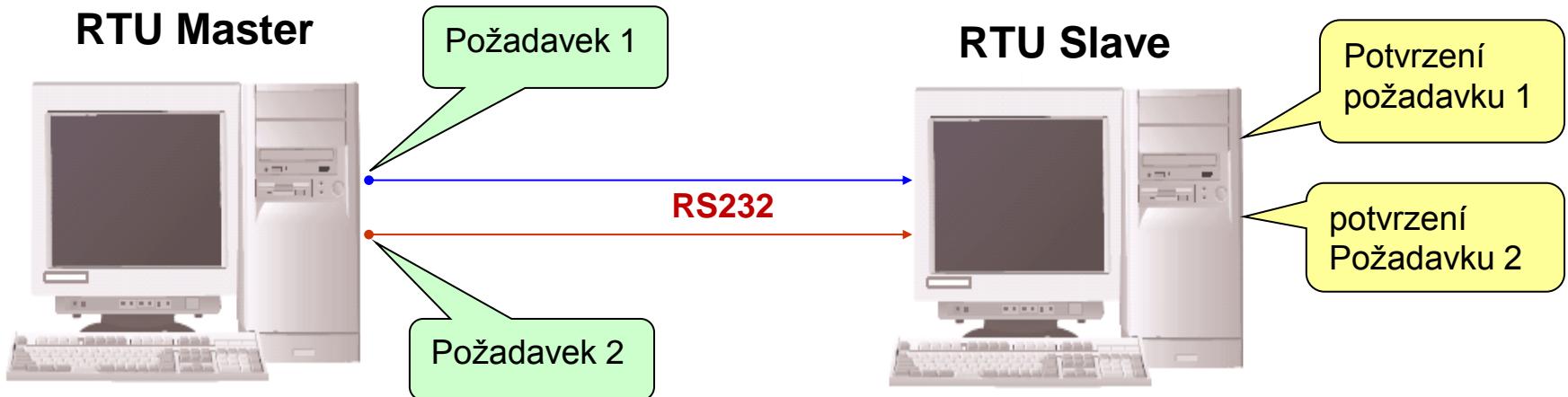
MODBUS RTU

Úlohy MR1, MR2, MR3, MR4, MR5, MR6

MR1M, MR1S, MR2M, MR2S, MR3M, MR3S, MR4M, MR4S, MR5M, MR5S, MR6M, MR6S



PC-PC (varianta C#)





Podpora pro PC

Class lib **Modbus.dll** - zdrojový kód **Modbus.cs**

```
namespace Modbus;
```

```
class ModbusRTU
```

```
// metody pro Modbus RTU
```

```
ushort Rdword(byte[] bf,int n);  
int Wrword(ushort val,byte[] bf,int n);
```

```
ushort Crc(byte[] bf,int len);  
int WrCrc(ushort crc,byte[] bf,int n);  
ushort RdCrc(byte[] bf,int n);
```

```
int WrOne(byte adr,byte fce,ushort reg,ushort val,byte[] bf);  
int Rd(byte adr,byte fce,ushort reg,ushort nbr,byte[] bf);  
int Wr(byte adr,byte fce,ushort reg,int nbr,byte[] vals,byte[] bf);
```

```
int AnsRd(byte adr,byte fce,int bytes,byte[] vals,byte[] bf);  
int AnsWr(byte adr,byte fce,ushort reg,ushort val,byte[] bf);  
int AnsErr(byte adr,byte fce,byte err,byte[] bf);
```



Užité metody třídy ModbusRTU v aplikaci z Modbus.dll

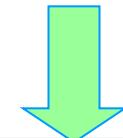
aplikační	pomocné
WrOne	RdWord
Rd	WrWord
AnsWr	Crc
AnsRd	WrCrc
AnsErr	RdCrc

Poznámka: v hlavním programu v sekci **using** přidat **Modbus**

úloha	str.
MR1	169
MR2	183
MR3	203
MR4	221
MR5	238
MR6	255



adr. slavu, kód funkce, adr. registrů a bitů, hodnoty, CRC



bfout

→ RS232

bfin

← RS232

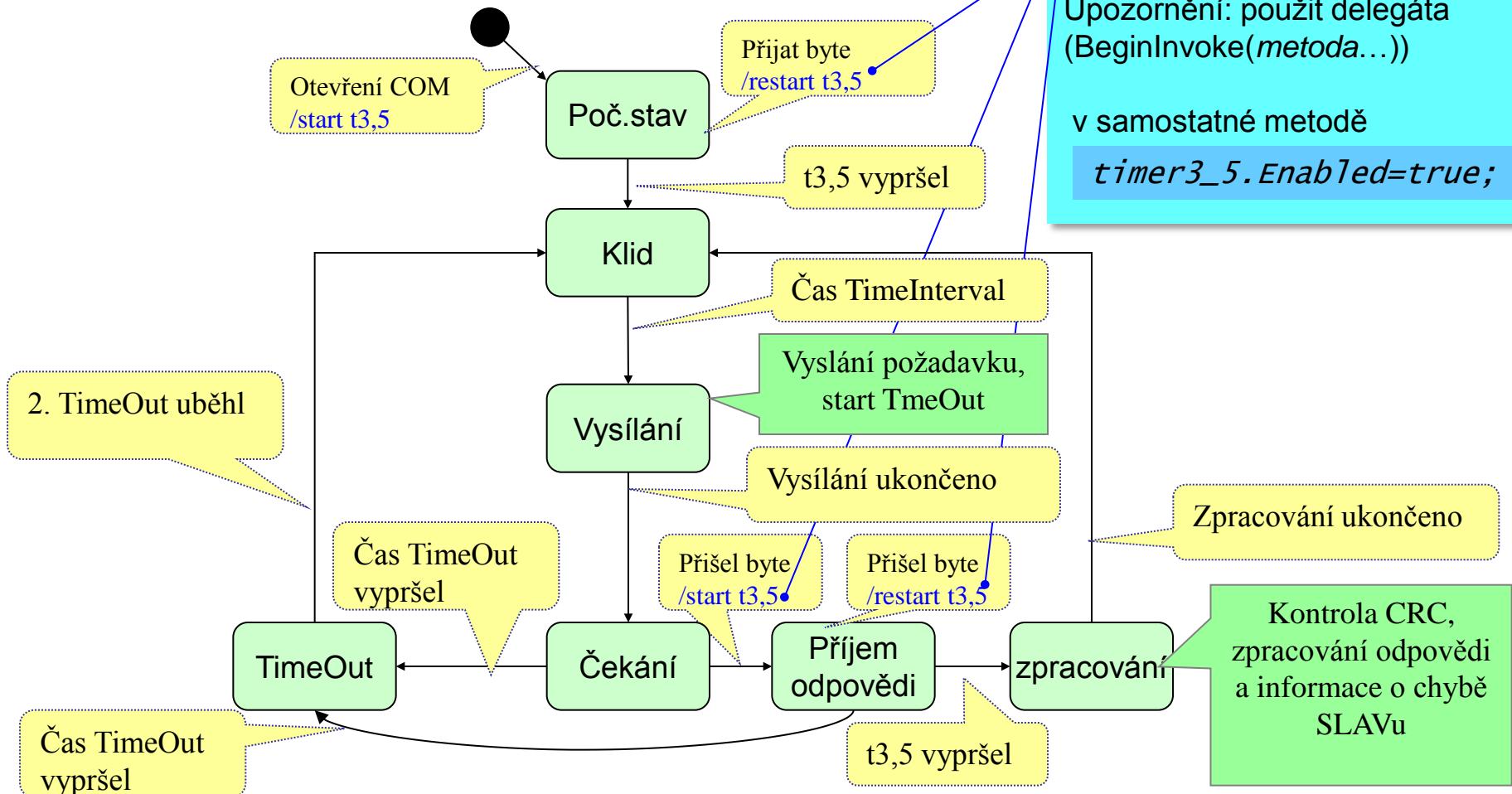
```
byte []bfin = new byte[256];  
byte []bfout = new byte[256];
```

bfout[0] **adresa slavu**
bfout[1] **kód funkce**

.

.

Master – zjednodušený stavový diagram





Master – počáteční stav

Click
(Open)

```
stav = Tstav.stPocatek;
Timer3_5.Enabled=true;
```

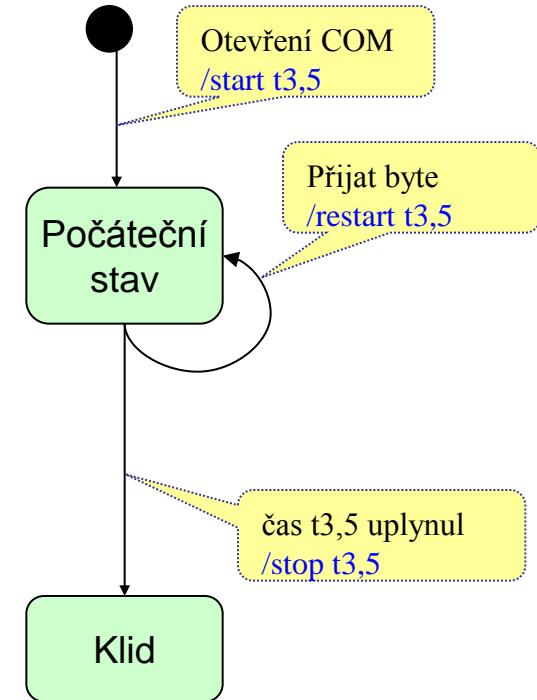
DataReceived

```
while(comPort.BytesToRead > 0){
    byte b = (byte)comPort.ReadByte();
    switch(stav){
        case Tstav.stPocatek: break;
        case Tstav.stCekani:
            .
        case Tstav.stPrijem:
            .
    }
    BeginInvoke(metoda ...);
    timer3_5.Enabled=true;
```

Tick
3_5

```
Timer3_5.Enabled=false;
switch(stav){
    case Tstav.stPocatek: stav=Tstav.stKlid;
    break;
```

Tstav stav;





Master – vyslání požadavku

Časovač Sample

střídavě každých cca 200 ms vysílá rámec s funcí **f1** a **f2**



ModbusRTU Mr;
bool prep;
Tstav stav;

Tick
Sample

```
if(comPort.IsOpen && stav == Tstav.stKlid)
{
```

```
    stav=Tstav.stVysilani;
```

```
    prep= !prep;
```

```
    if(prep) n=Mr.WrOne(ADR_S,...);
```

```
    else n=Mr.Rd(ADR_S,...);
```

n=Mr.Rd(ADR_S,...);

n=Mr.WrOne(ADR_S,...);

```
    n = Mr.WrCrc(Mr.Crc(BufferOut, n), BufferOut, n);
```

```
    comPort.Write(bfout,0,n);
```

```
    TimerOut.Interval=500;
```

```
    TimerOut.Enabled=true;
```

vyslání

aktivace TimeOutu

Stav čekání

```
    stav=Tstav.stCekani;
```

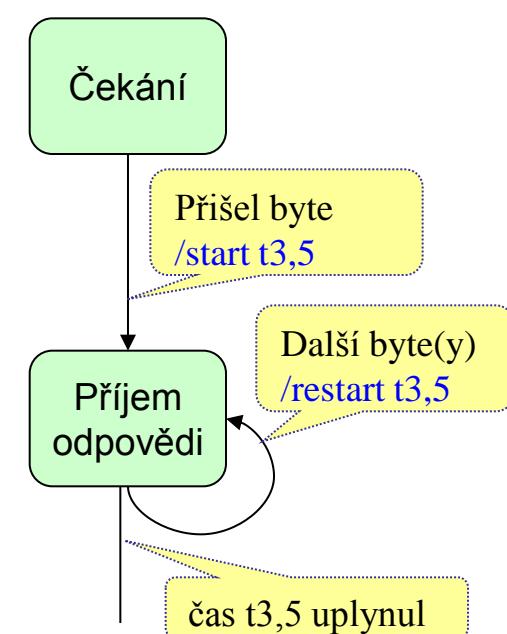


Master – příjem odpovědi

DataReceived

```
Tstav.stCekani:
    stav=Tstav.stPrijem;
    bfin[0]=b;
    ix=0;
    break;

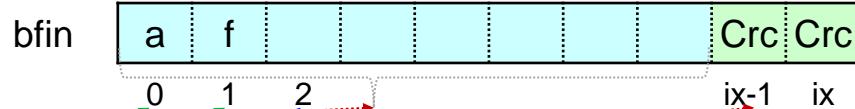
Tstav.stPrijem:
    ix++;
    bfin[ix]=b;
    break;
```



Tick
3_5

```
Timer3_5.Enabled=false;
switch(stav){
    case Tstav.stPocatek: stav=Tstav.stKlid;
                           break;
    case Tstav.stPrijem:
        . // zpracování odpovědi
```

Master – zpracování odpovědi



1. CRC

```
if (Mr.Crc(bfin,ix-1)!=Mr.RdCrc(bfin,ix-1)) {
    .. možná informace o chybné CRC
}
else {
```

2. adresa a kód funkce

```
adr_r=bfin[0];
kod_r=bfin[1];
```

3. reakce na odpověď

```
switch(kod_r){
    case f1:
        .
        break;
    case f2:
        .
        break;
    default: if(kod_r>=0x80)
    {
        er= bfin[2];
        switch(er){
            .
            informace o chybě stavu
        }
    } break;
}
stav=Tstav.stklid;
```

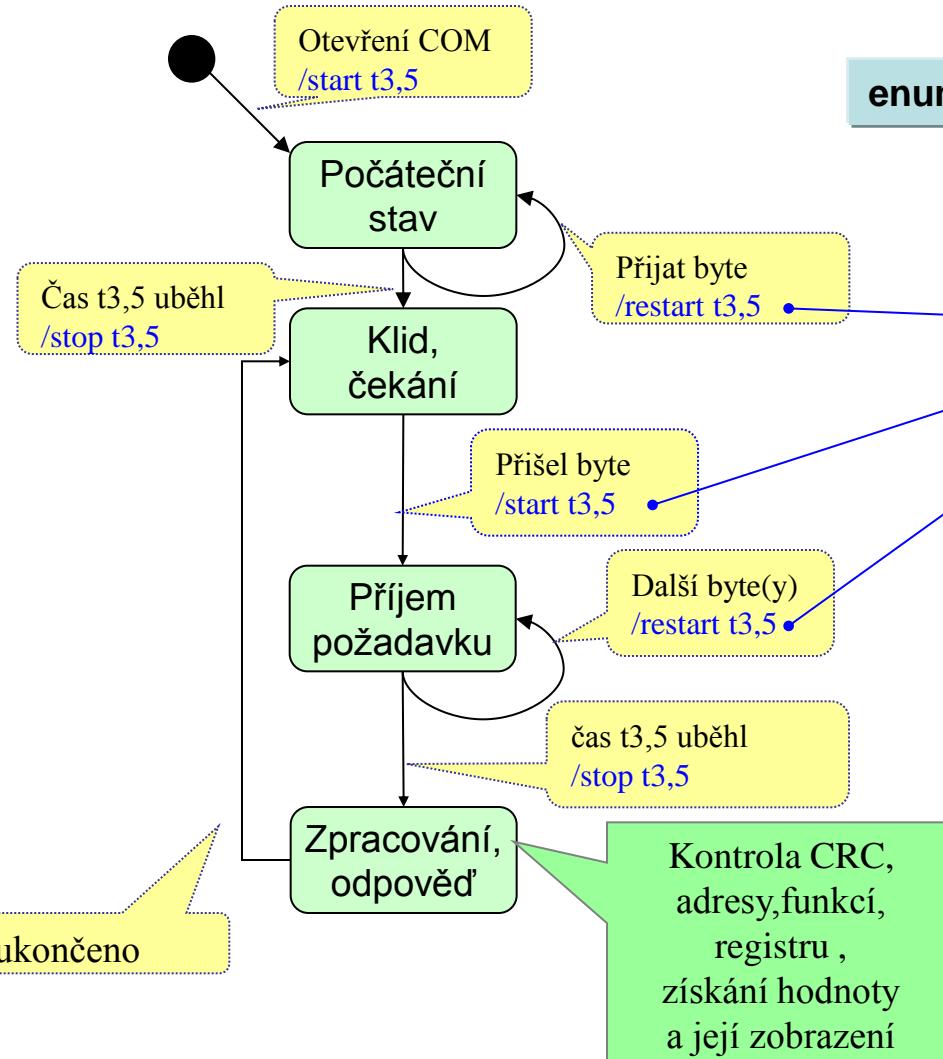
4. informace o chybě Slavu

úloha	str.
MR1	169
MR2	183
MR3	203
MR4	221
MR5	238
MR6	255

5. klidový stav



Slave – zjednodušený stavový diagram





Slave – počáteční stav

Click
(Open)

```
stav = Tstav.stPocatek;
Timer3_5.Enabled=true;
```

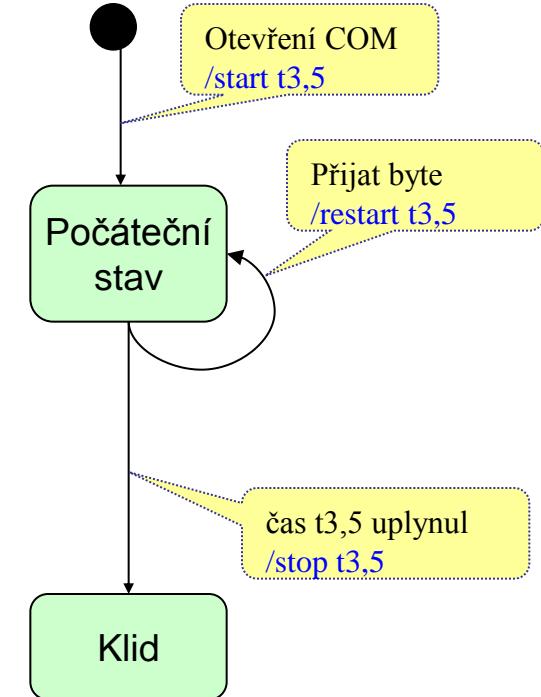
DataReceived

```
while(comPort.BytesToRead > 0){
    byte b = (byte)comPort.ReadByte();
    switch(stav){
        case Tstav.stPocatek:break;
        case Tstav.stKlid:
            .
            .
            .
        case Tstav.stPrijem:
            .
            .
            .
    }
    BeginInvoke(metoda...);
    timer3_5.Enabled=true;
```

Tick
3_5

```
Timer3_5.Enabled=false;
switch(stav){
    case Tstav.stPocatek: stav=Tstav.stKlid;
                            break;
    case Tstav.stPrijem:
        . // zpracování odpovědi
```

Tstav stav;

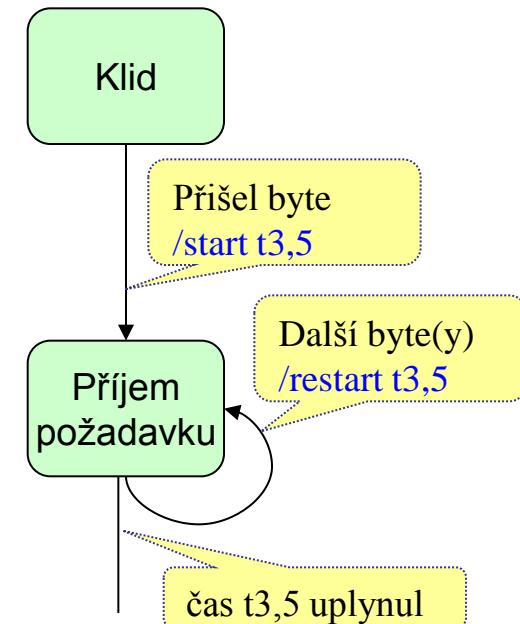




Slave – příjem požadavku

DataReceived

```
Tstav.stKlid:  
    stav=Tstav.stPrijem;  
    bfin[0]=b;  
    ix=0;  
    break;  
Tstav.stPrijem:  
    bfin[++ix]=b;  
    break;
```



Tick
3_5

```
case Tstav.stPrijem:  
    . // zpracování požadavku
```



Slave – zpracování požadavku

1. CRC

```
if(Mr.Crc(bfin,ix-1)!=Mr.RdCrc(bfin,ix-1) {  
    .. možná informace o chybné CRC  
}  
else {
```

2. adresa

```
adr_r=bfin[0];  
if(adr_r == ADR_S){
```

```
kod_r=bfin[1];
```

3. kód funkce

adresy objektů
a hodnoty

```
.. zpracování požadavku podle funkce a příprava odpovědi  
reg=Mr.RdWord(bfin,2);  
val=Mr.RdWord(bfin,4);  
.
```

kontrola požadavku

- funkce
- adresa objektu
- hodnoty

er=0;

Požadovaná funkce není ve Slavu implementována: **er=1;**
Požadovaná adresa objektu ve Slavu mimo rozsah: **er=2;**
Zapisovaná data do objektu ve Slavu mimo rozsah: **er=3;**





Slave – vyslání odpovědi

```
stav=Tstav.stVysilani;
```

```
if(er==0) n = Mr.AnsRd(ADR_S,kod_r,pocet,vals,bfout);
```

