www.triomotion.com

Trio Motion Technology Ltd.
Shannon Way, Tewkesbury,
Gloucestershire. GL20 8ND
United Kingdom
Tel: +44 (0)1684 292333
Fax: +44 (0)1684 297929

1000 Gamma Drive
Suite 206
Pittsburgh, PA 15238
United States of America
Ph: +1 412.968.9744
Fx: +1 412.968.9746

**Doc No.: TN20-70**
**Version: 1.0**
**Date: 11th August 2004**
**Subject: Description of Profibus Basic program for P297 daughter board.**

---

# Technical Note

## *Contents:*



## *Introduction:*

The P297 Profibus Daughter Board is used in Motion Coordinators MC206, MC216, MC224, Euro205 and Euro205X to add Profibus DP slave node functionality. On board the P297 is the popular SPC3 Profibus slave chip and this document describes the Trio BASIC program that must be run to initialise and control the operation of the chip.

The program sets up the SPC3 chip for cyclic data transfer only. Up to 16 words are transferred in and 16 words go out to the master on a cycle determined by the master. (PLC) Normally this transfer takes place on a 0.2 msec update at 12Mbaud, however please note that the true time for data to appear in the VR variables depends on the BASIC program so will vary depending on the Motion Coordinator processor speed and the Process number that the program runs on.

## *System Requirements:*

Motion Coordinator MC206, MC216, MC224, Euro205 or Euro205X.
P297 Profibus Daughter Board.
P297DR106.BAS Basic program to drive the daughter board.

## *Program Description:*

The first few lines reset the local variables to zero and then intialise some program variables. The user must change the value of 'node' to be the required Profibus node number and 'db' to the slot number of the daughter board. 'Vbase' can also be changed so that the incoming/outgoing data uses a different block of the Motion Coordinator's VRs.

```
restart:
RESET
node = 5 '    Profibus node address
debug = TRUE 'Set TRUE to get debug messages printed to terminal
db=1 '        Daughter Board slot number
vbase = 20 '  VRs for data transfer
localtimeout = 5000 'time in msec
VR(vbase+33)=0
```

Reset the chip and initialise all required registers.

```
GOSUB user_dps_reset
```

Main run loop, checks for interupts. If no "interupt" activity is detected after the localtimeout value has expired, the SPC3 chip is reset and everything restarts.

```
TICKS=localtimeout
REPEAT
    GOSUB dps2_ind
    'check local watchdog
    IF TICKS<0 THEN GOTO restart
    'has profbus watchdog timed out?
    IF timeout_flag=TRUE THEN GOTO restart
UNTIL FALSE
STOP
```

