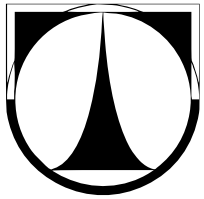


INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Implementace Modbus ASCII na PC

Ing. Josef Grosman

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

MODBUS – Aplikační vrstva

Funkční kód čtení | **Data požadavku**

02, 01
04, 03

Počáteční adresa objektu (2 byty)	Počet objektů (2 byty)

Funkční kód čtení | **Data odpovědi**

Počet bytů N (1 byte)	Hodnoty objektů (N bytů)

Funkční kód zápisu | **Data požadavku**

05, 06

Adresa objektu (2 byty)	Hodnota objektu (2 byty)

Bit: 0000H = 0; FF00H = 1

Funkční kód zápisu | **Data odpovědi**

Adresa objektu (2 byty)	Hodnota objektu (2 byty)

Funkční kód zápisu | **Data požadavku**

0F, 10

Počáteční adresa objektu (2 byty)	Počet objektů (2 byty)	Počet bytů N (1 byte)	Hodnoty objektů (N bytů)

Funkční kód zápisu | **Data odpovědi**

Počáteční adresa objektu (2 byty)	Počet objektů (2 byty)

MODBUS – Aplikační vrstva - kód chyby

Chybový funkční kód (funkční kód + 80H)	Kód chyby

Kód chyby 01	Neznámý kód funkce	Požadovaná funkce není ve Slavu implementována
Kód chyby 02	Neznámý objekt	Požadovaná adresa objektu ve Slavu mimo rozsah
Kód chyby 03	Chyba dat objektu	Zapisovaná data do objektu ve Slavu mimo rozsah

Vrstvový model

Funkce v aplikační vrstvě

Parametry

-funkční kód	pro čtení objektu	pro zápis jednoho objektu	pro zápis více objektů
-	adresa 1. objektu	adresa objektu	adresa 1. objektu
-	počet objektů	hodnota objektu	počet objektů
-			pole hodnot objektů

Výstup
- PDU (pole bytů)

Funkce v linková vrstvě protokol ASCII

Parametry
- adresa
- PDU

Zařazení počátečního znaku (:)
Transformace bytů na ASCII znaky
Výpočet a zařazení LRC
Zařazení koncových znaků (CR,LF)

Výstup
- rámec ASCII (pole bytů)

Smíšený model

Funkce pro protokol ASCIII

Parametry
- adresa
- funkční kód — pro čtení objektu — pro zápis jednoho objektu — pro zápis více objektů
- — adresa 1. objektu — adresa objektu — adresa 1. objektu
- — počet objektů — hodnota objektu — počet objektů
- — — — pole hodnot objektů

Zařazení počátečního znaku (:), transformace bytů na ASCII znaky,
výpočet a zařazení LRC, zařazení koncových znaků (CR,LF)

Výstup
- rámec ASCII (pole bytů)

MODBUS ASCII – Implementace na PC jazyk C#

C# Class lib Modbus.dll – zdrojový kód Modbus.cs

```
namespace Modbus;
public class ModbusASCII       //  metody pro Modbus ASCII

  byte AHex(byte c);
  byte HexAsc(byte b);

  byte RdByte(byte[] bf,int n);
  ushort RdWord(byte[] bf,int n);
  int WrByte(byte b,byte[] bf,int n);
  int WrWord(ushort w,byte[] bf,int n);

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

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

pomocné funkce pro převod mezi binárními hodnotami a znaky

pomocné funkce pro operace s daty

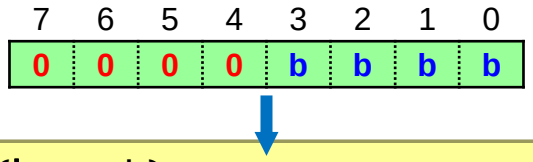
funkce pro požadavek (MASTER – Klient)

funkce pro odpověď (SLAVE – Server)

funkce pro zabezpečení LRC

funkce pro ukončení CR,LF

pomocné funkce pro převod mezi binárními hodnotami a znaky

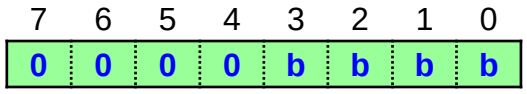


HEX → ASCII (vysílání)

```
byte HexAsc(byte b)
{
   if(b<10) return (byte)(b+'0');
   return (byte)(b-10+'A');
}
```

ASCII → HEX (příjem)

```
byte AHex(byte c)
{
  if((c>=(byte)'0')&&(c<=(byte)'9'))return (byte)(c-'0');
  if((c>=(byte)'A')&&(c<=(byte)'F'))return (byte)(c-'A'+10);
  return 0xFF;
}
```



pomocné funkce pro operace s daty

Pole bytů, kam budou data zapisována

Pozice, od které budou data zapisována

```
int WrByte(byte b,byte[] bf,int n)
{
  bf[n++] = HexAsc((byte)(b >> 4));
  bf[n++] = HexAsc((byte)(b & 0xF));
  return n;
}
```

aktuální počet bytů uložených v poli bf

Pole bytů, odkud budou data čtena

Pozice, od které budou data čtena