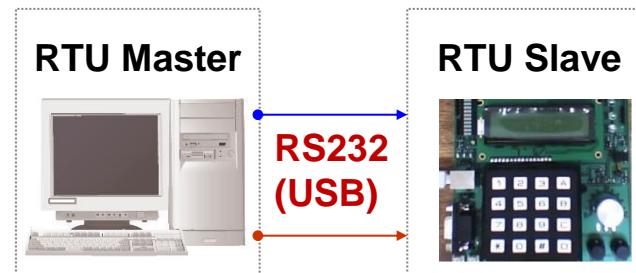
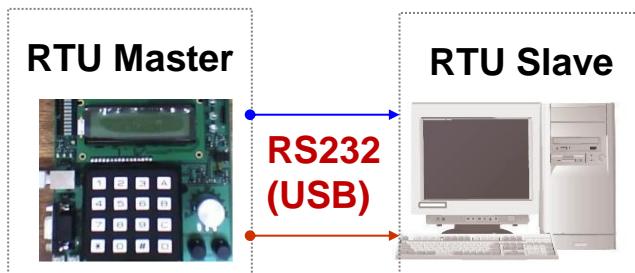
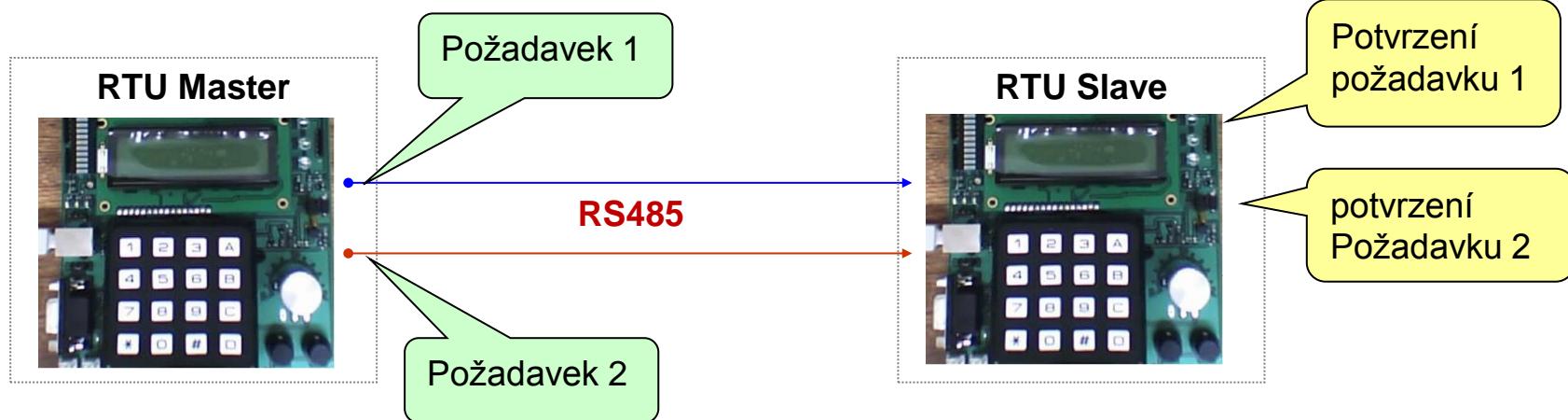
```
n = Mr.AnsWr(ADR_S,kod_r,reg,val,bfout);
```

```
if(er > 0) n=Mr.AnsErr(adr_r,(byte)(kod_r|0x80),er,bfout);
```

```
n = Mr.WrCrc(Mr.Crc(bfout,n),bfout,n);  
comPort.write(bfout,0,n);
```

```
stav=Tstav.stKlid;
```

mikropočítač – mikropočítač (PC – mikropočítač)





Podpora pro mikropočítáč prototypy fukcí **Modbus.H** – zdrojový kód **Modbus.C**

```
byte WrWord(word val,byte *bf);
word RdWord(byte *bf);
word MrtuRdCrc(byte *bf);
byte MrtuWrCrc(word crc,byte *bf );

byte MrtuWr(byte adr,byte fce,word reg,word nbr,byte *vals,byte *bf);
byte MrtuWrOne(byte adr,byte fce,word reg,word val,byte *bf);
byte MrtuRd(byte adr,byte fce,word reg,word val,byte *bf);

byte MrtuAnsErr(byte adr,byte fce,byte er,byte *bf);
byte MrtuAnsRd(byte adr,byte fce,byte reg,byte *vals,byte *bf);
byte MrtuAnsWr(byte adr,byte fce,word reg,word val,byte *bf);

word MrtuCrc(byte *bf, byte len);
```



Užité funkce v aplikaci ze souboru Modbus.C

aplikační	pomocné
MrtuWrOne	RdWord
MrtuRd	WrWord
MrtuAnsWr	MrtuCrc
MrtuAnsRd	MrtuWrCrc
MrtuAnsErr	MrtuRdCrc
Poznámka: v hlavním programu	#include "Modbus.H"

úloha	str.
MR1	169
MR2	183
MR3	203
MR4	221
MR5	238
MR6	255

adr. slavu, kód funkce, adr. registrů a bitů, hodnoty, CRC



bfout

→ RS232

bfin

← RS232

```
//globální
xbyte bfin[256],bfout[256];
```

bfout[0] **adresa slavu**

bfout[1] **kód funkce**

.

.

funkce pro vyslání zprávy:
-bf: pointer na pole znaků
-len: počet bytů k vyslání

```
void sendBuf(byte *bf,byte len)
{
    byte byteOut=*bf++;
    TI=0;
    SBUF=byteOut;
    while(--len)
    {
        byteOut=*bf++;
        while(!TI);
        SBUF=byteOut;
        TI=0;
    }
}
```



Časový interval 3,5 znaku – generování časovačem T1 v režimu 1

Formát UART: 8,N,2 → 11 bitů , $f_{bit} = 19200 \text{ bit/s} \rightarrow t_{bit} = 1/f_{bit}$

$$t_{3,5} = 3,5 \cdot 11 \cdot t_{bit} \approx 2 \text{ ms tik}$$

pro časovač T1 : $t_{3,5} = N3_5 \cdot 12/f_{osc}$

pro sériový kanál řízený časovačem T2 je $t_{bit} = 32 \cdot NBIT/f_{osc}$

$$t_{3,5} = 3,5 \cdot 11 \cdot 32 \cdot NBIT/fosc = N3_5 \cdot 12/fosc$$

$$N3_5 = NBIT \cdot 109 \rightarrow \#define N3_5 109*NBIT$$

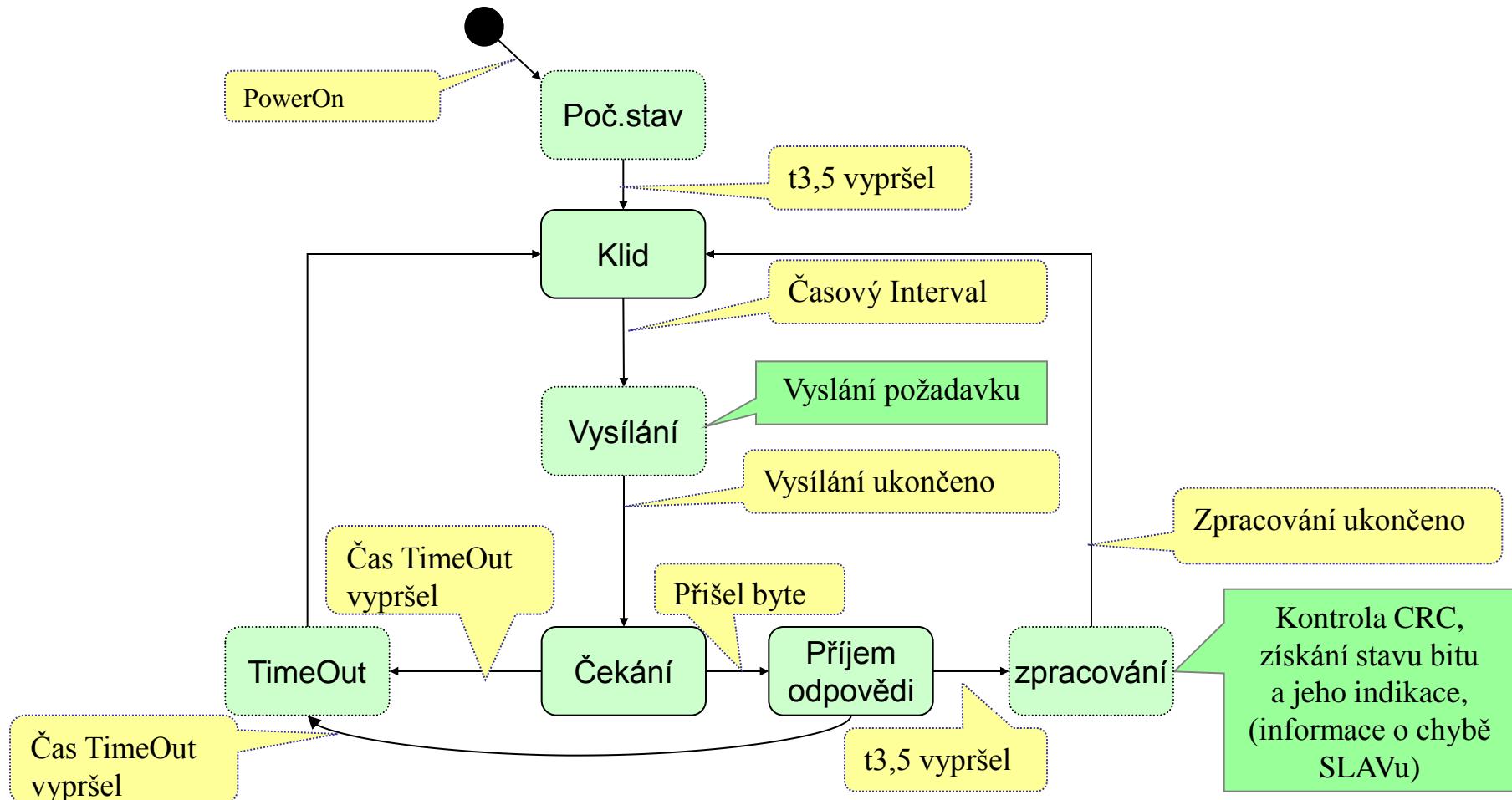
(re)start t3,5

```
TH1=(word)(-N3_5) >> 8;  
TL1=(byte)(-N3_5);  
TF1=0;  
TR1=1;
```

čas uplynul: přerušení nebo
1 → TF1

Poznámka: oba časovače T0 a T1 budou nastaveny v režimu 1: TMOD = 0x11;

Master – zjednodušený stavový diagram



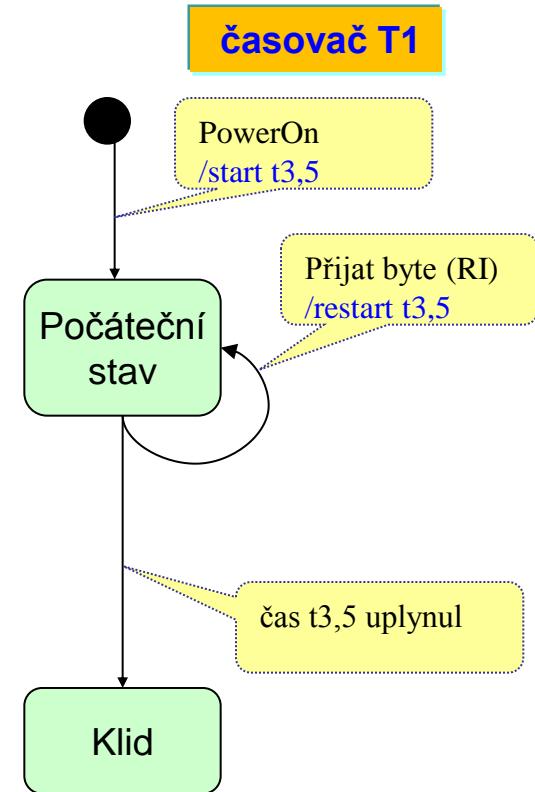


Master – počáteční stav

```

void main(void)
{
    .. // inicializace
    DIR485=0; /* na příjem */
    do
    {
        RI=0;
        TH1=(word)(-N3_5) >> 8;
        TL1=(byte)(-N3_5) ;
        TR1=1;
        while(!TF1);
        TF1=0;
        TR1=0;
    } while(RI);
    stav=stKlid;
    while(1)
    {
        .
        .
        .
        .
    }
}

```





Master – vyslání požadavku

střídavě každých cca 210 ms vysílá rámec s funcí **f1** a **f2**

časovač T0



```
if(++cnt_ticks>=N_TICKS && stav==stKlid)
{
    cnt_ticks=0;
    DIR485=1; /* na vysílání */
    prep=!prep;
    if(prep)itx=MrtuWrOne(ADR_S,...);
    else itx=MrtuRd(ADR_S,...);
    itx+=MrtuWrCrc(MrtuCrc(bfout,itx),bfout+itx);
    SendBuf(bfout,itx);
    stav=stCekani;
    DIR485=0; /* zpět na příjem */
}
```

```
#define N_TICKS 7
#define TIMEOUT 17
bit prep;
```

```
itx=MrtuRd(ADR_S,...)
itx=MrtuWrOne(ADR_S,...)
```

TimeOut

```
if(cnt_ticks>=TIMEOUT)
{
    cnt_ticks=0;
    LED_R=!LED_R; // signalizace TimeOutu
    stav=stKlid;
}
```

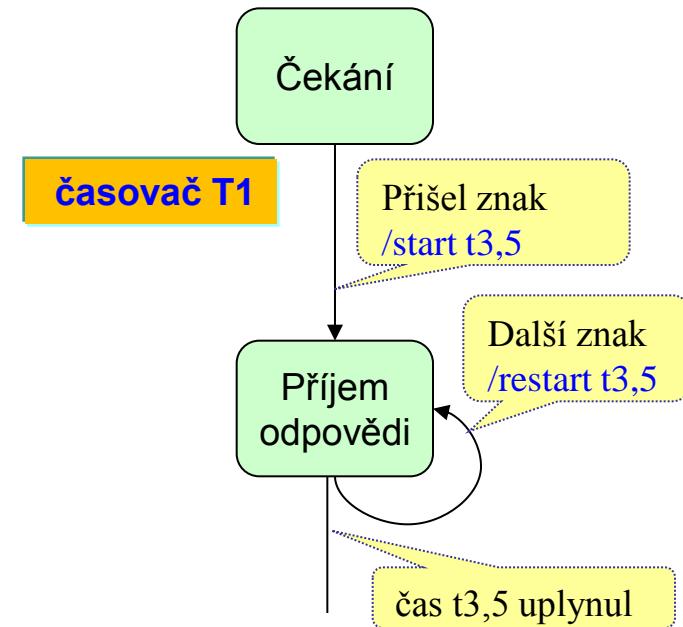


Master – příjem odpovědi

```

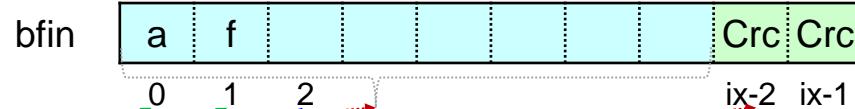
if (RI)
{
    byteIn=SBUF;
    RI=0;
    switch(stav)
    {
        case stCekani:
            ix=0;
            bfin[ix++]=byteIn;
            stav=stPrijem;
            TH1=(word)(-N3_5) >> 8;
            TL1=(byte)(-N3_5) ;
            TF1=0;
            TR1=1;
            break;
        case stPrijem:
            bfin[ix++]=byteIn;
            TH1=(word)(-N3_5) >> 8;
            TL1=(byte)(-N3_5) ;
            TF1=0;
            break;
    }
    if(TF1){ TR1=0;
}

```





Master – zpracování odpovědi



1. CRC

```
if(MrtuCrc(bfin,ix-2)!=MrtuRdcrc(bfin+ix-2)) {
    .. možná indikace chyby CRC
}
else {
```

2. adresa
a kód funkce

```
adr_r=bfin[0];
kod_r=bfin[1];
```

3. reakce
na odpověď

```
switch(kod_r){
    case f1:
        .
        break;
    case f2:
        .
        break;
    default: if(kod_r>=0x80)
    {
        er= bfin[2];
        switch(er){
            informace o chybě stavu
        }
    } break;
}
stav=stklid;
```

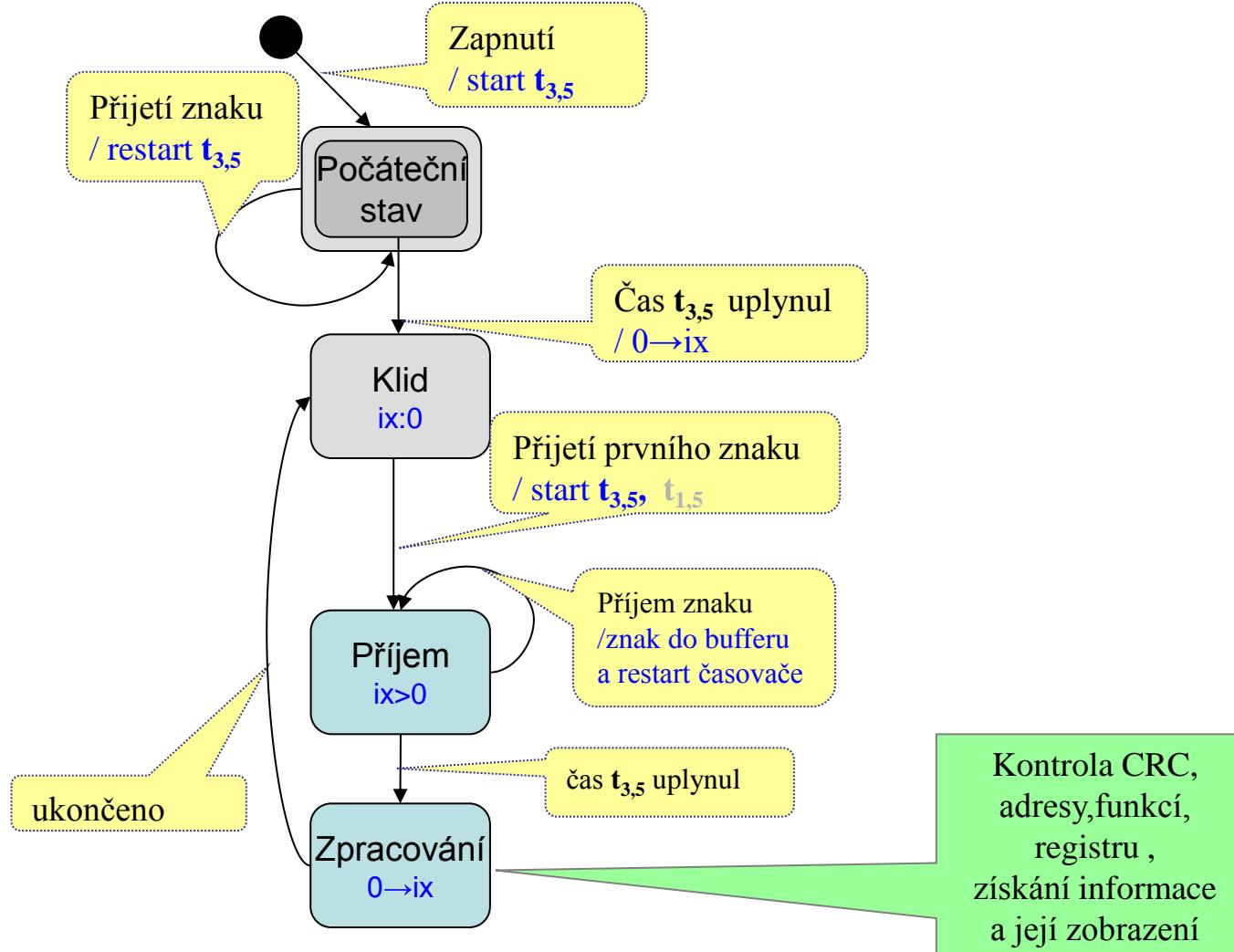
4. informace
o chybě Slavu

5. klidový stav

úloha	str.
MR1	169
MR2	183
MR3	203
MR4	221
MR5	238
MR6	255



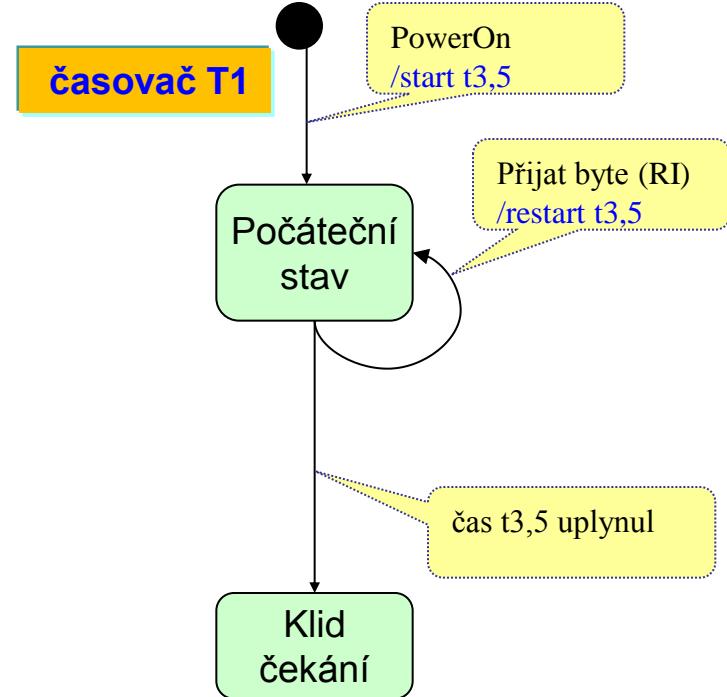
Slave – zjednodušený stavový diagram





Slave – počáteční stav

```
void main(void)
{
    .. // inicializace
    DIR485=0; /* na příjem */
    do
    {
        RI=0;
        TH1=(word) (-N3_5) >> 8;
        TL1=(byte) (-N3_5) ;
        TR1=1;
        while(!TF1);
        TF1=0;
        TR1=0;
    } while(RI);
    ix=0; // klidový stav
    while(1)
    {
        ..
    }
}
```





Slave – příjem požadavku

```
if (RI)
{
    bfin[ix++]=SBUF;
    RI=0;
    TH1=(word)(-N3_5) >> 8;
    TL1=(byte)(-N3_5) ;
    TF1=0;
    TR1=1;
}
```

```
if(TF1)
{
    TR1=0;
    . //zpracování požadavku

    .
    ix=0; // zpět do klidového stavu
}
```

časovač T1

Čekání
Příjem
odpovědi

Přišel znak
/(re)start t3,5

čas t3,5 uplynul
/stop t3,5



Slave – zpracování požadavku

1. CRC
a adresa

2. kód funkce

adresy objektů
a hodnoty

kontrola požadavku
- funkce
- adresa objektu
- hodnoty



```
if((MrtuCrc(bfin,ix-2)!=MrtuRdCrc(bfin+ix-2)) ||
   (bfin[0]!=ADR_S)){
    .. zpráva nepatří slavu, nebo chybná CRC - ignorování
}
else {
```

kod_r=bfin[1];

.. zpracování požadavku podle funkce a příprava odpovědi
 .
 reg=Rdword(bfin+2);
 val=Rdword(bfin+4);
 .

er=0;

Požadovaná funkce není ve Slavu implementována: **er=1;**
 Požadovaná adresa objektu ve Slavu mimo rozsah: **er=2;**
 Zapisovaná data do objektu ve Slavu mimo rozsah: **er=3;**



Slave – vyslání odpovědi

```
if(er==0)    itx=MrtuAnsRd(ADR_S,kod_r,pocet,vals,bufout);
```

```
        itx=MrtuAnsWr(ADR_S,kod_r,reg,val,bfout);
```

```
if(er)itx=MrtuAnsErr(adr_r,kod_r|0x80,er,bfout);
```

```
DIR485=1; /* na vysílání */  
itx+=MrtuWrCrc(MrtuCrc(bfout,itx),bfout+itx);  
SendBuf(bfout,itx);  
DIR485=0; /* zpět na příjem */
```

úloha	str.
MR1	169
MR2	183
MR3	203
MR4	221
MR5	238
MR6	255



Řídicí počítačové systémy

Úloha pro samostatná cvičení - MR1

Implementace protokolu MODBUS RTU na PC a mikropočítačích řady '51 pro uzly Master (Klient) i Slave (Server).

