16/10/2005

Aide à la programmation des communications Extralink

Le problème pour le programmeur qui ne poeu pas utiliser la librairie XA_dll fournie est de passer d'une fonction décritre dans le manuel, par exemple XA ('LEDON', adrmod), au programme qui effectuera la construction du message, son envoi et sa réception.

La méthodologie proposée est la suivante. Elle est basée sur le programme minimal **XA_mini.pas**, qui utilise les procédures du fichier **XA_dllmini.pas et** auxquels on pourra se reporter en cas de difficulté. Ce programm est en Pascal (Delphi), lisible sous éditeur de textes.

1 – Trouver l'équivalent numérique de la fonction

à l'aide de la table suivante et sachant que les fonctions notées Fxx ont pour équivalent numérique xx.

Numeros : 1 à 19 ; Code : 'F1' à 'F19',
Numero : 20 ; Code : 'WRB',
Numero : 21 ; Code : 'SETBIT',
Numero : 21 ; Code : 'WRB1',
Numero : 22 ; Code : 'RSTBIT',
Numero : 22 ; Code : 'WRB2',
Numero : 30 ; Code : 'WRI',
Numero : 31 ; Code : 'WRI1',
Numero : 32 ; Code : 'WRI2',
Numero : 40 ; Code : 'WRL',
Numero : 41 ; Code : 'WRL1',
Numero : 42 ; Code : 'WRL2',
Numero : 43 ; Code : 'WRL3',
Numero : 50 ; Code : 'HITRANS',
Numero : 50 ; Code : 'WRCB',
Numero : 51 ; Code : 'LOTRANS',
Numero : 51 ; Code : 'WRCB1',
Numero : 52 ; Code : 'HILOTRANS',
Numero : 52 ; Code : 'WRCB2',
Numero : 53 ; Code : 'WRCB3',
Numero : 54 ; Code : 'WRCB4',
Numero : 55 ; Code : 'WRCB5',
Numero : 56 ; Code : 'WRCB6',
Numero : 60 ; Code : 'DELAY',
Numero : 60 ; Code : 'WRCI',
Numero : 61 ; Code : 'BLINK',
Numero : 61 ; Code : 'WRCI1',
Numero : 62 ; Code : 'WRCI2',
Numero : 80 ; Code : 'LCDSET',
Numero : 81 ; Code : 'LCDSTR',
Numero : 81 ; Code : 'WRSTR',
Numero : 82 ; Code : 'LCDSTRPOS',
Numero : 98 ; Code : 'WRMEM',
Numero : 99 ; Code : 'RDMEM',
Numero : 100 ; Code : 'RDB',
Numero : 101 ; Code : 'RDI',
Numero : 101 ; Code : 'RDW',
Numero : 102 ; Code : 'RDBI',
Numero : 103 ; Code : 'RDL',
Numero : 104 ; Code : 'RDBL',
Numero : 105 ; Code : 'RDIL',
Numero : 106 ; Code : 'RDBIL',
Numero : 107 ; Code : 'RDLL',
Numero : 108 ; Code : 'RDBLL',
Numero : 109 ; Code : 'RDWLL',
Numero : 111 ; Code : 'RDLLL',
Numero : 120 ; Code : 'RDCB',
Numero : 121 ; Code : 'RDCW',

Numero : 121 ; Code : 'RDCI',

Numero : 123 ; Code : 'RDCL',
Numero : 127 ; Code : 'RDCLL',
Numero : 129 ; Code : 'RDCWLL',
Numero : 140 ; Code : 'SET629',
Numero : 190 ; Code : 'CONNECT',
Numero : 191 ; Code : 'DISCONNECT',
Numero : 192 ; Code : 'ACTIVEPACK',
Numero : 200 ; Code : 'OPEN',
Numero : 201 ; Code : 'CLOSE',
Numero : 202 ; Code : 'ADRPACK',
Numero : 202 ; Code : 'ADRIP',
Numero : 203 ; Code : 'SETADR',
Numero : 204 ; Code : 'LEDON',
Numero : 205 ; Code : 'LEDOFF',
Numero : 206 ; Code : 'RDVER',
Numero : 207 ; Code : 'RSTMOD',
Numero : 208 ; Code : 'TIMOUT',
Numero : 209 ; Code : 'SUBMSG',
Numero : 210 ; Code : 'I2CWR1',
Numero : 211 ; Code : 'I2CWR2',
Numero : 212 ; Code : 'I2CWR3',
Numero : 213 ; Code : 'I2CWR4',
Numero : 214 ; Code : 'I2CWR5',
Numero : 215 ; Code : 'I2CWR6',
Numero : 220 ; Code : 'I2CRD',
Numero : 250 ; Code : 'TJ',
Numero : 251 ; Code : 'TK',
Numero : 252 ; Code : 'RDN'