```
byte RdByte(byte[] bf,int n)
{
  return (byte)(AHex(bf[n])<< 4
                | AHex(bf[n+1]));
}
```

Pole bytů, kam budou data zapisována

Pozice, od které budou data zapisována

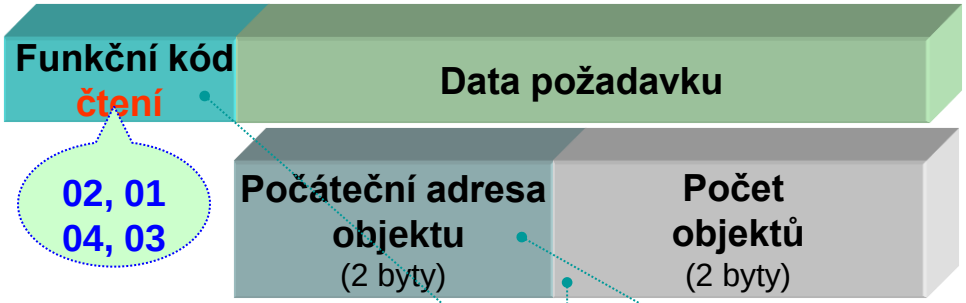
```
int WrWord(ushort w,byte[] bf,int n)
{
  bf[n++] = HexAsc((byte)(w >> 12));
  bf[n++] = HexAsc((byte)((w >> 8) & 0xF));
  bf[n++] = HexAsc((byte)((w >> 4) & 0xF));
  bf[n++] = HexAsc((byte)(w & 0xF));
  return n;
}
```

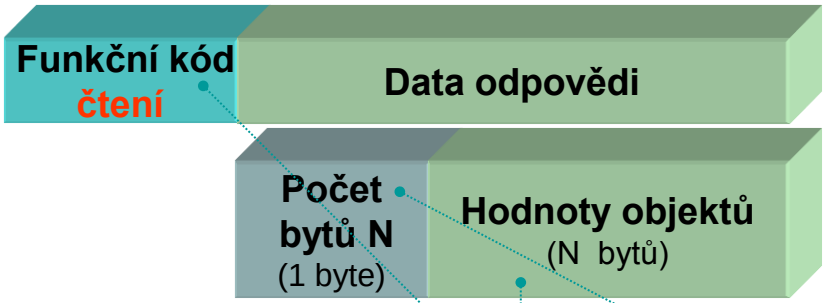
aktuální počet bytů uložených v poli bf

Pole bytů, odkud budou data čtena

Pozice, od které budou data čtena

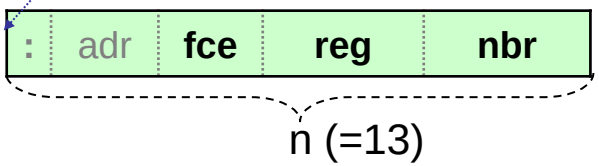
```
ushort RdWord(byte[] bf,int n)
{
  return (ushort)(AHex(bf[n] << 12 |
                  AHex(bf[n+1] << 8 |
                  AHex(bf[n+2] << 4 |
                  AHex(bf[n+3]);
}
```

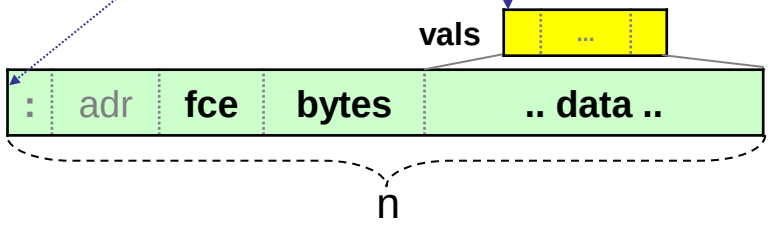


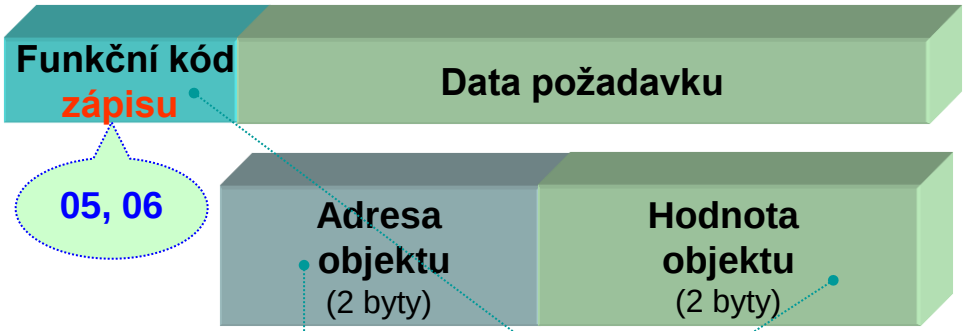



```
int Rd(byte adr,byte fce,ushort reg,
                    int nbr,byte[] bf)
{
   bf[0] = (byte)':';
   int n = WrByte(adr,bf,1);
   n = WrByte(fce,bf,n);
   n = WrWord(reg,bf,n);
   n = WrWord((ushort)nbr,bf,n);
   return n;
}
```

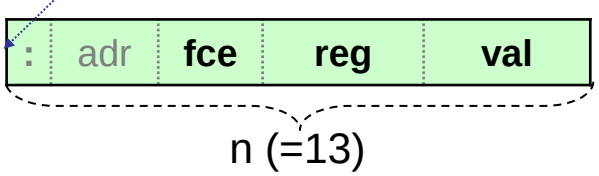
```
int AnsRd(byte adr,byte fce, int bytes,
                    byte[] vals,byte[] bf)
{
   bf[0] = (byte)':';
   int n = WrByte(adr,bf,1);
   n = WrByte(fce,bf,n);
   n = WrByte((byte)bytes,bf,n);
   for(int i=0;i<bytes;i++)
        n=WrByte(vals[i],bf,n);
   return n;
}
```

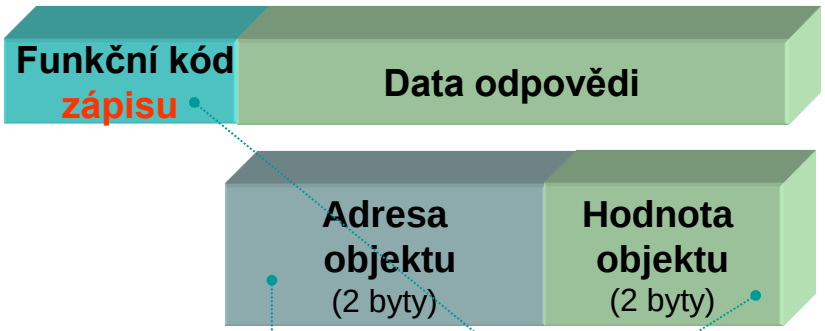




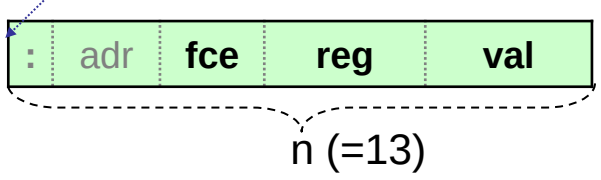


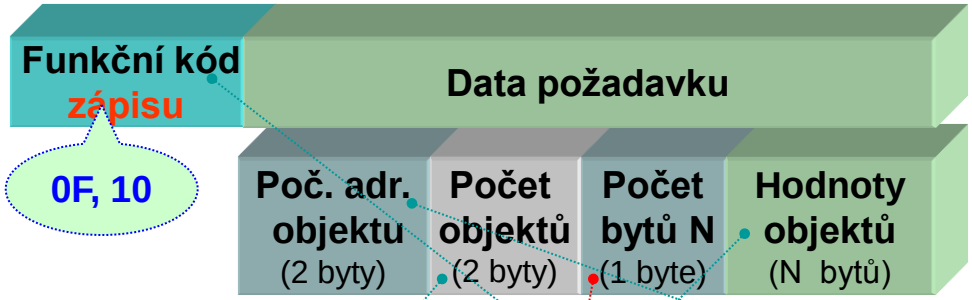