Požadované implementované funkce:

- zápis jediného vnitřního registru (Holding) do uzlu Slave,
- čtení bitového stavu (Coil) z uzlu Slave.

Rozhraní: RS232, standardní rámec 8,N,2

1. část: propojení PC – PC (C# MSVS)

2. část: propojení PC – mikropočítač

Rozhraní: RS485, standardní rámec 8,N,2

3. část: propojení mikropočítač – mikropočítač

Funkce pro podporu aplikace protokolu MODBUS:

v souboru Modbus.dll a Modbus.cs pro PC (C#),

v souboru Modbus.H a Modbus.C pro mikropočítač



MASTER (klient)

V pravidelných časových intervalech generuje 16 bitovou hodnotu a předává požadavek na zápis do uzlu SLAVE

V pravidelných časových intervalech generuje požadavek na čtení bitové informace z uzlu SLAVE a zobrazuje ji

kód fukce: 06
+ data

potvrzení

kód fukce: 01

data

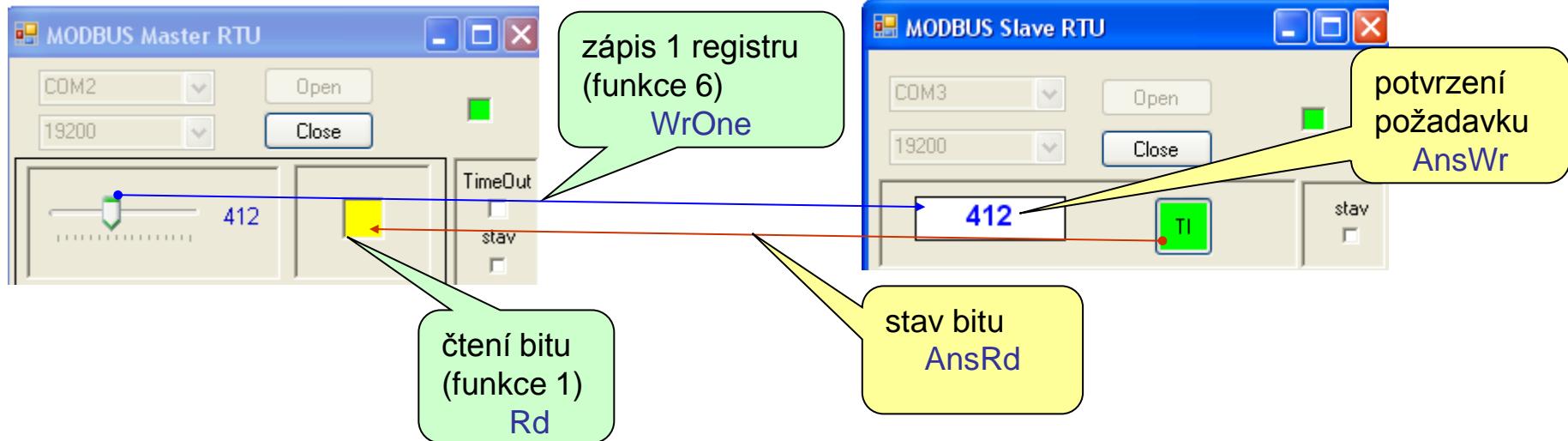
SLAVE (server)

Po příjmu hodnotu zobrazí a odešle potvrzovací odpověď

Po příjmu požadavku hodnotu bitu odešle



1.část : PC-PC (varianta C#)





Master – implementace na PC (klient)

Konfigurace:

Realizuje funkce (požadavky na server)

- požadavek na zápis jediného vnitřního registru (hodnota 0 až 1023) – funkční kód 6

metoda WrOne třídy ModbusRTU s kódem funkce 6 (FCE_WREG)

- požadavek na čtení bitové hodnoty – funkční kód 1

metoda Rd třídy ModbusRTU s kodem funkce 1 (FCE_RBIT)

Požadavky odesílat střídavě v pravidelných časových intervalech 200 ms,

jen když je sériový kanál otevřen a Master je ve stavu **klidu**

realizace časovačem intervalu

Implementovat generování čekacího TimeOut intervalu 500 ms na odpověď od Slave,
po vypršení TimeOutu vyčkat 500 ms a vátit se do stavu **klidu**

Zjednodušený příjem odpovědi

příchozí adresu Slave není nutno testovat, pouze správnost CRC

zpracovat jen odpověď na požadavek čtení bitu (FCE_RBIT)

informovat o chybové odpovědi od Slave

Implementace generování intervalu 3,5 znaku pro ukončení příjmu zprávy

Omezená (žádná) implementace generování intervalu 1,5 znaku



Master – vyslání požadavku

Časovač Sample

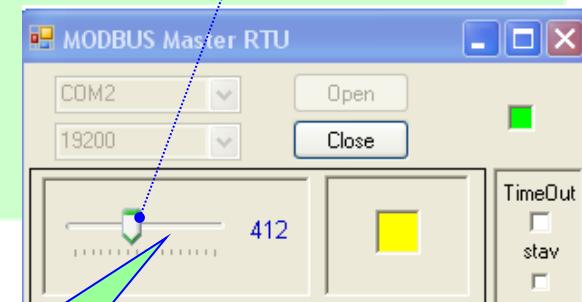
střídavě každých cca 200 ms vysílá rámec s funcí **1** (čtení bitu) a **6** (zápis registru)



ModbusRTU Mr;
bool prep;
Tstav stav;

Tick
Sample

```
if (comPort.IsOpen && stav == Tstav.stKlid)
{
    stav=Tstav.stVysilani;
    prep = !prep;
    if (prep) n=Mr.wrOne(ADR_S,FCE_WREG,REG_WR,pot,bfout)
    else n=Mr.Rd(ADR_S,FCE_RBIT,BIT_RD,1,bfout);
    n=Mr.WrCrc(Mr.Crc(bfout,n),bfout,n);
    comPort.Write(bfout,0,n);
    TimerOut.Interval=500;
    TimerOut.Enabled=true;
    stav=Tstav.stCekani;
}
```



0 až 1023



Master – zpracování odpovědi

1. CRC

```
if(Mr.Crc(bfin, ix-1) !=Mr.RdCrc(bfin, ix-1)
{
    .. možná informace o chybné CRC
}
else {
```

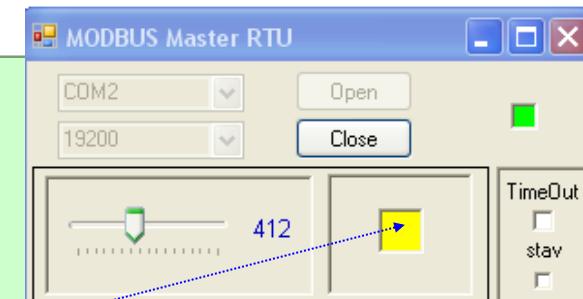
2. adresa
a kód funkce

```
adr_r=bfin[0];
kod_r=bfin[1];
```

3. reakce
na odpověď

```
if(kod_r== FCE_RBIT)
{
    pocet=bfin[2];
    val=bfin[3];
    if (adr_r==ADR_S && pocet==1)
        if (val & 1 ==1) { žlutá }
        else { bílá }
    }
else if (kod_r>=0x80)
{
    er= bfin[2];
    switch(er) {
        informace o chybě slavu
    }
}
stav=Tstav.stklid;
```

4. informace
o chybě Slavu





MINISTERSTVO ŠKOLSTVÍ,
MLÁDEŽE A TĚLOVYCHOVY



Slave – implementace na PC (server)

Konfigurace:

Přijímá požadavky od klienta a vrací odpovědi

- požadavek na zápis jediného vnitřního registru (hodnota 0 až 1023) – funkční kód 6, hodnotu zobrazí a vrací potvrzení o přijetí požadavku

metoda AnsWr třídy ModbusRTU s kódem přijaté funkce

- požadavek na čtení bitové hodnoty – funkční kód 1 a vrací stav požadovaného bitu

metoda AnsRd třídy ModbusRTU s kódem přijaté funkce

Kontrolovat přijatý požadavek a vracet chybovou odpověď v případě neimplementované funkce, neexistující adresy registru nebo bitu a hodnoty mimo rozsah

metoda AnsErr třídy ModbusRTU s upraveným kódem funkce a typem chyby

Skupinové vysílání ignorovat .

Implementace generování intervalu 3,5 znaku pro ukončení příjmu zprávy

Omezená (žádná) implementace generování intervalu 1,5 znaku



Slave – zpracování požadavku, kontrola položek, příprava odpovědi

1. CRC

```
if(Mr.Crc(bfin, ix-1) != Mr.RdCrc(bfin, ix-1)
{
    .. možná informace o chybné CRC
}
else {
```

2. adresa

```
adr_r=bfin[0];
if(adr_r == ADR_S)
{
```

3. kód funkce

```
kod_r=bfin[1];
er=0;
switch(kod_r) {
    case FCE_WREG:
        .
        .
    case FCE_RBIT:
        .
        .
default: er=1;
}
```





Slave – zpracování požadavku, kontrola položek, příprava odpovědi

FCE_WREG:

```
reg=Mr.RdWord(bfin,2);
val=Mr.RdWord(bfin,4);
if(reg!=REG_WR) er=2;
else if(val>1023) er=3;
else .. zobrazení hodnoty
if(er==0) n= Mr.Answr(ADR_S,kod_r,reg,val,bfout);
```



FCE_RBIT:

```
reg=Mr.RdWord(bfin,2);
pocet=Mr.RdWord(bfin,4);
if(reg!=BIT_RD || pocet!=1) er=2;
else {
    tlačítko -> vals
    n= Mr.Answrd(ADR_S,kod_r,1,vals,bfout);
}
```

byte []vals = byte[1];

4. chyba

if(er>0) n=Mr.Answerr(adr_r,(byte)(kod_r|0x80),er,bfout);

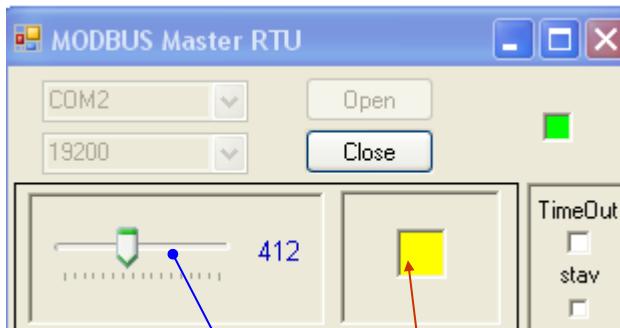
odeslání
odpovědi

```
n=Mr.WrCrc(Mr.Crc(bfout,n),bfout,n);
comPort.write(bfout, 0, n);
```

```
stav=Tstav.stklid;
```



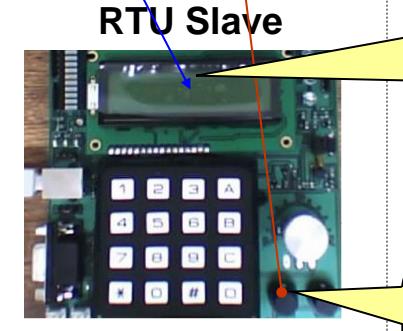
2.část : PC-mikropočítáč



potvrzení
požadavku
AnsWr

čtení bitu
(funkce 1)
Rd

zápis 1 registru
(funkce 6)
WrOne



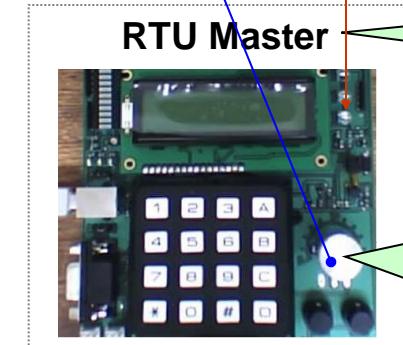
RTU Slave

potvrzení
požadavku
MrtuAnsWr

stav tlačítka
MrtuAnsRd



stav bitu
AnsRd



RTU Master

čtení bitu
(funkce 1)
MrtuRd

zápis hodnoty
potenciometru
(funkce 6)
MrtuWrOne



Master – implementace na mikropočítači (klient)

Konfigurace:

Realizuje funkce (požadavky na server)

- požadavek na zápis jediného vnitřního registru (hodnota 0 až 1023) – funkční kód 6
[aplikativní funkce MrtuWrOne s kódem funkce 6 \(FCE_WREG\)](#)
- požadavek na čtení bitové hodnoty – funkční kód 1
[aplikativní funkce MrtuRd s kódem funkce 1 \(FCE_RBIT\)](#)

Požadavky odesílat střídavě v pravidelných časových intervalech cca 200 ms,
jen když je Master ve stavu **klidu**

realizace časovačem T0

Implementovat generování čekacího TimeOut intervalu 500 ms na odpověď od Slave

Zjednodušený příjem odpovědi

příchozí adresu Slave není nutno testovat, pouze správnost CRC

zpracovat jen odpověď na požadavek čtení bitu (FCE_RBIT)

informace o chybě Slave: jen omezeně, nebo vůbec

Implementace generování intervalu 3,5 znaku pro ukončení příjmu zprávy časovačem T1

Omezená (žádná) implementace generování intervalu 1,5 znaku



Master – vyslání požadavku

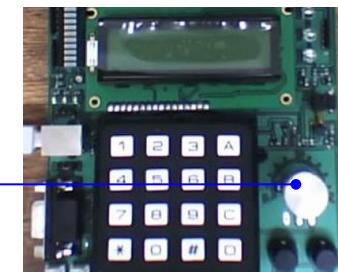
střídavě každých cca 200 ms vysílá rámec s funcí **1** (čtení bitu) a **6** (zápis registru)



```
#define N_TICKS 7
#define TIMEOUT 17
bit prep;
```

časovač T0

```
if(++cnt_ticks>=N_TICKS && stav==stKlid) {
    cnt_ticks=0;
    DIR485=1; /* na vysílání */
    prep=!prep;
    if(prep) { val= ... ;
        itx=MrtuWrOne(ADR_S,FCE_WREG,REG_WR,val,bfout); }
    else itx=MrtuRd(ADR_S,FCE_RBIT,BIT_RD,1,bfout);
    itx+=MrtuWrCrc(MrtuCrc(bfout,itx),bfout+itx);
    SendBuf(bfout,itx);
    stav=stCekani;
    DIR485=0; /* zpět na příjem */
}
```



TimeOut

```
if(cnt_ticks>=TIMEOUT)
{
    cnt=0;
    LED_R=!LED_R; // signalizace timeoutu
    stav=stKlid;
}
```



Master – zpracování odpovědi

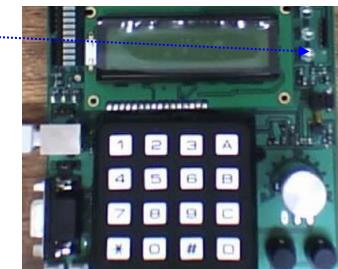
časovač T1

CRC

kód funkce

reakce
na odpověď

```
if(TF1)
{
    TR1=0;
    if(stav==stPrijem)
    {
        if(MrtuCrc(bfin,ix-2)==MrtuRdCrc(bfin+ix-2))
        {
            if ((kod_r=bfin[1]) == FCE_RBIT)
            {
                if (bfin[3] & 1) ... ; // LED svítí
                else ... ; // LED nesvítí
            }
            stav=stKlid;
        }
    }
}
```





Slave – implementace na mikropočítači (server)

Konfigurace:

Přijímá požadavky od klienta a vrací odpovědi

- požadavek na zápis jediného vnitřního registru (hodnota 0 až 1023) – funkční kód 6, hodnotu zobrazí a vrací potvrzení o přijetí požadavku

[aplikační funkce MrtuAnsWr s kódem přijaté funkce](#)

- požadavek na čtení bitové hodnoty – funkční kód 1 a vrací stav požadovaného bitu

[aplikační funkce MrtuAnsRd s kódem přijaté funkce](#)

Kontrolovat přijatý požadavek a vracet chybovou odpověď v případě neimplementované funkce, neexistující adresy registru nebo bitu a hodnoty mimo rozsah

[aplikační funkce MrtuAnsErr s upraveným kódem funkce a typem chyby](#)

Skupinové vysílání ignorovat .

Implementace generování intervalu 3,5 znaku pro ukončení příjmu zprávy časovačem T1
Omezená (žádná) implementace generování intervalu 1,5 znaku



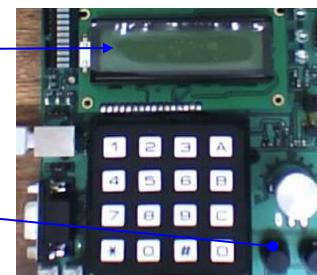
Slave – zpracování požadavku, kontrola položek, příprava odpovědi

CRC
a adresa

```
if((bfin[0]==ADR_S)&&(MrtuCrc(bfin,ix-2)==MrtuRdCrc(bfin+ix-2)))  
{
```

kód
funkce

```
er=0;  
switch(kod_r=bfin[1])  
{  
    case FCE_WREG:  
        .  
    case FCE_RBIT:  
        .  
    default: er=1;  
}
```

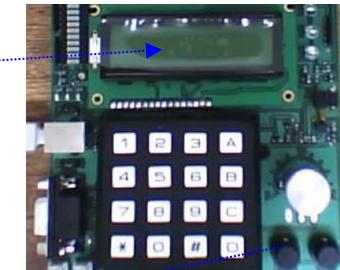




Slave – zpracování požadavku, kontrola položek, příprava odpovědi

FCE_WREG:

```
if((reg=Rdword(bfin+2))!=REG_WR) er=2;  
else if((val=Rdword(bfin+4))>1023) er=3;  
else printf(...);  
if(er==0) itx=MrtuAnsWr(ADR_S,kod_r,reg,val,bfout);  
break;
```



FCE_RBIT:

```
if((reg=Rdword(bfin+2))!=BIT_RD || (pocet=Rdword(bfin+4))!=1)er=2;  
else  
{  
    bity[0]= ... ;  
    itx=MrtuAnsRd(ADR_S,kod_r,1,bity,bfout);  
}  
break;
```

byte bity[1];

chyba

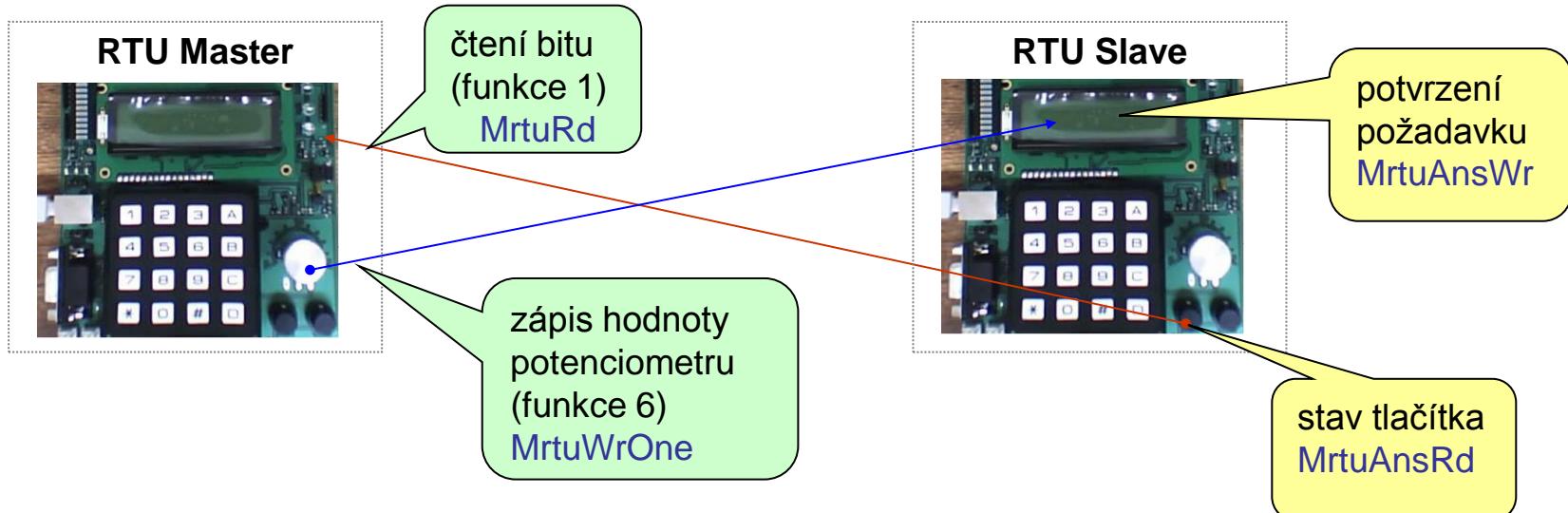
```
if(er)itx=MrtuAnsErr(adr_r,kod_r|0x80,er,bfout);
```

odeslání
odpovědi

```
DIR485=1; /* na vysílání */  
itx+=MrtuWrCrc(MrtuCrc(bfout,itx),bfout+itx);  
SendBuf(bfout,itx);  
DIR485=0; /* zpět na příjem */
```



3.část : mikropočítač – mikropočítač





Řídicí počítačové systémy

Úloha pro samostatná cvičení - MR2

Implementace protokolu MODBUS RTU na PC a mikropočítačích řady '51 pro uzly Master (Klient) i Slave (Server).

Požadované implementované funkce:

- zápis jediného bitového stavu (Coil) do uzlu Slave,
- čtení 16 bitového vnitřního registru (Holding) z uzlu Slave.

