TA7IO.dll en kommunikationsmodul

Genom att använda TA7IO.dll så kan ett VBA laddad program som tex iFIX eller Excel:

- Läsa och skriva data samt alla minnesceller i TA65xx/6711 ducar.
- Läsa och skriva data samt styra HIPCL i TA7Drv modulen.
- Hantera TA7Tree.txt filen för ett ”TA System7 ID-Träd”.
- Läsa och skriva data samt alla minnesceller i parameter fil TA7IOPar.par.
- Läsa ducars kommentarer från TA7Kxxx.txt filerna.

Detta möjliggör att trädstrukturen med ID-begrepp kan hämtas från TA System7 och implementeras i PC utan att någon driver eller databas behövs. Det kan röra sig om tiotusentals punkter.
Se iFix bilden TATree.grf som ett exempel.

Man kan även spara ändringar som görs på iFix bilder till TA7Progs datafiler så att de kommer med vid nästa UPDA (uppdatering av duc). Vidare kan man i VBA visa och ändra regulatorer, larmgränser mm via tex formulär. Egentligen kan man skapa alla färgbilder utan att ha någon SCADA support.

Varför inte testa här direkt (dvs om du aktiverat makro): Run

Vill du titta på VBA koden välj i Words meny:
Verktyg - Makro - Visual Basic Editor

Gränssnitt i VBA

Först måste DLL:en och dess enda funktion definieras, tex:

```
Private Declare Function TA7IO Lib "c:\Dynamics\APP\TA7IO\TA7IO.dll" _
                         (ByVal strQ As String, ByVal strA As String) As Long
```

Funktionen bygger på att du skickar en fråga i **strQ** och får ett svar i **strA**.
Observera dock att du först måste reservera plats i **strA** strängen (ett VBA / C++ problem).
Tex för att läsa MV(5) i DUC(7) kan du skriva:

```
strA = Space(100)
If TA7IO( ”DUC007MV5”, strA) = 0 Then
    MsgBox ”Svaret blev ” & strA
Else
    MsgBox ”Nu blev det fel:” & strA
If End
```

Funktionen returnerar alltså 0 om allt gick bra och svaret i **strA**.
Vid skrivning eller HIPCL kommando blir svaret "!OK" om allt gick bra.
Vid fel returneras ett felnummer och, om det finns plats i strängen, en feltext i **strA** .
Exempel på feltexter är "!BUSY", "!OFFLINE", "!SYNTAX", "!Error ..."

Dock måste DLL:en initieras först, så att den vet var ducarna kan nås (normalt via TA7Drv.exe) . Detta kan göras på två sätt. I båda fallen skickar man en initieringssträng i **strQ** och får versionsnummer mm som svar i **strA**. Väljer du **strQ="OPEN=<ipadr:UDPport>, <timeout>"** så kan du bara prata med ducarna, men väljer man istället **strQ="PATH=\\SCADA\Dynamics\APP\TA7Data\"** så tittar TA7IO.dll i filen TA7Duc.txt efter både UDP-porten till ducarna och vilka PLB:er som TA6711 ducar ligger under. Dessutom möjliggörs läsning och skrivning i filerna TA7IOPar.par och TA7Kxxx.txt.

Som standard blockerar/låser DLL:en anropande program under tiden den kommunicerar med ducarna. Men detta kan man ändra på genom att ställa in **strQ="OPT=NONBLOCKING"**. Då returnerar DLL:en med 0, dvs att funktionen gick bra, men **strA** innehåller **”!BUSY”**. Anropande program för då vänta lite, kanske 0.2 sekunder, och skicka samma fråga igen.