```
int WrOne(byte adr,byte fce,
   ushort reg,ushort val,byte[] bf)
{
  bf[0] = (byte)':';
  int n = WrByte(adr,bf,1);
  n = WrByte(fce,bf,n);
  n = WrWord(reg,bf,n);
  n = WrWord(val,bf,n);
  return n;
}
```

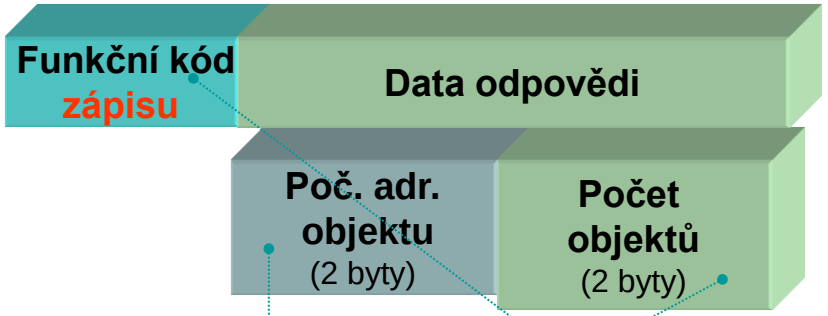





```
int AnsWr(byte adr,byte fce,
   ushort reg,ushort val,byte[] bf)
{
  bf[0] = (byte)':';
  int n = WrByte(adr,bf,1);
  n = WrByte(fce,bf,n);
  n = WrWord(reg,bf,n);
  n = WrWord(val,bf,n);
  return n;
}
```

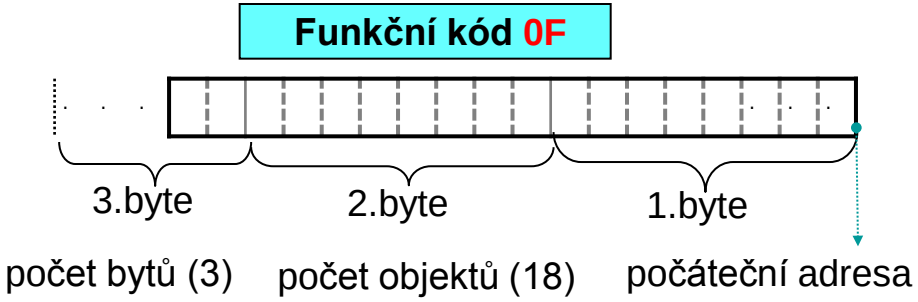







```
int Wr(byte adr, byte fce, ushort reg,
          int nbr,byte[] vals,byte[] bf)
{
  bf[0] = (byte)':';
  int n = WrByte(adr,bf,1);
  n = WrByte(fce,bf,n);
  n = WrWord(reg,bf,n);
  n = WrWord((ushort)nbr,bf,n);
  int bytes = 0;
  switch(fce)
  {
    case 16: bytes = 2 * nbr; break;
    case 15: bytes = (nbr+7)/8; break;
  }
  n = WrByte((byte)bytes,bf,n);
  for(int i=0;i<bytes;i++)
          n = WrByte(vals[i],bf,n);
  return n;
}
```

```
int AnsWr(byte adr,byte fce,
   ushort reg,ushort val,byte[] bf)
{
  bf[0] = (byte)':';
  int n = WrByte(adr,bf,1);
  n = WrByte(fce,bf,n);
  n = WrWord(reg,bf,n);
  n = WrWord(val,bf,n);
  return n;
}
```

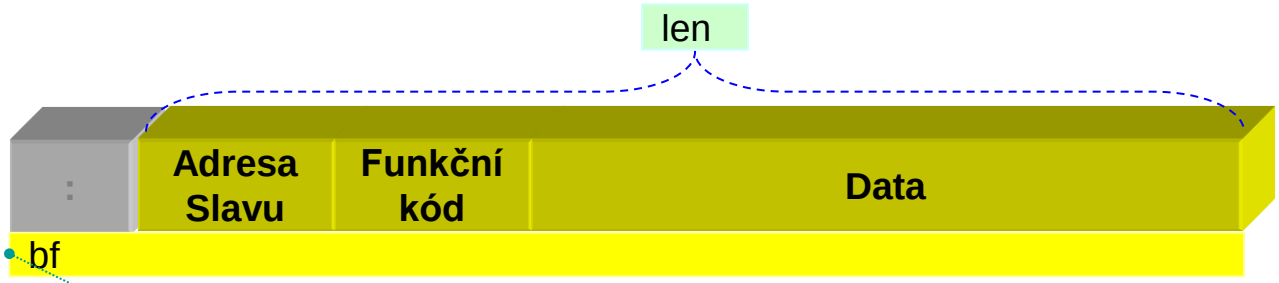


Chybová odpověď

Kód chyby 01	Neznámý kód funkce	Požadovaná funkce není ve Slavu implementována
Kód chyby 02	Neznámý objekt	Požadovaná adresa objektu ve Slavu mimo rozsah
Kód chyby 03	Chyba dat objektu	Zapisovaná data do objektu ve Slavu mimo rozsah

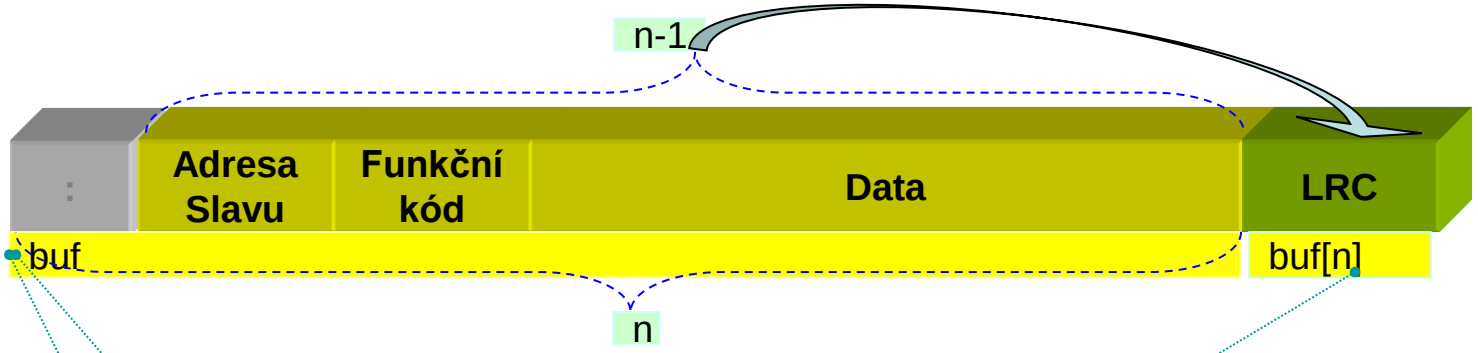