Rozhraní: RS232, standardní rámec 8,N,2

1. část: propojení PC – PC (C# MSVS)

2. část: propojení PC – mikropočítač

Rozhraní: RS485, standardní rámec 8,N,2

3. část: propojení mikropočítač – mikropočítač

Funkce pro podporu aplikace protokolu MODBUS:

v souboru Modbus.dll a Modbus.cs pro PC (C#),

v souboru Modbus.H a Modbus.C pro mikropočítač



MASTER (klient)

V pravidelných časových intervalech generuje 1bitovou informaci a předává požadavek na zápis do uzlu SLAVE

V pravidelných časových intervalech generuje požadavek na čtení 16bitové hodnoty z uzlu SLAVE a zobrazuje ji

SLAVE (server)

Po příjmu informaci zobrazí a odešle potvrzovací odpověď

Po příjmu požadavku hodnotu odešle

kód fukce: 05

+ data

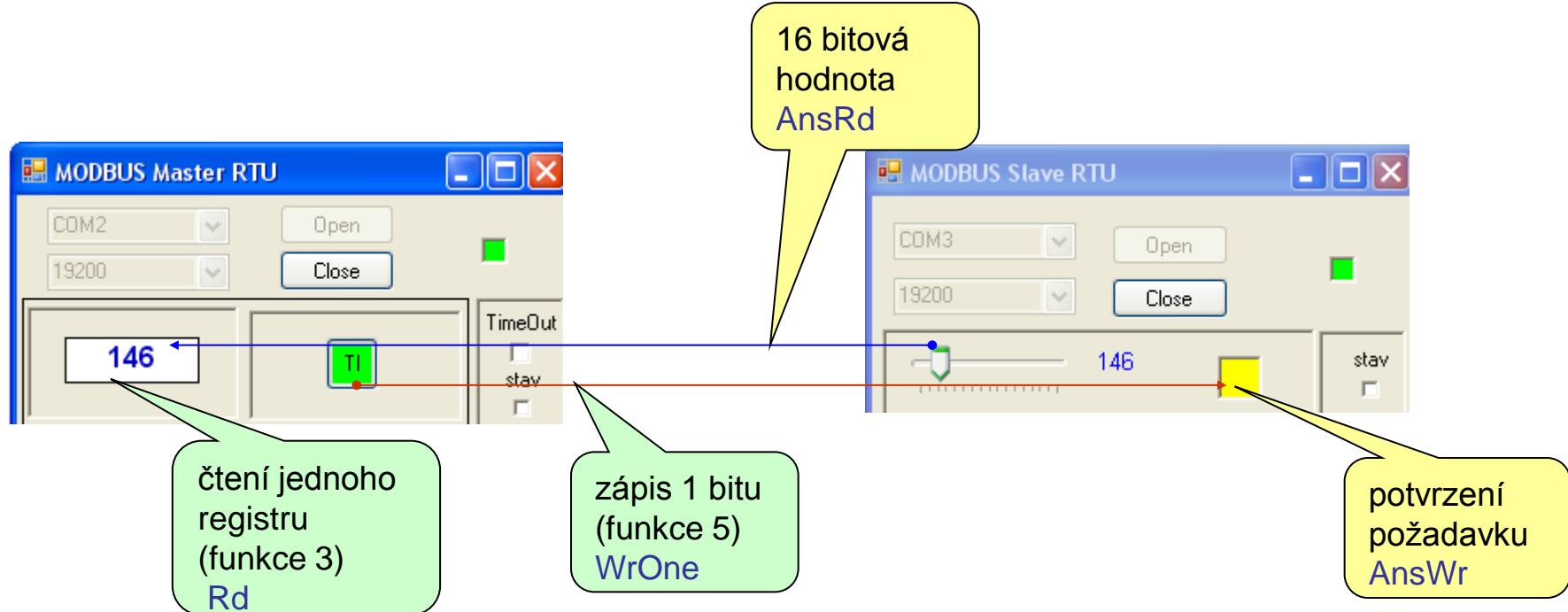
potvrzení

kód fukce: 03

data



1.část : PC-PC (varianta C#)





Master – implementace na PC (klient)

Konfigurace:

Realizuje funkce (požadavky na server)

- požadavek na zápis jediného bitového stavu – funkční kód 5
[metoda WrOne třídy ModbusRTU s kódem funkce 5 \(FCE_WBIT\)](#)
- požadavek na čtení 16 bitové hodnoty vnitřního registru – funkční kód 3
[metoda Rb třídy ModbusRTU s kodem funkce 3 \(FCE_RREG\)](#)

Požadavky odesílat střídavě v pravidelných časových intervalech 200 ms,
jen když je sériový kanál otevřen a Master je ve stavu **klidu**

realizace časovačem intervalu

Implementovat generování čekacího TimeOut intervalu 500 ms na odpověď od Slave,
po vypršení TimeOutu vyčkat 500 ms a várít se do stavu **klidu**

Zjednodušený příjem odpovědi

příchozí adresu Slave není nutno testovat, pouze správnost CRC

zpracovat jen odpověď na požadavek čtení registru (FCE_RREG)

informovat o chybové odpovědi od Slave

Implementace generování intervalu 3,5 znaku pro ukončení příjmu zprávy

Omezená (žádná) implementace generování intervalu 1,5 znaku



Master – vyslání požadavku

Časovač Sample

střídavě každých cca 200 ms vysílá rámec s funcí 3 (čtení registru) a 5 (zápis bitu)

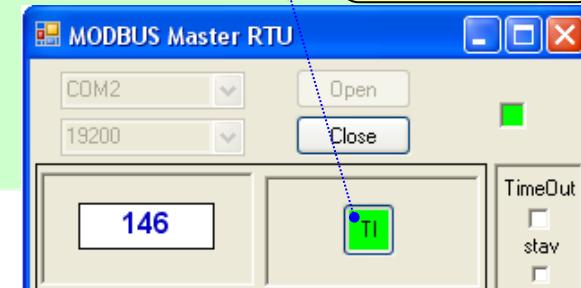


ModbusRTU Mr;
bool prep;
Tstav stav;

Tick
Sample

```
if (comPort.IsOpen && stav == Tstav.stklid)
{
    stav=Tstav.stVysilani;
    prep = !prep;
    if (prep) n=Mr.WrOne(ADR_S,FCE_WBIT,BIT_WR,val,bfout)
    else n=Mr.Rd(ADR_S,FCE_RREG,REG_RD,1,bfout);
    n=Mr.WrCrc(Mr.Crc(bfout,n),bfout,n);
    comPort.Write(bfout,0,n);
    TimerOut.Interval=500;
    TimerOut.Enabled=true;
    stav=Tstav.stCekani;
}
```

stisk: 0xFF00
jinak: 0





Master – zpracování odpovědi

1. CRC

```
if(Mr.Crc(bfin,ix-1)!=Mr.RdCrc(bfin,ix-1)
{
    .. možná informace o chybné CRC
}
else {
```

2. adresa
a kód funkce

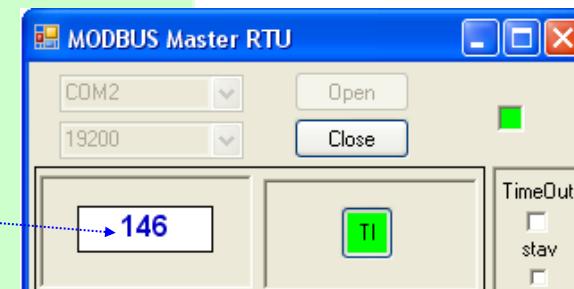
```
adr_r=bfin[0];
kod_r=bfin[1];
```

3. reakce
na odpověď

```
if (kod_r==FCE_RREG)
{
    pocet=bfin[2];
    val=Mr.Rdword(bfin,3);
    ..
}
```

4. informace
o chybě Slavu

```
else if (kod_r>=0x80)
{
    er = bfin[2]);
    switch (er) {
        informace o chybě slavu
    }
    stav=Tstav.stklid;
```





Slave – implementace na PC (server)

Konfigurace:

Přijímá požadavky od klienta a vrací odpovědi

- požadavek na zápis jediného bitového stavu – funkční kód 5, stav indikuje a vrací potvrzení o přijetí požadavku

metoda `AnsWr` třídy `ModbusRTU` s kódem přijaté funkce

- požadavek na čtení 16 bitové hodnoty – funkční kód 3 a vrací požadovanou hodnotu

metoda `AnsRd` třídy `ModbusRTU` s kódem přijaté funkce

Kontrolovat přijatý požadavek a vracet chybovou odpověď v případě neimplementované funkce, neexistující adresy registru nebo bitu a hodnoty mimo rozsah

metoda `AnsErr` třídy `ModbusRTU` s upraveným kódem funkce a typem chyby

Skupinové vysílání ignorovat .

Implementace generování intervalu 3,5 znaku pro ukončení příjmu zprávy

Omezená (žádná) implementace generování intervalu 1,5 znaku



Slave – zpracování požadavku, kontrola položek, příprava odpovědi

1. CRC

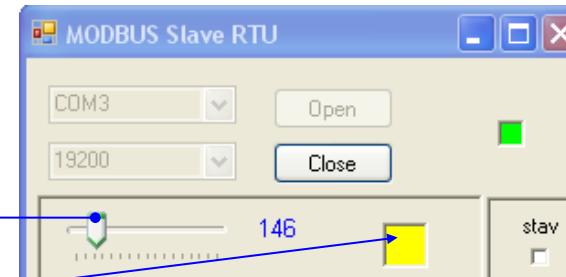
```
if(Mr.Crc(bfin,ix-1) !=Mr.RdCrc(bfin,ix-1)
{
    .. možná informace o chybné CRC
}
else {
```

2. adresa

```
adr_r=bfin[0];
if(adr_r == ADR_S)
{
```

3. kód funkce

```
kod_r=bfin[1];
er=0;
switch(kod_r) {
    case FCE_RREG:
        .
        .
    case FCE_WBIT:
        .
        .
    default: er=1;
}
```





Slave – zpracování požadavku, kontrola položek, příprava odpovědi

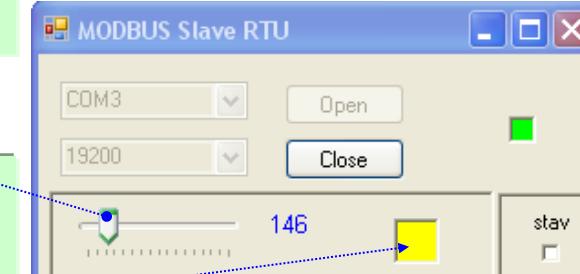
FCE_RREG:

```
reg=Mr.RdWord(bfin,2);
pocet=Mr.RdWord(bfin,4);
if(reg!=REG_RD || pocet!=1) er=2;
else .. MSB -> vals[0] a LSB -> vals[1]
if(er==0) n= Mr.AnswR(ADR_S,kod_r,2,vals,bfout);
```

byte []vals = byte[2];

FCE_WBIT:

```
reg=Mr.RdWord(bfin,2);
val=Mr.RdWord(bfin,4);
if (reg!=BIT_WR) er=2;
else switch (val) {
    case 0xFF00: .. žlutá
    case 0x0000: .. bílá
    default: er=3;
}
if(er==0) n= Mr.AnswR(ADR_S,kod_r,reg,val,bfout);
```



4. chyba

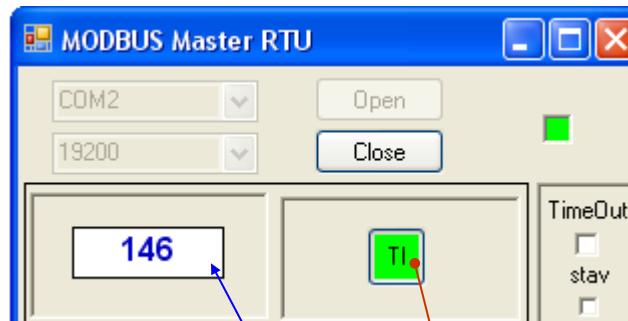
if(er>0) n=Mr.AnswErr(adr_r,(byte)(kod_r|0x80),er,bfout);

odeslání
odpovědi

```
n=Mr.WrCrc(Mr.Crc(bfout,n),bfout,n);
comPort.write(bfout, 0, n);
```



2.část : PC – mikropočítač



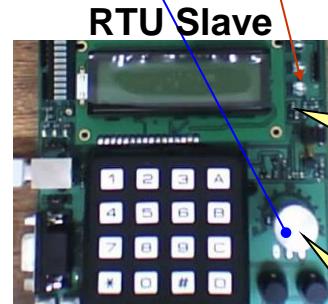
čtení jednoho registru
(funkce 3)
Rd

16 bitová hodnota
AnsRd



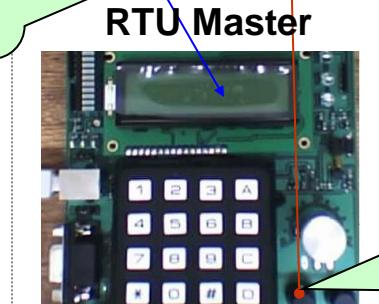
zápis 1 bitu
(funkce 5)
WrOne

potvrzení požadavku
AnsWr



potvrzení požadavku
MrtuAnsWr

hodnota potenciometru
MrtuAnsRd



zápis stavu tlačítka
(funkce 5)
MrtuWrOne

čtení jednoho registru
(funkce 3)
MrtuRd



Master – implementace na mikropočítači (klient)

Konfigurace:

Realizuje funkce (požadavky na server)

- požadavek na zápis jediného bitového stavu – funkční kód 5

[aplikativní funkce MrtuWrOne s kódem funkce 5 \(FCE_WBIT\)](#)

- požadavek čtení 16 bitové hodnoty vnitřního registru – funkční kód 3

[aplikativní funkce MrtuRd s kodem funkce 3 \(FCE_RREG\)](#)

Požadavky odesílat střídavě v pravidelných časových intervalech cca 200 ms,

jen když je Master ve stavu **klidu**

realizace časovačem T0

Implementovat generování čekacího TimeOut intervalu 500 ms na odpvěď od Slave

Zjednodušený příjem odpovědi

příchozí adresu Slave není nutno testovat, pouze správnost CRC

zpracovat jen odpověď na požadavek čtení registru (FCE_RREG)

informace o chybě Slave: jen omezeně, nebo vůbec

Implementace generování intervalu 3,5 znaku pro ukončení příjmu zprávy časovačem T1

Omezená (žádná) implementace generování intervalu 1,5 znaku



Master – vyslání požadavku

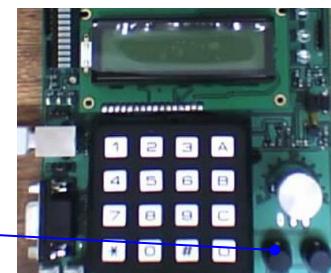
střídavě každých cca 210 ms vysílá rámec s funcí **3** (čtení bitu) a **5** (zápis registru)



časovač T0

```
if(++cnt_ticks==N_TICKS && stav==stK1id)
{
    cnt_ticks=0;
    DIR485=1; /* na vysílání */
    prep=!prep;
    if(prep){ val= ... ;
        itx=MrtuWrOne(ADR_S,FCE_WBIT,BIT_WR,val,bfout); }
    else itx=MrtuRd(ADR_S,FCE_RREG,REG_RD,1,bfout);
    itx+=MrtuWrCrc(MrtuCrc(bfout,itx),bfout+itx);
    SendBuf(bfout,itx);
    stav=stCekani;
    DIR485=0; /* zpět na příjem */
}
```

```
#define N_TICKS 7
#define TIMEOUT 17
bit prep;
```



TimeOut

```
if(cnt_ticks==TIMEOUT)
{
    cnt_ticks=0;
    LED_R=!LED_R; // signalizace TimeOutu
    stav=stK1id;
}
```



Master – zpracování odpovědi

časovač T1

CRC

kód funkce

reakce
na odpověď

```
if(TF1)
{
    TR1=0;
    if(stav==stPrijem)
    {
        if(MrtuCrc(bfin,ix-2)==MrtuRdCrc(bfin+ix-2))
        {
            if ((kod_r=bfin[1]) == FCE_RREG)
            {
                val=Rdword(bfin+3);
                printf(...);
            }
        }
        stav=stKlid;
    }
}
```





Slave – implementace na mikropočítači (server)

Konfigurace:

Přijímá požadavky od klienta a vrací odpovědi

- požadavek na zápis jediného bitového stavu – funkční kód 5,
stav indikuje a vrací potvrzení o přijetí požadavku

aplikativní funkce MrtuAnsWr s kódem přijaté funkce

- požadavek na čtení 16 bitové hodnoty – funkční kód 3 a vrací požadovanou hodnotu
aplikativní funkce MrtuAnsRd s kódem přijaté funkce

Kontrolovat přijatý požadavek a vracet chybovou odpověď v případě neimplementované funkce, neexistující adresy registru nebo bitu a hodnoty mimo rozsah

aplikativní funkce MrtuAnsErr s upraveným kódem funkce a typem chyby

Skupinové vysílání ignorovat .

Implementace generování intervalu 3,5 znaku pro ukončení příjmu zprávy časovačem T1
Omezená (žádná) implementace generování intervalu 1,5 znaku



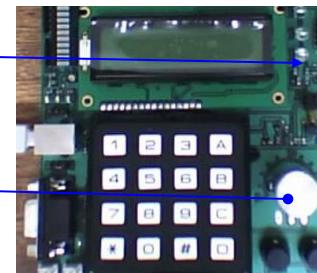
Slave – zpracování požadavku, kontrola položek, příprava odpovědi

CRC
a adresa

```
if((bfin[0]==ADR_S)&&(MrtuCrc(bfin,ix-2)==MrtuRdCrc(bfin+ix-2)))  
{
```

kód
funkce

```
er=0;  
switch(kod_r=bfin[1])  
{  
    case FCE_WBIT:  
        .  
    case FCE_RREG:  
        .  
    default: er=1;  
}
```

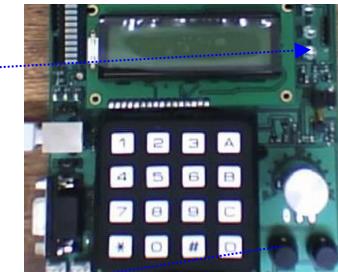




Slave – zpracování požadavku, kontrola položek, příprava odpovědi

FCE_WBIT:

```
if((reg=Rdword(bfin+2))!=BIT_WR) er=2;
else if((val=Rdword(bfin+4))!=0 && val!=0xFF00) er=3;
else LED_G ...;
if(er==0) itx=MrtuAnsWr(ADR_S,kod_r,reg,val,bfout);
break;
```



FCE_RREG:

```
if((reg=Rdword(bfin+2))!=REG_RD || (pocet=Rdword(bfin+4))!=1)er=2;
else {
    val = ...;
    vals[0]=val>>8;
    vals[1]=val;
    itx=MrtuAnsRd(ADR_S,kod_r,2,vals,bfout);
}
break;
```

byte vals[2];

chyba

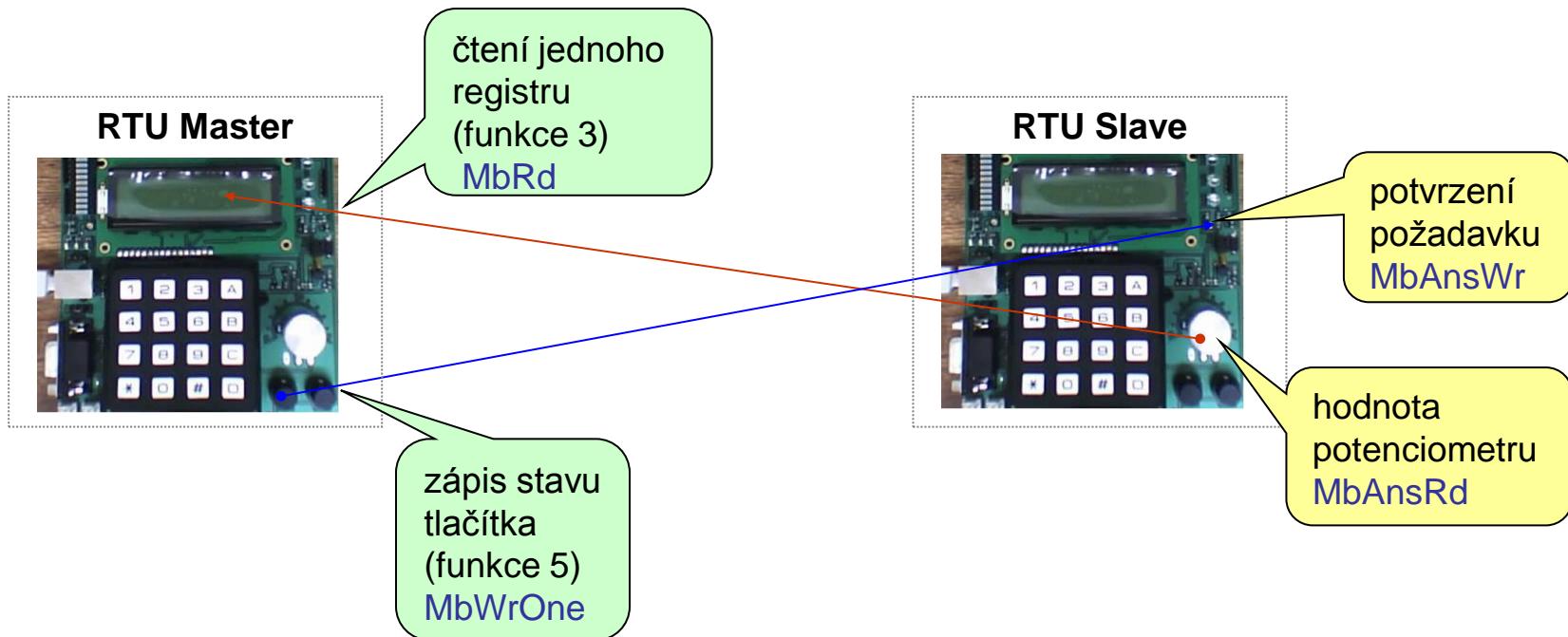
```
if(er)itx=MrtuAnsErr(adr_r,kod_r|0x80,er,bfout);
```

odeslání
odpovědi

```
DIR485=1; /* na vysílání */
itx+=MrtuWrCrc(MrtuCrc(bfout,itx),bfout+itx);
SendBuf(bfout,itx);
DIR485=0; /* zpět na příjem */
```



3.část : mikropočítač – mikropočítač



Řídicí počítačové systémy

Úloha pro samostatná cvičení - MR3

Implementace protokolu MODBUS RTU na PC a mikropočítačích řady '51 pro uzly Master (Klient) i Slave (Server).

Požadované implementované funkce:

- čtení 16bitového vnitřního registru (**Holding**) z uzlu Slave,
- čtení bitového stavu (**Coil**) z uzlu Slave.