Intialise the SPC3 chip. Most registers that are set in this routine have a description of the register name added as a comment and the reader can cross-reference this with the register details in the SPC3 manual.

```
user_dps_reset:

' Hardware reset SPC3:
PROFIBUS(db,1,808,8)
WA(50)
PROFIBUS(db,1,808,0)

' Hardware Watchdog Trigger *******

IF debug THEN PRINT "Initialising SPC3 Registers..."

PROFIBUS(db,1,8,4)'  Set Hardware Mode 1 byte: Force Offline
WA(50)

' Clear SPC3 Memory:

FOR byte=22 TO 1023
    PROFIBUS(db,1,byte,0)
NEXT byte

PROFIBUS(db,1,59,05)'    Set Ident High
PROFIBUS(db,1,58,149)'   Set Ident Low
PROFIBUS(db,1,22,node)'  Set Station Address

' Mode 0 registers must be set in OFFLINE mode:
```

```
PROFIBUS(db,1,6,0+0)'    Mode 0 low byte: SYNC and FREEZE mode are OFF
PROFIBUS(db,1,7,1+0+4)'  Mode 0 high byte: DP_MODE on, 10ms ticks
PROFIBUS(db,1,57,255)'   Disallow address change
PROFIBUS(db,1,10,255)'   timeout is 2.55 sec ?

' Define buffers for Dout/Din:

PROFIBUS(db,1,26,32)'    Length of 3 Dout buffers
PROFIBUS(db,1,27,60)'    Segment address of Dout buffer 1
PROFIBUS(db,1,28,64)'    Segment address of Dout buffer 2
PROFIBUS(db,1,29,68)'    Segment address of Dout buffer 3
PROFIBUS(db,1,30,32)'    Length of 3 Din buffers
PROFIBUS(db,1,31,72)'    Segment address of Din buffer 1
PROFIBUS(db,1,32,76)'    Segment address of Din buffer 2
PROFIBUS(db,1,33,80)'    Segment address of Din buffer 3

' Define diagnostic buffers:

PROFIBUS(db,1,36,10)'    Length of Diag buffer 1
PROFIBUS(db,1,37,10)'    Length of Diag buffer 2
PROFIBUS(db,1,38,14)'    Segment address of Diag buffer 1
PROFIBUS(db,1,39,18)'    Segment address of Diag buffer 2
PROFIBUS(db,1,14*8+6,4)' Load buffer with additional length
PROFIBUS(db,1,18*8+6,4)' Load buffer with additional length


' Define Auxiliary buffers:

PROFIBUS(db,1,40,16)'    Length of Aux buffer 1
PROFIBUS(db,1,41,16)'    Length of Aux buffer 2
PROFIBUS(db,1,42,1)'     Control bits for Auxiliary buffers
PROFIBUS(db,1,43,22)'    Segment address of Aux buffer 1
PROFIBUS(db,1,44,24)'    Segment address of Aux buffer 2

' Define Set_Slave_Address buffer:

PROFIBUS(db,1,45,4)'    Length of Set_Slave_Address buffer
PROFIBUS(db,1,46,26)'   Segment address of Set_Slave_Address buffer

' Define Set_Param buffer:

'PROFIBUS(db,1,47,12)'    Length of Set_Param buffer
PROFIBUS(db,1,48,27)'   Segment address of Set_Param buffer

' Define Check_Config buffer:

PROFIBUS(db,1,49,2)'    Length of Check_Config buffer
PROFIBUS(db,1,50,29)'   Segment address of Check_Config buffer

' Define Get_Config buffer:

PROFIBUS(db,1,51,2)'    Length of Get_Config buffer
PROFIBUS(db,1,52,30)'   Segment address of Get_Config buffer

' Pointer to configuration data:
cd_ptr=PROFIBUS(db,0,52)*8
IF debug THEN PRINT "get_cfg_ptr=";cd_ptr[0]
```

```
' Write configuration data:
PROFIBUS(db,1,cd_ptr,95)' number of input words:163
PROFIBUS(db,1,cd_ptr+1,111)' number of output words:16

' Set Indication Functions:

PROFIBUS(db,1,4,NOT(2+4+8))'
PROFIBUS(db,1,5,NOT(1+2+4+8+16+32))

' Set user watchdog value:

PROFIBUS(db,1,24,100)
PROFIBUS(db,1,25,100)
PROFIBUS(db,1,8,32)' Reset watchdog timer

' send the first input (input to master) buffer:
GOSUB get_din_ptr

' Fetch the first diagnosis buffer, initialize service bytes:
GOSUB get_diag_buf_ptr

PROFIBUS(db,1,10,255)'   set wd_baud_ctrl_val *****

PROFIBUS(db,1,8,1)'  Set Hardware Mode 1 byte: START_SPC3

RETURN
```

SPC3 event handler. In many systems this would be an interupt service routine but TrioBASIC does not have an interupt capability. Therefore this routine is called periodically from the main program and the Interupt Register of the SPC3 is polled to see if an interupt event has occurred.

```
dps2_ind:' This function is normally called when an interupt occurs

    intregl=PROFIBUS(db,0,0) ' Read Interupt Register Low Byte
    intregh=PROFIBUS(db,0,1) ' Read Interupt Register High Byte
    ' Combine 2 bytes to make one 16 bit interupt word.
    intr = (intregl + intregh*256)
```

If any bits are 1, then an event has occurred, so set the local timeout value to show that the Profibus is still alive.

```
    IF (intr AND 65519)<>0 THEN TICKS=localtimeout  ' 65519 is $FFEF
```

Now check the int register bits one at a time to see which ones need servicing and take the appropriate action.

```
    ' Set to go offline:
    IF intregl AND 1 THEN
        PROFIBUS(db,1,2,1) ' Acknowledge Interupt
    ENDIF
```

Go / Leave Data Exchange starts or stops the cyclic data transfer.

```
    ' Go/Leave Data Ex:
    IF intregl AND 2 THEN
        GOSUB go_leave_data_ex_function
        IF res=0 THEN timeout_flag=TRUE
```

```
    IF debug THEN PRINT "go/leave data ex = ";res[0]
    PROFIBUS(db,1,2,2) ' Acknowledge Interupt
ENDIF
```