```
int AnsErr(byte adr,byte fce,byte er,byte[] bf)
{
   bf[0] = (byte)':';
   int n = WrByte(adr,bf,1);
   n = WrByte(fce,bf,n);
   n = WrByte(er,bf,n);
   return n;
}
```

Zabezpečení - LRC




```
byte Lrc(byte[] bf,int len)
{
  byte lrc = 0;
  while(len>0) lrc += bf[len--];
  return (byte)(-lrc);
}
```

Zabezpečení - LRC




```
byte Lrc(byte[] bf,int len)
{
  byte lrc = 0;
  while(len>0) lrc += bf[len--];
  return (byte)(-lrc);
}
```

```
n = WrByte(Lrc(buf,n-1),buf,n));
```

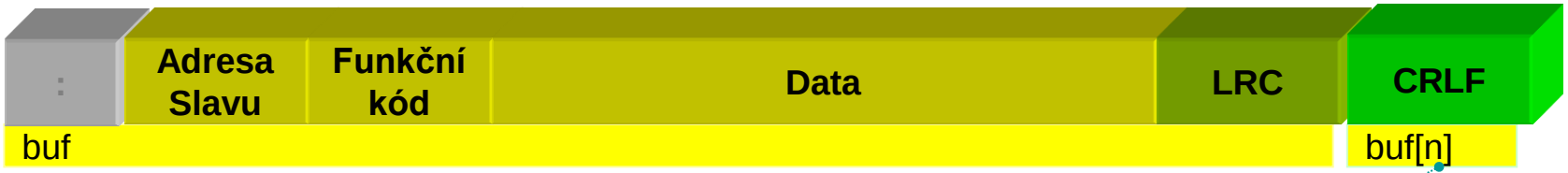
počet bytů po přidání LRC

Pole bytů, kam budou data zapisována

Pozice, od které budou data zapisována

```
int WrByte(byte b,byte[] bf,int n)
{
  bf[n++] = HexAsc((byte)(b >> 4));
  bf[n++] = HexAsc((byte)(b & 0xF));
  return n;
}
```

Ukončení - EoT



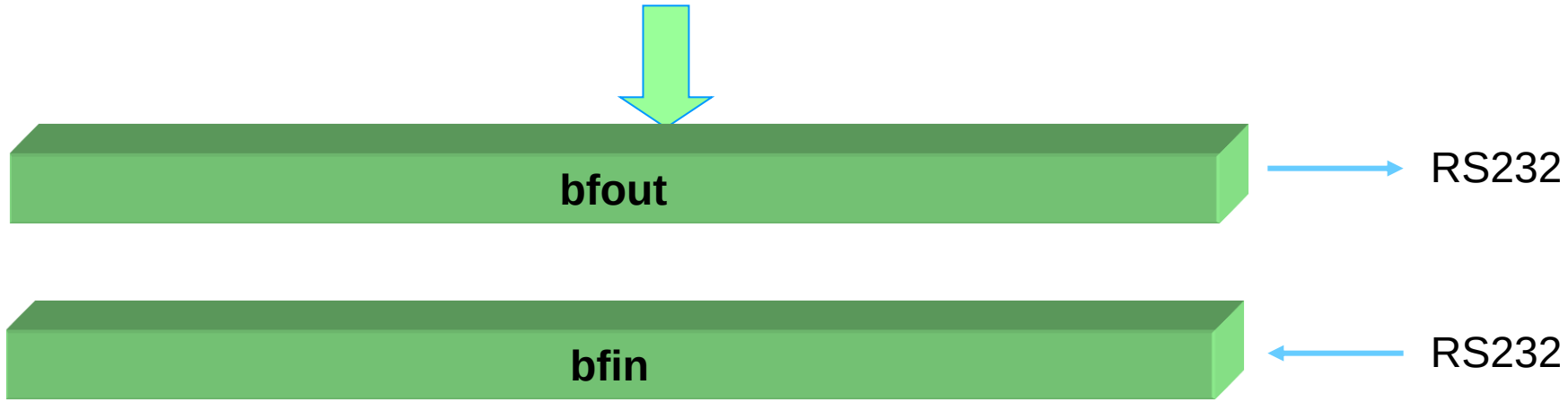

```
int WrEoT(byte[] bf,int n)
{
   bf[n++] = (byte)'\r';
   bf[n++] = (byte)'\n';
   return n;
}
```

Pozice, od které se CRLF uloží