Rozhraní: RS232, standardní rámec 8,N,2

- 1. část: propojení PC – PC (C# MSVS)**
- 2. část: propojení PC – mikropočítač**
Rozhraní: RS485, standardní rámec 8,N,2
- 3. část: propojení mikropočítač – mikropočítač**

Funkce pro podporu aplikace protokolu MODBUS:

- v souboru **Modbus.dll** a **Modbus.cs** pro PC (C#),
- v souboru **Modbus.H** a **Modbus.C** pro mikropočítač



MASTER (klient)

V pravidelných časových intervalech generuje požadavek na čtení 16bitové hodnoty z uzlu SLAVE a zobrazuje ji

V pravidelných časových intervalech generuje požadavek na čtení bitové informace z uzlu SLAVE a zobrazuje ji

kód funkce: 03

data

kód funkce: 01

data

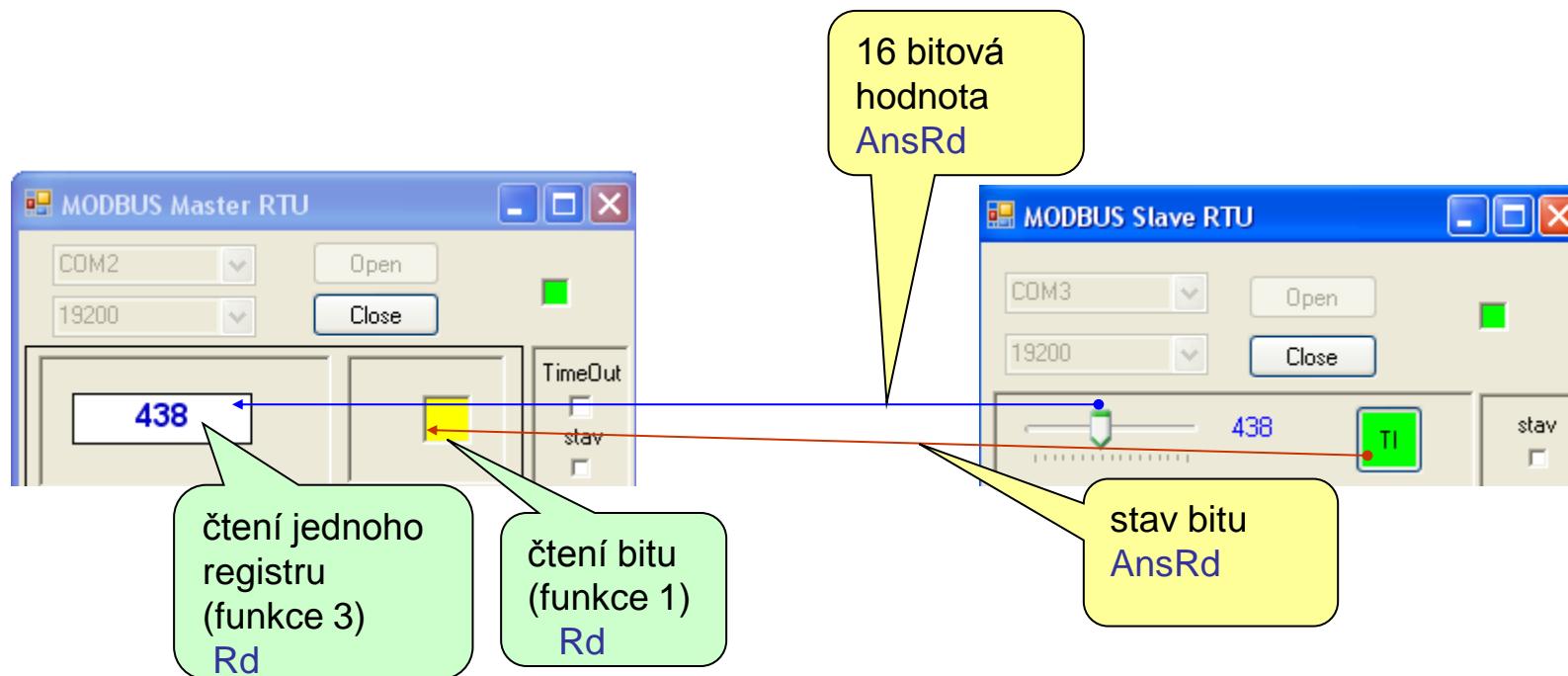
SLAVE (server)

Po příjmu požadavku hodnotu odešle

Po příjmu požadavku stavu bitu odešle



1.část : PC-PC (varianta C#)





Master – implementace na PC (klient)

Konfigurace:

Realizuje funkce (požadavky na server)

- požadavek na čtení 16bitové hodnoty vnitřního registru – funkční kód 3

metoda Rd třídy ModbusRTU s kódem funkce 3 (FCE_RREG)

- požadavek na čtení bitové hodnoty – funkční kód 1

metoda Rd třídy ModbusRTU s kodem funkce 1 (FCE_RBIT)

Požadavky odesílat střídavě v pravidelných časových intervalech 200 ms,

jen když je sériový kanál otevřen a Master je ve stavu **klidu**

realizace časovačem intervalu

Implementovat generování čekacího TimeOut intervalu 500 ms na odpvěď od Slave,
po vypršení TimeOutu vyčkat 500 ms a várít se do stavu **klidu**

Zjednodušený příjem odpovědi

příchozí adresu Slave není nutno testovat, pouze správnost CRC

zpracovat odpovědi na požadavky čtení registru (FCE_RREG)

a čtení bitu (FCE_RBIT)

informovat o chybové odpovědi od Slave

Implementace generování intervalu 3,5 znaku pro ukončení příjmu zprávy

Omezená (žádná) implementace generování intervalu 1,5 znaku



Master – vyslání požadavku

Časovač Sample

střídavě každých cca 200 ms vysílá rámec s funcí 1 (čtení bitu) a 3 (čtení registru)



ModbusRTU Mr;
bool prep;
Tstav stav;

Tick
Sample

```
if (comPort.IsOpen && stav == Tstav.stklid)
{
    stav=Tstav.stVysilani;
    prep = !prep;
    if (prep) n=Mr.Rd(ADR_S,FCE_RBIT,BIT_RD,1,bfout)
    else n=Mr.Rd(ADR_S,FCE_RREG,REG_RD,1,bfout);
    n=Mr.WrCrc(Mr.Crc(bfout,n),bfout,n);
    comPort.Write(bfout,0,n);
    TimerOut.Interval=500;
    TimerOut.Enabled=true;
    stav=Tstav.stCekani;
}
```



Master – zpracování odpovědi

1. LRC

```
if(Mr.Crc(bfin, ix-1) !=Mr.RdCrc(bfin, ix-1)
{
    .. možná informace o chybné CRC
}
else {
```

2. adresa
a kód funkce

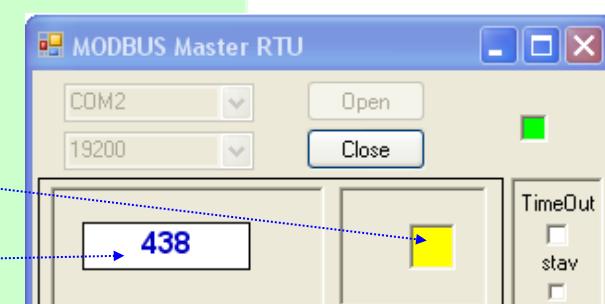
```
adr_r=bfin[0];
kod_r=bfin[1];
```

3. reakce
na odpověď

```
switch (kod_r) {
    case FCE_RBIT:
        .
        .
    case FCE_RREG:
        .
        .
default: if (kod_r>=0x80)
{
```

4. informace
o chybě Slavu

```
    er = bfin[2]);
    switch (er) {
        informace o chybě slavu
    }
}
stav=Tstav.stklid;
```





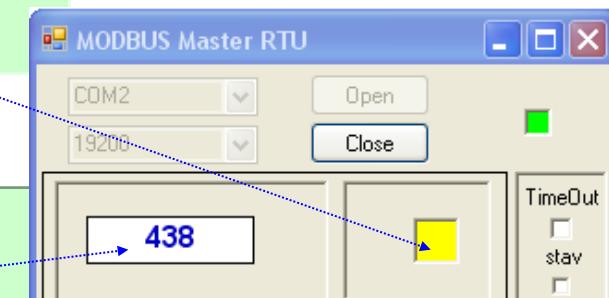
Master – zpracování odpovědi

FCE_RBIT:

```
begin
    pocet=bfin[2];
    val=bfin[3];
    if (adr_r=ADR_S) and (pocet=1) then
        if (val and 1)=1 then žlutá
        else bílá
    end
```

FCE_RREG:

```
begin
    pocet=bfin[2];
    val=Mr.RdWord(bfin,3);
    ...
end
```





Slave – implementace na PC (server)

Konfigurace:

Přijímá požadavky od klienta a vrací odpovědi

- požadavek na čtení 16bitové hodnoty – funkční kód 3 a vrací požadovanou hodnotu,
[metoda AnsRd třídy ModbusRTU s kódem přijaté funkce](#)
- požadavek na čtení bitové hodnoty – funkční kód 1 a vrací stav požadovaného bitu
[metoda AnsRd třídy ModbusRTU s kódem přijaté funkce](#)

Kontrolovat přijatý požadavek a vracet chybovou odpověď v případě neimplementované funkce, neexistující adresy registru nebo bitu a hodnoty mimo rozsah

[metoda AnsErr třídy ModbusRTU s upraveným kódem funkce a typem chyby](#)

Skupinové vysílání ignorovat .

Implementace generování intervalu 3,5 znaku pro ukončení příjmu zprávy

Omezená (žádná) implementace generování intervalu 1,5 znaku



Slave – zpracování požadavku, kontrola položek, příprava odpovědi

1. CRC

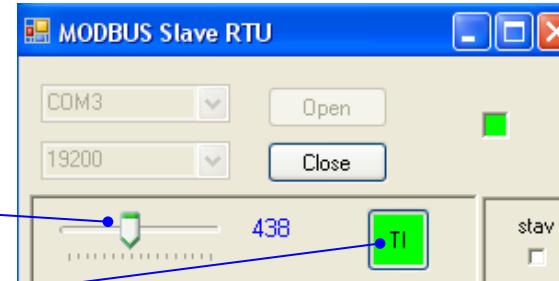
```
if(Mr.Crc(bfin, ix-1) != Mr.RdCrc(bfin, ix-1)
{
    .. možná informace o chybné CRC
}
else {
```

2. adresa

```
adr_r=bfin[0];
if(adr_r == ADR_S)
{
```

3. kód funkce

```
kod_r=bfin[1];
er=0;
switch(kod_r) {
    case FCE_RREG:
        .
        .
    case FCE_RBIT:
        .
        .
    default: er=1;
}
```





Slave – zpracování požadavku, kontrola položek, příprava odpovědi

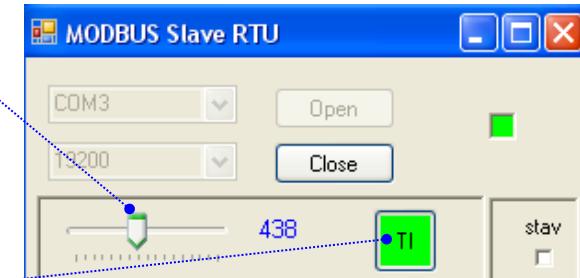
FCE_RREG:

```
reg=Mr.RdWord(bfin,2);
pocet=Mr.RdWord(bfin,4);
if(reg!=REG_RD || pocet!=1) er=2;
else .. MSB -> vals[0] a LSB -> vals[1]
if(er==0) n= Mr.AnSrd(ADR_S,kod_r,2,vals,bfout);
```

byte []vals = byte[2];

FCE_RBIT:

```
reg=Mr.RdWord(bfin,2);
pocet=Mr.RdWord(bfin,4);
if(reg!=BIT_RD || pocet!=1) er=2;
else .. tlačítka -> vals[0]
if(er==0) n= Mr.AnSRD(ADR_S,kod_r,1,vals,bfout);
```



4. chyba

if(er>0) n=Mr.AnSerr(adr_r,(byte)(kod_r|0x80),er,bfout);

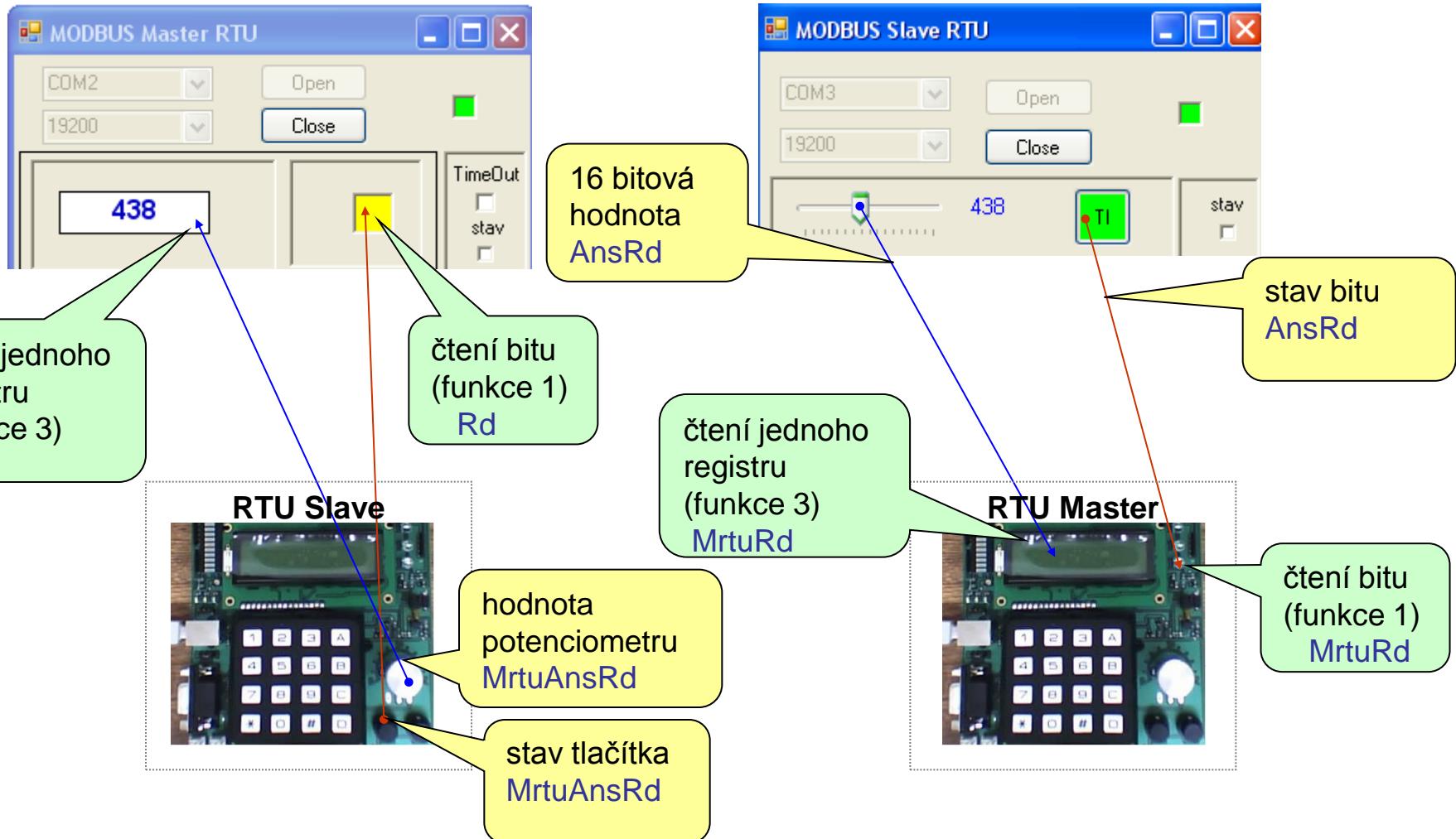
odeslání
odpovědi

```
n=Mr.WrCrc(Mr.Crc(bfout,n),bfout,n);
comPort.write(bfout, 0, n);
```

stav=Tstav.stklid;



2.část : PC-mikropočítač





Master – implementace na mikropočítači (klient)

Konfigurace:

Realizuje funkce (požadavky na server)

- požadavek čtení 16 bitové hodnoty vnitřního registru – funkční kód 3

[aplikativní funkce MrtuRd s kódem funkce 3 \(FCE_RREG\)](#)

- požadavek na čtení bitové hodnoty – funkční kód 1

[aplikativní funkce MrtuRd s kodem funkce 1 \(FCE_RBIT\)](#)

Požadavky odesílat střídavě v pravidelných časových intervalech cca 200 ms,
jen když je Master ve stavu **klidu**

realizace časovačem T0

Implementovat generování čekacího TimeOut intervalu 500 ms na odpvěď od Slave

Zjednodušený příjem odpovědi

příchozí adresu Slave není nutno testovat, pouze správnost CRC

zpracovat jen odpovědi na požadavky čtení registru (FCE_RREG)

a čtení bitu (FCE_RBIT)

informace o chybě Slave: jen omezeně, nebo vůbec

Implementace generování intervalu 3,5 znaku pro ukončení příjmu zprávy časovačem T1

Omezená (žádná) implementace generování intervalu 1,5 znaku



Master – vyslání požadavku

střídavě každých cca 210 ms vysílá rámec s funcí **1** (čtení bitu) a **3** (čtení registru)



časovač T0

```
if(++cnt_ticks>=N_TICKS && stav==stKlid)
{
    cnt_ticks=0;
    DIR485=1; /* na vysílání */
    prep=!prep;
    if(prep) itx=MrtuRd(ADR_S,FCE_RREG,REG_RD,1,bfout);
    else itx=MrtuRd(ADR_S,FCE_RBIT,BIT_RD,1,bfout);
    itx+=MrtuWrCrc(MrtuCrc(bfout,itx),bfout+itx);
    SendBuf(bfout,itx);
    stav=stCekani;
    DIR485=0; /* zpět na příjem */
}
```

```
#define N_TICKS 7
#define TIMEOUT 17
bit prep;
```

TimeOut

```
if(cnt_ticks>=TIMEOUT)
{
    cnt_ticks=0;
    LED_R=!LED_R; // signalizace TimeOutu
    stav=stKlid;
}
```

Master – zpracování odpovědi

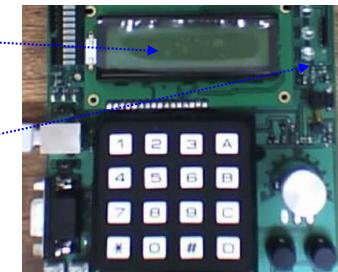
časovač T1

CRC

kód funkce

reakce
na odpověď

```
if(TF1)
{
    TR1=0;
    if(stav==stPrijem)
    {
        if(MrtuCrc(bfin,ix-2)==MrtuRdCrc(bfin+ix-2))
        {
            if ((kod_r=bfin[1]) == FCE_RREG)
            {
                val=Rdword(bfin+3);
                printf(...);
            }
            else if (kod_r == FCE_RBIT)
            {
                if (bfin[3] & 1) ... ; // LED svítí
                else ... ;           // LED nesvítí
            }
        }
        stav=stKlid;
    }
}
```





Slave – implementace na mikropočítači (server)

Konfigurace:

Přijímá požadavky od klienta a vrací odpovědi

- požadavek na čtení 16bitové hodnoty – funkční kód 3 a vrací požadovanou hodnotu,
[aplikativní funkce MrtuAnsRd s kódem přijaté funkce](#)
- požadavek na čtení bitové hodnoty – funkční kód 1 a vrací stav požadovaného bitu
[aplikativní funkce MrtuAnsRd s kódem přijaté funkce](#)

Kontrolovat přijatý požadavek a vracet chybovou odpověď v případě neimplementované funkce, neexistující adresy registru nebo bitu a hodnoty mimo rozsah

[aplikativní funkce MrtuAnsErr s upraveným kódem funkce a typem chyby](#)

Skupinové vysílání ignorovat .