Kommandon

OPEN[=<ipadr:UDPport>, <timeout>]	Öppnar en kommunikationsport till ducar. Porten är en nätverks UDP port med angiven IP-adress och portnummer samt med angiven timeout i ms. Vid OK returneras DLL:ens versionsnummer mm.
PATH[=<sökväg>]	Öppnar en sökväg till en katalog som innehåller TA7Duc.txt, TA7IOPar.par och TA7Kxxx.txt. Via den förstnämnda filen hitter DLL:en vilken UDP port som ducarna finns på. Vid OK returneras DLL:ens versionsnummer mm.
OPT[=NONBLOCKING, BLOCKING]	Sätter DLL:ens blocking mode, dvs om den skall returnera !BUSY eller vänta tills kommunikationen med duc är färdig.
DUC<n><tab><x>[=<värde>]	Kommunikation med ducar TA65xx/TA6711. Vid läsning returneras och vid skrivning skrivs <värde>. <n> är ducnummer 1-200. <tab> är en av ducens tabeller: logiska: IN, LL, HL, FI, UT, TC, TG, ILV, LF heltals: CNT, RT, TH flyttals: MV, SV, AUT, RV, TL, IFV tidkanaler: TCS, TGS <x> är index i tabell. <värde> formatet beror på tabelltyp: Logiska: 0 och 1 samt Till, Från, Auto för forcering. Heltal: -32768…+32767 Flyttal: ex. 3.14 eller 3.45e7 eller 10 TCS: tex "06:30-18:15 1111100” TGS: <TCS-format>,<TCS-format>,.. (fyra TCS)
IOPar<n>[SV,RV,TCS,TGS]<x>[=<värde>]	Läser och skriver i TA7IOPar.par, dvs den fil som TA7Prog använder vid uppdatering av ducar. <värde> enligt DUC kommando ovan.
Kom<n>[SV,RV,TC,TG,NR]<x>	Läser kommentarer från TA7Progs datafiler TA7Kxxx.txt. Fyra färdiga tabeller finns, emedan övriga nås via ”kommentars NR-tabellen”, se nedan.
DUC<x>MEM<m>-<n>[=hh hh hh ...] **IOPar<x>MEM<m>-<n>[=hh hh hh ...]**	Läser och skriver minnesceller direkt i DUC eller i TA7IOPar.par. Minnesadresserna <m> och <n> är i decimal form och får vara 0-1023, dock får skillnaden inte vara större än 32. Vad minnesadresserna innehåller beskrivs nedan. Data <hh …> anges i hexadecimal form.
[TCL,LV,NUM,PV]<x>[=<värde>]	för kommunikation med variabler i TA7Drv:s HIPCL del. <värde> lika ovan för DUC dock kan även ”*” anges vilket motsvarar att variabelt är oinitierad, se hjälp i HIPCL.
HIPCL ...	Här kan styrning av HIPCL göras, se dokumentation för HIPCL i TA7Drv. En sammanfattning följer här: KONV, START, STOPP, MNEM, DEBUG [HUCx, MAI, <subnamn>, <a1> <a2>] [n], [TCL, LV, NUM, PV](x[-y])[=z], INIVAR <typ> [<nLV> <nNUM> <nPV>], SHOWINIVAR
MAKETREE <trädfil> **FINDBILDIO <trädfil>** **SORTTREE <trädfil>**	Tre funktioner för ID-träd hantering enligt TA System7:s ID-Träd. De två första kommandona använd endast vid systemgenerering och använder ytterligare filer. <trädfil> skall innehålla sökväg och filnamn för trädfilen, dvs normalt: c:\Dynamics\APP\TA7IO\TA7Tree.txt

Kommentars-NR och minnesadresser i DUC & TA7IOPar.par

Genom kommando **"Kom<n>NR<x>"** kan samtliga duc-kommentarer i TA7Prog läsas.
Vad numren står för beskrivs i TA7Prog:s hjälpfil.
Här följer ett utdrag.

Tabell	Kom-NR	Antal
MV	0	16
GTYP	16	8
HL	24	16
LL	40	16
IN	56	16
FI	72	16
UT	88	16
AUT	104	6
TC	110	16
SV	126	32
REG	158	12
TG	170	4
DZON	174	4
DUMAX	178	3
UMIN	181	4
UMAX	185	4
OPT	189	4
KURV	193	6
DIFF	199	8
GRANS	207	12
RT	219	8
DUTY	227	8
ELOG	235	4
DLOG	239	8
PROJ	247	1
NAMN	248	1
ILV	249	32
IFV	281	32
DUMMY	313	1.
CNT	314	16

Genom kommando
"DUC<x>MEM<m>-<n>[=hh hh...]" och
"IOPar<x>MEM<m>-<n>[=hh hh...]"
kan samtliga minnesceller i DUC och motsvarande i fil TA7IOPar.par läsas och skrivas.

OBS att startadessen är i decimal form och att de verkliga fysiska adresserna i DUC som anges i TA7Prog:s hjälpfil inte används här. Detta för att underlätta användningen av TA7IO.dll.

Startadress	Byte / x	Tabell
0	26	**SYST**
26	3	**IN(1-16)**
74	5	**MV/LL/HL(1-16)**
154	15	**REG(1-12)**
334	3	**DZON(1-4)**
346	3	**DUMAX(1-3)**
355	3	**UMIN(1-4)**
367	3	**UMAX(1-4)**
379	19	**OPT(1-4)**
455	4	**KURV(1-6)**
479	2	**UT(1-16)**
511	1	Reserv
512	2	**FI(1-16)**
544	5	**TC(1-16)**
624	9	**GRÄNS(1-8)**
696	3	**SV(1-32)**
792	5	**GTYP(1-8)**
832	1	**AUT(1-6)**
838	5	**DIFF(1-8)**
878	9	**GRÄNS(9-12)**
914	8	**RT(1-8)**
978	4	**DUTY(1-8)**
1010	2	**ELOG(1-4)**
1018	6	Reserv

Detaljförklaring till minnesadresser

Byte och bit kodning per lagringstyp, ":" avskiljer byte, "'" avskiljer 4-bit.