```
n = WrEoT(buf,n);
```

Výsledný počet bytů zprávy pro vyslání

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



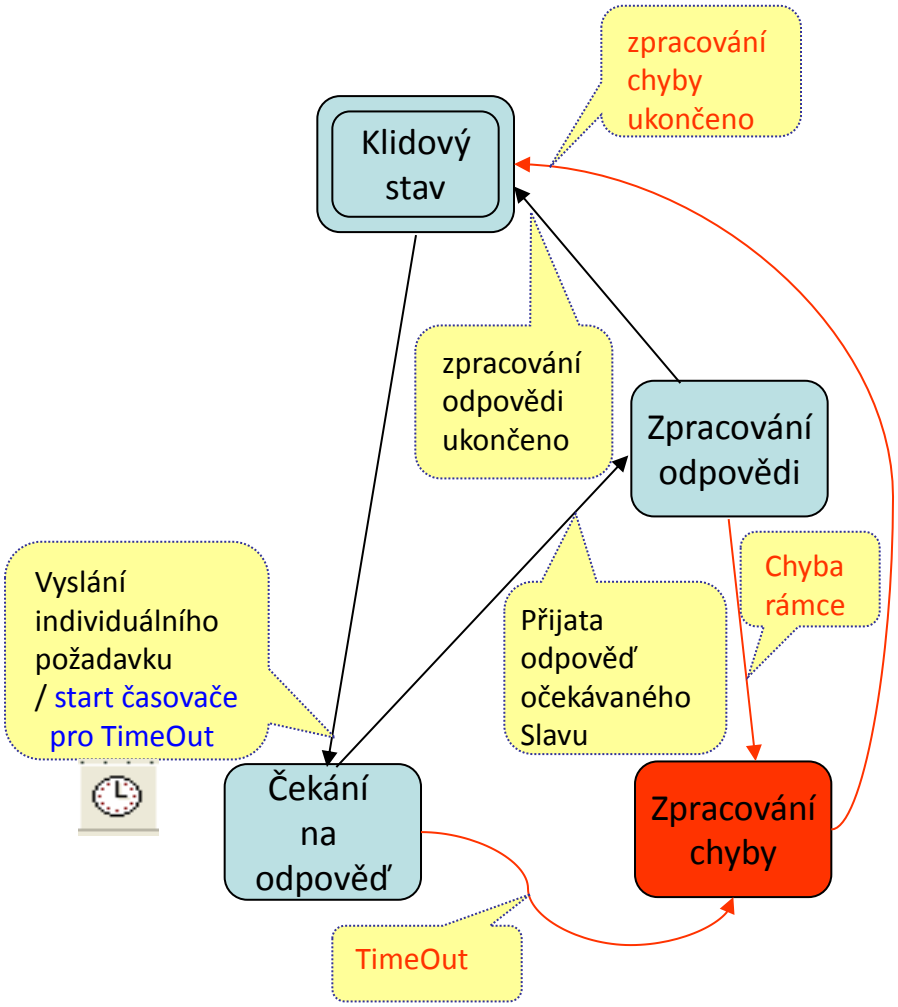

```
byte []bfin = new byte[512];
byte []bfout = new byte[512];
int  ix,n;
```

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

.

.

Master




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

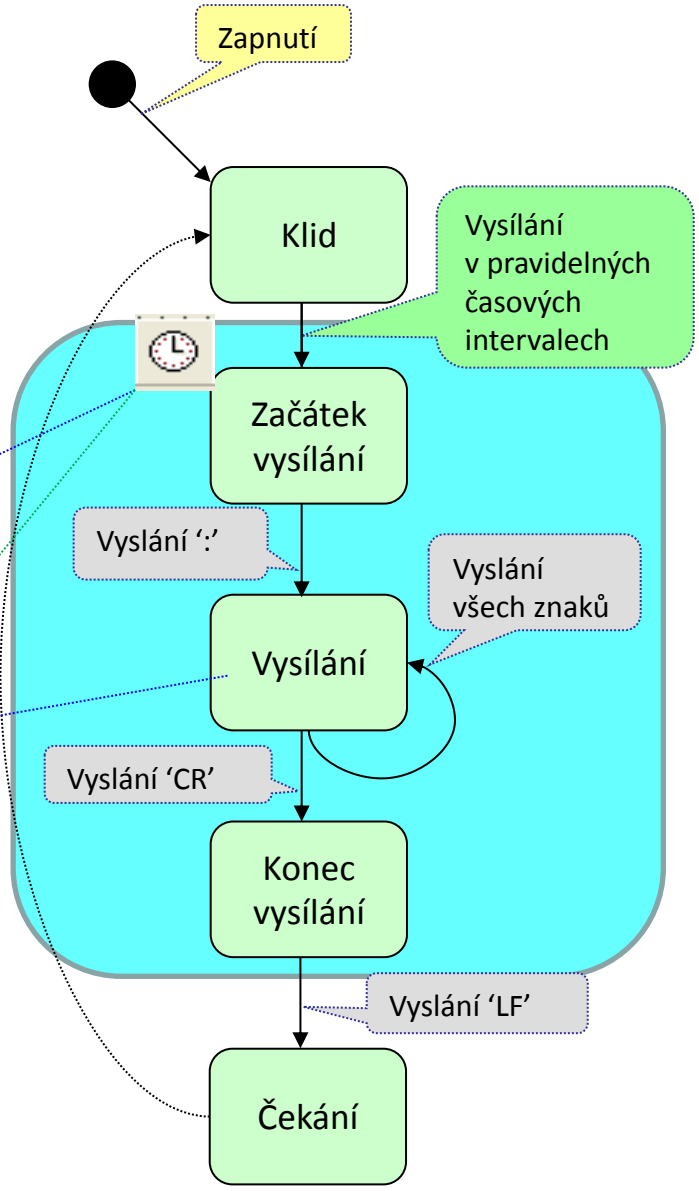
Master – vyslání požadavku

```
ModbusASCII Ma;
byte[] bfin = new byte[512];
byte[] bfout = new byte[512];
int ix,n;
Tstav stav;
```

```
n=Ma.WrOne(ADR_S,FCE_W,REG_WR,val,bfout);
```

```
n=Ma.Rd(ADR_S,FCE_R,REG_RD,1,bfout);
```

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



Master – vyslání požadavku

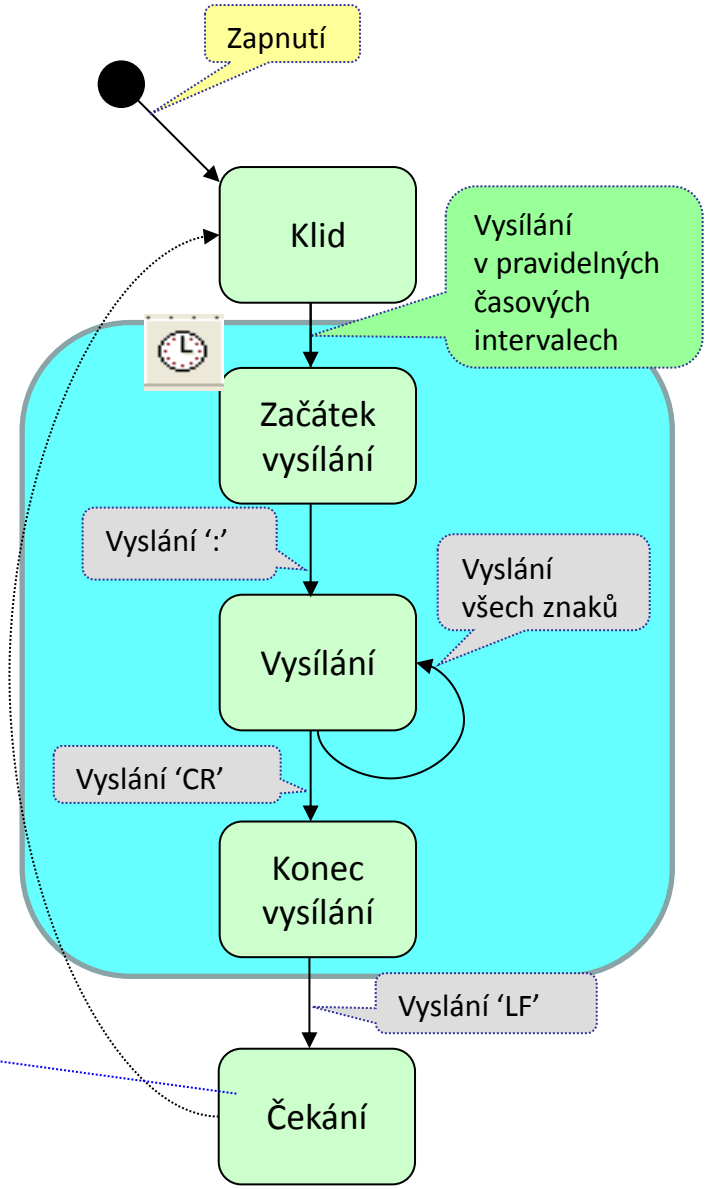
```
ModbusASCII Ma;
byte[] bfin = new byte[512];
byte[] bfout = new byte[512];
int ix,n;
Tstav stav;
```