Implementace generování intervalu 3,5 znaku pro ukončení příjmu zprávy časovačem T1
Omezená (žádná) implementace generování intervalu 1,5 znaku



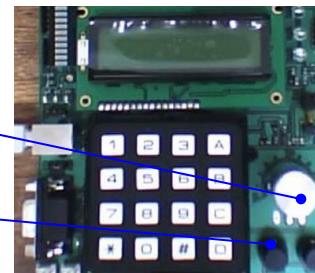
Slave – zpracování požadavku, kontrola položek, příprava odpovědi

CRC
a adresa

```
if((MrtuCrc(bfin, ix-2)==MrtuRdCrc(bfin+ix-2))&&(bfin[0]==ADR_S))  
{
```

kód
funkce

```
er=0;  
switch(kod_r=bfin[1])  
{  
    case FCE_RREG:  
        .  
    case FCE_RBIT:  
        .  
    default: er=1;  
}
```



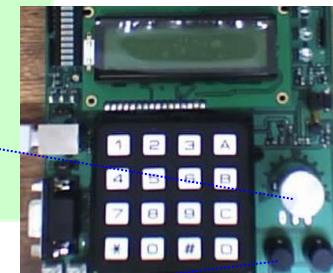


Slave – zpracování požadavku, kontrola položek, příprava odpovědi

FCE_RREG:

```
if((reg=Rdword(bfin+2))!=REG_RD || (pocet=Rdword(bfin+4))!=1)er=2;
else {
    val= ... ;
    vals[0]=val>>8;
    vals[1]=val;
    itx=MrtuAnsRd(ADR_S,kod_r,2,vals,bfout);
}
break;
```

byte vals[2];



FCE_RBIT:

```
if((reg=Rdword(bfin+2))!=BIT_RD || (pocet=Rdword(bfin+4))!=1)er=2;
else {
    vals[0]= ... ;
    itx=MrtuAnsRd(ADR_S,kod_r,1,bity,bfout);
}
break;
```

chyba

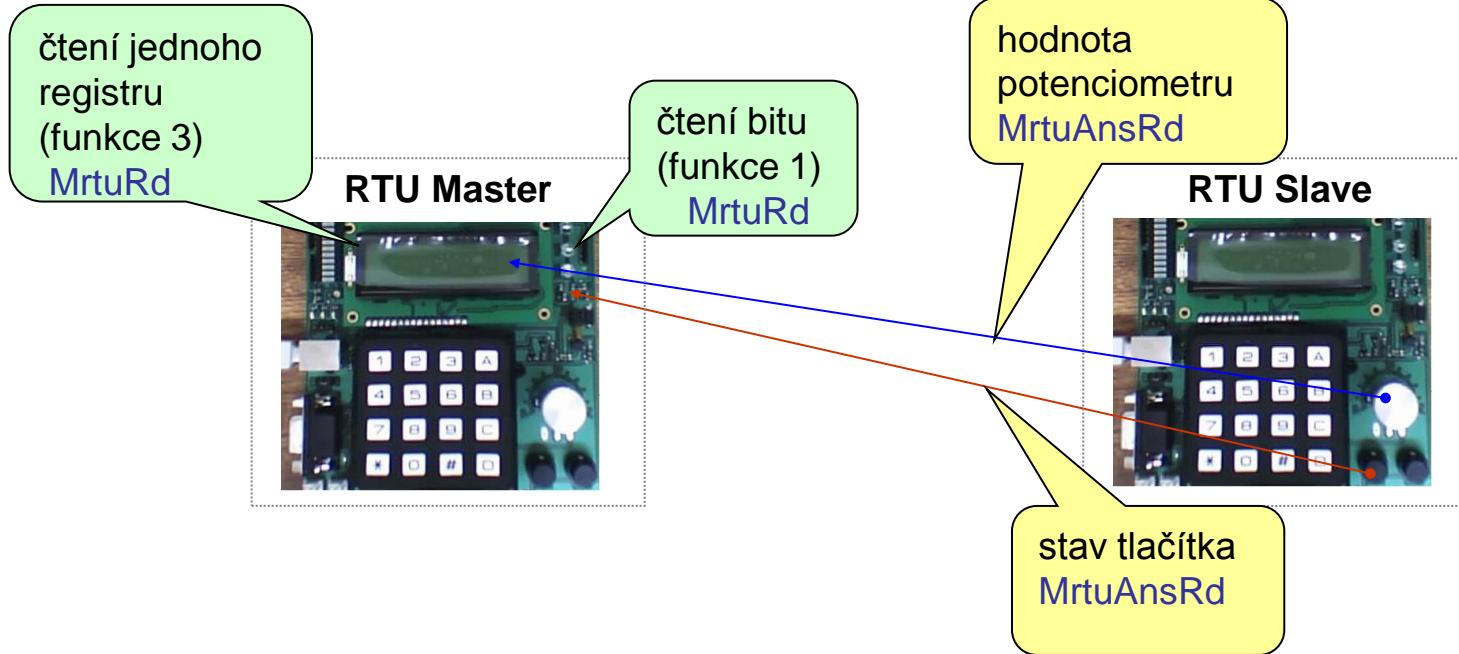
if(er)itx=MrtuAnsErr(adr_r,kod_r|0x80,er,bfout);

odeslání
odpovědi

```
DIR485=1; /* na vysílání */
itx+=MrtuWrCrc(MrtuCrc(bfout,itx),bfout+itx);
SendBuf(bfout,itx);
DIR485=0; /* zpět na příjem */
```



3.část : mikropočítač – mikropočítač





Řídicí počítačové systémy

Úloha pro samostatná cvičení - MR4

Implementace protokolu MODBUS RTU na PC a mikropočítačích řady '51 pro uzly Master (Klient) i Slave (Server).

Požadované implementované funkce:

- zápis jediného vnitřního registru (Holding) do uzlu Slave,
- zápis jediného bitového stavu (Coil) do uzlu Slave.

Rozhraní: RS232, standardní rámec 8,N,2

1. část: propojení PC – PC (C# MSVS)

2. část: propojení PC – mikropočítač

Rozhraní: RS485, standardní rámec 8,N,2

3. část: propojení mikropočítač – mikropočítač

Funkce pro podporu aplikace protokolu MODBUS:

v souboru Modbus.dll a Modbus.cs pro PC (C#),

v souboru Modbus.H a Modbus.C pro mikropočítač



MASTER (klient)

V pravidelných časových intervalech generuje 16 bitovou hodnotu a předává požadavek na zápis do uzlu SLAVE

V pravidelných časových intervalech generuje 1bitovou informaci a předává požadavek na zápis do uzlu SLAVE

kód fukce: 06
+ data

potvrzení

kód fukce: 05
+ data

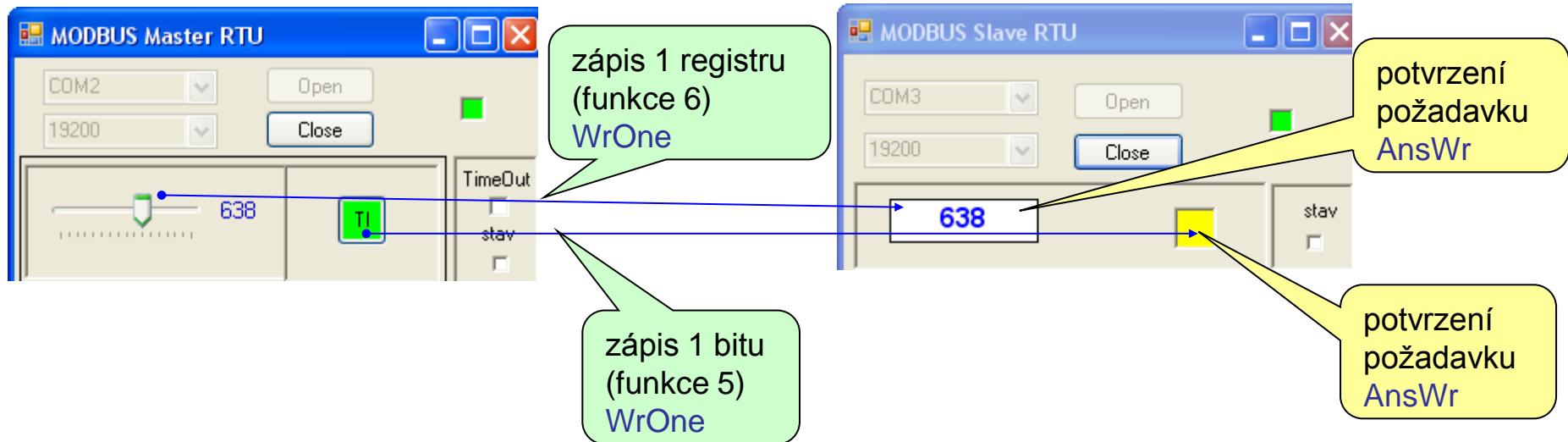
potvrzení

SLAVE (server)

Po příjmu hodnotu zobrazí a odešle potvrzovací odpověď

Po příjmu informaci zobrazí a odešle potvrzovací odpověď

1.část : PC-PC (varianta C#)





Master – implementace na PC (klient)

Konfigurace:

Realizuje funkce (požadavky na server)

- požadavek na zápis jediného vnitřního registru (hodnota 0 až 1023) – funkční kód 6
[metoda WrOne třídy ModbusRTU s kódem funkce 6 \(FCE_WREG\)](#)
- požadavek na zápis jediného bitového stavu – funkční kód 5
[metoda WrOne třídy ModbusRTU s kódem funkce 5 \(FCE_WBIT\)](#)

Požadavky odesílat střídavě v pravidelných časových intervalech 200 ms,
jen když je sériový kanál otevřen a Master je ve stavu **klidu**
realizace časovačem intervalu

Implementovat generování čekacího TimeOut intervalu 500 ms na odpvěď od Slave,
po vypršení TimeOutu vyčkat 500 ms a várat se do stavu **klidu**

Zjednodušený příjem odpovědi
příchozí adresu Slave není nutno testovat, pouze správnost CRC

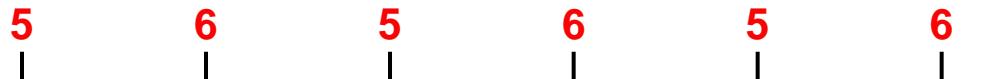
zpracovat jen chybové odpovědi od Slave a informovat o nich

Implementace generování intervalu 3,5 znaku pro ukončení příjmu zprávy
Omezená (žádná) implementace generování intervalu 1,5 znaku

Master – vyslání požadavku

Časovač Sample

střídavě každých cca 200 ms vysílá rámec s funcí 5 (zápis bitu) a 6 (zápis registru)

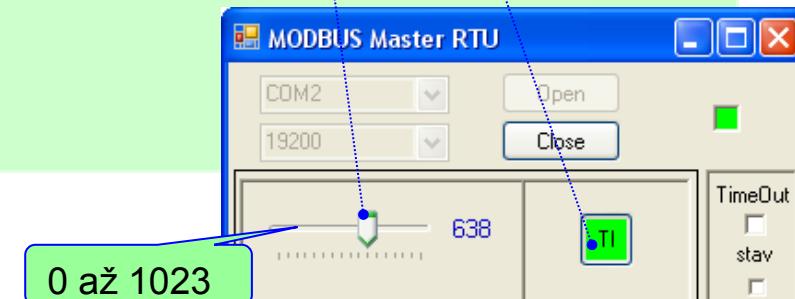


ModbusRTU Mr;
bool prep;
Tstav stav;

Tick
Sample

```
if (comPort.IsOpen && stav == Tstav.stklid)
{
    stav=Tstav.stVysilani;
    prep = !prep;
    if (prep) n=Mr.WrOne(ADR_S,FCE_WBIT,BIT_WR,val,bfout)
    else n=Mr.WrOne(ADR_S,FCE_WREG,REG_WR,pot,bfout);
    n=Mr.WrCrc(Mr.Crc(bfout,n),bfout,n);
    comPort.Write(bfout,0,n);
    TimerOut.Interval=500;
    TimerOut.Enabled=true;
    stav=Tstav.stCekani;
}
```

stisk: 0xFF00
jinak: 0





Master – zpracování odpovědi

1. CRC

```
if(Mr.Crc(bfin,ix-1)!=Mr.RdCrc(bfin,ix-1)
{
    .. možná informace o chybné CRC
}
```

2. adresa a kód funkce

```
adr_r=bfin[0];
kod_r=bfin[1];
```

3. informace o chybě Slavu

```
if (kod_r>=0x80)
{
    er= bfin[2];
    switch (er) {
        informace o chybě slavu
    }
}
stav=Tstav.stKlid;
```



Slave – implementace na PC (server)

Konfigurace:

Přijímá požadavky od klienta a vrací odpovědi

- požadavek na zápis jediného vnitřního registru (hodnota 0 až 1023) – funkční kód 6, hodnotu zobrazí a vrací potvrzení o přijetí požadavku

metoda `AnsWr` třídy `ModbusRTU` s kódem přijaté funkce

- požadavek na zápis jediného bitového stavu – funkční kód 5, stav indikuje a vrací potvrzení o přijetí požadavku

metoda `AnsWr` třídy `ModbusRTU` s kódem přijaté funkce

Kontrolovat přijatý požadavek a vracet chybovou odpověď v případě neimplementované funkce, neexistující adresy registru nebo bitu a hodnoty mimo rozsah

metoda `AnsErr` třídy `ModbusRTU` s upraveným kódem funkce a typem chyby

Skupinové vysílání ignorovat .

Implementace generování intervalu 3,5 znaku pro ukončení příjmu zprávy

Omezená (žádná) implementace generování intervalu 1,5 znaku



Slave – zpracování požadavku, kontrola položek, příprava odpovědi

1. CRC

```
if(Mr.Crc(bfin,ix-1) !=Mr.RdCrc(bfin,ix-1)
{
    .. možná informace o chybné CRC
}
else {
```

2. adresa

```
adr_r=bfin[0];
if(adr_r == ADR_S)
{
```

3. kód funkce

```
kod_r=bfin[1];
er=0;
switch(kod_r) {
    case FCE_WREG:
        .
        .
    case FCE_WBIT:
        .
        .
    default: er=1;
}
```





Slave – zpracování požadavku, kontrola položek, příprava odpovědi

FCE_WREG:

```
reg=Mr.Rdword(bfin,2);
val=Mr.Rdword(bfin,4);
if(reg!=REG_WR) er=2;
else if(val>1023) er=3;
else .. zobrazení hodnoty
if(er==0) n= Mr.Answr(ADR_S,kod_r,reg,val,bfout);
```



FCE_WBIT:

```
reg=Mr.Rdword(bfin,2);
val=Mr.Rdword(bfin,4);
if (reg!=BIT_WR) er=2;
else switch (val) {
    case 0xFF00: .. žlutá
    case 0x0000: .. bílá
    default: er=3;
}
if(er==0) n= Mr.Answr(ADR_S,kod_r,reg,val,bfout);
```

4. chyba

```
if(er>0) n=Mr.Answr(adr_r,(byte)(kod_r|0x80),er,bfout);
```

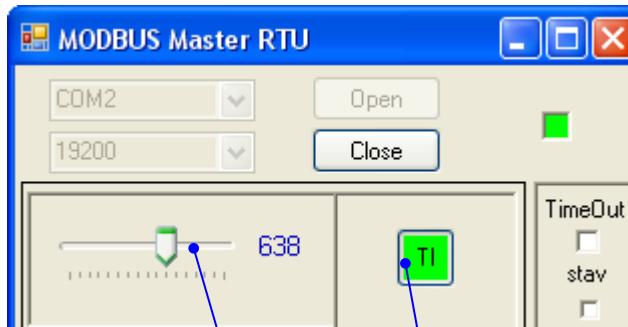
odeslání
odpovědi

```
n=Mr.WrCrc(Mr.Crc(bfout,n),bfout,n);
comPort.write(bfout, 0, n);
```

```
stav=Tstav.stklid;
```



2.část : PC – mikropočítač



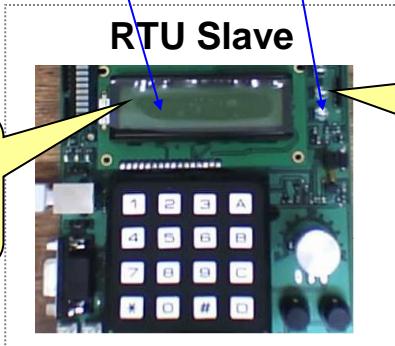
zápis 1 registru
(funkce 6)
WrOne

zápis 1 bitu
(funkce 5)
WrOne



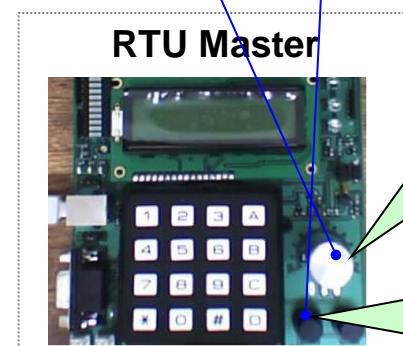
potvrzení
požadavku
AnsWr

potvrzení
požadavku
AnsWr



potvrzení
požadavku
MrtuAnsWr

potvrzení
požadavku
MrtuAnsWr



zápis hodnoty
potenciometru
(funkce 6)
MrtuWrOne

zápis stavu
tlačítka
(funkce 5)
MrtuWrOne



Master – implementace na mikropočítači (klient)

Konfigurace:

Realizuje funkce (požadavky na server)

- požadavek na zápis jediného vnitřního registru (hodnota 0 až 1023) – funkční kód 6
[aplikativní funkce MrtuWrOne s kódem funkce 6 \(FCE_WREG\)](#)
- požadavek na zápis jediného bitového stavu – funkční kód 5
[aplikativní funkce MrtuWrOne s kódem funkce 5 \(FCE_WBIT\)](#)

Požadavky odesílat střídavě v pravidelných časových intervalech cca 200 ms,
jen když je Master ve stavu **klidu**

realizace časovačem T0

Implementovat generování čekacího TimeOut intervalu 500 ms na odpvěď od Slave

Zjednodušený příjem odpovědi

příchozí adresu Slave není nutno testovat, pouze správnost CRC

zpracovat jen informace o chybě Slave: jen omezeně (žlutá LED), nebo vůbec

Implementace generování intervalu 3,5 znaku pro ukončení příjmu zprávy časovačem T1

Omezená (žádná) implementace generování intervalu 1,5 znaku



Master – vyslání požadavku

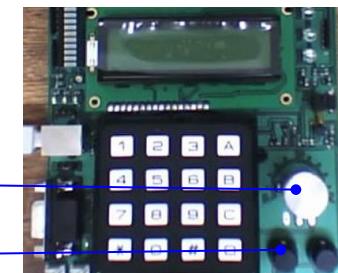
střídavě každých cca 200 ms vysílá rámec s funcí **5** (zápis bitu) a **6** (zápis registru)



```
#define N_TICKS 7
#define TIMEOUT 17
bit prep;
```

časovač T0

```
if(++cnt_ticks>=N_TICKS && stav==stKlid){
    cnt_ticks=0;
    DIR485=1; /* na vysílání */
    prep=!prep;
    if(prep) {val= ... ;
        itx=MrtuWrOne(ADR_S,FCE_WREG,REG_WR,val,bfout);}
    else { val= ... ;
        itx=MrtuWrOne(ADR_S,FCE_WBIT,BIT_WR,val,bfout);}
    itx+=MrtuWrCrc(MrtuCrc(bfout,itx),bfout+itx);
    SendBuf(bfout,itx);
    stav=stCekani;
    DIR485=0; /* zpět na příjem */
}
```



TimeOut

```
if(cnt_ticks==TIMEOUT)
{
    cnt=0;
    LED_R=!LED_R; // signalizace TimeOutu
    stav=stKlid;
}
```



Master – zpracování odpovědi

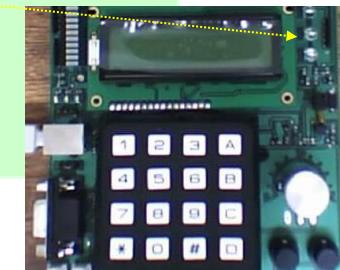
časovač T1

```
if(TF1)
{
    TR1=0;
    if(stav==stPrijem)
    {
        if(MrtuCrc(bfin,ix-2)==MrtuRdCrc(bfin+ix-2))
        {
            if ((kod_r=bfin[1]) >= 0x80)
                LED_Y=0;
            else LED_Y=1;
        }
        stav=stKlid;
    }
}
```

CRC

kód funkce

chybná odpověď





Slave – implementace na mikropočítači (server)

Konfigurace:

Přijímá požadavky od klienta a vrací odpovědi

- požadavek na zápis jediného vnitřního registru (hodnota 0 až 1023) – funkční kód 6, hodnotu zobrazí a vrací potvrzení o přijetí požadavku

aplikativní funkce MrtuAnsWr s kódem přijaté funkce

- požadavek na zápis jediného bitového stavu – funkční kód 5, stav indikuje a vrací potvrzení o přijetí požadavku

aplikativní funkce MrtuAnsWr s kódem přijaté funkce

Kontrolovat přijatý požadavek a vracet chybovou odpověď v případě neimplementované funkce, neexistující adresy registru nebo bitu a hodnoty mimo rozsah

aplikativní funkce MrtuAnsErr s upraveným kódem funkce a typem chyby

Skupinové vysílání ignorovat .

Implementace generování intervalu 3,5 znaku pro ukončení příjmu zprávy časovačem T1
Omezená (žádná) implementace generování intervalu 1,5 znaku



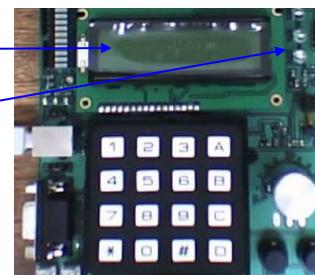
Slave – zpracování požadavku, kontrola položek, příprava odpovědi

CRC
a adresa

```
if((bfin[0]==ADR_S)&&(MrtuCrc(bfin,ix-2)==MrtuRdCrc(bfin+ix-2)))  
{
```

kód
funkce

```
er=0;  
switch(kod_r=bfin[1])  
{  
    case FCE_WREG:  
        .  
    case FCE_WBIT:  
        .  
    default: er=1;  
}
```

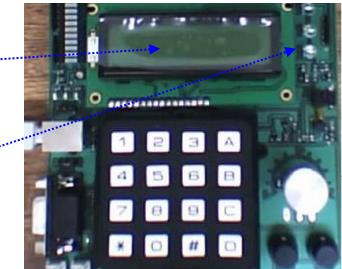




Slave – zpracování požadavku, kontrola položek, příprava odpovědi

FCE_WREG:

```
if((reg=Rdword(bfin+2))!=REG_WR) er=2;
else if((val=Rdword(bfin+4))>1023) er=3;
else printf(...);
if(er==0) itx=MrtuAnsWr(ADR_S,kod_r,reg,val,bfout);
break;
```



FCE_WBIT:

```
if((reg=Rdword(bfin+2))!=BIT_WR) er=2;
else if((val=Rdword(bfin+4))!=0 && val!=0xFF00) er=3;
else LED_G ...;
if(er==0) itx=MrtuAnsWr(ADR_S,kod_r,reg,val,bfout);
break;
```

chyba

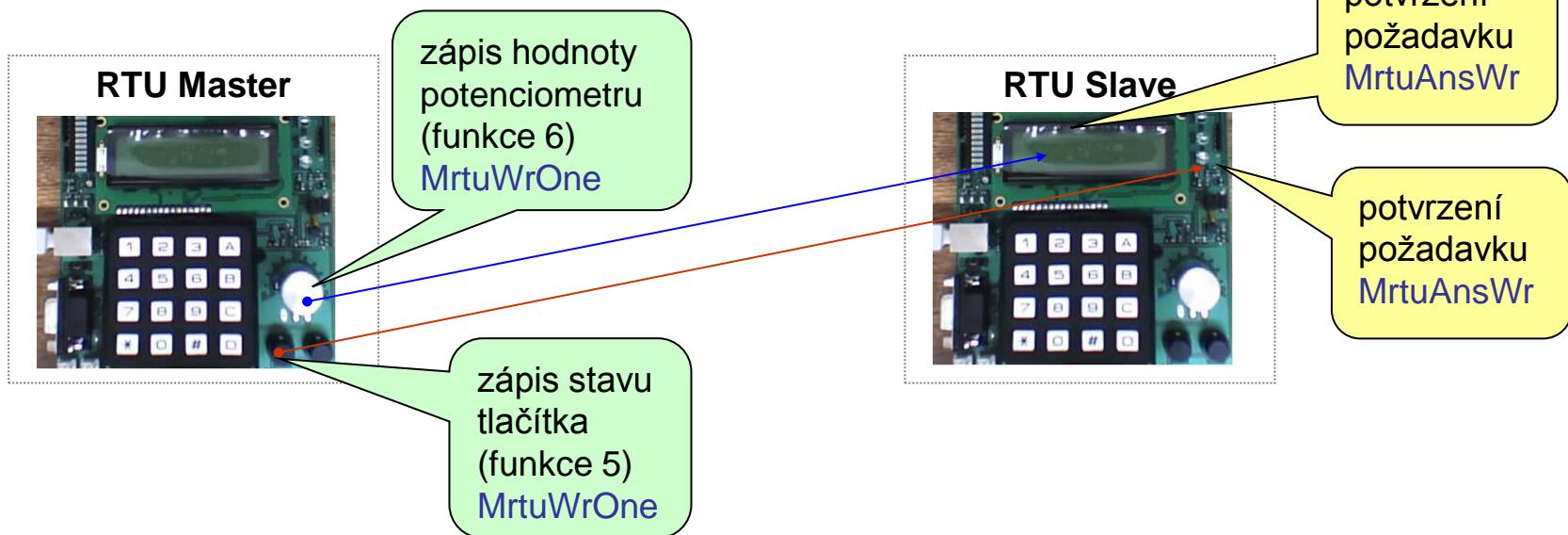
```
if(er)itx=MrtuAnsErr(adr_r,kod_r|0x80,er,bfout);
```

odeslání
odpovědi

```
DIR485=1; /* na vysílání */
itx+=MrtuWrCrc(MrtuCrc(bfout,itx),bfout+itx);
SendBuf(bfout,itx);
DIR485=0; /* zpět na příjem */
```



3.část : mikropočítač – mikropočítač



Řídicí počítačové systémy

Úloha pro samostatná cvičení - MR5

Implementace protokolu MODBUS RTU na PC a mikropočítačích řady '51 pro uzly Master (Klient) i Slave (Server).