2 – définir une procédure pour insérer un caractère à la fin du message

```
//*****  insertion d'un caractère à la fin du message
procedure XA_raw_INSERTB (octet : byte) ;
begin
Out_buffer [XA_lastcar_out] := octet ;
if XA_lastcar_out < 255 then inc (XA_lastcar_out) ;
somme := somme + octet ;
end ;
```

3 – définir une procédure d'insertion avec transcodification
qui ajoute directement un octet à la fin du message s'il n'est pas dans l'intervalle [26..30] et qui ajoute deux caractères sinon (26 suivi de octet – 26).

```
//***** insertion après transcodification
procedure XA_INSERTB (octet: byte) ;
begin
if octet in [26..30] then
  begin
  XA_raw_INSERTB (26) ;
  XA_raw_INSERTB (octet - 26) ;
  end
else
  XA_raw_INSERTB (octet) ;
end ;
```

4 – définir une procédure pour initialiser un message :

- insèrer le caractère 27 comme premier caractère,
- insérer 2 fois l'adresse du module
- si le message est une lecture, ajouter 1 à l'adresse du module,

```
//***** initialisation d'un message
procedure XA_createmsg (adrmod : byte ; read : boolean) ;
begin
XA_raw_INSERTB (27) ;
if read then XA_INSERTB (adrmod + adrmod + 1)
else XA_INSERTB (adrmod + adrmod) ;
XA_lastcar_out := 0 ;
end ;
```

5 – définir une procédure pour initialiser un message pour les modules réadressables :
on ajoute alors la longueur de la réponse attendue, et la fonction.

```
//***** pour readressables, on ajoute LNGREC et la fonction
procedure XA_build (adrmod, fonction, lngrec : byte) ;
begin
XA_createmsg (adrmod, true) ;
XA_INSERTB (lngrec) ;
XA_INSERTB (fonction) ;
end ;
```