```
n=Ma.WrOne(ADR_S,FCE_W,REG_WR,val,bfout);
```

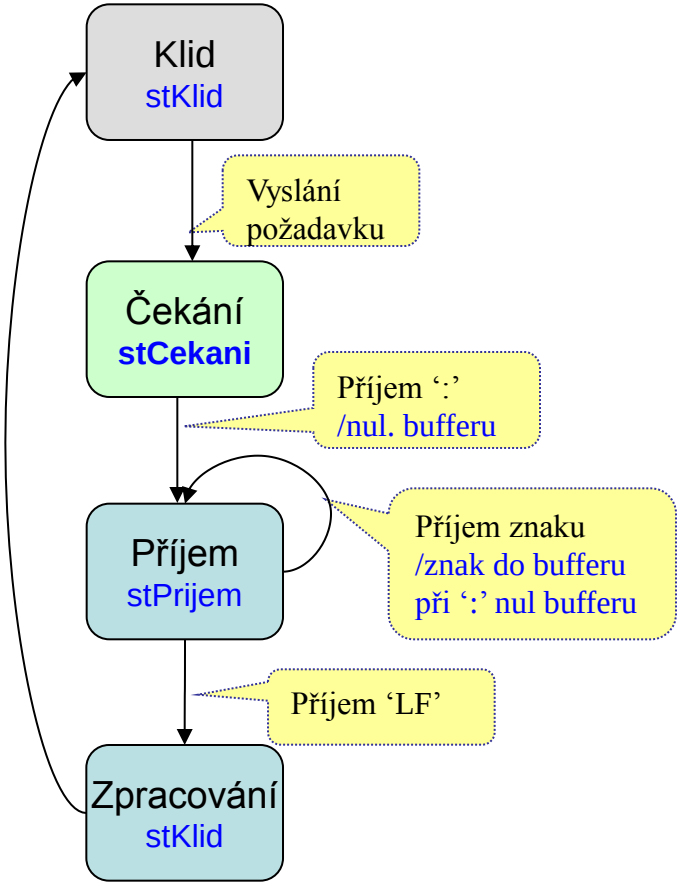
```
n=Ma.Rd(ADR_S,FCE_R,REG_RD,1,bfout);
```

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

```
timerOut.Enable=true;
stav=Tstav.stCekani;
```



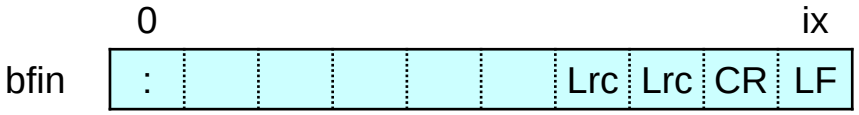
Master – příjem odpovědi



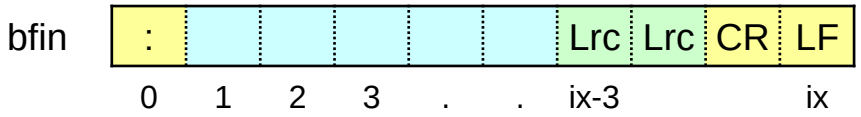