Požadované implementované funkce:

- čtení 16bitového vnitřního registru (**Holding**) z uzlu Slave,
- zápis jediného vnitřního registru (**Holding**) do uzlu Slave,

Rozhraní: RS232, standardní rámec 8,N,2

1. část: propojení PC – PC (C# MSVS)
 2. část: propojení PC – mikropočítač
- Rozhraní: RS485, standardní rámec 8,N,2**
3. část: propojení mikropočítač – mikropočítač

Funkce pro podporu aplikace protokolu MODBUS:

- v souboru Modbus.dll a Modbus.cs pro PC (C#),
- v souboru Modbus.H a Modbus.C pro mikropočítač



MASTER (klient)

V pravidelných časových intervalech generuje 16 bitovou hodnotu a předává požadavek na zápis do uzlu SLAVE

V pravidelných časových intervalech generuje požadavek na čtení 16bitové hodnoty z uzlu SLAVE a zobrazuje ji

kód fukce: 06
+ data

potvrzení

kód fukce: 03

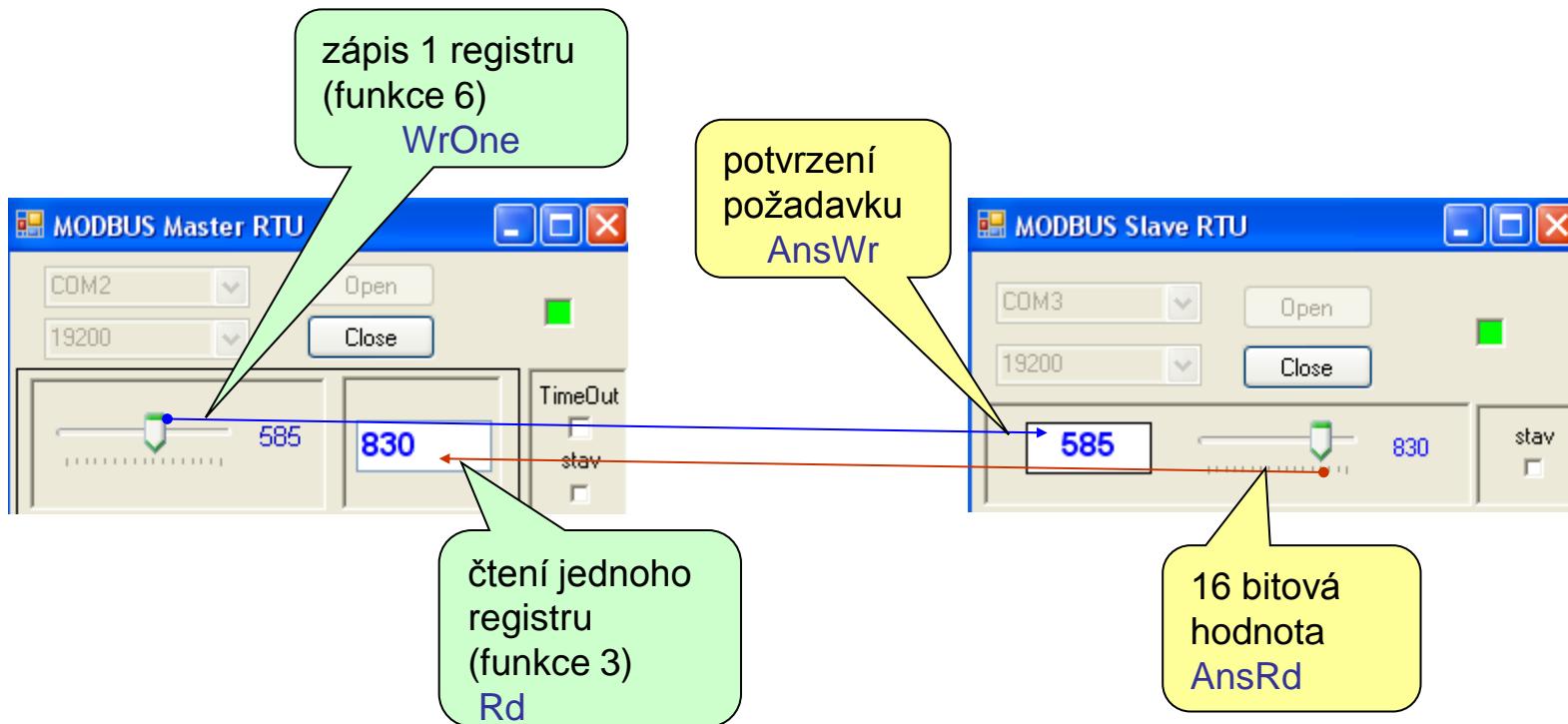
data

SLAVE (server)

Po příjmu hodnotu zobrazí a odešle potvrzovací odpověď

Po příjmu požadavku hodnotu odešle

1.část : PC-PC (varianta C#)





Master – implementace na PC (klient)

Konfigurace:

Realizuje funkce (požadavky na server)

- požadavek na zápis jediného vnitřního registru (hodnota 0 až 1023) – funkční kód 6

metoda WrOne třídy ModbusRTU s kódem funkce 6 (FCE_WREG)

- požadavek na čtení 16 bitové hodnoty – funkční kód 3

metoda Rd třídy ModbusRTU s kodem funkce 3 (FCE_RREG)

Požadavky odesílat střídavě v pravidelných časových intervalech 200 ms,

jen když je sériový kanál otevřen a Master je ve stavu **klidu**

realizace časovačem intervalu

Implementovat generování čekacího TimeOut intervalu 500 ms na odpověď od Slave,
po vypršení TimeOutu vyčkat 500 ms a vátit se do stavu **klidu**

Zjednodušený příjem odpovědi

příchozí adresu Slave není nutno testovat, pouze správnost CRC

zpracovat jen odpověď na požadavek čtení registru (FCE_RREG)

informovat o chybové odpovědi od Slave

Implementace generování intervalu 3,5 znaku pro ukončení příjmu zprávy

Omezená (žádná) implementace generování intervalu 1,5 znaku

Master – vyslání požadavku

Časovač Sample

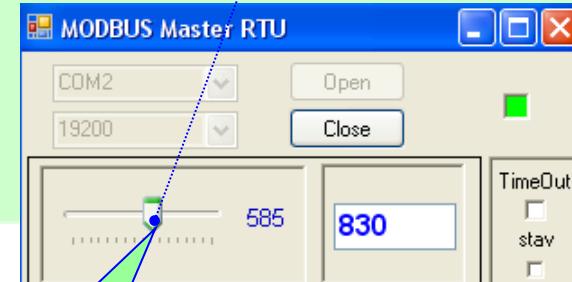
střídavě každých cca 200 ms vysílá rámec s funcí **3** (čtení bitu) a **6** (zápis registru)



ModbusRTU Mr;
bool prep;
Tstav stav;

Tick
Sample

```
if (comPort.IsOpen && stav == Tstav.stKlid)
{
    stav=Tstav.stVysilani;
    prep = !prep;
    if (prep) n=Mr.wrOne(ADR_S,FCE_WREG,REG_WR,pot,bfout)
    else n=Mr.Rd(ADR_S,FCE_RREG,REG_RD,1,bfout);
    n=Mr.WrCrc(Mr.Crc(bfout,n),bfout,n);
    comPort.Write(bfout,0,n);
    TimerOut.Interval=500;
    TimerOut.Enabled=true;
    stav=Tstav.stCekani;
}
```



0 až 1023



Master – zpracování odpovědi

1. CRC

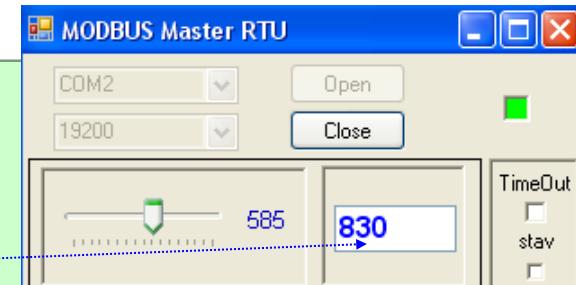
```
if(Mr.Crc(bfin, ix-1) !=Mr.RdCrc(bfin, ix-1)
{
    .. možná informace o chybné CRC
}
else {
```

2. adresa
a kód funkce

```
adr_r=bfin[0];
kod_r=bfin[1];
```

3. reakce
na odpověď

```
if(kod_r== FCE_RREG)
{
    pocet=bfin[2];
    val=Mr.Rdword(bfin,3);
    ..
else if (kod_r>=0x80)
{
    er= bfin[2];
    switch(er) {
        informace o chybě slavu
    }
}
stav=Tstav.stklid;
```



4. informace
o chybě Slavu



Slave – implementace na PC (server)

Konfigurace:

Přijímá požadavky od klienta a vrací odpovědi

- požadavek na zápis jediného vnitřního registru (hodnota 0 až 1023) – funkční kód 6, hodnotu zobrazí a vrací potvrzení o přijetí požadavku

metoda AnsWr třídy ModbusRTU s kódem přijaté funkce

- požadavek na čtení 16bitové hodnoty – funkční kód 3 a vrací požadovanou hodnotu,

metoda AnsRd třídy ModbusRTU s kódem přijaté funkce

Kontrolovat přijatý požadavek a vracet chybovou odpověď v případě neimplementované funkce, neexistující adresy registru nebo bitu a hodnoty mimo rozsah

metoda AnsErr třídy ModbusRTU s upraveným kódem funkce a typem chyby

Skupinové vysílání ignorovat .

Implementace generování intervalu 3,5 znaku pro ukončení příjmu zprávy

Omezená (žádná) implementace generování intervalu 1,5 znaku



Slave – zpracování požadavku, kontrola položek, příprava odpovědi

1. CRC

```
if(Mr.Crc(bfin, ix-1) != Mr.RdCrc(bfin, ix-1)
{
    .. možná informace o chybné CRC
}
else {
```

2. adresa

```
adr_r=bfin[0];
if(adr_r == ADR_S)
{
```

3. kód funkce

```
kod_r=bfin[1];
er=0;
switch(kod_r) {
    case FCE_WREG:
        .
        .
    case FCE_RREG:
        .
        .
    default: er=1;
}
```

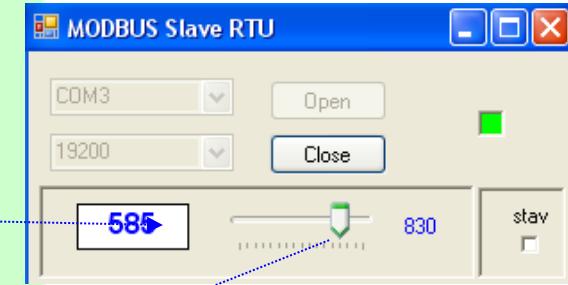




Slave – zpracování požadavku, kontrola položek, příprava odpovědi

FCE_WREG:

```
reg=Mr.Rdword(bfin,2);
val=Mr.Rdword(bfin,4);
if(reg!=REG_WR) er=2;
else if(val>1023) er=3;
else .. zobrazení hodnoty
if(er==0) n= Mr.Answr(ADR_S,kod_r,reg,val,bfout);
```



FCE_RREG:

```
reg=Mr.Rdword(bfin,2);
pocet=Mr.RdWord(bfin,4);
if(reg!=REG_RD || pocet!=1) er=2;
else .. MSB -> vals[0] a LSB -> vals[1]
if(er==0) n= Mr.Answrd(ADR_S,kod_r,2,vals,bfout);
```

byte []vals = byte[2];

4. chyba

if(er>0) n=Mr.Answr(adr_r,(byte)(kod_r|0x80),er,bfout);

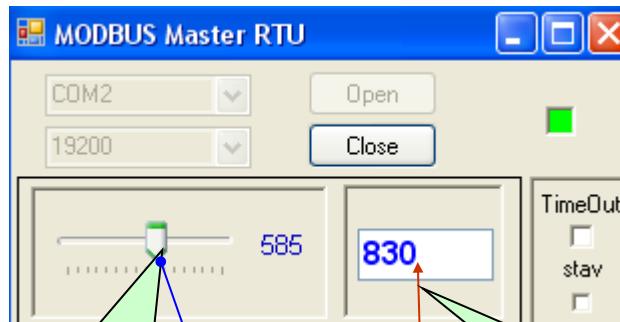
odeslání
odpovědi

```
n=Mr.WrCrc(Mr.Crc(bfout,n),bfout,n);
comPort.write(bfout, 0, n);
```

```
stav=Tstav.stklid;
```



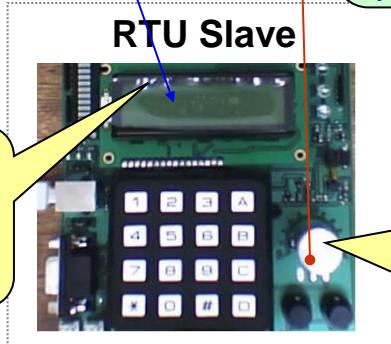
2.část : PC-mikropočítáč



zápis 1 registru
(funkce 6)
WrOne

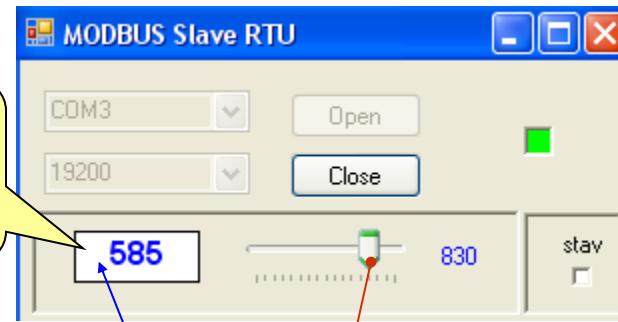
potvrzení
požadavku
AnsWr

čtení jednoho
registru
(funkce 3)
Rd



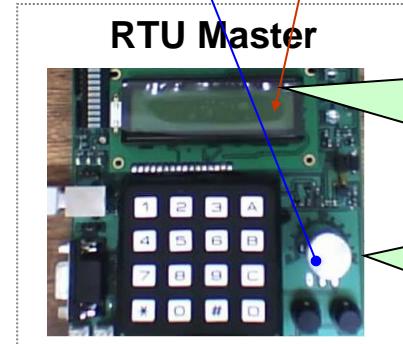
potvrzení
požadavku
MrtuAnsWr

hodnota
potenciometru
MrtuAnsRd



585

16 bitová
hodnota
AnsRd



čtení jednoho
registru
(funkce 3)
MrtuRd

zápis hodnoty
potenciometru
(funkce 6)
MrtuWrOne



Master – implementace na mikropočítači (klient)

Konfigurace:

Realizuje funkce (požadavky na server)

- požadavek na zápis jediného vnitřního registru (hodnota 0 až 1023) – funkční kód 6
[aplikativní funkce MrtuWrOne s kódem funkce 6 \(FCE_WREG\)](#)
- požadavek čtení 16 bitové hodnoty vnitřního registru – funkční kód 3
[aplikativní funkce MrtuRd s kódem funkce 3 \(FCE_RREG\)](#)

Požadavky odesílat střídavě v pravidelných časových intervalech cca 200 ms,
jen když je Master ve stavu **klidu**

realizace časovačem T0

Implementovat generování čekacího TimeOut intervalu 500 ms na odpověď od Slave

Zjednodušený příjem odpovědi

příchozí adresu Slave není nutno testovat, pouze správnost CRC
zpracovat jen odpověď na požadavek čtení registru (FCE_RREG)
informace o chybě Slave: jen omezeně, nebo vůbec

Implementace generování intervalu 3,5 znaku pro ukončení příjmu zprávy časovačem T1
Omezená (žádná) implementace generování intervalu 1,5 znaku



Master – vyslání požadavku

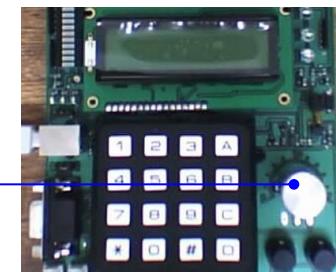
střídavě každých cca 200 ms vysílá rámec s funcí **3** (čtení registru) a **6** (zápis registru)



```
#define N_TICKS 7
#define TIMEOUT 17
bit prep;
```

časovač T0

```
if(++cnt_ticks>=N_TICKS && stav==stKlid) {
    cnt_ticks=0;
    DIR485=1; /* na vysílání */
    prep=!prep;
    if(prep) { val= ... ;
        itx=MrtuWrOne(ADR_S,FCE_WREG,REG_WR,val,bfout); }
    else itx=MrtuRd(ADR_S,FCE_RREG,REG_RD,1,bfout);
    itx+=MrtuWrCrc(MrtuCrc(bfout,itx),bfout+itx);
    SendBuf(bfout,itx);
    stav=stCekani;
    DIR485=0; /* zpět na příjem */
}
```



TimeOut

```
if(cnt_ticks>=TIMEOUT)
{
    cnt=0;
    LED_R=!LED_R; // signalizace timeoutu
    stav=stKlid;
}
```



Master – zpracování odpovědi

časovač T1

CRC

kód funkce

reakce
na odpověď

```
if(TF1)
{
    TR1=0;
    if(stav==stPrijem)
    {
        if(MrtuCrc(bfin,ix-2)==MrtuRdCrc(bfin+ix-2))
        {
            if ((kod_r=bfin[1]) == FCE_RREG)
            {
                val=Rdword(bfin+3);
                printf(...);
            }
        }
        stav=stKlid;
    }
}
```





Slave – implementace na mikropočítači (server)

Konfigurace:

Přijímá požadavky od klienta a vrací odpovědi

- požadavek na zápis jediného vnitřního registru (hodnota 0 až 1023) – funkční kód 6, hodnotu zobrazí a vrací potvrzení o přijetí požadavku

[aplikáční funkce MrtuAnsWr s kódem přijaté funkce](#)

- požadavek na čtení 16bitové hodnoty – funkční kód 3 a vrací požadovanou hodnotu, [aplikáční funkce MrtuAnsRd s kódem přijaté funkce](#)

Kontrolovat přijatý požadavek a vracet chybovou odpověď v případě neimplementované funkce, neexistující adresy registru nebo bitu a hodnoty mimo rozsah

[aplikáční funkce MrtuAnsErr s upraveným kódem funkce a typem chyby](#)

Skupinové vysílání ignorovat .