A signal has been detected and the automatic baudrate detection must be serviced.

```
' Baudrate Detect:
IF intregl AND 4 THEN
    is_reg=PROFIBUS(db,0,4) AND 48
    IF debug THEN PRINT "is_reg=";is_reg[0]
    IF (is_reg=16) OR (is_reg=32) THEN PROFIBUS(db,1,11,store_mintsdr)
    PROFIBUS(db,1,2,4) ' Acknowledge Interupt
    IF debug THEN PRINT "BR"
ENDIF
```

Watchdog timeout functions.

```
' WD_DP_Mode_Timeout:
IF intregl AND 8 THEN
    GOSUB wd_dp_mode_timeout_function
    IF debug THEN PRINT "WD_DP Mode timeout"
    PROFIBUS(db,1,2,8) ' Acknowledge Interupt
    timeout_flag=TRUE
ENDIF

' User watchdog has timed out:
IF intregl AND 16 THEN
    PROFIBUS(db,1,2,16) ' Acknowledge Interupt
ENDIF
```

New command / data responses

```
' New GC Command:
IF intregh AND 1 THEN
    GOSUB global_ctrl_command_function
    VR(vbase+32)=res
    IF debug THEN PRINT "GC Command = ";res[0]
    PROFIBUS(db,1,3,1) ' Acknowledge Interupt
ENDIF

' New SSA Data:
IF intregh AND 2 THEN
    PROFIBUS(db,1,3,2) ' Acknowledge Interupt
    IF debug THEN PRINT "New SSA Data = ";res[0]
ENDIF

' New Config Data:
IF intregh AND 4 THEN
    IF debug THEN PRINT "New Config Data"
    cfg_result = 0
    result = 0
    REPEAT
        'check configuration data until no conflict behaviour:
        cfg_ptr = PROFIBUS(db,0,50)*8' pointer to config data !
        config_data_len = PROFIBUS(db,0,49)
        IF config_data_len <> 2 THEN
```

```
            ' In this example there are 16 words in 16 words out
            PROFIBUS(db,1,9,16) ' write EN_CHG_CFG_BUFFER
            cfg_result=PROFIBUS(db,0,17) ' Read user_cfg_data_nok
        ELSE
            ' Length of the config o.k: check bytes
            config0=PROFIBUS(db,0,cfg_ptr)
            config1=PROFIBUS(db,0,cfg_ptr+1)
            IF (cfg_akt0 = config0) AND (cfg_akt1 = config1) THEN
                result = 0
                ' Desired config is equal the actual config
            ELSE
             IF(config0=95 AND config1=111)OR(config0=17 AND config1=33)THEN
                  cfg_akt0 = config0
                  cfg_akt1 = config1
                  result=2
              ELSE
                  result=1
              ENDIF
              IF result = 2 THEN
                  ' Program should acknowledge check_config configuration
                  user_io_data_len_ptr = res
                  PROFIBUS(db,0,16)
              ENDIF
            ENDIF
        ENDIF
    UNTIL cfg_result<>1
    PROFIBUS(db,1,3,4) ' Acknowledge Interupt
ENDIF

' New Param Data:
IF intregh AND 8 THEN
    prm_result = 0
    REPEAT
        prm_ptr = PROFIBUS(db,0,48)*8' ??? pointer to data
        param_data_len = PROFIBUS(db,0,47)
        IF debug THEN PRINT "pp=";prm_ptr[0],"  pdl=";param_data_len[0]
        ' Standard param_data_len is 7 bytes
        ' additional bytes are set in COM PROFIBUS (here 5)+7=12
        IF (param_data_len > 12) THEN
            ' Acknowledge not okay:
            b1=PROFIBUS(db,0,prm_ptr+8)
            IF b1 AND(PROFIBUS(db,0,prm_ptr+9)=170) THEN
                prm_result=PROFIBUS(db,0,15)
            ENDIF
        ELSE
            ' Acknowledge okay:
            prm_result=PROFIBUS(db,0,14)
            IF debug THEN PRINT "p_res=";prm_result[0]
        ENDIF
    UNTIL prm_result<>1

    ' Have received from master standard timeout value:
    store_mintsdr = PROFIBUS(db,0,prm_ptr+3)'for restart
    IF debug THEN PRINT "store_mintsdr=";store_mintsdr[0]
    running=TRUE
    VR(vbase+33)=1
```

```
    PROFIBUS(db,1,3,8) ' Acknowledge Interupt
ENDIF
```