SYST År : Månad : Dag : <2Byte>StartFördröjning{0-65535} : <2Byte>OffLineTid{0-65535} : 00 : 00 : Språk{2=svenska} : Inscan{0=32Hz/1=64Hz} : Utnollning{0=Ej/1=Ok} : DucTyp{1=6500upg, 2=6501v2, 3=6580v2, 4=6501v3a, 5=6580v3a, 6=6501v3d, 7=6580v3d, F=6505, 1A=6711} : I/O-1 : I/O-2 {FF=No, 1=Modem, 2=RS232C, 80=8DU, 88=12DI, 90=TALinje, A0=OPpanel, B8=6AU} : 00 : 00 : 00 : 00 : 00 : 00 : 00 : 00 : 00 : 00 : 00

IN Tdll'llll : FLLc'cccc : 0000'0000
T=Typ, d=Sek/Min, ll'llll=Larmtid, F=Fiktiv, LL=Larm{0..3}, c'cccc=Cnt enligt:
0'0000=0, 1'0100=S, 1'1000=M, 1'1100=T, 0'0100=1, 0'1000=10, 0'1100=100, 1'0101=1000

MV+LL+HL LlLh'skf : 00ff'gräns : 0DTT'TTTT : 0dtt'tttt : 0000'0000
Ll=LL-Larm(0..3), Lh=HL-Larm(0..3), skf.=SkalFaktor(0..8), ff=Filterfaktor(0..3), gräns=Gräns(0..8), D=LL-Sek/Min, TT'TTTT=LL-Larmtid, d=HL-Sek/Min, tt'tttt=HL-Larmtid

REG 00TT'0KII : DDNN'NUUU : 0000'0ZZZ : IN : BV : UT(0-8.eller.FlytTalsVar) : <Flyttal>P-tal : <Flyttal>I-tid : <Flyttal>D-tid
TT=Typ{01=PI/10=PID/11=P}, K=UtgTyp{0={I}ÖkaMin/1={K}Kaskad}, II=ReglerIntervall{01=1sek, 10=10sek, 11=60sek}, DD=DUMAX(0..3), ZZ'Z=DZON(0..4), UUU=UMAX(0..4), NNN=UMIN(0..4)

DZON,DUMAX, UMIN,UMAX (4+3+4+4)=15 Flyttal

OPT Rum : Ute : TG : Bv : MinG : OptV : NormV : MinV : UteK : Ind : <Flyttal>MinK : <Flyttal>MaxK : <3Byte>00

KURV InSignal : UtVärde : BrytPunkt : AntBryt

UT Ind : Larm

FI 0Dtt'tttt:0LL0'0000
D=Min/Sek, tt'tttt=Tidsfördröjning, LL=Larm(0/1/2/3)

TC Tt : Tm : Ft : Fm : Må Ti On To Fr Lö Sö 0

GRÄNS Låggräns, Höggräns, Hysteres 3 Flyttal

SV Flyttal

GTYP SkalFaktor(Flyttal) : Nollpunkt(2 byte signed heltal)

AUT AnalogVariabel

DIFF AnalogVariabel-1 : AnalogVariabel-2 : Gräns : LågLarm : HögLarm

RT IndVar : Tid(0=S/1=M/2=T) : <2Byte>Gräns1 : <2Byte>Gräns2 : Larm1 : Larm2
”<2Byte>GränsN” är 2 bytes heltal dock max 32767

DUTY Insignal : TillTid : FrånTid : Utsignal

ELOG Mätpunkt : TSA0'0000
T=<Logginterval(0=D/1=T)>, S=<0=Medel/1=Summa>, A=<0=Passiv/1=Aktiv>

Förklaring till Kodning av:

Flyttal 2-Byte Mantissa och 1-Byte exponent:
mmmm'mmmm'mmmm'mmmm:eeee'eeee, m=mantissa(signed), e=exponent(signed)
Flyttal = m/2^15 * 2^e, OBS! Hi:Lo (PC-int=Lo:Hi)

Digtal-Variabel 01-10=IN(1-16), 11-20=LL(1-16), 21-30=HL(1-16), 31-40=UT(1-16), 41-50=FI(1-16), 51-70=???, 71-90=ILV(1-32), 91-A0=TC(1-16), A1-A4=TG(1-4), A5-FF=???

Analog-Variabel 01-10=MV(1-16), 11-30=SV(1-32), 31-50=???, 51-60=IFV(1-16) 61-70=CNT(1-16), 71-94=RV(1-36), 95-A0=???, A1-A4=TL(1-4), A5-AC=RT(1-8), AD-BC=IFV(17-32), BD-C2=AUT(1-6), C3-E2=???, E3-E7=TH(1-4), E8-FF=???

Talen i Digital- och Analogvariabel är angivna i hexadecimal form.