Implementace generování intervalu 3,5 znaku pro ukončení příjmu zprávy časovačem T1
Omezená (žádná) implementace generování intervalu 1,5 znaku



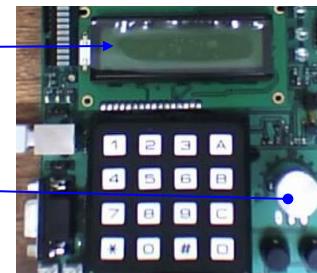
Slave – zpracování požadavku, kontrola položek, příprava odpovědi

CRC
a adresa

```
if((bfin[0]==ADR_S)&&(MrtuCrc(bfin,ix-2)==MrtuRdCrc(bfin+ix-2)))  
{
```

kód
funkce

```
er=0;  
switch(kod_r=bfin[1])  
{  
    case FCE_WREG:  
        .  
    case FCE_RREG:  
        .  
    default: er=1;  
}
```

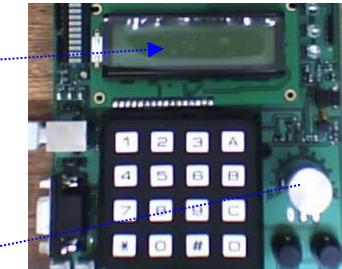




Slave – zpracování požadavku, kontrola položek, příprava odpovědi

FCE_WREG:

```
if((reg=RdWord(bfin+2))!=REG_WR) er=2;
else if((val=RdWord(bfin+4))>1023) er=3;
else printf(...);
if(er==0) itx=MrtuAnsWr(ADR_S,kod_r,reg,val,bfout);
break;
```



FCE_RREG:

```
if((reg=RdWord(bfin+2))!=REG_RD || (pocet=RdWord(bfin+4))!=1)er=2;
else {
    val= ... ;
    vals[0]=val>>8;
    vals[1]=val;
    itx=MrtuAnsRd(ADR_S,kod_r,2,vals,bfout);
}
break;
```

byte vals[2];

chyba

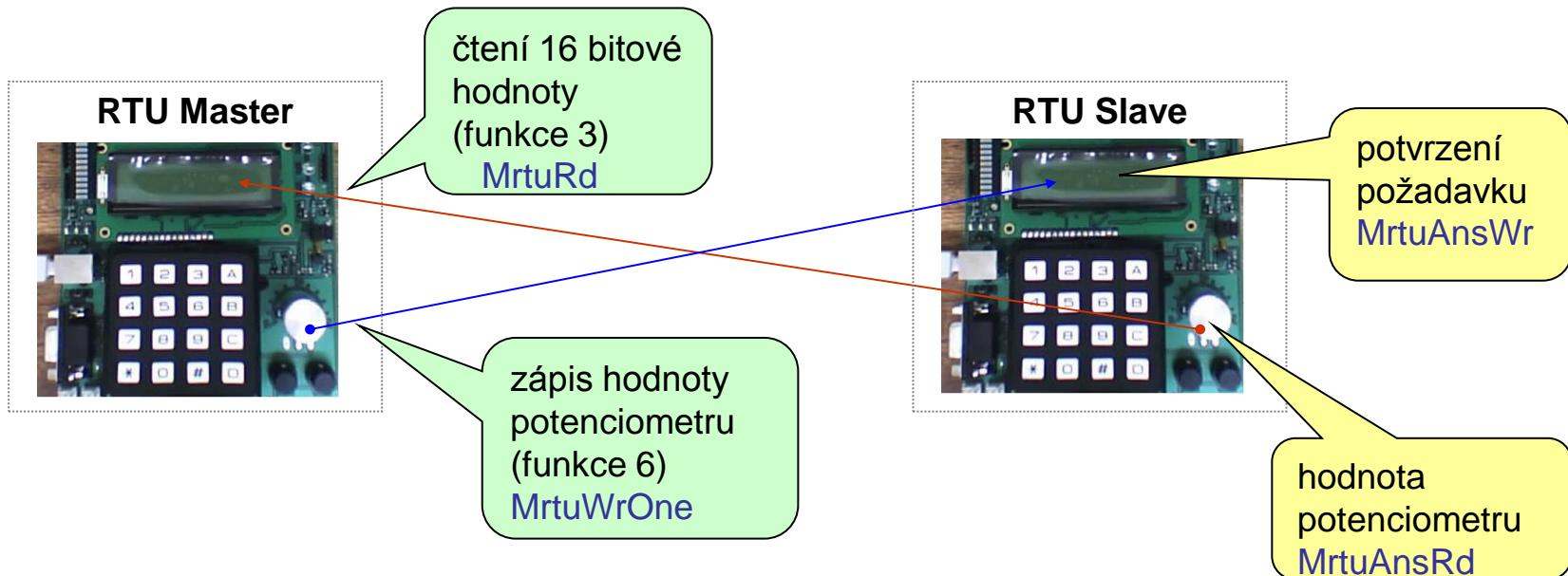
```
if(er)itx=MrtuAnsErr(adr_r,kod_r|0x80,er,bfout);
```

odeslání
odpovědi

```
DIR485=1; /* na vysílání */
itx+=MrtuWrCrc(MrtuCrc(bfout,itx),bfout+itx);
SendBuf(bfout,itx);
DIR485=0; /* zpět na příjem */
```



3.část : mikropočítač – mikropočítač



Řídicí počítačové systémy

Úloha pro samostatná cvičení - MR6

Implementace protokolu MODBUS RTU na PC a mikropočítačích řady '51 pro uzly Master (Klient) i Slave (Server).

Požadované implementované funkce:

- zápis jediného bitového stavu (Coil) do uzlu Slave,
- čtení bitového stavu (Coil) z uzlu Slave.

Rozhraní: RS232, standardní rámec 8,N,2

1. část: propojení PC – PC (C# MSVS)
 2. část: propojení PC – mikropočítač
- Rozhraní: RS485, standardní rámec 8,N,2**
3. část: propojení mikropočítač – mikropočítač

Funkce pro podporu aplikace protokolu MODBUS:

- v souboru Modbus.dll a Modbus.cs pro PC (C#),
- v souboru Modbus.H a Modbus.C pro mikropočítač



MASTER (klient)

V pravidelných časových intervalech generuje 1bitovou informaci a předává požadavek na zápis do uzlu SLAVE

V pravidelných časových intervalech generuje požadavek na čtení bitové informace z uzlu SLAVE a zobrazuje ji

SLAVE (server)

Po příjmu informaci zobrazí a odešle potvrzovací odpověď

kód fukce: 05
+ data

potvrzení

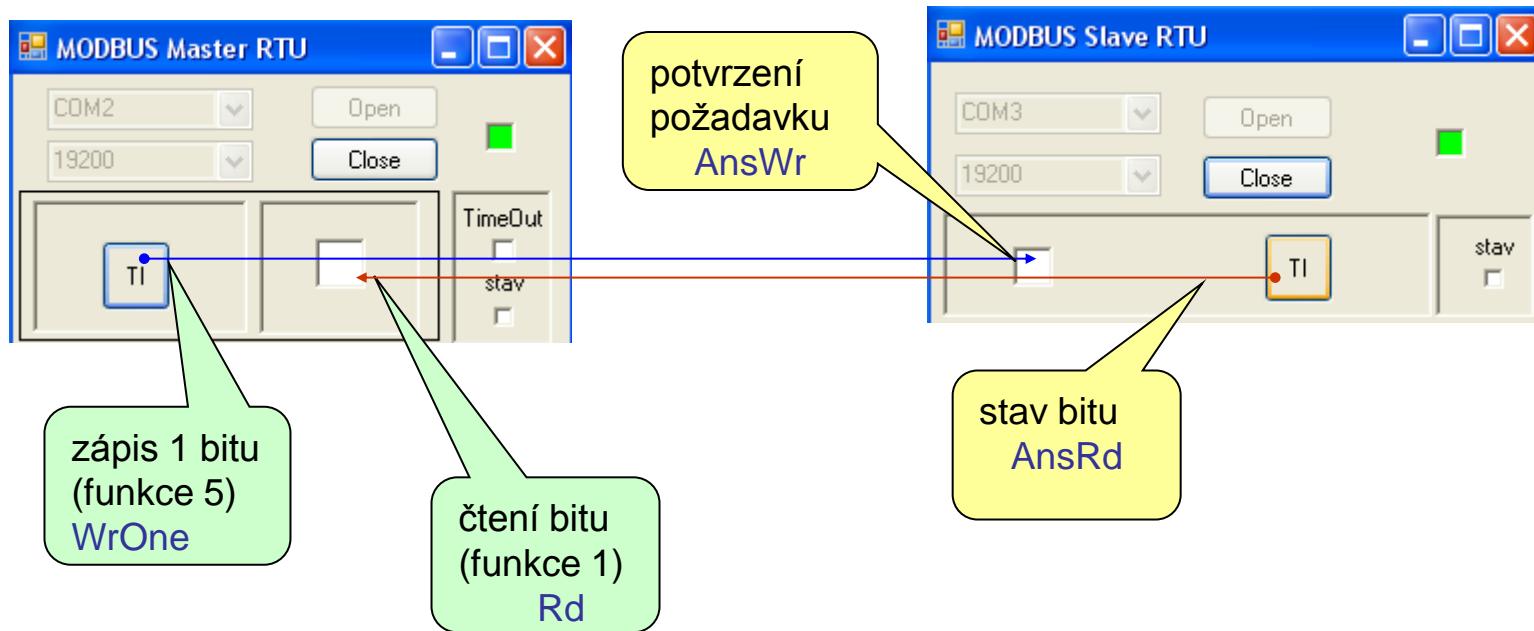
kód fukce: 01

data

Po příjmu požadavku hodnotu bitu odešle



1.část : PC-PC (varianta C#)





Master – implementace na PC (klient)

Konfigurace:

Realizuje funkce (požadavky na server)

- požadavek na zápis jediného bitového stavu – funkční kód 5

metoda WrOne třídy ModbusRTU s kódem funkce 5 (FCE_WBIT)

- požadavek na čtení bitové hodnoty – funkční kód 1

metoda Rd třídy ModbusRTU s kodem funkce 1 (FCE_RBIT)

Požadavky odesílat střídavě v pravidelných časových intervalech 200 ms,

jen když je sériový kanál otevřen a Master je ve stavu **klidu**

realizace časovačem intervalu

Implementovat generování čekacího TimeOut intervalu 500 ms na odpověď od Slave,

po vypršení TimeOutu vyčkat 500 ms a vátit se do stavu **klidu**

Zjednodušený příjem odpovědi

příchozí adresu Slave není nutno testovat, pouze správnost CRC

zpracovat jen odpověď na požadavek čtení bitu (FCE_RBIT)

informovat o chybové odpovědi od Slave

Master – vyslání požadavku

Časovač Sample

střídavě každých cca 200 ms vysílá rámec s funcí 1 (čtení bitu) a 5 (zápis bitu)

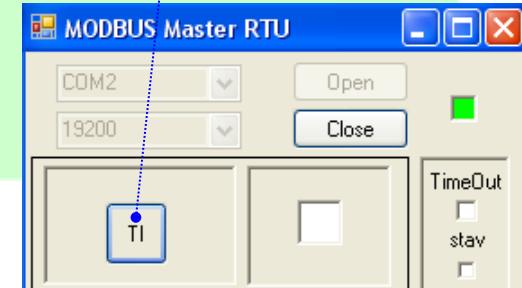


ModbusRTU Mr;
bool prep;
Tstav stav;

Tick
Sample

```
if (comPort.IsOpen && stav == Tstav.stKlid)
{
    stav=Tstav.stVysilani;
    prep = !prep;
    if (prep) n=Mr.WrOne(ADR_S,FCE_WBIT,BIT_WR,value,bfout)
    else n=Ma.Rd(ADR_S,FCE_RBIT,BIT_RD,1,bfout);
    n=Mr.WrCrc(Mr.Crc(bfout,n),bfout,n);
    comPort.Write(bfout,0,n);
    TimerOut.Interval=500;
    TimerOut.Enabled=true;
    stav=Tstav.stCekani;
}
```

stisk: 0xFF00
jinak: 0



Master – zpracování odpovědi

1. CRC

```
if(Mr.Crc(bfin,ix-1)!=Mr.RdCrc(bfin,ix-1)
{
    .. možná informace o chybné CRC
}
else {
```

2. adresa a kód funkce

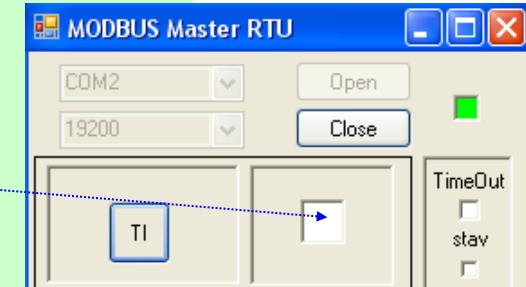
```
adr_r=bfin[0];
kod_r=bfin[1];
```

3. reakce na odpověď

```
if(kod_r== FCE_RBIT)
{
    pocet=bfin[2];
    val=bfin[3];
    if (adr_r==ADR_S && pocet==1)
        if (val & 1 ==1) { žlutá }
        else { bílá }
}
else if (kod_r>=0x80)
{
```

4. informace o chybě Slavu

```
    er= bfin[2];
    switch(er) {
        informace o chybě slavu
    }
}
stav=Tstav.stklid;
```





Slave – implementace na PC (server)

Konfigurace:

Přijímá požadavky od klienta a vrací odpovědi

- požadavek na zápis jediného bitového stavu – funkční kód 5, stav indikuje a vrací potvrzení o přijetí požadavku

metoda `AnsWr` třídy `ModbusRTU` s kódem přijaté funkce

- požadavek na čtení 1 bitové hodnoty – funkční kód 1 a vrací požadovanou hodnotu

metoda `AnsRd` třídy `ModbusRTU` s kódem přijaté funkce

Kontrolovat přijatý požadavek a vracet chybovou odpověď v případě neimplementované funkce, neexistující adresy registru nebo bitu a hodnoty mimo rozsah

metoda `AnsErr` třídy `ModbusRTU` s upraveným kódem funkce a typem chyby

Skupinové vysílání ignorovat .



Slave – zpracování požadavku, kontrola položek, příprava odpovědi

1. CRC

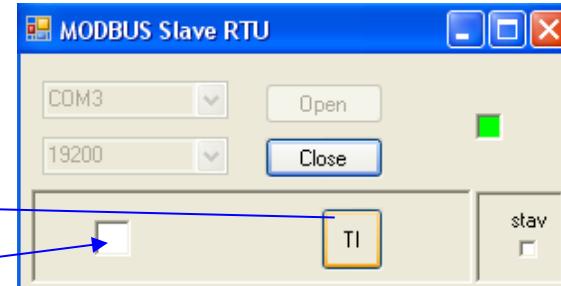
```
if(Mr.Crc(bfin, ix-1) != Mr.RdCrc(bfin, ix-1)
{
    .. možná informace o chybné CRC
}
else {
```

2. adresa

```
adr_r=bfin[0];
if(adr_r == ADR_S)
{
```

3. kód funkce

```
kod_r=bfin[1];
er=0;
switch(kod_r) {
    case FCE_RBIT:
        .
        .
    case FCE_WBIT:
        .
        .
    default: er=1;
}
```





Slave – zpracování požadavku, kontrola položek, příprava odpovědi

FCE_WBIT:

```
reg=Mr.RdWord(bfin,2);
val=Mr.RdWord(bfin,4);
if (reg!=BIT_WR) er=2;
else switch (val) {
    case 0xFF00: .. žlutá
    case 0x0000: .. bílá
    default: er=3;
}
if(er==0) n= Mr.Answr(ADR_S,kod_r,reg,val,bfout);
```



FCE_RBIT:

```
reg=Mr.RdWord(bfin,2);
pocet=Mr.RdWord(bfin,4);
if(reg!=BIT_RD || pocet!=1) er=2;
else .. tlačítka -> vals[0]
if(er==0) n= Mr.Answr(ADR_S,kod_r,1,vals,bfout);
```

byte []vals = byte[1];

4. chyba

if(er>0) n=Mr.Answr(adr_r,(byte)(kod_r|0x80),er,bfout);

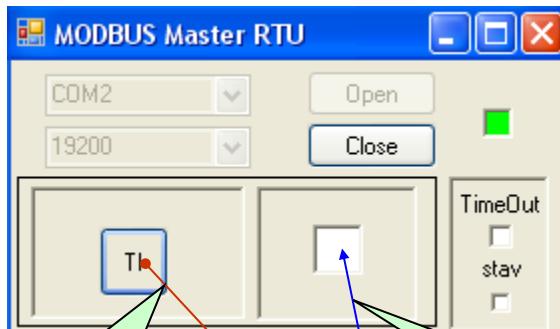
odeslání
odpovědi

n=Mr.WrCrc(Mr.Crc(bfout,n),bfout,n);
comPort.Write(bfout, 0, n);

stav=Tstav.stklid;

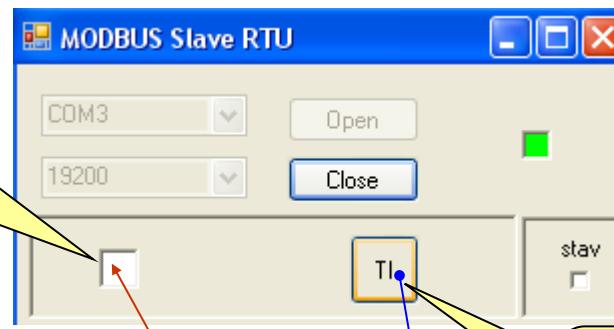


2.část : PC – mikropočítač



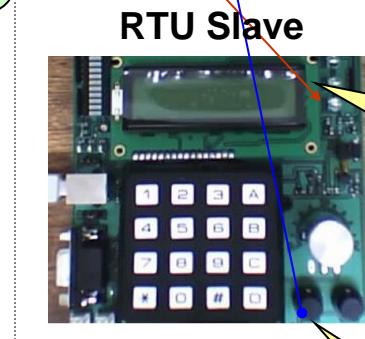
zápis 1 bitu
(funkce 5)
WrOne

potvrzení
požadavku
AnsWr



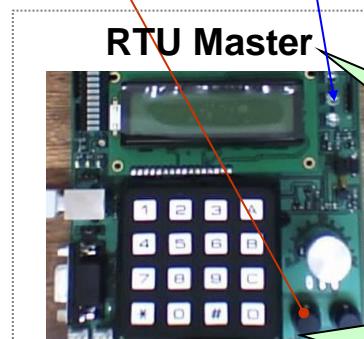
čtení bitu
(funkce 1)
Rd

1 bitová
hodnota
AnsRd



potvrzení
požadavku
MrtuAnsWr

stav tlačítka
MrtuAnsRd



čtení bitu
(funkce 1)
MrtuRd

zápis stavu
tlačítka
(funkce 5)
MrtuWrOne



Master – implementace na mikropočítači (klient)

Konfigurace:

Realizuje funkce (požadavky na server)

- požadavek na zápis jediného bitového stavu – funkční kód 5

[aplikativní funkce MrtuWrOne s kódem funkce 5 \(FCE_WBIT\)](#)

- požadavek na čtení bitové hodnoty – funkční kód 1

[aplikativní funkce MrtuRd s kodem funkce 1 \(FCE_RBIT\)](#)

Požadavky odesílat střídavě v pravidelných časových intervalech cca 200 ms,

jen když je sériový kanál otevřen a Master je ve stavu **klidu**

realizace časovačem T0 se základními tiky 30 ms ($30 * 7 = 210$)

Implementovat generování čekacího TimeOut intervalu 500 ms na odpověď od Slave
($30 * 17 = 510$)

Zjednodušený příjem odpovědi

příchozí adresu Slave není nutno testovat, pouze správnost CRC

zpracovat jen odpověď na požadavek čtení bitu (FCE_RBIT)

informace o chybě Slave: jen omezeně, nebo vůbec



Master – vyslání požadavku

střídavě každých cca 210 ms vysílá rámec s funcí **1** (čtení bitu) a **5** (zápis bitu)

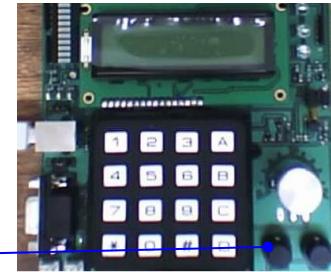


```
#define N_TICKS 7
#define TIMEOUT 17
bit prep;
```

časovač T0

```
if(++cnt_ticks>=N_TICKS && stav==stKlid)
{
    cnt_ticks=0;
    DIR485=1; /* na vysílání - pro RS485*/
    prep=!prep;
    if(prep) {val= ... ;
        itx=MrtuWrOne(ADR_S,FCE_WBIT,BIT_WR,val,bfout);}
    else itx=MrtuRd(ADR_S,FCE_RREG,REG_RD,1,bfout);
    itx+=MrtuWrCrc(MrtuCrc(bfout,itx),bfout+itx);
    SendBuf(bfout,itx);
    stav=stCekani;
    DIR485=0; /* zpět na příjem */}

```



TimeOut

```
if(cnt_ticks>=TIMEOUT)
{
    cnt_ticks=0;
    LED_R=!LED_R; // signalizace TimeOutu
    stav=stKlid;
}
```



Master – zpracování odpovědi

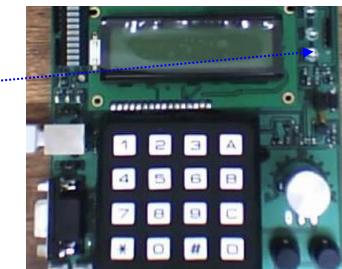
časovač T1

```
if(TF1)
{
    TR1=0;
    if(stav==stPrijem)
    {
        if(MrtuCrc(bfin,ix-2)==MrtuRdCrc(bfin+ix-2))
        {
            if ((kod_r=bfin[1]) == FCE_RBIT)
            {
                if (bfin[3] & 1) ... ; // LED svítí
                else ... ; // LED nesvítí •
            }
            stav=stKlid;
        }
    }
}
```

CRC

kód funkce

reakce
na odpověď





Slave – implementace na mikropočítači (server)

Konfigurace:

Přijímá požadavky od klienta a vrací odpovědi

- požadavek na zápis jediného bitového stavu – funkční kód 5, stav indikuje a vrací potvrzení o přijetí požadavku

aplikáční funkce MrtuAnsWr s kódem přijaté funkce

- požadavek na čtení bitové hodnoty – funkční kód 1 a vrací stav tlačítka

aplikáční funkce MrtuAnsRd s kódem přijaté funkce

Kontrolovat přijatý požadavek a vracet chybovou odpověď v případě neimplementované funkce, neexistující adresy registru nebo bitu a hodnoty mimo rozsah

aplikáční funkce MrtuAnsErr s upraveným kódem funkce a typem chyby

Skupinové vysílání ignorovat .



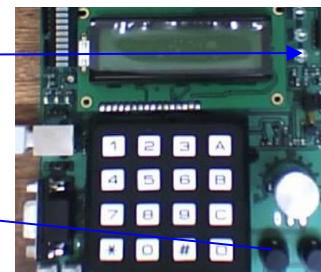
Slave – zpracování požadavku, kontrola položek, příprava odpovědi

CRC
a adresa

```
if((bfin[0]==ADR_S)&&(MrtuCrc(bfin,ix-2)==MrtuRdCrc(bfin+ix-2)))  
{
```

kód
funkce

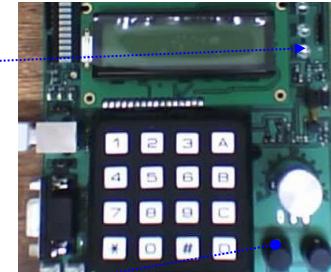
```
er=0;  
switch(kod_r=bfin[1])  
{  
    case FCE_WBIT:  
        .  
        .  
    case FCE_RBIT:  
        .  
        .  
    default: er=1;  
}
```



Slave – zpracování požadavku, kontrola položek, příprava odpovědi

FCE_WBIT:

```
if((reg=Rdword(bfin+2))!=BIT_WR) er=2;
else if((val=Rdword(bfin+4))!=0 && val!=0xFF00) er=3;
else LED_G ...;
if(er==0) itx=MrtuAnsWr(ADR_S,kod_r,reg,val,bfout);
break;
```



FCE_RBIT:

```
if((reg=Rdword(bfin+2))!=BIT_RD || (pocet=Rdword(bfin+4))!=1)er=2;
else {
    bity[0]= ... ;
    itx=MrtuAnsRd(ADR_S,kod_r,1,bity,bfout);
}
break;
```

byte bity[1];

chyba

```
if(er)itx=MrtuAnsErr(adr_r,kod_r|0x80,er,bfout);
```

odeslání
odpovědi

```
DIR485=1; /* na vysílání */
itx+=MrtuWrCrc(MrtuCrc(bfout,itx),bfout+itx);
SendBuf(bfout,itx);
DIR485=0; /* zpět na příjem */
```



3.část : mikropočítač – mikropočítač