Diagnotic buffer has changed – no action for P297, simply respond with interupt ack.

```
' Diagnostic Buffer Changed:
IF intregh AND 16 THEN
    PROFIBUS(db,1,3,16) ' Acknowledge Interupt
    IF debug THEN PRINT "diag buf changed"
ENDIF
```

New cyclic data has arrived from the master. This is read out of the SPC3 buffers and placed in VR variables as 16 bit signed integers. Outgoing VRs are also transferred to the SPC3 buffers and then a "buffer swap" is commanded. This means that our data goes out to the profibus and we "see" the next new data coming in from the master. (There are 2 copies of the data buffers)

```
' Write Read Data Changed:
IF intregh AND 32 THEN
    PROFIBUS(db,1,3,32) ' Acknowledge Interupt

    ' Handle update to outputs (from master):

    IF running<>0 THEN
      dout_buf_cmd=PROFIBUS(db,0,10)/16 AND 3
      dout_buf_ptr=PROFIBUS(db,0,27+dout_buf_cmd-1)*8
      vrb=vbase+16
      FOR ct=0 TO 15
        hibyte=PROFIBUS(db,0,dout_buf_ptr+(2*ct))
        lowbyte=PROFIBUS(db,0,dout_buf_ptr+(2*ct)+1)
        sint=INT(lowbyte+(hibyte*256))
        IF sint>32767 THEN sint=INT(sint-65536)
        VR(vrb+ct)=sint
      NEXT ct
    ENDIF
    PROFIBUS(db,0,11)' update output buffers

    ' Handle update to inputs (to master):

    din_buf_cmd=PROFIBUS(db,0,8)/16 AND 3
    din_buf_ptr=PROFIBUS(db,0,31+din_buf_cmd-1)*8
    FOR ct=0 TO 15
      sint=VR(vbase+ct)
      IF sint>32767 THEN sint=32767
      IF sint<-32768 THEN sint=-32768
      IF sint<0 THEN sint=sint+65536
      lowbyte=sint AND 255
      hibyte=(sint/256) AND 255

      PROFIBUS(db,1,din_buf_ptr+(2*ct),hibyte)
      PROFIBUS(db,1,din_buf_ptr+(2*ct)+1,lowbyte)
    NEXT ct
    PROFIBUS(db,0,9)' rotate input buffers

ENDIF

' Set EOI in DP_Din_Buffer_State_Machine:
```

```
    PROFIBUS(db,1,8,2)
    PROFIBUS(db,1,8,32)' Reset watchdog timer
    IF running THEN loops=loops+1
    IF loops=500 THEN PROFIBUS(db,0,9)

RETURN
```

This next section is where the actual "service routines" are for handling the different events.

```
wd_dp_mode_timeout_function:
    ' Profibus watchdog has run out.

    user_wd_state = (PROFIBUS(db,0,4)/64) AND 3

    len=6
    IF (PROFIBUS(db,0,12) AND 3)=1 THEN
        PROFIBUS(db,1,36,len)
    ELSE
        IF (PROFIBUS(db,0,12) AND 12)=4 THEN
            PROFIBUS(db,1,37,len)
        ELSE
            len=255
        ENDIF
    ENDIF
    running=FALSE
    VR(vbase+33)=0
    loops=0
    IF debug THEN
     PRINT "Profibus Timeout - Offline"
     PRINT ""
    ENDIF
RETURN

global_ctrl_command_function:
    res=PROFIBUS(db,0,60)
RETURN

go_leave_data_ex_function:
    res=(PROFIBUS(db,0,4)/16) AND 3
RETURN

get_diag_buf_ptr:
   chg_diag_b_new = 0
   chg_diag_b_old = 0

   diag_buffer_sm=PROFIBUS(db,0,12)
   IF (diag_buffer_sm AND 3)=1 THEN
        user_diag_buffer_ptr=PROFIBUS(db,0,38)*8
   ELSE
      IF (diag_buffer_sm AND 12)=4 THEN
         user_diag_buffer_ptr=PROFIBUS(db,0,39)*8
      ELSE
         user_diag_buffer_ptr=0
      ENDIF
   ENDIF
   PROFIBUS(db,0,13)' Make new diag buffer available
   user_diag_flag = TRUE
```

```
RETURN

get_din_ptr:
   din_buffer_sm=(PROFIBUS(db,0,8)/16) AND 3
   IF din_buffer_sm THEN
       user_din_ptr=PROFIBUS(db,0,31+(din_buffer_sm-1))*8
   ELSE
       user_din_ptr=0
   ENDIF
RETURN
```