```
if(b==(byte)':')
{
   stav = Tstav.stPrijem;
   bfin[ix=0] = b;
}
```

```
if(b==(byte)':') ix=0; else ix++;
bfin[ix] = b;
```

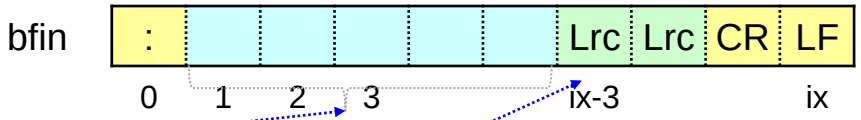
```
if(b==(byte)'\n'){
     .
     .
```



Master – zpracování odpovědi



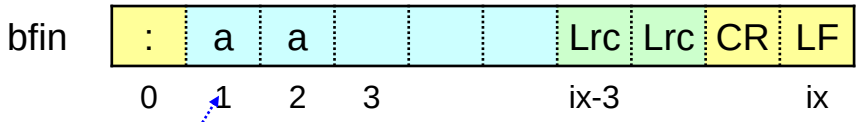
Master – zpracování odpovědi



1. LRC

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

Master – zpracování odpovědi



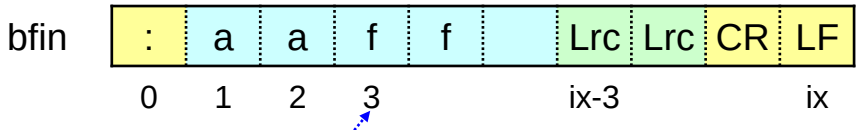
1. LRC

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

2. adresa

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

Master – zpracování odpovědi



1. LRC

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

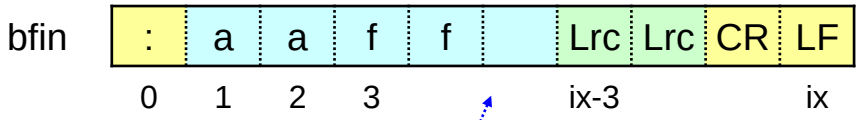
2. adresa

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

3. kód funkce

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

Master – zpracování odpovědi



1. LRC

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

2. adresa

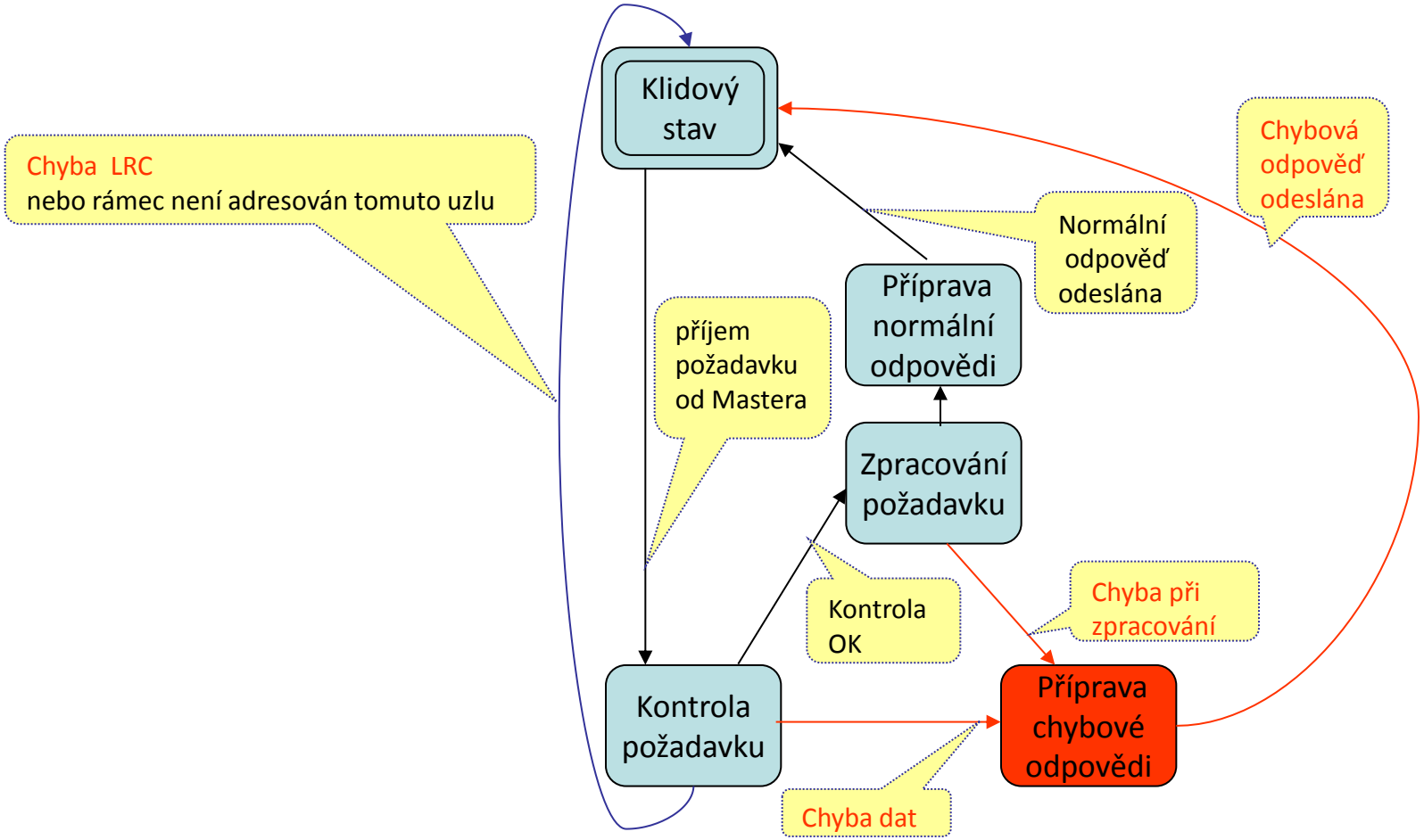
```
adr_r = Ma.RdByte(bfin,1);
```

3. kód funkce

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

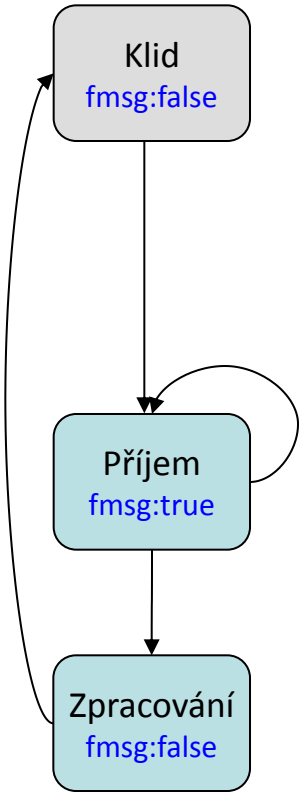
```
if(kod_r >= 0x80)
{
    .. informace o chybě SLAVE
}
else
{
    .. zpracování odpovědi na požadavek pro čtení ze SLAVE
     pocet = Ma.RdByte(bfin,5);
     valbyte1 = Ma.RdByte(bfin,7);
      .

}
```

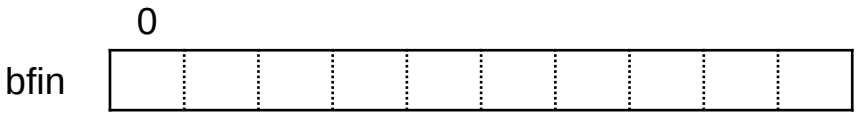
Slave



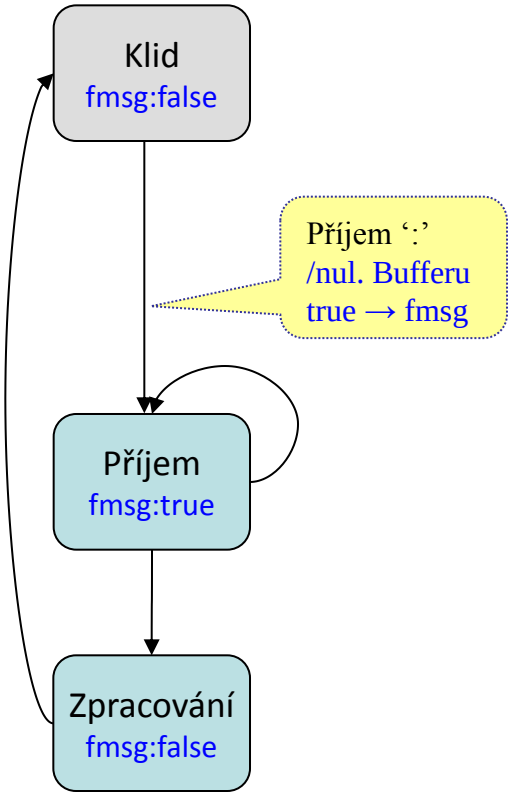