6 – la fonction **XAN** utilise toutes les précédentes pour construire un message
La structure des messages dépend du type des arguments et de la fonction.
Des intervalles numériques plus ou moins importants sont attribués à des fonctions fréquemment utilisées.
Le controleur de la base interprétera les codes fonctions et les partiés des adresses, et les adaptera notamment aux modules non réadressables.

```
//***** construction du message en fonction de la fonction
function XAN (fonction : longint ; adrmod  : longint ; arg1 : longint = 0 ; arg2 : longint = 0 ;
          arg3 : longint = 0 ; arg4 : longint = 0 ; arg5 : longint = 0) : longint ;
var foncbyte : byte ;
   entier : integer ;
   long : longint ;
begin
coderr := 0 ;
foncbyte := fonction and $ff ;
case foncbyte of
 01..19, 204, 205, 207 : begin                    // fonctions sans argument
        XA_build (adrmod, foncbyte, 1) ;
        XA_send ;
        end ;
 20..29, 80, 98 : begin                           // fonctions avec 1 byte
        XA_build (adrmod, foncbyte, 1) ;
        XA_insertB (arg1) ;
        XA_send ;
        end ;
 30..39 : begin                                   // fonctions avec 1 mot
        XA_build (adrmod, foncbyte, 1) ;
        XA_insertB (hi (arg1)) ;
        XA_insertB (lo (arg1)) ;
        XA_send ;
        end ;
 40..49 : begin                                   // fonctions avec 1 long
        XA_build (adrmod, foncbyte, 1) ;
        entier := arg1 shr 16 ;
        XA_INSERTB (hi (entier)) ;
        XA_INSERTB (lo (entier)) ;
        XA_INSERTB (hi (arg1)) ;
        XA_INSERTB (lo (arg1)) ;
        XA_send ;
        end ;
```

```
50..59 : begin                                  // fonctions avec 2 bytes
      XA_build (adrmod, foncbyte, 1) ;
      XA_insertB (arg1) ;
      XA_insertB (arg2) ;
      XA_send ;
      end ;
60..69 : begin                                  // fonctions avec 1 byte + 1 mot
      XA_build (adrmod, foncbyte, 1) ;
      XA_insertB (arg1) ;
      XA_insertB (hi (arg2)) ;
      XA_insertB (lo (arg2)) ;
      XA_send ;
      end ;
70..79 : begin
      XA_build (adrmod, foncbyte, 1) ;          // fonctions avec 1 byte + 1 long
      XA_insertB (arg1) ;
      entier := arg2 shr 16 ;
      XA_INSERTB (hi (entier)) ;
      XA_INSERTB (lo (entier)) ;
      XA_INSERTB (hi (arg2)) ;
      XA_INSERTB (lo (arg2)) ;
      XA_send ;
      end ;
100..119 : begin                                // lectures simples de N octets
      XA_build (adrmod, foncbyte, foncbyte - 98) ;
      XA_send ;
      end ;
120..135 : begin                                // ecritures 1 byte + lectures N octets
      XA_build (adrmod, foncbyte, foncbyte - 118) ;
      XA_insertB (arg1) ;
      XA_send ;
      end ;
140    : begin                                  // fonction spéciale LM629
      XA_build (adrmod, foncbyte, 1) ;
      XA_INSERTB (hi (arg2)) ; XA_INSERTB (lo (arg2)) ;
      XA_INSERTB (hi (arg3)) ; XA_INSERTB (lo (arg3)) ;
      XA_INSERTB (hi (arg4)) ; XA_INSERTB (lo (arg4)) ;
      XA_INSERTB (hi (arg5)) ; XA_INSERTB (lo (arg5)) ;
      XA_insertB (arg1) ;
      XA_send ;
      end ;

150..165 : begin                                // ecritures 1 byte + lectures N octets
      XA_build (adrmod, foncbyte, foncbyte - 133) ;
      XA_insertB (arg1) ;
      XA_send ;
      end ;
203    : begin                                  // adresse module réadressable
      XA_build (adrmod, foncbyte, 1) ;
      XA_insertB (arg1) ;
      XA_send ;
      end ;
206    : begin                                  // version DLL
       XA_build (adrmod, foncbyte, 3) ;
       XA_send ;
       end ;
210    : begin                                  // Ecritures I2C pures
      XA_createmsg (adrmod, false) ;
      XA_INSERTB (arg1) ;                       // 1 byte
      XA_send ;
      end ;
211    : begin
      XA_createmsg (adrmod, false) ;            // 2 bytes
      XA_INSERTB (arg1) ;
      XA_insertB (arg2) ;
      XA_send ;
      end ;
```

```
 212   : begin
       XA_createmsg (adrmod, false) ;                  // 3 bytes
       XA_INSERTB (arg1) ;
       XA_insertB (arg2) ;
       XA_insertB (arg3) ;
       XA_send ;
       end ;
 213   : begin
       XA_createmsg (adrmod, false) ;                   // 4 bytes
       XA_INSERTB (arg1) ;
       XA_insertB (arg2) ;
       XA_insertB (arg3) ;
       XA_insertB (arg4) ;
       XA_send ;
       end ;
 214   : begin
       XA_createmsg (adrmod, false) ;                   // 5 bytes
       XA_INSERTB (arg1) ;
       XA_insertB (arg2) ;
       XA_insertB (arg3) ;
       XA_insertB (arg4) ;
       XA_insertB (arg5) ;
       XA_send ;
       end ;
 220   : begin                                          // Lecture I2C pure
       XA_createmsg (adrmod, true) ;
       XA_INSERTB (arg1) ;
       XA_send ;
       end ;
 else
       coderr := 255 ;                                  // code fonction inconnu
 end ;
XAN := coderr ;
end ;
```

7 - La procédure XA_send termine le message en ajoutant la checksum, le caractère de fin et emet le message

```
//***** envoi du message
procedure XA_send ;
var ir, iw : byte ;
begin
XA_INSERTB (lo (somme)) ;
XA_raw_INSERTB (29) ;

--- emettre le message, recevoir la réponse dans Buffer_in jusqu'à avoir recu le caractère de fin (30)
--- ou à sortir en time-out. Si on n'est pas en time-out, on transfère le buffer en opérant la transcodification inverse:
si on rencontre le caractère 26, on le cumule avec le suivant.

ir := 0 ;
iw := 0 ;
for iw := 0 to 255 do
  Buffer [iw] := 0 ;
while Buffer_in [ir] <> 30 do
  begin
  Buffer [iw] := Buffer [iw] + Buffer_in [ir] ;
  If Buffer [ir] <> 26 then iw := iw + 1 ;
  End ;

--- on vérifie ensuite que tous la somme de tous les caractères est bien égale à 255 (caractère 30 excepté).
```

8 – écrire les fonctions nécessaires pour regrouper les octets du buffer
en mots de 16 ou 32 bits selon le langage et les besoins (**XA_get**).

Fin document