```
bool fmsg; // true – příjem, false – klid a zpracování
```

Slave – příjem požadavku

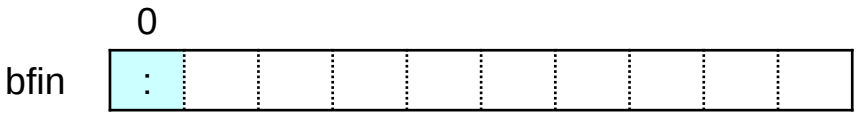




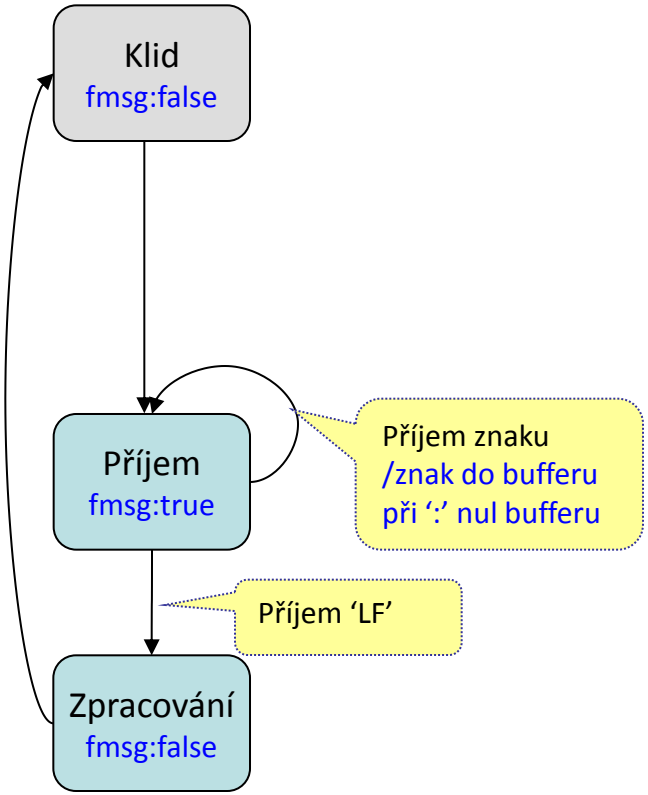
Slave – příjem požadavku




```
if(b==(byte)':')
{
   fmsg=true;
   bfin[ix=0] = b;
}
```



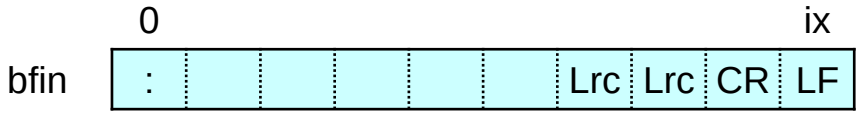
Slave – příjem požadavku



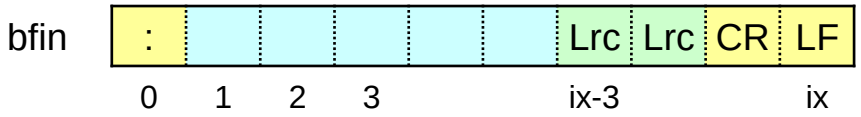

```
if(b==(byte)':')
{
   fmsg=true;
   bfin[ix=0] = b;
}
```

```
if(b==(byte)':') ix=0; else ix++;
bfin[ix] = b;
```

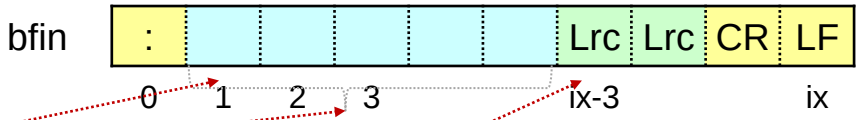
```
if(b==(byte)'\n' && fmsg)
{
   fmsg=false;
   .. zpracování požadavku
     .
}
```



Slave – zpracování požadavku



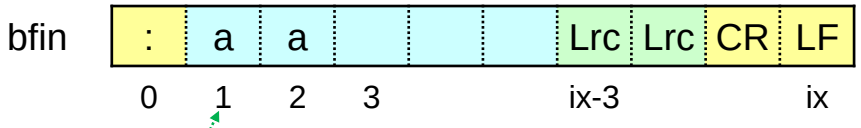
Slave – zpracování požadavku



1. LRC

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

Slave – zpracování požadavku



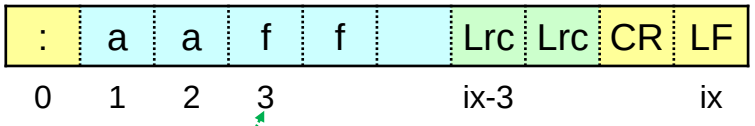
1. LRC

```
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) {
```

Slave – zpracování požadavku



1. LRC

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

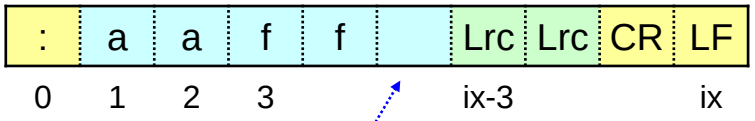
2. adresa

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

3. kód funkce

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

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);
```

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

Slave – vyslání odpovědi

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

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

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

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

Požadovaná funkce není ve Slavu implementována
1→er

Požadovaná adresa objektu ve Slavu mimo rozsah
2→er

Zapisovaná data do objektu ve Slavu mimo rozsah
3→er

Slave – zpracování chyby

Požadovaná funkce není ve Slavu implementována
1→er

Požadovaná adresa objektu ve Slavu mimo rozsah
2→er

Zapisovaná data do objektu ve Slavu mimo rozsah
3→er

```
er=0;
switch(kod_r){
   case FCE_1:
        .
   case FCE_2:
        .
   case FCE_N:
        .
   default: er=1;
}
```

```
reg=Ma.RdWord(bfin,..);
if(reg!=ADR_REG) er=2;
```

```
val=Ma.RdWord(bfin,..);
if(val<VAL_MIN || val>VAL_MAX) er=3;
```